Ukrainian Catholic University

Faculty of Applied Sciences

Data Science Master Programme

Decompose & Recommend

Linear Algebra final project report

Authors:
Anton Shcherbyna
Vadym Korshunov

24 January 2019

Abstract

Recommender systems play an important role in web services nowadays. In this work, we provide a detailed description of the domain and discuss two matrix factorizations algorithms for collaborative filtering - Funk SVD and Probabilistic Matrix Factorization. We perform experiments of those algorithms the movie rating data and discuss results. Also, we present a new factorization algorithm, which can handle non-linear relationships in the data, based on the idea of kernels - Kernel SVD.

1 Introduction

1.1 Recommender systems

A recommender system is an algorithm that is capable of predicting items for a user and rank these objects due to the user's subjective desire. The popularity of recommender systems has been increasing altogether with the popularity of the Internet. A lot of services, e.g., Amazon, Netflix, and Facebook have provided their users with the possibility to rate the goods, movies or to like the posts, so then they can estimate preferences of their users and offer them more relevant content. Nowadays there exist two main approaches to build such a system:

- Collaborative filtering (CF)
- Content-based filtering

Collaborative filtering approach based on a simple idea: we assume that if user Alice rated some item A and user Bob rated the same item A, then we can recommend to Bob some other items rated by Alice, which Bob didn't see. Here we also assume that if Bob didn't rate some probably relevant item, he just didn't familiar with it. More technically, in the CF we build user-item matrix, so its rows represent users and columns represents items. Elements of such matrix r_{ij} are ratings which user with index i gave to item with index j. Then we use this information to perform similarity search to find similar users (*memory-based system*) or use different machine learning techniques to predict user ratings for unrated items (*model-based system*).

The content-based approach is based on another simple idea: here we assume that if Alice rated some item A, then we can recommend similar to A item B. For this purposes we can build item-item matrix based on their properties or co-occurrences of such items. Then we build some classification model on items features and users ratings of them.

It worth to note that there also exist different modifications of such approaches and combinations of them (e.g., hybrid, knowledge-based, etc.), but in our work, we'll be interested specifically in the model-based collaborative filtering.

1.2 Collaborative filtering

Model-based collaborative filtering is one of the most widely spread approaches in recommender systems and has shown its efficiency [1]. The success of the model-based CF heavily relies on matrix decomposition or matrix factorization (MF), which was firstly used for recommendations during Netflix Prize competition and since then have become more popular. From the linear algebra

perspective matrix factorization is a solved problem, we have such well-studied methods like SVD and PCA. But the problem is that they aren't applicable in recommender systems setting as they require matrix to be without empty elements, however, the user-item matrix does have a lot of them. In this work, we'll look at the classic SVD from linear algebra to get a better understanding of what algorithms that will satisfy all constraints of recommender domain should be. Then we will look at two closely related algorithms proposed during Netflix Prize competition. Also, we will provide the results of the evaluation of those algorithms on the common dataset in this field.

2 Matrix factorization

In this section, we will describe classic SVD and its properties, which will give an intuition about how to construct similar to SVD algorithms capable of dealing with missing values.

To describe SVD regarding recommender systems, let's assume that our user-item matrix hasn't empty values or we simply imputed them with some value (e.g., mean or zero). Then we can perform classic SVD (we need to note that matrix imputation is meaningless from recommender perspective and can lead to wrong results, but here we do this only to show the idea behind classic SVD in recommender setting):

$$M = U\Sigma V^\top$$

We can interpret each of the matrices of the decomposition concerning the domain. Matrix U corresponds to the left-singular vectors, i.e., typical users. Matrix V corresponds to the right-singular vectors, which describes the typical items. The "weight" of the typical user/item is described by corresponding singular value. To obtain a low-rank approximation of the rating matrix R, we get k typical users vectors, items vectors, corresponded to the biggest singular values. For convention, let it will be U_k, Σ_k, and V_k.

