

Privrank for Social Media

Submitted By:

Ravi Teja Reddy Dodda (016693196)

Tanuja Reddy Maligireddy (016715348)

Harika Satti (016260504)

Submitted To:

Prof.

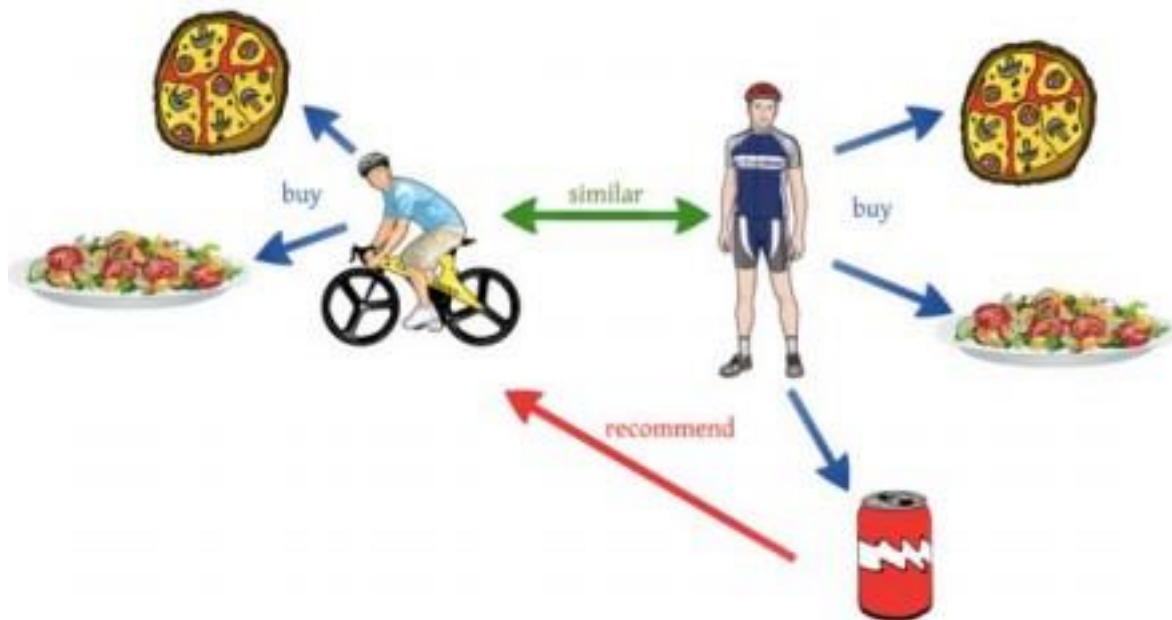
Chandrasekar Vuppalapati

Priv-Rank for Social Media

Collaborative Filtering:

Most websites like Amazon, YouTube, and Netflix use collaborative filtering as a part of their sophisticated recommendation systems.

Collaborative filtering is based on the idea that similar users tend to have similar tastes.



Collaborative filtering systems have many forms, but many common systems can be reduced to two steps:

1. Look for users who share the same rating patterns with the active user (the user whom the prediction is for).

2. Use the ratings from those like-minded users found in step 1 to calculate a prediction for the active user

EXPLANATION:

1. Loading the dataset

We currently have a dataset of users containing 4 things. UserID, gender, age and posts they liked.

For privacy reasons, we did not take fields which could be used to back track to the original user like Name, DOB, etc.

2. Looking for similar users / Pre-processing the dataset

Firstly, we normalize the data using Standard Scalar.

Then used the PCA (Principle Component Analysis) for dimensionality reduction from 3 to 1.

