Online-Updating Regularized Kernel Matrix Factorization Models for Large-Scale Recommender Systems

Steffen Rendle, Lars Schmidt-Thieme
Machine Learning Lab
Institute for Computer Science
University of Hildesheim, Germany
{srendle, schmidt-thieme}@ismll.uni-hildesheim.de

ABSTRACT

Regularized matrix factorization models are known to generate high quality rating predictions for recommender systems. One of the major drawbacks of matrix factorization is that once computed, the model is static. For real-world applications dynamic updating a model is one of the most important tasks. Especially when ratings on new users or new items come in, updating the feature matrices is crucial.

In this paper, we generalize regularized matrix factorization (RMF) to regularized kernel matrix factorization (RKMF). Kernels provide a flexible method for deriving new matrix factorization methods. Furthermore with kernels nonlinear interactions between feature vectors are possible. We propose a generic method for learning RKMF models. From this method we derive an online-update algorithm for RKMF models that allows to solve the new-user/ new-item problem. Our evaluation indicates that our proposed online-update methods are accurate in approximating a full retrain of a RKMF model while the runtime of online-updating is in the range of milliseconds even for huge datasets like Netflix.

Categories and Subject Descriptors

I.2.6 [**Learning**]: Parameter learning

General Terms

Algorithms, Experimentation, Measurement, Performance

1. INTRODUCTION

Regularized matrix factorization is known to be one of the most successful methods for rating prediction outperforming other methods like pearson-correlation based kNN or co-clustering [1, 14, 3]. One drawback is that the model (the factorization) is learned in batch mode. After having learned the model, it is applied for prediction. For lab evaluation this works fine. But in real-world scenarios like an online shop or a video rental service there is not such a distinction between training and prediction. In fact after the training phase visitors will generate new feedback (e.g. rate items). For users that have already rated a lot the user's profile will not change much, but for a new user each feedback that he gives will result in much change in his inferable taste. Therefore adding ratings to a user with a small rating profile should result in a much better model and thus to better predictions for this user. We will refer to this as the 'new-user problem'. Symmetrically for items, the 'new-item problem' is formulated. The scenario of new items and new users is important in almost any real-world recommender and is especially crucial in domains with changing content e.g. on a news website, TV program, etc. The new-user problem occurs in almost every application from online shopping to personalized websites.

In this work, first we generalize regularized matrix factorization (RMF) to regularized kernel matrix factorization (RKMF). A kernel function allows to transform the product of the factor matrices. Kernels like the s-shaped logistic function allow to impose bounds on the prediction (e.g. one to five stars) while still being differentiable. We will propose a gradient descent based algorithm for learning RKMF models. We will show that best k-rank SVD approximations are different from RKMF models and that for recommender systems RKMF clearly outperforms SVD as RKMF models do not need imputation and have a regularization term.

Secondly, we show how online-updates can be applied on a learned RKMF model without having to retrain the whole model. Our online-update methods especially targets the new-user and new-item problem. The proposed update methods are generic and applicable for all RKMF models. They are derived directly from the gradient descent method that we propose for learning RKMF models. In the evaluation we will see that the online-updates for new-user/new-item problems approximate the prediction of retraining the whole model very well. Both theoretical and empirical results show that the learning complexity of our online-updates is low. That makes RKMF models an ideal choice for real-world applications both in terms of runtime complexity and prediction quality.

In all, our contributions are as follows: (i) We introduce the model class of regularized kernel based matrix factorization for rating prediction in recommender systems. We provide a generic learning method and present three concrete instances of this class. (ii) We show how the major problem of updating features of new users and new items can be solved efficiently for the whole model class of regularized KMF. An extensive evaluation substantiates the ef-

RecSys'08, October 23–25, 2008, Lausanne, Switzerland.

fectiveness of our generic online-update-rules both in terms of prediction quality and runtime.

2. RECOMMENDER SYSTEMS

The central task in recommender systems is to predict the taste of a user. In this paper, we deal with the rating prediction problem that tries to predict how much a user likes a particular item. The estimated ratings can be used to recommend items to the user, e.g. what movies he might want to watch. The prediction is based on the user's feedback and feedback of other users in the past. Both feedback and prediction are supposed to be numerical values, e.g. one to five stars.

2.1 Rating Prediction

The problem of rating prediction can be seen as a matrix completion task, where a rating matrix R should be completed. The rows of R correspond to the users U and the columns to the items I. Thus the matrix has dimension $|U| \times |I|$. The entry $r_{u,i}$ of the matrix R contains the rating value of user u for item i. R is sparse, that means many values are unobserved (missing) values. We denote the set of observed ratings by S which contains triples (u, i, v) or $r_{u,i}$ of feedback from the user. The task of rating prediction is to complete R with an estimation $\hat{R}$.

2.2 New-User Problem / New-Item Problem

Typically, applications of recommender systems are dynamic. That means that new users sign in, new items are added and new ratings are given. On the other hand, many large-scale recommender models are static, e.g. correlation coefficients or nearest neighbors often are precomputed for collaborative filtering or factor matrices have been computed for matrix factorization. If after training these models, a new user signs in and gives some ratings, the recommender system has not adapted and gives poor recommendations. That means from the perspective of the user, the recommendations do not improve when he gives additional ratings. This is especially disappointing at the beginning when one expects large improvements from giving ratings. In this paper we deal with this problem both for new items and new users.

