The Dissertation Committee for Donghyuk Shin
certifies that this is the approved version of the following dissertation:

A Multi-Scale Framework for Graph Based Machine Learning Problems

Committee:

Inderjit S. Dhillon, Supervisor

Andrew B. Whinston

Lili Qiu

Deepayan Chakrabarti

A Multi-Scale Framework for Graph Based Machine Learning Problems

by

Donghyuk Shin, B.E.; B.Eco.; M.S.C.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2017

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor and mentor Professor Inderjit Dhillon. He has been extremely supportive with his indispensable guidance and encouragement, and constantly inspired me with his deep and broad insight in conducting research as well as in its presentation. Having him as an advisor has been invaluable for making the whole experience as a graduate student worthwhile and life-changing.

I gratefully acknowledge my Ph.D. committee members, Lili Qiu, Deepayan Chakrabarti, and Andrew Whinston. Suggestions and feedback from Professor Lili Qiu have helped reinforce research ideas, Professor Deepayan Chakrabarti has motivated and sharpened my thoughts with insightful questions and comments, and Professor Andrew Whinston has been my mentor for broadening my research to the field of business. I also had the great privilege to work with my academic grandfather Professor Beresford Parlett, whose expertise and vision have helped shaping this thesis.

I am grateful to my research internship mentors and friends Dr. Suleyman Cetintas and Dr. Kuang-Chih Lee from Yahoo Research for their support and advise. I also thank my lab-mates, colleagues and coauthors from the Center for Big Data Analytics and wish to extend my gratitude to Berkant Savas, Ambuj Tewari, Nagarajan Natarajan, Si Si, and David Inouye. I am fortunate to have met many great friends as well, who have made life in Austin an enriching experience.

Finally, and most importantly, I dedicate this thesis to my beloved wife, my dear parents and brother. It is their unconditional and unswerving love and support that made this thesis possible.

A Multi-Scale Framework for Graph Based Machine Learning Problems

Publication No. ____________

Donghyuk Shin, Ph.D.
The University of Texas at Austin, 2017

Supervisor: Inderjit S. Dhillon

Graph data have become essential in representing and modeling relationships between entities and complex network structures in various domains such as social networks and recommender systems. As a main contributor of the recent Big Data trend, the massive scale of graphs in modern machine learning problems easily overwhelms existing methods and thus sophisticated scalable algorithms are needed for real-world applications. In this thesis, we develop a novel *multi-scale framework* based on the divide-and-conquer principle as an effective and scalable approach for machine learning tasks involving large sparse graphs. We first demonstrate how our multi-scale framework can be applied to the problem of computing the spectral decomposition of massive graphs, which is one of the most fundamental low-rank matrix approximations used in numerous machine learning tasks. While popular solvers suffer from slow convergence, especially when the desired rank is large, our method exploits the clustering structure of the graph and achieves superior performance compared to existing algorithms in terms of both accuracy and scalability.

While the main goal of the divide-and-conquer approach is to efficiently compute solutions for the original problem, the proposed multi-scale framework further admits an attractive but less obvious feature that machine learning problems can benefit from. Particularly, we consider partial solutions of the subproblems computed in the process as localized models of the entire problem. By doing so, we can combine models at multiple scales from local to global and generate a holistic view of the underlying problem to achieve better performance than a single global view. We adapt such multi-scale view for the problems of link prediction in social networks and collaborative filtering in recommender systems with additional side information to obtain a model that can make accurate and robust predictions in a scalable manner.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Graph data have become increasingly important in representing and modeling dyadic relationships between entities and complex network structures with broad applications in the field of machine learning and data mining. It is one of the key contributing factors of the recent Big Data trend appearing in large-scale datasets from various applications, whenever there is a need to represent how things are connected or related to one another. Some canonical examples include hyperlinks between webpages in web graphs, friendships between users in social networks, user ratings of items in recommender systems and associations between genes and diseases in biological networks. The relationships that graphs in many real-world applications represent are unstructured and sparse. It is usually the case that they follow a pattern of sparsity that is not uniform, but irregular and highly skewed towards a few entities (e.g., power-law distribution) [80]. While these inherent characteristics make modern graph-based machine learning problems, such as identifying new relationships, quite challenging, the massive scale of such graphs has driven to an even greater need for developing scalable and sophisticated methods for graphs.

In this thesis, we develop a novel *multi-scale framework* based on the divide-and-conquer principle as an effective and scalable approach for machine learning tasks involving sparse graphs. While the divide-and-conquer paradigm has been a classical approach for various computer science problems (e.g., sorting and eigenvalue computation of tridiagonal matrices), it has not been widely employed in problems that arise in graph-based machine learning. The main idea is to first divide the original problem into several smaller subproblems defined only on a subset of data that are more manageable and efficient to solve. Then, the partial solutions from each subproblem are merged or conquered to obtain a solution for the original problem. A desirable property of the divide-and-conquer paradigm is that it can be naturally parallelized, since each subproblem can be solved independently, which is crucial for modern large scale applications.

However, developing a divide-and-conquer approach is usually nontrivial and the interconnected nature of graph data makes it an even more challenging task. In particular, a good algorithm should partition the graph such that (1) subproblems can be solved efficiently, and (2) solutions to the subproblems can be easily combined to give the solution to the original problem. We address these issues and demonstrate how our multi-scale framework can be effectively utilized for large scale machine learning problems with sparse graphs to obtain accurate and scalable solutions.

