TR 16-034

Big Data and Recommender Systems

David C. Anastasiu, Evangelia Christakopoulou, Shaden Smith, Mohit Sharma,
George Karypis

September 12, 2016

Big Data and Recommender Systems

David C. Anastasiu*[1], Evangelia Christakopoulou†[2], Shaden Smith‡[2], Mohit Sharma§[2] and George Karypis¶[2]

[1]Department of Computer Engineering, San José State University
[2]Department of Computer Science, University of Minnesota

1 Introduction

Companies such as Netflix and Amazon rely on the ability to recommend media content or products that users are likely to consume. Most companies use *recommender systems*, which are software that select products to recommend to individual customers. Recommender systems are ubiquitous in today's marketplace and have great commercial importance, as evidenced by the large number of companies that sell recommender systems solutions.

Successful recommender systems use past product purchase and satisfaction data to make high quality personalized recommendations. The volume of data available to recommender systems today is staggering and forces a total re-evaluation of the methods used to compute recommendations. Parallel and distributed computing, once a niche reserved for simulations and other scientific software, is at the heart of Big Data and must be at the forefront of algorithm design.

This paper is an overview of recommender systems in the era of Big Data. We highlight prevailing recommendation algorithms and how they have been adapted to operate in parallel computing environments. These include traditional parallel computing environments, such as OpenMP [25] and MPI [45], and also more recent distributed computing engines, such as MapReduce [29] and Spark [113]. Within the recommender systems context, we focus our discussion on scaling up two popular approaches, namely nearest neighbor and latent factor based recommendation.

*david.anastasiu@sjsu.edu
†evangel@cs.umn.edu
‡shaden@cs.umn.edu
§mohit@cs.umn.edu
¶karypis@cs.umn.edu

Table 1: Notation.

Symbol	Description
$\mathbf{X}$	matrix
$\boldsymbol{x}$	column vector
$\boldsymbol{x}^T$	row vector
m	number of users
n	number of items
$\mathbf{R}$	rating matrix of size $m \times n$
u	user
i	item
r_{ui}	rating of user u on item i
$\hat{r}_{ui}$	predicted rating of user u on item i
k	number of neighbors in neighborhood models
f	number of factors/latent dimensions
$\boldsymbol{p}_u$	user latent factor of size f for u
$\boldsymbol{q}_i$	item latent factor of size f for i
$\mathbf{P}$	matrix of size $m \times f$ containing the user latent factors
$\mathbf{Q}$	matrix of size $n \times f$ containing the item latent factors
λ	$l1$ regularization parameter
β	$l2$ regularization parameter

2 An Overview of Recommender Systems

Two common problems that recommender systems address are rating prediction and top-N recommendation. In the former, the goal is to compute the rating that a user would give for an item. In the latter, the object is to provide the user with a list of N items that they will find interesting and will likely enjoy.

Two of the most widely used approaches for computing recommendations are neighborhood-based and latent factor model-based algorithms. In the neighborhood-based approaches, the similarities between the items (item-based) or the users (user-based) are used to compute recommendations. In the latent factor model-based approaches, the users and the items are mapped in the same latent space and the items closest to a user in this space serve as the recommendations. Latent factor approaches have been shown to be superior for solving the problem of rating prediction. In contrast, item-based neighborhood approaches have proven to be superior for the top-N recommendation problem. We will discuss these approaches in the following sections.

Table 1 presents the notation that we will use in the rest of the paper.

2.1 Neighborhood-based collaborative filtering

Neighborhood-based methods rely on the similarity among users or items to generate recommendations.

2.1.1 User-based Methods

In the user-based neighborhood method [62], a set of users similar to the target user is identified as the user's neighborhood. The similarity among users is most often computed as the cosine similarity or Pearson correlation coefficient between the vectors denoting the two users' item ratings. By considering the k most similar users who have rated the target item i in the neighborhood of the uth user (known as the k-nearest-neighbor approach or kNN), the predicted rating of user u on item i is given by

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in \mathcal{N}_u^k(i)} \mathrm{sim}(u,v)(r_{vi} - \mu_v)}{\sum_{v \in \mathcal{N}_u^k(i)} \mathrm{sim}(u,v)}, \tag{1}$$

where $\mathrm{sim}(u,v)$ denotes the similarity between the users u and v, and μ_u and μ_v are the means of the ratings provided by users u and v, respectively. The symbol $\mathcal{N}_u^k(i)$ represents the set of the k most similar users to the user u, who have rated the item i.

2.1.2 Item-based Methods

The item-based neighborhood method [31, 85] computes the preference of user u on target item i by using the similarity to the target item of other items rated by the user. Analogous to the user-based neighborhood method, the cosine similarity or Pearson correlation coefficient can be used to measure similarity among items. The predicted rating of u on i is given by

$$\hat{r}_{ui} = \mu_i + \frac{\sum_{j \in \mathcal{N}_i^k(u)} \mathrm{sim}(i,j)(r_{uj} - \mu_j)}{\sum_{j \in \mathcal{N}_i^k(u)} \mathrm{sim}(i,j)}. \tag{2}$$

where $\mathrm{sim}(i,j)$ denotes the similarity between the items i and j, and μ_i and μ_j are the means of the ratings received by items i and j, respectively. The symbol $\mathcal{N}_i^k(u)$ represents the set of the k most similar items to the item i, that are rated by the user u.

In contrast to the standard item-based method which uses a predefined similarity measure like cosine or Pearson correlation, Sparse LInear Methods (SLIM) [73] learn the item-item relationships from the data instead. It has been shown that SLIM is one of the best methods for top-N recommendation. In SLIM, the rating for an item is predicted as a sparse aggregation of the existing ratings provided by the user

$$\hat{r}_{ui} = \boldsymbol{r}_u^T \boldsymbol{w}_i, \tag{3}$$

$\boldsymbol{w}_i$ is a sparse vector containing non-zero aggregation coefficients over all items. The sparse aggregation coefficient matrix $\mathbf{W}$ of size $m \times m$, capturing the item-item relationships is learned by minimizing the squared prediction error as follows

$$\begin{aligned}
\underset{\mathbf{W}}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{R} - \mathbf{R}\mathbf{W}||_F^2 + \frac{\beta}{2}||\mathbf{W}||_F^2 + \lambda||\mathbf{W}||_1 \\
\text{subject to} \quad & \mathbf{W} \geq 0 \\
& diag(\mathbf{W}) = 0.
\end{aligned} \tag{4}$$

Parameters λ and β enforce standard regularization and sparsity on the model parameters. The non-negativity constraint on $\mathbf{W}$ imposes the item-item relations to be positive. The constraint diag($\mathbf{W}$) = 0 is added to avoid trivial solutions (e.g., $\mathbf{W}$ corresponding to the identity matrix) and ensure that r_{ui} is not used to compute $\hat{r}_{ui}$. As a way of improving recommendations in very sparse datasets, Kabbur et al. [54] extended SLIM by learning the item-item similarity matrix as a product of two low-dimensional latent factor matrices.

