Learning Recommender Systems with Adaptive Regularization

Steffen Rendle
Social Network Analysis
University of Konstanz
78457 Konstanz, Germany
steffen.rendle@uni-konstanz.de

ABSTRACT

Many factorization models like matrix or tensor factorization have been proposed for the important application of recommender systems. The success of such factorization models depends largely on the choice of good values for the regularization parameters. Without a careful selection they result in poor prediction quality as they either underfit or overfit the data. Regularization values are typically determined by an expensive search that requires learning the model parameters several times: once for each tuple of candidate values for the regularization parameters. In this paper, we present a new method that adapts the regularization automatically while training the model parameters. To achieve this, we optimize simultaneously for two criteria: (1) as usual the model parameters for the regularized objective and (2) the regularization of future parameter updates for the best predictive quality on a validation set. We develop this for the generic model class of Factorization Machines which subsumes a wide variety of factorization models. We show empirically, that the advantages of our adaptive regularization method compared to expensive hyperparameter search do not come to the price of worse predictive quality. In total with our method, learning regularization parameters is as easy as learning model parameters and thus there is no need for any time-consuming search of regularization values because they are found on-the-fly. This makes our method highly attractive for practical use.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Parameter Learning*

General Terms

Algorithms, Experimentation, Measurement, Performance

Keywords

Regularization, Matrix Factorization, Tensor Factorization

1. INTRODUCTION

Recommender systems are an important tool with many applications e.g. online-shopping, video rental, personalized web sites, etc. Recently factorization models became very popular due to their success in several challenges including the Netflix prize[1]. Most of the research in this field focuses on new and improved models, among them are matrix factorization [16, 12, 18], probabilistic latent semantic analysis (PLSA) [6], SVD++ [7], SoRec [11], time-variant matrix factorization [8] for collaborative filtering or Tucker decomposition [20, 10, 14] and pairwise interaction tensor factorization (PITF) [15] for tag recommendation. However, the success of all the mentioned factorization models depends largely on the right choice of their regularization values. The reason is that factorization models deal with a very large number of model parameters – e.g. many millions of model parameters for the winning approaches of the Netflix prize. This makes all factorization models very prone to overfitting, i.e. a low prediction error on the training data, but a high error on future test cases. L2-regularization is commonly applied where regularization parameters serve as a complexity control that can prevent overfitting. But if the regularization is chosen too large, the model will not learn (fit the training data) and if it is chosen too small the model will overfit. In both cases, the prediction quality will be poor and thus the choice of the regularization values is crucial. For factorization models, the common way to determine these regularization values is a time-consuming search over a set of candidates (often grid search [1]). For each candidate value, the model parameters are trained – a process that might take hours on large scale problems for each of the candidate values for the regularization. After learning several models, the regularization value with the best predictive quality on a withheld validation set is taken. This process is very time consuming because many models have to be learned.

Even though factorization models are very sensitive to the regularization values and the search for these values is very expensive, there is little research to overcome this issue. For the practical application of factorization models it is crucial to have a fast approach to determine regularization values. Even more because of the sensitivity it is desired that regularization values are chosen automatically such that wrong values cannot be set accidentally by inexperienced users. Finally, an automatic approach should work with several loss functions, be easy to implement and integrate into the prevalent existing algorithms.

In this work, we propose a new method that meets these

[1]`http://www.netflixprize.com/`

requirements. (1) It integrates the search for regularization values directly into the learning algorithm for model parameters. That means the model parameters have to be learned only once and during this process the regularization values are automatically found on-the-fly. This is done by adapting the regularization values such that in the next update step of model parameters the error on a validation set is minimized. (2) The runtime complexity of our proposed algorithm for adaptive regularization is the same as the one of standard stochastic gradient descent (SGD) which is the standard method for optimizing factorization models. This makes the algorithm applicable to large scale problems like recommender systems with millions of data records. (3) Compared to the common approach of expensive search over a set of candidate regularization values, our approach has also the advantage that it searches not just a predefined subspace of candidates, but the search space is unrestricted. Furthermore, models with many regularization parameters can be solved efficiently (e.g. one regularization parameter per factor dimension) which is not feasible with traditional grid search where the search space is exponential in the number of regularization parameters. (4) The proposed algorithm works with several losses and is easy to integrate into existing SGD algorithms. This makes our approach highly attractive for practical use of factorization models.

We analyze our proposed approach empirically on the Netflix and Movielens datasets and show that the prediction quality of adaptive regularization is comparable to models where the regularization values have been searched using expensive grid search.

2. RELATED WORK

2.1 Validation Set Based Approaches

Usually regularization values are determined by a search for optimal values using a withheld validation set. There are different approaches how the search is performed. The most common one is to search a space of candidate values for regularization and to learn a separate model for each candidate. Using the withheld validation set, the model for each regularization value can be evaluated and the best one is chosen. For factorization models, grid search [1] is usually performed where a predefined grid of candidates is set up and on each candidate a model is trained and evaluated (see fig. 1). Even though such an uninformed search is very expensive in terms of runtime, it is the dominating method for factorization models (e.g. [7, 15, 13]).

For other models, there are more sophisticated methods where an informed search is performed. Examples are Chapelle et al. [2] for SVMs and Larsen et al. [9] for neural networks. Both guide the search for regularization parameters by gradient methods: Chapelle et al. optimize the kernel parameters and Larsen et al. the weight decay. Although both work with different models, similar to us they alternate between optimizing regularization/ weight decay and model parameters. But unlike our approach both do not intertwine the adaptive regularization with the learning of model parameters. Instead they iterate between a complete training of model parameters and a complete training of regularization hyperparameters. Even though such an informed search reduces the number of models that have to be learned, still several models have to be fitted. And unlike grid search, learning these several models cannot be parallelized trivially. In contrast to this, our approach integrates both steps and requires only to learn a single model. Besides a more concise learning of model and regularization parameters, our intertwined approach benefits from its ongoing adaptation of both sets of parameters because the regularization can be adapted to the current state of the learning: At first, when the model parameters are far from convergence, regularization will be (almost) zero, increasing to its final level when the model parameters (start to) converge.

Also for support vector machines (SVMs), Hastie et al. [5] have proposed regularization paths that can be used for speeding up the search for the optimal regularization value. Regularization paths allow to find all possible solutions as a function of the regularization value and can be computed in the same complexity as fitting a single model. Like our approach, the runtime for finding the regularization values is roughly doubled compared to standard learning with predefined regularization values. Besides the different model of SVMs and factorization models, we deal with a large number of regularization parameters (see fig. 6).

In total, to the best of our knowledge our approach is the first one that allows to adapt regularization parameters efficiently for the prevalent model and learning method in recommender systems: i.e. factorization models with stochastic gradient descent learning.

