Gaussian Ranking by Matrix Factorization

Harald Steck
Netflix Inc.
Los Gatos, California
hsteck@netflix.com

ABSTRACT

The ranking quality at the top of the list is crucial in many real-world applications of recommender systems. In this paper, we present a novel framework that allows for pointwise as well as listwise training with respect to various ranking metrics. This is based on a training objective function where we assume that, for given a user, the recommender system predicts scores for all items that follow approximately a *Gaussian* distribution. We motivate this assumption from the properties of implicit feedback data. As a model, we use matrix factorization and extend it by non-linear activation functions, as customary in the literature of artificial neural networks. In particular, we use non-linear activation functions derived from our Gaussian assumption. Our preliminary experimental results show that this approach is competitive with state-of-the-art methods with respect to optimizing the Area under the ROC curve, while it is particularly effective in optimizing the head of the ranked list.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—Data Mining

Keywords

Recommender Systems; Collaborative Filtering; Matrix Factorization; Learning to Rank

1. INTRODUCTION

In many applications, the recommendation quality at the head of the ranked list of items is crucial. There has been a large body of work on training collaborative filtering approaches as to maximize various ranking metrics, see eg [5, 8, 17, 2, 18, 10, 20, 21, 14, 13, 12]. This is a difficult optimization problem, as recommender systems typically predict a continuous-valued score for each item, based on which the items are then ranked for a given user. The mapping from scores to ranks is not smooth, however, which makes

it hard to optimize ranking metrics directly during training. For this reason, most approaches make smooth approximations to the ranking metric as to enable gradient descent for computationally efficient training. In many cases, a specific training method has been developed for each particular ranking metric, eg [18, 10, 20, 14, 13, 12].

In this paper, we develop an approach for training a recommender system toward various ranking metrics, as chosen by the researcher. We illustrate its flexibility in terms of ranking metrics by optimizing normalized Discounted Cumulative Gain (nDCG), which emphasizes the top of the ranked list, as well as the Area Under the ROC Curve (AUC), which gives equal importance to the ranking of items across the entire list. Moreover, we develop a pointwise variant as well as a listwise one. While the listwise variant requires a more involved training algorithm, we find in our experiments that it yields improved results when optimizing toward the head of the ranked list, while the required computation time is comparable to the one of the pointwise approach.

In our approach, we take advantage of two well-known properties of implicit feedback data, namely that it is typically *binary* and *sparse*. Binary implicit feedback data are ubiquitous in many applications of recommender systems, like clicks on links, purchases of goods, or plays of songs and movies. Moreover, the data are typically extremely sparse, as each user provides feedback on only a small number of items, and no feedback on the vast majority. Usually, the density of the data is around 1% for many publicly available data sets, while it is often orders of magnitudes lower in industrial settings.

Moreover, we specifically focus on the matrix factorization (MF) model (see Section 2). MF has been found to be one of the best single models for collaborative filtering in the literature, see eg [6]. In this paper, we take advantage of the fact that it predicts the score for an item by summing over numerous latent dimensions. This fact, combined with the central limit theorem in statistics, as well as with the properties of implicit feedback data mentioned above, motivated us to approximate the distribution of the predicted scores by a *Gaussian* distribution in this paper (see Section 3). Furthermore, we present several approximations to this Gaussian assumption, which results in different *activation functions*—some of which are also commonly used in artificial neural network models (see Section 4.2). We also present a novel activation function that is particularly efficient and simple, as it is only slightly different from the root mean squared error (RMSE), which is perhaps the most popular accuracy metric in the recommender literature. We

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author(s). Copyright is held by the owner/author(s).

RecSys'15, September 16–20, 2015, Vienna, Austria.

ACM 978-1-4503-3692-5/15/09.

DOI: <http://dx.doi.org/10.1145/2792838.2800185>.

use these activation functions as to map between the scores of the items as predicted by the MF model, and the corresponding ranks of the items. The obtained ranks can then be used in various ranking metrics—we use nDCG and AUC in this paper (see Section 4.1). The resulting training objective function and algorithm are outlined in Section 5. In Section 6, we compare our approach to related work in the literature, and point out its advantages. In our experiments on a publicly available data set as well as on a proprietary one (see Section 7), we find that the presented approach is competitive with state-of-the-art ranking methods regarding AUC, while it is particularly effective in optimizing the head to the ranked list.

In summary, the contributions of this paper are as follows:

- We present a novel training approach for ranking problems, tailored to sparse binary data and the MF model,
- which allows for different ranking metrics like AUC or nDCG.
- This approach can also be understood from the perspective of training neural networks—but with the difference that we use a ranking metric as objective.
- We developed a pointwise as well as listwise variant, where the latter is particularly effective for optimizing the head of the ranked list in our experiments.

2. MATRIX FACTORIZATION MODEL

Matrix factorization (MF) has proven to be one of the most accurate approaches to collaborative filtering in the literature of recommender systems, eg see [6] and references therein. For a given user u , the score $s_{u,i}$ of an item i is determined by the dot product of the latent item and user vectors, $\vec{p}_i$ and $\vec{v}_u$, respectively:

$$s_{u,i} = \vec{p}_i^\top \vec{v}_u = \sum_k p_{i,k} v_{u,k} \quad (1)$$

where $k = \{1, \dots, K\}$ is the index over the latent dimensions. In matrix notation, this reads $\vec{s}_u = P \vec{v}_u$ with score vector $\vec{s}_u = [s_{u,1}, \dots, s_{u,N}]^\top$ and matrix $P = [\vec{p}_1, \dots, \vec{p}_N]^\top$ regarding all items $i \in \{1, \dots, N\}$.

