

ORIE 5355/INFO 5370 HW 2: Recommendation systems

- Name:
- Net-id:
- Date:
- Late days used for this assignment:
- Total late days used (counting this assignment):
- People with whom you discussed this assignment:

Notes

We have marked questions in **green**. Please put answers in the default color. You'll want to write text answers in "markdown" mode instead of code. In Jupyter notebook, you can go to Cell > Cell Type > Markdown, from the menu. Please carefully read the late days policy and grading procedure [here](#). In that link, we also give some tips on exporting your notebook to PDF, which is required for GradeScope submission.

Some of the questions can be interpreted in multiple ways. That is always true in data science. You'll need to make judgment calls for what analysis to do. For the homework, you'll still receive full points for any "reasonable" choice. Briefly explain and justify your choices. Also feel free to ask questions on EdStem.

Disclosure and survey

Type "I understand" in the next box to attest to the following statements.

I understand that I have to submit this homework assignment in two places:

- gradescope -- where I will submit a pdf of the assignment and tag each page of my assignment with the problem to which it corresponds.
- github (via classroom50) -- where I will commit and push up this notebook and the rest of my code, in a runnable format.

I understand that I will reach out to the teaching staff via office hours or EdStem at least 5 days before the homework is due if I have any issues submitting the assignment on either of these platforms, such as I am not on the class roster. I understand that I may receive points off (up to full value of the homework assignment) if I do not submit to both locations.

I understand that the teaching staff prefers that I make multiple commits on github (via classroom50) as I am completing the assignment (for example, after I finish each problem), and that this information may be used to assess overall understanding, progress, and qualitative assessment/grading.

I understand that I must fully disclose my usage of AI in the box below **and** at the top of each problem. I will follow the AI usage policy, and am responsible for everything that I submit. I understand that quizzes and other assessments may evaluate understanding of what I submitted, including asking me to interpret what code does.

After you finish the homework, please complete the following (short, anonymous) post-homework [survey](#) and include the survey completion code below.

AI disclosure. In the below box, please write 1-3 sentences about your use of AI in the assignment **overall**. Example responses include:

- None
- "I vibe-coded all the code for the assignment. I didn't even try to understand or parse the code"
- "I referred to ChatGPT for interpreting parts of the assignment, and to help me write code, but ultimately I understood each of its outputs and either typed/copied in small pieces code that it gave me. I wrote more than half the code myself".

Note that AI disclosure is part of the grade for each component, and you are ultimately responsible for your learning.

Please also note that AI disclosure is part of each question below

Conceptual component

Reading 1: Stitch Fix

Go through the "Algorithms tour" [here](#). It's a great view of the combination of algorithms used by a modern e-commerce company.

AI disclosure for this reading: In the next box, describe any AI use for this reading and how it was used, or write "None".

1) How do they use a combination of "latent" factors and explicit features to gain the benefits of collaborative filtering (matrix factorization) while mitigating cold-start problems?

2) How do they match clients with human stylists who make the final decision? Does it remind you of anything we learned in class?

3) A customer says, "My previous purchases were for work, but now I want casual clothes." Give one way the customer should be able to steer the recommendations. How could a conversational LLM help, and why would adding an LLM alone not guarantee that the customer's request is respected? Answer in at most three sentences.

Reading 2: LLM-based recommendations at Netflix

Please read at least the **Introduction (Section 1)**, **Section 3 (Problem Setting)**, and **Section 4 (Methodology)** of [GenRec: An LLM-Backed Recommendation Ranker at Netflix](#).

This paper describes an LLM-based recommender that broadly follows the paradigm from class: use data about users and items to learn personalized user preferences, and account for additional objectives and business requirements when making recommendations. (Some of these requirements are incorporated during model training itself, differing from the 2 stage approach we overviewed in class). You do not need to focus on every implementation detail, but it may be of interest.

AI disclosure for this reading: In the next box, describe any AI use for this reading and how it was used, or write "None".

1) What does GenRec try to predict about a user's interactions with items? What data does it use for this prediction task?

Give two examples of prediction targets/training signals the authors prefer, explain how these relate to Netflix's longer-term goal, and whether there is a gap between them. Answer in no more than four sentences.

2) Beyond prediction accuracy, what other business rules or objectives matter when deciding what to recommend? Give two examples from the paper and briefly explain how GenRec incorporates them. Answer in no more than four sentences.

