COMP40320 – Recommender Systems

Michael O'Mahony
19th February 2016

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 a basic implementation of the item-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 the algorithm implemented in the framework.

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). 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`.

Test Set (u.test)

This file also consists of a percentage of each user's original ratings (these ratings are different from those contained 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 based on prediction accuracy (RMSE) and coverage.

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 and test sets. Note that this file is not used in the standard item-based CF algorithm (but could be used, for example, to compute item-item similarities based on genres).

CF Framework

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

Item-based CF Algorithm

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

Calculating item-item similarity: the framework contains an implementation of the *Pearson* similarity metric (class `PearsonMetric` 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. *Cosine* 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 PearsonMetric();
        ...
double sim = metric.getSimilarity(p1, p2);
```

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

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

Forming item neighbourhoods: once the pairwise similarity between all items has been calculated, the next step is to form a neighbourhood for each item. The framework contains an implementation of the *Nearest Neighbour* approach (class `NearestNeighbourhood` in package `alg.ib.neighbourhood`). In this approach, for each item, the *k* most similar items are selected as neighbours.

Note that the `NearestNeighbourhood` class extends the abstract class `Neighbourhood` (also in package `alg.ib.neighbourhood`); any new neighbourhood formation approaches can be easily integrated into the framework if they also extend the `Neighbourhood` abstract class (please follow this approach).

Choosing a predictor: the final step is to compute the predicted rating for the target user on the target item. The framework contains a basic implementation – referred to as the *Simple Average Approach* in the lecture notes. The implementation of this approach can be found in class `SimpleAveragePredictor` in package `alg.ib.predictor`.

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

Executing the Item-based CF Algorithm

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

```
// configure the item-based CF algorithm - set the predictor, neighbourhood and
similarity metric ...
Predictor predictor = new SimpleAveragePredictor();
Neighbourhood neighbourhood = new NearestNeighbourhood(100);
SimilarityMetric metric = new PearsonMetric();
```

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

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

// 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 item-based CF algorithm (class `ItemBasedCF` in package `alg.ib`) 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);
ItemBasedCF ibcf = new ItemBasedCF(predictor, neighbourhood, metric, reader);
```

The final step is to evaluate the algorithm by generating predictions for the test 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. In addition to displaying RMSE over all test set items, the RMSE when the actual test set ratings are 1, 2, 3, 4 and 5 are also displayed.

Note that an output file (saved in folder `results`), which contains the individual predictions calculated for each of the test set items, is also created. The format of this file is as follows: each line consists of a `<user_id item_id actual_rating predicted_rating>` tuple. RMSE is calculated over the data in this file.

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

Double RMSE = eval.getRMSE();
if(RMSE != null) System.out.printf("RMSE: %.6f\n", RMSE);

for(int i = 1; i <= 5; i++)
{
	RMSE = eval.getRMSE(i);
	if(RMSE != null)
		System.out.printf("RMSE (true rating = %d): %.6f\n", i, RMSE);
}

double coverage = eval.getCoverage();
System.out.printf("coverage: %.2f%s\n", coverage, "%");
```