# Reproducing and Prototyping Recommender Systems in R

Ludovik Çoba, Panagiotis Symeonidis, Markus Zanker

Free University of Bozen-Bolzano, 39100, Bozen-Bolzano, Italy
{lucoba,psymeonidis,Markus.Zanker}@unibz.it

Abstract. In this paper we describe recsys, an open source extension package in R for rapid prototyping and intuitive assessment of recommender system algorithms. Due to its wide variety of implemented packages and functionalities, R language represents a popular choice for many tasks in Data Analysis. This package replicates the most popular collaborative filtering algorithms for rating and binary data and we compare results with the Java-based LensKit implementation for the purpose of benchmarking the implementation. Therefore this work can also be seen as a contribution in the context of replication of algorithm implementations and reproduction of evaluation results. Users can easily tune available implementations or develop their own algorithms and assess them according to the standard methodology for offline evaluation. Thus this package should represent an easily accessible environment for research and teaching purposes in the field of recommender systems.

### 1 Introduction

R represents a popular choice in Data Analytics and Machine Learning. The software has low setup cost and contains a large selection of packages and functionalities to enhance and prototype algorithms with compact code and good visualization tools. Thus R represents a suitable environment for exploring the field of recommender systems. Therefore we present and contribute a novel R package that reproduces several of the most popular recommender algorithms for Likert scaled as well as binary rating values. The functionality of this framework is mainly focused on prototyping and educational purposes due to the compact code representation and the interactive way of invoking and visualizing results in R. We took advantage of the large selection of packages in R to implement the algorithms included in our package. We achieve competing performance due to highly vectorized and mixed R/C++ implementations.

We therefore introduce  $\operatorname{rrecsys}[1, 2]$  by presenting a general overview of the implemented algorithms and the evaluation methodology. A few code examples are included in an appendix. Furthermore, we proceed by comparing results of recsys with those of the Lenskit library. We also include evidence on runtime performance that document the efficient implementation of algorithms in R.

## 2 rrecsys Package

rrecsys has a modular structure as well as includes expansion capabilities. The core of the package includes the implementation of several popular algorithms, an evaluation component and a couple of auxiliary parts for data analysis and convergence detection. Next we concisely describe the included algorithms.

#### 2.1 Algorithms

**Baseline and Popularity:** The included baseline predictors are the global mean rating (Global Average), item's mean rating (Item Average), user's mean ratings (User Average) as well as an unpersonalized Most Popular method that determines item popularity based on the total number of (positive) ratings.

Item Based K-nearest neighbors: Given a target user and her positively rated items, the algorithm will identify the k-most similar items for each target *a* and will rank them according to aggregated similarities with the different targets as described by Sarwar et al. [14]. For similarity measures we develop the *cosine* similarity and the adjusted cosine similarity.

Items a and b are considered as rating vectors a and b in the user space. Cosine similarity measures the cosine angle between those vectors:

$$sim(\boldsymbol{a}, \boldsymbol{b}) = cos(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{|\boldsymbol{a}| * |\boldsymbol{b}|}$$
(1)

The *adjusted cosine similarity* is computed by offsetting the user average on each co-rated pair on two item vectors. If  $I_a$  and  $I_b$  is the set of users that rated correspondingly item a and b the adjusted cosine similarity is measured as:

$$sim(\boldsymbol{a}, \boldsymbol{b}) = \frac{\sum\limits_{u \in I_a \cap I_b} (R_{ua} - \overline{R_u}) * (R_{ub} - \overline{R_u})}{\sqrt{\sum\limits_{u \in I_a \cap I_b} (R_{ua} - \overline{R_u})^2 * \sum\limits_{u \in I_a \cap I_b} (R_{ub} - \overline{R_u})^2}}$$
(2)

Where  $\overline{R_u}$  is the average rating of user u, computed as:

$$\overline{R_u} = \sum_{i \in I_u} \frac{R_{ui}}{|I_u|} \tag{3}$$

Where  $I_u$  is the set of items rated by user u.

Once similarities among all items are computed a neighborhood might be formed by choosing the items with the highest similarity value. Prediction over a target user u and item a are calculated as the weighted sum [14].

$$p_{u,a} = \frac{\sum_{\text{all similar items},k} (s_{a,k} * R_{u,k})}{\sum_{\text{all similar items},k} s_{a,k}}$$
(4)

User Based K-nearest neighbors: Herlocker et al. [7] proposed the algorithm that finds similarities among users instead among items. In our implementation we consider the *cosine* similarity and the *Pearson correlation*. User u and v are considered as rating vectors u and v in the item space. Cosine similarity measures the cosine angle between those vectors using Formula 1

Instead the *Pearson correlation* is measured by offsetting the user average on co-rated pairs among the user vectors:

$$sim(\boldsymbol{u}, \boldsymbol{v}) = Pearson(\boldsymbol{u}, \boldsymbol{v}) = \frac{\sum\limits_{i \in I_u \cap I_v} (R_{ui} - \overline{R_u}) * (R_{vi} - \overline{R_v})}{\sqrt{\sum\limits_{i \in I_u \cap I_v} (R_{ui} - \overline{R_u})^2 * \sum\limits_{i \in I_u \cap I_v} (R_{vi} - \overline{R_v})^2}}$$
(5)

Where  $\overline{R_u}$  and  $\overline{R_v}$  are correspondingly the average ratings of user u and user v, computed like in Formula 3.

