Clustering and Constructing User Coresets to Accelerate Large-scale Top-K Recommender Systems

Department of Computer Science, University of California, Los Angeles, CA, USA
{junyu,patrickchen,chohsieh,weiwang}@cs.ucla.edu

ABSTRACT
Top-K recommender systems aim to generate few but satisfactory personalized recommendations for various practical applications, such as item recommendation for e-commerce and link prediction for social networks. However, the numbers of users and items can be enormous, thereby leading to myriad potential recommendations as well as the bottleneck in evaluating and ranking all possibilities. Existing Maximum Inner Product Search (MIPS) based methods treat the item ranking problem for each user independently and the relationship between users has not been explored. In this paper, we propose a novel model for clustering and navigating for top-K recommenders (CANTOR) to expedite the computation of top-K recommendations based on latent factor models. A clustering-based framework is first presented to leverage user relationships to partition users into affinity groups, each of which contains users with similar preferences. CANTOR then derives a coreset of representative vectors for each affinity group by constructing a set cover with a theoretically guaranteed difference to user latent vectors. Using these representative vectors in the coreset, approximate nearest neighbor search is then applied to obtain a small set of candidate items for each affinity group to be used when computing recommendations for each user in the affinity group. This approach can significantly reduce the computation without compromising the quality of the recommendations. Extensive experiments are conducted on six publicly available large-scale real-world datasets for item recommendation and personalized link prediction. The experimental results demonstrate that CANTOR significantly speeds up matrix factorization models with high precision. For instance, CANTOR can achieve 355.1x speedup for inferring recommendations in a million-user network with 99.5% precision@1 to the original system while the state-of-the-art method can only obtain 93.7x speedup with 99.0% precision@1.

KEYWORDS
Large-scale top-K recommender systems; Latent factor models; Approximate nearest neighbor search

1 INTRODUCTION
Building large-scale personalized recommender systems has already become a core problem in many online applications since the explosive growth of internet users in the recent decade. For example, user-item recommender systems achieve many successes in e-commerce markets [23] while link prediction in social networks can be treated as a variant of recommender systems [2, 33]. To establish recommender systems, latent factor models for collaborative filtering have become popular because of their effectiveness and simplicity. More precisely, each user or item can be represented as a low-dimensional vector in a latent space so that the inner products between user and item vectors are capable of indicating the user-item preferences. Furthermore, these latent vectors can then be learned by optimizing a loss function with sufficient training data. For instance, matrix factorization [19] has been empirically shown to outperform conventional nearest-neighbor based approaches in a wide range of application domains [11].

After obtaining user and item latent vectors, to make item recommendations for each user, recommender systems need to calculate the inner products for all user-item pairs. Although learning user and item latent vectors is efficient and scalable for most existing models, recommender systems can take an enormous amount of time in evaluating all user-item pairs. More specifically, the time complexity of learning latent vectors is only proportional to the number of user-item pairs in the training data which is a small subset of all possible user-item pairs, but finding the top recommendations entails examining all \( O(mn) \) inner products between all \( m \) users and \( n \) items. As a result, the quadratic complexity becomes a hurdle for large-scale recommender systems. For example, it can take more than a day to compute and rank all preference scores, and consequently the systems cannot be updated on a daily basis [12]. In order to make large-scale recommender systems practical, it is critical to accelerate the process of computing and ranking the inner products of user and item latent vectors, in order to efficiently obtain the top-K recommendations for all users.

To accelerate the computation of inner products, the maximum inner product search (MIPS) [27, 31, 34] is one of the feasible approaches. Locality sensitive hashing (LSH) [16] and PCA tree [32] may be applied to solve MIPS after reducing the problem to nearest-neighbor search. To reduce the computation for making recommendations for a given user, one may find a small group of candidate items whose latent vectors have large inner products with the user’s latent vector using clustering algorithms [7], or sort entries of each dimension in the latent vectors separately by some greedy algorithms [12, 34]. In essence, most of the existing MIPS algorithms

This paper is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.

WWW '20, April 20–24, 2020, Taipei, Taiwan. ACM, New York, NY, USA, 11 pages.
https://doi.org/10.1145/3366623.3380283
adopt a two-stage strategy, decomposing the computation into a preparation process and a prediction process. In the preparation stage, these methods will construct suitable data structures [34] or reduce the number of ranking candidates [7], and these prepared data structures are used to conduct efficient maximum inner product search for query vectors in the inference stage. However, most of these existing MIPS algorithms have the following two weaknesses, making them often impractical for real applications: (1) they only focus on optimizing the inference speed for a given user at the cost of considerable preparation time, but for recommender systems, the overall execution time (including both preparation and inference time) matters more because the system needs to be re-trained frequently as new data arrive. (2) All the MIPS approaches aim to quickly identify the top item set for any query vector. However, in recommender systems queries are not arbitrary vectors. They are user latent factors and usually have very strong clustering structure, which is ignored in most of the MIPS algorithms.

In order to speed up the overall execution time, our main idea is to exploit the relationships between users. More precisely, users with similar latent factors are more likely to share similar item preferences which may be reflected by their high inner products. However, existing methods for accelerating recommender systems do not consider user relationships and the distribution of user latent vectors. For instance, existing greedy strategies [12, 34] only consider the values of item latent vectors. Some studies based on proximity graphs [26, 35] and clustering algorithms [7] also solely reduce the search space of items. In the inference stage, these approaches treat the recommendation to each user as an independent query to the data structures and algorithms. As a consequence, it can be extremely time-consuming, especially with myriad users and enormous spaces of candidate items.