However, both the traditional item-based methods as well as SLIM capture only pairwise relations between items and are not capable of capturing higher-order relations. Mukund et al. [31] developed HOKNN (Higher-Order k-NN), in which the most similar items are identified not only for each individual item, but also for sufficiently frequent itemsets that are present in the active user's basket. Similarly, Christakopoulou et al. [22] introduced HOSLIM (Higher-Order Sparse LInear Method), which learns both item-item and itemset-item similarities.

Another potential drawback of the item-based methods is that they estimate only a single model for all the users. In many cases, there are differences in users' behavior, which cannot be captured by a single model. Recently, GLSLIM [23] was proposed, which combines global and local SLIM models in a personalized way and automatically identifies the appropriate user subsets.

The learning of item-item similarities in SLIM [73], HOSLIM [22] and GLSLIM [23] for large datasets can be scaled by learning similarities in parallel for every target item i, as every column of the sparse aggregation coefficient matrix can be learned independently from the other columns.

2.2 Latent factor model-based collaborative filtering

2.2.1 Matrix factorization

Matrix factorization approaches are suited for the rating prediction task and have gained high popularity since the Netflix Prize [64, 66, 98]. They assume that the user-item rating matrix $\mathbf{R}$ is low rank and can be computed as a product of two matrices known as the user and the item latent factors ($\mathbf{P}$ and $\mathbf{Q}$ respectively). Then, the predicted rating for the user u on the item i is given by

$$\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i.$$

The completed matrix $\hat{\mathbf{R}} = \mathbf{P}\mathbf{Q}^T$ is used to serve the recommendation to the user for the items for which his/her preferences were unknown in the original matrix $\mathbf{R}$.

The user and the item latent factors are estimated by minimizing a regularized square loss

$$\underset{\mathbf{P},\mathbf{Q}}{\text{minimize}} \quad \frac{1}{2}\sum_{r_{ui}\in\mathbf{R}} \left(r_{ui} - \mathbf{p}_u^T\mathbf{q}_i\right)^2 + \frac{\beta}{2}\left(||\mathbf{P}||_F^2 + ||\mathbf{Q}||_F^2\right), \tag{5}$$

where the parameter β controls the Frobenius norm regularization to prevent overfitting.

2.2.2 Singular Value Decomposition (SVD)

The key idea of SVD models [30] is to factorize the user-item rating matrix to a product of two lower rank matrices, one containing the user factors and the other containing the item factors. Since conventional SVD is undefined in the presence of missing values, Cremonesi et al [24] developed a method, called PureSVD, which treats all the missing values as zeros. PureSVD is very well suited for the top-N recommendation task, and estimates the rating matrix $\mathbf{R}$ as

$$\hat{\mathbf{R}} = \mathbf{U}\mathbf{\Sigma}\mathbf{Q}^T,$$

where $\mathbf{U}$ is a $n \times f$ orthonormal matrix, $\mathbf{Q}$ is an $m \times f$ orthonormal matrix and $\mathbf{\Sigma}$ is an $f \times f$ diagonal matrix, containing the f largest singular values. It can be noted that the matrix $\mathbf{P}$ representing the user factors can be derived by

$$\mathbf{P} = \mathbf{U}\mathbf{\Sigma}.$$

The matrices $\mathbf{P}$ and $\mathbf{Q}$ can be estimated by solving the following optimization problem with orthonormal constraints

$$\begin{aligned} \underset{\mathbf{P},\mathbf{Q}}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{R} - \mathbf{P}\mathbf{Q}^T||_F^2 + \frac{\beta}{2}\left(||\mathbf{P}||_F^2 + ||\mathbf{Q}||_F^2\right) \\ \text{subject to} \quad & \mathbf{P}^T\mathbf{P} = \mathbf{I} \\ & \mathbf{Q}^T\mathbf{Q} = \mathbf{I}, \end{aligned} \tag{6}$$

where $\mathbf{I}$ is the identity matrix.

2.3 Contextual Recommendations

Beyond the users' rating history, in multiple cases, there is also a plethora of contextual data available that can be used to enhance recommendation quality [1, 12, 74, 78, 82]. Contexts include time, location, text, or any additional information associated with the users, the items or the ratings themselves.

In literature, there are works which use context as a means to pre-filter or post-filter the recommendations made [1]. The state-of-the-art in contextual recommendation [56, 83, 91] is treating every context as a different dimension, thus creating a tensor. Then, instead of having a rating matrix of two dimensions (users and items), we have a tensor of three dimensions (users, items, and time) or of four dimensions (users, items, location, and time) etc.

3 Scaling up nearest-neighbor approaches

Finding nearest neighbors is a critical component in recommender systems. In this section, we discuss recent trends in scaling up nearest neighbor identification to Big Data. Filtering and Approximate Nearest Neighbor (ANN) based approaches rely on data structures to reduce the number of computed similarities, yet provide error guarantees on identified neighborhoods. Other approaches simply choose a subset of users or items to search for neighbors in, without guaranteeing high quality results. Additional performance gains are achieved via multithreaded and distributed systems. To simplify the discussion, we will refer to the user or item rating profile as an *object* and the values it contains as *features.*

Identifying neighbors has been studied extensively for several decades in the form of two related problems. The k-nearest neighbor (kNN) search problem seeks to find the k objects that have the highest proximity to a query object. The ϵ-nearest neighbor (ϵNN) search (or *similarity search*) problem finds all objects with a similarity of at least ϵ to the query. When the proximity function is a metric, the problem is also known as *radius search*, and seeks to find all objects within ϵ distance to the query. Recommender systems initially compute and store the kNN or ϵNN for each user or item, which are known as *kNN graph construction* (kNNG) and *all-pairs similarity search* (APSS) or *similarity join* problems, respectively.

