Collaborative Ranking

Suhrid Balakrishnan
AT&T Labs–Research
180 Park Ave.
Florham Park, NJ
suhrid@research.att.com

Sumit Chopra
AT&T Labs–Research
180 Park Ave.
Florham Park, NJ
schopra@research.att.com

ABSTRACT

Typical recommender systems use the root mean squared error (RMSE) between the predicted and actual ratings as the evaluation metric. We argue that RMSE is not an optimal choice for this task, especially when we will only recommend a few (top) items to any user. Instead, we propose using a ranking metric, namely normalized discounted cumulative gain (NDCG), as a better evaluation metric for this task. Borrowing ideas from the learning to rank community for web search, we propose novel models which approximately optimize NDCG for the recommendation task. Our models are essentially variations on matrix factorization models where we also additionally learn the features associated with the users and the items for the ranking task. Experimental results on a number of standard collaborative filtering data sets validate our claims. The results also show the accuracy and efficiency of our models and the benefits of learning features for ranking.

Categories and Subject Descriptors

* [*]: *

General Terms

*

Keywords

Recommender Systems, Learning to Rank, Collaborative Ranking, NDCG, RMSE

1. INTRODUCTION AND MOTIVATION

The past decade has seen a large number of web-based recommendation systems deployed with great success in diverse domains. Their surge can be attributed to a variety of factors. Among them, a key factor is the rather limitless choice of items available to a user in a multitude of applications. A partial list of extremely popular web-based recommender services are: Netflix, for movies, Amazon.com for products, Pandora, Last.fm, and iTunes Genius for music, YouTube's Recommended For You, for online videos, Facebook's Other People You May Know, for social networking, What Should I Read Next, for books etc.

Before a typical recommender system can make personalized recommendations for a user, it needs to elicit that users preferences. This is done by soliciting numeric (and now ubiquitous) "star" ratings from the user for select items. For instance, Netflix makes this the first step for a user as soon as she creates an account on their website; they ask a new user to rate some minimum number of preselected movies/TV shows on a 5 point star scale ranging from 1 = *Hate it*, to 5 = *Love it*. After sufficiently many ratings have been collected, collaborative filtering (CF) [16] techniques are applied to the entire dataset (all users, all items). CF models form the core of most recommender systems. They work by extrapolating unobserved user-item preferences from preference information collected from the target user, and the preferences of all the other users. Finally, recommendations are made, and the user can be shown the items estimated to be the most preferred by her. This is usually done by using CF models to first estimate the preferences of all the items; next, the items are sorted by these estimated preferences; and finally, a (small) subset of the top items are shown to the user as her recommendations.

Since many recommender systems operate in this paradigm of using explicit ratings as a surrogate for user preferences, a natural way to train and evaluate such systems has emerged. In this view, the recommendation task reduces to predicting the ratings for an unseen user-item pair. In order to evaluate the performance of such systems, the collected ratings data from all the users, is partitioned into a training portion (the training set) and a disjoint/non-overlapping testing portion (the test set). Models are learned on the training set and evaluated on the test set (held-out ratings not seen during training). Since the recommendation task is framed as predicting the ratings for unseen user-item pairs, evaluation consists of quantifying how well the recommender predicts the test user-item ratings. This is essentially a regression task, and the Mean Squared Error (MSE) between the model predictions and the actual test ratings is a natural choice for an evaluation metric. Models with lower test MSE have better predictions, on average, than models with higher MSE. Furthermore, once test MSE is the evaluation criterion, training MSE is also automatically the first choice for a loss function while training these models.

While this has been a completely valid and a successful approach to building recommender systems, we, along with a few other researchers [20, 6], believe that using MSE criteria to evaluate such systems is a suboptimal fit to the recommendation task. The reasons are two-fold. First, as mentioned earlier, the way most recommenders are used in practice is to generate a top-k list of the items to show each user. Therefore, only recommendations that the system estimates to be highly rated by her will ever be shown to a user. A user is never shown any item which the system believes she will not give a high rating to. Since MSE places equal emphasis on all the ratings, high and low, minimizing MSE results in training models that predict the low ratings as accurately as the high ones. Therefore it appears a lot of extra work is done unnecessarily, in trying to solve a harder problem than we need to. As an example, in the case of Netflix, since we know beforehand that we will only show (estimated) 4/5-star items to a user, requiring our system to be good at predicting a 1-star rating accurately seems wasteful of modeling capacity. The second main criticism of MSE as a training criterion is that most of the time, the actual predicted ratings values themselves are not even shown to the users directly (Netflix is a notable exception). If the only use for the numerical values of the predictions is generating a top-k list, precision in the predicted value of these ratings also doesn't seem to be necessary. Instead, getting the order of the items right seems critical. In particular, the correct order at the top of the list appears to be key.

For these reasons we argue that instead of using MSE, a ranking metric will be better suited to evaluating recommendation systems. In particular, in this paper we show that Normalized Discounted Cumulative Gain (NDCG) is a particularly good fit as a metric for evaluating such systems. NDCG (see Section 2.1 for mathematical details) was developed with ranking in information retrieval as the target application [12]. Given relevance values (typically on an ordinal scale) for a set of items (web-pages) returned as response to a search query, NDCG can score any list of permutations of these items. NDCG was designed so that the list with the highest relevance items in the top ranked positions is the one which gets the maximum score. Since a recommender system will be used to show users only a few suggested items, our focus is precisely on the items populating the top of the list. As long as low ranked items can be distinguished from the high ranked ones, estimating their predicted ratings is of little to no consequence to NDCG and hence to the ranking task. We call this approach to recommendation *Collaborative Ranking*.

Motivated by the work in the domain of learning-to-rank for web-search, we propose two classes of models for the collaborative ranking problem; point-wise models and pair-wise models. The idea behind both these models is to learn a parametric function which assigns a relevance score to every input, which is a user-item pair. The distinction between the two models lie in the way the parameters of the function are learned to optimize the ranking metric.