In this paper, we propose a novel model for clustering and navigating for top-K recommenders (CANTOR) that leverages the knowledge of user relationships to accelerate the process of generating recommendations for all users with a given latent factor model. CANTOR consists of two stages: preparation and prediction. In the preparation stage, we aim to cluster users sharing similar interests into affinity groups and compute a small set of preferred items for each affinity group. More specifically, the user vectors (generated from a given latent factor model) are used in clustering affinity groups. To further accelerate the preparation time, a user coreset of few representative vectors are derived for each affinity group, and are used to obtain a small set of preferred items for users in this group by an efficient approximate nearest neighbor search algorithm. Finally, in the prediction stage, the top-K recommendations for each user can be retrieved by ranking these preferred items of the corresponding affinity group, which can be done much more efficiently than evaluating and ranking all items. We summarize our contributions as follows:

- Clustering users into affinity groups based on the distribution of user latent factors provides significant speedup of the prediction process, compared to conventional approaches that independently deal with each user. The representative vectors of the affinity groups offer a theoretically guaranteed precision for users with similar preferences. Approximate nearest neighbor search is applied to efficiently retrieve the satisfactory recommendations for each user from a small set of candidate items.
- Experiments conducted on six publicly available datasets demonstrate that CANTOR can significantly accelerate large-scale top-K recommender systems for both item recommendation and personalized link prediction. An in-depth analysis then indicates the robustness and effectiveness of the proposed framework.

2 PROBLEM STATEMENT

In this section, we first introduce the notations and then formally define the objective of this paper. Suppose that we have an incomplete $m \times n$ one-class matrix $R = \{R_{ij}\} \in \{0, 1\}^{m \times n}$, where $m$ and $n$ are the numbers of users and items in the system. $R_{ij} = 1$ if user $i$ prefers item $j$ in the training data; otherwise, $R_{ij} = 0$. Based on $R$, a matrix factorization based algorithm learns $d$-dimensional user and item latent vectors, denoted by $P \in \mathbb{R}^{m \times d}$ and $Q \in \mathbb{R}^{n \times d}$ respectively, where $\hat{R} = PQ^T \in \mathbb{R}^{m \times n}$ reflects the underlying preferences. To compute top-$K$ recommendations for each user, we need to find items with the $K$ highest scores among $\hat{R}(i) = \{\hat{R}_{ij} | j' \in 1 \ldots m\}$. Note that $m = n$ for personalized link prediction in social networks, where the goal is to suggest other users as recommended items.

Although matrix factorization models can be learned expeditiously when $R$ is sparse, inferring the top-$K$ recommendations requires computing and sorting the scores $\hat{R}_{ij}$ of all items $j$ for each user $i$. As a result, the inference process can be time-consuming with an $O(ndm)$ time complexity which becomes intractable when $n$ and $m$ are large. To address this problem, the goal of this paper is to speed up the inference time of top-$K$ recommenders with a high precision. More specifically, given the trained matrices $P$ and $Q$, we aim to propose an efficient approach that approximates the top-$K$ recommended items for each user.

3 CONSTRUCTING USER CORESETS FOR TOP-K RECOMMENDER SYSTEMS

In this section, we present CANTOR for accelerating top-$K$ recommender systems, starting with several key preliminary ideas.

3.1 Preliminary

In order to leverage the relationship between users, we first formally define the affinity groups of users in recommender systems as follows:

**Definition 1.** (Affinity Group) An affinity group $A_x$ is a set of users sharing similar interests in items. Even though any similarity metrics may be used in this paper, we adopt cosine similarity as the metric to define the affinity groups.

By this definition, the sets of satisfactory recommendations should be similar for users in the same affinity group. This suggests that the top recommendations for all users in an affinity group are confined to a small subset of the items and such item subset can be learned by
examining only a few carefully selected users in the group, leading to the following definition of the preferred item set.

**Definition 2.** (Preferred Item Set) A preferred item set $c$ for an affinity group is a set of (potentially) satisfactory items for the users in the group, and the size of the preferred item set is usually much smaller than the total number of items, i.e., $|c| \ll n$.

Therefore, we only need to examine the preferred item set to generate top recommendations, leading to significant time saving over the alternative of examining all items.

In order to robustly generate the preferred item set for each affinity group, we generate a few representatives from the group to compute the preferred item set. This is statistically more robust than using only the “centroid” user in the latent space, and is more computationally efficient than using all users in the group.

**Definition 3.** (User Coreset of an Affinity Group) A $\delta$-user coreset $s_i$ of an affinity group $A_t$ is a (small) set of latent representative vectors to preserve the item preference of the users in $A_t$ such that $\forall q \in Q, i \in A_t$:

$$\left| p_i q^T - N_{s_i}(p_i) q^T \right| \leq \delta,$$

where $N_{s_i}(p_i) \in s_i$ is the nearest coreset representative for $p_i$; $\delta > 0$ is a small enough constant.

The user interests in the affinity group can be well captured by the representative vectors in the user coreset. Note that the representative vectors do not have to be identical to actual user latent vectors in the group.