Concerning implicit feedback data, one typically distinguishes between *positive* and *negative* items. *Positive* items are defined as the ones for which the user provided (positive) implicit feedback. In contrast, all other items are typically considered *negative* items. Note that the items in the negative class may be further subdivided into items that the user truly does not like, items that the user may like but did not have time yet to provide feedback on, etc. Even though a single negative class is a simplification, it makes practical approaches feasible, and has been adopted as the common approach in the literature, eg [5, 8, 10, 17]. In the remainder of this paper, the number of positive and negative items of a given user u is denoted by $N_u^{(+)}$ and $N_u^{(-)} = N - N_u^{(+)}$, respectively.

In this paper, we use the asymmetric matrix factorization model (AMF) introduced by [9] and later extended to SVD++ in [6]: the latent user-vector is defined as the sum of the latent item-vectors $\vec{q}_j$ of the positive items j of the user:

$$\vec{v}_u = \frac{1}{\sqrt{N_u^{(+)}}} \sum_{j \in I(u)} \vec{q}_j = Q \vec{h}_u \quad (2)$$

Figure 1: AMF viewed as a NN (see Section 2).

where $I(u)$ is the set of positive items in the feedback history of user u ; we define matrix $Q = [\vec{q}_1, \dots, \vec{q}_N]$ as well as the user's feedback vector $\vec{h}_u$, where $h_{u,j} = 1/\sqrt{N_u^{(+)}}$ if $j \in I(u)$, and 0 otherwise. Compared to an unconstrained latent user vector that has to be learned for each individual user in the basic MF model, the constraint in Eq. 2 results in a considerable reduction in the number of model parameters. Consequently, the parameters underlying the latent user vector $\vec{v}_u$ can be estimated more accurately from the sparse data, where $N_u^{(+)} \ll N$ for a typical user u . An additional advantage is the simplicity of the fold-in, ie computing $\vec{v}_u$ from the user's feedback history $\vec{h}_u$.

As to render notation in this paper independent of a particular MF model, we use θ as to refer to any model parameter, eg the elements in the matrices P and Q . More specifically, $\vec{\theta}_u$ refers to the vector of all parameters that determine the latent user vector $\vec{v}_u$, ie all $q_{j,k}$ where $j \in I(u)$; and $\vec{\theta}_i = \vec{p}_i$ refers to the parameters of item i .

In the remainder of this paper we will regularly refer to ideas from artificial neural networks (NN). Fig. 1 illustrates that AMF can be understood in terms of a NN: the user's feedback vector $\vec{h}_u$ corresponds to the input layer of the NN. The matrix Q maps $\vec{h}_u$ to the hidden layer in the NN, which corresponds to the user vector $\vec{v}_u$ in AMF. Multiplication with matrix P yields the values of the output layer $\vec{s}_u$, which are the scores of the items in AMF. Finally, as we will discuss in the remainder of this paper, a (non-linear) activation function is applied to the scores as to obtain the final output $\vec{r}_u$ of the NN. Note that the data points are indexed by the users u , while the observed dimensions within each data point are indexed by the items i .

3. DEFINITION OF RANKING PROBLEM

This section provides a formal definition of the ranking problem considered in this paper. While we use AMF in this paper, note that the presented learning-to-rank approach is also applicable to other MF models. Our objective is to minimize a rank loss function, like AUC or nDCG, denoted in general by

$$\sum_u L(\vec{y}_u, \vec{s}_u) \quad (3)$$

where the losses L are summed over all users u (we use an unweighted sum for brevity of notation). The class label for negative and positive items is denoted by $y_{u,i} \in \{0, 1\}$, and the corresponding vector regarding all items as $\vec{y}_u$; $\vec{s}_u$ is the vector of predicted scores, as defined above. As to train the MF model, in the remainder of this paper we derive L as a function of the model parameters θ , based on the following assumptions:

1. binary labels: there are only *positive* items i (with label $y = 1$) and negative items (with label $y = 0$), as typical for implicit feedback data.
2. sparsity: for each user, the number of positive items is extremely small compared to the number of negative items: $N^{(+)} \ll N^{(-)}$. In recommender applications, the sparsity is typically $N^{(+)} / N < 1\%$, often by orders of magnitude.
3. for each user u , the distribution of the predicted scores $s_{u,i}$ follows a distribution $\mathcal{P}$. While the empirical estimate of $\mathcal{P}$ may be determined directly from the scores, we assume $\mathcal{P}$ to approximate a Gaussian, ie $\mathcal{N}(0, \text{std}_u^2)$ with zero mean and standard deviation std_u . This assumption is not only a useful working hypothesis in practice, as illustrated in Section 7, but it may also be motivated as follows: due to data sparsity, the vast majority of items belongs to the same (ie negative) class. Hence, the contribution of the positive items to the overall distribution of all items can be ignored in good approximation. Moreover, the score $s_{u,i}$ (as defined in Eq. 1) may be viewed as a sum of scores, ie $s_{u,i} = \sum_k s_{u,i,k}$ over the latent dimensions k , where $s_{u,i,k} = p_{i,k} v_{u,k}$. If the scores $s_{u,i,k}$ regarding different latent dimensions k are viewed as approximately independent random variables, then the central limit theorem implies that their summation (over k) results in scores $s_{u,i}$ that can be expected to be approximately normally distributed.

