Geometry Aware Mappings for High Dimensional Sparse Factors

Avradeep Bhowmik
University of Texas at Austin, Austin, TX

Nathan Liu
Google, Mountain View, CA

Erheng Zhong
Yahoo! Labs, Sunnyvale, CA

Badri Narayan Bhaskar
Yahoo! Labs, Sunnyvale, CA

Suju Rajan
Yahoo! Labs, Sunnyvale, CA

Abstract

While matrix factorisation models are ubiquitous in large scale recommendation and search, real time application of such models requires inner product computations over an intractably large set of item factors. In this manuscript we present a novel framework that uses the inverted index representation to exploit structural properties of sparse vectors to significantly reduce the run time computational cost of factorisation models. We develop techniques that use geometry aware permutation maps on a tessellated unit sphere to obtain high dimensional sparse embeddings for latent factors with sparsity patterns related to angular closeness of the original latent factors. We also design several efficient and deterministic realisations within this framework and demonstrate with experiments that our techniques lead to faster run time operation with minimal loss of accuracy.

1 INTRODUCTION

Latent factor models like matrix [17, 18] and tensor [23, 24] factorisation are a ubiquitous class of techniques with a wide range of applications, mainly in personalised search and recommendation systems, where each user i and each item j is assumed to be associated with latent factors $\mathbf{u}_i, \mathbf{v}_j \in \mathbb{R}^k$ respectively and the matrix of interactions (ratings, click/no-click, etc.) $\mathbf{R} = [r_{ij}]$ for the i^{th} user with the j^{th} item is modelled as the product of their respective latent factors as $r_{ij} \sim \mathbf{u}_i^\top \mathbf{v}_j$ or a monotonic function thereof.

While substantial amount of work [17, 18, 25] has been dedicated to learning the latent factors given interaction data in a scalable manner, an often overlooked problem is the computational efficiency of deploying the learned factors for real time recommendation.

The commonly used brute force retrieval of top-κ relevant items for any user i requires score computation of the corresponding latent factor $\mathbf{u}_i$ with every single item factor $\mathbf{v}_j \ \forall \ j = \{1, 2, \cdots N\}$, which is often an intractably large set. Pre-computing scores during the learning step is often impractical, for instance, in online news recommendation, where user interests change very rapidly and new items keep cropping up all the time. Moreover, while changing latent factors can be learned dynamically, arbitrary changes in latent factors would require updates to the entire set of pre-computed scores.

A greatly preferable alternative would be to design a technique that automatically discards irrelevant items per user, and thereby significantly reduces the search space for top-κ recommendations. This manuscript does exactly this, by exploiting structural properties of sparse vectors using the inverted index representation.

1.1 Sparse Factors and the Inverted Index Representation

Suppose the factors for users and items were very sparse, inner product between factors with non-overlapping sparsity patterns (non-zero's in different indices) would compute to 0. In such a case for every user, relevant items would be such that the corresponding user factor and items factor have more or less matching sparsity pattern.

The inverted index representation [11, 4], widely used in information retrieval tasks, is particularly appropriate to exploit this property. In our setup, this involves storing the list of items using a data structure where each index is associated with all the items whose corresponding latent factors are non-zero in that index.

During recommendation, for each user, we extract the set of indices $\mathcal{I}_\mathbf{u}$ in which the corresponding user fac-

Appearing in Proceedings of the 19^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 51.

tor $\mathbf{u}$ is non-zero, and retrieve using the inverted index notation, the set of items which are also non-zero in the corresponding indices in $\mathcal{I}_{\mathbf{u}}$. Inner product computation is then required only over this significantly smaller set, rather than the full item set.

1.2 Conflicting Sparsity Pattern

Clearly, the success of the inverted index representation relies on using factors with significant "conflict" in their sparsity patterns[1].

Low dimensional factors are almost always dense, and to avoid losing too much information, introducing sparsity would need to be accompanied with an increase in dimensionality. Unfortunately, most learning algorithms that promote sparsity (like LASSO) cannot necessarily ensure conflicting sparsity patterns. A more reliable alternative is post-processing factors (possibly dense, and learned using any appropriate algorithm) to obtain high dimensional sparse embeddings such that for original factors that are "close" to each other (high inner product), the corresponding sparse maps would have significant overlap in sparsity patterns, and vice versa.

To our knowledge, we are the first to explicitly tackle this exact problem setup, to use the inverted index representation to discard irrelevant factors with conflicting sparsity pattern. Our main contributions are as follows:

1. We introduce a novel framework consisting of a meta-algorithm that uses geometry aware permutation maps on a tessellated unit sphere to obtain sparse embeddings for latent factors with sparsity patterns related to angular closeness of the original factors.

2. We provide several deterministic realisations for the meta-algorithm that are efficient with respect to time and space complexity and satisfy desirable properties

3. We demonstrate the efficacy of our methods with extensive experimental evaluation

1.3 Notation

$\mathbb{S}^k$ refers to the surface of the k-dimensional Euclidean unit sphere. We use (usually subscripted) blackboard bold font $\mathbb{P}$ to denote permutations, always in p-dimensional space unless indicated otherwise, where $p > k$. Finally, we use $\boldsymbol{\phi} : \mathbb{S}^k \mapsto \mathbb{R}^p$ to refer to our sparse mapping function that maps factors $\mathbf{z}$ on the k-dimensional hypersphere to a sparse p-dimensional vector $\boldsymbol{\phi}(\mathbf{z})$. Zero padded vectors are denoted by diacritics as $\ddot{\mathbf{z}}$.