A key issue in using these models in the recommendation setting, as opposed to web-search ranking, is the lack of availability of explicit input features. To this end we propose a novel solution this problem, which involves learning these features while simultaneously learning the parameters of the ranking function during the optimization of the loss function. We validate our claims by running our models on a number of standard real-world collaborative filtering data sets. The results show the efficiency and accuracy of our models. We also show the benefits of learning the input features tuned to the ranking task at hand.

In Section 2 we briefly discuss the outline of the state of the learning-to-rank community and the types of models people use for web-search ranking. Section 3 discusses in detail the our proposed models for the collaborative ranking problem. We then discuss related work and evaluation in Section 4. Next, we turn to our experiments and results in Section 5. Finally, we end with our conclusions and discuss future work in Section 6.

2. LEARNING TO RANK

The web-search task is related to finding information on the Internet. In the standard setting, users provide a search engine with a set of search terms (the query), and the search engine returns a list of relevant hyperlinks (listings or impressions). The user may then follow hyperlinks that look relevant, or refine her search etc. Research effort has focused on many specific aspects of this task. One prominent area of recent research attention has been automated web-search result ranking. In particular, motivated by the observation that most users focus their attention on only a handful of impressions typically placed on the top of the first page of the search result, one desired characteristic is that this list of impressions returned by the search engine should be ranked according to their relevance to the search query. In the web-search literature this task, concerned with finding models that return listings by strongly taking into account their rank, is known as the "learning to rank" (LTR) problem.

Under most circumstances, in LTR, the ranking problem is treated as a supervised learning problem, where access to a labeled training data $\mathcal{D}$ is assumed. A canonical training data set $\mathcal{D}$ consists of a list of query-impression pairs (q_i, l_i) labeled with the corresponding relevance score y_i: $\mathcal{D} = \{(q_i, l_i, y_i) : i \in [1 \ldots n]\}$. The relevance labels, y_i are typically editorial judgments on some ordinal scale $\mathcal{Y}$ (e.g., $y = 1$, not relevant to $y = 5$, very relevant), and the query-impression pairs are represented by a fixed length feature vectors $\mathbf{x}_{ql} \in \mathbb{R}^d$. The exact features used by search engines are proprietary, but generally include quantities measuring the textual overlap between the query and the listing body text, anchor text, the URL and document title. They may also include quantities specific to the query, like length of the query, or quantities specific to the listing: it's pagerank, linkage characteristics etc. Learning involves estimating the parameters of a model which takes as input the features $\mathbf{x}_{ql}$ associated with the query-impression pair (q, l), and produces the correct relevance score y.

2.1 NDCG

Recall that our goal is to produce ranking mechanisms that position relevant impressions in the very top of a ranked list extremely well. As mentioned earlier, it is known that the top few impressions get much more attention (clicks as well as views) than impressions further down the page [18]. Further down the list attention drops off even more rapidly, and

many users rarely go beyond the first page of results [18]. Reflecting these concerns, an evaluation metric that has become very popular in the LTR community is Normalized Discounted Cumulative Gain or NDCG [12].

One can calculate the NDCG scores for any permutation of a set of items whose relevance labels are known. NDCG has one user-defined parameter and two functions that make it desirable in the ranking setting. The parameter k is a cutoff parameter, and determines how many items in the ranked list to consider. The two functions are the *gain function* and the *discount function*. The gain function allows a user to set the significance of each relevance level. The discount function makes items lower down in the ranked list contribute less to the NDCG score. More specifically, let $\mathbf{y}$ be a vector of the relevance values for a sequence of items (e.g., the impressions associated with one query). Let π denote a permutation over the sequence of items in $\mathbf{y}$. π, for example, would be the order of the listings returned by a trained ranking algorithm. Then π_q is the index of the qth item in π and y_{π_q} is the actual relevance value of this item. The Discounted Cumulative Gain (DCG) for this permutation π is defined as:

$$\text{DCG@}k(\mathbf{y}, \pi) = \sum_{q=1}^{k} \frac{2^{y_{\pi_q}} - 1}{\log_2(2+q)}.$$

The gain function is a power of two in this definition (minus 1), and the discount function has a logarithmic decay as we move down the ranked list. Normalized Discounted Cumulative Gain (NDCG) can then be defined as:

$$\text{NDCG@}k(\mathbf{y}, \pi) = \frac{\text{DCG@}k(\mathbf{y}, \pi)}{\text{DCG@}k(\mathbf{y}, \pi^*)}, \tag{1}$$

where π^* is a permutation over the sequence of items which corresponds to any perfect ordering based on the relevance scores; an ordering where no item with a low relevance score appears earlier than any item with a higher relevance score in the ordering (in other words, with the items sorted by $\mathbf{y}$, and ties broken arbitrarily).

Ideally, one would like to learn the parameters of the ranking model so that it directly maximizes the NDCG of the predicted rankings. However the fact that the metric is discontinuous everywhere presents significant challenges while optimizing it. Furthermore, the inherent "sort" associated with it makes the design of smooth relaxations/approximations of this metric equally difficult. To this end several authors [5, 3, 15] have proposed a number of solutions for the LTR problem, which involve a surrogate loss function whose minimization will loosely approximate the maximization of the NDCG metric. We describe some of these LTR proposals at a high level next.

2.2 Learning to Rank Approaches

Models in the LTR paradigm can be grouped into three main categories: point-wise, pair-wise and the so called list-wise approaches. The differences are mainly with respect to the form of loss function and training data used.

Point-wise models estimate a parametric function $h(\mathbf{x}_{ql})$ that takes as input the features associated with each query-impression item and produces as output a relevance score. In order to train the parameters of this function, one can either use a regression loss when the predicted relevance scores are continuous ($h : \mathbb{R}^d \to \mathbb{R}$) [5], or a classification loss when the relevance scores are discrete ($h : \mathbb{R}^d \to \mathcal{Y}$) [15]. Typically, while training, the performance on a ranking metric, such as NDCG, is mimicked by the judicious choice of the target response. For instance in [5], the authors work with a exponentiation of the actual relevance values as the targets for regression. This rescales the relevance value y to a value closer to the numerator of the NDCG metric. Thus point-wise models are essentially regression/classification models for web-search query-impression, relevance data. Note that in point-wise models, the distinction between queries is typically expressed only through appropriate query-impression features, $\mathbf{x}_{ql}$. In other words, the query q and impression l only define appropriate features; these examples with their features $\mathbf{x}_{ql}$ and associated responses are subsequently modeled independently. Underlying the modeling is the assumption that the query-impression features $\mathbf{x}_{ql}$ are informative enough to distinguish their given relevance levels. For the models themselves, flexible and powerful (non-linear) regression/classification methods like gradient boosted regression trees are typically used. The advantage of such point-wise models is that learning models is relatively straightforward, and more importantly, these methods scale well to large size problems.

