Maddie Waldie
March 7, 2024
COEN 169

COEN 169 - Project 2

About my code:

I've attached a zip file of my code, as well as copy and pasted my code to the bottom of this report. To run each piece of code, follow these instructions:

1. Unzip the folder
2. Open terminal, and cd to the Project 2 folder: `cd Project2`
3. Run the files using python3:
    - `python3 ubfCosine.py`
    - `python3 ubfPearson.py`
    - `python3 ubfIUFPearson.py`
    - `python3 ubfCaseModPearson.py`
    - `python3 ibfCosine.py`
    - `python3 myAlgorithm.py`

By default, the files will use test5.txt, and will create recommendations based on this file. If you want to use test10.txt or test20.txt, you will need to go into each file and change the instances of "test5" to "test10" or "test20."

Results Discussion: *Compare the accuracy of the various algorithms. Please include a table to show the Mean Absolute Error (MAE) of your different algorithms that you obtain from the online submission system. Do you think your results are reasonable? Can you analyze the advantages and disadvantages of the algorithms?*

The table below shows the Mean Absolute Error (MAE) results for various collaborative filtering algorithms, with an overall MAE computed across three different files. The user-based collaborative filtering (UBF) methods include cosine similarity, Pearson correlation, and an inverse user frequency variant, as well as a modification involving case sensitivity in Pearson correlation. Additionally, item-based collaborative filtering (IBF) using cosine similarity is assessed, as well as an algorithm I created using euclidean distance.

Algorithm	MAE of Given 5	MAE of Given 10	MAE of Given 20	Overall MAE
UBF Cosine Similarity	0.837689133425034	0.782	0.769653708883959	0.795025447381382
UBF Pearson Correlation	0.8388145554582	0.782	0.7694607890415	0.795312756526
UBF Pearson Correlation Inverse User Frequency	0.8398149305989	0.780333333333	0.7733191858782	0.796872434739
UBF Pearson Correlation Case Modification	0.8343128673252	0.7771666666666	0.7709076878556	0.79326054835

IBF Cosine Similarity	0.830811554332875	0.78	0.76531301244333	0.790428501067148
Own Algorithm	0.858321870701513	0.839666666666667	0.842866788849233	0.847151535051716

Analyzing the results, it is evident that the MAE values vary across different collaborative filtering algorithms. Let's delve into the findings and discuss the advantages and disadvantages of each algorithm:

User-Based Collaborative Filtering (UBF):

- **Cosine Similarity:** Achieves the lowest overall MAE of 0.795, making it a relatively accurate method.
- **Pearson Correlation:** Performs similarly to Cosine Similarity but with a slightly higher overall MAE of 0.7953.
- **Inverse User Frequency:** Exhibits a higher overall MAE of 0.7969 compared to the other UBF methods, suggesting it may not be as effective in predicting user preferences.
- **Pearson Correlation Case Modification:** Performs competitively with an overall MAE of 0.7933, indicating that the modification might offer a slight improvement.

Item-Based Collaborative Filtering (IBF):

- **Cosine Similarity:** Provides a good performance with an overall MAE of 0.7904, making it comparable to the UBF methods.

Own Algorithm:

- **Euclidean Distance:** Shows the highest overall MAE of 0.8472, suggesting that this algorithm may not be as accurate in predicting user preferences compared to other methods.

Advantages and Disadvantages:

- **Cosine Similarity (UBF and IBF):**
  - **Advantages:** Simple and effective in capturing item or user similarity.
  - **Disadvantages:** May not handle sparse data well, and its performance might degrade with a large number of users or items.
- **Pearson Correlation (UBF):**
  - **Advantages:** Captures linear relationships between users or items.
  - **Disadvantages:** Sensitive to outliers, and modifications might not always yield significant improvements.
- **Inverse User Frequency (UBF):**
  - **Advantages:** Tries to mitigate the impact of frequently rated items.
  - Disadvantages: Results in a higher overall MAE, indicating that the inverse user frequency might not be a highly effective modification.
- **Euclidean Distance (Own Algorithm):**
  - **Advantages:** Captures the overall distance between user-item pairs.
  - Disadvantages: Yields the highest overall MAE, suggesting it might not perform as well as other algorithms in this specific context.

