Matchbox: Large Scale Online Bayesian Recommendations

David Stern
Microsoft Research Ltd.
Cambridge, UK
dstern@microsoft.com

Ralf Herbrich
Microsoft Research Ltd.
Cambridge, UK
rherb@microsoft.com

Thore Graepel
Microsoft Research Ltd.
Cambridge, UK
thoreg@microsoft.com

ABSTRACT

We present a probabilistic model for generating personalised recommendations of items to users of a web service. The Matchbox system makes use of content information in the form of user and item meta data in combination with collaborative filtering information from previous user behavior in order to predict the value of an item for a user. Users and items are represented by feature vectors which are mapped into a low-dimensional 'trait space' in which similarity is measured in terms of inner products. The model can be trained from different types of feedback in order to learn user-item preferences. Here we present three alternatives: direct observation of an absolute rating each user gives to some items, observation of a binary preference (like/ don't like) and observation of a set of ordinal ratings on a user-specific scale. Efficient inference is achieved by approximate message passing involving a combination of Expectation Propagation (EP) and Variational Message Passing. We also include a dynamics model which allows an item's popularity, a user's taste or a user's personal rating scale to drift over time. By using Assumed-Density Filtering (ADF) for training, the model requires only a single pass through the training data. This is an on-line learning algorithm capable of incrementally taking account of new data so the system can immediately reflect the latest user preferences. We evaluate the performance of the algorithm on the MovieLens and Netflix data sets consisting of approximately 1,000,000 and 100,000,000 ratings respectively. This demonstrates that training the model using the on-line ADF approach yields state-of-the-art performance with the option of improving performance further if computational resources are available by performing multiple EP passes over the training data.

Categories and Subject Descriptors

G.3 [**Mathematics of Computing**]: Probability and Statistics; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information Filtering*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services

General Terms

Algorithms, Experimentation, Performance

1. INTRODUCTION

One of the great promises of the Web is to connect all the participating people, products, companies and institutions. For these connections to be useful it is necessary to select the few fruitful connections and disregard the many worthless connections. Automatic recommender systems can aid users in this selection [7, 19], some of the best known being found at Amazon, [18], Netflix [1], and Yahoo!. The task at hand is to predict, for a particular user, which items they will be interested in. There are two sources of information which are typically used to achieve this.

- **Content (Meta-Data)** The content-based approach makes use of descriptions (feature vectors) of both users and items. Users may be described by properties such as age and gender and items may be described by properties such as author and manufacturer. Typical examples of content based recommendation systems include web search engines and social matchmaking sites.

- **Collaborative Filtering** The collaborative filtering approach uses only the identities (IDs) of users and items. Implicit descriptions of the user and items are obtained from a (sparse) matrix of ratings of items by users [3]. We can learn about a user by the items they have previously rated and the users who have rated items in common with them.

Typically both of these types of information are available and ideally we would like to use both of them to make predictions. When a user is relatively new to the system, predictions should be improved by making use of extra information about the user, thus addressing the well known 'cold-start' problem [11]. However, once we have observed sufficient user ratings we want to be able to make fully personalized predictions for the user based on their specific ratings rather than their features.

In addition it is desirable that a recommender system can be applied flexibly in a wide variety of scenarios. Frequently, data on user preferences is in the form of ratings on an ordinal scale (for example Netflix [1]) where each user's interpretation of this scale may be different. In other cases we may only have data showing which items were clicked on by a user, where we assume that a user clicking on an item provides implicit evidence that it is interesting to them (for example the Google news recommender system [6]).

In order to provide up-to-date recommendations and react rapidly to a user's evolving tastes we need the system to be able to be trained rapidly, ideally incrementally, as new data arrives. It is crucial that the system is scalable, with memory and processor requirements increasing in proportion to the number of users and items and not in proportion to the total number of previous ratings.

In this work we present a model based on a compressed representation of the matrix of the values of items for users. In the literature the prototypical method of this kind is based on a singular value decomposition (see, for example [12]). We represent each user and item by a vector of features. In order to take account of user-specific and item-specific tastes we can include a binary feature for every user ID and item ID in the set of features (only one of which is active for each user and item). Each feature is associated with a latent 'trait' vector and the linear combination of the trait vectors for a particular user or item, weighted by the feature values for that user or item, provides a total trait vector for the user or item. We model the value that an item has for a user as the inner product between the trait vector for the item and the trait vector for the user. Key contributions of this work are:

1. User and item metadata are integrated to produce good recommendations in a cold-start situation which will automatically become more personalised for longer term users (Section 2.1).

2. The model for user feedback is flexible. We present three alternatives: direct observation of an absolute rating each user gives to some items, observation of a binary preference (like/ don't like) and observation of a set of ordinal ratings on a user-specific scale (Section 2.3).

3. A dynamics model which allows an item's popularity, a user's taste or a user's personal rating scale to drift over time (Section 2.4).

4. Efficient inference is performed with a novel combination of Variational Message Passing (VMP) and Expectation Propagation (EP) (Section 3.1).

5. Assumed-Density Filtering (ADF) can be used to give an on-line training method that can incrementally take account of new data so the system can immediately reflect the latest user preferences (Section 3.3).

In Sections 4.1 and 4.2 we present experimental results for the Netflix and MovieLens data sets.

2. A PROBABILISTIC RATING MODEL

2.1 The Bi-linear Rating Model

Initially, let us assume that the recommender system receives tuples $(\mathbf{x}, \mathbf{y}, \Phi, r)$ of user descriptions $\mathbf{x} \in \mathbb{R}^n$, item descriptions $\mathbf{y} \in \mathbb{R}^m$, other features describing the context of this rating $\mathbf{\Phi} \in \mathbb{R}^f$ and ratings $r \in \mathbb{R}$. We define the K dimensional user *trait vector* as $\mathbf{s} = \mathbf{U}\mathbf{x}$ where $\mathbf{U}$ is a $K \times n$ matrix of latent user traits where each element u_{ki} is the contribution of feature i to user trait dimension k. Similarly we define the K dimensional item trait vector as $\mathbf{t} = \mathbf{V}\mathbf{y}$ for an $K \times m$ item trait matrix, $\mathbf{V}$. In general we also model a bias, $b = \Phi^\top \mathbf{w}$ where $\mathbf{w}$ is a latent set of weights. Frequently the features Φ will be a combination of the user description features $\mathbf{x}$ and the item description features $\mathbf{y}$ only so in these cases we may write $b = \mathbf{x}^\top \mathbf{u} + \mathbf{y}^\top \mathbf{v}$ where $\mathbf{u}$ and $\mathbf{v}$ are the latent user and item biases.

Now, the rating, r, is modeled as

$$p(r|\mathbf{s}, \mathbf{t}, b) = \mathcal{N}(r|\mathbf{s}^\top \mathbf{t} + b, \beta^2)$$

where β is the standard deviation of the observation noise. Thus we adopt a bi-linear form in which the similarity between a user and an item is given by the inner product of a vector of user traits and a vector of item traits. The expected rating is proportional to the lengths $||\mathbf{s}||$ and $||\mathbf{t}||$ of the trait vectors as well as the cosine of the angle between them. The parameter K is chosen such that $K \ll n$ and $K \ll m$ so computational resource requirements (in time and memory) should scale as $m + n$ rather than mn.