Programming component

Helper code

```
In [ ]: import numpy as np
import pandas as pd
import os, sys, math
import matplotlib.pyplot as plt
import pickle
def load_pickle(filename):
    with open(filename, "rb") as f:
        data = pickle.load(f)
    return data

def load_user_item_vectors(type_name = 'rating_all_zero'):
    # ratings = load_pickle('data/{}_ratings'.format(type_name))
    book_vectors = load_pickle('data/{}_dict_book_factor'.format(type_name))
    user_vectors = load_pickle('data/{}_dict_reader_factor'.format(type_name))
    return book_vectors, user_vectors
```

In this homework, we are giving you trained user and book item vectors using a GoodReads dataset. Goodreads is a social cataloging website that allows individuals to search its database of books, annotations, quotes, and reviews. There are multiple types of interactions that a user can have with a book: add books to a list of books they intend to read ("short-list" the book), indicate they have read books before, and review books they have read. A user may have interacted with a book, even if they never left any explicit rating or review. Ratings actually left by users range from 1 to 5 (stars), with 5 being the most positive and 1 being the most negative.

Here, we work with multiple types of data that capture interactions between books and users as training data for a recommendation system. For each "type" of rating data, we used the raw ratings data and trained user and item vectors using a Python package (<https://berkeley-reclab.github.io/>), which implements matrix factorization in cases where there are missing entries in a matrix.

There are 2 types of rating/interaction data that we used:

- `Rating_all_missing_zero` : Numeric values more than 0 indicate the star ratings given. Missing values are replaced with 0's, so that there are no missing ratings. In other words, if a user has not left any reviews for a book, we assume that the user would rate the book 0 star.
- `Rating_interaction_zero` : Numeric values more than 0 indicate the ratings given. Now, we replace missing values with 0's, *only if the user interacted with that book in the past*. In other words, if a user has not left any reviews for a book *and* has interacted with the book, we assume that the user would rate the book 0 star. Note that after such replacement, we would still get some missing values, since not all users have interacted with all the books.

To make it easier for you, we do not provide you the ratings data directly, but only what you'll be working with: user and item vectors trained using these data.

```
In [1]: book_vectors_allmissing0, user_vectors_allmissing0 = load_user_item_vectors(type_name = 'rating_all_zero')
book_vectors_interact0, user_vectors_interact0 = load_user_item_vectors(type_name = 'rating_interaction_zero')
```

```
In [19]: def get_shapes_and_ranges(book_vectors, item_vectors):
print(np.shape(book_vectors), np.shape(item_vectors))
```

```
In [20]: get_shapes_and_ranges(book_vectors_allmissing0, user_vectors_allmissing0)
get_shapes_and_ranges(book_vectors_interact0, user_vectors_interact0)
```

```
(200, 10) (1000, 10)
(200, 10) (1000, 10)
```

Problem 1: Predictions and recommendations with different data types

AI disclosure for this problem: In the next box, describe any AI use for this problem and how it was used, or write "None".

1a) What do different data types mean?

What is `Rating_interaction_zero` trying to capture -- why would we fill in books that someone interacted with but did not rate as a 0? (Hint: connect to conceptual reading from HW1). Answer in no more than 3 sentences.

What are some potential problems you see with using `Rating_all_missing_zero` for recommendations? Answer in no more than 3 sentences.

1b) Generating predictions

Fill in the following function that takes in a user matrix (where each row is 1 user vector) and an item matrix (where each row is 1 item vector), and returns a matrix of predicted ratings for each user and item, where each entry is associated with the corresponding user (row number) and item (column number)

```
In [21]: def get_predictions(user_vectors, book_vectors):  
         pass # your code here
```

```
In [ ]:
```

Output the predictions for first 10 items for the first user, using each of the 2 data types.

For example, the predictions for one of the data types are:

Ratings for first 10 items, `Ratings_all_missing_zero`:

```
[ 0.08 0.29 0.063 1.57 -0.186 0.055 0.011 -0.088 -0.895 -0.012]
```

```
In [ ]:
```

Do a scatterplot of the predicted rating for two data types. (Each dot represents one user and one book, with X axis being predicted ratings using `Rating_interaction_zero` data and Y axis being predicted rating using `Rating_all_missing_zero` ratings). Describe what you see in no more than 2 sentences.

In []:

1c) From predictions to recommendations (without capacity constraints)

Fill in the following function that takes in the matrix of predicted ratings for each user and item, and returns a dictionary where the keys are the user indices and the values are a list of length "number_top_items" indicating the recommendations given to that user

```
In [22]: def get_recommendations_for_each_user(predictions, number_top_items = 10):  
         pass
```

Output the recommendations for the first user, using each of the 2 data types.

For example, from the `Ratings_all_missing_zero` dataset, you should get: [57, 55, 56, 81, 50, 78, 58, 86, 77, 96]

In []:

Fill in the following function that takes in the (top 10) recommendations for each user and the total number of items in the catalog, and outputs a histogram for how often each item is to be recommended. Count all catalog items, including those recommended zero times. For example, if there are 18 items, and 10 of them were never recommended, 5 of them were recommended once each, and 3 of them were recommended five times each, then you would have bars at 0, 1, and 5, of height 10, 5, and 3, respectively.

```
In [ ]: def show_frequency_histograms(recommendations, number_items=200):  
         pass
```

Show the histograms for both data types. Describe what you observe in no more than 3 sentences. For example, discuss how often is the most recommended item recommended, how that compares to the least recommended items, and what that could mean for recommendations in various contexts.