In conclusion, the Cosine Similarity algorithm, both in user-based and item-based variants, was the overall best-performing method with the lowest Mean Absolute Error (MAE) of 0.7904 for item-based and 0.795 for user-based. These results indicate its effectiveness in capturing user-item similarity and providing accurate

recommendations. On the other hand, the Euclidean Distance-based algorithm that I created consistently exhibits the highest MAE values across all evaluation metrics, making it the least accurate method among those tested. While Euclidean Distance captures the overall distance between user-item pairs, its higher MAE values suggest that it may not be as suitable for this specific recommendation system compared to other collaborative filtering approaches like Cosine Similarity. Therefore, based on the provided results, Cosine Similarity stands out as the preferred choice for accurate collaborative filtering predictions, while Euclidean Distance lags behind in terms of performance.

User-Based Collaborative Filtering Cosine Similarity Code:

```
import math

# Initialize user_data dictionary to store user ratings
user_data = {}

# Read data from the training set
try:
with open("train.txt", "r") as file:
train_users = set() # Set to store unique user IDs in the training set
for line in file:
u, m, r = [int(i) for i in line.split()] # Extract user ID, movie ID, and rating
if u not in train_users:
user_data[u] = {} # Create a dictionary for each user
train_users.add(u)
user_data[u][m] = r # Store the rating for the user and movie
except FileNotFoundError:
print("error: File train.txt not found")

# Read data from test20.txt
try:
with open("test20.txt", "r") as file2:
test_users = set() # Set to store unique user IDs in the test set
for line in file2:
u, m, r = [int(i) for i in line.split()] # Extract user ID, movie ID, and rating
if u not in test_users:
user_data[u] = {} # Create a dictionary for each test user
test_users.add(u)
user_data[u][m] = r # Store the rating for the test user and movie
except FileNotFoundError:
print("error: File test20.txt not found")

# Iterate through test_users and train_users to find shared movies
similarities = {}
testmean = 0
testcount = 0
trainmean = 0
traincount = 0
numRaters = 0 # Number of users that have rated a particular movie
totalusers = len(test_users) + len(train_users)
```

```
for test_user in test_users:
# Set up list of similarities for each test user
similarities[test_user] = []

for train_user in train_users:
test_movies = user_data[test_user].keys()
train = [] # List of train_user's ratings given to shared movies
test = [] # List of test_user's ratings given to shared movies
similarityUsers = set()
similarityUsers.add(test_user)

# The ratings in train[] and test[] correspond to the same movie
for m in test_movies:
if user_data[test_user][m] == 0: # Movie not rated
continue
if m in user_data[train_user]:
train.append(user_data[train_user][m])
test.append(user_data[test_user][m])
trainmean += user_data[train_user][m]
traincount += 1
testmean += user_data[test_user][m]
testcount += 1
numRaters += 1

trainmean = trainmean / traincount
testmean = testmean / testcount
numerator = 0

# Cosine similarity calculation
dot_product = sum(train[i] * test[i] for i in range(len(test)))
test_mag = math.sqrt(sum(test[i] ** 2 for i in range(len(test))))
train_mag = math.sqrt(sum(train[i] ** 2 for i in range(len(train))))

similarity = dot_product / (test_mag * train_mag) if (test_mag * train_mag) != 0 else
0

# Append the calculated similarity and train_user to the similarities list
similarities[test_user].append((similarity, train_user))

# Sort similarities in descending order
similarities[test_user].sort(reverse=True)
```

```
k = 15 # Set the number of similar users to consider

output = []
for test_user in test_users:
test_movies = user_data[test_user].keys()
for m in test_movies:
if user_data[test_user][m] != 0:
continue
count = 0
train_ratings = []
train_similarities = []

# Iterate through similarities for each test user and gather ratings and similarities
for x in similarities[test_user]:
similarity = x[0]
train_user = x[1]

# Skip over training users that did not rate the target movie
if m not in user_data[train_user]:
continue

train_ratings.append(user_data[train_user][m])
train_similarities.append(similarity)
count += 1

if count == k: # Limit the number of similar users considered
break

numerator = sum(train_ratings[i] * train_similarities[i] for i in
range(len(train_ratings)))
denominator = sum(abs(train_similarities[i]) for i in range(len(train_similarities)))

# Calculate the final predicted rating
rating = numerator / denominator if denominator != 0 else 0

# Ensure the rating is between 1 and 5
rating = max(1, min(5, round(rating)))

# Save results to the output list
output.append(f"{test_user} {m} {rating}\n")

# Write output to a file
```

```
with open("result20.txt", "w") as file:
file.writelines(output)
```

