Speeding Up the Xbox Recommender System Using a Euclidean Transformation for Inner-Product Spaces

Yoram Bachrach
Microsoft Research

Yehuda Finkelstein
Microsoft R&D

Ran Gilad-Bachrach
Microsoft Research

Liran Katzir
Computer Science, Technion

Noam Koenigstein
Microsoft R&D

Nir Nice
Microsoft R&D

Ulrich Paquet
Microsoft Research

ABSTRACT

A prominent approach in collaborative filtering based recommender systems is using dimensionality reduction (matrix factorization) techniques to map users and items into low-dimensional vectors. In such systems, a higher inner product between a user vector and an item vector indicates that the item better suits the user's preference. Traditionally, retrieving the most suitable items is done by scoring and sorting all items. Real world online recommender systems must adhere to strict response-time constraints, so when the number of items is large, scoring all items is intractable.

We propose a novel order preserving transformation, mapping the maximum inner product search problem to Euclidean space nearest neighbor search problem. Utilizing this transformation, we study the efficiency of several (approximate) nearest neighbor data structures. Our final solution is based on a novel use of the PCA-Tree data structure in which results are augmented using paths one hamming distance away from the query (neighborhood boosting). The end result is a system which allows approximate matches (items with relatively high inner product, but not necessarily the highest one). We evaluate our techniques on two large-scale recommendation datasets, Xbox Movies and Yahoo Music, and show that this technique allows trading off a slight degradation in the recommendation quality for a significant improvement in the retrieval time.

Categories and Subject Descriptors

H.5 [**Information systems**]: Information retrieval—*retrieval models and ranking, retrieval tasks and goals*

Keywords

Recommender systems, matrix factorization, inner product search, fast retrieval

RecSys'14, October 6–10, 2014, Foster City, Silicon Valley, CA, USA.

ACM 978-1-4503-2668-1/14/10 ...$15.00.
http://dx.doi.org/10.1145/2645710.2645741.

1. INTRODUCTION

The massive growth in online services data gives rise to the need for better information filtering techniques. In the context of recommender systems the data consists of (1) the item catalog; (2) the users; and (3) the user feedback (ratings). The goal of a recommender system is to find for every user a limited set of items that have the highest chance to be consumed. Modern recommender systems have two major parts. In the first part, the learning phase, a model is learned (offline) based on user feedback[1]. In the second part, the retrieval phase, recommendations are issued per user (online). This paper studies the scalability of the retrieval phase (the second part) in massive recommender systems based on matrix factorization. Specifically, we introduce a new approach which offers a trade-off between running time and the quality of the results presented to a user.

Matrix Factorization (MF) is one of the most popular approaches for collaborative filtering. This method has repeatedly demonstrated better accuracy than other methods such as nearest neighbor models and restricted Boltzmann machines [2, 8]. In MF models, users and items are represented by latent feature vectors. A Bayesian MF model is also at the heart of the Xbox recommendation system [16] which serves games, movies, and music recommendations to millions of users daily. In this system, users and items are represented by (low-dimensional) vectors in $\mathbb{R}^{50}$. The quality of the match between a user u represented by the vector $\mathbf{x}_u$ and the item i represented by the vector $\mathbf{y}_i$ is given by the inner product $\mathbf{x}_u \cdot \mathbf{y}_i$ between these two vectors. A higher inner product implies a higher chance of the user consuming the item.

The Retrieval Problem: Ideally, given a user u represented by a vector $\mathbf{x}_u$, all the item vectors $(\mathbf{y}_1, \ldots, \mathbf{y}_n)$ are examined. For each such item vector $\mathbf{y}_i$, its match quality with the user $(\mathbf{x}_u \cdot \mathbf{y}_i)$ is computed, and the items sorted according to their match quality. The items with the highest match quality in the list are then selected to form the final list of recommendations. However, the catalog of items is often too large to allow an exhaustive computation of all the inner products within a limited allowed retrieval time.

The Xbox catalog consists of millions of items of various kinds. If a linear scan is used, millions of inner product computations are required for each single recommendation. The

[1]This phase cannot be done entirely offline when a context is used to issue the recommended items.

user vectors can take into account contextual information[2] that is only available during user engagement. Hence, the complete user vector is computed *online* (at runtime). As a result, the retrieval of the recommended items list can only be performed online, and cannot be pre-computed offline. This task constitutes the single most computational intensive task imposed on the online servers. Thereby, having a fast alternative for this process is highly desirable.

Our Contribution: This paper shows how to significantly speed up the recommendation retrieval process. The optimal item-user match retrieval is relaxed to an approximate search: retrieving items that have a high inner product with the user vector, but not necessarily the highest one. The approach combines several building blocks. First, we define a novel transformation from the inner product problem to a Euclidean nearest neighbor problem (Section 3). As a pre-processing step, this transformation is applied to the item vectors. During item retrieval, another transformation is applied to the user vector. The item with the smallest Euclidean distance in the transformed space is then retrieved. To expedite the nearest neighbor search, the PCA-Tree [21] data structure is used together with a novel neighborhood boosting scheme (Section 4).

To demonstrate the effectiveness of the proposed approach, it is applied to an Xbox recommendations dataset and the publicly available Yahoo Music dataset [8]. Experiments show a trade-off curve of a slight degradation in the recommendation quality for a significant improvement in the retrieval time (Section 5). In addition, the achievable time-accuracy trade-offs are compared with two baseline approaches, an implementation based on Locality Sensitive Hashing [1] and the current state of the art method for approximate recommendation in matrix-factorization based CF systems [13]. We show that for a given required recommendation quality (accuracy in picking the optimal items), our approach allows achieving a much higher speedup than these alternatives.

Notation: We use lower-case fonts for scalars, bold lower-case fonts for vectors, and bold upper-case fonts for matrices. For example, x is a scalar, $\mathbf{x}$ is a vector, and $\mathbf{X}$ is a matrix. Given a vector $\mathbf{x} \in \mathbb{R}^d$, let x_i be the measure in dimension i, with $(x_1, x_2, \ldots, x_d)^T \in \mathbb{R}^d$. The norm is denoted by $\|\cdot\|$; in Euclidean space $\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{d} x_i^2}$. We denote by $\mathbf{x} \cdot \mathbf{y}$ a dot product (inner product) between $\mathbf{x}$ and $\mathbf{y}$. Finally, we use $\left(a, \mathbf{x}^T\right)^T$ to denote a concatenation of a scalar a with a vector $\mathbf{x}$.