In []:

Problem 2: Cold start -- recommendations for new users

AI disclosure for this problem: In the next box, describe any AI use for this problem and how it was used, or write "None".

In this part of the assignment, we are going to ask you to tackle the "cold-start" problem with matrix-factorization based recommendation systems. The above recommendation techniques worked when you had access to past data for each user, such as interactions or explicit ratings. However, it doesn't work as well when a new user has just joined the platform and so the platform doesn't have any data.

You should also see a comma-separated values file (user_demographics.csv) that contains basic demographic information on each user. Each row describes one user, and has four attributes: 'User ID', 'Wealth', 'Age group' and 'Location'. (This data is synthetic, not real data from GoodReads; we make it up as illustrations.)

User ID is the unique identifier associated with each user, and it is in the same order as the user_vectors, and in the same indexing as the ratings (be careful about 0 and 1 indexing in Python).

Wealth is a non-negative, normalized value indicating the average wealth of the neighborhood in which the user is, where we normalized it such that each Location has similar wealth distributions. Age group describes the age of the user. Location describes the region that the user is from.

```
In [24]: demographics = pd.read_csv("data/user_demographics.csv")
demographics.head()
```

```
Out[24]:
```

	User ID	Wealth	Age group	Location
0	1	1.833101	50 to 64	America
1	2	2.194996	18 to 34	America
2	3	2.216195	18 to 34	Europe
3	4	0.838690	50 to 64	Asia Pacific
4	5	2.109313	18 to 34	America

We are now going to pretend that we don't have the personalized ratings/interactions history for the last 100 users, and thus don't have their user vectors. Rather, let's pretend that these are new users to the platform, and you are able to get the above demographics from their browser cookies/IP address. Now, we're going to try to recommend items for them anyway. For this part, we'll exclusively use the `Rating_interaction_zero` data.

(In other words, we're going to use the first 900 users for training, and then evaluate on the remaining 100 users)

```
In [ ]: existing_user_vectors = user_vectors_interact0[0:900,:]
existing_user_demographics = demographics.iloc[0:900,:]
new_user_demographics = demographics.iloc[900:,:]
```

2a) Predictions for new users [Simple]

Fill in the following function that takes in: the demographics of a single new user, the demographics of all the existing users in your platform, and the user vectors of all the existing users, and outputs a 'predicted' user vector for the new user to use until we get enough data for that user.

For this question, we ask you to use the following simple method to construct the vector for the new user. Each user is classified as "Low" or "High" wealth based on whether their Wealth score is below or above the median of about 1.70. Then, we simply construct a mean user vector for "Low" and "High" wealth, based on the 900 users (take the average vector among users with "Low" and "High" Wealth, respectively.). The corresponding mean vector is then used for each new user.

For example, using this method, you should find that the vector for the second user (index "1") is:

```
array([-0.183, -0.149, -0.141, -0.199, -0.166, -0.272, -0.02 , 0.137, -0.12 , 0.022])
```

Note: our answers below are based on the median *not rounded* to 2 decimal places, i.e., we use the full median to many decimal places.

```
In [26]: existing_user_demographics.Wealth.median()
```

```
Out[26]: 1.7026180771992308
```

```
In [27]: def get_user_vector_for_new_user(new_user, existing_user_demographics, existing_user_vectors):  
         pass
```

```
In [ ]:
```

Output the mean vector predicted for the first user (index 0) in `new_user_demographics` .

```
In [ ]:
```

For each of the 100 "new" users, use your model to retrieve a user vector for that user, and then your functions from Problem 1 to get predicted ratings and top-10 recommendations.

Plot a scatterplot between the ratings predicted by the demographic model and the ratings predicted by the full model from Problem 1. Each point in the scatter plot should correspond to one user and one item, and so your scatterplot should have 100*200 points.

For example, for the first user-item pair (index 0 user, index 0 item), your prediction using the basic demographic should round to -0.0012, and using the full model should round to 0.3145. So one point in the scatter plot would be approximately (-0.0012, 0.3145).

```
In [ ]:
```

Comment on the above. What is the potential "loss" from using demographics since we do not have access to the full data?

2b) Predictions for new users [Using KNN or another model]

Fill in the following function that takes in: the demographics of a single new user, the demographics of all the existing users in

your platform, and the user vectors of all the existing users, and outputs a 'predicted' user vector for the new user to use until we get enough data for that user.

Now, use K nearest neighbors or some other machine learning method, and report the same things as in 2a.