User-Based Collaborative Filtering Pearson Correlation Method Code:

```
import math

# Initialize user_data dictionary to store user ratings
user_data = {}

# Read data from the training set
try:
with open("train.txt", "r") as file:
train_users = set() # Set to store unique user IDs in the training set
for line in file:
u, m, r = [int(i) for i in line.split()] # Extract user ID, movie ID, and rating
if u not in train_users:
user_data[u] = {} # Create a dictionary for each user
train_users.add(u)
user_data[u][m] = r # Store the rating for the user and movie
except FileNotFoundError:
print("error: File train.txt not found")

# Read data from test20.txt
try:
with open("test5.txt", "r") as file2:
test_users = set() # Set to store unique user IDs in the test set
for line in file2:
u, m, r = [int(i) for i in line.split()] # Extract user ID, movie ID, and rating
if u not in test_users:
user_data[u] = {} # Create a dictionary for each test user
test_users.add(u)
user_data[u][m] = r # Store the rating for the test user and movie
except FileNotFoundError:
print("error: File test20.txt not found")

# Iterate through test_users and train_users to find shared movies
similarities = {}
testmean = 0
testcount = 0
trainmean = 0
traincount = 0
numRaters = 0 # Number of users that have rated a particular movie
totalusers = len(test_users) + len(train_users)
```

```
for test_user in test_users:
# Set up list of similarities for each test user
similarities[test_user] = []

for train_user in train_users:
test_movies = user_data[test_user].keys()
train = [] # List of train_user's ratings given to shared movies
test = [] # List of test_user's ratings given to shared movies
similarityUsers = set()
similarityUsers.add(test_user)

# The ratings in train[] and test[] correspond to the same movie
for m in test_movies:
if user_data[test_user][m] == 0: # Movie not rated
continue
if m in user_data[train_user]:
train.append(user_data[train_user][m])
test.append(user_data[test_user][m])
trainmean += user_data[train_user][m]
traincount += 1
testmean += user_data[test_user][m]
testcount += 1
numRaters += 1

trainmean = trainmean / traincount
testmean = testmean / testcount
numerator = 0

# Dot product calculation for the numerator
for i in range(len(test)):
numerator += (train[i] - trainmean) * (test[i] - testmean)

test_mag = math.sqrt(sum((test[i] - testmean) ** 2 for i in range(len(test))))
train_mag = math.sqrt(sum((train[i] - trainmean) ** 2 for i in range(len(train))))

denominator = test_mag * train_mag

# Prediction calculation
if denominator == 0:
prediction = 0
else:
prediction = (numerator / denominator)
```

```
# Append the calculated prediction and train_user to the similarities list
similarities[test_user].append((prediction, train_user))

# Sort similarities in descending order
similarities[test_user].sort(reverse=True)

k = 15 # Set the number of similar users to consider

output = []
for test_user in test_users:
test_movies = user_data[test_user].keys()
for m in test_movies:
if user_data[test_user][m] != 0:
continue
count = 0
train_ratings = []
train_similarities = []

# Iterate through similarities for each test user and gather ratings and similarities
for x in similarities[test_user]:
prediction = x[0]
train_user = x[1]

# Skip over training users that did not rate the target movie
if m not in user_data[train_user]:
continue

train_ratings.append(user_data[train_user][m])
train_similarities.append(prediction)
count += 1

if count == k: # Limit the number of similar users considered
break

numerator = sum(train_ratings[i] * train_similarities[i] for i in
range(len(train_ratings)))
denominator = sum(abs(train_similarities[i]) for i in range(len(train_similarities)))

# Calculate the final predicted rating
rating = numerator / denominator if denominator != 0 else 0
```

```
# Ensure the rating is between 1 and 5
rating = max(1, min(5, round(rating)))

# Save results to the output list
output.append(f"{test_user} {m} {rating}\n")

# Write output to a file
with open("result5.txt", "w") as file:
file.writelines(output)
```