3.1 Filtering based approaches

In recent years, a number of nearest neighbor methods have been proposed for sparse vectors that work by ignoring (or filtering) object pairs that cannot be neighbors. The vector representing a user or item rating profile is inherently sparse, as a user often consumes and rates few of the overall items.

Search methods avoid comparing objects that have no features in common by using an *inverted index* data structure. The index consists of a set of lists, one for each feature among all objects, such that the jth list contains pairs (i, $r_{i,j}$) for all objects i that have a non-zero $r_{i,j}$ value for the jth feature. Additional savings can be achieved by relying on proximity thresholds. Chaudhuri et al. [20] were first to show that, given a predefined feature processing order, there comes a point in the feature processing when the un-processed query features do not have enough weight to lead to a proximity of at least ϵ with any object in the index. One can thus identify all potential neighbors for an object by checking only the inverted lists associated with the first few query features.

Bayardo et al. [15] used the idea in [20], along with a predefined object processing order, to design a filtering framework for solving the APSS problem. For each object, the framework first identifies a list of *candidates* by consulting the inverted index lists for the first few features in the query object. Each candidate is then vetted by computing a quick upper bound on its similarity with the query and comparing it to the threshold ϵ. Finally, those candidates that pass this filtering test have their similarities with the query computed and checked against ϵ. Note that the framework has a null error guarantee. It outputs the same result as when computing all pairwise similarities and then filtering those below ϵ.

The framework proposed by Bayardo et al. [15] has been extended for cosine similarity by a number of different methods [6, 10, 68, 84, 103]. In their method `L2AP`, Anastasiu and Karypis [6] introduced more stringent bounds and new filtering strategies within the filtering framework, which lead to an order of magnitude efficiency improvement over previous state-of-the-art methods. The key to the success of `L2AP`, and its namesake (L2-norm All-Pairs), is the use of the L2-norm of the remaining (unprocessed) features in the candidate and query vectors to compute several powerful similarity upper bounds which lead to significant efficiency savings in the filtering framework.

In the kNNG construction context, most proposed methods are approximate, relying on data [77] or neighborhood improvement [33] heuristics to find some of the nearest neighbors. In contrast, Anastasiu and Karypis [7] developed `L2Knng`, an exact kNNG construction filtering-based method. The main idea in `L2Knng` is to bootstrap the search with a quickly constructed approximate graph and then use the minimum similarities in neighborhoods as filtering criteria.

While we focused our summary on cosine similarity as the proximity function of choice, which has been shown to work well in many recommender systems settings, filtering has been shown to work well with many other proximity functions when applied to sparse data, including string similarity, Overlap similarity, and the Dice, Jaccard, and Tanimoto coefficients [9, 15, 104, 105].

Finding nearest neighbors is inherently a memory bound operation. As a result, shared memory parallel methods [8, 11] focus on keeping threads busy and minimizing resource contention. Existing distributed solutions for nearest neighbor graph construction generally use the MapReduce framework. Most of these methods rely on the framework's built-in features to aggregate (reduce) partial similarities of object pairs computed in mappers [13, 28, 35, 69]. While some filtering strategies can be used to avoid generating some partial similarities, these methods often suffer from high communication costs which make then inefficient for large datasets [5]. Another category of MapReduce methods use a mapper-only scheme, with no reducers [4, 5, 99]. They partition the set of objects into subsets (blocks) and apply serial nearest-neighbor search methods on block pairs. Certain block comparisons can be eliminated by relying on block-level filtering techniques. Alabduljalil et al. also investigated distributed load balancing strategies [99] and cache-conscious performance optimizations for the local searches [4].

3.2 Approximate Nearest Neighbors

Some neighborhood based schemes sometimes rely on latent decomposition as a way to capture inherent similarities between users or items. A number of different dimensionality reduction techniques, such as Singular Value Decomposition (SVD), Principal Component Analysis (PCA) [79, 101] or Latent Semantic Indexing (LSI) [30, 49] have been used to learn dense latent vectors representing users [16] or both users and items [65].

The techniques used to accelerate finding neighbors for dense objects differ based on the number of dimensions. Tree-based methods [18, 37, 38, 67, 96] have been shown to be most effective for low-dimensional embeddings (≤ 20 dimensions). They work by partitioning the search space, allowing neighbor searches to be prioritized within grids close to the one the query object is in [19]. For higher number of dimensions, Locality Sensitive Hashing (LSH) [44, 51] based algorithms find most nearest neighbors and provide theoretic error guarantees in the expectation.

LSH methods rely on families of functions that hash *signatures* of similar objects to the same bucket with high probability. Function families have been defined for hamming distance, L-p norms, cosine similarity, and the Jaccard coefficient. Parameters in LSH methods allow tuning the error rate at the cost of efficiency. Recent methods have focused on improving the serial efficiency of LSH by estimating object proximity [27, 32], probing nearby hashes [71], or taking advantage of data statistics [40, 41]. In the parallel domain, Sundaram et al. [97] proposed a parallel LSH search scheme that achieved query times of 1–2.5 ms in a database of more than 1 billion tweets. Yu et al. [108] focused instead on optimizing the kNN computation kernel in a multithreaded environment, after first retrieving neighborhood candidates from LSH tables. Their kernel fuses proximity calculation with neighborhood selection and achieves a 4x performance improvement over state-of-the-art baselines.

3.3 Sampling based approaches

A number of methods have been developed that reduce the complexity of finding neighbors by selecting a subset of the objects to compare each object against. Sarwar et al. [87] and Das et al. [26] propose clustering the objects and then computing similarities only for objects in the same cluster. Das et al. [26] use LSH or PLSI to enhance the clustering process.

