

LEARNING TO RANK FOR COLLABORATIVE FILTERING

Jean-Francois Pessiot, Tuong-Vinh Truong, Nicolas Usunier, Massih-Reza Amini, Patrick Gallinari

*Department of Computer Science, University of Paris VI
104 Avenue du President Kennedy, 75016 Paris, France
{first_name.last_name}@lip6.fr*

Keywords: Collaborative Filtering, Recommender Systems, Machine Learning, Ranking.

Abstract: Up to now, most contributions to collaborative filtering rely on rating prediction to generate the recommendations. We, instead, try to correctly rank the items according to the users' tastes. First, we define a ranking error function which takes available pairwise preferences between items into account. Then we design an effective algorithm that optimizes this error. Finally we illustrate the proposal on a standard collaborative filtering dataset. We adapted the evaluation protocol proposed by (Marlin, 2004) for rating prediction based systems to our case, where pairwise preferences are predicted instead. The preliminary results are between those of two reference rating prediction based methods. We suggest different directions to further explore our ranking based approach for collaborative filtering.

1 INTRODUCTION

With the emergence of e-commerce, a growing number of commercial websites are using recommender systems to help their customers find products to purchase. The goal of such systems is to generate personalized recommendations for each user, i.e. to filter out a potentially huge set of items, and to extract a subset of N items that best matches his tastes or needs. The most successful approach to date is called collaborative filtering; the main underlying idea is to identify users with similar tastes and use them to generate the recommendations. Collaborative filtering is particularly suited to recommend cultural products like movies, books and music, and is extensively used in many online commercial recommender systems, like Amazon.com or CDNow.com.

A simple way to model a user's preferences is to assign to each item a numerical score which measures how much he likes this item. All items are then ordered according to those scores, from the user's top favorites to the ones he's less interested in. In the standard collaborative filtering framework, those scores are ordinal ratings from 1 to 5. Each user has only rated a few items, leaving the majority of them unrated. Most collaborative filtering methods are based

on a rating prediction approach: taking the available ratings as input, their goal is to predict the missing ratings. The recommendation task simply consists in recommending each user the unrated items with the highest predictions.

Due to its simplicity and the fact that it easily accommodates with objective performance evaluation, the rating prediction approach is the most studied in the collaborative filtering literature. Previous works include classification (Breese et al., 1998), regression (Herlocker et al., 1999), clustering (Chee et al., 2001), dimensionality reduction ((Canny, 2002), (Srebro and Jaakkola, 2003)) and probabilistic methods ((Hofmann, 2004), (Marlin, 2003)). As they all reduce the recommendation task to a rating prediction problem, those methods share a common objective: predicting the missing ratings as accurately as possible. However, from the recommendation perspective, the order over the items is more important than their ratings. Our work is therefore a ranking prediction approach: rather than trying to predict the missing ratings, we predict scores that respect pairwise preferences between items, i.e. preferences expressing that one item is preferred to another. Using those pairwise preferences, our goal is to improve the quality of the recommendation process.

The rest of the paper is organized as follows. Section 2 presents our approach, then details the algorithm and the implementation. Experimental protocol and results are given in section 3. We conclude and give some suggestions to improve our model in section 4.

2 CF AS A RANKING TASK

2.1 Motivation

Most previous works in the collaborative filtering literature have a common approach where they decompose the recommendation task into two steps: rating prediction and recommendation. Of course, once the ratings are predicted, the latter is trivially accomplished by sorting the items according to their predictions and recommending to each user the items with the highest predictions. As they reduce the recommendation task to a rating prediction problem, all these rating prediction approaches have the same objective: predicting the missing ratings as accurately as possible. Such objective seems natural for the recommendation task, and it has been the subject of a great amount of research in the collaborative filtering domain (Marlin, 2004). The rating based formulation is simple and easily accommodates with later performance evaluation. However, it is important to note that rating prediction is only an intermediate step toward recommendation, and that other alternatives may be considered.