4. DECOMPOSED LOSS FUNCTION

The rank loss L is typically defined as a function of the ranks $r_{u,i}$, which themselves are determined by the scores $s_{u,i}$,¹ and $s_{u,i}$ are a function of the model parameters θ . As to minimize L by stochastic gradient descent (SGD), its partial derivative w.r.t. each model parameter θ is required. Using the chain rule, its derivative decomposes accordingly (where indices are omitted for brevity):

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial r} \frac{\partial r}{\partial s^{(n)}} \frac{\partial s^{(n)}}{\partial s} \frac{\partial s}{\partial \theta} \quad (4)$$

where we have additionally inserted the normalized score $s^{(n)}$. Comments on each of the four terms are in order:

1. L as a function of the ranks rather than the scores is outlined in Section 4.1 in terms of two popular rank metrics, AUC and nDCG.
2. an item's rank $r_{u,i}$ depends on its normalized score $s_{u,i}^{(n)}$ according to

$$r_{u,i}(s_{u,i}^{(n)}) = \max \left\{ 1, N \cdot \left(1 - \text{CDF}(s_{u,i}^{(n)}) \right) \right\}, \quad (5)$$

where N is the number of all items, and $\text{CDF}(s) = \int_{-\infty}^s \mathcal{P}(x) dx$ is the cumulative density function of the distribution $\mathcal{P}$ of the scores. This equation follows immediately as illustrated in Figure 2. Note that we treat the ranks $r \in [1, N]$ as continuous variables in this approach. For simplicity, we assume that $\mathcal{P}$ is a normalized Gaussian distribution, see Assumption 3 above, so that CDF is the *probit* function.

¹Ranks are determined by sorting in descending order, ie the highest score results in the first rank.

Figure 2: Assuming a Gaussian distribution of the scores, the Gaussian CDF (black line) provides the mapping from score s to rank r via the 'normalized reverse' rank ρ . Also shown are approximations by the logistic function (blue dash-dotted line) and the piecewise quadratic function of Eq. 13 (red dashed).

3. the normalized score depends on the (unnormalized) score $s_{u,i}$ according to $s_{u,i}^{(n)} = (s_{u,i} - \mu_u) / \text{std}_u$, where μ_u and std_u are the mean and standard deviation of the scores $s_{u,i}$ across all items i for a given user u . For simplicity, we assume that std_u does not change (noticeably) during a gradient descent update. Hence, we have $\partial s^{(n)} / \partial s = 1 / \text{std}_u$. Moreover, we assume that std_u is the same for each user in the pointwise approach—this assumption is relaxed in Section 4.3.
4. the score $s_{u,i}$ depends on the model parameters $\vec{\theta}$; eg for $s_{u,i} = \vec{p}_i^\top \vec{v}_u$, we have $\partial s_{u,i} / \partial p_{i,k} = v_{u,k}$, see Eq. 1.

The first two terms of the chain rule in Eq. 4 are interesting and will be discussed in the following sections.

4.1 Rank-Loss Functions

This section discusses the rank loss L as a function of the ranks $r_{u,i}$, regarding the first term in Eq. 4. In the following, we consider the loss for a given user, and hence drop the index u in the notation. Note that the loss L is minimized by minimizing the *negative* of the following rank metrics. Here we outline the equations for AUC and nDCG, which—like many rank metrics—depend explicitly only on the ranks $r^{(+)}$ of the positive items, while the negative items enter only indirectly by affecting the ranks of the positives. Also other rank-loss functions may be used in this approach as long as they are differentiable with respect to the rank, when the rank is considered a continuous variable. Loss functions that *cannot* be used in this framework, are for instance, recall@ n , precision@ n etc, as they are not smooth at rank n .

4.1.1 Area under the Curve

The Area Under the ROC Curve (AUC) is commonly used for evaluating the ranking of the *entire* list of items (rather than the head of the ranked list). It can be defined as

$$\text{AUC} = \frac{1}{N^{(+)} N^{(-)}} \sum_{i=1}^{N^{(+)}} \sum_{j=1}^{N^{(-)}} \mathbf{I}(s_i^{(+)} > s_j^{(-)}) \quad (6)$$

with the indicator function $\mathbf{I}(\cdot) = 1$ if its argument is true, and 0 otherwise; $N^{(+)}$ and $N^{(-)}$ are the number of positive and negative items, respectively, and $N^{(+)} + N^{(-)} = N$. AUC determines the fraction of times when the score $s_i^{(+)}$ of a positive item i is correctly predicted to be larger than the score $s_j^{(-)}$ of a negative item j . AUC is equivalent to the Wilcoxon-Mann-Whitney statistic and to the fraction of concordant pairs. Replacing the indicator function by a smooth function immediately leads to a popular training method, Bayesian Personalized Ranking (BPR) [10].

Instead of evaluating $N^{(+)} \cdot N^{(-)}$ pairwise comparisons, however, it can be computationally more efficient to sort the N scores in time $\mathcal{O}(N \log N)$, and then use the ranks $r_i^{(+)}$ of the positive items:

$$\text{AUC} = \frac{1}{N^{(+)}N^{(-)}} \left[(N+1)N^{(+)} - \binom{N^{(+)}+1}{2} - \sum_{i=1}^{N^{(+)}} r_i^{(+)} \right] \quad (7)$$

It is easy to see the equivalence of Eqs. 6 and 7, eg see [3]. AUC is hence linear in the ranks $r_i^{(+)}$; the derivative of Eq. 7 is hence a constant for each positive item i ,

$$\frac{\partial L}{\partial r_i^{(+)}} = -\frac{\partial \text{AUC}}{\partial r_i^{(+)}} = \frac{1}{N^{(+)}N^{(-)}}. \quad (8)$$

