Factorization Machines with libFM

STEFFEN RENDLE, University of Konstanz

Factorization approaches provide high accuracy in several important prediction problems, for example, recommender systems. However, applying factorization approaches to a new prediction problem is a nontrivial task and requires a lot of expert knowledge. Typically, a new model is developed, a learning algorithm is derived, and the approach has to be implemented.

Factorization machines (FM) are a generic approach since they can mimic most factorization models just by feature engineering. This way, factorization machines combine the generality of feature engineering with the superiority of factorization models in estimating interactions between categorical variables of large domain. LIBFM is a software implementation for factorization machines that features stochastic gradient descent (SGD) and alternating least-squares (ALS) optimization, as well as Bayesian inference using Markov Chain Monto Carlo (MCMC). This article summarizes the recent research on factorization machines both in terms of modeling and learning, provides extensions for the ALS and MCMC algorithms, and describes the software tool LIBFM.

Categories and Subject Descriptors: I.2.6 [**Artificial Intelligence**]: Learning—*Parameter learning*; I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*

General Terms: Algorithms, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Factorization model, matrix factorization, tensor factorization, recommender system, collaborative filtering, factorization machine

ACM Reference Format:
Rendle, S. 2012. Factorization machines with libFM. ACM Trans. Intell. Syst. Technol. 3, 3, Article 57 (May 2012), 22 pages.
DOI = 10.1145/2168752.2168771 http://doi.acm.org/10.1145/2168752.2168771

1. INTRODUCTION

Recently, factorization models have attracted a lot of research in the fields of intelligent information systems and machine learning. They have shown excellent prediction capabilities in several important applications, for example, recommender systems. The most well studied factorization model is matrix factorization [Srebro and Jaakkola 2003], which allows us to predict the relation between two categorical variables. Tensor factorization models are an extension for relations over several categorical variables; among the proposed tensor factorization approaches are Tucker Decomposition [Tucker 1966], Parallel Factor Analysis [Harshman 1970], or Pairwise Interaction Tensor factorization [Rendle and Schmidt-Thieme 2010]. For specific tasks, specialized factorization models have been proposed that take noncategorical variables into account, for example, SVD++ [Koren 2008], STE [Ma et al. 2011], FPMC

Author's address: S. Rendle, Social Network Analysis, University of Konstanz; email: steffen.rendle@uni-konstanz.de.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from the Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701, USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2012 ACM 2157-6904/2012/05-ART57 $10.00
DOI 10.1145/2168752.2168771 http://doi.acm.org/10.1145/2168752.2168771

[Rendle et al. 2010] (for a set categorical variable), timeSVD++ [Koren 2009b], and BPTF [Xiong et al. 2010] (for an additional numerical variable). For the basic matrix factorization model, many learning and inference approaches have been studied—among them, (stochastic) gradient descent, alternating least-squares (e.g., [Pilászy et al. 2010]), variational Bayes [Lim and Teh 2007], and Markov Chain Monto Carlo (MCMC) inference [Salakhutdinov and Mnih 2008a]. However, for more complex factorization models, only the most simple learning method of gradient descent is mostly available.

Even though factorization models have a high prediction quality in many applications, it is nontrivial to work with them. For each problem that cannot be described with categorical variables, a new specialized model has to be derived, and a learning algorithm has to be developed and implemented. This is very time-consuming, error-prone, and only applicable for experts in factorization models.

On the other hand, in practice, the typical approach in machine learning is to describe data with feature vectors (a preprocessing step aka feature engineering) and to apply a standard tool for example, LIBSVM [Chang and Lin 2011] for support vector machines, a toolbox such as Weka [Hall et al. 2009], or a simple linear regression tool. This approach is easy and applicable even for users without in-depth knowledge about the underlying machine-learning models and inference mechanisms.

In this article, factorization machines (FM) [Rendle 2010] are presented. FMs combine the high-prediction accuracy of factorization models with the flexibility of feature engineering. The input data for FMs is described with real-valued features, exactly like in other machine-learning approaches such as linear regression, support vector machines, etc. However, the internal model of FMs uses factorized interactions between variables, and thus, it shares with other factorization models the high prediction quality in sparse settings, like in recommender systems. It has been shown that FMs can mimic most factorization models just by feature engineering [Rendle 2010]. This article summarizes the recent research on FMs, including learning algorithms based on stochastic gradient descent, alternating least-squares, and Bayesian inference using MCMC. FMs and all presented algorithms are available in the publicly available software tool LIBFM. With LIBFM, applying factorization models is as easy as applying standard tools, such as SVMs or linear regression.

The article is structured as follows: (1) the FM model and its learning algorithms that are available in LIBFM are introduced; (2) several examples, for input data are given, and the relation to specialized factorization models is shown; (3) the LIBFM software is briefly introduced; and (4) experiments are conducted.

2. FACTORIZATION MACHINE MODEL

Let us assume that the data of a prediction problem is described by a design matrix $X \in \mathbb{R}^{n \times p}$, where the ith row $\mathbf{x}_i \in \mathbb{R}^p$ of X describes one case with p real-valued variables and where y_i is the prediction target of the ith case (see Figure 1 for an example). Alternatively, one can describe this setting as a set S of tuples $(\mathbf{x}, y)$, where (again) $\mathbf{x} \in \mathbb{R}^p$ is a feature vector and y is its corresponding target. Such a representation with data matrices and feature vectors is common in many machine-learning approaches, for example, in linear regression or support vector machines (SVM).

Factorization machines (FM) [Rendle 2010] model all nested interactions up to order d between the p input variables in $\mathbf{x}$ using factorized interaction parameters. The factorization machine (FM) model of order $d = 2$ is defined as

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \sum_{f=1}^{k} v_{j,f}\, v_{j',f}, \tag{1}$$

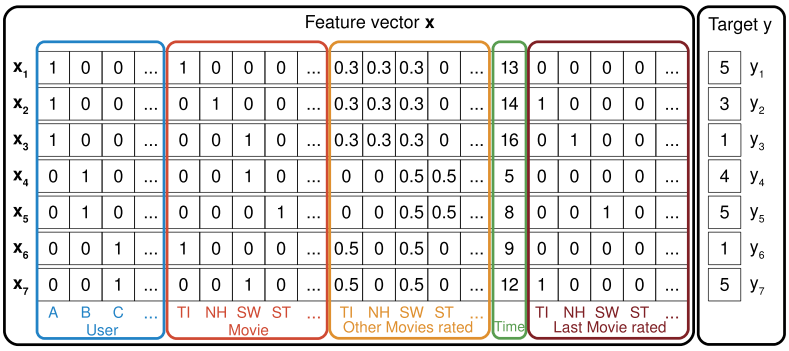


Fig. 1. Example (from Rendle [2010]) for representing a recommender problem with real valued feature vectors $\mathbf{x}$. Every row represents a feature vector $\mathbf{x}_i$ with its corresponding target y_i. For easier interpretation, the features are grouped into indicators for the active user (blue), active item (red), other movies rated by the same user (orange), the time in months (green), and the last movie rated (brown).

where k is the dimensionality of the factorization and the model parameters $\Theta = \{w_0, w_1, \ldots, w_p, v_{1,1}, \ldots v_{p,k}\}$ are

$$w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^p, \quad V \in \mathbb{R}^{p \times k}. \tag{2}$$

The first part of the FM model contains the unary interactions of each input variable x_j with the target—exactly as in a linear regression model. The second part with the two nested sums contains all pairwise interactions of input variables, that is, $x_j x_{j'}$. The important difference to standard polynomial regression is that the effect of the interaction is not modeled by an independent parameter $w_{j,j}$ but with a factorized parametrization $w_{j,j} \approx \langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle = \sum_{f=1}^{k} v_{j,f} v_{j',f}$ which corresponds to the assumption that the effect of pairwise interactions has a low rank. This allows FMs to estimate reliable parameters even in highly sparse data where standard models fail. The relation of FMs to standard machine-learning models is discussed in more detail in Section 4.3.

In Section 4, it will also be shown how FMs can mimic other well known factorization models, including matrix factorization, SVD++, FPMC, timeSVD, etc.

Complexity. Let N_z be the number of nonzero elements in a matrix X or vector $\mathbf{x}$.

$$N_z(X) := \sum_i \sum_j \delta(x_{i,j} \neq 0), \tag{3}$$

where δ is the indicator function

$$\delta(b) := \begin{cases} 1, & \text{if } b \text{ is true} \\ 0, & \text{if } b \text{ is false} \end{cases}. \tag{4}$$

The FM model in Equation (1) can be computed in $\mathcal{O}(k\, N_z(\mathbf{x}))$ because it is equivalent [Rendle 2010] to

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{j=1}^{p} w_j x_j + \frac{1}{2} \sum_{f=1}^{k} \left[\left(\sum_{j=1}^{p} v_{j,f} x_j \right)^2 - \sum_{j=1}^{p} v_{j,f}^2 x_j^2 \right]. \tag{5}$$

The number of model parameters $|\Theta|$ of an FM is $1 + p + k\, p$ and thus linear in the number of predictor variables (= size of the input feature vector) and linear in the size of the factorization k.