In particular, considering the typical use of recommendation where each user is shown the top- N items without their predicted scores (Deshpande and Karypis, 2004), we think that correctly sorting the items is more important than correctly predicting their ratings. Although these two objectives look similar, they are not equivalent from the recommendation perspective. Any method which correctly predicts all the ratings will also correctly sort all the items. However, two methods equally good at predicting the ratings may perform differently at predicting the rankings. Figure 1 shows a simple example: let $[2, 3]$ be the true ratings of items A and B respectively, $r_1 = [2.5, 3.6]$ and $r_2 = [2.5, 2.4]$ be two prediction vectors obtained from two different methods. r_1 and r_2 are equivalent with respect to the squared error¹ (both errors equal $0.5^2 + 0.6^2$), while only r_1 predicts the correct rankings, as it scores B higher than A . A more detailed study of the performance evaluation problem in col-

¹That is the squared difference between a true rating and the prediction

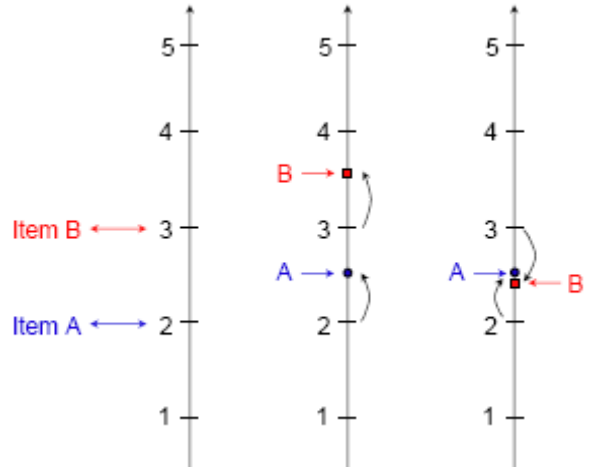


Figure 1: Left scale: true ratings for items A and B . Middle and right scales: prediction vectors r_1 and r_2 . The squared error equals 0.61 for both r_1 and r_2 . The ranking error equals 0 for r_1 and 1 for r_2 .

laborative filtering can be found in (Herlocker et al., 2004).

This work proposes an alternative to the traditional rating prediction approach: our goal is to correctly rank the items rather than to correctly predict their ratings. To achieve this goal, we first define a ranking error that penalizes wrong ranking predictions between two items. Then we propose our model, and design an effective algorithm to minimize the ranking error.

2.2 Formal Definition and Notation

2.2.1 Definition

We assume that there are a set of n users and p items, and that each user has rated at least one item. A CF training instance is a triplet of the form (x, a, r) where x , a and r are respectively a user index in $\mathcal{X} = \{1, \dots, n\}$, an item index in $\mathcal{Y} = \{1, \dots, p\}$ and a rating value in $\mathcal{V} = \{1, \dots, v\}$. For each user's available ratings, we construct the set of pairwise preferences $y_x = \{(a, b) | a \in \mathcal{Y}, b \in \mathcal{Y}, r_x(a) > r_x(b)\}$, where $r_x(a)$ denotes the rating of item a provided by user x . Every pair $(a, b) \in y_x$ is called a *pairwise preference* which means that user x prefers item a to item b .

2.2.2 Utility Functions

Utility functions are the most natural way to represent preferences. An utility function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ depicts the user preferences over a given item as a real-valued variable. Thus, if an user $x \in \mathcal{X}$ prefers

an item a over an item b , this preference can simply be represented by the inequality $f(x, a) > f(x, b)$. To make recommendations for a given user x , it then comes natural to present items in a decreasing order of ranks with respect to the output of $f(x, \cdot)$.

