machine learning & knowledge extraction

MDPI

Article

A Matrix Factorization Algorithm for Efficient Recommendations in Social Rating Networks Using Constrained Optimization

Nicholas Ampazis *, Theodoros Emmanouilidis and Flora Sakketou

Department of Financial and Management Engineering, University of the Aegean, Kountouriotou 41, 82100 Chios, Greece

* Correspondence: n.ampazis@fme.aegean.gr

Received: 4 June 2019; Accepted: 9 August 2019; Published: 11 August 2019

Abstract: In recent years the emergence of social media has become more prominent than ever. Social networking has become the de facto tool used by people all around the world for information discovery. Consequently, the importance of recommendations in a social network setting has urgently emerged, but unfortunately, many methods that have been proposed in order to provide recommendations in social networks cannot produce scalable solutions, and in many cases are complex and difficult to replicate unless the source code of their implementation has been made publicly available. However, as the user base of social networks continues to grow, the demand for developing more efficient social network-based recommendation approaches will continue to grow as well. In this paper, following proven optimization techniques from the domain of machine learning with constrained optimization, and modifying them accordingly in order to take into account the social network information, we propose a matrix factorization algorithm that improves on previously proposed related approaches in terms of convergence speed, recommendation accuracy and performance on cold start users. The proposed algorithm can be implemented easily, and thus used more frequently in social recommendation setups. Our claims are validated by experiments on two real life data sets, the public domain Epinions.com dataset and a much larger dataset crawled from Flixster.com.

Keywords: collaborative filtering; recommender systems; social networks; matrix factorization; constrained optimization

1. Introduction

Matrix factorization in collaborative filtering recommender systems is usually performed by unconstrained gradient descent for learning the feature components of the user and item factor matrices [1]. This is essentially a "black box" approach, where apart from the minimization of an objective function (usually the Root Mean Squared Error (RMSE) over the known ratings), generally no other information or knowledge is taken into account during the factorization process. However this approach alone cannot solve efficiently a majority of recommendation problems. The most notable example is the Netflix Prize problem, which dealt with a very sparse, high-dimensional dataset with more than 100 million training patterns. The winning solutions of the "Bellkor's Pragmatic Chaos" team [2–4], made apparent that a huge number of latent factor models, clustering, and K-nearest neighbors approaches was necessary to be combined (both linearly and non-linearly) in order to provide an accurate final solution.

In earlier work where our focus was on training feedforward neural networks [5] we showed that, during training, it is useful to incorporate additional "incremental conditions", i.e., conditions involving

quantities that must be optimized incrementally at each epoch of the learning process (by the term "epoch" we denote a training cycle of presenting the entire training set). We thus formulated a general problem, whereby we sought the minimization of the objective function representing the distance of the network's outputs from preset target values, subject to other constraints that represented the additional knowledge. We then demonstrated how to formulate the general problem of incorporating the additional knowledge as a constrained optimization task, whose solution lead to a powerful generic learning algorithm accounting for both target and incremental conditions. Advancing that work, in a recent study [6], and similarly to other proposed methods [7], we cast the factorization of the user-by-item ratings matrix as a feedforward neural network training problem, and thus concentrated on the development of constrained optimization methods that could lead to efficient factorization algorithms. In that study, we introduced a general constrained matrix factorization framework that we refer to as FALCON (Factorization Algorithms for Learning with Constrained OptimizatioN), and presented two examples of algorithms which can be derived from that framework, that incorporate additional knowledge about learning the factor matrices. The first example (FALCON-M), incorporated an extra condition that seeks to facilitate factor updates in long narrow valleys of the objective function landscape, thus avoiding getting trapped in suboptimal solutions. The second example (FALCON-R), considered the importance of regularization on the success of the factorization models and adapted the regularization parameter automatically while training.

In a social rating network, recommendations for a user can be produced on the basis of the ratings of the users that have direct or indirect social relations with the given user. This approach is supported by sociological models [8], and their verification due to the increasing availability of online social network data [9]. The models propose that people tend to relate to other people with similar attributes, and due to the effects of social influence, related people in a social network, in turn, influence each other to become even more similar. Thorough surveys summarizing the various approaches proposed for social recommender systems in general, can be found in [10,11].

In this paper, we exploit this information within the FALCON framework and propose a matrix factorization algorithm for recommendation in social rating networks, called SocialFALCON. Here the additional information is incorporated as an extra condition that, during training, seeks to render the latent feature vector of each user as close as possible to the weighted average of the latent feature vectors of his direct neighbors, without compromising the need for decreasing the objective function. By achieving maximum possible alignment between the successive feature vectors updates of each user to that of his direct neighbors, the latent features of users indirectly connected in the social network also become dependent and hence social network influence gets propagated during training. An important benefit from this approach also arises for cold start users, that is, for users who have expressed only a few ratings. These users are more dependent on the propagation of social infuence compared to users with more ratings, and thus the SocialFALCON model, through the enforcement of the constraint that their latent features is close to those of their neighbors, can learn user feature vectors for this challenging category of users as well.

To evaluate the effectiveness of our approach in terms of prediction accuracy and computational efficiency, we present experimental results to compare its performance with the following baseline and state of the art algorithms for matrix factorization in recommender systems:

- Regularized SVD (RegSVD): This method is the baseline matrix factorization approach which does not take into account the social network [12]. We compare our approach against this method in order to evaluate the improvement in performance induced by the social network information.
- Probabilistic Matrix Factorization (PMF): This is an extension of the baseline matrix factorization which bounds the range of predictions by introducing non-linearities in the prediction rule [13]. Similarly to RegSVD we compare our approach against this method in order to evaluate the importance of utilizing social network information.
- SVD++: This is also a matrix factorization approach which also does not take into account any social network information [14]. However SVD++ takes into account additional information

in the factorization model in the form of implicit feedback for each user's item preference history. We compare our approach against this method since it achieves state of the art performance in recommender system tasks against which other algotihms should be compared, and because it provides a paradigm for extending the baseline matrix factorization approach with additional information.

- SocialMF: This is a very closely related algorithm to our proposed approach but utilizes the social network information in a different way in the factorization model [15].

Detailed derivations and descriptions of the above algorithms are presented in the next sections. To elucidate on the transparency of the results, we have implemented each algorithm from scratch in low-level C with no dependencies on additional libraries, and we've made the source codes publicly available with documentation and additional supporting materials from Github as described in Section 8.

Due to the lack of publicly available social rating network datasets, the performance of all algorithms was evaluated on the two standard social rating network datasets utilized and introduced respectively in [15], namely the existing Epinions dataset and a large scale dataset that was gained from Flixster.com. Again for transparency reasons in the reported results, we've made publicly available all the training/validation cross-validation splits that we utilized in our experiments as described in Section 7.

The rest of this paper is organized as follows: The problem definition of matrix factorization in recommender systems is presented in Section 2. SVD++, which we consider in our benchmarks as a state of the art extension of the basic matrix factorization model, and which utilizes implicit rating information, is discussed in Section 3. Section 4 discusses the recommendation problem in a social rating network setting and describes SocialMF which is the most closely related method to our approach. The general constrained optimization FALCON framework is outlined in Section 5. In Section 6 we derive the SocialFALCON algorithm from the FALCON framework. In the same section we also discuss some desirable properties of the algorithm, and its advantages over the SocialMF approach, especially with regards to computational complexity. The real life data sets used in our experiments are described in Section 7, and the experiments are reported in Section 8. Finally, in Section 9 conclusions are drawn and we outline some interesting directions for future work.

2. Matrix Factorization for Recommender Systems

In its basic form, matrix factorization for recommender systems describes both users and items by vectors of (latent) factors which are inferred from the user-by-item rating matrix. High correlation between user and item factors leads to a recommendation of an item to a particular user [16].