Multilinearity. An appealing property of FMs is multilinearity, that is, for each model parameter $\theta \in \Theta$, the FM is a linear combination of two functions g_θ and h_θ that are independent of the value of θ [Rendle et al. 2011].

$$\hat{y}(\mathbf{x}) = g_\theta(\mathbf{x}) + \theta\, h_\theta(\mathbf{x}) \quad \forall \theta \in \Theta, \tag{6}$$

with

$$h_\theta(\mathbf{x}) = \frac{\partial \hat{y}(\mathbf{x})}{\partial \theta} = \begin{cases} 1, & \text{if } \theta \text{ is } w_0 \\ x_l, & \text{if } \theta \text{ is } w_l \\ x_l \sum_{j \neq l} v_{j,f}\, x_j, & \text{if } \theta \text{ is } v_{l,f} \end{cases}. \tag{7}$$

The definition of g_θ is omitted because in the following it is never used directly. If its value should be computed, the equation $g_\theta(\mathbf{x}) = \hat{y}(\mathbf{x}) - \theta\, h_\theta(\mathbf{x})$ will be used instead.

Expressiveness. The FM model can express any pairwise interaction, provided that k is chosen large enough. This follows from the fact that any symmetric positive semidefinite matrix W can be decomposed into $V\,V^t$ (e.g., Cholesky decomposition). Let W be any pairwise interaction matrix that should express the interactions between two distinct variables in an FM. W is symmetric, and as the FM does not use the diagonal elements (because $j' > j$ in Eq. (1)), any value—and especially also arbitrary large values—for the diagonal elements are possible, which will make W positive semidefinite.

Please note that this is a theoretical statement about the expressiveness. In practice, $k \ll p$, because the advantage of FMs is the possibility to use a low-rank approximation of W, and thus, FMs can estimate interaction parameters even in highly sparse data—see Section 4.3 for a comparison to polynomial regression which uses the full matrix W for modeling interactions.

Higher-Order FMs. The FM model of order $d = 2$ (Eq. (1)) can be extended by factorizing ternary and higher-order variable interactions. The higher-order FM model [Rendle 2010] reads

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{l=2}^{d} \sum_{j_1=1}^{p} \cdots \sum_{j_d=j_{d-1}+1}^{p} \left(\prod_{i=1}^{l} x_{j_i} \right) \sum_{f=1}^{k_l} \prod_{i=1}^{l} v_{j_i,f}, \tag{8}$$

with model parameters

$$w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^p, \quad \forall l \in \{2, \ldots, d\} : V^l \in \mathbb{R}^{p \times k_l}. \tag{9}$$

Also for higher-order interactions, the nested sums in Eq. (8) can be decomposed for a more efficient computation. In the remainder of this article, we will deal only with second order FMs because in sparse settings—where factorization models are especially appealing—typically higher-order interactions are hard to estimate [Rendle and Schmidt-Thieme 2010]. Nevertheless, most formulas and algorithms can be transfered directly to higher-order FMs, as they share the property of multilinearity with second-order FMs.

3. LEARNING FACTORIZATION MACHINES

Three learning methods have been proposed for FMs: stochastic gradient descent (SGD) [Rendle 2010], alternating least-squares (ALS) [Rendle et al. 2011], and Markov Chain Monte Carlo (MCMC) inference [Freudenthaler et al. 2011]. All three of them are available in LIBFM.

3.1. Optimization Tasks

Optimality of model parameters is usually defined with a loss function l where the task is to minimize the sum of losses over the observed data S.

$$\text{OPT}(S) := \underset{\Theta}{\text{argmin}} \sum_{(\mathbf{x},y)\in S} l(\hat{y}(\mathbf{x}|\Theta), y). \tag{10}$$

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, a loss function can be chosen. For example, for regression least-squares loss,

$$l^{\text{LS}}(y_1, y_2) := (y_1 - y_2)^2, \tag{11}$$

or for binary classification ($y \in \{-1, 1\}$),

$$l^{\text{C}}(y_1, y_2) := -\ln \sigma(y_1\, y_2), \tag{12}$$

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

FMs usually have a large 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 [Srebro et al. 2005] or Tikhonov regularization.

$$\text{OPTREG}(S, \lambda) := \underset{\Theta}{\text{argmin}} \left(\sum_{(\mathbf{x},y)\in S} l(\hat{y}(\mathbf{x}|\Theta), y) + \sum_{\theta\in\Theta} \lambda_\theta \theta^2 \right), \tag{13}$$

where $\lambda_\theta \in \mathbb{R}^+$ is the regularization value for the model parameter θ. It makes sense to use individual regularization parameters for different parts of the model. In LIBFM, model parameters can be grouped—for example, one group for the parameters describing the users, one group for the items, one for time, etc. (see Figure 1 for an example of groups)—and each group uses an independent regularization value. Moreover, each factorization layer $f \in \{1, \ldots, k\}$ as well as the unary regression coefficients $\mathbf{w}$ and w_0 can have an individual regularization (again with groups). In total, the regularization structure of LIBFM is

$$\lambda^0, \quad \lambda^w_\pi, \quad \lambda^v_{f,\pi}, \quad \forall \pi \in \{1, \ldots, \Pi\}, \forall f \in \{1, \ldots, k\}, \tag{14}$$

where $\pi : \{1, \ldots, p\} \to \{1, \ldots, \Pi\}$ is a grouping of model parameters. That means, for example, the regularization value for $v_{l,f}$ would be $\lambda^v_{f,\pi(l)}$.

Probabilistic Interpretation. Both loss and regularization can also be motivated from a probabilistic point of view (e.g., Salakhutdinov and Mnih [2008b]). The least-squares loss corresponds to the assumption that the target y is Gaussian distributed with the prediction as mean

$$y|\mathbf{x}, \Theta \sim \mathcal{N}(\hat{y}(\mathbf{x}, \Theta), 1/\alpha). \tag{15}$$

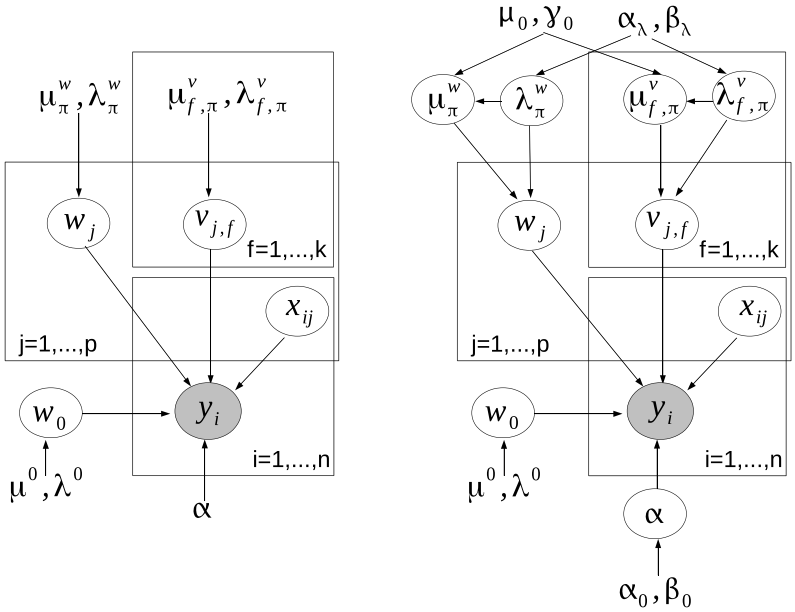
For binary classification, a Bernoulli distribution is assumed.

$$y|\mathbf{x}, \Theta \sim \text{Bernoulli}(b(\hat{y}(\mathbf{x}, \Theta))), \tag{16}$$

where $b : \mathbb{R} \to [0, 1]$ is a link function, typically the logistic function σ or the cumulative distribution function (CDF) of a standard normal distribution (Φ).

L2 regularization corresponds to Gaussian priors on the model parameters.

$$\theta|\mu_\theta, \lambda_\theta \sim \mathcal{N}(\mu_\theta, 1/\lambda_\theta) \tag{17}$$



(a) Factorization machine (FM). (b) FM with hyperpriors.

Fig. 2. Graphical representation of the variables involved in a standard factorization machine. (a) The variables are target y, input features $\mathbf{x}$, model parameters $w_0, w_j, v_{j,f}$, and hyperparameters/ priors μ, λ, α. (b) The priors are extended by hyperpriors $\Theta_0 = \{\alpha_0, \alpha_\lambda, \beta_0, \beta_\lambda, \gamma_0, \mu_0\}$ which allows the MCMC algorithm (Algorithm 3) to find the prior parameters automatically [Freudenthaler et al. 2011].

The prior mean μ_θ should be grouped and organized the same way as the regularization values λ_θ (see Eq. (14)).

The graphical model for the probabilistic view can be seen in Figure 2(a). The maximum a posteriori (MAP) estimator for this model (with $\alpha = 1$, $\mu_\theta = 0$) is the same as the optimization criterion of Eq. (13).