We consider that each user and each item is described by an (unknown) k -length vector, where k is a fixed integer. We also consider that the form of a utility function f is a dot product between the user vector and the item vector $\forall (x, a) \in \mathcal{X} \times \mathcal{Y}, f(x, a) = \langle i_a, u_x \rangle$, where $i_a \in \mathbb{R}^k$ and $u_x \in \mathbb{R}^k$ are respectively the vector descriptions of item a and user x . We define U as the n -by- k matrix where the x -th row contains the user vector u_x , and I as the k -by- p matrix where the a -th column contains the item vector i_a . The product UI denominates a n -by- p utility matrix associated to the utility function f , where $(UI)_{xa} = f(x, a)$.

2.2.3 Cost Function

With respect to the previous notations, a ranking prediction is correct if $f(x, a) > f(x, b)$ when $(a, b) \in y_x$. We can now define the cost function D as the number of pairwise preferences $(a, b) \in y_x$ wrongly predicted by the utility function over all users in the training set:

$$D(U, I) = \sum_x \sum_{(a, b) \in y_x} [[(UI)_{xa} \leq (UI)_{xb}]] \quad (1)$$

where $[[pr]]$ is equal to 1 if the predicate pr holds and 0 otherwise. The objective of learning is to find the best representations for users and items such that the number of pairwise preferences wrongly predicted, D , is the lowest possible. This is an optimisation problem which consists to find I and U minimizing D . As D is not differentiable we optimise its exponential upperbound using the inequality $[[x \leq 0]] \leq e^{-x}$:

$$D(U, I) \leq \underbrace{\sum_x \sum_{(a, b) \in y_x} e^{(UI)_{xb} - (UI)_{xa}}}_{\mathcal{E}(U, I)} \quad (2)$$

The exponential upper-bound is convex separately in U and I , so standard optimisation algorithms can be applied for its minimisation. In order to regularize the factorization, we also add two penalty terms to the exponential objective function \mathcal{E}

$$\mathcal{R}(U, I) = \sum_x \sum_{(a, b) \in y_x} e^{(UI)_{xb} - (UI)_{xa}} + \mu_U \|U\|^2 + \mu_I \|I\|^2 \quad (3)$$

where $\|R\|^2$ denotes the Frobenius norm of a matrix R , and is computed as the sum of its squared elements: $\|R\|^2 = \sum_{ij} R_{ij}^2$. It is a standard penalty function used to regularize matrix factorization problems (Srebro et al., 2004). The regularization is controlled by the positive coefficients μ_U, μ_I and must be carefully

chosen to avoid overfitting the model on the training data. The optimization problem reduces to :

$$(U^*, I^*) = \underset{U, I}{\operatorname{argmin}} \mathcal{R}(U, I)$$

where argmin returns the matrices U^* and I^* that minimize the cost function \mathcal{R} .

2.3 Model

2.3.1 Optimization

The objective function \mathcal{R} is convex over each variable U and I separately, it is however not convex over both variables simultaneously. To minimize \mathcal{R} , we propose a two steps optimization procedure, which consists in alternatively fixing one variable U or I and minimizing \mathcal{R} with respect to the other. Each minimization step is performed using gradient descent, and those steps are repeated until convergence. Algorithm 1 depicts this procedure.

Algorithm 1: A learning Algorithm for CF

Input :

- The set of pairwise preferences $\forall x \in \mathcal{X}, y_x = \{(a, b) | a \in \mathcal{Y}, b \in \mathcal{Y}, r_x(a) > r_x(b)\}$

Initialize:

- Initialize $U^{(1)}$ and $I^{(1)}$ at random
- $t \leftarrow 1$

repeat

- $U^{(t+1)} \leftarrow \underset{U^{(t)}}{\operatorname{argmin}} \mathcal{R}(U^{(t)}, I^{(t)})$
- $I^{(t+1)} \leftarrow \underset{I^{(t)}}{\operatorname{argmin}} \mathcal{R}(U^{(t+1)}, I^{(t)})$
- $t \leftarrow t + 1$

until convergence of $\mathcal{R}(U, I)$;

