**Recommender System Assignment 02**

**Abdullah Islam 21k-4896**

**Queston 1:**

**Part1**

#### a) MovieLens

File: u.data file  
Used columns in data:

* User ID
* Movie ID
* Rating (1 to 5)
* Timestamp
* 4 and 5 stars are frequent
* 1 star is rare
* Some users rated 500+ movies
* Many users rated < 10 movies
* No missing values
* Dataset is balanced and clean

#### Film Trust

Columns:

1. User ID
2. Item ID
3. Rating (0.5 to 4.0)
4. Timestamp

* Ratings are between 0.5 and 4
* Most ratings are 3+ or above
* Most ratings are 3 and above
* Some users rated dozens of movies; others rated only 1 or 2
* No null values
* Ratings are not in 1–5 scale

#### Yahoo

Columns:

1. User ID
2. Movie ID
3. Rating (1–5)
4. Date (optional)

* Majority ratings are 4 and 5 stars
* Some movies are rated 100s of times
* Many movies have 1 rating (low freq.)

**Distribution**

* Mostly clean
* No missing values
* Large dataset with more sparsity

#### Book Crossing

Columns:

1. User ID
2. ISBN (Book ID)
3. Rating (0–10)

* Most ratings lie below 5 and 10
* Many books have 1 rating
* Some books have 500+ ratings
* High sparsity (many users rated only once)
* Null values possible in book metadata
* Requires cleanin

**Part2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Paper Title** | **Authors** | **Year** | **Use** |

|  |  |  |  |
| --- | --- | --- | --- |
| **MovieLens** | | | |
| MovieLens Beliefs Dataset: Collecting Pre-Choice Data for  Recommender Systems | Guy Aridor, Duarte Goncalves, et al | 2024 | User beliefs + ratings to analyze  recommendation behavior |
| A Re-visit of the Popularity Baseline in Recommender Systems | Yitong Ji, Aixin Sun, et al. | 2020 | Tested effectiveness of popularity-based recommendations |
| **FilmTrust** | | | |
| FilmTrust: Movie Recommendations from Social  Networks | Jennifer Golbeck | 2006 | Introduced trust- based  recommendation system using  FilmTrust |
| Weighted Item Ranking for Matrix Factorization | Haiyang Zhang, Ivan Ganchev, et al. | 2017 | Enhanced accuracy using weighted item ranking on FilmTrust |
| **Yahoo** | | | |
| A Re-visit of the Popularity Baseline | Yitong Ji, Aixin Sun, et al. | 2020 | Compared baseline methods including Yahoo dataset |
| **Book Crossing** | | | |
| Social Network Analysis on Book- Crossing | Mohammed Al-Taie, Seifedine Kadry | 2012 | Analyzed user-book networks using SNA |

**Question 2:**

import numpy as np

from collections import defaultdict

def nbcf\_algorithm(ratings, num\_users, num\_items, possible\_ratings, alpha=1.0): print("Starting NBCF Algorithm...")

pup = defaultdict(lambda: defaultdict(lambda: alpha)) pip = defaultdict(lambda: defaultdict(lambda: alpha))

cup = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: alpha))))

cip = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: alpha))))

uc = defaultdict(lambda: defaultdict(lambda: 0)) ic = defaultdict(lambda: defaultdict(lambda: 0))

ijc = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: 0))) uvc = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: 0))) num\_ratings = len(possible\_ratings)

for u in range(num\_users): for y in possible\_ratings:

uc[u][y] = num\_ratings \* alpha for i in range(num\_items):

for y in possible\_ratings:

ic[i][y] = num\_ratings \* alpha for i in range(num\_items):

for j in range(num\_items): for y in possible\_ratings:

ijc[i][j][y] = num\_ratings \* alpha for u in range(num\_users):

for v in range(num\_users): for y in possible\_ratings:

uvc[u][v][y] = num\_ratings \* alpha

print("Initialized probability and counter dictionaries.") for u in range(num\_users):

user\_ratings = [(i, r) for (user, i, r) in ratings if user == u and r != 0] if not user\_ratings:

continue

for i, y in user\_ratings: uc[u][y] += 1

pup[u][y] = (uc[u][y]) / (sum(uc[u].values()) + num\_ratings \* alpha) ic[i][y] += 1

pip[i][y] = (ic[i][y]) / (sum(ic[i].values()) + num\_ratings \* alpha) for j, k in user\_ratings:

if j != i:

ijc[i][j][y] += 1

cip[j][k][i][y] = (ijc[i][j][y] \* cip[j][k][i][y] + 1) / (ijc[i][j][y] + num\_ratings \* alpha) users\_rated\_i = [(v, r) for (v, item, r) in ratings if item == i and r != 0]

for v, k in users\_rated\_i: if v != u:

uvc[u][v][y] += 1

cup[v][k][u][y] = (uvc[u][v][y] \* cup[v][k][u][y] + 1) / (uvc[u][v][y] + num\_ratings \*

alpha)

if u % 100 == 0:

print(f"Processed user {u}/{num\_users}") print("NBCF Algorithm completed.")

return pup, pip, cup, cip

def main(): ratings = [

(0, 1, 1), (0, 2, 2), (0, 3, 2), (0, 4, 5), (0, 6, 4), (0, 7, 3), (0, 8, 5),

(1, 0, 1), (1, 1, 5), (1, 2, 3), (1, 4, 2), (1, 5, 3), (1, 6, 4), (1, 7, 3),

(2, 0, 1), (2, 1, 1), (2, 2, 2), (2, 4, 2), (2, 5, 4), (2, 6, 4), (2, 7, 5),

(3, 0, 3), (3, 1, 2), (3, 2, 2), (3, 3, 3), (3, 5, 1), (3, 6, 3), (3, 7, 2),

(4, 0, 5), (4, 1, 1), (4, 2, 5), (4, 3, 5), (4, 4, 4), (4, 5, 4), (4, 6, 5), (4, 7, 2)

]

num\_users = 5

num\_items = 9

possible\_ratings = [1, 2, 3, 4, 5]

alpha = 1.0

print("Running NBCF on synthetic dataset...")

pup, pip, cup, cip = nbcf\_algorithm(ratings, num\_users, num\_items, possible\_ratings, alpha) print("\nSample Prior Probabilities (pup):")

for u in range(num\_users): for y in possible\_ratings:

if pup[u][y] != alpha:

print(f"P(r\_{u} = {y}) = {pup[u][y]:.4f}") print("\nSample Prior Probabilities (pip):") for i in range(num\_items):

for y in possible\_ratings: if pip[i][y] != alpha:

print(f"P(r\_{i} = {y}) = {pip[i][y]:.4f}")

print("\nSample Conditional Probabilities (cup):") for u in range(num\_users):

for v in range(num\_users): for y in possible\_ratings:

for k in possible\_ratings:

if cup[v][k][u][y] != alpha:

print(f"P(r\_{v} = {k} | r\_{u} = {y}) = {cup[v][k][u][y]:.4f}") print("\nSample Conditional Probabilities (cip):")

for i in range(num\_items): for j in range(num\_items):

for y in possible\_ratings: for k in possible\_ratings:

if cip[j][k][i][y] != alpha:

print(f"P(r\_{j} = {k} | r\_{i} = {y}) = {cip[j][k][i][y]:.4f}")

if name == " main ": main()

**Question 03**

# Part1

In this study, path coefficient analysis was used to check how one thing affects another. Like, if people think the system works well (PP), they start trusting it (TR). Then, if they trust the system, they also trust the products it shows (TP). And if they trust the products, they might choose from them (IC).It also showed that performance (PP) doesn’t directly affect the final

choice (IC). Trust is in the middle and makes the real difference. Also, ChatGPT (BingChat) made people trust more than Amazon, maybe because it feels more fair and not selling anything. One more thing was checked — brand awareness. It helped when people didn’t know the brand well (low or medium), but didn’t matter when they already knew the brand. So, overall, this analysis helped understand that trust is very important in recommendation systems, and ChatGPT did a better job in building that trust.