[1]Two sparse vectors have a conflicting sparsity pattern if the set of indices of non-zero elements for the two vectors are disjoint or have a very small intersection. For example $[9, 0, 8, 0, 0]$ and $[0, 6, 0, 7, 3]$ have non-zero elements in non-overlapping sets of indices $\{0, 2\}$ and $\{1, 3, 4\}$ respectively

2 PROBLEM SETUP

Consider the following setup. We are given a set of N factors $[\mathbf{z}_1; \cdots; \mathbf{z}_N] = \mathbf{Z} \in \mathcal{Z} \subseteq \mathbb{S}^k$, where $\mathbb{S}^k = \{\mathbf{x} \in \mathbb{R}^k : \|\mathbf{x}\|_2 = 1\}$ is the unit sphere in k-dimensional Euclidean space $\mathbb{R}^k$. For example, in a recommendation setting the set of factors $\mathbf{Z}$ could be the concatenated set $\mathbf{Z} = [\mathbf{U}; \mathbf{V}]$ of user features $\mathbf{U}$ and object features $\mathbf{V}$.

The "compatibility" between two factors $\mathbf{z}_i, \mathbf{z}_j$ is measured as $r_{ij} = \mathbf{z}_i^\top \mathbf{z}_j$. In the context of recommendations, $\mathbf{z}_i = \mathbf{u}_i$ is the i^{th} user factor, and $\mathbf{z}_j = \mathbf{v}_j$ is the j^{th} item factor, and r_{ij} is the interaction (rating, click/no-click).

This notion of "compatibility" between factors on the unit sphere $\mathbb{S}^k$ is captured by the angular distance[2] metric $d(\cdot, \cdot)$ which is defined for any two factors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^k$ as

$$d(\mathbf{x}, \mathbf{y}) = 1 - \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2}$$

For factors $\mathbf{z}_i, \mathbf{z}_j \in \mathbf{Z}$, $\|\mathbf{z}\|_2 = 1$, therefore $d(\mathbf{z}_i, \mathbf{z}_j) = 1 - \mathbf{z}_i^\top \mathbf{z}_j = 1 - r_{ij}$. Clearly, factors which are more compatible have a low angular distance in their Euclidean vector representations and vice versa.

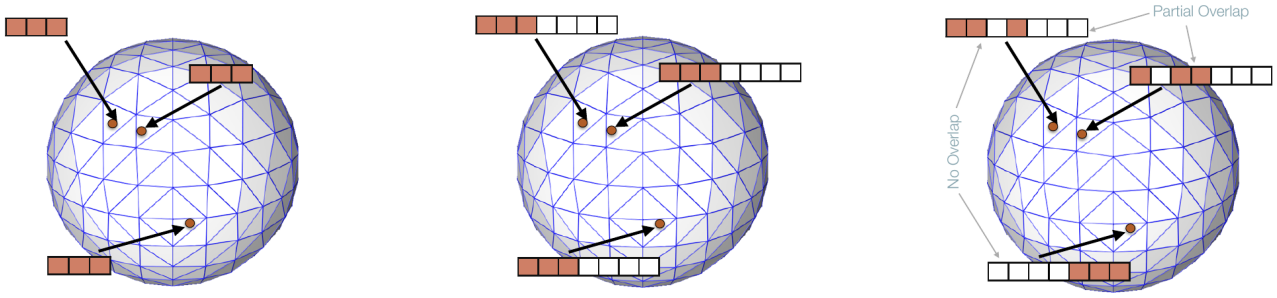
The objective is to find a map $\boldsymbol{\phi} : \mathcal{Z} \mapsto \mathbb{R}^p$ that maps factors in $\mathcal{Z}$ to sparse vectors in a p-dimensional space $\mathbb{R}^p$, where $p > k$.

As described earlier, the inverted index representation is useful in extracting vectors which have overlapping sparsity patterns. Hence, the mapping $\boldsymbol{\phi}$ should be such that if two factors $\mathbf{z}_i$ and $\mathbf{z}_j$ have a low angular distance between them, their corresponding mappings $\boldsymbol{\phi}(\mathbf{z}_i)$ and $\boldsymbol{\phi}(\mathbf{z}_j)$ should have similar sparsity patterns. Conversely, if $\mathbf{z}_i$ and $\mathbf{z}_j$ have a high angular distance between them, their corresponding mappings $\boldsymbol{\phi}(\mathbf{z}_i)$ and $\boldsymbol{\phi}(\mathbf{z}_j)$ should have conflicting sparsity patterns.

3 A GEOMETRY AWARE SCHEMA FOR SPARSE MAPPING

The requirements on $\boldsymbol{\phi}(\cdot)$ elaborated on in the preceding section can be captured effectively with the following intuitive observation. Suppose there exists a

[2]alternatively, one minus standard cosine similarity



(a) Tessellating unit sphere into regions, and marking each $\mathbf{z}$

(b) Padding with zeros, to obtain $\ddot{\mathbf{z}} \sim \{\mathbf{z}; \mathbf{0}\}$

(c) Applying region based permutation, to get $\boldsymbol{\phi}(\mathbf{z}) = \mathbb{P}(\ddot{\mathbf{z}})$

Figure 1: A pictorial representation of the sparse mapping technique- (a) Tessellating the unit sphere and associating each factor with its corresponding tile (b) Padding each factor with zeros to make it p-dimensional (c) Applying to each zero-padded vector the permutation specific to its tile on the tessellated unit sphere

mapping between sparsity patterns and regions on the surface of the unit sphere such that neighbouring regions of the unit sphere get similar sparsity patterns and vice versa. Then the requirements on the mapping function can be satisfied by setting $\boldsymbol{\phi}$ to map every factor with the sparsity pattern that depends on which region on the unit sphere the factor lies.

Our proposed framework uses this intuition to design a mapping function $\boldsymbol{\phi}(\cdot)$ that maps compatible (angularly close) factors to overlapping sparsity patterns and incompatible factors to conflicting sparsity patterns. The steps involved are described below (and summarised as the meta-algorithm in Algorithm (1)). We show concrete realisations for the meta-algorithm in section (4).

3.1 Defining a Schema

First we define a schema consisting of tessellating the unit sphere and a permutation map for each tile or region of the tessellated unit sphere.