Output : U and I

As the objective function \mathcal{R} is not convex in both U and I , the proposed algorithm may lead to a local minima of \mathcal{R} . The derivatives of \mathcal{R} computed at each step of the algorithm for the gradient descent are:

$$\begin{aligned} \frac{\partial \mathcal{R}}{\partial I_{jd}} &= \sum_x \left[\sum_a U_{xd} e^{(I^T u_x)_j - (I^T u_x)_a} \delta_{ja}^x \right. \\ &\quad \left. - \sum_b U_{xd} e^{(I^T u_x)_b - (I^T u_x)_j} \delta_{bj}^x \right] + 2\mu_I I_{jd} \\ \frac{\partial \mathcal{R}}{\partial U_x} &= \sum_{(a, b) \in y_x} (i_b - i_a) e^{(I^T u_x)_b - (I^T u_x)_a} + 2\mu_U u_x \end{aligned}$$

where u_x is the x -th row of U , i_a is the a -th column of I , $(I^T u_x)_j$ is the j^{th} component of the matrix product between I and u_x , and $\delta_{ja}^x = 1$ if $(j, a) \in y_x$ and 0 otherwise.

2.3.2 Implementation and Complexity Analysis

The most important challenge of CF recommendation systems is to be able to manage a huge volume of data in real time. For example, the MovieLens dataset², that we considered in our experiments, is constituted of 1 million ratings for 6,040 users and 3,706 movies. As a result CF approaches involve a major constraint of requiring very high computing resources. The real-time computing and recommending constraints force recommendation engines to be scalable in terms of number of users and items. In order to fulfill these constraints, our system learn the parameters U^* and I^* offline and makes recommendations online. In the following, we present the computational complexities of these operations and show that both complexities are linear in number of items, users or rating's values.

Offline Learning Complexity Learning the parameters of the recommendation system goes over the computation of the gradient of \mathcal{R} (equation 3) with respect to U and I (algorithm 1). This computation requires to consider the p^2 pairwise preferences over all items and is often unrealistic in real-life CF applications. However, similarly to (Amini et al., 2005), we can show that the objective function \mathcal{R} can be rewritten as follows :

$$\mathcal{R}(U, I) = \sum_x \sum_{r \in \mathcal{V}} \left(\sum_{a | r_x(a) < r} e^{(UI)_{xa}} \times \sum_{b | r_x(b) = r} e^{-(UI)_{xb}} \right) + \mu_U \|U\|^2 + \mu_I \|I\|^2$$

for which the computation is linear with respect to the number of items p . More precisely, the computational complexity of this function is $O(np(v+k))$ (where n is the number of users, v the number of possible rating values and k the dimension of the space representation). Using a similar decomposition for both gradients, the complexity of each iteration of our algorithm is $O(np(v+k))$. The total complexity is then $O(Tnp(v+k))$, where T is the maximum number of iterations of our algorithm.

Recommendation Complexity To make recommendation for a user, the system computes a corresponding score by multiplying the user matrix and the item matrix. The complexity of this operation

²<http://www.grouplens.org/>

is $O(pk)$, the top h items are then sorted and those with the highest scores are presented as recommended items to the user. The complexity of a recommendation is thus equal to $O(p(k+h \log h))$.

3 EXPERIMENTS

3.1 Experimental Protocol and Error Measure

In order to evaluate our approach, we are going to measure the ability of our method to generalize to unseen pairwise preferences. In these experiments, the available ratings for each user are split into an observed set, and a held out set; each set is then used to generate a pairwise preferences set. The first one is used for training, and the second one for testing the performance of the method. Note that this protocol only measures the ability of a method to generalize to other pairwise preferences provided by the same users who were used for training the method.

Testing is done by first partitioning each user's ratings into a set of observed items, and a set of held out items; each set is then used to generate a pairwise preferences set: a training one and a test one. One way to choose the set of held out items is to randomly pick K items among the user's ratings. Since CF datasets are already sparse, for each user we only pick 2 items for testing and leave the rest for training; we call this protocol *all-but-2*.

