Collaborative Filtering with Apache Mahout

Sebastian Schelter
Technische Universität Berlin, Germany
ssc@apache.org

Sean Owen
Myrrix Ltd.
srowen@apache.org

ABSTRACT

Apache Mahout [1] is an Apache-licensed, open source library for scalable machine learning. It is well known for algorithm implementations that run in parallel on a cluster of machines using the MapReduce [2] paradigm.

Besides that, Mahout offers one of the most mature and widely used frameworks for non-distributed Collaborative Filtering. We give an overview of this framework's functionality, API and featured algorithms.

Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous

Keywords

Collaborative Filtering, Open Source

1. INTRODUCTION

Today's internet users face an ever increasing amount of data, which makes it constantly harder and more time consuming to pick out the interesting pieces of information from all the noise. This situation has triggered the development of recommender systems: intelligent filters that learn about the users' preferences and figure out the most relevant information for them.

We believe that it is of high importance to provide high quality, publicly available, open source software for implementing recommender systems. Therefore, we present the collaborative filtering framework of the *Apache Mahout* [1] library for scalable data mining and machine learning. Whilst Mahout also provides algorithm implementations to compute recommendations in batch [11] on a MapReduce cluster, we put our focus on the functionality it offers for developing single-machine recommendation systems.

In this work, we present Mahout's flexible collaborative filtering framework, which features a broad range of algorithm implementations and provides all necessary building blocks for real-world recommender systems. Its main design goals consist of processing efficiency, ease of use, integratibility with different datastores and extensibility for both scientific and industrial usage.

2. RECOMMENDERS AND DATA

At the heart of Collaborative filtering applications lie user-item interactions. Mahout models those as a *(user,item,value)* triple in a *Preference* object. For memory efficiency, only numeric identifiers are allowed. The *PreferenceArray* encapsulates a set of interactions belonging to either a user or an item.

The dataset holding all known interactions is represented by a *DataModel*. This class provides a variety of convenient accessor methods like *getNumUsers()* to find the number of users in the data or *getPreferencesForItem(itemID)* to get all interactions for a particular item. Mahout offers several implementations that are able to manage interaction data in memory, on disk, in relational databases and key-value stores.

```
DataModel dataModel = new FileDataModel(new
    File("movielens.csv"));
PreferenceArray prefsOfUser = dataModel.
    getPreferencesFromUser(userID);
```

Listing 1: loading interaction data

The basic interface for all of Mahout's recommenders is modeled in the *Recommender* class, which offers the functionality of recommending items for a particular user (*top-N recommendation*) as well as estimating the preference of a user towards an unknown item (*rating prediction*).

In real-world usecases, one might not want to include all existing items when computing recommendations. This could be either due to latency constraints or requirements of a business usecase (e.g. some items might be temporarily out of stock). For such a scenario, Mahout provides a customizable *CandidateItemsStrategy* class, which is responsible for fetching all items that might be recommended for a particular user. Furthermore, a user can provide a *Rescorer* to postprocess recommendations, e.g. to emphasize special offers.

```
List<RecommendedItem> topItems =
    recommender.recommend(userID,10);
float preference = recommender.
    estimatePreference(userID,itemID);
```

Listing 2: top-N recommendations and rating prediction

3. NEIGHBORHOOD METHODS

Neighborhood methods form the most established and widely used approach to collaborative filtering. They are also known as k-nearest neighbor methods as they compute recommendations by

either finding users with similar taste or items that have been similarly rated. Mahout provides implementations for both, the user-based and the item-based approach. In a *UserBasedRecommender* [9], a *UserNeighborhood* selects users that act as a jury for finding items to recommend.

Due to its simplicity and scalability, the item-based approach [8, 10] represents the most widely deployed recommendation algorithm. It can present the items most similar to a given item (a popular non-personalized way of recommending) and can provide preferences for items as justification for recommendations. In Mahout, the item-based approach is realized by an *ItemBasedRecommender* together with a measure to compute similarities between items, represented by an *ItemSimilarity*. Implementations for lots of popular measures such Pearson correlation, cosine similarity, Jaccard coefficient or loglikelihood ratio [3] are available. Similar to interaction data, precomputed item similarities can be loaded from disk or a relational database.

```
ItemBasedRecommender recommender = new
    GenericItemBasedRecommender(dataModel,
    new PearsonCorrelationSimilarity(
    dataModel));
List<RecommendedItem> similarItems =
    recommender.mostSimilarItems(itemID,3);
```

Listing 3: computing most similar items

4. LATENT FACTOR MODELS

During the *Netflix* contest, another class of collaborative filtering algorithms has become very popular. The so called *latent factor models* [7] are based on projecting the user-item interactions onto a lower-dimensional feature space. In Mahout, the *SVDRecommender* computes recommendations from such a projection.

The necessary decomposition of the interaction data is computed by a *Factorizer*. Mahout offers implementations of standard models such as Simon Funk's SGD [4], SVD++ [6] and parallel implementations of weighted matrix factorization using Alternating Least Squares [5, 12].

```
Factorizer factorizer = new
    SVDPlusPlusFactorizer(dataModel,
    numFeatures,numIterations);
Recommender recommender = new SVDRecommender
    (dataModel,factorizer);
List<RecommendedItem> topItems = recommender
    .recommend(userID,10);
```

Listing 4: recommendations using matrix factorization

5. EVALUATION

Mahout offers tools to evaluate the prediction quality of a recommender on a random split of the data. For explicit feedback data, a *RecommenderEvaluator* can compute the mean average error as well as the root mean squared error. In the case of implicit feedback data, the *RecommenderIRStatsEvaluator* computes statistics such as precision, recall, normalized discounted cumulative gain and related measures.

```
RecommenderEvaluator eval = new
    RMSRecommenderEvaluator();
RecommenderBuilder recoBuilder = ...
DataModelBuilder dataModelBuilder = ...
eval.evaluate(recoBuilder,dataModelBuilder,
    dataModel,trainingRatio,testRatio);
```

6. FUTURE DIRECTIONS

Although we consider Mahout to be a versatile, mature framework for building recommenders, there is still a lot of room for improvements and extensions.

The evaluation framework should help users with finding parameters such as the number of nearest items or users to use in neighborhood methods or the learning rate, regularization constant and number of features for latent factor models.

Furthermore, Mahout should offer recommenders that integrate valuable side information such as content attributes, temporal data and social links. Such functionality would be extremely beneficial for production usecases.

On the technical side, we would like to see support for out-of-core operations for datasets that do not fit into main memory but should still be managable by a single machine. Furthermore there should be explicit support for realtime updates and folding-in new users and items.

7. REFERENCES

[1] Apache Mahout, http://mahout.apache.org.

[2] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, 2008.

[3] T. Dunning. Accurate methods for the statistics of surprise and coincidence. *Comput. Linguist.*, 19:61–74, 1993.

[4] S. Funk. Netflix Update: Try This at Home, http://sifter.org/ simon/journal/20061211.html. 2006.

[5] Y. Hu, Y. Koren, and C. Volinsky. Collaborative Filtering for Implicit Feedback Datasets. ICDM, pages 263–272, 2008.

[6] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. KDD, pages 426–434, 2008.

[7] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42:30–37, 2009.

[8] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[9] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. CSCW, pages 175–186, 1994.

[10] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. WWW, pages 285–295, 2001.

[11] S. Schelter, C. Boden, and V. Markl. Scalable Similarity-Based Neighborhood Methods with MapReduce. RecSys, pages 163–170, 2012.

[12] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-Scale Parallel Collaborative Filtering for the Netflix Prize. AAIM, pages 337–348, 2008.