Scalable Collaborative Bayesian Preference Learning

Mohammad Emtiyaz Khan **Young Jun Ko** **Matthias Seeger**
École Polytechnique Fédérale de Lausanne, Switzerland

Abstract

Learning about users' utilities from preference, discrete choice or implicit feedback data is of integral importance in e-commerce, targeted advertising and web search. Due to the sparsity and diffuse nature of data, Bayesian approaches hold much promise, yet most prior work does not scale up to realistic data sizes. We shed light on why inference for such settings is computationally difficult for standard machine learning methods, most of which focus on predicting explicit ratings only. To simplify the difficulty, we present a novel expectation maximization algorithm, driven by expectation propagation approximate inference, which scales to very large datasets without requiring strong factorization assumptions. Our utility model uses both latent bilinear collaborative filtering and non-parametric Gaussian process (GP) regression. In experiments on large real-world datasets, our method gives substantially better results than either matrix factorization or GPs in isolation, and converges significantly faster.

1 Introduction

We are interested in modelling datasets of N users expressing preferences or making discrete choices among M items, with the goal to predict utility rankings and to make informed recommendations. Scenarios of this kind are manifold in targeted advertising, social networks, and e-commerce (where items are movies in a video streaming portal or books, video games and other merchandise sold online). They come with a range of challenges. Preference statements, partial rankings or other implicit feedback (e.g., number of clicks or view durations) are diffuse, uncalibrated sources of information, relative at best, while traditional machine learning frameworks are focused on predicting explicit targets with controlled range and noise levels. However, while users are unwilling to provide much explicit feedback, preference statements are linked to user behavior which can easily be recorded on the fly. Due to the very large scale of many applications (with N and M in the millions), any conceivable dataset is extremely sparse: the vast majority of user-item pairs receives no coverage at all. Any competitive approach has to capture dependencies in this highly incomplete matrix.

Indirect observations from diverse sources, data sparsity and richness in latent structure strongly motivate approaches grounded in Bayesian statistics. Latent bilinear factors combined with nonparametric regression from meta-data provide flexible utility models, which can be linked to observations through a wide range of likelihood functions. However, Bayesian inference is analytically intractable, and most current approximations do not scale up to applications of interest. In this paper, we clarify sources of computational difficulties when applying standard Bayesian inference to preference or discrete choice modelling. Our arguments shed light on why the "proxy problem" of predicting explicit ratings receives so much more attention.

Drawing on ideas from [24], we show how these difficulties can be circumvented by a non-standard expectation maximization (EM) approach. Our algorithm applies to models which combine bilinear latent factors and Gaussian processes, yet remains operational in a regime far beyond standard variational EM methodology. Driven by expectation propagation (EP) or other local variational approximations, it can be used with a wide range of likelihood functions. It comes with simple closed-form updates, is easily implemented, and its dominating computations are embarrassingly parallelizable. We compare our method to prior work on a number of real-world datasets, ranging from a few thousand to about 3 million observations, and obtain significant improvements in substantially less time.

Appearing in Proceedings of the 17^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2014, Reykjavik, Iceland. JMLR: W&CP volume 33.

2 Bayesian Models for Preferences

We index the N users by n and the M items by i. Each user is associated with a latent *utility function* $\boldsymbol{z}_n$ over items. A large value of z_{in} indicates a high interest in item i. A discrete choice model consists of a specification of $\boldsymbol{z}_n$, possibly dependent on covariates, and a likelihood function linking utilities to observations.

For the sake of simplicity, we focus only on pairwise preferences, yet the methods discussed here apply to other data types as well (including implicit feedback and discrete choice data). We observe P_n preference pairs $\mathbb{O}_n = \{(i,j) : i \succ j\}$ for user n, where $i \succ j$ states that item i is preferred over j. A common model for such observations is the *probit* likelihood [3, 8] defined as $p(i \succ j|\boldsymbol{z}_n) = \Phi(z_{in} - z_{jn})$, where Φ is the standard Gaussian cumulative distribution function. Another popular option is the *logit* likelihood $p(i \succ j|\boldsymbol{z}_n) = (1+e^{-(z_{in}-z_{jn})})^{-1}$ [11]. We also observe an item feature-vector $\boldsymbol{x}_i$ per item i.

Utilities can conveniently be modelled via Gaussian processes (GP) [2, 3, 8], $z_{in} = z_n(\boldsymbol{x}_i)$, where $z_n(\cdot) \sim \mathrm{GP}(0, K)$ and $K(\boldsymbol{x}, \boldsymbol{x}')$ a covariance function between items. Some of the early work was focused on a single user, e.g. [5, 4]. Multi-user preference learning based on a hierarchical model with dependent GP priors is proposed in [2]. Given item and user features, the utility function can also be modelled by a joint GP prior of tensor product structure [3]. A recent approach by [8] models utility differences $z_{in} - z_{jn}$ directly using linear combination of GPs. Unfortunately, these methods do not scale to datasets with more than a few hundred items, e.g., the method in [3] scales cubically in the number of observed pairs, that of [8] scales cubically in the number of distinct pairs. For modern datasets containing millions of pairs, this is not acceptable.

Another popular approach is to use matrix factorization [18], with utilities $z_{in} = \boldsymbol{v}_i^T \boldsymbol{u}_n + \mu_i$, where $\boldsymbol{v}_i, \boldsymbol{u}_n$ are length D latent factors and μ_i is a bias parameter for item i. Denote $\boldsymbol{U} = [\boldsymbol{u}_n]^T \in \mathbb{R}^{N \times D}$, $\boldsymbol{V} = [\boldsymbol{v}_i]^T \in \mathbb{R}^{M \times D}$. Item features can be incorporated by adding regression factors [20, 1], e.g., by defining $z_{in} = \boldsymbol{v}_i^T \boldsymbol{u}_n + \mu_i + \boldsymbol{x}_i^T \boldsymbol{w}_n$, where $\boldsymbol{w}_n$ are regression weights. Matrix factorization models work particularly well for users or items which are well supported by data, but can fail to generalize well to those with very few observations (the "cold-start problem"), and appending a regression term to the utility function can be particularly beneficial in these cases.

