Non-Personalized Recommender Assignment

In this assignment, you will implement some non-personalized recommenders. In particular, you will implement raw and damped item mean recommenders and simple and advanced association rule recommenders.

You will implement these recommenders in the LensKit toolkit.

Downloads and Resources

- Project template (from Coursera)
- LensKit for Teaching website (links to relevant documentation)
- JavaDoc for included code
- Fastutil API docs documents the Fastutil optimized data structure classes that are used in portions of LensKit.

The project template contains support code, the build file, and the input data that you will use.

Input Data

The input data contains the following files:

- `ratings.csv` contains user ratings of movies
- `movies.csv` contains movie titles
- `movielens.yml` is a LensKit data manifest that describes the other input files

Getting Started

To get started with this assignment, unpack the template and import it in to your IDE as a Gradle project. The assignment video demonstrates how to do this in IntelliJ IDEA.

Mean-Based Recommendation

The first two recommenders you will implement will recommend items with the highest average rating.

With LensKit's scorer-model-builder architecture, you will just need to write the recommendation logic once, and you will implement two different mechanisms for computing item mean ratings.

You will work with the following classes:

- `MeanItemBasedItemRecommender` (the *item recommender*) computes top-N recommendations based on mean ratings. You will implement the logic to compute such recommendation lists.
- `ItemMeanModel` is a *model class* that stores precomputed item means. You will not need to modify this class, but you will write code to construct instances of it and use it in your item recommender implementation.
- `ItemMeanModelProvider` computes item mean ratings from rating data and constructs the model. It computes raw means with no damping.
- `DampedItemMeanModelProvider` is an alternate builder for item mean models that computes damped means instead of raw means. It takes the damping term as a parameter. The configuration file we provide you uses a damping term of 5.

There are `// TODO` comments in all places where you need to write new code.

Computing Item Means

Modify the `ItemMeanModelProvider` class to compute the mean rating for each item.

Recommending Items

Modify the `MeanItemBasedItemRecommender` class to compute recommendations based on item mean ratings. For this, you need to:

1. Obtain the mean rating for each item
2. Order the items in decreasing order
3. Return the N highest-rated items

Computing Damped Item Means

Modify the `DampedItemMeanModelProvider` class to compute the damped mean rating for each item. This formula uses a damping factor α, which is the number of 'fake' ratings at the global mean to assume for each item. In the Java code, this is available as the field `damping`.

The damped mean formula, as you may recall, is:

$$s(i) = \frac{\sum_{u \in U_i} r_{ui} + \alpha\mu}{|U_i| + \alpha}$$

where μ is the *global* mean rating.

Example Outputs

To help you see if your output is correct, we have provided the following example correct values:

ID	Title	Mean	Damped Mean
2959	*Fight Club*	4.259	4.252
1203	*12 Angry Men*	4.246	4.227

Association Rules

In the second part of the assignment, you will implement two versions of an association rule recommender.

The association rule implementation consists of the following code:

- `AssociationItemBasedItemRecommender` recommends items using association rules. Unlike the mean recommenders, this recommender uses a *reference item* to compute the recommendations.
- `AssociationModel` stores the association rule scores between pairs of items. You will not need to modify this class.
- `BasicAssociationModelProvider` computes an association rule model using the basic association rule formula ($P(X \wedge Y)/P(X)$).
- `LiftAssociationModelProvider` computes an association rule model using the lift formula ($P(X \wedge Y)/P(X)P(Y)$).

Computing Association Scores

Like with the mean-based recommender, we pre-compute product association scores and store them in a model before recommendation. We compute the scores between *all pairs* of items, so that the model can be used to score any item. When computing a single recommendation from the command line, this does not provide much benefit, but is useful in the general case so that the model can be used to very quickly compute many recommendations.

The `BasicAssociationModelProvider` class computes the association rule scores using the following formula:

$$P(i|j) = \frac{P(i \wedge j)}{P(j))} = \frac{|U_i \cap U_j|/|U|}{|U_j|/|U|}$$

In this case, j is the *reference* item and i is the item to be scored.

