Learning to Rank Recommendations with the k-Order Statistic Loss

[Extended Abstract]

Jason Weston
Google Inc.
76 9th Avenue, New York
New York, NY 10011 USA
jweston@google.com

Hector Yee
Google Inc.
901 Cherry Avenue,
San Bruno, CA 94066 USA
hyee@google.com

Ron J. Weiss
Google Inc.
76 9th Avenue, New York
New York, NY 10011 USA
ronw@google.com

ABSTRACT

Making recommendations by learning to rank is becoming an increasingly studied area. Approaches that use stochastic gradient descent scale well to large collaborative filtering datasets, and it has been shown how to approximately optimize the mean rank, or more recently the top of the ranked list. In this work we present a family of loss functions, the k-order statistic loss, that includes these previous approaches as special cases, and also derives new ones that we show to be useful. In particular, we present (i) a new variant that more accurately optimizes precision at k, and (ii) a novel procedure of optimizing the mean maximum rank, which we hypothesize is useful to more accurately cover all of the user's tastes. The general approach works by sampling N positive items, ordering them by the score assigned by the model, and then weighting the example as a function of this ordered set. Our approach is studied in two real-world systems, Google Music and YouTube video recommendations, where we obtain improvements for computable metrics, and in the YouTube case, increased user click through and watch duration when deployed live on www.youtube.com.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

learning to rank, loss functions, stochastic gradient, collaborative filtering, matrix factorization

1. INTRODUCTION

While low-rank factorizations have been a standard tool for recommendation for a number of years [2] optimizing them using a ranking criterion is a relatively recent and increasingly popular trend amongst researchers and practicioners alike. Methods like CofiRank [7], CLiMF [5], or Wsabie [9] all learn, in some manner, to rank items that a user prefers at the top of the ranked list of items. Such loss functions are natural because the end product of such systems is usually to suggest a few recommendations to the user by choosing the top scoring (top ranked) items that the model predicts. However, in contrast to methods like SVD, where the user-item rating matrix is factorized in a least-squares sense, the optimization is not always straightforward. For example, it is difficult to optimize a metric like precision@k directly by gradient descent due to the discontinuities introduced by ranking. That is, when two items at the top of the list switch rank, the objective function score can change drastically. On the other hand, when two items at the bottom of the list switch rank there is no change in the objective at all. Researchers have found many approximations to the ranking functions they would like to optimize that are amenable to gradient descent. Note that several methods in the information retrieval domain have been proposed, e.g. [10, 11], but they have mostly focused on linear models rather than factor models. Those methods are not often applicable to recommendation tasks which typically involve thousands or millions of users and thousands or millions of items to rank, thus direct modeling of the full rank user-item matrix is infeasible. Hence, factored ranking models are desirable.

Due to the size of recommendation datasets learning via stochastic gradient descent (SGD) training is an attractive option, however again not all loss functions are easily optimized in this manner. For example the ListNet [10], SVM$_{map}$ [11] or OWPC (ordered weighted pairwise classification) [6] objectives involve computing the rank of positive examples which is too slow in an SGD step when there are hundreds of thousands or millions of items. Two choices of ranking loss that can be trained via SGD are the AUC (area under the curve) [4, 3] and WARP (weighted approximately ranked pairwise) [**?**] losses. However, both methods ignore the fact that there are multiple positive items per user and treat those items independently, clearly an incorrect assumption.

In this work we propose a new class of loss functions, the k-order statistic loss, which generalizes the existing AUC and WARP methods, as well as providing novel choices of loss, by taking into account the *set* of positive examples during the gradient steps. In particular, it can more accuratel optimize precision at k than the WARP loss, which WARP was designed for. Secondly, it can optimize novel metrics like maximum mean rank which we believe are useful for accurately covering all of the user's tastes when recommending

RecSys ACM RECSYS 2013 (OCT 12-16, 2013)

items. Experiments on real-world datasets indicate the usefulness of our approach.

