Collaborative Metric Learning

Cheng-Kang Hsieh[‡], Longqi Yang[†],Yin Cui[†], Tsung-Yi Lin[†], Serge Belongie[†], Deborah Estrin[†]
[‡]UCLA; [†]Cornell Tech
[‡]changun@cs.ucla.edu; [†]{ly283, yc984, tl483, sjb344, destrin}@cornell.edu

ABSTRACT

Metric learning algorithms produce distance metrics that capture the important relationships among data. In this work, we study the connection between metric learning and collaborative filtering. We propose Collaborative Metric Learning (CML) which learns a joint metric space to encode not only users' preferences but also the user-user and item-item similarity. The proposed algorithm outperforms state-of-the-art collaborative filtering algorithms on a wide range of recommendation tasks and uncovers the underlying spectrum of users' fine-grained preferences. CML also achieves significant speedup for Top-K recommendation tasks using off-the-shelf, approximate nearest-neighbor search, with negligible accuracy reduction.

1. INTRODUCTION

The notion of *distance* is at the heart of many fundamental machine learning algorithms, including K-nearest neighbor, K-means and SVMs. Metric learning algorithms produce a distance metric that captures the important relationships among data and is an indispensable technique for many successful machine learning applications, including image classification, document retrieval and protein function prediction. [43, 45, 26, 52, 32]. In this paper we apply metric learning to collaborative filtering problems and compare it to state-of-the-art collaborative filtering approaches.

Given a set of objects in which we know certain pairs of objects are "similar" or "dissimilar," the goal of metric learning is to learn a distance metric that respects these relationships. Specifically, we would like to learn a metric that assigns smaller distances between similar pairs, and larger distances between dissimilar pairs. For instance, in face recognition, we may wish to learn a metric that assigns small distances to genuine (same identity) face pairs and large distances to impostor (different identity) pairs, based on image pixels [43, 29].

Mathematically, a metric needs to satisfy several conditions, among which the triangle inequality is the most crucial to the generalization of a learned metric [29, 51]. The triangle inequality states that for any three objects, the sum of any two pairwise distances should be greater than or equal to the remaining pairwise distance. This implies that, given the information: "x is similar to both y and z," a learned metric will not only pull the stated two pairs closer, but also pull the remaining pair (y, z) relatively close to one another.[1] We can see this as a *similarity propagation* process, in which the learned metric propagates the known similarity information to the pairs whose relationships are unknown.

This notion of similarity propagation is closely related to collaborative filtering (CF). In collaborative filtering, we also observe the relationships between certain (user,item) pairs, i.e., the users' ratings of items, and would like to generalize this information to other unseen pairs [31, 40]. For example, the most well-known CF technique, **matrix factorization**, uses the dot product between user/item vectors to capture the known ratings, and uses the dot product of those vectors to predict the unknown ratings [25].

However, in spite of their conceptual similarity, in most cases matrix factorization is not a metric learning approach because dot product does not satisfy the crucial triangle inequality [36, 42]. As we will illustrate in Section 3.6, although matrix factorization can capture users' general interests, it may fail to capture the finer grained preference information when the triangle inequality is violated, which leads to suboptimal performance, as seen in Section 4. Also, while we can predict the ratings with a matrix factorization system, we cannot reliably determine the user-user and item-item relationships with such a system, which significantly limits the interpretability of resulting model.[2]

In this paper, we propose **collaborative metric learning** (CML) which learns a joint user-item metric to encode not only users' preferences but also the user-user and item-item similarity. We focus on the collaborative filtering problem for implicit feedback and show that CML naturally captures such relationships and outperforms state-of-the-art algorithms in a wide range of recommendation domains, including books, news and photography, by up to 13.95% in recall rates for pure rating-based CF and 17.66% for CF with content information.

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052639

[1] (y, z)'s distance is bounded by the sum of distances between (x, y) and (x, z): $d(y, z) \leq d(x, y) + d(x, z)$.

[2] A common heuristic is to measure latent vectors' cosine similarity, but this heuristic leads to misleading results when the triangle inequality is not satisfied as shown in Section 3.6.

Beyond the improvement in accuracy, an exciting property of CML is its capability of uncovering fine-grained relationships among users' preferences. In Section 4.5, we demonstrate this capability using a Flickr photographic dataset. We demonstrate how the visualization of the learned metric can reveal user preferences for different scenes and objects, and uncovers the underlying spectrum of users' preferences.

We show CML's capability of integrating various types of item features, including images, texts, and tags, through a probabilistic interpretation of the model. Finally, we also demonstrate that the efficiency of CML's Top-K recommendation tasks can be improved by more than 2 orders of magnitude using the off-the-shelf Locality-Sensitive Hashing (LSH), with only negligible reduction in accuracy [4].[3]

The remainder of this paper is organized as follows. In Section 2 we review relevant prior work on metric learning and collaborative filtering. In Section 3 we propose our CML model along with a suite of regularization and training techniques. In Section 4 we evaluate CML relative to the state-of-the-art recommendation algorithms and show how the learned metric can uncover users' fine-grained preferences. In Section 5 we conclude and discuss future work.

2. BACKGROUND

In this section we review the background of metric learning and collaborative filtering with a particular focus on collaborative filtering for implicit feedback.

2.1 Metric Learning

Let $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ be a collection of data over the input space $\mathbb{R}^m$. The label information in metric learning is specified in the form of pairwise constraints, including the set of known similar pairs, denoted by

$$\mathcal{S} = \{(x_i, x_j) | x_i \text{ and } x_j \text{are considered similar}\},$$

and the set of dissimilar pairs, denoted by

$$\mathcal{D} = \{(x_i, x_j) | x_i \text{ and } x_j \text{are considered dissimilar}\}.$$

The most original metric learning approach attempts to learn a Mahalanobis distance metric

$$d_A(x_i, x_j) = \sqrt{(x_i - x_j)^T A (x_i - x_j)}\ ,$$

where $A \in \mathbb{R}^{m \times m}$ is a positive semi-definite matrix [51]. This is equivalent to projecting each input x to a new space $\mathbb{R}^m$ in which their euclidean distances obey the desired constraints. There are many different ways to create such a metric. The most original approach was to globally solve the following convex optimization problem:

$$\min_A \textstyle\sum_{(x_i,x_j)\in\mathcal{S}} d_A(x_i, x_j)^2$$

$$\text{s.t. } \textstyle\sum_{(x_i,x_j)\in\mathcal{D}} d_A(x_i, x_j)^2 \geq 1 \text{ and } A \succeq 0.$$

More recent approaches tend to use non-linear transformation functions, e.g., kernel trick or neural nets, to improve the metric accuracy [49, 48, 8, 21].