We define the 'new-user problem' as follows: The profile $C(u, \cdot)$ of a user u grows from 0 ratings to k ratings. Where C is defined as:

$$C(u, i) := \{r_{u',i'} \in S | u' = u \wedge i' = i\} \quad (1)$$

Symmetrically, we define the 'new-item problem' as following: The profile $C(\cdot, i)$ of a item i grows from 0 ratings to k ratings.

3. RELATED WORK

There are many approaches for rating prediction. Long-established is collaborative filtering based on the k-nearest-neighbor method (kNN) [4, 11, 1]. Other approaches use latent semantic models [6], classifiers [13], etc. Another class of models is based on matrix factorization (MF). MF techniques have shown to be very effective on several datasets including Netflix [15, 14, 1].

There is already research in kernels for matrix factorization. Zhang et al. [16] present non-negative MF on kernels. Their motivation is to approximate a matrix where the rows specify objects and the columns specify attributes. They map the entries within the attribute dimension in a higher dimension and use kernels to perform multiplications between two attribute vectors (columns). Our approach differs as we use the kernel between user and item vectors (row and column). Furthermore we use regularization to learn the model. In [14] Takacs et al. propose to apply a rounding function on parts of the output of MF. If their rounding function is applied to the whole output, i.e. the dot product, this can be seen as a special case of a kernel. Similarly Salakhutdinov and Mnih [9] suggest to use the logistic function to bound the output of the dot product. The kernel approach that we suggest is more general.

Updating k-rank SVD models has already been studied in the field of recommender systems [2, 10]. As we will see, best k-rank SVD and regularized MF are different. For the task of rating prediction where there is a huge number of missing values, SVD suffers from imputation and overfitting. A detailed discussion on the differences between SVD and regularized (K)MF can be found in section 4.5.

Online updates for kernel methods like SVMs or kernel regression have been studied e.g. by Kivinen et al. [7]. In contrast to this work we present online updates for regularized kernel matrix factorization.

If there are attributes on users (e.g. demographic data) or attributes on items (e.g. genre, actors) this information can be used for recommendation. Especially when little rating information on a specific user or item is present like in the new-user and new-item problem (aka 'cold start problem'), attribute information can improve the prediction quality [12, 5]. In this paper we deal with problems without any attribute information.

4. REGULARIZED KERNEL MATRIX FACTORIZATION

In this section, we first introduce matrix factorization and then we generalize this to kernel matrix factorization. As MF models have a huge number of parameters, for optimizing a regularization term is added to prevent overfitting. We propose a stochastic gradient descent approach to learn RKMF models in general and provide update rules for three models, i.e. linear, logistic and linear with non-negative constraints. At the end of this section we compare RKMF to best k-rank SVD. Sometimes people refer to RMF as regularized SVD which is not accurate as this is not a singular value decomposition. In this paper we use the term of SVD for real singular value decomposition and MF for matrix factorization in two matrices. We will see that our proposed RKMF models are better suited and provide better results for rating prediction than best k-rank SVD. The reasons are that (1) SVD tends to overfit because of missing regularization and (2) SVD needs imputation of missing values.

4.1 Matrix Factorization (MF)

Matrix factorization is the task of approximating the true unobserved ratings-matrix R by $\hat{R} : |U| \times |I|$. With $\hat{R}$ being the product of two feature matrices $W : |U| \times k$ and $H : |I| \times k$, where the u-th row w_u of W contains the k features that describe the u-th user and the i-th row h_i of H contains k corresponding features for the i-th item.

$$\hat{R} = W \cdot H^t \quad (2)$$

Or equivalently:

$$\hat{r}_{u,i} = \langle w_u, h_u \rangle = \sum_{f=1}^{k} w_{u,f} h_{i,f} \tag{3}$$

Often bias terms are added which are equivalent to centering the approximization, so that only residuals have to be learned:

$$\hat{r}_{u,i} = b_{u,i} + \sum_{f=1}^{k} w_{u,f} h_{i,f} \tag{4}$$

Normally the bias term $b_{u,i}$ is something like the global average, user average or item average, but also could be the result of another prediction algorithm. In our experiments we set $b_{u,i}$ to the global average rating, i.e. $\underset{r_{u,i} \in S}{\text{avg}}\ r_{u,i}$.

4.2 Kernel Matrix Factorization (KMF)

Like matrix factorization, kernel matrix factorization (KMF) uses two feature matrices that contain the features for users and items, respectively. But the interactions between the feature vector w_u of a user and the feature vector h_i of an item are kernelized:

$$\hat{r}_{u,i} = a + c \cdot K(w_u, h_i) \tag{5}$$

The terms a and c are introduced to allow rescaling the predictions. For the kernel $K : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ one can use one of the following well-known kernels:

$$K_l(w_u, h_i) = \langle w_u, h_u \rangle \qquad \text{linear} \tag{6}$$

$$K_p(w_u, h_i) = (1 + \langle w_u, h_u \rangle)^d \qquad \text{polynomial} \tag{7}$$

$$K_r(w_u, h_i) = \exp\left(-\frac{||w_u - h_u||^2}{2\sigma^2}\right) \qquad \text{RBF} \tag{8}$$

$$K_s(w_u, h_i) = \phi_s(b_{u,i} + \langle w_u, h_u \rangle) \qquad \text{logistic} \tag{9}$$

$$\text{with} \quad \phi_s(x) := \frac{1}{1 + e^{-x}}$$

It is obvious that normal matrix factorization like in equation (4) can be expressed with $a = b_{u,i}$ and $c = 1$ and the linear kernel K_l.