Although this bilinear modelling approach comes with highly scalable algorithms for *explicit* ratings data, these do not apply to preference observations. For one, sensible preference likelihoods are non-Gaussian, and approximate inference techniques have to be used. However, the main computational burden is present even if Gaussian likelihoods are used, and arises from the asymmetry between items and users: observations couple z_{in}, z_{jn} for different items i and j, which makes estimating $\boldsymbol{V}$ difficult. As shown in the Appendix, in an EM algorithm that integrates over $\boldsymbol{U}$ and estimates $\boldsymbol{V}$, the M-step involves minimizing a quadratic function $\boldsymbol{v}^T \boldsymbol{A} \boldsymbol{v} + \boldsymbol{a}^T \boldsymbol{v}$, where $\boldsymbol{v} = \mathrm{vec}(\boldsymbol{V}) \in \mathbb{R}^{MD}$ and $\boldsymbol{A}$ is an unstructured matrix of size $MD \times MD$. This problem is highlighted in [21] for the case of Gaussian likelihoods.

Common solutions to circumvent these problems include factorizing the posterior [18] or using variational inference based on crude local variational bounds [11]. In particular in data-scarce situations, the implied loss of accuracy is considerable. Moreover, factorization assumptions do not work well for GP models. For methods based on EP, factorization assumptions also compromise algorithmic convergence.

In this paper, we show that neither factorization assumptions nor sub-standard inference approximations are really required for scalable Bayesian multi-user preference learning. We make use of a utility model featuring *both* matrix factorization and GP terms, which allows for significantly better results than either in isolation. Building on ideas from [24], we devise an efficient EM algorithm. Our method comes with a closed-form M step update of $\boldsymbol{V}$, based on statistics of size M^2 only (compared to M^2D^2 in the standard EM algorithm). Our algorithm generalizes to many likelihoods, allowing us to model different types of data.

3 A New Utility Model

We propose a new utility model, containing both matrix factorization and GP parts, as shown below. Here, μ_i is an item-specific bias, and $s_n(\cdot)$ is drawn from a GP with covariance function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $\boldsymbol{u}_n$ is drawn from a standard Gaussian distribution.

$$z_{in} = \boldsymbol{v}_i^T \boldsymbol{u}_n + \mu_i + s_n(\boldsymbol{x}_i), \quad \boldsymbol{s}_n \sim \mathrm{GP}(0, \sigma_s^2 K) \qquad (1)$$
$$\boldsymbol{u}_n \sim \mathcal{N}(0, \boldsymbol{I}), \quad p(\mathbb{O}_n|\boldsymbol{z}_n) = \prod_{(i,j)\in\mathbb{O}_n} \boldsymbol{\Phi}(z_{in} - z_{jn})$$

The observation set $\mathbb{O}_n$ of P_n preference pairs from the n-th user is modeled using the likelihood function shown above. These observations strongly couple elements of $\boldsymbol{z}_n$ since each likelihood depends on z_{in} and z_{jn}. We denote these couplings by a sparse matrix $\boldsymbol{B}_n$ of size $P_n \times M$. A row of this matrix pair correspond to an observed pair $i \succ j$, and has all zero except for $+1$ at i and -1 at j. The likelihood of $\mathbb{O}_n$ is then completely specified by a vector $\tilde{\boldsymbol{z}}_n = \boldsymbol{B}_n \boldsymbol{z}_n$.

Algorithm 1 EM for collaborative preference learning

Initialize $\boldsymbol{V}, \boldsymbol{\mu}, \sigma_s^2$.
Compute the Kernel matrix $\boldsymbol{K}$ and its Cholesky $\boldsymbol{L}$.
Initialize $\boldsymbol{\Sigma} \leftarrow \boldsymbol{V}\boldsymbol{V}^T + \sigma_s^2\boldsymbol{K}$ and $\boldsymbol{W} \leftarrow \boldsymbol{L}^{-1}\boldsymbol{V}$
while not converged **do**
  % E-step in parallel
  Initialize global statistics $\boldsymbol{s}$ and $\boldsymbol{S}$.
  for all users $n = 1, \ldots, N$, in parallel **do**
    Run EP to compute $\boldsymbol{\beta}_n$, $\boldsymbol{\pi}_n$ as in (4).
    Compute $\boldsymbol{E}_n$ and $\boldsymbol{e}_n$ using (7) and (10).
    Add these to $\boldsymbol{s}$ and $\boldsymbol{S}$ as in (13) and (15).
  end for
  % M-step
  $[\boldsymbol{W}, \sigma_s^2] \leftarrow \texttt{fastEigs}(\boldsymbol{W}, \sigma_s^2, \boldsymbol{L}, \boldsymbol{S})$. See Sec. 4.
  Update $\boldsymbol{V} \leftarrow \boldsymbol{L}\boldsymbol{W}$ and $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{s}$.
  Update $\boldsymbol{\Sigma} \leftarrow \boldsymbol{V}\boldsymbol{V}^T + \sigma_s^2\boldsymbol{K}$
end while

4 Scalable Variational EM Algorithm

We will now derive a scalable variational EM algorithm for our model. We will integrate out the $\boldsymbol{u}_n$ and $\boldsymbol{s}_n$, and estimate the parameters $\boldsymbol{\theta} = \{\boldsymbol{V}, \boldsymbol{\mu}, \sigma_s^2\}$. Estimating $\boldsymbol{V}$ (rather than integrating it out) leads to a computationally efficient algorithm, a choice which is justified if N is reasonably larger than M. For simplicity, we do not learn the Kernel hyperparameters within our framework, but note that the prior variance σ_s^2 is learned at no additional computational cost.

We seek θ that maximizes the marginal likelihood:

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} \int p(\mathbb{O}_n|\tilde{\boldsymbol{z}}_n) p(\boldsymbol{u}_n) p(\boldsymbol{s}_n)\, d\boldsymbol{u}_n d\boldsymbol{s}_n$$

The key idea towards an efficient algorithm is to work with $\boldsymbol{z}_n$ and $\tilde{\boldsymbol{z}}_n$, instead of $(\boldsymbol{u}_n, \boldsymbol{s}_n)$:

$$p(\boldsymbol{z}_n|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{z}_n|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad \boldsymbol{\Sigma} := \sigma_s^2\boldsymbol{K} + \boldsymbol{V}\boldsymbol{V}^T \tag{2}$$

$$p(\tilde{\boldsymbol{z}}_n|\boldsymbol{\theta}) = \mathcal{N}(\tilde{\boldsymbol{z}}_n|\boldsymbol{B}_n\boldsymbol{\mu}, \boldsymbol{B}_n\boldsymbol{\Sigma}\boldsymbol{B}_n^T) \tag{3}$$

Importantly, all parameters $\boldsymbol{\theta}$ reside in the prior $p(\boldsymbol{z}_n)$, while the likelihoods $p(\mathbb{O}_n|\tilde{\boldsymbol{z}}_n)$ are parameter-free. In an EM algorithm, this leads to an efficient M-step. Each E-step is also simplified by working with $\tilde{\boldsymbol{z}}_n$ which are of much shorter length compared to $\boldsymbol{z}_n$ (since $P_n \ll M$). The EM algorithm is outlined in Algorithm 1. We give details for each of its steps now.