# Part 2

Latent means analysis was used to compare Amazon and BingChat. It looked at which system makes people more likely to choose the suggested products (IC). The result showed that

BingChat had higher score than Amazon. The difference was 0.275, which means people liked ChatGPT suggestions more. Maybe because it gives better and more useful recommendations by understanding user needs. This tells us that ChatGPT works better than Amazon in helping users decide what to pick. It also helps system makers know what users like more.

# Part3

This part explains how trust moves from system to product, and how brand awareness (BA) changes this effect.First, people trust the recommender system (TR), then that trust goes to the products (TP), and then they decide to choose them (IC). This is called **trust transfer**.Now, **brand awareness** plays a role. When people don’t know the brand much (low or medium BA), then this trust path (PP → TR → TP → IC) works well. But when brand is already famous (high BA), people already trust it, so recommender system doesn’t matter much.So, this means AI systems like ChatGPT help more when brand is not well-known. They help people trust new products and make better choices.

# Part4

Figure 2 shows two scatter plots for Amazon and BingChat. It explains how trust in recommender (TR) is linked to trust in products (TP), and how this changes with brand awareness (BA).

On X-axis we have TR, and on Y-axis TP. The points are shown with different signs for brand awareness:

* "○" for low BA
* "□" for medium BA
* "x" for high BA

Solid lines mean strong connection, dashed lines mean weak or no connection. At low and

medium BA, lines are solid, so TR and TP are strongly related. But at high BA, lines are dashed , meaning people already trust the product, so recommender doesn’t help much.

Both Amazon and BingChat show same pattern, but BingChat works better overall because it gives more useful and smart suggestions.

This figure helps show that AI recommenders like ChatGPT help more when the brand is not popular.

