Latent Factor Models with Additive and Hierarchically-smoothed User Preferences

Amr Ahmed *
Google Inc.
amra@google.com

Bhargav Kanagal*
Google Inc.
bhargav@google.com

Sandeep Pandey*
Twitter
spandey@twitter.com

Vanja Josifovski*
Google Inc.
vanjaj@google.com

Lluis Garcia Pueyo*
Google Inc.
lgpueyo@google.com

Jeff Yuan
Yahoo! Research,
yuanjef@yahoo-inc.com

ABSTRACT

Items in recommender systems are usually associated with annotated attributes such as brand and price for products; agency for news articles, etc. These attributes are highly informative and must be exploited for accurate recommendation. While learning a user preference model over these attributes can result in an interpretable recommender system and can hands the cold start problem, it suffers from two major drawbacks: data sparsity and the inability to model *random effects*. On the other hand, latent-factor collaborative filtering models have shown great promise in recommender systems; however, its performance on rare items is poor. In this paper we propose a novel model LFUM, which provides the advantages of both of the above models. We learn user preferences (over the attributes) using a personalized *Bayesian* hierarchical model that uses a combination (*additive model*) of a globally learned preference model along with user-specific preferences. To combat data-sparsity, we smooth these preferences over the item-taxonomy using an efficient forward-filtering and backward-smoothing inference algorithm. Our inference algorithms can handle both discrete attributes (e.g., item brands) and continuous attributes (e.g., item prices). We combine the user preferences with the latent-factor models and train the resulting collaborative filtering system end-to-end using the successful BPR ranking algorithm. In our extensive experimental analysis, we show that our proposed model outperforms several commonly used baselines and we carry out an ablation study showing the benefits of each component of our model.

Categories and Subject Descriptors

G.3 [**Probability And Statistics**]: Statistical Computing; I.2.6 [**Computing Methodologies**]: Artificial Intelligence-Learning

General Terms

Algorithms, Experimentation, Performance

*The work was performed at Yahoo! Research.

WSDM'13, February 4–8, 2013, Rome, Italy.

Keywords

Recommendation, Latent Variable Models, Collaborative Filtering, Inference, Factor Models

1. INTRODUCTION

Personalized recommender systems are ubiquitous with applications to computational advertising, content suggestions, e-commerce and search engines. These systems use the past behavior of users to recommend items to them that are likely to be of interest. Significant advancements have been made in recent years to improve the accuracy of such personalized recommendation systems and to scale the algorithms to large amount of data.

One of the most extensively used techniques in these recommendation systems is the latent factor model and its variants [2, 13, 14, 16] (see Koren et al. [15] for an excellent survey). The key idea behind latent factor models is to project the users and items into a smaller dimensional space (such lower dimensional projections are called *factors*), thereby clustering similar users and items. Subsequently, the interest (similarity) of a user to an unrated item is measured and the most similar item(s) is recommended to the user. While factor models have been very successful with the Netflix contest top performers [5, 13] and tag recommendation [19] being some of the success stories, these models have certain shortcomings. Much of these techniques do not take into account, the auxiliary information that is typically associated with items. In most domains, the items being recommended have a rich set of attributes which are highly indicative of users' preferences, i.e., they determine the users' intent to purchase the item. For instance, in the movie recommendation application, there is information about the director, lead actor, duration of the movie, time of release, all of which play a significant role in the user selecting the movie. Movies that have reputed directors are usually chosen despite lack of user interest as modeled by a latent factor model. In the retail product recommendation, we have attributes such as the brand and price that determines user's interest in the product. The other disadvantage with this approach is that of sparsity, commonly referred to as the *cold-start* problem.

In order to leverage such auxiliary information, there has been a number of efforts to build content-based recommender systems [6]. While content-based approaches work well in the presence of cold-start, they are outperformed by the latent factor models in warm-start scenarios, i.e., when we have enough training data. Hybrid recommendation models [9, 8, 6] that combine collaborative and content information have been proposed to overcome this issue. However, as we will show in our experiments, these hybrid methods are expensive to to train and does not perform well when the

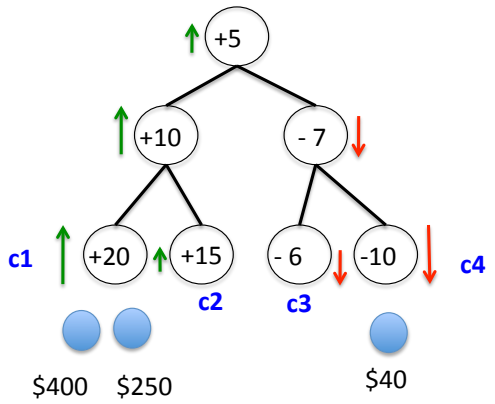

Figure 1: Illustrating the role of hierarchy and additive model in learning user preferences. Each internal node gives the user preferred mean price as an addition over the global mean price at the same node. The user bought items shaded at the leaves of the tree and our model inferred the number inside each internal node of the user's tree. See Section 3.1 for more details.

attributes of the items live in a very high-dimensional space. Furthermore, it is not easy to augment these hybrid models with additional background information, such as a taxonomy over items.

In this paper we propose the LFUM (Latent Factor augmented with User preferences Model) which combines latent factor models along with a user preference models for improving personalization in recommendation. A key to our model is that we capture the user preferences over each of the item attributes, using an addition of a global preference model and a per-user personalized preference model. We smooth these user preferences over a *taxonomy* of the items, to alleviate the aforementioned sparsity problem. Taxonomies are commonly available for many domains [1, 12]. For example, consider the price attribute: Suppose that Alice bought an expensive laptop. In this case, our model can learn that Alice buys `electronics` items at a price $200 more than the global average. The preferences in our model are learned using a *hierarchical additive* model. We develop an efficient forward filtering, backward-smoothing algorithm over the taxonomy. Continuing with the above example, if Alice buys expensive electronics, then that increases the likelihood that she would purchase an expensive phone (even though she has not purchased any phone). We support both discrete-valued (e.g., brand) and continuous-valued (e.g., price) attributes. For continuous variables, we extend the well known Kalman filtering/smoothing [11] algorithm to multiple interacting hierarchies using an additive model. Furthermore, we combine these user preferences with a latent factor model and optimize an efficient lower bound on the joint objective function using the Bayesian personalized ranking algorithm of Rendle et al. [18]. We apply LFUM to the real world problem of *smart ad* selection for Display advertising and demonstrate significant improvements in the recommendation accuracy using our approaches.

Outline: The rest of this paper is organized as follows. First in Section 2 we establish notations, formalize the problem and review relevant background. Then in Section 3 we describe detail our model. In Section 4 we give an efficient inference algorithm and in Section 5-6 we show a comprehensive evaluation of our model against several baseline. Section 7 covers related work and we conclude in Section 8.

2. PRELIMINARIES

In this Section we provide background for the various concepts in the paper. We begin with the notations used in the paper and explain commonly used recommender systems in the literature, namely latent factor models and hybrid recommender systems.