Expectation Propagation. The first obvious difficulty is the intractable integral due to the non-Gaussian likelihoods. We address this issue by EP [13]. In a nutshell, likelihoods $p(\mathbb{O}_n|\tilde{\boldsymbol{z}}_n)$ are replaced by Gaussian functions giving rise to a Gaussian approximation, as shown below.

$$q(\boldsymbol{z}_n) \propto e^{\boldsymbol{\beta}_n^T\tilde{\boldsymbol{z}}_n - \frac{1}{2}\tilde{\boldsymbol{z}}_n^T \operatorname{diag}(\boldsymbol{\pi}_n)\tilde{\boldsymbol{z}}_n}\mathcal{N}(\boldsymbol{z}_n|\boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{4}$$

The EP parameters $\boldsymbol{\beta}_n$, $\boldsymbol{\pi}_n$ are adjusted by a simple iterative algorithm (see [6] for a detailed algorithm).

In our framework, EP runs efficiently since only marginals of $q(\tilde{\boldsymbol{z}}_n)$ are required which are of much smaller size than those of $q(\boldsymbol{z}_n)$ (see Eq. (3)). We use parallel updating[1] EP [6], iterating between recomputing marginals and doing all local EP updates in parallel. The former scales as $O(P_n^3)$, while the latter is a simple operation for the probit link [13]. In our experiments, EP converged after few (5-7) parallel rounds in nearly all instances. This fast and robust convergence is presumably due to no factorization assumptions on the posterior. In contrast, the method of [18] requires sequential updating, damping and careful message scheduling to ensure EP convergence.

Note that both the probit and logit likelihoods are log-concave, and EP updates are guaranteed to be well-defined in exact arithmetic and result in positive entries in $\boldsymbol{\pi}_n$ [19]. On the other hand, EP can be problematic for numerical reasons, in particular in the faster parallel updating variant. In our experiments, we also consider a simpler local variational inference approximation due to Jaakkola [9], which applies to the logit likelihood (replace probit in Eq. (1) by logit). This approximation is numerically stable, but can perform worse than EP [16]. In our context here, it can be seen as a different way of choosing $\boldsymbol{\pi}_n$, $\boldsymbol{\beta}_n$, so all other computations remain unchanged.

Computation of Sufficient Statistics. The EM criterion depends on statistics of $q(\boldsymbol{z}_n)$, but fortunately, these can be expressed in terms of much smaller statistics of $q(\tilde{\boldsymbol{z}}_n)$. Since EP approximates the non-Gaussian likelihoods by a Gaussian, the resulting statistics are similar to Gaussian PPCA [22] and are shown here in Eq. (5) and (8). In Eq. (6) and (9), we use the Woodbury formula to express the statistics in terms of a P_n length vector $\boldsymbol{e}_n$ and a square matrix $\boldsymbol{E}_n$ of size $P_n \times P_n$. These are defined in terms of $\boldsymbol{\pi}_n$ and $\boldsymbol{\beta}_n$ in Eq. (7) and (10).

$$\operatorname{Cov}(\boldsymbol{z}_n) = [\boldsymbol{\Sigma}^{-1} + \boldsymbol{B}_n \operatorname{diag}(\boldsymbol{\pi}_n)\boldsymbol{B}_n^T]^{-1} \tag{5}$$

$$= \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\boldsymbol{B}_n^T\boldsymbol{E}_n\boldsymbol{B}_n\boldsymbol{\Sigma} \tag{6}$$

$$\text{where } \boldsymbol{E}_n^{-1} = \operatorname{diag}(\boldsymbol{\pi}_n)^{-1} + \boldsymbol{B}_n\boldsymbol{\Sigma}\boldsymbol{B}_n^T \tag{7}$$

$$\mathbb{E}(\boldsymbol{z}_n) = \operatorname{Cov}(\boldsymbol{z}_n)[\boldsymbol{B}_n^T\boldsymbol{\beta}_n + \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}] \tag{8}$$

$$= \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{B}_n^T\boldsymbol{e}_n \tag{9}$$

$$\text{where } \boldsymbol{e}_n = \boldsymbol{E}_n[\operatorname{diag}(\boldsymbol{\pi}_n)^{-1}\boldsymbol{\beta}_n - \boldsymbol{B}_n\boldsymbol{\mu}] \tag{10}$$

Computing $\boldsymbol{E}_n$ and $\boldsymbol{e}_n$ requires $O(P_n^3)$ computations only. We give expressions for numerically safe computation of these updates in the supplementary material.

[1] In contrast, sequential updating EP as in [13] would run much slower in our context.

Fortunately, we never have to compute $\mathrm{Cov}(\boldsymbol{z}_n)$ and $\mathbb{E}(\boldsymbol{z}_n)$ explicitly, as we show next.

M-step criterion. The criterion to be minimized depends on moments of $q(\boldsymbol{z}_n)$, as shown below.

$$\min_\theta \sum_{n=1}^N \mathbb{E}_{q(\boldsymbol{z}_n)}[-\log p(\boldsymbol{z}_n|\theta)] \tag{11}$$

$$= \frac{N}{2}\left[\log|2\pi\boldsymbol{\Sigma}| + \mathrm{tr}(\boldsymbol{\Sigma}^{-1}\boldsymbol{C})\right] \tag{12}$$

$$\boldsymbol{C} := \frac{1}{N}\sum_{n=1}^N \mathrm{Cov}(\boldsymbol{z}_n) + [\boldsymbol{\mu} - \mathbb{E}(\boldsymbol{z}_n)][\boldsymbol{\mu} - \mathbb{E}(\boldsymbol{z}_n)]^T$$