The idea behind matrix factorization is very simple. We define the corresponding matrix factorization $\hat{\boldsymbol{R}}_K$ of the user-by-item ratings matrix $\boldsymbol{R} \in \mathbb{R}^{N \times M}$ as

$$\hat{\boldsymbol{R}}_K = \boldsymbol{U}^T \boldsymbol{V} \tag{1}$$

where $\boldsymbol{U} \in \mathbb{R}^{K \times N}$ and $\boldsymbol{V} \in \mathbb{R}^{K \times M}$, in order to approximate all the unknown elements of $\boldsymbol{R}$.

For the recommendation problem, $\boldsymbol{R}$ has many unknown elements, which cannot be treated as zero, and the application of formal Singular Value Decomposition (SVD), e.g., by Lanczos' method [17] would provide erroneous results. Thus, for this case, the approximation task can be defined as follows:

Let $\hat{r}_{ui}$ denote how the u-th user would rate the i-th item, according to the factorization model, and e_{ui} denote the error on the (u, i)-th known rating example (training pattern):

$$\hat{r}_{ui} = \sum_{k=1}^{K} u_{uk} v_{ki} \tag{2}$$

and

$$e_{ui} = r_{ui} - \hat{r}_{ui}. \tag{3}$$

In order to minimize this error (and consequently the error over all training patterns) we can apply the stochastic version of the gradient descent method on each $(1/2) \cdot e_{ui}^2$ to find a local minimum. Hence the elements of $\boldsymbol{U}$ and $\boldsymbol{V}$ can be updated as follows:

$$u_{uk}^{'} = u_{uk} + \eta \cdot e_{ui} \cdot v_{ki}, \;\; v_{ki}^{'} = v_{ki} + \eta \cdot e_{ui} \cdot u_{uk} \tag{4}$$

where η is the learning rate.

To better generalize on unseen examples, we can apply regularization with factors λ_u and λ_v for the user and item factors respectively, in order to prevent large weights:

$$u_{uk}^{'} = u_{uk} + \eta \cdot (e_{ui} \cdot v_{ki} - \lambda_u \cdot u_{uk}), \;\; v_{ki}^{'} = v_{ki} + \eta \cdot (e_{ui} \cdot u_{uk} - \lambda_v \cdot v_{ki}). \tag{5}$$

Thus, Equation (5) are the user and item factors updates when minimizing the following objective function by performing unconstrained stochastic gradient descent on $\mathbf{U}_u$ and $\mathbf{V}_i$ for all users u and all items i:

$$\mathcal{L} = \frac{1}{2} \sum_{u=1}^{N} \sum_{i=1}^{M} I_{ui} (r_{ui} - \mathbf{U}_u^T \mathbf{V}_i)^2 + \frac{\lambda_u}{2} \sum_{u=1}^{N} ||\mathbf{U}_u||^2 + \frac{\lambda_v}{2} \sum_{i=1}^{M} ||\mathbf{V}_i||^2. \tag{6}$$

In the above equation I_{ui} is an indicator function that is equal to 1 if user u rated item i and equal to 0 otherwise, and "$||.||$" denotes the Euclidean norm. In this linear model the predictions are usually clipped to the $[min_r, max_r]$ range appearing in the ratings matrix $\boldsymbol{R}$.

It is also customary to adjust the predicted rating by accounting for user and item biases, that is, for systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others. We can encapsulate these effects using *baseline estimates* [14] as follows:

Denote by μ the overall average rating. A baseline estimate for an unknown rating $\hat{r}_{ui}$ is denoted by b_{ui} and accounts for the user and item effects:

$$b_{ui} = \mu + \beta_u + \gamma_i. \tag{7}$$

The vectors $\boldsymbol{\beta} \in \mathbb{R}^N$ and $\boldsymbol{\gamma} \in \mathbb{R}^M$ contain the biases for all users and items respectively, and the elements β_u and γ_i indicate the biases of user u and item i respectively. These can be also learned by gradient descent if we plugin into Equation (6) the baseline corrected expression for $\hat{r}_{ui}$ as:

$$\hat{r}_{ui} = b_{ui} + \mathbf{U}_u^T \mathbf{V}_i = \mu + \beta_u + \gamma_i + \mathbf{U}_u^T \mathbf{V}_i. \tag{8}$$

In this case the updates for β_u and γ_i are given by:

$$\beta_u^{'} = \beta_u + \eta \cdot (e_{ui} - \lambda_\beta \cdot \beta_u), \;\; \gamma_i^{'} = \gamma_i + \eta \cdot (e_{ui} - \lambda_\gamma \cdot \gamma_i) \tag{9}$$

where λ_β and λ_γ are regularization parameters for the user and item biases respectively. This approach is usually refered to in the literature as Regularized SVD (RegSVD) [12].

Instead of using a simple linear model, which can make predictions outside of the range of valid rating values, the dot product between user and item specific feature vectors can be passed through the logistic function (sigmoid) $s(z) = 1/(1 + exp(-z))$, which bounds the range of predictions within $[0, 1]$. In this case the objective function is modified as follows:

$$\mathcal{L} = \frac{1}{2}\sum_{u=1}^{N}\sum_{i=1}^{M} I_{ui}(r_{ui} - g(\theta_{ui}))^2 + \frac{\lambda_u}{2}\sum_{u=1}^{N}||\mathbf{U}_u||^2 + \frac{\lambda_v}{2}\sum_{i=1}^{M}||\mathbf{V}_i||^2 + \frac{\lambda_\beta}{2}||\boldsymbol{\beta}||^2 + \frac{\lambda_\gamma}{2}||\boldsymbol{\gamma}||^2 \quad (10)$$

where

$$\theta_{ui} = \beta_u + \gamma_i + \mathbf{V}_i^T\mathbf{U}_u \quad (11)$$

and $g(z) = (max_r - min_r) * s(z) + min_r$. This model is usually refered to as Probabilistic Matrix Factorization (PMF) [13].

Usually the learning rate and the regularization values for each latent factor are determined by a search for optimal values that optimize the performance on a withheld validation set. There are various approaches on how this search is performed, but the most common approach is by grid search [18], where a predefined grid of candidates is set up and on each candidate a model is trained and evaluated on the validation set. It is obvious that such a scheme is very expensive in terms of runtime as many different models have to be trained and then evaluated, neverthess it is the dominating method for parameter selection in matrix factorization models.

3. SVD++

SVD++ [14] is a state of the art extension to the RegSVD model by taking into account implicit information. The most natural choice for implicit feedback is the item preference history, which tells us about the items for which the user has expressed a preference (either explicit or implicit). Thus, the idea is to add a second set of item factors, relating each item i to a factor vector $\mathbf{Y}_i \in \mathbb{R}^K$, and model the factor vector of each user as a function of all factor vectors $\mathbf{Y}_i$, for which the user has expressed a preference.

Following [12,14], SVD++ gives the predicted rating of user u on item i as follows:

$$\hat{r}_{ui} = b_{ui} + \mathbf{V}_i^T(\mathbf{U}_u + |R_u|^{-\frac{1}{2}}\sum_{j\in R_u}\mathbf{Y}_j) \quad (12)$$

where b_{ui} is given by Equation (7) and the set R_u contains the items rated by user u (for which we may or may not know the exact explicit rating). For example, in the Netflix challenge there was extra information in the qualifying set that users had rated certain movies but the actual value of each rating was withheld in order to be predicted by the contestants.

Therefore, in order to predict an unknown rating $\hat{r}_{ui}$, as in (8), SVD++ maintains a vector of factors $\mathbf{U}_u$ for each user but this vector is also complemented by the sum $|R_u|^{-\frac{1}{2}}\sum_{j\in R_u}\mathbf{Y}_j$ that represents the combination of explicit and implicit feedback.

