Recommender system: Using singular value decomposition as a matrix factorization approach

Robin Witte

Introduction

The task of a recommender system is to recommend items, that fit the user's taste. The approach, that makes use of only user activities of the past, is termed *collaborative filtering*. Given a not fully specified user-item ratings Matrix, a collaborative filtering algorithm estimates robustly all missing entries. This project report describes a collaborative filtering algorithm, that uses *singular value decomposition* (SVD) as a form of matrix factorization.

Matrix factorization

In the basic matrix factorization approach, the $m \times n$ ratings matrix R is approximatly factorized into an $m \times k$ matrix U and an $n \times k$ matrix V:

$$R \approx UV^T \tag{1}$$

In an intuitive way, the columns of U represent the affinities of the users for k different concepts and the columns of V the affinities of the items for the same k concepts. Thus each rating r_{ui} in R can be approximatly expressed as follows:

$$r_{ui} \approx \hat{r}_{ui} = \sum_{k=1}^{K} (u_{uk} \cdot v_{ik}) \tag{2}$$

The matrix factorization can be performed by a *truncated* SVD of rank $k \ll \min\{m, n\}$, which is given as:

$$R \approx Q_k \Sigma_k P_k^T \tag{3}$$

This solution is the best rank-K approximation of R with respect to the *root mean squared error* (RMSE). To achieve the matrix factorization described above, the matrices U and V are defined as follows:

$$\begin{aligned} U &= Q_k \Sigma_k \\ V &= P_k \end{aligned} \tag{4}$$

Problem Definition

In practice most of the users rate only a small subset of the entire set of items, so we are only given a small sample of ratings $\mathcal{T} = \{(u_1, i_1, r_1), (u_2, i_2, r_2), ..., (u_t, i_t, r_t)\}$ and the matrix R is not fully specified. Thus the SVD can not be calculated in a regular way. To solve this problem, Simon Funk introduced an algorithm, that uses a gradient descent to calculate the SVD [1]. For this method the error is defined by:

$$\epsilon_{ui} = r_{ui} - \hat{r}_{ui} = r_{ui} - \sum_{k=1}^{K} (u_{uk} \cdot v_{ik}) \tag{5}$$

As described in [2] minimizing the *sum of squared error* (SSE) minimizes the RMSE. Hence the convex optimization problem is defined as:

$$\begin{aligned} (U^*, V^*) &= \underset{\substack{U \in \mathbb{R}^{m \times k} \\ V \in \mathbb{R}^{n \times k}}}{\operatorname{argmin}} \sum_{(u,i) \in \mathcal{T}} \epsilon_{ui}^2 \\ &= \underset{\substack{U \in \mathbb{R}^{m \times k} \\ V \in \mathbb{R}^{n \times k}}}{\operatorname{argmin}} \sum_{(u,i) \in \mathcal{T}} \left(r_{ui} - \sum_{k=1}^{K} (u_{uk} \cdot v_{ik}) \right)^2 \end{aligned} \tag{6}$$

By taking the partial derivatives with respect to u_{uk} and v_{ik} we get update rules for the parameters (details in [3]):

$$\begin{aligned} u_{uk} &+= -\lambda \frac{\partial}{\partial u_{uk}} \epsilon_{ui}^2 = \lambda(\epsilon_{ui} v_{ik}) \\ v_{ik} &+= -\lambda \frac{\partial}{\partial v_{ik}} \epsilon_{ui}^2 = \lambda(\epsilon_{ui} u_{uk}) \end{aligned} \tag{7}$$

In this case, λ is an arbitrary number and describes a learning rate. With random initial values, these update rules describe an iterative algorithm to approximate the SVD and therefore to build the model for the recommender system using the sample $\mathcal{T}$.

Regularization

As described by Simon Funk a regularzation is recommended to discourage u_{uk} and v_{ik} from being large [1]. The idea is to perform a *Tikhonov regularization*, and thus penalizing the magnitude of the features. The regularized optimization problem is defined as:

$$\begin{aligned} (U^*, V^*) = \underset{\substack{U \in \mathbb{R}^{m \times k} \\ V \in \mathbb{R}^{n \times k}}}{\operatorname{argmin}} \sum_{(u,i) \in \mathcal{T}} \Bigg[& \epsilon_{ui}^2 \\ & + \gamma \left(\left(\sum_{k=1}^{K} u_{uk} \right)^2 + \left(\sum_{k=1}^{K} v_{ik} \right)^2 \right) \Bigg] \end{aligned} \tag{8}$$

The appropriate update rules are:

$$\begin{aligned} u_{uk} &+= \lambda(\epsilon_{ui} v_{ik} - \gamma u_{uk}) \\ v_{ik} &+= \lambda(\epsilon_{ui} u_{uk} - \gamma v_{ik}) \end{aligned} \tag{9}$$

The strength of regularization is described by γ.

