COMP47580 – Recommender Systems & Collective Intelligence

Non-personalised Recommender Framework

The following provides a description of the dataset and the non-personalised recommender framework that are used in this part of the assignment. To help get you started, the framework contains an implementation of a non-personalised algorithm, in which the similarity between two movies is given by the Jaccard index calculated over the set of genres associated with each movie. This algorithm is ready to run. The framework is designed to be extensible and it readily facilitates the integration of the additional similarity metrics which you need to implement in this part of the assignment.

Dataset

The assignment dataset consists of:

- 95,379 ratings from 1,135 users on 1,660 movies.
- Ratings are based on a 0.5–5.0 point scale, in increments of 0.5, where 5.0 is the maximum rating and 0.5 is the minimum rating.

The data was obtained from the MovieLens system (https://movielens.org/). This dataset includes the following files.

Training Set (train.txt)

The training set consists of a randomly selected 80% of the user-item ratings from the dataset. The file format is as follows: each line consists of a `<user_id,item_id,rating>` rating tuple. For example, the first line in the file is: `16387,3360,4.5`, which indicates that user `16387` assigned a rating of `4.5` to item `3360`.

Test Set (test.txt)

This file consists of 10% of the ratings from the dataset (these ratings are different from those contained in the training set).

Note that this file is not used in this assignment on non-personalised recommender algorithms.

Item Descriptions (movies-sample.txt)

This file contains information about the movies. Each line of this file represents a movie which is formatted as follows: `<item_id,title,genre_1|genre_2|…|genre_N>`. For example, the first line in this file is: `1,Toy Story (1995), Adventure|Animation|Children|Comedy|Fantasy` which shows the set of genres associated with the movie with ID `1` and title `Top Story (1995)`.

The movie IDs correspond to those used in the training and test sets.

Genome Tags (genome-tags.csv) and Genome Scores (genome-scores-sample.txt)

The tag genome contains tag relevance scores for movies. Each movie has a value for every one of the 1,128 tags in the genome.

The tag genome encodes how strongly movies exhibit particular properties represented by tags (atmospheric, thought-provoking, realistic, etc.). The tag genome was computed using a machine learning algorithm on user-contributed content including tags, ratings, and textual reviews.

The genome is split into two files. The file "genome-scores-sample.txt" contains movie-tag relevance data in the following format: `<movieId,tagId,relevance>`. The movie IDs correspond to those used in the training and test sets.

The second file, "genome-tags.csv", provides the tag descriptions for the tag IDs in the genome file, in the following format: `<tag_id,tag>`.

Non-personalised Recommender Framework

A framework to implement and evaluate non-personalised recommender algorithms has been created and made available to you. Further, a non-personalised algorithm (in which movie similarity is computed using Jaccard over movie genres) has been implemented using this framework and is ready to run and generate recommendations. In what follows, a brief description of this framework is provided along with instructions on how to execute the code.

Non-personalised Recommender Algorithm

This algorithm makes recommendations for a given *target item*. The key step in the algorithm is to calculate the similarities between items. Non-personalised recommender algorithms are characterised by the similarity metric used.

Calculating item-item similarity: the framework contains an implementation of the *Genre* similarity metric (class `GenreMetric` in package `alg.np.similarity.metric`). Note that this class implements the interface `SimilarityMetric` (also in package `alg.np.similarity.metric`). All similarity metrics should implement the `SimilarityMetric` interface. This way, it is very easy to change the code to switch to a different similarity metric. For example, if the code says:

```
SimilarityMetric metric = new GenreMetric(reader);
```

then, to switch to, for example, the *Genome* similarity metric, you just need to change this line of code as follows:

```
SimilarityMetric metric = new GenomeMetric(reader);
```

In this assignment, your task is to implement a number of different similarity metrics.

Executing the Non-personalised Recommender Algorithm

Class `ExecuteNP` in package `alg.np` executes the algorithm. To begin, the paths and filenames of the item description file, genome scores file, training and test sets are specified as follows:

```
// set the paths and filenames of the item file, genome scores file, train file and
// test file ...
String folder = "ml-20m-2018-2019";
String itemFile = folder + File.separator + "movies-sample.txt";
String itemGenomeScoresFile = folder + File.separator + "genome-scores-sample.txt";
String trainFile = folder + File.separator + "train.txt";
String testFile = folder + File.separator + "test.txt";
```

Next, an instance of the `DatasetReader` class (in package `util.reader`) is created to read in the above files. An instance of the similarity metric (class `GenreMetric` in package `alg.np.similarity.metric`) is then created; the constructor takes as argument an instance of the `DatasetReader` class. Lastly, an instance of the non-personalised algorithm (class `NonPersonalisedRecommender` in package `alg.np`) is created; the constructor takes as arguments instances of the `DatasetReader` and `SimilarityMetric` classes described above.

```
DatasetReader reader = new DatasetReader(itemFile, itemGenomeScoresFile,
                              trainFile, testFile);
SimilarityMetric metric = new GenreMetric(reader);
NonPersonalisedRecommender alg = new NonPersonalisedRecommender(reader, metric);
```

The final step is to evaluate the algorithm by making recommendations for each item in the dataset. This is achieved by creating an instance of the `Evaluator` class (in package `util.np.evaluator`). This class contains a number of methods to evaluate the performance of the algorithm using various criteria.

The following code is included in class `ExecuteNP`. It displays the top 3 recommendations which are made for a sample of 5 movies. This class is ready to run.

```
// display the top-k recommendations for five items
int k = 3; // the number of recommendations to be made for each target item
Evaluator eval = new Evaluator(alg, reader, k);
Map<Integer,Item> items = reader.getItems();
int[] itemIds = {3668, 3639, 3578, 87232, 3681};
for (int itemId: itemIds) {
      Item item = items.get(itemId);
      eval.printRecs(item);
}
```