2. K-ORDER STATISTIC LOSS

We consider the general recommendation task of ranking a set of items $\mathcal{D}$ for a given user, the returned list should have the most relevant items at the top. To solve this task, we are given a training set of users $\mathcal{U}$ each with a set of known ratings. We consider the case where each user has purchased / watched / liked a set of items, which are considered as positive ratings. No negative ratings are given. All non-positive rated items are thus considered as having an unknown rating[1]. We define the set $\mathcal{D}_u$ to be the positive items for user u. We consider factorized models of the form

$$f_d(u) = \frac{1}{|\mathcal{D}_u|} \sum_{i \in \mathcal{D}_u} V_i^\top V_d,$$

where V, which is an $m \times |\mathcal{D}|$ matrix, one vector for each item, contains the parameters to be learnt. We can further define $f(u)$ to be the vector of all item scores $1, \ldots, |\mathcal{D}|$ for the user u. To learn f one typically minimizes an objective function of the following form:

$$\sum_{u=1}^{|\mathcal{U}|} L(f(u), \mathcal{D}_u)$$

where L is the loss function, which measures the discrepency between the known ratings $\mathcal{D}_u$ and the predictions for user u. The well-known AUC loss (sometimes known as the margin ranking loss) [4, 3] is defined as:

$$L_{AUC}(f(u), \mathcal{D}_u) = \sum_{d \in \mathcal{D}_u} \sum_{\bar{d} \in \mathcal{D} \setminus \mathcal{D}_u} \max\Big(0, 1 - f_d(u) + f_{\bar{d}}(u)\Big).$$

To optimize it by stochastic gradient descent, one selects a user, a positive item and a negative item at random, and makes a gradient step, corresponding to one term in the double sum of the equation above. Repeated updates gradually visit all the terms.

The AUC loss is known to not optimize well the top of the rank list. Another set of loss functions called the OWPC loss [6] and its SGD counterpart, WARP loss [**?**], attempt to focus on the top of the list. The loss is defined as:

$$L_{WARP}(f(u), \mathcal{D}_u)) = \sum_{d \in \mathcal{D}_u} \Phi\Big(rank_d(f(u))\Big) \qquad (1)$$

where $\Phi(\eta)$ converts the rank of a positive item d to a weight. Here, the rank of d is defined as

$$rank_d(u) = \sum_{\bar{d} \notin \mathcal{D}_u} I\Big(f_d(u) \leq 1 + f_{\bar{d}}(u)\Big), \qquad (2)$$

where I is the indicator function.

Choosing $\Phi(\eta) = C$ for any positive constant C is equivalent to the AUC loss. However, a weighting such as $\Phi(\eta) = \sum_{i=1}^{\eta} 1/i$ pays more attention to optimizing the top of the ranked list. Unfortunately, training such an objective by SGD directly is not tractable as eq. 2 sums over all items, which is too slow to compute per gradient update. The WARP loss [**?**] was proposed to solve this problem. Using WARP, the $rank_d(u)$ is replaced with a sampled approximation: sample N items $\bar{d}$ until a violation is found, i.e. $f_u(d) < 1 + f_{\bar{d}}(u)$ and then approximate the rank with $|\mathcal{D} \setminus \mathcal{D}_u|/N$.

While OWPC/WARP provides a generalized class of loss functions including AUC as a special case, note that it still treats each positive item independently via the sum in eq. 1. In contrast, many evaluation metrics that we are interested in do not treat positive examples in this way. For example, precision at 1 only cares if one of the positives is at the top of the ranked list, and does not care about the position of the others. We thus generalize the above loss functions by proposing the k-Order Statistic (k-OS) loss as follows. For a given user u, let o be the vector of indices indicating the order of the positive examples in the ranked list:

$$f_{\mathcal{D}_{U\,o_1}}(u) > f_{\mathcal{D}_{U\,o_2}}(u) > \cdots > f_{\mathcal{D}_{U\,o_{|s|}}}(u).$$

[1]The binary rating case described is rather common in many real world recommendation tasks, especially for those where ratings are harvested from implicit feeback.

Algorithm 1 K-OS algorithm for picking a positive item.

We are given a probability distribution P of drawing the i^{th} position in a list of size K. This defines the choice of loss function.
Pick a user u at random from the training set.
Pick $i = 1, \ldots, K$ positive items $d_i \in \mathcal{D}_u$.
Compute $f_{d_i}(u)$ for each i.
Sort the scores by descending order, let $o(j)$ be the index into d that is in position j in the list.
Pick a position $k \in 1, \ldots, K$ using the distribution P.
Perform a learning step using the positive item $d_{o(k)}$.

