User-user Programming Assignment

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 Teaching website (links to relevant documentation and the LensKit tutorial video)
- JavaDoc for included code

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).
- An IDE; I recommend IntelliJ IDEA Community Edition.

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.
- 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 for item 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:

$$cos(u,i) = \frac{\vec{u} \cdot \vec{i}}{\|\vec{u}\|_2 \|\vec{i}\|_2} = \frac{\sum_t u_t i_t}{\sqrt{\sum_t u_t^2}\sqrt{\sum_t i_t^2}}$$

- 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.

Example Output

Command:

~~~
```
./gradlew predict -PuserId=42 -PitemIds=11,414,424,812
```
~~~

Output:

```
predictions for user 42:
  11 (Star Wars: Episode IV - A New Hope (1977)): 4.169
  414 (Batman Forever (1995)): 3.300
  424 (Schindler's List (1993)): 4.254
  812 (Aladdin (1992)): 4.020
```

Command:

~~~
```
./gradlew recommend -PuserId=42,91
```
~~~

Output:

```
recommendations for user 42:
  13 (Forrest Gump (1994)): 4.292
  424 (Schindler's List (1993)): 4.254
  629 (The Usual Suspects (1995)): 4.239
  857 (Saving Private Ryan (1998)): 4.169
  11 (Star Wars: Episode IV - A New Hope (1977)): 4.169
  105 (Back to the Future (1985)): 4.153
  807 (Seven (a.k.a. Se7en) (1995)): 4.103
  280 (Terminator 2: Judgment Day (1991)): 4.025
  812 (Aladdin (1992)): 4.020
  9802 (The Rock (1996)): 4.014
recommendations for user 91:
  238 (The Godfather (1972)): 4.519
  550 (Fight Club (1999)): 4.459
  424 (Schindler's List (1993)): 4.449
  278 (The Shawshank Redemption (1994)): 4.379
  24 (Kill Bill: Vol. 1 (2003)): 4.308
  807 (Seven (a.k.a. Se7en) (1995)): 4.295
  194 (Amelie (2001)): 4.220
  77 (Memento (2000)): 4.189
  393 (Kill Bill: Vol. 2 (2004)): 4.113
  38 (Eternal Sunshine of the Spotless Mind (2004)): 4.103
```

Submitting

As with all programming assignments in this class, you are strongly encouraged to work with a partner. Not only does working with a partner make the assignment easier and more educational, it also makes it faster to grade! When submitting the assignment, please include a name and ID number for both you and your partner. Only one submission is necessary. Submit your code as a zip file to the TA (taijala@cs.umn.edu).

To create this zip 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`.

Further Exploration

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