### Table 1: Summary of notations and their descriptions.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m, n$</td>
<td>numbers of users and items</td>
</tr>
<tr>
<td>$d$</td>
<td>number of dimensions for latent vectors</td>
</tr>
<tr>
<td>$K$</td>
<td>number of top recommendations</td>
</tr>
<tr>
<td>$R \in \mathbb{R}^{m \times n}$</td>
<td>one-class preference matrix</td>
</tr>
<tr>
<td>$P \in \mathbb{R}^{m \times d}$</td>
<td>user latent vectors for all users</td>
</tr>
<tr>
<td>$Q \in \mathbb{R}^{n \times d}$</td>
<td>item latent vectors for all items</td>
</tr>
<tr>
<td>$\hat{P} \in \mathbb{R}^{c \times d}$</td>
<td>sampled user latent vectors</td>
</tr>
<tr>
<td>$u_t$</td>
<td>number of sub-sampled users</td>
</tr>
<tr>
<td>$r$</td>
<td>number of affinity groups for $m$ users</td>
</tr>
<tr>
<td>$A$</td>
<td>set of $r$ affinity groups, where $A = { A_t</td>
</tr>
<tr>
<td>$A_i$</td>
<td>centroid vectors for the affinity group $A_t$</td>
</tr>
<tr>
<td>$z(p)$</td>
<td>affinity group indicator for the user vector $p$</td>
</tr>
<tr>
<td>$P_t$</td>
<td>latent vectors of users in the affinity group $A_t$</td>
</tr>
<tr>
<td>$C(p, K)$</td>
<td>indexes of top-$K$ items with full $p^T Q$ evaluation.</td>
</tr>
<tr>
<td>$s_i$</td>
<td>the user coreset for the affinity group $A_t$</td>
</tr>
<tr>
<td>$N_{s_i}(p_i)$</td>
<td>the nearest coreset representative in $s_i$ for $p_i$</td>
</tr>
<tr>
<td>$c_t$</td>
<td>reduced item set of top-$K$ items for the affinity group $A_t$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>similarity threshold in adaptive representative selection</td>
</tr>
<tr>
<td>$w$</td>
<td>number of new representatives for outliers</td>
</tr>
<tr>
<td>$G$</td>
<td>proximity graph of the item vectors</td>
</tr>
<tr>
<td>$cfs$</td>
<td>the size of dynamic lists of nearest neighbors</td>
</tr>
</tbody>
</table>

Figure 1 shows the general framework of CANTOR. The framework consists of two stages: preparation and prediction. In the preparation stage as shown in Algorithm 1, the $m$ user latent vectors $P$ are first sub-sampled as $\hat{P}$ and clustered into $r$ affinity groups $A_t$ with a centroid vector $s_t$ computed from the corresponding user vectors $P_t$, where $t = 1 \ldots r$. For each affinity group $A_t$, we aim at deriving a small user coreset $s_t$. To do so, we propose an adaptive representative selection method (Algorithm 2) to greedily construct a set cover of user latent vectors for each affinity group after mathematically proving that the set covers can be the coresets of affinity groups. Finally, a small preferred item set $c_t$ can be obtained by approximate nearest neighbor search using its coreset $s_t$ for each affinity group. In the prediction stage (Algorithm 4), CANTOR first locates the corresponding affinity group $A_t$ for each user and then ranks the small number of items in the preferred item set $c_t$, thereby efficiently providing satisfactory recommendations. In sum, Table 1 further summarizes all major notations in this paper.

### 3.2 Framework Overview

To overcome the hurdle of extremely long preparation time experienced by conventional methods, we propose to exploit similarities between user vectors in the latent space for acceleration as shown in Algorithm 1.

**Affinity Group Modeling by User Clustering.** Most of the conventional algorithms only rely on similarities of item latent vectors [30] and proximity graphs [14, 35] to accelerate the recommendation, and have not used the relationships between users and
Algorithm 1: Preparation Process for CANTOR

Input: User latent vectors \( P \); item latent vectors \( Q \); degree of each user \( \text{deg}_i \); the number of desired recommendation \( K \).

Output: Centroid vectors \( \mathbf{c}_t \) and preferred item sets \( \mathbf{c}_t \) for each affinity cluster \( A_t \) for \( t = 1 \ldots r \).

Hyperparameter: Number of affinity groups \( r \); small world graph search size \( efs \); number of sub-sampled users \( u \);
\( \hat{P} = \text{Multinomial} \_\text{Sampling}(P, \text{deg}_i, u); \hat{P} \in \mathbb{R}^{u \times d} \)
\( \mathbf{c}_1, \ldots, \mathbf{c}_r = 0; I = 0, I \in \mathbb{R}^{n \times 1} \)

repeat
\( i = 1, \ldots, r \) do
\( v_i = \sum_{j \in \{j \mid i = j \}} \hat{P}[j] \);
\( v_i = v_i / \|v_i\|_2 \);
\( I = \max_2 \alpha_i^T \hat{P} \);
until Convergence;

\( G = \text{CreateProximityGraph}(Q, efs) \);
\( c_1, \ldots, c_r = 0, \ldots, 0 \);
\( I = \max_2 \alpha_i^T \hat{P} \);
for \( i = 1, \ldots, r \) do
\( \hat{P}_i = \{p_j \mid j \in \hat{P}, 1 \leq j \leq i \} \);
\( s_i = \text{AdaptiveClustering}(\hat{P}_i, e, w) \);
for \( q \in s_i \) do
\( \hat{I}_i = \text{QueryProximityGraph}(G, s, K) \);
\( c_t = \hat{c}_t \cup \hat{I}_i \);

return \( c_t, \mathbf{v}_t \) for all \( t = 1, \ldots, r \).

Since our task is to accelerate the maximum inner product search, the centroid vector \( \mathbf{c}_t \) for each affinity group \( A_t \) can then be updated by the maximum cosine similarity criteria as:

\[
\mathbf{u}_t = \frac{\sum_{l=1}^{\left\| P_l \right\|} P_{l i}}{\left\| \sum_{l=1}^{\left\| P_l \right\|} P_{l i} \right\|_2},
\]

where \( P_l = \{p_j \mid z(p_j) = t \} \) contains the latent vectors of users that belong to the affinity group \( A_t \). Therefore, each affinity group \( A_t \) can obtain a centroid vector \( \mathbf{c}_t \) by iteratively running Equations (1) and (2). However, iteratively performing Equations (1) and (2) can still cost a long computational time when the number of users \( m \) is large. To address this issue, we propose to sub-sample a portion of the \( m \) user latent vectors to learn the centroid vectors. Moreover, we sample the latent vectors based on the degree distribution in the one-class matrix \( R \). For example, Figure 2a shows that degree distribution of users usually follows a power-law distribution. Hence, instead of using a uniform sampling, we sample user \( i \) with a probability proportional to a log function of its degree as:

\[
P(X = i) \propto \log \sum_{j=1}^{n} R_{ij}.
\]

where \( X \) denotes the random variable of the target sampling process. We will later show in Theorem 2 that error of approximation based on sub-sampling will be asymptotically bounded.