One benefit from using a kernel like the logistic one is that values are naturally bound to the application domain and do not have to be cut, e.g. a prediction of 6.2 stars in a scenario with at most 5 stars is not possible with the proposed logistic kernel. Secondly, kernels can provide non-linear interactions between user and item vectors. Finally, another benefit is that kernels lead to different models that can be combined in an ensemble. The Netflix challenge has shown that ensembling (e.g. blending) many models achieves the best prediction quality [1, 15, 14].

4.3 Non-negative Matrix Factorization

Non-negative matrix factorization is similar to matrix factorization but poses additional constraints on the feature matrices W and H. It is required that all elements of both matrices are non-negative. The motivation is to eliminate interactions between negative correlations which has been successfully applied in some collaborative filtering algorithms. With the linear kernel reasonable meta parameters are:

$$a := r_{\min}, \quad c := r_{\max} - r_{\min} \tag{10}$$

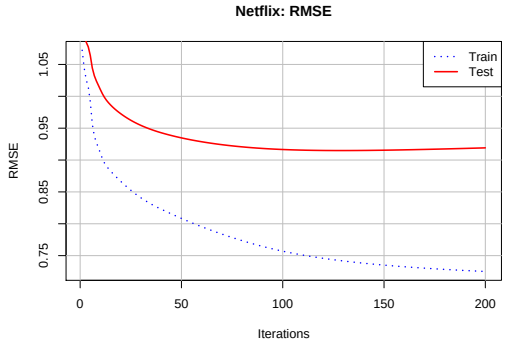

Figure 1: Fit on training and evaluation set ('probe') of regularized linear MF on Netflix.

4.4 Learning Matrix Factorization Models

The most important part for learning MF models is dealing with overfitting. Factorization models have a huge number of parameters, i.e. $O(k \cdot (|U| + |I|))$, that should approximate a matrix with lots of missing values. An illustrative example is the Netflix datasets with about 480,000 users and 17,000 items which leads to about 50 million free parameters for MF with $k = 100$. For Netflix the number of given ratings is about 100 million, that means R has about 8,060 million missing values. It is obvious that learning 50 million parameters from 100 million ratings will lead to overfitting. So strategies for overfitting play a central role for learning algorithms. For MF usually two strategies are proposed: (1) regularization and (2) early stopping. A second problem is the high number of missing values. Thus the optimization of model parameters is done only with respect to the observed values S of R. For optimizing wrt RMSE, the task is:

$$\underset{W,H}{\text{argmin}}\, E(S, W, H) \tag{11}$$

with

$$E(S, \hat{R}) := E(S, W, H) := \sum_{r_{u,i} \in S} (r_{u,i} - \hat{r}_{u,i})^2 \tag{12}$$

4.4.1 *Regularization*

Instead of learning the optimal fit of $W \cdot H^t$ on the observed values, a regularization term is added to the optimization task. Usually Tikhonov regularization aka ridge regression is used where a parameter λ controls the regularization. Hence, the optimization task is:

$$\underset{W,H}{\text{argmin}}\, \text{Opt}(S, W, H) \tag{13}$$

with

$$\text{Opt}(S, W, H) := E(S, W, H) + \lambda \left(||W||_F^2 + ||H||_F^2\right) \tag{14}$$

4.4.2 *Optimization by Gradient Descent*

For optimizing formula (14) different techniques might be used. For normal matrix factorization often stochastic gradient descent is used. We propose to use this also for KMF, so for optimizing only the partial derivative of K has to calculated. Also minimizing another (differentiable, monotonic) loss function than RMSE is easy, because only $E(S, W, H)$

```
1: procedure OPTIMIZE(S, W, H)
2:     initialize W, H
3:     repeat
4:         for r_{u,i} ∈ S do
5:             for f ← 1, ..., k do
6:                 w_{u,f} ← w_{u,f} − α ∂/∂w_{u,f} Opt({r_{u,i}}, W, H)
7:                 h_{i,f} ← h_{i,f} − α ∂/∂h_{i,f} Opt({r_{u,i}}, W, H)
8:             end for
9:         end for
10:    until Stopping criteria met
11:    return (W, H)
12: end procedure
```

Figure 2: Learning KMF by gradient descent.

has to be differentiated. In all, the generic learning algorithm is outlined in Figure 2. The parameter α is called the learning rate or step size. With the regularization term, avoiding overfitting by early stopping becomes less important. Nevertheless, early stopping can be applied to speed up the training process. In the simple case, the stopping criterion is a fixed number of iterations that could be optimized by a holdout method on the training set.