2.2 Hierarchical Bayesian Approaches

Another direction to find regularization parameters are Bayesian approaches [3] that do not work with a validation set but use a hierarchical model with hyperpriors on the prior distribution (i.e. regularization). This allows to group parameters like we do and due to hyperpriors partial pooling is achieved which decreases overfitting. Salakhutdinov and Mnih [17] proposed a hierarchical Bayesian model for matrix factorization and Xiong et al. [21] for a three mode PARAFAC model but both are only applicable for least squares loss. Furthermore Bayesian approaches are typically optimized with Markov Chain Monte Carlo (MCMC) whereas most factorization models are optimized with SGD (e.g. [12, 11, 7, 8, 15]) due to its simplicity and low computational complexity. Our method allows to be integrated with little effort into such existing (S)GD methods for factorization models and work for different losses.

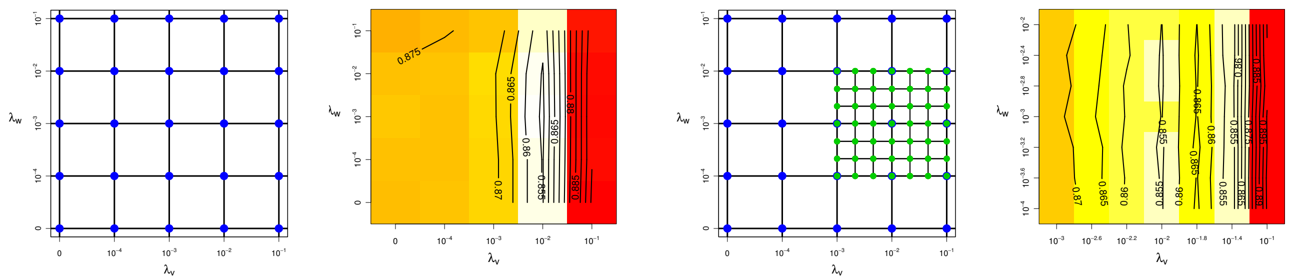
To summarize, our method integrates the search for regularization values directly into the learning algorithm of the model parameters, it allows to learn many regularization values and can be applied for several losses and finally is very easy to implement. We show this for Factorization Machines which generalize a wide variety of factorization models.

3. PROBLEM SETTING

Before introducing our method we shortly recapitulate some of the most successful factorization models for recommender systems and discuss how they are regularized and learned.

3.1 Factorization Models

Matrix Factorization (MF) [19] can be applied to settings with two variables over categorical domains, e.g. users U and items I in a recommender system. The matrix factorization model finds a latent description (a vector of k factors) for each entity, e.g. $\mathbf{v}_u \in \mathbb{R}^k$ for a user $u \in U$. The model



(a) Initial Grid (b) RMSE Heatmap (c) Refined Grid (d) RMSE Heatmap

Figure 1: Typical grid search for finding regularization values (here $\lambda \in \mathbb{R}^2$): (a) Grid of candidates $\Lambda = \{(0,0),(0,10^{-1}),(10^{-1},0),\ldots\}$. (b) For each candidate $\lambda \in \Lambda$, the model parameters are trained on S_T and their quality is measured on a withheld validation set S_V. (c) Finer grid in the neighborhood of the best value (here $\lambda = (0.01, 0.001)$). (d) For each of the new candidates model parameters are trained and tested.

equation corresponds to the dot product of two entities (e.g. a user u and an item i):

$$\hat{y}(u,i) := \langle \mathbf{v}_u, \mathbf{v}_i \rangle, \quad \mathbf{V} \in \mathbb{R}^{(U \cup I) \times k} \tag{1}$$

In this model, the latent factors $\mathbf{V}$ are the model parameters that have to be learned. Often bias terms are added to the matrix factorization model [12]: A global bias w_0 that usually holds the mean over the target. And secondly, bias terms for each entity are added, e.g. w_i for the item i. The matrix factorization model with such additional biases reads:

$$\hat{y}(u,i) := w_0 + w_i + w_u + \langle \mathbf{v}_u, \mathbf{v}_i \rangle. \tag{2}$$

Here $w_0 \in \mathbb{R}$ and $\mathbf{w} \in \mathbb{R}^{U \cup I}$ are additional model parameters.

For problems such as tag recommendation or context-awareness where more than two variables are involved, e.g. $C_1, \ldots, C_j$, tensor factorization models can be applied. Examples are Tucker Decomposition (TD) [20] or Parallel Factor Analysis (PARAFAC) [4]. For tag recommender systems, a model using pairwise interactions (PITF) [15] has been proposed and shown to outperform TD and PARAFAC on this task. The PITF model for j variables and biases can be written as:

$$\hat{y}(c_1,\ldots,c_j) := w_0 + \sum_{l=1}^{j} w_{c_l} + \sum_{l_1=1}^{j} \sum_{l_2>l_1}^{j} \langle \mathbf{v}_{c_{l_1}}, \mathbf{v}_{c_{l_2}} \rangle \tag{3}$$

where the model parameters are $w_0 \in \mathbb{R}$, $\mathbf{w} \in \mathbb{R}^{C_1 \cup \ldots C_j}$ and $\mathbf{V} \in \mathbb{R}^{(C_1 \cup \ldots C_j) \times k}$.

There are lots of specialized models that extend general factorization models for special cases. E.g. SVD++ [7] for movie recommendation that adds implicit feedback indicators to MF:

$$\hat{y}(u,i) := w_0 + w_i + w_u + \langle \mathbf{v}_u, \mathbf{v}_i \rangle + \frac{1}{\sqrt{N(u)}} \sum_{l \in N(u)} \langle \mathbf{v}_i, \mathbf{v}_l \rangle$$

where $N(u)$ is the set of all other items a user u has ever selected (e.g. rented or viewed). Or a k-nearest-neighbor (KNN) model [7] where neighborhood weights are treated as model parameters:

$$\hat{y}(u,i) := w_0 + w_i + w_u + \frac{1}{\sqrt{R(u)}} \sum_{l \in R(u)} w_{i,l,1}(y_{u,l} - b_{u,l}) + \frac{1}{\sqrt{N(u)}} \sum_{l \in N(u)} w_{i,l,2} \tag{4}$$

Here $R(u)$ is the set of items where the ratings are known.