Though, in general, the main goal of the divide-and-conquer approach is to efficiently compute solutions for the original problem, it reveals an interesting and appealing aspect for machine learning tasks. Particularly, we can view partial solutions of the subproblems as localized models of the entire problem. Thus, instead of simply using these localized models to compute a global solution for the original task, we can combine models at multiple scales from local to global generated by our divide-and-conquer approach in order to achieve better performance for the underlying problem. By taking such multi-scale view of the data, we are able to obtain a model that can make accurate and robust predictions in various machine learning problems such as link prediction and recommender systems.

To this end, we demonstrate the efficiency of our multi-scale approach on three important machine learning problems including (1) spectral decomposition and singular value decomposition of graphs, (2) link prediction in social network analysis and (3) recommender systems with additional information. We propose a novel multi-scale framework that achieves superior performance compared to other state-of-the-art methods in a robust and scalable manner. Particularly, we show the first aspect of efficiently computing a global solution by applying our multi-scale framework to the spectral decomposition and singular value decomposition problems. The second aspect of our proposed method, i.e., exploiting multi-scale view of the data, is shown for the problem of link prediction, where we combine models at various scales to make accurate and robust predictions. Another important application that has gained great interest is recommender systems, where interactions between users and items can be represented as a bipartite graph.

First, we consider the spectral decomposition of massive graphs, which is one of the most informative and fundamental matrix approximations. There are cases when only very few eigenvectors are needed, such as the leading eigenvector for PageRank [40] or the Fiedler vector for spectral clustering [81], where classical single-vector iterative algorithms (e.g., power method) [58, 84] are widely used. However, it is often the case that a reasonably large number k of eigenvalues and eigenvectors are needed. This computation is required in various machine learning applications such as link prediction, recommender systems and semi-supervised learning. The aforementioned single-vector algorithms fall short in such case as they are restricted to computing only the leading eigenvector or have difficulty in computing multiplicities or clusters of eigenvalues. To avoid such problems, block versions of iterative algorithms, such as the randomized SVD [36] or block Lanczos [75], can be employed. Nonetheless, these methods generally suffer from slow convergence due to its random initialization and does not scale well with sufficiently large k. In this thesis, we develop a multi-scale framework to compute large number of dominant eigenpairs of the entire graph in an efficient and scalable manner. Specifically, we exploit the cluster structure of the graph to obtain a good initialization for standard eigensolvers, which is based on the key observations that the union of all cluster's subspace is close to that of the original graph. From this, we extend the developed method for spectral decomposition to computing the more general singular value decomposition of large-scale graphs and demonstrate its effectiveness.

Next we consider the problem of link prediction, which is an important task in social network analysis [67]. Comprehensive proximity measures between users that are based on the structural properties of the graph (e.g., Katz) have been shown to be quite effective for link prediction. The core assumption is that users with a high proximity score implies they are close and hence will have a good chance of forming a link in the future. However, such measures typically involve high computational costs making it infeasible for large scale graphs in the big data era. An alternative is to use a low rank approximation of the graph to obtain a good estimation of the proximity measure. Due to our proposed framework, we can generate a multi-scale view of the graph and efficiently compute low-rank approximations at different levels of granularity. Our approach considers scores from local to global views of the network based on the multi-scale low-rank approximation, whereas traditional methods use a single low-rank representation of the graph resulting in a single prediction score. As a result, we are able to capture more information from local views within clusters, where most of the new links appear, while capturing links between clusters through a more global view at the same time.

For the third and final problem, we investigate recommender systems, where the data can also be represented as a graph encoding relationships between users and items [2, 56]. While such user-item matrix lies at the core of recommender systems, numerous recommendation scenarios involving additional information about users, items and their interactions have emerged in recent years. In this thesis, we exploit information beyond the user-item matrix of conventional collaborative filtering settings and adapt a multi-scale view of the problem to enhance recommendation quality. The type of information additional to the user-item matrix can be classified into two main categories related to its source: (1) interaction information associated with the interplay between users and items, and (2) rich side-information about users and items [91]. The former considers contextual information such as time and location, while the latter includes examples such as profile, user-generated content and social network information. We discuss each case in two real-life applications: mobile application (app) recommendation and blog recommendation. For the app recommendation problem, we develop a novel algorithm that utilizes app usage patterns from multiple scales to make dynamic recommendations. For the blog recommendation problem, we study how to effectively leverage side-information of users and items with the goal of achieving improved recommendation performance.

1.1 Outline

The thesis discusses multi-scale approaches for three important problems using graphs as the main data source in machine learning and data mining. Chapter 2 discusses the first part of the thesis, where we explore the fundamental problem of computing spectral decompositions of massive graphs, especially when reasonably large number of eigenvalues and eigenvectors are needed. We present a novel multi-scale spectral decomposition method (MSEIGS) based on the divide-and-conquer principle. Both theoretical and empirical analysis is presented to highlight the effectiveness of our method. We also propose an early-termination strategy to efficiently compute quality approximations. The usefulness of our method is demonstrated in two important applications that require large number of eigenvectors of graphs. In the subsequent section, we propose a multi-scale singular value decomposition method (MSSVDS) by extending MSEIGS to general rectangular matrices.