Plugging in Equation (12) into (6) and taking into account regularization parameters λ_β and λ_γ for the user and item biases as in Equation (9), we can learn $\mathbf{U}_u$, $\mathbf{V}_i$, and $\mathbf{Y}_i$ by gradient descent updates as follows:

$$\mathbf{U}_u^{'} = \mathbf{U}_u + \eta\cdot(e_{ui}\cdot\mathbf{V}_i - \lambda_u\mathbf{U}_u), \quad (13)$$

$$\mathbf{V}_i^{'} = \mathbf{V}_i + \eta\cdot(e_{ui}\cdot(\mathbf{U}_u + |R_u|^{-\frac{1}{2}}\sum_{j\in R_u}\mathbf{Y}_j) - \lambda_v\mathbf{V}_i), \quad (14)$$

$$\mathbf{Y}_j^{'} = \mathbf{Y}_j + \eta\cdot(e_{ui}\cdot|R_u|^{-\frac{1}{2}}\mathbf{V}_i - \lambda_y\mathbf{Y}_j), \quad \forall j \in R_u, \quad (15)$$

where λ_y is the regularization parameter for the items for which the user has expressed a preference.

4. SocialMF

In a social rating network, each user u also has a set N_u of other users (direct neighbors) with which she has a connection. We denote by T_{uv} the value of social "trust" [15] that user u has on user v as a real number in [0, 1]. A value of 0 means that there is no trust and a value of 1 means full trust from u to v. In practice, in most social networks the trust values are binary with $T_{uv} = 0$ meaning that user u does not "follow" user v, and $T_{uv} = 1$ meaning that u "follows" v. The trust values can be stored in a matrix $T = [T_{uv}]_{NxN}$, where usually each row is normalized so that $\sum_{u=1}^{N} T_{uv} = 1$. Note that the matrix T is asymmetric in general.

SocialMF [15] is a probabilistic model-based solution which incorporates the propagation of trust in the model, and which has been shown to improve both in terms of the quality of recommendations and speed over previously proposed approaches such as the Social Trust Ensemble (STE) proposed in [19]. The main idea behind SocialMF is that the latent feature vectors of a user should be made dependent on the feature vectors of his direct neighbors in the social network. Using this idea, latent features of users indirectly connected in the social network will be also dependent and hence the trust gets propagated. Thus, latent factors for all users and items are learned jointly from the ratings and the social graph. SocialMF therefore models the users' feature vectors as:

$$\hat{\mathbf{U}}_u = \sum_{v \in N_u} T_{uv} \mathbf{U}_v \tag{16}$$

where $\hat{\mathbf{U}}_u$ is the estimated latent feature vector of u given the feature vectors of his direct neighbors.

To this end, SocialMF extends the basic PMF model by considering the following objective function:

$$\begin{aligned} \mathcal{L}(R, T, U, V) = \frac{1}{2} \sum_{u=1}^{N} \sum_{i=1}^{M} I_{ui} (r_{ui} - g(\mathbf{U}_u^T \mathbf{V}_i))^2 + \frac{\lambda_u}{2} \sum_{u=1}^{N} ||\mathbf{U}_u||^2 + \frac{\lambda_v}{2} \sum_{i=1}^{M} ||\mathbf{V}_i||^2 \\ + \frac{\lambda_t}{2} \sum_{u=1}^{N} ||(\mathbf{U}_u - \sum_{v \in N_u} T_{uv} \mathbf{U}_v)||^2, \end{aligned} \tag{17}$$

where $g(\mathbf{U}_u^T \mathbf{V}_i)$ is the predicted rating $\hat{r}_{ui}$ of user u on item i. As in PMF, λ_u, λ_v and λ_t are regularization parameters and I_{ui} is an indicator function that is equal to 1 if user u rated item i and equal to 0 otherwise.

We can find a local minimum of the objective function in Equation (17) by performing gradient descent on $\mathbf{U}_u$ and $\mathbf{V}_i$ for all users u and all items i:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{U}_u} = \sum_{i=1}^{M} I_{ui} \mathbf{V}_i g^{'}(\mathbf{U}_u^T \mathbf{V}_i)(g(\mathbf{U}_u^T \mathbf{V}_i) - r_{ui}) + \lambda_u \mathbf{U}_u \\ + \lambda_t \left((U_u - \sum_{v \in N_u} T_{uv} U_v) - \sum_{v | u \in N_v} T_{vu} (U_v - \sum_{w \in N_v} T_{vw} U_w) \right), \end{aligned} \tag{18}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{V}_i} = \sum_{u=1}^{N} I_{ui} \mathbf{U}_u g^{'}(\mathbf{U}_u^T \mathbf{V}_i)(g(\mathbf{U}_u^T \mathbf{V}_i) - r_{ui}) + \lambda_v \mathbf{V}_i. \tag{19}$$

Note that the derivative of the objective function with respect to the item feature vectors $\mathbf{V}_i$ is simply the same to that of the PMF model.

A variation of SocialMF has been proposed in [20] in which the authors train separate matrix factorization models for each category c that the items belong to, and for which a user u has issued a trust statement towards v, given that u and v simultaneously have ratings in a category c.

5. Overview of the Falcon Framework

Based on the discussion in the previous sections, it is clear that in all approaches matrix factorization in recommender systems involves minimization by unconstrained gradient descent of a suitably chosen objective function with respect to the user and item features and biases. In our approach we suppose that there are additional relations to be satisfied, that represent additional knowledge we wish to incorporate into the learning mechanism, and involve all the available features $\mathbf{U}_u$, $\mathbf{V}_i$, β_u and γ_i (u = 1 $\cdots$ N and i = 1 $\cdots$ M).

For the rest of our discussion it is convenient to group all the features into a single column vector $\boldsymbol{w}$ (which we will refer to as the "weight" vector) as follows:

$$\boldsymbol{w} = (\beta_1 \cdots \beta_N, \mathbf{U}_1^T \cdots \mathbf{U}_N^T, \gamma_1 \cdots \gamma_M, \mathbf{V}_1^T \cdots \mathbf{V}_M^T). \tag{20}$$

Before introducing the form of the additional relations, we note that in this work we'll concentrate on the minimization of the Mean Squared Error (MSE) objective function given by:

$$\mathcal{L} = \frac{1}{2} \sum_{u=1}^{N} \sum_{i=1}^{M} I_{ui}(r_{ui} - g(\theta_{ui}))^2, \tag{21}$$

with the quantities involved explained in Section 2. We will also adopt an epoch-by-epoch (i.e., *batch*) optimization framework with the following objectives:

1. At each epoch of the learning process, the vector $\boldsymbol{w}$ will be incremented by $d\boldsymbol{w}$, so that the search for an optimum new point in the space of $\boldsymbol{w}$ is restricted to a hypersphere of known radius δP centered at the point defined by the current $\boldsymbol{w}$

$$d\boldsymbol{w}^T d\boldsymbol{w} = (\delta P)^2. \tag{22}$$

2. At each epoch, the objective function $\mathcal{L}$ must be decremented by a quantity δQ, so that, at the end of learning, $\mathcal{L}$ is rendered as small as possible. To first order, we can substitute the change in $\mathcal{L}$ by its first differential and demand that

$$d\mathcal{L} = \delta Q. \tag{23}$$

Within the FALCON framework originally introduced in [6], we can define a certain quantity Φ that we also wish to incrementally maximize at each epoch subject to the objectives defined above. Consequently the learning rule can be derived by solving the following constrained optimization problem:

$$\begin{aligned} &\text{Maximize } \Phi \ (\Phi = max) \ \text{ w.r.t } d\boldsymbol{w} \\ \text{subject to } \quad &d\boldsymbol{w}^T d\boldsymbol{w} = (\delta P)^2 \\ &d\mathcal{L} = \delta Q. \end{aligned}$$