Pair-wise models also estimate a function, $g(\mathbf{x}_{ql})$ that scores each query-impression item[1] $g : \mathbb{R}^d \to \mathbb{R}$. However, the training loss for pair-wise models is based on pairs of impressions with the same query, $\{(q_i, \mathbf{x}_{q_i l_i}, y_i), (q_j, \mathbf{x}_{q_j l_j}, y_j)\}$ where $q_i = q_j$, but $l_i \neq l_j$. The pair-wise loss depends on the order of the relevance labels of the items in the pair. The loss is low when the predicted order is correct, or in other words, when the more relevant item in a pair is predicted to have a higher score than the less relevant item. In our example, with $y_j > y_i$ the loss would be low for $g(\mathbf{x}_{q_j l_j}) > g(\mathbf{x}_{q_i l_i})$. The loss would be high when there is a mistake in the order. For instance, in our previous example, when $y_j > y_i$, but $g(\mathbf{x}_{q_j l_j}) < g(\mathbf{x}_{q_i l_i})$. Examples of pairwise approaches are RankBoost [8], RankNet and LambdaRank [3]. The intuition for these models is that a function that can order well should also be able to rank the items well.

We also briefly give pointers to list-wise approaches, such as, AdaRank and PermuRank [21, 22] which are beyond the scope of our work (and may be an interesting avenue of future work). These methods have been devised to iteratively optimize specialized ranking performance measures like NDCG, mean-average precision etc.

We next proceed to outlining how these LTR approaches (point-wise and pair-wise) can be applied to the recommendation problem. A number of questions naturally crop up. How exactly do we set up the recommendation problem in an analogous fashion to ranking? What is the concept corresponding to a query in the collaborative filtering setting? What features can we use? What choice of parametric functions will perform well? How do we modify the point-wise

[1]There are analogous classification based pairwise models as well, but we will be concerned with regression models in this paper.

and pair-wise loss to optimize performance on NDCG? We tackle these and other issues next.

3. COLLABORATIVE RANKING

Recall that the objective of a recommender system is to make effective recommendations given the ratings data R, on a set of m users $\mathcal{U}$, and n items $\mathcal{I}$. In our notation, we will use the index $j \in \mathcal{U}$ to index over the users and the index $i \in \mathcal{I}$ to index over the items. Thus, we can refer to an individual rating for item i and user j by R_{ij}. Since in practice a typical user rates only a small set of items, the ratings data R is usually very sparse.

We first outline latent factor models [14], a standard class of collaborative filtering models that are very popular and effective. In a latent factor model, we learn d-dimensional vectors representing "factors" for each user and each item. Typically, d is much smaller than either m or n, and it models our belief that a small number of unobserved factors are sufficient to provide accurate ratings. We then make predictions for a rating $\hat{R}_{ij}$, by taking the dot product of the factors for item i and user j. Mathematically:

$$\hat{R}_{ij} = \mathbf{v}_i^{\mathrm{T}} \mathbf{u}_j,$$

where the factors for each item i and user j, are the d-dimensional vectors $\mathbf{v}_i$ and $\mathbf{u}_j$.[2] We will also refer to the collection of all the user factors with $\mathbf{U}$, and similarly the collection of all the item factors with $\mathbf{V}$.

When MSE is the evaluation and training criterion, this leads to parameter estimation using modifications of the following primary objective function:

$$\min_{\mathbf{u},\mathbf{v}} \sum_{\{i,j\} \in R} (R_{ij} - \mathbf{v}_i^{\mathrm{T}} \mathbf{u}_j)^2 + \lambda \left(\sum_{i=1}^{n} ||\mathbf{v}_i||^2 + \sum_{j=1}^{m} ||\mathbf{u}_j||^2 \right). \quad (2)$$

The differences in various latent factor models are usually in the form of the regularization of the factors. In the formulation above, from Probabilistic Matrix Factorization (PMF, [19]) factor regularization is in the form of a squared ℓ_2 norm on the factors. We overload the notation of R slightly to also denote indexing over the non-zero ratings in the sparse matrix ($\{i,j\} \in R$). Also, test MSE is given by $1/|R^*| \sum_{\{i,j\} \in R^*} (R^*_{ij} - \mathbf{v}_i^{\mathrm{T}} \mathbf{u}_j)^2$, where once again we overload the notation of the test ratings matrix R^*. In particular, $|R^*|$ denotes the number of test user-item pairs.

As argued earlier, we believe that the squared loss over the predicted and actual ratings is suboptimal for top-k recommendations. This is because MSE loss treats all wrongly predicted ratings equally. We instead propose to use mean NDCG@k (Equation 1) as the evaluation metric. We craft suitable ranking losses borrowing ideas from the LTR community in order to learn models which are more accurate for the ranking task. We call this paradigm collaborative ranking (as in [20]). Following LTR methods (see Section 2), we propose point-wise and pair-wise solutions to the problem.

3.1 Point-wise Models

[2]Note that an error term is also typically included to complete the model specification. Gaussian noise is typically the error form used.

The first set of methods we propose are similar to the regression based point-wise approaches used in the LTR community. The key insight is an idea proposed in [5] and also in [15]; perfect regression (or classification) will also result in perfect NDCG. Analyzing DCG, Cossock and Zhang were able to bound DCG errors by regression errors [5]. Similarly, Li et. al in [15] were able to bound DCG errors by classification errors. Practically, this implies that regression (or classification) is a feasible path towards optimizing for ranking. These results are favorable because both regression and classification are extremely well studied and scalable.