The models described so far can be generalized to Factorization Machines (FM) [13] which work with real-valued input vectors $\mathbf{x} \in \mathbb{R}^p$. It has been shown [13] that FMs can mimic several factorization models among them are (biased) MF, SVD++, KNN and PITF, just by feature engineering, i.e. definition of the input vectors $\mathbf{x}$ using binary indicator variables (e.g. one variable for each user, each item, etc.). A FM models all nested d-way interactions between any combination of d variables with the target[2]. A 2-way FM is defined as:

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{l=1}^{p} w_l\, x_l + \sum_{l_1=1}^{p} \sum_{l_2>l_1}^{p} \langle \mathbf{v}_{l_1}, \mathbf{v}_{l_2} \rangle\, x_{l_1}\, x_{l_2} \tag{5}$$

with model parameters $w_0 \in \mathbb{R}$, $\mathbf{w} \in \mathbb{R}^p$ and $\mathbf{V} \in \mathbb{R}^{p \times k}$. The model equation of FMs can be computed efficiently because eq. (5) is equivalent to:

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{l=1}^{p} w_l\, x_l + \frac{1}{2} \sum_{f=1}^{k} \left(\left(\sum_{l=1}^{p} v_{l,f}\, x_l \right)^2 - \sum_{l=1}^{p} v_{l,f}^2\, x_l^2 \right)$$

Due to the expressiveness of FMs, we develop our adaptive regularization approach for this model class. This leads to self-adaptive approaches of all subsumed models.

3.2 Optimization

Before a model can be applied, the values of the model parameters Θ (e.g. $\Theta = (w_0, \mathbf{w}, \mathbf{V})$ for FMs) have to be learned from a set of observed data S. Let S be a set of observed cases $(\mathbf{x}, y) \in \mathbb{R}^{p+1}$, where for classification $y \in \{-1, 1\}$.

3.2.1 Optimization Tasks

Optimality of model parameters is usually defined with a loss function l where the task is to minimize the sum of

[2]In this work, we will deal only with $d = 2$.

losses over the data S:

$$\textsc{Opt}(S) := \operatorname*{argmin}_{\Theta} \sum_{(\mathbf{x},y)\in S} l(\hat{y}(\mathbf{x}|\Theta), y). \tag{6}$$

Note that we add the model parameters Θ to the model equation and write $\hat{y}(\mathbf{x}|\Theta)$ when we want to stress that $\hat{y}$ depends on a certain choice of Θ. Depending on the task, the loss function can be chosen. E.g. for regression, least squares loss:

$$l^{\mathrm{LS}}(y_1, y_2) := (y_1 - y_2)^2, \tag{7}$$

or for binary classification:

$$l^{\mathrm{C}}(y_1, y_2) := -\ln \sigma(y_1\, y_2) \tag{8}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid/ logistic function.

3.2.2 Regularization

All the presented factorization models usually have a huge number of model parameters Θ – especially if k is chosen large enough. This makes them prone to overfitting. To overcome this, typically L2 regularization is applied which can be motivated by maximum-margin [19], Tikhonov regularization or MAP estimator with Gaussian priors [18]:

$$\textsc{OptReg}(S, \lambda) := \operatorname*{argmin}_{\Theta} \left(\sum_{(\mathbf{x},y)\in S} l(\hat{y}(\mathbf{x}|\Theta), y) + \sum_{\theta\in\Theta} \lambda_\theta \theta^2 \right). \tag{9}$$

Here $\lambda_\theta \in \mathbb{R}_+$ is the regularization parameter for the model parameter θ. The major issue of such regularization approaches is that their success depends largely on the choice of the value of the regularization parameter λ. If λ is chosen too large, the model cannot fit the training data and thus does not learn. If λ is chosen too small, the model overfits the training data which means the training error will be small but the error on future test cases will be large. As mentioned before, overfitting is easy to achieve with models of high expressiveness, i.e. lots of model parameters, like in factorization models. That means regularization succeeds only if λ is chosen correctly.

Multiple Priors.

In fact this problem even gets more difficult, as one does not deal with only one regularization parameter $\lambda \in \mathbb{R}_+$ that is shared by all model parameters, but instead model parameters are grouped and each group has an independent regularization parameter λ. So if there are c groups, λ is a c-dimensional vector: $\lambda \in \mathbb{R}_+^c$. A simple example for FMs would be that the global bias w_0 has its own regularization parameter λ_0, all bias parameters $w_1, \ldots, w_p$ share one regularization parameter λ_w and each level f of the k dimensions of the factors $\mathbf{V}$ share one parameter, e.g. λ_f for $v_{1,f}, \ldots, v_{p,f}$. In total this leads to $c = k + 2$ regularization parameters. Such a grouping is typically chosen in advance because it stems from the structure of the factorization model. In the following, we will allow several regularization parameters and we denote with λ_θ the regularization parameter that belongs to the model parameter θ.

3.2.3 Learning Algorithm

For optimizing complex models with respect to $\textsc{OptReg}$, usually iterative algorithms are applied. Especially popular for factorization models are gradient descent (GD) methods because they are simple and are easily applicable to a wide variety of loss functions. This method has been shown to solve large problems e.g. the Netflix prize [12, 7, 8] efficiently. The idea of GD is that given a parameter setting Θ^t, the model parameters can be improved to Θ^{t+1} by performing a step in the opposite direction of the gradient of the loss function. To speed up learning, stochastic GD (SGD) is used where a case $(\mathbf{x}, y)$ is randomly drawn from the observed data S and the update is performed only with respect to this case:

$$\theta^{t+1} = \theta^t - \alpha \left(\frac{\partial}{\partial \theta^t} l(\hat{y}(\mathbf{x}|\Theta^t), y) + 2\,\lambda\,\theta^t \right) \tag{10}$$

where α is the learning rate or step size. GD works with any differentiable loss, e.g. for least squares, the gradient is

$$\frac{\partial}{\partial\theta} (\hat{y}(\mathbf{x}|\Theta) - y)^2 = 2\, (\hat{y}(\mathbf{x}|\Theta) - y)\, \frac{\partial}{\partial\theta} \hat{y}(\mathbf{x}|\Theta) \tag{11}$$

or for classification

$$\frac{\partial}{\partial\theta} - \ln \sigma\,(\hat{y}(\mathbf{x}|\Theta)\, y) = (\sigma\,(\hat{y}(\mathbf{x}|\Theta)\, y) - 1)\; y\, \frac{\partial}{\partial\theta} \hat{y}(\mathbf{x}|\Theta). \tag{12}$$

Finally, the gradients of the model equation for factorization machines are [13]:

$$\frac{\partial}{\partial\theta} \hat{y}(\mathbf{x}) = \begin{cases} 1, & \text{if } \theta \text{ is } w_0 \\ x_i, & \text{if } \theta \text{ is } w_i \\ x_i \left(\sum_{j=1} x_j\, v_{j,f} - x_i\, v_{i,f} \right), & \text{if } \theta \text{ is } v_{i,f} \end{cases} \tag{13}$$