In the following, we give detailed optimization rules for three variants of RKMF, i.e. a linear kernel, a logistic kernel and a linear kernel with non-negative constraints. In the derivation, we will discard all positive constants that can be integrated in the learning rate α or the regularization λ. The partial derivations of $\text{Opt}(\{r_{u,i}\}, W, H)$ are:

$$\frac{\partial \text{Opt}(\{r_{u,i}\}, W, H)}{\partial w_{u,f}} \propto (\hat{r}_{u,i} - r_{u,i}) \cdot \frac{\partial}{\partial w_{u,f}} K(w_u, h_i) + \lambda w_{u,f} \tag{15}$$

$$\frac{\partial \text{Opt}(\{r_{u,i}\}, W, H)}{\partial h_{i,f}} \propto (\hat{r}_{u,i} - r_{u,i}) \cdot \frac{\partial}{\partial h_{i,f}} K(w_u, h_i) + \lambda h_{i,f} \tag{16}$$

In all, only the partial derivative of $K(w_u, h_i)$ is kernel specific.

4.4.3 Linear kernel

Optimizing formula (14) with gradient descent (figure 2) for a KMF with a linear kernel corresponds to normal regularized matrix factorization. For the complete update rules (15) and (16) we need the partial derivation of $K_l(w_u, h_i)$:

$$\frac{\partial}{\partial w_{u,f}} K(w_u, h_i) = h_{i,f}, \quad \frac{\partial}{\partial h_{i,f}} K(w_u, h_i) = w_{u,f} \tag{17}$$

For initialization of both feature matrices $w_{u,f}$ and $h_{i,f}$, small random values around 0 should be used. This way, in the beginning $\hat{r}_{u,i}$ is near the bias term.

4.4.4 Logistic kernel

For the logistic kernel (9), the update rules require to differentiate $K_s(w_u, h_i)$ which includes the logistic function:

$$\frac{\partial}{\partial w_{u,f}} K(w_u, h_i) \propto h_{i,f} \cdot \phi_s^2(b_{u,i} + \langle w_u, h_u \rangle) \cdot e^{-b_{u,i} - \langle w_u, h_u \rangle} \tag{18}$$

$$\frac{\partial}{\partial h_{i,f}} K(w_u, h_i) \propto w_{u,f} \cdot \phi_s^2(b_{u,i} + \langle w_u, h_u \rangle) \cdot e^{-b_{u,i} - \langle w_u, h_u \rangle} \tag{19}$$

Again, the features can be initialized with small random values around 0 when the following bias term is used:

$$b_{u,i} = -\ln\left(\frac{c}{g-a}\right) \tag{20}$$

The other hyper-parameters for the logistic kernel are $a = r_{\min}$ and $c = r_{\max} - r_{\min}$.

4.4.5 Non-Negative constraints

With a linear kernel and non-negative constraints on both W and H, the update rules can use a projection step to ensure non-negative elements:

$$w_{u,f} \leftarrow \max\left(0, w_{u,f} - \alpha \frac{\partial}{\partial w_{u,f}} \text{Opt}(\{r_{u,i}\}, W, H)\right) \tag{21}$$

$$h_{i,f} \leftarrow \max\left(0, h_{i,f} - \alpha \frac{\partial}{\partial h_{i,f}} \text{Opt}(\{r_{u,i}\}, W, H)\right) \tag{22}$$

The derivations are the same as in formula (15), (16) and (17). In contrast to unconstrained KMF, when dealing with non-negative constraints, the non-negative model cannot be centered around the global average with a bias term because otherwise no rating below the average would be possible. Thus, a initialization around 0 would lead to predictions around $r_{\min}$. A better initialization for non-negative matrix factorization is to set the values of both W and H s.t. the predictions are near the global average, which leads to $w_{u,f} = h_{i,f} = \sqrt{\frac{g - r_{\min}}{k \cdot (r_{\max} - r_{\min})}} + noise$

4.5 SVD versus Regularized KMF

Singular Value Decomposition (SVD) is a technique for decomposing a matrix into three matrices. With our notation that would be $R = W' \Sigma H'$ with $W' : |U| \times |U|$, $\Sigma : |U| \times |I|$ and $H' : |I| \times |I|$ where Σ is a diagonal matrix containing the singular values. One can show, that the best k-rank approximation $\hat{R}$ of R is given by using only the k largest singular values and setting the remaining singular values to zero. This means that one could reduce the number of columns of W' and H' to k and obtaining two matrices $W : |U| \times k$ and $H : |I| \times k$ that give the best approximation of $W \cdot H^t$ to R. As SVD is a well-studied technique in many fields, e.g. image analysis or numerics, the question arises why not to use it in recommender systems. As we indicated before the task of matrix approximation in recommender systems differs from other fields like image analysis.

First of all, in recommender systems we deal with a huge amount of missing/ unobserved values. E.g. for Netflix the sparsity factor is about 99%. Take care that sparsity in recommender systems means missing/ unobserved values whereas in SVD literature often zero values are meant. Before an SVD can be calculated, the missing values have to be estimated ('imputation'). Choosing 0 as missing value is obviously not a good idea because then most of the predicted ratings in $\hat{R}$ would be around 0 as well. A better idea is to use another prediction algorithm to estimate the missing values. In a simple case that could be the column or row mean. The SVD can then be calculated on this full matrix. An efficient implementation should recenter the matrix (e.g. around the column mean) before applying a standard SVD algorithm. A second problem for both MF and SVD is overfitting. As indicated before, in regularized MF usually regularization and early stopping is applied to avoid overfitting.