Algorithm 2 K-OS WARP loss

Initialize model parameters (mean 0, std. deviation $\frac{1}{\sqrt{d}}$).
repeat
  Pick a positive item d using Algorithm 1.
  Set $N = 0$.
  repeat
    Pick a random item $\bar{d} \in \mathcal{D} \setminus \mathcal{D}_u$.
    $N = N + 1$.
  until $f_{\bar{d}}(u) > f_d(u) - 1$ or $N \geq |\mathcal{D} \setminus \mathcal{D}_u|$
  if $f_{\bar{d}}(y) > f_d(u) - 1$ **then**
    Make a gradient step to minimize:
      $\Phi(\frac{|\mathcal{D} \setminus \mathcal{D}_u|}{N}) \max(0, 1 + f_{\bar{d}}(u) - f_d(u))$.
    Project weights to enforce constraints, e.g. if $||V_i|| > C$ then set $V_i \leftarrow (CV_i)/||V_i||$.
  end if
until validation error does not improve.

Algorithm 3 K-OS AUC loss

Initialize model parameters (mean 0, std. deviation $\frac{1}{\sqrt{d}}$).
repeat
  Pick a positive item d using Algorithm 1.
  Pick a random item $\bar{d} \in \mathcal{D} \setminus \mathcal{D}_u$.
  if $f_{\bar{d}}(u) > f_d(u) - 1$ **then**
    Make a gradient step to minimize:
      $\max(0, 1 + f_{\bar{d}}(u) - f_d(u))$.
    Project weights to enforce constraints, e.g. if $||V_i|| > C$ then set $V_i \leftarrow (CV_i)/||V_i||$.
  end if
until validation error does not improve.

The k-OS loss is then defined as:

$$L_{K\text{-OS}}(f(u), \mathcal{D}_u)) = \frac{1}{Z} \sum_{i=1}^{|\mathcal{D}_u|} P\Big(\frac{i}{|\mathcal{D}_u|}\Big) \Phi\Big(rank_{\mathcal{D}_u o_i}(f(u))\Big)$$

where $Z = \sum_i P\Big(\frac{i}{|\mathcal{D}_u|}\Big)$ normalizes the weights induced by P. $P(\frac{j}{100})$ is the weight assigned to the j^{th} percentile of the ordered positive items. Different choices of P result in different loss functions. $P(j) = C$ for all j and any positive constant C results in the original WARP or AUC formulations. Choices where $P(i) > P(j)$ for $i < j$ result in paying more attention to positive items that are at the top of the ranked list, and tends to ignore the lower ranked positives. This should have the effect of improving precision and recall at the top whilst sacrificing some of the user's taste preferences. Conversely, choosing $P(i) < P(j)$ for $i < j$ should focus more on improving the worst ranked positives in the user's rating set. We hypothesize that this may more accurately cover all of the user's tastes, and try to measure this in our experiments using the mean maximum rank metric.

To optimize k-OS easily via SGD we make the following simplification. During each SGD step we draw, for a random user, K random positives and order them by $f(u)$. Then the P distribution only takes on K possible values. The overall method is detailed in Algorithms 1, 2 and 3, for both AUC and WARP generalizations. In the majority of our experiments we use $P(j) = 1$ if $j = k/N$, and 0 otherwise, and leave k as a hyperparemater. That is, we simply always select the positive in the k^{th} position in the list.

3. EXPERIMENTS

We conducted experiments on three large scale, real world tasks: artist recommendation and track recommendation using proprietary data from Google Play Music (`http://music.google.com`), and video recommendation from YouTube (`http://www.youtube.com`). In all cases, the datasets consist of a large set of anonymized users, where for each user there is a set of associated items based on their watch/listen history. The user-item matrix is hence a sparse binary matrix. The approximate dataset sizes are given in Table 1.

To construct evaluation data, we randomly selected 5 items for testing per user, and kept them apart from training. At prediction time for the set of test users we then ranked all unrated items (i.e. items that they have not watched/listened to that are present in the training set) and observe where the 5 test items are in the ranked list of recommendations. We then evaluate the following metrics: mean rank (the position in the ranked list, averaged over all test items and users), mean maximum rank (the position of the lowest ranked item out of the 5 test items, i.e. the furthest from the top, averaged over all test users), precision at 1 and 10 (P@1 and P@10), and recall at 1 and 10 (R@1 and R@10).