An important special case of this model is the case when only the identity of each user and item is known and no additional properties. In this case, the model reduces to a collaborative filtering approach if we represent user i by the feature vector $\mathbf{x} := \mathbf{e}_i$ and item j by $\mathbf{y} := \mathbf{e}_j$, where $\mathbf{e}_i$ denotes the ith unit vector, i.e., the vector with 1 in position i and 0 everywhere else. If the bias is zero then the expression for the expected rating simplifies to $E[r] = \mathbf{u}_i^\top \mathbf{v}_j$. Here $\mathbf{u}_i$ and $\mathbf{v}_j$ are the ith and jth column of $\mathbf{U}$ and $\mathbf{V}$ respectively. In other words, each user and each item is represented by a K-dimensional vector in trait space directly. To make the connection to standard methods, suppose we have a complete matrix $\mathbf{R}$ of ratings for n users and m items. Then the singular-value decomposition (SVD) yields the optimal solution for the matrices $\mathbf{U}$ and $\mathbf{V}$ in the least-squares sense, that is

$$(\mathbf{U}_{\mathrm{SVD}}, \mathbf{V}_{\mathrm{SVD}}) := \underset{(\mathbf{U},\mathbf{V})}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{m} (\mathbf{u}_i^\top \mathbf{v}_j - r_{ij})^2.$$

This function can be directly minimized but due to the large sparsity of the matrix $\mathbf{R}$ in real-world tasks (for example, 1.2% for the Netflix data set [1]) regularization needs to be employed [17, 12, 2].

2.2 Factorizing Prior

The model parameters to be learned are the variables $\mathbf{U}$ and $\mathbf{V}$ which determine how users and items are mapped to the K dimensional trait space and $\mathbf{w}$, the value of bias features for the rating. We represent our prior beliefs about the values of these parameters by independent Gaussian distributions on each of the components of the matrices $\mathbf{U}$ and $\mathbf{V}$ and the vector $\mathbf{w}$. For example we have

$$p(\mathbf{U}) = \prod_{k=1}^{K} \prod_{i=1}^{n} \mathcal{N}(u_{ki}; \mu_{ki}, \sigma_{ki}^2).$$

We choose this factorizing prior because it reduces memory requirements to two parameters (a mean and standard deviation) for each component and it allows us to perform efficient inference (see Section 3.1).

2.3 Feedback Models

One advantage of the approach described in this paper is that it allows us to model different types of user-item rating data in a flexible way. Up to this point we have assumed that the rating, r, is observed directly, however this need not be the case.

2.3.1 *Ordinal Regression*

A common scenario is that users provide feedback about which items they like or dislike via an ordinal scale. For example, on Netflix, users are asked to rate movies from one to five stars. These ranks can only be compared, but not subtracted from one another. In addition, each user's interpretation of the scale may be different and the mapping from rank to latent rating may not be linear. We assume that for each user-item pair for which data is available we observe a rank $l \in 1, \cdots, L$. We relate the latent rating r to ranks l via a cumulative threshold model [4]. For each user, u, we maintain user-specific thresholds $\mathbf{b}_u \in \mathbb{R}^{L-1}$ which divide the latent rating axis into L consecutive intervals $(b_{u(i-1)}, b_{u(i)})$ of varying length each of which representing the region in which this user gives the same rank to an item. Formally, we define a generative model of a ranking as $p(l = a|\mathbf{b}_u, r) =$

$$\begin{cases} \prod_{i=1}^{a-1} \mathbb{I}(r > \tilde{b}_{u(i-1)}) \prod_{i=a}^{L-1} \mathbb{I}(r < \tilde{b}_{u(i-1)}) & \text{if } 1 < a < L \\ \prod_{i=1}^{L-1} \mathbb{I}(r < \tilde{b}_{u(i-1)}) & \text{if } a = 1 \\ \prod_{i=1}^{L-1} \mathbb{I}(r > \tilde{b}_{u(i-1)}) & \text{if } a = L \end{cases} \tag{1}$$

where

$$p(\tilde{b}_{ui}|b_{ui}, \tau) = \mathcal{N}(\tilde{b}_{ui}; b_{ui}, \tau^2)$$

and we place an independent Gaussian prior on the thresholds so $p(b_{ui}) = \mathcal{N}(b_{ui}; \mu_i, \sigma_i^2)$. The indicator function $\mathbb{I}(\cdot)$ is equal to 1 if the proposition in the argument is true and 0 if it is false.

2.3.2 *Binary ('click') Probit Model*

A special case of the ordinal regression feedback model is for $L = 1$, $\tau = 0$, $\sigma_0^2 = 0$ and $\mu_0 = 0$. This corresponds to observing a binary variable $c \in \{\text{TRUE}, \text{FALSE}\}$ which is related to the latent rating by,

$$p(c = \text{TRUE}) = p(r > 0).$$

This is useful in the situation where we are only provided with binary information about whether or not a user likes a particular item. For example, a user may provide feedback to a news recommender service by which stories they click on [6].

2.4 Dynamics

Ideally a recommender system needs to be able to track non-stationary data. A user's tastes will drift with time and an item's popularity may rise and fall with changing trends. In addition, if we use the user-specific threshold model discussed in Section 2.3.1 a user's individual rating scale may also drift with time. One well known phenomenon where this may occur is 'anchoring' where a user tends to be more likely to give an item a high rating if they have recently given other items a high rating.

We model these dynamics by assuming that the latent variables $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{w}$ (and $\mathbf{b}$ if we use the ordinal regression feedback model) drift with time by the addition of Gaussian noise each time step. For the example of the threshold model we have $p(b_l^{(t+1)}|b_l^{(t)}) = \mathcal{N}(b_l^{(t+1)}; b_l^{(t)}, \gamma^2)$ where t is an index over time steps. At time t_0 we use the prior $p(b_i^{(0)}) = \mathcal{N}(b_i; \mu_i, \sigma_i^2)$. Analogous models may be used for each of the other latent variables.

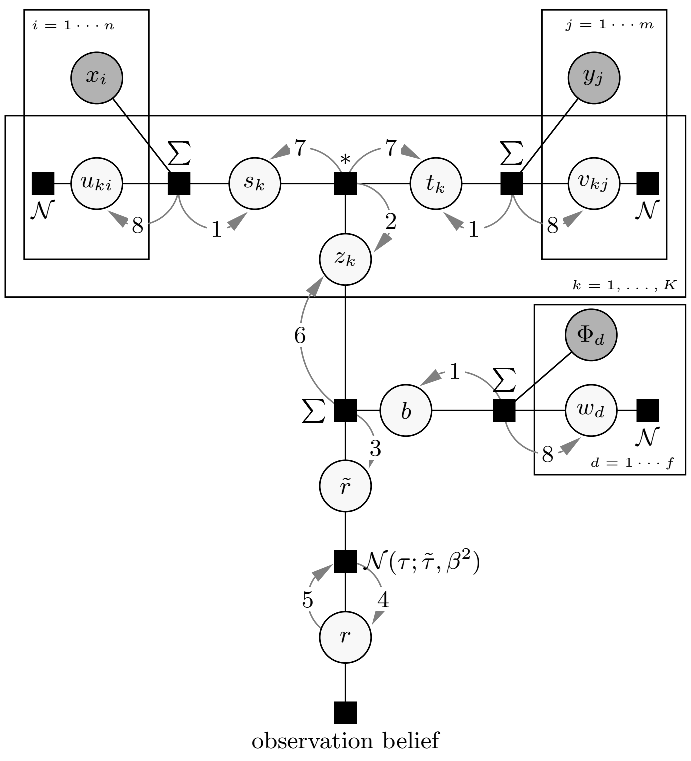


Figure 1: Factor Graph for Bi-linear Rating Model. The large rectangles or *plates* indicate parts of the graph which are repeated with repetition indexed by the variable in the corner of the plate. The factors labeled Σ are sum factors of the form $\mathbb{I}[z = x + y]$. The numbered arrows correspond to messages. Messages (1) are the messages from the sum factors mapping from the user/ item/ bias descriptions to the latent trait space, $m_{\Sigma \to s_k}(s_k)$, $m_{\Sigma \to t_k}(t_k)$, and $m_{\Sigma \to b}(b)$. Message (2) is the message $m_{* \to z_k}(z_k)$, from the product factor. Message (3) is $m_{\Sigma \to \tilde{r}}(\tilde{r})$, the message from the total sum factor to the latent rating $\tilde{r}$. Message (4) is the message from the Gaussian noise factor $\mathcal{N}(\tau; \tilde{r}, \beta^2)$ to the noisy latent rating r. The rest of the messages are the reverse messages back to the prior.