Gradients. For direct optimization of the loss functions, the derivatives are for least-squares regression,

$$\frac{\partial}{\partial\theta} l^{\mathrm{LS}}(\hat{y}(\mathbf{x}|\Theta), y) = \frac{\partial}{\partial\theta} \left(\hat{y}(\mathbf{x}|\Theta) - y\right)^2 = 2\left(\hat{y}(\mathbf{x}|\Theta) - y\right) \frac{\partial}{\partial\theta}\hat{y}(\mathbf{x}|\Theta), \tag{18}$$

or for classification,

$$\frac{\partial}{\partial\theta} l^{\mathrm{C}}(\hat{y}(\mathbf{x}|\Theta), y) = \frac{\partial}{\partial\theta} - \ln\sigma\left(\hat{y}(\mathbf{x}|\Theta)\, y\right) = \left(\sigma\left(\hat{y}(\mathbf{x}|\Theta)\, y\right) - 1\right) y \frac{\partial}{\partial\theta}\hat{y}(\mathbf{x}|\Theta). \tag{19}$$

Finally, due to multilinearity of the FM model, the partial derivative of the model equation with respect to θ corresponds to h_θ (Eq. (7)).

$$\frac{\partial}{\partial\theta}\hat{y}(\mathbf{x}|\Theta) = h_\theta(\mathbf{x}). \tag{20}$$

3.2. Stochastic Gradient Descent (SGD)

Stochastic gradient descent (SGD) algorithms are very popular for optimizing factorization models as they are simple, work well with different loss functions, and have low computational and storage complexity. Algorithm 1 shows how FMs can be optimized

ALGORITHM 1: Stochastic Gradient Descent (SGD)

Input: Training data S, regularization parameters λ, learning rate η, initialization σ
Output: Model parameters $\Theta = (w_0, \mathbf{w}, \mathbf{V})$
$w_0 \leftarrow 0$; $\mathbf{w} \leftarrow (0, \ldots, 0)$; $\mathbf{V} \sim \mathcal{N}(0, \sigma)$;
repeat
  for $(\mathbf{x}, y) \in S$ **do**
    $w_0 \leftarrow w_0 - \eta \left(\frac{\partial}{\partial w_0} l(\hat{y}(\mathbf{x}|\Theta), y) + 2\,\lambda^0\, w_0 \right)$;
    for $i \in \{1, \ldots, p\} \wedge x_i \neq 0$ **do**
      $w_i \leftarrow w_i - \eta \left(\frac{\partial}{\partial w_i} l(\hat{y}(\mathbf{x}|\Theta), y) + 2\,\lambda^w_{\pi(i)}\, w_i \right)$;
      for $f \in \{1, \ldots, k\}$ **do**
        $v_{i,f} \leftarrow v_{i,f} - \eta \left(\frac{\partial}{\partial v_{i,f}} l(\hat{y}(\mathbf{x}|\Theta), y) + 2\,\lambda^v_{f,\pi(i)}\, v_{i,f} \right)$;
      end
    end
  end
until *stopping criterion is met*;

with SGD [Rendle 2010]. The algorithm iterates over cases $(\mathbf{x}, y) \in S$ and performs updates on the model parameters.

$$\theta \leftarrow \theta - \eta \left(\frac{\partial}{\partial \theta} l(\hat{y}(\mathbf{x}), y) + 2\,\lambda_\theta\, \theta \right), \tag{21}$$

where $\eta \in \mathbb{R}^+$ is the learning rate or step size for gradient descent.

Complexity. The SGD algorithm for FMs has a linear computational and constant storage complexity. For one iteration over all training cases, the runtime complexity of SGD is $\mathcal{O}(k\, N_z(X))$ because, for each single case $(\mathbf{x}, y) \in S$, the complexity for the gradient steps is $\mathcal{O}(k \sum_{i=1}^{p} \delta(x_i \neq 0)) = \mathcal{O}(k\, N_z(\mathbf{x}))$.

Hyperparameters. For performing SGD, there are several critical hyperparameters.

—*Learning rate* η: The convergence of SGD depends largely on η: if η is chosen too high, the algorithm does not converge, and if it is chosen too small, convergence is slow. Typically, η is the first hyperparameter that should be determined.
—*Regularization* λ: As noted in Section 3.1, the generalization capabilities of FMs and thus the prediction quality, depends largely on the choice of the regularization λ. The regularization values are typically searched on a separate holdout set, for example, using grid search. As there are several regularization parameters (see Eq. (14)), the grid has exponential size and thus this search is very time-consuming. To make the search more feasible, the number of regularization parameters is usually reduced, for example, groups are dropped and all factor layers use the same regularization value.
—*Initialization* σ: The parameters for the factorized interactions ($\mathbf{V}$) have to be initialized with nonconstant values. In LIBFM, the values are sampled from a zero-mean normal distribution with standard deviation σ. Typically small values are used for σ.

SGD with Adaptive Regularization. In Rendle [2012], it has been shown how the regularization values can be adapted automatically in SGD, while the model parameters are learned. LIBFM includes the adaptive regularization algorithm proposed there and extends it with groups.

3.3. Alternating Least-Squares/Coordinate Descent

The optimization approach of SGD is based on iterating over cases (rows) of the training data and performing small steps in the direction of a smaller loss. Coordinate descent or alternating least-squares (ALS) takes another approach by minimizing the loss per model parameter. For least-squares regression with L2-regularization, the optimal value θ^* for one model parameter θ given all remaining parameters $\Theta \setminus \{\theta\}$ can be calculated directly [Rendle et al. 2011] as

$$\begin{aligned}
\theta^* &= \operatorname*{argmin}_{\theta} \left(\sum_{(\mathbf{x},y)\in S} \left(\hat{y}(\mathbf{x}|\Theta) - y\right)^2 + \sum_{\theta\in\Theta} \lambda_\theta \theta^2 \right) \\
&= \operatorname*{argmin}_{\theta} \left(\sum_{(\mathbf{x},y)\in S} \left(g_\theta(\mathbf{x}|\Theta \setminus \{\theta\}) + \theta\, h_\theta(\mathbf{x}|\Theta \setminus \{\theta\}) - y\right)^2 + \sum_{\theta\in\Theta} \lambda_\theta \theta^2 \right) \\
&= \frac{\sum_{i=1}^n (y - g_\theta(\mathbf{x}_i|\Theta \setminus \{\theta\}))\, h_\theta(\mathbf{x}_i|\Theta \setminus \{\theta\})}{\sum_{i=1}^n h_\theta(\mathbf{x}_i)^2 + \lambda_\theta} \\
&= \frac{\theta \sum_{i=1}^n h_\theta^2(\mathbf{x}_i) + \sum_{i=1}^n h_\theta(\mathbf{x}_i)\, e_i}{\sum_{i=1}^n h_\theta(\mathbf{x}_i)^2 + \lambda_\theta},
\end{aligned} \tag{22}$$

where e_i is the 'error' term/ residual of the ith case.

$$e_i := y_i - \hat{y}(\mathbf{x}_i|\Theta). \tag{23}$$

This allows us to derive a least-squares learning algorithm (see Algorithm 2) that iteratively solves a least-squares problem per model parameter and updates each model parameter with the optimal (local) solution,

$$\theta \leftarrow \theta^*. \tag{24}$$

This is performed iteratively over all parameters until convergence.

Complexity. The main effort of ALS learning (Eq. (22)) is computing the following two quantities.

$$\sum_{i=1}^n h_\theta(\mathbf{x}_i)^2, \quad \sum_{i=1}^n h_\theta(\mathbf{x}_i) e_i = \sum_{i=1}^n h_\theta(\mathbf{x}_i) \left(y_i - \hat{y}(\mathbf{x}_i|\Theta)\right). \tag{25}$$

With a trivial implementation, updating one model parameter would require computing the model equation $\hat{y}(\mathbf{x}_i)$, and the gradient $h_\theta(\mathbf{x}_i)$ for each training case $\mathbf{x}_i$ where the corresponding column j is nonzero ($x_{i,j} \neq 0$). For example, for updating the model parameter $v_{j,f}$, this would render the complexity $\mathcal{O}(\sum_{i=1}^n \delta(x_{i,j} \neq 0)\, k\, N_z(\mathbf{x}_i))$. In total, this would have to be computed for each of the $1 + p\,(k+1)$ model parameters.

In Rendle et al. [2011], it has been shown how one full iteration over all model parameters Θ can be done efficiently in $\mathcal{O}(N_z(X)\, k)$ by precomputing the caches $e \in \mathbb{R}^n$ (see Eq. (23)) and $\boldsymbol{Q} \in \mathbb{R}^{n\times k}$ such that

$$q_{i,f} := \sum_{l=1}^p v_{l,f}\, x_{i,l}, \tag{26}$$

which allows us to compute h quickly in $\mathcal{O}(1)$.

$$h_{v_{l,f}}(\mathbf{x}_i) = x_{i,l} \left(q_{i,f} - v_{l,f}\, x_{i,l}\right). \tag{27}$$

ALGORITHM 2: Alternating least squares (ALS)

Input: Training data S, regularization parameters λ, initialization σ
Output: Model parameters $\Theta = (w_0, \mathbf{w}, \mathbf{V})$
$w_0 \leftarrow 0$; $\mathbf{w} \leftarrow (0, \ldots, 0)$; $\mathbf{V} \sim \mathcal{N}(0, \sigma)$;
repeat
  $\hat{\mathbf{y}} \leftarrow$ predict all cases S;
  $\mathbf{e} \leftarrow \mathbf{y} - \hat{\mathbf{y}}$;
  $w_0 \leftarrow w_0^*$;
  for $l \in \{1, \ldots, p\}$ **do**
    $w_l \leftarrow w_l^*$;
    update e;
  end
  for $f \in \{1, \ldots, k\}$ **do**
    init $q_{\cdot,f}$;
    for $l \in \{1, \ldots, p\}$ **do**
      $v_{l,f} \leftarrow v_{l,f}^*$;
      update e, q;
    end
  end
until *stopping criterion is met*;

Now calculating θ^* for the lth parameter is in $\mathcal{O}(\sum_{i=1}^{n} \delta(x_{i,l} \neq 0))$. Also, updating each cache value q and e can be done in constant extra time (see Rendle et al. [2011]).