$$M_k = U_k\Sigma_k V_k^\top$$

Such approximation is the best in all k-rank matrices, i.e

$$R_k = \underset{\hat{R}:\ rank(\hat{R})=k}{\operatorname{argmin}} \sum_{u,\ i}(R_{ui} - \hat{R}_{ui})^2$$

Moreover, we also state that it is the best in all matrices with rank $\leq k$ because the smaller rank of approximation will give the smaller explained variance of the estimator. So, R_k is the solution to the optimization:

$$R_k = \underset{\hat{R}:\ rank(\hat{R})\leq k}{\operatorname{argmin}} \sum_{u,\ i}(R_{ui} - \hat{R}_{ui})^2$$

The solution is valid also for mean squared error (MSE) because Frobenius norm and MSE are the same up to the constant. We will use root MSE in further experiments for bigger interpretability of the results.

2.1 Funk SVD

Simon Funk proposed an alternative [2] in 2006, which has similar properties to the SVD, but with specific improvement: we can ignore the empty elements and perform direct optimization with non-empty ratings. Name "Funk SVD" isn't correspond to the truly SVD method, rather shows an evolution of the ideas of SVD in recommender systems domain. The main assumption that the user's rating for an item depends on the similarity between user-feature and item-feature vectors, whose dimension is smaller than both number of users and items.

$$\hat{r}_{ui} = p_u^\top q_i \iff \hat{R} \approx PQ$$

The $\boldsymbol{p_u}$ and $\boldsymbol{q_i}$ for each user-item pair are vectors, and easy to see that we can write it as a matrix product. Similarly, we can define biased version of the above equality, i.e incorporate into the model bias by user and item, and also mean non-trainable constant μ:

$$\hat{r}_{ui} = \mu + a_u + b_i + p_u^\top q_i \iff \hat{R} \approx PQ + \boldsymbol{a}\mathbf{1}^\top + \mathbf{1}\boldsymbol{b}^\top + (\mu,\ \dots,\ \mu)\mathbf{1}^\top$$

Optimization task will be the same as for singular value decomposition, we will find the matrices $P,\ Q$ (and parameters $\vec{a},\ \vec{b}$) that minimize functional:

$$L(P,\ Q,\ a,\ b) = \sum_{u,\ i}(r_{ui} - \hat{r}_{ui})^2 \to \min$$

In the non-biased option, matrix $\hat{R}$ has rank equal to or smaller than the size of the feature space. Let it will be $k \ll min(\text{number of users, number of items}) \implies rank(\hat{R}) \leq min(rank(P),\ rank(Q)) \leq k$. Hence, the optimization task will be the same as in the ordinal SVD, because cost functions and the optimization domains are identical in the two cases.
Also, we can introduce l_2 regularization term to constrain the size of the parameters of the model:

$$L_\lambda(P,\ Q,\ a,\ b) = \sum_{u,\ i}(r_{ui} - \hat{r}_{ui})^2 + \lambda(a_u^2 + b_i^2 + ||p_u||^2 + ||q_i||^2)$$

Let $e_{ui} = r_{ui} - \hat{r}_{ui}$. We'll perform stochastic gradient descent, which is a first order optimization, to find the gradients with a respect to the parameters:

$$\begin{cases} \frac{\partial L}{\partial a_u} = -(e_{ui} - \lambda a_u) \\ \frac{\partial L}{\partial b_i} = -(e_{ui} - \lambda b_i) \\ \frac{\partial L}{\partial \boldsymbol{p_u}} = -(e_{ui}\boldsymbol{q_i} - \lambda \boldsymbol{p_u}) \\ \frac{\partial L}{\partial \boldsymbol{q_i}} = -(e_{ui}\boldsymbol{p_u} - \lambda \boldsymbol{q_i}) \end{cases}$$

The algorithm has $O(nm \cdot epochs)$ time complexity and $O((k+1)(n+m))$ space complexity, because we have three nested loops and store $(k+1)(n+m)$ trainable parameters correspondingly. Need to notice, the computed above complexity is the worst case when the matrix has all non-empty elements, but in real life rating matrix has a very small number of existed ratings, so computation will be much quicker and derived complexity asymptotics need only for convention and understanding how situation will change "on the infinity".

