# Recommender System Project: Probabilistic Approch to SLIM and MF Techniques

Flavio Di Palo
flavio.dipalo@mail.polimi.it

Alessio Baccelli
alessio.baccelli@mail.polimi.it

October 2018

## 1 Introduction

The aim of this project is to explore if the introduction of a probabilistic constraint during the learning phase of the SLIM[1] and Matrix Factorization[4] algorithms can lead to an improvement on the performance with respect to the traditional approach.

## 2 SLIM Problem Statement

Given $U$ the set of all the users and $I$ the set of all the items, we want to apply a constraint on the similarity matrix $W$, having dimensions $I \times I$, that defines how much a given item is similar to the other one.
In particular we are going to explore if a constraint that forces the sum of the column of the Similarity matrix equal to 1 during the learning process can lead to an improvement.

### 2.1 Slim RMSE

Given an User Rating Matrix $A$ with dimensions $m \times n$, the aim of the SLIM RMSE algorithm is to compute, with an iterative approach, a sparse matrix $W$ with dimensions $n \times n$ that is the result of the following optimization problem:

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \frac{1}{2}\|A - AW\|_F^2 + \frac{\beta}{2}\|W\|_F^2 + \lambda\|W\|_1 \\ \text{subject to} \quad & W > 0,\ diag(W) = 0 \end{aligned}$$

We now want to add, as a constraint, that every column of the W matrix sums to 1 at each step of the learning process. Therefore, we add the constraint:

$$w_{1,j} + w_{2,j} + ... + w_{m,j} = 1 \quad \forall\ j = 0, 1, ..., m$$

where the element $w_{i,j}$ is the element of the W matrix at the $i$-th row and $j$-th column. Doing so we mean to give a probabilistic meaning to the elements present on the rows of W matrix.

## 3 Solution Approach and Metodology

### 3.1 Constraint Formalization

In order to achieve the result we used constraint optimization methods that allowed us to project the gradient and search for the optimum in a constrained space[2]. Considering the optimization problem in the form:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \textit{subject to} \quad & A\mathbf{x} = \mathbf{b} \end{aligned}$$

where $f : R^n \to R,\ A \in R^{m\times n},\ m < n,\ rankA = m,\ \mathbf{b} \in R^m$
In this case the specific structure of the constraint set allows us to compute the projection operator $\Pi$ using the *orthogonal projector*. Specifically, $\Pi[\mathbf{x}]$ can be defined using the orthogonal projector matrix $\mathbf{P}$ given by:

$$P = I_n - A^T(AA^T)^{-1}A$$

Considering that the projected gradient algorithm update has the form

$$x^{(k+1)} = \Pi[x^{(k)} - \alpha_k \Delta f(x^{(k)})]$$

we can express the gradient update exploiting projector matrix P as follows:

$$\Pi[x^{(k)} - \alpha_k \Delta f(x^{(k)})] = x^{(k)} - \alpha_k P \Delta f(x^{(k)})$$

assuming that $x^{(k)} \in \Omega$ where $\Omega$ is the considered constraint

### 3.2 Implementation and Testing

We have implemented a SLIM RMSE Recommender algorithm in Cython out implmentation is able to work on both the constrained and unconstarained problem. Here we present the results obtained comparing the classical algorithm to the constrained one. We made a comparison between the two on the Movielens 1M and on the Movielens 10M dataset. On the latter, we also performed different kind of tests, by taking an item-item KNN collaborative Recommender as a benchmark. The first test was done by taking into account only the 67% of the least popular items, in order exclude the popularity bias. The next type of analysis we performed was done by deleting 33% and 66% of the interactions at random. In the end, we made two tests, by taking only users and items with at least 10 interactions in the first one and by setting the threshold to 50 in the second one.
In all the tests we used the 60% of the dataset for training, the 20% for validation and the 20% for testing. The hyperparameters for the SLIM RMSE algorithm have been tuned with a Bayesian Search Algorithm.

## 4 Results

### 4.1 Movielens 1M

In the following table is presented the comparison between the SLIM RMSE with and without normalization over the Movielens 1M dataset:

| **Algorithm** | **Results** | | | | | | **Configuration** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AUC** | **Precision** | **Recall** | **MAP** | **NDCG** | **MRR** | **l1 norm** | **l2 norm** | **topK** |
| Slim RMSE (non normalized) | 0.516 | 0.408 | 0.049 | 0.330 | 0.113 | 0.612 | 0.01 | 0.01 | 50 |
| Slim RMSE (normalized) | 0.512 | 0.412 | 0.048 | 0.335 | 0.112 | 0.608 | 0.01 | 0.01 | 50 |

