COMP 40320 Recommender Systems

Case-based Recommender Assignment
Code and Dataset Description

The following provides a description of the dataset and the case-based recommender framework that is available to you for use in the assignment. To help get you started, the framework contains an implementation of a basic case-based recommender algorithm, which is ready to run. The framework is designed to be extensible; for example, it readily facilitates the integration of new case and feature similarity metrics etc. The document also describes the output produced by the algorithm implemented in the framework.

Dataset

The assignment dataset consists of:

- 39609 ratings from 1000 users for 1073 movies. Ratings are based on a 0.5–5.0 point scale (in 0.5 point intervals), where 5 is the maximum rating and 0.5 is the minimum rating.
- Movie metadata (title, genre(s), director(s) actor(s))

The dataset was obtained from Flixster (http://www.flixster.com).

Three files have been created from this dataset for use in the assignment. The content and format of these files are described below.

Training Set (trainData.txt)

All recommendations are based on this data, which consists of 80% of each user's set of ratings. The file format is as follows: each line consists of a `<userId movieId rating review_text>` tuple (tab delimited). For example, the first entry in the file is:

```
170827209       3463    5.0     very funny and it has sumthings in it that we can…
```

which indicates that user `170827209` assigned a rating of `5.0` to movie `3463`. The review text provided by the user for the movie is also included (`very funny and it has sumthings in it that we can…`).

Test Set (testData.txt)

This file consists of 20% of each user's set of ratings (which are different from those contained in the training set). These are the ratings that are used to compute and compare the recommendation performance (using, for example, precision and recall) of different case-based recommender algorithms. The file format is as above.

Movie Metadata (movies.txt)

This file contains movie metadata. The file format is as follows: each line consists of a `<movieId title genre(s) director(s) actor(s)>` tuple (tab delimited). For example, the first entry in the file is:

```
4402    A Nightmare on Elm Street      Horror, Mystery        Wes Craven      Heather
Langenkamp, Johnny Depp, Robert Englund, John Saxon
```

The movie ids correspond to those used in the training and test sets. Each movie can have multiple genres, directors and actors.

Case-based Recommender Framework

A framework to implement and evaluate case-based recommender algorithms has been created and made available to you. Further, a fully functioning basic case-based recommender algorithm 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 and how to perform evaluation.

Case-based Recommender Algorithm

This algorithm generates ranked lists of recommendations for users. To configure the algorithm a `CaseSimilarity` object (package `alg.cases.similarity`) and a `Recommender` object (package `alg.recommender`) need to be instantiated.

The framework contains examples of two case similarity metrics (classes `OverlapCaseSimilarity` and `JaccardCaseSimilarity` in package `alg.cases.similarity`). Note that these classes implement the interface `CaseSimilarity` (also in package `alg.cases.similarity`). Any new case similarity metric class you create should also implement the `CaseSimilarity` interface. Using this approach, it is very easy to change your code to switch to a different case similarity metric. For example, if your code says:

```
CaseSimilarity caseSimilarity = new OverlapCaseSimilarity();
       ...
double sim = caseSimilarity.getSimilarity(c1, c2);
```

To switch to the `JaccardCaseSimilarity` case similarity metric you just need to change the first line:

```
CaseSimilarity caseSimilarity = new JaccardCaseSimilarity();
       ...
double sim = caseSimilarity.getSimilarity(c1, c2);
```

The framework also contains an implementation of one case-based recommender algorithm (class `MeanRecommender` in package `alg.recommender`). The `MeanRecommender` class extends the `Recommender` abstract class (also in package `alg.recommender`). As above, this approach ensures that any new case-based recommender algorithms can be easily integrated into the framework by extending this abstract class.

Executing the Case-based Recommender Algorithm

Class `Execute` in package `alg` executes the algorithm. To begin, the paths and filenames of the training, test and movie metadata files are specified as follows:

```
// set the paths and filenames of the training, test and movie metadata files …
String trainFile = "dataset" + File.separator + "trainData.txt";
String testFile = "dataset" + File.separator + "testData.txt";
String movieFile = "dataset" + File.separator + "movies.txt";
```

Next, an instance of the DatasetReader (in package `util.reader`) class is created to read in the data from the above files:

```
DatasetReader reader = new DatasetReader(trainFile, testFile, movieFile);
```

Then the case similarity metric and recommender are set as follows:

```
// configure the case-based recommendation algorithm - set the case similarity and
recommender
CaseSimilarity caseSimilarity = new OverlapCaseSimilarity();
Recommender recommender = new MeanRecommender(caseSimilarity, reader);
```

In the above, the constructor of the Recommender class takes as arguments instances of the `CaseSimilarity` and `DatasetReader` classes described previously.

The final step is to evaluate the case-based recommender algorithm by generating recommendations for test users. This is achieved by creating an instance of the `Evaluator` class (in package `util.evaluator`). This class is used to compute the precision and recall provided by the algorithm for various recommendation list sizes (list size is specified by the variable `topN`).

```
// evaluate the case-based recommender
Evaluator eval = new Evaluator(recommender, reader);

System.out.println("topN Recall Precision");
for(int topN = 5; topN <= 50; topN += 5)
      System.out.println(topN + " " + eval.getRecall(topN) + " " +
      eval.getPrecision(topN));
```