Recall that in the LTR setting, there are queries, impressions, and corresponding relevance labels, and that query-impression pairs give rise to the features in these models. For our recommender task however, we only have sparse ratings data R. Hence it is unclear how to directly apply the above regression (or classification) type approaches used in LTR to the collaborative ranking task. In particular, we need to clearly define the analogous concepts to a query, the query-impression features and the relevance values. Recall that LTR problem essentially involves ranking a set of impressions in response to a query. Analogously, one can view the problem of recommendations, as ranking a set of items for a particular user. Hence a user in the recommendation task correspond to a query in web-search. Next, while we wish to use mean test NDCG as an evaluation metric, unlike in LTR, we do not have explicit user-item features with which to train the models. Conveniently, this is exactly what we get as a result of training a latent factor model. Finally, with queries being equivalent to users, and features being defined, the relevance values for query-impressions, intuitively correspond to the user-item rating values that we have observed.

This leads to our first proposal. It is a two step procedure. The first step involves training a latent factor model using RMSE to obtain the user and item factors. In our experiments we used the PMF model described in Equation 2. After the model is trained, the factors for item i and user j are concatenated to form a feature vector associated with the item-user pair (i,j). These form "features" for each rating of the item-user pair. In the second step, using the features learned from the first step as input, we apply a regression based point-wise LTR algorithm to optimize for ranking. We call this simple two-stage method $\mathrm{CR}_{\triangleright\mathrm{MF}}$, where the $\triangleright$ symbol is used to denote that it is point-wise technique, and the MF stands for features estimated using matrix factorization.

We have found any class of regression function g that has sufficient capacity works well with $\mathrm{CR}_{\triangleright\mathrm{MF}}$. We experimented with various non-linear regressors with roughly equal success. In particular, we tried gradient boosted trees [9, 15], random forests [2, 17] and multi-layer neural networks [3].

In more detail, the procedure for training a $\mathrm{CR}_{\triangleright\mathrm{MF}}$ model based on a neural network for g is as follows:

1. Given ratings data R, we first train a d-dimensional factor model, such as PMF, which minimizes the objective function in Equation 2. This results in a set of item and user factors $\mathbf{V}$ and $\mathbf{U}$.

2. We then create a new training dataset $\mathcal{D}$ for rank-

based training as follows: For every observed training rating R_{ij} from user i and movie j, we create a single fixed length feature vector $\mathbf{x}_{ij} = [\mathbf{v}_i; \mathbf{u}_j]$ of length $2k$. The symbol ";" denotes column-wise concatenation. Further, for every target rating R_{ij} we also create a modified response y_{ij} given by $y_{ij} = 2^{R_{ij}} - 1$. This rescaling of the response to better reflect NDCG gain was suggested in [5] and was found to work well in [15]. Thus, we have a data set $\mathcal{D} = \{(\mathbf{x}_z, y_z)\}$ with $z = \{i, j\} \in R$, indexing all the training ratings.

3. Using this labeled data set $\mathcal{D}$ we then learn a suitable regression function g, with parameters $\boldsymbol{\theta}$, to map user-item features to the rescaled rating value. Thus g is a function such that $g : \mathbb{R}^{2k} \rightarrow \mathbb{R}$. Learning involves minimizing the MSE on the rescaled responses:

$$\sum_{(\mathbf{x}_z, y_z) \in \mathcal{D}} (y_z - g(\mathbf{x}_z))^2. \quad (3)$$

This minimization is accomplished using stochastic gradient descent. In particular, for every item-user pair (i, j), its feature vector $\mathbf{x}_{ij} = [\mathbf{v}_i; \mathbf{u}_j]$ is forward propagated through the function g to compute the output and the loss. The gradient of the loss with respect to the parameters of the function is computed using back-propagation, and the parameters $\boldsymbol{\theta}$ are updated. We use learning rate annealing and early stopping as a way to regularize the parameters.

4. At test time, given a user-item pair to make a prediction for, we construct the corresponding user-item features, and obtain a score using the learnt function g.

5. Once our model has predicted the scores of all the item-user pairs, a ranked list of movies for any user can be trivially obtained by sorting the movies according to their scores for that user.

One issue with this model is its two step nature. In the first phase the features for $\text{CR}_{\triangleright\text{MF}}$ are learned using MSE minimization, and it is not clear that these are the optimal features to use for the ranking task. Ideally one would want to use features best suited for the final task at hand. Using neural networks as the ranking function provides us with an attractive opportunity of learning the parameters of the network while simultaneously learning the item-user feature representation tuned for the ranking task. This results in a unified model for this problem, which is entirely ranking based and estimated from the data. The feature learning aspect of this work is inspired by models for Natural Language Processing problems [1, 4].

We call this unified model $\text{CR}_{\triangleright\text{LF}}$ (LF for Learned Factors). At a high level, $\text{CR}_{\triangleright\text{LF}}$ is like a typical factor model, in that it has d-dimensional factors $\mathbf{U}$ and $\mathbf{V}$ for all training users and items. Additionally though, $\text{CR}_{\triangleright\text{LF}}$ has a set of parameters $\boldsymbol{\theta}$ that functionally determine how to compute predicted responses, which are scaled ratings. Both the factors and the model parameters $\boldsymbol{\theta}$, are learned simultaneously for the ranking task. We train this model using the following EM-like algorithm:

1. We rescale the ratings as $y_{ij} = 2^{R_{ij}} - 1$.

2. The factors $\mathbf{U}$ and $\mathbf{V}$ are initialized to some random values. One can potentially initialize the factors in other ways, such as, using the output of the matrix factorization procedure.

3. Next we fix the factors $\mathbf{U}$ and $\mathbf{V}$, and estimate the parameters $\boldsymbol{\theta}$ of the ranking function, which takes these features as input and produces a rescaled rating as output. This training is done by minimizing the MSE over the rescaled target response and the output of the regression function, using a stochastic gradient descent procedure (similar to what we described for $\text{CR}_{\triangleright\text{MF}}$).