Given these formulas, a SGD algorithm for factorization machines [13] can be sketched (see figure 2). Note that this algorithm also can learn all subsumed model classes (see section 3.1) like matrix factorization or PITF.

```
1:  procedure SolveOptReg($S, \lambda$)
2:      $w_0 \leftarrow 0$
3:      $\mathbf{w} \leftarrow (0, \ldots, 0)$
4:      $\mathbf{V} \sim \mathcal{N}(0, \sigma)$
5:      repeat
6:          for $(\mathbf{x}, y) \in S$ do
7:              $\partial_0 \leftarrow \frac{\partial}{\partial w_0} l(\hat{y}(\mathbf{x}|\Theta), y)$
8:              $w_0 \leftarrow w_0 - \alpha\ (2\,\lambda_0\, w_0 + \partial_0)$
9:              for $i \in \{1, \ldots, p\} \wedge x_i \neq 0$ do
10:                 $\partial_i \leftarrow \frac{\partial}{\partial w_i} l(\hat{y}(\mathbf{x}|\Theta), y)$
11:                 $w_i \leftarrow w_i - \alpha\ (2\,\lambda_w\, w_i + \partial_i)$
12:                 for $f \in \{1, \ldots, k\}$ do
13:                     $\partial_{i,f} \leftarrow \frac{\partial}{\partial v_{i,f}} l(\hat{y}(\mathbf{x}|\Theta), y)$
14:                     $v_{i,f} \leftarrow v_{i,f} - \alpha\ (2\,\lambda_f\, v_{i,f} + \partial_{i,f})$
15:                 end for
16:             end for
17:         end for
18:     until stopping criterion is met
19:     return $\Theta := (w_0, \mathbf{w}, \mathbf{V})$
20: end procedure
```

Figure 2: Learning model parameters Θ of a FM by stochastic gradient descent for OptReg (eq. 9) given a regularization vector $\lambda \in \mathbb{R}_+^c$.

4. ADAPTIVE REGULARIZATION

In the following, we show how the search for optimal regularization values can be integrated directly into SolveOptReg (fig. 2). We start with the definition of the optimization task for regularization values and develop a fast and simple learning algorithm.

4.1 Optimization Task for Regularization

Typically, optimal regularization values $\lambda^* \in \mathbb{R}_+^c$ for factorization models are determined using a holdout method. Therefore the available data S is split into two disjoint data sets: $S = S_T \cup S_V$. On S_T, the model parameters are optimized for the regularized loss objective (OptReg, eq. 9) given a regularization constant λ. Using the validation set S_V, the quality of the learned model parameters can be estimated. As the training S_T and validation S_V sets are disjoint, the quality on S_V gives a reliable estimate of the future success of the parameter setting – as long as the size of the validation set is sufficiently large and the future data is generated by the same process as the validation data S_V. Working with disjoint training and validation sets is one of the core principles in machine learning. In our case, the task is to find the regularization values λ^* that lead to the lowest error on the validation set

$$\lambda^* := \operatorname*{argmin}_{\lambda \in \mathbb{R}_+^c} \sum_{(\mathbf{x},y)\in S_V} l\left(\hat{y}(\mathbf{x}|\textsc{OptReg}(S_T,\lambda)), y\right). \qquad (14)$$

This is a nested optimization task where the outermost objective is to determine the best regularization values λ^* that minimize the loss on the validation set S_V. This loss on the validation set depends not directly on λ. Instead it depends on the model equation $\hat{y}$ which itself depends only on the model parameters. However, the model parameters are the solution of $\Theta^*|\lambda = \textsc{OptReg}(S_T, \lambda)$, i.e. the regularized optimization criterion on the training data S_T, thus these optimal model parameters depend on λ. So in total, we are looking for a λ^* such that the corresponding optimal model parameters $\Theta^*|\lambda^*$ with respect to the regularized loss on S_T lead to a minimal error on the validation set S_V.

Alternating Optimization.

A straightforward idea is to use an alternating optimization, i.e. starting with an initial guess of (Θ, λ) and alternate between improving Θ while λ is fixed and improving λ while Θ is fixed. This process is repeated up to convergence. For the first task, i.e. optimizing Θ for a given λ, we already have a solution (alg. SolveOptReg). But the second task (eq. 14) is not trivial. If Θ is kept constant here, i.e. $\textsc{OptReg}(S_T, \lambda)$ is replaced by the current model parameters Θ^t, then the optimization task (eq. 14) reads

$$\lambda^*|\Theta^t := \operatorname*{argmin}_{\lambda \in \mathbb{R}_+^c} \sum_{(\mathbf{x},y)\in S_V} l\left(\hat{y}(\mathbf{x}|\Theta^t)), y\right). \qquad (15)$$

In this task the right hand side is independent of λ. This means the gradient vanishes

$$\frac{\partial}{\partial \lambda} L(S_V, \Theta^t) = \frac{\partial}{\partial \lambda} \sum_{(\mathbf{x},y)\in S_V} l(\hat{y}(\mathbf{x}|\Theta^t), y) = 0 \qquad (16)$$

and thus every value $\lambda \in \mathbb{R}$ would be optimal w.r.t. equation (15) which is obviously not the right approach to solve the original task (eq. 14).

4.2 Optimizing Future Model Parameters

Alternating optimization using eqs. (9) and (14) seems to be impossible as the gradient of any λ vanishes when Θ is kept constant. The problem is that λ does not appear explicitly in the optimization formula (eq. 15). However if we remember the role of lambda in eq. (14), it can be seen that λ steers the search for the model parameters Θ on S_T which are then used in S_V. This can also be seen in the SGD update rule for the model parameters (eq. 10) where λ appears. That means with a simple trick, we can reformulate the model equation such that it depends explicitly on λ: It is known that after the next update on θ, the value of θ will be the one of eq. (10). This future value θ^{t+1} depends on λ. So the future model equation after updating the model parameters Θ can be written as:

$$\hat{y}(\mathbf{x}|\Theta^{t+1}) = w_0^{t+1} + \sum_{l=1}^{p} w_l^{t+1}\, x_l + \sum_{l_1=1}^{p} \sum_{l_2>l_1}^{p} \langle \mathbf{v}_{l_1}^{t+1}, \mathbf{v}_{l_2}^{t+1} \rangle\, x_{l_1}\, x_{l_2} \qquad (17)$$

That means instead of expressing the model equation with the current parameters Θ^t, it can be formulated to depend on the parameters after the next update Θ^{t+1}. Replacing each θ^{t+1} in eq. (17) with the right hand side of eq. (10), results in a model equation which depends on λ:

$$\begin{aligned}
\hat{y}(\mathbf{x}|\Theta^{t+1}) &= w_0^t - \alpha \left(\frac{\partial l(\hat{y}(\mathbf{x}|\Theta^t), y)}{\partial w_0^t} + 2\,\lambda_0\, w_0^t \right) \\
&\quad + \sum_{l=1}^{p} x_l \left(w_l^t - \alpha \left(\frac{\partial l(\hat{y}(\mathbf{x}|\Theta^t), y)}{\partial w_l^t} + 2\,\lambda_w\, w_l^t \right) \right) \\
&\quad + \sum_{l_1=1}^{p} \sum_{l_2>l_1}^{p} \sum_{f=1}^{k} \Bigg[x_{l_1} \left(v_{l_1,f}^t - \alpha \left(\frac{\partial l(\hat{y}(\mathbf{x}|\Theta^t), y)}{\partial v_{l_1,f}^t} + 2\,\lambda_f\, v_{l_1,f}^t \right) \right) \\
&\qquad x_{l_2} \left(v_{l_2,f}^t - \alpha \left(\frac{\partial l(\hat{y}(\mathbf{x}|\Theta^t), y)}{\partial v_{l_2,f}^t} + 2\,\lambda_f\, v_{l_2,f}^t \right) \right) \Bigg] \qquad (18)
\end{aligned}$$

