

# A Survey of Latent Factor Models for Recommender Systems and Personalization

Data Mining Course Project

Johns Hopkins University

Shalin Shah  
sshah100@jhu.edu

## Abstract

Recommender systems aim to personalize the experience of a user and are critical for businesses like retail portals, e-commerce websites, book sellers, streaming movie websites and so on. The earliest personalized algorithms use matrix factorization or matrix completion using algorithms like the singular value decomposition (SVD). There are other more advanced algorithms, like factorization machines, Bayesian personalized ranking (BPR), and a more recent Hebbian graph embeddings (HGE) algorithm. In this work, we implement BPR and HGE and compare our results with SVD, Non-negative matrix factorization (NMF) using the MovieLens dataset.

*Keywords:* Recommender Systems, Personalization Algorithms, Hebbian Graph Embeddings, Open Source, Python, Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), Bayesian Personalized Ranking

## 1 Introduction

The earliest personalization algorithms were based on matrix factorization or matrix completion, based on singular value decomposition (SVD) [1] [2]. An  $L_2$  loss is a common choice to learn the latent factors. Bayesian personalized ranking (BPR) [3] [4] is a recent algorithm that learns the latent factors of users and items and places a zero mean Gaussian prior on the latent factors. Non-negative matrix factorization [5] is similar to SVD, but the user and item latent factors are kept positive. Hebbian graph embeddings (HGE) [6] uses a graph to learn the embeddings of users and items based on propagating the embeddings across graph neighborhoods, similar to PageRank [7]. Other personalization algorithms are compared in our paper [8].

We implement HGE and BPR and we use the surprise library [9] for SVD

and NMF.

[https://surprise.readthedocs.io/en/stable/matrix\\_factorization.html](https://surprise.readthedocs.io/en/stable/matrix_factorization.html)

We compare all four algorithms on the MovieLens 600 dataset (small) [10].

## 2 The Algorithms

### 2.1 Non-negative Matrix Factorization (NMF)

NMF factorizes the user-movie matrix into two matrices  $W$  and  $H$  such that both are non-negative. The original matrix  $A$  is also assumed to be non-negative (which is true in the case of movie ratings which are between 0 and 5). Algorithms for NMF are described in the paper [5]. There are also algorithms based on non-negative least squares which put a non-negativity constraint on the least squares formulation.

NMF factorizes the user-movie matrix  $A$  into two matrices  $W$  and  $H$

$$A = WH$$

If  $A$  is an  $m \times n$  matrix of  $m$  users and  $n$  movies, then  $W$  is an  $m \times p$  matrix and  $H$  is a  $p \times n$  matrix. The columns and rows of  $W$  and  $H$  are  $p$  dimensional which is computed to be of dimension significantly lesser than the dimensions of  $A$  which creates a low dimensional approximation to the original matrix.

The ratings for users in the matrix  $A$  are computed using the following:

$$a_i = Wh_i$$

where,  $a_i$  is the  $i^{th}$  column vector of  $A$  which could represent ratings of a user for all movies, and  $h_i$  is the  $p$  dimensional  $i^{th}$  column vector of the matrix  $H$ .

NMF is very applicable to the problem of recommender systems as it makes it easier to interpret the results. We use the surprise library [9] to compute the NMF matrices and to do inference.

### 2.2 Hebbian Graph Embeddings (HGE)

An embedding is a numerical vector representing an entity. For instance a user is an entity, a movie is an entity, products in a retail website are entities. Hebbian graph embeddings [6] learns the embeddings of users and movies and computes recommendations based on an inner product between a user embedding and a movie embedding. It is a derivative-free method that uses multidimensional centrality updates like PageRank [7]. This algorithm is patent pending [11].

The algorithm is shown in figure 1. Users and movies induce a bipartite preference graph where there is an undirected edge between a user and a movie if the user has rated that movie with a rating  $\geq 4$  i.e. the user liked the movie.

The embeddings of users and movies are initialized using a 0 mean Gaussian with a diagonal covariance matrix  $\sigma^2 I$ .

$$\begin{aligned} u_i &\sim N(\vec{0}, \sigma^2 I) \\ v_i &\sim N(\vec{0}, \sigma^2 I) \end{aligned}$$

Embeddings are propagated across the graph and so, embeddings of users and movies are jointly learnt in the same space. Previously, we used our algorithm for link prediction and for item-item recommendations. But as we do in this article, the algorithm can just as easily applied to learn embeddings for personalization using the user-movie bipartite graph.

Let  $u_i$  be the vector that represents a user and let  $v_i$  represent a vector for the movie. If there is an edge between user  $u$  and movie  $v$  in the training data, the embeddings of both are propagated using a Gaussian perturbed propagation rule (similar to annealing).

$$\begin{aligned} g_i &\sim N(v_i, \sigma^2 I) \\ u_i &= u_i + g_i \\ h_i &\sim N(u_i, \sigma^2 I) \\ v_i &= v_i + h_i \end{aligned}$$

The value of  $\sigma^2$  could be anywhere from 0.01 to 10 depending on the size of the problem. We use 0.01 for our purposes. Ablation studies show that the variance does not have a large effect on the results as long as the variance is not too large.