Step I: Tessellating the unit sphere:

A natural way of defining regions corresponding to angular distance is via tiles on the surface of a tessellated unit sphere. An M-order tessellation for our purpose is specified completely by a set of M distinct tessellating vectors $\Gamma = \{\mathbf{a}_i : i = 1, 2, \cdots M, \|\mathbf{a}_i\| = 1\} \subset \mathbb{S}^k$. Each tile or region $\gamma_{\mathbf{a}}$ associated with a specific tessellating vector $\mathbf{a}$ under this schema is defined simply as the set of points which are closest to the said tessellating vector, that is,

$$\gamma_{\mathbf{a}} = \{\mathbf{x} \in \mathbb{S}^k : d(\gamma_{\mathbf{a}}, \mathbf{x}) \leq d(\gamma_{\mathbf{a}'}, \mathbf{x}) \ \forall \ \mathbf{a}' \in \Gamma, \mathbf{a} \neq \mathbf{a}'\}$$

Therefore, the boundary between regions under such a tessellating scheme consists of points on the unit sphere which are equidistant from two or more tessellating vectors. Note that this is similar to the concept of a Voronoi tessellation in a metric space.

Step II: Associating every region with a permutation:

The next step is to associate a permutation $\mathbb{P}_{\mathbf{a}}$ with each $\mathbf{a} \in \Gamma$, we denote the set of all such permutations by $\mathbb{P}_\Gamma = \{\mathbb{P}_{\mathbf{a}} : \mathbf{a} \in \Gamma\}$. The main requirement for this mapping is that nearby tessellating vectors should get mapped to similar permutations. An informal existence argument can be made for this using the fact that the set of permutations, when represented as the vertices of a Birkhoff polytope [5], can be embedded on the surface of a unit hypersphere [22].

3.2 Processing Factors based on the Schema

Given a schema $(\Gamma, \mathbb{P}_\Gamma)$, defining the map $\boldsymbol{\phi}(\cdot)$ for a set of factors $\mathbf{Z}$ consists of the following steps

Step I: Associating every factor to a region:

The associated region for a factor $\mathbf{z}$ is specified by the closest (in angular distance) tessellating vector $\mathbf{a}_{\mathbf{z}} \in \Gamma$ to the factor, as determined by the following optimisation problem

$$\mathbf{a}_{\mathbf{z}} = \arg\min_{\mathbf{a} \in \Gamma} d(\mathbf{a}, \mathbf{z}) \tag{1}$$

This is, in general, a difficult optimisation problem, with a search space over an intractably large discrete domain, but as we shall see in the succeeding sections, many tessellating schemata admit efficient solutions, exact or approximate, to this problem.

Step II: Zero-padding factors:

The next step is simply to zero-pad the k-dimensional vector with $p - k$ zeros to make it p-dimensional. Denote the zero-padded factor for $\mathbf{z}$ by $\ddot{\mathbf{z}}$.

Step III: Applying region specific permutation

Finally, we apply region specific permutations as defined by $\mathbb{P}_\Gamma$ to the zero-padded vector. Say $\mathbf{a_z}$ is the tessellating region associated with a factor $\mathbf{z}$, and $\mathbb{P}_{\mathbf{a_z}}$ is the corresponding p-dimensional permutation associated with $\mathbf{a_z}$, then the mapping $\boldsymbol{\phi}$ maps $\mathbf{z}$ to $\mathbb{R}^p$ by using the permutation operator $\mathbb{P}_{\mathbf{a_z}}$ on the zero-padded factor $\ddot{\mathbf{z}}$ as

$$\boldsymbol{\phi}(\mathbf{z}) = \mathbb{P}_{\mathbf{a_z}}(\ddot{\mathbf{z}}) \tag{2}$$

A pictorial[3] depiction of the technique has been shown in Figure (1). With proper selection of tessellation schema and permutation map, factors in nearby tiles have similar sparsity patterns (high overlap in non-zero entries) in their sparse maps and vice versa.

Algorithm 1 Sparse-mapping meta-algorithm

procedure SCHEMA(M)
  Define tessellating set of M vectors $\Gamma \subset \mathbb{S}^k$
  Define permutation map $\mathbb{P}_{\mathbf{a}}$ for each $\mathbf{a} \in \Gamma$
  return $\Gamma, \mathbb{P}_\Gamma$
end procedure

procedure PROCESSFACTORS($\Gamma, \mathbb{P}_\Gamma, \mathbf{Z}$)
  for each $\mathbf{z} \in \mathbf{Z}$ **do**
    associate region as $\mathbf{a_z} = \arg\min_{\mathbf{a}\in\Gamma} d(\mathbf{a}, \mathbf{z})$
    zero-pad $\tilde{\mathbf{z}} = [\mathbf{z}; 0]$
    apply permutation to get $\boldsymbol{\phi}(\mathbf{z}) = \mathbb{P}_{\mathbf{a_z}}(\tilde{\mathbf{z}})$
  end for
  return $\boldsymbol{\phi}(Z)$
end procedure

3.3 Desiderata for a good schema

Apart from finding effective tessellation schemata and permutation maps for each tessellation schema, there are multiple challenges that need to be taken into consideration when designing a particular instance for this meta algorithm. Note that for a given schema, processing the factors involves two potentially computationally intensive steps defined in equations (1) and (2). Hence, any schema for tessellation and permutation should be such that both (1) and (2) should be efficiently computable. Another concern would be controlling the storage complexity.

The first natural technique that is immediate as a tessellation schema is hypersphere point picking where random points are picked uniformly distributed on the surface of the unit hypersphere (see for instance [19, 8, 21, 13]). Owing to [20], we have a simple method of doing this by generating M independent and identically distributed points from the standard k-dimensional multivariate Gaussian distribution and normalising them. Since the standard Gaussian distribution is spherically symmetric, the resulting points are uniformly distributed on the surface of the unit hypersphere.