This future model equation $\hat{y}(\mathbf{x}|\Theta^{t+1})$ can be used in eq. (14), which results in the following task:

$$\lambda^*|\Theta^t := \operatorname*{argmin}_{\lambda \in \mathbb{R}_+^c} \sum_{(\mathbf{x},y)\in S_V} l\left(\hat{y}(\mathbf{x}|\Theta^{t+1}), y\right). \qquad (19)$$

This optimization task answers the question: *What is the best value of λ such that the next update on Θ generates the smallest error on the validation set?* Now alternating between eqs. (9) and (19) is possible.

In general, this idea can be used for any iterative optimization algorithm where the update on the model parameter explicitly depends on the regularization value. In the following, we show how this can be done efficiently for SGD.

Gradients.

As the model parameters are usually optimized with SGD, we propose to use SGD as well for optimizing λ w.r.t. the task of eq. (19). The SGD-update for λ given a case $(\mathbf{x}, y) \in S_V$ is

$$\lambda^{t+1} = \lambda^t - \alpha \frac{\partial}{\partial \lambda} l\left(\hat{y}(\mathbf{x}|\Theta^{t+1}), y\right) \qquad (20)$$

where the gradients of the loss function are the same as in eqs. (11) and (12), e.g. for least squares it reads

$$\frac{\partial}{\partial \lambda}(\hat{y}(\mathbf{x}|\Theta^{t+1}) - y)^2 = 2\,(\hat{y}(\mathbf{x}|\Theta^{t+1}) - y)\frac{\partial}{\partial \lambda}\hat{y}(\mathbf{x}|\Theta^{t+1}),$$

and for logistic classification loss:

$$\frac{\partial}{\partial \lambda} - \ln \sigma\left(\hat{y}(\mathbf{x}|\Theta^{t+1})\, y\right) = \left(\sigma\left(\hat{y}(\mathbf{x}|\Theta^{t+1})\, y\right) - 1\right)\, y\, \frac{\partial}{\partial \lambda}\hat{y}(\mathbf{x}|\Theta^{t+1}).$$

Finally, the gradients of the future model equation (eq. 17) w.r.t. λ have to be calculated. As the update equation of L2-regularized gradient descent is a linear function in λ and FMs are linear functions in each single model parameter, this is always easy to derive. We show this now for the example where there are k+2 regularization parameters: one parameter (λ_0) for w_0, one (λ_w) that is shared by $w_1, \ldots, w_p$ and for each of the factorization dimensionality $f = 1, \ldots, k$, there is one regularization parameter λ_f which is shared by $v_{1,f}, \ldots, v_{p,f}$. First, the gradient of λ_0 w.r.t. the future model equation (eq. 18) is

$$\frac{\partial}{\partial \lambda_0}\hat{y}(\mathbf{x}|\Theta^{t+1}) = -2\,\alpha\, w_0^t,$$

for λ_w the gradient reads

$$\frac{\partial}{\partial \lambda_w}\hat{y}(\mathbf{x}|\Theta^{t+1}) = -2\,\alpha \sum_{i=1}^{p} w_i^t x_i$$

and for each λ_f

$$\frac{\partial}{\partial \lambda_f}\hat{y}(\mathbf{x}|\Theta^{t+1}) = -2\alpha\left[\sum_{i=1} x_i v_{i,f}^{t+1} \sum_{j=1} x_j v_{j,f}^t - \sum_{j=1} x_j^2 v_{j,f}^{t+1} v_{j,f}^t\right]$$

Note that the derivative of factors depends both on current $v_{i,f}^t$ and future $v_{i,f}^{t+1}$ parameters (see eq. 10).

4.3 Alternating SGD Algorithm

These gradients allow to derive an alternating SGD algorithm for eq. (14). Instead of solving OptReg (eq. 9) and the optimization for λ (eq. 19) one after another, they can be solved simultaneously by alternating single SGD-steps: i.e. drawing one case from S_T and perform an update step (eq. 10) on the model parameters Θ and then draw one case from S_V and perform an update on λ with eq. (20). This means while learning Θ, the regularization parameters are adapted.

Fast Computation.

Now only one issue in the optimization of λ remains. Each parameter θ^{t+1} in the future model eq. (18) depends on the gradient $\frac{\partial l(\hat{y}(\mathbf{x}|\Theta^t),y)}{\partial \theta^t}$ of the training case $(\mathbf{x}, y) \in S_T$ which will update θ next. Instead of computing this gradients for every θ^{t+1} on every λ-update, this value can be approximated by the last gradient of θ which we denote by ∂_θ. That means when a model parameter is updated in a θ-update step, its gradient is stored in ∂_θ and can be used later for the corresponding λ-updates where the approximated future model parameter $\tilde{\theta}^{t+1}$ is

$$\tilde{\theta}^{t+1} := \theta^t - \alpha\left(\partial_\theta + 2\,\lambda\,\theta^t\right) \approx \theta^{t+1} \qquad (21)$$