	SVD	Regularized MF		
		linear	logistic	lin. non-neg.
Netflix RMSE	0.946 [8]	0.915	0.918	0.914

Table 1: RMSE results on Netflix probe for RKMF and k-rank SVD.

In contrast to this, choosing the best k-rank approximation of an SVD will lead to overfitting of $\hat{R}$.

As we have seen, in the domain of recommender systems SVD has several drawbacks in contrast to RMF. First of all the high number of missing values that have to be estimated and the lack of regularization leads to overfitting. Table 1 shows a comparison of SVD to several regularized matrix factorization methods. The evaluation has been done on the Netflix dataset, where Netflix 'probe' was used as test dataset. We compare the RKMF prediction quality to the best SVD results reported by [8]. Their best SVD model uses the Lanczos algorithm with imputation by EM. This SVD model has best quality on the 'probe' set with 10 dimensions – for more dimensions they report overfitting. Our regularized KMF methods use 40 dimensions ($k = 40$) and we do not observe overfitting (see fig. 1 for the linear case). In fact, even if we enlarge the number of dimensions in RKMF (e.g. $k = 100$) the quality still increases. This evaluation has shown, that regularization is important for successfully learning a model and avoiding overfitting. In the rest of this paper we will not deal with SVD any more, as regularized matrix factorization is obviously the better method for the task of recommender systems.

5. ONLINE UPDATES

In this section we provide methods for solving the new-user and new-item problem. That means it is assumed that an existing factorization, i.e. (W, H), is given and then a new rating comes in. We provide methods for updating the factorization (W, H) both in case for a user with a small rating profile and an item with a small profile. These methods will use the same update rules as in the last section, that means the derivations for training the model can be reused for online-updates. We will describe all methods in terms of the new-user problem. Of course everything can be applied to the new-item problem as well because KMF models are symmetric.

5.1 Training KMF models

Obviously, retraining the whole KMF model with the algorithm in fig. 2 after a new rating comes in is not applicable at all as it has complexity $O(|S| \cdot k \cdot i)$ where i is the number of iterations before early stopping is applied. In the Netflix use-case with $k = 40$, $i = 120$ and $|S| = 100,000,000$ this would lead to about 480 billion feature updates. In this paper, we propose an approximization method that updates the matrices of an existing model and that has complexity $O(|C(u,\cdot)| \cdot k \cdot i)$ where $C(u,\cdot)$ is the current profile of the user.

5.2 Approximating Updates

Let $\hat{R}_S$ denote the factorization calculated from the observed values S of R by the algorithm in fig. 2. Afterwards a new rating $r_{u,i}$ comes in. First of all, an exact reconstruction of $\hat{R}_{S\cup\{r_{u,i}\}}$ from $\hat{R}_S$ cannot be calculated as (1) in stochastic gradient descent the sequence of how ratings in S are visited is important and (2) information between iterations propagates through the matrices. The best that can be done is to approximate $\hat{R}_{S\cup\{r_{u,i}\}}$ from $\hat{R}_S$.

```
1: procedure UserUpdate(S, W, H, r_{u,i})
2:     S ← S ∪ {r_{u,i}}
3:     return UserRetrain(S, W, H, u)
4: end procedure

5: procedure UserRetrain(S, W, H, u*)
6:     initialize u*-th row in W
7:     repeat
8:         for r_{u,i} ∈ C(u*, ·) do
9:             for f ← 1, ..., f do
10:                w_{u,f} ← w_{u,f} − α ∂/∂w_{u,f} Opt(S, W, H)
11:            end for
12:        end for
13:    until Stopping criteria met
14:    return (W, H)
15: end procedure
```

Figure 3: Online updates for new-user problem.

We propose the algorithm UserUpdate (see fig. 3) for solving the new-user problem. This algorithm retrains the whole feature vector for this user and keeps all other entries in the matrix fixed. The motivation for this algorithm is the assumption that the model build from S and the model build from $S\cup\{r_{u,i}\}$ is mostly the same from a global perspective. But if user u is a new-user, his (local) features might change a lot from the new rating $r_{u,i}$. That is why we fully retrain this user and keep the other features fixed, as we assume them to be already the best guess.

The time complexity of UserRetrain is $O(|C(u,\cdot)|\cdot k\cdot i)$. In our evaluation we will see, that retraining users for each incoming rating becomes less important when the user profile increases. That is why $|C(u,\cdot)|$ usually is small. Besides time complexity and good quality (see evaluation), one of the major advantages of UserUpdate is that it is generic and applicable to any RKMF model. That means that no kernel specific algorithm or additional update formulas have to be designed.

5.3 Further Speedup

We have argued that retraining a user u on a new incoming rating $r_{u,i}$ is very important for a user with a small profile. With growing profiles, the update is less important (see fig. 5). In cases where an additional speedup is needed, one can apply several rules that determine if user-retrain can be skipped. We will present approaches that define a probability whether online-updates are performed or not. Depending on this probability the recommender can skip some online-update steps.