2. BACKGROUND AND RELATED WORK

In this section we will explain the problem of finding best recommendations in MF models and review possible approaches for efficient retrieval of recommendations.

2.1 Matrix Factorization Based Recommender Systems

In MF models, each user u is associated with a user-traits vector $\mathbf{x}_u \in \mathbb{R}^d$, and each item i with an item-traits vector $\mathbf{y}_i \in \mathbb{R}^d$. The predicted rating of a user u to an item i is denoted by $\hat{r}_{ui}$ and obtained using the rule:

$$\hat{r}_{ui} = \mu + b_u + b_i + \mathbf{x}_u \cdot \mathbf{y}_i \ , \qquad (1)$$

[2]The contextual information may include the time of day, recent search queries, etc.

where μ is the overall mean rating value and b_i and b_u represent the item and user biases respectively. The above model is a simple baseline model similar to [14]. It can be readily extended to form the core of a variety of more complex MF models, and adapted to different kinds of user feedback.

While μ and b_i are important components of the model, they do not effect the ranking of items for any given user, and the rule $\tilde{r}_{ui} = b_i + \mathbf{x}_u \cdot \mathbf{y}_i$ will produce the same set of recommendations as that of Equation 1. We can also concatenate the item bias b_i to the user vector and reduce our prediction rule to a simple dot product: $\tilde{r}_{ui} = \bar{\mathbf{x}}_u \cdot \bar{\mathbf{y}}_i$, where $\bar{\mathbf{x}}_u \triangleq (1,\ \mathbf{x}_u^T)^T$, and $\bar{\mathbf{y}}_i \triangleq (b_i,\ \mathbf{y}_i^T)^T$. Hence, computing recommendations in MF models amounts to a simple search in an inner product space: given a user vector $\bar{\mathbf{x}}_u$, we wish to find items with vectors $\bar{\mathbf{y}}_i$ that will maximize the inner product $\bar{\mathbf{x}}_u \cdot \bar{\mathbf{y}}_i$. For the sake of readability, from this point onward we will drop the bar and refer to $\bar{\mathbf{x}}_u$ and $\bar{\mathbf{y}}_u$ as $\mathbf{x}_u$ and $\mathbf{y}_i$. We therefore focus on the problem of finding maximal inner product matches as described above.

2.2 Retrieval of Recommendations in Inner-Product Spaces

The problem of efficient retreival of recommendations in MF models is relatively new, but it has been discussed in the past [10, 11, 13]. In real-world large scale systems such as the Xbox Recommender, this is a concrete problem, and we identified it as the main bottleneck that drains our online resources.

Previous studies can be categorized into two basic approaches. The first approach is to propose new recommendation algorithms in which the prediction rule is not based on inner-product matches. This was the approach taken by Khoshneshin et al. [10], who were first to raise the problem of efficient retrieval of recommendations in MF models. In [10] a new model is proposed in which users and items are embedded based on their Euclidean similarity rather than their inner-product. In a Euclidean space, the plethora of algorithms for nearest-neighbor search can be utilized for an efficient retrieval of recommendations. A similar approach was taken by [11] where an item-oriented model was designed to alleviate retrieval of recommendations by embedding items in a Euclidean space. While these methods show significant improvements in retrieval times, they deviate from the well familiar MF framework. These approaches which are based on new algorithms do not benefit the core of existing MF based recommender systems in which the retrieval of recommendations is still based on inner-products.

The second approach to this problem is based on designing new algorithms to mitigate maximal inner-product search. These algorithms can be used in any existing MF based system and require only to implement a new data structure on top of the recommender to assist at the retrieval phase. For example, in [13] a new IP-Tree data structure was proposed that enables a branch-and-bound search scheme in inner-product spaces. In order to reach higher speedup values, the IP-Tree was combined with spherical user clustering that allows to pre-compute and cache recommendations to similar users. However, this approach requires prior knowledge of all the user vectors which is not available in systems such as the Xbox recommender where ad-hoc contextual information is used to update the user vectors. This work was later continued in [18] for the general problem of maximal inner-product search, but these extensions showed effective-

ness in high-dimensional sparse datasets which is not the case for vectors generated by a MF process.

This paper builds upon a novel transformation that reduces the maximal inner-product problem to simple nearest neighbor search in a Euclidean space. On one hand the proposed approach can be employed by any classical MF model, and on the other hand it enables using any of the existing algorithms for Euclidean spaces. Next, we review several alternatives for solving the problem in a Euclidean Space.

2.2.1 Nearest Neighbor in Euclidean Spaces

Locality Sensitive Hashing (LSH) was recently popularized as an effective approximate retrieval algorithm. LSH was introduced by Broder et al. to find documents with high Jaccard similarity[4]. It was later extended to other metrics including the Euclidean distance [9], cosine similarity [5], and earth mover distance [5].

A different approach is based on space partitioning trees: KD-trees [3] is a data structure that partitions $\mathbb{R}^d$ into hyper-rectangular (axis parallel) cells. In construction time, nodes are split along one coordinate. At query time, one can search of all points in a rectangular box and nearest neighbors efficiently. Several augmented splits are used to improve the query time. For example, (1) Principal component axes trees (PCA-Trees) transform the original coordinates to the principal components [21]; (2) Principal Axis Trees (PAC-Trees) [15] use a principal component axis at every node; (3) Random Projection Trees (RPT) use a random axis at each node [6]; and (4) Maximum Margin Trees (MMT) use a maximum margin axis at every node [20]. A theoretical and empirical comparison for some variants can be found [19].

Our approach makes use of PCA-trees and combines it with a novel neighborhood boosting scheme. In Section 5 we compare to alternatives such as LSH, KD-Trees, and PAC-Trees. We do not compare against MMT and RPT as we don't see their advantage over the other methods for the particular problem at hand.