3. INFERENCE

Given a stream of rating tuples $(\mathbf{x}, \mathbf{y}, \mathbf{\Phi}, r)$ we train the model in order to learn posterior distributions over the values of the parameters $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{w}$. This can be accomplished efficiently by message passing. The algorithm sketched below involves computing a sequence of messages. We used the Infer.net library to perform these computations [15].

3.1 Message Passing for the Core Model

The model described in Section 2.1 can be further factorized by introducing some intermediate latent variables z_k to represent the result of each component, $s_k t_k$ of the inner product. That is, $p(z_k|s_k, t_k) = \mathbb{I}(z_k = s_k t_k)$. Now the latent rating (before adding noise) is given by $p(\tilde{r}|\mathbf{z}, b) = \mathbb{I}(\tilde{r} = \sum_k z_k + b)$. From Section 2.1 we can see that $p(s_k|\mathbf{U}, \mathbf{x}) = \mathbb{I}(s_k = \sum_i u_{ki} x_i)$ and $p(t_k|\mathbf{V}, \mathbf{y}) = \mathbb{I}(t_k = \sum_j v_{kj} y_j)$ and

$p(b|\mathbf{w}, \Phi) = \mathbb{I}(b = \sum_i \Phi_i w_i)$.

Therefore the joint distribution of all the variables factorizes as $p(\mathbf{s}, \mathbf{t}, \mathbf{U}, \mathbf{V}, \mathbf{w}, \mathbf{z}, \tilde{r}, r|\mathbf{x}, \mathbf{y}, \mathbf{\Phi}) =$

$$p(r|\tilde{r})p(\tilde{r}|\mathbf{z}, b)p(b|\mathbf{w}, \Phi)p(\mathbf{U})p(\mathbf{V})p(\mathbf{w}) \cdot \prod_{k=1}^{K} p(z_k|s_k, t_k)p(s_k|\mathbf{U}, \mathbf{x})p(t_k|\mathbf{V}, \mathbf{y}).$$

The posterior distribution over the $\mathbf{U}$ $\mathbf{V}$ and $\mathbf{w}$ variables given an observed rating tuple, $(\mathbf{x}, \mathbf{y}, \Phi, r)$, is given by summing out the latent variables: $p(\mathbf{U}, \mathbf{V}, \mathbf{w}|r, \mathbf{x}, \mathbf{y}, \Phi) \propto$

$$\int_{\mathbf{s}}\int_{\mathbf{t}}\int_{\mathbf{z}}\int_{\tilde{r}} p(\mathbf{s}, \mathbf{t}, \mathbf{U}, \mathbf{V}, \mathbf{w}, \mathbf{z}, \tilde{r}, r|\mathbf{x}, \mathbf{y}, \mathbf{f}) d\mathbf{s} d\mathbf{t} d\mathbf{z} d\tilde{r}. \quad (2)$$

The factor graph for this model is shown in Figure 1. A factor graph is a bipartite graph with (square) factor nodes corresponding to factors in a function and (circular) variable nodes representing variables in the function. The edges of the graph reveal the dependencies of factors on variables [10].

Message passing is used to compute the marginal of a joint distribution by assuming a full factorization of the joint distribution and in this way approximating each factor as follows (for an example distribution $p(\mathbf{v})$ of a set of variables $\mathbf{v}$):

$$p(\mathbf{v}) \propto \prod_f f(\mathbf{v}) \approx \prod_{\hat{f}} \underbrace{\prod_{i \in V(\hat{f})} m_{\hat{f} \to i}(v_i)}_{\hat{f}(\mathbf{v})} = \prod_i \underbrace{\prod_{\hat{f} \in \hat{F}(i)} m_{\hat{f} \to i}(v_i)}_{\hat{p}(v_i)},$$

where $V(\hat{f})$ is the set of all variable indices involved in the approximate factor $\hat{f}$ and $\hat{F}(i)$ is the set of all approximate factors in which variable v_i is involved. Message passing is about best approximating each factor f by a factor $\hat{f}$. This optimization is achieved by minimizing an α-divergence (a generalization of the Kullback-Leibler divergence) between $\hat{p}(\mathbf{v}) \cdot f(\mathbf{v})/\hat{f}(v)$ (see [14] for details). We approximate all factors by Gaussian densities so all messages have the functional form of a Gaussian density.

For all of the factors in the model (Figure 1) apart from the product factor ($\mathbb{I}(z_k = s_k \cdot t_k)$, labeled * in the diagram) we choose to minimise the Kullback-Leibler (KL) divergence $\mathrm{KL}(f||\hat{f})$ between the true and approximate factors (alpha-divergence with $\alpha \to 0$). This corresponds to the Sum-Product algorithm (for exact messages) and Expectation Propagation (for approximate messages) [10, 16]. For the exact factors we compute factor to variable messages according to the general update equation for a message from a factor f to a variable v:

$$m_{f \to v}(v) = \int f(\mathbf{v}) \prod_{v_j \in V(f) \setminus v} m_{v_j \to f(\mathbf{v})}(v_j) d\mathbf{v} \quad (3)$$

(which follows directly from the distributive law of sums and products). The messages are denoted on the factor graph by arrows and the numbers indicate the order in which we compute them. Since all messages are Gaussian these computations are exact for all factors in the model apart from the product factor.

3.1.1 Pre-Processing

Firstly we pre-compute the sums for the linear Gaussian input models. For example for the user model we have (from equation 3), $m_{\Sigma \to s_k}(s_k) =$

$$\int \mathbb{I}(s_k = \sum_i u_{ki} x_i) \prod_{i=1}^{n} \mathcal{N}(u_{ki}; \mu_{ki}, \sigma_{ki}^2) dU \quad (4)$$

$$= \mathcal{N}\left(s_k; \sum_i \mu_{ki} x_i, \sum_i \sigma_{ki}^2 x_i\right). \quad (5)$$

This is also computed for the item model and the bias model and are labeled '1' in Figure 1.

3.1.2 Iterative Message Passing Schedule

Next we compute the message from the product factor (labeled '2' in Figure 1). This factor presents us with a difficulty because if we were to use the EP approximation here the variance of the message would always grow. To see this, note that given $m_{z_k \to *}(z_k) = \mathcal{N}(z_k; \mu_k, \sigma_k^2)$ the most likely value for s_k is $s_k = \mu_k / t_k$. This gives rise to a true bi-modal hyperbolic message density on both s_k and t_k (one mode for $t_k > 0$ and one mode for $t_k < 0$). Since the Kullback–Leibler distance attempts to capture the support of the distribution, the variance of the approximation messages $m_{* \to s_k}$ and $m_{* \to t_k}$ will grow to cover both modes of the distribution.