2.1 Notations

Let $\mathcal{I}$ be a set of items and $\mathcal{U}$ be a set of users. For each $u \in \mathcal{U}$, we let $\mathcal{I}_u$ denote the set of items the user interacted with (ex. purchased before). Each item $I \in \mathcal{I}$ is described by a set of continuous (such as price) and discrete (such as brand) attributes. We let $I.x$ denote a continuous attribute of I and $x \in \mathcal{X}$, where $\mathcal{X}$ is the set of continuous attributes. Similarly, we let $I.y$ denote a discrete attribute of I for $y \in \mathcal{Y}$, where $\mathcal{Y}$ is the set of discrete attributes. In addition, each item is associated with a category. We use $\pi(I)$ to denote the category of item I. The categories themselves are arranged in a tree-like taxonomy (or a hierarchy). For a given node in the tree, say n, we abuse notation a a bit, and we let $\pi(n)$ denote the parent of node n in the tree, and $\mathcal{I}_n$ denotes the children of node n. If n is a leaf category, then $\mathcal{I}_n$ is a set of items, and if n is an internal node, then $\mathcal{I}_n$ is a set of subcategories. We let L denote the depth of the tree, where the root is at level 0 and the leaf categories are at level $L-1$. We also use $T(n)$ to denote the subtree rooted at node n. Our goal is to learn user preferences over attributes for each category in the tree and subsequently, use these preferences to augment a factor model, thereby improving recommendation of items to users.

2.2 Latent Factor Models

The input to a recommender system is a sparse (partially populated) user-item matrix where the entries correspond to an interaction between a user and an item – either in terms of a rating or a purchase. The goal of the recommender system is to predict, for each user u, a ranked list of items. We assume that each user u and item i can be represented by latent factors $\mathbf{v_u}$ and $\mathbf{v_i}$ respectively, which are vectors of size $1 \times K$. User u's affinity/interest in item i (denoted by z_{ui}) is assumed to follow this model:

$$z_{ui} = \langle \mathbf{v_u}, \mathbf{v_i} \rangle$$

The learning problem here is to determine the best values for $\mathbf{v_u}$ and $\mathbf{v_i}$ (for all users u and all items i) based on the given rating matrix; we denote these parameters by Γ. While traditional approaches to matrix factorization try to regress over the known entries of the matrix, a more successful approach is the recently proposed Bayesian personalized ranking (BPR) [18]. Here, the trick is to do regression directly over the *ranks* of the items, rather than the actual ratings since the goal is to construct ranked lists. Also, we only have *implicit feedback* from the users (i.e., we will have a rating between 1 to 5, but only know that the user made a purchase). In this case, regression over the actual numbers of purchases is not meaningful. The main objective in BPR is to discriminate between items bought by the user from those that were not bought. In other words, we need to learn a ranking function R_u for each user u that ranks u's interesting items higher than the non-interesting items. In other words, if item i appears in user u's purchase list B_u and item j does not appear in B_u, then we must have $R_u(i) > R_u(j)$. For this, we need to have: $z_{ui} > z_{uj}$. Based on the above arguments, our likelihood function $p(R_u|\Gamma)$ is:

$$p(R_u|\Gamma) = \prod_{u \in U} \prod_{i \in B_u} \prod_{j \notin B_u} \sigma(z_{ui} - z_{uj})$$

Following Rendle et al. [18], we approximate the non-smooth, non-differentiable expression $z_{ui} > z_{uj}$ using the logistic sigmoid function $\sigma(z_{ui} - z_{uj})$, where $\sigma(z) = \frac{1}{1+e^{-z}}$. We use a Gaussian prior $N(0, \sigma)$

over all the factors in Γ and compute the MAP (maximum a posteriori) estimate of Γ. The posterior over Γ (which needs to be maximized) is given by:

$$\begin{aligned} p(\Gamma|R_u) &= p(\Gamma)p(R_u|\Gamma) \\ &= p(\Gamma)\prod_{u\in U}\prod_{i\in B^u}\prod_{j\notin B_u}\sigma(z_{ui}-z_{uj}) \end{aligned}$$

2.3 Hybrid Recommender Systems

As described in Section 1, hybrid recommender systems [6] combine latent factor models along with the content-based approaches based on user and item features. Much of the approaches work by first fitting a regression function over the item features against the target feedback and subsequently modeling the residual with a latent factor model. We can represent hybrid recommender systems in our context using the following affinity expression.

$$z_{ij} = \langle v_u, v_i\rangle + \sum_{x\in\mathcal{X}} W_x i.x + \sum_{y\in\mathcal{Y}} W_y i.y$$

The first term in the expression corresponds to the traditional latent factor model. In the second term, W_x denotes the weights for each of the discrete attribute set $\mathcal{X}$ (Note that we abuse notation slightly: $i.x$ corresponds to the value of attribute x of item i). Similarly, the third term corresponds to the weights learned over the continuous attributes. In practice, we can extend hybrid recommender systems to learn both the weights and the user/item factors simultaneously. However, such an approach is problematic in two aspects: First, the weights are learned over all the users, thus we cannot model user personalization. Further more, we cannot learn per-user weights owing to the sparsity of the data. In addition, it is unclear how to learn weights for the continuous attributes without discretizing them and as such introducing additional noise. In the next section we will introduce our LFUM which addresses these problems in a principled way.

3. AUGMENTED LATENT FACTOR MODELS: LFUM

This section describes our proposed approach, a Latent Factor augmented with a User preference Model (LFUM). We will begin by first describing the intuition behind the model and then give a formal description. Our model is a hybrid model that combines the observed item attributes with a latent factor model. However, our model is novel since it doesn't learn a regression function over item attributes but rather learn a user-specific probability distribution over item attributes. Let us define $q_x(i.x|u)$ to be the probability that user u likes item i based on item's i value of attribute x. We abuse notation slightly and write $q^u_x(i.x)$ to denote $q_x(i.x|u)$. One could imagine that we can combine these probabilities for each attributed and learn a linear function that predicts the user affinity to each item. However, the observed attributes might not be enough to explain why the user would buy item i instead of item j. For example, the user might prefer item i over item j based on an attribute that is not observed (for instance, the shape or the color of the item). Therefore, we define $P(R_u|\Gamma, q)$ to denote the *ranking* probabilities for user i. As we made explicit in this probability, the raking depends on q and is parameterized by a set of parameters Γ to be defined later. Our goal thus is to solve the following optimization problem:

$$\begin{aligned} &\max_\Gamma \prod_u P(R_u|\Gamma) \\ = &\max_\Gamma \int_q P(q)\prod_u P(R_u|\Gamma, q)dq \end{aligned} \quad (1)$$

where $q = \{q_x(.)\}_{x\in\mathcal{X}} \cup \{q_y(.)\}_{y\in\mathcal{Y}}$, is the product probability of all item attributes. In essence, Eq (1) treats q as a *hidden variable* and integrates over it to define the marginal observed rankings for all users. There are several advantages to treating q as a hidden variable: 1) it makes it easier to combine continuous and discrete attributes (with varying cardinalities) as we map each attribute's contribution into a number between [0,1]; 2) treating q as a probability distribution opens the door for using several hierarchical smoothing that combats data sparsity and allows us to introduce regularization in a principled way. To fully define our model we need to answer the following three questions:

1. How to define q for both discrete and continuous attributes?
2. How to define $P(R_u|\Gamma, q)$?
3. How to optimize the seemingly intractable objective function in Eq (1)?

In the following two subsections, we will answer the first two questions and then in Section 4, we will tackle the third question.

3.1 User Preference Models

3.1.1 The Intuition: Background Subtraction

Before delving into the technical details of the generative process, we begin by giving an overview of our model. Let us assume that items represent products, and that we have only one continuous attribute (price of the item), and one discrete attribute (brand of the item). Moreover, let the nodes in the taxonomy represent product categories. Let us first consider preferences over item prices. Given the items purchased in the dataset, one could postulate that there is a global (across-users) average price for each category, this is not the average price of items under the category, but rather the average price of the *actual purchases* of items under the category. For example, under the category `smart phones`, the average price of phones purchased by the users is say $200 even though the average price of the distinct phones might be much higher (or lower). However, this modeling assumption is rather inadequate as it ignores user personalized preferences over the category. To remedy that, we could hypothesize that users' personal preferences can be modeled as an addition over the global preference. For example, a user might be more inclined to buy expensive smart phones and as such her mean price over this category is above the category's global mean price. Similarly, another user might have a mean price that is much smaller than the global mean price, i.e. she prefers to buy the cheapest smart phones. Therefore, we can look at the global price of a given category as the average across all users, and we can model each user's preferences as an addition over it (either positive or negative).