[3]Matrix Factorization can also be sped up with a specifically-designed asymmetric hashing [42]. We provide a comparison in Section 4.4.

2.2 Metric Learning for kNN

The above global optimization essentially attempts to learn a distance metric that pulls all similar pairs together, and pushs dissimilar pairs apart. This objective, however, is not always feasible. On the other hand, Weinberger et al. showed that if the learned metric is to be used for k-nearest neighbor classification, it is sufficient to just learn a metric that makes each object's k-nearest neighbors be the objects that share the same class label with that object (as opposed to clustering all of the similar object together) [49]. This objective is much more attainable, and we adopt this relaxed notion of metric learning in our model as our goal is also to find the kNN items to recommend to each user.

Specifically, given an input x, we refer to the data points we desire to be the closest to x as its *target neighbors*. We can imagine x's target neighbors establishing a perimeter that differently labeled inputs should not invade. The differently labeled inputs that invade the perimeter are referred to as *impostors*. The goal of learning, in general, is to learn a metric that minimizes the number of impostors [49].

One of the most well-known models of this type is **large margin nearest neighbor** (LMNN) [49] that uses two loss terms to formulate this idea. Specifically, LMNN defines a *pull loss* that pulls the target neighbors of an input x closer:

$$\mathcal{L}_{pull}(d) = \sum_{j \rightsquigarrow i} d(x_i, x_j)^2,$$

where $j \rightsquigarrow i$ denotes that the input j is input i's target neighbor. In addition, LMNN defines a *push loss* that pushes away the impostors from the neighborhood and maintains a margin of safety around the kNN decision boundaries:

$$\mathcal{L}_{push}(d) = \sum_{i, j \rightsquigarrow i} \sum_k (1 - y_{ik})[1 + d(x_i, x_j)^2 - d(x_i, x_k)^2]_+,$$

where indicator function $y_{ik} = 1$ if input i and k are of the same class, otherwise $y_{ik} = 0$, and $[z]_+ = \max(z, 0)$ is the standard hinge loss. The complete loss function of LMNN is a weighted combination of $\mathcal{L}_{pull}(d)$ and $\mathcal{L}_{push}(d)$ and can be optimized through semidefinite programming.

2.3 Collaborative Filtering

We now turn our attention to collaborative filtering (CF), in particular, collaborative filtering with implicit feedback. Traditional collaborative filtering algorithms were based on the user similarity computed by heuristics, such as cosine similarity. The recommendations were made by aggregating the ratings of the K-nearest users who are the most similar to the query user [40, 31].

Over the last decade, matrix factorization (MF) has become the most popular CF approach due to its superior performance [25, 19, 1]. The original MF models were designed to model users' explicit feedback by mapping users and items to a latent factor space, such that user-item relationships (i.e., ratings) can be captured by their latent factors' dot product. Specifically, let r_{ij} denote user i's rating to item j, we learn user vector $\mathbf{u}_i \in \mathbb{R}^r$ and item vector $\mathbf{v}_j \in \mathbb{R}^r$, such that their dot product $\mathbf{u}_i^T \mathbf{v}_j$ approximate r_{ij} [25]. This formulation leads to the optimization problem that minimizes the mean squared error on the set of known

ratings:

$$\min_{\mathbf{u}_*,\mathbf{v}_*} \sum_{r_{ij}\in\mathcal{K}} (r_{ij} - \mathbf{u}_i^T\mathbf{v}_j)^2 + \lambda_u\|\mathbf{u}_i\|^2 + \lambda_v\|\mathbf{v}_i\|^2,$$

where $\mathcal{K}$ is the set of known ratings; λ_u and λ_v are hyper-parameters that regularize the L^2-norm of $\mathbf{u}_*$ and $\mathbf{v}_*$.

2.3.1 Implicit Feedback

Besides explicit feedback, there are many other signals that can be utilized to infer users' preferences, such as likes, bookmarks, click-through, etc., which are collectively referred to as *implicit feedback* [19, 38]. Implicit feedback is usually more abundant and less biased than its explicit counterpart [22, 20], and has received much attention in the research community [19, 35, 38, 3, 46, 16].

However, it is problematic to apply traditional matrix factorization to implicit feedback, mainly for two reasons: first, with implicit feedback, we only observe positive feedback (i.e., $r_{ij} = 1, \forall r_{ij} \in \mathcal{K}$). We cannot ignore the unobserved user-item interactions, otherwise it will lead to trivial but useless solutions (e.g., collapsing all the latent vectors to a single point). Also, we cannot assume these unobserved interactions as negative either, as we do not know the fact that these interactions did not happen was because the user did not like the item or the user was not aware of it [19].

To address these issues, Hu et al. and Pan et al. proposed **weighted regularized matrix factorization** (WRMF) [19, 35] that includes all the unobserved user-item interactions as negative samples and uses a case weight $c_{i,j}$ to reduce the impact of these uncertain samples, i.e.,

$$\min_{\mathbf{u}_*,\mathbf{v}_*} \sum_{r_{ij}\in\mathcal{K}} c_{ij}(r_{ij} - \mathbf{u}_i^T\mathbf{v}_j)^2 + \lambda_u\|\mathbf{u}_i\|^2 + \lambda_v\|\mathbf{v}_i\|^2,$$

where case weight c_{ij} is larger for observed positive feedback and smaller for unobserved interactions.

2.3.2 Bayesian Personalized Ranking

As the above discussion shows, with implicit feedback, the notion of "ratings" become less precise. As a result, more recent matrix factorization models start to move away from estimating a specific set of ratings to, instead, modeling the relative preferences (or orders) between different items. **Bayesian personalized ranking** (BPR) proposed by Rendle et al. [38] is a well-known example of this type. Let $\mathcal{D}_i$ be a set of item pairs $(j,\ k)$ where user i has interacted with item j but not item k, assuming user i might be more interested in item j than item k, BPR minimizes the pairwise ranking loss:

$$\min_{\mathbf{u}_*,\mathbf{v}_*} \sum_{i\in\mathcal{I}} \sum_{(j,k)\in\mathcal{D}_i} -log\ \sigma(\mathbf{u}_i^T\mathbf{v}_j - \mathbf{u}_i^T\mathbf{v}_k) + \lambda_u\|\mathbf{u}_i\|^2 + \lambda_v\|\mathbf{v}_j\|^2,$$

where σ is the sigmoid function.