In order to compare the ranking prediction accuracies of the different methods, we define the *mean ranking error* (MRE), which counts the mean number of prediction errors over the test pairwise preferences. Assuming n users and 2 test items per user as in the all-but-2 protocol:

$$MRE = \frac{1}{n} \sum_x [|(I^T u_x)_{a_x} \leq (I^T u_x)_{b_x}|]$$

where (a_x, b_x) is the test pairwise preference of user i .

3.2 Dimensionality Reduction for Rating Prediction

We compare our approach with two dimensionality reduction methods used for rating prediction: weighted Singular Value Decomposition and Generalized Non-Negative Matrix Factorization. In this subsection, we briefly describe them and explain how they are applied to the rating prediction task.

Definitions In the following, R is a n -by- p matrix, k is a positive integer with $k < np$, and W is a n -by- p matrix of positive weights. The Frobenius norm of R is defined as: $\|R\|^2 = \sum_{ij} R_{ij}^2$. In its simplest form, the goal of matrix factorization is to find the best k -rank approximation of R in respect to the Frobenius norm, i.e. to find the k -rank matrix \hat{R} minimizing $\|R - \hat{R}\|^2$. As most standard matrix factorization methods are unable to handle missing elements in R , recent approaches propose to optimize the weighted Frobenius norm instead of the standard Frobenius norm, i.e.: $\|W \odot (R - \hat{R})\|^2$, where \odot is the elementwise Schur product. Missing elements R_{ij} are simply handled by setting corresponding W_{ij} to 0.

Singular Value Decomposition SVD is a standard method for dimensionality reduction; it is used to decompose R into a product ASV^T where A , S , V are n -by- p , p -by- p , p -by- p matrices respectively. In addition, A and V are orthogonal, and S is a diagonal matrix where S_{ii} is the i^{th} largest eigenvalue of DD^T ; the columns of A and V are the eigenvectors DD^T and $D^T D$ respectively, and are ordered according to the values of the corresponding eigenvalues. The main property is that the k -rank approximation of R obtained with SVD is optimal in respect to the Frobenius norm. While SVD cannot be used when some entries of the target matrix R are missing, the weighted SVD (wSVD) approach proposed by (Srebro and Jaakkola, 2003) can handle such missing data by optimizing the weighted Frobenius norm. Its simplest implementation consists of an EM-like algorithm, where SVD is iteratively applied to an updated low-rank approximation of R . Although very simple to implement, this method suffers from a high algorithmic complexity ($O(lnp^2 + lp^3)$ where l is the number of iterations), making it difficult to use on real CF datasets.

Non-negative Matrix Factorization NMF is a matrix factorization method proposed by (Lee and Seung, 1999). Given a non-negative n -by- p matrix R (i.e. all the elements of R are non-negative real numbers), NMF computes a k -rank decomposition of R under non-negativity constraints. The motivation of NMF lies in those constraints, as their authors argue that they allow the decomposition of an object as the sum of its parts. Formally, we seek a product of two non-negative matrices U , I of sizes n -by- k and k -by- p respectively, optimal in respect to the Frobenius norm. The corresponding optimization problem is not convex, thus only local minima

are achievable; those can be found using Lee’s multiplicative updates. Although NMF was not designed to work with missing data, (Dhillon and Sra, 2006) recently proposed the Generalized Non-negative Matrix Factorization (GNMF) which optimizes the weighted Frobenius norm.

Application to CF . Previous matrix factorization methods are applied in a similar way to the CF task: given n users and p items, we consider the n -by- p target matrix R and the n -by- p weights matrix W . If the user x provided the rating r for the item a , then $R_{xa} = r$ and $W_{xa} = 1$; if the rating is unknown, then $W_{ij} = 0$. The matrix factorization of R is driven by the optimization of the weighted Frobenius norm; for both weighted SVD and GNMF, unknown ratings are randomly initialized. The ratings predictions for unrated items are given by the k -rank matrix resulting from the matrix factorization. The real predicted ratings induce a total order over the unrated items; in the following, we will compare our ranking predictions to the ones obtained by weighted SVD and GNMF.