The second part of the thesis discusses two important machine learning applications that can benefit from the proposed multi-scale approach. Chapter 3 illustrates how our multi-scale view can be adapted to the problem of link prediction in social networks. We propose a novel multi-scale link prediction framework (MSLP) that captures and combines both local and global information to improve predictions in a scalable manner. The three main phases of our method are presented in different sections: hierarchical clustering, subspace approximation and multi-scale prediction. Extensive experimental results on large scale real-world datasets are given to support our claims.

Lastly, we discuss recommender systems with information beyond the traditional user-item matrix in Chapters 4 and 5. We study the two main categories of information available in addition to the user-item matrix, contextual information and side-information of users and items, each in real-life applications: mobile application (app) recommendation and blog recommendation. In Chapter 4, we address the problem of predicting the next app that a user will use, given his or her recent interaction information. We show how users can be represented by their app usage pattern at different scales from individidual to global level, which is utilized to develop a novel collaborative filtering method that incorporates dynamic interaction information. In Chapter 5, we explore the problem of blog recommendation in one of the most popular microblogging services. We present a novel boosted inductive matrix completion (BIMC) method, which is an additive low-rank model for user-blog preferences consisting of two components; one component captures the low-rank structure of follow relationships and the other captures the latent structure using side-information. In both recommendation scenarios, the proposed methods are shown to greatly enhance recommendation quality on real-world datasets.

1.2 Contributions

This thesis makes the following conceptual and concrete contributions:

- We look at machine learning problems with graphs as the primary data source and show that our multi-scale approach based on the divide-and-conquer paradigm can be applied to achieve superior performance and scalability.

- We propose a novel multi-scale spectral decomposition method (MSEIGS) that is capable of computing large number of eigenvalues and eigenvectors of the entire graph. MSEIGS outperforms other widely used solvers in terms of convergence speed and approximation quality. Furthermore, our method is naturally parallelizable and exhibits significant speedups in shared-memory parallel settings.

- We extend MSEIGS to the more general case of computing the singular value decomposition. The proposed multi-scale singular value decomposition method (MSSVDS) naturally shares the advantages of MSEIGS and outperforms other popular solvers.

- We propose a flexible multi-scale link prediction framework (MSLP) for scalable link prediction. Our method makes predictions by combining information from multiple scales of the network, resulting in a more accurate and robust model than other state-of-the-art methods.

- We propose new algorithms for real-world recommender systems where additional information sources are available. Specifically, we utilize contextual information and side-information of users and items for app recommendation and blog recommendation, respectively. The proposed methods adapt our multi-scale view and effectively exploit such additional information in a scalable manner yielding significant improvements in recommendation performance.

1.3 Related Work

Here we highlight previous work closely related to the research in this thesis.

Multi-scale Spectral Decomposition: The spectral decomposition of large and sparse graphs is a fundamental tool that lies at the core of numerous algorithms in varied machine learning tasks. Practical examples include spectral clustering [81], link prediction in social networks [94], recommender systems with side-information [77, 92], densest k-subgraph problem [83] and graph matching [85]. Most of the existing eigensolvers for sparse matrices employ the single-vector version of iterative algorithms, such as the power method and Lanczos algorithm [58]. The Lanczos algorithm iteratively constructs the basis of the Krylov subspace to obtain the eigendecomposition, which has been extensively investigated and applied in popular eigensolvers, e.g., eigs in Matlab (ARPACK) [61] and PROPACK [59]. However, it is well known that single-vector iterative algorithms can only compute the leading eigenvalue/eigenvector (e.g., power method) or have difficulty in computing multiplicities/clusters of eigenvalues (e.g., Lanczos) [84]. In contrast, the block version of iterative algorithms using multiple starting vectors, such as the randomized SVD [36] and block Lanczos [75], can avoid such problems and utilize efficient matrix-matrix operations (e.g., Level 3 BLAS) with better caching behavior.

While these are the most commonly used methods to compute the spectral decomposition of a sparse matrix, they do not scale well to large problems, especially when hundreds of eigenvalues/eigenvectors are needed. Furthermore, none of them consider the clustering structure of the sparse graph. One exception is the classical divide and conquer algorithm by [22], which partitions the tridiagonal eigenvalue problem into several smaller problems that are solved separately. Then it combines the solutions of these smaller problems and uses rank-one modification to solve the original problem. However, this method can only be used for tridiagonal matrices and it is unclear how to extend it to general sparse matrices.

Multi-scale Link Prediction: Link prediction refers to the problem of inferring new interactions among members in a network. The first systematic treatment of the problem appeared in [67], where a variety of proximity measures, such as Common Neighbors [79] and Katz [52] were used as effective methods for link prediction. In addition to unsupervised approaches, there is also rising interest in supervised approaches for link prediction [39, 68]. In supervised link prediction, node and/or edge features are extracted from the network and link prediction is treated as a classification problem. However, engineering good features and handling the class imbalance problem are still challenging tasks.

Many popular proximity measures that are used for link prediction have high computational complexity and do not scale well to large-scale networks. A great deal of recent work has been devoted to speed up the computation. For example, [102] truncates the series expansion of Katz and only considers paths of limited length. In [97, 28], dimensionality reduction methods, such as the eigen-decomposition, are used to construct low-rank approximations of a graph, which are then used to compute approximated proximity measures. The more recent work in [13] applies the Lanczos/Stieltjes procedure to iteratively compute upper and lower bounds of a single Katz value and shows that these eventually converge to the real Katz value.