Through this loss function, BPR essentially attempts to minimize the error of predicting, between a pair of items, which one the user prefers. This loss function is also equivalent to optimizing the Area Under ROC Curve (AUC) for each user [39]. However, an issue with the BPR loss is that it does not sufficiently penalize the items that are at a lower rank. It produces suboptimal results for Top-K recommendation tasks where only the items ranked within the Top-K matter [53]. A popular way to improve the Top-K recommendation is to adopt a *weighted ranking loss* that penalizes the positive items at a lower rank [50, 27, 28] and will be described further in Section 3.2.

3. COLLABORATIVE METRIC LEARNING

The above discussion highlights the fact that, by moving from explicit feedback to implicit feedback, the focus of collaborative filtering is no longer about estimating a specific rating matrix but about capturing users' relative preferences for different items.

In this section we describe CML as a more natural way to capture such relative relationships. The high level idea of CML is as follows: we model the observed implicit feedback as a set of user-item pairs $\mathcal{S}$ that we know have positive relationships and learn a user-item joint metric to encode these relationships. Specifically, the learned metric pulls the pairs in $\mathcal{S}$ closer and pushes the other pairs relatively further apart. This process, due to the triangle inequality, will also cluster 1) the users who co-like the same items together, and 2) the items that are co-liked by the same users together. Eventually, the nearest neighbor items for any given user will become:

- the items liked by this user previously, and
- the items liked by other users who share a similar taste with this user previously.

In other words, by learning a metric that obeys the known positive relationships, we propagate these relationships not only to other user-item pairs, but also to those user-user and item-item pairs for which we did not directly observe such relationships. In the following, we formulate this idea more formally.

3.1 Model Formulation

We represent each user and each item with a user vector $u_i \in \mathcal{R}^r$ and an item vector $v_j \in \mathcal{R}^r$. We learn these vectors in a way that their euclidean distance, i.e.,

$$d(i,j) = \|\mathbf{u}_i - \mathbf{v}_j\|,$$

will obey user i's relative preferences for different items, namely an item this user liked will be closer to this user than other items he did not like. We use the following loss function to formulate such a constraint:

$$\mathcal{L}_m(d) = \sum_{(i,j)\in\mathcal{S}} \sum_{(i,k)\notin\mathcal{S}} w_{ij}[m + d(i,j)^2 - d(i,k)^2]_+, \quad (1)$$

where j is an item user i liked, k is an item he did not like; $[z]_+ = max(z, 0)$ denotes the standard hinge loss, w_{ij} is a ranking loss weight (described later), and $m > 0$ is the safety margin size.

Figure 1 illustrates the gradients resulting from this loss function.[4] For items the user likes, their gradients move inward to create a smaller radius. For impostor items, which are the items the user did not like but which invade the perimeter, their gradients move outward from the user until they are pushed out of the perimeter by a safe margin.

This loss function is similar to that of LMNN but with three important differences:

- Each user's target neighbors are all the items he liked, and there is no target neighbor for items.

[4]The outline of this figure is inspired by [49].

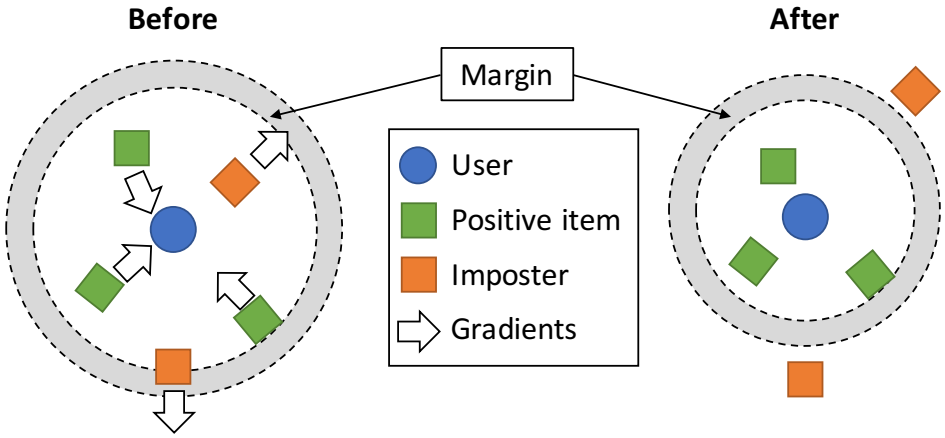

Figure 1: An illustration of collaborative metric learning. The hinge loss defined in Eq. 1 creates a gradient that *pulls* positive items closer to the user and *pushes* the intruding impostor items (i.e., items that the user did not like) away until they are beyond the safety margin.

- We do not have the $\mathcal{L}_{pull}$ term because an item can be liked by many users, and it is not feasible to pull it closer to all of them. However, our *push* loss pulls the positive items closer to the user when there are impostors.
- We adopt a weighted ranking loss to improve the Top-K recommendations described in the next subsection.

3.2 Approximated Ranking Weight

We use a rank-based weighting scheme, called Weighted Approximate-Rank Pairwise (WARP) loss, proposed by Weston et al. to penalize items at a lower rank [50]. Given a metric d, let J denote the total number of items and $rank_d(i,j)$ denote the rank of item j in user i's recommendations [5], we penalize a positive item j based on its rank by setting

$$w_{ij} = log(rank_d(i,j) + 1).$$

This scheme penalizes a positive item at a lower rank much more heavily than one at the top, and produces the state-of-the-art results in many prior works [50, 53, 28]. However, computing $rank_d(i,j)$ at each gradient descent step is rather expensive.