Feel free to prepare data/train a model outside this function, and then use your trained model within the function.

```
In [28]: def get_user_vector_for_new_user_knn(new_user, existing_user_demographics, existing_user_vectors):  
         pass
```

Output the predicted vector for the first user in `new_user_demographics` .

```
In [ ]:
```

Justify your choice of model. If you used K nearest neighbors, then how did you decide upon your distance function? If you used another model, how does that model weight the different demographics in importance (either implicitly or explicitly)?

Problem 3: Predictions under capacity constraints

AI disclosure for this problem: In the next box, describe any AI use for this problem and how it was used, or write "None".

Above, you should have observed that if we just recommend the top items for each user, some items get recommended quite a bit, and many items do not get recommended at all. Here, we are going to ask you to implement recommendations under capacity constraints.

Throughout this part, assume that you only have 5 copies of each item that you recommend, and that you will only recommend 1 item to each user. In other words, you cannot recommend the same item more than 5 times, and so there are exactly 1000 items in stock (representing 200 unique books) for your 1000 users.

We'll continue exclusively using the "ratings with interaction0" data.

Now, let's assume that users are entering the platform sequentially in order of index. So the index 0 user comes first, index 1

user comes second, etc.

3a) Naive recommendations under capacity constraints

First, let's pretend that we were naively recommending the predicted favorite item to each user. Of course, with unlimited capacity, each user would be recommended their predicted favorite. With capacity constraints, the favorite items of the users who come in later might already have reached their capacity, and so they have to be recommended an item further down their list.

Do the following: simulate users coming in sequentially, in order of index. For each user, recommend to them their predicted favorite item that is still available. So the first user will get their favorite item, but the last few users will almost certainly not receive any of their top few predicted items. For each user, keep track of what the rank of the item that they were ultimately recommended was, according to the predicted ranking over items for that user.

For example, you'll see that the first user was recommended their favorite item, but the last user was recommended their 129th favorite item.

Plot the resulting rankings in 2 ways: 1) A line plot, where the X axis is the user index and the Y axis is the rank of the item that they were recommended. and 2) A histogram of how often each rank shows up. (the X axis is the (binned) rank, and the Y axis is the count of that bin).

In all cases, you will plot the rank according to the user's predicted ranking.

In []:

3b) Optimal recommendations under capacity constraints -- maximum weight matching

Now let's do "optimal" recommendations with capacity, using maximum weight matching. Create the same two plots as above. Describe what you observe compared to the naive recommendations above.

We suggest you use the `scipy.optimize.linear_sum_assignment` function with `maximize=True` when using predicted ratings as weights. In that case, `np.tile` might also come in handy to create 5 copies of each item.

In []:

Of course, in reality you don't observe all the users at the same time -- they come in one by one, and you need to create a recommendation for the first user before the 50th user shows up. Here, let's pretend that users show up in batches of 100. So the first 100 users at the same time, next 100, etc. In this case, you can do "batched maximum weight matching," where you run maximum weight matching for the first 100 together to determine recommendations. Then, you do the same thing for the next 100 users with the items that are remaining, etc.

Implement the above, show the same two plots as above, and describe what you observe. Note that this part requires careful attention for how many of each item remain after each round.

In []:

3c) Score functions for recommendations under capacity constraints

Here, we are working with just 200 items and 1000 users, and so batched maximum weight matching is feasible to run. In practice, with millions of items, that might not be an effective strategy. Now, we ask you to implement the score function approach from class.

You should normalize the predicted ratings between 0 and 1 so that you are not dividing by a negative or close to 0 average rating before proceeding. (Just in this case, you can apply this normalization once using the entire prediction matrix before the simulation, keeping the mean predicted ratings fixed during the simulation. In practice, since we don't observe users before they arrive, we would have to use historical data to perform this normalization).

Implement the above and run the same simulation as part 3a, show the same two plots, and describe what you observe.

Note that, as above, you will plot the rank according to the user's predicted ranking, NOT the score function ranking. (Plotting by score function ranking will result in just a straight line where everyone gets their top ranked item, which is incorrect).

For this part, use the following score function:

$$\frac{r_{ij}}{\bar{r}_j} \sqrt{C_j}$$

HINT: In your code, for each user i you will:

1. Retrieve the ratings r_{ij} for each item j .

2. Divide each r_{ij} by the mean item rating $\bar{r}_j$ and multiply by the sqrt of the current capacity for that item.
3. Sort the items by the above modified score, and recommend the best item according to the modified score.

In []:

In []:

In []:

Comment for entire homework: In this homework, we haven't been careful with what is "training" data and what is "test" data. For example, in 3c, you're using average ratings from customers who haven't shown up yet in your simulation. In Problem 2, when training the user/book vectors we used data from customers that we are then pretending we haven't seen data from. In practice, and for the class project, you should be more careful. Such train/test/validation pipelines should be a core part of what you learn in machine learning classes.