5.3.1 *Profile Size*

A reasonable assumption is that the larger the profile of a specific user is, the less important is retraining his profile on each new rating. Thus, the probability of retraining a user

```
1: procedure AddRating(S, W, H, r_{u,i})
2:     S ← S ∪ {r_{u,i}}
3:     return UpdateRating(S, W, H, r_{u,i})
4: end procedure

5: procedure RemoveRating(S, W, H, r_{u,i})
6:     S ← S \ {r_{u,i}}
7:     return UpdateRating(S, W, H, r_{u,i})
8: end procedure

9: procedure UpdateRating(S, W, H, r_{u,i})
10:    if P_u(train|r_{u,i}) > random then
11:        (W, H) ← UserRetrain(S, W, H, u)
12:    end if
13:    if P_i(train|r_{u,i}) > random then
14:        (W, H) ← ItemRetrain(S, W, H, i)
15:    end if
16:    return (W, H)
17: end procedure
```

Figure 4: General algorithm for online-updates.

could decay with increasing profile size:

$$P_u\left(\text{train}|r_{u,i}\right) = \gamma^{|C(u,\cdot)|}, \quad \gamma \in (0,1) \tag{23}$$

$$P_u\left(\text{train}|r_{u,i}\right) = \max\left(1, \frac{m}{|C(u,\cdot)|}\right), \quad m \in \mathbb{N}^+ \tag{24}$$

With (24) the average/expected runtime complexity is $O(m \cdot k \cdot i)$ and thus independent of the profile size.

5.3.2 Expected Impact

Another approach would be retraining if the new rating $r_{u,i}$ is expected to change the features. To measure the expected impact of the new rating on a user's profile one could use the error between the prediction of this rating and the true value. The larger this error is the more the rating is expected to change the features. To map the error to the interval $[0, 1]$ we propose to use a smooth monotonically increasing function like tanh:

$$P\left(\text{train}|r_{u,i}\right) = \tanh\left(\left(\hat{r}_{u,i} - r_{u,i}\right)^2\right) \tag{25}$$

5.4 General Update Problem

In the previous part of this section the algorithms and methods have been described from the perspective of the new-user problem. However, all methods and algorithms can be directly transfered to the new-item problem by exchanging users with items.

In general, a new rating $r_{u,i}$ might influence the features of both user u and item i. If both profiles are small, one could update both feature vectors. The outcome of this is a general update algorithm (fig. 4) that first performs an online update of the user's features and then an online update of the item's features. Both online updates are only executed if they are not pruned by one of the rules $P\left(\text{train}|r_{u,i}\right)$. This way, it is more likely to perform an update on a small profile than on a large one.

The proposed algorithm UpdateRating is not limited to new ratings, but can also be applied for removing ratings or changing a rating. A sketch for this task is found in fig. 4.

6. EVALUATION

In the evaluation we want to examine how our proposed online-update methods perform on real world problems. The goal of the update methods is (1) to approximate the quality of fully retraining as good as possible and (2) to have low runtime.

6.1 Evaluation Protocol

We simulate the new-user problem the following way:

1. Create a new-user-scenario:
   (a) Pick n% of the users and put them in U_t.
   (b) for each unknown user $u \in U_t$ do
      i. Split the ratings in $C(u,\cdot)$ in two disjoint sets T_u and V_u. The size of T_u is $\min\{m, \frac{|C(u,\cdot)|}{2}\}$ and so $V_u = C(u,\cdot) \setminus T_u$.
      ii. Remove all of the ratings $C(u,\cdot)$ from the set of known ratings S: $S \leftarrow S \setminus C(u,\cdot)$
2. Train the model on S: $(W, H) \leftarrow \text{Optimize}(S, W, H)$
3. Evaluate the new-user-scenario:
   for $j = 1, \ldots, m$ do
   (a) for each unknown user $u \in U_t$ do
      i. add one rating $r_{u,i} \in T_u$ to S
      ii. update the model: $(W, H) \leftarrow \text{UserUpdate}(S, W, H, r_{u,i})$
      iii. calculate error $\text{se}_u^j := E(V_u, W, H)$ on V_u
   (b) calculate $\text{rmse}^j := \sqrt{\frac{1}{\sum_{u:|T_u|\geq j} |V_u|} \cdot \sum_{u:|T_u|\geq j} \text{se}_u^j}$

With this protocol, one can examine how well a model can adapt to a user's taste with an increasing size of the user's rating profile. It is obvious that the protocol simulates a true online scenario where the rating profile of several new users increase from one up to m ratings. Please note that we update and evaluate the models after adding each single rating and not only after having added one rating for all users. Thus we can see the quality of the immediate response on a user's recommendations. The evaluation protocol for the new-item problem is symmetric to the protocol of new-users.

6.2 Datasets

In our experiments we evaluate on two movie recommendation datasets, i.e. Netflix[1] and Movielens[2]. Netflix is the state-of-the-art evaluation dataset for recommender systems with over 100 million ratings of about 480,000 users and 17,000 items. There are two Movielens datasets, we choose the larger one which contains about 1 million ratings by 6040 users and 3706 items.