Weston et al. proposed to estimate $rank_d(i,j)$ through a sequential sampling procedure that repeatedly samples a negative item until we find an impostor [50]. Specifically, let N denote the number of negative items we need to sample to find an impostor k that has non-zero loss in Eq. 1, the $rank_d(i,j)$ is then approximated as $\lfloor \frac{J}{N} \rfloor$. This procedure is similar to *negative sample mining* commonly used in object detection where easy but non-informative negative samples dominate the training set [7]. The number of negative samples N is usually bounded by a constant $U = 10$ or 20 to avoid an extended sampling time [27, 53]. In our work, however, we replace this sequential procedure with a parallel procedure to utilize the massive parallelism enabled by modern GPUs:

1. For each user-item pair (i,j), sample U negative items in parallel and compute the hinge loss in Eq. 1.
2. Let M denote the number of imposts in U samples, $rank_d(i,j)$ is then approximated as $\lfloor \frac{J \times M}{U} \rfloor$

It might seem wasteful to always sample U negative items. However, empirically, we find that CML pushes the positive items to a high rank rather quickly after the first few epochs. Therefore, in most cases, we need to sample a similar number of negative items in order to find an impostor, even with the sequential procedure.

[5] $0 \leq rank_d(i,j) < J$, and the top item's $rank_d(i,j) = 0$

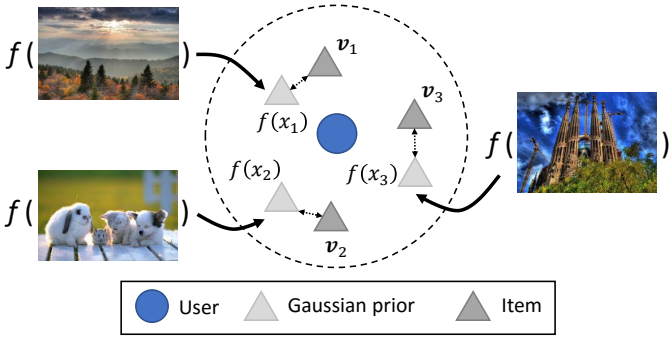

Figure 2: A learnable transformation function f is used to project item features (e.g., image pixels) into the user-item joint space. The projections are treated as a Gaussian prior for items' locations.

3.3 Integrating Item Features

As mentioned in Section 2.1, the original idea of metric learning is to learn a transformation function f that projects the raw inputs to an euclidean space [51]. We adopt a similar idea to integrate item features that are often available in a recommendation system; such as items' text descriptions, tags, or image pixels.[6] Let $\mathbf{x}_j \in \mathcal{R}^m$ denote the m-dimensional raw feature vector of item j, as illustrated in Figure 2, we learn a transformation function f that projects $\mathbf{x}_j$ to the joint user-item space described earlier. As the

[6] Note that the same formulation can be applied to user features as well.

projection $f(\mathbf{x}_j)$ should, to some extent, capture item j's characteristics, we penalize the item j's eventual location in the space (i.e., $\mathbf{v}_j$) when $\mathbf{v}_j$ deviates away from $f(\mathbf{x}_j)$. Specifically, let θ denote the parameters of the function f, we define the following $L2$ loss function:

$$\mathcal{L}_f(\theta, \mathbf{v}_*) = \sum_j \|f(\mathbf{x_j}, \theta) - \mathbf{v}_j\|^2.$$

This loss function essentially treats $f(\mathbf{x}_j)$ as a Gaussian prior to $\mathbf{v}_j$, and we fine-tune the location of $\mathbf{v}_j$ when we have more information about it (i.e., more ratings). Note that the function f is trainable, and during the training, we simultaneously minimize $\mathcal{L}_f$ and metric loss $\mathcal{L}_m$ described earlier to make function f and $\mathbf{v}_*$ mutually inform each other. Specifically, the transformation function f is informed by $\mathbf{v}_*$ and learns to pick up the features that are most relevant to users' preferences; and $\mathbf{v}_*$ is informed by f in a way that the items with similar features will tend to be clustered together and improve the metric accuracy especially for less-rated items. We choose **multi-layer perceptron** (MLP) with dropout as our transformation function f for its superior representational capacity and ease of training [14, 5].

3.4 Regularization

A proper regularization scheme is crucial to the feasibility of the proposed model. Our model essentially projects users and items to a joint r-dimensional space. The number of dimensions determines the representational capacity of the model. However, a kNN-based model like the one we propose is known to be ineffective in a high-dimensional space if the data points spread too widely (i.e., the curse of dimensionality) [11]. Therefore, we bound all the user/item $\mathbf{u}_*$ and $\mathbf{v}_*$ within a unit sphere, i.e.,

$$\|\mathbf{u}_*\|^2 \leq 1 \text{ and } \|\mathbf{v}_*\|^2 \leq 1,$$

to ensure the robustness of the learned metric. Note that, unlike many matrix factorization models, we do not regularize the L^2-norm of $\mathbf{v}_*$ or $\mathbf{u}_*$. Regularizing L^2-norm creates a gradient that pulls every object toward the origin. It is not applicable here because the origin in our metric space does not have any specific meaning.

Another regularization technique we use is covariance regularization recently proposed by Cogswell et al. [9] used to reduce the correlation between activations in a deep neural network. We found the same principle is also useful in de-correlating the dimensions in the learned metric. Let $\mathbf{y}^n$ denote an object's latent vector where an object can be a user or an item, and n indexes the object in a batch of size N. The covariances between all pairs of dimensions i and j form a matrix C:

$$C_{ij} = \frac{1}{N}\sum_n (y_i^n - \mu_i)(y_j^n - \mu_j),$$

where $\mu_i = \frac{1}{N}\sum_n y_i^n$. We define the loss $\mathcal{L}_c$ to regularize the covariances:

$$\mathcal{L}_c = \frac{1}{N}(\|C\|_f - \|diag(C)\|_2^2),$$

where $\|\cdot\|_f$ is the Frobenius norm. As covariances can be seen as a measure of linear redundancy between dimensions, this loss essentially tries to prevent each dimension from being redundant and encourages the whole system to more efficiently utilize the given space.

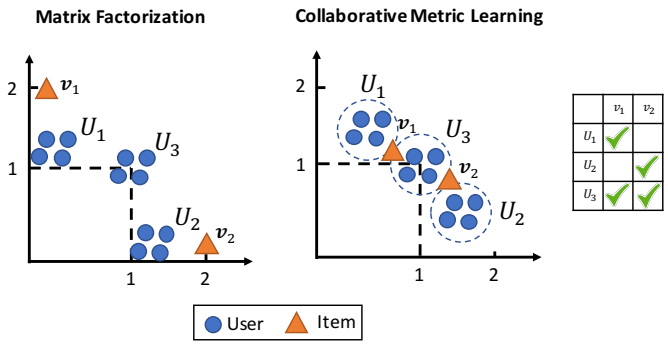

Figure 3: Example latent vector assignments for matrix factorization and CML. The table on the right shows user/item's preference relationships.