3. REDUCIBLE SEARCH PROBLEMS

A key contribution of this work is focused on the concept of efficient reductions between search problems. In this section we formalize the concept of a search problem and show efficient reductions between known variants.

We define a search problem as:

DEFINITION 1. *A search problem $S(\mathcal{I}, \mathcal{Q}, s)$ consists of an instance set of n items $I = \{i_1, i_2, \ldots, i_n\} \in \mathcal{I}$, a query $\mathfrak{q} \in \mathcal{Q}$, and a search function*

$$s : \mathcal{I} \times \mathcal{Q} \to \{1, 2, \ldots, n\} \ .$$

Function s retrieves the index of an item in I for a given query $\mathfrak{q}$. The goal is to pre-process the items with $g : \mathcal{I} \to \mathcal{I}'$ such that each query is answered efficiently. The pre-processing g can involve a transformation from one domain to another, so that a transformed search problem can operate on a different domain. The following definition formalizes the reduction concept between search problems:

DEFINITION 2. *A search problem $S_1(\mathcal{I}, \mathcal{Q}, s_1)$ is reducible to a search problem $S_2(\mathcal{I}', \mathcal{Q}', s_2)$, denoted by $S_1 \leq S_2$, if there exist functions $g : \mathcal{I} \to \mathcal{I}'$ and $h : \mathcal{Q} \to \mathcal{Q}'$ such that*

$$j = s_1(I, \mathfrak{q}) \ \text{ if and only if } \ j = s_2(g(I), h(\mathfrak{q})) \ .$$

This reduction does not apply any constraints on the running time of g and h. Note that g runs only once as a pre-processing step, while h is applied at the query time. This yields a requirement that h has a $O(1)$ running time. We formalize this with the following notation:

DEFINITION 3. *We say that $S_1 \leq_{O(f(n))} S_2$ if $S_1 \leq S_2$ and the running time of g and h are $O(f(n))$ and $O(1)$ respectively.*

For a query vector in $\mathbb{R}^d$, we consider three search problems in this paper: **MIP**, the maximum inner product from n vectors in $\mathbb{R}^d$ ($\text{MIP}_{n,d}$); **NN**, the nearest neighbor from n vectors in $\mathbb{R}^d$ ($\text{NN}_{n,d}$); **MCS**, the maximum cosine similarity from n vectors in $\mathbb{R}^d$ ($\text{MCS}_{n,d}$). They are formally defined as follows:

Instance: A matrix of n vectors $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n]$ such that $\mathbf{y}_i \in \mathbb{R}^d$; therefore $\mathcal{I} = \mathbb{R}^{d \times n}$.

Query: A vector $\mathbf{x} \in \mathbb{R}^d$; hence $\mathcal{Q} = \mathbb{R}^d$.

Objective: Retrieve an index according to

$$\begin{aligned} s(\mathbf{Y}, \mathbf{x}) &= \operatorname*{argmax}_i \mathbf{x} \cdot \mathbf{y}_i && \text{MIP}_{n,d} \\ s(\mathbf{Y}, \mathbf{x}) &= \operatorname*{argmin}_i \|\mathbf{x} - \mathbf{y}_i\| && \text{NN}_{n,d} \\ s(\mathbf{Y}, \mathbf{x}) &= \operatorname*{argmax}_i \frac{\mathbf{x} \cdot \mathbf{y}_i}{\|\mathbf{x}\| \, \|\mathbf{y}_i\|} && \text{MCS}_{n,d} \ , \end{aligned}$$

where i indicates column i of $\mathbf{Y}$.

The following section shows how transformations between these three problems can be achieved with $\text{MCS}_{n,d} \leq_{O(n)} \text{MIP}_{n,d} \leq_{O(n)} \text{NN}_{n,d+1}$ and $\text{NN}_{n,d} \leq_{O(n)} \text{MCS}_{n,d+1} \leq_{O(n)} \text{MIP}_{n,d+1}$.

3.1 Order Preserving Transformations

The triangle inequality does not hold between vectors $\mathbf{x}$, $\mathbf{y}_i$, and $\mathbf{y}_j$ when an inner product compares them, as is the case in **MIP**. Many efficient search data structures rely on the triangle inequality, and if **MIP** can be transformed to **NN** with its Euclidian distance, these data structures would immediately become applicable. Our first theorem states that **MIP** can be reduced to **NN** by having an Euclidian metric in one more dimension than the original problem.

THEOREM 1. $\text{MIP}_{n,d} \leq_{O(n)} \text{NN}_{n,d+1}$

Proof: Let $\phi \triangleq \max_i \|\mathbf{y}_i\|$ and preprocess input with: $\tilde{\mathbf{y}}_i = g(\mathbf{y}_i) = \left(\sqrt{\phi^2 - \|\mathbf{y}_i\|^2}, \mathbf{y}_i^T\right)^T$. During query time: $\tilde{\mathbf{x}} = h(\mathbf{x}) = \left(0, \mathbf{x}^T\right)^T$. As

$$\begin{aligned} \|\tilde{\mathbf{x}}\|^2 &= \|\mathbf{x}\|^2 \\ \|\tilde{\mathbf{y}}_i\|^2 &= \phi^2 - \|\mathbf{y}_i\|^2 + \|\mathbf{y}_i\|^2 = \phi^2 \\ \tilde{\mathbf{x}} \cdot \tilde{\mathbf{y}}_i &= \sqrt{\phi^2 - \|\mathbf{x}_i\|^2} \cdot 0 + \mathbf{x} \cdot \mathbf{y}_i = \mathbf{x} \cdot \mathbf{y}_i \end{aligned}$$

we have

$$\|\tilde{\mathbf{x}} - \tilde{\mathbf{y}}_i\|^2 = \|\tilde{\mathbf{x}}\|^2 + \|\tilde{\mathbf{y}}\|^2 - 2\tilde{\mathbf{x}} \cdot \tilde{\mathbf{y}}_i = \|\mathbf{x}\|^2 + \phi^2 - 2\mathbf{x} \cdot \mathbf{y}_i \ .$$

Finally, as ϕ and $\mathbf{x}$ are independent of index i,

$$j = \operatorname*{argmin}_i \|\tilde{\mathbf{x}} - \tilde{\mathbf{y}}_i\|^2 = \operatorname*{argmax}_i \mathbf{x} \cdot \mathbf{y}_i \ .$$

■

Theorem 1 provides the main workhorse for our proposed approach (Section 4). In the remaining of this section, we present its properties as well the related transformations.