However, it is immediately apparent that for any randomised schema for Γ, solving the optimisation problem (1) for any $\mathbf{z} \in \mathbf{Z}$ would require an exhaustive search involving an explicit computation of distance scores with every $\mathbf{a} \in \Gamma$. Since M can be really large (often super exponential in k, as we shall see in section (4)), this is computationally infeasible. Moreover, because this requires explicit generation and storage of all M tessellating vectors, the space complexity required is also prohibitively high.

A similar argument holds for the design of the permutation map as well. The space of permutations over p coordinates is $O(2^{plogp})$ in p, and assigning permutations one by one to each of the M tessellating vectors would be infeasible.

A one-stop solution to both of these problems is to have a deterministic function-based schema for both the tessellation and permutation map. That is, given a factor $\mathbf{z} \in \mathbf{Z} \subset \mathbb{S}^k$, finding the tessellating vector closest to it should involve evaluating a function of $\mathbf{z}$ which is computationally efficient in terms of k, and for a given tessellating vector $\mathbf{a} \in \Gamma$, computing the corresponding permutation $\mathbb{P}_{\mathbf{a}}$ should involve evaluating a function of $\mathbf{a}$ which can be done efficiently in terms of p. Therefore, mapping for each factor can be done separately in a two-step process without additional explicit storage or computation of Γ or $\mathbb{P}_\Gamma$.

4 DETERMINISTIC SCHEMATA

In this section, we describe some concrete realisations for the general schema described earlier. In particular, we specify simple techniques for tessellating the unit sphere, as well as associating a permutation map with each of these tessellations, and generalise each of these techniques to define a broader class of realisations for our schemata. We will show that finding $\boldsymbol{\phi}(\mathbf{z})$ given $\mathbf{z}$ using each of these techniques only involves two efficient deterministic function computations, hence they avoid all of the pitfalls described in the previous section. Note that obtaining $\boldsymbol{\phi}(\mathbf{z})$ for each $\mathbf{z}$ can be done separately for each $\mathbf{z}$ in parallel.

4.1 Tessellating the Unit Sphere

The first step in the meta-algorithm is to tessellate the unit sphere. We start off by describing a simple scheme in section (4.1.1) and then generalise the idea to specify a larger class of tessellation schemes in sec-

[3] parts of the image adapted from the web

tion (4.1.2).

4.1.1 Directional tessellation

Consider the ternary base set $\mathcal{B} = \{-1, 0, 1\}$. Say $\mathcal{A} = \mathcal{B}^k - \{0\}^k$ is the set of all[4] non-zero k-length ternary vectors formed out of elements of $\{-1, 0, 1\}$. The tessellating vector set is formed from the normalised versions of these vectors, that is

$$\Gamma = \{\mathbf{a} = \frac{\tilde{\mathbf{a}}}{\|\tilde{\mathbf{a}}\|} : \tilde{\mathbf{a}} \in \mathcal{A}\} \tag{3}$$

Clearly in this case, $M = |\Gamma| = 3^k - 1$. However, finding the closest tessellating vector given any $\mathbf{z} \in \mathbf{Z}$ can be done efficiently.[5]

Lemma 1. *Given a factor $\mathbf{z} \in \mathbf{Z} \subset \mathbb{S}^k$, solving equation (1) for Γ as in (3) can be performed efficiently in $O(klogk)$ time using Algorithm (2), and requires no explicit storage of the tessellating set Γ.*

The time complexity of Algorithm (2) is $O(klogk)$ since the limiting operation is a sorting operation which can be done in $O(klogk)$ time. Proof of correctness of this algorithm is presented in the supplementary material.

4.1.2 Directional tessellation using D-ary Base set

This is exactly the same as the previous case, except the tessellation is done with a D-ary base set $\mathcal{B}_D = \{-1, -\frac{D-1}{D}, \cdots, -\frac{1}{D}, 0, \frac{1}{D}, \cdots, \frac{D-1}{D}, 1\}$ instead of a ternary base set- the corresponding vector set is $\mathcal{A}_D = \mathcal{B}_D^k - \{0\}^k$ and tessellating set is $\Gamma_D = \{\mathbf{a} = \frac{\tilde{\mathbf{a}}}{\|\tilde{\mathbf{a}}\|} : \tilde{\mathbf{a}} \in \mathcal{A}_D\}$. Clearly, the ternary base set $\{-1, 0, 1\}$ is the same as $\mathcal{B}_D$ with $D = 2$.

The algorithm (2) for ternary base sets no longer applies directly to the D-ary case. In fact, getting an exact solution is, in general, difficult for this schema. However, we can still get an ϵ-approximation to the closest tessellating vector (the corresponding algorithm is provided in the supplement).

Lemma 2. *For any vector $\mathbf{z} \in \mathbf{Z}$, say $\mathbf{a}_\mathbf{z}^*$ is the true solution to equation (1) with Γ obtained the D-ary base set. Then, a tessellating vector $\tilde{\mathbf{a}}_\mathbf{z}$ can be obtained such that $d(\mathbf{a}_\mathbf{z}^*, \tilde{\mathbf{a}}_\mathbf{z}) \leq \epsilon$, where $\epsilon \sim O(k/D^2)$, using an algorithm that takes $O(k)$ time, and requires no explicit storage of the tessellating set Γ_D.*

[4]For example, for $k = 2$, we have $\mathcal{A} = \{[-1, -1], [-1, 0], [-1, 1], [0, 1], [1, 1]\}$.