Adding biases

Equation 2 tries to capture the interactions between users and items that produce the different rating values. However, much of the observed variation in rating values is due to effects associated with either users or items, known as biases or intercepts, independent of any interactions [4]. To deal with this, Koren et al. describe the following first-order approximation of the bias involved in rating r_{ui}:

$$b_{ui} = \mu + b_i + b_u \tag{10}$$

The overall average rating is denoted by μ. The parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average. Biases extend Equation 2 as follows:

$$\hat{r}_{ui} = \mu + b_u + b_i + \sum_{k=1}^{K} (u_{uk} \cdot v_{ik}) \tag{11}$$

In this case, the observed rating is broken down into its four components: global average, item bias, user bias, and user-item interaction. This allows each component to explain only the part of a signal relevant to it [4]. The biased regularized optimization problem is defined as:

$$\begin{aligned} (U^*, V^*) = \underset{\substack{U \in \mathbb{R}^{m \times k} \\ V \in \mathbb{R}^{n \times k}}}{\operatorname{argmin}} & \sum_{(u,i) \in \mathcal{T}} \\ & \left[\left(r_{ui} - \mu - b_u - b_i - \sum_{k=1}^{K} (u_{uk} \cdot v_{ik}) \right)^2 \right. \\ & \left. + \gamma \left(\left(\sum_{k=1}^{K} u_{uk} \right)^2 + \left(\sum_{k=1}^{K} v_{ik} \right)^2 + b_u^2 + b_i^2 \right) \right] \end{aligned} \tag{12}$$

The appropriate update rules are:

$$\begin{aligned} b_u &+= \lambda(\epsilon_{ui} - \gamma b_u) \\ b_i &+= \lambda(\epsilon_{ui} - \gamma b_i) \\ u_{uk} &+= \lambda(\epsilon_{ui} v_{ik} - \gamma u_{uk}) \\ v_{ik} &+= \lambda(\epsilon_{ui} u_{uk} - \gamma v_{ik}) \end{aligned} \tag{13}$$

Implementation

The implementation of the presented recommender system was build after the model of *Surprise* [(1)]. This python package was programmed by Nicolas Hug and is a *Python scikit* for recommender systems. The algorithm presented in this report uses the biased and reularized update rules described above. As postulated by Simon Funk choosing a fixed number of epochs for the gradient descent results in the best overall performance [1].

Initialization:

The learning rate λ is set to 0.005 and the regularization term γ to 0.001. The user factors u_{uk} and the item factors v_{ik} are both initalized with a standard normal distribution (mean = 0; standard deviation = 1). The user biases b_u and the item biases b_i are all set to zero. The approximation of the SVD is computed in 20 epochs with $K = 150$ factors.

Fit:

In every epoch, every presented rating is used once to improve the model. First the error is computed as described above. Subsequently this error is used to update all parameters after the rules in equation 13. u_{uk} and v_{ik} are updated for all K factors.

Prediction:

Equation 11 is applied, in order to predict ratings for given users and given items. If either the item or the user are not represented in U or V, the dot product is ignored. Moreover, the summand b_u is deleted, if the user is unknown and b_i, if the item is unknown. Additionaly a rounding to a precision of 0.1 is performed.

Analysis

In order to verify the algorithm, a dataset containing 461806 ratings was applied (ratings from 0 to 4). With this dataset a 5-fold cross validation was conducted. The result is presented in table 1.

Table 1 Results of 5-fold cross-validation (MAE: mean absolute error; time in seconds)

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE	0.507	0.510	0.510	0.511	0.512	0.510	0.0017
MAE	0.293	0.295	0.295	0.293	0.295	0.294	0.0008
Fit time	18.801	18.623	17.771	21.286	21.492	19.595	1.5073
Test time	0.680	0.600	0.610	0.710	0.640	0.648	0.0417

(1) `http://www.surpriselib.com`

Conclusion

The results demonstrate that SVD is a adequate choice for recommender systems. The use of a gradient descent algorithm leads to an efficient and fast implementation. Even if the number of epochs is fixed, the cross validation showed a reproducible fast convergence of the algorithm. One of the major drawbacks of the presented method is the missing online learning functionality. Once computed, the model is static. As described by Matthew Brand there are various different approaches to include updating in regularized matrix factorization models [5]. Since for real-world applications dynamic updating a model is one of the most important tasks, this shoud be included in future works. Certainly there are also various other possible extensions.

The presented parameter values were chosen with reference of Simon Funk [1] and Nicolas Hug [6]. They should be adapted by testing different values to improve the results.

In conclusion, SVD is a proper method to build a recommender system. Compared to other algorithms the presented implementation is short and plain, but leads to a very good result.

References

1. Simon Funk. Netflix update: Try this at home. URL `http://sifter.org/~simon/journal/20061211.html`.
2. Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of machine learning research*, 10(Mar):623–656, 2009.
3. Adam Wagman. Netflix svd derivation. URL `http://sifter.org/%7Esimon/journal/20070815.html`.
4. Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009. ISSN 0018-9162.
5. Matthew Brand. Fast online svd revisions for lightweight recommender systems. *Society for Industrial and Applied Mathematics*, 2003.
6. Nicolas Hug. Surprise documentation, 2015. URL `https://surprise.readthedocs.io/en/stable/#`.