3.5 Training Procedure

The complete objective function of the proposed model is as follows:

$$\min_{\theta, \mathbf{u}_*, \mathbf{v}_*} \mathcal{L}_m + \lambda_f \mathcal{L}_f + \lambda_c \mathcal{L}_c$$
$$\text{s.t.} \quad \|\mathbf{u}_*\|^2 \leq 1 \text{ and } \|\mathbf{v}_*\|^2 \leq 1,$$

where λ_f and λ_c are hyperparameters that control the weight of each loss term. We minimize this constrained objective function with Mini-Batch Stochastic Gradient Descent (SGD) and control the learning rating using *AdaGrad* [10], as suggested in [28]. Our training procedure is as follows:

1. Sample N positive pairs from $\mathcal{S}$
2. For each pair, sample U negative items and approximate $rank_d(i, j)$ as described in Section 3.2
3. For each pair, keep the negative item k that maximizes the hinge loss and form a mini-batch of size N.
4. Compute gradients and update parameters with a learning rate controlled by AdaGrad.
5. Censor the norm of $\mathbf{u}_*$ and $\mathbf{v}_*$ by $\mathbf{y}' = \frac{\mathbf{y}}{\max(\|\mathbf{y}\|, 1)}$.
6. Repeat this procedure until convergence.

3.6 Relation to Other Models

In this subsection, we describe the relation between CML and other collaborative filtering models. At a high level, the formulation of CML is similar to that of BPR or other pairwise matrix factorization models described in Section 2.3.1. However, the fact that these matrix factorization models rely on dot product, which does not satisfy the triangle inequality, leads to two important consequences illustrated in the following.

Figure 3 shows three equally-sized groups of users labeled as U_1, U_2 and U_3, where U_1 liked item v_1, U_2 liked item v_2, and U_3 liked *both* item v_1 and v_2. Figure 3 shows a *stable* setting for a matrix factorization system. The setting is stable in a way that the dot product between user/item vectors= 2 when the user liked the item, otherwise their dot-product= 0. However, an important observation is that the dot-product between the item v_1 and item v_2 is 0 even if U_3 like both of them. This violates the triangle inequality because the positive relationships between the pairs (U_3, v_1)

Table 1: Dataset Statistics.

	CiteULike	BookCX	Flickr	Medium	MovieLens20M	EchoNest
Domain	Paper	Book	Photography	News	Movie	Song
# Users	7,947	22,816	43,758	61,909	129,797	766,882
# Items	25,975	43,765	100,000	80,234	20,709	260,417
# Ratings	142,794	623,405	1,372,621	2,047,908	9,939,873	7,261,443
Concentration[a]	33.47%	33.10%	13.48%	55.38%	72.52%	65.88%
Features Type	Tags	Subjects	Image Features	Tags	Genres, Keywords	NA
# Feature Dim.	10,399	7,923	2,048	2,313	10,399	NA

[a]Concentration is defined as the percentage of the ratings that concentrate on the Top 5% of the items.

and (U_3, v_2) are not propagated to (v_1, v_2). Such a violation leads to two undesirable consequences:

1. Although the MF model captures the most prominent factors at its two axises, it does not capture the fine-grained preferences present in users U_3's feedback.
2. The latent vectors of the MF model do not reliably capture the item-item or user-user similarity.

Figure 3 also shows a typical solution for the proposed CML model. Due to the loss function defined in Eq. 1, CML will pull the item v_1 and item v_2 relatively close to each other due to U_3's preferences (in relative to other items not shown in this figure). At the same time, because CML maintains the triangle inequality, the user-user and item-item similarity is also encoded in their euclidean distances in this joint space.

Previous works also considered different euclidean embedding models for collaborative filtering problems. Khoshneshin et al. considered using euclidean embedding for explicit feedback, which is different from the implicit feedback problems considered in this work [23]. Koenigstein et al. also proposed an euclidean embedding model to enable fast similar item retrieval, but their model only captures item-item relationships [24]. Bachrach et al. proposed to transform a pre-trained dot-product space into an approximate euclidean embedding that preserves the item order [2]. Our approach is different as we learn an euclidean metric directly and is able to outperform the state-of-the-art dot-product models.

4. EXPERIMENTS

We conduct thorough experiments to evaluate CML's performance. We show CML's superior accuracy over the state-of-the-art recommendation algorithms in a wide range of recommendation domains. We demonstrate CML's advantage of utilizing off-the-shelf approximate nearest neighbor search algorithms to massively speed up the Top-K recommendation tasks. Finally, we show how the created metric, through a proper visualization, can uncover users' fine-grained preferences and the underlying preference spectrum.

4.1 Datasets

We use datasets from 6 different domains with varying sizes and difficulties to evaluate our model. Table 1 shows statistics of these datasets. For CiteULike data [47], we use the tags of the papers as the item features. For BookCX [54] data, we include users that have more than 5 ratings, and use the book subjects from `OpenLibrary.org` as item features (in a Bag-Of-Word encoding). For Flickr data, we crawl a user-item graph consisting of users and the photos they *favorite* on Flickr. We crawl 100,000 images and use the image features extracted from Deep Residual Net [15] as the item features. For Medium data, we use a subset of news recommendation data collected from `Medium.com` in the prior work [18], and use the article tags as the item features. For MovieLens data [13], we include the ratings greater or equal to 4 as positive feedback as suggested by the prior works on implicit feedback [38, 47], and include the users with more than 10 ratings. We use the genres and plot keywords from `imdb.com` as item features for movies. Finally, for EchoNest data [34], we include the songs a user listened to for at least 5 times as positive feedback. We do not use item features for EchoNest data.

4.2 Evaluation Methodology

We divide each user's ratings into training/validation/test sets in a 60%/20%/20% split. Users who have less than 5 ratings are only included in the training set. The predicted ranking is evaluated on recall rates for Top-K recommendations, which are widely-used performance metrics for implicit feedback [46, 47].

4.2.1 Baselines

We compare CML's recommendation accuracy to 3 collaborative filtering baselines described in Section 2.3, including:

- **WRMF** Weighted Regularized Matrix Factorization [19, 35], the implicit MF model that uses an additional case weight to model unobserved interactions.
- **BPR** Bayesian Personalized Ranking [38] that uses pairwise *log-sigmoid* loss.
- **WARP** Weighted Approximate-Rank Pairwise (WARP) loss based MF model that produces the state-of-the-art Top-K recommendation results [28, 53].