Algorithm 1 *Funk SVD*

1. Download user-item matrix R
2. Initialize the weights of the model $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{a}$, $\boldsymbol{b}$ with random variable from standard normal distribution
3. Set parameters *epochs* - number of training epochs, λ - regularization parameter, γ - learning rate, n - number of users, m - number of items, k - number of latent factors
4. if training phase: For $m = 1$ to *epochs*:
   (a) For each element $r_{ui} \neq$ Null of matrix R:
      i. Compute $e_{ui} = r_{ui} - \hat{r}_{ui}$
      ii. Compute gradients $\frac{\partial L}{\partial a_u}$, $\frac{\partial L}{\partial b_i}$, $\frac{\partial L}{\partial \boldsymbol{p_u}}$, $\frac{\partial L}{\partial \boldsymbol{q_i}}$
      iii. Update parameters $\begin{cases} p_u \leftarrow p_u - \gamma \frac{\partial L}{\partial \boldsymbol{p_u}} \\ q_i \leftarrow q_i - \gamma \frac{\partial L}{\partial \boldsymbol{q_i}} \\ a_u \leftarrow a_u - \gamma \frac{\partial L}{\partial a_u} \\ b_i \leftarrow b_u - \gamma \frac{\partial L}{\partial b_i} \end{cases}$
5. if test phase (input user index u and item index i):
   (a) Get p_u and q_i
   (b) Compute and return $\hat{r}_{ui} = \mu + a_u + b_i + p_u^\top q_i$

2.2 Probabilistic Matrix Factorization

PMF approach is pretty similar to the previous one, but it has a deeper explanation based on the probability theory and a couple of enhancements compared to the Funk SVD. Firstly we need to introduce a probabilistic model for ratings. Let's suppose that the rating of each user for each movie is normally distributed around $p_i^T q_j$ with some random noise σ^2:

$$p(R|P, Q, \sigma^2) = \prod_i \prod_j [N(r_{ij}|p_i^T q_j, \sigma^2)]^{I_{ij}},$$

where $N(x|\mu, \sigma^2)$ is a gaussian distribution and I_{ij} is a indicator function, which equals to 1 if i-th user rated j-th movie and 0 otherwise.

Now we can use MLE to get point estimates of the p_i and q_j $(\forall i, j)$. Let's write down the log likelihood function:

$$log(p, q|r, \sigma^2, \sigma_p^2, \sigma_q^2) = -\frac{1}{2\sigma^2} \sum_i \sum_j I_{ij}(r_{ij} - p_i^T q_j)^2 - \frac{1}{2\sigma_p^2} \sum_i p_i^T p_i - \frac{1}{2\sigma_j^2} \sum_j q_j^T q_j -$$
$$\frac{1}{2}((\sum_i \sum_j I_{ij}) log(\sigma^2) + NDlog(\sigma_p^2) + MDlog(\sigma_q^2)$$

Here we have σ_p^2 - variance over users feature vectors, σ_q^2 - variance over items feature vectors (they are the hyperparameters of the model). Now maximizing log-likelihood we get the following cost function:

$$J(p,q) = \frac{1}{2}\sum_i \sum_j I_{ij}(r_{ij} - p_i^T q_j)^2 + \frac{\sigma^2}{2\sigma_p^2}\sum_i ||p_i||_{Frob}^2 + \frac{\sigma^2}{2\sigma_q^2}\sum_j ||q_j||_{Frob}^2$$

Also, it's important to apply a sigmoid function over dot product $q_i^T p_j$ because it scales the rating to the [0, 1] interval so that model won't output ratings higher than some upper bound. In order to get proper results we also need to scale input data: $\hat{x} = \frac{x-1}{4}$ - input data scaling to the [0, 1].