Once similarities among all items are computed a neighborhood might be formed by choosing the items with the highest similarity value. Prediction over a target user u and item a are calculated as the weighted sum [14]:

$$p_{u,a} = \overline{R_u} + \frac{\sum_{\text{all similar items},k} (s_{a,k} * R_{u,k})}{\sum_{\text{all similar items},k} s_{a,k}}$$
(6)

Weighted Slope One: proposed by Lemire et al [10] performs prediction for a missing rating  $\hat{r}_{ui}$  for user u on item i as the following average:

$$\hat{r}_{ui} = \frac{\sum_{\forall r_{uj}} (dev_{ij} + r_{uj})c_{ij}}{\sum_{\forall r_{uj}} c_{ij}}.$$
(7)

The average deviation rating  $dev_{ij}$  between co-rated items is defined by:

$$dev_{ij} = \sum_{\forall u \in users} \frac{r_{ui} - r_{uj}}{c_{ij}}.$$
(8)

Where  $c_{ij}$  is the number of co-rated items between items *i* and *j* and  $r_{ui}$  is an existing rating for user *u* on item *i*. The Weighted Slope One takes into account both, information from users who rated the same item and the number of observed ratings.

Simon Funk's SVD: Matrix factorization methods are used in recommender systems to derive a set of latent factors, from the user  $\times$  item rating matrix, to characterize both users and items by this vector of factors. The useritem interaction are modeled as inner product of the latent factors space [4]. Accordingly each item *i* will be associated with a vector of factors  $V_i$ , and each user *u* is associated with a vector of factors  $U_u$ . An approximation of the rating of a user *u* on an item *i* can be derived as the inner product of their factor vectors:

$$\hat{R}_{ui} = \mu + b_i + b_u + U_u * V_i^T$$
(9)

Where  $\mu$  is the overall average rating and  $b_u$  and  $b_i$  indicate the deviation due to user u and item i from the mean rating.

The U(user) and V(item) factor matrices are cropped to k features and initialized at small values. Each feature is trained until convergence (where convergence specifying the number of updates to be computed on a feature before considering it converged, it can be either chosen by the user or calculated automatically by the package). On each loop the algorithm predicts  $\hat{R}_{ui}$ , calculates the error and the factors are updated as follows:

$$e_{ui} = R_{ui} - \dot{R}_{ui} \tag{10}$$

$$V_{ik} \leftarrow V_{ik} + \lambda * (e_{ui} * U_{uk} - \gamma * V_{ik}) \tag{11}$$

$$U_{uk} \leftarrow U_{uk} + \lambda * (e_{ui} * V_{ik} - \gamma * U_{uk}) \tag{12}$$

The attribute  $\lambda$  represents the learning rate, while  $\gamma$  corresponds to the regularization term.

In addition, the following two algorithms address the One Class Collaborative Filtering problem (OCCF).

**Bayesian Personalized Ranking:** The algorithm has been introduced by [12]. It turns the OCCF into a ranking problem by implicitly assuming that users prefer items they have already interacted with another time. Instead of applying rating prediction techniques, BPR ranks candidate items for a user without calculating a "virtual" rating. The overall goal of the algorithm is to find a personalized total ranking  $>_u \subset I^2$  for any user  $u \in Users$  and pairs of items  $(i, j) \in I^2$  that meet the properties of a total order (totality, anti-symmetry, transitivity).

Weighted Alternated Least Squares: We compute a low-rank approximation matrix  $\hat{R} = (\hat{R}_{ij})_{m \times n} = U * V^T$ , where U and V are the usual feature matrix cropped to k features as introduced by [11]. Weighted low-rank aims to determine  $\hat{R}$  such that minimizes the Frobenius loss of the following objective function:

$$\mathcal{L}(\hat{R}) = \mathcal{L}(U, V) = \sum_{ij} W_{ij} (R_{ij} - U_i * V_j^T)^2 + \lambda * (\|U_i\|_F^2 + \|V_i\|_F^2)$$
(13)

The regularization term weighted by  $\lambda$  is added to prevent over-fitting. The expression  $\|.\|_F$  denotes the Frobenius norm. The alternated least square algorithms optimization process solves partial derivatives of  $\mathcal{L}$  with respect to each entry U and V,  $\frac{\partial \mathcal{L}(U,V)}{\partial U_i} = 0$  with fixed V and  $\frac{\partial \mathcal{L}(U,V)}{\partial V_j} = 0$  with fixed U, to compute  $U_i$  and  $V_i$ . Then U and V are initialized with random Gaussian numbers with mean zero and small standard deviation and are updated until convergence. The matrix  $W = (W_{ij})_{m \times n} \in \mathbb{R}^{m \times n}_+$  is a non-negative weight matrix that assigns confidence values to observations (hence the name weighted ALS).

