Mach Learn (2016) 104:223–242
DOI 10.1007/s10994-016-5579-3

Factorizing LambdaMART for cold start recommendations

Phong Nguyen[1] · **Jun Wang**[1] · **Alexandros Kalousis**[2]

Received: 23 March 2016 / Accepted: 28 June 2016 / Published online: 21 July 2016

Abstract Recommendation systems often rely on point-wise loss metrics such as the mean squared error. However, in real recommendation settings only few items are presented to a user. This observation has recently encouraged the use of rank-based metrics. LambdaMART is the state-of-the-art algorithm in learning to rank which relies on such a metric. Motivated by the fact that very often the users' and items' descriptions as well as the preference behavior can be well summarized by a small number of hidden factors, we propose a novel algorithm, LambdaMART matrix factorization (LambdaMART-MF), that learns latent representations of users and items using gradient boosted trees. The algorithm factorizes LambdaMART by defining relevance scores as the inner product of the learned representations of the users and items. We regularise the learned latent representations so that they reflect the user and item manifolds as these are defined by their original feature based descriptors and the preference behavior. We also propose to use a weighted variant of NDCG to reduce the penalty for similar items with large rating discrepancy. We experiment on two very different recommendation datasets, meta-mining and movies-users, and evaluate the performance of LambdaMART-MF, with and without regularization, in the cold start setting as well as in the simpler matrix completion setting. The experiments show that the factorization of LambdaMart brings significant performance improvements both in the cold start and the matrix completion settings. The incorporation of regularisation seems to have a smaller performance impact.

Editors: Thomas Gärtner, Mirco Nanni, Andrea Passerini, and Celine Robardet.

✉ Alexandros Kalousis
Alexandros.Kalousis@hesge.ch

Phong Nguyen
phongnguyen@expedia.com

Jun Wang
jwang1@expedia.com

1 Expedia, Inc., 12, Rue du Lac, Geneva, Switzerland

2 Business Informatics Department, University of Applied Sciences, Western Switzerland, 7, Route de Drize, Carouge, Switzerland

Keywords Recommendation systems · Matrix factorization · LambdaMart · Regularization

1 Introduction

Most recommendation algorithms minimize a point-wise loss function such as the mean squared error or the mean average error between the predicted and the true user preferences. For instance in matrix factorization, a learning paradigm very popular in recommendation problems, state-of-the-art approaches such as Srebro et al. (2005), Abernethy et al. (2006) and Agarwal and Chen (2010) minimize the squared error between the inner product of the learned low-rank representations of users and items and the respective true preference scores. Such cost functions are clearly not appropriate for recommendation problems since what matters there is the rank order of the preference scores and not their absolute values, i.e. items that are very often top ranked should be highly recommended. It is only recently that recommendation methods have started using ranking-based loss functions in their optimization problems. Cofirank (Weimer et al. 2007), a collaborative ranking algorithm, does maximum-margin matrix factorization by optimizing an upper bound of the Normalized Discounted Cumulative Gain measure, NDCG, a ranking-based loss function (Järvelin and Kekäläinen 2000). However, like many recommendation algorithms, it cannot address the cold start problem, i.e. it cannot recommend new items to new users.

In preference learning (Fürnkranz and Hüllermeier 2010) we learn the preference order of documents for a given query. Preference learning algorithm are used extensively in search engines and IR systems and optimize ranking-based loss functions such as NDCG. Probably the best known example of such algorithms is LambdaMART (Burges 2010), the state-of-the-art learning to rank algorithm. Its success is due to its ability to model preference order using features describing side-information of the query-document pairs in a very flexible manner.

In this paper we develop a new recommendation algorithm with an emphasis on the cold start problem and the exploitation of user and item side-information. We take the view that users and items can be described by a small number of latent factors the inner product of which generates the preference scores. The new algorithm can be thought of as a variant of LambdaMART in which instead of directly learning the user-item preferences, as LambdaMART does, we learn latent factors that describe the users and the items and use them to compute the user-item preference scores. Essentially our algorithm does a low-rank matrix factorization of the preferences matrix. On the latent representations of the users and items we define data-based regularisers that reflect the user and item manifolds as these are established from their side-information and the preference matrix. We evaluate the performance of our algorithm on two very different recommendation applications under two scenarios: matrix completion and cold start recommendations. We compare its performance to a number of baselines, amongst which LambdaMART, and demonstrate significant performance improvements.

2 Preliminaries

We are given a (sparse) preference matrix, $\mathbf{Y}$, of size $n \times m$. The (i, j) non-missing entry of $\mathbf{Y}$ represents the preference score of the ith user for the jth item in recommendation problems or the relevance score of the jth document for the ith query in learning to rank problems;

the larger the value of the (i, j) entry the larger the relevance or preference is. In addition to the preference matrix, we also have the descriptions of the users and items. We denote by $\boldsymbol{c}_i = (c_{i1}, \ldots, c_{id})^{\mathrm{T}} \in \mathbb{R}^d$, the d-dimensional description of the ith user, and by $\mathbf{C}$ the $n \times d$ user description matrix, the ith row of which is given by the $\boldsymbol{c}_i^T$. Similarly, we denote by $\boldsymbol{d}_j = (d_{j1}, \ldots, d_{jl})^{\mathrm{T}} \in \mathbb{R}^l$, the l-dimensional description of the jth item and by $\mathbf{D}$ the $m \times l$ item description matrix, the jth row of which is given by the $\mathbf{d}_j^T$.

As already mentioned when recommending an item to a user we only care about the rank order of y_{ij} and not the actual preference score value y_{ij}. Thus in principle a preference learning algorithm does not need to predict the exact value of y_{ij} but only its rank order as this is induced by the preference vector $\boldsymbol{y}$. We will denote by $\boldsymbol{r}_{\mathbf{c}_i}$ the target rank vector of the ith user. We construct $\boldsymbol{r}_{\mathbf{c}_i}$ by ordering in a decreasing manner the ith user's non-missing preference scores over the items; its kth entry, $\boldsymbol{r}_{\mathbf{c}_i k}$, is the rank of the kth non-missing preference score, with the highest preference score having a rank of one. In the inverse problem, i.e. matching users to items, we will denote by $\boldsymbol{r}_{\boldsymbol{d}_j}$ the target rank vector of the jth item, given by ordering the jth item's non-missing relevance scores for the users.

2.1 Evaluation metric

In applications of preference learning, such as recommendation, information retrieval, etc, only a few top-ranked items are finally shown to the users. As a result appropriate evaluation measures for preference learning focus on the correctness of the top-ranked items. One such, very often used, metric is the *Discounted Cumulative Gain* (DCG) (Järvelin and Kekäläinen 2000), which is defined as follows:

$$\mathrm{DCG}(\boldsymbol{r}, \boldsymbol{y})@k = \sum_{i=1}^{m} \frac{2^{y_i} - 1}{\log_2(r_i + 1)} I(r_i \leq k) \tag{1}$$

where k is the truncation level at which DCG is computed and I is the indicator function which returns 1 if its argument holds otherwise 0. $\boldsymbol{y}$ is the m dimensional ground truth relevance vector and $\boldsymbol{r}$ is a rank vector that we will learn. The DCG score measures the match between the given rank vector $\boldsymbol{r}$ and the rank vector of the relevance score vector $\boldsymbol{y}$. It is easy to check that if the rank vector $\boldsymbol{r}$ correctly preserves the order induced by $\boldsymbol{y}$ then the DCG score will achieve its maximum. Due to the log in the denominator the DCG score will incur larger penalties when misplacing top items compared to low end items, emphasizing like that the correctness of top items.

Since the DCG score also depends on the length of the relevance vector $\boldsymbol{y}$, it is often normalized with respect to its maximum score, resulting to what is known as the Normalized DCG (NDCG), defined as:

$$\mathrm{NDCG}(\boldsymbol{y}, \hat{\boldsymbol{y}})@k = \frac{\mathrm{DCG}(r(\hat{\boldsymbol{y}}), \boldsymbol{y})@k}{\mathrm{DCG}(r(\boldsymbol{y}), \boldsymbol{y})@k} \tag{2}$$

$r(\cdot)$ is a rank function that outputs the rank vector, in decreasing order, of its input argument vector. Thus the vector $r(\boldsymbol{y})$ is the rank vector of the ground truth relevance vector $\boldsymbol{y}$ and $r(\hat{\boldsymbol{y}})$ is the rank vector of the predicted relevance vector $\hat{\boldsymbol{y}}$ provided by the learned model. With normalization, the value of NDCG ranges from 0 to 1, the larger the better. In this paper, we will also use this metric as our evaluation metric.