However there is a problem with this approach as it ignores the taxonomy. It is quite natural to assume that both the global and personalized preferences are smooth over the taxonomy. This has the advantage of combating data sparsity. To see this point, we refer the reader to Figure 1. This figure depicts the preferences of a given user over the prices of different categories. In this figure, we observed two purchases of this user under category $c1$ and we deduced from them, that under category $c1$, the user's average price is above the global average price. Moreover, we observed a single purchase from this user under category $c4$ and we concluded that the user prefers inexpensive items under this category (i.e the user average price is below the global average price). However, what can we say about the user's preferences over categories $c2$ and $c3$? Intuitively we expect $c2$ to follow the same pattern as $c1$ and $c3$ to follow the same pattern as $c4$. However, the exact deviations from the global mean prices depend on two factors: First, how confident

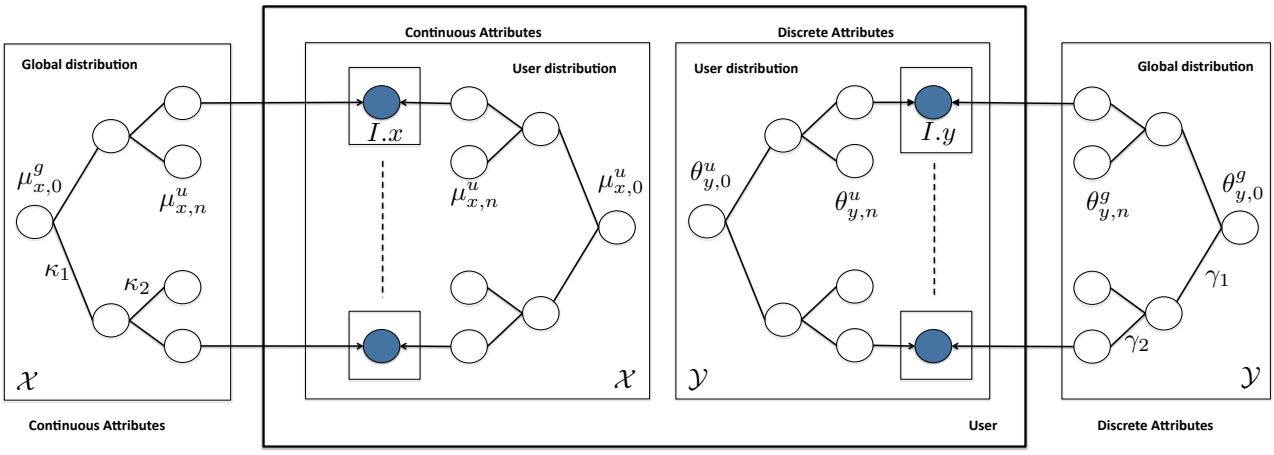

Figure 2: The graphical model, see text for more details, Section 2 for notations and Algorithm 1 for the full generative process

we are about our estimate for $c1$ and $c4$ and second, the variance of prices over the tree (we will formalize these notions in the generative process shortly). However, the main message of this figure is that given a few observations about each user, we could still have a hypothesis about each user's preferences over *all* the categories in the tree. Informally, this is possible, since we can pool data across users to create hypotheses about the global preferences and then use these hypotheses to draw conclusions about each user (a process called smoothness – Section 4).

The same intuition applies also to discrete attributes such as the brand of the item. Given all purchases, one could form a global brand preference over each category and then use it as a prior over the preferences of each user. This process is known in the literature as smoothness with a background model. However, this process alone does not achieve smoothness over close-by nodes in the taxonomy. For example, let us assume that a user bought an iPhone, i.e a smart phone with "Apple" as the brand. Since smart phone is a subcategory of "electronic gadget", we could infer that this user prefers to also buy other "Apple" products under other subcategories of "electronic gadgets", such as "laptops" for instance. However, if we don't observe any purchase from the user under the "electronic gadgets" category, we might *shrink* our estimate of the user's preferences over "electronic gadgets" to the global distribution. We will see that our model achieves this property using an additive model as we detail below.

3.1.2 The Generative Model

Armed with the above intuitions, we are now ready to detail the generative process with reference to Figure 2 and Algorithm 1. For a continuous attribute x, we let $\mu^g_{x,n}$ denotes the global mean value of this attribute at node n in the taxonomy. We connect those means over the tree using a Gaussian-cascade process, where each node's value acts as the mean of a normal distribution that is used to generate the children of node n. Put it another way, the mean of node n is generated from a normal distribution centered at the mean value of its parent in the tree. The variance of this normal distribution depends on the level of the node n, and we denote it as $\kappa_{Level(n)}$. We expect the variance to be high for levels close to the root and small for levels close to the leaf categories. Similarly, for each user, we let $\mu^u_{x,n}$ denotes user u's deviation at node n from the global mean at the same node. Again we connect those user-specific deviations over the tree using a Gaussian-Cascade process with the same variances used in the global process. We use the same variances since estimating a user-specific variance is prone to over-fitting. Moreover, those variances act as priors and during inference we compute a posterior value for each user based on his observations (see Section 4.1). Finally, for each user, we observe a set of purchased items under each leaf category (depicted as shaded nodes in Figure 2 and as I^u_n in the generative process, i.e. item purchased by

Algorithm 1 Description of the generative model

Require: set of users U , set of items I
Generate global preferences:
for all $x \in \mathcal{X}$ // *Continuous Attributes* **do**
  for all categories n starting from the root downward: **do**
    Draw $\mu^g_{x,n} \sim N(\mu^g_{s,\pi(n)}, \kappa_{\text{Level}(n)})$
  end for
end for
for all $y \in \mathcal{Y}$ // *Discrete Attributes:* **do**
  for all categories n starting from the root downward: **do**
    Draw $\theta^g_{y,n} \sim \text{Dir}(\gamma_{\text{Level}(n)}\theta^g_{y,\pi(n)})$
  end for
end for
Generate User Preferences
for all $u \in \mathcal{U}$ **do**
  for all $x \in \mathcal{X}$ // *Continuous Attributes* **do**
    for all categories n starting from the root downward: **do**
      Draw $\mu^u_{x,n} \sim N(\mu^u_{x,\pi(n)}, \sigma_{\text{Level}(n)})$
    end for
  end for
  for all $y \in \mathcal{Y}$ // *Discrete Attributes:* **do**
    for all categories n starting from the root downward: **do**
      Draw $\theta^u_{y,n} \sim \text{Dir}(\gamma_{\text{Level}(n)}\theta^u_{y,\pi(n)})$
    end for
  end for
  Generate Attributes of Items the user interacted with
  for all $i \in \mathcal{I}_u$ **do**
    for all $x \in \mathcal{X}$ **do**
      $i.x \sim N(\mu^g_{x,\pi(i)} + \mu^u_{x,\pi(i)}, \kappa_L)$
    end for
    for all $y \in \mathcal{Y}$ **do**
      $i.y \sim \text{Multinomial}(\theta^u_{y,\pi(i)} + \gamma_L\theta^g_{y,\pi(i)})$
    end for
  end for
end for

the user under leaf category n). The final step in the generative process is to generate the attributes of those items from a normal distribution centered at the sum of the global mean and the user-specific deviation under each category (step 27). The variance of this distribution is κ_L where L is the depth of the tree (recall that the leaf categories, which are the parents of the items, appear at level $L-1$ in the tree) . We should note here that the item attributes are not user specific, however, we still generate them for each user but rather from a different user-specific distribution. This is the key step that allows the model to connect global means with each user local mean as we will see during posterior inference. Put in another way, the item attributes acts as constraints that forces the model during inference to factor the observed prices into a global and user-specific components.