#### 2.2 Recommendation List

The library is currently distributing two different methodologies for computing the top-N recommendation. The first is the Highest Predicted Ratings (HPR), which proposes a sorted list based on the highest computed rating values by an algorithm. The second method is the Most Frequent (MF), that determines the top-N list based on the most frequent items available in the neighborhood of an user or item. This methodology is known to produce better performance than HPR [9, 15].

#### 2.3 Evaluation

The evaluation module is based on the k-fold cross-validation method. A stratified random selection procedure is applied when dividing the rated items of each user into k folds such that each user is uniformly represented in each fold, i.e. the number of ratings of each user in any fold differs at most by one. For k-fold cross validation each of the k disjunct fractions of the ratings are used k - 1 times for training (i.e.  $\subset R_{train}$ ) and once for testing (i.e.  $\subset R_{test}$ ). Practically, ratings in  $R_{test}$  are set as missing in the original dataset and predictions/recommendations are compared to  $R_{test}$  to compute the performance measures.

We included the most popular performance metrics according to the survey in [8]. These are mean absolute error(MAE), root mean squared error(RMSE), Precision, Recall, F1, True and False Positives, True and False Negatives, normalized discounted cumulative gain (NDCG), rank score, area under the ROC curve (AUC) and catalog coverage[5].

#### **3** Experimental results

In this Section, we compare our recess library with the popular Lenskit [3] Java library.

In Figure 1, we compare both libraries in terms of RMSE and MAE, using 5-fold cross validation. Lenskit and receys were configured both with the same algorithms and evaluation methodology. We used the MovieLens100K dataset [6] for these experiments as the purpose is not only the results per se, but to demonstrate the reproducibility of the results derived from LensKit.

For the experiments in Figure 1 on the FunkSVD, we have tuned its parameters as follows, the latent space is set to 100 features, the learning rate is set to 0.001, the regularization term is set to 0.015. In the case of the item based k-nearest neighbor algorithm, we have set the number of nearest neighbors to 100, and adjusted cosine similarity is the similarity measure. In the case of the user-based k-nearest neighbor algorithm, we have set the number of nearest neighbors to 100, and used Pearson as similarity measure.

As shown in Figure 1, all the reported results demonstrate our ability to clearly reproduce and replicate the same results with those of Lenskit in terms of both RMSE and MAE while other libraries failed to do so [13]. The insignificant differences with LensKit are the result of a random distribution of items in the k-folds. Since BPR and wALS are not implemented in LensKit it is impossible for us to compare results.

In Table 1 we show the performance of *rrecsys* based on the latest implementations with R/C++ code on the MovieLens100K dataset running on the same machine. It is noticeable that *rrecsys* performs similarly to *LensKit*. We compared only optimized algorithms. In future we will provide optimized implementation of more state of the art algorithms.



Fig. 1. Benachmark with LensKit.

Framew.	IBKNN (neigh.)			UBKNN (neigh.)			Funk SVD (feat.)			WSlopeOne
	10	100	500	10	100	500	10	100	500	
rrecsys	14.91s	14.99s	15.67s	8.77s	8.87s	9.48s	6.23s	15.34s	54.71s	49.46s
Lenskit	14.17s	14.62s	15.28s	60.04s	61.48s	62.32s	4.32s	11.05s	46.82s	16.41s

Table 1. Evaluation times with optimized implementations using R/C++ and comparison with LensKit.

## 4 Using the library

In this Section, we introduce an executable script in R for running some of the functionalities of recess in order to demonstrate its intuitive use. Please notice that due to space limitations in this paper we do not describe all commands in detail. The library is distributed with a full range of vignettes and a manual describing all available functionalities<sup>1</sup>.