2.2 LambdaMART

The main difficulty in learning preferences is that rank functions are not continuous and have combinatorial complexity. Thus most often instead of the rank of the preference scores the pairwise order constraints over the items' preferences are used. LambdaMART is one of the most popular algorithms for preference learning which follows exactly this idea (Burges 2010). Its optimization problem relies on a distance distribution measure, cross entropy, between a learned distribution[1] that gives the probability that item j is more relevant than item k from the true distribution which has a probability mass of one if item k is really more relevant than item j and zero otherwise. The final loss function of LambdaMart defined over all users i and overall the respective pairwise preferences for items j, k, is given by:

$$\mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}}) = \sum_{i=1}^{n} \sum_{\{jk\} \in Z} |\varDelta \mathrm{NDCG}^i_{jk}| \log(1 + e^{-\sigma(\hat{y}_{ij} - \hat{y}_{ik})}) \tag{3}$$

where Z is the set of all possible pairwise preference constraints such that in the ground truth relevance vector holds $y_{ij} > y_{ik}$, and $\varDelta\mathrm{NDCG}^i_{jk}$ is given by:

$$\varDelta\mathrm{NDCG}^i_{jk} = \mathrm{NDCG}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i) - \mathrm{NDCG}(\boldsymbol{y}_i^{jk}, \hat{\boldsymbol{y}}_i)$$

where $\boldsymbol{y}_i^{jk}$ is the same as the ground truth relevance vector $\boldsymbol{y}_i$ except that the values of y_{ij} and y_{ik} are swapped. Note that this equal to the NDCG difference that we get if we swap the $\hat{y}_{ij}$, $\hat{y}_{ik}$, estimates. Thus the overall loss function of LambdaMART Eq. 3 is the sum of the logistic losses on all pairwise preference constraints weighted by the respective NDCG differences. Since the NDCG measure penalizes heavily the error on the top items, the loss function of LambdaMART has also the same property. LambdaMART minimizes its loss function with respect to all $\hat{y}_{ij}$, $\hat{y}_{ik}$, and its optimization problem is:

$$\min_{\hat{\mathbf{Y}}} \mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}}) \tag{4}$$

Yue and Burges (2007) have shown empiricially that solving this problem also optimizes the NDCG metric of the learned model. The partial derivative of LambdaMART's loss function with respect to the estimated scores $\hat{y}_{ij}$ is

$$\frac{\partial \mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}})}{\partial \hat{y}_{ij}} = \lambda^i_j = \sum_{\{k|jk\} \in Z} \lambda^i_{jk} - \sum_{\{k|kj\} \in Z} \lambda^i_{kj} \tag{5}$$

and λ^i_{jk} is given by:

$$\lambda^i_{jk} = \frac{-\sigma}{1 + e^{\sigma(\hat{y}_{ij} - \hat{y}_{ik})}} |\varDelta\mathrm{NDCG}^i_{jk}| \tag{6}$$

With a slight abuse of notation below we will write $\frac{\partial \mathcal{L}(y_{ij}, \hat{y}_{ij})}{\partial \hat{y}_{ij}}$ instead of $\frac{\partial \mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}})}{\partial \hat{y}_{ij}}$, to make explicit the dependence of the partial derivative only on y_{ij}, $\hat{y}_{ij}$ due to the linearity of $\mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}})$.

LambdaMART uses Multiple Additive Regression Trees (MART) (Friedman 2001) to solve its optimization problem. It does so through a gradient descent in the functional space

[1] This learned distribution is generated by the sigmoid function $P^i_{jk} = \frac{1}{1+e^{\sigma(\hat{y}_{ij} - \hat{y}_{ik})}}$ of the estimated preferences $\hat{y}_{ij}$, $\hat{y}_{ik}$.

that generates preference scores from item and user descriptions, i.e. $\hat{y}_{ij} = f(\boldsymbol{c}_i, \boldsymbol{d}_j)$, where the update of the preference scores at the t step of the gradient descent is given by:

$$\hat{y}_{ij}^{(t)} = \hat{y}_{ij}^{(t-1)} - \eta \frac{\partial \mathcal{L}(y_{ij}, \hat{y}_{ij}^{(t-1)})}{\partial \hat{y}_{ij}^{(t-1)}} \tag{7}$$

or equivalently:

$$f^{(t)}(\boldsymbol{c}_i, \boldsymbol{d}_j) = f^{(t-1)}(\boldsymbol{c}_i, \boldsymbol{d}_j) - \eta \frac{\partial \mathcal{L}(y_{ij}, f^{(t-1)}(\boldsymbol{c}_i, \boldsymbol{d}_j))}{\partial f^{(t-1)}(\boldsymbol{c}_i, \boldsymbol{d}_j)} \tag{8}$$

where η is the learning rate. We terminate the gradient descent when we reach a given number of iterations T or when the validation loss NDCG starts to increase. We approximate the derivative $\frac{\partial \mathcal{L}(y_{ij}, \hat{y}_{ij}^{(t-1)})}{\partial \hat{y}_{ij}^{(t-1)}}$ by learning at each step t a regression tree $h^{(t)}(\boldsymbol{c}, \boldsymbol{d})$ that fits it by minimizing the sum of squared errors.

Thus at each update step we have

$$f^{(t)}(\boldsymbol{c}_i, \boldsymbol{d}_j) = f^{(t-1)}(\boldsymbol{c}_i, \boldsymbol{d}_j) + \eta h^{(t)}(\boldsymbol{c}_i, \boldsymbol{d}_j) \tag{9}$$

which if we denote by γ_{tk} the prediction of the kth terminal node of the $h^{(t)}$ tree and by h_{tk} the respective partition of the input space, we can rewrite as:

$$f^{(t)}(\boldsymbol{c}_i, \boldsymbol{d}_j) = f^{(t-1)}(\boldsymbol{c}_i, \boldsymbol{d}_j) + \eta \gamma_{tk} I((\boldsymbol{c}, \boldsymbol{d}) \in h_{tk}) \tag{10}$$

we can further optimize over the γ_{tk} values to minimize the loss function of Eq. 3 over the instances of each h_{tk} partition using Newton's approximation. The final preference estimation function is given by:

$$\hat{y} = f(\boldsymbol{c}, \boldsymbol{d}) = \sum_{t=1}^{T} \eta h^{(t)}(\boldsymbol{c}, \boldsymbol{d}) \tag{11}$$

or

$$\hat{y} = f(\boldsymbol{c}, \boldsymbol{d}) = \sum_{t=1}^{T} \eta \gamma_{tk} I((\boldsymbol{c}, \boldsymbol{d}) \in h_{tk}) \tag{12}$$

LambdaMart is a very effective algorithm for learning to rank problems, see e.g. Burges et al. (2011) and Donmez et al. (2009). It learns non-linear relevance scores, $\hat{y}_{ij}$, using gradient boosted regression trees. The number of the parameters it fits is given by the number of available preference scores (this is typically some fraction of $n \times m$); there is no regularization on them to prevent overfitting. The only protection against overfitting can come from rather empirical approaches such as constraining the size of the regression trees or by selecting learning rate η.

3 Factorized Lambda-MART

Motivated by the fact that very often the users' and the items' descriptions and their preference relations can be well summarized by a small number of hidden factors we learn a low-dimensional hidden representation of users and items using gradient boosted trees, MART. We define the relevance score as the inner product of the new representation of the users and items; this has the additional advantage of introducing a low rank regularization in the learned preference matrix.

Concretely, we define the relevance score of the ith user and jth item by $\hat{y}_{ij} = \boldsymbol{u}_i^{\mathrm{T}} \boldsymbol{v}_j$, where $\mathbf{u}_i$ and $\mathbf{v}_j$ are the r-dimensional user and item latent descriptors. We denote by $\mathbf{U} : n \times r$ and $\mathbf{V} : m \times r$ the new representation matrices of users and items. The dimensionality of r is a small number, $r << \min(n, m)$. The loss function of Eq. 3 now becomes:

$$\mathcal{L}_{MF}(\mathbf{Y}, \hat{\mathbf{U}}, \hat{\mathbf{V}}) = \sum_{i=1}^{n} \sum_{\{jk\} \in Z} |\Delta \mathrm{NDCG}_{jk}^{i}| \log(1 + e^{-\sigma(\hat{\boldsymbol{u}}_i(\hat{\boldsymbol{v}}_j - \hat{\boldsymbol{v}}_k))}) \tag{13}$$

The partial derivatives of this loss function with respect to $\hat{\boldsymbol{u}}_i$, $\hat{\boldsymbol{v}}_j$, are given by:

$$\frac{\partial \mathcal{L}_{MF}(\mathbf{Y}, \hat{\mathbf{U}}, \hat{\mathbf{V}})}{\partial \hat{\boldsymbol{u}}_i} = \sum_{j=1}^{m} \frac{\partial \mathcal{L}_{MF}(\mathbf{Y}, \hat{\mathbf{Y}})}{\partial \hat{y}_{ij}} \frac{\partial \hat{y}_{ij}}{\partial \hat{\boldsymbol{u}}_i} = \sum_{j=1}^{m} \lambda_j^i \frac{\partial \hat{y}_{ij}}{\partial \hat{\boldsymbol{u}}_i} \tag{14}$$

$$\frac{\partial \mathcal{L}_{MF}(\mathbf{Y}, \hat{\mathbf{U}}, \hat{\mathbf{V}})}{\partial \hat{\boldsymbol{v}}_j} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}_{MF}(\mathbf{Y}, \hat{\mathbf{Y}})}{\partial \hat{y}_{ij}} \frac{\partial \hat{y}_{ij}}{\partial \hat{\boldsymbol{v}}_i} = \sum_{i=1}^{n} \lambda_j^i \frac{\partial \hat{y}_{ij}}{\partial \boldsymbol{v}_j} \tag{15}$$

Note that the formulation we give in Eq. 13 is very similar to those used in matrix factorization algorithms. Existing matrix factorization algorithms used in collaborative filtering recommendation learn the low-rank representation of users and items in order to complete the sparse preference matrix $\mathbf{Y}$ (Srebro et al. 2005; Weimer et al. 2007; Abernethy et al. 2006; Agarwal and Chen 2010), however these approaches cannot address the cold start problem.

