Nonlinear Latent Factorization by Embedding Multiple User Interests

[Extended Abstract]

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

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

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

ABSTRACT

Classical matrix factorization approaches to collaborative filtering learn a latent vector for each user and each item, and recommendations are scored via the similarity between two such vectors, which are of the same dimension. In this work, we are motivated by the intuition that a user is a much more complicated entity than any single item, and cannot be well described by the same representation. Hence, the variety of a user's interests could be better captured by a more complex representation. We propose to model the user with a richer set of functions, specifically via a set of latent vectors, where each vector captures one of the user's latent interests or tastes. The overall recommendation model is then nonlinear where the matching score between a user and a given item is the maximum matching score over each of the user's latent interests with respect to the item's latent representation. We describe a simple, general and efficient algorithm for learning such a model, and apply it to large scale, real-world datasets from YouTube and Google Music, where our approach outperforms existing techniques.

Categories and Subject Descriptors

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

Keywords

collaborative filtering, learning to rank, matrix factorization, nonlinear models

1. INTRODUCTION

Standard latent models for recommendation are linear factorizations of the user-item matrix of preferences. These models are then applied at test time for recommendations of new items. For example, singular value decomposition (SVD) and its variants, such as SVD++ (see [2] for a good review), factorize the user-item matrix in a least-squares sense where the matrix elements are considered to be ratings. Methods like CofiRank [8], CLiMF [6] or Wsabie [10] optimize a ranking loss instead with the goal of ranking higher valued user-item pairs above lower-scoring items for the given user, but the model still has the same linear form. In a linear factorization both a given user and a given item are assigned an m-dimensional vector of latent features, where typically m is small, between ten and a few hundred dimensions. To score each item one scores its m dimensional "embedding" against the m-dimensional user embedding, and orders the items by this score. This means that the representation of the user has the same capacity as the representation of a single item.

In this work we challenge the use of that common modeling approach as we believe a user should have a much richer latent representation. Intuitively, the model of a user should be able to encode that they may have many different tastes concerning different items, whereas a single item representation only has to encode a single item. If the item set to be recommended is a large, diverse set, we believe this limitation will have a larger impact. For example, consider the set of all products sold by an online store like Amazon. There is likely only very weak correlation between the user's tastes in different product areas, e.g. between the movies they might like, and the household products they wish to purchase. Yet, the standard factorization models forces that all these interests are encoded in the m latent factors, and diverse topics are forced to share these features. Even within a more focused recommendation area, such as music or movies the issue may be the same, for example a user's interest in history documentaries may have little correlation with their interest in romantic comedies. One could argue that one can simply increase the item embedding dimensionality to capture more interests in the user representation, but there are reasons not to do this (both practical, and generalization-based, especially for the long tail). Practically, the model becomes larger in memory which can be an issue in production. Secondly, as it is larger it would take longer to train well, and may overfit as the item representations have also become larger. We believe that employing a nonlinearity and a higher capacity user model, and keeping the item representation small, makes more sense as we will describe.

Our solution is to model a user with T latent vectors, each of dimension m, that model the user's latent tastes. Each item still has a single m-dimensional latent feature vector. The score between a user and a given item is the maximum match between the user tastes and the given item. The remainder of the paper is as follows. We first describe previous work in Section 2. We then present our model in Section 3, detailing the objective function and learning procedure to train such a model by gradient descent. Experimental results on large scale recommendation tasks from Google Music and YouTube are reported in Section 4. We finally sketch some directions for future work in Section 5.

2. RELATED WORK

We have already talked about the most related linear factorization models in the introduction, but the method we propose in this work generalizes those to a nonlinear model. Several types of nonlinear models have been tried before for recommendation. The authors of [3] proposed a nonlinear matrix factorization approach with Gaussian processes by using a kernelized form for the model. They report very good results on 1M MovieLens and EachMovie, however their approach may have scalability issues for larger problems. The authors of [5] applied Restricted Boltzmann machines for collaborative filtering, which is a kind of neural network that introduces nonlinearities via Gaussian hidden units. They obtained results slightly better than SVD, and an ensemble of the two methods performs very well. Our method can also be considered to be a particular kind of neural network, utilizing max nonlinearities rather than the standard sigmoids, although it is quite different from that of [5].