We can see that the resulted objective function of PMF is the same as SVD except for regularization terms and sigmoid trick. In SVD we set regularization parameter independently of σ_p^2 and σ_q^2, but in PMF due to probabilistic interpretation, regularization rates are defined by initial variances.

For more details, please refer to the original paper [3].

2.3 Kernel SVD

Standard Funk SVD algorithm supposes rating depends on linear similarity measure (or dot product) between user-feature and item-feature vectors. But what if real user-feature and item-feature do form the non-linear manifolds? In this case, we propose to find non-linear transformation $\phi : R^n \to R^m$ from one space to another space, where we can apply the linear similarity measure, and obtain the rating. For simplifying, consider one function for users and items spaces. The scalar product between non-linearly transformed vectors called *kernel*:

$$K(x,\ y) = \phi(x)^\top \phi(y),$$

Let $\boldsymbol{p_u}$ and $\boldsymbol{q_i}$ also represent the users and items vectors correspondingly. We will predict the output rating as $K(p_u,\ q_i)$ and apply the same optimization procedure as in the Funk SVD algorithm:

$$L(P,\ Q) = \textstyle\sum_{u,\ i}(r_{ui} - \phi(p_u)^\top \phi(q_i))^2 \to \min$$

The main difference between Funk SVD and our Kernel SVD is the updating rule in the stochastic gradient descent because of the function ϕ influences on the gradient:

$$\begin{cases} \frac{\partial L}{\partial \boldsymbol{p_u}} = 2(r_{ui} - \hat{r}_{ui})\frac{\partial L}{\partial \boldsymbol{p_u}}\phi(p_u)^\top \phi(q_i) = 2(r_{ui} - \hat{r}_{ui})J_\phi(p_u)\phi(q_i) \\ \frac{\partial L}{\partial \boldsymbol{q_i}} = 2(r_{ui} - \hat{r}_{ui})\frac{\partial L}{\partial \boldsymbol{q_i}}\phi(p_u)^\top \phi(q_i) = 2(r_{ui} - \hat{r}_{ui})J_\phi(q_i)\phi(p_u), \end{cases}$$

where J_ϕ function is Jacobian of the ϕ, i.e. first-order approximation of the non-linear transformation.

We didn't consider the biased option of the algorithm, because we believe that non-linearity ϕ can incorporate this property into itself. Also important to note, that from a theoretical perspective we must bound the class of possible functions ϕ, because $K(x,\ y)$ must be a proper inner product in the output space of the ϕ. But on the first stage, we omitted these and concentrated

only on the numerical evaluation.

The Jacobian J_ϕ was computed numerically via central finite differences, i.e:

$$J_\phi(x)_{ij} \approx \hat{J}_\phi(x)_{ij} = \frac{\phi_i(\boldsymbol{x}+h\boldsymbol{e_j})-\phi_i(\boldsymbol{x}+h\boldsymbol{e_j})}{h}, \text{ where}$$

$e_j = (0_1,\ ...,\ 1_j,\ ...,\ 0_n)$ and h is the very small increment. The finite difference algorithm has $O(h^2)$ accuracy, so we can take $h = 10^{-5}$ and obtain 10^{-10} accuracy for every element of the $J_\phi(x)$.

The overall algorithm will be the same as Funk SVD, but the update rule will use new gradients with respect to the parameters, as derived above. Numerical computation of the jacobian changes the algorithms complexity: $O(epochs \cdot nm \cdot g^2)$ for time complexity, where g is the dimension of the resulted space after applying ϕ. If $g = 10$, then algorithm will work 100 times slower than ordinal Funk SVD. Unfortunately, we can't avoid such problem, because in other way we must compute derivatives theoretically, what can be difficult procedure.