We also compared CML with item features (denoted as **CML+F**) to 3 state-of-the-art hybrid collaborative filtering algorithms, including:

- **FM** Factorization Machine [37], a generalized MF model that captures interactions between categorical variables by projecting them into a joint dot-product space. We only estimate the interactions between (user, item) and (user, item features) as suggested in [28, 27].
- **VBPR** Visual BPR [16] partitions rating dimensions into visual and non-visual factors. The items' visual factors are a linear projection of image features. We replace the linear projection with a more general MLP model.
- **CDL** Collaborative Deep Learning [47] learns item offset vectors from item features through a denoise autoencoder specifically designed for text data. We replace the autoencoder with a more general MLP model applicable to a broader range of features.

Table 2: Recall@50 and Recall@100 on the test set. (# dimensions $r = 100$) The best performing method is boldfaced. $*$, $**$, $***$ indicate $p \leq 0.05$, $p \leq 0.01$, and $p \leq 0.001$ based on the Wilcoxon signed rank test suggested in [41].

	WRMF	BPR	WARP	CML	*ours vs. best*	FM	VBPR	CDL	CML+F	*ours vs. best*
Recall@50										
CiteULike	0.2437	0.2489	0.1916	**0.2714*** **	9.03%	0.1668	0.2807	**0.3375** **	0.3312	-1.86%
BookCX	0.0910	0.0812	0.0801	**0.1037*** **	13.95%	0.1016	0.1004	0.0984	**0.1147*** **	12.89%
Flickr	0.0667	0.0496	0.0576	**0.0711*** **	6.59%	NA	0.0612	0.0679	**0.0753*** **	10.89%
Medium	0.1457	0.1407	0.1619	**0.1730*** **	6.41%	0.1298	0.1656	0.1682	**0.1780*** **	5.82%
MovieLens	0.4317	0.3236	0.4649	**0.4665**	0.34%	0.4384	0.4521	0.4573	**0.4617***	0.96%
EchoNest	0.2285	0.1246	0.2433	**0.2460**	1.10%	NA	NA	NA	NA	NA
Recall@100										
CiteULike	0.3112	0.3296	0.2526	**0.3411*** **	3.37%	0.2166	0.3437	0.4173	**0.4255** **	1.96%
BookCX	0.1286	0.1230	0.1227	**0.1436*** **	11.66%	0.1440	0.1455	0.1428	**0.1712*** **	17.66%
Flickr	0.0821	0.0790	0.0797	**0.0922*** **	12.30%	NA	0.0880	0.0909	**0.1048*** **	15.29%
Medium	0.2112	0.2078	0.2336	**0.2480*** **	6.16%	0.1900	0.2349	0.2408	**0.2531*** **	5.10%
MovieLens	0.5649	0.4455	0.5989	**0.6022**	0.55%	0.5561	0.5712	0.5943	**0.5976**	0.55%
EchoNest	0.2891	0.1655	0.3021	**0.3022**	0.00%	NA	NA	NA	NA	NA

Note that, we train all the above models with the WARP loss for its superior performance over the BPR loss [53], and use the same MLP model in CDL, VBPR, and CML for feature extraction to ensure a fair comparison. We use *LightFM*'s implementation of BPR, WARP and FM [28], which is a popular CF library used in various competitions. We implement CML and the rest of the baselines to run on GPU using *Theano* [44]. We validate our implementation of the baselines by comparing the results to that from [16, 46, 12]

All hyperparameters are tuned to perform the best on the validation set. We set $m = 0.5$, $\lambda_f = 1$ and $\lambda_c = 10$ except for MovieLens where $\lambda_f = 0.5$. We use the MLP with a 256-dimensional hidden layer, 50% dropout, and *ReLu* as the activation function [5]. For MF baselines, λ_v and λ_u are tuned to their best. The data and implementations are open-sourced for reproducibility [17]. We evaluate different latent vector sizes r from $r = 10$ to $r = 100$. The general trend is the same across different vector sizes. For the sake of space, we only show the results for $r = 100$.[7]

4.3 Recommendation Accuracy

Table 2 shows recommendation accuracy in Recall@50 and Recall@100 for different datasets. We make the following observations: First, **CML** (without using item features) significantly outperforms other pure CF algorithms (the group on the left) by up to 13.95% in BookCX, Flickr, and Medium datasets. This is mainly due to the fact that **CML**'s metric space can better capture users' preferences as described in Section 3.6. We also observe that **CML** tends to show better performance in the domains where users' personal interests play a major role (e.g., photography, books, etc.). In contrast, for MovieLens and EchoNest, where items' popularity plays a more important role, **CML** shows no significant improvement. This characteristic is also present in the rating concentration ratio of these two datasets (Table 1), where more than 65% of the ratings concentrating on the Top 5% of the items. In general, models using WARP loss (i.e., **WARP** and our **CML**) perform much better than **BPR**, the only exception is CiteULike, the smallest dataset, where **BPR**'s log-sigmoid loss shows better performance.

[7]VBPR's user vectors contain both visual and non-visual parts and are two times larger than that of other models.

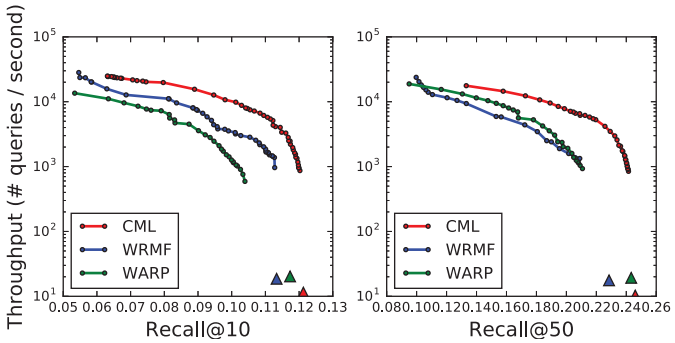

Figure 4: Performance of Top-K recommendation tasks sped up with location-sensitive hashing (LSH). The triangles at the lower right corners represent the performance of brute-force search from the three algorithms.