Koenigstein et al. [60] use importance sampling, first introduced by Bengio et al. [17], to lower the cost of deriving pairwise relations between items. Every item is assigned a probability proportional to its empirical frequency in the training set and then the items are sampled according to their proposed distribution. Kabbur et al. [54] sample the zero entries in the training matrix and use them along with all the non-zero entries.

4 Scaling Up Latent Factor Approaches

The success of factorization-based approaches has led to a wealth of research on scaling up factorization to Big Data. These approaches span many optimization algorithms which feature different strategies for parallelization.

4.1 Top-N Recommendation

The SVD is at the core of many top-N recommendation algorithms. Computing the SVD of a large, sparse matrix is often done with Lanczos bidiagonalization [47, 93]. The bidiagonalization algorithm is iterative and relies on multiplying the sparse matrix $\mathbf{R}$ by a dense vector. Sparse matrix-vector multiplication is found in many algorithms for high performance computing and Big Data and has a large body of research [58, 70, 102, 112]. SLEPc [48] is a mature, high performance software library that computes the SVD with Lanczos bidiagonalization.

It is not always necessary to compute the SVD of the full ratings matrix. Halko et al. [46] survey and extend randomized SVD algorithms which have high accuracy when the factorization is low rank [46]. Sarwar et al. [86] developed an *incremental* SVD algorithm for when $\mathbf{R}$ is not constant and new users are being introduced. Instead of total recomputation, the factorization is only updated to incorporate the additional ratings.

4.2 Rating Prediction

Equation (5) is a non-convex, computationally expensive problem often called *matrix completion*. Several optimization algorithms have been successfully applied to large scale datasets.

4.2.1 Alternating Least Squares (ALS)

ALS was one of the first matrix completion algorithms applied to large scale data [114]. The main idea in ALS is based on the observation that, if we solve Equation (5) for one latent factor at a time, the solution has a linear least squares solution. ALS is an iterative algorithm which first minimizes with respect to $\mathbf{P}$ and then $\mathbf{Q}$. The process is repeated until convergence.

Let $\mathbf{r}_u$ be the vector of all ratings supplied by user u. $\mathbf{H}_u$ is an $|\mathbf{r}_u| \times f$ matrix whose rows are the feature vectors $\mathbf{q}_i$, for each item i rated in $\mathbf{r}_u$. Similarly, $\mathbf{r}_i$ is the vector of all ratings supplied for item i, and $\mathbf{H}_i$ is an $|\mathbf{r}_i| \times f$ matrix. ALS proceeds by updating all p_u followed by all q_i:

$$\begin{aligned}\mathbf{p}_u &\leftarrow \left(\mathbf{H}_u^T\mathbf{H}_u + \beta\mathbf{I}\right)^{-1}\mathbf{H}_u^T\mathbf{r}_u, \quad \forall u \in 1,\dots,m\\ \mathbf{q}_i &\leftarrow \left(\mathbf{H}_i^T\mathbf{H}_i + \beta\mathbf{I}\right)^{-1}\mathbf{H}_i^T\mathbf{r}_i, \quad \forall i \in 1,\dots,n.\end{aligned} \tag{7}$$

The computational complexity of ALS is $\mathcal{O}(f^2|\mathbf{R}| + f^3(m+n))$ per iteration. Zhou et al. [114] identified opportunities for parallelism in that all $\mathbf{p}_u$ can be independently computed, as can all $\mathbf{q}_i$. ALS can be computed on a large distributed system as long as $\mathbf{P}$ and $\mathbf{Q}$ fit in memory on each node. Gates et al. [42] showed how to perform the dense linear algebra in Equation (7) to achieve high throughput on modern CPUs and GPUs. Schelter et al. [88] developed a MapReduce implementation. ALS is also the algorithm of choice in the Spark machine learning library, MLlib [72].

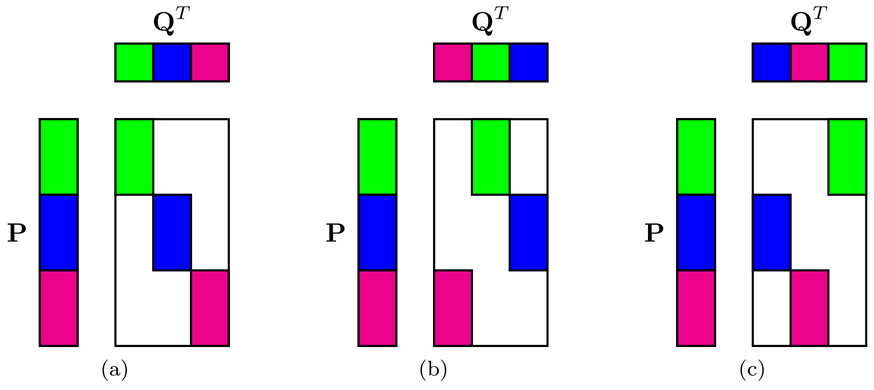

Figure 1: Stratified SGD with three workers. Colored blocks represent sets of ratings and the rows of **P** and **Q** which model them. Each block of ratings can be processed in parallel.

4.2.2 Stochastic Gradient Descent (SGD)

SGD is an optimization algorithm that trades a large number of iterations for a low computational complexity. SGD is a conceptually simple method, in which an iteration consists of selecting ratings at random and updating the factorization based on the local gradient. Updates are of the form:

$$\begin{aligned} e_{ui} &\leftarrow r_{ui} - \mathbf{p}_u^T \mathbf{q}_i, \\ \mathbf{p}_u &\leftarrow \mathbf{p}_u + \eta \left(e_{ui} \mathbf{q}_i - \beta \mathbf{p}_u \right), \\ \mathbf{q}_i &\leftarrow \mathbf{q}_i + \eta \left(e_{ui} \mathbf{p}_u - \beta \mathbf{q}_i \right), \end{aligned} \tag{8}$$

where η is a hyperparameter representing the learning rate. The complexity of Equation 8 is linear in f, resulting in a total complexity of $\mathcal{O}(f|\mathbf{R}|)$ per iteration. The low complexity and simple implementation of SGD has led to it being widely adopted by researchers and industry alike.

SGD is less straightforward than ALS to parallelize. Since processing a rating updates rows of both **P** and **Q**, special care must be taken to prevent the same rows from being modified at the same time (called a *race condition*). There are two broad approaches for parallelizing SGD.