Very little work has been done using hierarchical structures for link prediction. One exception is the method proposed by [19], which works by sampling a number of competitive hierarchical random graphs from a large pool of such graphs. Each sampled graph is associated with a probability indicating the strength of community structure over the original network. The probability of a link appearing between any two nodes is averaged over the corresponding connecting probability on the sampled graphs. However, to predict potential links, this algorithm needs to enumerate and average over almost all possible hierarchical partitions of a given network and thus is very costly to compute even for small networks. Compared with [19], our algorithm is much more efficient in terms of speed and thus can be scaled up to large-scale link prediction problems with millions of users.

Collaborative Filtering with Interactional Context: Previous research has shown that contextual information can enhance the performance of recommender systems in various applications. Based on the classification of context introduced in [27], context can be distinguished into two types: *representational* and *interactional*. The majority of context-aware recommender systems have investigated the use of representational context, such as time, location and weather [8]. Some context-aware neighborhood-based models [17, 43] incorporate contextual information in the similarity measure between users. Another approach described in [10] introduces item splitting, where item ratings are split into two virtual items based on a given contextual condition. The virtual items are used instead of the original ones in different collaborative filtering algorithms and a rating is predicted for the virtual item corresponding to the current user's context. The context-aware latent factor model in [9] includes attribute-based context variables as biases to appropriately learn the model parameters. In the tensor factorization method proposed in [49], additional dimension for contextual information is added to the standard user-item ratings matrix.

There has been limited work for recommender systems that incorporate interactional context. Hariri, et. al [38] also consider the problem of recommending the next track given a sequence of tracks recently played. Each sequence of tracks in a hand-compiled playlist database is first encoded as a sequence of latent topics (using topic modeling). Frequent patterns of topics are discovered from the topic sequences using a pattern-mining algorithm. These sequential patterns are then used to predict relevant topics given a user session. The predicted topics are used to post-filter an initial ranking produced by a traditional recommendation algorithm. The method is applicable only when the number of topics is small or the maximum length of a session is short. Enumerating all possible sequences can be computationally infeasible otherwise. Moreover, recommendations are not personalized as sequential patterns are mined from the entire population. In the domain of mobile application, recent work on predicting app launch [105] uses both representational (location, time) and interactional (app launch) context information to engineer features. However, the approach in [105] uses only the first app in a given session, which is considered as the trigger. In contrast, our approach considers all apps launched in the current session.

Collaborative Filtering with Side Information: In general, various sources of information additional to the traditional user-item matrix can boost recommendation performance. Recommender systems with side information is by no means new and numerous methods have been proposed based on the type of side information they utilize, such as user generated content [55, 112, 37, 76, 64, 5], user/item profile or attribute [3, 14], social network [47, 70] and context information [78]. A recent comprehensive survey of the state-of-the-art methods can be found in [91].

One of the main approaches that extend MC with side information is the Collective Matrix Completion (CMC) model [96, 14], where the goal is to jointly recover a collection of matrices with shared low-rank structure. In [112], the user-item matrix and the user-user similarity matrix based on tags information are jointly factorized to facilitate better recommendations. Recent work on CMC provides consistency guarantees under certain assumptions [33], which can be restrictive due to imposing a common structure. Recommender systems with social networks are mostly based on the latent factor model with additional constraints in the objective such as latent factors being similar between connected users [47, 70]. Another approach is the regression-based latent factor model proposed by [3], where attribute information is integrated into the model. However, the proposed method does not scale well to large datasets. Graph-based methods have also been extended to incorporate side information. For example, [55] constructs a multi-partite graph with social and tag information, which does not scale well with additional side-information or when features are represented as a dense matrix. Lastly, user generated content, such as reviews and comments, have been exploited by analyzing sentiment information [76, 64]. In most cases, methods are either specialized for a particular source of information or do not scale well with large number of features and lack theoretical guarantees.

Closely related to Tumblr blog recommendation is the Who-To-Follow system in Twitter [34]. Previous approaches for followee recommendation include a probabilistic model based on probabilistic latent semantic analysis proposed by [54]. In [111], a community-based approach is proposed, where matrix factorization is applied independently to each of the discovered communities. However, both methods do not consider any other explicit user/blog features. In [37], follower/followee as well as content (tweets) information is used to represent users in a similarity-based collaborative filtering method. Similarly, [5] first identifies a list of candidate followees, which are the 2-hop neighbors in the follower graph, and then refines the list using content-based profiles of users. Graph-based methods that use proximity measures between nodes have also been applied to followee recommendation [108]. One major drawback is that these methods can not efficiently deal with the inductive setting. Furthermore, none of the existing methods consider images nor user activity information, which is also available in Twitter.

There has been limited work on employing deep learning methods for recommender systems. One exception is the music recommendation method proposed by [82]. In [82], the traditional matrix factorization is combined with a deep convolutional neural network to learn a function that maps music content features to corresponding latent factors. Another exception is the work by [30], where a recurrent neural network is trained to capture semantics of text documents that is used in a content-based recommender system. Both studies have shown deep learning as a promising approach for recommender systems.