Secondly, among the hybrid algorithms that utilize item features, **CML+F** shows up to 17.66% improvement over the second best algorithms. We found that **CML+F** along with other algorithms that uses MLP as a feature extractor can more effectively utilize item features than **FM** models that uses latent factors to model the item features. This is particularly evident when item features are noisy (e.g., the user-generated tags). On the other hand, compared to **VBPR** and **CDL**, besides the advantages mentioned earlier, the unit-norm-bounded euclidean space of **CML+F** is easier for neural networks to approximate as opposed to MF models' unbounded dot-product space [30, 43]. The only exception to this is CiteULike dataset, where **CML-F** is slightly outperformed by **CDL** in recall@50.

4.4 Top-K Recommendations with LSH

An important advantage of CML is its capability of significantly speeding up Top-K recommendation tasks with off-the-shelf approximate nearest-neighbor (ANN) algorithms, such as **location-sensitive hashing** (LSH). We demonstrate this advantage with EchoNest data, which is the largest dataset used in this paper, and with a well-known LSH library used in the industry, called *Annoy* [6]. On the other hand, the Top-K recommendation search problem of matrix

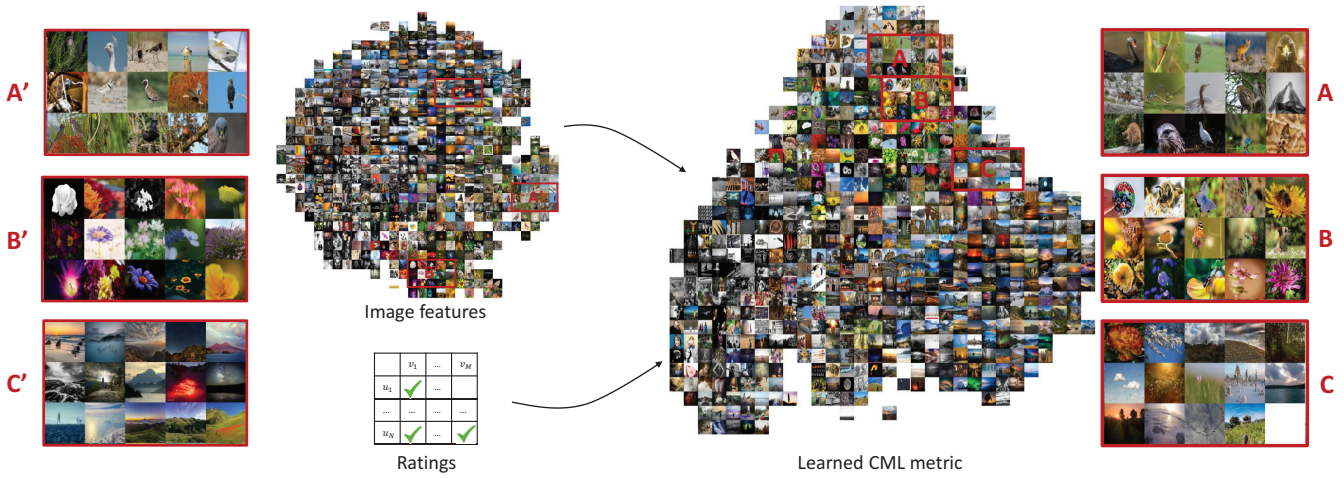

Figure 5: t-SNE embedding of CML metric. CML not only clusters visually-similar objects together but also learns the relevance between objects from users' preferences. For example, CML pulls the images of birds(A), flowers(B), and natural landscape(C) closer while they are far away in the original feature space (left).

factorization is equivalent to the maximum-inner-product search (MIPS) problem, which is known to be tricky to optimize due to the dot product's violation of the triangle inequality [42]. In the following, we compare CML to the MF models sped up with an asymmetric LSH recently proposed by Shrivastava et al. [42], which is the state-of-the-art LSH approach for MIPS[8].

As a standard LSH benchmarking method [6], we sweep a large number of parameter settings for each approach, including the number of LSH trees, the number of buckets to search, and the parameters specifically for MIPS search [42], and only plot the results that are on the accuracy-efficiency frontier[9]. As shown in Figure 4, although the **CML**'s brute-force search is the slowest, **CML** benefits considerably from LSH. For example, **CML** gains 106x and 86x speedup with only 2% of reduction in recall@10 and recall@50. It is also the fastest among the three algorithms given the same accuracy. For example, given recall@10= 0.1, **CML** is more than 8x faster than other MF algorithms. MF algorithms also suffer much larger accuracy reduction for recall@50.

4.5 Metric Visualization

Figure 5 shows the t-SNE embedding of the learned **CML** metric for Flickr images [33]. We observe that CML not only clusters the similar objects together (based on the image features) but also learns to pull images of *relevant* objects closer to reflect users' preferences. For example, CML pulls the images of birds, flowers, and natural landscape closer in the embedding while these images are further away in the original feature space (shown on the left). In addition, we also observe an interesting image preference spectrum: from the embedding's left to its right, we see images from close-up human portrait, to indoor scenes, to cityscapes and to natural landscapes. This spectrum, again, does not exhibit in the original image feature spaces, and is learned by CML to reflect users' fine-grained preferences for photos.

[8]BPR is omitted from this experiment for its low accuracy.
[9]The evaluation is done on an AWS g2.2xlarge instance with Intel Xeon E5-2670 CPU.

5. CONCLUSIONS

In this work, we study the connection between metric learning and collaborative filtering. We propose **collaborative metric learning** (CML), which encodes user-item relationships and user-user/item-item similarity in a joint metric space, and provide a suite of regularization, feature fusion and training techniques to make such a model feasible. We demonstrate CML's superior accuracy over the state-of-the-art collaborative filtering algorithms in a wide range of recommendation domains. We demonstrate how CML's Top-K recommendation tasks can be massively sped up by an off-the-shelf, approximate nearest neighbor, search algorithm, and demonstrate how the visualization of CML uncovers users' fine-grained preferences and the underlying preference spectrum.

By moving from explicit feedback to implicit feedback, the focus of collaborative filtering has also moved from estimating specific set of ratings to capturing users' relative preferences for different items [38]. As an alternative to matrix factorization, our proposed CML algorithm captures such relationships in a more intuitive way and can better propagate such information through user-item pairs.

These results suggest the promising new research direction of applying metric-based algorithms, such as kNN, K-means and SVMs, to collaborative filtering. While we only addressed item features in this paper, future work should consider user features, such as those extracted from individual-usage traces as well [18]. End-to-end metric-based approaches could integrate such data to improve the model's representational capacity and interpretability.