User-Based Collaborative Filtering Pearson Correlation Inverse User Frequency Code:

```
import math

# Initialize user_data dictionary to store user ratings
user_data = {}

# Read data from the training set
try:
with open("train.txt", "r") as file:
train_users = set() # Set to store unique user IDs in the training set
for line in file:
u, m, r = [int(i) for i in line.split()] # Extract user ID, movie ID, and rating
if u not in train_users:
user_data[u] = {} # Create a dictionary for each user
train_users.add(u)
user_data[u][m] = r # Store the rating for the user and movie
except FileNotFoundError:
print("error: File train.txt not found")

# Read data from test20.txt
try:
with open("test20.txt", "r") as file2:
test_users = set() # Set to store unique user IDs in the test set
for line in file2:
u, m, r = [int(i) for i in line.split()] # Extract user ID, movie ID, and rating
if u not in test_users:
user_data[u] = {} # Create a dictionary for each test user
test_users.add(u)
user_data[u][m] = r # Store the rating for the test user and movie
except FileNotFoundError:
print("error: File test20.txt not found")

# Iterate through test_users and train_users to find shared movies
similarities = {}
testmean = 0
testcount = 0
trainmean = 0
traincount = 0
numRaters = 0 # Number of users that have rated a particular movie
totalusers = len(test_users) + len(train_users)
```

```
for test_user in test_users:
# Set up list of similarities for each test user
similarities[test_user] = []

for train_user in train_users:
test_movies = user_data[test_user].keys()
train = [] # List of train_user's ratings given to shared movies
test = [] # List of test_user's ratings given to shared movies
similarityUsers = set()
similarityUsers.add(test_user)

# The ratings in train[] and test[] correspond to the same movie
for m in test_movies:
if user_data[test_user][m] == 0: # Movie not rated
continue
if m in user_data[train_user]:
train.append(user_data[train_user][m])
test.append(user_data[test_user][m])
trainmean += user_data[train_user][m]
traincount += 1
testmean += user_data[test_user][m]
testcount += 1
numRaters += 1

trainmean = trainmean / traincount
testmean = testmean / testcount
numerator = 0

# Dot product calculation for the numerator
for i in range(len(test)):
numerator += (train[i] - trainmean) * (test[i] - testmean)

test_mag = math.sqrt(sum((test[i] - testmean) ** 2 for i in range(len(test))))
train_mag = math.sqrt(sum((train[i] - trainmean) ** 2 for i in range(len(train))))

denominator = test_mag * train_mag
iuf = math.log10(numRaters / totalusers)

# Prediction calculation
if denominator == 0:
prediction = 0
else:
```

```
prediction = (numerator / denominator) * iuf

# Append the calculated prediction and train_user to the similarities list
similarities[test_user].append((prediction, train_user))

# Sort similarities in descending order
similarities[test_user].sort(reverse=True)

k = 15 # Set the number of similar users to consider

output = []
for test_user in test_users:
test_movies = user_data[test_user].keys()
for m in test_movies:
if user_data[test_user][m] != 0:
continue
count = 0
train_ratings = []
train_similarities = []

# Iterate through similarities for each test user and gather ratings and similarities
for x in similarities[test_user]:
prediction = x[0]
train_user = x[1]

# Skip over training users that did not rate the target movie
if m not in user_data[train_user]:
continue

train_ratings.append(user_data[train_user][m])
train_similarities.append(prediction)
count += 1

if count == k: # Limit the number of similar users considered
break

numerator = sum(train_ratings[i] * train_similarities[i] for i in
range(len(train_ratings)))
denominator = sum(abs(train_similarities[i]) for i in range(len(train_similarities)))

# Calculate the final predicted rating
rating = numerator / denominator if denominator != 0 else 0
```

```
# Ensure the rating is between 1 and 5
rating = max(1, min(5, round(rating)))

# Save results to the output list
output.append(f"{test_user} {m} {rating}\n")

# Write output to a file
with open("result20.txt", "w") as file:
file.writelines(output)
```

