LEIBNIZ UNIVERSITÄT HANNOVER

FAKULTÄT FÜR ELEKTROTECHNIK UND INFORMATIK
INSTITUT FÜR VERTEILTE SYSTEME

Multi-purpose Library of Recommender System Algorithms for the Item Prediction Task

Bachelor Thesis

eingereicht von

JULIUS KOLBE

am 11. Juni 2013

Erstprüfer	:	Prof. Dr. techn. Wolfgang Nejdl
Zweitprüfer	:	Jun.-Prof. Dr. rer. nat. Robert Jäschke
Betreuer	:	Ernesto Diaz-Aviles

EHRENWÖRTLICHE ERKLÄRUNG

Hiermit versichere ich, die vorliegende Bachelor Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die wörtlich oder inhaltlich aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Hannover, den 11. Juni 2013

Julius Kolbe

ABSTRACT

In this thesis I will give an introduction to recommender systems, provide an overview over other recommender system libraries and datasets available to try out the algorithms. After that I will describe different recommender algorithms and evaluation metrics I implemented in my work followed by an explanation on how to use them. Additionally I will provide the result of the tests.

ZUSAMMENFASSUNG

Kurze Zusammenfassung des Inhaltes in deutscher Sprache…

CONTENTS

1 INTRODUCTION

1.1 MOTIVATION

The library together with this document shall provide a "cookbook" for recommender systems. With the simple syntax and the interactivity of Python it is aimed at beginners to simply experiment with different algorithms. Especially the interactivity is missing in the already existing libraries because none of them is written in Python.

1.2 TASK (WHAT A RECOMMENDER SYSTEM DOES)

A Recommender System works in a scenario with users, items and interactions users can have with items. Such a scenario could be an online shop, where the interactions are purchases of items by users or a video platform, where the users interact with items (videos) by watching them. Based on the past interactions of the users a Recommender System searches for items a user haven't interacted with yet but the probability that he will interact is maximized.

The interactions can be implicit like purchases or clicks, then the scenario is also called item prediction. When the feedback is provided explicit like ratings the scenario is called rating prediction. In this work the focus lies on implicit feedback or item prediction. However ratings can be interpreted as the strength of implicit feedback. For example how often a user purchased an item. Some algorithms implemented in this library can use this information but none will explicitly predict ratings like it's usual in rating prediction scenarios.

1.3 CONTRIBUTIONS

The main contribution of my work is the interactive library I wrote [2]. Also in this document I provide explanations about the algorithms implemented in the library and an extensive user manual of the library.

2 BACKGROUND

2.1 EVALUATION METHODS

To evaluate a recommender algorithm we have to split up the database into one for training and one for evaluation. There are different methods to split the database but in the library only one is implemented.

2.1.1 *Leave-one-out Protocol*

The Leave-one-out Protocol means, that we take one interaction of each user out of the database for training and use it for validation. The item the recommender has to predict is also called hidden item.

Now we can test for each user if the algorithm is capable to predict this missing interaction.

2.1.2 *Evaluation metrics*

These are a selection of different metrics to rate the recommendations. By default the evaluations are executed with only one hidden item but generally the metrics should also work with more than just one.

2.1.2.1 *Hitrate/Recall@N*

This metrics lets the recommender recommend N items. If the hidden item is under the N recommended items, the recommender got a hit [5, 7]. So the hitrate is

$$\text{Hitrate} = \frac{\text{Numberofhits}}{\text{Numberofhiddenitems}}$$

This metric is very intuitive you can for example imagine that you show the user 10 items then Recall@10 would be the chance of showing the user an item he will interact with. But this metric doesn't take the number of recommended items into account.

2.1.2.2 *Precision*

The precision[7] is

$$\text{Precision} = \frac{\text{numberofhits}}{\text{numberofrecommendeditems}}$$

As you can clearly see this metric is taken the number of recommended items into account. Which will probably lead to worse results as the number of recommended items increases.