Chapter 2

Spectral Decomposition of Massive Graphs

Spectral decomposition of large-scale graphs is one of the most informative and fundamental matrix approximations. Specifically, we are interested in the case where the top-k eigenvalues and eigenvectors are needed, where k is in the hundreds. This computation is needed in various machine learning applications such as semi-supervised classification, link prediction and recommender systems. The data for these applications is typically given as sparse graphs containing information about dyadic relationship between entities, e.g., friendship between pairs of users. Supporting the current big data trend, the scale of these graphs is massive and continues to grow rapidly. Moreover, they are also very sparse and often exhibit clustering structure, which should be exploited. However, popular solvers, such as subspace iteration, randomized SVD [36] and the classical Lanczos algorithm [58], are often too slow for very big graphs.

A key insight is that the graph often exhibits a clustering structure and the union of all cluster's subspaces turns out to have significant overlap with the dominant subspace of the original matrix, which is shown both

The materials presented in this chapter have been published in [95]. All the co-authors contributed equally in the related publication.

theoretically and empirically. Based on this observation, we propose a novel divide-and-conquer approach to compute the spectral decomposition of large and sparse matrices, called MSEIGS, which exploits the clustering structure of the graph and achieves faster convergence than state-of-the-art solvers. In the divide step, MSEIGS employs graph clustering to divide the graph into several clusters that are manageable in size and allow fast computation of the eigendecomposition by standard methods. Then in the conquer step, eigenvectors of the clusters are combined to initialize the eigendecomposition of the entire matrix via block Lanczos. As shown in our analysis and experiments, MSEIGS converges faster than other methods which do not consider the clustering structure of the graph. To speedup the computation, we further divide the subproblems into smaller ones and construct a hierarchical clustering structure; our framework can then be applied recursively as the algorithm moves from lower levels to upper levels in the hierarchy tree. Moreover, our proposed algorithm is naturally parallelizable as the main steps can be carried out independently for each cluster. On the SDWeb dataset with more than 82 million nodes and 3.6 billion edges, MSEIGS takes only about 2.7 hours on a single-core machine while Matlab's `eigs` function takes about 4.2 hours and randomized SVD takes more than 6 hours. Using 16 cores, we can cut this time to less than 40 minutes showing that our algorithm obtains good speedups in shared-memory settings. We further extend MSEIGS to computing the more general Singular Value Decomposition (SVD) of a sparse matrix and propose a multi-scale singular value decomposition (MSSVDS) method, which naturally inherits the advantages of MSEIGS.

While our proposed algorithm is capable of computing highly accurate eigenpairs, it can also obtain a much faster approximate eigendecomposition with modest precision by prematurely terminating the algorithm at a certain level in the hierarchy tree. This early termination strategy is particularly useful as it is sufficient in many applications to use an approximate eigendecomposition. We apply MSEIGS and its early termination strategy to two real-world machine learning applications: label propagation for semi-supervised classification and inductive matrix completion for recommender systems. We show that both our methods are much faster than other methods while still attaining good performance. For example, to perform semi-supervised learning using label propagation on the Aloi dataset with 1,000 classes, MSEIGS takes around 800 seconds to obtain an accuracy of 60.03%; MSEIGS with early termination takes less than 200 seconds achieving an accuracy of 58.98%, which is more than 10 times faster than a conjugate gradient based semi-supervised method [48].

2.1 Spectral Decomposition

Suppose we are given a graph $G = (\mathcal{V}, \mathcal{E}, A)$, which consists of n vertices and $|\mathcal{E}|$ edges such that an edge between any two vertices i and j represents their similarity w_{ij}. The corresponding adjacency matrix A is a $|\mathcal{V}| \times |\mathcal{V}|$ sparse matrix with (i, j) entry equal to w_{ij} if there is an edge between i and j and 0 otherwise. We consider the case where G is an undirected graph, i.e., A is symmetric. The goal is to efficiently compute the top-k eigenvalues $\lambda_1, \cdots, \lambda_k$ ($|\lambda_1| \geq \cdots \geq |\lambda_k|$) and their corresponding eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \cdots \mathbf{u}_k$ of A, which form the best rank-k approximation of A. That is,

$$A \approx U_k \Lambda_k U_k^T,$$

where Λ_k is a $k \times k$ diagonal matrix with the k largest eigenvalues of A and $U_k = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k]$ is an $n \times k$ orthonormal matrix.

The block Lanczos algorithm is one of the most successful and widely used methods for computing the dominant k eigenvalues and eigenvectors of a symmetric matrix A [75, 84]. It can be viewed as a generalization of the classical single-vector Lanczos method. The basic idea of block Lanczos is to use an $n \times b$ initial matrix V_0 to recursively construct the *Krylov subspace* of A. After j steps, the algorithm generates an orthonormal basis $\hat{Q}_j = \left[Q_1, Q_2, \cdots, Q_j\right]$ of the j-th Krylov subspace of A on V_0: $K_j(A, V_0) = \text{span}\{V_0, AV_0, A^2V_0, \cdots, A^{j-1}V_0\}$. Simultaneously with the iteration, a sequence of block tridiagonal matrices $\hat{T}_j$ is obtained, each of which is an orthonormal projection of A onto $K_j(A, V_0)$:

$$\hat{T}_j = \hat{Q}_j^T A \hat{Q}_j = \begin{bmatrix} D_1 & B_1^T & \cdots & 0 \\ B_1 & D_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & B_j^T \\ 0 & \cdots & B_j & D_{j+1} \end{bmatrix}.$$

Subsequently, the *Rayleigh-Ritz* procedure is applied by using the extreme eigenpairs $(\hat{\lambda}_i, \hat{\mathbf{u}}_i)$ of $\hat{T}_j$ to obtain the Ritz values $\hat{\lambda}_i$ and Ritz vectors $\hat{Q}_j\hat{\mathbf{u}}_i$ as the approximate eigenpairs $(\bar{\lambda}_i, \bar{\mathbf{u}}_i)$ of A.