And then we created the similarity matrix, or we can say correlation matrix.

```
1 0.8858748713068504 0.18655321615857778 0.16298502474900822 0.17916769966444834
0 0.8858748713068504 1 0.5209461793376426 0.4337...
1 0.18655321615857778 0.5209461793376426 1 0.440...
2 0.16298502474900822 0.43375864429872935 0.4402...
3 0.17916769966444834 0.9871659981611819 0.19675...
4 0.3011344088809562 0.6420260046728776 0.901507...
5 0.4748585089133869 0.5772489055053566 0.734660...
6 0.16676561513505195 0.030987077991099676 0.048...
7 0.6272130696695033 0.3798951759044207 0.398446...
8 0.19705911003745458 0.8509605270950726 0.47385...
9 0.4135983413862216 0.5483055695315037 0.717238
```

3. Recommend person to user

| |  |  |  |  |
|---|---|---|---|---|
|  |  |  |  |  |
|  | |  |  |  |
|  |  |  |  | |
|  |  | |  | |
|  |  |  |  |  |

Now that we have users that are most similar to us, we need to find the likeability of each post not seen by our current user. For this we use the following likeliness equation:

$$\sum_{i=1}^n x_i \cdot r_i$$

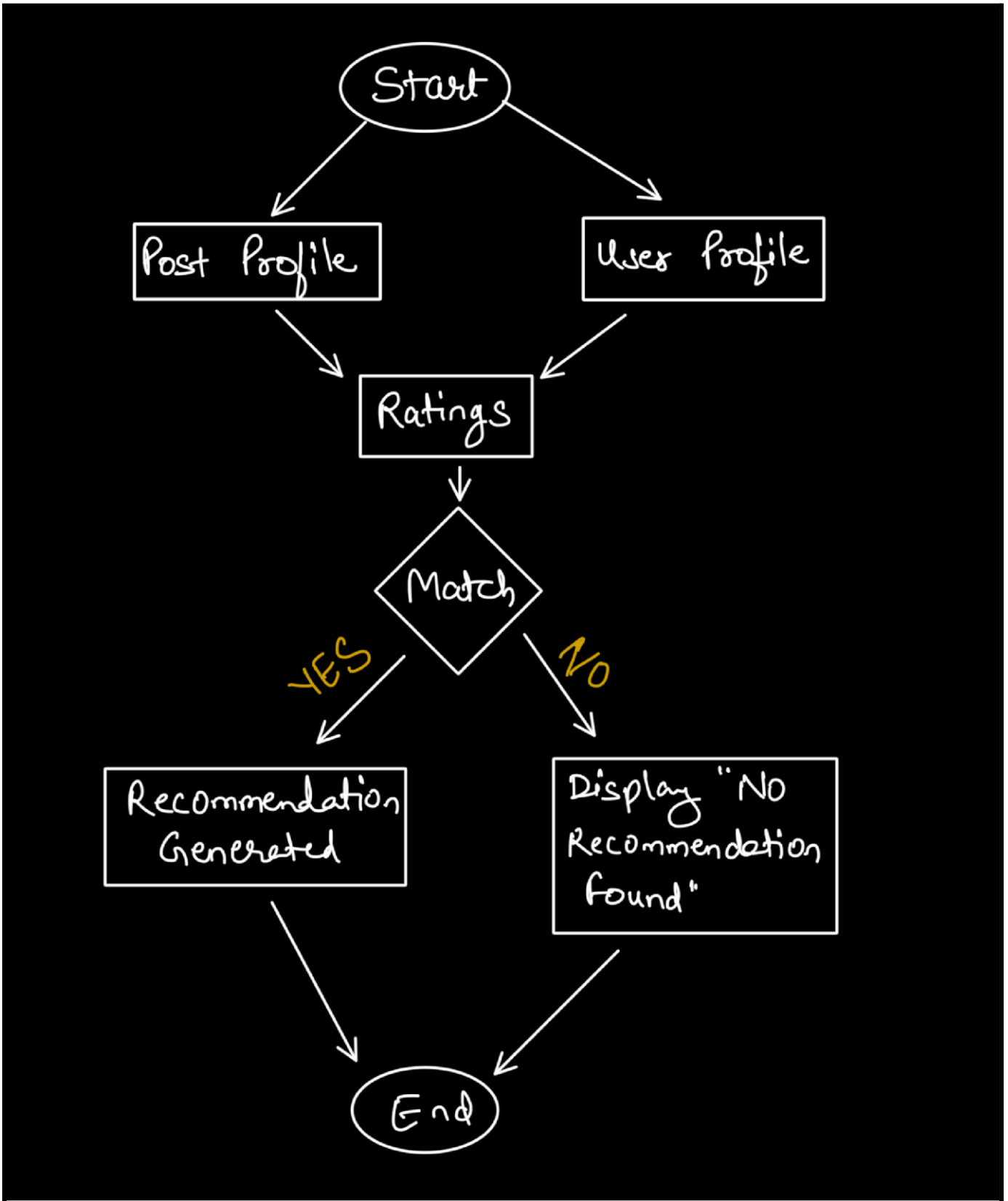
Here “x” is the likeability value of the current user, “i” is the user and “r” is the likeability of that person/user. We perform the above operation on every not visited post and then reverse sort the result. The first value i.e., post with the highest value is the one which is most likely that the user will like.