We estimate probabilities by counting: $P(i)$ is the fraction of users in the system who purchased item i; $P(i \wedge j)$ is the fraction that purchased both i and j.

Implement the association rule computation in this class.

Computing Recommendations

Implement the recommendation logic in `AssociationItemBasedItemRecommender` to recommend items related to a given reference item. As with the mean recommender, it should compute the top N recommendations and return them.

Computing Advanced Association Rules

The `LiftAssociationModelProvider` recommender uses the *lift* metric that computes how much more likely someone is to rate a movie i when they have rated j than they would have if we do not know anything about whether they have rated j:

$$s(i|j) = \frac{P(j \wedge i)}{P(i)P(j)}$$

Example Outputs

Following is the correct output for the basic association rules with reference item 260 (*Star Wars*), as generated with `./gradlew runBasicAssoc -PreferenceItemm=260`:

```
2571 (Matrix, The (1999)): 0.916
1196 (Star Wars: Episode V - The Empire Strikes Back (1980)): 0.899
4993 (Lord of the Rings: The Fellowship of the Ring, The (2001)): 0.892
1210 (Star Wars: Episode VI - Return of the Jedi (1983)): 0.847
356 (Forrest Gump (1994)): 0.843
5952 (Lord of the Rings: The Two Towers, The (2002)): 0.841
7153 (Lord of the Rings: The Return of the King, The (2003)): 0.830
296 (Pulp Fiction (1994)): 0.828
1198 (Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)): 0
480 (Jurassic Park (1993)): 0.789
```

And lift-based association rules for item 2761 (*The Iron Giant*):

```
631 (All Dogs Go to Heaven 2 (1996)): 4.898
2532 (Conquest of the Planet of the Apes (1972)): 4.810
3615 (Dinosaur (2000)): 4.546
1649 (Fast, Cheap & Out of Control (1997)): 4.490
340 (War, The (1994)): 4.490
1016 (Shaggy Dog, The (1959)): 4.490
```

```
2439 (Affliction (1997)): 4.490
332 (Village of the Damned (1995)): 4.377
2736 (Brighton Beach Memoirs (1986)): 4.329
3213 (Batman: Mask of the Phantasm (1993)): 4.317
```

Running your code

The Gradle build file we have provided is set up to automatically run all four of your recommenders. The following Gradle targets will do this:

- `runMean` runs the raw mean recommender
- `runDampedMean` runs the damped mean recommender
- `runBasicAssoc` runs the basic association rule recommender
- `runLiftAssoc` runs the advanced (lift-based) association rule recommender

You can run these using the IntelliJ Gradle runner (open the Gradle panel, browse the tree to find a task, and double-click it), or from the command line:

```
./gradlew runMean
```

The association rule recommenders can also take the reference item ID on the command line as a `referenceItem` parameter. For example:

```
./gradlew runLiftAssoc -PreferenceItem=1
```

The IntelliJ 'Run Configuration' dialog will allow you to specify additional 'script parameters' to the Gradle invocation.

Debugging

If you run the Gradle tasks using IntelliJ's Gradle runner, you can run them under the debugger to debug your code.

The Gradle file also configures LensKit to write log output to log files under the `build` directory. If you use the SLF4J logger (the `logger` field on the classes we provide) to emit debug messages, you can find them there when you run one of the recommender tasks such as `runDampedMean`.

Submitting

You will submit a compiled `jar` file containing your solution. To prepare your project for submission, run the Gradle `prepareSubmission` task:

```
./gradlew prepareSubmission
```

This will create file `nonpers-submission.jar` under `build/distributions` that contains your final solution code in a format the grader will understand. Upload this `jar` file to the Coursera assignment grader.

Grading

Your grade for each part will be based on two components:

- Outputting items in the correct order: 75%
- Computing correct scores for items (within an error tolerance): 25%

The parts themselves are weighted equally.