However, the speed-up comes at the price of higher memory consumption for the caches. The approach presented in Rendle et al. [2011] has an additional memory complexity of $\mathcal{O}(nk)$ because of the Q-cache. LIBFM provides a more efficient implementation with only $\mathcal{O}(n)$ memory complexity for the Q-cache (see Algorithm 2). The idea is that the model parameters are updated per layer f (i.e., first all parameters $v_{1,1}, v_{2,1}, v_{3,1}, \ldots$, then $v_{1,2}, v_{2,2}, v_{3,2}, \ldots$, etc.), and in each layer, only the cache values Q of the same layer have to be present. This means that LIBFM stores (and updates) only the Q-cache for one layer (and thus the storage is $\mathcal{O}(n)$), and when changing the layer, the Q-values of the new layer are computed/initialized. The initialization of the Q-values per layer has no negative effect on the overall computational complexity.

Hyperparameters. A clear advantage of ALS over SGD is that ALS has no learning rate as hyperparameter. However, two important hyperparameters remain: regularization and initialization. Finding good regularization values is especially computational-expensive.

Classification. The ALS algorithm described so far is restricted to least-squares regression and cannot solve classification tasks. LIBFM contains classification capabilities for ALS/coordinate descent, which is based on using a probit-link function. This approach is motivated from the probabilistic interpretation (Section 3.1) and will be described at the end of the MCMC section.

3.4. Markov Chain Monte Carlo (MCMC) Inference

The Bayesian model used so far can be seen in Figure 2. Both ALS and SGD learn the best parameters Θ which are used for a point estimate of $\hat{y}$. MCMC is a Bayesian inference technique that generates the distribution of $\hat{y}$ by sampling. For MCMC

inference in FMs using Gibbs sampling, the conditional posterior distributions for each model parameter are [Freudenthaler et al. 2011]

$$\theta|X, y, \Theta \setminus \{\theta\}, \Theta_H \sim \mathcal{N}(\tilde{\mu}_\theta, \tilde{\sigma}_\theta^2), \tag{28}$$

where

$$\tilde{\sigma}_\theta^2 := \left(\alpha \sum_{i=1}^{n} h_\theta(\mathbf{x}_i)^2 + \lambda_\theta\right)^{-1}, \tag{29}$$

$$\tilde{\mu}_\theta := \tilde{\sigma}_\theta^2 \left(\alpha\, \theta \sum_{i=1}^{n} h_\theta^2(\mathbf{x}_i) + \alpha \sum_{i=1}^{n} h_\theta(\mathbf{x}_i)\, e_i + \mu_\theta\, \lambda_\theta\right), \tag{30}$$

and Θ_H are the hyperparameters

$$\Theta_H := \{(\mu^0, \lambda^0), (\mu_\pi^w, \lambda_\pi^w), (\mu_{f,\pi}^v, \lambda_{f,\pi}^v) : \forall \pi \in \{1, \ldots, \Pi\}, \forall f \in \{1, \ldots, k\}\}. \tag{31}$$

When comparing the conditional posterior of model parameters for MCMC (Eq. (30)) and the ALS solution (Eq. (22)), it can be seen that both are very similar, that is, $\theta^* = \tilde{\mu}_\theta$ with $\alpha = 1$ and $\mu_\cdot = 0$. The difference is that MCMC samples from the posterior distribution, while ALS uses the expected value.

A major advantage of MCMC over ALS and SGD is that it allows us to integrate the regularization parameters Θ_H into the model, which avoids a time-consuming search for these hyperparameters. For integration of Θ_H, the Bayesian FM model is extended (Figure 2) by placing distributions on the priors (hyperprior distributions). For each pair $(\mu_\theta, \lambda_\theta) \in \Theta_H$ of prior parameters, a Gamma distribution is assumed for λ_θ and a normal distribution for μ_θ. That is,

$$\mu_\pi^w \sim \mathcal{N}(\mu_0, \gamma_0 \lambda_\pi^w), \quad \lambda_\pi^w \sim \Gamma(\alpha_\lambda, \beta_\lambda), \quad \mu_{f,\pi}^v \sim \mathcal{N}(\mu_0, \gamma_0 \lambda_{f,\pi}^v), \quad \lambda_{f,\pi}^v \sim \Gamma(\alpha_\lambda, \beta_\lambda), \tag{32}$$

where μ_0, γ_0, as well as α_λ and β_λ, describe the hyperprior distributions. Finally, a Gamma distribution is also placed on α.

$$\alpha \sim \Gamma(\alpha_0, \beta_0). \tag{33}$$

In total, the hyperpriors lead to the following new parameters Θ_0.

$$\Theta_0 := \{\alpha_0, \beta_0, \alpha_\lambda, \beta_\lambda, \mu_0, \gamma_0\}. \tag{34}$$

MCMC allows us to integrate Θ_H into the inference process, that is, values for Θ_H are found automatically by sampling from their corresponding conditional posterior distributions [Freudenthaler et al. 2011].

$$\alpha|y, X, \Theta_0, \Theta \sim \Gamma\left(\frac{\alpha_0 + n}{2}, \frac{1}{2}\left[\sum_{i=1}^{n}(y_i - \hat{y}(\mathbf{x}_i|\Theta))^2 + \beta_0\right]\right), \tag{35}$$

$$\lambda_\pi^\cdot|\Theta_0, \Theta_H \setminus \{\lambda_\pi^\cdot\}, \Theta \sim \Gamma\left(\frac{\alpha_\lambda + p^\pi + 1}{2}, \frac{1}{2}\left[\sum_{j=1}^{p} \delta(\pi(j) = \pi)(\theta_j - \mu_\theta)^2 + \gamma_0(\mu_\pi^\cdot - \mu_0)^2 + \beta_\lambda\right]\right), \tag{36}$$

$$\mu_\pi^\cdot|\Theta_0, \Theta_H \setminus \{\lambda_\pi^\cdot\}, \Theta \sim \mathcal{N}\left((p^\pi + \gamma_0)^{-1}\left[\sum_{j=1}^{p} \delta(\pi(j) = \pi)\theta_j + \gamma_0\mu_0\right], \frac{1}{(p^\pi + \gamma_0)\lambda_\pi^\cdot}\right), \tag{37}$$

with

$$p^{\pi} := \sum_{j=1}^{p} \delta(\pi(j) = \pi). \tag{38}$$

Complexity. The Gibbs sampler for MCMC inference is sketched in Algorithm 3 and has the same complexity as the ALS algorithm. This follows directly from the observation that for both algorithms, the same summations have to be computed for the conditional posterior distribution in MCMC and the expected value in ALS. The overhead of MCMC is the inference over the Θ_H, that is, computing the posteriors (Eqs. (35), (36), and (37)), but even with a straightforward implementation, this is in $\mathcal{O}(k\, N_z(X))$.

Hyperparameters. A major advantage of MCMC is that the regularization values Θ_H are automatically determined. This comes at the price of introducing parameters for hyperpriors Θ_0. However, (1) the number of hyperpriors $|\Theta_H|$ is smaller than the number of regularization parameters $|\Theta_0|$, and (2) more importantly, MCMC is typically insensitive to choices of Θ_0. That is, a trivial choice for the values of Θ_0 works well. In LIBFM, the following trivial values for Θ_0 are used: $\alpha_0 = \beta_0 = \alpha_\lambda = \beta_\lambda = \gamma_0 = 1$ and $\mu_0 = 0$.

The only hyperparameter that remains for MCMC is the initialization σ. In general here, one can even use a value of 0 (which is not possible for ALS and SGD), because MCMC's posterior uncertainty will identify the factorization; however, choosing a proper value can speed up the sampler. One can usually see in the first few samples if an initialization σ is a good choice.

Classification. The MCMC Algorithm 3 solves regression tasks. It can be extended for binary classification by mapping the normal distributed $\hat{y}$ to a probability $b(\hat{y}) \in [0, 1]$ that defines the Bernoulli distribution for classification [Gelman et al. 2003]. That means, the MCMC algorithm will predict the probability that a case is of the positive class. LIBFM uses the CDF of a normal distribution as mapping, that is, $b(z) = \Phi(z)$, because the posteriors are then easy to sample from.