However, the updates can be expressed in terms of moments of $q(\tilde{\boldsymbol{z}}_n)$, avoiding explicit computation of $q(\boldsymbol{z}_n)$. We show this first for $\boldsymbol{\mu}$. Setting the derivative with respect to $\boldsymbol{\mu}$ to zero, gives us the update shown below. We use Eq. (9) to get the second equality.

$$\hat{\boldsymbol{\mu}} = \sum_{n=1}^N \frac{\mathbb{E}(\boldsymbol{z}_n)}{N} = \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{s}, \text{with } \boldsymbol{s} := \sum_{n=1}^N \frac{\boldsymbol{B}_n\boldsymbol{e}_n}{N} \tag{13}$$

To update $\boldsymbol{V}$ and σ_s^2, we first plug the $\hat{\boldsymbol{\mu}}$ in the objective (12), to get the following new value of $\boldsymbol{C}$ which depends only on $\boldsymbol{e}_n$ and $\boldsymbol{E}_n$ through $\boldsymbol{S}$, defined in Eq. (15) (see detailed derivation in the Appendix).

$$\boldsymbol{C} := \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\boldsymbol{S}\boldsymbol{\Sigma} \tag{14}$$

$$\boldsymbol{S} := \frac{1}{N}\sum_n \boldsymbol{B}_n^T(\boldsymbol{E}_n - \boldsymbol{e}_n\boldsymbol{e}_n^T)\boldsymbol{B}_n + \boldsymbol{s}\boldsymbol{s}^T \tag{15}$$

All we need from the E-step computation for user n are statistics $\boldsymbol{e}_n$ and $\boldsymbol{E}_n$ of size $O(P_n^2)$. The global statistics $\boldsymbol{s}$, $\boldsymbol{S}$ are easily accumulated alongside the E-step computations. We will now derive updates of $\boldsymbol{V}$ and σ_s^2, without explicitly forming $\boldsymbol{C}$.

Update for $\boldsymbol{V}$ and σ_s^2. The objective of Eq. 12 can be minimized in closed form for these parameters as well. Let $\boldsymbol{L}$ be the Cholesky of $\boldsymbol{K} + \varepsilon\boldsymbol{I}$. Note that we add a jitter[2]term $\varepsilon > 0$ in order to improve the condition number of the matrix [15]. Defining $\boldsymbol{W} := \boldsymbol{L}^{-1}\boldsymbol{V}$ and $\tilde{\boldsymbol{C}} := \boldsymbol{L}^{-1}\boldsymbol{C}\boldsymbol{L}^{-T}$, we can rewrite the objective:

$$\min_\theta \log|\tilde{\boldsymbol{\Sigma}}| + \mathrm{tr}(\tilde{\boldsymbol{\Sigma}}^{-1}\tilde{\boldsymbol{C}}), \quad \tilde{\boldsymbol{\Sigma}} := \sigma_s^2\boldsymbol{I} + \boldsymbol{W}\boldsymbol{W}^T \tag{16}$$

This is the negative log likelihood for a probabilistic PCA model, which can be solved analytically [22]. Let $\boldsymbol{\Lambda}$ be the diagonal matrix of the D largest eigenvalues of $\tilde{\boldsymbol{C}}$, and $\boldsymbol{R}$ be the matrix of corresponding orthonormal eigenvectors. A global minimizer of (16) is

$$\hat{\boldsymbol{W}} = \boldsymbol{R}(\boldsymbol{\Lambda} - \hat{\sigma}_s^2\boldsymbol{I})^{1/2}, \quad \hat{\sigma}_s^2 = \frac{\mathrm{tr}(\tilde{\boldsymbol{C}}) - \mathrm{tr}(\boldsymbol{\Lambda})}{M - D}. \tag{17}$$

[2] This term is added to the covariance everywhere.

Algorithm 2 Fast matrix-vector multiplications

Require: Input vector $\boldsymbol{a}$ and $\boldsymbol{W}, \sigma_s^2, \boldsymbol{L}, \boldsymbol{S}$
Ensure: Fast MVMs $\boldsymbol{a} \leftarrow \tilde{\boldsymbol{C}}\boldsymbol{a}$
$\tilde{\boldsymbol{a}} \leftarrow \sigma_s^2\boldsymbol{a} + \boldsymbol{W}(\boldsymbol{W}^T\boldsymbol{a})$
$\boldsymbol{b} \leftarrow \boldsymbol{L}(\boldsymbol{S}(\boldsymbol{L}^T\tilde{\boldsymbol{a}}))$
$\tilde{\boldsymbol{b}} \leftarrow \sigma_s^2\boldsymbol{b} + \boldsymbol{W}(\boldsymbol{W}^T\boldsymbol{b})$
$\boldsymbol{a} \leftarrow \tilde{\boldsymbol{a}} + \tilde{\boldsymbol{b}}$

A naive computation of $\hat{\boldsymbol{W}}$, $\hat{\sigma}_s^2$ would require the explicit computation of $\boldsymbol{C}$, $\tilde{\boldsymbol{C}}$, and the eigendecomposition of the latter, which is $O(M^3)$ with a large constant. A better way is to use an iterative scheme to find the $D \ll M$ leading eigenvectors requiring a single matrix-vector multiplication(MVM) with $\tilde{\boldsymbol{C}}$ per round. Eq. (18-20) and Algorithm 2 show how a linear operator $\boldsymbol{a} \mapsto \tilde{\boldsymbol{C}}\boldsymbol{a}$ can be implemented given only $\boldsymbol{S}, \boldsymbol{L}$ and the old values of $\boldsymbol{W}$ and σ_s^2, thus avoiding $O(M^3)$ cost for forming $\tilde{\boldsymbol{C}}$ explicitly.

$$\tilde{\boldsymbol{C}}\boldsymbol{a} = \boldsymbol{L}^{-1}\boldsymbol{C}\boldsymbol{L}^{-T}\boldsymbol{a} = \boldsymbol{L}^{-1}(\boldsymbol{\Sigma} - \boldsymbol{\Sigma}\boldsymbol{S}\boldsymbol{\Sigma})\boldsymbol{L}^{-T}\boldsymbol{a} \tag{18}$$

$$= \tilde{\boldsymbol{\Sigma}}\boldsymbol{a} - \tilde{\boldsymbol{\Sigma}}\boldsymbol{L}\boldsymbol{S}\boldsymbol{L}^T\tilde{\boldsymbol{\Sigma}}\boldsymbol{a} \tag{19}$$

$$\tilde{\boldsymbol{\Sigma}}\boldsymbol{a} = \sigma_s^2\boldsymbol{a} + \boldsymbol{W}(\boldsymbol{W}^T\boldsymbol{a}) \tag{20}$$

In total, we require $O(D)$ MVMs giving us a cost of $O(M^2D)$. Computation of Eq. (17) using fast MVMs of Algorithm 2 is referred to as `fastEigs` in Algorithm 1. Similarly, $\mathrm{tr}(\tilde{\boldsymbol{C}})$ can be computed in $O(M^2D)$, as well.

