Assignment 1

DSCI 641 - Recommender Systems – Winter 2024

OVERVIEW

In this assignment, you will explore two recommender systems data sets and compute some basic recommendations.

The deliverables for this assignment are **two Jupyter notebooks**. Submit the notebook files (`.ipynb`), along with PDF (preferred) or HTML exports of them, to Blackboard. All notebooks should be a well-organized and readable; see my notes and video on writing effective notebooks for what this means.

This assignment is due **11:59 PM** on **Saturday, January 27.**

Read the entire assignment before beginning; there are tips at the end. In Week 2's lecture, we will be looking at example code.

DATA SETS

For this assignment, you need to use two data sets.

- The MovieLens 25M data set from https://grouplens.org/datasets/movielens/
- Amazon rating data from the product category of your choice from https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/

For the Amazon data, you will need to download two files for your category:

- The **ratings-only** CSV file (from the section "*Small*" *subsets for experimentation*)
- The **metadata** JSON file (from the section *Per-category files*)

REQUIREMENTS

In a separate notebook for each data set, address the questions below.

Basic Statistics

- How many items are in the data set? How many users? How many ratings?
- User activity:
    - What is the distribution of ratings-per-user?
- Item statistics

    - What is the item popularity curve (the distribution of ratings-per-item)? A Lorenz curve or a CDF plot on a log x scale is a good way to look at this; a rank-frequency plot on a log-log scale is also a good means.
    - What is the distribution of average ratings for items?

Non-personalized Recommendation

- What are the 10 most popular items (the items with the most ratings)? Show the item ID, item title, and the number of ratings.
- What are the 10 items with the highest average ratings (with their titles and average ratings)?
- What are the 10 items with the highest *damped* average ratings, with a damping factor$\gamma = 5$? Recall that the damped mean is computed by:

$$\tilde{r}_i = \frac{\left(\sum_{r_{ui} \in R_i} r_{ui}\right) + \gamma \bar{r}}{|R_i| + \gamma}$$

  Show both the damped and undamped mean for these items. You can also use the bias model without user biases to compute these means.

Item Similarity

In this section, we're going to start looking at *similar movies*, and using that to generate basic recommendation lists. For this section, just use the MovieLens data.

Probability

We're going to start with probability-based relatedness:

- What are the 5 most popular movies among users who also rated *Toy Story* (not counting *Toy Story* itself)?
- What are the 5 most popular movies among users who also rated *10 Things I Hate About You?*
- Pick another movie of your choice to generate a top-5 list for.
- The previous questions use conditional probability ($P[j|i]$, where i is the reference item). Answer the same questions using *lift* ($P[j|i]/P[j]$) instead. Do these lists make more or less sense?

Next, let's look at rating values to measure the similarity between movies (*item-based collaborative filtering*, even though we aren't recommending for users yet).

Rating Similarity

First, prepare the data by removing all items with fewer than 5 ratings, mean-centering the ratings (for each rating, subtract the item's mean rating), and normalizing each item's rating to be a unit vector. Convert the ratings into a *sparse matrix* for efficient computation. You can normalize either before or after turning the data frame into a sparse matrix. This allows fast cosine similarities.

As we will discuss in Week 2 lecture, there are reasons for each of these:

- When we mean-center the vectors (so each item has a mean of 0), computing a cosine similarity is almost same as computing the Pearson correlation.
- If two vectors are unit vectors ($\|\vec{x}\| = \|\vec{y}\| = 1$), then the cosine similarity is just the dot product ($\cos(\vec{x}, \vec{y}) = \vec{x} \cdot \vec{y}$).
- If $\vec{x}$ is a column vector, then $M\vec{x}$ computes the similarity between $\vec{x}$ and *every row* of M, yielding a vector of the results. This matrix-vector multiplication operation is heavily optimized in linear algebra packages like Numpy and Scipy, so we get fast comparisons between our target item and the other items in the database.

Task: Once you have computed your normalized rating matrix, compute top-5 lists for the same 3 movies above based on cosine similarity between rating vectors.

Content-Based Similarity

The final item similarity is to use the "Tag Genome" files to do *content-based* similarity between items. To prepare:

1. Load the tag genome file into a data frame. It will have one row for each (item, tag) combination.
2. Use *pivot* to convert the tag genome into a wide data frame (one row for each item, and column for each tag
3. As with ratings, normalize each row to be a unit vector. Optionally, you can mean-center them first (if you want, compare results of this both with and without mean-centering!).

You can treat Pandas data frames as matrices, so the same matrix-vector multiplication operations you used for similarities above (and probably used for the lift computations) work for comparing items.

Task: compute top-5 lists for the same 3 movies above based on cosine similarity between tag genome vectors.

Recommending for Users

This section also is only on the MovieLens data set. Pick 2 users from the data set by whatever criteria you wish (you may also create a vector of your own ratings for at least 15 movies in lieu of one of the users).

For each of these users, generate two different *item-based* top-5 recommendation lists using 2 item similarity methods of your choice from the previous section. Score items using the **weighted average** method: if w_{ij} is the score for item j with respect to reference item i by one of the methods above (conditional probability, lift, rating similarity, or content similarity), then:

$$s(i|u) = \frac{\sum_{i \in I_u} r_{ui} w_{ij}}{\sum_{i \in I_u} r_{ui}}$$

While you can compute this in a fully vectorized fashion, users have few enough items that it won't be too expensive to loop over the user's items to do the final scoring computation. You don't want to loop over all target items, though.

Task: compute top-5 recommendation lists for the users, ordered by this scoring function.

Reflection

Write 2-3 paragraphs at the end of your MovieLens notebook about what you learned from this assignment, what surprised you, and — to the extent you can judge — which methods seemed to produce more useful or interesting recommendation lists.

SUBMITTING

Submit 2 notebook files, one for each data set. Clean up, organize, and document your notebooks before submitting; see the guidance linked in the introduction.

For each notebook, submit 2 things:

- The .ipynb source file.
- The exported PDF or HTML file.

TIPS

Recommendation data is pretty big, and I have deliberately selected a large data file for you to work with to encourage efficient code.

You will need to make extensive use of **vectorized computation** in this assignment: performing operations on entire NumPy arrays or Pandas data frame columns at a time, without writing Python loops (everything in this assignment except perhaps the user-based recommendation can be done without writing a single Python loop). We will see some examples of this in the example code in Week 2; the example exploration notebook from Week 1 also has some examples.

Sometimes, some clever work with grouping and/or joins will help you exploit vectorization. For example, my standard approach to subtract item means is this:

1. Compute item means with `groupby` and `mean`, and make sure the result is a data frame with item IDs on the index and a column for means:

```
item_stats = ratings.groupby('item')['rating'].mean()
item_stats = item_stats.to_frame('item_mean')
```

2. Join this with the rating frame:

   ```
   rate_work = ratings.join(item_stats, on='item')
   ```

3. Compute the normalized ratings:

   ```
   rate_work['nrating'] = (rate_work['rating']
       - rate_work['item_mean'])
   ```

Sparse matrices (SciPy's `csr_array`) will be useful for computing similarities. It is also very good for computing co-occurrences. We will see an example of such co-occurrences in Week 2's lecture and example code. Pandas Indexes are useful for mapping between user and item IDs (which may be arbitrary values) and 0-based row or column numbers in a matrix.

You can use the "Latest Small" dataset instead of 25M for initial testing and debugging your code; it is in the same format as 25M, but is much smaller (100K ratings) so computations are faster and take less memory.