Covering both modes is undesirable, we just need to make sure that the model chooses a single mode locally to break the symmetry. Therefore, for this factor we choose to minimize the KL divergence with the arguments swapped, $\mathrm{KL}(\hat{f}||f)$, which is equivalent to a local variational approximation [20, 14] ($\alpha \to 0$). The variational update equations for the messages to and from the product factor $f(s, t, z) = \mathbb{I}(z = s \cdot t)$

$$m_{* \to z}(z) = \mathcal{N}\left(z; \langle s \rangle \langle t \rangle, \langle s^2 \rangle \langle t^2 \rangle - \langle s \rangle^2 \langle t \rangle^2\right),$$

$$m_{* \to s}(s) = \mathcal{N}\left(s; \frac{\langle m_{z \to *} \rangle \langle t \rangle}{\langle t^2 \rangle}, \frac{\langle m_{z \to *}^2 \rangle - \langle m_{z \to *} \rangle^2}{\langle t^2 \rangle}\right) (6)$$

corresponding to messages (2) and (7) in Figure 1 respectively. Here, $\langle t \rangle$ denotes the mean of the approximate (Gaussian) marginal distribution $p(t)$ and $\langle t^2 \rangle$ denotes the non-centered second moment of the marginal $p(t)$ (the message for $m_{* \to t}$ is obtained from $m_{* \to s}$ by swapping the roles of s and t). These messages have the undesirable property that the variance of $m_{* \to s}$ scales according to the precision of $m_{t \to *}$ and not the variance. In other words, a large uncertainty in $m_{t \to *}$ leads to a large reduction in uncertainty in $m_{* \to s}$. The approximate inference algorithm reduces uncertainty every time that these messages get updated.

Note that the inputs to the computation of $m_{* \to z}$ are the current estimates of the marginal distributions $p(t)$ and $p(s)$. The marginal distributions are given by the product of the incoming messages to the variable (labeled 7 and 1 in Figure 1), for example $p(t_k) = m_{* \to t_k}(t_k) \cdot m_{t_k \to *}(t_k)$. Since message $m_{* \to t_k}$ is initially unavailable we initialize it to a uniform distribution and as we iterate the message passing schedule (2..7) we can obtain better estimates of the value of this message until eventual convergence.

Message (3) from the total sum factor is calculated by an analogous equation to equation (4) (belief propagation) and message (4) corresponds to simply adding β^2 to the variance of message (3) (this also corresponds to standard belief propagation). See [8] for more details on these computations.

Message (5) is the message from the observation factor. For a direct observation of the value of the rating this message would be a delta function centered on the observed

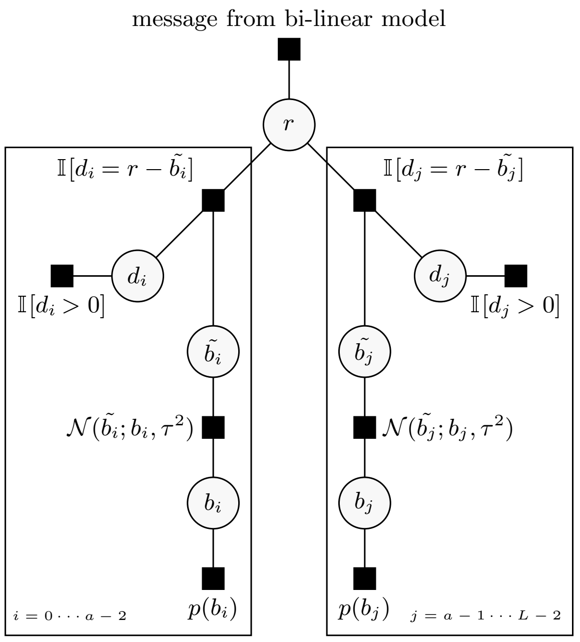

message from bi-linear model
r
$\mathbb{I}[d_i = r - \tilde{b_i}]$
$\mathbb{I}[d_j = r - \tilde{b_j}]$
d_i
$\mathbb{I}[d_i > 0]$
d_j
$\mathbb{I}[d_j > 0]$
$\tilde{b_i}$
$\tilde{b_j}$
$\mathcal{N}(\tilde{b_i}; b_i, \tau^2)$
$\mathcal{N}(\tilde{b_j}; b_j, \tau^2)$
b_i
b_j
$i = 0 \cdots a-2$
$p(b_i)$
$p(b_j)$
$j = a-1 \cdots L-2$

Figure 2: Factor graph for user-specific ordinal regression output model. This represents the model given in equation 1. Inference is performed by message passing.

value, a, that is, $\mathbb{I}(r = a)$. If another output model is used then this message will result from performing a message passing schedule on that model and will be a function of the downward message (4).

Message (6) is the Gaussian sum-product algorithm message from a sum factor to its summonds (derived from equation 3, see [8]) and messages (7) are calculated using equation (6).

The message passing schedule (2...7) is iterated until the marginal distribution of the predicted rating, $p(r)$, no longer changes.

3.1.3 Post-Processing

Finally messages (8) (for example $m_{\Sigma \to v_{kj}}$) are calculated and the posterior distribution in each case is obtained by multiplying this message by the prior. For example, for the variables v_{kj} with prior $\mathcal{N}(v_{kj}; \mu_{kj}, \sigma_{kj}^2)$ the posterior is updated as $p(v_{kj}) := m_{\Sigma \to v_{kj}}(v_{kj}) \cdot \mathcal{N}(v_{kj}; \mu_{kj}, \sigma_{kj}^2)$.

3.2 Feedback Models

Inference for the ordinal regression observation model is performed by approximate Gaussian EP message passing on the factor graph (Figure 2). The message update equations are described in [8] (and can be computed with Infer.net). This calculation is performed for each iteration of the schedule described in section 3.1. Note that now the marginal distribution for each user threshold must be stored.

The binary probit model is a special case of the ordinal regression model (see Section 2.3.2) [8].

3.3 Assumed-Density Filtering and EP

Once the posterior marginals for the variables $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{w}$ have been calculated as described in section 3.1 then we can discard all of the messages and continue, using this posterior as the prior for the variables for the next rating. In this way we pass once through the data, in an on-line fashion, incrementally incorporating each additional rating into the model beliefs. This method is a form of Assumed-Density Filtering (ADF) [16]. There are several advantages of adopting this approach. Firstly, the memory overhead is small as no messages need to be stored. Secondly, the on-line algorithm can immediately take account of new data when it becomes available, which may be desirable if we wish to produce up-to-date recommendations in practice.