The only two changes that have to be made to Algorithm 3 for classification is (1) that for prediction, $\hat{y}$ is transformed by Φ, and (2) that instead of regressing to y, the regression target y' is sampled in each iteration from its posterior that has a truncated normal distribution.

$$y'_i|\mathbf{x}_i, y_i, \Theta \sim \begin{cases} \mathcal{N}(\hat{y}(\mathbf{x}_i, \Theta), 1)\, \delta(y'_i < 0), & \text{if } y_i \text{ has the negative class} \\ \mathcal{N}(\hat{y}(\mathbf{x}_i, \Theta), 1)\, \delta(y'_i \geq 0), & \text{if } y_i \text{ has the positive class} \end{cases}. \tag{39}$$

Sampling from this distribution is efficient [Robert 1995].

As noted before ALS, for regression, can be seen as a simplification of MCMC where the model parameters are not sampled but their expectation value is taken in each update. A classification option for ALS is available in LIBFM that follows the same idea, and instead of sampling from the truncated normal (as it is done in MCMC), for classification with ALS, the expected value of the truncated normal is computed.

3.5. Summary

An overview of the properties of the learning algorithms in LIBFM can be found in Table I.

ALGORITHM 3: Markov Chain Monte Carlo Inference (MCMC)

Input: Training data S, Test data S_{test}, initialization σ
Output: Prediction $\hat{\mathbf{y}}_{\text{test}}$ for the test cases
$w_0 \leftarrow 0$; $\mathbf{w} \leftarrow (0, \ldots, 0)$; $\mathbf{V} \sim \mathcal{N}(0, \sigma)$;
$\#_{\text{samples}} \leftarrow 0$;
repeat
  $\hat{\mathbf{y}} \leftarrow$ predict all cases S;
  $\mathbf{e} \leftarrow \mathbf{y} - \hat{\mathbf{y}}$;
  Update the hyperparameters:
  sample α from eq. (35);
  for $(\mu_\pi, \lambda_\pi) \in \Theta_H$ **do**
    sample λ_π from eq. (36);
    sample μ_π from eq. (37);
  end
  Update the model parameters:
  sample w_0 from $\mathcal{N}(\tilde{\mu}_{w_0}, \tilde{\sigma}^2_{w_0})$;
  for $l \in \{1, \ldots, p\}$ **do**
    sample w_l from $\mathcal{N}(\tilde{\mu}_{w_l}, \tilde{\sigma}^2_{w_l})$;
    update e;
  end
  for $f \in \{1, \ldots, k\}$ **do**
    init $q_{\cdot,f}$;
    for $l \in \{1, \ldots, p\}$ **do**
      sample $v_{l,f}$ from $\mathcal{N}(\tilde{\mu}_{v_{l,f}}, \tilde{\sigma}^2_{v_{l,f}})$;
      update e, q;
    end
  end
  $\#_{\text{samples}} \leftarrow \#_{\text{samples}} + 1$;
  $\hat{\mathbf{y}}^*_{\text{test}} \leftarrow$ predict all cases S_{test};
  $\hat{\mathbf{y}}_{\text{test}} \leftarrow \hat{\mathbf{y}}_{\text{test}} + \hat{\mathbf{y}}^*_{\text{test}}$;
until *stopping criterion is met*;
$\hat{\mathbf{y}}_{\text{test}} \leftarrow \frac{1}{\#_{\text{samples}}} \hat{\mathbf{y}}_{\text{test}}$;

Table I. Properties of the Learning Algorithms in LIBFM

Algorithm	SGD	ALS	MCMC
Runtime Complexity	$\mathcal{O}(k\, N_z(X))$	$\mathcal{O}(k\, N_z(X))$	$\mathcal{O}(k\, N_z(X))$
Storage Complexity	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Regression	yes	yes	yes
Classification	yes	yes	yes
Hyperparameters	initialization, regularization values λs, learning rate η	initialization, regularization values λs	initialization, hyperpriors (insensitive)

4. RELATED WORK AND APPLICATION OF FACTORIZATION MACHINES

First, examples for input data are shown, including how they relate to other specialized factorization models. Note that FMs are not restricted to the choices presented here. Second, other generic factorization models are compared to FMs. Third, FMs are compared to polynomial regression.

4.1. Expressing Factorization Models with Factorization Machines

In this section, the generality of FMs will be discussed by comparing them to other specialized state-of-the-art factorization models. This also shows how to apply FMs by defining the input data (i.e., features). It is crucial to note that, in practice, only the feature vector $\mathbf{x}$ has to be defined; the rest is done implicitly by the FM—neither an explicit reformulation of the model equation nor developing of new prediction nor a learning algorithms is necessary. The analysis of the FM model equation that is done in this section is just to show the theoretical relations to other models.

4.1.1. Matrix Factorization. Assume data about two categorical variables U (e.g., a user) and I (e.g., an item) should be used in a FM. The straightforward way to describe a case $(u, i) \in U \times I$ is to use a feature vector $\mathbf{x} \in \mathbb{R}^{|U|+|I|}$ with binary indicator variables, that is,

$$(u, i) \rightarrow \mathbf{x} = (\underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|U|}, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|I|}), \tag{40}$$

where the uth entry in the first part of $\mathbf{x}$ is 1, the ith entry on the second part of $\mathbf{x}$ is 1, and the rest is 0 (e.g., see the first two groups of Figure 1). Using this data in an FM, the FM will be exactly the same as a (biased) matrix factorization model (FM) [Paterek 2007; Srebro et al. 2005].

$$\hat{y}(\mathbf{x}) = \hat{y}(u, i) = w_0 + w_u + w_i + \sum_{f=1}^{k} v_{u,f}\, v_{i,f}. \tag{41}$$

4.1.2. Pairwise Interaction Tensor Factorization. If three categorical variables should be described, for example, U, I, and T (e.g., tags), a straightforward representation with a feature vector would be $\mathbf{x} \in \mathbb{R}^{|U|+|I|+|T|}$.

$$(u, i, t) \rightarrow \mathbf{x} = (\underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|U|}, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|I|}, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|T|}). \tag{42}$$

An FM using this data representation would be similar to the pairwise interaction tensor factorization model (PITF) [Rendle and Schmidt-Thieme 2010].

$$\hat{y}(\mathbf{x}) = \hat{y}(u, i, t) = w_0 + w_u + w_i + w_t + \sum_{f=1}^{k} v_{u,f}\, v_{i,f} + \sum_{f=1}^{k} v_{u,f}\, v_{t,f} + \sum_{f=1}^{k} v_{i,f}\, v_{t,f}. \tag{43}$$

The difference between this FM and the original PITF is that this FM contains lower-order interactions and shares the factors V between interactions. Besides this, both approaches are identical.

4.1.3. SVD++ and FPMC. Assume that there are two categorical variables (e.g., U and I) and one set-categorical variable (e.g., $\mathcal{P}(L)$). One simple representation of this data would be $\mathbf{x} \in \mathbb{R}^{|U|+|I|+|L|}$.

$$(u, i, \{l_1, \ldots, l_m\}) \rightarrow \mathbf{x} = (\underbrace{0, \ldots, 1, 0, \ldots}_{|U|}, \underbrace{0, \ldots, 1, 0, \ldots}_{|I|}, \underbrace{0, \ldots, 1/m, 0, \ldots, 1/m, 0, \ldots}_{|L|}), \tag{44}$$

where each of the m elements of the set $\{l_1, \ldots, l_m\}$ is described by a nonzero value, for example, by $1/m$ in the corresponding column (e.g., see the first three groups of Figure 1). With this data, the FM will be equivalent to

$$\hat{y}(\mathbf{x}) = \hat{y}(u, i, \{l_1, \ldots, l_m\}) = \overbrace{w_0 + w_u + w_i + \underbrace{\langle \mathbf{v}_u, \mathbf{v}_i \rangle + \frac{1}{m}\sum_{j=1}^{m} \langle \mathbf{v}_i, \mathbf{v}_{l_j} \rangle}_{FPMC}}^{SVD++} + \frac{1}{m}\sum_{j=1}^{m} w_{l_j} + \frac{1}{m}\sum_{j=1}^{m} \langle \mathbf{v}_u, \mathbf{v}_{l_j} \rangle + \frac{1}{m^2}\sum_{j=1}^{m}\sum_{j'>j}^{m} \langle \mathbf{v}_{l_j}, \mathbf{v}_{l_{j'}} \rangle. \quad (45)$$

If implicit feedback is used as input for $\{l_1, \ldots, l_m\}$, the FM just sketched is almost the same as the SVD++ model [Koren 2008; Salakhutdinov and Mnih 2008b; Takács et al. 2009]. The first part (annotated with *SVD++*) is exactly the same as the original SVD++ [Koren 2008]; the second part (second line in Eq. (45)) contains some additional interactions. If sequential information is used as input for $\{l_1, \ldots, l_m\}$, the FM is very similar to the Factorized Personalized Markov Chain (FPMC) [Rendle et al. 2010]—especially if the FM is optimized for ranking (like FPMC) almost all terms that are in the FM model but not in the FPMC model will vanish (see [Rendle 2010] for details). If social information is used as input (e.g., friends), the FM is similar to the Social Trust Ensemble (STE) [Ma et al. 2011].

4.1.4. BPTF and TimeSVD++. If time should be included, the most simple approach is to treat time as a categorical variable (e.g., each day is a level) and apply the same encoding as in Eq. (42). The FM with this data is similar to the time-aware BPTF model [Xiong et al. 2010]. The differences are that BPTF uses a ternary PARAFAC model over the three categorical variables (user, item, time), whereas FM uses factorized pairwise interactions. Moreover, BPTF has an additional regularizer over the time variables. In Freudenthaler et al. [2011], it has been shown that the FM works indeed better than the more complex BPTF model.