[5]Note that the naïve algorithm that thresholds every index of $\mathbf{z}$ on -0.5 or 0.5 to get an element of $\mathcal{B}$ does not give an exact solution since we are working with angular distance metrics

Algorithm 2 Region Specification on Γ defined on $\mathcal{B}$

procedure TessVector($\mathbf{z}$)
  Sort $\mathbf{z}$ in desc. order of abs. value to get $\mathbf{z}_\downarrow$
  Let π be the sorting order, that is,

$$\mathbf{z}_\downarrow^j = |\mathbf{z}^{\pi(j)}| \geq \mathbf{z}^{\pi(j+1)}| = \mathbf{z}_\downarrow^{j+1} \ \forall \ j$$

  Compute scaled cumsum $\mathbf{z}_s$ from $\mathbf{z}_\downarrow$ as
  for $\iota = 1, 2, \cdots k$ **do**
    Set $\mathbf{z}_s^\iota = \frac{\sum_{j=1}^{\iota} \mathbf{z}_\downarrow^j}{\sqrt{\iota}}$
  end for
  Let $\iota^* = \text{argmax}_\iota \mathbf{z}_s^\iota$ be the index of the maximum value of $\mathbf{z}_s$
  Define the index set (where $\mathbf{z}_\downarrow^{\pi^{-1}(k)} = |\mathbf{z}^k| \ \forall k$)

$$I_\mathbf{z} = \{\pi^{-1}(1), \pi^{-1}(2), \cdots \pi^{-1}(\iota^*)\}$$

  Compute the tessellating vector $\mathbf{a}_\mathbf{z}$ as

$$\mathbf{a}_\mathbf{z}^\iota = \begin{cases} \frac{sign(\mathbf{z}^\iota)}{\sqrt{|I_\mathbf{z}|}} & \text{if } \iota \in I_\mathbf{z} \\ 0 \quad \text{otherwise} \end{cases}$$

  return $\mathbf{a}_\mathbf{z}$
end procedure

Therefore, if $D \gg \sqrt{k}$, the true tessellating vector can be obtained within a very small tolerance. The algorithm to obtain this and the proof of this lemma is provided in the supplementary material. Clearly, this schema generates a finer tessellation of the unit sphere with increasing D.

4.2 Sparse Mapping in High Dimensions

The next step in the method is to assign a sparsity pattern to every tessellating vector. We start off by describing a simple permutation map in section (4.2.1) and then generalise the idea to define a larger class of maps in section (4.2.2). Note that each sparsity pattern can also be defined as a function (that depends on the corresponding tessellating vector $\mathbf{a}_\mathbf{z}$) that maps a coordinate of $\mathbf{z}$ to a specific coordinate of $\boldsymbol{\phi}(\mathbf{z})$.

4.2.1 One Hot Encoding

A simple encoding scheme is the following. Consider the ternary tessellation scheme as in Section (4.1.1). Note that for tessellating vectors obtained from a ternary scheme, there is a one-one correspondence between each $\tilde{\mathbf{a}} \in \mathcal{A}$ and each $\mathbf{a} \in \Gamma$ since

$$\mathbf{a}^j = \frac{sign(\tilde{\mathbf{a}}^j)}{(\text{num of non-zeros in } \tilde{\mathbf{a}})^{1/2}} \ \forall \ j$$

Therefore, a simple permutation scheme for this tessellation can be obtained for $p = 3k$ as follows. For

every $t = 1, 2, \cdots k$ and $i = 1, 2, \cdots p$ set

$$\boldsymbol{\phi}(\mathbf{z})^i = [\mathbb{P}_{\mathbf{a}_z}(\ddot{\mathbf{z}})]^i = \begin{cases} \mathbf{z}^t & \text{if } i = 3t \text{ and } \tilde{\mathbf{a}}_{\mathbf{z}}^t = 1 \\ \mathbf{z}^t & \text{if } i = 3t+1 \text{ and } \tilde{\mathbf{a}}_{\mathbf{z}}^t = 0 \\ \mathbf{z}^t & \text{if } i = 3t+2 \text{ and } \tilde{\mathbf{a}}_{\mathbf{z}}^t = -1 \\ 0 & \text{otherwise} \end{cases}$$

This basically pads each coordinate of $\mathbf{z}$ with two extra zeros and permutes it within each 3-index segment thus obtained depending on the value of the corresponding coordinate of $\tilde{\mathbf{a}}_{\mathbf{z}}$. A similar schema can be obtained for $p = Dk$ for the D-ary tessellation.

The permutation thus obtained is related to the geometry of the tessellation- for any two $\mathbf{a}_1, \mathbf{a}_2 \in \Gamma$, the Kendall-Tau distance[6] between their corresponding permutations $\mathbb{P}_{\mathbf{a}_1}, \mathbb{P}_{\mathbf{a}_2}$ is exactly equal to the ℓ_1 distance between the unnormalised vectors $\tilde{\mathbf{a}}_1, \tilde{\mathbf{a}}_2$. A bound for Spearman's footrule can also be obtained[9].

It is easy to see that this mapping also has the desirable property that for any $\mathbf{z} \sim \mathbf{a}$ and $\mathbf{z}' \sim \mathbf{a}'$, suppose for a particular index $j \in \{1, \cdots k\}$, we have that τ_j and τ'_j are the corresponding index map for $\boldsymbol{\phi}(\cdot)$. That is, $\boldsymbol{\phi}(\mathbf{z})^{\tau_j} = \mathbf{z}^j$ and $\boldsymbol{\phi}(\mathbf{z}')^{\tau'_j} = \mathbf{z}'^j$. Then, $\tau_j = \tau'_j$ if and only if $\mathbf{a}_j = \mathbf{a}'_j$. Moreover, the list of possible τ_j is unique for any j and depends only on j, and not on $\mathbf{a}$. This ensures that sparsity patterns overlap only for neighbouring tessellating regions, uniformly.