4. After one epoch of parameter training in step 3, we fix the parameters $\boldsymbol{\theta}$ and train the features $\mathbf{U}$ and $\mathbf{V}$. This step also uses stochastic gradient descent. However instead of computing the gradients respect to the parameters $\boldsymbol{\theta}$, we compute the gradient of the loss with respect to the feature vector $\mathbf{x}_{ij} = [\mathbf{v}_i; \mathbf{u}_j]$ using back-propagation and then update the feature vector.

5. We iterate steps 3 and 4 until validation error does not decrease and return the learned $\mathbf{U}$, $\mathbf{V}$ and $\boldsymbol{\theta}$.

Our experiments show this to be an accurate and efficient technique for collaborative ranking. We next turn to pair-wise approaches for collaborative ranking.

3.2 Pair-wise Models

Starting with [11], researchers have focused attention on pairwise training losses for the ranking problem. In the LTR community, SVMrank [13] by Joachims, RankNet and LambdaRank [3] by Burges et.al., have been particularly influential. Pair-wise techniques are attractive because eliciting explicit ratings (or relevance scores) suffers from a drawback known as *calibration* [10], which can be illustrated with two examples. First, users often have incompatible ratings on the same scale. For instance, on a ratings scale of 1 to 5, a rating of 4 for some user A might be comparable to a rating of 5 for another user B. Second, sometimes users start off by rating items in an overly generous (or harsh) manner and retrospectively come to regret those initial ratings after seeing some number of items. Instead of getting accurate predictions of relevance scores of individual items, pair-wise techniques bypass the issues associated with calibration by focusing directly on learning the rank order between a pair of items correctly.

As in the case of point-wise models, the ideas behind the pair-wise models introduced for the LTR problem can be directly applied for solving the collaborative ranking problem, so long as we can define appropriate user-item features. The procedure at the heart of a pair-wise approach is to learn a parametric function $g(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$, parameterized by $\boldsymbol{\theta}$, which operates on single user-item feature vector $\mathbf{x}_{ij}$ and returns a score. However, the approach is pair-wise because the parameters $\boldsymbol{\theta}$ are trained using pairs of examples. This leads to a model that can make test predictions quickly, because it operates on a single user-item feature vector; pairs of examples are only used in training, they are not needed at test time.

Our first proposal is an analogous pair-wise extension to the two stage point-wise model $\text{CR}_{\triangleright\text{MF}}$. We denote this

model with the symbol $\text{CR}_{\bowtie\text{MF}}$, where $\bowtie$ refers to pair-wise training. The detailed procedure is as follows:

1. Given the ratings data R, we first train a d-dimensional factor model to obtain item and user factors $\mathbf{V}$ and $\mathbf{U}$.

2. We then create a dataset $\mathcal{D}$, with the same features as the point-wise features, namely, for every item-user pair (i, j) we generate a feature vector $\mathbf{x}_{ij}$, obtained by concatenating the factors associated with item i and user j: $\mathbf{x}_{ij} = [\mathbf{v}_i; \mathbf{u}_j]$. The target response y_{ij} for this item-user pair is set to the original rating $y_{ij} = R_{ij}$ (no rescaling in this case). At the end of this step, we have a training data set $\mathcal{D} = \{(\mathbf{x}_z, y_z)\}$ with $z = \{i, j\} \in R$.

3. We then learn the parameters $\boldsymbol{\theta}$ of our scoring function $g(\mathbf{x})$, on pairs of examples from $\mathcal{D}$ by setting up an appropriate probabilistic regression [3]. Let a pair of ratings, on two different items, from a particular user be indexed by z_1 and z_2. Define $o_{z_1} = g(\mathbf{x}_{z_1})$ as the model output (the predicted rating score), and $o_{z_1 z_2} = g(\mathbf{x}_{z_1}) - g(\mathbf{x}_{z_2})$ as the difference between the predicted scores on a pair of items from the same user. Let $P_{z_1 > z_2}$ denote the probability under the model that item z_1 is ranked higher than item z_2 for this user. Following [3] we use a logistic function $\sigma(q) = \frac{1}{1+e^{-q}}$ to model this probability conditioned on the model outputs. Thus:

$$P_{z_1 > z_2} = \frac{e^{o_{z_1 z_2}}}{1 + e^{o_{z_1 z_2}}} = \frac{e^{g(\mathbf{x}_{z_1}) - g(\mathbf{x}_{z_2})}}{1 + e^{g(\mathbf{x}_{z_1}) - g(\mathbf{x}_{z_2})}}. \quad (4)$$

   Learning the model involves estimating the parameters $\boldsymbol{\theta}$ of the score function g, such that the model produces a high probability $P_{z_1 > z_2}$ for a pair of samples for which $y_{z_1} > y_{z_2}$, and low probability when $y_{z_1} < y_{z_2}$. This is accomplished by minimizing a cross entropy loss function. In particular, let $Y_{z_1 > z_2} = I(y_{z_1} - y_{z_2})$ be an indicator function which takes on the values in $\{0, 1\}$ depending on whether $y_{z_1} > y_{z_2}$ or not. Then the loss over a single pair of examples is given by:

$$\log(1 + e^{o_{z_1 z_2}}) - Y_{z_1 > z_2} o_{z_1 z_2}. \quad (5)$$

   We used a a multi-layer neural network for the scoring function g in our experiments, and we minimize this loss using stochastic gradient descent.

4. Step 3 is repeated for all the pairs of examples for every user until the error on the validation set stops decreasing. The loss over the entire data set is given by

$$\sum_{z_1, z_2 \in \mathcal{D}} \left(\log(1 + e^{o_{z_1 z_2}}) - Y_{z_1 > z_2} o_{z_1 z_2}\right). \quad (6)$$

5. At test time, given a user-item pair to make a prediction for, we construct the corresponding user-item features, and obtain a prediction by the output of the function g with learned parameters θ. Ranking follows by sorting the predicted scores for each user.