We compute the hit rates for HGE in tables 1 and 2 and results show that HGE performs very competitively as compared to SVD, NMF and BPR. Though, the difference between HGE and SVD is not very significant when the dimensionality is large. This difference could reduce further if the dimensionality is increased beyond 100.

Also, HGE is embarrassingly parallelizable which means that it can be applied to very large graphs with billions of nodes quite easily if a cluster of servers is available.

### 2.3 Singular Value Decomposition (SVD)

SVD is used to factorize non-square matrices, and efficient algorithms are available in MATLAB. We use the surprise library [9] to compute SVD and to do inference. SVD is described in detail in my article [1].

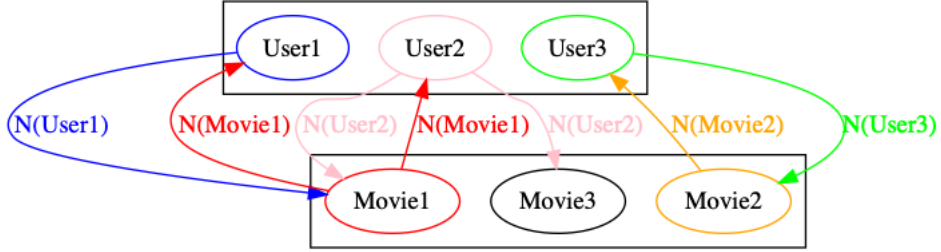


Figure 1: Hebbian Graph Embeddings (Bipartite User-Movie Graph)

SVD factorizes the user-movie matrix  $A$  into three matrices:

$$A \sim U\Sigma V^T$$

The entries in  $\Sigma$  are called singular values. The dimensionality of the problem is the number of singular values kept and the rest discarded. This is called rank reduction.

So,  $U\Sigma$  contains the latent factors of the users and  $\Sigma V^T$  contains the latent factors of the movies.

If  $A$  is an  $m \times n$  matrix,  $U\Sigma$  in its rank reduce form is an  $m \times k$  matrix and  $\Sigma V^T$  is a  $k \times n$  matrix in its rank reduced form. SVD was used extensively in the Netflix recommender systems challenge [2].

As the results show, SVD performs quite well especially if the dimensionality is large. It performs competitively against NMF and BPR and is also quite close to HGE. Increasing the dimensionality above 100 may improve its results further.

## 2.4 Bayesian Personalized Ranking (BPR)

BPR is a recent algorithm [3] which learns the embeddings of entities and also puts a zero mean Gaussian prior on the latent factors.

The algorithm is quite simple. It first samples a random user, and then samples an item that the user interacted with and another item that the user did not interact with (an item is considered a preference of a user if the user gave it a rating of  $\geq 4$ ).

If user  $u$  prefers item  $i$  against item  $j$ , the preference is quantified using the following equation:

$$P(i > j|u) = \frac{e^{x_{ui}}}{e^{x_{ui}} + e^{x_{uj}}}$$

where,  $x_{ui} = \langle v_u v_i \rangle$

where  $v_u$  is the user’s latent factor and  $v_i$  is the item latent factor.

The Gaussian prior results in an  $L_2$  regularization term in the loss function.

BPR performs very well even if the dimension of the factors is smaller. But SVD outperforms BPR, even more noticeably, when the dimension is increased, as shown in table 2 and figure 2.

BPR can be easily implemented in a parallel distributed algorithm, and can be used even when the number of entities is in millions (though this would need a cluster of servers, if the dimensionality is large).

Our open source implementation of BPR is available on GitHub:

<https://github.com/shah314/BPR>

### 3 The MovieLens Dataset

The MovieLens [10] Small data consists of about 600 users, 9000 movies and 100,000 ratings.

<https://grouplens.org/datasets/movielens/latest/>

We use this data set for all our experiments. We divide the movies rated by a user into a 30% test set and a 70% training set for each user. We compute the hit rates in table 1 on the train and the test sets. In table 2, we use only the test set across varying dimensionality.

### 4 Results

The results of our analysis is shown in tables 1 and 2 and in figure 2.

We compute the results as hit rates computed on the 30% held out test set. Let  $r_{ij}$  be the rating computed by the algorithm for the  $i^{th}$  user for the  $j^{th}$  movie. Sorting all movies in decreasing order of  $r_{ij}$  gives a ranked list of movies for a particular user. If any one of the movies occurring in a user’s test set occurs in the top 10 movies of the sorted list, it is considered a hit and assigned a value of 1. The number of hits divided by the number of users gives the hit rate.