This constrained optimization problem can be solved analytically by a method similar to the constrained gradient ascent technique introduced in optimal control in [21], and leads to a generic update rule for $\boldsymbol{w}$ as follows:

First, we introduce suitable Lagrange multipliers λ_1 and λ_2 to take into account Equations (22) and (23) respectively. If δP is small enough, the changes to Φ induced by changes in $\boldsymbol{w}$ can be approximated by the first differential $d\Phi$. Thus, secondly, we introduce the function ϕ, whose differential is defined as

$$d\phi = d\Phi + \lambda_1(d\mathcal{L} - \delta Q) + \lambda_2[(\delta P)^2 - ||d\boldsymbol{w}||^2]. \tag{24}$$

On evaluating the differentials involved in the right hand side, we readily obtain

$$d\phi = \boldsymbol{F} \cdot d\boldsymbol{w} + \lambda_1(\boldsymbol{G} \cdot d\boldsymbol{w}) + \lambda_2[(\delta P)^2 - ||d\boldsymbol{w}||^2], \tag{25}$$

where $\boldsymbol{G}$ and $\boldsymbol{F}$ are given by

$$\boldsymbol{G} = \partial\mathcal{L}/\partial\boldsymbol{w}\ , \quad \boldsymbol{F} = \partial\Phi/\partial\boldsymbol{w}. \tag{26}$$

To maximize $d\phi$ at each epoch, we demand that

$$d^2\phi = (\boldsymbol{F} - \lambda_1\boldsymbol{G} - 2\lambda_2 d\boldsymbol{w}) \cdot d^2\boldsymbol{w} = 0 \tag{27}$$

and

$$d^3\phi = -2\lambda_2||d^2\boldsymbol{w}||^2 < 0. \tag{28}$$

Hence, the factor multiplying $d^2\boldsymbol{w}$ in Equation (27) should vanish, and therefore we obtain

$$d\boldsymbol{w} = -\frac{\lambda_1}{2\lambda_2}\boldsymbol{G} + \frac{1}{2\lambda_2}\boldsymbol{F}. \tag{29}$$

Equation (29) constitutes the weight update rule, provided that λ_1 and λ_2 can be evaluated in terms of known quantities. This can be done as follows:

From Equations (22), (26) and (27) we obtain

$$\lambda_1 = \frac{I_{GF} - 2\lambda_2\delta Q}{I_{GG}} \tag{30}$$

with I_{GG} and I_{GF} given by

$$I_{GG} = ||\boldsymbol{G}||^2, \quad I_{GF} = \boldsymbol{G} \cdot \boldsymbol{F}. \tag{31}$$

It remains to evaluate λ_2. To this end, we substitute (29) into (23) to obtain

$$4\lambda_2^2(\delta P)^2 = I_{FF} + \lambda_1^2 I_{GG} - 2\lambda_1 I_{GF} \tag{32}$$

where I_{FF} is given by

$$I_{FF} = ||\boldsymbol{F}||^2. \tag{33}$$

Finally, we substitute (30) into (32) and solve for λ_2 to obtain

$$\lambda_2 = \frac{1}{2}\left[\frac{I_{GG}(\delta P)^2 - (\delta Q)^2}{I_{FF}I_{GG} - I_{GF}^2}\right]^{-1/2}. \tag{34}$$

Note that the positive square root value has been chosen for λ_2 in order to satisfy Equation (28).

Let us now discuss our choice for δQ. This choice is dictated by the demand that the quantity under the square root in Equation (34) should be positive. It is easy to show that the term $I_{FF}I_{GG} - I_{GF}^2$ is always positive by the Cauchy–Schwarz inequality [5]. Now, since $I_{GG} = ||\boldsymbol{G}||^2 \geq 0$, it follows that care must be taken to ensure that $I_{GG}(\delta P)^2 > (\delta Q)^2$. The simplest way to achieve this is to select δQ adaptively by setting $\delta Q = -\xi\delta P\sqrt{I_{GG}}$ with $0 < \xi < 1$. Consequently, the proposed generic weight update algorithm has two free parameters, namely δP and ξ.

6. The SocialFALCON Algorithm

The most important aspect that we will now discuss is the definition of the quantity Φ that we wish to incrementally maximize at each epoch. Rather than adopting the approach of SocialMF which imposes a target condition on the latent feature vector of a user as the weighted average of the latent feature vectors of his direct neighbors, we will adopt and epoch-by-epoch approach so as to incrementally maximize at each epoch the alignment of the user's feature vector update to the weighted average of the feature vectors updates of his direct neighbors at the immediately preceding epoch.

We thus introduce the quantity:

$$\Phi = \sum_{u=1}^{N} d\beta_{u_t} \sum_{v \in N_u} T_{uv} d\beta_{v_{t-1}} + \sum_{u=1}^{N} d\mathbf{U}_{u_t}^T \sum_{v \in N_u} T_{uv} d\mathbf{U}_{v_{t-1}} \tag{35}$$

where $d\mathbf{U}_{u_t}$ is the user u feature vector update at the present epoch and $\sum_{v \in N_u} T_{uv} d\mathbf{U}_{v_{t-1}}$ is the weighted average of the latent feature vector updates of his direct neighbors v at the immediately preceding epoch. Similarly, $d\beta_{u_t}$ is the update of the bias of user u at the current epoch and $\sum_{v \in N_u} T_{uv} d\beta_{v_{t-1}}$ is the weighted average of the updates of the biases of the user's direct neighbors at the immediately preceding epoch. Since within our constrained learning framework the whole weight vector $\boldsymbol{w}$ updates have constant moduli equal to δP (by Equation (22)), maximization of Φ amounts to minimization of the angle between each user's feature vector updates at the present epoch and the weighted average of the latent feature vector updates of his direct neighbors preceding that epoch.

Hence, for the factorization problem, at each epoch of the learning process we can restore from the weight vector $\boldsymbol{w}$ the user feature vectors $\mathbf{U}_u$ using Equation (20), and update them according to Equations (29) and (26) as:

$$\begin{aligned} d\mathbf{U}_u &= -\frac{\lambda_1}{2\lambda_2} \frac{\partial \mathcal{L}}{\partial \mathbf{U}_u} + \frac{1}{2\lambda_2} \frac{\partial \Phi}{\partial \mathbf{U}_u} \\ &= -\frac{\lambda_1}{2\lambda_2} \sum_{i=1}^{M} I_{ui} \cdot e_{ui} \cdot g^{'}(\theta_{ui}) \cdot \mathbf{V}_i + \frac{1}{2\lambda_2} \sum_{v \in N_u} T_{uv} d\mathbf{U}_{v_{t-1}}. \end{aligned} \tag{36}$$

Similarly the user biases will be updated by:

$$d\beta_u = \frac{\lambda_1}{2\lambda_2} \sum_{i=1}^{M} I_{ui} \cdot e_{ui} \cdot g^{'}(\theta_{ui}) + \frac{1}{2\lambda_2} \sum_{v \in N_u} T_{uv} d\beta_{v_{t-1}}. \tag{37}$$

As far as the updates of the item feature vectors $\mathbf{V}_i$ and item biases γ_i are concerned, there are two possible alternatives. The first is to simply update them by:

$$d\mathbf{V}_i = -\frac{\lambda_1}{2\lambda_2} \frac{\partial \mathcal{L}}{\partial \mathbf{V}_i} \text{ and } d\gamma_i = -\frac{\lambda_1}{2\lambda_2} \frac{\partial \mathcal{L}}{\partial \gamma_i} \tag{38}$$

since there are no dependencies of Φ as defined in Equation (35) from any $\mathbf{V}_i$ or γ_i.