If it is known that the transformed $\tilde{\mathbf{Y}} = [\tilde{\mathbf{y}}_1, \tilde{\mathbf{y}}_2, \ldots, \tilde{\mathbf{y}}_n]$ is in a manifold, as given above, we might expect to recover $\mathbf{Y}$ by reducing back with $\text{NN}_{n,d} \leq_{O(n)} \text{MIP}_{n,d-1}$. However, in the general case the transformation is only possible by increasing the dimensionality by one again:

THEOREM 2. $\text{NN}_{n,d} \leq_{O(n)} \text{MIP}_{n,d+1}$

Proof: The preprocessing of the input: $\tilde{\mathbf{y}}_i = g(\mathbf{y}_i) = \left(\|\mathbf{y}_i\|^2, \mathbf{y}_i^T\right)^T$. During query time: $\tilde{\mathbf{x}} = h(\mathbf{x}) = \left(1, -2\mathbf{x}^T\right)^T$. We have $\tilde{\mathbf{x}} \cdot \tilde{\mathbf{y}}_i = \|\mathbf{y}_i\|^2 - 2\mathbf{x} \cdot \mathbf{y}_i$. Finally,

$$j = \operatorname*{argmax}_i \tilde{\mathbf{x}} \cdot \tilde{\mathbf{y}}_i = \operatorname*{argmin}_i \|\mathbf{x}\|^2 + \|\mathbf{y}_i\|^2 - 2\mathbf{x} \cdot \mathbf{y}_i$$
$$= \operatorname*{argmin}_i \|\mathbf{x} - \mathbf{y}_i\|^2 .$$

■

MIP search can also be embedded in a **MCS** search by increasing the dimensionality by one:

THEOREM 3. $\text{MIP}_{n,d} \leq_{O(n)} \text{MCS}_{n,d+1}$

Proof: Preprocessing and query transformation are identical to Theorem 1. The preprocessing of the input: $\phi \triangleq \max_i \|\mathbf{y}_i\|$ and let $\tilde{\mathbf{y}}_i = g(\mathbf{y}_i) = \left(\sqrt{\phi^2 - \|\mathbf{y}_i\|^2}, \mathbf{y}_i^T\right)^T$. During query time: $\tilde{\mathbf{x}} = h(\mathbf{x}) = \left(0, \mathbf{x}^T\right)^T$. Finally,

$$j = \operatorname*{argmax}_i \frac{\tilde{\mathbf{x}} \cdot \tilde{\mathbf{y}}_i}{\|\tilde{\mathbf{x}}\| \|\tilde{\mathbf{y}}_i\|} = \operatorname*{argmax}_i \frac{\mathbf{x} \cdot \mathbf{y}_i}{\|\mathbf{x}\| \phi} = \operatorname*{argmax}_i \mathbf{x} \cdot \mathbf{y}_i .$$

■

However, **MCS** is simply **MIP** searching over normalized vectors:

THEOREM 4. $\text{MCS}_{n,d} \leq_{O(n)} \text{MIP}_{n,d}$

Proof: The preprocessing of the input: $\tilde{\mathbf{y}}_i = g(\mathbf{y}) = \frac{\mathbf{y}_i}{\|\mathbf{y}_i\|}$. During query time: $\tilde{\mathbf{x}} = h(\mathbf{x}) = \mathbf{x}$. Finally,

$$j = \operatorname*{argmax}_i \tilde{\mathbf{x}} \cdot \tilde{\mathbf{y}}_i = \operatorname*{argmax}_i \frac{\mathbf{x} \cdot \mathbf{y}_i}{\|\mathbf{x}\| \|\mathbf{y}_i\|} .$$

■

Our final result states that a **NN** search can be transformed to a **MCS** search by increasing the dimensionality by one:

THEOREM 5. $\text{NN}_{n,d} \leq_{O(n)} \text{MCS}_{n,d+1}$

Proof: Same reduction as in Theorem 1. The preprocessing of the input: $\phi \triangleq \max_i \|\mathbf{y}_i\|$ and $\tilde{\mathbf{y}}_i = g(\mathbf{y}_i) = \left(\sqrt{\phi^2 - \|\mathbf{y}_i\|^2}, \mathbf{y}_i^T\right)^T$. During query time: $\tilde{\mathbf{x}} = h(\mathbf{x}) = \left(0, \mathbf{x}^T\right)^T$. Thus by Theorem 1,

$$j = \operatorname*{argmax}_i \frac{\tilde{\mathbf{x}} \cdot \tilde{\mathbf{y}}_i}{\|\tilde{\mathbf{x}}\| \|\tilde{\mathbf{y}}_i\|} = \operatorname*{argmax}_i \frac{\mathbf{x} \cdot \mathbf{y}_i}{\|\mathbf{x}\| \phi} = \operatorname*{argmax}_i \mathbf{x} \cdot \mathbf{y}_i$$
$$= \operatorname*{argmin}_i \|\tilde{\mathbf{x}} - \tilde{\mathbf{y}}_i\|^2 .$$

■

Next, we utilize Theorem 1 for speeding up retrieval of recommendations in Xbox and other MF based recommender systems.