3.3 Dataset

We used a public movie rating dataset called MovieLens; it contains 1,000,209 ratings collected from 6,040 users over 3,706 movies. Ratings are on a scale from 1 to 5. The dataset is 95.5% sparse. For each user we held out 2 test items using the all-but-2 protocol, leaving 988,129 ratings for training and 12,080 for testing. This was done 5 times, generating a total of 10 bases for the evaluation. All the results presented below are averaged over the generated bases.

3.4 Results

We compared our approach to GNMF and wSVD for several values of the matrix rank k . We stopped our algorithm after 50 iterations of our two steps procedure; GNMF was stopped after 1000 iterations. We also simplified the regularization problem by fixing $\mu_A = \mu_X$ for both GNMF and our approach; several regularization values were tried, and the presented MRE results correspond to the best ones. GNMF was used with $\mu_U = \mu_I = 1$, and our ranking approach with $\mu_U = \mu_I = 100$. The main results are:

	GNMF	wSVD	Ranking
k	9	8	8
MRE	0.2658	0.2770	0.2737

Discussion The optimal values for k are almost identical for the three approaches; this is not surprising for GNMF and wSVD, as they are very similar methods (their only difference lies in the additional non-negativity constraints for GNMF). But this is interesting for our ranking approach, and it seems that explaining the users pairwise preferences is as difficult as explaining their ratings, as they require the same number of hidden factors.

Although not equivalent, the ranking error used for evaluation is closely related to the ranking error optimized by our approach, while GNMF and wSVD optimize a squared error measuring how well they predict the ratings. This is why these primary results are a bit disappointing, as we would have logically expected our approach to have the best ranking error. The good performance of GNMF is not surprising considering that it already performed well (at least better than wSVD) with respect to rating prediction (Pessiot et al., 2006). Concerning wSVD, its ranking error could be improved by increasing the number of iterations, but the high algorithmic complexity makes it difficult to use on real datasets such as MovieLens, especially when the number of items is high. In our experiments, we had to stop it after only 20 iterations due to its extreme slowness. Besides, this wSVD is also limited by its lack of regularization, which is usually used to avoid the overfitting problem.

Further directions need to be explored to complete and improve those primary results. The first direction concerns user level normalization: when we minimize the sum of errors (the sum of squared errors for each rating in GNMF and wSVD, the sum of ranking errors for each pairwise preference in our approach), users who have rated lots of items tend to be associated with higher errors; thus the learning phase focuses on those users, while ignoring the others. This problem can be avoided if we give each user the same importance by considering normalized errors, i.e. by dividing each user’s error by the number of his pairwise preferences. The mean ranking error we define for evaluation is in fact a normalized error, as we only consider one test pairwise preference for each user. This is why we expect that learning with normalized errors will give better experimental results.

A second direction we want to explore is a more careful study of stopping criteria. We stopped GNMF and our ranking approach after fixed numbers of iterations, which seemed to correspond to empirical convergence. In future experiments, we will rather stop them when the training errors stop decreasing, which will allow us a more thorough comparison of the three methods with respect to the training time.

Another question we need to study concerns the

regularization. It is an important feature of a learning algorithm as it is used to prevent overfitting the training data, thus avoiding bad predictions on unseen data. In both GNMF and our ranking approach, μ_U and μ_I are the regularization terms. Setting $\mu_U = \mu_I = 0$ means no regularization; and the higher they are, the more matrix norms are penalized. In our experiments we fixed $\mu_U = \mu_I$ for simplicity. By doing this, we implicitly gave equal importance for each variable of our model. In future works, we will study the exact influence of those regularization terms, and how they should be fixed.