After learning the centroids \( \mathbf{c}_1, \ldots, \mathbf{c}_r \in \mathbb{R}^d \) and the corresponding user latent vectors \( P_1, \ldots, P_r \) for \( r \) affinity groups \( A_1, \ldots, A_r \), the preferred item set \( \mathbf{c}_t \) for each group \( A_t \) can be constructed so that user vectors \( P_t \) only need to search over this set of preferred items for top recommendations. However, the naive approach to generate \( c_t \) would require \( O(n d) \) operations to examine all \( n \) items in order to derive the set candidates for each user in \( A_t \). Each affinity group \( A_t \) would need \( O(|P_t| n d) \) operations for considering all \( |P_t| \) users in the group to construct the preferred item set \( c_t \).

Coreset Construction as Finding a Set Cover. To accelerate the process of constructing the preferred item set \( c_t \) for an affinity group \( A_t \), we want to find a \( \delta \)-user coreset of \( A_t \), and use it only instead of whole \( A_t \) to construct \( c_t \). We achieve this by first defining the idea of \( \epsilon \)-set cover, and then show that each \( \epsilon \)-set cover corresponds to a \( \delta \)-coreset.
For an affinity group

Given an

The proof is shown in Appendix A. Therefore, we could construct a user coreset with an arbitrarily small δ by finding a cover with a greater ε.

Another nice property is that we could find an ε-cover of the state-of-the-art performance in ANNS. Specifically, a proximity graph navigation [26, 35] which has shown to be same sampling process of P generating user vectors p1 belonging to A_i. We will have following result:

Theorem 2. For an affinity group A_i, given any query q, an ε-cover of k samples {p_i} drawn from P_{A_i} would satisfy following inequality with probability at least 1 - γ:

\[ \min_{q} \left( \|N_{A_i}(p_i)q^T - p_iq^T\| \right) \leq \delta + \sqrt{\frac{2\log(1/\gamma)}{k}}. \]

Note that we demonstrate the proof in Appendix B.

Theorem 2 indicates that we could construct an ε-cover of sub-sampled vectors to have an asymptotically guaranteed difference of inner-product values to true distributions within the same affinity group. Consequently, our task becomes finding an ε-cover of all A_i's and constructing the preferred item set c_i of it. Hence, we propose a fast adaptive representative selection method to efficiently construct an ε-cover with sub-sampled user latent vectors for each affinity group as summarized in Algorithm 2. For each affinity group A_i, the adaptive representative selection method is applied to obtain a few representative ε-cover s_i. If there exists at least one user whose latent vector has cosine similarity lower than ε to all representative vectors, the algorithm iteratively generates more representatives until every user has high cosine similarity to at least one representative vector. As a result, the number of ε-cover |s_i| must be less than or equal to the number of user vectors in the cluster |P_i|, and in practice, |s_i| ≪ |P_i| in most cases. Note that the adaptive representative selection method is applied on each affinity group A_i independently. Next, the ε-cover s_i will be utilized to construct the preferred item set to reduce complexity from O(|P_i|nd) to O(|s_i|nd).

Proximity Graph Navigation for Preferred Item Set Construction. To avoid examining all n items (O(nd) complexity) in preferred item set construction, we apply an approximate nearest neighbor search (ANNS) method to accelerate the computation. We adopt a model based on proximity graphs [26, 35] which has shown the state-of-the-art performance in ANNS. Specifically, a proximity graph is generated in which item vectors are nodes and nodes of similar item vectors are connected by edges. Since the item degree in recommender systems tends to follow a power-law distribution as illustrated in Figure 2b, this proximity graph has the small world properties [5] with sparse edges that offer high reachability between nodes. Hence, we apply the model of hierarchical navigable small world graphs [25, 26] to obtain the preferred item set c_i for each affinity group A_i.

Algorithm 2: Adaptive Representative Selection

**Input:** User latent vectors for an affinity group P, the number of iterations T, the threshold ε, the number of new representatives w

**Output:** Representative vectors s

1. Initialize s = ∅;
2. I = \arg \max_i s^T P;
3. repeat
   4. for i = 1 . . . |s| do
      5. \[ s_i = \sum_{j \in \{j | |j-i| = i\}} P[j]; \]
      6. \[ s_i = s_i / \|s_i\|; \]
      7. I = \arg \max_i s^T P;
      8. Outliers = \{j | s_i^T P[j] < ε\};
      9. for j ∈ Outliers do
         10. Draw i from 1 . . . w;
         11. I[j] = |s| + i;
         12. if Outliers ≠ ∅ then
            13. Append w vectors to s;
      14. until Outliers = ∅;
      15. Outliers = \{j | s_i^T P[j] < ε\};
16. Append P_{Outliers} to s;
17. return s.

To construct the proximity graph of item vectors Q as a hierarchical small world graph G, we iteratively insert the item vectors into the graph, where each node q has a list E(q) of at most efs approximate nearest neighbors that could be dynamically updated when inserting other item vectors, where efs is a hyperparameter. In addition, the edges in the graph are organized as a hierarchy so that edges connecting items that have a high inner product value of their corresponding item vectors are at the bottom layers and edges connecting item vectors that have low inner product values are at the top layers, thereby shrinking the search spaces for nearest neighbors. Let L(e) denote the corresponding layer of edge e. Given two edges e_i and e_j, if L(e_i) > L(e_j), then the nodes connected by edge e_i has a smaller inner product score than that of edge e_j.