This approximation $\tilde{\theta}^{t+1}$ is used for the λ-steps, i.e. in eq. (18) and its derivatives (eq. 20). The size of the approximating error depends on the size of the change in the model

```
1: procedure SolveOptAdaptiveReg($S_T, S_V$)
2:     $w_0 \leftarrow 0$
3:     $\mathbf{w} \leftarrow (0, \ldots, 0)$
4:     $\mathbf{V} \sim \mathcal{N}(0, \sigma)$
5:     $\lambda \leftarrow (0, \ldots, 0)$
6:     $\partial \leftarrow (0, \ldots, 0)$
7:     repeat
8:         for $(\mathbf{x}, y) \in S_T$ do
9:             $\partial_0 \leftarrow \frac{\partial}{\partial w_0} l(\hat{y}(\mathbf{x}|\Theta), y)$
10:            $w_0 \leftarrow w_0 - \alpha\,(2\,\lambda_0\, w_0 + \partial_0)$
11:            for $i \in \{1, \ldots, p\} \wedge x_i \neq 0$ do
12:                $\partial_i \leftarrow \frac{\partial}{\partial w_i} l(\hat{y}(\mathbf{x}|\Theta), y)$
13:                $w_i \leftarrow w_i - \alpha\,(2\,\lambda_w\, w_i + \partial_i)$
14:                for $f \in \{1, \ldots, k\}$ do
15:                    $\partial_{i,f} \leftarrow \frac{\partial}{\partial v_{i,f}} l(\hat{y}(\mathbf{x}|\Theta), y)$
16:                    $v_{i,f} \leftarrow v_{i,f} - \alpha\,(2\,\lambda_f\, v_{i,f} + \partial_{i,f})$
17:                end for
18:            end for
19:            $(\mathbf{x}', y') \sim S_V$    ▷ draw a case from valid. set
20:            $\lambda_0 \leftarrow \max\left(0, \lambda_0 - \alpha \frac{\partial}{\partial \lambda_0} l\left(\hat{y}(\mathbf{x}'|\tilde{\Theta}), y'\right)\right)$
21:            $\lambda_w \leftarrow \max\left(0, \lambda_w - \alpha \frac{\partial}{\partial \lambda_w} l\left(\hat{y}(\mathbf{x}'|\tilde{\Theta}), y'\right)\right)$
22:            for $f \in \{1, \ldots, k\}$ do
23:                $\lambda_f \leftarrow \max\left(0, \lambda_f - \alpha \frac{\partial}{\partial \lambda_f} l\left(\hat{y}(\mathbf{x}'|\tilde{\Theta}), y'\right)\right)$
24:            end for
25:        end for
26:    until stopping criterion is met
27:    return $\Theta := (w_0, \mathbf{w}, \mathbf{V})$
28: end procedure
```

Figure 3: Integrating the search for regularization parameters into the learning algorithm for model parameters. This way, adaptive regularization learns the regularization values automatically while learning the model parameters. See section 4.2 for the gradients.

parameters between updates. With a small step size α the approximation error is also small. Furthermore the approximation error decreases and finally vanishes in convergence of model parameters and regularization.

Algorithm.

The final algorithm is very simple and requires only little extensions to the standard SGD algorithm.

1. Update model parameters as usual. Store for each model parameter the last gradient step, i.e. $\partial_\theta \leftarrow \frac{\partial l(\hat{y}(\mathbf{x}|\Theta^t),y)}{\partial \theta^t}$. As this value is anyway computed during the SGD on the model parameters, this has no extra overhead in runtime but doubles the number of parameters to store.

2. After each update on the model parameters, make an SGD-update on the regularization values (eq. 20) using the approximated future model parameter (eq. 21).

Figure 3 shows the extended SGD algorithm that adapts regularization during parameter learning. To shorten notation and because within the algorithm it does not make sense to keep track of time t, we replaced θ^t with θ and $\tilde{\theta}^{t+1}$ with $\tilde{\theta}$.

Table 1: Datasets statistics about the number of users, items and number of instances/ rows in the train, validation and test set.

Dataset	# users	# items	size train	size validation	size test
Movielens	6,040	3,900	792,164	8,002	200,042
Netflix	480,189	17,700	100,340,104	140,403	1,408,789

4.4 Properties of Adaptive Regularization

To summerize, adaptive regularization has the following properties:

1. The runtime complexity of adaptively regularized model parameter learning (fig. 3) is the same as the one of standard learning (fig. 2) where λ is fixed. The only computational overhead of adaptive regularization compared to the basic algorithm are the additional gradient steps on λ. In each update, computing the gradients for all regularization parameters has the same complexity as computing the gradients of the model parameters, i.e. $O(k\, m(\mathbf{x}))$[3]. As we propose to do a regularization update for each model update, the runtime is roughly doubled. Compared to this, learning hyperparameters with non-integrated approaches (e.g. grid-search, see fig. 1 or informed search) is much more time-consuming because it means learning one model for each of the candidates.
2. The proposed algorithm adapts the regularization values automatically that means there is no danger that an inexperienced user sets wrong values to these critical parameters.
3. Most optimization algorithms for factorization models are based on SGD. Our proposed approach is easy to integrate into such SGD algorithms and is applicable to many loss functions.

5. EVALUATION

In the evaluation, we investigate the predictive performance of our proposed adaptive regularization method (SGDA, fig. 3) compared to the standard approach of SGD (fig. 2) with expensive grid search. Moreover, we provide deeper insights into the process of learning optimal hyperparameters λ using SGDA.

5.1 Methodology

5.1.1 *Experimental Reproducibility*

For sake of experimental reproducibility we evaluate on two well-known datasets: Movielens 1M (ML1M) and Netflix (see table 1). We split Movielens into a train and test set using a random 80/20-split. Concerning Netflix we use the train (incl. *probe*) and quiz (the *leaderboard* set) split from the challenge as train and test data sets – this allows comparison to the results reported on the Netflix leaderboard. The experiments are conducted using the publicly available software LIBFM[4] that we extend with adaptive SGDA learning. We use the factorization machine of LIBFM to mimic biased matrix factorization (MF) (see eq. 2) and the KNN model with learned similarities (eq. 4) with implicit feedback (KNN++) and without (KNN) – we run it with a neighborhood size of 256 following the pruning protocol of Koren [7].

[3]Where $m(\mathbf{x})$ is the number of non-zero values in $\mathbf{x}$.
[4]`http://www.libfm.org/`

5.1.2 *Protocol*

For SGDA, we run SOLVEOPTADAPTIVEREG (fig. 3) to predict the test set. For ML1M we use a random subsample of 1% of the training data for validation S_V. For Netflix we use a 10% subsample of the Netflix *probe* set as validation set S_V because *probe* is representative for test[5]. As SGDA finds the regularization values automatically, there is no need to specify any regularization hyperparameter in advance. For determining the number of iterations, we use the predicted quality on the validation set S_V.

The setup for SGD is more complicated because regularization parameters have to be found. As the search for individual regularization parameters for each factor layer $f \in \{1, \ldots, k\}$ using grid search is not feasible, we restrict the search space and only search for λ_w and one shared λ_v for all factor layers. For Movielens, we apply a grid search like the one in fig. 1 using the 1% validation sample; the best values are $\lambda_w = 0$ and $\lambda_v = 0.01$ for the MF model. For the much larger and more time-consuming Netflix dataset, we rely on the hyperparameter search from [7]: for KNN we use $\lambda_w = 0.002$ and for MF $\lambda_w = \lambda_v = 0.04$ which was reported for the related SVD++ model. In favor of SGD, after regularization values have been searched, we retrain the models on the whole dataset (i.e. merging S_V and S_T) with the best λ found.