Algorithm 1 TransformAndIndex($\mathbf{Y}$, d')

input: item vectors $\mathbf{Y}$, depth $d' \leq d+1$
output: tree t
compute ϕ, $\boldsymbol{\mu}$, $\mathbf{W}$
$\mathcal{S} = \emptyset$
for $i = 1 : n$ **do**
  $\tilde{\mathbf{y}}_i = g(\mathbf{y}_i)$; $\mathcal{S} = \mathcal{S} \cup \tilde{\mathbf{y}}_i$
end for
return $T \leftarrow$ PCA-Tree($\mathcal{S}$, d')

4. AN OVERVIEW OF OUR APPROACH

Our solution is based on two components, a reduction to a Euclidian search problem, and a PCA-Tree to address it. The reduction is very similar to that defined in Theorem 1, but composed with an additional shift and rotation, so that the **MIP** search problem is reduced to **NN** search, with all vectors aligned to their principal components.

4.1 Reduction

We begin with defining the first reduction function following Theorem 1. Let $\phi \triangleq \max_i \|\mathbf{y}_i\|$, and

$$\mathbf{y}_i^\star = g_1(\mathbf{y}_i) = \left(\sqrt{\phi^2 - \|\mathbf{y}_i\|^2}, \mathbf{y}_i^T\right)^T$$
$$\mathbf{x}^\star = h_1(\mathbf{x}) = \left(0, \mathbf{x}^T\right)^T , \quad (2)$$

which, when applied to $\mathbf{Y}$, gives elements $\mathbf{y}_i^\star \in \mathbb{R}^{d+1}$. This reduces **MIP** to **NN**. As **NN** is invariant to shifts and rotations in the input space, we can *compose* the transformations with PCA rotation and still keep an equivalent search problem.

We mean-center and rotate the data: Let $\boldsymbol{\mu} = \frac{1}{n}\sum_i \mathbf{y}_i^\star$ be the mean after the first reduction, and $\mathbf{M} \in \mathbb{R}^{d+1 \times n}$ a matrix with $\boldsymbol{\mu}$ replicated along its columns. The SVD of the centered data matrix is

$$(\mathbf{Y}^\star - \mathbf{M}) = \mathbf{W}\boldsymbol{\Sigma}\mathbf{U}^T ,$$

where data items appear in the columns of $\mathbf{Y}^\star$. Matrix $\mathbf{W}$ is a $(d+1)$ by $(d+1)$ matrix. Each of the columns of $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_{d+1}]$ defines an orthogonal unit-length eigenvector, so that each $\mathbf{w}_j$ defines a hyperplane onto which each $\mathbf{y}_i^\star - \boldsymbol{\mu}$ is projected. Matrix $\mathbf{W}$ is a *r*otation matrix that aligns the vectors to their principal components. [3] We define the centered rotation as our second transformation,

$$\tilde{\mathbf{y}}_i = g_2(\mathbf{y}_i^\star) = \mathbf{W}^T(\mathbf{y}_i^\star - \boldsymbol{\mu})$$
$$\tilde{\mathbf{x}} = h_2(\mathbf{x}^\star) = \mathbf{W}^T(\mathbf{x}^\star - \boldsymbol{\mu}) . \quad (3)$$

The composition

$$g(\mathbf{y}_i) = g_2(g_1(\mathbf{y}_i)), \quad h(\mathbf{x}) = h_2(h_1(\mathbf{x})) \quad (4)$$

still defines a reduction from **MIP** to **NN**. Using $\tilde{\mathbf{y}}_i = g(\mathbf{y}_i)$, gives us a transformed set of input vectors $\tilde{\mathbf{Y}}$, over which an Euclidian search can be performed. Moreover, after this transformation, the points are rotated so that their components are in decreasing order of variance. Next, we index the transformed item vectors in $\tilde{\mathbf{Y}}$ using a PCA-Tree data structure. We summarize the above logic in Algorithm 1.

[3]Notice that $\boldsymbol{\Sigma}$ is not included, as the Euclidian metric is invariant under rotations of the space, but not shears.

Algorithm 2 PCA-Tree($\mathcal{S}$, δ)

input: item vectors set $\mathcal{S}$, depth δ
output: tree t
if $\delta = 0$ **then**
  return new leaf with $\mathcal{S}$
end if
$j = d + 1 - \delta$ *// principal component at depth δ*
$m = \text{median}(\{\tilde{y}_{ij} \text{ for all } \tilde{\mathbf{y}}_i \in \mathcal{S}\})$
$\mathcal{S}_{\leq} = \{\tilde{\mathbf{y}}_i \in \mathcal{S} \text{ where } \tilde{y}_{ij} \leq m\}$
$\mathcal{S}_{>} = \{\tilde{\mathbf{y}}_i \in \mathcal{S} \text{ where } \tilde{y}_{ij} > m\}$
t.leftChild = PCA-Tree($\mathcal{S}_{\leq}$, $\delta - 1$)
t.rightChild = PCA-Tree($\mathcal{S}_{>}$, $\delta - 1$)
return t

4.2 Fast Retrieve with PCA-Trees

Building the PCA-Tree follows from a the KD-Tree construction algorithm on $\tilde{\mathbf{Y}}$. Since the axes are aligned with the $d+1$ principal components of $\mathbf{Y}^{\star}$, we can make use of a KD-tree constriction process to get a PCA-Tree data structure. The top $d' \leq d+1$ principal components are used, and each item vector is assigned to its representative leaf. Algorithm 2 defines this tree construction procedure.

At the retrieval time, the transformed user vector $\tilde{\mathbf{x}} = h(\mathbf{x})$ is used to traverse the tree to the appropriate leaf. The leaf contains the item vectors in the neighborhood of $\tilde{\mathbf{x}}$, hence vectors that are on the same side of all the splitting hyperplanes (the top principal components). The items in this leaf form an initial candidates set from which the top items or nearest neighbors are selected using a direct ranking by distance.

The number of items in each leaf decays exponentially in the depth d' of the tree. By increasing the depth we are left with less candidates hence trading better speedup values with lower accuracy. The process allows achieving different trade-offs between the quality of the recommendations and an allotted running time: with a larger d', a smaller proportion of candidates are examined, resulting in a larger speedup, but also a reduced accuracy. Our empirical analysis (Section 5) examines the trade-offs we can achieve using our PCA-trees, and contrasts this with trade-offs achievable using other methods.