Computational Complexity. To summarize, the E-step of our EM algorithm has running costs $O(\sum_n P_n^3)$, but is embarrassingly parallelizable with respect to users. Given the Cholesky factor $\boldsymbol{L}$, an M-step costs $O(M^2D)$. Overall, we require storing only two $M \times M$ matrices. The only $O(M^3)$ computation in our algorithm is the Cholesky decomposition of $\boldsymbol{K} + \varepsilon\boldsymbol{I}$, which has to be done only once up front. These figures compare favorably to the standard EM algorithm for matrix factorization, where an E-step costs $O(ND^3 + \sum_n D^2P_n)$ (which can be slightly cheaper), but each M-step requires $O(M^2D^2)$ memory as well as solving a dense system of size $(MD) \times (MD)$. Such computations are not tractable at large scales.

Nonparametric PCA. Finally, how does our method relate to the nonparametric PCA algorithm (NPCA) of [24]? First, they neither address models with non-Gaussian likelihoods nor with couplings $\boldsymbol{B}_n$, but are concerned with explicit ratings and Gaussian noise only. Their method can be lifted to our setting, which we did for experimental comparisons. Second, they do not make use of item features $\boldsymbol{x}_i$. Third, they cannot impose low rank constraints in the form of a $\boldsymbol{V}$ factor, with $D \ll M$. This point is worth elaborating on. NPCA corresponds to updating the full-rank

	Sushi		MovieLens		
	A	B	100K	1M	10M
#Items	10	100	1.5k	3.5k	9.5k
#Users	5k	5k	715	4.6k	45.6k
#Pairs	45	45	187k	3.5M	18.5M
#TestPairs	5k	5k	1.3k	9.9k	77k

Table 1: List of datasets. For MovieLens, we choose the items with ratings 5 and 1, and create all possible pairs. We remove the users and movies with no ratings, and keep 4 pairs per user for testing. The number obtained after this processing are shown in the Table.

covariance $\boldsymbol{\Sigma}$ directly. The M-step update is simply $\boldsymbol{\Sigma} \leftarrow \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\boldsymbol{S}\boldsymbol{\Sigma}$. It costs two $O(M^3)$ matrix multiplications, but avoids more complex algorithms like `eigs`. The E-step is identical to ours. We compare our algorithm to NPCA in Section 5, where we find that NPCA exhibits overfitting for smaller datasets, while requiring a very large number of iterations for convergence on the largest problems.

Speeding up E-Step Computations. E-step computations in our method scale as $O(P_n^3)$ per user n, which can be a bottleneck. In particular, a small number of users with many pairs can dominate the computational load even with parallel processing. One way to get around this issue is to employ sparse GP approximations [23]. Namely, with $W \ll M$ inducing variables, E-step computations can be done in $O(P_n(W+D)^2)$ if $P_n > W + D$. In our experiments in Section 5, we circumvent the problem by randomly subsampling the observations for users above a threshold. While this introduces stochasticity into the EM optimization, our results indicate that similar optima are reached for different runs and even different numbers of likelihoods sampled for each user. A more advanced idea would be to select likelihoods from a larger pool via greedy optimization of information-theoretic criteria [12].

5 Results

We present experimental results on real-world datasets listed in Table 1. Methods are evaluated on a test set of randomly drawn pairs. Similar to [14], our evaluation metric is the log-loss which is defined as $-\log_2 q(i \succ j|\mathcal{D})$, averaged over test pairs $(i \succ j)$. Here, $q(i \succ j|\mathcal{D})$ is the posterior predictive probability. See Appendix for details on an exact expression to compute the probability. This loss is close to 0 for correct predictions with high confidence, and 1 for random coin flips. It depends on predictive uncertainties, not just on predictive means. Code is available at `https://github.com/yjk21/sbcpl`.

Methods	Error	$\log\psi$, $\log\sigma_s^2$
GPfull-Laplace	0.897	(2 , -2, -2, 2)
GPitem-Laplace	0.910	(-2, 0)
GPitem-EP	0.875	(-2, 1)
GPVU-EP	**0.798**	(0, -0.96)
GPVU-Jaakkola	0.854	(0, -1.55)
VU-Jaakkola	0.856	

Table 2: Comparison of preference learning methods on Sushi-A dataset ($N = 200$ users, $M = 10$ items). GP denotes Gaussian process, VU matrix factorization, GPVU a combined model. Laplace, EP, Jaakkola denote different inference approximations (see text for details). The error is average test log-loss, the θ column provides hyperparameter values $(\log\psi, \log\sigma_s^2)$. For GPfull, these parameterize the item, then the user covariance function.

5.1 Comparison with Prior Work

We present a comparison of our method, combining matrix factorization with Gaussian processes, with methods that use one or the other. Since most prior work on GP preference learning is not scalable, we restrict ourselves to small datasets.

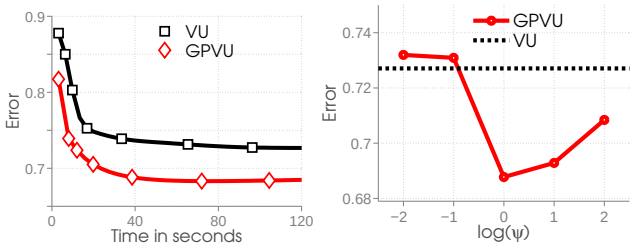
The Sushi datasets contain preferences of 5000 Japanese customers for different types of sushi [10] (sizes given in Table 1). Sushi-A has 10 sushi types while Sushi-B has 100. Each user was asked to rank 10 sushi each. For Sushi-A these were all the sushis, while for Sushi-B 10 items were randomly drawn from the complete set (different draw for each user). Ranking over 10 sushis gives us 45 preference statements per user. Moreover, each Sushi type comes with a 18-dimensional feature vector $\boldsymbol{x}_i$ (2 binary, 4 numerical and a 12-state categorical feature). User features are provided as well (9 dimensions).