For simplicity, let E(q, l) denote the list of nodes connected to node q by edges in the l-th layer. Finally, the hierarchical small world graph G of item vectors Q can be constructed in O(dn log n) [26, 35], where n is the total number of items; efs is treated a constant hyperparameter. Note that efs controls the trade-off between efficiency and accuracy for searching nearest neighbors because it decides the size of search space and the potential coverage of real nearest neighbors.

The hierarchical small world graph G provides the capability of efficiently querying K nearest neighbors of a vector q with a hierarchical greedy search algorithm. More specifically, we can greedily traverse the graph G by navigating the query vector from the bottom layer to the top layer to derive K approximate nearest neighbors to q as shown in Algorithm 3 with a O(d log n) time complexity for each query. For each affinity group A_i, we perform a small world graph query to approximate C(s_i, K) for each representative vector s_i ∈ s_i. The preferred item set c_i can then be constructed by taking the union operation to individual top-K sets.
Algorithm 3: QueryProximityGraph

**Input:** Hierarchical small world graph G; the query vector \( q \); the number of the output approximate nearest neighbors \( K \)

**Output:** \( K \) nearest vectors in \( G \)

1. \( p = \) Randomly select an entry node in \( G \) ;
2. For \( l = 1 \) to \( L \) do
3. \( q = \arg \max_{r \in \{p'|\{p\} \cap E(p,l) \neq \emptyset\} } q^T r; \)
4. return \( K \) Nearest Nodes in \( E(p,L) \) to \( q \) ;

Algorithm 4: Prediction Process for CANTOR

**Input:** User latent vectors \( p_i \); item latent vectors \( Q \); Number of top recommendations \( K \)

**Output:** The indices of estimated top-\( K \) recommendations for the user \( i \).

\[ z(p_i) = \arg \max_r u_r^T p_i; \]
\[ \text{logits} = p_i^T Q \left[ c_{1}(p_i) \right]; \]
\[ \text{topIndices} = \text{argsort}(\text{logits}, K); \]

\[ \text{return} \text{topIndices}. \]

as

\[ c_t = \bigcup_{i=1}^{s_t} C(s_t, i, K). \]  

(4)

### 3.4 Prediction Stage

To predict top recommendations for a user with the latent vector \( p_i \), CANTOR relies on the clustering model parameterized by the centroid vector \( u_t \in \mathbb{R}^d \) and the preferred item set \( c_t \) for each affinity group \( A_t \). More precisely, we first compute the affinity group indicator \( z(p) \) as:

\[ z(p_i) = \arg \max_r u_r^T p_i, \]  

(5)

and evaluate full vector matrix product \( p^T Q_I \) over the corresponding item vectors of the preferred item set \( Q_I, I = \{ j | j \in c_{e_i(p_j)} \} \). The computed results are then sorted to provide the final top-\( K \) recommendations for the user. Algorithm 4 shows the procedure of the prediction process.

## 4 EXPERIMENTS

In this section, we conduct extensive experiments and in-depth analysis to demonstrate the performance of CANTOR.

### 4.1 Experimental Settings

**Experimental Datasets.** We evaluate the performance in two common tasks: item recommendation and personalized link prediction, using six publicly available real-world large-scale datasets as shown in Table 2. For the task of item recommendation, the MovieLens 20M dataset (MovieLens) [15] consists of 20-million ratings between users and movies; the Last.fm 360K dataset (Last.fm) [6] contains the preferred artists of about 360K users; the dataset of Amazon ratings (Amazon) includes ratings between millions of users and items [20]. For the task of personalized link prediction, we follow the previous study [12] to construct three social networks among users: YouTube, Flickr, and Wikipedia [20]. Note that four of the six experimental datasets, Amazon, YouTube, Flickr, and Wikipedia, are available in the Koblenz Network Collection [20].

**Evaluation Metrics.** To measure the quality of an approximate algorithm for top-\( K \) recommendation we evaluate the top-\( K \) approximated recommendations with Precision@\( K \) (P@\( K \)), which is defined by

\[ \frac{1}{m} \sum_{i} \frac{|Y_i^K \cap S_i^K|}{K}, \]

where \( Y_i^K \) and \( S_i^K \) are the top-\( K \) items recomputed by the approximate algorithm and full inner-product computations for user \( i \); \( m \) is the number of users. To measure the speed of each algorithm, we report the speedup defined by the ratio of wall clock time consumed by the full set of \( O(mn) \) inner products to find the top-\( K \) recommendations divided by the wall clock time of the approximate algorithm.

**Baseline Methods.** To evaluate our proposed CANTOR, we consider the following five algorithms as the baseline methods for comparison.

- \( \epsilon \)-approximate link prediction (\( \epsilon \)-Approx) [12] sorts entries of the latent factor for each dimension to construct a guaranteed approximation of full inner products.
- Greedy-MIPS (GMIPS) [34] is a greedy algorithm for solving the MIPS problem with a trade-off controlled by varying a computational budget parameter in the algorithm.
- SVD-softmax (SVDS) [30] is a low-rank approximation approach for fast softmax computation. We vary the rank of SVD to control the latent factor for each dimension to construct a guaranteed \( \epsilon \)-approximation of full inner products.
- Fast Graph Decoder (FGD) [35] directly applies small world graph on all items \( Q \) and navigates to derive recommended items with user latent vectors as queries on the proximity graph. It also serves a direct baseline of only using proximity graph navigation.
- Learning to Screen (L2S) [7] is the first clustering-based method on fast prediction in NLP tasks with the state-of-the-art results on inference time but suffers from long preparation time. CANTOR is inspired by the clustering step in L2S, thus L2S serves as a