The main procedure of block Lanczos is listed in Algorithm 1, which can be expressed in the following three-term recurrence:

$$Q_{j+1}B_j = AQ_j - Q_jD_j - Q_{j-1}B_{j-1}^T.$$

Typically, convergence is checked by examing the residual matrix R as $A\hat{Q}_j - \hat{Q}_j\hat{T}_j = RE_j$, where $E_j = [0, 0, \cdots, I_b]$ (I_b is the identity matrix of size b). Note that $\hat{Q}_j$ forms an invariant subspace of the range space of A when $R = 0$. If the residuals $\|A\bar{\mathbf{u}}_i - \bar{\lambda}_i\bar{\mathbf{u}}_i\|$ are small enough, we stop the procedure and output $(\bar{\lambda}_i, \bar{\mathbf{u}}_i)$, $i = 1, \cdots, k$, as the approximate eigenpairs.

Algorithm 1: Block Lanczos

```
Input  : $n \times n$ symmetric sparse matrix $A$, rank $k$ and $n \times b$
         initial matrix $V_0$.
Output: The approximate dominant $k$ eigenpairs $(\bar{\lambda}_i, \bar{\mathbf{u}}_i)$ of $A$ for
         $i = 1, \cdots, k$.
1 Initialize block Lanczos: $B_0 = 0$; $Q_0 = 0$; $Q_1 = V_0$
2 for $j = 1, 2, \cdots$ do
3     $R = AQ_j - Q_{j-1}B_{j-1}^T$        // Let $R$ be orthogonal to $Q_{j-1}$
4     $D_j = Q_j^T R$        // Obtain $D_j$ by projecting $R$ onto $Q_j$
5     $R = R - Q_jD_j$        // Let $R$ be orthogonal to $Q_j$
6     $Q_{j+1}B_j = R$        // QR-factorization of $R$ to obtain $B_j$
       and $Q_{j+1}$
7     Form $\hat{T}_j$ and $\hat{Q}_j$ and compute the top-$k$ eigenpairs $(\hat{\lambda}_i, \hat{\mathbf{u}}_i)$ of
       $\hat{T}_j$ to obtain the Ritz values $\bar{\lambda}_i = \hat{\lambda}_i$ and Ritz vectors
       $\bar{\mathbf{u}}_i = \hat{Q}_j\hat{\mathbf{u}}_i$ of $A$.
8     If the residuals $\|A\bar{\mathbf{u}}_i - \bar{\lambda}_i\bar{\mathbf{u}}_i\|$, $i = 1, \cdots, k$, are sufficiently
       small, then stop and output $(\bar{\lambda}_i, \bar{\mathbf{u}}_i)$ as the approximate
       eigenpairs.
9 end
```

More recently, the randomized SVD proposed by [36] has gained popularity due to its simple implementation, applicability to large-scale datasets, and existence of theoretical approximation error bounds. It is equivalent to subspace iteration, which is the block version of the power method, with a Gaussian random matrix as the initial matrix. In contrast to block Lanczos that uses $K_j(A, V_0)$, randomized SVD (or subspace iteration) constructs a basis for $A^j V_0$ and then restricts A to this subspace to obtain the decomposition discarding information from previous iterations. As a consequence, block Lanczos can achieve better performance than randomized SVD with the same number of iterations.

While these are the most commonly used methods to compute the spectral decomposition of a sparse symmetric matrix, they do not scale well to large problems, especially when hundreds of eigenvalues and eigenvectors are needed. Furthermore, none of them consider the clustering structure of the sparse graph and use random initialization. In this chapter, we propose a novel multi-scale spectral decomposition method (MSEIGS), which embodies the clustering structure of A to achieve faster convergence. We begin first describing the single-level version of MSEIGS.

2.2 Single-level Division

Our proposed multi-scale spectral decomposition algorithm, which can be used as an alternative to Matlab's `eigs` function, is based on the *divide-and-conquer* principle to utilize the clustering structure of the graph. It consists of two main phases: in the divide step, we divide the problem into several smaller subproblems such that each subproblem can be solved efficiently and independently; in the conquer step, we use the solutions from each subproblem as a good initialization for the original problem and achieve faster convergence compared to existing solvers which typically start from random initialization.

2.2.1 Divide Step

We first use clustering to partition the sparse matrix A into c^2 submatrices as

$$A = D + \Delta = \begin{bmatrix} A_{11} & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & A_{cc} \end{bmatrix}, \; D = \begin{bmatrix} A_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A_{cc} \end{bmatrix}, \; \Delta = \begin{bmatrix} 0 & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & 0 \end{bmatrix}, \tag{2.1}$$

where each diagonal block A_{ii} is a $m_i \times m_i$ matrix, D is a block diagonal matrix and Δ is the matrix consisting of all off-diagonal blocks of A. We then compute the dominant r ($r \leq k$) eigenpairs of each diagonal block A_{ii} independently, such that $A_{ii} \approx U_r^{(i)} \Lambda_r^{(i)} (U_r^{(i)})^T$, where $\Lambda_r^{(i)}$ is a $r \times r$ diagonal matrix with the r dominant eigenvalues of A_{ii} and $U_r^{(i)} = [\mathbf{u}_1^{(i)}, \mathbf{u}_2^{(i)}, \cdots, \mathbf{u}_r^{(i)}]$ is an orthonormal matrix with the corresponding eigenvectors.