4.1.2 Normalized Discounted Cumulative Gain

The normalized Discounted Cumulative Gain (nDCG) emphasizes the head of the list. It is defined as

$$\text{nDCG} = \frac{DCG}{DCG^{(\text{opt})}}, \quad (9)$$

where $DCG^{(\text{opt})}$ is determined by the optimal ranking. While DCG can deal with graded relevance, it reads

$$\text{DCG} = \sum_{i=1}^{N^{(+)}} \frac{1}{\log(r_i^{(+)} + 1)} \quad (10)$$

in the binary setting if negative items have zero relevance. The derivative of Eq. 9, which is needed in Eq. 4, reads

$$\frac{\partial L}{\partial r_i^{(+)}} = -\frac{\partial \text{nDCG}}{\partial r_i^{(+)}} = \frac{1}{DCG^{(\text{opt})}} \frac{1}{(r_i^{(+)} + 1) \log^2(r_i^{(+)} + 1)}. \quad (11)$$

4.2 Activation Functions

This section is concerned with the second term in Eq. 4, ie $\partial r / \partial s^{(n)}$. Eq. 5 provides the link between scores and ranks for a general CDF. The two most important special cases are: (1) if the scores follow a *uniform* distribution (on a finite interval), then the ranks are a *linear* function of the scores. This justifies the hinge loss of the scores as to optimize AUC, as discussed in more detail in [16]. In case (2), the scores $s_{u,i}$ (approximately) follow a *Gaussian* distribution. This was motivated in the third item in Section 3. In this case, the probit function provides the link between the scores and the ranks. It is well known that the probit is well approximated by the logistic or sigmoid function

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad (12)$$

with appropriate scaling: $\text{probit}(s) \approx \sigma(s\sqrt{8/\pi})$, see eg [1]. Moreover, a computationally efficient approximation can be obtained by a piecewise quadratic function,

$$\text{quad}(x) = \begin{cases} 0 & \text{if } x \leq -1 \\ (1+x)^2/2 & \text{if } -1 < x \leq 0 \\ 1 - (1-x)^2/2 & \text{if } 0 < x < 1 \\ 1 & \text{if } 1 \leq x \end{cases} \quad (13)$$

when using the scaling: $\text{probit}(s) \approx \text{quad}(s/\sqrt{6})$. These approximations are illustrated in Fig. 2.

An even simpler approximation that we found to work well in practice, is obtained when we use the fact that among the positive items only those with a positive score matter (ie toward the top of the ranked list), while the (few) ones with negative scores can be ignored (ie toward the bottom of the ranked list): it is the *rank squared error*

$$\text{rankSE}(x) = [1 - ([1 - x]_+)^2]_+, \quad (14)$$

where $[x]_+ = x$ if $x > 0$, and 0 otherwise. It is easy to see that $\text{quad}(x) = \text{rankSE}(x)$ for the relevant values, ie for $x > 0$.

4.3 Listwise Ranking

In this section, we consider listwise ranking, ie we take into account properties of the entire list of items for a given user. First, this allows us to overcome the assumption that the distribution of the scores has a fixed standard deviation $\text{std}_u = \text{const}$, independent of the user. Now we allow for different values of std_u for different users u . This can efficiently be estimated from the empirical distribution of the scores $s_{u,i}$ for a given user u .

Second, as discussed in Section 4.1, many rank losses depend explicitly on the ranks of only the *positive* items. Positive items typically have large scores and hence lie in the tail of the distribution of all the scores (of positive and negative items). Due to the sparsity in the tail, the mapping by means of the activation function may result in noisy estimates of their ranks as a function of their scores. In addition, the scores cannot be expected to follow exactly a Gaussian distribution in practice (as it is a simplifying assumption after all). For these two reasons, in the listwise variant, we simply sort all the items by their scores as to directly obtain their exact ranks, which are denoted by $\hat{r}_{u,i}$. Then, in Eq. 4, the derivative of the rank loss $\partial L(r)/\partial r$ can be evaluated immediately at $r = \hat{r}_{u,i}$. The second term in Eq. 4, ie $\partial r / \partial s^{(n)}$, can be approximated by mapping the rank $\hat{r}_{u,i}$ back to the corresponding score $\hat{s}_{u,i}^{(n)}$. This is analytically tractable if we assume a logistic activation function σ : given $\hat{r}_{u,i}$, we first compute the normalized reverse rank $\hat{\rho}_{u,i} = 1 - \hat{r}_{u,i}/N$, see Fig 2. Then, given that $\hat{\rho}_{u,i}$ and $\hat{s}_{u,i}^{(n)}$ are linked via $\hat{\rho}_{u,i} = \sigma(\hat{s}_{u,i}^{(n)})$, see Fig. 2, one obtains

$$\frac{\partial r}{\partial \hat{s}^{(n)}} = \sigma'(\hat{s}^{(n)}) = \sigma(\hat{s}^{(n)})(1 - \sigma(\hat{s}^{(n)})) = \hat{\rho}(1 - \hat{\rho}), \quad (15)$$

where the indices are omitted for brevity. Note that this is computationally efficient as no logistic or exponential function has to be computed explicitly.