Since we use neural networks for the function g, in a similar manner to the point-wise case, we have the ability to learn a unified model in the pair-wise setting too. We denote this proposal where we additionally learn the factors by $\text{CR}_{\bowtie\text{LF}}$, and the EM style algorithm to fit this model follows an analogous recipe: fix the factors $\mathbf{U}$, $\mathbf{V}$ and estimate $\boldsymbol{\theta}$; then fix $\boldsymbol{\theta}$ and estimate the factors $\mathbf{U}$, $\mathbf{V}$.

3.2.1 A Heuristic for Pair-wise Models

Careful readers would have observed that the complexity of the pair-wise procedures appears quite large; training on pairs of ratings is burdensome, especially when most collaborative filtering datasets have millions of observed ratings. We do point out that we only construct pairs from ratings by the same user, and so for our sparse ratings matrix, the number of such pairs far less than $O(|R|^2)$. However, considering even all ratings pairs from every user is quite challenging because many datasets contain many active/heavy users, who have thousands of ratings each.

Another issue associated with the pair-wise models, which was raised in the LambdaRank paper [3], bears resemblance to the issue with using MSE evaluation metric in the point-wise models. In particular, while training, giving equal emphasis to all the pairs per user might be suboptimal when NDCG is used as an evaluation metric[3]. This is because NDCG cares more about higher rating values, both via the gain function, and the discount function (see Equation 1). In other words, trying to get g to learn an ordering between a pair of items rated '1' and '2' is not very useful. In LambdaRank, the authors address this shortcoming by modifying the gradient update steps while cycling through all (per user) pairs. In particular, they create all possible pairs for every user, but assign weights to each pair, so that the pairs involving higher ratings, such as ('5','4'), are assigned a higher weight than those involving lower ratings, such as ('2', '1'). During the gradient update of the parameters, this results in a larger gradient step (and thus a bigger change in g) for pairs involving higher ratings, and a smaller step for pairs involving lower ratings. In other words, pairs with higher ratings have more influence on g. They also show that empirically, this procedure leads to locally optimal NDCG [7].

We instead dispatch with both of these problems, namely, of large number of pairs and of dealing with non-informative pairs for NDCG, with one simple heuristic. For each user, instead of creating all pairs of items, we only create pairs which have at least one of the top ratings class the user has rated any item with. For instance, if a particular user has used values '5', '4', '3', '2', to rate various items, then for this user we only form pairs which consists of at least one item rated either a '5' or a '4'. The procedure is as follows:

1. For every user we collect all the items which she has rated and sort them according to their ratings. Let us denote this set of items by $\mathcal{S}_j$ for a user j.

2. We pick the highest rated item from $\mathcal{S}_j$, and pair it with all the items in the set which are rated less than the rating of the picked item.

3. We then remove this item from the set $\mathcal{S}_j$.

[3]Note that LambdaRank was a web-search LTR proposal and not about recommendations. We have taken the liberty of translating the ideas in question to our setting for clarity and continuity.

4. We repeat steps 2 and 3 until we have exhausted all the items corresponding to the top 2 rating classes for that user.

The intuition behind this is that the pairs containing the top two ratings classes for every user (say '4's and '5's), are what we need to be concerned with, when we are evaluating NDCG. As pointed out in [5] and [15], in order to have good test NDCG performance, the learned function necessarily need not have perfect regression (or classification) performance. For instance, adding the same constant to every rating score will not affect the NDCG performance. Indeed, what matters are just two things: A. The ability to distinguish reliably between the highly rated items, say, between an item rated '4' and an item rated '5'. B. The ability to accurately distinguish highly rated items from low rated items, say between an item rated a '4' and an item rated a '1'. Our heuristic captures both of these requirements. A. is addressed by making pairs of items with ratings of the form ('5','4'). B. is encoded by only makings pairs of items with at least one high rating, i.e., pairs of the form ('5','1') or ('4', '2'). Since being able to order a ('1', '2') pair is likely not useful at all to test NDCG, we don't create any such pairs with our heuristic.

4. RELATED WORK AND EVALUATION

We now turn to related proposals in the literature. Indeed, not many other researchers have focused on matrix factorization from a ranking perspective. As we mentioned in the introduction, the de-facto standard is to perform matrix factorization using MSE/RMSE. But, before we discuss some of the related proposals, we point out here that evaluating recommenders on the ranking task is complicated by the fact that we have a fixed number of ratings to use both for training the models and evaluating them (as opposed to evaluating a deployed system on live users, for example). We address this issue next.

4.1 Evaluation

In particular, training and experimental evaluation with a single fixed ratings dataset suffers from a rather direct trade-off between having more ratings for training, and thus decreasing the bias in the model predictions, and having more ratings for testing, and decreasing the bias in the evaluation[4]. Our metric of choice, NDCG, is particularly vulnerable to this trade-off.

As an example, consider the standard training validation (probe) split for the Netflix dataset, where the validation dataset consists of about three ratings per user. In this case, after training on the training dataset, we wouldn't even be able to evaluate NDCG@10. As a more relevant example, using some fixed number of randomly sampled ratings, say 20 per user will result in high variance estimates of NDCG@10.

Since we are making the case for collaborative ranking, we prefer more accurate/lower variance evaluations, and thus our preferred train-test split has many more test ratings per user than training ratings. Indeed, this form of experimental setup (large test dataset, smaller training dataset) was also the one favored by CofiRank [20]. Cremonesi et al. [6], pursue a different form of evaluation. In what follows, we briefly outline these two proposals and highlight the similarities and differences compared to our methods.

[4]LTR approaches do not suffer from this issue as they simply evaluate on test/ independent queries

4.2 Cremonesi et al.

In their recent paper, Cremonesi et al. point out the inadequacies of MSE/RMSE as an evaluation metric [6]. Like us, they argue that a top-k based recommendation scheme is a better and more natural fit with how the recommender will finally be used. In their paper, they show that methods trained to perform well on MSE/RMSE, do not behave particularly well on the top-k recommendation task. They further propose straightforward sparse SVD (as SVD requires a full rank matrix, the impute zero values for unseen ratings) as a competitive method for the top-k recommendation task. However, there are significant differences between their work and ours. The biggest difference is in how they evaluate their top-k recommenders. Cremonesi et al. do not consider NDCG; they evaluate using precision/recall on just a small number of high ratings and also a large number of items sampled at random. They settle on this scheme in order to train on a much larger set of ratings. While this is reasonable, we prefer our evaluation using NDCG which avoids random sampling and has very little bias. Another issue that Cremonesi et al. bring up is that popular items seem to heavily sway the performance of algorithms on their precision/recall evaluation metric. Again, we feel this is mainly due to the nature of their evaluation metric; in experiments we do not report, both sparse SVD and top-pop (a non-personalized recommender that simply returns the items sorted by the number of training occurrences) perform non-competitively compared to the methods we evaluate.