We randomly drew training and test sets as follows. For Sushi-A, we subsampled $N = 200$ users, then extracted 3 training and 1 test pair per user. This very small dataset size allowed us to compare against methods which scale cubically in the total number of training pairs. For Sushi-B, we used all $N = 5000$ users, and extracted 10 training and 1 test pair per user.

Methods we compare differ in two aspects – the model and the inference method. We denote different methods as 'model'-'method'. The method can be EP, the local bound of [9] (Jaakkola), or the Laplace approximation [3]. Model can be either GP based, matrix factorization based or a combination of the two. We refer to these as GP, VU and GPVU, respectively. We now give a brief description of all these methods. The first method, which we refer to as GPfull-Laplace, is

Methods	D=10	D=20	D=30
VU-Jaakkola	0.730	0.715	0.720
GPVU-Jaakkola	**0.690**	0.700	0.710

Table 3: Comparison of preference learning methods on Sushi-B dataset ($N = 5000$ users, $M = 100$ items). VU is a matrix factorization model, GPVU adds a GP part dependent on item features.



(a) (b)

Figure 1: Further comparative results on Sushi-B. (a): GPVU-Jaakkola and VU-Jaakkola (with $D = 10$) converge in about the same number of iterations, GPVU outperforms VU. (b): Solid line shows test error for GPVU with different values of $\log \psi$ (kernel hyperparameter), dashed line shows test error for VU.

proposed in [3]. They use a joint Gaussian process over item and user features, $z_{in} = s(\boldsymbol{x}_i, \boldsymbol{y}_n)$, with a tensor product covariance matrix $\boldsymbol{K} \otimes \boldsymbol{K}_u$, where $\boldsymbol{K}$ is the item covariance, $\boldsymbol{K}_u$ the user covariance. We use `Matlab` code provided by the authors. GPitem-Laplace and GPitem-EP use a Gaussian process over items only, $z_{in} = s_n(\boldsymbol{x}_i)$, with a single shared covariance matrix $\boldsymbol{K}$. For all the GP based models, we use the squared-exponential covariance function $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/\psi)$, where $\psi > 0$ is a hyperparameter. The hyperparameters (ψ, σ_s^2) were selected on a grid to minimize test log-loss. Note that GPfull-Laplace has separate parameters for the item and user covariance function. For all the GP based methods, we add a jitter term with $\epsilon = 0.1$ to make sure that the Kernel matrix is well-conditioned.

VU-Jaakkola employs a matrix factorization model only, $z_{in} = \boldsymbol{v}_i^T \boldsymbol{u}_n$. Note that there are no item biases μ_i. This model is fitted by the standard EM algorithm described in the Appendix. Finally, GPVU is our proposed approach, which we ran with EP and Jaakkola approximate inference, but without item biases μ_i. Note that in our method, σ_s^2 is learned as part of the algorithm, and only ψ has to be selected. In terms of scalability, GPfull-Laplace is the most expensive method by far, followed by VU-Jaakkola. The GPitem-Laplace can be implemented to run very efficiently, but we simply used the code provided by [3].

The results of our comparison on Sushi-A are shown in Table 2. Comparing GPfull-Laplace to GPitem-Laplace, we see that the inclusion of user features provide little improvements of about 1% only, in line with findings reported in [8]. EP does significantly better than the Laplace approximation: 4% improvement for GPitem. The posterior with threshold likelihoods is highly skewed, and the Laplace approximation can perform poorly. However, the most significant improvements are obtained by adding a matrix factorization term (7% from GPitem-EP to GPVU-EP). Our proposed approach GPVU-EP improves on all others significantly. Finally, VU-Jaakkola gives almost the same performance as GPVU-Jaakkola. Due to the small number of items, a purely collaborative approach cannot be improved much by adding a GP, but this does help for cold-start.

We also compared VU-Jaakkola (matrix factorization only) with GPVU-Jaakkola (our combined model) on the larger Sushi-B dataset with $M = 100$ items, results are shown in Table 3. Here, adding the GP part leads to modest improvements. We also see that prediction errors increase for larger latent dimension D, indicating the onset of overfitting. A sample run for $D = 10$ is shown in Figure 1(a). For this small dataset, there is no big difference in convergence time, but GPVU attains better performance. Figure 1(b) demonstrates the sensitivity of GPVU results w.r.t. the kernel hyperparameter $\log \psi$.

Overall, there is a clear advantage of our GPVU method over GP-only methods. Our EP inference outperforms other options (Laplace, Jaakkola). Our edge over VU (matrix factorization only) is less pronounced for these small datasets with few items, but becomes significant on large and sparse dataset.

5.2 Results on MovieLens Dataset

We now address larger problems, based on the three MovieLens[3] datasets, consisting of 100K, 1M and 10M explicit ratings (integers 1–5) for movies, along with a range of user and item features (we make use of the latter only). We create three preference learning problems, similar to what is done in [14]. We only use ratings 5 and 1, removing users with no such ratings. We create a pool of training pairs $i \succ j$ for every i, j s.t. i is rated 5, j is rated 1. The number of users, movies and training pairs are shown in Table 1. We sample four pairs per user for testing, removing them from the training pool. Just as above, we use average test log-loss as performance metric. For GP, we add a jitter term of $\epsilon = 1$ to make the Kernel matrix well-conditioned.

[3] `www.grouplens.org/datasets/movielens/`

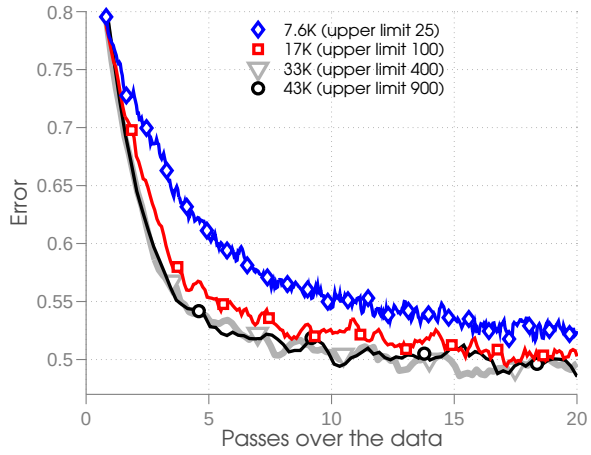


Figure 2: Experiment running GPVU-EP on MovieLens-100K, with $D = 30$ latent dimensions and $\psi = 1$. Legend shows different upper limits on P_n, along with the sum of subsampled pairs over the user. The horizontal axis unit is the number of pairs updated on, in terms of passes over the total training pool. Markers are set after every 20 EM iterations. A method with fewer training pairs (e.g., 7.6k) runs more EM iterations in a given time, but converges slower than a method with many training pairs (e.g., 43k). Importantly, all methods eventually reach the same test error plateau.