Acknowledgement

We appreciate the anonymous reviewers for their helpful comments and feedback. This research is partly funded by AOL-Program for Connected Experiences, a Google Focused Research Award and further supported by the small data lab at Cornell Tech which receives funding from UnitedHealth Group, Google, Pfizer, RWJF, NIH and NSF.

6. REFERENCES

[1] D. Agarwal and B.-C. Chen. flda: matrix factorization through latent dirichlet allocation. In *WSDM'10*, pages 91–100. ACM, 2010.

[2] Y. Bachrach, Y. Finkelstein, R. Gilad-Bachrach, L. Katzir, N. Koenigstein, N. Nice, and U. Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *RecSys'14*, pages 257–264. ACM, 2014.

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

[4] M. Bawa, T. Condie, and P. Ganesan. Lsh forest: self-tuning indexes for similarity search. In *WWW'05*, pages 651–660. ACM, 2005.

[5] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. *An MIT Press book in preparation.*, 2015.

[6] E. Bernhardsson. Annoy. `https://github.com/spotify/annoy`, 2016.

[7] O. Canévet and F. Fleuret. Efficient sample mining for object detection. In *ACML*, 2014.

[8] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR'05*, volume 1, pages 539–546. IEEE, 2005.

[9] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra. Reducing overfitting in deep networks by decorrelating representations. *ICLR'15*, 2015.

[10] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[11] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[12] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Mymedialite: a free recommender system library. In *RecSys'11*, pages 305–308. ACM, 2011.

[13] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), Dec. 2015.

[14] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

[16] R. He and J. McAuley. Vbpr: visual bayesian personalized ranking from implicit feedback. *arXiv:1510.01784*, 2015.

[17] C.-K. Hsieh. Collaborative metric learning. `https://github.com/changun/CollMetric`, 2016.

[18] C.-K. Hsieh, L. Yang, H. Wei, M. Naaman, and D. Estrin. Immersive recommendation: News and event recommendations using personal digital traces. In *WWW'16*, pages 51–62. International World Wide Web Conferences Steering Committee, 2016.

[19] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM'08*, pages 263–272. Ieee, 2008.

[20] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR'05*, pages 154–161. Acm, 2005.

[21] D. Kedem, S. Tyree, F. Sha, G. R. Lanckriet, and K. Q. Weinberger. Non-linear metric learning. In *NIPS'12*, pages 2573–2581, 2012.

[22] D. Kelly and J. Teevan. Implicit feedback for inferring user preference: a bibliography. In *ACM SIGIR Forum*, volume 37, pages 18–28. ACM, 2003.

[23] M. Khoshneshin and W. N. Street. Collaborative filtering via euclidean embedding. In *RecSys'10*, pages 87–94. ACM, 2010.

[24] N. Koenigstein and Y. Koren. Towards scalable and accurate item-oriented recommendations. In *RecSys'13*, pages 419–422. ACM, 2013.

[25] Y. Koren, R. Bell, C. Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[26] M. Köstinger, M. Hirzer, P. Wohlhart, P. M. Roth, and H. Bischof. Large scale metric learning from equivalence constraints. In *CVPR'12*, pages 2288–2295. IEEE, 2012.

[27] M. Kula. Metadata embeddings for user and item cold-start recommendations. *arXiv:1507.08439*, 2015.

[28] M. Kula. Lightfm. `https://github.com/lyst/lightfm`, 2016.

[29] B. Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2012.

[30] B. Kumar, G. Carneiro, and I. Reid. Learning local image descriptors with deep siamese and triplet convolutional networks by minimising global loss functions. *arXiv:1512.09272*, 2015.

[31] G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[32] W. Liu and I. W. Tsang. Large margin metric learning for multi-label prediction. In *AAAI'15*, pages 2800–2806, 2015.

[33] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[34] B. McFee, T. Bertin-Mahieux, D. P. Ellis, and G. R. Lanckriet. The million song dataset challenge. In *WWW'12*, pages 909–916. ACM, 2012.

[35] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *ICDM'08*, pages 502–511. IEEE, 2008.

[36] P. Ram and A. G. Gray. Maximum inner-product search using cone trees. In *KDD'12*, pages 931–939. ACM, 2012.

[37] S. Rendle. Factorization machines. In *ICDM'10*, pages 995–1000. IEEE, 2010.

[38] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI'09*, pages 452–461. AUAI Press, 2009.

[39] S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM'10*, pages 81–90. ACM, 2010.

[40] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW'94*, pages 175–186. ACM, 1994.

[41] G. Shani and A. Gunawardana. Evaluating recommendation systems. In *Recommender systems handbook*, pages 257–297. Springer, 2011.

[42] A. Shrivastava and P. Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *NIPS'14*, pages 2321–2329, 2014.

[43] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR'14*, pages 1701–1708, 2014.

[44] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv:1605.02688*, 2016.

[45] J. Wan, D. Wang, S. C. H. Hoi, P. Wu, J. Zhu, Y. Zhang, and J. Li. Deep learning for content-based image retrieval: A comprehensive study. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 157–166. ACM, 2014.

[46] C. Wang and D. M. Blei. Collaborative topic modeling for recommending scientific articles. In *KDD'11*, pages 448–456. ACM, 2011.

[47] H. Wang, N. Wang, and D.-Y. Yeung. Collaborative deep learning for recommender systems. In *KDD'15*, pages 1235–1244. ACM, 2015.

[48] J. Wang, H. T. Do, A. Woznica, and A. Kalousis. Metric learning with multiple kernels. In *NIPS'11*, pages 1170–1178, 2011.

[49] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009.

[50] J. Weston, S. Bengio, and N. Usunier. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine learning*, 81(1):21–35, 2010.

[51] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning with application to clustering with side-information. *NIPS'03*, 15:505–512, 2003.

[52] Z. E. Xu, M. Chen, K. Q. Weinberger, and F. Sha. From sbow to dcot marginalized encoders for text representation. In *CIKM'12*, pages 1879–1884. ACM, 2012.

[53] T. Zhao, J. McAuley, and I. King. Improving latent factor models via personalized feature projection for one class recommendation. In *CIKM'15*, pages 821–830. ACM, 2015.

[54] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW'05*, pages 22–32. ACM, 2005.