An even more interesting option for updating the item feature vectors and biases is to supplement the quantity Φ with a term involving an incremental maximization target that we seek to achieve for each $\mathbf{V}_i$ and γ_i on an epoch-by-epoch basis. To this end, we adopt the approach that we have proposed in algorithm FALCON-M as described in [6]. According to that approach, the quantity that we wish to maximize is the alignment between the item feature vector updates at the present and immediately preceding epoch. Due to the constant moduli of the updates of the whole vector $\boldsymbol{w}$ (Equation (22), this also amounts to minimization of the angle between the vectors of the successive weight updates for each $\mathbf{V}_i$, and thus can suppress zig-zagging and allow learning to proceed along relatively smooth paths. In a sense this is equivalent to adding a momentum term in the updates of each $\mathbf{V}_i$ (and γ_i), with the important difference that both the coefficients of the gradient (i.e., learning rate) and momentum are suitably adapted at each epoch of the learning process by the constained update rule of Equation (29).

Note that for this approach to work we should redefine Φ as:

$$\Phi = \sum_{u=1}^{N} d\beta_{u_t} \sum_{v \in N_u} T_{uv} d\beta_{v_{t-1}} + \sum_{u=1}^{N} d\mathbf{U}_{u_t}^T \sum_{v \in N_u} T_{uv} d\mathbf{U}_{v_{t-1}} + \sum_{i=1}^{M} d\gamma_{i_t} d\gamma_{i_{t-1}} + \sum_{i=1}^{M} d\mathbf{V}_{i_t}^T d\mathbf{V}_{i_{t-1}} \tag{39}$$

where $d\mathbf{V}_{i_t}$ and $d\mathbf{V}_{i_{t-1}}$ are the item i feature vector updates at the present and immediately preceding epoch respectively (similarly for the item i bias updates $d\gamma_{i_t}$ and $d\gamma_{i_{t-1}}$).

Again we can restore from the weight vector $\boldsymbol{w}$ the item feature vectors $\mathbf{V}_i$ using Equation (20), and update them according to Equations (29) and (26) as:

$$\begin{aligned} d\mathbf{V}_i &= -\frac{\lambda_1}{2\lambda_2} \frac{\partial \mathcal{L}}{\partial \mathbf{V}_i} + \frac{1}{2\lambda_2} \frac{\partial \Phi}{\partial \mathbf{V}_i} \\ &= -\frac{\lambda_1}{2\lambda_2} \sum_{u=1}^{N} I_{ui} \cdot e_{ui} \cdot g^{'}(\theta_{ui}) \cdot \mathbf{U}_u + \frac{1}{2\lambda_2} d\mathbf{V}_{i_{t-1}}. \end{aligned} \tag{40}$$

Similarly the updates for the item biases will be given by:

$$d\gamma_i = -\frac{\lambda_1}{2\lambda_2} \sum_{u=1}^{N} I_{ui} \cdot e_{ui} \cdot g^{'}(\theta_{ui}) + \frac{1}{2\lambda_2} d\gamma_{i_{t-1}}. \tag{41}$$

Equations (36), (40), (37), and (41) thus constitute our proposed SocialFALCON algorithm for updating the user features, item features, user biases, and item biases respectively.

6.1. Desirable Properties of SocialFALCON

As is the case with SocialMF, the SocialFALCON model makes the feature vector of each user to be dependent on the feature vectors of his direct neighbors. Since those neighbors are in turn connected to other users, recursively, indirect connections in the social network propagate their social infuence across the entire network. In addition, even for users that have expressed no ratings (cold start users) but have social connections, there still remains a social update term in their factor update rule, which means that latent features of those users will at least adapt towards those of their neighbors. Therefore, despite not having any expressed ratings, feature vectors for these users will be learned as well.

The SocialFALCON model has three further desirable properties. First, despite its seemingly complex derivation, the factor update rule for both users and items (and their biases) turns out to be quite simple since it only constitutes from a term proportional to the gradient of the objective function of Equation (21), and an extra term proportional to the gradient of Φ as given by Equation (39) with respect to each feature vector. Both of these terms are quite easy to compute as it can be seen from Equations (36), (37), (40) and (41). For the case of user factors and their biases, the extra term is a social term adapting the update towards the weighted average of the factors' updates of the direct neighbors at the immediately preceding epoch, whereas for the item factors and biases the extra term acts as momentum.

Second, all the factor update rules can also be viewed as standard gradient descent with regularization. This is enforced by the hard constraint of Equation (22) that restricts the norm of the weight vector update $d\boldsymbol{w}$ to a hypersphere (which means that the compoments of the vector $\boldsymbol{w}$ cannot grow in an uncontrollable fashion), and can be more easily seen if we rewrite, for example, Equation (40) as:

$$
\begin{aligned}
d\mathbf{V}_i &= -\frac{\lambda_1}{2\lambda_2}\sum_{u=1}^{N} I_{ui}\cdot e_{ui}\cdot g^{'}(\theta_{ui})\cdot \mathbf{U}_u + \frac{1}{2\lambda_2} d\mathbf{V}_{i_{t-1}} \\
&= -\frac{\lambda_1}{2\lambda_2}\sum_{u=1}^{N} I_{ui}\cdot e_{ui}\cdot g^{'}(\theta_{ui})\cdot \mathbf{U}_u + \frac{1}{2\lambda_2}(\mathbf{V}_i - \mathbf{V}_{i_{t-1}}). \qquad (42)
\end{aligned}
$$

The same argument applies, by expansion of the weighted average in the social term, to user factor updates. An important benefit here is that both the coefficient multiplying the gradient as well as the reguralization coefficient are automatically adapted at each epoch by the current values of λ_1 and λ_2.

Finally, the algorithm has only two free parameters, namely δP and ξ which, as we'll discuss in the experimental results section, are not very sensitive to the exact setting of their values.

6.2. Complexity Analysis of SocialFALCON

The main overhead to complete an epoch is in evaluating the gradient of the objective function with respect to the feature vectors of users and items. As it can been seen from Equation (21), the number of operations per feature vector needed to complete this evaluation is proportional to the number of rating examples in the training set. In addition we need to calculate the gradient of Φ as given by Equation (39) with respect to the feature vectors. Following the notation used in [15], we assume that the average number of ratings per user is $\bar{r}$, and the average number of direct neighbors per user is $\bar{t}$. As reported in the same paper, the computational complexity of computing the gradients of SocialMF's objective function $\mathcal{L}$ with respect to the number of users in the social rating network is $O(N\bar{r}K + N\bar{t}^2K)$. The $\bar{t}^2$ factor is justified if we take a closer look at Equation (18) which computes the derivative of Equation (17) with respect to each user feature vector $\mathbf{U}_u$ so as to update it by gradient descent. In order to evaluate this derivative, for each user u, one needs to take the following steps: (a) Find his direct neighbors and aggregate their feature vectors, (b) find the users v in the social network *to which* u is a direct neighbor, and (c) $\forall v$ find their direct neighbors w and aggregate their feature vectors.

In the SocialFALCON model the updates of the user feature vectors are given by Equation (36). As it can be readily seen from that equation, the evaluation of its second term only requires step (a) as described above. This makes the total omputational complexity of evaluating the gradients of (21) and (39) with respect to the number of users $O(N\bar{r}K + N\bar{t}K)$. Therefore SocialFALCON is $\frac{\bar{r}+\bar{t}^2}{\bar{r}+\bar{t}}$ times faster than SocialMF in computing the gradient in each training epoch as it scales linearly with the number of user connections. Since usually the rating matrix R and trust matrix T are very sparse, $\bar{t}$ and $\bar{r}$ are relatively small, and therefore both SocialMF and SocialFALCON scale linearly with respect to the total number of users in the social rating network. However in a large social network where the number of average number of direct neighbors per user is large, the speedup factor provided by SocialFALCON becomes profound and crucially important in the ability to effectively train a recommender.