Table 1: Results on Testing Set, full dataset

### 4.2 Movielens 10M

#### 4.2.1 Least popular items

In the following tables is presented the comparison between the SLIM RMSE with and without normalization over the Movielens 10M dataset. The first table is obtained with the full dataset, while the second one by eliminating the top 33% popular items:

| **Algorithm** | **Results** | | | | | | **Configuration** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AUC** | **Precision** | **Recall** | **MAP** | **NDCG** | **MRR** | **l1 norm** | **l2 norm** | **topK** |
| Slim RMSE (non normalized) | 0.358 | 0.242 | 0.073 | 0.184 | 0.111 | 0.417 | 0.001 | 0.01 | 100 |
| Slim RMSE (normalized) | 0.325 | 0.208 | 0.063 | 0.138 | 0.099 | 0.369 | 0.01 | 0.001 | 100 |
| KNN | 0.362 | 0.236 | 0.084 | 0.160 | 0.119 | 0.414 | - | - | - |

Table 2: Results on Testing Set, full dataset

| **Algorithm** | **Results** | | | | | | **Configuration** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AUC** | **Precision** | **Recall** | **MAP** | **NDCG** | **MRR** | **l1 norm** | **l2 norm** | **topK** |
| Slim RMSE (non normalized) | 0.325 | 0.194 | 0.066 | 0.129 | 0.096 | 0.361 | 0.001 | 0.01 | 100 |
| Slim RMSE (normalized) | 0.329 | 0.203 | 0.065 | 0.136 | 0.103 | 0.371 | 0.01 | 0.001 | 100 |
| KNN | 0.357 | 0.234 | 0.085 | 0.158 | 0.119 | 0.409 | - | - | - |

Table 3: Results on Testing Set, without top 33% top popular items

#### 4.2.2 Deleting random interactions

In the following tables is presented a comparison the SLIM RMSE with and without normalization over the Movielens 10M dataset. The first table is obtained by deleting 33% of the the interactions (taken at random), while the second table was obtained by increasing the percentage of deleted interactions up to 66%.

| Algorithm | Results | | | | | | Configuration | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AUC** | **Precision** | **Recall** | **MAP** | **NDCG** | **MRR** | **l1 norm** | **l2 norm** | **topK** |
| Slim RMSE (non normalized) | 0.213 | 0.104 | 0.053 | 0.065 | 0.066 | 0.218 | 0.001 | 0.01 | 100 |
| Slim RMSE (normalized) | 0.241 | 0.125 | 0.059 | 0.078 | 0.080 | 0.251 | 0.01 | 0.001 | 100 |
| KNN | 0.221 | 0.110 | 0.058 | 0.066 | 0.069 | 0.225 | - | - | - |

Table 4: Results on Testing Set, deleting 33% of interactions

| Algorithm | Results | | | | | | Configuration | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AUC** | **Precision** | **Recall** | **MAP** | **NDCG** | **MRR** | **l1 norm** | **l2 norm** | **topK** |
| Slim RMSE (non normalized) | 0.107 | 0.044 | 0.043 | 0.031 | 0.044 | 0.101 | 0.001 | 0.01 | 100 |
| Slim RMSE (normalized) | 0.104 | 0.045 | 0.040 | 0.029 | 0.043 | 0.100 | 0.01 | 0.001 | 100 |
| KNN | 0.092 | 0.038 | 0.035 | 0.025 | 0.034 | 0.087 | - | - | - |

Table 5: Results on Testing Set, deleting 66% of interactions

#### 4.2.3 K interactions

In the following tables is presented a comparison the SLIM RMSE with and without normalization over the Movielens 10M dataset. The first table is obtained without considering items and users with less than 10 interactions, the second one by setting the threshold at 50.

| Algorithm | Results | | | | | | Configuration | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AUC** | **Precision** | **Recall** | **MAP** | **NDCG** | **MRR** | **l1 norm** | **l2 norm** | **topK** |
| Slim RMSE (non normalized) | 0.356 | 0.234 | 0.072 | 0.181 | 0.109 | 0.409 | 0.001 | 0.01 | 100 |
| Slim RMSE (normalized) | 0.420 | 0.301 | 0.084 | 0.239 | 0.137 | 0.483 | 0.01 | 0.001 | 100 |
| KNN | 0.359 | 0.236 | 0.083 | 0.159 | 0.118 | 0.413 | - | - | - |

Table 6: Results on Testing Set, threshold set to 10

| Algorithm | Results | | | | | | Configuration | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AUC** | **Precision** | **Recall** | **MAP** | **NDCG** | **MRR** | **l1 norm** | **l2 norm** | **topK** |
| Slim RMSE (non normalized) | 0.351 | 0.300 | 0.051 | 0.208 | 0.095 | 0.416 | 0.001 | 0.01 | 100 |
| Slim RMSE (normalized) | 0.515 | 0.393 | 0.068 | 0.318 | 0.143 | 0.606 | 0.01 | 0.001 | 100 |
| KNN | 0.398 | 0.283 | 0.054 | 0.190 | 0.106 | 0.473 | - | - | - |

Table 7: Results on Testing Set, threshold set to 50

## 5 Matrix Factorization Problem Statement

Matrix factorization is a class of collaborative filtering algorithms that works by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices. The idea behind matrix factorization is to represent users and items in a lower dimensional latent space, basing on the assumption that there are ”latent features” that are able to summarize the information contained in the original dataset.