4.2.2 Parse Tree Based Encoding

A more general scheme for this can be obtained in the following manner. Consider the ternary tessellation scheme of Section (4.1.1). Start with a $\boldsymbol{\phi}(\mathbf{z})$ as a p-length vector of all zeros. For constructing a parse-tree of depth δ, initialise by mapping the first $\delta - 1$ coordinates of $\mathbf{z}$ to specific coordinates of $\boldsymbol{\phi}(\mathbf{z})$ using, say, the one-hot encoding scheme. At each subsequent step $j = \delta, \delta+1, \cdots k$, use a sliding window of size δ to read the unnormalised tessellating vector $\tilde{\mathbf{a}}$, such that δ coordinates are read at a time. At step j of the reading process, we read the segment $\tilde{\mathbf{a}}_\delta^j = [\tilde{\mathbf{a}}^{j-\delta}, \cdots, \tilde{\mathbf{a}}^j]$.

Since each coordinate of $\tilde{\mathbf{a}}$ can take three possible values $\{-1, 0, 1\}$, we can construct a parse tree of depth δ containing 3^δ leaf nodes which is traversed based on the δ-length segment thus read. Thus, each non-leaf node at depth $t \in \{0, 1, \cdots (\delta - 1)\}$ branches out into three child sub-trees corresponding to $\tilde{\mathbf{a}}^{j-\delta+t}$ being -1, 0 or 1.

Map the remaining coordinates of $\boldsymbol{\phi}(\mathbf{z})$ in the following way. At each step j of the reading process, maintain an index counter $\tau_j \in \{1, 2, \cdots p\}$ that defines the current index at time j. Associate each leaf node of the parse tree with a corresponding "action" $f(\cdot)$ to perform on the counter to move it to the current index. That is, given the previous index τ_{j-1} and the segment $\tilde{\mathbf{a}}_\delta^j = [\tilde{\mathbf{a}}^{j-\delta}, \cdots, \tilde{\mathbf{a}}^j]$ read at time step j, compute the next position for the counter as $\tau_j = f(\tau_{j-1}; \tilde{\mathbf{a}}_\delta^j)$. Finally, map the j^{th} coordinate of $\mathbf{z}$ to the current index of $\boldsymbol{\phi}(\mathbf{z})$ as $\boldsymbol{\phi}(\mathbf{z})^{\tau_j} = \mathbf{z}^j$. Repeat the same process for each step $j = \delta, \delta+1, \cdots k$.

For example, the one-hot encoding is a special case of this with $\delta = 1$. Suppose we have already mapped $\mathbf{z}^1$ to $\mathbf{z}^{j-1}$. At step j, if the leaf node corresponding to $\mathbf{a}_{\mathbf{z}}^j$ is 1, the corresponding action is to set the counter τ_j at time j to the coordinate $\tau_j = 3j$. Similarly, if $\mathbf{a}_{\mathbf{z}}^j$ is a 0, we set the counter to the coordinate $\tau_j = 3j+1$ and if it is -1, we set $\tau_j = 3j+2$. The corresponding coordinate of $\boldsymbol{\phi}(\mathbf{z})$ gets mapped as $\boldsymbol{\phi}(\mathbf{z})^{\tau_j} = \mathbf{z}^j$.

Other more complicated actions $f(\cdot)$ are possible. In particular, we describe some examples in the supplement which have the property that for any $\mathbf{a}, \mathbf{a}'$ at any step $\tau_j = \tau'_j$ if and only if $[\mathbf{a}^{j-t}, \cdots \mathbf{a}^j] = [\mathbf{a}'^{j-t}, \cdots \mathbf{a}'^j]$ for some $t > \delta$. Moreover, list of possible τ_j for any j depends uniquely only on j, and not on $\mathbf{a}$ itself. The parse-tree procedure can be extended to the case of a D-ary tessellation scheme in a simple manner by considering parse trees where each non-leaf node has D child nodes.

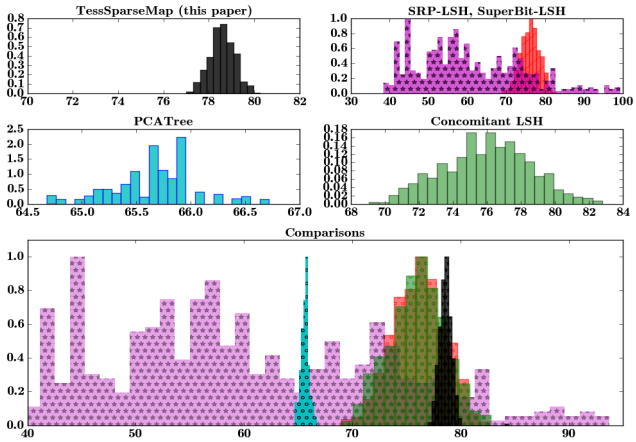
The mapping procedure is deterministic and time efficient- suppose each action $f(\cdot)$ in an $O(1)$ operation (most actions like shifts, etc. are of this form), the effective time complexity is $O(k\delta)$. The final space complexity of storing $\boldsymbol{\phi}(\mathbf{z})$ for each $\mathbf{z}$ is no more than $O(klogp)$ using the inverted index representation.

5 DISCUSSION

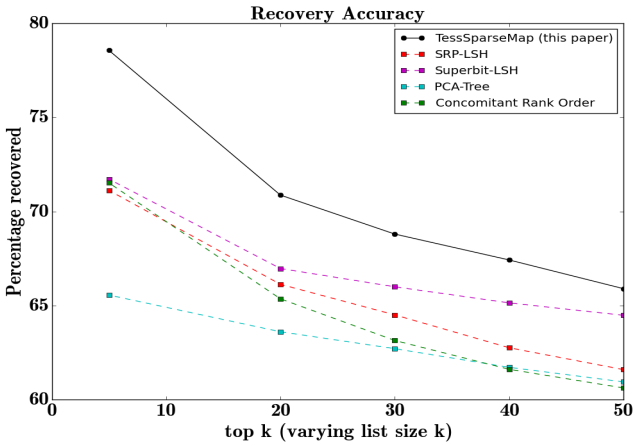
We would like to note that while the preceding discussion assumed normalised factors, we do not actually need the factors (or, indeed, even the tessellating vectors) themselves to have unit norm, since we are anyway looking at the angular distance metric. In particular, Algorithm (2) for finding the closest (in terms of angular distance) tessellating vector $\mathbf{a}$ for a given factor $\mathbf{z}$ is scale invariant in terms of $\mathbf{z}$ as well as the set of $\mathbf{a}$.