We should note here that once the gradient has been evaluated, a relatively small number of additional operations as given by Equations (31), (33), (34) and (30) (which is independent of the number of training examples) is needed to complete the update. In addition the updates of all user and item factors at the immediately preceding epoch should be stored since they are utilized by the SocialFALCON update rule. This additional computational burden however is very small compared to the calculation of the gradient since it just involves the evaluation of three inner vector products as given by Equations (31) and (33). The sizes of these vectors are equal to the size of vector $\boldsymbol{w}$ (Equation (20)), thus the number of operations involved is $(K+1)*(N+M)$. This is confirmed by the actual CPU times measured in our experimental results.

7. Datasets

The availability of publicly available social rating network datasets is extremely limited, presumably due to the sensitive nature of social network data and the proprietary rights of social networks. To the best of our knowledge there are only very few public social rating network datasets: The Epinions.com dataset, and a dataset that was crawled from Flixster.com and has been made available by the authors of [15].

The Epinions dataset (http://www.trustlet.org/wiki/Downloaded_Epinions_dataset) that we used has 49,289 users, 664,824 ratings and 487,183 connections between users. Possible rating values are discrete integers in the range (1,5). The average number of ratings per user is 16.5 and each user has on average 14.3 direct neighbors. The social relations in the dataset are directed.

The Flixster dataset (http://www.cs.sfu.ca/$\sim$sja25/personal/datasets) dataset has 787,213 users, 8,196,077 ratings and 7,058,819 relations between users. Each user, on average, has rated 55.5 items and has 48.6 social relations. Even though the dataset was crawled by [15] as a directed network, the social relations are undirected. This means that either duplicates should be removed or be handled appropriately in the code (we opted for the later choice). Possible rating values are ten discrete numbers in the range (0.5,5), and in our implementation we scaled rating values by 2 to be integers in the range (1,10).

In order to evaluate the performance of all algorithms considered in the next section, we used five-fold cross-validation (CV). We used stratified sampling on user ratings so that, in each fold, 80% of the ratings of each user were chosen randomly and used for training and the remaining 20% of user's ratings were used for evaluation. Due to this sampling scheme, users with only one rating in the dataset were inevitably excluded from the evaluation set. We have made publicly available (http://labs.fme.aegean.gr/ideal/socialfalcon-datasets/) both the database schema as well as the five-fold CV training/validation splits that we've used in the experiments.

8. Experimental Results

We compared the proposed SocialFALCON algorithm with four other factorization models: RegSVD and PMF as described in Section 2, SVD++ as described in Section 3 and SocialMF as described in Section 4. All algorithms were implemented in low-level C and were compiled with the GNU gcc 4.6.3 compiler. We should point out that the codebase for RegSVD, PMF and SVD++ has been carefully developed both in terms of minimizing their runtime and increasing their prediction accuracy, due to their prior utilization by one of the authors as member in "The Ensemble" team which was a runner up for the Netflix Prize competition (http://netflixprize.com/leaderboard). For the code implementation of SVD++, it should be noted that a naive implementation of gradient descent is very ineffective, because if the implicit items are updated at each training example then the training process will be very slow. In our implementation we have used the "looping trick" proposed in [22] for NSVD1, which is a predecessor of SVD++ originally introduced by [12]. This modification produces exactly the same results as the naive implementation of gradient descent, however it makes a significant difference in running time, especially for large scale problems.

To the best of our knowledge the authors of [15] have not provided any source code implementation of SocialMF. The only known publicly available code implementation of SocialMF is within MyMediaLite (http://www.mymedialite.net/), which is a lightweight, multi-purpose library of recommender system algorithms. An important bug fix to that code implementation has also been provided by one of the authors of the present paper (http://mymedialite.net/download/Changes), which was contributed while developing our own implementation. Nonetheless, the codes for all algorithms presented in this section have been rewritten from scratch and have been made publicly available from Github (https://github.com/namp/SocialFALCON). The experiments were run on a machine with an Intel Core2 Quad CPU (Q9400 @2.66GHz) with 4GB RAM, running Ubuntu 12.04 LTS.

The prediction error was measured in RMSE on the evaluation set of each split defined as:

$$RMSE = \sqrt{\frac{1}{|E|} \sum_{r_{ui} \in E} (r_{ui} - \hat{r}_{ui})^2} \tag{43}$$

where $|E|$ is the size of the evaluation set E, r_{ui} is the actual rating and $\hat{r}_{ui}$ is the prediction.

Table 1 summarises the RMSE of RegSVD, PMF, SVD++, SocialMF and the proposed SocialFALCON algorithm on the Epinions and Flixster datasets. We report results for two choices of latent dimensions, specifically $K = 5$ and $K = 10$ as was also reported in [15], where it was shown that increasing K in Epinions did not improve the results, while increasing K in Flixster improved the results slightly. For all algorithms, each RMSE result is reported along with two numbers: The average number of epochs required to achieve it and the average time (in seconds) spent at each epoch. The RMSE numbers reported are the average of the five-fold CV splits. Note that for the Flixster dataset the average reported RMSE for all algorithms has been divided by 2 in order to compensate for reporting the error on the original rating values in the range (0.5,5). For fairness in the comparison, for all algorithms, each run was initialized from a common random seed, and their relevant parameters were carefully adjusted to achieve the best possible performance. These parameters were found by utilising grid search and are reported in Appendix A.

Table 1. Experimental results for Epinions and Flixster. In each table cell we report the triplet: RMSE/average number of epochs/time spent in each epoch (in secs).

Features (K)	Algorithm	Epinions			Flixster		
5	RegSVD	1.0490098	61.6	0.064946	0.8577111	166.4	2.1827098
	PMF	1.0492854	54	0.0963092	0.868567	14.4	2.3057142
	SVD++	1.0485256	62	0.145221	**0.8543018**	144	2.9196802
	SocialMF	1.048803	3180.6	1.0123998	0.8856209	979	55.8514706
	SocialFALCON	**1.0428028**	216.6	0.3194868	0.8554456	1040.2	3.8049924
10	RegSVD	1.0490002	61.6	0.0941402	0.8483767	187.8	2.2727028
	PMF	1.049246	53.8	0.1374766	0.8590072	15.2	2.8283334
	SVD++	1.0483972	61.2	0.2286722	**0.845011**	180.4	4.134036
	SocialMF	1.0487536	3179.6	1.6876002	0.8760801	983.6	72.0185296
	SocialFALCON	**1.0427142**	216.2	0.6428872	0.8484094	1082	5.0869388

For the Epinions dataset, as highlighted in bold, SocialFALCON improves the RMSE of RegSVD, PMF, SVD++, and SocialMF on both choices of latent dimensions. In both cases, we notice that the RMSE of SocialFALCON is 0.6% lower compared to that of RegSVD, 0.62% lower than the RMSE of PMF, and 0.55% lower compared to the RMSE of SVD++. Empirical data on recommender systems research have shown that such levels of improvement are indeed significant [23]. For example during the Netflix Prize competition, yearly progress prizes of $50,000 US were offered for the best improvement of at least 1% over the previous year's result [24]. The closely related SocialMF method seems to perform only marginally better than RegSVD and PMF, and marginally worse than SVD++. Naturally, as explained in the beginning of this section, the codes for RegSVD, PMF, and SVD++ have been very carefully implemented due to their utilization in the Netflix prize and thus their performance is expected to be quite good. However the same argument is also true for the code implementation of SocialMF. In addition all RMSEs are reported with optimal parameters found by grid search, and to this end, it remains unclear why we have not been able to confirm in our evaluation the much improved performance of SocialMF over PMF on the Epinions dataset as reported in [15]. A possible explanation for this is that in their experiments they used a different version of the Epinions dataset and, of course, our CV splits are different. Another explanation for the poor performance they reported for PMF is perhaps due to poor code implementation. Still, the results that we obtained by SocialMF for the Epinions dataset are better than those reported on their paper.