Stratified methods are based on the observation that, if two ratings do not overlap, meaning they have neither a row or column in common, then they can be updated with Equation (8) at the same time. Gemulla et al. [43] first used this technique with DSGD, which imposes a grid on **R** and identifies blocks which can be processed in parallel. This strategy is illustrated in Figure 1. Stratification has proven to be an effective strategy for parallelizing SGD and has been extended in works on multithreaded environment [75, 76, 115], distributed systems [100, 111], and even GPUs [53].

Asynchronous methods rely on the stochastic nature of SGD to allow overlapping updates. Teflioudi et al. first presented this technique with ASGD [100], an SGD algorithm for distributed computing environments. During ASGD, nodes maintain working copies of $\mathbf{P}$ and $\mathbf{Q}$ (e.g., locally modified) and updates are asynchronously communicated several times per iteration. Overlapping updates are averaged with the master copy and sent to workers. Petroni and Querzoni extended this work with GASGD [80], which uses graph partitioning to distribute $\mathbf{R}$ in order to reduce conflicting updates. Recht et al. introduced Hogwild! [81], a parallel SGD algorithm for multithreaded environments which simply applies Equation (8) in parallel without concern for overlapping entries. When $\mathbf{R}$ is sufficiently sparse, updates to $\mathbf{P}$ and $\mathbf{Q}$ are unlikely to overlap and convergence is still observed.

4.2.3 Coordinate Descent (CCD++)

Coordinate descent is a class of optimization algorithms which update one parameter of the output at a time. Yu et al. [109] explored coordinate descent algorithms for matrix completion and developed CCD++, a parallel algorithm for multithreaded and distributed environments. CCD++ updates columns of $\mathbf{P}$ and $\mathbf{Q}$ in sequence, with a single parameter update taking the form

$$p_{us} \leftarrow \frac{\sum_{r_{ui} \in \mathbf{R}} (r_{ui} - \mathbf{p}_u^T \mathbf{q}_i + p_{us} q_{is}) q_{is}}{\beta + \sum_{r_{ui} \in \mathbf{R}} q_{is}^2}. \tag{9}$$

If all $(r_{ui} - \mathbf{p}_u^T \mathbf{q}_i)$ are pre-computed, CCD++ has a complexity of $\mathcal{O}(f|\mathbf{R}|)$ per iteration, matching SGD. Each column entry is independent and can thus be computed in parallel.

4.2.4 Emerging Algorithms

While ALS, SGD, and CCD++ have received the majority of attention, special note should be made of emerging methods based on the alternating direction method of multipliers (ADMM). ADMM itself is not a new algorithm, but has recently garnered much attention due to its potential for high scalability in parallel algorithms. One such recent work is DS-ADMM [110], a stochastic variant of ADMM for matrix factorization. AO-ADMM [50] is another recent ADMM work for matrix factorization and interestingly can handle objectives for both the top-N and rating prediction problems.

4.3 Extensions to Tensor Factorization

Section 2.3 discussed contextual information as a means to improve recommendation quality. This is often accomplished with *tensor factorization.*

The canonical polyadic decomposition (CPD) is a generalization of the SVD to tensors that is often used for top-N recommendation. Several parallel algorithms have been designed for the MapReduce paradigm, including GigaTensor [55] and Haten2 [52]. High performance software for distributed systems include DFacTo [21] and SPLATT [94]. The Tucker decomposition is another SVD generalization and has been studied in multithreaded [14] and distributed [59] environments.

Rating prediction with tensors also uses the CPD and extends Equation (5). Shao [89] first explored parallel ALS and SGD algorithms for multithreaded systems. Shin and Kang [92] developed MapReduce algorithms based on ALS. Karlsson et al. [57] presented ALS and CCD++ algorithms for high performance distributed systems. Smith et al. [95] later developed high performance ALS, SGD, and CCD++ algorithms and made them available in SPLATT.

5 Online Recommendation

So far, the paper has focused on ways to compute recommendation models quickly. In large-scale recommender systems, beyond model computation, it is crucial that

- the recommendations are presented to the user very fast, and that
- the recommender system is able to handle new data that comes in continuously.

Below, we present some approaches that large-scale systems employ in order to tackle these challenges.

5.1 Efficient Retrieval of Recommendations

Koenigstein et al. [60,61] propose the use of metric trees, which are binary space-partitioning trees, for efficient retrieval of recommendations in the context of neighborhood-based [60] and matrix factorization [61] methods. The authors also suggest even faster approximate retrieval by clustering users and precomputing recommendations for the cluster centers. The system then provides, as an approximate result, the cluster center recommendation for the cluster that the user belongs in.

Yin et. al. [106] precompute f lists of ordered items, where every list corresponds to a latent factor. When an online query comes, the top-N items are returned from the f lists in a priority list. The threshold-based algorithm [36] is employed, for updating the priority of the current list, as well as the threshold store. Yin et. al. [107] based on the previous algorithm, also present a clustering-based branch and bound algorithm for fast computation of point-of-interest (POI) recommendation. The POI vectors containing the POI attributes are first clustered into buckets and then within each bucket, they are sorted based on their length. This happens offline. At the time of online recommendation, when a query appears, the buckets are sorted according to the inner product of the query and the upper bound vector of the bucket.

5.2 Cold-Start Recommendation

In real-world recommender systems, thousands of new users and items are introduced constantly, as well as new ratings for the existing users and items.

Thus, recommender systems must have an online component which updates the model and computes recommendations using the newest data.

There are two types of updates that are usually executed when new data becomes available. A periodic rebuild, which takes place on a daily or weekly basis, retrains the model based on all available data. Conversely, a continuous online update computes the recommendations incrementally without retraining the model. Koren [63] and Aizenberg et al. [3] present a way of computing this continuous online update for new users, by employing a model which does not parameterize users. Thus, the model can handle new users as soon as they provide feedback to the system.