Next, note that our schema is generic and does not depend on the specific learning algorithm used for the latent factors. Moreover, because of the geometric nature of our framework, it can work for all kinds of factors irrespective of spherical symmetry properties of the factor distribution. For factors which are known to have clustered form, a simple extension of our algorithm would involve a non-uniform tessellation scheme with finer granularity near the cluster centres.

[6] minimum number of pairwise inversions required to convert one permutation to another, see [9]



(a) Histograms of percentage of discarded items over users



(b) Recovery Accuracy for Synthetic Data

Figure 2: Synthetic Data: our method discards more items on average with lower variance across users while having higher recovery accuracy (histogram y-axes scaled for uniformity)

A discussion on non-uniform tessellation is provided in the supplementary material.

5.1 Related Work

The closest line of work to our method is locality sensitive hashing [14, 12] which looks at approximate nearest neighbour extraction. However, while hashing is mostly concerned with dimensionality reduction (faster score computation by decreasing the effective k), our problem has more to do with direct reduction of effective search space (faster retrieval by reducing effective N); in fact, the two methods are independent enough that they can augment each other. Hash functions have been developed for different distance metrics like Hamming distance [12], Euclidean distance [3], Jaccard Similarity [7], etc. The most popular hash function for angular similarity is the sign-random-projection hash (SRP-LSH) [6] which generates random hyperplanes and assigns to a factor the sign of the projection of that factor on each hyperplane. A more recent alternative to this called Superbit-LSH [15] orthogonalises the random vectors before projection. Another variation uses l concomitant rank order statistics [10] instead of signed projection to compute an l-ary hash code. Yet another line of work computes hash functions by constructing spatial partitioning trees, specifically the PCA-tree [27] which splits each factor at the median along principal eigenvectors.

For standard hashing methods, the usual way is to extract relevant items for each user by computing the Hamming distance with the hash functions of corresponding item factors and returning the closest items. In our setup that is not feasible since that would defeat the entire purpose of not having to compare against every item. These methods would apply to our setup by computing exact hash matches using tree-based data structures. However, each instance of LSH divides factors into regions with rigid boundaries, which tends to throw away too many items[7], especially for factors at the edges. In contrast, since a geometry aware schema is tuned to each factor separately, our method by design also captures similarity with overlapping regions and soft boundaries.

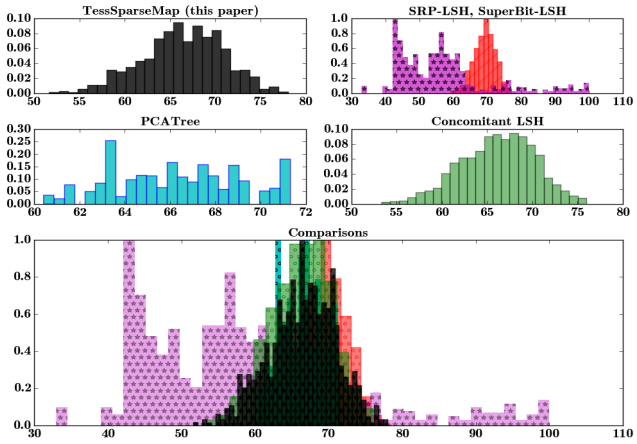
Uniformly tessellating the unit sphere deterministically is a hard problem. Various heuristics [16, 2, 26] exist that use, for example, arguments from physics to find minimum energy configurations for charges on a sphere. Embedding permutations on the unit sphere is even more difficult, see for instance [22]. However, unlike our methods, all of these techniques require computationally expensive exhaustive search approaches for finding the correct tessellation for a given factor, or finding the appropriate permutation map.

6 EXPERIMENTS

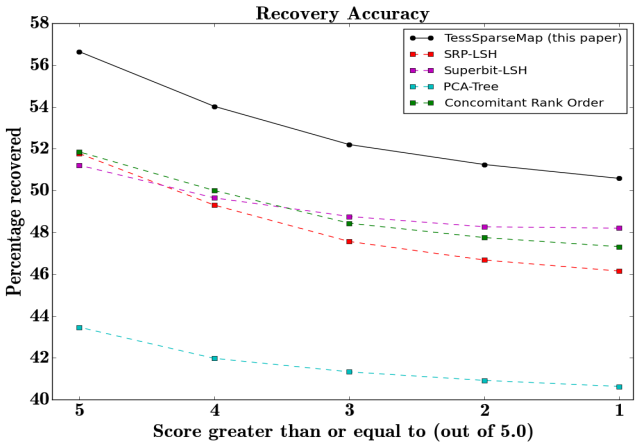
We perform experimental evaluation of our procedure on both synthetic data and the MovieLens100k dataset [1] consisting of ratings compiled from the MovieLens website. Using recommendation system terminology, one set of factors will be referred to as user factors, the other set as item factors and the inner product between user and item factors will be referred to as the rating.

For our method, we feed the factors, after some thresholding, to a schema that uses the ternary tessellation of Section (4.1.1) and a parse-tree based permutation map (described in the supplement) to get sparse representations. We use these representations to extract

[7] in our experiments, LSH is boosted by coalescing all items collected by multiple instances of random hashing



(a) Histograms of percentage of discarded items over users



(b) Recovery Accuracy for MovieLens Data

Figure 3: MovieLens Data: for comparable percentage of discarded items, recovery accuracy is much higher for our method compared to baselines (histogram y-axes scaled for uniformity)

exactly those items as would be extracted by the inverted index data structure applied to our sparse factors. The performance is compared against the following baselines- sign-random-projection hash (SRP-LSH) [6], Superbit-LSH [15], Concomitant rank order statistics [10] and PCA-tree [27].