Other machine learning models consider max-based nonlinearities as we do, e.g. convolutional networks for image tasks use max-pooling [4]. To the best of our knowledge, however, they have not been used in latent models as in our approach, and have similarly not been applied to recommendation tasks.

Perhaps closest to our approach is the work of [1] where multiple profiles for user modeling (micro-profiling) are used for context-aware recommendation.

3. MAX INTEREST LATENT FACTORS

Standard linear user-item factorizations are of the form:

$$f(u,d) = U_u^\top V_d \tag{1}$$

where u is the given user, $u \in \{1,\dots,|\mathcal{U}|\}$, and d is an item to be recommended, $d \in \{1,\dots,|\mathcal{D}|\}$. The score assigned for the given user and item, $f(u,d) = S_{ud} = U_u^\top V_d$, can be seen as the prediction of a missing element of the (full rank) matrix S of dimension $|\mathcal{U}| \times |\mathcal{D}|$, which is approximated by the low rank factorization of S by U and V. The matrix U, which is of size $m \times |\mathcal{U}|$, encodes the latent factors for the users, and a $m \times |\mathcal{D}|$ matrix V encodes those of the items. Hence, every user and every item has m latent factors.

The key idea of the proposed model is to define T interest vectors per user, where the user part of the model is written as $\hat{U}$ which is an $m \times |\mathcal{U}| \times T$ tensor. Hence, we also write $\hat{U}_{iu} \in \mathbb{R}^m$ as the m-dimensional vector that represents the i^{th} of T possible interests for user u. The item part of the model is the same as in the classical user-item factorization models, and is still denoted as a $m \times |\mathcal{D}|$ matrix V. The new scoring model is defined as:

$$f(u,d) = \max_{i=1,\dots,T} \hat{U}_{iu}^\top V_d. \tag{2}$$

For any given item, we are now computing T dot products, rather than one, when comparing it to the user, and taking the highest scoring. The intuition here is that the user is modeled with T differing tastes and the item matches with one of them best, captured via the max function. Said differently, for each user, the set of items is partitioned into T subsets, where the partitioning can vary across users. For each partition a different scoring function is applied.

Algorithm 1 MaxMF SGD algorithm.

```
Initialize model parameters (we use mean 0, standard deviation $\frac{1}{\sqrt{m}}$), unless parameters are already initialized during MapReduce (Algorithm 2).
repeat
  For user $u$ randomly pick a positive item $d \in \mathcal{D}_u$.
  Compute $f(u,d)$.
  Set $N = 0$.
  repeat
    Pick a random item $\bar{d} \in \mathcal{D} \setminus \mathcal{D}_u$.
    $N = N + 1$.
  until $f(u,\bar{d}) > f(u,d) - 1$ or $N \geq |\mathcal{D} \setminus \mathcal{D}_u|$
  if $f(u,\bar{d}) > f(u,d) - 1$ then
    Make a gradient step to minimize:
        $L\big(\frac{|\mathcal{D}\setminus\mathcal{D}_u|}{N}\big)\max(0, 1 + f(u,\bar{d}) - f(u,d))$.
    (i.e. equations (3)-(6) for $f(u,d)$ defined by (2)).
    Project weights to enforce constraints, e.g. if $||V_i|| > C$ then set $V_i \leftarrow (CV_i)/||V_i||$ (and similar for $U$).
  end if
until validation error does not improve.
```