User-Based Collaborative Filtering Pearson Correlation Case Modification Code:

```
import math

# Initialize user_data dictionary to store user ratings
user_data = {}

# Read data from the training set
try:
with open("train.txt", "r") as file:
train_users = set()
for line in file:
u, m, r = [int(i) for i in line.split()]
if u not in train_users:
user_data[u] = {}
train_users.add(u)
user_data[u][m] = r
except FileNotFoundError:
print("error: File train.txt not found")

# Read data from test20.txt
try:
with open("test20.txt", "r") as file2:
test_users = set()
for line in file2:
u, m, r = [int(i) for i in line.split()]
if u not in test_users:
user_data[u] = {}
test_users.add(u)
user_data[u][m] = r
except FileNotFoundError:
print("error: File test20.txt not found")

# Iterate through test_users and train_users to find shared movies
similarities = {}
testmean = 0
testcount = 0
trainmean = 0
traincount = 0
numRaters = 0
totalusers = len(test_users) + len(train_users)
```

```
for test_user in test_users:
similarities[test_user] = []

for train_user in train_users:
test_movies = user_data[test_user].keys()
train = []
test = []
similarityUsers = set()
similarityUsers.add(test_user)

for m in test_movies:
if user_data[test_user][m] == 0:
continue
if m in user_data[train_user]:
train.append(user_data[train_user][m])
test.append(user_data[test_user][m])
trainmean += user_data[train_user][m]
traincount += 1
testmean += user_data[test_user][m]
testcount += 1
numRaters += 1

trainmean = trainmean / traincount
testmean = testmean / testcount
numerator = 0

for i in range(len(test)):
numerator += (train[i] - trainmean) * (test[i] - testmean)

test_mag = math.sqrt(sum((test[i] - testmean) ** 2 for i in range(len(test))))
train_mag = math.sqrt(sum((train[i] - trainmean) ** 2 for i in range(len(train))))

denominator = test_mag * train_mag

if denominator == 0:
prediction = 0
else:
prediction = (numerator / denominator)

# Case modification: Consider user and item biases
prediction = prediction + (testmean + trainmean) / 2
```

```
similarities[test_user].append((prediction, train_user))

similarities[test_user].sort(reverse=True)

k = 15

output = []
for test_user in test_users:
test_movies = user_data[test_user].keys()
for m in test_movies:
if user_data[test_user][m] != 0:
continue
count = 0
train_ratings = []
train_similarities = []

for x in similarities[test_user]:
prediction, train_user = x

if m not in user_data[train_user]:
continue

train_ratings.append(user_data[train_user][m])
train_similarities.append(prediction)
count += 1

if count == k:
break

numerator = sum(train_ratings[i] * train_similarities[i] for i in
range(len(train_ratings)))
denominator = sum(abs(train_similarities[i]) for i in range(len(train_similarities)))

rating = numerator / denominator if denominator != 0 else 0

rating = max(1, min(5, round(rating)))

output.append(f"{test_user} {m} {rating}\n")

with open("result20_modified.txt", "w") as file:
file.writelines(output)
```