2.1.2.3 *F1*

The F1 metric[7] tries to balance hitrate and precision by taking both into account.

$$F1 = \frac{2 * \text{Hitrate} * \text{Precision}}{(\text{Recall} + \text{Precision})}$$

2.1.2.4 *Mean Reciprocal Hitrate*

This metric counts the hits but punishes them the more the lower they appear in the list of recommendations. So if the hidden item appears first in the list of recommendations the hit counts as one, but when it is in the second position the hit already counts only as one half and so on.

2.1.2.5 *Area under the ROC (AUC)*

AUC[6] counts the number of items the recommender rates higher than the hidden item, normalize it by the number of items the recommender can rate higher. Sum this up for every user and again normalize by the number of users.

To get an implicit score of each item the recommender recommends all items in a list sorted by decreasing score. This is in fact the same as for the other metrics only that the recommender can recommend as many items as possible.

2.2 DATASETS FOR TESTING

In the WWW there are several anonymized datasets available to try out recommender systems. Following I will introduce three of them.

2.2.1 *MovieLens*

MovieLens[1] is a database provided by GroupLens, a research lab at the University of Minnesota. One of their research areas is recommender systems and they built an application where users rate movies and then get recommendations for movies the could like. The MovieLens dataset is the ratings gathered by this application. For this work I will interpret the rating as intensity of interaction between users and items for example the number of times the user saw this movie.

The dataset is available in three different sizes:

- 100,000 ratings
- 1 million ratings
- 10 million ratings

For the experiments the smallest dataset is totally sufficient, with the larger datasets the computation time gets too long for just trying something out.

2.2.2 *Million Song Dataset*

The million song dataset[3] is a large database of features and media data of a million songs. For a challenge they also provided the listening history of over 1 million user. To present I will use a subset of this dataset to keep the computing time required reasonable low so it's easier for others to retrace the results.

2.2.3 *SNAP*

3 RELATED WORK

There is a wide range of projects providing implemantions for recommender system. Some of them are described in this chapter to give a quick overview and comparison.

3.1 MyMediaLite

MyMediaLite[**?**] is an open source project developed at the University of Hildesheim and provides several algorithm for rating prediction and item prediction. It is written in C# and is used with a command line interface. It also provides a graphical interface to demonstrate recommender algorithms

3.2 PREA (Personalized Recommendation Algorithms Toolkit)

PREA[**?**] is an open source project written in Java. It provides a wide range of recommender algorithms and evaluation metrics to test them. It is maintained by the Georgia Institute of Technology.

3.3 Apache Mahout

Mahout[**?**] is an open source library in java. It is implemented on top of Apache Hadoop, so it uses the map/reduce paradigm. This means it can run on different independent computers.

3.4 Duine Framework

The Duine Framework [**?**] is an open source project written in java by the Telematica Instituut/Novay. The recommender of the Duine Framework combines multiple prediction techniques to exploit the strengths of the different techniques and to avoid their weaknesses.

3.5 Cofi

Cofi [**?**]provides an algorithm for the rating prediction task called Maximum Margin Matrix Factorization. It is open source and written in C++.

3.6 Lenskit

Lenskit [**?**] is a toolkit which provides several recommender algorithms and an infrastructure to evaluate them. It is an open source project by the University of Minnesota

4 RECOMMENDATION ALGORITHMS

In this chapter I will roughly explain how the algorithms I've implemented work. For further explanations please refer to the cited papers.

4.1 NON-PERSONALIZED ALGORITHMS

In this chapter I will describe two very simple and basic recommendation algorithms I implemented for comparison with the more sophisticated algorithms.

4.1.1 *Constant*

The constant recommender algorithm counts the number of interactions for each item and sorts this in decreasing order of interactions. Then it recommends the top items of this list. So it recommends the items which are the most popular over all users and doesn't do any personalizations.

4.1.2 *Random*

The random recommender just picks items randomly.

4.2 K-NEAREST-NEIGHBOR

This class of recommendation algorithms works by searching neighbors of either items or users based on a similarity function which is the cosine in this library.

