Programming User-User Collaborative Filtering.

In this assignment, you will implement a user-user collaborative filter for LensKit.

LensKit provides a flexible implementation of user-user collaborative filtering, but for this assignment we would like you to implement it (mostly) from scratch.

Specifically, we're going to have you build a model-free user-user collaborative filtering scorer that predicts a target user's movie rating for a target item by going through the following process:

1. First, you will adjust each user's rating vector by subtracting that user's mean rating from each of their ratings (this corrects for the fact that some users think 5 stars is anything worth seeing and others think 3 stars is very good).
2. Next, you will identify the set of other users who have rated the target item and who have a history of rating items similarly to the target user; specifically, we'll limit this set to the 30 users with the highest cosine similarity between their adjusted rating vectors and the target user's adjusted rating vector. This similarity measures the angle between the vectors, which is highest when both users have the rated the same items and have given those items the same rating (this won't be perfectly the case here, since we're predicting for unrated items).
3. Then you will combine the mean-adjusted ratings from these "neighbor" users, weighted by their cosine similarity with the target user — i.e., the more similar the other user's ratings, the more their rating of the target item influences the prediction for the target user.
4. Finally, re-adjust the prediction back the target user's original rating scale by adding the target user's mean rating back into the prediction.

Once you've written code to do this, the program will give either specific predictions or predictions for top-10 recommended unrated items for the selected users; you do not have to code this part as it's already built into LensKit.

Start by downloading the project template. This is a Gradle project; you can import it into your IDE directly (IntelliJ users can open the build.gradle file as a project). This contains a `SimpleUserUserItemScorer` class that you need to finish implementing, along with the Gradle files to build and run it.

Downloads and Resources

- Project template (on course website)
- LensKit for Learning website (links to relevant documentation and the LensKit tutorial video)

Additionally, you will need:

- Java — download the Java 8 JDK. On Linux, install the OpenJDK 'devel' package (you will need the devel package to have the compiler).

- A development environment

Basic Requirements

Implement scoring in this class as follows:

- Use user-user collaborative filtering.
- Compute user similarities by taking the cosine between the users' mean-centered rating vectors (that is, subtract each user's mean rating from their rating vector, and compute the cosine between those two vectors). LensKit's `Vectors` class can help you with this; it provides functions to compute dot products, euclidian norms, and means.
- For each item's score, use the 30 most similar users who have rated the item and whose similarity to the target user is positive.
- Refuse to score items if there are not at least 2 neighbors to contribute to the item's score.
- Use mean-centering to normalize ratings for scoring. That is, compute the weighted average of each neighbor v's offset from average ($r_{v,i} - \mu_v$), then add the user's average rating μ_u. Like this, where $N(u;i)$ is the neighbors of u who have rated i and $cos(u,v)$ is the cosine similarity between the rating vectors for users u and v:

$$p_{u,i} = \mu_u + \frac{\sum_{v \in N(u;i)} cos(u,v)(r_{v,i} - \mu_v)}{\sum_{v \in N(u;i)} |cos(u,v)|}$$

- Remember, cosine similarity is defined as follows:

$$cos(u,v) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\|_2 \|\vec{v}\|_2} = \frac{\sum_i u_i v_i}{\sqrt{\sum_i u_i^2}\sqrt{\sum_i v_i^2}}$$

- Do not use any code or classes from the `lenskit-knn` module; we want you to code this yourself.

Running the Recommender

You can run the recommender by using Gradle; the `predict` target will generate predictions for the user specified with `userId` and the items specified with `itemIds` (see next section for examples). `recommend` will produce top-10 recommendations for a user.

User-user CF has an interesting penchant for recommending really obscure things. We've also provided a configuration for a hybrid recommender that blends the collaborative

filtering output with popularity information to prefer more popular items. To run this version of your recommender, use `recommendBlended`.

All recommender-running tasks will send debug output to a log file under `build`.

Example Output

Command:

```
./gradlew predict -PuserId=320 -PitemIds=260,153,527,588
```

Output:

```
predictions for user 320:
  153 (Batman Forever (1995)): 2.841
  260 (Star Wars: Episode IV - A New Hope (1977)): 4.549
  527 (Schindler's List (1993)): 4.319
  588 (Aladdin (1992)): 3.554
```

Command:

```
./gradlew recommend -PuserId=320
```

Output:

```
recommendations for user 320:
  858 (Godfather, The (1972)): 4.562
  2360 (Celebration, The (Festen) (1998)): 4.556
  318 (Shawshank Redemption, The (1994)): 4.556
  8638 (Before Sunset (2004)): 4.512
  7371 (Dogville (2003)): 4.511
  922 (Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)): 4.503
  1217 (Ran (1985)): 4.497
  44555 (Lives of Others, The (Das leben der Anderen) (2006)): 4.491
  2859 (Stop Making Sense (1984)): 4.486
  1089 (Reservoir Dogs (1992)): 4.479
```

Command:

```
./gradlew recommendBlended -PuserId=320
```

Output:

```
recommendations for user 320:
  318 (Shawshank Redemption, The (1994)): 0.999
  858 (Godfather, The (1972)): 0.999
  58559 (Dark Knight, The (2008)): 0.995
  1089 (Reservoir Dogs (1992)): 0.995
  7153 (Lord of the Rings: The Return of the King, The (2003)): 0.994
```

```
1258 (Shining, The (1980)): 0.989
1210 (Star Wars: Episode VI - Return of the Jedi (1983)): 0.989
79132 (Inception (2010)): 0.988
1080 (Monty Python's Life of Brian (1979)): 0.986
4973 (Amelie (Fabuleux destin d'Am?lie Poulain, Le) (2001)): 0.982
```

Submitting

As with the Course 1 assignments, you will submit a compiled `jar` file to be graded. To create this file, please use the pre-created archive functionality in the Gradle build:

```
./gradlew prepareSubmission
```

This will ensure that your submission contains all required files. It will produce a submission file in `build/distributions`.

Submit the `uu-submission.jar` file to Coursera for greading.

Grading

Your grade will be based on your output over randomly selected users:

- 75% for the scorer ordering items correctly
- 25% for computing the correct scores (within an error threshold)

Further Exploration

Try different similarity functions and normalization strategies to see what difference they make in user predictions.