<table>
<thead>
<tr>
<th>Task</th>
<th>Item Recommendation</th>
<th>Dataset</th>
<th>MovieLens</th>
<th>Last.fm</th>
<th>Amazon</th>
</tr>
</thead>
<tbody>
<tr>
<td>#(Users)</td>
<td>138,493</td>
<td>359,293</td>
<td>2,146,057</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#(Items)</td>
<td>26,744</td>
<td>160,153</td>
<td>1,230,915</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Task | Personalized Link Prediction | Dataset | YouTube | Flickr | Wikipedia |
|------|--------------------------------|---------|--------|--------|
| #(Users) | 1,503,841 | 1,580,291 | 1,682,759 |
| #(Items) | 1,503,841 | 1,580,291 | 1,682,759 |
Table 3: Comparisons of top-$K$ recommendation results on six datasets in two tasks. Note that P@$K$ measures the precision of approximating the top-$K$ recommendations of full inner-product computations. SU indicates the ratio of speedup based on the original full inner product time of inferring top-$K$ recommendations. For example, 9.4x means the computation time of the method is 9.4 times faster than the full inner product computation time. PT means the preparation time and IT represents the inference time in prediction process. The time units of seconds, minutes, and hours are represented as s, m, and h, respectively. Computation time of the full inner product method for each dataset is 71s (MovieLens), 1,017s (Last.fm), 92,828s (Amazon), 56,824s (Youtube), 71,653s (Flickr), and 72,723s (Wikipedia).

<table>
<thead>
<tr>
<th>Task</th>
<th>Item Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MovieLens</td>
</tr>
<tr>
<td></td>
<td>SU  PT  IT  P@1  P@5</td>
</tr>
<tr>
<td>e-Approx</td>
<td></td>
</tr>
<tr>
<td>GMIPS</td>
<td>0.7x  0.19x  99.00s  0.753  0.671</td>
</tr>
<tr>
<td>SVDS</td>
<td>3.9x  N/A  18.41s  1.000  0.972</td>
</tr>
<tr>
<td>FGD</td>
<td>1.0x  0.10x  69.00s  1.000  1.000</td>
</tr>
<tr>
<td>L2S</td>
<td>2.8x  4.94s  20.10s  1.000  0.999</td>
</tr>
<tr>
<td>CANTOR</td>
<td>9.4x  6.17s  1.000  1.000  0.999</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>Personalized Link Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YouTube  Flickr  Wikipedia</td>
</tr>
<tr>
<td></td>
<td>SU  PT  IT  P@1  P@5</td>
</tr>
<tr>
<td>e-Approx</td>
<td></td>
</tr>
<tr>
<td>GMIPS</td>
<td>0.1x  0.3m  129.2h  0.364  0.432</td>
</tr>
<tr>
<td>SVDS</td>
<td>1.4x  N/A  11.12h  0.987  0.965</td>
</tr>
<tr>
<td>FGD</td>
<td>1.0x  0.03m  15.30h  0.965  0.963</td>
</tr>
<tr>
<td>L2S</td>
<td>44.8x  10.28m  80.5m  0.989  0.981</td>
</tr>
<tr>
<td>CANTOR</td>
<td>6.9x  135.93m  0.79m  0.984  0.968</td>
</tr>
<tr>
<td></td>
<td>112.7x  7.75m  0.63m  0.993  0.985</td>
</tr>
</tbody>
</table>

In our implementation, random sub-sampling is applied to choose a subset of users for training L2S. Note that [34] has shown that Greedy-MIPS outperforms other MIPS algorithms including LSH-MIPS [27, 31], Sampling MIPS [3] and PCA-MIPS [1], so we omit those other MIPS algorithms in our comparisons. Although bandit-based methods [13, 21, 22] have elegant mathematical properties and theoretical bounds, we did not include them originally because they generally perform worse than other methods in practical cases. For example, SCLUB [21], which is one of the state-of-the-art bandit-based approaches, only achieves 0.81x and 0.62x speedups on the Amazon and Wikipedia datasets respectively. SVDS can efficiently decompose the preference matrix as its preparation process is slow so that the overall speedup is further degraded. SVDS search to retrieve recommendations for each user. Although L2S is the state-of-the-art algorithm for approximate nearest neighbor retrieval, it is a direct baseline. In our implementation, random sub-sampling is applied to choose a subset of users for training L2S.

### Implementation Details

For each dataset, the LIBMF library [8] is used to train a non-negative MF (NMF) model. More specifically, the number of dimensions for latent vectors is 10 while the models are trained with all data for 100 iterations. Note that we adopt NMF models because of the restrictions of e-Approx, but CANTOR does not have any limitation on matrix types. We implement CANTOR in Python with NumPy optimized by BLAS [4]. For the baseline methods, the implementations of GMIPS, SVDS, FGD, and L2S are provided by the original authors and highly-optimized while we utilize an efficient C++ implementation of e-Approx. All experiments were run on a 64-bit Linux Ubuntu 16.04 server with 512 GB memory and single thread regime on an Intel® Xeon® CPU E5-2698 v4 @2.2 GHz.

### Hyperparameters in CANTOR

For the general settings of hyperparameters in CANTOR, we fix the number of sub-sampled user latent vectors $r$ as 50,000 and the number of clusters $\epsilon$ as 8. For adaptive representative selection, we set the number of iterations in the adaptive selection $T$ as 10 and the similarity threshold $\epsilon$ as 0.99. The number of new representatives $w$ in adaptive representative selection algorithm is set as 8. We also tune the size of dynamic nearest neighbor lists $\beta$ in the construction of hierarchical small graphs for each dataset to achieve acceptable accuracy scores. As a result, the selections of $\beta$ are 200, 200, 1,500, 500, 1,500, and 100 for the datasets MovieLens, Last.fm, Amazon, YouTube, Flickr, and Wikipedia, respectively.