4.2.1 Boosting Candidates With Hamming Distance Neighborhoods

While the initial candidates set includes many nearby items, it is possible that some of the optimal top K vectors are indexed in other leafs and most likely the adjacent leafs. In our approach we propose boosting the candidates set with the item vectors in leafs that are on the "wrong" side in at most one of the median-shifted PCA hyperplane compared to $\tilde{\mathbf{x}}$. These vectors are likely to have a small Euclidean distance from the user vector.

Our PCA-Tree is a complete binary tree of height d', where each leaf corresponds to a binary vector of length d'. We supplement the initial candidates set from the leaf of the user vector, with all the candidates of leafs with a Hamming distance of '1', and hence examine candidates from d' of the $2^{d'}$ leafs. In Section 5.1.1 we show that this approach is instrumental in achieving the best balance between speedup and accuracy.

5. EMPIRICAL ANALYSIS OF SPEEDUP-ACCURACY TRADEOFFS

We use two large scale datasets to evaluate the speedup achieved by several methods:

1. **Xbox Movies** [12] – This dataset is a Microsoft propriety dataset consisting of 100 million binary $\{0, 1\}$ ratings of more than 15K movies by 5.8 million users. We applied the method used in [12] to generate the vectors representing items and users.

2. **Yahoo! Music** [8] – This is a publicly available ratings dataset consisting of 252,800,275 ratings of 624,961 items by 1,000,990 users. The ratings are on a scale of 0-100. The users and items vectors were generated by the algorithm in [7].

From both datasets we created a set of item vectors and user vectors of dimensionality $d = 50$. The following evaluations are based on these vectors.

Speedup Measurements and Baselines.

We quantify the improvement of an algorithm A over another (naive) algorithm A_0 by the following term:

$$\text{Speedup}_{A_0}(A) = \frac{\text{Time taken by Algorithm } A_0}{\text{Time taken by Algorithm } A}. \quad (5)$$

In all of our evaluations we measure the speedup with respect to the same algorithm: a naive search algorithm that iterates over all items to find the best recommendations for every user (i.e. computes the inner product between the user vector and each of the item vectors, keeping track of the item with the highest inner product found so far). Thus denoting by T_{naive} the time taken by the naive algorithm we have: $T_{naive} = \Theta(\#\text{users} \times \#\text{items} \times d)$.

The state of the art method for finding approximately optimal recommendations uses a combination of IP-Trees and user cones [13]. In the following evaluation we dubbed this method *IP-Tree*. The IP-Tree approach assumes all the user vectors (queries) are computed **in advance** and can be clustered into a structure of user cones. In many real-world systems like the Xbox recommender the user vectors are computed or updated online, so this approach cannot be used. In contrast, our method does not require having all the user vectors in advance, and is thus applicable in these settings.

The IP-Tree method relies on an adaptation of the branch-and-bound search in metric-trees [17] to handle nearest neighbor search in inner-product spaces. However, the construction of the underlaying metric-tree data structure, which is a space partitioning tree, is not adapted to inner-product spaces (it partitions vectors according to Euclidean proximity). By using the Euclidean transformation of Theorem 1, we can utilize the data structures and algorithms designed for Euclidean spaces in their original form, without adaptations that may curb their effectiveness. Next, we show that our approach achieves a **superior computation speedup**, despite having no access to any prior knowledge about the user vectors or their distribution. [4]

[4]We focus on online processing time, i.e. the time to choose an item to recommend for a target user. We ignore the computation time required by offline preprocessing steps.

Theorem 1 allows using various approximate nearest-neighbor algorithms for Euclidean spaces, whose performance depends on the specific dataset used. We propose using PCA-Trees as explained in Section 4.2, and show that they have an excellent performance for both the Xbox movies and Yahoo! music datasets, consisting of low dimensionality dense vectors obtained by matrix factorization. A different and arguably more popular approach for finding approximate-nearest-neighbors in Euclidean spaces is Locality-Sensitive Hashing (LSH) [1]. In the evaluations below we also include a comparison against LSH. We emphasize that using both our PCA-Trees approach and LSH techniques is only enabled by our Euclidean transformation (Theorem 1).

Our approximate retrieval algorithms introduce a trade-off between accuracy and speedup. We use two measures to quantify the quality of the top K recommendations. The first measure *Precision@K* denotes how similar the approximate recommendations are to the optimal top K recommendations (as retrieved by the naive approach):

$$Precision@K \triangleq \frac{|L_{rec} \cap L_{opt}|}{K} , \quad (6)$$

where L_{rec} and L_{opt} are the lists of the top K approximate and the top K optimal recommendations respectively. Our evaluation metrics only consider the items at the top of the approximate and optimal lists.[5]

A high value for *Precision* implies that the approximate recommendations are very similar to the optimal recommendations. In many practical applications (especially for large item catalogs), it is possible to have low *Precision* rates but still recommend very relevant items (with a high inner product between the user and item vectors). This motivates our second measure *RMSE@K* which examines the preference to the approximate items compared to the optimal items:

$$RMSE@K \triangleq \sqrt{\frac{1}{K}\sum_{k=1}^{K}\Big(L_{rec}(k) - L_{opt}(k)\Big)^2} , \quad (7)$$

where $L_{rec}(k)$ and $L_{opt}(k)$ are the scores (predicted ratings) of the k'th recommended item in the approximated list and the optimal list respectively. Namely, $L_{rec}(k)$ and $L_{opt}(k)$ are the values of inner products between the user vector and k'th recommended item vector and optimal item vector respectively. Note that the amount $\Big(L_{rec}(k) - L_{opt}(k)\Big)$ is always positive as the items in each list are ranked by their scores.

5.1 Results

Our initial evaluation considers three approximation algorithms: *IP-Tree*, *LSH*, and our approach (Section 4.2). Figure 1(a) depicts *Precision@10* for the Xbox Movies dataset (higher values indicate better performance). The *Precision* values are plotted against the average speedup values they enable. At very low speedup values the LSH algorithm shows the best trade-off between precision and speedup, but when higher speedup values are considered the LSH performance drops significantly and becomes worst. One possible reason for this is that our Euclidean transformation results in transformed vectors with one dimension being very large compared with the other dimensions, which is a difficult input distribution for LSH approaches.[6] In contrast, the tree-based approaches (IP-Tree and our approach) show a similar behavior of a slow and steady decrease in *Precision* values as the speedup increases. The speedup values of our approach offers a better precision-vs-speedup tradeoff than the IP-tree approach, though their precision is almost the same for high speedup values.