Algorithm 2 MaxMF MapReduce algorithm.

```
Initialize $V$ randomly (mean 0, standard deviation $\frac{1}{\sqrt{m}}$).
Define model $f_1(u,d) = \frac{1}{|\mathcal{D}_u|}\sum_{i\in\mathcal{D}_u} V_i^\top V_d$.
Train $f_1(u,d)$ using Algorithm 1.
Define $f_2(u,d) = \max_i \hat{U}_{iu}^\top V_d^*$, where $V^* = V$ from $f_1$.
for each user $u$ (in parallel) do
  Train $\hat{U}_u$, but keep $V^*$ fixed, i.e. run Algorithm 1 but only invoke the gradient updates (3)-(4) and not (5)-(6).
end for
```

Stochastic Gradient Descent Training.

Now that we have described the model, the next step is to describe how to train it. We could learn such a model using a regression (least squares) approach such as in SVD, but in this work we focus on learning to rank as it has been observed to perform well on several recommendation tasks previously [6, 10]. Our starting point is the objective of the linear factorization model, Wsabie [9], which learns the model parameters by minimizing:

$$\sum_{u\in\mathcal{U}}\sum_{d\in\mathcal{D}}\sum_{\bar{d}\notin\mathcal{D}_u} L\big(rank_d(u)\big)\max(0, 1 + f(u,\bar{d}) - f(u,d)).$$

Here $\mathcal{D}_u$ denotes the set of items that the user has purchased / watched / listened to (depending on the context) which we refer to as positive items, i.e. we are in a binary rating setting rather than the real-valued rating setting e.g. of the Netflix challenge. In fact, the binary setting is a very common one in real-world problems, in our experience more common than the rating-based one. In the absence of negative data, the above objective tries to rank all the positive items as highly as possible. Here, $rank_d(u)$ is the rank of the positive item d relative to all the negative items:

$$rank_d(u) = \sum_{\bar{d} \notin \mathcal{D}_u} I(f(u,d) \leq 1 + f(u,\bar{d})),$$

and $L(\eta)$ converts the rank to a weight. Choosing $L(\eta) = C\eta$ for any positive constant C optimizes the mean rank, whereas a weighting such as $L(\eta) = \sum_{i=1}^{\eta} 1/i$ optimizes the top of the ranked list, as described in [7]. To train with such an objective, stochastic gradient has previously been employed. For speed the computation of $rank_d(u)$ is then replaced with a sampled approximation: sample N items $\bar{d}$ until a violation is found, i.e. $\max(0, 1+f(u,\bar{d})-f(u,d))) > 0$ and then approximate the rank with $|\mathcal{D} \setminus \mathcal{D}_u|/N$.

Although our model is nonlinear, we can still use almost the same procedure as used for the linear model in [10], we just need to compute the relevant gradients for our model. Specifically, for a given triple $(u, d, \bar{d})$, let us denote the maximum scoring user interest vector for d as $\hat{U}_{ut}$ and $\hat{U}_{u\bar{t}}$ for $\bar{d}$, where

$$\forall j \neq t : \hat{U}_{tu}^\top V_d > \hat{U}_{ju}^\top V_d, \qquad \forall j \neq t : \hat{U}_{\bar{t}u}^\top V_d > \hat{U}_{ju}^\top V_{\bar{d}}.$$

For a violating triple we then need to update for the user model:

$$\hat{U}_{tu} \leftarrow \hat{U}_{tu} + \lambda L(\tfrac{|\mathcal{D} \setminus \mathcal{D}_u|}{N}) V_d, \tag{3}$$

$$\hat{U}_{\bar{t}u} \leftarrow \hat{U}_{\bar{t}u} - \lambda L(\tfrac{|\mathcal{D} \setminus \mathcal{D}_u|}{N}) V_{\bar{d}}, \tag{4}$$

and for the item model:

$$\hat{V}_d \leftarrow \hat{V}_d + \lambda L(\tfrac{|\mathcal{D} \setminus \mathcal{D}_u|}{N}) \hat{U}_{ut}, \tag{5}$$

$$\hat{V}_{\bar{d}} \leftarrow \hat{V}_d - \lambda L(\tfrac{|\mathcal{D} \setminus \mathcal{D}_u|}{N}) \hat{U}_{u\bar{t}}, \tag{6}$$

where λ is the learning rate. At each step, one typically also enforces that $||U_{ij}|| \leq C$ and $||V_i|| \leq C$, for all i and j, as a means of regularization.