Another approach is to use a separate time variable per user (i.e., making the user-time interaction explicit). The input data would be $\mathbf{x} \in \mathbb{R}^{|U|+|I|+|U||T|}$ with binary indicators for the user, item, and a user-specific day indicator. With this data, the FM model would be equivalent to.

$$\begin{aligned}\hat{y}(\mathbf{x}) = \hat{y}(u, i, t) &= w_0 + w_u + w_i + w_{(u,t)} + \sum_{f=1}^{k} v_{u,f}\, v_{i,f} + \sum_{f=1}^{k} v_{(u,t),f}\, v_{i,f} + \sum_{f=1}^{k} v_{(u,t),f}\, v_{u,f} \\ &= w_0 + \underbrace{w_u + w_{(u,t)}}_{b_u(t)} + w_i + \sum_{f=1}^{k} \underbrace{(v_{u,f} + v_{(u,t),f})}_{v_{u,f}(t)}\, v_{i,f} + \sum_{f=1}^{k} v_{(u,t),f}\, v_{u,f}. \end{aligned} \quad (46)$$

This model captures the 'day-specific variability' of biases $b_u(t)$ and of factors $v_{u,f}(t)$, exactly as in the TimeSVD model of Koren [2009b]. Extending the feature vectors of the FM with implicit feedback indicators (see Section 4.1.3) and a linear indicator for the time will result in the TimeSVD++ model (the *time* group of Figure 1 is an example for a linear time indicator).

4.1.5. Nearest Neighbor Models. When other numerical measurements are available, for example, other ratings $r_1, r_2, \ldots$, the same user has given to items $l_1, l_2, \ldots \in I$ etc., this can be encoded in a feature vector $\mathbf{x} \in \mathbb{R}^{|I|+|I|}$.

$$(i, \{(r_1, l_1), \ldots, (r_m, l_m)\}) \rightarrow \mathbf{x} = (\underbrace{0, \ldots, 1, \ldots, 0}_{|I|}, \underbrace{0, \ldots, r_1/m, 0, \ldots, r_m/m, \ldots, 0}_{|I|}). \quad (47)$$

The FM model with this data would be equivalent to

$$\hat{y}(\mathbf{x}) = \hat{y}(i, \{(r_1, l_1), \ldots, (r_m, l_m)\}) = w_0 + w_i + \frac{1}{m}\sum_{j=1}^{m} r_j w_{l_j} + \overbrace{\frac{1}{m}\sum_{j=1}^{m} r_j \langle \mathbf{v}_i, \mathbf{v}_{l_j} \rangle}^{\textit{Factorized KNN}} + \frac{1}{m^2}\sum_{j=1}^{m}\sum_{j'>j}^{m} r_j r_{j'} \langle \mathbf{v}_{l_j}, \mathbf{v}_{l_{j'}} \rangle \quad (48)$$

This model is similar to the factorized nearest neighbor model [Koren 2010].

Another possible approach to encode the data would be to use separate rating indicators per item and one for the user, that is, $\mathbf{x} \in \mathbb{R}^{|I|+|U|+|I||I|}$. That means the rating indicators in Eq. (47) would be in a separate block for each item. An FM of order $d = 1$ with this data would be equivalent to

$$\hat{y}(\mathbf{x}) = \hat{y}(i, u, \{(r_1, l_1), \ldots, (r_m, l_m)\}) = w_0 + w_i + w_u + \frac{1}{m}\sum_{j=1}^{m} r_j w_{i,l_j}. \quad (49)$$

This approach is identical to the nonfactorized nearest neighbor model of Koren [2008]. It can be combined with the implicit feedback idea which results in the KNN++ model [Koren 2008].

4.1.6. Attribute-Aware Models. There are several attempts to integrate attribute information about users and items into recommender systems. It is very simple to use such information in an FM. A straightforward approach is to add the attributes of an item (or user), such as genre, actor, etc., to the input vector $\mathbf{x}$.

Assume the input vector consists of such item attributes and an indicator variable for the user.

$$(u, a_1^i, \ldots, a_m^i) \rightarrow \mathbf{x} = (\underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|U|}, \underbrace{a_1^i, \ldots, a_m^i}_{\text{attributes of item } i}). \quad (50)$$

With this data, the FM would be equivalent to

$$\hat{y}(\mathbf{x}) = \hat{y}(u, a_1^i, \ldots, a_m^i) = w_0 + w_u + \sum_{j=1}^{m} a_j^i w_j + \underbrace{\sum_{j=1}^{m} a_j^i \langle \mathbf{v}_u, \mathbf{v}_j \rangle}_{\textit{'attribute mapping'}} + \sum_{j=1}^{m}\sum_{j'>j}^{m} a_j^i a_{j'}^i \langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle. \quad (51)$$

This is almost identical to the attribute-aware approach in Gantner et al. [2010] that uses a linear regression to map item attributes to factors (see the highlighted part '*attribute mapping*' of Eq. (51))—the only difference is that the FM contains biases as well as additional interactions between item attributes (e.g., between genre and actors).

If the input vector of standard matrix factorization (Eq. (40)) is extended by attribute information of the user (e.g., demographics) and attributes of the item, the FM

would correspond to the attribute-aware model proposed in Agarwal and Chen [2009]. Again, the difference is that the FM contains additional interactions within user and item attributes (e.g., interaction between a user's age and gender).

4.2. Other Generic Factorization Models

There are other attempts for a more generic factorization model. In Agarwal and Chen [2009], a matrix factorization model is extended with regression priors. That is, the mean of the normal distributed priors of factors is a linear regression model. FMs can mimic this approach because for any hierarchical model using normal distributed priors, the mean value of the prior (and thus also a linear regression model for the prior mean) can be added as covariates to the feature vectors. On the other hand, MF with regression priors is much more restricted than FMs, because MF itself is limited to interactions of two categorical variables, and thus, the MF model with regression priors is not appropriate for tasks with interactions over more than two variables, for example, tag recommendation or context aware recommendation. FMs include (pairwise) interactions between any number of variables (also not limited to categorical ones).

SVDfeature [Chen et al. 2011] is another generic factorization model. Similar to Agarwal and Chen [2009], in SVDfeature, a matrix factorization model is extended with linear regression terms for the factors and for the bias term. However, compared to FMs, it shares the same shortcomings of Agarwal and Chen [2009]: only interactions between two categorical variables can be factorized. That means it is not able to mimic state-of-the-art context-aware recommenders, tag recommenders, etc. Moreover, for SVDfeature, only SGD learning has been proposed, whereas LIBFM features MCMC inference, which is much more simple to apply, as there is no learning rate, and the regularization values are automatically determined. An advantage of SVDfeature over LIBFM is that due to the more restrictive model equation, it has an improved learning algorithm (following Koren [2008]) for speeding up learning in factor regression terms.

4.3. Relation to Polynomial Regression

In Rendle [2010] it has been shown that FMs can be seen as polynomial regression (or SVM with inhomogeneous polynomial kernel) using factorized parameter matrices. Polynomial regression of order $d = 2$ can be defined as

$$\hat{y}^{\text{PR}}(\mathbf{x}) := w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j}^{p} w_{j,j'} x_j x_{j'}, \tag{52}$$

with model parameters

$$w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^p, \quad W \in \mathbb{R}^{p \times p}. \tag{53}$$

Comparing this to the FM model (Eq. (1)) one can see that FMs use a factorization for pairwise interactions, whereas polynomial regression uses an independent parameter $w_{j,j'}$ per pairwise interaction. This difference is crucial for the success of FMs in sparse settings, for example, recommender systems or other prediction problems involving categorical variables of large domain. FMs can estimate pairwise interactions $w_{j,j'}$ for pairs (j, j') even if none or only little observation about the pair is present, because a low-rank assumption is made ($w_{j,j'} \approx \langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle$), that is, it is assumed that the interaction of pair (j, j') and (j, j^*) have something in common. In polynomial regression, both pairs are completely independent (a priori).

5. LIBFM SOFTWARE

LIBFM[1] is an implementation of factorization machines. It includes the SGD, ALS, and MCMC algorithms described in Section 3 3 for regression and classification tasks. FMs of order $d = 2$ are implemented.

5.1. Data Format

The input data format of LIBFM is the same as for SVM^{light} [Joachims 1999] and LIBSVM [Chang and Lin 2011]. For very large-scale data that does not fit into main memory, LIBFM has a binary data format where only parts have to be kept in main memory. A converter from the standard text format to the binary data format is available.

5.2. Example

All major options are available over an easy-to-use command line interface. An example call for learning a dataset with MCMC inference would be

```
./libFM -method mcmc -task r -dim '1;1;8' -init_stdev 0.1 -iter 100
-test ml1m-test.libfm -train ml1m-train.libfm -out ml1m-test.pred,
```

where `dim` specifies the factorization dimensions: 0/1 if w_0 should be included, 0/1 if $\mathbf{w}$ should be included, and $k \in \mathbb{N}$ (here $k = 8$) for the dimensionality of V. `init_stdev` is the standard deviation for initialization, that is, σ of Algorithm 3. `iter` is the number of samples that are drawn.