Similar to LambdaMART we will seek a function f that will optimize the $\mathcal{L}_{MF}$ loss function. To do so we will learn functions of the latent profiles of the users and items from their side information. We will factorize $f(\mathbf{c}, \mathbf{d})$ by

$$f(\mathbf{c}, \mathbf{d}) = f_u(\mathbf{c})^{\mathrm{T}} f_v(\mathbf{d}) = \mathbf{u}^{\mathrm{T}} \mathbf{v} = \hat{y}$$

where $f_u : \mathcal{C} \to \mathcal{U}$ is a learned user function that gives us the latent factor representation of user from his/her side information descriptor $\mathbf{c}$; $f_v : \mathcal{D} \to \mathcal{V}$ is the respective function for the items. We will follow the LambdaMART approach described previously and learn an ensemble of trees for each one of the f_u and f_v functions. Concretely:

$$\hat{\mathbf{u}}_i = f_u(\mathbf{c}_i) = \sum_{t=1}^{T} \eta h_u^{(t)}(\mathbf{c}_i) \tag{16}$$

$$\hat{\mathbf{v}}_j = f_v(\mathbf{d}_j) = \sum_{t=1}^{T} \eta h_v^{(t)}(\mathbf{d}_j) \tag{17}$$

Unlike standard LambdaMART the trees we will learn are multi-output regression trees, predicting the complete latent profile of users or items. Now at each step t of the gradient descent we learn the $h_u^{(t)}(\mathbf{c})$ and $h_v^{(t)}(\mathbf{d})$ trees that fit the negative of the partial derivatives given at Eqs. 14, 15. We learn the trees by greedily optimizing the sum of squared errors over the over the dimensions of the partial gradients. The t gradient descent step for $\boldsymbol{u}_i$ is given by:

$$\boldsymbol{u}_i^{(t)} = \mathbf{u}_i^{(t-1)} - \sum_{j=1}^{m} \frac{\partial \mathcal{L}_{MF}(\mathbf{Y}, \hat{\mathbf{Y}})}{\partial \hat{y}_{ij}} \Big|_{\hat{y}_{ij} = \hat{\boldsymbol{u}}_i^{(t-1)\mathrm{T}} \hat{\boldsymbol{v}}_j^{(t-1)}} \frac{\partial \hat{\boldsymbol{u}}_i^{\mathrm{T}} \hat{\boldsymbol{v}}_j^{(t-1)}}{\partial \hat{\boldsymbol{u}}_i}$$

and for $\boldsymbol{v}_j$ by:

$$\boldsymbol{v}_j^{(t)} = \boldsymbol{v}_j^{(t-1)} - \sum_{i=1}^{n} \frac{\partial \mathcal{L}_{MF}(\mathbf{Y}, \hat{\mathbf{Y}})}{\partial \hat{y}_{ij}}\bigg|_{\hat{y}_{ij}=\hat{\boldsymbol{u}}_i^{(t-1)\mathrm{T}} \hat{\boldsymbol{v}}_j^{(t-1)}} \frac{\partial \hat{\boldsymbol{u}}_i^{(t-1)\mathrm{T}} \hat{\boldsymbol{v}}_j}{\partial \hat{\boldsymbol{v}}_j}$$

The functions of each step are now:

$$f_{\boldsymbol{u}}^{(t)}(\boldsymbol{c}) = f_u^{(t-1)}(\boldsymbol{c}) + \eta h_u^{(t)}(\boldsymbol{c}) \tag{18}$$

$$f_{\boldsymbol{v}}^{(t)}(\boldsymbol{d}) = f_v^{(t-1)}(\boldsymbol{d}) + \eta h_v^{(t)}(\boldsymbol{d}) \tag{19}$$

which give rise to the final form of the functional estimates that we already gave in Eqs. 16 and 17 respectively. Optimizing for both $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ at each step of the gradient descent results to a faster convergence, than first optimizing for one while keeping the other fixed and vice versa. We will call the resulting algorithm LambdaMART Matrix Factorization and denote it by LM-MF.

4 Regularization

In this section, we will describe a number of different regularization methods to constrain in a meaningful manner the learning of the user and item latent profiles. We will incorporate different regularizers in the gradient boosting tree algorithm to avoid overfitting during the learning of the f_u and f_v functions. We will do so in a manner that will reflect the original user and item manifolds as well as their relationships as these are described in the preference matrix.

4.1 Input–output space regularization

LambdaMART-MF learns a new representation of users and items. We will regularize these representations by constraining them by the geometry of the user and item spaces as this is given by the **c** and **d** descriptors respectively. Based on these descriptors we will define user similarities and item similarities, which we will call input-space similarities in order to make explicit that they are computed on the side-information vectors describing the users and the items. In addition to the input-space similarities we will also define what we will call output space similarity which will reflect the similarities of users(items) according to the respective similarities of their preference vectors. We will regularize the learned latent representations by the input/output space similarities constraining the former to follow the latter.

To define the input space similarities we will use the descriptors of users and items. Concretely given two users $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$ we measure their input space similarity $s_{U_{in}}(\boldsymbol{c}_i, \boldsymbol{c}_j)$ using the heat kernel over their descriptors as follows:

$$s_{U_{in}}(\boldsymbol{c}_i, \boldsymbol{c}_j; \sigma) = e^{-\sigma ||\boldsymbol{c}_i - \boldsymbol{c}_j||^2} \tag{20}$$

where σ is the inverse kernel width of the heat kernel; we set its value to the squared inverse average Euclidean distance of all the users in the $\mathcal{C}$ space, i.e. $\sigma = ((\frac{1}{n} \sum_{ij} ||\boldsymbol{c}_i - \boldsymbol{c}_j||)^2)^{-1}$. By applying the above equation over all user pairs, we get the $\mathbf{S}_{U_{in}} : n \times n$ user input similarity matrix. We will do exactly the same to compute the $\mathbf{S}_{V_{in}} : m \times m$ item input similarity matrix using the item descriptors $\mathbf{D}$.

To define the output space similarities we will use the preference vectors of users and items. Given two users i and j and their preference vectors $\boldsymbol{y}_{i\cdot}$ and $\boldsymbol{y}_{j\cdot}$, we will measure their output space similarity $s_{U_{out}}(\mathbf{y}_{i\cdot}, \mathbf{y}_{j\cdot})$ using NDCG@k since this is the metric that we want to optimize. To define the similarities, we first compute the NDCG@k on the $(\boldsymbol{y}_{i\cdot}, \boldsymbol{y}_{j\cdot})$ pair as well as on the $(\boldsymbol{y}_{j\cdot}, \boldsymbol{y}_{i\cdot})$, because the NDCG@k is not symmetric, and compute the distance $d_{\text{NDCG@}k}(\mathbf{y}_{i\cdot}, \mathbf{y}_{j\cdot})$ between the two preference vectors as the average of the two previous NDCG@k measures which we have subtracted from one:

$$d_{\text{NDCG@}k}(\mathbf{y}_{i\cdot}, \mathbf{y}_{j\cdot}) = \frac{1}{2}((1 - \text{NDCG@}k(\mathbf{y}_{i\cdot}, \mathbf{y}_{j\cdot})) + (1 - \text{NDCG@}k(\mathbf{y}_{j\cdot}, \mathbf{y}_{i\cdot})))$$

We define finally the output space similarity $s_{U_{out}}(\mathbf{y}_{i\cdot}, \mathbf{y}_{j\cdot})$ by the exponential of the negative distance:

$$s_{U_{out}}(\mathbf{y}_{i\cdot}, \mathbf{y}_{j\cdot}) = e^{-d_{\text{NDCG@}k}(\mathbf{y}_{i\cdot}, \mathbf{y}_{j\cdot})} \tag{21}$$

The resulting similarity measure gives high similarity to preference vectors that are very similar in their top-*k* elements, while preference vectors which are less similar in their top-*k* elements will get much lower similarities. We apply this measure over all user preference vector pairs to get the $\mathbf{S}_{U_{out}} : n \times n$ user output similarity matrix. We do the same for items using now the $\mathbf{y}_{\cdot i}$ and $\mathbf{y}_{\cdot j}$ preference vectors for each ijth item pair to get the $\mathbf{S}_{V_{out}} : m \times m$ item output similarity matrix.