This is illustrated in Figure 3 when combined with AUC and nDCG: for the choice of AUC, one obtains

$$L'_{\text{AUC}}(\hat{\rho}) := \frac{\partial L}{\partial r} \cdot \frac{\partial r}{\partial \hat{s}^{(n)}} \propto \hat{\rho}(1 - \hat{\rho}). \quad (16)$$

Note that, for the top half of the ranked list ($\hat{\rho} > 0.5$), the gradient $L'(\hat{\rho})$ monotonically *decreases* toward the top of the ranked list (ie toward $\hat{\rho} = 1$). In contrast, as illustrated by the other two curves in Figure 3, which show the behavior for nDCG, the derivative as a function of $\hat{\rho}$ reads

$$L'_{\text{nDCG}}(\hat{\rho}) := \frac{\partial L}{\partial r} \cdot \frac{\partial r}{\partial \hat{s}^{(n)}} \propto \frac{\hat{\rho}(1-\hat{\rho})}{((1-\hat{\rho})N+1) \log^2((1-\hat{\rho})N+1)}. \quad (17)$$

as obtained by combining Eqs. 15 and 11. Note that $(1-\hat{\rho})N = \hat{r} \in \{1, \dots, N\}$. As shown in Figure 3, $L'_{\text{nDCG}}(\hat{\rho})$ *increases* with growing $\hat{\rho}$.

Between these two opposite behaviors (ie decreasing vs increasing derivatives for AUC vs nDCG), the *constant* gradient is a trivial but interesting case. It hence can be expected to optimize the ranking with emphasis that is partially on the head and partially on the entire list. Obviously the corresponding function has to be linear in the scores. The hinge loss and the linear rectifier are two functions that are linear unless they are zero. While the former is used as a loss in support vector machines, the latter is applied as an activation function in deep NN.

There is also another way of obtaining a nearly constant gradient, resulting in a partially head-heavy ranking loss: in our pointwise variant, one may choose an activation function with an associated standard deviation $\text{std}_{\text{activ}}$ that is considerably *larger* than std_u of the actual distribution of the scores. Then the resulting ranking loss is increasingly head-heavy as $\text{std}_{\text{activ}}$ becomes larger compared to std_u . The reason is that, for $\text{std}_{\text{activ}} \gg \text{std}_u$, the gradient decreases more slowly than the one resulting from the proper standard deviation (only if $\text{std}_{\text{activ}} = \text{std}_u$, the mapping from the scores to the ranks is correct in the sense of Eq. 5). In fact, we find this idea to work quite effectively in our experiments in Section 7: we use AUC as the loss function, but choose an activation function with $\text{std}_{\text{activ}} \gg \text{std}_u$ as to render the resulting loss head-heavy. This is indicated by probit^(w), logistic^(w) and rankSE^(w) in Tables 1 and 2, which show that the resulting rankings are competitive toward the head of the ranked list, for the price of being degraded with respect to AUC.

5. TRAINING OBJECTIVE

Having discussed the rank loss L and several of its variants above, we now can add the regularization terms as to obtain the final training objective function to be minimized:

$$\sum_u \sum_{i \in I(u)} \left[L(s_{u,i}) + \lambda \cdot (\|\vec{\theta}_u\|^2 + \|\vec{\theta}_i\|^2) \right] + \sum_{j \in I} \gamma \cdot ([s_{u,j}]_+)^2 \quad (18)$$

The details are as follows. First, in the sum over the set $I(u)$ of positive items, we have the loss function L , which commonly is a function of the positive items only (see Section 4.1), as well as the usual ℓ_2 -regularization terms for the model parameters $\vec{\theta}_u$ and $\vec{\theta}_i$ concerning the user u and items i , ie the parameters $p_{i,k}$ and $q_{i,k}$ with $i \in I(u)$ in AMF. The ℓ_2 -regularization may be viewed as the logarithm of a Gaussian prior distribution with zero mean and variance $1/(2\lambda)$.

Second, the sum over all items $j \in I$ in Eq. 18 enforces the (approximately Gaussian) distribution of the scores $s_{u,j}$ about zero. This reflects our main assumption, as outlined in Section 3, namely that the scores follow approximately

Figure 3: The derivative of the loss $\frac{\partial L}{\partial r} \frac{\partial r}{\partial \hat{s}^{(n)}}$, ie the first two terms in Eq. 4, as a function of the ‘normalized reverse’ rank ρ : as ρ approaches the top of the list ($\rho \approx 1$), the gradient *decreases* for $L = -\text{AUC}$ (solid line), while it *increases* for $L = -\text{nDCG}$, for different lengths N of the ranked list, $N = 10,000$ (dashed line) and 10 million (dash-dotted line). The latter two curves are re-scaled along the y-axis. See Sections 4.3 and 4.1 for details.

the *posterior* distribution $s_{u,j} \sim \mathcal{N}(0, \text{std}_u^2)$. As to obtain such a posterior distribution with the desired variance std_u^2 , one may use a *prior* distribution $s_{u,j} \sim \mathcal{N}(0, 1/(2\gamma))$, where γ is chosen appropriately. The logarithm of this prior immediately yields the additional term $\gamma \cdot s_{u,j}^2$. For simplicity, we have chosen γ to be independent of user u here.

The derivative of Eq. 18 may be interpreted as the force that pulls an item’s score toward the target values 0 or 1 during training. This interpretation immediately originates from the perspective of physics when Eq. 18 is interpreted as an energy. In this picture, γ may be viewed as the degree with which an item’s score is pulled toward *zero*. During training with the feedback data, this is counter-balanced by having the scores of the positive items being pulled toward a *positive* target value (like 1) due to the rank loss L . A proper choice of γ is hence crucial for the obtained results; the value of γ can be determined such that the desired rank metric on a validation set is optimized.