5.3. Parameter Setup

In the following, a few practical hints for applying LIBFM to a prediction problem are given.

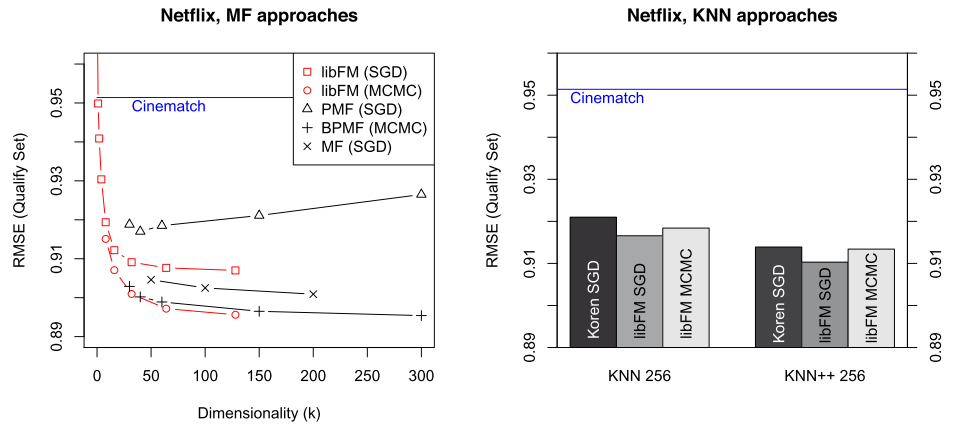
(1) For inexperienced users, it is advisable to use MCMC inference, as it is the most simple one to work with.
(2) When a predictive model for a new dataset is built, one should start with a low factorization dimensionality (e.g., $k = 8$) and first determine the standard deviation for initialization (`-init_stdev`), because proper values will speed-up the MCMC sampler.
(3) Several values for `init_stdev` should be tested (e.g., 0.1, 0.2, 0.5, 1.0). The success can be quickly seen on the first few iterations by monitoring the training error or, even better by using a holdout set for validation.
(4) After an appropriate `init_stdev` has been determined, MCMC can be run with a large number of iterations and larger factorization dimensionality k. The accuracy and convergence can be monitored on the output of libFM.

For MCMC, no other hyperparameters have to be specified. Other methods (ALS and SGD) require more hyperparameters to tune (see Section 3 3).

5.4. Ranking

LIBFM also contains methods for optimizing an FM model with respect to ranking [Liu and Yang 2008] based on pairwise classification [Rendle et al. 2009]. Ranking is not available over the command line but can be embedded into existing software. An example for embedding LIBFM is provided in the software tool *Tag Recommender* (also available with source code).

[1]Available with source code from `http://www.libfm.org/`.



(a) Matrix factorization (MF). (b) Nearest neighborhood (KNN).

Fig. 3. Prediction error of LIBFM with SGD and MCMC learning using MF (see Eq. 40) and KNN (see Eq. 49) input data. (a) Compare to the MF approaches PMF (SGD) [Salakhutdinov and Mnih 2008b], BPMF (MCMC) [Salakhutdinov and Mnih 2008a], and MF (SGD) [Koren 2008]. (b) Compare to the corresponding KNN approach of Koren [2008].

6. EVALUATION

In Section 4.1, it was shown that FMs are able to mimic many factorization models. Now this will be substantiated by comparing the LIBFM implementation empirically to several well studied factorization models. The success will be measured by root mean-square error (RMSE) for regression tasks and F1-measure for ranking tasks (see Gunawardana and Shani [2009] for a summary of evaluation metrics for recommender systems).

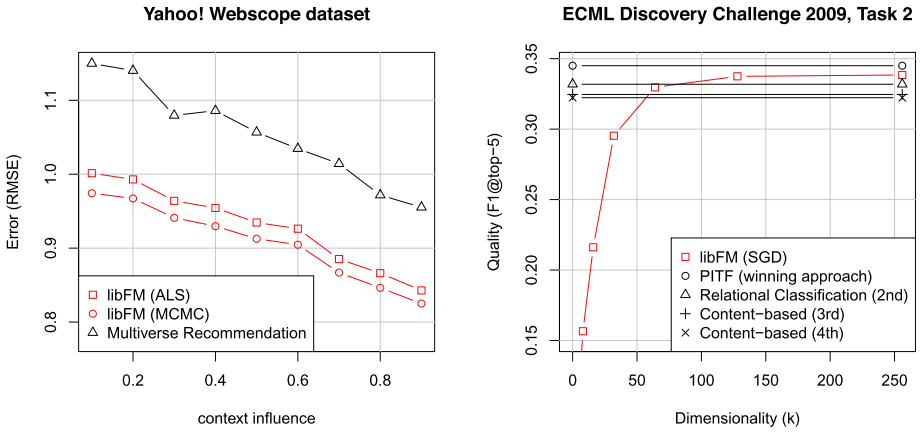
6.1. Rating Prediction

In recommender systems, the most well studied data set is the Netflix challenge[2] which includes about 100,000,000 ratings of about 480,000 users for 17,770 items. All our reported results are obtained on the Netflix *quiz* set (i.e., the same test set as on the public leaderboard from the Netflix challenge).

6.1.1. Matrix Factorization (MF). The most successful approaches on Netflix are based on matrix factorization (e.g., [Jahrer et al. 2010; Koren 2009a; Takács et al. 2009]). For MF, many different variations and learning approaches have been proposed, for example, ALS [Pilászy et al. 2010], MCMC [Salakhutdinov and Mnih 2008a], Variational Bayes [Lim and Teh 2007; Stern et al. 2009], but mostly SGD variants (e.g., [Koren 2008; Salakhutdinov and Mnih 2008b]). Thus, even for the simple MF model, the prediction quality reported differ largely. We want to investigate how good the learning methods of LIBFM are by setting up an FM with MF indicators (see Eq. (40)), which is equivalent to biased MF. With this setting, all compared approaches share the same model but differ in learning algorithm and implementation.

Figure 3(a) shows a comparison of LIBFM (using SGD and MCMC[3]) to the SGD approaches of PMF [Salakhutdinov and Mnih 2008b] and the MF (SGD) approach of [Koren 2008], as well as the BPMF approach using MCMC inference [Salakhutdinov

[2]http://www.netflixprize.com/
[3]FM (SGD) results are from Rendle [2012], FM (MCMC) results from Freudenthaler et al. [2011].



(a) Context-aware recommendation. (b) Tag recommendation.

Fig. 4. LIBFM for context-aware recommendation compared to Multiverse Recommendation [Karatzoglou et al. 2010]. Compares LIBFM for the task of tag recommendation to the best four approaches on Task 2 of the ECML/PKDD Discovery Challenge 2009.

and Mnih 2008a]. For LIBFM with SGD, we use the regularization values ($\lambda_\theta = 0.04$) that have been reported in Koren [2008] for the related SVD++ model. It can be seen that the MCMC approaches have the lowest error, and the MCMC sampler of LIBFM outperforms the sampler of the BPMF model slightly.

6.1.2. Nearest Neighbor Models. Traditionally, nearest neighbor models have attracted a lot of research in the recommender system community (e.g., [Linden et al. 2003; Sarwar et al. 2001; Zheng and Xie 2011]). On the Netflix challenge, the best performing neighborhood approaches are based on treating the similarities between items as model parameters which are learned, that is, KNN (Eq. (49)) [Koren 2008] and factorized KNN (Eq. (48)) [Koren 2010]. Again, we want to see how well LIBFM can mimic these models just by feature engineering. We set up the input data for LIBFM such that the FM model corresponds to the KNN and KNN++ (i.e., with additional implicit indicators) as described in Koren [2008]. We use the same pruning protocol to restrict to 256 neighbors, and for SGD, we use the same regularization ($\lambda_\theta = 0.002$) as reported in Koren [2008]. Figure 3(b) shows that LIBFM with MCMC and SGD achieves comparable quality to the approach of Koren [2008].

6.2. Context-Aware Recommendation

Secondly, LIBFM has been studied on the problem of context-aware recommendation [Rendle et al. 2011]. In context-aware recommendation, besides the user and item, there is other information about the rating event available, for example, the location of the user at the time of his rating, the mood, etc. As FMs can work with any number of features, they can be applied easily to this task. Figure 4(a)[4] shows a comparison of LIBFM using ALS and MCMC to the state-of-the art approach *Multiverse Recommendation* [Karatzoglou et al. 2010], which outperforms other context-aware methods, such as item splitting [Baltrunas and Ricci 2009] and the multidimensional approach of Adomavicius et al. [2005].

[4]Please see Karatzoglou et al. [2010] and Rendle et al. [2011] for details about the experimental setup.

6.3. Tag Recommendation

The last experiment shows the applicability of LIBFM to ranking. We compare LIBFM for the task of tag recommendation (e.g., Lipczak and Milios [2011]) using input data, as in Eq. (42). With this data, LIBFM mimics the PITF model [Rendle and Schmidt-Thieme 2010] which was the best performing approach[5] on Task 2 of the ECML/ PKDD Discovery Challenge 2009. Figure 4(b) shows a comparison of the prediction quality of LIBFM to PITF and the second to fourth best models of the Discovery Challenge: relational classification [Marinho et al. 2009] and the models of the third [Lipczak et al. 2009] and fourth place [Zhang et al. 2009].