### 4.2 Performance Comparison

To fairly compare the performance, for each dataset, we tune the parameters such that each method can roughly achieve 0.99 P@1 accuracy. Table 3 shows the efficiency and the precision scores of CANTOR and all baseline methods on six datasets. Note that since the open-sourced library of GMIPS does not provide the breakdown of execution time into preparation and prediction time, the reported time includes both preparation and prediction processes. Among the baseline methods, FGD performs the best because it exploits the state-of-the-art algorithm for approximate nearest neighbor search to retrieve recommendations for each user. Although L2S is the most efficient baseline in the inference process, its preparation process is slow so that the overall speedup is further degraded. SVDS can efficiently decompose the preference matrix as its preparation process, but it still requires to examine all items many times to achieve sufficient accuracy so that the acceleration is unsatisfactory. In addition, it is worth noting that, although e-Approx theoretically needs fewer multiplications than the full evaluation, it actually does not provide any acceleration in practice. Similar to bandit-based
methods, this is because each dimension is independently processed so that the model cannot benefit from any low-level optimization for linear algebra operations.

Our approach CANTOR significantly outperforms all of the baseline methods in accelerating the overall execution time to provide top-K recommendations in all datasets. More specifically, CANTOR has similar inference time for the prediction process to that of L2S (that also reduces the candidate item sets for less computation) but the preparation process of CANTOR is much faster. This is because similarities between user latent vectors are well leveraged to avoid unnecessary and redundant computation.

4.3 Number of Sub-Sampled User Latent Vectors \( u \)

Since only a small subset of user latent vectors will be sub-sampled for user clustering, we verify how the number of sub-sampled users affects the performance in both efficiency and accuracy. Figure 3 illustrates the P@1 scores and the ratios of speedup of CANTOR for different numbers of sampled users in six datasets. It is obvious that smaller number of sampled user latent vectors is accompanied with greater speedup and lower P@1 score. However, CANTOR can generally achieve 99% P@1 scores after sampling more than around 10^4 users in all datasets. In other words, the distribution of the whole dataset can be preserved by sampling a small portion of users. For example, the Wikipedia dataset needs to sample only 5% of all users for high accuracy of recommendations.

4.4 Trade-off in Proximity Graph Construction

Proximity graph plays an important role in CANTOR while the hyperparameter \( efs \) controls a trade-off between the efficiency and accuracy for generating the preferred item sets. Figure 4 depicts the P@1 scores and the speedup ratios of CANTOR for different \( efs \) in six datasets. Obviously, a too-small \( efs \) leads to unsatisfactory approximation and low accuracy scores. More precisely, the P@1 scores considerably drops when \( efs \) is below 10^2. On the other hand, CANTOR works well when \( efs \) is greater than 10^3 in all datasets. Hence, we suggest to tune \( efs \) in the range between 10^2 and 10^3 to reach a balance between efficiency and accuracy.

4.5 Number of Affinity Groups \( r \)

Figure 5 shows the performance of CANTOR with different numbers of user affinity groups \( r \) in six datasets. When there are more affinity groups, the sizes of preferred item sets shrink because of fewer representative vectors for each cluster. As a consequence,
CANTOR with larger group numbers considers fewer items in each affinity group, thereby achieving greater speedup. It is also noted that the great speedup comes with slight drop in accuracy. For example, there is only a 0.1% drop from \( r = 2 \) to \( r = 16 \). From these results, we suggest to set the number of user clusters \( r \) as a reasonable large number.

### 4.6 Effectiveness of Adaptive Representative Selection

The adaptive representative selection (ARS) method as shown in Algorithm 2 is important for CANTOR to accelerate the preparation process, so we also evaluate its effectiveness and robustness. Figure 6 illustrates the preparation time of CANTOR in six datasets with and without applying ARS. As a result, CANTOR using adaptive representative selection significantly outperforms the one without using ARS across all datasets when achieving similar accuracy. This further demonstrates the effectiveness and robustness of the adaptive representative selection method.

### 5 RELATED WORK

In this section, we discuss the related work in collaborative filtering for recommender systems and maximum inner product search.

#### 5.1 Collaborative Filtering for Recommender Systems

Collaborative filtering (CF) [9] is one of the most popular solutions for recommendation problems, including the task of top-\( K \) recommender systems in this paper. Moreover, the low-rank assumption in CF further leads to the prominence of latent factor models or matrix factorization (MF) [19]. For example, Kang et al. [17] exploited MF models to optimize numerical ratings for top-\( K \) recommenders while Rendle et al. [29] observed pairwise implicit feedback in a one-class preference matrix and enhanced the personalized ranking performance of MF models. However, MF models can be time-consuming in inferring recommendations. More specifically, although MF models can be trained efficiently with sparse preference matrices, the number of possible recommendations can be enormous when the numbers of users and items are massive. To tackle this problem, Duan et al. [12] proposed to separately compute dot-product results in each dimension so that some items can be discarded if their dot-product values are below a threshold for specific dimensions. However, separately processing different dimensions and discarding certain entries not only lead to inaccuracy, but also give up the opportunity to take advantage of low-level runtime optimization like BLAS [4] as shown in our experiments. Moreover, this approach does not reduce the number of possible
recommendations. On the other hand, although some of the previous studies [18, 28] achieve acceleration by group recommendation, users in a certain group would receive identical recommendations that can be unsatisfactory for individual users. To the best of our knowledge, this paper is the first work leveraging each user information to accelerate the inference process for top-$K$ recommender systems without accuracy loss.