**Question 04**

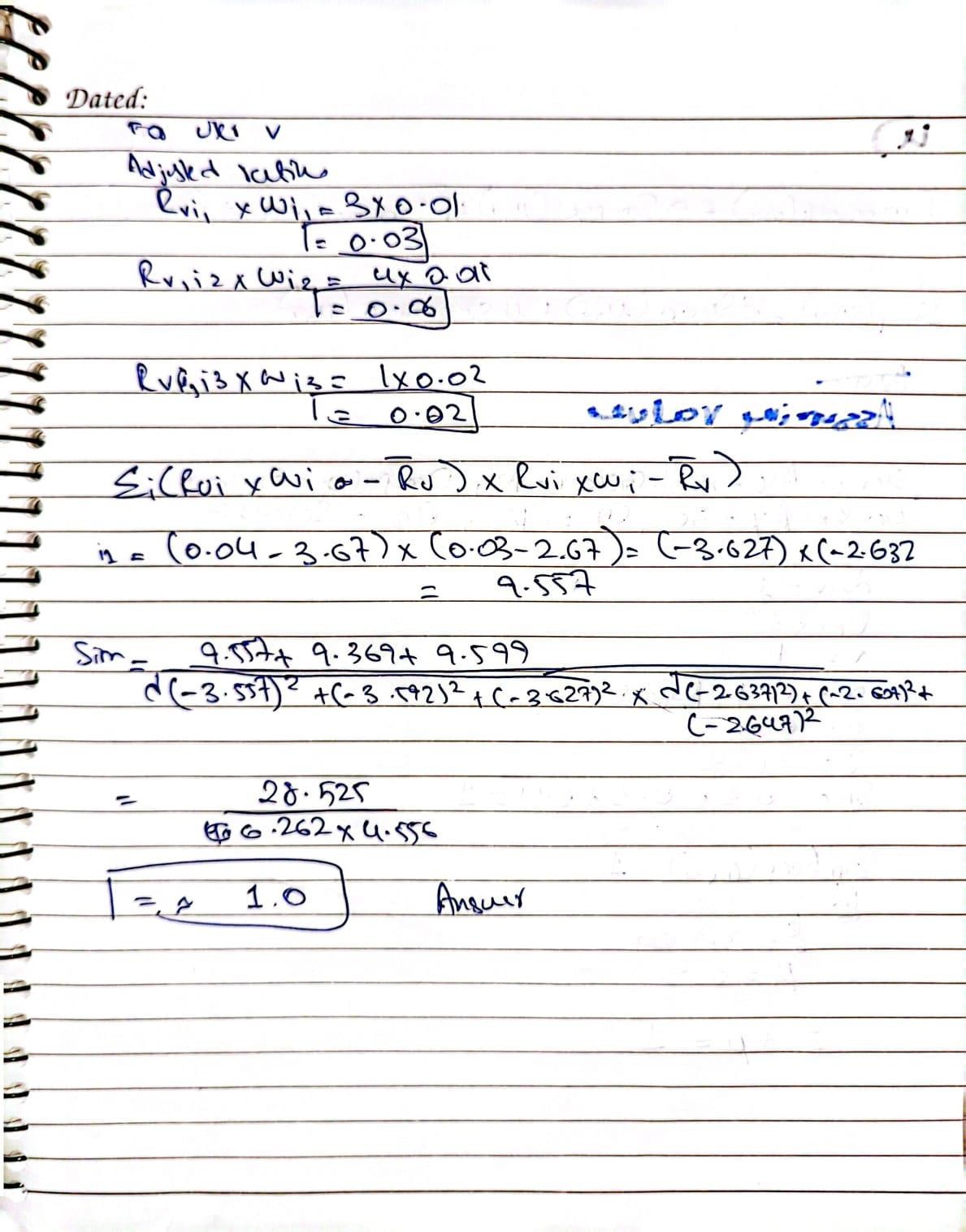
# Part 1

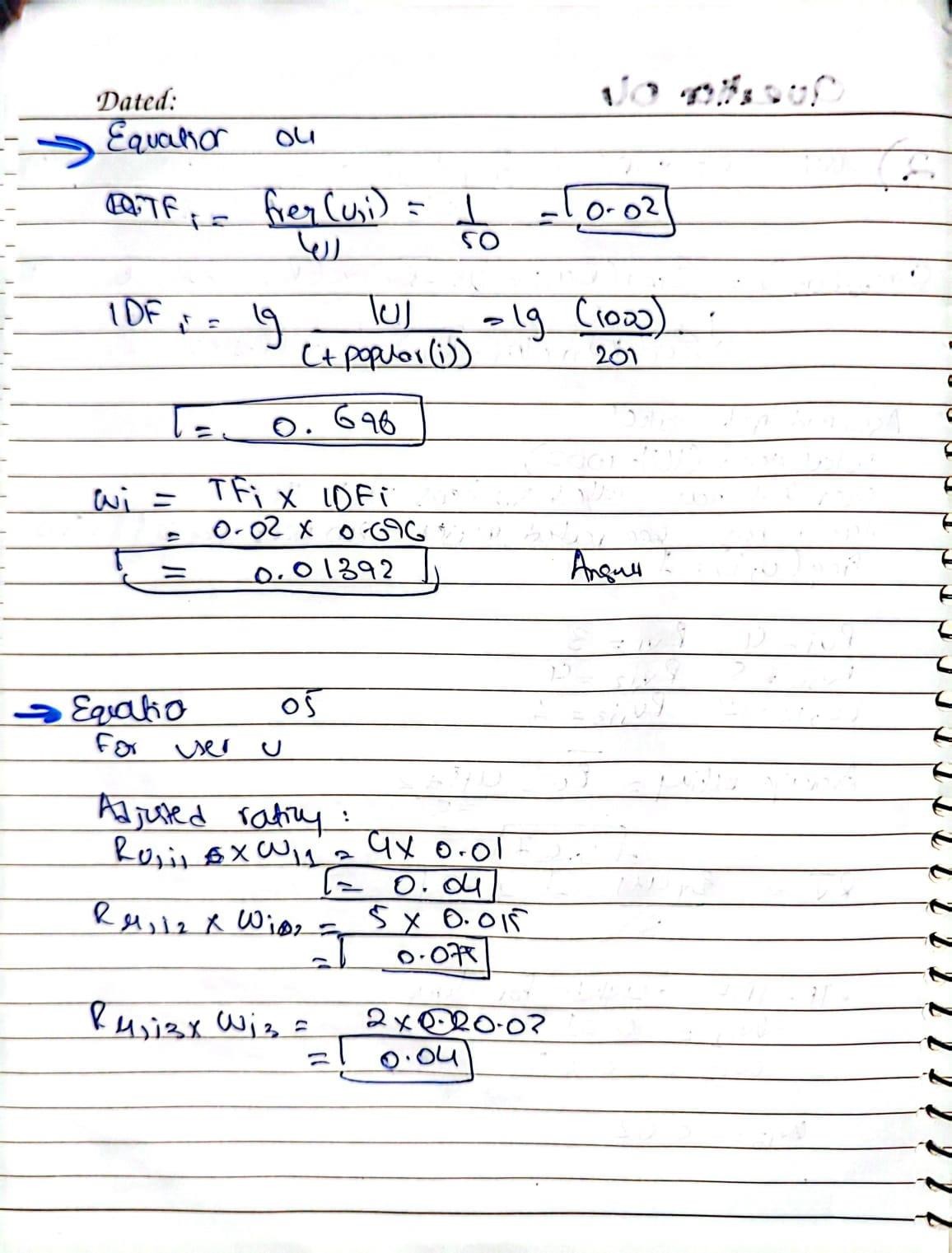
The Improved TF-IDF based method helps fix a common problem in collaborative filtering (CF), where popular items get too much focus. For example, movies like *Shawshank Redemption* are rated by many people, so they affect user similarity more than they should. This can make recommendations less accurate. To fix this, TF-IDF is used. It gives more weight to less popular items, because those better reflect a user’s actual taste. TF checks how often a user interacted with an item, and IDF reduces the value if many users rated that item. Their product gives a new weight, which is used in user similarity calculations. By doing this, less common items have more impact, and recommendations become more personal. Tests on MovieLens showed this