For the much larger Flixster dataset, for both choices of latent dimensions, SocialFALCON outperforms PMF and SocialMF, and performs as good as RegSVD (which apparently performed

remarkably well on this task considering its model simplicity). SVD++ performs marginally better than SocialFALCON and RegSVD, which can be attributed to the large number of implicit ratings in the evaluation sets. In general, the RMSE results reported for all algorithms are better than the results for Epinions. An explanation for this has been given in [15] by noticing that the items in Epinions are heterogeneous (belonging to categories such as DVD players, cameras, printers, laptops, etc.), while the items in Flixster are from a single caterory, namely movies. Possibly this makes the rating signals quite more accurate than those of Epinions, and thus makes the factorization task easier in general. Despite the large number of social connections in the dataset, apparently SocialMF provided the worst RMSE score among all the algorithms tested. In this case also we have been unable to verify the claims of significant RMSE improvement over PMF for the Flixster dataset and, in general, the SocialMF's results reported in [15]. It is also interesting to note how much SocialFALCON appears to improve the RMSE over SocialMF, which is 3.4% for $K = 5$ and 3.15% for $K = 10$.

Overall, the proposed SocialFALCON algorithm performs well and it is competitive to state of the art matrix factorisation methods. More important however, is that it can effectively handle large datasets. This is supported by the other two numbers, apart from the RMSE, that are shown in each cell of Table 1. SocialFALCON and SocialMF are batch methods, and hence they require a much larger number of epochs than stochastic algorithms such as RegSVD, PMF and SVD++. The reason is that batch methods only have one chance to update the factors after each presentation of the whole training set, whereas stochastic methods make thousands of updates in one epoch, up to the $N \times M$ size of the user-by-item ratings matrix. This explains the significant difference, for example, between PMF and SocialFALCON or SocialMF in the average number of epochs to achieve the reported RMSE score. We should mention that SocialFALCON's average number of epochs, as reported for the various tasks in Table 1, are higher than should be needed in practice. This is because its best scores were reported for small stepsizes δP discovered by the grid search, as shown in Appendix A, even thought quite comparable RMSE scores where attained with larger stepsizes and hence fewer epochs. Similar performances were recorded with $0.5 < \delta P < 2.0$ and $0.7 < \xi < 0.9$, indicating that results are not very sensitive to the exact values of the parameters. Following the example of [25], we performed additional runs of SocialFALCON using common values ($\delta P = 1.0$) and ($\xi = 0.85$) on the two datasets. Deterioration of the RMSE compared to the scores reported with the optimal parameters shown in Table 1, was in both cases never more than 2%, even though for the case of Flixster there was more than 30% reduction on the required number of epochs. The larger scale problem was more sensitive to the selection of δP, whereas it was not so sensitive to the selection of ξ (for which a value around 0.85 worked well in both cases).

The third number shown in each cell of Table 1, reports the average time required to complete an epoch and is quite informative as it is a measure of the computational complexity of each algorithm. The numbers show that the complexity/epoch of the SocialFALCON algorithm is not significantly higher than that of RegSVD and PMF and even SVD++. This is true in spite of the fact that our implementation of SVD++ is very efficient as we have utilised the "looping-trick" mentioned in the beginning of this section, which significantly reduces SVD++'s computational burden. Note also that according to our discussion in Section 6.2, SocialFALCON should theoretically be $\frac{\bar{r}+\bar{t}^2}{\bar{r}+\bar{t}}$ times faster than SocialMF. Since the Flixster data set is denser than the Epinions data set, the improvement over runtime efficiency for Flixster should be more prominent than that for Epinions. As we can see from Table 1, indeed SocialMF is much slower than SocialFALCON. The epoch of SocialFALCON is 2.9 times faster than that of SocialMF for Epinions and 14.5 times faster for Flixster.

Cold Start Users

Based on the findings of the previous section where, in general, the performance of SocialFALCON compared to SocialMF was significantly better, we also investigated the performance of these two closely related methods only on cold start users. As reported in [15], in both Flixster and Epinions more than 50% of users are cold start users and thus the efficiency of recommendations on this challenging

class of users becomes very important. As cold start users we define users that have less than five ratings in each dataset and at least two, since users with only one rating were not included in the evaluation set of each fold. To this end, we created a subset of the evaluation set of each CV fold which contained only cold start users.

Table 2 summarises the RMSE of SocialMF and the proposed SocialFALCON algorithm for cold start users. For the Epinions dataset, as highlighted in bold, SocialFALCON improves the RMSE of SocialMF on both choices of latent dimensions. For the Flixster dataset, in both cases (especially for $K = 10$), SocialMF performs marginally better than SocialFALCON. On this dataset which has denser social relations, this can be attributed to the hard preset target distance metric incorporated into the SocialMF factorization model that, at the end of learning, the latent feature vector of each user is rendered equal to the weighted average of the latent feature vectors of his direct neighbors. On the other hand SocialFALCON seeks to minimize that distance incrementally by aligning the corresponding updates so as to render the latent feature vector of each user as close as possible to the weighted average of the latent feature vectors of his direct neighbors at the end of learning. Thus, for cold start users, social connections alone, might play a slightly more dominant role for providing recommendations in SocialMF than in SocialFALCON, which comes of course with a higher computational cost as was discuseed in the previous sections.

Table 2. Experimental results for Epinions and Flixster, cold start users only.

Features	Algorithm	Epinions	Flixster
5	SocialMF	1.1265016	**1.0904493**
	SocialFalcon	**1.1101022**	1.1067439
10	SocialMF	1.126648	**1.1094216**
	SocialFalcon	**1.1102382**	1.109602

9. Conclusions

In this paper, we proposed an efficient constrained matrix factorization algorithm called SocialFALCON, for providing recommendations in social rating networks. The algorithm derives from the FALCON generic constrained matrix factorization which has been previously proposed by authors of this paper. The FALCON framework allows the incorporation of additional knowledge into the learning mechanism for determining the user and item factor matrices. In the case of SocialFALCON, this additional knowledge is embedded into mathematical constraints that, during learning, drive the feature vector of each user to be dependent on the feature vectors of his direct neighbors in the social network. Similarly to related proposed approaches, this allows the propagation of social influence within the factorization model, which has been shown to be an important factor in the social sciences, in social network analysis and in trust-based recommendations. The propagation of social infuence also allows for providing recommendations to cold start users that have not expressed many ratings, since their feature vectors can be learned through their social relations. However, unlike similar approaches the proposed algorithm has reduced computational complexity, can be implemented easily, and thus be utilized more frequently in social recommendation setups.

Experimental results on two publicly available datasets showed that the algorithm improves on baseline and state of the art factorization methods as well as on previously proposed related approaches in terms of convergence speed and recommendation accuracy. To elucidate on the transparency of our results, we have made publicly available the source code of the proposed algorithm, as well as the codes of the algorithms against which it was compared, along with the datasets used in the experiments.

One of the most attractive features of this algorithm is its potential for suggesting several interesting directions for further improvements. In the same framework for constrained matrix factorization, it is possible to augment it with further information about learning in social rating networks, including methods for embedding graph theory measures and indices into the social trust values and incorporating implicit direct neighbor ratings as utilized in SVD++. It is the concerted

incorporation of such detailed information into the same algorithm that will hopefully lead to increasingly efficient matrix factorization training schemes combining fast learning, good scalability properties, and powerful generalization capabilities on predicted ratings for unseen items.

Author Contributions: Conceptualization, N.A.; Methodology, N.A. and T.E.; Software, N.A. and T.E.; Validation, T.E.; Writing–original draft, N.A. and T.E.; Writing–review & editing, F.S.