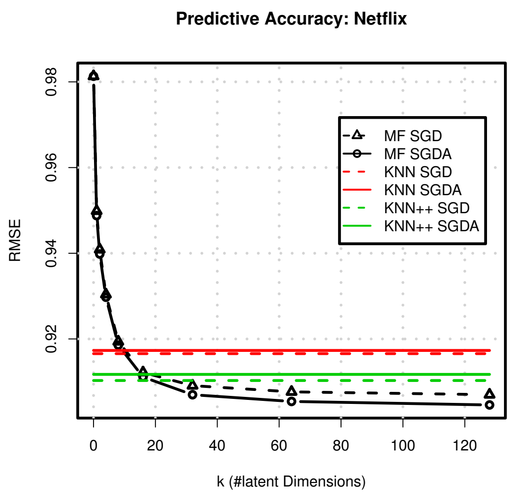
5.2 Results

5.2.1 *Predictive Accuracy*

Figure 4 compares the predictive accuracy of SGDA to SGD. It can be seen that on all variants, both methods perform comparable. Note that SGDA achieves this quality without any predefined regularization values which are crucial for SGD and very expensive to determine.

Despite SGDA variants make only use of 99% (ML1M) and 99.9% (Netflix) of the train data (1% and 0.1% of train become the validation datasets), they outperform their SGD counterparts slightly. The reason is that SGDA allows to use many regularization parameters without any additional cost: we use $k + 1$ regularization parameters for SGDA (λ_w and λ_f, $f \in \{1, \ldots, k\}$). As described before, for SGD we use only two regularization parameters λ_w and λ_v due to the exponential complexity of grid search. The increased flexibility of SGDA in modeling the *importance*[6] of each latent dimension allows for more flexible regularization and thus better predictions with higher numbers of latent dimensions overcompensating the lack of 1% or 0.1% of the training

[5]The test set (qualify) of Netflix is a temporal split where rare users are oversampled which means a random subsample from train would not be a representative validation set.
[6]Higher regularization equals to lower importance and vice versa.



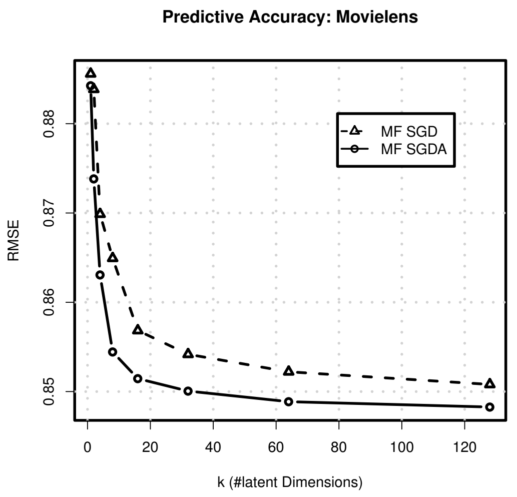


Figure 4: Predictive accuracy of Matrix Factorization and variants of learned KNN with 256 neighbors [7]. All methods are learned with SGD and SGDA. Despite SGDA does not perform any expensive hyperparameter search for regularization values, its predictive accuracy is as good as SGD which uses expensive grid search.

data. Moreover, instead of searching a grid of candidates like SGD, SGDA has an unrestricted search space for the regularization values which is another advantage of SGDA over grid-search based SGD.

Nearest neighbor models (KNN) [7] show another difference between SGD and SGDA. Since both methods have no factored features the number of regularization parameters is the same. Without the advantage of a more flexible regularization, SGDA suffers from the reduced amount of training data. Thus, SGDA performs slightly worse than SGD with optimal hyperparameters. A simple retraining of SGDA on the whole dataset S after λ has been found – like we have done for SGD – is expected to close the performance gap.

5.2.2 Convergence

When numerical optimization algorithms like gradient descent methods are applied the question of convergence arises automatically. Figure 5 shows for both evaluation datasets the relationship between train, validation and test error for SGDA based matrix factorization. Unlike the train error which is typically used for determining convergence, the validation error behaves identically to the test error. Shape and slope as a function of the number of iterations for validation and test are the same. Thus, the number of iterations necessary for SGDA can be determined automatically by monitoring the slope of the validation error.

5.2.3 Evolution of λ

Since one regularization parameter is learned per latent dimension, it is possible to assess single dimensions by the corresponding regularization parameter, i.e. more important latent dimensions have lower regularization values. These different regularization levels for each dimension are depicted in figure 6 which shows the distributions of regularization parameters per latent dimension sorted in ascending order of their mean value.

Since the different regularization values per dimension in this plot are the result of a learning process, they indicate that having more flexible regularization is better than one

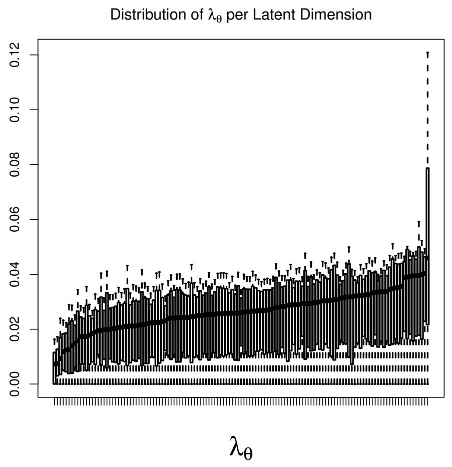


Figure 6: Series of boxplots showing distributions of all regularization values λ on Netflix for SGDA-MF with $k = 128$ features.

regularization value for all dimensions. In general, the more dimension enter a factorization model, the more regularization is necessary to avoid overfitting. However, instead of regularizing all dimensions equally like standard regularization search does, SGDA and its individually *optimized* regularization approach, do not assign the same regularization value to all dimensions but discriminate between more and less important dimensions.

5.2.4 Size of Validation Set S_V

Up to now, the size of the validation set was 1% and 0.1% of the training set for Movielens and Netflix respectively. Since the validation set guides the search for regularization values, we want to study the influence of different sizes (see fig. 7). First of all, the validation set should be representative for the test set. This way one can measure the quality of the current model parameters by predicting the validation

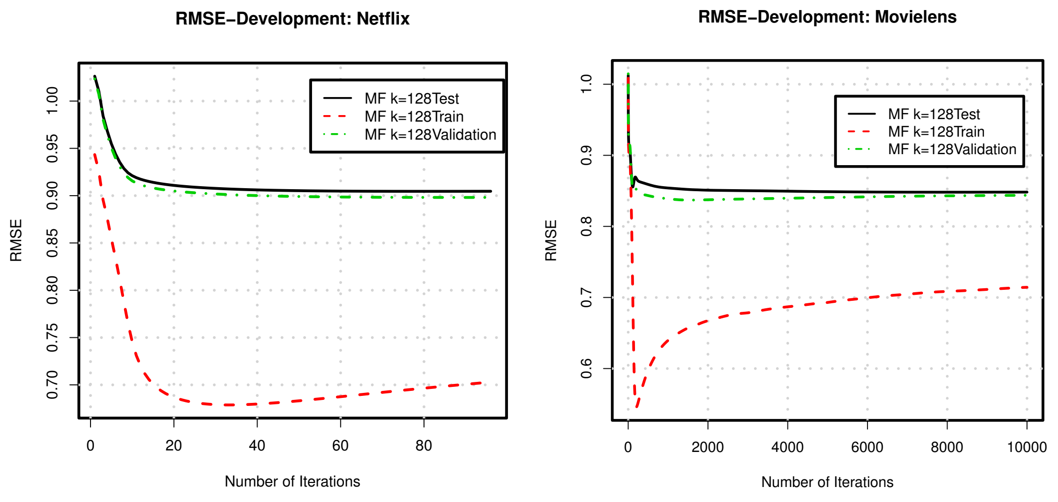


Figure 5: Development of RMSE relative to the number of full gradient descent iterations of SGDA. The quality on the validation set can be used for selecting the number of iterations.