We model the discrete attributes of the items in a similar fashion but with a Dirichlet-Multinomial cascade instead (step 7-11 for the global distributions and 19-23 for the user specific distributions). Recall that the Dirichlet is the conjugate prior for the Multinomial distribution. Usually a Dirichlet distribution is specified as $Dir(\rho)$ for a given parameter vector ρ, thus the mean and variance of the sampled multinomials depend on ρ. A more expressive way of specifying a Dirichlet distribution is as $Dir(\gamma\theta)$, here γ acts as the variance and θ acts as the mean. The higher the value of γ the less the variance of the distribution. We use this representation to connect nodes over the taxonomy as the case of the Gaussian distribution with θ playing the role of μ and γ playing the role of κ (see steps 9 and 21). We expect γ to be small for nodes close to the root (to give high variances) and high for nodes close to the leaf (to give small variances). Finally to generate the discrete attributes of items purchased by a user, we use an additive model that combines the global and local distributions of the item's category (step 30). In this case, we weight the global preference by a scalar value γ_L which specifies how much the user preference can deviate from the global distribution (this plays the same role as κ_L in step 27).

3.2 Item Ranking: $P(R_u|\Gamma, q)$

How can we use the learned user preferences to perform recommendation? In other words, how can we define the item preference ranking probabilities, $P(q)\prod_u P(R_u|\Gamma, q)$? Using the principle of BPR, we first let z_{ui} denotes user u's affinity towards item i and we define it as follows:

$$z_{ui} = \sum_{x\in\mathcal{X}} \alpha_x q_x^u(i.x) + \sum_{y\in\mathcal{Y}} \beta_y q_y^u(i.y) + \epsilon_{ui} \quad (2)$$

where α and β are attribute-specific weights that determine the importance of this attribute in computing the affinity. The preference probability is computed as follows:

$$q_x^u(i.x) = N\big(\mu^g_{x,\pi(i)} + \mu^u_{x,\pi(i)}, \kappa_L\big) \quad (3)$$

$$q_y^u(i.y) = \text{Multinomial}\big(\theta^g_{y,\pi(i)} + \gamma_L\theta^u_{y,\pi(i)})\big) \quad (4)$$

The final component in computing z_{ui} is thus ϵ_{ui} which is a random noise that accounts for interactions not explained by the observed attributes (which is termed as a random effect model in the statistics literature [10]). We model ϵ_{ui} using a latent factor model as follows:

$$\epsilon_{ui} = < v_u, v_i > \quad (5)$$

where $< ., . >$ denotes dot product and $v_u \in \mathcal{R}^K$ is the user factor and $v_i \in \mathcal{R}^K$ is the item factor. The parameters of this system are:

$$\Gamma = \{\alpha_x\}_{x\in\mathcal{X}}, \{\beta_y\}_{y\in\mathcal{Y}}, \{v_u\}_{u\in\mathcal{U}} \text{ and } \{v_i\}_{i\in\mathcal{I}}.$$

4. PARAMETER LEARNING

We seek to optimize the following objective function:

$$\max_\Gamma \int_q P(q)\prod_u P(R_u|\Gamma, q)dq$$

However, this objective integrates over the space of all probability distributions, which makes the optimization intractable. To remedy this, we instead optimize the following variational lower bound:

$$\max_\Gamma \int_q P(q)\prod_u P(R_u|\Gamma, q)dq \quad (6)$$

$$\geq \max_\Gamma \prod_u P(R_u|\Gamma, \hat{q}) \quad (7)$$

where $\hat{q}$ is the map estimate of q, i.e., the best distribution q that explains the user preferences over the item attributes. The above lower bound gives rise to the following simple algorithm. First learn the map estimate of q using the observed attributes of the items bought by each user, and then use $\hat{q}$ to learn the value of Γ that best explains the item rankings by each user, R_u. We will tackle each of these problems in the following sections.

4.1 Posterior Inference of Preferences

Given a set of user-item interactions, the goal is to learn a posterior distribution over q, $\hat{q}$. This is equivalent to learning the posterior values of the global parameters μ^g, θ^g and the posterior values of the user specific parameters μ^u, θ^u. We address the continuous case in Section 4.1.1 and the discrete case in Section 4.1.2.

4.1.1 The Continuous Case

Let $\mathcal{I}_u$ be the set of items purchased by a given user, and let $\mathcal{I}_g$ the set of all purchased items by all users. Our goal is to compute $P(\mu^g_x|\mathcal{I}_g)$ and $P(\mu^u_x|\mathcal{I}_g, \mathcal{I}_u)$ for all users u and for all attributes x, where $\mu^g_x = (\mu^g_0, \mu^g_1, \cdots, \mu^g_N)$, and $\mu^u_x = (\mu^u_0, \mu^u_1, \cdots, \mu^u_N)$, where N is the total number of nodes in the taxonomy. We give below the inference algorithm for a given attribute x, thus the following algorithm is repeated for each attribute separately.

The problem of this posterior computation is the dependencies between μ^g_x and μ^u_x for all $u \in \mathcal{U}$ because of the additive form of the observation model (Step 27 in Algorithm 1). Therefore we resort to an **alternating algorithm** similar to loopy belief propagation, that computes each of these quantities conditioned on the other. Lets first assume that we know μ^x_g. Thus conditioning on μ^g_x, the inference problem over user's means **decouples** into a set of $|\mathcal{U}|$ independent problems. For user u, we just need to compute $P(\mu^u_x|\mathcal{I}_u, \mu^g_x)$. Once we compute this quantity for all users, we alternate and compute $P(\mu^g_x|\mathcal{I}_g, \{\mu^u_x\}_{u\in\mathcal{U}})$. We **repeat this alternating procedure** until convergence, where convergence is assessed by measuring changes in the value of μ^g_x. The algorithm is summarized below:

1. Repeat
   (a) $\forall u \in \mathcal{U}$:
      i. Compute $P(\mu^u_x|\mathcal{I}_u, \mu^g_x)$.
   (b) Compute $P(\mu^g_x|\mathcal{I}_g, \{\mu^u_x\}_{u\in\mathcal{U}})$
2. Until convergence.

We focus here first on $P(\mu^u_x|\mathcal{I}_u, \mu^g_x)$. This inference problem is known in the literature as multi-scale Kalman filter [11], albeit with a **shifted observations** here. Recall that the value of the x attribute of all items I in $\mathcal{I}_u$ are generated from a normal distribution with

mean = $\mu^g_{x,\pi(I)} + \mu^u_{x,\pi(I)}$. Since we hold μ^g_x fixed, this is equivalent to generating the values $I.x - \mu^g_{x,\pi(I)}$ for all $I \in \mathcal{I}_u$ form the leaf categories of the user tree $\mu^u_{x,\pi(I)}$, which is exactly the posterior inference problem addressed by the multi-scale KF algorithm [11].

The multi-scale KF algorithm [11] proceeds in two stages: upward phase (from the leafs to the root) and downward phase (from the root to leaf). The upward phase is known as the filtering phase. In this phase we compute the posterior probability of each node conditioned on its immediate children. Once we reach the root of the tree, the root has now received information from all nodes in the tree, and we move to the second downward phase known as the smoothing phase. In this phase, we propagate information down the tree to get the posterior mean of each node conditioned on the whole tree (not only its children). The recursive equations for both phases are given below in the following two subsections and interested readers can refer to [11] for more details. We first define the following quantities used by both the forward and backward phase:

$$\Psi_n = \sum_{i=1}^{\text{Level}(n)} \kappa_i \quad (8)$$

$$F_n = \Psi_{\text{Level}(n)-1}\left[\Psi_{\text{Level}(n)}\right]^{-1}$$

where Ψ_n is the prior variance of node n and F is the prior covariance between the mean of node n and its parent.