When new items are introduced into the system, the basic neighborhood-based or latent factor based approaches can not be applied to compute personalized recommendations, because there are no prior preferences associated with those items. This is known as the cold-start recommendation problem. To solve this problem, cold-start recommender systems take the characteristics of the items being recommended into account in addition to the prior preferences of the users. Regression-based latent factor models (RLFM) [2] is a general technique that can also work in item cold-start scenarios. AFM [39] learns item attributes to latent feature mapping by learning a factorization of the rating matrix into user and item latent factors. User-specific Feature-based Similarity Models (UFSM) [34] learn a personalized linear combination of user-independent similarity functions known as global similarity functions for cold-start top-N item recommendations. Similar to UFSM, Sharma et al [90] developed the Feature-based factorized Bilinear Similarity Model (FBSM), which learns a bilinear similarity function to capture pairwise dependencies between the features for cold-start top-n item recommendations.

6 Conclusion

We presented an overview of recommender systems in the era of Big Data, focusing on two specific challenges: how to scale up finding nearest neighbors and how to scale latent factor recommendation methods. For each category of methods, we presented both serial and parallel advances in obtaining efficient and effective solutions. Most existing recommendation methods assume all ratings are given as input and do not change. An important future direction for recommender systems in Big Data is developing parallel methods that can efficiently update recommendation models or provide online recommendations without loss of quality.

References

[1] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[2] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 19–28. ACM, 2009.

[3] Natalie Aizenberg, Yehuda Koren, and Oren Somekh. Build your own music recommender by modeling internet radio streams. In *Proceedings of the 21st international conference on World Wide Web*, pages 1–10. ACM, 2012.

[4] Maha Alabduljalil, Xun Tang, and Tao Yang. Cache-conscious performance optimization for similarity search. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 713–722, New York, NY, USA, 2013. ACM.

[5] Maha Ahmed Alabduljalil, Xun Tang, and Tao Yang. Optimizing parallel algorithms for all pairs similarity search. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 203–212, New York, NY, USA, 2013. ACM.

[6] David C. Anastasiu and George Karypis. L2ap: Fast cosine similarity search with prefix l-2 norm bounds. In *30th IEEE International Conference on Data Engineering*, ICDE '14, 2014.

[7] David C. Anastasiu and George Karypis. L2knng: Fast exact k-nearest neighbor graph construction with l2-norm pruning. In *24th ACM International Conference on Information and Knowledge Management*, CIKM '15, 2015.

[8] David C. Anastasiu and George Karypis. Pl2ap: Fast parallel cosine similarity search. In *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, IA3 '15, pages 8:1–8:8, New York, NY, USA, 2015. ACM.

[9] David C. Anastasiu and George Karypis. Efficient identification of tanimoto nearest neighbors. In *The 3rd IEEE International Conference on Data Science and Advanced Analytics*, DSAA '16, 2016.

[10] Amit Awekar and Nagiza F. Samatova. Fast matching for all pairs similarity search. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '09, pages 295–300, Washington, DC, USA, 2009. IEEE Computer Society.

[11] Amit Awekar and Nagiza F Samatova. Parallel all pairs similarity search. In *Proceedings of the 10th International Conference on Information and Knowledge Engineering*, IKE '11, 2011.

[12] Linas Baltrunas, Bernd Ludwig, and Francesco Ricci. Matrix factorization techniques for context aware recommendation. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 301–304. ACM, 2011.

[13] Ranieri Baraglia, Gianmarco De Francisci Morales, and Claudio Lucchese. Document similarity self-join with mapreduce. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 731–736, Washington, DC, USA, 2010. IEEE Computer Society.

[14] Muthu Baskaran, Benoît Meister, Nicolas Vasilache, and Richard Lethin. Efficient and scalable computations with sparse tensors. In *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, pages 1–6. IEEE, 2012.

[15] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 131–140, New York, NY, USA, 2007. ACM.

[16] Robert M Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 43–52. IEEE, 2007.

[17] Yoshua Bengio, Jean-Sébastien Senécal, et al. Quick training of probabilistic neural nets by importance sampling. In *AISTATS*, 2003.

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

[19] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 97–104, New York, NY, USA, 2006. ACM.

[20] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 5–, Washington, DC, USA, 2006. IEEE Computer Society.

[21] Joon Hee Choi and S Vishwanathan. DFacTo: Distributed factorization of tensors. In *Advances in Neural Information Processing Systems*, pages 1296–1304, 2014.

[22] Evangelia Christakopoulou and George Karypis. Hoslim: Higher-order sparse linear method for top-n recommender systems. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 38–49. Springer, 2014.

[23] Evangelia Christakopoulou and George Karypis. Local item-item models for top-n recommendation. In *Proceedings of the Tenth ACM Conference on Recommender Systems*, RecSys '16. ACM, 2016.

[24] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM.

[25] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[26] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

[27] Anirban Dasgupta, Ravi Kumar, and Tamas Sarlos. Fast locality-sensitive hashing. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1073–1081, New York, NY, USA, 2011. ACM.

[28] G. De Francisci, C. Lucchese, and R. Baraglia. Scaling out all pairs similarity search with mapreduce. *Large-Scale Distributed Systems for Information Retrieval*, page 27, 2010.

[29] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[30] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990.

[31] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

[32] Wei Dong, Moses Charikar, and Kai Li. Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 123–130, New York, NY, USA, 2008. ACM.

[33] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 577–586, New York, NY, USA, 2011. ACM.

[34] Asmaa Elbadrawy and George Karypis. User-specific feature-based similarity models for top-n recommendation of new items. *ACM Trans. Intell. Syst. Technol.*, 6(3):33:1–33:20, April 2015.

[35] Tamer Elsayed, Jimmy Lin, and Douglas W. Oard. Pairwise document similarity in large collections with mapreduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, HLT-Short '08, pages 265–268, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[36] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003.

[37] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977.

[38] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, June 1998.

[39] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, Steffen Rendle, and Schmidt-Thie Lars. Learning attribute-to-feature mappings for cold-start recommendations. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 176–185, Washington, DC, USA, 2010. IEEE Computer Society.

[40] Jinyang Gao, Hosagrahar Visvesvaraya Jagadish, Wei Lu, and Beng Chin Ooi. Dsh: Data sensitive hashing for high-dimensional k-nn search. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 1127–1138, New York, NY, USA, 2014. ACM.