### 5.1 FunkSVD

The original algorithm proposed by Simon Funk in his blog post [3] factorized the user-item rating matrix R as the product of two lower dimensional matrices, the first one has a row for each user, while the second has a column for each item. The row or column associated to a specific user or item is referred to as latent factors. The predicted ratings can be computed as $\tilde{R} = HW$ where $\tilde{R} \in R^{users \times items}$ is the estimated user-item rating matrix, $H \in R^{users \times latentfactors}$ contains the user's latent factors and $W \in R^{latentfactors \times items}$ the item's latent factors.
Specifically, the predicted rating user u will give to item i is computed as:

$$\tilde{r}_{ui} = \sum_{f=0}^{nfactors} H_{u,f} W_{f,i}$$

It is possible to tune the expressive power of the model by changing the number of latent factors. Increasing the number of latent factor will improve personalization

The objective of FunkSVD is to minimizes the following function:

$$\arg\min_{H,W} \|R - \tilde{R}\|_{\mathrm{F}} + \alpha\|H\| + \beta\|W\|$$

Where $\|.\|_{\mathrm{F}}$ is defined to be the frobenius norm.

## 6 Solution Approach and Metodology

### 6.1 Constraint Formalization

In the Matrix Factorization scenario we are forcing the matrices H and W to have every column summing to 1 at each step of the learning process. We apply the same constrained optimization technique described in Section 3.1 to the two Matrices

### 6.2 Implementation and Testing

We implemented a Cython version of the FunkSVD algorithm that is able to work both in the Noramalization framework discussed above and the classical FunkSVD scenario. Our algorithm are trained by considering 60% of the dataset for training and the remaining 40% equally splitted between testing set and validation set. Hyperparmeter tuning has been performed with a Bayesian Search algorithm, for each test we also report the value of the hyperparameters for the best configuration.

# 7 Results

## 7.1 Movielens 1M

In the following table is presented the comparison between the two version of FunkSVD algorithm on the Movielens1M dataset:

| **Algorithm** | **Results** | | | | | | **Configuration** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AUC** | **Precision** | **Recall** | **MAP** | **MRR** | **NDCG** | **sgd mode** | **num factors** | **user reg** |
| FunkSVD (non normalized) | 0.201 | 0.0988 | 0.0256 | 0.0539 | 0.201 | 0.0421 | adagrad | 30 | 1e-06 |
| FunkSVD (normalized) | 0.206 | 0.098 | 0.024 | 0.054 | 0.206 | 0.042 | adam | 20 | 1e-09 |

Table 8: Results on Testing Set, full dataset

## 7.2 Epinions

In the following table is presented the comparison between the two version of FunkSVD algorithm on the Epinions dataset:

| **Algorithm** | **Results** | | | | | | **Configuration** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AUC** | **Precision** | **Recall** | **MAP** | **MRR** | **NDCG** | **sgd mode** | **num factors** | **user reg** |
| FunkSVD (non normalized) | 0.027 | 0.009 | 0.014 | 0.009 | 0.0245 | 0.0108 | adagrad | 20 | 0.001 |
| FunkSVD (normalized) | 0.0263 | 0.00877 | 0.01255 | 0.0085 | 0.02476 | 0.0251 | adam | 110 | 0 |

Table 9: Results on Testing Set, full dataset

## 7.3 Book Crossing

In the following table is presented the comparison between the two version of FunkSVD algorithm on the Book Crossing dataset, in order to reduce the completion time for our tests we considered only random 50% of the interaction in the dataset:

| **Algorithm** | **Results** | | | | | | **Configuration** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AUC** | **Precision** | **Recall** | **MAP** | **MRR** | **NDCG** | **sgd mode** | **num factors** | **user reg** |
| FunkSVD (non normalized) | 0.01392 | 0.00424 | 0.00916 | 0.00691 | 0.01311 | 0.00338 | adam | 50 | 0 |
| FunkSVD (normalized) | 0.00680 | 0.0022 | 0.00466 | 0.00308 | 0.0063 | 0.00258 | adam | 70 | 0 |

Table 10: Results on Testing Set, 50% interactions dataset

# 8 Conclusion

The most important conclusion we can derive from our work is that the Normalized approach proposed is able to achieve better performance on dense dataset than the standard SLIM approach. As we have seen in section 4.2.3 the normalized approach is able to outperform both the standard SLIM approach and the collaborative KNN recommender. For what concerns Matrix Factorization

Techniques, the normalized approach proposed does not achieve significative performance improvements.

## References

[1] Xia Ning and George Karypis. *SLIM: Sparse Linear Methods for Top-N Recommender Systems.* Data Mining (ICDM), 2011 IEEE 11th International Conference on

[2] Stephen Boyd, Neal Parikh, Eric Chu Borja Peleato and Jonathan Eckstein. *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers.* Foundations and Trends in Machine Learning Vol. 3, No. 1 (2010) 1–122

[3] Funk, Simon. *"FunkSVD proposal",https://sifter.org/simon/journal/20061211.html*

[4] Koren, Yehuda; Bell, Robert; Volinsky, Chris *"Matrix Factorization Techniques for Recommender Systems". Computer. 42 (8): 30–37.*