Detailed Results MRE results for several values of the rank k :

k	7	8	9	10	11
GNMF	0.2696	0.2688	0.2658	0.2679	0.2684

k	5	6	7	8	9
wSVD	0.2847	0.2862	0.2803	0.2770	0.2786

k	6	7	8	9	10
Ranking	0.2752	0.2744	0.2737	0.2743	0.2753

4 CONCLUSION AND PERSPECTIVES

The rating prediction approach is still actively used and studied in collaborative filtering problems. Proposed solutions come from various machine learning fields such as classification, regression, clustering, dimensionality reduction or density estimation. Their common approach is to decompose the recommendation process into a rating prediction step, and the recommendation step. But from the recommendation perspective, we think other alternatives than rating prediction should be considered. In this paper, we proposed a new ranking approach for collaborative filtering: instead of predicting the ratings as most methods do, we predict scores that respect pairwise preferences between items, as we think correctly sorting the items is more important than correctly predicting their ratings. We proposed a new algorithm for ranking prediction, defined a new evaluation protocol and compared our approach to two rating prediction approaches. While the primary results are not as good as we expected with respect to the mean ranking error, we are confident they can be explained and improved by studying user level normalization, convergence criteria and regularization. We are planning to explore the relations between collaborative filtering and other tasks such as text analysis (e.g. text segmentation,

(Caillet et al., 2004)) and multitask learning (Ando and Zhang, 2005), in order to extend our work to other frameworks such as semi-supervised learning (Amini and Gallinari, 2003).

ACKNOWLEDGEMENTS

The authors would like to thank Trang Vu for her helpful comments. This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors view.

REFERENCES

- Amini, M.-R. and Gallinari, P. (2003). Semi-supervised learning with explicit misclassification modeling. In Gottlob, G. and Walsh, T., editors, *IJCAI*, pages 555–560. Morgan Kaufmann.
- Amini, M.-R., Usunier, N., and Gallinari, P. (2005). Automatic text summarization based on word-clusters and ranking algorithms. In *Proceedings of the 27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23*, Lecture Notes in Computer Science, pages 142–156. Springer.
- Ando and Zhang (2005). A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*.
- Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*.
- Caillet, M., Pessiot, J.-F., Amini, M.-R., and Gallinari, P. (2004). Unsupervised learning with term clustering for thematic segmentation of texts. In *Proceedings of the 7th Recherche d'Information Assistée par Ordinateur, Avignon, France*, pages 648–656. CID.
- Canny, J. (2002). Collaborative filtering with privacy via factor analysis. *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*.
- Chee, S., Han, J., and Wang, K. (2001). Rectree: An efficient collaborative filtering method. In *Data Warehousing and Knowledge Discovery*.
- Deshpande, M. and Karypis, G. (2004). Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*.
- Dhillon, I. S. and Sra, S. (2006). Generalized nonnegative matrix approximations with bregman divergences. *NIPS*.
- Herlocker, J., Konstan, J., and Riedl, J. (1999). An algorithmic framework for performing collaborative filtering.
- Herlocker, J., Konstan, J., Terveen, L., and Riedl, J. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*.
- Hofmann, T. (2004). Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115.
- Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*.
- Marlin, B. (2003). Modeling user rating profiles for collaborative filtering. *Advances in Neural Information Processing Systems*.
- Marlin, B. (2004). Collaborative filtering: A machine learning perspective.
- Pessiot, J.-F., Truong, V., Usunier, N., Amini, M., and Gallinari, P. (2006). Factorisation en matrices non-negatives pour le filtrage collaboratif. In *3eme Conference en Recherche d'Information et Applications (CORIA'06)*, pages 315–326, Lyon.
- Srebro, N. and Jaakkola, T. (2003). Weighted low rank approximation. In *ICML '03. Proceedings of the 20th international conference on machine learning*.
- Srebro, N., Rennie, J. D. M., and Jaakkola, T. S. (2004). Maximum-margin matrix factorization. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*. MIT Press, Cambridge, MA.