Table 1: Hit Rates (30% held out test set) @ position 10

<b>DataSet</b>	<b>Algorithm</b>	<b>Dimension</b>	<b>Test</b>	<b>Train</b>
MovieLens 600	HGE	10	53.2%	77%
MovieLens 600	NMF	10	3.6%	7.2%
MovieLens 600	BPR	10	61.5%	73.6%
MovieLens 600	SVD	10	37.4%	64.9%
MovieLens 600	Random	10	3.1%	4.8%

Table 2: Varying Dimensionality Hit Rates (30% held out test set) @ position 10 on MovieLens 600

<b>Algorithm</b>	<b>10</b>	<b>20</b>	<b>30</b>	<b>40</b>	<b>50</b>	<b>75</b>	<b>100</b>
HGE	53.2	55	55.8	57.1	57	55.3	58.9
NMF	3.6	13.6	24.1	25	32	38	36.6
BPR	61.5	37.6	36.3	35.1	30	36	37.4
SVD	37.4	40.4	47	42	43.2	47.5	53

Table 1 shows the hit rates of all four algorithms with a dimensionality of 10 on the test and train data sets. HGE, BPR and SVD perform quite well, but NMF performs poorly (this is because of the low dimension as we see in table 2).

Table 2 and Figure 2 show the hit rates across varying dimensionality. NMF is very sensitive to the dimension, as the results show. For dimensionality of 100, there is only a small difference between HGE and SVD. HGE and SVD outperform BPR and NMF. The random row in table 1 shows the hit rate without any training (i.e. just initializing the latent factors randomly using a zero mean Gaussian with a small diagonal covariance matrix).

## 5 Varying Dimensionality

We ran all four algorithms for dimensionality ranging from 10 to 100 and find results shown in table 2. The dimension could be increased beyond 100 but that may require more computing resources. As the results show, NMF and SVD are very sensitive to the dimension. NMF starts off with poor results, but its results improve as the dimension is increased, and its hit rate is almost identical to BPR for a dimensionality of 100. SVD also improves significantly as the size of the latent factors is increased. HGE and BPR perform well even if the dimension is small, which implies that if computing resources are scarce, they might be the first of the few algorithms to try.

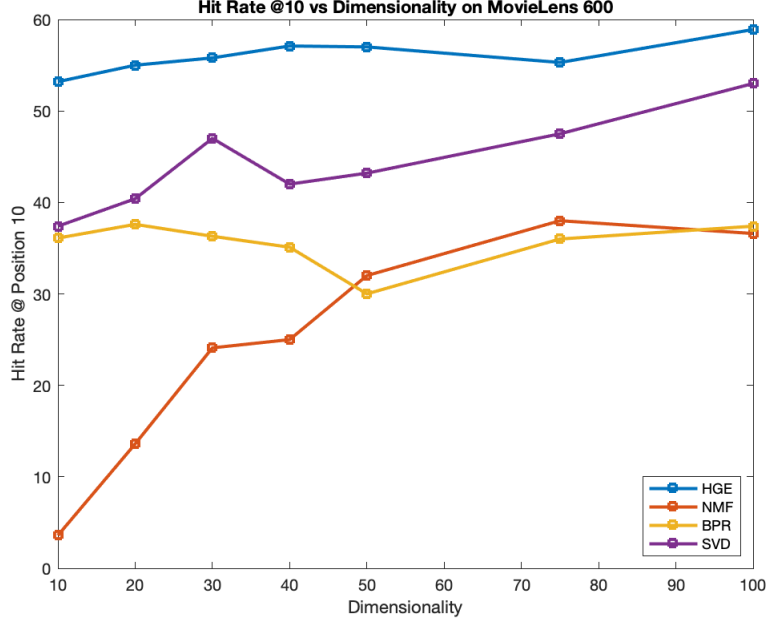


Figure 2: Hit Rates @ Position 10 with Varying Dimensionality

## 6 Conclusion

I compared four algorithms for personalization in a recommender system on the MovieLens data set. As the results shown, HGE and SVD outperform BPR and NMF, especially when the dimensionality is high. More experiments are needed to see if NMF and BPR catch up if the dimensionality is increased beyond 100. BPR was recently used in a hierarchical model in [12] which shows promising results.

It is worth noting that HGE and BPR are very easy to parallelize using a distributed algorithm which makes them highly applicable to larger data sets of millions of users and movies. We note some of the results in [8], though this is the first paper applying HGE to user-movie personalization using the bipartite user-movie graph.

It remains to be seen if the results carry forward if the number of users and movies are increased. MovieLens has a 1M data set which could be used to perform further experiments, but this might need a cluster of servers, for efficient computation.

## References

- [1] Shalin Shah. Introduction to matrix factorization for recommender systems. Technical report, Johns Hopkins University, 2018.
- [2] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.
- [3] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.
- [4] Alfredo Láinez Rodrigo and Luke de Oliveira. Distributed bayesian personalized ranking in spark.
- [5] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.
- [6] Shalin Shah and Venkataramana Kini. Hebbian graph embeddings. *arXiv preprint arXiv:1908.08037*, 2019.
- [7] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.
- [8] Venkataramana Kini and Shalin Shah. Comparison of Three Recent Personalization Algorithms. 2020.
- [9] Nicolas Hug. Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174, 2020.
- [10] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.
- [11] Shalin Shah and Venkataramana Kini. Link prediction using hebbian graph embeddings, 2020. US Patent App. 16/855,761.
- [12] Yuchen Zhang, Amr Ahmed, Vanja Josifovski, and Alexander Smola. Taxonomy discovery for personalized recommendation. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 243–252, 2014.
- [13] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.