5.2 Maximum Inner Product Search

Maximum inner product search (MIPS) can be treated as a closely related problem to MF based top-$K$ recommender systems. Shrivastava and Li [31] and Neyshabur and Srebro [27] proposed to reduce MIPS to nearest neighbor search (NNS) and then solve NNS by Locality Sensitive Hashing (LSH) [16]. PCA tree [32] partitions the space according to the directions of principal components and shows better performance in practice. Bachrach et al. [1] showed tree-based approaches can be used for solving MIPS but the performance is poor for high dimensional data. Malkov et al. [25], Malkov and Yashunin [26] recently developed an NNS algorithm based on small world graph. Zhang et al. [35] applied the MIPS-to-NNS reduction and showed that graph-based approach performs well on neural language model prediction. Some algorithms were proposed to directly tackle MIPS problem instead of transforming to NNS. For example, Yu et al. [34] proposed Greedy-MIPS and showed a significant improvement over LSH and tree-based approaches.

Another branch of research exploited sampling techniques with the (MAD) problem discussed in [3]. However, their goal is to find top-$K$ pairs among all $m \times n$ for the users in the affinity group. Moreover, we mathematically prove that user coresets can be efficiently constructed by set covers of sub-sampled user latent vectors with an asymptotically guaranteed bound. Experimental results demonstrate that CANTOR significantly outperforms existing MIPS and approximate MF algorithms for accelerating top-$K$ recommender systems. In particular, CANTOR achieves 355x and 29x speedup on the largest Wikipedia and Amazon datasets in two tasks while the accuracy scores still remain to be 99% for both P@1 and P@5. Moreover, the results of analysis also show the effectiveness and robustness of CANTOR.

6 CONCLUSIONS

In this paper, we propose a novel framework for accelerating large-scale top-$K$ recommender systems by exploiting user relationships and the redundancy of user vectors in the latent space. Our model, CANTOR, first clusters users into affinity groups, thereby determining as a user coreset of representative vectors for each group so that only a limited number of preferred items need to be examined for the users in the affinity group. Moreover, we mathematically prove that user coresets can be efficiently constructed by set covers of sub-sampled user latent vectors with an asymptotically guaranteed bound. Experimental results demonstrate that CANTOR significantly outperforms existing MIPS and approximate MF algorithms for accelerating top-$K$ recommender systems. In particular, CANTOR achieves 355x and 29x speedup on the largest Wikipedia and Amazon datasets in two tasks while the accuracy scores still remain to be 99% for both P@1 and P@5. Moreover, the results of analysis also show the effectiveness and robustness of CANTOR.

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for their helpful comments. The work was partially supported by NSF DGE-1829071, NSF IIS-1719097, Intel, and Facebook.

APPENDIX

A PROOF OF THEOREM 1

Proof. Without loss of generality, we assume that vectors in $A_{r}$, $Q$, and $s_{t}$ have unit norms. $\forall q \in Q, i \in A_{r}$, we have:

$$\|p_{i} q^{T} - N_{s_{t}}(p_{i}) q^{T}\| = \|p_{i} - N_{s_{t}}(p_{i})\| q^{T} \| q^{T}\|
\leq \sqrt{d} \|p_{i} - N_{s_{t}}(p_{i})\| \leq \sqrt{d} \|p_{i} - N_{s_{t}}(p_{i})\|_{2}^{2}
= \sqrt{d} (\|p_{i}\|_{2}^{2} + \|N_{s_{t}}(p_{i})\|^{2} - 2N_{s_{t}}(p_{i}) q^{T} q^{T}) \leq \sqrt{d} (2 - 2\epsilon) = \delta,$$

where we define $\delta = \sqrt{d} (2 - 2\epsilon)$. $(a)$ follows from the fact that $\|\cdot\|_{2} \leq \sqrt{d} \|\cdot\|_{2}$, where $d$ is the dimension of the vector. $(b)$ follows from the condition of theorem. □

B PROOF OF THEOREM 2

Proof. Since $s_{t}$ is a $\epsilon$ set cover of $p_{s}$, there exist a $\delta$ such that $s_{t}$ is a $\delta$-user coreset of $p_{s}$. Therefore, for any given query $q$ and vector $p_{t}$ sampled from $P_{A_{r}}$, we have:

$$\|N_{s_{t}}(p_{t}) q^{T} - p_{t} q^{T}\| = \|N_{s_{t}}(p_{t}) q^{T} - p_{t} q^{T} + p_{t} q^{T} - p_{t} q^{T}\|
\leq \|N_{s_{t}}(p_{t}) q^{T} - p_{t} q^{T}\| + \|p_{t} q^{T} - p_{t} q^{T}\| \leq \delta + \|p_{t} q^{T} - p_{t} q^{T}\|.$$

Since $p_{t}$ and $p_{t}$ follow the same distribution, $p_{t}$ and $p_{t}$ will have same expectation value and we have:

$$\mathbb{E}[\|N_{s_{t}}(p_{t}) q^{T} - p_{t} q^{T}\|] = \mathbb{E}[\delta + \|p_{t} q^{T} - p_{t} q^{T}\|] = \delta + \mathbb{E}[\|p_{t} q^{T} - p_{t} q^{T}\|] \quad (a) \leq \delta + \mathbb{E}[\|p_{t} q^{T}\| - \mathbb{E}[p_{t} q^{T}\| = \delta,$$

where $(a)$ follows the Jensen’s inequality. Therefore, by Hoeffding’s inequality, with probability at least $1 - \gamma$:

$$\frac{1}{k} \sum_{i=1}^{k} \|N_{s_{t}}(p_{i}) q^{T} - p_{i} q^{T}\| \leq \delta + \sqrt{\frac{2 \log (1/\gamma)}{k}}.$$

By the fact that for any set $S$, $\min(S) \leq \text{mean}(S)$, we will have:

$$\min_{i} \left(\|N_{s_{t}}(p_{i}) q^{T} - p_{i} q^{T}\| \leq \frac{1}{k} \sum_{i=1}^{k} \|N_{s_{t}}(p_{i}) q^{T} - p_{i} q^{T}\| \leq \delta + \sqrt{\frac{2 \log (1/\gamma)}{k}}. \right.$$ □