The whole procedure is outlined in Algorithm 1, and is scalable to medium-size datasets, where the model fits in memory. Unfortunately, there is a major problem when the dataset becomes large. For example if we would like to apply this method to a dataset with 1 billion users (more than 1 billion unique users visit YouTube each month) and latent dimension $m = 64$, this gives a model of size > 238GB (even for $T = 1$). This is too big for most machines, and even if it did fit, it might also be very slow to train. We therefore consider a MapReduce-based training solution to deal with this issue.

MapReduce Training.

Our solution to scale this algorithm is to consider an alternating optimization: first optimize over the item embeddings V and then train the user embeddings U. In the first phase where we train V, the user model U is unknown. One option would be to initialize U to random values, however V is unlikely to be trained well unless the process is iterated enough to allow U to also converge (but similarly U will be training with an impoverished V in early iterations). Instead, in the first phase we build a model that factors out U completely:

$$f_1(u,d) = \frac{1}{|\mathcal{D}_u|} \sum_{i \in \mathcal{D}_u} V_i^\top V_d.$$

i.e. we consider that $U_u \triangleq \frac{1}{|\mathcal{D}_u|} \sum_{i \in \mathcal{D}_u} V_i$. Note that this is reminiscent of the item-based embedding of the user employed in SVD++ [2]. We then train using the stochastic gradient descent (SGD) approach of Algorithm 1 which is now tractable, at least if the number of items is say less than a few million. This gives us a good initialization for the item embeddings V. In the second phase we fix V to V^* and train the user models only:

$$f_2(u,d) = \max_{i,\ldots,T} \hat{U}_{iu}^\top V_d^*,$$

This decouples the parameters for each user, allowing them to be trained independently. We thus train all users in parallel using MapReduce: the mappers collect and emit the relevant training triples keyed by user, and the reducers train each user independently without requiring a single machine to have access to the entire model. The reducers can then emit their $\bar{U}_u$ vectors to save the entire model to disk for further training iterations. I.e. one can then go back and retrain V given fixed U and so on, although we only considered a two-phase setup in our experiments. In the two-phase case we do not even need to save the user model to disk if we are interested in batch offline predictions. We can both train each user and perform the ranking of items in the reduce phase and output the recommendations to disk instead, hence the user model is never actually stored.

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. Our baseline model is Wsabie and mean rank (Rank), precision at 1 and 10 (P@N) and recall at 1 and 10 (R@N) metrics are given relative to that model. Decreases in rank and increases in R@N and P@N indicate improvements.

Method	Rank	P@1	P@10	R@1	R@10
SVD	+26%	-7.9%	-1.5%	-6.7%	-0.75%
Wsabie	-	-	-	-	-
MaxMF T=3	-3.9%	-3.1%	-0.18%	-4.4%	-0.15%
MaxMF T=5	-8%	-0.46%	+1.3%	-0.63%	+1.9%
MaxMF T=10	-11%	+0.33%	+3.1%	+0.33%	+3.2%

Table 3: Google Music Track Recommendation

Method	Rank	P@1	R@1	P@10	R@10
Wsabie	-	-	-	-	-
MaxMF T=2	-7.7%	+10%	+9.8%	+12%	+10%
MaxMF T=3	-12%	+20%	+20%	+21%	+20%
MaxMF T=5	-15%	+24%	+28%	+26%	+30%
MaxMF T=10	-17%	+30%	+33%	+32%	+34%