The approximation can be improved by passing over the data several times, using the full EP message passing schedule. In this case we must store the messages into the $\mathbf{U}$, $\mathbf{V}$, $\mathbf{w}$ variables: $m_{\Sigma \to u_{ki}}(u_{ki})$, $m_{\Sigma \to v_{kj}}(v_{kj})$ and $m_{\Sigma \to w_n}(w_n)$ in addition to the posterior marginal distributions $p(\mathbf{U})$, $p(\mathbf{V})$ and $p(\mathbf{w})$ (equal to the product of these messages with the priors). The first time we pass through the data the updates are exactly the same as ADF except that we now store these messages. In subsequent passes through the data we obtain the messages from the marginals by dividing out the stored incoming messages. For example for the user messages:

$$m_{u_{ki} \to \Sigma}(u_{ki}) = \frac{p(u_{ki})}{m_{\Sigma \to u_{ki}}(u_{ki})}.$$

Since all messages and marginals are Gaussian distributions this is a straightforward computation. Once these messages are obtained for the data point in question we perform the iterative procedure of section 3.1 in order to compute updated messages (labeled 8 in Figure 1) to the user and item variables: $m'_{\Sigma \to u_{ki}}(u_{ki})$, $m'_{\Sigma \to v_{kj}}(v_{kj})$ and $m'_{\Sigma \to w_n}(w_n)$. Then we update the marginal distribution to be the product of the outgoing and incoming messages. For example for the user messages:

$$p'(u_{ki}) = m'_{\Sigma \to u_{ki}}(u_{ki}) m_{u_{ki} \to \Sigma}(u_{ki}),$$

and store the latest messages $m'_{\Sigma \to u_{ki}}(u_{ki})$, $m'_{\Sigma \to v_{kj}}(v_{kj})$ and $m'_{\Sigma \to w_n}(w_n)$ in place of the previous versions. See [5] for more details of this approach.

One downside of the full EP method is that the memory requirements scale in proportion to the number of ratings in the data set. This means that for large data sets the full forward backward approach may not be feasible, unless the data is broken into batches which are trained in sequence. However, we show in Section 4.1 that performance is still competitive even if on line ADF is performed. Also, if we assume a continuous stream of data is arriving (effectively an infinite data set) then an on-line approach makes even more sense from a practical point of view.

3.4 Dynamics

When using ADF for training, dynamics is trivial to implement: with each time step add the variance of the dynamics factor to the variance of the variable for which dynamics is being modeled. For example with the threshold model, after each day we update the variance of threshold beliefs by $\sigma_{ul}^{2(t+1)} := \sigma_{ul}^{2(t)} + \gamma^2$ where $\sigma_{ul}^{2(t)}$ is the variance of the marginal belief for threshold level l for user u at time step t. In a practical system ADF would probably be used so this is the method by which dynamics would be applied.

For full EP with dynamics, we divide each variable into separate copies, one for each time step in which it is involved with an observation. In this case we must cache the forward and backward messages into each copy of each variable as well as the marginal distributions for the copies. After each sweep forwards through the data we must perform a backward pass when the backward messages and marginals are updated from future observations, thus smoothing backwards in time. For a variable, say b, the backward messages from the noise factor, $f(b^{(t-1)}, b^{(t)}) = \mathcal{N}(b^{(t)}; b^{(t-1)}, \gamma^2)$, are calculated by $m'_{f(b^{(t-1)},b^{(t)})\to b^{(t-1)}}(b^{(t-1)}) =$

$$\int \mathcal{N}(b^{(t)}; b^{(t-1)}, \gamma^2) \frac{p(b^{(t)})}{m_{f(b^{(t-1)},b^{(t)})\to b^{(t)}}(b^{(t)})} db^{(t)},$$

and the marginal distribution for the variable $b^{(t-1)}$ is updated by

$$p'(b^{(t-1)}) = p(b^{(t-1)}) * \frac{m'_{f\to b^{(t-1)}}}{m_{f\to b^{(t-1)}}}.$$

The procedure is described in more detail in [5].

3.5 Parallel Inference

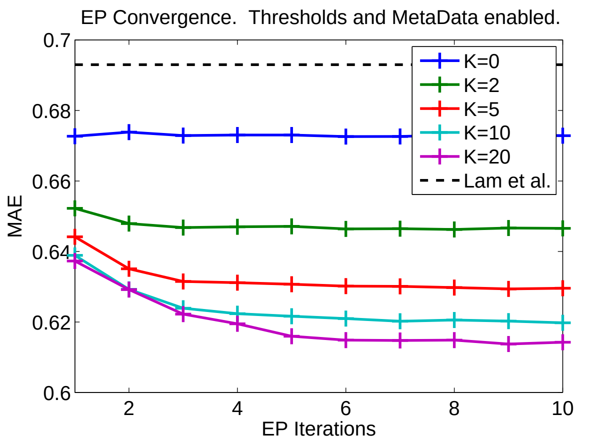
In general, fully factorised message passing algorithms (as described in [14]) can be parallelised as long as we take care that messages and cached marginals are always consistent. Frequently one would cache the marginal, $p(v_i)$, and calculate messages from a variable to a factor, $m_{i\to f}$, by dividing a cache of $p(v_i)$ by the message $m_{f\to i}$. As long as both the cache $p(v_i)$ and the incoming messages $m_{f\to i}$ are updated in one atomic step, computations of messages can be parallelised. We exploited parallelism in training on the Netflix data set (see Section 4.2). For an X core machine we divided the movies into X partitions and processed the sequences of ratings for each of these partitions in parallel. We used a monitor to ensure that two ratings by the same user are always processed in sequence to avoid a race condition. This method gave a 4× speedup on an 8 core machine, allowing us to process the 100,000,000 Netflix ratings in 2 hours with $K = 30$.

4. EXPERIMENTS

4.1 MovieLens

Firstly we investigate the predictive performance on the MovieLens data set. The data set contains 1,000,206 ratings of 3,952 movies by 6,040 users. Ratings are on an ordinal scale from 1 to 5. The data is 95.7% sparse. In addition to the ID for each user and item some meta data is also provided for 88% of the users and 98% of the items. The meta data provided is shown in Table 1.

In order to measure accuracy in a cold start situation we adopt the methodology of [11] and [13]. We randomly divide the users into two sets: a test set containing 10% of the users and a training set containing the rest of the users. First the model is trained on all the ratings by the training users according to the procedure outlined in Section 3.1. For each of the test users we train the model on a random subset of T% of their ratings for $T = 5\%$ and 75%. Then we use the model to predict the rest of the ratings for that user. Following [11] we report the Mean Absolute Error (MAE), $\frac{1}{N}\sum_{i=1}^{N} |\hat{r}_i - r_i|$, where r_i is the true rating and $\hat{r}_i$ is the predicted rating. We compare to their best result (MAE=0.6927 for $T = 75\%$) in each of the figures in this section. Note that we include the result of [11] on our plots for $T = 5\%$ although their training procedure had $T = 75\%$.



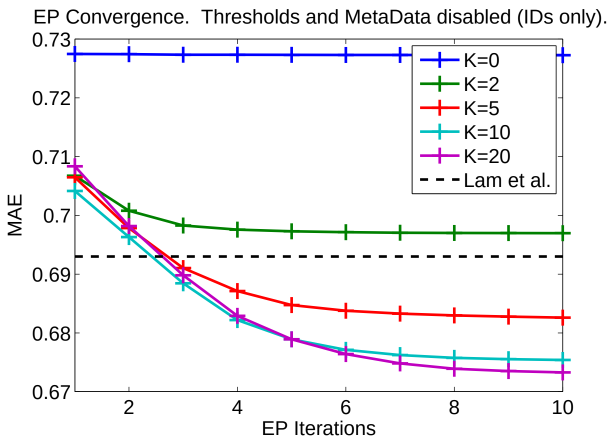


Figure 3: MovieLens Test MAE as a function of number of EP iterations, $T = 75\%$. The best result presented by [11] is included for comparison.