[5]Note that for this evaluation the *recall* is completely determined by the *precision*.

Method	Enabled by Theorem 1	Prior knowledge	Neighborhood boosting
IP-Tree	no	user vectors	not allowed
KD-Tree	yes	none	allowed
PCA-Tree	yes	none	allowed
PAC-Tree	yes	none	not allowed

Table 1: A summary of the different tree approaches. IP-Tree is the baseline from [13], which requires prior knowledge of the users vectors. All other approaches (as well as LSH) were not feasible before Theorem 1 was introduced in this paper.

Figure 1(b) depicts the *RMSE@10* (lower values indicate better performance) vs. speedup for the three approaches. The trend shows significantly superior results for our *PCA-Tree* approach, for all speedup values. Similarly to Figure 1(a), we see a sharp degradation of the *LSH* approach as the speedup increases, while the tree-based approaches show a trend of a slow increase in RMSE values as the speedup increases. We note that even for high speed-up values, which yield low precision rates in Figure 1(a), the RMSE values remain very low, indicating that very high quality of recommendations can be achieved at a fraction of the computational costs of the naive algorithm. In other words, the recommended items are still very relevant to the user, although the list of recommended items is quite different from the optimal list of items.

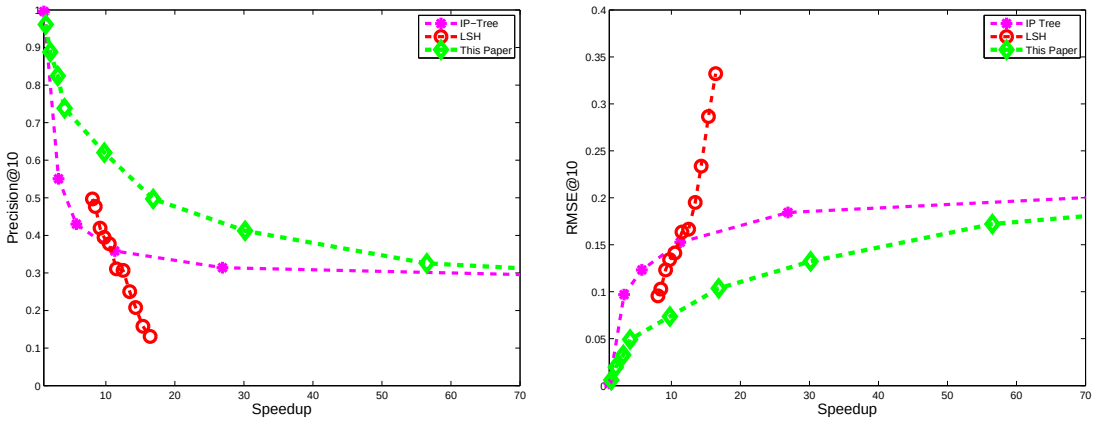
Figure 2 depicts *Precision@10* and *RMSE@10* for the Yahoo! Music dataset. The general trends of all three algorithms seem to agree with those of Figure 1: *LSH* starts better but deteriorates quickly, and the tree-based approaches have similar trends. The scale of the *RMSE* errors in Figure 1(b) is different (larger) because the predicted scores are in the range of 0-100, whereas in the Xbox Movies dataset the predictions are binary.

The empirical analysis on both the Xbox and Yahoo! datasets shows that it is possible to achieve excellent recommendations for very low computational costs by employing our Euclidean transformation and using an approximate Euclidean nearest neighbor method. The results indicate that tree-based approaches are superior to an LSH based approach (except when the required speedup is very small). Further, the results indicate that our method yields higher quality recommendations than the IP-trees approach [13]. Note that we also compared *Precision@K* and *RMSE@K* for other K values. While the figures are not included in this paper, the trends are all similar to those presented above.

5.1.1 Comparing Different Tree Approaches

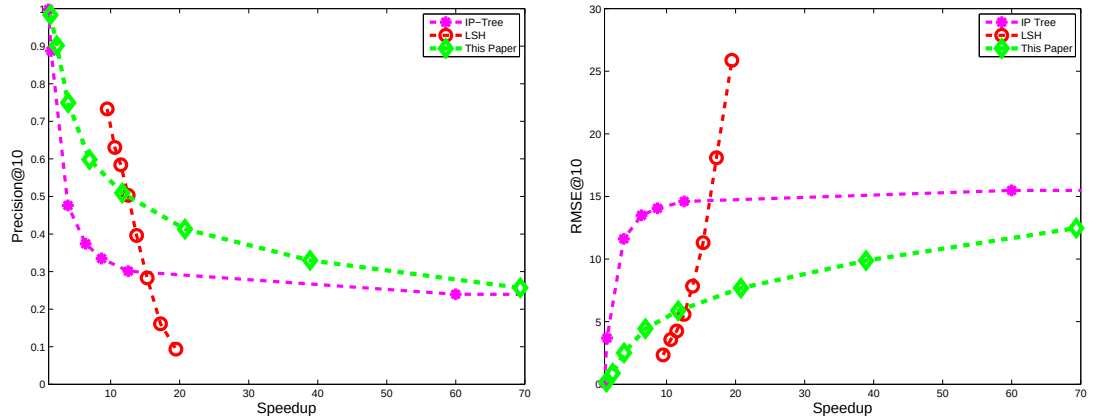
A key building block in our approach is aligning the item vectors with their principal components (Equation 3) and using PCA-Trees rather than KD-Trees. Another essential

[6]The larger dimension is the auxiliary dimension ($\sqrt{\phi^2 - \|\mathbf{y}_i\|^2}$) in Equation 2.



