A Practical Guide to the Universal Recommender

Jérôme Kunegis

July 22, 2010

Abstract

This is a short how-to guide for the Universal Recommender [4], a Java programming library providing data structures and algorithms for recommender systems. The Universal Recommender supports link prediction, rating prediction and recommendation in arbitrary multirelational networks. This document gives a brief overview of its main functions, and is supplemented by the actual program library documentation.

1 Introduction

The Universal Recommender is a programming library that implements data structures and algorithms for recommender systems. The basic ideas behind the Universal Recommender are described in the 2009 white paper *The Universal Recommender* [4].

The Universal Recommender is a programming library written in Java. It can be natively imported using the Maven framework. The Universal Recommender is available as the package `de.dailab.recommender`. This guide is intended to provide a hands-on introduction to using the Universal Recommender. It should be used in conjunction with the Javadoc documentation. For readability, this guide does not mention the package in which classes are located–they can be inferred easily.

The guide begins with a definition of the term *recommender*. Then, key concepts are described in increasing order of complexity: datasets, predictors, recommenders and evaluation methods. We provide a glossary of recommendation terms at the end.

2 Motivation: What is a Recommender?

A recommender is an algorithm that, given a node of a network as input, returns a ranked list of other nodes of the same network. The networks in question typically contain nodes such as users, items and features which are connected rating, friendship and other links.

Recommenders are typically classified by the input and output they take (e.g. user-user or user-item recommenders), by the specific algorithm used (e.g.

neighborhood-based or latent) and by the type of intermediary data that is computed (e.g. memory-based or model-based). The Universal Recommender can be used to implement any kind of recommendation algorithm for any multirelational dataset, as explained in the next section.

3 Datasets

To use a recommender, a dataset is needed. In the Universal Recommender, datasets are represented by the class `Dataset`.

A dataset is a network of entities connected by relationships. Both entities and relationships have a type; these are called entity types and relationship types. Additionally, entities can have metadata attached to them, and relationships can have weights.

The classes `EntityType` and `RelationshipType` represent entity and relationship types, respectively. Both classes simply contain a string.

In a `Dataset`, the entities and relationships are grouped by their types. A `Dataset` thus contains several `EntitySet`s and `RelationshipSet`s.

In an `EntitySet`, the entities of a given type are represented by contiguous integer IDs starting at zero. Single entities can be represented by the class `Entity`, which contains an `EntityType` and an integer ID.

In a `RelationshipSet`, all relationships of a given type are represented by a matrix. Each relationship set specifies which entity types are connected. If these are the same, the relationship type is unipartite and the matrix is square. If they are different, the relationship type is bipartite and the matrix is rectangular. For unipartite relationship types, there is a distinction between the symmetric and asymmetric case. In practice, the symmetric case is very rare. Relationships can also have weights. Each relationship set specifies the range of possible weights: unweighted, positively weighted (only positive weights), signed (positive and negative weights) and weighted (any weights). The difference between the signed and weighted case is subtle: In the signed case, the weight value zero has the specific meaning of "neutral", whereas this is not the case for weighted relationship types. A signed relationship type would for instance be the friend/foe relations in the Slashdot Zoo [3]. A weighted relationship type would be the five star rating scale of Netflix [1].

Each `RelationshipSet` contains an adjacency or biadjacency matrix that includes all relationships of the given type. Matrices are represented by the interface `Matrix`. All implementations of it are sparse, memory-held matrices. No third-party matrix library is used.

To get started quickly, specific datasets can be tried out. The following subsections describe the various ways of loading datasets into memory.

3.1 Semantic Stores

The Universal Recommender supports reading datasets from so-called *semantic stores*. A semantic store is a repository of semantic triples that can be queried

using the language *SPARQL*.

To load a dataset from a semantic store, use the class `SemanticStoreDataset`. This class extends `Dataset` and reads out data from a semantic store in its constructor. To configure the location of the semantic store, pass a `Sparqlable` object. The preferred implementation of this interface is `SemanticStoreConnection2`, and is configured via its constructor, which takes a URL and a username/password pair.

3.2 Databases

The package `de.dailab.recommender.db` can be used to load a dataset from a database. In that case, each entity type and each relationship type has its own table. The configuration and loading is done through the class `DbDataset`.

3.3 The Graph Store

The Graph Store is a collection of network datasets located on the server `munin` that can be used with the Universal Recommender. The Graph Store contains two types of datasets: semantic datasets and unirelational datasets. All Graph Store datasets can be loaded using the classes in the two following packages:

- `de.dailab.recommender.graph.datasets`
- `de.dailab.recommender.graph.unirelationaldatasets`

To load the datasets, the directory `corpus@munin:/data/corpora/graph/` must be mounted locally and the variable `$GRAPH_DIR` must be set to this directory. Instead of mounting the directory, it could also be copied from `munin`, if enough disk place space is available.

4 Prediction

Prediction of links in a dataset is one of the main uses of the Universal Recommender. A predictor is an algorithm that takes a pair of entities and returns a floating point value describing the proximity or similarity of the two entities. The exact meaning of the prediction score varies from one prediction algorithm to another.

The interface `Predictor` denotes individual prediction algorithms. To compute predictions in a given dataset, the method `Predictor.build(Dataset)` returns a `PredictorModel`. A predictor model can then be used to compute prediction for that specific dataset. `PredictorModel` is an interface that has a method `predict()` which takes two entities and returns a prediction

The interface `Predictor` does not specify the meaning of scores, and each predictor will typically document the possible returned scores. The only requirement on scores are:

- Higher scores denote higher proximity or similarity

- A score of zero is neutral

In particular, no typical order of magnitude is specified, meaning that some predictors return very small or very large values; this has to be supported by callers. Also, there is no upper limit in most predictors, meaning that scores cannot be interpreted as percentages.