method gives better results than the usual CF approach, with lower errors like MAE and RMSE.

# Part2

# 





**Part3**

The User Characteristics Model is made to improve the traditional collaborative filtering (CF)

method. Traditional CF mainly looks at user ratings, but it ignores other things, like age, gender, and occupation, which can really affect what users like. By using these characteristics, it helps solve problems like the cold start problem. This problem happens when new users don’t have enough ratings for recommendations.

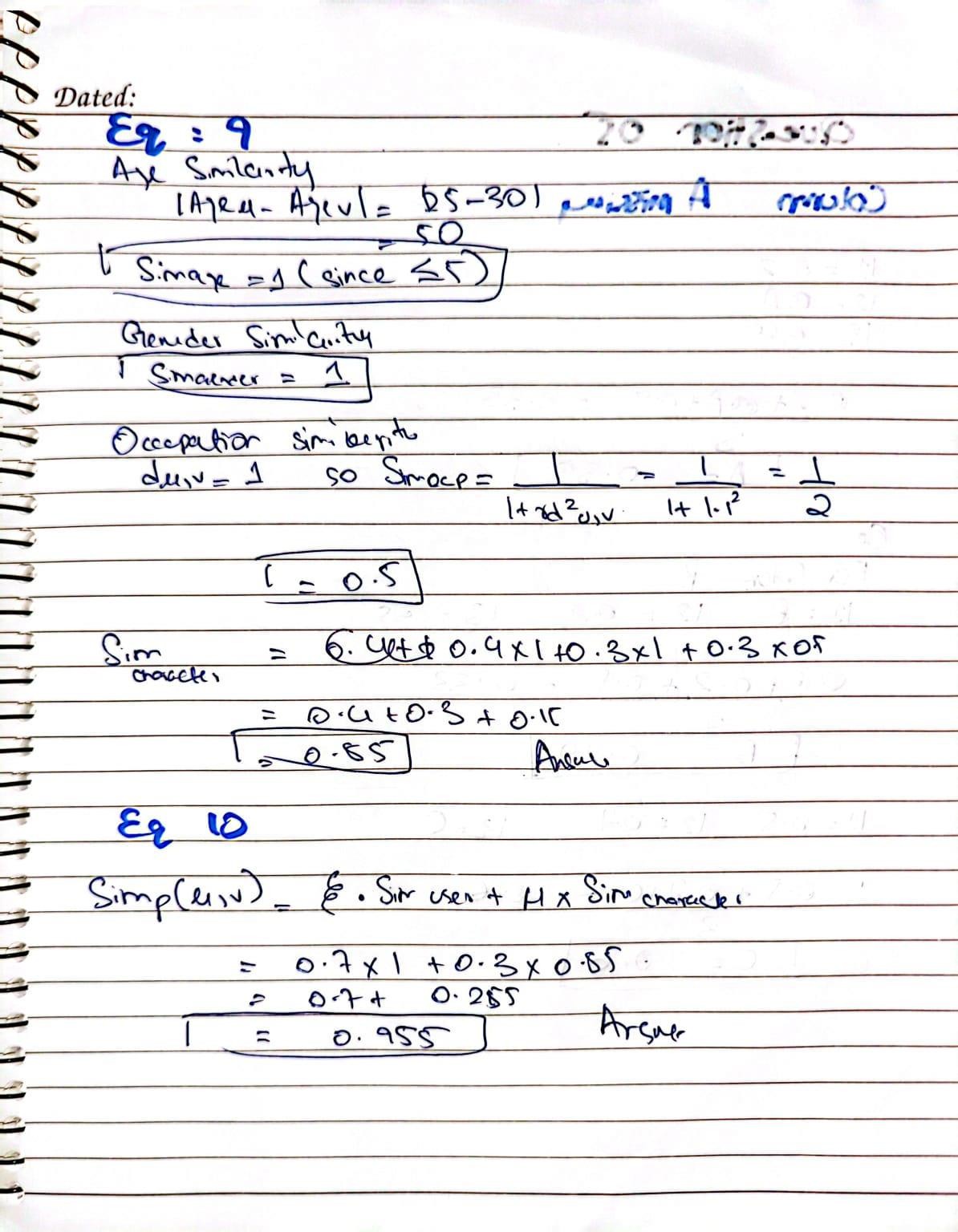
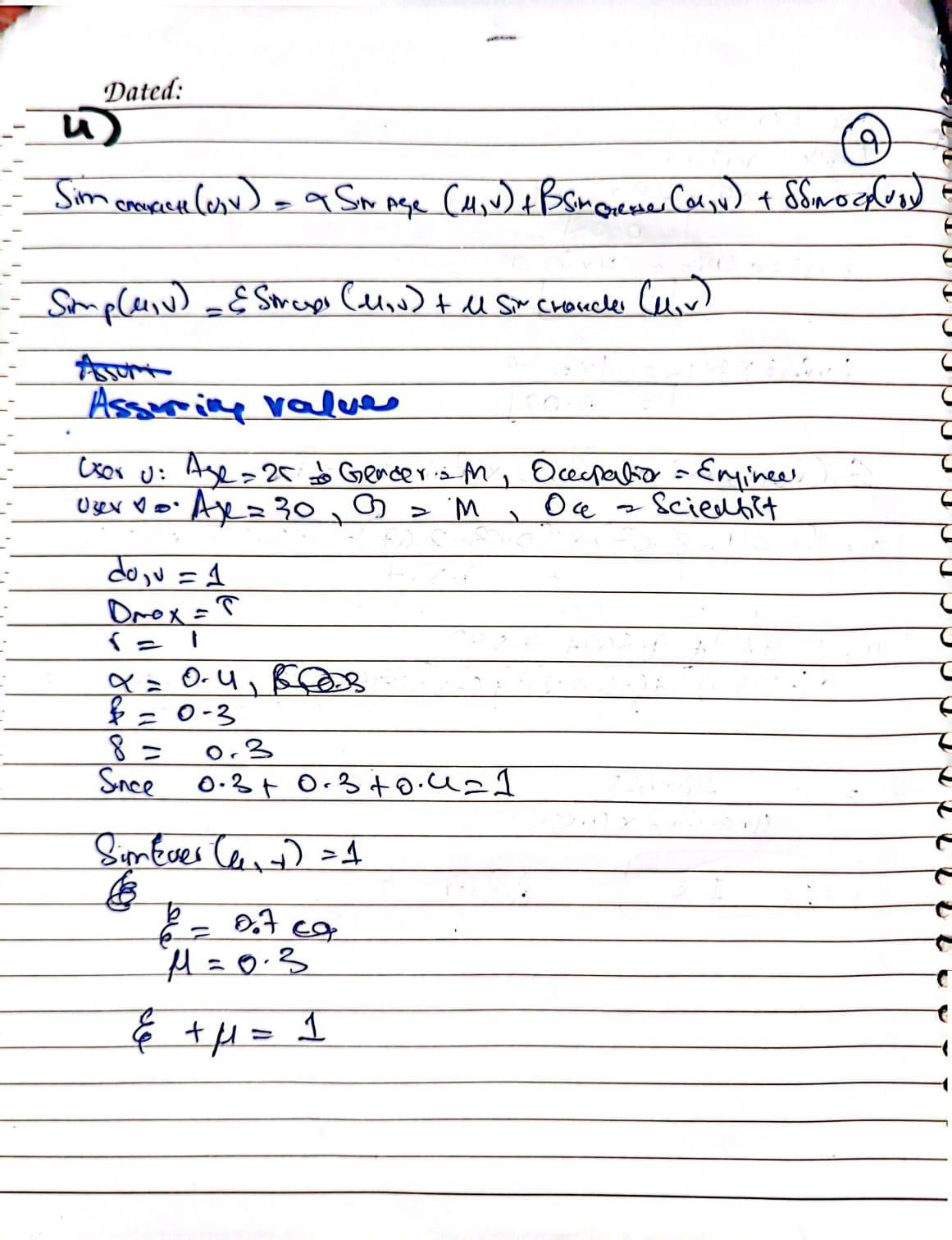
* Age Similarity: This method uses a fuzzy function to check if two users are similar in age. If they are close in age (within 5 years), they are considered very similar. The similarity decreases the more their ages differ, and it becomes zero if the age difference is greater than 25 years.
* Occupation Similarity: This is based on how similar two users’ jobs are. The method uses a special formula to measure this similarity based on the job categories in a classification tree.
* Gender Similarity: This one is simple. If two users are of the same gender, they are considered similar (similarity = 1). If they are of different genders, they are not (similarity