To regularize the user and item latent profiles, we will use graph laplacian regularization and force them to reflect the manifold structure of the users and items as these are given by the input and outpout space similarity matrices. Concretely, we define the user and item regularizers $\mathcal{R}_U$ and $\mathcal{R}_V$ as follows:

$$\mathcal{R}_U = \mu_1 ||\hat{\mathbf{U}}^{\mathrm{T}} \mathbf{L}_{U_{in}} \hat{\mathbf{U}}||_F^2 + \mu_2 ||\hat{\mathbf{U}}^{\mathrm{T}} \mathbf{L}_{U_{out}} \hat{\mathbf{U}}||_F^2 \tag{22}$$

$$\mathcal{R}_V = \mu_1 ||\hat{\mathbf{V}}^{\mathrm{T}} \mathbf{L}_{V_{in}} \hat{\mathbf{V}}||_F^2 + \mu_2 ||\hat{\mathbf{V}}^{\mathrm{T}} \mathbf{L}_{V_{out}} \hat{\mathbf{V}}||_F^2 \tag{23}$$

where the four laplacian matrices $\mathbf{L}_{U_{in}}, \mathbf{L}_{U_{out}}, \mathbf{L}_{V_{in}}$ and $\mathbf{L}_{V_{out}}$ are defined as $\mathbf{L} = \mathbf{D} - \mathbf{S}$ where $\mathbf{D}_{ii} = \sum_j \mathbf{S}_{ij}$ and $\mathbf{S}$ are the corresponding similarity matrices. μ_1 and μ_2 are regularization parameters that control the relative importance of the input and output space similarities respectively.

4.2 Weighted NDCG cost

In addition to the graph laplacian regularization over the latent profiles, we also provide a soft variant of the NDCG loss used in LambdaMART. Recall that in NDCG the loss is determined by the pairwise difference incurred if we exchange the position of two items j and k for a given user i. This loss can be unreasonably large even if the two items are similar to each other with respect to the similarity measures defined above. A consequence of such large penalties will be a large deviance of the gradient boosted trees under which similar items will not anymore fall in the same leaf node. To alleviate that problem we introduce a weighted NDCG difference which takes into account the items' input and output similarities, which we define as follows:

$$\mathbf{S}_V = \mu_1 \mathbf{S}_{V_{in}} + \mu_2 \mathbf{S}_{V_{out}} \tag{24}$$

$$\Delta \text{WNDCG}^i_{jk} = \Delta \text{NDCG}^i_{jk}(1 - s_{V_{jk}}) \tag{25}$$

Under the weighted variant if two items j and k are very similar the incurred loss will be by construction very low leading to a smaller loss and thus less deviance of the gradient boosted trees for the two items.

4.3 Regularized LambdaMART-MF

By combing the input–output space regularization and the weighted NDCG with LM-MF given in Eq. 13 we obtain the regularised LambdaMART matrix factorization the objective function of which is

$$\mathcal{L}_{RMF}(\mathbf{Y}, \hat{\mathbf{U}}, \hat{\mathbf{V}}) = \sum_{i=1}^{n} \sum_{\{jk\} \in Z} |\Delta \text{WNDCG}^i_{jk}| \log(1 + e^{-\sigma(\hat{\boldsymbol{u}}_i(\hat{\boldsymbol{v}}_j - \hat{\boldsymbol{v}}_k))}) + \mathcal{R}_U + \mathcal{R}_V \quad (26)$$

Its partial derivatives with respect to $\hat{\boldsymbol{u}}_i$, $\hat{\boldsymbol{v}}_j$, are now given by:

$$\begin{aligned}\frac{\partial \mathcal{L}_{RMF}(\mathbf{Y}, \hat{\mathbf{U}}, \hat{\mathbf{V}})}{\partial \hat{\boldsymbol{u}}_i} &= \sum_{j=1}^{m} \lambda^i_j \frac{\partial \hat{y}_{ij}}{\partial \hat{\boldsymbol{u}}_i} + 2\mu_1 \sum_{j \in \mathbb{N}^i_{U_{in}}} s_{U_{in},ij}(\hat{\boldsymbol{u}}_i - \hat{\boldsymbol{u}}_j) \\ &+ 2\mu_2 \sum_{j \in \mathbb{N}^i_{U_{out}}} s_{U_{out},ij}(\hat{\boldsymbol{u}}_i - \hat{\boldsymbol{u}}_j) \quad (27)\\ \frac{\partial \mathcal{L}_{RMF}(\mathbf{Y}, \hat{\mathbf{U}}, \hat{\mathbf{V}})}{\partial \hat{\boldsymbol{v}}_j} &= \sum_{i=1}^{n} \lambda^i_j \frac{\partial \hat{y}_{ij}}{\partial \boldsymbol{v}_j} + 2\mu_1 \sum_{i \in \mathbb{N}^j_{V_{in}}} s_{V_{in},ij}(\hat{\boldsymbol{v}}_j - \hat{\boldsymbol{v}}_i) \\ &+ 2\mu_2 \sum_{i \in \mathbb{N}^j_{V_{out}}} s_{V_{out},ij}(\hat{\boldsymbol{v}}_j - \hat{\boldsymbol{v}}_i) \quad (28)\end{aligned}$$

where $\mathbb{N}^i_{U_{in}}$ is the set of the k nearest neighbors of the ith user defined on the basis of the input similarity.

To optimize our final objective function, Eq. 26, we learn the latent profiles of users and items by gradient boosted trees. We will call the resulting algorithm Regularized LambdaMART Matrix Factorization and denote it by LM-MF-Reg. The algorithm is described in Algorithm 1. At iteration t, we first compute the partial derivatives of the objective function at point $(\hat{\mathbf{U}}^{t-1}, \hat{\mathbf{V}}^{t-1})$. Then we fit the trees $h^t_{\boldsymbol{u}}$ and $h^t_{\boldsymbol{v}}$ for the user and item descriptions respectively. Finally, we update the predictions of $\hat{\mathbf{U}}$ and $\hat{\mathbf{V}}$ according to the output of regression trees. The learning process is continued until the maximum number of trees is reached or the early stop criterion is satisfied. In all our experiments, the maximum number of trees is set by 15000. The early stopping criterion is no validation set error improvement in 200 iterations.

5 Experiments

We will evaluate the two basic algorithms that we presented above, LM-MF and LM-MF-Reg, on two recommendation problems, meta-mining and MovieLens, and compare their performance to a number of baselines.

Meta-mining (Hilario et al. 2011) applies the idea of meta-learning or learning to learn to the whole DM process. Data mining workflows and datasets are extensively characterised by side information. The goal is to suggest which data mining workflow should be applied on which dataset in view of optimizing some performance measure, e.g. accuracy in classification problems, by mining past experiments. Recently Nguyen et al. (2012a) have proposed tackling the problem as a hybrid recommendation problem: dataset-workflow pairs can be seen as user-

Algorithm 1 Regularized LambdaMART Matrix Factorization

```
Input: $\mathbf{C}$, $\mathbf{D}$, $\mathbf{Y}$, $\mathbf{S}_{\mathbf{U}_{\mathbf{out}}}$, $\mathbf{S}_{\mathbf{U}_{\mathbf{in}}}$, $\mathbf{S}_{\mathbf{V}_{\mathbf{out}}}$, $\mathbf{S}_{\mathbf{V}_{\mathbf{in}}}$, $\mu_1$, $\mu_2$, $\eta$, $r$, and $T$
Output: $f_{\boldsymbol{u}}$ and $f_{\boldsymbol{v}}$
initialize: $f_{\boldsymbol{u}}^0(\boldsymbol{c})$ and $f_{\boldsymbol{v}}^0(\boldsymbol{d})$ with random values
initialize $t = 1$
repeat
  a) compute $\boldsymbol{r}_{\boldsymbol{u}_i}^t = -\frac{\partial \mathcal{L}_{RMF}(\mathbf{Y},\hat{\mathbf{U}},\hat{\mathbf{V}})}{\partial \hat{\boldsymbol{u}}_i}\Big|_{\hat{\mathbf{U}}=\hat{\mathbf{U}}^{t-1},\hat{\mathbf{V}}=\hat{\mathbf{V}}^{t-1}}$ according to equation 27, for $i = 1$ to $n$
  b) compute $\boldsymbol{r}_{\boldsymbol{v}_j}^t = -\frac{\partial \mathcal{L}_{RMF}(\mathbf{Y},\hat{\mathbf{U}},\hat{\mathbf{V}})}{\partial \hat{\boldsymbol{v}}_j}\Big|_{\hat{\mathbf{U}}=\hat{\mathbf{U}}^{t-1},\hat{\mathbf{V}}=\hat{\mathbf{V}}^{t-1}}$ according to equation 28, for $j = 1$ to $m$
  c)fit a multi-output regression tree $h_{\boldsymbol{u}}^t$ for $\{(\boldsymbol{c}_i, \boldsymbol{r}_{\boldsymbol{u}_i}^t)\}_{i=1}^n$
  d)fit a multi-output regression tree $h_{\boldsymbol{v}}^t$ for $\{(\boldsymbol{d}_i, \boldsymbol{r}_{\boldsymbol{v}_i}^t)\}_{i=1}^m$
  e) $f_{\boldsymbol{u}}^t(\boldsymbol{c}_i) = f_{\boldsymbol{u}}^{t-1}(\boldsymbol{c}_i) + \eta h_{\boldsymbol{u}}^t(\boldsymbol{c}_i)$
  f) $f_{\boldsymbol{v}}^t(\boldsymbol{d}_j) = f_{\boldsymbol{v}}^{t-1}(\boldsymbol{d}_j) + \eta h_{\boldsymbol{v}}^t(\boldsymbol{d}_j)$
until converges or t=T
```