4.3 CofiRank

A proposal much closer to our work is CofiRank [20]. The motivation and development of CofiRank is exactly the same as that of our models. They argue for NDCG being a better training/evaluation metric for recommendations, and the experimental protocol we follow is essentially the same as theirs (to allow for comparison). In CofiRank, the authors fit a maximum margin matrix factorization model (matrix factorization with a trace norm regularization on the factors) to minimize various losses including NDCG@k. There are a few very important differences between our proposals. Perhaps the biggest difference lies in our pair-wise and point-wise training loss which is different than CofiRank's proposal to directly minimize training NDCG@k. In the same way that evaluation is difficult in this setting, it is not clear that minimizing training NDCG@k will give a model that performs well on test NDCG@k, particularly when the train and test data are of different sizes. This may be why in the CofiRank paper, the authors report that Cofi with a regression (squared) loss (Cofi-reg) performs competitively/better that Cofi model trained using NDCG@k (Cofi-NDCG). In our experiments, we also consistently found Cofi-reg to be better than Cofi-NDCG. Other minor differences between CofiRank and us have to do with the form regularization used in the matrix factorization model. While Cofi uses MMMF, we use PMF [19], which has a slightly different form of regularization.

5. EXPERIMENTS

We now describe the experiments we performed to evaluate our methods. We evaluate our proposals on several well known collaborative filtering data sets. The data sets are primarily for movie (and TV shows on DVD) recommendation, and are comprised of a collection of ratings given by a set of users to a set of movies. In particular, we used the MovieLens data, and the EachMovie data. These data sets vary drastically in their size and composition. Table 1 gives some basic statistics associated with each one.

Our experimental evaluation closely follows the experimental evaluations in the CofiRank paper. For every user, we randomly sample a fixed number N of ratings and place them in the training set. The remaining ratings by that user are placed in the test set. We experimented with $N = 10$, 20, and 50 training ratings per user. Since we evaluate NDCG@10, we require that users have rated at least 20, 30, and 60 items respectively. Users not meeting these requirements are dropped from both training and test data sets. In addition, we require that for an item to be considered, it should have been rated by at least 5 users in the data set (train+test). Items not meeting this criteria are also dropped. This filtering results in a slight decrease in the number of users and items in the data sets. The original number of users and items are reported in the first two columns of Table 1. For data sets corresponding to different values of N, the average number of users and items after filtering are reported in the last two columns of Table 1. For both the data sets, for different values of N, we generate 10 replicate data sets, each with a different random samples for the training and test set. We report the average NDCG@10 over these 10 replicates for each N. NDCG at other cutoffs gave us different numerical values but qualitative exactly the same results.

5.1 Results

In all our models the parametric scoring function g was a two layer fully connected neural network. The first layer (the hidden layer) was a standard perceptron layer consisting of a linear module followed by a tanh non-linearity applied component-wise to the output of the linear module. The second layer (the output layer) was a fully connected linear layer, which used the output of the hidden layer as input and produced a score. The number of units in the hidden layer were fixed for all the experiments and were set to 400. Since it is a scoring function, the output layer consisted of a single unit. Other hyper parameters associated with the model, such as the learning rates, and the number of training epochs for early stopping were chosen once per data set using a separate training/validation data set sampled with similar characteristics. We did not do further parameter optimization. The results shown are for models learned with these fixed parameters.

As the baseline model, we chose standard Probabilistic Matrix Factorization (PMF) which is fit by minimizing regularized MSE (Equation 2). We also compared our models against CofiRank [20]. We set the factor dimensionality, $d = 50$ in all our experiments. We trained and evaluated the point-wise models $\mathrm{CR}_{\triangleright\mathrm{MF}}$ and $\mathrm{CR}_{\triangleright\mathrm{FL}}$. We also trained and evaluated the pair-wise models $\mathrm{CR}_{\bowtie\mathrm{MF}}$ and $\mathrm{CR}_{\bowtie\mathrm{FL}}$. The features for $\mathrm{CR}_{\triangleright\mathrm{MF}}$ and $\mathrm{CR}_{\bowtie\mathrm{MF}}$ were obtained from the baseline PMF model.

Our results on the MovieLens, and EachMovie data sets are reported in Tables 2 and 3. Overall, the results show some intuitive patterns. All methods get better at making ranking recommendations with increasing number of available training ratings N per user. Furthermore, we see that the baseline results are quite reasonable. In other words, in line with our expectations, minimizing training MSE is not a terrible proxy to train on for providing ranking recommendations. Since training for min MSE is the de-facto standard, our results provide reassurance for using this approach. CofiRank did not do well in our evaluation. We trained all three versions of CofiRank discussed in [20], namely Cofi-NDCG, Cofi-ord (ordinal loss), and Cofi-reg. In our results, we report the performance of the best performing model of the three. We use the untuned hyper-parameter settings as described in [20]. In our experiments we found that in many cases, Cofi-reg was the best method.

However the models we propose in this paper are the clear winners. They provide superior ranking recommendation performance. We show substantial gains in NDCG even when we use the fixed factors obtained by minimizing the MSE, just by using a different yet straightforward predictive model such as $\mathrm{CR}_{\triangleright\mathrm{MF}}$. This validates our hypothesis that the MSE minimization is not optimal for making recommendations. Furthermore, tuning the factors according to the task at hand is critical and can lead to even more improvement. This is evident from the performance of our learned feature models, $\mathrm{CR}_{\triangleright\mathrm{FL}}$ and $\mathrm{CR}_{\bowtie\mathrm{FL}}$. Our experiments do not provide evidence for the superiority of pair-wise or point-wise models. As a practical suggestion, we thus recommend using point-wise models, over their pair-wise counterparts, due to their efficiency and scalability.