After obtaining the r dominant eigenpairs of each A_{ii}, we can sort all cr eigenvalues from the c diagonal blocks and select the k largest eigenvalues (in terms of magnitude) and the corresponding eigenvectors. More specifically, suppose that we select the top-k_i eigenpairs of A_{ii} and construct an $m_i \times k_i$ orthonormal matrix $U_{k_i}^{(i)} = [\mathbf{u}_1^{(i)}, \mathbf{u}_2^{(i)}, \cdots, \mathbf{u}_{k_i}^{(i)}]$, then we concatenate all $U_{k_i}^{(i)}$'s and form an $n \times k$ orthonormal matrix Ω as

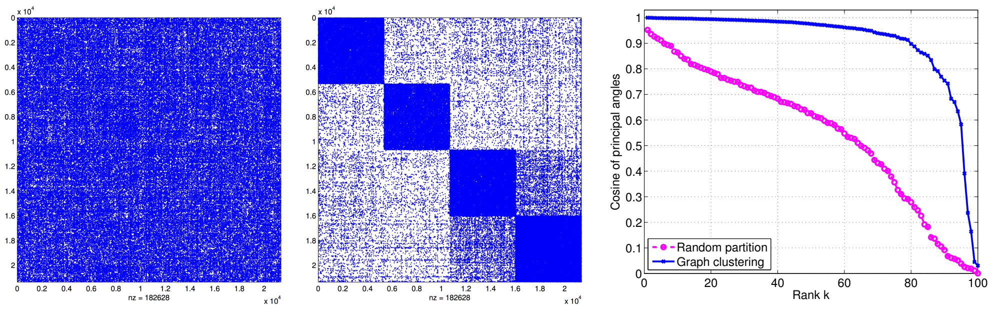
$$\Omega = U_{k_1}^{(1)} \oplus U_{k_2}^{(2)} \oplus \cdots \oplus U_{k_c}^{(c)}, \tag{2.2}$$

where $\sum_i k_i = k$ and $\oplus$ denotes direct sum, which can be viewed as the sum of the subspaces spanned by $U_{k_i}^{(i)}$. Note that Ω is exactly the k dominant eigenvectors of D. After obtaining Ω, we can use it as a starting subspace for the eigendecomposition of A in the conquer step. We next show that if we use graph clustering to generate the partition of A in (2.1), then the space spanned by Ω is close to that of U_k, which makes the conquer step more efficient. We use principal angles [66] to measure the closeness of two subspaces. Since Ω and U_k are orthonormal matrices, the j-th principal angle between subspaces spanned by Ω and U_k is $\theta_j(\Omega, U_k) = \arccos(\sigma_j)$, where σ_j, $j = 1, 2, \cdots, k$, are the singular values of $\Omega^T U_k$ in descending order. In Theorem 2.2.1, we show that $\Theta(\Omega, U_k) = \mathrm{diag}(\theta_1(\Omega, U_k), \cdots, \theta_k(\Omega, U_k))$ is related to the matrix Δ. The proof is given in Appendix 1.1.

Theorem 2.2.1. *Suppose $\lambda_1(D), \cdots, \lambda_n(D)$ (in descending order of magnitude) are the eigenvalues of D. Assume there is an interval $[\alpha, \beta]$ and $\eta \geq 0$ such that $\lambda_{k+1}(D), \cdots, \lambda_n(D)$ lies entirely in $[\alpha, \beta]$ and the k dominant eigenvalues of A, $\lambda_1, \cdots, \lambda_k$, lie entirely outside of $(\alpha - \eta, \beta + \eta)$, then*

$$\|\sin(\Theta(\Omega, U_k))\|_2 \leq \frac{\|\Delta\|_2}{\eta}, \quad \|\sin(\Theta(\Omega, U_k))\|_F \leq \sqrt{k}\frac{\|\Delta\|_F}{\eta}.$$

As we can see, $\Theta(\Omega, U_k)$ is influenced by Δ, thus we need to find a partition such that $\|\Delta\|_F$ is small in order for $\|\sin(\Theta(\Omega, U_k))\|_F$ to be small.