Table 1 Dataset statistics

	Users	Items	Ratings	% comp. **Y**	*d*	*l*
Meta-mining	65	35	2275	100.0	113	214
MovieLens 100 K	943	1682	100K	6.3	4	19
MovieLens 1 M	6040	3900	1M	4.2	4	19

item pairs which are related by the relative performance achieved by the workflows applied on the datasets. Here we go one step further (Nguyen et al. 2012a) and treat meta-mining as a learning to rank problem.

MovieLens[2] is a well-known benchmark dataset for collaborative filtering. It provides one to five star ratings of users for movies, where each user has rated at least twenty movies. The dataset provides also limited side information on users and items. It has been most often used in non cold start settings, due to the difficulty in exploiting the side information. It comes in two versions: the first contains one hundred thousand ratings (100 K) and the second one million (1M). We experiment with both of them.

In Table 1 we give a basic description of the three different datasets, namely the numbers of ratings, users, items, the numbers of user and item descriptors, d and l respectively, and the percentage of available entries in the **Y** preference matrix. The meta-mining dataset has a quite extensive set of features giving the side information and a complete preference matrix, it is also considerably smaller than the two MovieLens variants. The MovieLens datasets are characterized by a large size, the limited availability of side information, and the very small percentage of available entries in the preference matrix. Overall we have two recommendation problems with very different characteristics.

5.1 Recommendation tasks

We will evaluate the performance of the algorithms we presented in two different variants of the cold start problem. In the first, that we will call *User Cold Start*, we will evaluate the quality of the recommendations generated for unseen users when the set of items over which we provide recommendations is fixed. In the second variant, which we will call *Full Cold*

[2] http://grouplens.org/datasets/movielens/.

Start, we will provide suggestions over both unseen users and items. We will also evaluate the performance of the algorithms in a *matrix completion* setting; here we will randomly remove items from the users preference lists and predict the preferences of the removed items with a model learned on the remaining observed ratings. In model-based collaborative filtering, matrix completion has been usually addressed by low-rank matrix factorization algorithms that do not exploit side information. We want to see whether exploiting the side information can bring substantial improvements in the matrix completion performance.

Note that provision of recommendations in the cold start setting is much more difficult than the matrix completion since in the former we do not have historical data for the new users and new items over which we need to provide the recommendations and thus we can only rely on their side information to infer the latter.

5.2 Comparison baselines

As first baseline we will use LambdaMART (LM) (Burges 2010), in both cold start variants as well as in the matrix completion setting. To ensure a fair comparison of our methods against LamdaMART we will train all of them using the same values over the hyperparameters they share, namely learning rate, size of regression trees and number of iterations. In the matrix completion we will also add CofiRank (CR) as a baseline, a state-of-the-art matrix factorization algorithm in collaborative ranking (Weimer et al. 2007). CR minimizes an upper bound of NDCG and uses the l_2 norm to regularize the latent factors. Its objective function is thus similar to those of our methods and LambdaMART's but it learns directly the latent factors by gradient descent without using the side information.

For the two cold start evaluation variants, we will also have as a second baseline a memory-based approach, one of the most common approaches used for cold start recommendations (Bell and Koren 2007). In the user cold start setting we will provide item recommendations for a new user using its nearest neighbors. This user memory-based (UB) approach will compute the preference score for the jth item as: $\hat{y}_j = \frac{1}{|\mathbb{N}|}\sum_{i\in\mathbb{N}} y_{ij}$, where $\mathbb{N}$ is the set of the 5-nearest neighbors for the new user for which we want to provide the recommendations. We compute the nearest neighbors using the Euclidean distance on the user side information. In the full cold start setting, we will provide recommendations for a new user-item pair by joining the nearest neighbors of the new user with the nearest neighbors of the new item. This full memory-based (FB) approach will compute the preference score for the ith user and jth item as: $\hat{y}_{ij} = \frac{1}{|\mathbb{N}_i|\times|\mathbb{N}_j|}\sum_{c\in\mathbb{N}_i}\sum_{d\in\mathbb{N}_j} y_{cd}$, where $\mathbb{N}_i$ is the set of the 5-nearest neighbors for the new user and $\mathbb{N}_j$ is the set of the 5-nearest neighbors for the new item. Both neighborhoods are computed using the Euclidean distance on the user and item side-information features respectively.

Finally we will also experiment with the LambdaMART variant in which we use the weighted NDCG cost that we described in Sect. 4.2 in order to evaluate the potential benefit it brings over the plain NDCG; we will call this method LMW.

5.3 Meta-mining

The meta-mining problem we will consider is the one provided by Nguyen et al. (2012a). It consists of the application of 35 feature selection plus classification workflows on 65 real world datasets with genomic microarray or proteomic data related to cancer diagnosis or prognosis, mostly from National Center for Biotechnology Information.[3] The preference

[3] http://www.ncbi.nlm.nih.gov/.

score that corresponds to each dataset-workflow pair is given by a number of significance tests on the performances of the different workflows that are applied on a given dataset. More concretely, if a workflow is significantly better than another one on the given dataset it gets one point, if there is no significant difference between the two workflows then each gets half a point, and if it is significantly worse it gets zero points. Thus if a workflow outperforms in a statistically significant manner all other workflows on a given dataset it will get $m-1$ points, where m is the total number of workflows (here 35). In the matrix completion and the full cold start settings we compute the preference scores with respect to the *training* workflows since for these two scenarios the total number of workflows is less than 35. In addition, we rescale the performance measure from the 0–34 interval to the 0–5 interval to avoid large preference scores from overwhelming the NDCG due to the exponential nature of the latter with respect to the preference score.

To describe the datasets and the data mining workflows we use the same characteristics that were used in Nguyen et al. (2012a). Namely 113 dataset characteristics that give statistical, information-theoretic, geometrical-topological, landmarking and model-based descriptors of the datasets and 214 workflow characteristics derived from a propositionalization of a set of 214 tree-structured generalized workflow patterns extracted from the ground specifications of DM workflows.

5.3.1 Evaluation setting

We fix the parameters that LambdaMART uses to construct the regression trees to the following values: the maximum number of nodes for the regression trees is three, the minimum percentage of instances in each leaf node is 10 % and the learning rate, η, of the gradient boosted tree algorithm to 10^{-2}. To build the ensemble trees of LM-MF and LM-MF-Reg we use the same parameter settings as the ones we use in LambdaMART. We select their input and output regularization parameters μ_1 and μ_2 in the grid $[0.1, 1, 5, 7, 10]^2$, by three-fold inner cross-validation. We fix the number of nearest neighbors for the Laplacian matrices to five. To compute the output-space similarities, we used the NDCG similarity measure defined in Eq. 21 where the truncation level k is set to the truncation level at which each time we report the results.

To build the different recommendation scenarios, we have proceeded as follows. In matrix completion we randomly select N workflows for each dataset to build the training set. We choose $N \in \{5, 10, 15\}$. For each dataset, we use ten workflows, different from the N ones selected in training, for validation and we use the rest for testing. This scenario emulates a 86, 71 and 57 % of missing values in the preference matrix. Since in the matrix completion setting the numbers of users and items are fixed, we fix the number of hidden factors r to $\min(n, m) = 35$ for all three matrix factorization algorithms, LM-MF, LM-MF-Reg and CofiRank. For this baseline we used the default parameters as these are provided in Weimer et al. (2007). We report the average NDCG@5 measure on the test workflows of each dataset.