6. CONCLUSIONS AND FUTURE WORK

In this paper we argue for collaborative ranking as a better way to make recommendations. Inspired by recent work in the LTR community, we present novel models for the collaborative ranking problem, and their accompanying efficient algorithms. Our experiments show that our proposals are accurate and that learning features for the ranking task produces the best performing models. In future work, we would like to extend our models to richer models like list-wise LTR approaches. Other avenues for future work include analyzing and formalizing our pair-wise heuristic, and improving the scalability of the pair-wise algorithms. Lastly, we plan on extending these results to bigger data sets, such as Netflix, and to data sets from different domains, such as music recommendation, and book recommendations.

7. ACKNOWLEDGMENTS

We would like to acknowledge discussions with Bob Bell, always a source of great ideas and inspiration.

8. REFERENCES

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[2] L. Breiman. Random forests. In *Machine Learning*, volume 45(1), 2001.

[3] C. Burges. From ranknet to lambdarank to lambdamart: An overview. In *Microsoft Research*

Technical Report MSR-TR-2010-82, 2010.

[4] R. Collobert and J. Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 160–167, New York, NY, USA, 2008. ACM.

[5] D. Cossock and T. Zhang. Statistical analysis of bayes optimal subset ranking. *IEEE Transactions on Information Theory*, 54(11):5140–5154, 2008.

[6] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM.

[7] P. Donmez, K. M. Svore, and C. J. Burges. On the local optimality of lambdarank. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 460–467, New York, NY, USA, 2009. ACM.

[8] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003.

[9] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[10] S. Hacker and L. von Ahn. Matchin: eliciting user preferences with an online game. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 1207 – 1216, New York, NY, USA, 2009. ACM.

[11] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In P. J. Bartlett, B. Schölkopf, D. Schuurmans, and A. J. Smola, editors, *Advances in Large Margin Classifiers*, pages 115–132. MIT Press, 2000.

[12] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20:422–446, October 2002.

[13] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM.

[14] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30 – 37, 2009.

[15] P. Li, C. J. C. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting, 2007.

[16] P. Melville and V. Sindhwani. Recommender Systems. 2010.

[17] A. Mohan, Z. Chen, and K. Q. Weinberger. Web-search ranking with initialized gradient boosted regression trees. *Journal of Machine Learning Research, Workshop and Conference Proceedings*, 14:77–89, 2011.

[18] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 521–530, New York, NY, USA, 2007. ACM.

[19] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Neural Information Processing Systems (NIPS)*, 2007.

[20] M. Weimer, A. Karatzoglou, Q. V. Le, and A. Smola. Cofi-rank: Maximum margin matrix factorization for collaborative ranking. In *Neural Information Processing Systems (NIPS)*, 2007.

[21] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 391–398, New York, NY, USA, 2007. ACM.

[22] J. Xu, T.-Y. Liu, M. Lu, H. Li, and W.-Y. Ma. Directly optimizing evaluation measures in learning to rank. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 107–114, New York, NY, USA, 2008. ACM.

Table 1: The table showing the statistics associated with the three data sets which we used in the experiments. The column "Avg. Users (10/20/50)" (and "Avg. Items (10/20/50)") reports the average number of users (and items) in the three data sets generated after randomly sampling 10, 20 and 50 ratings per user as part of the training data.

Data set	No. of Users	No. of Items	No. of Ratings	Ratings Scale	Avg. Users (10/20/50)	Avg. Items (10/20/50)
MovieLens	943	1682	100000	1 – 5	941/743/496	1094/1205/1279
Eachmovie	61265	1623	2811718	0 – 1	36656/28170/14350	1526/1564/1592

Table 2: The table reports NDCG@10 by various models on the MovieLens data set using different training set sizes. The mean and the standard deviation over the 10 folds is reported.

Method	N = 10	N = 20	N = 50
Baseline	0.6916 ± 0.0051	0.7087 ± 0.0031	0.7317 ± 0.0053
CofiRank	0.6994 ± 0.0079	0.7125 ± 0.0031	0.7204 ± 0.0040
$CR_{\triangleright MF}$	0.7026 ± 0.0022	0.7103 ± 0.0021	0.7242 ± 0.0051
$CR_{\triangleright FL}$	$\mathbf{0.7220} \pm 0.0043$	$\mathbf{0.7202} \pm 0.0031$	$\mathbf{0.7316} \pm 0.0070$
$CR_{\bowtie MF}$	0.7023 ± 0.0040	0.7129 ± 0.0033	0.7312 ± 0.0053
$CR_{\bowtie FL}$	$\mathbf{0.7184} \pm 0.0031$	$\mathbf{0.7221} \pm 0.0034$	$\mathbf{0.7360} \pm 0.0058$

Table 3: The table reports NDCG@10 by various models on the EachMovie data set using different training set sizes. The mean and the standard deviation over the 10 folds is reported.

Method	N = 10	N = 20	N = 50
Baseline	0.6950 ± 0.0019	0.7028 ± 0.0006	0.7269 ± 0.0007
CofiRank	0.7109 ± 0.0050	0.7271 ± 0.0074	0.7346 ± 0.0077
$CR_{\triangleright MF}$	0.6959 ± 0.0021	0.7409 ± 0.0013	0.7469 ± 0.0013
$CR_{\triangleright FL}$	$\mathbf{0.7205} \pm 0.0010$	$\mathbf{0.7620} \pm 0.0005$	$\mathbf{0.7593} \pm 0.0013$
$CR_{\bowtie MF}$	0.7038 ± 0.0016	0.7359 ± 0.0007	0.7367 ± 0.0010
$CR_{\bowtie FL}$	$\mathbf{0.7200} \pm 0.0012$	$\mathbf{0.7442} \pm 0.0004$	$\mathbf{0.7503} \pm 0.0006$