[41] Jinyang Gao, H.V. Jagadish, Beng Chin Ooi, and Sheng Wang. Selective hashing: Closing the gap between radius search and k-nn search. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 349–358, New York, NY, USA, 2015. ACM.

[42] Mark Gates, Hartwig Anzt, Jakub Kurzak, and Jack Dongarra. Accelerating collaborative filtering using concepts from high performance computing. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 667–676. IEEE, 2015.

[43] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM, 2011.

[44] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers.

[45] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[46] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. 2009.

[47] Vicente Hernández, José E. Román, and Andrés Tomás. A robust and efficient parallel svd solver based on restarted lanczos bidiagonalization. *Electronic Transactions on Numerical Analysis*, 31:68–85, 2008.

[48] Vicente Hernández, José E. Román, and Vicente Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):351–362, 2005.

[49] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

[50] Kejun Huang, Nicholas D Sidiropoulos, and Athanasios P Liavas. A flexible and efficient algorithmic framework for constrained matrix and tensor factorization. *arXiv preprint arXiv:1506.04209*, 2015.

[51] 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. ACM, 1998.

[52] Inah Jeon, Evangelos E Papalexakis, U Kang, and Christos Faloutsos. Haten2: Billion-scale tensor decompositions. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1047–1058. IEEE, 2015.

[53] Jing Jin, Siyan Lai, Su Hu, Jing Lin, and Xiaola Lin. Gpusgd: A gpu-accelerated stochastic gradient descent algorithm for matrix factorization. *Concurrency and Computation: Practice and Experience*, 2015.

[54] Santosh Kabbur, Xia Ning, and George Karypis. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 659–667. ACM, 2013.

[55] U Kang, Evangelos E. Papalexakis, Abhay Harpale, and Christos Faloutsos. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–324. ACM, 2012.

[56] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86. ACM, 2010.

[57] Lars Karlsson, Daniel Kressner, and André Uschmajew. Parallel algorithms for tensor completion in the cp format. *Parallel Computing*, 2015.

[58] Kamer Kaya, Bora Uçar, and Ümit V Çatalyürek. Analysis of partitioning models and metrics in parallel sparse matrix-vector multiplication. In *International Conference on Parallel Processing and Applied Mathematics*, pages 174–184. Springer, 2013.

[59] Oguz Kaya and Bora Uçar. High-performance parallel algorithms for the tucker decomposition of higher order sparse tensors. Technical report, Inria-Research Centre Grenoble–Rhône-Alpes, 2015.

[60] Noam Koenigstein and Yehuda Koren. Towards scalable and accurate item-oriented recommendations. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 419–422. ACM, 2013.

[61] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 535–544. ACM, 2012.

[62] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, March 1997.

[63] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[64] Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81:1–10, 2009.

[65] Yehuda Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):1, 2010.

[66] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.

[67] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 614–623, New York, NY, USA, 1998. ACM.

[68] Dongjoo Lee, Jaehui Park, Junho Shim, and Sang-goo Lee. An efficient similarity join algorithm with cosine similarity predicate. In *Proceedings of the 21st International Conference on Database and Expert Systems Applications: Part II*, DEXA'10, pages 422–436, Berlin, Heidelberg, 2010. Springer-Verlag.

[69] Jimmy Lin. Brute force and indexed approaches to pairwise document similarity comparisons with mapreduce. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 155–162, New York, NY, USA, 2009. ACM.

[70] Xing Liu, Mikhail Smelyanskiy, Edmond Chow, and Pradeep Dubey. Efficient sparse matrix-vector multiplication on x86-based many-core processors. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 273–282. ACM, 2013.

[71] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 950–961. VLDB Endowment, 2007.

[72] Xiangrui Meng, Joseph Bradley, B Yuvaz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *JMLR*, 17(34):1–7, 2016.

[73] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506. IEEE, 2011.

[74] Xia Ning and George Karypis. Sparse linear methods with side information for top-n recommendations. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 155–162. ACM, 2012.

[75] Yusuke Nishioka and Kenjiro Taura. Scalable task-parallel sgd on matrix factorization in multicore architectures. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 1178–1184. IEEE, 2015.

[76] Jinoh Oh, Wook-Shin Han, Hwanjo Yu, and Xiaoqian Jiang. Fast and robust parallel sgd matrix factorization. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 865–874. ACM, 2015.

[77] Youngki Park, Sungchan Park, Sang-goo Lee, and Woosung Jung. Greedy filtering: A scalable algorithm for k-nearest neighbor graph construction. In *Database Systems for Advanced Applications*, volume 8421 of *Lecture Notes in Computer Science*, pages 327–341. Springer-Verlag, 2014.

[78] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[79] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[80] Fabio Petroni and Leonardo Querzoni. Gasgd: stochastic gradient descent for distributed asynchronous matrix completion via graph partitioning. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 241–248. ACM, 2014.

[81] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[82] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 635–644. ACM, 2011.

[83] Steffen Rendle and Lars Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 81–90. ACM, 2010.

[84] Leonardo A. Ribeiro and Theo Härder. Efficient set similarity joins using min-prefixes. In *Proceedings of the 13th East European Conference on Advances in Databases and Information Systems*, ADBIS '09, pages 88–102, Berlin, Heidelberg, 2009. Springer-Verlag.

[85] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, New York, NY, USA, 2001. ACM.

[86] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28. Citeseer, 2002.

[87] Badrul M Sarwar, George Karypis, Joseph Konstan, and John Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the fifth international conference on computer and information technology*, volume 1, 2002.

[88] Sebastian Schelter, Christoph Boden, Martin Schenck, Alexander Alexandrov, and Volker Markl. Distributed matrix factorization with mapreduce using a series of broadcast-joins. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 281–284. ACM, 2013.

[89] Weijia Shao. Tensor completion. Master's thesis, Universität des Saarlandes Saarbrücken, 2012.

[90] Mohit Sharma, Jiayu Zhou, Junling Hu, and George Karypis. Feature-based factorized bilinear similarity model for cold-start top-n item recommendation. In *Proceedings of the 15th SIAM International Conference on Data Mining, SDM*, volume 15, pages 190–198. SIAM, 2015.

[91] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. Tfmap: optimizing map for top-n context-aware recommendation. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 155–164. ACM, 2012.