What we did for Privacy?

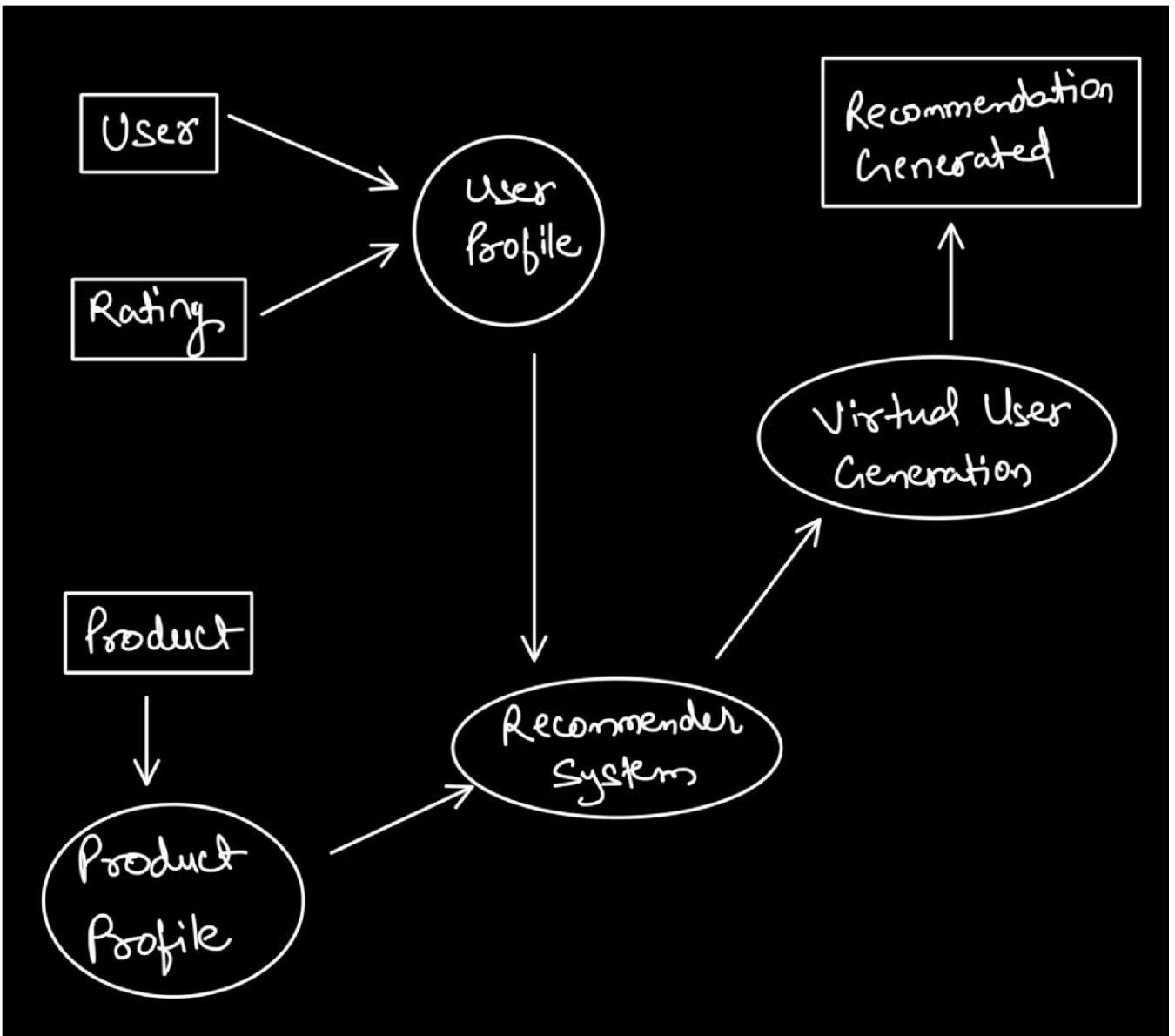
We obfuscated the data 10 - 20% like converting the likes to dislikes and vice versa. So, we could not easily backtrack to the user from their choices. This process was done on the user's end.