Hyperparameters (C, learning rate) were chosen using a portion of the training set for validation, although for memory and speed reasons we limited the embedding dimension to be $m = 64$. As we trained our model, K-os, with a ranking criteria which includes the WARP loss and AUC losses as special cases, we consider those as our baselines, and report relative changes in metrics compared to them. For K-os in all cases we used $K = 5$ in Algorithm 1, i.e we sample 5 positive items. After ordering them by score, we then select the item in the k^{th} position. We report re-

Table 1: Recommendation Datasets

Dataset	Music: Artists	Music: Tracks	YouTube
Number of Items	≈75k	≈700k	≈500k
Train Users	Millions		
Test Users	Tens of Thousands		

Table 2: Google Music Artist Recommendation. The baseline model is AUC. Mean/Max Rank, P@N and R@N metrics are given relative to it. Decreases in rank and increases in R@N and P@N indicate improvements. K-os uses AUC via Algorithm 3.

Method	Mean Rank	Max Rank	P@1	P@10	R@1	R@10
SVD	+254%	+284%	+1.6%	-2.5%	+0.72%	-2.1%
WARP	+23%	+26%	+25%	+14%	+25%	+13%
AUC	-	-	-	-	-	-
K-os k=1	+159%	+194%	+1.3%	-5%	-0.1%	-5.6%
K-os k=2	+65%	+80%	+9%	+0.3%	+7%	-0.4%
K-os k=3	+15%	+20%	+10%	+3.9%	+9%	+3.6%
K-os k=4	-2.7%	-1.6%	+6%	+3%	+5.9%	+2.7%
K-os k=5	-2.2%	-3.7%	-25%	-8%	-24%	-8%

Table 3: Google Music Artist Recommendation with WARP baseline. K-os uses WARP via Algorithm 2.

Method	Mean Rank	Max Rank	P@1	P@10	R@1	R@10
SVD	+187%	+205%	-19%	-14%	-19%	-14%
WARP	-	-	-	-	-	-
K-os k=1	+195%	+224%	-1.3%	-5.2%	-1.8%	-5.6%
K-os k=2	+88%	+110%	+1%	-0.4%	+0.7%	-0.6%
K-os k=3	+23%	+32%	-1%	-0.4%	-1.6%	-0.4%
K-os k=4	-7.7%	-6.4%	-3%	-2%	-4%	-2%
K-os k=5	-16%	-18%	-14%	-7%	-14%	-7%

Table 4: Google Music Track Recommendation

Method	Mean Rank	Max Rank	P@1	P@10	R@1	R@10
WARP	-	-	-	-	-	-
K-os k=1	+323%	+271%	+16%	+3.3%	+17%	+4.3%
K-os k=2	+209%	+199%	+22%	+14%	+23%	+15%
K-os k=3	+50.7%	+61%	+22%	+19%	+22%	+20%
K-os k=4	-44.1%	-40.9%	+9.1%	+15%	+12%	+16%
K-os k=5	-54.7%	-54.8%	-50%	-32%	-50%	-33%

Table 5: YouTube Video Recommendation

Method	Mean Rank	Max Rank	P@1	P@10	R@1	R@10
SVD	+56%	+45.3%	-54%	-57%	-54%	-96%
WARP	-	-	-	-	-	-
K-os k=1	+119%	+101%	+14%	+6.5%	+12%	+3.5%
K-os k=2	+55%	+71%	+7%	+6%	+5.8%	+4.8%
K-os k=3	+10%	+19%	-1.4%	+1.3%	-1.2%	+2.1%
K-os k=4	-10%	-13%	-10%	-6.4%	-8%	-3.8%
K-os k=5	-14%	-23%	-36%	-32%	-34%	-30%
K-os k<2	+119%	+101%	+14%	+6.5%	+12%	+3.5%
K-os k<3	+76%	+84%	+10%	+6.7%	+10%	+4.7%
K-os k<4	+39%	+54%	+6.1%	+5.6%	+6%	+5.1%
K-os k<5	+16%	+24%	+2.6%	+1%	+2.7%	+0.4%
K-os k>1	-4.3%	-5.8%	-12%	-10%	-12%	-9.9%
K-os k>2	-6.4%	-10%	-28%	-25%	-28%	-25%
K-os k>3	-18%	-26%	-23%	-23%	-20%	-20%
K-os k>4	-14%	-23.1%	-36%	-32%	-34%	-30%

sults for different values of k to show its effect. On YouTube and the Google Music artist recommendation task we also compare to SVD. Results on the three datasets are given in Tables 2, 3, 4 and 5.