(a) Precision vs. Speedup (b) RMSE vs. Speedup

Figure 1: Performance against speedup values for the Xbox Movies dataset top 10 recommendations



(a) Precision vs. Speedup (b) RMSE vs. Speedup

Figure 2: Performance against speedup values for the Yahoo! Music dataset top 10 recommendations

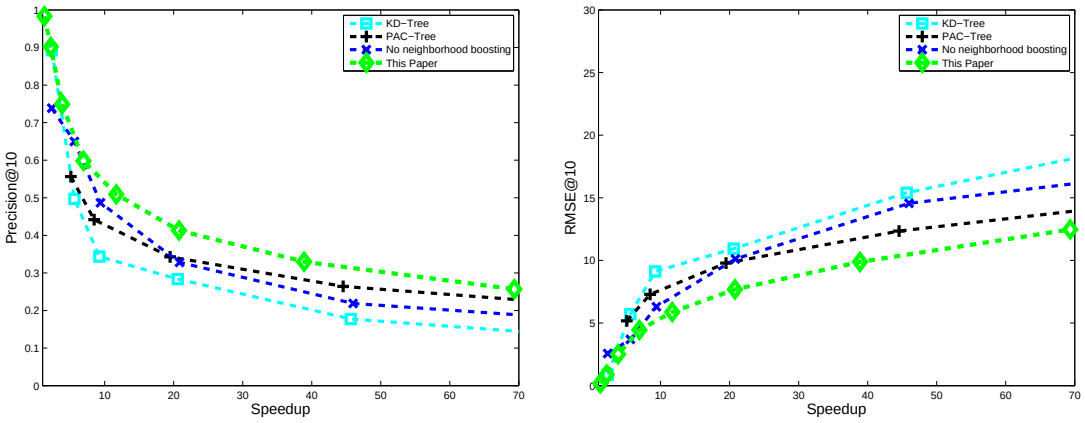
ingredient in our approach is the neighborhood boosting of Section 4.2.1. One may question the vitality of PCA-Trees or the neighborhood boosting to our overall solution. We therefore present a detailed comparison of the different tree based approaches. For the sake of completeness, we also included a comparison to *PAC-Trees* [15]. Table 1 summarizes the different data structures. Except the IP-Tree approach, all of these approaches were not feasible before Theorem 1 was introduced in this paper. Note that neighborhood boosting is possible only when the tree splits are all based on a single consistent axis system. It is therefore prohibited in IP-Tees and PAC-Trees where the splitting hyperplanes are ad-hoc on every node.

We compare the approach proposed in this paper with simple KD-Trees, PAC-Trees, and with PCA-Trees without neighborhood boosting (our approach without neighborhood boosting). Figure 3 depicts *Precision@10* and *RMSE@10* on the Yahoo! Music dataset. As the speedup levels increase, we notice an evident advantage in favor of PCA aligned trees over KD-Trees. When comparing PCA-Trees without neighborhood boosting to PAC-Trees we see a mixed picture: For low speedup values PCA-Trees perform better, but for higher speedup values we notice an eminent advantage in favor of PAC-Trees. To conclude, we note the overall advantage for the method proposed in this paper over any of the other tree based alternatives both in terms of *Precision* and *RMSE*.

6. CONCLUSIONS

We presented a novel transformation mapping a maximal inner product search to Euclidean nearest neighbor search, and showed how it can be used to speed-up the recommendation process in a matrix factorization based recommenders such as the Xbox recommender system.

We proposed a method for approximately solving the Euclidean nearest neighbor problem using PCA-Trees, and empirically evaluated it on the Xbox Movie recommendations and the Yahoo Music datasets. Our analysis shows that our approach allows achieving excellent quality recommendations at a fraction of the computational cost of a naive approach, and that it achieves superior quality-speedup tradeoffs compared with state-of-the-art methods.



(a) Precision vs. Speedup

(b) RMSE vs. Speedup

Figure 3: Comparing tree based methods for the Yahoo! Music dataset top 10 recommendations

7. REFERENCES

[1] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468, 2006.

[2] Robert M. Bell and Yehuda Koren. Lessons from the netflix prize challenge. *SIGKDD Explor. Newsl.*, 2007.

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[4] Andrei Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, pages 21–29, 1997.

[5] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, pages 380–388, 2002.

[6] Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pages 537–546, 2008.

[7] Gideon Dror, Noam Koenigstein, and Yehuda Koren. Yahoo! music recommendations: Modeling music ratings with temporal dynamics and item taxonomy. In *Proc. 5th ACM Conference on Recommender Systems*, 2011.

[8] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. The Yahoo! music dataset and KDD-Cup'11. *Journal Of Machine Learning Research*, 17:1–12, 2011.

[9] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998.

[10] Mohammad Khoshneshin and W. Nick Street. Collaborative filtering via euclidean embedding. In *Proceedings of the fourth ACM conference on Recommender systems*, 2010.

[11] Noam Koenigstein and Yehuda Koren. Towards scalable and accurate item-oriented recommendations. In *Proc. 7th ACM Conference on Recommender Systems*, 2013.

[12] Noam Koenigstein and Ulrich Paquet. Xbox movies recommendations: Variational Bayes matrix factorization with embedded feature selection. In *Proc. 7th ACM Conference on Recommender Systems*, 2013.

[13] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. Efficient retrieval of recommendations in a matrix factorization framework. In *CIKM*, 2012.

[14] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 2009.

[15] James McNames. A fast nearest-neighbor algorithm based on a principal axis search tree. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(9):964–976, September 2001.

[16] Ulrich Paquet and Noam Koenigstein. One-class collaborative filtering with random graphs. In *Proceedings of the 22nd international conference on World Wide Web*, WWW '13, pages 999–1008, 2013.

[17] Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[18] Parikshit Ram and Alexander Gray. Maximum inner-product search using cone trees. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2012.

[19] Parikshit Ram and Alexander G. Gray. Which space partitioning tree to use for search? In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *NIPS*, pages 656–664, 2013.

[20] Parikshit Ram, Dongryeol Lee, and Alexander G. Gray. Nearest-neighbor search on a time budget via max-margin trees. In *SDM*, pages 1011–1022. SIAM / Omnipress, 2012.

[21] Robert F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*,