[92] Kijung Shin and U. Kang. Distributed methods for high-dimensional and large-scale tensor factorization. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 989–994, Dec 2014.

[93] Horst D Simon and Hongyuan Zha. Low-rank matrix approximation using the lanczos bidiagonalization process with applications. *SIAM Journal on Scientific Computing*, 21(6):2257–2274, 2000.

[94] Shaden Smith and George Karypis. A medium-grained algorithm for distributed sparse tensor factorization. In *30th IEEE International Parallel & Distributed Processing Symposium (IPDPS'16)*, 2016.

[95] Shaden Smith, Jongsoo Park, and George Karypis. An exploration of optimization algorithms for high performance tensor completion. In *Proceedings of the 2016 ACM/IEEE conference on Supercomputing*, 2016, to appear.

[96] Robert F. Sproull. Refinements to nearest-neighbor searching ink-dimensional trees. *Algorithmica*, 6(1):579–589, 1991.

[97] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *Proc. VLDB Endow.*, 6(14):1930–1941, September 2013.

[98] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of machine learning research*, 10(Mar):623–656, 2009.

[99] Xun Tang, Maha Alabduljalil, Xin Jin, and Tao Yang. Load balancing for partition-based similarity search. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 193–202, New York, NY, USA, 2014. ACM.

[100] Christina Teflioudi, Faraz Makari, and Rainer Gemulla. Distributed matrix completion. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 655–664. IEEE, 2012.

[101] Michael E. Tipping and Chris M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 61:611–622, 1999.

[102] Richard Vuduc, James W Demmel, and Katherine A Yelick. Oski: A library of automatically tuned sparse matrix kernels. In *Journal of Physics: Conference Series*, volume 16, page 521. IOP Publishing, 2005.

[103] Jiannan Wang, Guoliang Li, and Jianhua Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 85–96, New York, NY, USA, 2012. ACM.

[104] Chuan Xiao, Wei Wang, and Xuemin Lin. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. *Proc. VLDB Endow.*, 1(1):933–944, August 2008.

[105] Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. Efficient similarity joins for near duplicate detection. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 131–140, New York, NY, USA, 2008. ACM.

[106] Hongzhi Yin, Bin Cui, Ling Chen, Zhiting Hu, and Xiaofang Zhou. Dynamic user modeling in social media systems. *ACM Transactions on Information Systems (TOIS)*, 33(3):10, 2015.

[107] Hongzhi Yin, Xiaofang Zhou, Yingxia Shao, Hao Wang, and Shazia Sadiq. Joint modeling of user check-in behaviors for point-of-interest recommendation. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1631–1640. ACM, 2015.

[108] Chenhan D. Yu, Jianyu Huang, Woody Austin, Bo Xiao, and George Biros. Performance optimization for the k-nearest neighbors kernel on x86 architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 7:1–7:12, New York, NY, USA, 2015. ACM.

[109] Hsiang-Fu Yu, Cho-Jui Hsieh, I Dhillon, et al. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 765–774. IEEE, 2012.

[110] Zhi-Qin Yu, Xing-Jian Shi, Ling Yan, and Wu-Jun Li. Distributed stochastic admm for matrix factorization. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1259–1268. ACM, 2014.

[111] Hyokun Yun, Hsiang-Fu Yu, Cho-Jui Hsieh, SVN Vishwanathan, and Inderjit Dhillon. Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. *Proceedings of the VLDB Endowment*, 7(11):975–986, 2014.

[112] Albert-Jan Yzelman and Dirk Roose. High-level strategies for parallel shared-memory sparse matrix-vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 2014.

[113] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010.

[114] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Algorithmic Aspects in Information and Management*, pages 337–348. Springer, 2008.

[115] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. A fast parallel sgd for matrix factorization in shared memory systems. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 249–256. ACM, 2013.

David Anastasiu is an Assistant Professor in the Department of Computer Engineering at San José State University, in San José, CA. His research interests fall broadly at the intersection of data mining, high performance computing, information retrieval, and cloud and distributed computing. Much of his work has been focused on scalable and efficient methods for analyzing sparse data. He has developed serial and parallel methods for identifying near neighbors, methods for characterizing how user behavior changes over time, and methods for personalized and collaborative presentation of Web search results. His work has been published in many top-tier conferences and journals, and he serves on the program committees of several IEEE and ACM conferences.

Evangelia Christakopoulou is a PhD Candidate in the Department of Computer Science & Engineering at the University of Minnesota, Twin Cities. Her research interests lie in the intersection of data mining and machine learning, with a special emphasis on recommender systems. The focus of her research is the development of algorithms for improving the top-N recommendation quality. Her work has been published in several top-tier venues and she is a student member of ACM.

Shaden Smith is a PhD Candidate in the Department of Computer Science & Engineering at the University of Minnesota, Twin Cities. His research interests span high performance computing, tensor decomposition, and sparse linear algebra. The bulk of his research covers parallel algorithms for tensor decomposition. Shaden has been published in several top venues and is a student member of ACM, IEEE, and SIAM.

Mohit Sharma is a PhD Candidate in the Department of Computer Science & Engineering at the University of Minnesota, Twin Cities. His research primarily focuses on recommender systems. He has developed methods for cold-start Top-N item recommendations that takes into account the interaction among the features of the items.

George Karypis is a Distinguished McKnight University Professor and an ADC Chair of Digital Technology at the Department of Computer Science & Engineering at the University of Minnesota, Twin Cities. His research interests span the areas of data mining, high performance computing, information retrieval, collaborative filtering, bioinformatics, cheminformatics, and scientific computing. He has coauthored over 260 papers on these topics and two books ("Introduction to Protein Structure Prediction: Methods and Algorithms" (Wiley, 2010) and "Introduction to Parallel Computing" (Publ. Addison Wesley, 2003, 2nd edition)). He is serving on the program committees of many conferences and workshops on these topics, and on the editorial boards of the IEEE Transactions on Knowledge and Data Engineering, ACM Transactions on Knowledge Discovery from Data, Data Mining and Knowledge Discovery, Social Network Analysis and Data Mining Journal, International Journal of Data

Mining and Bioinformatics, the journal on Current Proteomics, Advances in Bioinformatics, and Biomedicine and Biotechnology.