The comparison is two-fold. First, we compute the recovery accuracy- what proportion of relevant items was actually recovered by each of the above methods after discarding certain items. Next, for each of the methods, we compute for each user the proportion of items that are discarded. The percentage of items discarded showed a large variance for some of the baselines, hence we display them as histograms. The supplementary material also contains a set of figures that plot recovery accuracy against achieved sparsity, as well as the mean percentage of discarded items across methods for synthetic and real data.

Note that the percentage of items discarded has a direct relationship with the speed-up achieved- eg, if η proportion of items are discarded, size of item list for score computation reduces to $(1-\eta)$ which results in a $\frac{1}{1-\eta}$-fold increase in speed.

6.1 Synthetic Data

For synthetic data, we randomly generate factors $\mathbf{U}$ and $\mathbf{V}$ using the standard normal distribution and construct the "rating matrix" $\mathbf{R} = \mathbf{U}\mathbf{V}^\top$. We set the factor matrix Z by concatenating the factors $\mathbf{U}$ and $\mathbf{V}$ as $\mathbf{Z} = [\mathbf{U}; \mathbf{V}]$. Performance of different methods on these factors are evaluated with respect the true rating matrix $\mathbf{R}$. Experiments show that our methods achieves superior performance compared to the baselines, by obtaining both higher percentage of discarded items (figure (2a)) as well as higher accuracy (figure (2b)). With close to 80% of the items discarded on an average, our method achieves a nearly five-fold speed-up compared to the standard retrieval technique.

6.2 MovieLens Data

We use the MovieLens100k dataset to learn low dimensional factors $\mathbf{U}$ and $\mathbf{V}$ for users and items respectively. The exact same procedure as for synthetic data is then repeated with the learned user and item factors. Experiments show that for comparable performance in percentage of discarded items (figure (3a)), our method achieves much higher recovery accuracy (figure (3b)) as compared to the baselines. With around 70% items discarded on average, our method would result in over three-fold speed-up in retrieval.

7 CONCLUSION AND FUTURE WORK

In this manuscript we presented a novel framework that exploits structural properties of sparse vectors to significantly reduce the run time computational cost of factorisation models. We developed techniques that use geometry aware permutation maps on a tessellated unit sphere to obtain high dimensional sparse embeddings for factors with sparsity patterns related to angular closeness of the original factors. We also provided deterministic and efficient realisations for the framework. Future work for this would involve the design of better tessellation and sparse mapping schema for this framework and theoretical analyses of the same.

Acknowledgements

This work was done while Avradeep Bhowmik and Nathan Liu were at Yahoo! Labs, Sunnyvale, CA.

References

[1] Movielens 100k dataset. *GroupLens Research*.

[2] E. L. Altschuler, T. J. Williams, E. R. Ratner, R. Tipton, R. Stong, F. Dowla, and F. Wooten. Possible global minimum lattice configurations for thomson's problem of charges on a sphere. *Physical Review Letters*, 78(14):2681, 1997.

[3] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 459–468. IEEE, 2006.

[4] V. N. Anh and A. Moffat. Inverted index compression using word-aligned binary codes. *Information Retrieval*, 8(1):151–166, 2005.

[5] R. B. Bapat and T. E. Raghavan. *Nonnegative matrices and applications*, volume 64. Cambridge University Press, 1997.

[6] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

[7] O. Chum, J. Philbin, A. Zisserman, et al. Near duplicate image detection: min-hash and tf-idf weighting. In *BMVC*, volume 810, pages 812–815, 2008.

[8] J. Cook. Rational formulae for the production of a spherically symmetric probability distribution. *Mathematics of Computation*, 11(58):81–82, 1957.

[9] P. Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 262–268, 1977.

[10] K. Eshghi and S. Rajaram. Locality sensitive hash functions based on concomitant rank order statistics. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 221–229. ACM, 2008.

[11] E. Gabrilovich and S. Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI*, volume 7, pages 1606–1611, 2007.

[12] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[13] J. Hicks and R. Wheeling. An efficient method for generating uniformly distributed points on the surface of an n-dimensional sphere. *Communications of the ACM*, 2(4):17–19, 1959.

[14] P. Indyk and R. 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. ACM, 1998.

[15] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian. Super-bit locality-sensitive hashing. In *Advances in Neural Information Processing Systems*, pages 108–116, 2012.

[16] A. Katanforoush and M. Shahshahani. Distributing points on the sphere, i. *Experimental Mathematics*, 12(2):199–209, 2003.

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

[18] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *The Journal of Machine Learning Research*, 11:19–60, 2010.

[19] G. Marsaglia et al. Choosing a point from the surface of a sphere. *The Annals of Mathematical Statistics*, 43(2):645–646, 1972.

[20] M. E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Communications of the ACM*, 2(4):19–20, 1959.

[21] J. Newman. Various techniques used in connection with random digits. *NBS Appl. Math. Series*, 36:12, 1951.

[22] S. M. Plis, T. Lane, and V. D. Calhoun. Directional statistics on permutations. *arXiv preprint arXiv:1007.2450*, 2010.

[23] A. Shashua and T. Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings of the 22nd international conference on Machine learning*, pages 792–799. ACM, 2005.

[24] K. Takeuchi, R. Tomioka, K. Ishiguro, A. Kimura, and H. Sawada. Non-negative multiple tensor factorization. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 1199–1204. IEEE, 2013.

[25] C. Teflioudi, F. Makari, and R. Gemulla. Distributed matrix completion. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 655–664. IEEE, 2012.

[26] M. Tegmark. An icosahedron-based method for pixelizing the celestial sphere. *The Astrophysical Journal*, 470:L81, 1996.

[27] N. Verma, S. Kpotufe, and S. Dasgupta. Which spatial partition trees are adaptive to intrinsic dimension? In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 565–574. AUAI Press, 2009.