= 0).

These three types of similarities (age, gender, and occupation) are combined using weights. The weights are assigned based on how important each characteristic is. This helps in calculating overall similarity between users.

The User Characteristics Model helps make better recommendations, especially when there are not many ratings. It helps with new users who don’t have much rating data yet by looking at their age, gender, and occupation. This way, it improves the accuracy of the recommendations and solves problems where there is not enough data.

# Part4



**Part5**

def optimal\_weight\_search(): min\_mae = float('inf')

min\_rmse = float('inf') best\_xi = None best\_mu = None

for mu in [i / 20 for i in range(0, 21)]: xi = 1 - mu

predicted\_ratings = predict\_ratings(xi, mu) actual\_ratings = get\_actual\_ratings()

mae, rmse = calculate\_errors(predicted\_ratings, actual\_ratings)

if mae < min\_mae or (mae == min\_mae and rmse < min\_rmse): min\_mae = mae

min\_rmse = rmse best\_xi = xi best\_mu = mu

return best\_xi, best\_mu, min\_mae, min\_rmse

best\_xi, best\_mu, best\_mae, best\_rmse = optimal\_weight\_search() print(f"Optimal ξ: {best\_xi}, μ: {best\_mu}, MAE: {best\_mae}, RMSE: {best\_rmse}")

**Question 05**