Note that there is no term in Eq. 18 that explicitly tries to pull scores toward a *negative* target value. For this reason, it is not necessary for the ‘prior’ to pull *negative* scores toward zero. In fact, it is beneficial to modify the last term in Eq. 18 from $\gamma \cdot s_{u,i}^2$ to $\gamma \cdot ([s_{u,i}]_+)^2$, where $[x]_+ = x$ if $x > 0$, and 0 otherwise; ie, an item’s score $s_{u,i}$ is pulled toward zero only if it is positive. We found this modification to yield improved ranking results in our experiments.

Eq. 18 can efficiently be minimized by stochastic gradient descent (SGD). The necessary derivatives of the different variants of the rank loss are outlined in the previous sections. For computational efficiency, the sum over all items I in Eq. 18, and hence the iteration at line 11 in Algorithm 1, are sub-sampled, like eg in [8, 17]. Moreover, note that line 16 of Algorithm 1 refers to the negative items, while the last term in Eq. 18 refers to both positive and negative items—this difference is negligible in practice as $N^{(+)} \ll N^{(-)}$. The iterative training algorithm for AMF has the same structure as the one for training SVD++ [6]: for each user, one first

Algorithm 1 SGD for AMF to minimize pointwise loss L .

```
1: Input:: learning rate  $\eta$ , parameters  $\lambda, \gamma$ , training data.
2: Output: latent vectors  $\vec{p}_i, \vec{q}_i$  for all items  $i$ .
3: do:
4:  $\forall i: \vec{p}_i \leftarrow \text{random initialization}$ 
5:  $\forall i: \vec{q}_i \leftarrow \text{random initialization}$ 
6: repeat
7:   for user  $u$  do
8:      $\Lambda \leftarrow 0$ 
9:      $\vec{\epsilon} \leftarrow 0$ 
10:     $\vec{v}_u \leftarrow \sum_{j \in I_u} \vec{q}_j / \sqrt{N_u^{(+)}}$ 
11:    for all items  $i \in I(u)$  do
12:      if  $i \in I(u)$  then
13:         $\text{err} \leftarrow -L'(\vec{p}_i^\top \vec{v}_u)$ 
14:         $\lambda_i \leftarrow \lambda$ 
15:         $\Lambda \leftarrow \Lambda + \lambda$ 
16:      else
17:         $\text{err} \leftarrow -\gamma[\vec{p}_i^\top \vec{v}_u]_+$ 
18:         $\lambda_i \leftarrow 0$ 
19:       $\vec{\epsilon} \leftarrow \vec{\epsilon} + \text{err} \cdot \vec{p}_i / \sqrt{N_u^{(+)}}$ 
20:       $\vec{p}_i \leftarrow \vec{p}_i + \eta \cdot (\text{err} \cdot \vec{v}_u - \lambda_i \cdot \vec{p}_i)$ 
21:    for  $i \in I(u)$  do
22:       $\vec{q}_i \leftarrow \vec{q}_i + \eta \cdot (\vec{\epsilon} - \Lambda \cdot \vec{q}_i)$ 
23: until convergence
```

iterates through all the items i as to update the parameters $\vec{p}_i$, while the updates for the parameters $\vec{q}_i$ are aggregated in the temporary vector $\vec{\epsilon}$ and in the regularization value Λ . After that, in a second iteration at line 21 in Algorithm 1, the parameters $\vec{q}_i$ get updated based on the temporary vector. Algorithm 1 provides the pseudo code for pointwise ranking with AMF, using the derivative of the loss $L' = \frac{\partial L}{\partial r} \frac{\partial r}{\partial s^{(n)}} \frac{\partial s^{(n)}}{\partial s}$, which gets evaluated using score $s_{u,i} = \vec{p}_i^\top \vec{v}_u$, see also Eq. 4. The fourth derivative-term in Eq. 4 is included in lines 19 and 20 in Algorithm 1.

In the listwise training (see Section 4.3), there is an additional iteration over the items as to compute their scores, and then their ranks and the scores' standard deviation. Only after that, the SGD updates in line 11 and following are made analogous to the pointwise approach.

In addition, we use dropout learning [15]. It combines the random subspace method [4] with parameter sharing across the models in the (implicit) ensemble, which results in a reduced generalization error.

6. RELATED WORK

In the recommender literature, many learning-to-rank approaches are *pairwise* methods. The most well-known approach is perhaps BPR [10], which optimizes AUC. As to optimize head-heavy rank metrics, custom-tailored approaches to optimize mean-reciprocal-rank (MRR) and mean-average-precision (MAP) were presented in [13, 12]. Moreover, weighting schemes [19, 20] and sampling schemes for the negative items [21] were proposed, and may be viewed as extensions to BPR.

In contrast, the presented approach considers all items concerning a user u at once, rather than each user-item pair (u, i) separately. If we assume that the number of sampled negative items is the same as the number of positive items, ie $N_u^{(+)}$ for user u , then all the SGD updates for a

user u can be computed by first sorting the scores in time $\mathcal{O}(N_u^{(+)} \log(N_u^{(+)})$) in the listwise approach (see Section 4.3). Note that this accounts for all pairwise comparisons between positive and negative items. In the pairwise approach, this would require much more time, ie $\mathcal{O}((N_u^{(+)})^2)$.

While minimizing RMSE on the observed ratings is associated with the 'rating prediction' problem in the recommender literature, minimizing the squared error with proper sampling and weighting of negatives [5, 8, 17] was also successfully used for ranking problems, and outperformed other ranking methods, like eg [2]. The presented approach essentially replaces the RMSE objective with a rank loss.

A listwise approach to training recommender systems has been proposed in [14]. It was motivated from top-1 recommendations, and was experimentally shown to outperform Cofi-rank [18] regarding nDCG for the top items. This approach uses the *softmax* activation function and the *cross entropy* as loss function. Interestingly, the same approach is also popular for training artificial neural networks. We included this approach as an additional baseline in our experiments.