In the user cold start scenario, we will evaluate the performance in providing accurate recommendations for new datasets. To do so we do a leave-one-dataset-out. We train the different methods on 64 datasets and evaluate their recommendations on the left-out dataset. Since there are no missing workflows as previously we also fix the number of hidden factors r to $\min(n, m) = 35$. In the full cold start scenario on top of the leave-one-dataset-out we also randomly partition the set of workflows in two sets, where we use 70 % of the workflows for training and the remaining 30 % as the test workflows for the left-out dataset. That is the number of workflows in the train set is equal to $\lfloor 0.7 \times 35 \rfloor = 24$ which defines the number of hidden factors r. We use the 11 remaining workflows as the test set. For the both cold start

Table 2 NDCG@5 results on meta-mining for the *matrix completion* setting

	$N = 5$	$N = 10$	$N = 15$
CR	0.5755	0.6095	0.7391
LM	0.6093	0.6545	0.7448
δ_{CR}	$p = 0.4567$ (=)	$p = 0.0086$ (+)	$p = 1$ (=)
LMW	0.5909	0.6593	0.7373
δ_{CR}	$p = 1$ (=)	$p = 0.0131$ (+)	$p = 0.6985$ (=)
δ_{LM}	$p = 0.2626$	$p = 0.0736$ (=)	$p = 1$ (=)
LM-MF	0.6100	**0.6556**	0.7347
δ_{CR}	$p = 0.0824$ (=)	$p = 0.0255$(+)	$p = 0.6985$ (=)
δ_{LM}	$p = 0.7032$ (=)	$p = 1$ (=)	$p = 0.7750$ (=)
LM-MF-Reg	**0.61723**	0.6473	**0.7458**
δ_{CR}	$p = 0.0471$ (+)	$p = 0.0824$ (=)	$p = 0.6985$ (=)
δ_{LM}	$p = 0.9005$ (=)	$p = 0.8955$ (=)	$p = 0.2299$ (=)

N is the number of workflows we keep in each dataset for training. For each method, we give the comparison results against the CofiRank and LambdaMart methods in the rows denoted by δ_{CR} and δ_{LM} respectively. More precisely we report the p values of the McNemar's test on the numbers of wins/losses and denote by (+) a statistically significant improvement, by (=) no performance difference and by (−) a significant loss. In bold, the best method for a given N

scenarios, we report the average testing NDCG measure. We compute the average NDCG score at the truncation levels of $k = 1, 3, 5$.

For each of the two algorithms we presented we compute the number of times we have a performance win or loss compared to the performance of the baselines. On these win/loss pairs we do a McNemar's test of statistical significance and report its results, we set the significance level at $p = 0.05$. For the matrix completion we denote the results of the performance comparisons against the CofiRank and LambdaMART baselines by δ_{CR} and δ_{LM} respectively. We give the complete results in table 2. For the user cold start we denote the results of the performance comparisons against the user-memory based and the LambdaMART baselines by δ_{UB} and δ_{LM} respectively, Table 3a. For the full cold start we denote by δ_{FB} and δ_{LM} the performance comparisons against the full-memory-based and the LambdaMART baselines respectively, table 3b.

5.3.2 Results

Matrix completion CR achieves the lowest performance for $N = 5$ and $N = 10$, compared to the other methods; for $N = 15$ the performances of all methods are very similar, Table 2. The strongest performance advantages over the CR baseline appear at $N = 10$; there LMW and LM-MF are significantly better and LM-MF-Reg is close to being significantly better, p-value=0.0824. At $N = 5$ only LM-MF-Reg is significantly better than CR. When it comes to LM and its new variants that we propose here, neither its factorised variant, i.e. LM-MF, nor the regularised version of LM-MF bring any performance improvement for matrix completion in the meta-mining problem.

Table 3 NDCG@k results on meta-mining for the two cold start settings

	$k = 1$	$k = 3$	$k = 5$
(a) *User cold start*			
UB	0.4367	0.4818	0.5109
LM	0.5219	0.5068	0.5135
δ_{UB}	$p = 0.0055$ (+)	$p = 0.1691$ (=)	$p = 0.9005$ (=)
LMW	0.5008	0.5232	0.5168
δ_{UB}	$p = 0.0233$ (+)	$p = 0.2605$ (=)	$p = 0.5319$ (=)
δ_{LM}	$p = 0.4291$ (=)	$p = 0.1845$ (=)	$p = 0.8773$ (=)
LM-MF	0.5532	**0.5612**	**0.5691**
δ_{UB}	$p = 0.0055$ (+)	$p = 0.0086$ (+)	$p = 0.3210$ (=)
δ_{LM}	$p = 0.0636$ (=)	$p = 0.0714$ (=)	$p = 0.1271$ (=)
LM-MF-Reg	**0.5577**	0.5463	0.5569
δ_{UB}	$p = 0.0055$ (+)	$p = 0.0086$ (+)	$p = 0.3815$ (=)
δ_{LM}	$p = 0.0636$ (=)	$p = 0.1056$ (=)	$p = 0.2206$ (=)
(b) *Full cold start*			
FB	0.46013	0.5329	0.5797
LM	0.5192	0.5231	0.5206
δ_{FB}	$p = 0.0335$ (+)	$p = 0.9005$ (=)	$p = 0.0607$ (=)
LMW	0.5294	0.5190	0.5168
δ_{FB}	$p = 0.0086$ (+)	$p = 1$ (=)	$p = 0.0175$ (−)
δ_{LM}	$p = 1$ (=)	$p = 1$ (=)	$p = 0.5218$ (=)
LM-MF	0.5554	**0.6156**	0.5606
δ_{FB}	$p = 0.0175$ (+)	$p = 0.0175$ (+)	$p = 0.6142$ (=)
δ_{LM}	$p = 0.0171$ (+)	$p = 0.0048$ (+)	$p = 0.0211$ (+)
LM-MF-Reg	**0.5936**	0.5801	**0.5855**
δ_{FB}	$p = 0.0007$ (+)	$p = 0.1041$ (=)	$p = 1$ (=)
δ_{LM}	$p = 0.0117$ (+)	$p = 0.0211$ (+)	$p = 0.0006$ (+)

For each method, we give the comparison results against the user, respectively full, memory-based and LambdaMart methods in the rows denoted by δ_{UB}, respectively δ_{FB}, and δ_{LM}. The table explanation is as before. In bold, the best method for a given k

User Cold Start Here LM-MF and LM-MF-Reg achieve performance improvements over both the UB and the LM baselines for all values of the k truncation level, table 3a. At $k = 1$ both of them beat in a statistically significant manner the UB baseline, and they are also close to beating in a statistical significant manner the LM baseline as well, p-value=0.0636. At $k = 3$ both beat in a statistically significant manner UB but not LM, at $= 5$ there are no significant differences. Finally, note the LambdaMART variant that makes use of the weighted NDCG, LMW, does not seem to bring an improvement over the plain vanilla NDCG, LM. Overall while factorization and regularization using user and item information bring performance improvements over the plain vanilla LM these fall short from being statistically significant.

Full Cold Start Here the performance advantage of LM-MF and LM-MF-Reg is even more pronounced, table 3b. Both of them beat in a statistically significant manner the LM baseline for all values k. They also beat the FB baseline in a statistically significant manner at $k = 1$ with LM-MF beating it also in a statistical significant manner at $k = 3$ as well. Overall in the full cold start problem the introduction of factorization in LM brings statistical significant performance improvements; the use of user and item side information also seems to bring performance benefits over the factorised variant of LM.

5.4 MovieLens

The MovieLens recommendation dataset has a quite different morphology than that of the meta-mining. The descriptions of the users and the items are much more limited; users are described only by four features: sex, age, occupation and location, and movies by their genre which takes 19 different values, such as comedy, sci-fi to thriller, etc. We model these descriptors using one-of-N representation.

In the MovieLens dataset the side information has been mostly used to regularize the latent profiles and not to predict them, see e.g. Abernethy et al. (2006) and Agarwal and Chen (2010). Solving the cold start problem in the presence of so limited side information is a rather difficult task and typical collaborative filtering approaches make only use of the preference matrix $\mathbf{Y}$ to learn the user and items latent factors.

5.4.1 Evaluation setting

We use the same baseline methods as in the meta-mining problem. We train LM with the following hyper-parameter setting: we fix the maximum number of nodes for the regression trees to 100, the minimum percentage of instances in each leaf node to 1 % for the cold start problems and 0 % for the matrix completion problem, and the learning rate, η, of the gradient boosted tree algorithm to 10^{-2}. As in the meta-mining problem for the construction of the ensemble of regression trees in LM-MF and LM-MF-Reg we use the same setting for the hyperparameters they share with LM. We optimize their μ_1 and μ_2 regularization parameters within the grid $[0.1, 1, 5, 10, 15, 20, 25, 50, 80, 100]^2$ using five-fold inner cross validation for the cold start problems. For the matrix completion problem, we manually set these two parameters to one. We fix the number of factors r to 50 and the number of nearest neighbors in the Laplacian matrices to five. As in the meta-mining problem we used the NDCG similarity measure to define output-space similarities with the truncation level k set to the same value as the one at which we evaluate the methods.