The package is loaded on the Comprehensive R Archive Network (CRAN), therefore download, installation and loading of the package requires the execution of the following two functions:

```
install.packages("rrecsys"); library(rrecsys)
```

Once the package is loaded the dataset MovieLens  $Latest^2$  will be available within the environment. A setup of the data is required to define possible limits and the structure of the dataset. Users can explore the dataset by checking the

 $<sup>^1</sup>$  https://cran.r-project.org/package=rrecsys

<sup>&</sup>lt;sup>2</sup> We redistribute the MovieLens Latest datasets for demonstration purposes only. Please notice that these datasets change over time and are not appropriate for reporting experimental results.

number of ratings, its sparsity or even by modifying it to contain a specific number of ratings for each item\user.

```
data("mlLatest100k")
m <- defineData(mlLatest100k, minimum = 1, maximum = 5,
halfStar = TRUE)
sparsity(m); numRatings(m); rowRatings(m); colRatings(m)
#Crop the dataset to contain at least 200 ratings on each user and
10 ratings on each item.
smallmlLatest <- m[rowRatings(m) >= 200, colRatings(m) > 10]
```

The following code shows how to train a model (e.g., ub10) on an algorithm (e.g., UBKNN), which can be used for either rating prediction (e.g., p) or item recommendation (e.g., rHPR and rMF).

```
ub10 <- rrecsys(smallmlLatest, "UBKNN", neigh = 10, simFunct = 1)
p <- predict(ub10)
rHPR <- recommendHPR(ub10, topN = 10)
#pt is the positive threshold for recommending an item.
rMF <- recommendMF(ub10, topN = 10, pt = 3)</pre>
```

The following code shows how we generate the k-folds. Same fold distribution can be used to evaluate different algorithms.

The output of the evaluation function looks like as follows:

## 5 Conclusions

This paper contributed a recently released package for prototyping and interactively demonstrating recommendation algorithms in R. It comes with a nice range of implemented standard algorithms for Likert scaled and binary ratings. Reported results demonstrate that it reproduces results of the Java-based Lenskit toolkit. Thus it remains to hope that this effort will be of use for the field of recommender systems and the large R user community.

## References

- Çoba, L., Zanker, M.: rrecsys: an r-package for prototyping recommendation algorithms. In: Guy, I., Sharma, A. (eds.) Poster Track of the 10th ACM Conference on Recommender Systems (RecSys 2016) (RecSysPosters). No. 1688 in CEUR Workshop Proceedings, Aachen (2016), http://ceur-ws.org/Vol-1688/#paper-12
- [2] Çoba, L., Zanker, M.: Replication and reproduction in recommender systems research evidence from a case-study with the recess library. In: 30th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2017, Arras, France, June, 2017, Proceedings. Springer International Publishing, Cham (2017)
- [3] Ekstrand, M.D., Ludwig, M., Kolb, J., Riedl, J.T.: Lenskit: A modular recommender framework. In: Proceedings of the Fifth ACM Conference on Recommender Systems. pp. 349–350. RecSys '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2043932.2044001
- [4] Funk, S.: Netflix Update: Try this at Home (2006), http://sifter.org/ simon/journal/20061211.html
- [5] Gunawardana, A., Shani, G.: A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. The Journal of Machine Learning Research 10, 2935–2962 (2009)
- [6] Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. ACM Trans. Interact. Intell. Syst. 5(4), 19:1–19:19 (Dec 2015)
- [7] Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 230–237. SIGIR '99, ACM, New York, NY, USA (1999)
- [8] Jannach, D., Zanker, M., Ge, M., Gröning, M.: 13th International Conference on E-Commerce and Web Technologies, chap. Recommender Systems in Computer Science and Information Systems – A Landscape of Research, pp. 76–87. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
- [9] Karypis, G.: Evaluation of item-based top-n recommendation algorithms. In: Proceedings of the Tenth International Conference on Information and Knowledge Management. pp. 247–254. CIKM '01, ACM, New York, NY, USA (2001)
- [10] Lemire, D., Maclachlan, A.: Slope one predictors for online rating-based collaborative filtering. In: SDM. vol. 5, pp. 1–5. SIAM (2005)
- [11] Pan, R., Zhou, Y., Cao, B., Liu, N.N., Lukose, R., Scholz, M., Yang, Q.: One-class collaborative filtering. Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on pp. 502–511 (2008)
- [12] Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-thieme, L.: BPR : Bayesian Personalized Ranking from Implicit Feedback. Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence cs.LG, 452–461 (2009), http://dl.acm.org/citation.cfm?id=1795167
- [13] Said, A., Bellogín, A.: Comparative Recommender System Evaluation: Benchmarking Recommendation Frameworks. RecSys pp. 129–136 (2014), http://dx.doi.org/10.1145/2645710.2645746
- [14] Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: 10th Int. Conference on the World Wide Web. pp. 285–295 (2001)
- [15] Symeonidis, P., Nanopoulos, A., Papadopoulos, A.N., Manolopoulos, Y.: Collaborative recommender systems: Combining effectiveness and efficiency. Expert Syst. Appl. 34(4), 2995–3013 (May 2008)