Filtering: upward phase

We begin at node n at level $L-1$ on the tree. The children of this node are the shifted attributes of the items purchased by the user under category n, $\mathcal{I}_{u,n}$. We compute the probability of this node based on those children, $P(\mu^u_{x,n}|\mu^g_{x,n}, \mathcal{I}_{u,n}) = N(\phi^u_{x,n}, \sigma^u_{x,n})$, where ϕ, σ are given as follows:

$$\sigma^u_{x,n} = \frac{\Psi_n \kappa_L}{\kappa_L + |\mathcal{I}_{u,n}|\Psi_n} \quad (9)$$

$$\phi^u_{x,n} = \frac{\sigma^u_{x,n}}{\kappa_L} \sum_{I \in \mathcal{I}_{u,n}} I.x - \hat{\mu}^g_{x,n} \quad (10)$$

We repeat the above computations for all nodes n at Level $L-1$. We then move one level at a time upward computing $P(\mu^u_{x,n}|\mu^g_{x,n}, \mathcal{I}_{u,T(n)}) = N(\phi^u_{x,n}, \sigma^u_{x,n})$ for each node n at levels $L-2, L-3, \cdots, 0$ as follows, where $T(n)$ is the tree rooted at node n as defined in Section 2:

- For all nodes $m \in \mathcal{I}_n$. i.e. the children of node n under consideration, compute the following:

$$\phi^u_{x,n|m} = F_m \phi^u_{x,m} \quad (11)$$

$$\sigma^u_{x,n|m} = F_m^2 \sigma^u_{x,m} + F_m \kappa_{\text{level}(m)} \quad (12)$$

  where the above computes the filtering probability of node n conditioned on child m only (denoted by $n|m$ above).

- Now combine those estimates to find the filtering probability of node n based on all of its children as follows:

$$\sigma^u_{x,n} = \left[\Psi_n^{-1} + \sum_{m \in \mathcal{I}_n} [\sigma^u_{x,n|m}]^{-1} - {\Psi_n}^{-1}\right]^{-1} \quad (13)$$

$$\phi^u_{x,n} = \sigma^u_{x,n} \sum_{m \in \mathcal{I}_n} \phi^u_{x,n|m} \, [\sigma^u_{x,n|m}]^{-1} \quad (14)$$

Smoothing: downward phase

The recurrence in the upward phase ends at the root of the tree. and we now have the posterior probability of root itself from the filtering phase, that is $\hat{\mu}^u_{x,0} = \phi^u_{x,0}$ and $\hat{\sigma}^u_{x,0} = \sigma^u_{x,0}$, where we note that the posterior probability of each node n in the tree is distributed as $N(\hat{\mu}^u_{x,n}, \hat{\sigma}^u_{x,n})$. Now we propagate this information down the tree computing the posterior probability of each node n by combining the filtering probabilities from its children, with the smoothed probability from its parent as follows:

$$\hat{\mu}^u_{x,n} = \phi^u_{x,n} + \sigma^u_{x,n} F_n \left[\hat{\mu}^u_{x,\pi(n)} - \mu^u_{x,\pi(n)|n}\right]\left[\sigma^u_{x,\pi(n)|n}\right]^{-1} \quad (15)$$

$$\hat{\sigma}^u_{x,n} = \sigma^u_{x,n} + {\sigma^u_{x,n}}^2 {F_n}^2 \left[\hat{\sigma}^u_{x,\pi(n)} - \sigma^u_{x,\pi(n)|n}\right]\left[\sigma^u_{x,\pi(n)|n}\right]^{-2} \quad (16)$$

where the quantities $\pi(n)|n$ are those computed in the upward phase (Eq 11,12).

The above concludes the multi-scale Kalman Filtering Algorithm. It scales linearly with the number of nodes in the tree and as such it is efficient. Once we compute the smoothed posterior values for each user, we now repeat the same algorithm but over the global tree. The equation remains exactly the same, except that Eq 9 and 10 become:

$$\sigma^g_{x,n} = \frac{\Psi_n \kappa_L}{\kappa_L + |\mathcal{I}_{g,n}|\Psi_n} \quad (17)$$

$$\phi^g_{x,n} = \frac{\sigma^g_{x,n}}{\kappa_L} \sum_{u \in \mathcal{U}} \sum_{I \in \mathcal{I}_{u,n}} I.x - \hat{\mu}^u_{x,n} \quad (18)$$

which means that the observation to the global tree are shifted by each user's smoothed posterior mean. The rest of the steps of the upward and downward recurrences remains the same.

4.1.2 The Discrete Case

In this subsection we address the similar posterior problem addressed in the previous subsection but over discrete attributes. Our goal is to compute $P(\theta^g_{y,n}|\mathcal{I}_g)$ and $P(\theta^u_{y,n}|\mathcal{I}_g)$. We could use a similar alternating algorithm similar to the one described for the continuous case. At each step of this algorithm we run a Dirichlet-Multinomial algorithm called *Forward filtering, backward sampling* [20] and iterate it until convergence. However, this approach is expensive as it is slow to converge. Hence, we resort to an approximation algorithm due to [7] known as the maximum-path algorithm over trees. This algorithm proceeds in two phases as well: upward phases and downward phases. In the upward phase, we generate *pseudo* observations for internal nodes in the tree given the observations at their children. In the downward phase, we smooth each node's distribution with its parent. We give the algorithm below for the user specific distribution and then for the global distribution. Let $C^u_{y,n}$ be the counts of observed values of attribute y at node n. C is a vector whose components gives the observed counts for each possible value of y. For example, if y is the brand, then each component of C would contains the number of items purchases from this brand at node n. If n is a leaf category, then C is already observed.

- Upward phase: in which, we compute $C^u_{y,n}$ for all nodes n above the leaf category levels (at the leaf C is observed).

$$C^u_{y,n} = \sum_{m \in \mathcal{I}_n} C^u_{y,m} \quad (19)$$

  Which has the intuitive interpretation that $C^u_{y,n}$ is just the sum of the observed counts at the subtree rooted at node n.

- Downward phase: Once we reach the root in the upward phase, we compute the smoothed distribution of the root node as follows:

$$\hat{\theta}^u_{y,0} = \text{Normalize}(C^u_{y,0} + \gamma_0) \quad (20)$$

where Normalize just makes the input vector sums to one. We then proceed downward computing the smoothed distribution at all *internal* nodes as follows:

$$\hat{\theta}^u_{y,n} = \text{Normalize}(C^u_{y,n} + \gamma_{\text{Level}(n)}\hat{\theta}^u_{y,\pi(n)}) \quad (21)$$

The above algorithm computed the posterior over θ^u_y for each user and then we ran it again to compute the global distribution θ^g_y. The same equation above holds except that at the leaf categories $C^g_{y,n}$: for a leaf category n θ^g_y is computed using $\mathcal{I}_{g,n}$ which is the set of observed purchases from all users under category n. Finally, the posterior distribution used to predict how user u is likely to purchase an item under a leaf category n is computed using: $\text{Normalize}(C^u_{y,n} + \gamma_{\text{Level}(n)}\hat{\theta}^u_{y,\pi(n)} + \gamma_{\text{Level}(n)}\hat{\theta}^g_{y,n})$.