6.3 Methodology

We run the proposed evaluation protocol (see section 6.1) for both the new-user problem and the new-item problem on Netflix and Movielens. For Movielens we choose $n = 10\%$ and for Netflix $n = 1\%$. The profile size for each user and

[1] http://www.netflixprize.com
[2] http://www.grouplens.org

item resp. is grown from 0 to $m = 50$ ratings as indicated in the evaluation protocol. Each Movielens experiment is run 10 times where the evaluation folds (the unknown users and items resp.) do not overlap (cv-style). For the Netflix dataset we rerun the experiment four times on non overlapping user/ item sets. We report the mean of the RMSE results as described in the evaluation protocol. For each experiment we run three types of RKMF, i.e. a linear kernel, a logistic kernel and a linear kernel with non-negative constraints. For Netflix each RKMF model has $k = 40$ features and for Movielens $k = 10$ (see table 2).

As we want to examine how good online-updates approximate a full retrain, we train and evaluate a second model (W^*, H^*) at the end of the j-loop (3.). This model (W^*, H^*) is generated by a full retrain, i.e. by $\text{OPTIMIZE}(S, W^*, H^*)$, on all ratings S. Thus (W^*, H^*) is the model that the online-updates try to approximate. To measure how good the approximation is in terms of prediction quality, we calculate the RMSE on the same evaluation set as the online updates, i.e. on $\bigcup_{u:|T_u|\geq j} V_u$. For Netflix we measure the quality of a full retrain only for $j = \{10, 25, 50\}$ because for Netflix fully retraining a second model in each iteration (m=50), for each kernel (3), each run (4) and both new-user and new-item problem (2) would have cost about $50 \cdot 4 \cdot 3 \cdot 2 \cdot 10$ hours = 500 days of CPU runtime. Nevertheless, our evaluation is computationally expensive. In total all experiments took 54 days of CPU runtime. All experiments were run on machines of the same type (CPU, RAM).

6.4 Quality

Figure 5 shows the evaluation of the new-user- and the new-item-problem on both Netflix and Movielens. As one can see, the error curves of the full retrain and the online updates of all three kernel methods are quite similar. Especially for non-negative RMF the online-updates are almost the same as the full retrain. For linear and logistic RMF the difference between online-update and full retrain are about 1% in the worst case. This shows that the proposed online-updates approximate the quality of fully retraining the model very well.

Secondly, all three factorization methods show promising quality results on the datasets. E.g. with non-negative RMF a user profile size of 7 ratings is enough to obtain a RMSE of below 95% which means beating the overall RMSE of Netflix's Cinematch system. With a user-profile size of 22 ratings, a mean RMSE of bellow 90% is achieved. For new items the prediction task on Netflix is harder as there are much more users than items. But also in this case 18 ratings on a new item's profile would beat the overall RMSE of Cinematch. And about 50 ratings are sufficient to break the 90% barrier.

Furthermore, the evaluation shows that for the new-user/ new-item problem it is important that models use new rating information. A static model without updates would not improve with new ratings and thus the error for all profile sizes would be as worse as at the beginning.

6.5 Speedup

As we have already seen, the quality of the approximation with online-updates is almost as good as a full retrain of the model. Now we compare the runtimes between online-updates and full retrains. Table 2 shows an overview of training times of several RKMF methods and the online-update costs. It is obvious, that online-updates are clearly faster than retraining the whole model. E.g. with linear RKMF on Netflix, retraining costs about 12 hours whereas an update as described in algorithm (3) takes only 0 to 15 ms depending on the profile size $|C(u, \cdot)|$ of the user/ the item. These empirical results match to the theoretical complexity that is $O(|C(u, \cdot)| \cdot k \cdot i)$ for online-updates instead of $O(|S| \cdot k \cdot i)$ for full retraining, where S is the set of all ratings whereas $C(u, \cdot)$ are the ratings of a specific user.

Dataset	Training	Regularized MF		
		linear	logistic	non-neg.
Movielens	Online Update	0-1 ms	0-10 ms	0-1 ms
	Retrain	18 s	3.5 min	25 s
	Features k	10	10	10
	# Iterations	30	270	50
Netflix	Online Update	0-15 ms	0-15 ms	0-18 ms
	Retrain	11.6 h	10.2 h	13.8 h
	Features k	40	40	40
	# Iterations	120	120	120

Table 2: Runtime of full retrain and runtime of proposed online updates wrt to profile size $C(u, \cdot)$ (for 'new-user' problem).

7. CONCLUSION

In this paper we have proposed the class of regularized kernel matrix factorization (RKMF) and a generic learning algorithm based on gradient descent. We have provided generic online-update methods for RKMF models that are based on the same gradient descent step that are also used for training the model. For static rating prediction, RMF are known to be one of the best models with regard to prediction quality. The drawback of RMF models is that once the factorization is computed, they cannot handle updates. Our online-updates make RMF and the more general RKMF class applicable for dynamic real-world scenarios where new users sign in and item catalogs are extended. We have shown that the proposed online-updates approximate the quality of fully retraining the model very well. On the other hand, both empirical and theoretical results for runtime complexity of online-updates make RKMF models feasible for huge datasets like Netflix and other dynamic real-world applications.

Acknowledgements

The authors gratefully acknowledge the partial co-funding of their work through the European Commission FP7 project MyMedia (www.mymediaproject.org) under the grant agreement no. 215006. For your inquiries please contact info@mymediaproject.org.

8. REFERENCES

[1] R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM*, pages 43–52. IEEE Computer Society, 2007.