This also allows to prevent inference attacks such as based on user preference we might be able to infer their gender etc., thereby increasing privacy as done in PrivRank research paper.

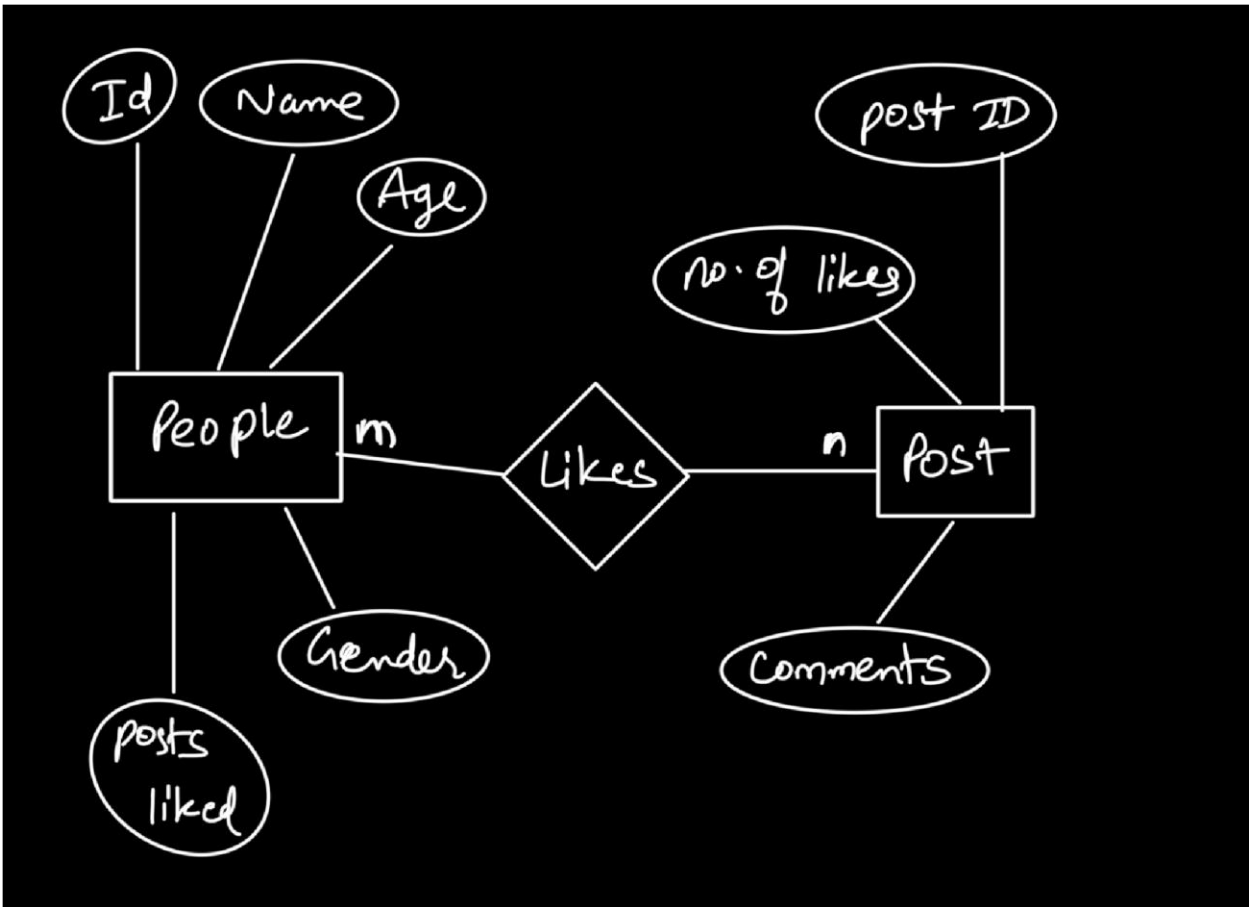
Flowchart:



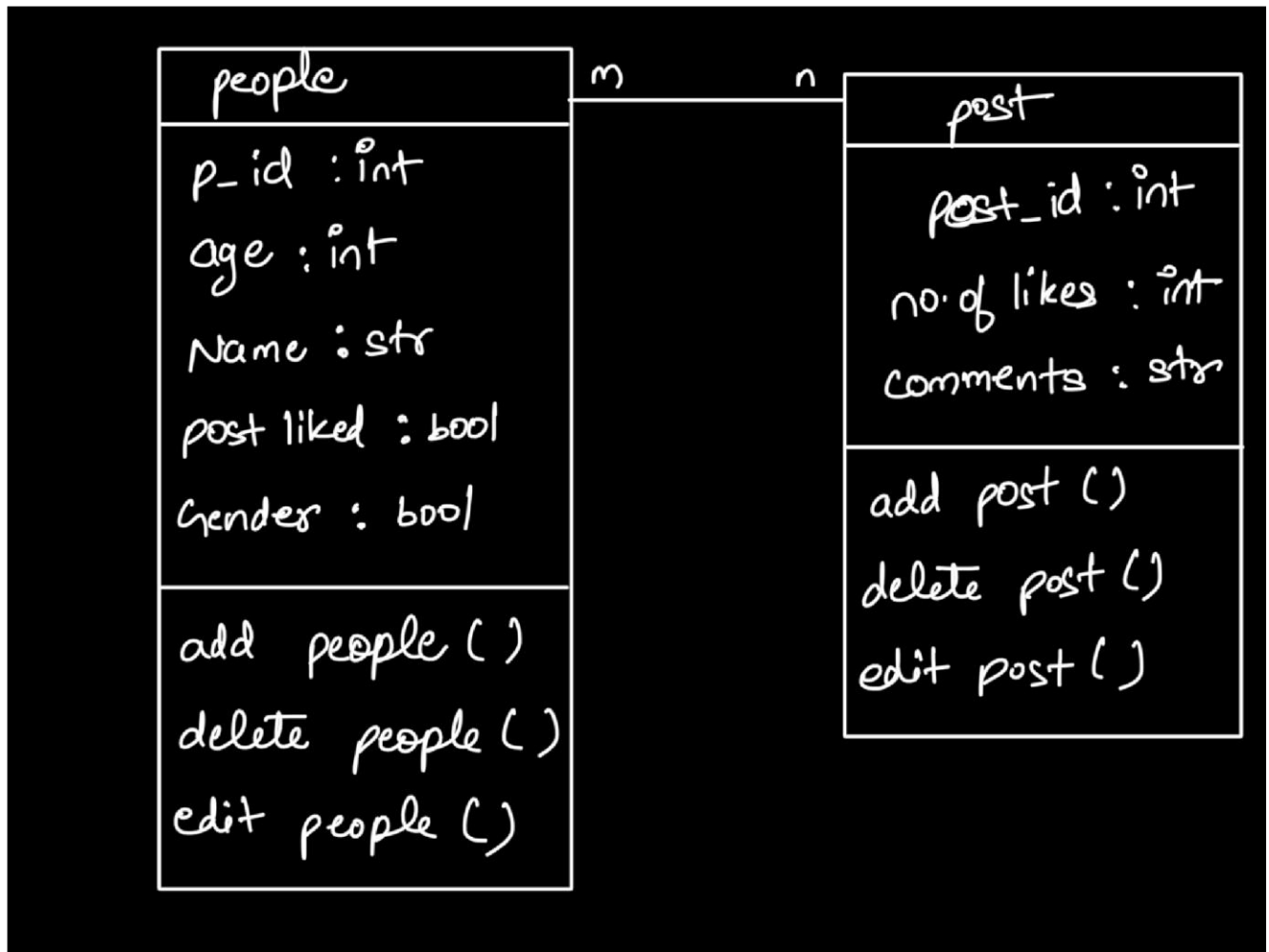
Data Flow Diagram:



ER- Diagram:



UML Diagram:



Code:

From *pre-process.py* file:

```
import csv
import random

def write_csv(matrix):
    with open('similarity.csv', 'w', newline='') as file:
        writer = csv.writer(file, delimiter=' ')
        for row in matrix:
            writer.writerow(row)
```

```

def f():
    N = 1000 # 1000 -> N; Also need to change in main.py; obfuscation()
call
    res = [[1 for _ in range(0, N)] for _ in range(0, N)]
    for i in range(0, N):
        for j in range(i, N):
            if i != j:
                x = random.random()
                res[i][j] = x
                res[j][i] = x
    return res

result = f()
write_csv(result)

```

From *main.py* file:

```

import csv
import random
from fastapi import FastAPI, Request, Form
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates

app = FastAPI()

templates = Jinja2Templates(directory="templates")
app.mount("/static", StaticFiles(directory="static"), name="static")

@app.get("/", response_class=HTMLResponse)

```



```

async def input_form(request: Request):
    return templates.TemplateResponse("form.html", {"request": request})

@app.post("/", response_class=HTMLResponse)

```

```

async def recommended_users(request: Request,
                             liked_users: str = Form(...)):
    recommended_users_arr = liked_users.split(',')
    recommended_users_arr = [int(e) for e in recommended_users_arr]

    # 1000 -> N; Also need to change in preprocess.py
    recommended_users_arr = obfuscation(recommended_users_arr, 10, 1000)
    recommended_users_arr = predict(recommended_users_arr)
    res = []
    for id, score in recommended_users_arr:
        res.append(f"                {id}                                {score}")
    return templates.TemplateResponse("result.html", {"request": request,
"res": res})

```

```

def predict(arr):
    arr=set(arr)
    matrix=get_similarity_matrix()
    N=len(matrix)

    scores=[[0, i] for i in range(0, N)]
    for liked_item in arr:
        for idx in range(0, N):
            scores[idx][0] += matrix[liked_item][idx]
    scores.sort(reverse=True)
    res=[]
    for score, val in scores:
        if val not in arr:
            res += [[val, score]]
        if len(res) == 10:

```

```
        break
    return res

def get_similarity_matrix():
    with open('similarity.csv', newline='') as file:
        reader=csv.reader(file, delimiter=' ')
        matrix=[]
        for row in reader:
            matrix += [[float(e) for e in row]]
        return matrix

def obfuscation(arr, THRESHOLD, N):
    THRESHOLD=int((THRESHOLD*N)/100)
    THRESHOLD=min(THRESHOLD, N)
    indexes=random.sample(range(0, N), THRESHOLD)
    res=set()

    for idx in set(indexes+arr):
        if random.randint(0, 1):
            res.add(idx)
    return sorted(res)
```

Output:

PrivRank

Enter User Ids

PrivRank Result

Recommending Item Number | Prediction Score

| | |
|-----|--------------------|
| 22 | 33.676141454315825 |
| 479 | 32.99377932313922 |
| 807 | 32.181812879717626 |
| 253 | 32.03791261799418 |
| 384 | 31.762891705476598 |
| 828 | 31.62691467981335 |
| 43 | 31.543402498974093 |
| 216 | 31.509598629446977 |
| 441 | 31.389902617410257 |
| 961 | 31.38700923767059 |