We compare our method GPVU-EP with NPCA-EP, where nonparametric PCA [24] is discussed at the end of Section 4. NPCA does not use item features. It differs from VU above (which cannot be run at this scale) in that the covariance $\mathbf{\Sigma}$ is not low-rank constrained. In both cases, we speed up the E-step by subsampling pairs from the pool for each user, a strategy which is similar to Bayesian personalized ranking (BPR) [17]. We do this by setting an upper limit on P_n (number of pairs per user), then subsampling for users whose pool size is larger. Sampling is repeated independently in each iteration.

In our first experiment, we run GPVU-EP with different upper limits on P_n, giving rise to different number of training pairs seen by each EM iteration. Results on MovieLens-100K are shown in Figure 2. While the runs converge at different speeds, they all eventually reach the same test set performance, which demonstrates that our subsampling strategy does not bias final results on this dataset. In the context of these experiments, the upper limit on P_n can therefore be treated as a tuning parameter which determines the speed of convergence. In all further experiments, we fix the cap on P_n to 400 pairs, which corresponds to a total of 33K pairs in Figure 2.

In Table 4, we show the effect of latent dimensionality D and hyperparameter ψ for GPVU-EP run on MovieLens-100K. We observe overfitting for D larger than 30. For the larger MovieLens datasets, overfitting does not occur, but the test error does not improve significantly for $D > 30$. For the kernel hyperparameter, a value of $\log \psi = 1$ works best. In all further experiments, we fix $D = 30$ and $\log \psi = 1$. We use EP as our inference algorithm since it gives better results than Jaakkola (see Figure in the Appendix).

D	Error
5	0.589
10	0.526
20	0.504
30	**0.485**
50	0.490

$\log(\psi)$	Error	$\log(\sigma_s^2)$
-2	0.533	-0.17
-1	0.532	-0.17
0	0.493	-0.13
1	**0.483**	-0.07
2	0.495	-0.05

Table 4: Experiments running GPVU-EP on MovieLens-100K. Left: Different latent dimensionalities D for $\psi = 1$. Right: Different values for $\log \psi$ for $D = 30$. We also show $\log \sigma_s^2$, as learned by the EM algorithm.

Method	100k	1M	10M
NPCA-EP	0.525	0.375	0.576
GPVU-EP	**0.483**	**0.368**	**0.390**

Table 5: Average test log-loss for NPCA-EP and GPVU-EP on MovieLens (100K, 1M, 10M). Scores are averaged over 5 repetitions each (different seeds), as well as over the last few EM iterations of each run. GPVU outperforms NPCA in all cases, the differences being statistically significant for 100K and 10M.

Table 5 provides results for NPCA-EP vs. our GPVU-EP across the 3 MovieLens datasets (also see Figure in the Appendix). NPCA exhibits overfitting on the smaller 100K dataset, while the low rank assumption ($D = 30$) leads to valid generalization for GPVU. MovieLens-1M is a fairly densely observed dataset (more so than 10M, see Table 1): NPCA and GPVU show similar performance. Our method exhibits much better convergence on the largest dataset MovieLens-10M, while NPCA does not properly converge after a reasonable runtime. GPVU significantly outperforms NPCA for 100K and 10M. For the largest 10M dataset, both methods were stopped after 100 passes over the training pool. NPCA did not seem to have converged even after this large number of iterations.

We also implemented a map-reduce version of our code that runs E-step in parallel, using multi-threading in C++ with Boost. In our implementation, each thread manages a local copy of the sufficient statistic matrix which are then added in the end in parallel, requiring only logarithmic number of operations in the number

Pairs	100K	Pairs	1M	Pairs	10M
7.6k	.7s, 1s	56k	7s, 3s	-	-
17k	1s, 1s	147k	9s, 3s	1M	1m, 20s
33k	7s, 1s	349k	1m, 3s	2M	7m, 20s
43k	27s, 1s	534k	15m, 3s	3M	58m, 20s

Table 6: Running times of GPVU-EP on MovieLens datasets (100K, 1M, 10M), for different total numbers of training pairs. For each dataset, the first column shows the total number of training pairs obtained after subsampling, the second quotes average time for an E-step and M-step (in s(econds) or m(inutes)). M-step times are independent of the number of training pairs.

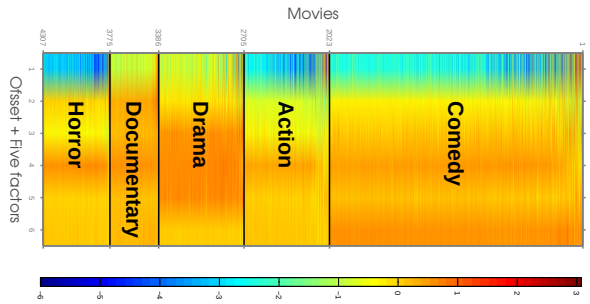


Figure 3: Visualization of latent factors for Movielens-10M. First rows shows $\boldsymbol{\mu}$. Rest 5 show top 5 columns of $\boldsymbol{V}$. Movies are grouped under 5 genres, clearly showing the expected clustering of factors.

of threads. The parallel reduction was done using Intel's Threading Building Blocks. We opted for a shared memory environment for ease of implementation, although a distributed implementation might be more appropriate. In Table 6, we provide running times of our GPVU-EP on MovieLens datasets different sizes and number of subsampled training pairs, as obtained on a workstation with Intel 2.8GHz i7-2600S processor and 4 cores. Our method scale well to large datasets. We see that on 4 cores, we can run up to 3M pairs within an hour. On 32 cores, we get further reduction enabling us to process 3M pairs within 10 minutes.