4.1.3 Summary

To summarize, given a set of continuous item attributes $\mathcal{X}$ and discrete item attributes $\mathcal{Y}$, our goal is to learn a user preference over each of them. This preference is learned as a distribution over those attributes. To combat sparsity, we parametric these distribution as an additive model of two parts: a global component which is shared across all the users, and a user-specific components that *shifts* the global component to personalize the distribution. We learn those two parts for each continuous and discrete attribute using upward-downward algorithms. In the case of continuous attributes, we use an iterative multi-scale Kalman algorithm until convergence. And in the discrete case we use a non-iterative two pass approximate algorithm. Each of these algorithms scales linearly with the number nodes of the tree. We noticed in practice that the iterative Kalman filtering algorithm converges in 5 to 10 iterations. While the output of these algorithm are useful by itself as we demonstrated in Figure 1, we use it as input to augment a latent-factor model to perform the end-end recommendations as we detail in the next subsection Finally, we treat the variances over the edges of the trees κ_i and γ_i as parameters and we estimate them using cross validation with regard to their effect of the end-end recommendation. To reduce the number of parameters we use the following parameterization :

$$\kappa_i = \frac{\kappa}{i} \qquad \gamma_i = \gamma * i \quad (22)$$

This parameterization implements our intuition that the variances of the parameters dimension as we go down the tree.

4.2 Learning Item Ranking Parameters: Γ

In this subsection, we describe how to find the Γ that maximizes $P(R|\Gamma, q)$. We decided to exploit recent advances in learning factorization models and rely on the so-called discriminative Bayesian personalized ranking (BPR) which has been shown to significantly outperform generative training [18].Based on Section 2, we seek to optimize the following log-likelihood function:

$$\begin{aligned} p(\Gamma|R_u, q) &= p(\Gamma)p(R_u|\Gamma, q) \quad (23)\\ &= p(\Gamma)\prod_{u\in U}\prod_{i\in\mathcal{I}_u}\prod_{j\notin\mathcal{I}_u} p(z_{ui} > z_{uj}|\Gamma, q) \end{aligned}$$

where, we approximate the non-smooth, non-differentiable expression $x_{ui} > x_{uj}$ using the logistic sigmoid function $\sigma(x_{ui} - x_{uj})$, where $\sigma(z) = \frac{1}{1+e^{-z}}$, which results in maximizing:

$$\sum_u \sum_{i\in\mathcal{I}_u} \sum_{j\notin\mathcal{I}_u} \ln\sigma(z_{ui} - z_{uj}) - \lambda||\Gamma||^2 \quad (24)$$

where, λ is the regularization constant which arise from a Gaussian prior over the parameters. and $||\Theta||^2$ is given by the following expression:

$$||\Gamma||^2 = \sum_u ||v_u||^2 + \sum_i ||v_u||^2 + \sum_{x\in\mathcal{X}} \alpha_x^2 + \sum_{y\in\mathcal{Y}} \beta_y^2$$

We use stochastic gradient descent to optimize the above function. At each iteration, we pick a triplet (u, i, j) where the user u bought item i and did not buy item j and define its associated loss $L(u, i, j)$ as follows:

$$\begin{aligned} L(u,i,j) &= \ln\sigma(z_{ui} - z_{uj}) - \lambda||\Gamma||^2\\ &= \ln\sigma\Big[\langle v_u, v_i - v_j\rangle + \sum_{x\in\mathcal{X}} \alpha_x\big(q^u_x(i.x) - q^u_x(j.x)\big)\\ &+ \sum_{y\in\mathcal{Y}} \beta_y\big(q^u_y(i.y) - q^u_y(j.y)\big)\Big] - \lambda||\Gamma||^2 \end{aligned}$$

Now, we need to compute the gradients of $L(u, i, j)$ with respect to the parameters $\Gamma = \{v_u, v_i, v_j, \alpha_x, \beta_y\}$ and perform a gradient step over those parameters. It is quite straightforward to show that this would give rise to the following update rules:

$$\begin{aligned} v_u &= v_u + \rho\Big(c_{uij}(v_i - v_j) - \lambda v_u\Big)\\ v_i &= v_i + \rho\Big(c_{uij}v_u - \lambda v_i\Big)\\ v_j &= v_j + \rho\Big(-c_{uij}v_u - \lambda v_j\Big)\\ \alpha_x &= \alpha_x + \rho\Big(c_{uij}(q^u_x(i.x) - q^u_x(j.x)) - \lambda\alpha_x\Big)\\ \beta_x &= \beta_x + \rho\Big(c_{uij}(q^u_y(i.y) - q^u_y(j.y)) - \lambda\beta_y\Big) \end{aligned} \quad (25)$$

Where ρ is the learning rate and $c_{uij} = 1 - \sigma(z_{ui} - z_{uj})$. We iterate sampling a tuple and updating the parameters until convergence (we used 100 epochs, where each epoch is a pass over the training data).

4.2.1 Hierarchical Extension: hLFUM

Since we have access to a taxonomy over items, we can also constraint the item factors to be smooth over the taxonomy using an additive model as follows. Recall that items appear at level L in the taxonomy. We define v_i as follows:

$$v_i = \sum_{l=0}^{L} w_{\pi^l(i)} \quad (26)$$

where $\pi^l(i)$ is the l^{th} parent of item i in the taxonomy. And w_n is a latent factor assigned to each internal node in the taxonomy. Thus for two sibling items i, i', they will share all the nodes starting from their parent up until the root, and as such only differ on $w_i, w_{i'}$ in the above summation and such $v_i, v_{i'}$ would become close to each other in the latent space. A similar additive model have been proposed before in [17]. The update rules in (25) for v_u, α_x, β_y remains the same using v_i defined in (26). A new update rule for w should be added and can be derived using the chain rule as follows:

$$\begin{aligned} w_{\pi^l(i)} &= w_{\pi^l(i)} + \rho\Big(c_{uij}v_u - \lambda w_{\pi^l(i)}\Big), l = 0, 1, \cdots L\\ w_{\pi^l(j)} &= w_{\pi^l(j)} + \rho\Big(-c_{uij}v_u - \lambda w_{\pi^l(j)}\Big), l = 0, 1, \cdots L \end{aligned}$$

which means that we update each weight on the path form the root to the item using the same gradient we used to update v_i in (25) before (this is easy to see from the chain rule over (26)).

5. EXPERIMENTAL SETUP

5.1 Dataset

To evaluate our proposed models, we used a log of user online transactions obtained from a major search engine, email provider and online shopping site and join it with a dataset of user conversion on display advertising where the ads correspond to products. The dataset contains information about the historical purchases of users over a period of 3 months and their response to display advertising Ads shown to them. We fully anonymize the users by dropping the original user identifier and assigning a new, sequential numbering of the records. For anonymity reasons, we report results over a sample of the above data. In the sample, we have about 40,000 users with average 2.3 purchases/ad response. We have 500,000 distinct individual products that are purchased/clicked upon in the dataset, which is mapped to a publicly available shopping taxonomy [1] which gives the price and the brand for each item as well. We use price as an example of a continuous attribute and brand as an example of a discrete attribute. The (resulting) taxonomy we use has about 1.5 million individual products in the leaf level organized into a taxonomy 3 levels deep, with around 1500 nodes at lowest level, 270 at the middle level and 23 top level categories. For each user, we select a transaction and mark all subsequent transactions/ad responses into the test dataset. All previous transactions/ ad responses are used for training. The last transaction/ad response in the training dataset is used for cross-validation and first transaction/ad response in the test dataset is used for prediction and reporting the error estimates.