7. EXPERIMENTS

This section outlines our results on the 10 million MovieLens data [7] and our proprietary Netflix data.

7.1 MovieLens Data

In the 10 million MovieLens data [7], 69,878 users have rated 10,677 movies. As to simulate binary 'play' data for use in our approach, we binarized the MovieLens data by retaining all ratings of 3 or higher as positives (which may indicate some user satisfaction), while discarding the lower ratings. This resulted in a data set of 8.2 million plays, with a sparsity of about 1%. All other user-movie pairs are considered as 'negative', as defined in Section 2. We split this data set into a training set (95%) and a disjoint test set (5%). We repeated this random split five times, and report mean and standard deviation. Given that the disjoint test set does not contain any movies played by the user in the training set, those movies are filtered out from the user's recommendations. We used AMF with $K = 50$ latent dimensions.

We evaluated the ranked list of items with respect to AUC and nDCG. Besides nDCG, there are various ranking metrics that summarize the ranking quality with a certain emphasis on the top of the list in a *single* number, like mean average precision (MAP), mean reciprocal rank (MRR), etc. Instead, we provide results w.r.t. $\text{recall}@n$ for several n , which provides detailed insights in the ranking quality at different positions in the ranked list. In the interest of space, we do not report $\text{precision}@n$, as it is proportional to $\text{recall}@n$ for the same test data and the same value n for each user, see eg [17].

When training the AMF model, we re-scaled the inputs of the activation functions such that their widths agreed with rankSE in Eq. 14 (rather than with the probit function)—this allowed as to use the target values 1 (instead of $\sqrt{6}$) and 0 for the positive and negative items in the pointwise approach, respectively. In the training objective function in Eq. 18 we used the parameters $\lambda = 10^{-7}$ and $\gamma = 0.3$ for this data set for the pointwise variants, as determined by a grid search. The step size in SGD was $\eta = 0.001$. Choosing

Table 1: Ranking results on the MovieLens data [7].² Based on five repetitions of the experiments, the standard deviation (std) is ≤ 0.002 for recall and nDCG, and ≤ 0.001 for AUC. Values within std of the best result are highlighted.

trained	loss	activ. fct.	recall@ 10	recall@ 30	recall@ 80	recall@ 500	recall@ 3000	AUC	nDCG
point-wise	'AUC'	probit ^(w)	0.265	0.432	0.602	0.883	0.986	0.975	0.426
		logistic ^(w)	0.266	0.432	0.602	0.883	0.986	0.976	0.427
		rankSE ^(w)	0.265	0.431	0.600	0.881	0.986	0.975	0.427
list-wise	nDCG	logistic	0.272	0.433	0.592	0.820	0.891	0.901	0.421
	AUC	logistic	0.217	0.386	0.564	0.876	0.993	0.978	0.386
point-w.	RMSE	linear	0.250	0.410	0.579	0.867	0.988	0.974	0.413
list-w.	X ent.	softmax	0.256	0.420	0.589	0.872	0.989	0.975	0.417
pair-w.	AUC by BPR		0.241	0.404	0.577	0.877	0.992	0.979	0.408

that large a value for γ leads to an increased emphasis of the ranking quality at the head of the ranked list, as discussed at the end of Section 4.3. For this reason, we use probit^(w), logistic^(w) and rankSE^(w) in Table 1, as to indicate that the AUC rank loss is combined with a *wide* activation function. Table 1 reflects this fact clearly, as 'AUC' with all three *wide* activation functions obtains high values with respect to recall toward the top as well as regarding nDCG. Moreover, these three variants of wide activation functions lead to very similar results, as expected from Figure 2.

The listwise variants of the proposed approach adapt the activation function to the width of the distribution of the scores of each user, ie $\text{std}_{\text{activ}} = \text{std}_u$. Consequently, the chosen rank loss, nDCG or AUC, determines whether the ranking is optimized regarding the head of the ranked list or concerning the entire list. Table 1 illustrates this flexibility of the presented approach, as either recall @ 10 and 30 is optimized, or AUC, depending on the chosen rank loss. Interestingly, Table 1 also shows that the presented approach trained with nDCG loss is much more head-heavy than expected: while it achieves high recall @ 10 and 30, the ranking quality deteriorates very quickly below the head of the ranked list, reflected by an extremely low AUC. This also degrades nDCG on the entire list. While not fully clear from the theoretical side, this behavior is actually desirable in real-world applications of recommender systems, where only a very limited number of items can be shown to the user due to space constraints.

7.2 Baseline Approaches

Table 1 also shows the results of three competing approaches for comparison, whose hyper-paramters were also optimized by grid search.² As to optimize the root mean squared error (RMSE)² on binary data we followed the ideas outlined in [5, 8, 17], ie we randomly sampled negative items in addition to using the positive items in the implicit feedback data. This is a crucial change to the popular approach used for 'rating prediction', eg [11, 6]. As a result, this approach is geared toward optimizing the ranking, and was found to outperform other ranking methods, like eg [2]. Table 1 shows that optimizing RMSE with sampled negative items yields solid ranking results.

BPR² [10] optimizes AUC in Table 1, as expected. Not surprisingly, as the head's contribution to the ranking of the entire list is small, the ranking at the top is rather unim-

portant, and consequently the head-heavy metrics are lower than for RMSE optimization.