5 Recommendation

In the task of recommendation, the input is an entity (the source), and the output is a list of scored entities.

The returned entities are returned as an iterator over `Recommendation` object. This is to make it possible for implementations to make only the computations needed for the number of required entities, and also allows pagination. A `Recommendation` simply contains an entity and a score, which can be seen as a prediction value if the recommender has an underlying predictor.

The interface `Recommender` works analogously to `Predictor`; it has a `build(Dataset)` method which returns a `RecommenderModel`. A recommender model can then be used to compute recommendation with the `recommend()` methods.

There are two basic types of recommenders: path recommenders and latent recommenders. Path recommenders work by following paths in the network; latent recommenders build a latent model of the dataset and use fast vector similarity measures for prediction.

5.1 Path Recommenders

Path recommenders are a class of recommendation algorithms that perform a usually breadth-first search in the network to find entities. The default path recommender is given by `PathRecommender()`. Path recommenders can be configured by `Path` objects, which specify an algorithm for search through a network. The default path is `AllPath2`, which performs a breadth-first search in the network and returns weighted sums of paths. Using `RelationshipPath`, `CompoundPath` and `ParallelPath`, paths can be specified that follow exact relationship type sequences.

The `build()` method of path recommenders does not compute anything; all computation is done in `recommend()`.

The runtime and memory usage of path recommenders is usually not predictable, as it depends on network topology. Path recommenders are not considered scalable to large datasets.

5.2 Latent Recommenders

Latent recommenders compute a latent model of the dataset, and use fast similarity measures in this model to compute recommendations. The default latent recommender is given by `LatentRecommender`. Latent recommenders are intended to be scalable to large network sizes.

The default and simplest latent recommender is based on the eigenvalue decomposition of the network's adjacency matrix.

Latent recommenders correspond compute a matrix decomposition in the general sense [2]. The decomposed matrix is the adjacency matrix, the Laplacian, or any other characteristic graph matrix. Matrix decompositions are usually the eigenvalue of singular value decompositions, but some others are such as "mask decomposition" and generalized Laplacians are implemented. In all cases, the result are two eigenvector matrices U and V, and a eigenvalues Λ. The terms *eigenvector* and *eigenvalue* are to be understood in a general sense here.

By construction, latent recommenders are able to compute predictions, and the returned recommendation scores are exactly the prediction values.

To compute a prediction for the entity pair (i, j), the three vectors U_i, V_j and $\mathrm{diag}(\lambda)$ are combined. The type of combination can be adjusted and is represented by the interface `Similarity`. Examples are the dot product, the inverted Euclidean distance or the "Gaussian" function. It is also possible to apply spectral transformations to Λ, using the class `SpectralTransformation`.

6 Evaluation

By *evaluation*, we understand the computation of several predictors and recommenders on a specific dataset with the goal of comparing their performances at these tasks. Evaluations are done for recommendation research, for testing the Universal Recommender, and for testing the consistency and plausibility of new datasets.

The unit test `TestEvaluation` performs prediction and recommendation evaluations on a set of representative datasets from the Graph Store. The classes `PredictorEvaluation` and `RecommenderEvaluation` can be used to perform an evaluation of the default predictors and recommenders on any dataset.

Both types of evaluations are implemented by splitting the dataset's relationships into a training and a test set. The training set is then used to compute predictions or recommendations, which are then compared to the the relationships in the test set. Evaluation results are given in various error measures for prediction, and in various precision variants for recommendation.

7 Glossary

Several technical terms are used throughout the Java code and in associated literature. Several of these terms can be confusing, and it is important to distinguish them. The following is a glossary of such terms:

Dataset A network of entities connected by relationships.

Entity A node in a dataset. An entity has an entity type and an integer ID.

Entity type The type of an entity in a dataset. Entity types are represented by strings. Typical examples are users and movies.

Predictor An algorithm that computes a similarity or proximity value given two entities in a dataset. Predictors have to be "compiled" into a predictor model to compute predictions.

Predictor model A model that was built using a given predictor and dataset. A predictor model can compute predictions given two entities.

Recommender An algorithm that computes a list of scored entities when given a weighted list of entities (the sources). A recommender has to be "compiled" into a recommender model to compute recommendations.

Recommender model A model that was built using a given recommender and dataset. A recommender model can compute recommendations when given a weighted list of entities.

Relationship A relationship connects two entities in a dataset. A relationship can be optionally weighted, and has a specific relationship type.

Relationship type The type of a relationship. Relationship types are represented by strings. Typical examples are friendship, ratings and a hyperlink.

References

[1] Bennett, J., and Lanning, S. The Netflix prize. In *Proc. KDD Cup* (2007), pp. 3–6.

[2] Kunegis, J., and Lommatzsch, A. Learning spectral graph transformations for link prediction. In *Proc. Int. Conf. on Machine Learning* (2009), pp. 561–568.

[3] Kunegis, J., Lommatzsch, A., and Bauckhage, C. The Slashdot Zoo: Mining a social network with negative edges. In *Proc. Int. World Wide Web Conf.* (2009), pp. 741–750.

[4] Kunegis, J., Said, A., and Umbrath, W. The Universal Recommender. White paper, 2009.