We will evaluate and compare the different methods under the matrix completion scenario in the MovieLens 100 k dataset and the two cold start scenarios in both MovieLens datasets. In the matrix completion problem, we randomly select N movies per user, $N \in \{5, 10, 15\}$, to build the training set, ten different movies for validation and the rest for testing. This is referred as the *weak* generalization procedure defined in Weimer et al. (2007). In the 100 k MovieLens dataset this procedure generates 95.28, 91.78 and 88.84 % missing preferences respectively over the 6.3 % originally observed preferences. This brings the number of observed entries to less than 1 % for all three values of N. In the user cold start problem, we randomly select 50 % of the users for training and we test the recommendations that the learned models generate on the remaining 50 %. In the full cold start problem, we use the same separation to training and

Table 4 NDCG@10 results on the MovieLens 100 K dataset for the *matrix completion* setting

	$N = 5$	$N = 10$	$N = 15$
CR	0.6254	0.6459	0.6555
LM	0.6046	0.6130	0.6229
δ_{CR}	$p = 1$ (−)	$p = 1$ (−)	$p = 1$ (−)
LM-MF	**0.6755**	**0.6740**	**0.6697**
δ_{CR}	$p = 1\mathrm{e}^{-5}$ (+)	$p = 0.0088$ (+)	$p = 0.3594$ (=)
δ_{LM}	$p = 2\mathrm{e}^{-16}$ (+)	$p = 2\mathrm{e}^{-16}$ (+)	$p = 7\mathrm{e}^{-11}$ (+)
LM-MF-Reg	0.6738	0.6720	**0.6697**
δ_{CR}	$p = 4\mathrm{e}^{-5}$ (+)	$p = 0.0395$ (+)	$p = 0.0723$ (=)
δ_{LM}	$p = 2\mathrm{e}^{-16}$ (+)	$p = 2\mathrm{e}^{-16}$ (+)	$p = 2\mathrm{e}^{-11}$ (+)

N is the number of movies we keep per user for training. The table has the same interpretation as Table 2

testing users and in addition we also randomly divide the item set to two equal size subsets, where we use one for training and the other for testing. Thus the learned models are evaluated on users and items that have never been seen in the model training phase. We give the results for the 100 k and 1 M variants of the MovieLens dataset for the user cold start scenario in Table 5a and for the full cold start scenario in Table 5b.

5.4.2 Results

Matrix completion In Table 4 we can see that for MovieLense 100 k the worst performance matrix completion performance is achieved by LM for all the three values of observed preferences per user (N parameter); LM is statistically significantly worse when compared against CR, LM-MF and LM-MF-Reg. The best performance is achieved by LM-MF and LM-MF-Reg; both are statistically significantly better against the two baselines for $N = 5$ and $N = 10$. For $N = 15$, LM-MF-Reg is also close to being significantly better than CR with a p-value of 0.0723. Overall, we see that when we have a very small number of observed preferences, <1 % of all possible references, LM is performing worse than all matrix factorization algorithms. On the other hand the introduction of latent variables and factorization in LM, i.e. LM-MF, brings statistically significant performance improvements; the use of regularization based on side-information does not seem to bring any improvement over the factorised variant of LM. The advantage of both LM-MF and LM-MF-Reg is much more pronounced compared to the baseline comparison methods when the number of observed preferences per user is very low, $N = 5, 10$.

User cold start In the user cold start scenario, Table 5a, we can see first that both LM-MF and LM-MF-Reg beat in a statistically significant manner the UB baseline in both MovieLens variants for all values of the truncation parameter, with the single exception of LM-MF at 100 k and $k = 5$ for which the performance difference is not statistically significant. LM and its weighted NDCG variant, LMW, are never able to beat UB; LM is even statistically significantly worse compared to UB at 1M for $k = 5$. Moreover both LM-MF and its regularized variant beat in a statistically significant manner LM in both MovieLens variants and for all values of k. LM-MF-Reg has a small advantage over LM-MF for 100 K, it achieves a higher average NDCG score, however this advantage disappears when we move to the 1M dataset, which is rather normal since due to the much larger dataset size there is less need for additional regularization. Moreover since we do a low-rank matrix factorization, we have

Table 5 NDCG@k results on the two MovieLens datasets for the two cold start settings

	100 K		1 M	
	$k = 5$	$k = 10$	$k = 5$	$k = 10$
(a) *User cold start*				
UB	0.6001	0.6159	0.6330	0.6370
LM	0.6227	0.6241	0.6092	0.6453
δ_{UB}	$p = 0.2887$ (=)	$p = 0.8537$ (=)	$p = 0.0001$ (−)	$p = 0.1606$ (=)
LMW	0.6252	0.6241	0.6455	0.6450
δ_{UB}	$p = 0.0878$ (=)	$p = 0.5188$ (=)	$p = 0.0120$ (+)	$p = 0.1448$ (=)
δ_{LM}	$p = 0.7345$ (=)	$p = 0.9029$ (=)	$p = 0.0000$ (+)	$p = 0.5219$ (=)
LM-MF	0.6439	0.6455	**0.6694**	**0.6700**
δ_{UB}	$p = 0.0036$ (+)	$p = 0.1171$ (=)	$p = 0.0000$ (+)	$p = 0.0000$ (+)
δ_{LM}	$p = 2e^{-06}$ (+)	$p = 6e^{-06}$ (+)	$p = 0.0000$ (+)	$p = 0.0000$ (+)
LM-MF-Reg	**0.6503**	**0.6581**	**0.6694**	**0.6705**
δ_{UB}	$p = 4e^{-05}$ (+)	$p = 0.0001$ (+)	$p = 0.0000$ (+)	$p = 0.0000$ (+)
δ_{LM}	$p = 3e^{-06}$ (+)	$p = 0.0000$ (+)	$p = 0.0000$ (+)	$p = 0.0000$ (+)
(b) *Full cold start*				
FB	0.5452	0.5723	0.5339	0.5262
LM	0.5486	0.5641	0.5588	0.5597
δ_{FB}	$p = 0.7817$ (=)	$p = 0.4609$ (=)	$p = 1e^{-06}$ (+)	$p = 0.0001$ (+)
LMW	0.5549	0.5622	0.55737	0.5631
δ_{FB}	$p = 0.4058$ (=)	$p = 0.2129$ (=)	$p = 9e^{-06}$ (+)	$p = 1e^{-06}$ (+)
δ_{LM}	$p = 0.0087$ (+)	$p = 0.8830$ (=)	$p = 0.1796$ (=)	$p = 3e^{-06}$ (+)
LM-MF	**0.5893**	**0.5876**	**0.5733**	**0.5750**
δ_{FB}	$p = 0.0048$ (+)	$p = 0.2887$ (=)	$p = 0.0000$ (+)	$p = 0.0000$ (+)
δ_{LM}	$p = 3e^{-06}$ (+)	$p = 0.0001$ (+)	$p = 0.0000$ (+)	$p = 0.0000$ (+)
LM-MF-Reg	0.5699	0.57865	**0.5736**	0.5683
δ_{FB}	$p = 0.0142$ (+)	$p = 0.1810$ (=)	$p = 0.0000$ (+)	$p = 4e^{-07}$ (+)
δ_{LM}	$p = 0.0310$ (+)	$p = 0.0722$ (=)	$p = 0.0000$ (+)	$p = 0.0000$ (+)

For each method, we give the comparison results against the user, respectively full, memory-based and LambdaMART methods in the rows denoted by δ_{UB}, respectively δ_{FB}, and δ_{LM}. The table explanation is as before. In bold, the best method for a given k

$r = 50$ for the roughly 6000 users. which already is on its own a quite strong regularization rendering also unnecessary the need for the input/output space regularizers.

Full cold start A similar picture arises in the full cold start scenario, Table 5b. With the exception of MovieLens 100 K at $k = 10$ both LM-MF and LM-MF-Reg beat always in a statistically significant the FB baseline. Note also that now LM as well as its weigthed NDCG variant are significantly better than the FB baseline for the 1M dataset and $k = 5, 10$. In adition the weigthed NDCG variant is significantly better than LM for 100 k and 1M at $k = 5$ and $k = 10$ respectively. Both LM-MF and LM-MF-Reg beat in a statistically significant manner LM, with the exception of 100 k at $k = 10$ where there is no significant difference. Unlike the user cold start problem, now it is LM-MF that achieves the best overall performance.

5.5 Discussion

We performed a number of experiments with the new variants of LM that we presented in this paper on three recommendation datasets with very different characteristics. The Meta-Mining dataset is a very small recommendation dataset with only 65 “users” and 35 “items”, with quite detailed descriptions of the users and the items and full preference information available. The movielens datasets are traditional recommendation benchmark datasets with few thousands users and items, limited number of features describing them, and very low percentages of observed preferences. Not surprisingly the observed performance patterns differ over the two dataset types.

In the Meta-Mining dataset the introduction of factorization to LM does not bring any significant performance improvement over plain LM for the matrix completion experiments. In the user cold start experiment while the factorization brings nominal performance improvements over the performance of plain LM these fall short from being statistically significant. In the full cold start the introduction of factorization now brings statistically significant performance improvements; the incorporation of regularisation, LM-MF-reg, seems also to bring performance improvements over LM-MF.