set and see this as an approximation of the (unknown) test error. Furthermore, it has to be large enough that it cannot be overfit by the regularization values. As the regularization values are learned on the validation set, a large number of regularization values with a small validation set might lead to overfitting. Of course, the larger the validation set, the better the approximation is supposed to be. Figure 7 shows the Pearson correlation coefficient between the validation error measured after each full data iteration and the corresponding test error. While less than 100 validation instances do not suffice to accurately approximate the test error with the validation error, the approximation steadily improves by increasing the size of validation set $|S_V|$. Already 8000 cases (1% of train) have a correlation of 0.971 and $80,000$ (10% of train) a correlation of 0.9996.

However, the improved approximation comes at the cost of higher test errors since the training data reduces. Figure 7 shows this trade-off for an MF model with $k = 32$ dimensions. Even though very small validation sets (e.g. 8 and 80 cases) allow to learn the model parameters from almost all data, they lead to low prediction quality because the validation set cannot find reliable regularization parameters. A validation set of 8000 cases which does not diminish the train data set too much, leads to the best test error as the approximation between validation error and test error is high 0.971 and the training set is still large (99%). Any further increase in the validation data set improves the approximation further, however, the final error increases as too much training data is used for the test error approximation and for learning the regularization values λ_θ but not for training the model parameters Θ.

Note that this trade-off between the size of the validation set and the prediction quality on test exists not just for SGDA but for any method that uses a validation set for hyperparameter tuning (also grid-search). The experiment shows that even though a trade-off exists, all reasonable choices (800 cases = 0.1% to $80,000$ cases = 10%) lead to good results.

6. CONCLUSION AND FUTURE WORK

In this work, a new approach for finding regularization values for factorization models has been proposed. Our approach integrates the search for regularization values directly into the learning algorithm for model parameters. This avoids a time-consuming search for regularization values that is usually done by techniques like grid search and requires training the model parameters several times. Furthermore our adaptive regularization method can handle a large number of regularization parameters without any impact on the runtime complexity. We developed a learning algorithm with adaptive regularization for Factorization Machines that subsume a variety of factorization models including matrix factorization and specialized models. In our experiments, we have shown on the large Netflix dataset that our adaptive regularization method achieves similar predictive quality as the standard learning technique where the regularization parameters have been chosen carefully in advance by an expensive search.

Besides the regularization value, the learning rate is another hyperparameter of SGD algorithms. In basic factorization models like MF, there is typically only one parameter for the learning rate, however, more complex models like the integrated neighboorhood SVD++ model [7] use different learning rates. Studying adaption strategies for learning rates is an interesting direction for future research.

7. ACKNOWLEDGMENTS

I would like to thank Christoph Freudenthaler for many fruitful discussions and his valuable comments.

8. REFERENCES

[1] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[2] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46:131–159, 2002. 10.1023/A:1012450327387.

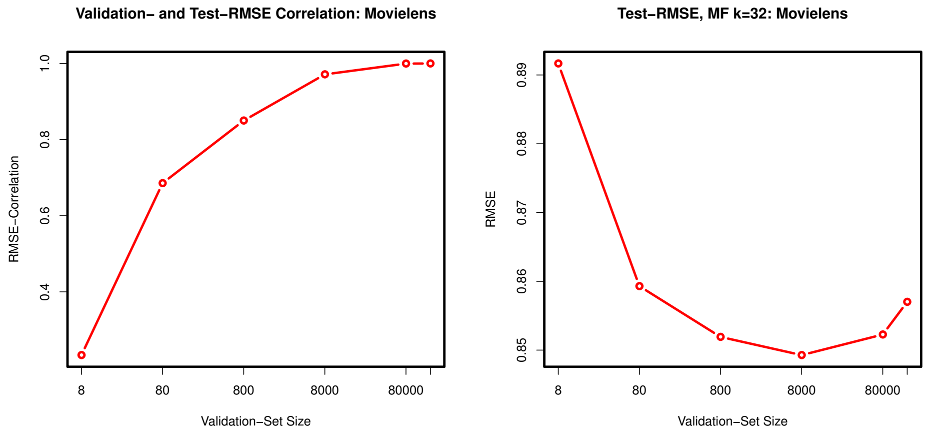


Figure 7: Influence of the size of the validation set. The left figure shows the correlation between the error on the validation set and the test set. The larger the validation data, the closer is its error to the test error. The right plot shows the test error depending on the validation size. As larger validation sets lead to smaller training sets, there is a trade-off between training and validation size.

[3] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2nd edition, 2003.

[4] R. A. Harshman. Foundations of the parafac procedure: models and conditions for an 'exploratory' multimodal factor analysis. *UCLA Working Papers in Phonetics*, pages 1–84, 1970.

[5] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5:1391–1415, December 2004.

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

[7] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434, New York, NY, USA, 2008. ACM.

[8] Y. Koren. Collaborative filtering with temporal dynamics. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, New York, NY, USA, 2009. ACM.

[9] J. Larsen, C. Svarer, and L. N. Andersen. Adaptive regularization in neural network modeling. In *Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science*, volume 1524, pages 113–132. Springer-Verlag, 1998.

[10] L. D. Lathauwer, B. D. Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21(4):1253–1278, 2000.

[11] H. Ma, H. Yang, M. R. Lyu, and I. King. Sorec: social recommendation using probabilistic matrix factorization. In *Proceeding of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 931–940, New York, NY, USA, 2008. ACM.

[12] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 39–42, 2007.

[13] S. Rendle. Factorization machines. In *Proceedings of the 10th IEEE International Conference on Data Mining*. IEEE Computer Society, 2010.

[14] S. Rendle, L. B. Marinho, A. Nanopoulos, and L. Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *KDD '09: Proceeding of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2009. ACM.

[15] S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM '10: Proceedings of the third ACM international conference on Web search and data mining*, pages 81–90, New York, NY, USA, 2010. ACM.

[16] J. D. M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005.

[17] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the 25th International Conference on Machine Learning*, volume 25, 2008.

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

[19] N. Srebro, J. D. M. Rennie, and T. S. Jaakola. Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems 17*, pages 1329–1336. MIT Press, 2005.

[20] L. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966.

[21] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. 2010.