COMP40320 – Recommender Systems

Michael O'Mahony
7th February 2017

The following provides a description of the dataset and the CF framework that are used in this assignment. 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 assignment dataset consists of:

- 137,938 ratings from 1,674 users on 1,245 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 (http://movielens.umn.edu). This dataset includes the following files which are used in the assignment.

Training Set (train.txt)

All predictions to be generated are based on this data, which consists of a randomly selected 90% of the 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: `82,74787,4.0`, which indicates that user `82` assigned a rating of `4.0` to item `74787`.

Test Set (test.txt)

This file consists of the remaining 10% of ratings from the dataset (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. Since the actual rating values are provided, 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 (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: `73319,Leap Year (2010),Comedy|Romance` which indicates that there are two genres (`Comedy` and `Romance`) associated with the movie with ID `73319` and title `Leap Year (2010)`.

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).

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

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. However, genome scores will be used in this assignment to implement a content-based recommender.

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_ML20M` 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(10);
SimilarityMetric metric = new PearsonMetric();
```

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

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

// 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, itemGenomeScoresFile, 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.

Note that the output file (saved in folder `results`), which contains the individual predictions calculated for each of the test set ratings, 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());

// Write to output file
eval.writeResults(outputFile);

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

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