The $\mathbf{x}$ and $\mathbf{y}$ vectors are sparse binary, with a 1 present for each true feature. The bias uses the same features as the latent embedding so $b = \mathbf{x}^\top \mathbf{u} + \mathbf{y}^\top \mathbf{v}$.

We perform experiments using both the ordinal regression observation model and the direct observation model. Where the ordinal regression observation model is used we use the median of the predictive distribution $p(l|\mathbf{x}, \mathbf{y}, \Phi)$ in order to make the optimal decision to minimize MAE error measure. If the direct observation model is used then we use the mean of the predictive distribution $p(r|\mathbf{x}, \mathbf{y}, \mathbf{\Phi})$.

Figure 3 shows the effect of performing additional forward-backward passes of EP. The best result from [11] is included for comparison. Firstly we can note that more EP iterations help improve performance, and the larger the number of latent dimensions, K, the more forward-backward passes are needed in order to achieve convergence. In this figure we also compare the vanilla version of the model with no meta data (only user ID and item ID and the direct observation model - no ordinal regression) with the full model including all meta data features (Table 1) and user-specific ordinal

Job			
0	other or none specified	11	lawyer
1	academic/educator	12	programmer
2	artist	13	retired
3	clerical/admin	14	sales / marketing
4	college / grad student	15	scientist
5	customer service	16	self-employed
6	doctor / health care	17	technician / engineer
7	executive / managerial	18	tradesman / craftsman
8	farmer	19	unemployed
9	homemaker	20	writer
10	student		

0	**Age**< 18	4	$45 \leq$ **Age** < 49
1	$18 \leq$ **Age** < 25	5	$50 \leq$ **Age** < 55
2	$25 \leq$ **Age** < 34	6	**Age** > 55
3	$35 \leq$ **Age** < 44		

Gender	
0	Male
1	Female

Movie Genre			
0	Action	11	Musical
1	Adventure	12	Mystery
2	Animation	13	Romance
3	Children's	14	Thriller
4	Comedy	15	Sci-Fi
5	Crime	16	War
6	Documentary	17	Western
7	Drama		
8	Fantasy		
9	Film Noir		
10	Horror		

Table 1: Meta data provided by the MovieLens data set, left: user meta data, right: item meta data. In addition to these features we also use the IDs of the users and the IDs of the items. The x and y vectors are sparse binary, with a 1 present for each true feature.

regression threshold model. The full model strongly outperforms the vanilla model. In addition the vanilla model benefits more each additional EP iteration, taking more iterations before achieving best performance. Each sweep through the 1,000,000 ratings takes approximately 5 minutes on a 3.6GHz Pentium 4, for $K = 20$.

Figure 4 compares the performance of the model with and without the user-specific threshold model. The thresholds appear to help enormously to improve performance. Note that the threshold model outperforms [11] even with ADF (one EP iteration) and K=0 (zero latent dimensions!). For the case where K=0 the model is a simple linear model of the features in combination with the user-specific threshold model. To understand why the model performs so well, even with K=0 consider Figure 5. This shows a histogram of rating frequency for some randomly chosen users from the MovieLens data set. Note how each user seems to mainly use only a few rating levels each so accuracy is greatly improved by using the user-specific threshold model.

Figure 6 compares the performance of the model with and without the additional meta data features. We compare performance for $T = 5\%$ and $T = 75\%$. The meta data features seem to improve performance, especially for higher values of K. The threshold features are especially helpful in the case where $T = 5\%$ as here we face a severe cold start challenge - making predictions having seen only 5% of a user's ratings. In this case the meta data features push the performance above that of of [12] even though they used $T = 75\%$ for training and we only used $T = 5\%$.

4.2 Netflix

In October 2006, Netflix released a large movie rating data set and challenged the data mining, machine learning and computer science communities to develop systems that could beat the accuracy of their internal system Cinematch by a certain amount. The data were collected between October 1998 and December 2005 and represent the distribution of all ratings Netflix obtained during this time period. The training data set consists of 100,480,507 ratings from 480,189 randomly-chosen, anonymous users on 17,770 movie titles.

K	2	5	10	20	30
ADF	0.944	0.936	0.93	0.927	0.926
EP3	0.941	0.93	0.924	0.916	0.914
Lim & Teh (2007)	-	0.937	0.924	0.917	0.914

Table 2: Netflix RMSE scores for different settings. ADF is a single EP pass through the data. The EP3 run involves splitting the data into batches of 2,000,000 ratings and passing over each batch 3 times using EP before discarding the messages and moving on to the next batch (necessary to avoid running out of memory). With only ID features, as is the case for Netflix, the model is similar to the approach of [12] and their results are included for comparison. By using $K = 50$ and performing 20 EP iterations on 5,000,000 ratings, and then incorporating the rest of the ratings by ADF gives an RMSE of 0.910.

As part of the training data, Netflix also provides test data (the 'probe' set), containing 1,408,395 ratings. Since there is no information available per user and movie other than their IDs, we use unit vectors $\mathbf{x} := \mathbf{e}_i$ and $\mathbf{y} := \mathbf{e}_j$ to represent a user i and a movie j. We report root mean squared error (RMSE) scores according to convention for this data set. We use the ordinal regression prediction model and our prediction for each test data point is the expected rating value under the distribution over ratings $\sum_{a=1}^{5} a \cdot p(l = a|\mathbf{x}, \mathbf{y}, \mathbf{\Phi})$.

Table 2 shows the performance of the model for various settings. ADF corresponds to a single EP pass through the data. The EP3 run was performed by splitting the data into batches of 2,000,000 ratings and passing over each batch 3 times using EP before discarding the messages $m_{\Sigma \to u_{ki}}(u_{ki})$, $m_{\Sigma \to v_{kj}}(v_{kj})$ and $m_{\Sigma \to w_n}(w_n)$ and moving on to the next batch (necessary to avoid running out of memory). Note that we achieve comparable results to [12] with the EP3 approach.

It is interesting to visualize the learned embedding of some users and movies into trait space for $K = 2$ (see Figure