Figure 3 shows a visualization of learned parameters for Movielens-10M. An advantage of the Bayesian approach is the quantification of uncertainty. To demonstrate this, we picked all comedy, action, and drama movies from Movielens-10M, and simulated users with specific movie-tastes by generating training-pairs in a controlled way. For example, a comedy-preferring user is represented using observation-pairs where comedy movies are preferred over other genres. We held out 100 test-pairs for each kind of movie, and computed predictions using $\boldsymbol{\theta}$ previously learned on Movielens-10M. Figure 4 shows these predictions as a function of number of training examples for a user who prefers comedy and another user who prefers action. We see,

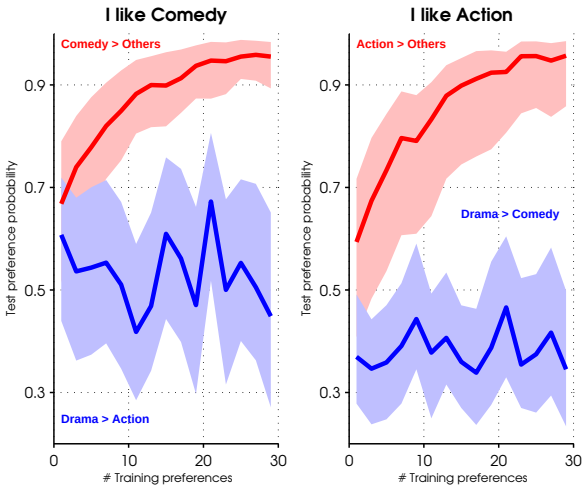


Figure 4: Demonstration of uncertainty-quantification with our Bayesian approach. See text for details.

e.g. for comedy-preferring user, that prediction probabilities for test-pairs where comedy is preferred, approach 1 as the number of training pairs is increased. Interestingly, the predictions for pairs where drama is preferred over action stays neutral (around 0.5) throughout. The same is true for an action-preferring user. Our Bayesian framework quantifies uncertainties reliably, making many confident predictions close to 1 while also keeping uncertain predictions close to 0.5.

6 Conclusions

We presented a novel approach for Bayesian multi-user preference learning, modelling a user's utility by the sum of a latent bilinear collaborate filtering term and a Gaussian process indexed by item features. This combination yields substantially improved results over using one or the other on its own. The bilinear part is particularly effective for users with much data, while the GP part is valuable in cold start situations. We have clarified why learning from pair preferences is computationally more difficult than from explicit ratings. By modifying the standard EM approach to this setting, we obtain a new algorithm with simple closed-form updates, which scales to very large datasets without imposing factorization assumptions or restricting the way in which likelihoods are approximated.

In future work, we will explore ways to speed up the E-step, as noted at the end of Section 4. We will extend our method to discrete choice (or partial ranking) data by way of the multivariate probit likelihood. Finally, we will explore stochastic versions of our algorithm [7].

Acknowledgements: This work was funded by an ERC starting grant (277815-SCALABIM).

References

[1] D. Agarwal and B. Chen. Regression-based latent factor models. In *ACM SIGKDD 15th Int. Conf. Knowledge Discovery and Data Mining*, pages 19–28, 2009.

[2] A. Birlutiu, P. Groot, and T. Heskes. Multi-task preference learning with an application to hearing aid personalization. *Neurocomputing*, 73(79):1177–1185, 2010.

[3] E. Bonilla, S. Guo, and S. Sanner. Gaussian process preference elicitation. In *NIPS 23*, pages 262–270, 2010.

[4] E. Brochu, N. De Freitas, and A. Ghosh. Active preference learning with discrete choice data. In *NIPS 20*, pages 409–416. 2008.

[5] W. Chu and Z. Ghahramani. Preference learning with Gaussian processes. In *ICML 22*, pages 137–144, 2005.

[6] M. van Gerven, B. Cseke, F. de Lange, and T. Heskes. Efficient Bayesian multivariate fMRI analysis using a sparsifying spatio-temporal prior. *Neuroimage*, 50:150–161, 2010.

[7] M. Hoffman, D. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *JMLR*, 14:1303–1347, 2013.

[8] N. Houlsby, J. Hernandez-Lobato, F. Huszar, and Z. Ghahramani. Collaborative Gaussian processes for preference learning. In *NIPS 25*, pages 2105–2113, 2012.

[9] T. Jaakkola. *Variational Methods for Inference and Estimation in Graphical Models*. PhD thesis, MIT, 1997.

[10] T. Kamishima. Nantonac collaborative filtering: Recommendation based on order responses. In *ACM SIGKDD 9th Int. Conf. Knowledge Discovery and Data Mining*, 2003.

[11] N. Koenigstein and U. Paquet. Xbox movies recommendations: Variational Bayes matrix factorization with embedded feature selection. In *Proceedings of the 7th ACM Conference on Recommender Systems*, 2013.

[12] N. D. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In *NIPS 15*, pages 609–616, 2003.

[13] T. Minka. Expectation propagation for approximate Bayesian inference. In *Uncertainty in AI 17*, 2001.

[14] A. Mnih and Y. W. Teh. Learning label trees for probabilistic modelling of implicit feedback. In *NIPS 25*, pages 2825–2833, 2012.

[15] Radford M. Neal. Monte Carlo implementation of Gaussian process models for Bayesian classification and regression. Technical Report 9702, Department of Statistics, University of Toronto, January 1997.

[16] H. Nickisch and C. Rasmussen. Approximations for binary Gaussian process classification. *JMLR*, 9:2035–2078, 2008.

[17] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Uncertainty in AI 25*, pages 452–461, 2009.

[18] T. Salimans, U. Paquet, and T. Graepel. Collaborative learning of preference relations. In *Proceedings of the 6th ACM Conference on Recommender Systems*, 2012.

[19] M. Seeger. Bayesian inference and optimal design for the sparse linear model. *JMLR*, 9:759–813, 2008.

[20] D. Stern, R. Herbrich, and T. Graepel. Matchbox: Large scale online Bayesian recommendations. In *Proceedings of the 18th International Conference on World Wide Web*, pages 111–120, 2009.

[21] G. Takács and D. Tikk. Alternating least squares for personalized ranking. In *Proceedings of the 6th ACM Conference on Recommender Systems*, pages 83–90, 2012.

[22] M. Tipping and C. Bishop. Probabilistic principal component analysis. *J. Roy. Stat. Soc. B*, 61(3):611–622, 1999.

[23] M. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *AISTATS 12*, pages 567–574, 2009.

[24] K. Yu, S. Zhu, J. Lafferty, and Y. Gong. Fast nonparametric matrix factorization for large-scale collaborative filtering. In *Proceedings of ACM SIGIR*, pages 211–218, 2009.