As non-linear transformation ϕ we considered polynomial combinations of the input vector, i.e all possible products of all combinations of vector's coordinates. For example, let dim $Domain(\phi) = 3$ and degree of transformation is 2:

$$v = (a,\ b,\ c) \implies \phi(v) = (1,\ a,\ b,\ c,\ ab,\ ac,\ bc,\ a^2,\ b^2,\ c^2)$$

Obviously, it's non-trivial to write the explicit expression for such function, so we used already written numerical function in the Scikit-learn library [4].

3 Experiments

In our experiments, we tested algorithms on the open dataset MovieLens-100k. Dataset describes 100836 five-star rating across 9742 movies and other meta-information from MovieLens resource [5], a movie recommendation service. These data were created by 610 users and collected between 1996 and 2018 years. We were interested only in the user-film interactions, so we considered only the user-film-rating table and constructed user-item matrix for preprocessing and data exploration purposes.

We performed grid search over the set of hyperparameters σ_p^2, σ_q^2, D. Also, we tried different regularization parameters (for SVD only, for PMF amount of regularization is defined by variances) and different learning rates. Kernel SVD algorithm we tested only on few hyperparameters without regularization due to the long running time. All performance measures were done by cross-validation over five different folds. You can find the results of the best models with corresponding values of hyperparameters in table 1.

Name	**Latent space size**	**Number of epochs**	**Mean RMSE (5 folds)**
VanillaSVD	10	20	0.9457
VanillaPMF	10	1000	0.9613
SurpriseSVD	10	20	0.9363
SurprisePMF	10	100	0.9667
KernelSVD	$3 \to 10$	20	0.9817

Table 1. Evaluation results

We see that our implementations perform on the same level as implementations from Surprise library. We had an assumption that PMF will perform better than SVD, as it has a better probabilistic explanation, but we found out that the performance of both models is almost equal. It can be explained in such way: both our implementation and ones from Surprise are vanilla versions of the original algorithm (we didn't use all improvements for PMF such as automatic complexity control and constraints). Although we thought that Kernel SVD would work better than all algorithms, but we faced problems during the optimization step and observed slow convergence of the algorithm. Final score of such method was slightly smaller than PMF. We think that the reason of such behaviour is a numerical computation of the Jacobian matrix.

The code with implemented algorithms and experiments can be found via the link [6].

4 Conclusion & Future work

In this report, we described the importance of efficient and scalable matrix factorization algorithms for recommender domain. We explained two widely used methods and proposed our new approach based on the idea of a kernel transformation. We evaluated our implementations of the algorithms and compared them to the widely used library Surprise [7]. We got pretty close results. Small deviations can be explained by the stochastic nature of the optimization process and little differences in the implementation. We proposed Kernel SVD algorithm, but didn't achieve any significant improvementd due to optimizational problems.

In future work, we can look at other approaches to matrix factorization, such as SVD++, which allow using implicit ratings, and Non-negative MF. Also, we can improve the proposed kernel method by solving the problem not numerically, but with the help of the OLS, thus we will be able to use true kernel trick, which can accelerate the algorithm's working time [8].

References

[1] Neil D. Lawrence, Raquel Urtasun. Non-linear Matrix Factorization with Gaussian Processes. ICML, 2009

[2] Simon Funk. Funk-SVD. https://sifter.org/simon/journal/20061211.html

[3] Ruslan Salakhutdinov, Andriy Mnih. Probabilistic Matrix Factorization. NIPS, 2007.

[4] Scikit-learn library. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Polynomia

[5] MovieLens. https://grouplens.org/datasets/movielens/

[6] Code with experiments. https://github.com/Vaden4d/UCU/tree/master/lareport

[7] Surprise library. http://surpriselib.com

[8] Xinyuan Sun. Kernel Methods for Collaborative Filtering. 2016.