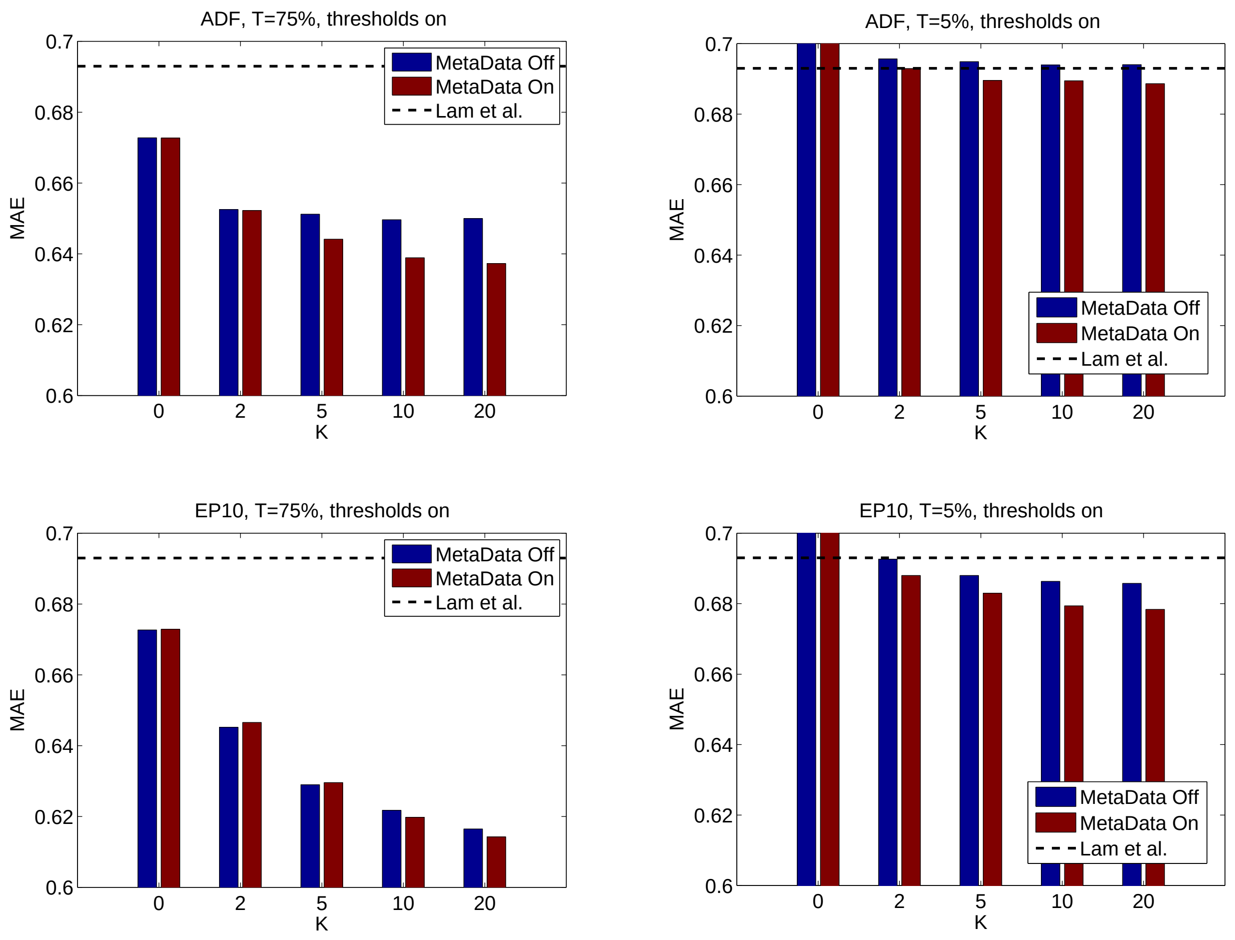


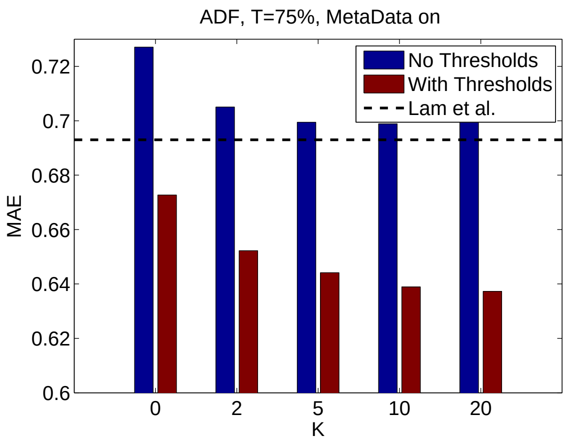
Figure 6: Movielens Test MAE, left:T=75%, right: T=5%. Comparing performance with and without meta data features. 'Meta data off' means only ID features are included. The top plot is produced by training the model by ADF (a single EP iteration) and the bottom plot is produced by training the model on 10 EP iterations.

7). In this plot the means of the first trait dimension are plotted against the means for the second trait dimension for each user and each item. Note that the learned embedding separates drama movies as much as possible from action moves in inner product distance. For any given user i in this space, movies j in the 'preference cone' $\{\mathbf{t}_j : \mathbf{s}_i^\top \mathbf{t}_j + b > c\}$ are preferred. We have depicted such a cone in the graph. Pictorially, a high ranking of a movie by a user leads to both a shift and rotation of the user towards the movie vector as well as a shift and rotation of the movie towards the user. This will bring other movies into the preference cone. This plot was produced by training with ADF. As further EP iterations are performed the users and movies become more evenly distributed in a ball in trait space allowing the model to separate more categories of movies.

One training procedure which boosts performance while retaining an incremental training method for incorporating new ratings was to first perform a number of full EP iterations on the ratings of a random subset of 5% of the users (5,000,000 ratings), called the 'seed' users. Since we have a large amount of training data per movie, even 5% of the data is enough to learn a good embedding of the movies in trait space. Then we incorporate the ratings of the rest of the users incrementally using ADF. For 20 iterations of EP on the seed users and $K = 50$ we achieve an RMSE of 0.910 on the probe set. It is worth pointing out that by the standards of many real world applications even the Netflix dataset is tiny. In a practical on-line setting where new ratings are arriving continuously we effectively have an infinite amount of data and, as the amount of data becomes very large, ADF could approach the same performance as full EP.

4.3 Advert Click Prediction for AdCenter

Finally, to illustrate a different type of application of a recommender system, we applied the model to the analysis of logs of clicks on adverts displayed above the search results on Live.com. The data consisted of 8,000,000 page views, each corresponding to a single user search query. At each page view three adverts are displayed ('impressions'). For each of the 24,000,000 impressions we know whether or not the advert was clicked on. We also have a large amount of meta data for the advert such as the ad title, advertiser account number and so on. Besides the advert information, the only information we have about the user is their query. We also have a number of features which are related to both the user and advert, for example, the 'matchtype' feature which indicates the degree of match between the advert and query, depending on what keywords the advertiser has selected as



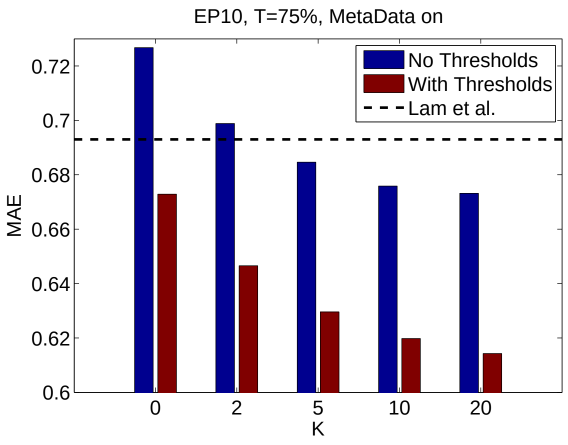


Figure 4: Movielens Test MAE, T=75%. The ordinal regression observation model (thresholds) is compared to the direct observation model (no thresholds). The top plot is produced by training the model by ADF (a single EP iteration) and the bottom plot is produced by training the model with 10 EP iterations.

being relevant to their products and services.