7. CONCLUSION AND FUTURE WORK

Factorization machines (FM) combine the flexibility of feature engineering with factorization models. This article summarizes the recent research on FMs and presents three efficient inference methods based on SGD, ALS, and MCMC. Also extensions are presented, among them classification for MCMC and ALS, as well as grouping of variables.

The properties of FMs have been discussed both theoretically in terms of complexity, expressiveness, and relations to other factorization models, as well as with an empirical evaluation. It has been shown that FMs can mimic several specialized factorization models—for certain, FMs are not restricted to these examples. Empirical results show that the prediction quality of the described inference algorithms for FMs is comparable to the best inference approaches for specialized models in the area of recommender systems. In total, that means that the generality of FMs does not come to the prize of a low prediction accuracy or a high computational complexity. All presented algorithms are implemented in the publicly available software tool LIBFM.

There are several directions for future work on FMs. First, due to the generality of FMs, they are supposed to be interesting for a wide variety of prediction problems, especially problems involving categorical variables with large domains might benefit from FMs. Studying FMs using LIBFM on such problems is highly interesting. Second, the complexity of the inference methods for FMs could be reduced, because the algorithms proposed so far make no use of repeating patterns in the input data which could be exploited for an additional speed-up. Third, the software implementation LIBFM could be extended by higher-order interactions ($d \geq 3$).

ACKNOWLEDGMENTS

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

REFERENCES

ADOMAVICIUS, G., SANKARANARAYANAN, R., SEN, S., AND TUZHILIN, A. 2005. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Info. Syst. 23,* 1, 103–145.

AGARWAL, D. AND CHEN, B.-C. 2009. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. ACM, New York, NY, 19–28.

BALTRUNAS, L. AND RICCI, F. 2009. Context-based splitting of item ratings in collaborative filtering. In *Proceedings of the third ACM Conference on Recommender Systems (RecSys'09)*. ACM, New York, NY, 245–248.

[5]The winning model was an ensemble of several PITF models and a postprocessing step. For a fair comparison, here the F1-score for a single PITF model without postprocessing is reported. Both ensembling and postprocessing could be done the same way for the LIBFM predictions, as well.

CHANG, C.-C. AND LIN, C.-J. 2011. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol. 2*, 27:1–27:27.

CHEN, T., ZHENG, Z., LU, Q., ZHANG, W., AND YU, Y. 2011. Feature-based matrix factorization. Tech. rep. APEX-TR-2011-07-11, Apex Data & Knowledge Management Lab, Shanghai Jiao Tong University.

FREUDENTHALER, C., SCHMIDT-THIEME, L., AND RENDLE, S. 2011. Bayesian factorization machines. In *Proceedings of the NIPS Workshop on Sparse Representation and Low-rank Approximation*.

GANTNER, Z., DRUMOND, L., FREUDENTHALER, C., RENDLE, S., AND LARS, S.-T. 2010. Learning attribute-to-feature mappings for cold-start recommendations. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'10)*. IEEE Computer Society, Los Alamintos, CA, 176–185.

GELMAN, A., CARLIN, J. B., STERN, H. S., AND RUBIN, D. B. 2003. *Bayesian Data Analysis* 2nd Ed. Chapman and Hall/CRC.

GUNAWARDANA, A. AND SHANI, G. 2009. A survey of accuracy evaluation metrics of recommendation tasks. *J. Mach. Learn. Res. 10*, 2935–2962.

HALL, M., FRANK, E., HOLMES, G., PFAHRINGER, B., REUTEMANN, P., AND WITTEN, I. H. 2009. The weka data mining software: An update. *SIGKDD Explor. Newsl. 11*, 10–18.

HARSHMAN, R. A. 1970. Foundations of the parafac procedure: Models and conditions for an 'exploratory' multimodal factor analysis. *UCLA Working Papers in Phonetics*, 1–84.

JAHRER, M., TÖSCHER, A., AND LEGENSTEIN, R. 2010. Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'10)*. ACM, New York, NY, 693–702.

JOACHIMS, T. 1999. *Making Large-Scale Support Vector Machine Learning Practical*. MIT Press, Cambridge, MA, 169–184.

KARATZOGLOU, A., AMATRIAIN, X., BALTRUNAS, L., AND OLIVER, N. 2010. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the 4th ACM Conference on Recommender Systems (RecSys'10)*. ACM, New York, NY, 79–86.

KOREN, Y. 2008. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*. ACM, New York, NY, 426–434.

KOREN, Y. 2009a. The bellkor solution to the Netflix grand prize.

KOREN, Y. 2009b. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. ACM, New York, NY, 447–456.

KOREN, Y. 2010. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Trans. Knowl. Discov. Data 4*, 1:1–1:24.

LIM, Y. J. AND TEH, Y. W. 2007. Variational Bayesian approach to movie rating prediction. In *Proceedings of the KDD Cup and Workshop*.

LINDEN, G., SMITH, B., AND YORK, J. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *Inter. Comput. IEEE 7,* 1, 76–80.

LIPCZAK, M., HU, Y., KOLLET, Y., AND MILIOS, E. 2009. Tag sources for recommendation in collaborative tagging systems. In *Proceedings of the ECML-PKDD Discovery Challenge Workshop*.

LIPCZAK, M. AND MILIOS, E. 2011. Efficient tag recommendation for real-life data. *ACM Trans. Intell. Syst. Technol. 3,* 1, 2:1–2:21.

LIU, N. N. AND YANG, Q. 2008. Eigenrank: A ranking-oriented approach to collaborative filtering. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'08)*. ACM, New York, NY, 83–90.

MA, H., KING, I., AND LYU, M. R. 2011. Learning to recommend with explicit and implicit social relations. *ACM Trans. Intell. Syst. Technol.* Article 29.

MARINHO, L. B., PREISACH, C., AND SCHMIDT-THIEME, L. 2009. Relational classification for personalized tag recommendation. In *Proceedings of the ECML-PKDD Discovery Challenge Workshop*.

PATEREK, A. 2007. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of the KDD Cup Workshop 13th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'07)*. 39–42.

PILÁSZY, I., ZIBRICZKY, D., AND TIKK, D. 2010. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the 4th ACM Conference on Recommender Systems (RecSys'10)*. ACM, New York, NY, 71–78.

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

RENDLE, S. 2012. Learning recommender systems with adaptive regularization. In *Proceedings of the 5th ACM International Conference on Web Search and Data Mining (WSDM'12)*. ACM, New York, NY, 133–142.

RENDLE, S., FREUDENTHALER, C., GANTNER, Z., AND SCHMIDT-THIEME, L. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI09)*.

RENDLE, S., FREUDENTHALER, C., AND SCHMIDT-THIEME, L. 2010. Factorizing personalized Markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*. ACM, New York, NY, 811–820.

RENDLE, S., GANTNER, Z., FREUDENTHALER, C., AND SCHMIDT-THIEME, L. 2011. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th ACM SIGIR Conference on Reasearch and Development in Information Retrieval*.

RENDLE, S. AND SCHMIDT-THIEME, L. 2010. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the third ACM International Conference on Web Search and Data Mining (WSDM'10)*. ACM, New York, NY, 81–90.

ROBERT, C. P. 1995. Simulation of truncated normal variables. *Stat. Comput. 5*, 121–125.

SALAKHUTDINOV, R. AND MNIH, A. 2008a. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the 25th International Conference on Machine Learning*.

SALAKHUTDINOV, R. AND MNIH, A. 2008b. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20*.

SARWAR, B., KARYPIS, G., KONSTAN, J., AND RIEDL, J. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*. ACM Press, New York, NY, 285–295.

SREBRO, N. AND JAAKKOLA, T. 2003. Weighted low rank approximation. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*.

SREBRO, N., RENNIE, J. D. M., AND JAAKOLA, T. S. 2005. Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems 17*, MIT 1329–1336.

STERN, D. H., HERBRICH, R., AND GRAEPEL, T. 2009. Matchbox: Large-scale online Bayesian recommendations. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*. ACM, New York, NY, 111–120.

TAKÁCS, G., PILÁSZY, I., NÉMETH, B., AND TIKK, D. 2009. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res. 10*, 623–656.

TUCKER, L. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika 31*, 279–311.

XIONG, L., CHEN, X., HUANG, T.-K., SCHNEIDER, J., AND CARBONELL, J. G. 2010. Temporal collaborative filtering with Bayesian probabilistic tensor factorization. In *Proceedings of the SIAM International Conference on Data Mining (SIAM)*. 211–222.

ZHANG, N., ZHANG, Y., AND TANG, J. 2009. A tag recommendation system based on contents. In *Proceedings of the ECML-PKDD Discovery Challenge Workshop*.

ZHENG, Y. AND XIE, X. 2011. Learning travel recommendations from user-generated gps traces. *ACM Trans. Intell. Syst. Technol. 2*, 2:1–2:29.

Received January 2012; revised January 2012; accepted February 2012