In the MovieLens datasets the factorization of LM brings significant performance improvements in all three experiments we performed, i.e. matrix completion, user and item cold start, over the plain vanilla LM. The addition of regularization, LM-MF-Reg, does not bring significant performance benefits over LM-MF with the exception of the user cold start experiment on the small MovieLens dataset which seems to benefit from the regulariser.

The limited benefit of LM-MF and LM-MF-Reg in the MetaMining experiments, despite the extensive descriptors of users and items, can be explained both by the particularities of the MetaMining problem as well as by the dataset size. It is well known that getting appropriate descriptors for datasets for the metalearning problem is a quite challenging task which still has not been addressed in a satisfactory manner (Brazdil et al. 2009). The size of the MetaMining dataset is also quite small compared to that typically seen in recommendation problems. This is particularly pronounced in the matrix completion setting where we have generated very large levels of missing preference scores, between 57 and 86 % of the total number of preference scores, resulting in hardly any performance difference between the different methods. In the two cold start settings we have a considerably larger proportion of available training data, which allows for a better performance differentiation of the methods, based on how well they exploit the available training data for learning.

6 Related work

Most of the work related to ours comes from matrix factorization. Nguyen et al. (2012b) proposed a hybrid recommendation algorithm for the meta-mining problem that was learning metrics (projections) of the user and item spaces to a common space; the inner product of the projections was the preference score. We used gradient descent to optimise a squared loss in which we regularised the projections of users and items to the common space to reflect the output space similarities generated from the preference matrix. The current paper in a sense generalises that work by learning non-linear mappings of the user and item features to the latent space and optimizing a ranking-based loss. Rao et al. (2015) propose an algorithm for matrix completion in collaborative filtering which uses the same Laplacian regularisers as we do over the input space and optimises a square loss using conjugate gradient. Li and Yeung (2009) do matrix factorization using a squared loss function and apply Laplacian regularisation on the user side. Similar Laplacian regularisers have been used for matrix completion in Kalofolias et al. (2014) with a squared loss coupled with low rank regularisation on the completed matrix together and the Laplacian regularisers imposed directly on the columns and rows of the completed matrix. Cai et al. (2011) learn using gradient descent non-negative matrix factorizations with a divergence or a squared loss function coupled with a Laplacian regulariser on the user part. All of the above do matrix completion using squared loss functions and with the exception of Nguyen et al. (2012b) do not handle the cold start recommendation problem. The use of Laplacian regularisers is rather common and they have been used in very different settings such as matrix factorization for transfer learning (Long et al. 2012). Other relevant works include Chen et al. (2013), where the authors do matrix factorization, with a quadratic loss function using gradient boosting, and Lee and Lin (2015), where the authors also do matrix factorization using as cost function NDCG which they optimize with lambda gradient as done in LambdaRank algorithm (Burges 2010).

7 Conclusion

Since only top items are observable by users in real recommendation systems, we believe that ranking loss functions that focus on the correctness of the top item predictions are more appropriate for this kind of problem. LambdaMART is a state of the art learning to rank algorithm the loss function of which does focus on the correctness of the top items predictions. We build on LambdaMART and we propose two basic variants of it. Motivated by the fact that in recommendation problems the descriptions of users and items are governed by a small number of latent factors we cast the learning to rank problem as the learning of a low-rank matrix factorization of the preference martix; the learned factors correspond to low dimensional descriptions of users and items and the user-item preference scores are now computed as inner products of the latent representations of the users and items. Moreover we regularise the learning of the latent factors imposing Laplacian regularisers that constrain them to reflect the user and item manifolds as these are given by their feature descriptions and the preference matric.

We experimented with the two new LambdaMART variants on a couple of very different recommendation datasets, MetaMining and MovieLens, in three different recommendation settings, matrix completion, cold start, and full cold start. In the MetaMining dataset the matrix factorization brings significant performance improvements only for the full cold start setting; in the MovieLens dataset the improvements are significant in all three recommendation settings. The incorporation of the regularisation on the factorized LambdaMART variant seems to improve performance only on a small subset of the experiments, namely full cold start in the MetaMining dataset and the user cold start on the small MovieLens dataset.

References

Abernethy, J., Bach, F., Evgeniou, T., & Vert, J. P. (2006). Low-rank matrix factorization with attributes. arXiv preprint arXiv:cs/0611124

Agarwal, D., & Chen, B. C. (2010). flda: Matrix factorization through latent dirichlet allocation. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pp. 91–100. ACM, New York, NY, USA. doi:10.1145/1718487.1718499.

Bell, R. M., & Koren, Y. (2007). Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Seventh IEEE international conference on data mining, 2007. ICDM 2007.* (pp. 43–52). IEEE.

Brazdil, P., Giraud-Carrier, C. G., Soares, C., & Vilalta, R. (2009). *Metalearning—Applications to data mining. Cognitive Technologies*. Berlin: Springer. doi:10.1007/978-3-540-73263-1.

Burges, C. J. (2010). From ranknet to lambdarank to lambdamart: An overview. *Learning*, *11*, 23–581.

Burges, C. J., Svore, K. M., Bennett, P. N., Pastusiak, A., & Wu, Q. (2011). Learning to rank using an ensemble of lambda-gradient models. *Journal of Machine Learning Research-Proceedings Track*, *14*, 25–35.

Cai, D., He, X., Han, J., & Huang, T. S. (2011). Graph regularized nonnegative matrix factorization for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(8), 1548–1560.

Chen, T., Li, H., Yang, Q., & Yu, Y. (2013). General functional matrix factorization using gradient boosting. In *Proceedings of the 30th international conference on machine learning, ICML 2013*, Atlanta, GA, USA, 16–21 June 2013, pp. 436–444.

Donmez, P., Svore, K. M., & Burges, C. J. (2009). On the local optimality of lambdarank. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (pp. 460–467). ACM

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*(5), 1189–1232.

Fürnkranz, J., & Hüllermeier, E. (2010). *Preference learning*. Berlin: Springer.

Hilario, M., Nguyen, P., Do, H., Woznica, A., & Kalousis, A. (2011). Ontology-based meta-mining of knowledge discovery workflows. In N. Jankowski, W. Duch, & K. Grabczewski (Eds.), *Meta-learning in computational intelligence*. Berlin: Springer.

Järvelin, K., & Kekäläinen, J. (2000). IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on research and development in information retrieval* (pp. 41–48). ACM

Kalofolias, V., Bresson, X., Bronstein, M. M., & Vandergheynst, P. (2014). Matrix completion on graphs. CoRR abs/1408.1717

Lee, G., & Lin, S. (2015). Lambdamf: Learning nonsmooth ranking functions in matrix factorization using lambda. In *2015 IEEE international conference on data mining, ICDM 2015*, Atlantic City, NJ, USA, November 14–17, 2015 (pp. 823–828).

Li, W., & Yeung, D. (2009). Relation regularized matrix factorization. In *IJCAI 2009, Proceedings of the 21st international joint conference on artificial intelligence*, Pasadena, California, USA, July 11–17, 2009, pp. 1126–1131.

Long, M., Wang, J., Ding, G., Shen, D., & Yang, Q. (2012). Transfer learning with graph co-regularization. In J. Hoffmann, & B. Selman (Eds.), *Proceedings of the twenty-sixth AAAI conference on artificial intelligence*, July 22–26, 2012, Toronto, Ontario, Canada. AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4958.

Nguyen, P., Wang, J., Hilario, M., & Kalousis, A. (2012a). Learning heterogeneous similarity measures for hybrid-recommendations in meta-mining. In *IEEE 12th international conference on data mining (ICDM)* pp. 1026–1031. doi:10.1109/ICDM.2012.41.

Nguyen, P., Wang, J., Hilario, M., & Kalousis, A. (2012b). Learning heterogeneous similarity measures for hybrid-recommendations in meta-mining. In *12th IEEE international conference on data mining, ICDM 2012*, Brussels, Belgium, December 10–13, 2012, pp. 1026–1031.

Rao, N., Yu, H., Ravikumar, P., & Dhillon, I. S. (2015). Collaborative filtering with graph information: Consistency and scalable methods. In *Advances in neural information processing systems 28: Annual conference on neural information processing systems 2015*, December 7–12, 2015, Montreal, Quebec, Canada, pp. 2107–2115.

Srebro, N., Rennie, J. D. M., & Jaakkola, T. S. (2005). Maximum-margin matrix factorization. In L. K. Saul, Y. Weiss, & L. Bottou (Eds.), *Advances in neural information processing systems* (Vol. 17, pp. 1329–1336). Cambridge, MA: MIT Press.

Weimer, M., Karatzoglou, A., Le, Q. V., & Smola, A. (2007). Maximum margin matrix factorization for collaborative ranking. In *Advances in neural information processing systems*

Yue, Y., & Burges, C. (2007). On using simultaneous perturbation stochastic approximation for ir measures, and the empirical optimality of lambdarank. In *NIPS Machine Learning for Web Search Workshop*.