OnlineCF

Online Collaborative Filtering

Contents

Overview.

- The problem of collaborative filtering is typically defined as the task of inferring consumer preferences: Given an observed set of product preferences for a set of users, can we accurately predict the unobserved preferences?
- Notation. We define a collaborative filtering problem as a distribution D over triples (a, b, r) A × B × R where A and B are finite sets of size n and m respectively. We are given a set M of triples {(a, b, r)} and want to find a function f (a, b) which minimizes the expected squared error
- Typically, we think of A as our set of users, B as our set of products, and r as user a's "rating" of product b. In most movie recommendation datasets, r is a number in {1, 2, 3, 4, 5} as in the number of "stars", although in other settings we may only be given r {0, 1}, as in liked/disliked.

Usage.

- Basic overview of usage steps are:
- Convert. convert input data into OnlineCF compatible format.
- Configuration and Train. set parameters for training
- Load
- Predict

Convert.

- Since, currently, we support one input path, we need to convert input data and combine set of triples, item and user features into one file. In order to implement custom parsing of input data, use InputConverter class Below is example for Movie Lens dataset converter.

```
MovieLensConverter converter = new MovieLensConverter();
converter.convert(pathToPreferences, pathToMovieGenres, convertedOutputPath);
```

Configuration and Train.

- In order to achieve good performance in prediction we need to configure iteration count, matrix rank and matrix factorization update functions.

```
OnlineCF recommender = new OnlineCF();
recommender.setInputPreferences(convertedOutputPath);
recommender.setIteration(150);
recommender.setMatrixRank(3);
recommender.setSkipCount(1); // after how many steps we should synchronize values in each task
recommender.setUpdateFunction(MeanAbsError.class);
recommender.setOutputPath(outputFileName);
recommender.train();
```

Load.

- After training, model will be saved into output file by default In order to use prediction functions we need to load it.

```
recommender.load(pathToTrainedModel, false);
```

Predict.

```
// estimate score
double estimatedScore = recommender.estimatePreference(userId, itemId);

// estimate user similarities
double userSimilarity = recommender.calculateUserSimilarity(user1, user2);
// Pair<K, V> - where K predicted similar user, V predicted similarity score
List<Pair<Long, Double>> similarUsers = recommender.getMostSimilarUsers(userId, count);

// estimate item similarities
double itemSimilarity = recommender.calculateItemSimilarity(item1, item2);
// Pair<K, V> - where K predicted similar item, V predicted similarity score
List<Pair<Long, Double>> similarItems = recommender.getMostSimilarItems(itemId, count);
```

Complementary functions

- There some classes which can be useful to know while using Online Collaborative Filtering.
- InputConverter. For parsing input data and converting into OnlineCF compatible format. (see MovieLensConverter)
- OnlineUpdate.Function. For matrix factorization functions It will be used while training and estimating user preference (see MeanAbsError)

References

- Online Collaborative Filtering. Jacob Abernethy, Kevin Canini, John Langford, Alex Simma http://canini.me/research_files/OnlineCollaborativeFiltering.pdf