As already outlined in Section 6, the approach that optimizes the cross-entropy and uses the softmax activation function² in Table 1 was motivated in [14] to optimize top-1 recommendations, and it was also shown to outperform Cofi-rank [18] concerning nDCG at the top. In Table 1 regarding the top of the ranked list, however, it only improves over the variants optimized toward AUC and RMSE, while it is outperformed by all our pointwise and listwise variants that optimize the head.

7.3 Netflix Data

The results on our proprietary data agree with the insights from the MovieLens data above. We considered all movies and TV shows available in our US catalog (for simplicity, we ignored videos available on Netflix outside the US, as a global model that can deal with different availabilities of videos in different countries is beyond the scope of this paper). The split into disjoint training and test data was performed based on the time-stamp when a video was played, as to simulate the situation of the real-world system: we randomly picked a test-day in 2014, and the preceding 6 months as the training period. We used binary data that captured whether a user played a video or not during the training or test periods, respectively. If a user played the same video in the training and the test data, we removed it from the test data. We sampled a random subset of about 500,000 anonymized users. This was repeated five times, and we report the mean and standard deviation (std) of the test-metrics.

Table 2 depicts the results, where we focus on the very top of the recommended list, as this is crucial in the real-world application. We also provide AUC regarding the entire list as additional information. The hyper-parameters in the training objective were again determined by grid search for each model. The results are relative to the baseline model, for which we picked the one that optimizes RMSE regarding positive items combined with randomly sampled negative items, as described in the previous section. The proposed pointwise variants with a wide activation function, as well as the listwise optimization of nDCG again perform well at the top of the list. Regarding the AUC metric, BPR as well as our listwise optimization of AUC (and the RMSE baseline model itself) achieve the highest AUC (within the noise level).

²For fair comparisons, we also used the AMF (instead of the standard MF) model for all competing approaches.

Table 2: Ranking results on our proprietary Netflix data set, as a relative improvement over training toward RMSE with proper sampling of negative items.² The uncertainty (std) is 2 % for recall and 1% for AUC (percentage points). Values within std of the best result are highlighted.

loss fct.	activ. fct.	recall @				AUC
		1	5	10	30	
'AUC'	probit ^(w)	28%	16%	9%	2%	-4%
	logistic ^(w)	30%	15%	7%	-1%	-13%
	rankSE ^(w)	28%	16%	9%	1%	-4%
nDCG	logistic	32%	17%	9%	2%	-5%
AUC	logistic	24%	9%	2%	-3%	0%
RMSE	linear	--	--	--	--	--
X ent.	softmax	28%	14%	8%	0%	-1%
AUC by BPR		8%	-1%	-2%	-3%	1%

8. CONCLUSIONS

In this paper, we presented a novel learning-to-rank approach that takes advantage of two typical properties of implicit feedback data, namely that they are sparse and binary. This motivated us to make the approximation that the scores of the items, as predicted by a matrix factorization model, follow approximately a Gaussian distribution. Based on this assumption, we derived several non-linear activation functions. This enabled us to optimize toward different ranking metrics efficiently. In our experiments, we found that the presented approach is competitive with respect to the Area under the Curve, while it is particularly effective for optimizing the head of the ranked list.

Given that the used asymmetric matrix factorization model can be viewed as a shallow neural network, future work includes the extension of the presented learning-to-rank approach to deep neural networks.

9. ACKNOWLEDGMENTS

I would like to thank Caitlin Smallwood, Carlos Gomez-Urbe and Roelof van Zwol for their encouragement and support of this work.

10. REFERENCES

- [1] C. Bishop. *Neural networks for pattern recognition*. Oxford, 1995.
- [2] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-N recommendation tasks. In *ACM Conference on Recommender Systems*, pages 39–46, 2010.
- [3] D. J. Hand and R. J. Till. A simple generalization of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45:171–86, 2001.
- [4] T. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Learning*, 20:832–44, 1998.
- [5] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *IEEE International Conference on Data Mining (ICDM)*, 2008.
- [6] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 426–34, 2008.
- [7] MovieLens. Univ. of Minnesota, homepage: <http://grouplens.org/datasets/movielens/>.
- [8] R. Pan, Y. Zhou, B. Cao, N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *IEEE International Conference on Data Mining (ICDM)*, 2008.
- [9] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *KDD Cup*, 2007.
- [10] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 452–61, 2009.
- [11] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.
- [12] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, A. Hanjalic, and N. Oliver. TFMAR: optimizing MAP for top-n context-aware recommendation. In *ACM SIGIR conference*, 2012.
- [13] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. CLiMF: Learning to maximize reciprocal rank with collaborative less-is-more filtering. In *ACM Conference on Recommender Systems (RecSys)*, 2012.
- [14] Y. Shi, M. Larson, and A. Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *ACM Conference on Recommender Systems (RecSys)*, 2010.
- [15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–58, 2014.
- [16] H. Steck. Hinge rank loss and the area under the ROC curve. In *Proceedings of the European Conference on Machine Learning (ECML)*, 2007.
- [17] H. Steck. Training and testing of recommender systems on data missing not at random. In *ACM Conference on Knowledge Discovery and Data Mining*, pages 713–22, 2010.
- [18] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. Cofi rank–maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.
- [19] J. Weston, H. Yee, and R. Weiss. WSABIE: Scaling up to large vocabulary image annotation. In *Int. Joint Conference on Artificial Intelligence (IJCAI)*, 2011.
- [20] J. Weston, H. Yee, and R. Weiss. Learning to rank recommendations with the k-order statistic loss. In *ACM Conference on Recommender Systems (RecSys)*, 2013.
- [21] W. Zhang, T. Chen, J. Wang, and Y. Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *ACM SIGIR Conference*, 2013.