5.2 Metrics

We use the AUC metric described below to compare our model with the baseline systems. AUC is a commonly used metric for testing the quality of rank orderings. Suppose the list of items to rank is X and our test transaction is B. Also suppose $r(x)$ is the numerical rank of the item x according to our model (from $1 \ldots n$). Then, the formula to compute AUC is given by:

$$\frac{1}{|B||X \setminus B|} \sum_{x \in B, y \in X \setminus B} \delta(r(x) < r(y))$$

While we could have used an alternate metric such as precision/recall at a given rank in the list, it requires selecting a suitable rank value. In contrast, AUC combines the prediction performance over all ranks into a single number and is therefore convenient to use.

6. EVALUATION

In this subsection, we study the performance of our proposed model LFUM and compare it with several baselines. We user the following system for comparison:

- LFUM is the model proposed in this paper with NO taxonomy over the item factors.
- hLFUM is the model proposed in this paper with a taxonomy over the item factors.
- LF is a latent factor model. i..e uses no item attributes
- hLF is latent factor model with a taxonomy over the item factors.
- UM: is a user preference model only. That is no latent factors are used. We just train the smoothed user preferences. (i.e. we set ϵ_{ui} to zero)
- HybridLF: this is the hybrid LF model described in Section 2.3.

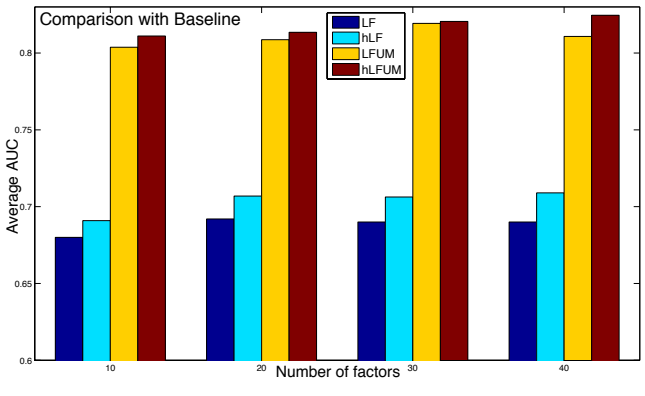

Figure 3: Comparing LFUM and hLFUM against several baselines along different number of factors

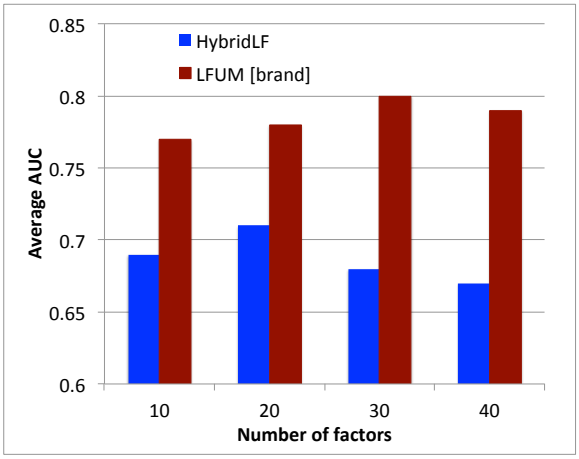

Figure 4: Comparing our approach LFUM with a fancier version of the standard hybridLF described in Section 2.3. The figure shows that our approach is more efficient in utilizing the extra information (brand in this case).

All models are trained using the BPR algorithm and all regularization parameters are tuned on a validation set.

6.1 Comparison to Baselines

in Figure 3 we compare LF, hLF, LFUM and hLFUM over different number of factors. As evident form the figure hLFUM and LFUM significantly outperformed models that ignore user preferences. Notice that LF does not use taxonomy in any way but hLF used taxonomy to constraint the item factors and combat sparsity. Nevertheless, hLF is till lagging even behind LFUM which does not impose a taxonomy constraint over the latent factors. It should be noted that both LFUM and hLFUM used taxonomy to smooth the user preferences, but only hLFUM used the taxonomy to smooth the item factors as well.

Second, we compare with the standard HybridLF model from Section 2.3. To make the comparison fair we only use the brand attribute to avoid introducing any bias in the result due to improper thresholding of the price attribute (note that our model does not need any such thresholding but we still use only the brand attribute in this experiment). The range of the brand attribute is 17K, thus we have $\mathcal{Y} = 17K$, since we need to add a binary variable for each brand. Furthermore, to make the HybridLF stronger, we stipulate that $W_y = W_y^u + W_y^0 + W_y^c$, i.e., we use an additive model to define W_y that combines user-specific weights with background

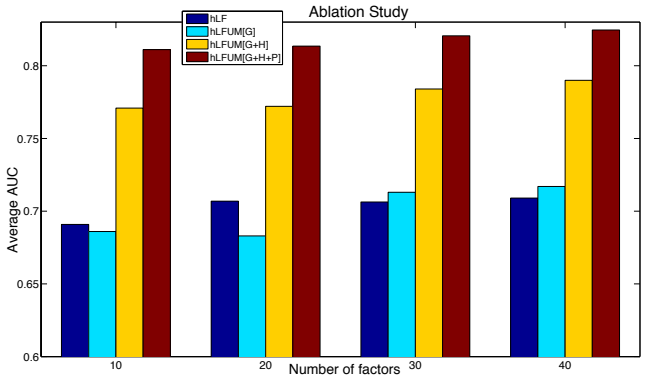

Figure 5: Studying the effect of the method used to learn user preferences on the final performance of our model. The figure shows that each part of our construction adds to the final result

weights W_y^0 and the item's category weights. We found that this form performed the best as opposed to using a global only weight vector, W_y^0, or a user-specific only weight vector, W_y^u. The results are shown in Figure 4. As evident, our model outperforms that baseline which confirms that our approach does not suffer from attributes with high cardinalities and that our model makes better use of the item attributes.

6.2 Ablation Study

In this subsection we study the contribution of each part of our model to understand the source of our improvement over the baselines. In Figure 5, we first study the smoothness effect over the user preference models. We parameterize LFUM and hLFUM with the the technique used to compute the user preferences over items as follows:

- G: means that we ignore the user specific distribution and the hierarchy and only compute the global preferences over each leaf category individually and use it for all users.
- G+H: same as G but we in addition we use the tree to smooth the computation of the global preferences over item attributes. However , no user personalization is used.
- G+H+P: is the full model described in Section 3 where the preferences are computed using an additive process smoothed over the taxonomy.

We compare these configurations against a hLF model by varying the number of factors. As obvious from this figure using only the global preference with no smoothing results in minimal improvement and sometime even degrade performance for small number of factors. As the number of factors increases, the model learned to user the extra factor to counter that detrimental effect. However, smoothing those global preference over the tree improves the performance. At first this might be puzzling, but this effect can be attribute to the wisdom of the crow effect. In other words, if the majority of the users prefer a given brand, then there is a chance that recommending this brand under a category that interest the user already (which can be inferred from the LF part of the model) would help the recommendation process. Finally adding user personalization significantly improved the result.

