Collective Intelligence – Recommender System Practical Notes

Michael O'Mahony
20th February 2015

The following provides a description of the dataset and the CF framework that is available to you for use in the practicals. To help get you started, the framework contains an implementation of the user-based CF algorithm, which is ready to run. The framework is designed to be extensible; for example, it readily facilitates the integration of new similarity metrics, neighbourhood formation approaches etc. The document also describes the output produced by algorithms implemented in the framework and, in particular, the file that you need to upload to the online submission system.

Dataset

The practical dataset consists of:
* 100,000 ratings from 943 users on 1682 movies.
* Each user has rated at least 20 movies.
* Users and items are numbered consecutively from 1.
* Ratings are based on a 1-5 point scale, where 5 is the maximum rating and 1 is the minimum rating.

The data was obtained from the MovieLens system (http://movielens.umn.edu). The data was collected during the seven-month period from September 19th, 1997 through April 22nd, 1998.

This dataset is used to create the following files which you will use during the practicals.

Training Set (u.train)

All predictions to be generated are based on this data, which consists of 80% of each user's original ratings. The file format is as follows: each line consists of a `<user_id item_id rating>` tuple. For example, the first line in the file is: `1 45 5` which indicates that user `1` assigned a rating of `5` to item `45`.

Probe Set (u.probe)

This file consists of 10% of each user's original ratings (which are different from those present in the training set). These are the ratings for which you need to make a prediction. The file format is as above. The actual ratings are also supplied; this allows you to generate predictions using different CF algorithms and to compare the performance of each algorithm by, for example, computing predictive accuracy (using RMSE) and coverage.

Test Set (u.test)

This file is similar to the probe set, and again consists of 10% of each user's ratings (again, these ratings are different to those contained in both the training and probe sets). However, the actual ratings are **not** provided in this file. Once you have experimented with and evaluated a variety of CF algorithms using the probe set, predictions for the test set should then be generated and uploaded to the sonline submission system – the RMSE on these predictions is then used when compiling the leader board.

Item Descriptions (u.item)

Finally, a file containing information about the items (movies) is provided; this is a '|' separated list of:

```
movie id | movie title | release date | video release date | IMDb URL | unknown |
Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama |
Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War
| Western |
```

The last 19 fields are the genres, a 1 indicates the movie is of that genre, a 0 indicates it is not; movies can be in several genres at once. The movie ids correspond to those used in the training, test and probe sets.

CF Framework

A framework to implement and evaluate CF algorithms has been created and made available to you. Further, a fully functioning user-based CF algorithm has been implemented using this framework and is ready to go and generate predictions. 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.

User-based CF Algorithm

This algorithm calculates a predicted rating for a *target user* on a *target item*. There are three key components to the user-based CF algorithm:

Calculating user similarity: the framework contains an implementation of the *Cosine* similarity metric (class `Cosine` in package `similarity.metric`). Note that this class implements the interface `SimilarityMetric` (also in package `similarity.metric`). If you need to implement a new similarity metric (e.g. *Pearson* similarity), ensure that it also implements the `SimilarityMetric` interface. If you do it this way, it will be very easy to change your code to switch to a different similarity metric. For example, if your code says:

```
SimilarityMetric metric = new Cosine();
        ...
double sim = metric.getSimilarity(p1, p2);
```

then to switch to, for example, the *Pearson* similarity metric, you just need to change the first line:

```
SimilarityMetric metric = new Pearson();
        ...
double sim = metric.getSimilarity(p1, p2);
```

Forming a neighbourhood: once the similarity between the target user and all other users who have rated the target item have been calculated, the next step is to form the target user's neighbourhood. The framework contains an implementation of the *Nearest Neighbour* approach (class `kNearestNeighbourhood` in package `alg.ub.neighbourhood`). In this approach, the *k* most similar users to the target user are selected as neighbours.

In a similar manner to the above, the `kNearestNeighbourhood` class implements the `Neighbourhood` interface (also in package `alg.ub.neighbourhood`); any new neighbourhood formation approaches can be easily integrated into the framework if these also implement the `Neighbourhood` interface.

Choosing a predictor: the final step is to compute the predicted rating for the target user on the target item. The framework contains a simple implementation (class `MeanPredictor` in package `alg.ub.predictor`), where the predicted rating is computed by taking an average of the neighbours' ratings for the target item.

The `MeanPredictor` class implements the `Predictor` interface (in package `alg.ub.predictor`); again, this approach ensures that any new predictors can be easily integrated into the framework by implementing this interface.

Executing the User-based CF Algorithm

Class `ExecuteUB` in package `alg.ub` executes the algorithm. To begin, the similarity metric, neighbourhood formation approach and the predictor are set as follows:

```
// configure the user-based CF algorithm - set the predictor, neighbourhood and
similarity metric ...
Predictor predictor = new MeanPredictor();
Neighbourhood neighbourhood = new kNearestNeighbourhood(10);
SimilarityMetric metric = new Cosine();
```

Next, the paths and filenames of the item file, training and test (or probe) sets and the output file are specified:

```
// set the paths and filenames of the item file, train file and test file ...
String itemFile = "dataset" + File.separator + "u.item";
String trainFile = "dataset" + File.separator + "u.train";
String testFile = "dataset" + File.separator + "u.probe";

// set the path and filename of the output file ...
String outputFile = "results" + File.separator + "predictions.txt";
```

Next, an instance of the DatasetReader (in package `util.reader`) class is created to read in the above data sets. Then an instance of the user-based CF algorithm (class `UserBasedCF` in package `alg.ub`) is created; the constructor takes as arguments instances of the `Predictor`, `Neighbourhood`, `SimilarityMetric` and `DatasetReader` classes described above.

```
DatasetReader reader = new DatasetReader(itemFile, trainFile, testFile);
UserBasedCF ubcf = new UserBasedCF(predictor, neighbourhood, metric, reader);
```

The final step is to evaluate the algorithm by generating predictions for the test set (or probe set) items. This is achieved by creating an instance of the `Evaluator` class (in package `util.evaluator`). This class is used to calculate the RMSE and coverage provided by the algorithm, as well as creating the output file. The output file contains the individual predictions calculated for each of the test set (or probe set) items. The format of this file is as follows: each line consists of a `<user_id item_id actual_rating predicted_rating>` tuple if the probe set is specified or a `<user_id item_id predicted_rating>` tuple if the test set is specified. This latter output file is the one that you need to upload to the online submission system.

```
Evaluator eval = new Evaluator(ubcf, reader.getTestData());
eval.writeResults(outputFile);

Double RMSE = eval.getRMSE();
if(RMSE != null) System.out.println("RMSE: " + RMSE);

for(int i = 1; i <= 5; i++)
{
	RMSE = eval.getRMSE(i);
	if(RMSE != null) System.out.println("RMSE (true rating = " + i + "): " + RMSE);
}

double coverage = eval.getCoverage();
System.out.println("coverage: " + coverage + "%");
```