Because of the number of features available, including ones which indicate the degree of correspondence between an advert and a user, a simple linear model ($K = 0$) performs very strongly at this task so most of the features available were just used for the bias (the Φ vector). For $K > 0$ we used user and item features derived from the words in the query and the ad title. The $\mathbf{x}$ vector was chosen to be a sparse binary vector of the occurrence of words in the query from a lexicon of popular words and the $\mathbf{y}$ vector was chosen to be a vector indicating the occurrence of popular words in the advert title. The feedback model is the binary probit model (Section 2.3.2) where 'click' means positive feedback (c = TRUE) and 'no click' means negative feedback (c = FALSE). Figure 8 shows an embedding of query words and ad title words for $K = 2$ and training with ADF on a subset of the data. With two latent dimensions, the model appears to learn to separate navigational queries in one dimension from other queries by putting words like 'com' and 'Site' in one direction. After further EP iterations the adverts and queries

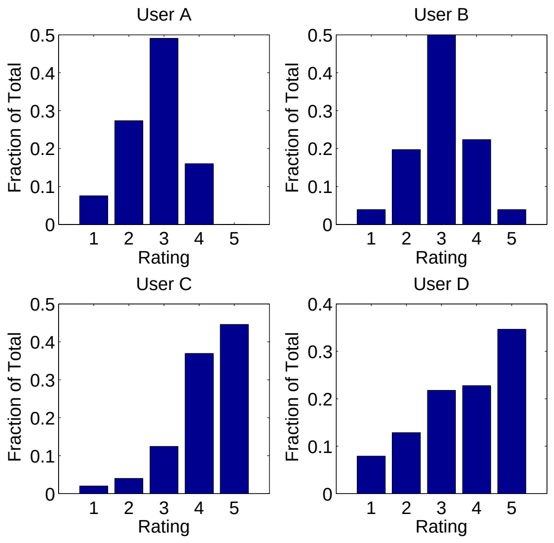


Figure 5: Relative frequency of use of each rating level for four users chosen at random from the Movielens data set. Notice that user A never uses rating 5 but users C and D both find 5 their most popular rating. None of the users frequently use rating 1.

become more distributed in trait space, leading to better performance.

5. CONCLUSIONS AND FUTURE WORK

We have presented a large scale recommender system in which user and item meta data are integrated. Experiments on the MovieLens data set show that meta data features are helpful, particularly for dealing with the cold-start problem with users new to the system. The user feedback model is flexible and results show that an ordinal regression model for user feedback can greatly improve accuracy. For training, inference is achieved by a novel combination of Variational Message Passing and EP. We can achieve state-of-the-art performance if we use ADF for training which is an on-line, incremental method so recommendations can always be up to date. Results are improved further by using full EP for training but in practice we anticipate using ADF as new data will continuously be arriving and the performance of ADF will approach that of full EP.

In order to predict what items may be of interest to a user we potentially have to consider all available items. In order to make this process more efficient we are investigating two approaches: hashing [6] and Kd trees [9] and this is the main focus of future work. The system presented here is being developed for deployment in a commercial on-line service.

6. REFERENCES

[1] Netflix Cinematch: *http://www.netflix.com*.

[2] R. M. Bell and Y. Koren. Lessons from the Netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9:75–79, 2007.

[3] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th ACM Conference*

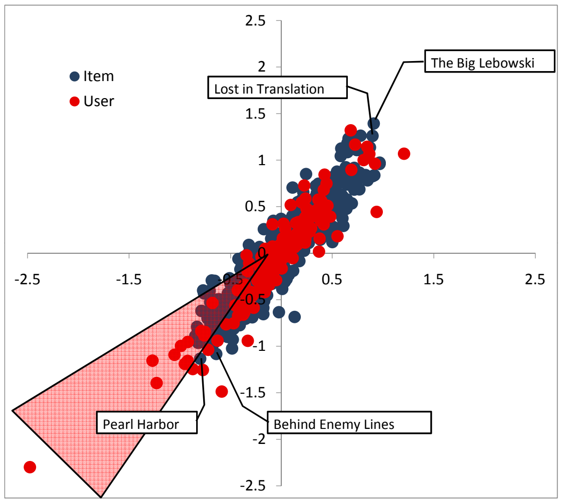


Figure 7: An embedding of 100 users and 500 movies into a 2-dimensional trait space (for a run with $K = 2$). The mean of the first user trait $(u_{(0)i})$ is plotted against the mean of the second user trait $(u_{(1)j})$ to produce the red dots. The mean of the first movie trait $(v_{(0)i})$ is plotted against the mean of the second movie trait $(v_{(1)j})$ to produce the blue dots. Some movie titles are labeled. The triangle suggests the region within which user 2114400 (the dot in the bottom left) would like movies because similarity between users and moves is measured by the inner product. Note how the learned embedding separates drama movies (top-right corner) as much as possible from action movies (bottom-left corner).

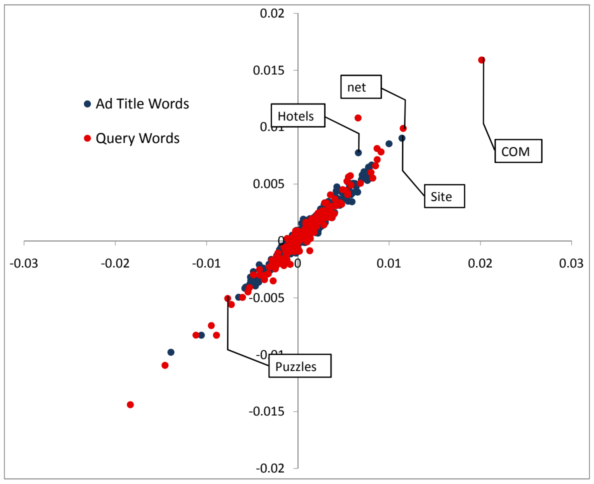


Figure 8: An embedding of 190 query words and 339 advert title words into a 2-dimensional trait space (for a run with $K = 2$). The mean of the first query word trait $(u_{(0)i})$ is plotted against the mean of the second query word trait $(u_{(1)j})$ to produce the red dots. The mean of the first advert word trait $(v_{(0)i})$ is plotted against the mean of the second advert word trait $(v_{(1)j})$ to produce the blue dots. Some words are labeled.

on Uncertainty in Artificial Intelligence, pages 34–52, 1998.

[4] W. Chu and Z. Ghahramani. Gaussian processes for ordinal regression. pages 1019–1041, 2005.

[5] P. Dangauthier, R. Herbrich, T. Minka, and T. Graepel. Trueskill through time: Revisiting the history of chess. In *Advances in Neural Information Processing Systems 20*, pages 337–344, 2008.

[6] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 271–280, New York, NY, USA, 2007. ACM Press.

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

[8] R. Herbrich, T. Minka, and T. Graepel. TrueSkill(TM): A Bayesian skill rating system. In *Advances in Neural Information Processing Systems 20*, pages 569–576, 2007.

[9] M. K. Hughey and M. W. Berry. Improved query matching using kd-trees: A latent semantic indexing enhancement. *Information Retrieval*, 2:287–302, 2004.

[10] F. R. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47(2):498–519, 2001.

[11] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, pages 208–211, 2008.

[12] Y. J. Lim and Y. W. Teh. Variational Bayesian approach to movie rating prediction. In *Proceedings of KDD Cup and Workshop*, 2007.

[13] Benjamin Marlin. Collaborative filtering: A machine learning perspective. Master's thesis, University of Toronto, 2004.

[14] T. Minka. Divergence measures and message passing. Technical Report MSR-TR-2007-173, Microsoft Research Ltd., 2005.

[15] T. Minka, J.M. Winn, J.P. Guiver, and A. Kannan. Infer.NET 2.2, 2009. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

[16] Thomas Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, MIT, 2001.

[17] A. Mnih R. Salakhutdinov and G. Hinton. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th Annual International Conference on Machine Learning*, pages 791–798, 2007.

[18] J. B. Schafer, J. Konstan, and J. Riedi. Recommender systems in E–commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166, 1999.

[19] H. R. Varian and P. Resnick. Recommender systems. *Communications of the ACM*, 40:56–58, 1997.

[20] J. M. Winn. *Variational message passing and its application*. PhD thesis, Department of Physics, University of Cambridge, 2003.