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:

After you finish the homework, please complete the following (short, anonymous) post-homework survey: https://forms.gle/7tFZUoDszbDeDKmV6 and include the survey completion code below.

Question 0 [2 points]

Survey completion code:

We have marked questions in blue . Please put answers in black (do not change colors). 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.

Conceptual component

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

1) How do they use a combination of "latent" factors and explicit features to gain the benefits of collaborative filtering (matrix factorization) while not being susceptible to cold start issues?

In [ ]:

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

In [ ]:

3) How do they manage their inventory to ensure that they have enough items that future customers will want?

In [ ]:

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_ratings_and_factors(type_name = 'interaction'):
    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 ratings, 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.

Here, we work with multiple types of interactions as training data for a recommendation system. For each "type" of rating data, we give you the raw ratings data, as well as user and item vectors trained using a Python package (https://berkeley-reclab.github.io/) that implements matrix factorization in cases where there are missing entries in a matrix. The "ratings" data is in a "sparse matrix"/dictionary format, meaning that the dictionary keys are of the kind (user, item), and the dictionary value is the corresponding value. Not all pairs are in the matrix, indicating that that value is missing or at its default value.

There are 4 types of rating/interaction data:

- `Interaction` : a "1" indicates the user has interacted with the book at some point in the past, either by saying that they intend to read it, have read it, or have given it a rating. If it is missing, that means the user has not interacted with the book.
- `Explicit Rating` : explicit ratings. Numeric values indicate the ratings given. If it is missing, that means the user has not rated the book.
- `Rating_all_zero` : explicit ratings. Numeric values more than 0 indicate the ratings given. Now, we replace missing values from above with "zeros," so that there are no missing ratings.

- `Rating_interaction_zero` : explicit ratings. Numeric values more than 0 indicate the ratings given. Now, we replace missing values from above with "zeros," only if the user interacted with that book in the past.

In [ ]:
```
ratings_interactions, book_vectors_interactions, user_vectors_interactions = load_rati
ratings_explicit, book_vectors_explicit, user_vectors_explicit = load_ratings_and_fact
ratings_allmissing0, book_vectors_allmissing0, user_vectors_allmissing0 = load_ratings
ratings_interact0, book_vectors_interact0, user_vectors_interact0 = load_ratings_and_f
```

In [ ]:
```
def get_shapes_and_ranges(ratings, book_vectors, item_vectors):
    print(len(ratings), np.shape(book_vectors), np.shape(item_vectors), min(ratings.va
```

In [ ]:
```
get_shapes_and_ranges(ratings_interactions, book_vectors_interactions, user_vectors_ir
get_shapes_and_ranges(ratings_explicit, book_vectors_explicit, user_vectors_explicit)
get_shapes_and_ranges(ratings_allmissing0, book_vectors_allmissing0, user_vectors_allm
get_shapes_and_ranges(ratings_interact0, book_vectors_interact0, user_vectors_interact
```

```
12238 (200, 10) (1000, 10) 1 1
8324 (200, 10) (1000, 10) 1 5
200000 (200, 10) (1000, 10) 0 5
12238 (200, 10) (1000, 10) 0 5
```

Problem 1: Predictions and recommendations with different data types

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_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 [ ]:
```
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 4 data types.

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

Ratings for first 10 items, interactions: [-0.003 0.01 0.002 -0.001 0.003 0.007 -0.01 0.007 0.001 0.003]

In [ ]:

Do a scatterplot of the predicted rating for the "interaction" and "explicit ratings" types. (Each dot represents one user and one book, with X axis being predicted ratings using interaction data and Y axis being predicted rating using explicit 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 dictioanry 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 [ ]:
```
def get_recommendations_for_each_user(predictions, number_top_items = 10):
    pass
```

Output the recommendations for the first user, using each of the 4 data types.

For example, from the "Interaction" dataset, you should get: [182, 198, 19, 100, 104, 73, 30, 199, 164, 74]

In [ ]:

Fill in the following function that takes in the (top 10) recommendations for each user, and outputs a histogram for how often each item is to be recommended. 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):
    pass
```

Show the histograms for the "interact" and "explicit" 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.

Problem 2: Cold start -- recommendations for new users

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 reach 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 have four attributes: 'User ID', 'Wealth', 'Age group' and 'Location'.

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

Out[ ]:

	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 browswer cookies/IP address. Now, we're going to try to recommend items for them anyway. For this part, we'll exclusively use the "ratings with interaction0" data.

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 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 correpsonding 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])

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

Out[ ]:
```
1.7026180771992308
```

In [ ]:
```
def get_user_vector_for_new_user(new_user, existing_user_demographics, existing_user_v
    pass
```

In [ ]:
```

```

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

In [ ]:
```

```

2b) [Bonus, 3 points] 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.

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

```

```

In [ ]:
```
def get_user_vector_for_new_user_knn(new_user, existing_user_demographics, existing_us
    pass
```

Output the mean vector predicted 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)?

In [ ]:

2c) Comparing predictions from "true" user vector and from above

For each of the 100 "new" users, use either your model from 2a or 2b ("demographic model") to retrieve a user vector for that user, and then your functions from Problem 1 to get predicted ratings and top-10 recommendations. First, 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 user), your prediction using the basic demographic should be -0.0011902780252621872, and using the full model should be 0.31447640890118356. So one point in the scatter plot would be (-0.0011902780252621872, 0.31447640890118356).

In [ ]:

Now for each new user, calculate the mean rating (according to the "full" model in Problem 1) for the 10 items recommended to that user, by each of the demopgraphic and "full" models. Output a scatterplot for the two mean ratings, where each point correpsonds to 1 user (and so you will have 100 points in your scatter plot). For example, for the first new user, the associated point is (2.4880867541832146, 0.5305243424156764).

In [ ]:

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

In [ ]:

Problem 3: Predictions under capacity

constraints

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 predicting 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 [ ]:

3b) [Bonus -- 4 pts] Optimal recommendations under capacity constraints -- maximum weight matching

[2 points] 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. In that case, `np.tile` might also come in handy to create 5 copies of each items.

In [ ]:

[2 points] 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's 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.

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

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. Normalize each r_{ij} for by mean item rating 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.