4.2.1 *Item Based*

For this algorithm the database has to be represented as a matrix where the rows correspond to the users and the columns to the items. Then the entry (i,j) represents the number of transactions which happened between the ith user and the jth item.

The algorithm interprets the columns of the matrix i.e. the items as vectors and computes there similarities by computing their cosine. To build the model the algorithm computes the n most similar items of each item.

To compute recommendations for user U the algorithm then computes the union of the n most similar items of each item U interacted with. From this set the items U already interacted with are removed. For each item remaining in this set we compute the sum of its similarities to the items U interacted with. Finally these items are sorted in decreasing order of this sum of similarities and the first n items will be recommended[5].

4.2.2 *User Based*

The user based k-Nearest-Neighbor is very similar to the item based. But instead of interpreting the columns as vectors we interpret the lines or users of the matrix as vectors and compute their similarities to other users.

Then for each item i we sum up the similarities between U and the users who interacted with i. Again we remove all items U already interacted with, sort in decreasing order fo the sum and recommend the first n items.

4.3 MATRIX FACTORIZATION

All matrix factorization techniques build two matrices in the model building phase. These matrices are supposed to represent abstract features of each item and user. For recommendation the dot product of the feature vector of an user and an item gives a score whith which we can sort the items and recommend the best suitable ones. The process of presenting a large matrix M as two smaller matrices W and H so that M = W*H is also called singular value decomposition.

Each of the implemented algorithms train the model with stochastic gradient descent. In each iteration the model is trained with a randomly chosen user, a randomly chosen item the user interacted with, called the positive item and a randomly chosen item the user didn't interacted with yet, called the negative item. The features of the user and the negative and the positive item are then trained according to the derivative of a loss function.

4.3.1 *BPRMF*

BPMRF uses the logloss to train the model. The logloss is defined as

$$\mathrm{logLoss}(a, y) = \log(1 + \exp(-ay))$$

And the derivative of the log loss is

$$\frac{\partial}{\partial y}(\log(1 + \exp(-ay))) = -\frac{a}{\exp(ay) + 1}$$

For further informations please refer to [6]

4.3.2 *RankMFX*

RankMFX uses the hingeLoss. It is defined as

$$\mathrm{hingeLoss}(a, y) = \max(0, 1 - ay)$$

And its derivative

$$\frac{\partial}{\partial y}(\max(0, 1 - ay)) = \begin{cases} -y & ay < 1 \\ 0 & \text{otherwise} \end{cases}$$

See also [Paper for citation?]

4.3.3 *Ranking SVD (Sparse SVD)*

Ranking SVD uses the quadratic loss and the difference between the predicted score of the positive item and the negative minus the actual score of the positive item.[4]

5 EXPERIMENTS

5.1 EXECUTION

5.2 RESULTS

5.3 COMPARISON

6 DESIGN AND IMPLEMENTATION

6.1 General structure

6.2 Interfaces

7 USER MANUAL

7.1 PRIMITIVE ALGORITHMS

7.2 K-NEAREST NEIGHBOR

7.3 BPRMF

7.4 RANKMFX

7.5 RANKING SVD (SPARSE SVD)

8 CONCLUSIONS

8.1 FUTURE WORK

8.2 OUTLOOK

BIBLIOGRAPHY

[1] Movielens data sets. URL http://grouplens.org/node/73.

[2] recsyslab. URL https://github.com/Foolius/recsyslab.

[3] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[4] Michael Jahrer and Andreas Töscher. Collaborative filtering ensemble for ranking. In *Proc. of KDD Cup Workshop at 17th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, KDD*, volume 11, 2011.

[5] George Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, pages 247–254, New York, NY, USA, 2001. ACM. ISBN 1-58113-436-3. doi: 10.1145/502585.502627. URL http://doi.acm.org/10.1145/502585.502627.

[6] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press. ISBN 978-0-9749039-5-8. URL http://dl.acm.org/citation.cfm?id=1795114.1795167.

[7] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. Application of dimensionality reduction in recommender system – a case study. In *IN ACM WEBKDD WORKSHOP*, 2000.