For the first dataset, Google Music artist recommendation, we report two sets of results. Table 2 gives performance of K-OS using AUC (Algorithm 3) relative to standard AUC training. Table 3 gives performance of K-OS using WARP (Algorithm 2) relative to standard WARP training. We also compare to SVD, which is outperformed by both AUC and WARP ranking losses, presumably because they are better at optimizing these ranking metrics as has been observed before [9]. Note that a strongly performing model has a small mean/max rank, and large values of precision/recall, hence we are looking for negative percentage changes in rank but positive changes in the other metrics. In both the AUC and WARP cases the choice of k in K-OS gives clear control over the loss function. Small values of k tend to optimize precision and recall metrics as they focus on the top ranked positives in the set. Larger values of k tend to optimize mean maximum rank as they focus on the bottom ranked positives in the set. For example, the choice of $k = 5$ in Table 2 gives improved rank metrics over the AUC baseline, at the expense of decreases in precision and recall. Conversely, choices of $k \leq 4$ give improved precision and recall metrics over the AUC baseline at the expense of larger rank metrics. Note that $k = 1$ does not give the best precision improvements as you might at first expect ($k = 2$ is better). We hypothesize that this is because concentrating too much on only the top ranked positive makes the overall model suffer from not seeing enough training data with varying labels. The same observation is true in the track recommendation dataset as well.

The second dataset, Google Music track recommendation, is comprised of the same set of anonymized users, but with items represented at the track rather than the artist level. That means there are more items to rank ($\approx$700k rather than $\approx$75k) so one could expect bigger differences between the methods as the task is more difficult. Table 4 shows larger improvements over the WARP baseline both in rank metrics ($k \geq 4$) and precision and recall metrics ($k \leq 4$). In this case $k = 4$ is actually a sweet spot which gives improvements in all metrics compared to the baseline.

The third dataset, YouTube video recommendation, also shows improvements in metrics for various choices of k. Again, we see smooth transitions from optimizing max or mean rank metrics versus precision or recall at the top as we vary k. In these experiments as well as showing results for single values of k we also report distributions P where we sample uniformly at random different values of k. For example, K-OS $k < 4$ in the table means that we select uniformly at random one of the top 3 positives after ordering the 5 sampled positives. The conclusions are similar to those of the experiments in the previous datasets.

YouTube Live Experiment.

We next tried our method using the K-OS loss (using WARP and $k = 5$, $N = 5$) in the live YouTube video recommendation system where we attempted to improve an already strong baseline machine learning system [1]. In our experiments above we measured rank, precision and recall. However, all these metrics are merely a proxy for the online metrics that matter more, such as video click through rate and duration of watch. When evaluating our method in the real-world system, it gave statistically signicant increases in both click through rate and watch duration (approximately 1%) compared to using the standard WARP loss, which in turn was superior to the AUC loss.

4. CONCLUSIONS

In this paper we have introduced a general class of ranking loss functions for training large-scale factorized recommendation models. This class generalizes several well known loss functions such as AUC and WARP and also provides new choices of objective function. In particular, by focusing the training on more highly ranked items one can obtain better precision and recall metrics compared to those existing approaches. Alternatively, by focusing the training on lower ranked items one can obtain better mean or maximum rank metrics. Depending on the overall goal, both of these approaches may be useful. We hypothesize that the latter improves the overall diversity of the recommendations which in live YouTube experiments resulted in more engaged users. Future work could try to understand further the impact of these loss function choices on such end goals.

5. REFERENCES

[1] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

[2] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[3] D. Grangier and S. Bengio. A discriminative kernel-based model to rank images from text queries. *PAMI*, 30:1371–1384, 2008.

[4] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. *NIPS*, pages 115–132, 1999.

[5] 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 *Proceedings of the sixth ACM conference on Recommender systems*, pages 139–146. ACM, 2012.

[6] N. Usunier, D. Buffoni, and P. Gallinari. Ranking with ordered weighted pairwise classification. *ICML*, 2009.

[7] M. Weimer, A. Karatzoglou, Q. Le, A. Smola, et al. Cofirank-maximum margin matrix factorization for collaborative ranking. *NIPS*, 2007.

[8] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, pages 2764–2770, 2011.

[9] J. Weston, C. Wang, R. Weiss, and A. Berenzweig. Latent collaborative retrieval. *ICML*, 2012.

[10] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *ICML*, 2008.

[11] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR*, pages 271–278, 2007.