Item-Based Collaborative Filtering Cosine Similarity Code:

```
import math

# Initialize user_data dictionary to store user ratings
user_data = {}

# Read data from the training set
try:
with open("train.txt", "r") as file:
train_users = set() # Set to store unique user IDs in the training set
for line in file:
u, m, r = [int(i) for i in line.split()] # Extract user ID, movie ID, and rating
if u not in train_users:
user_data[u] = {} # Create a dictionary for each user
train_users.add(u)
user_data[u][m] = r # Store the rating for the user and movie
except FileNotFoundError:
print("error: File train.txt not found")

# Read data from test20.txt
try:
with open("test20.txt", "r") as file2:
test_users = set() # Set to store unique user IDs in the test set
for line in file2:
u, m, r = [int(i) for i in line.split()] # Extract user ID, movie ID, and rating
if u not in test_users:
user_data[u] = {} # Create a dictionary for each test user
test_users.add(u)
user_data[u][m] = r # Store the rating for the test user and movie
except FileNotFoundError:
print("error: File test20.txt not found")

# Iterate through test_users and train_users to find shared movies
similarities = {}
testmean = 0
testcount = 0
trainmean = 0
traincount = 0
numRaters = 0 # Number of users that have rated a particular movie
totalusers = len(test_users) + len(train_users)
```

```
for test_user in test_users:
# Set up list of similarities for each test user
similarities[test_user] = []

for train_user in train_users:
test_movies = user_data[test_user].keys()
train = [] # List of train_user's ratings given to shared movies
test = [] # List of test_user's ratings given to shared movies
similarityUsers = set()
similarityUsers.add(test_user)

# The ratings in train[] and test[] correspond to the same movie
for m in test_movies:
if user_data[test_user][m] == 0: # Movie not rated
continue
if m in user_data[train_user]:
train.append(user_data[train_user][m])
test.append(user_data[test_user][m])
trainmean += user_data[train_user][m]
traincount += 1
testmean += user_data[test_user][m]
testcount += 1
numRaters += 1

trainmean = trainmean / traincount
testmean = testmean / testcount
numerator = 0

# Cosine similarity calculation
dot_product = sum(train[i] * test[i] for i in range(len(test)))
test_mag = math.sqrt(sum(test[i] ** 2 for i in range(len(test))))
train_mag = math.sqrt(sum(train[i] ** 2 for i in range(len(train))))

similarity = dot_product / (test_mag * train_mag) if (test_mag * train_mag) != 0 else
0

# Append the calculated similarity and train_user to the similarities list
similarities[test_user].append((similarity, train_user))

# Sort similarities in descending order
similarities[test_user].sort(reverse=True)
```

```
k = 35 # Set the number of similar users to consider

output = []
for test_user in test_users:
test_movies = user_data[test_user].keys()
for m in test_movies:
if user_data[test_user][m] != 0:
continue
count = 0
train_ratings = []
train_similarities = []

# Iterate through similarities for each test user and gather ratings and similarities
for x in similarities[test_user]:
similarity = x[0]
train_user = x[1]

# Skip over training users that did not rate the target movie
if m not in user_data[train_user]:
continue

train_ratings.append(user_data[train_user][m])
train_similarities.append(similarity)
count += 1

if count == k: # Limit the number of similar users considered
break

numerator = sum(train_ratings[i] * train_similarities[i] for i in
range(len(train_ratings)))
denominator = sum(abs(train_similarities[i]) for i in range(len(train_similarities)))

# Calculate the final predicted rating
rating = numerator / denominator if denominator != 0 else 0

# Ensure the rating is between 1 and 5
rating = max(1, min(5, round(rating)))

# Save results to the output list
output.append(f"{test_user} {m} {rating}\n")

# Write output to a file
```

```
with open("result20.txt", "w") as file:
file.writelines(output)
```