[2] M. Brand. Fast online svd revisions for lightweight recommender systems. In *SIAM International Conference on Data Mining*, 2003.

[3] T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. In *ICDM*

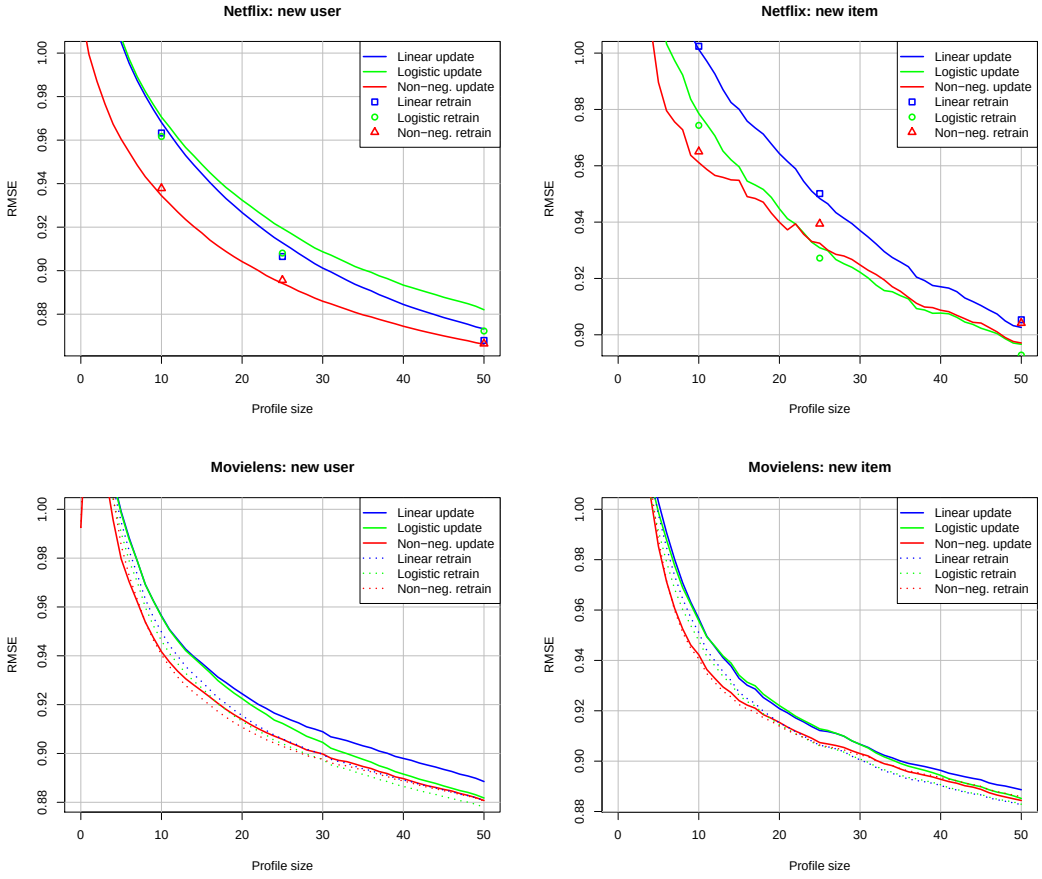

Figure 5: New-user/ new-item problem on Movielens and Netflix. Curves show the RMSE of online-updates (see protocol) compared to a full retrain.

'05: Proceedings of the Fifth IEEE International Conference on Data Mining, pages 625–628, Washington, DC, USA, 2005. IEEE Computer Society.

[4] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[5] S. Hauger, K. Tso, and L. Schmidt-Thieme. Comparison of recommender system algorithms focusing on the new-item and user-bias problem. In *Proceedings of 31th Annual Conference of the Gesellschaft fuer Klassifikation (GfKl), Freiburg*, 2007.

[6] T. Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, 2004.

[7] J. Kivinen, A. Smola, and R. Williamson. Online learning with kernels. *Signal Processing, IEEE Transactions on*, 52(8):2165–2176, Aug. 2004.

[8] M. Kurucz, A. A. Benczúr, and B. Torma. Methods for large scale svd with missing values. In *KDDCup 2007*, 2007.

[9] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, volume 20, 2008.

[10] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Proceedings of the 5th International Conference in Computers and Information Technology*, 2002.

[11] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *World Wide Web*, pages 285–295, 2001.

[12] A. Schein, A. Popescul, L. Ungar, and D. Pennock. Generative models for cold-start recommendations. In *Proceedings of the 2001 SIGIR Workshop on Recommender Systems*, 2001.

[13] L. Schmidt-Thieme. Compound classification models for recommender systems. In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM 2005)*, pages 378–385, 2005.

[14] G. Takacs, I. Pilaszy, B. Nemeth, and D. Tikk. On the gravity recommendation system. In *KDDCup 2007*, 2007.

[15] M. Wu. Collaborative filtering via ensembles of matrix factorization. In *KDDCup 2007*, pages 43–47, 2007.

[16] D. Zhang, Z.-H. Zhou, and S. Chen. Non-negative matrix factorization on kernels. In Q. Yang and G. I. Webb, editors, *PRICAI*, volume 4099 of *Lecture Notes in Computer Science*, pages 404–412. Springer, 2006.