Table 4: YouTube Video Recommendation

Method	Rank	P@1	p@10	R@1	R@10
SVD	+56%	-54%	-57%	-54%	-96%
Wsabie	-	-	-	-	-
MaxMF T=2	-3.2%	+8.8%	+13%	+9.9%	+14%
MaxMF T=3	-6.2%	+16%	+18%	+17%	+19%
MaxMF T=5	-9%	+22%	+23%	+23%	+23%
MaxMF T=10	-11%	+26%	+26%	+28%	+26%

4. EXPERIMENTS

We conducted experiments on three large scale, real world tasks: music recommendation using proprietary data from Google Music[1], of both artists and individual tracks, and video recommendation from YouTube[2]. 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 sizes of the datasets 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 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: rank (the position in the ranked list, averaged over all test items), precision at 1 and 10 (P@1 and P@10), and recall at 1 and 10 (R@1 and R@10).

Hyperparameters (λ, C) 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, MaxMF (Max-nonlinearity Matrix Factorization), with a ranking criteria, we consider our baseline to be the same type of model, but without nonlinearity, which is the Wsabie model of [10]. So we used the same λ and C that are optimal for Wsabie and then report results for different values of T for MaxMF with those settings, using the MapReduce based training scheme. On Google Music Artist Recommendation and the YouTube tasks we also compare to SVD (L2-optimal matrix factorization for the complete matrix with log-odds weighting on the columns, which downweights the importance of the popular features, as that worked better than uniform weights). (For YouTube, we trained the SVD on a (large) subset due to scalabilty issues.) Note, however, that SVD was shown to be inferior to Wsabie on several datasets previously [10]. Hence, as Wsabie is our main baseline, we report relative changes in metrics when using other methods compared to it. Results on the three datasets are given in Tables 2, 3 and 4.

The first dataset, Google Music artist recommendation (Table 2), MaxMF gave only relatively small gains compared to Wsabie. SVD performed worse in all metrics. Note that a strongly performing model has a small mean rank, and large precision and recall values, hence we are looking for negative percentage changes in mean rank but positive changes in the other metrics. SVD likely underperforms since we are measuring ranking based metrics, which it does not optimize at training time. In other respects, the models are very similar. For MaxMF, as we increased the number of user interests T that we model, mean rank decreases from 3.9%, 8% and 11% compared to Wsabie, for $T = 3$, $T = 5$ and $T = 10$ respectively. Changes in precision and recall are smaller. We hypothesize that the relatively small size of this dataset compared to the other two is the reason why MaxMF has smaller gains.

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, ≈700k rather than ≈75k, and correspondingly more items per user. This means the task is harder, since there are more items to rank. However, per user there are more items, which might mean that training nonlinear interest models per user is less likely to overfit, since there is more data for training. As is shown in Table 3 we get considerable gains using our method on this dataset compared to the Wsabie baseline, up to ≈30% gains in precision and recall using $T = 10$ user interest vectors.

The third dataset, YouTube video recommendation, requires us to rank 500k videos. As shown in Table 4 the gains are also quite large, up to around 25% improvement in recall and precision compared to the baseline model.

[1]http://music.google.com
[2]http://www.youtube.com

5. CONCLUSIONS

In this paper, we have introduced a highly scalable method for learning a nonlinear latent factorization. Experiments on three large scale, real-world datasets indicate the efficacy of the approach.

Further work could try explore further algorithms, and to analyse and understand these results further. For example, we could explore whether we can learn the optimal number of tastes T per user rather than leaving this parameter fixed.

6. REFERENCES

[1] L. Baltrunas and X. Amatriain. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARSâĂŹ09)*, 2009.

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

[3] N. D. Lawrence and R. Urtasun. Non-linear matrix factorization with gaussian processes. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 601–608. ACM, 2009.

[4] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995.

[5] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[6] 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.

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

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

[9] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *Intl. Joint Conf. Artificial Intelligence, (IJCAI)*, pages 2764–2770, 2011.

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