1. Generate 10k users. Split them into 10 groups. Each group has 1k users, and users in the same group have similar interest (interest tags: finance, sports, technology, movie, travelling, politics, education, adventure, games, and music). The user table (USER) has columns: 1) ID of 6 digits. 2) isCreator = False. 3) tag. 4) followerCount (number of follower)
2. Generate 300 creators. 30 creators of each interest tag. The creators and users share the same table and creators have isCreator=True. A creator is also a user. Generally, a creator has much more followers than normal users. Save the users in “data/users.csv”.
3. Generate an engagement table (ENGAGEMENT), which contains columns 1) engagement actor ID 2) engagement receiver ID 3) receiver’s isCreator flag and engagement counts of 4) like, 5) save, 6) reshare, 7) privateShare, 8)videoView60, 9) profileVisit, and 10) follow (follow = true means user follows a creator).
   1. For each engagement, generate a random count from 0 to 10, if engagement actor and receiver have the same tags, otherwise generate the count from 0 to 4.
   2. Make sure the engagements are sparse, meaning that a user may engage in another user in at most 4 engagements.
   3. Each user (actor) engage in 10 creators (7 have same tags, and 3 doesn’t have), and 5 normal users (3 have same tags, and 2 have different tag).
   4. A user won’t engagement with himself.
   5. A creator with more followers will receive more engagements.
   6. A creator can also engage in other users (as well as creators).
   7. Generate an engagement-generating Python file.
   8. Different engagement has different importance, and the table should contain a variable evaluating the weight of the edge.
   9. Save the engagement table in “data/engagements.csv”.
4. Load the saved table file. We can treat the table ENGAGEMENT as a graph. Use GNN (Graph nueral network) to generate embedding of users (and creators) so that two more close users in the graph will have more similar embedding. Introduce different types of GNN, and which one is more reasonable to use.
5. After training embedding, use KNN to find the most 10 close (based on cos similarity of two embedding) creators for each user. To evaluate the GNN, compute the percentage of similar creators that have the same tags as the users, for each users.
6. For GNN, please show detailed and theoretical derivation, formulation, and equation to build the network. And specify how GNN evaluate engagements to train the embedding. We need inductive GNN and the graph is weighted. Think about how to choose the model (GraphSAGE or GATs), and how to make adjustment to allow inductive ability and dealing weight graph.
7. Every code is in Python. If GPU is not available, allow use of CPU.
8. Properly set the random seed so that the experiment can be reproduced.
9. Consider a problem: Now if we have a more recently updated engagement table, do we have to retrain the whole GNN to obtain the new embedding? If no, what can be done to effective and quickly update the embedding? If yes, why?
10. Update edge weights/add new edges. 1) Update 10% edge weights by adding a random count from 1 to 2 to an engagement. if receiver is a normal user, and a random count from 2-4 if receiver is a creator. 2) Add 10% new edges. If the receiver is a creator, add a random count