Funding: F.S. is supported by a PhD Scholarship by the State Scholarships Foundation (IKY), Greece.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Algorithms' Parameters

In this section we list the learning parameters (discovered by grid search) with which the algorithms achieved their best performance on each dataset, as presented in the results tables (Table 1 for all users and Table 2 for cold start users only).

In order to boost the performance of RegSVD, PMF, SVD++ and SocialMF we used different learning rates for the user features and biases (η_u and η_{bu} respectively) and for the item features and biases (η_v and η_{bv} respectively). SVD++ also requires two further parameters for the training of the implicit item factors, namely η_y and λ_y which are the learning rate and regularisation parameter respectively.

SocialFALCON has only two free parameters, namely δP and ξ.

- **RegSVD (Epinions)**: η_u=0.003, η_v=0.00005, η_{bu}=0.003, η_{bv}=0.003, λ_u=0.015, λ_v=0.1, λ_β=0.015, λ_γ=0.015
- **RegSVD (Flixster)**: η_u=0.02, η_v=0.005, η_{bu}=0.02, η_{bv}=0.005, λ_u=0.1, λ_v=0.1, λ_β=0.01, λ_γ=0.01
- **PMF (Epinions)**: η_u=0.005, η_v=0.005, η_{bu}=0.005, η_{bv}=0.005, λ_u=0.1, λ_v=0.1, λ_β=0.01, λ_γ=0.01
- **PMF (Flixster)**: η_u=0.015, η_v=0.005, η_{bu}=0.015, η_{bv}=0.005, λ_u=0.01, λ_v=0.01, λ_β=0.01, λ_γ=0.01
- **SVD++ (Epinions)**: η_u=0.003, η_v=0.003, η_{bu}=0.003, η_{bv}=0.003, η_y=0.003,λ_u=0.015, λ_v=0.5, λ_β=0.015, λ_γ=0.015, λ_y=1.0
- **SVD++ (Flixster)**: η_u=0.003, η_v=0.003, η_{bu}=0.003, η_{bv}=0.003, η_y=0.003,λ_u=0.015, λ_v=0.5, λ_β=0.015, λ_γ=0.015, λ_y=1.0
- **SocialMF (Epinions)**: η_u=0.0001, η_v=0.0001, η_{bu}=0.0001, η_{bv}=0.0001, λ_u=0.1, λ_v=0.1, λ_β=0.1, λ_γ=0.1, λ_t=0.1
- **SocialMF (Flixster)**: η_u=0.0001, η_v=0.0001, η_{bu}=0.0001, η_{bv}=0.0001, λ_u=0.1, λ_v=0.1, λ_β=0.1, λ_γ=0.1, λ_t=0.1
- **SocialFALCON (Epinions)**: δP=1.0, ξ=0.8
- **SocialFALCON (Flixster)**: δP=0.3, ξ=0.95

References

1. Takács, G.; Pilászy, I.; Németh, B.; Tikk, D. On the Gravity Recommendation System. In Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining, San Jose, CA, USA, 12–15 August 2007; pp. 22–30.
2. Koren, Y. The BellKor Solution to the Netflix Grand Prize. Available online: http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf (accessed on 11 August 2019).
3. Toscher, A.; Jahrer, M.; Bell, R. The Big Chaos Solution to the Netflix Grand Prize. Available online: http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf (accessed on 11 August 2019).
4. Piotte, M.; Chabbert, M. The Pragmatic Theory Solution to the Netflix Grand Prize. Available online: http://www.netflixprize.com/assets/GrandPrize2009_BPC_PragmaticTheory.pdf (accessed on 11 August 2019).
5. Perantonis, S.J.; Ampazis, N.; Virvilis, V. A Learning Framework for Neural Networks Using Constrained Optimization Methods. *Ann. Oper. Res.* **2000**, *99*, 385–401. [CrossRef]
6. Ampazis, N.; Emmanouilidis, T. FALCON: A matrix factorization framework for recommender systems using constrained optimization. *Intell. Decis. Technol.* **2015**, *9*, 221–232. [CrossRef]
7. He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; Chua, T.S. Neural Collaborative Filtering. In Proceedings of the 26th International Conference on World Wide Web, Perth, Australia, 3–7 April 2017; pp. 173–182. [CrossRef]

8. Wasserman, S.; Faust, K. *Social Network Analysis: Methods and Applications*; Cambridge University Press: Cambridge, UK, 1994; Volume 8.
9. Crandall, D.; Cosley, D.; Huttenlocher, D.; Kleinberg, J.; Suri, S. Feedback Effects Between Similarity and Social Influence in Online Communities. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, NV, USA, 24–27 August 2008; pp. 160–168.
10. Tang, J.; Hu, X.; Liu, H. Social recommendation: A review. *Soc. Netw. Anal. Min.* **2013**, *3*, 1113–1133. [CrossRef]
11. Yang, X.; Guo, Y.; Liu, Y.; Steck, H. A survey of collaborative filtering based social recommender systems. *Comput. Commun.* **2014**, *41*, 1–10. [CrossRef]
12. Paterek, A. Improving regularized singular value decomposition for collaborative filtering. In Proceedings of the KDD Cup and Workshop, San Jose, CA, USA, 12 August 2007.
13. Salakhutdinov, R.; Mnih, A. Probabilistic Matrix Factorization. In Proceedings of the 20th International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 3–6 December 2007; pp. 1257–1264.
14. Koren, Y. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, NV, USA, 24–27 August 2008; ACM: New York, NY, USA, 2008; pp. 426–434.
15. Jamali, M.; Ester, M. A matrix factorization technique with trust propagation for recommendation in social networks. In Proceedings of the Fourth ACM Conference on Recommender Systems, Barcelona, Spain, 26–30 September 2010; ACM: New York, NY, USA, 2010; pp. 135–142.
16. Koren, Y.; Bell, R.; Volinsky, C. Matrix Factorization Techniques for Recommender Systems. *Computer* **2009**, *42*, 30–37. [CrossRef]
17. Cullum, J.; Willoughby, R. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Volume 1, Theory*; Classics in Applied Mathematics, Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2002.
18. Chang, C.C.; Lin, C.J. LIBSVM: A library for support vector machines. *Acm Trans. Intell. Syst. Technol.* **2011**, *2*, 27:1–27:27. [CrossRef]
19. Ma, H.; King, I.; Lyu, M.R. Learning to recommend with social trust ensemble. In Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, Boston, MA, USA, 19–23 July 2009; ACM: New York, NY, USA, 2009; pp. 203–210.
20. Yang, X.; Steck, H.; Liu, Y. Circle-based Recommendation in Online Social Networks. In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Beijing, China, 12–16 August 2012; ACM: New York, NY, USA, 2012; pp. 1267–1275. [CrossRef]
21. Bryson, A.E.; Denham, W.F. A Steepest-Ascent Method for Solving optimum Programming Problems. *J. Appl. Mech.* **1962**, *29*, 247–257. [CrossRef]
22. Takács, G.; Pilászy, I.; Németh, B.; Tikk, D. A unified approach of factor models and neighbor based methods for large recommender systems. In Proceedings of the 2008 First International Conference on the Applications of Digital Information and Web Technologies (ICADIWT), Ostrava, Czech Republic, 4–6 August 2008; pp. 186–191.
23. Feuerverger, A.; He, Y.; Khatri, S. Statistical Significance of the Netflix Challenge. *Stat. Sci.* **2012**, *27*, 202–231. [CrossRef]
24. Bennett, J.; Lanning, S. The Netflix Prize. In Proceedings of the KDD-Cup and Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, CA, USA, 12–15 August 2007.
25. Jacobs, R.A. Increased rates of convergence through learning rate adaptation. *Neural Netw.* **1988**, *1*, 295–307. [CrossRef]

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).