In Figure 6 we performed another ablation study to study if the improvement brought by LFUM are only due to the item preferences. In this case we compare the UM model against LFUM using 40 factors and by varying the technique used to smooth the preferences. As shown in this figure, while UM is still a strong baseline, LFUM outperforms it which shows the importance of using a latent

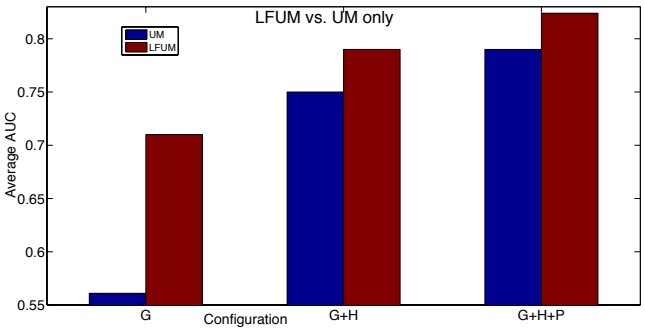

Figure 6: Comparing the performance of the full model against a model that only uses items' attributes

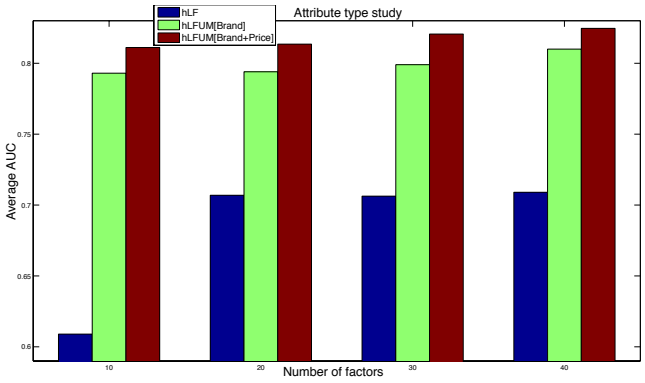

Figure 7: Studying the effect of different attribute types on the final result

factor model to account for the unobserved sources that draws the user to a given item.

6.3 Study of Attribute Type

In this subsection we study the effect of attribute type on the final result. We compare three models: hLF, hLFUM trained with brand preferences only and hLFUM model trained with both brand and price preferences. Both LFUM and hLFUM uses the default preference learning mechanisms (G+H+P). As can be seen from the result both attributes bring improvements, although we notices in general that brand preferences brings more improvements due to brand-loyalty: a user who like "Apple" product for instance, would still buy "Apple" product even if "Apple" increase the price of its products. Nevertheless, price by itself still provides a strong signal.

6.4 Effect of Item Frequency

We compare the performance of our approach hLFUM and with the baseline hLF as the frequency of items changes. As shown in Figure 8, our approach significantly outperforms the baseline especially when the item frequency is small (i.e. cold-start items). We omit the result of hybrid LF as it was worse than hLF as we showed in Figure 4

7. RELATED WORK

There is a wealth of literature about latent factor models for recommendation (see [15] for a survey). Conventional recommendation algorithms do not use user/item attributes, which can limit their performance when the feedback data is sparse. In recent work, several authors propose to augment the latent factor model with item

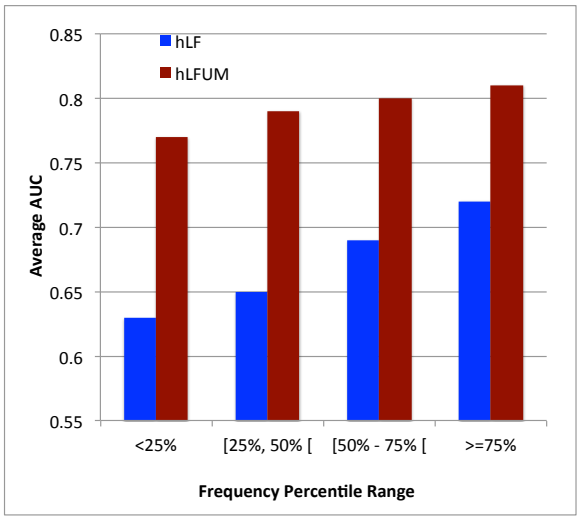

Figure 8: Illustrating the difference in performance between hLF and hLFUM as a function of item's popularity. Each cell in the x-axis represent items in a given frequency percentile range. Note that our approach consistently gives good results across different frequency ranges. Number of factors = 30.

and user attributes. This work was pioneered by Agarwal et al. [2]. However, the authors used very high-dimensional item and user attributes and use them to constraint the projection of users and items into the latent space. In contrast, in our work we only use few attributes (two in our study, brand and price) and therefore this approach is not applicable in our settings. Moreover our approach is unique as it first uses an additive model to preprocess the attributes before combining them with the latent factor model. This has the advantage of turning a discrete attribute with a large cardinality into a single informative feature. For instance, an alternative approach is to add a binary feature for each distinct brand, however that will produce a lot of noise and significantly slow down learning.

Our work is also related to hybrid recommender systems which combine two or more techniques, such as content and collaborative-filtering, to make recommendations [6]. We make several novel contributions in this work. We derive user preferences (e.g., brand, price) in terms of the global preference model and a per-user personalized model. The former captures the global trend and is very useful for sparse users, while the latter allows us to capture the individual characteristics for the dense users. In other words, for new users we leverage the global model, and as we get more and more data about a user, his/her preference model gets increasingly personalized. Moreover, we exploit the taxonomy while learning these preference models (in contrast to Yu et al. [21]). This allows us to deal with sparsity, e.g., preferences over different phones can be propagated to understand preferences over siblings in the hierarchy such as computers, printers, as well as parents such as electronics.

8. CONCLUSIONS

In this paper we presented a novel approach to integer content-based and latent factor based models. We learn a user preference model over item attributes that factors out common parts shared across users via an additive model, thus allowing user preferences to be represented in a discriminative fashion. Moreover, we showed how to integrate these learned preferences with a latent factor model and train the resulting system end-end to optimize a Bayesian discriminative ranking loss function. Our approach can handle continuous and discrete attributes and the complexity of learning and inference is weakly dependent no the cardinalities of the item attributes. We demonstrated the efficacy of our approach in a smart ad retrieval task with very promising results over several baselines. In the future we plan to incorporate the dynamics of user interests [4] and scale our inference to hundred of millions of users using techniques from [3]

9. REFERENCES

[1] Pricegrabber. http://www.pricegrabber.com/.

[2] D. Agarwal and B.-C. Chen. Regression-based latent factor models. In *KDD*, pages 19–28, 2009.

[3] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola. Scalable inference in latent variable models. In *WSDM*, pages 123–132, 2012.

[4] A. Ahmed, Y. Low, M. Aly, V. Josifovski, and A. J. Smola. Scalable distributed inference of dynamic user interests for behavioral targeting. In *KDD*, pages 114–122, 2011.

[5] J. Bennett, S. Lanning, and N. Netflix. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[6] R. D. Burke. Hybrid recommender systems: Survey and experiments. *User Model. User-Adapt. Interact.*, 12(4):331–370, 2002.

[7] P. Cowans. Probabilistic document modelling. 2006.

[8] A. Gunawardana and C. Meek. Tied boltzmann machines for cold start recommendations. In *RecSys*, pages 19–26, 2008.

[9] A. Gunawardana and C. Meek. A unified approach to building hybrid recommender systems. In *RecSys*, pages 117–124, 2009.

[10] P. D. Hoeff. Bilinear mixed effects models for dyadic data. *Journal of the American Statistical Assosciation*, 2005.

[11] A. W. K. Chou and A. Benveniste. Multiscale recursive estimation, data fusion, and regularization. *IEEE Transactions on Automatic Control*, 39(3), 1994.

[12] N. Koenigstein, G. Dror, and Y. Koren. Yahoo! music recommendations: modeling music ratings with temporal dynamics and item taxonomy. In *RecSys*, 2011.

[13] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, pages 426–434, 2008.

[14] Y. Koren. Collaborative filtering with temporal dynamics. In *KDD*, pages 447–456, 2009.

[15] Y. Koren and R. M. Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 145–186. 2011.

[16] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[17] A. Mnih. Taxonomy-informed latent factor models for implicit feedback. In *KDD Cup, KDD 2011*, 2011.

[18] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-thieme. L.s.: Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, 2009.

[19] S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM*, 2010.

[20] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. Hierarchical dirichlet processes. Dec. 2006.

[21] K. Yu, J. D. Lafferty, S. Zhu, and Y. Gong. Large-scale collaborative prediction using a nonparametric random effects model. In *ICML*, page 149, 2009.