Own Algorithm Code:

```
import math

# Initialize user_data dictionary to store user ratings
user_data = {}

# Read data from the training set
try:
with open("train.txt", "r") as file:
train_users = set() # Set to store unique user IDs in the training set
for line in file:
u, m, r = [int(i) for i in line.split()] # Extract user ID, movie ID, and rating
if u not in train_users:
user_data[u] = {} # Create a dictionary for each user
train_users.add(u)
user_data[u][m] = r # Store the rating for the user and movie
except FileNotFoundError:
print("error: File train.txt not found")

# Read data from test20.txt
try:
with open("test20.txt", "r") as file2:
test_users = set() # Set to store unique user IDs in the test set
for line in file2:
u, m, r = [int(i) for i in line.split()] # Extract user ID, movie ID, and rating
if u not in test_users:
user_data[u] = {} # Create a dictionary for each test user
test_users.add(u)
user_data[u][m] = r # Store the rating for the test user and movie
except FileNotFoundError:
print("error: File test20.txt not found")

# Iterate through test_users and train_users to find shared movies
similarities = {}
testmean = 0
testcount = 0
trainmean = 0
traincount = 0
numRaters = 0 # Number of users that have rated a particular movie
totalusers = len(test_users) + len(train_users)
```

```
for test_user in test_users:
# Set up list of similarities for each test user
similarities[test_user] = []

for train_user in train_users:
test_movies = user_data[test_user].keys()
train = [] # List of train_user's ratings given to shared movies
test = [] # List of test_user's ratings given to shared movies
similarityUsers = set()
similarityUsers.add(test_user)

# The ratings in train[] and test[] correspond to the same movie
for m in test_movies:
if user_data[test_user][m] == 0: # Movie not rated
continue
if m in user_data[train_user]:
train.append(user_data[train_user][m])
test.append(user_data[test_user][m])
trainmean += user_data[train_user][m]
traincount += 1
testmean += user_data[test_user][m]
testcount += 1
numRaters += 1

trainmean = trainmean / traincount
testmean = testmean / testcount
numerator = 0

# Euclidean distance calculation
euclidean_distance = math.sqrt(sum((train[i] - test[i]) ** 2 for i in
range(len(test))))

# Append the calculated distance and train_user to the similarities list
similarities[test_user].append((euclidean_distance, train_user))

# Sort similarities in ascending order (lower distance is better)
similarities[test_user].sort()

k = 15 # Set the number of similar users to consider

output = []
for test_user in test_users:
```

```
test_movies = user_data[test_user].keys()
for m in test_movies:
if user_data[test_user][m] != 0:
continue
count = 0
train_ratings = []
train_distances = []

# Iterate through similarities for each test user and gather ratings and distances
for x in similarities[test_user]:
distance = x[0]
train_user = x[1]

# Skip over training users that did not rate the target movie
if m not in user_data[train_user]:
continue

train_ratings.append(user_data[train_user][m])
train_distances.append(distance)
count += 1

if count == k: # Limit the number of similar users considered
break

numerator = sum(train_ratings[i] / (train_distances[i] + 1e-10) for i in
range(len(train_ratings)))
denominator = sum(1 / (train_distances[i] + 1e-10) for i in
range(len(train_distances)))

# Calculate the final predicted rating
rating = numerator / denominator if denominator != 0 else 0

# Ensure the rating is between 1 and 5
rating = max(1, min(5, round(rating)))

# Save results to the output list
output.append(f"{test_user} {m} {rating}\n")

# Write output to a file
with open("result20.txt", "w") as file:
file.writelines(output)
```

Screenshots of MAEs:

UBF Cosine:

Hello, Waldie, Madeleine

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.837689133425034
MAE of GIVEN 10 : 0.782
MAE of GIVEN 20 : 0.769653708883959
OVERALL MAE : 0.795025447381382

You have already submitted 22 times.

GOOD LUCK!

UBF Pearson:

Hello, Waldie, Madeleine

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.838814555458297
MAE of GIVEN 10 : 0.782
MAE of GIVEN 20 : 0.769460789042153
OVERALL MAE : 0.795312756526022

You have already submitted 17 times.

GOOD LUCK!

UBF IUF Pearson:

Hello, Waldie, Madeleine

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.839814930598975
MAE of GIVEN 10 : 0.780333333333333
MAE of GIVEN 20 : 0.773319185878268
OVERALL MAE : 0.79687243473978

You have already submitted 15 times.

GOOD LUCK!

UBF Case Based Pearson:

Hello, Waldie, Madeleine

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.834312867325247
MAE of GIVEN 10 : 0.777166666666667
MAE of GIVEN 20 : 0.770907687855696
OVERALL MAE : 0.793260548350025

You have already submitted 18 times.

GOOD LUCK!

IBF Cosine:

Hello, Waldie, Madeleine

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.830811554332875
MAE of GIVEN 10 : 0.78
MAE of GIVEN 20 : 0.76531301244333
OVERALL MAE : 0.790428501067148

You have already submitted 24 times.

GOOD LUCK!

My Algorithm:

Hello, Waldie, Madeleine

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.858321870701513
MAE of GIVEN 10 : 0.839666666666667
MAE of GIVEN 20 : 0.842866788849233
OVERALL MAE : 0.847151535051716

You have already submitted 25 times.

GOOD LUCK!