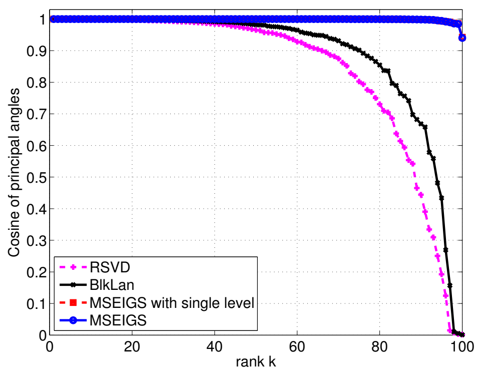


(a) Random partition (b) Graph clustering (c) Cosine of principal angles

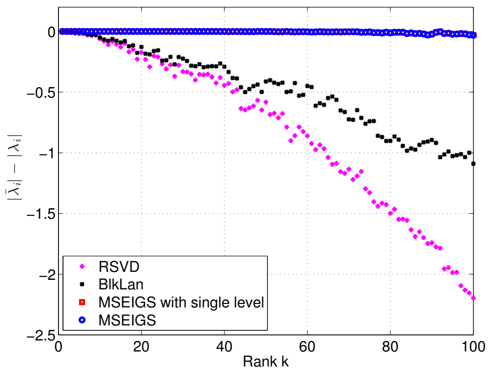
Figure 2.1: Comparison of random partition and graph clustering on the CondMat dataset (collaboration network with 21,362 nodes and 182,628 edges). (a) Spy plot of randomly partitioned graph. (b) Spy plot of graph partitioned into 4 clusters using Metis (nodes are reordered by their cluster indexes). (c) Cosine of principal angles of $\Theta(\Omega, U_k)$ using random partitioning and graph clustering to compute Ω.

Assuming that the graph has clustering structure, we apply graph clustering algorithms to partition A to generate small $\|\Delta\|_F$. In general, the goal of graph clustering is to find clusters such that there are many edges within clusters and only a few between clusters, i.e., make $\|\Delta\|_F$ small. Various graph clustering software can be used to generate the partitions, e.g., Metis [51], Graclus [24], GEM [104] and Nerstrand [60]. Figure 2.1 shows a comparison of the cosine values of $\Theta(\Omega, U_k)$ with different Ω for the CondMat dataset, a collaboration network with 21,362 nodes and 182,628 edges. We compute Ω using random partitioning and graph clustering, where we cluster the graph into 4 clusters using Metis. Spy plots of each case are shown in Figures 2.1(a) and 2.1(b), respectively, illustrating the clustering structure of the CondMat dataset with more than 85% of edges appearing within clusters for graph clustering, whereas there are less than 30% of such edges for random partitioning. As shown in

Figure 2.1(c), more than 80% of principal angles have cosine values that are greater than 0.9 with graph clustering, while this ratio drops to 5% with random partitioning. This illustrates that (1) the effectiveness of graph clustering to reduce $\Theta(\Omega, U_k)$; (2) the subspace spanned by Ω from graph clustering is close to that of U_k.



(a) Cosine of principal angles.



(b) Difference between computed and exact eigenvalues.

Figure 2.2: Comparison of RSVD, BlkLan, MSEIGS with single level and MSEIGS on the CondMat dataset with the same number of iterations (5 steps). (a) shows $\cos(\Theta(\bar{U}_k, U_k))$, where $\bar{U}_k$ consists of the computed top-k eigenvectors and (b) shows the difference between the computed eigenvalues and the exact ones.

2.2.2 Conquer Step

After obtaining Ω from the clusters (diagonal blocks) of A, we use Ω to initialize the spectral decomposition solver for A. In principle, we can use different solvers such as randomized SVD (RSVD) and block Lanczos (BlkLan). In our divide-and-conquer framework, we focus on using block Lanczos due to its superior performance as compared to RSVD as discussed in Section 2.1. In Figure 2.2(a), we compare block Lanczos with RSVD in terms of $\cos(\Theta(\bar{U}_k, U_k))$ for the CondMat dataset, where $\bar{U}_k$ consists of the approximate k dominant eigenvectors. Similarly in Figure 2.2(b), we show that the eigenvalues computed by block Lanczos are more closer to the true eigenvalues. In other words, block Lanczos needs less iterations than RSVD to achieve similar accuracy. For the CondMat dataset, block Lanczos takes 7 iterations to achieve mean of $\cos(\Theta(\bar{U}_k, U_k))$ to be 0.99, while RSVD takes more than 10 iterations to obtain similar performance. It is worth noting that there are a number of improved versions of block Lanczos [32, 7], and we show in the experiments that our method achieves superior performance even with the simple version of block Lanczos.

The single-level version of our proposed MSEIGS algorithm is given in Algorithm 2. Some remarks on Algorithm 2 are in order: (1) $\|A_{ii}\|_F$ is likely to be different among clusters and larger clusters tend to have more influence over the spectrum of the entire matrix. Thus, we select the rank r for each cluster i based on the ratio $\|A_{ii}\|_F / \sum_i \|A_{ii}\|_F$; (2) We use a small number of additional eigenvectors in step 4 (similar to RSVD) to improve the effectiveness of block Lanczos; (3) It is time consuming to test convergence of the Ritz pairs in block Lanczos (steps 7, 8 of Algorithm 1), thus we test convergence after running a few iterations of block Lanczos; (4) Better quality of clustering, i.e., smaller $\|\Delta\|_F$, implies higher accuracy of MSEIGS. From Figures 2.2(a) and 2.2(b), we can observe that the single-level MSEIGS performs much better than block Lanczos and RSVD.

Algorithm 2: MSEIGS with single level

```
Input  : $n \times n$ symmetric sparse matrix $A$, target rank $k$ and
         number of clusters $c$.
Output: The approximate dominant $k$ eigenpairs $(\bar{\lambda}_i, \bar{\mathbf{u}}_i)$ of $A$ for
         $i = 1, \cdots, k$.
1 Generate $c$ clusters $A_{11}, \cdots, A_{cc}$ by performing graph clustering on
    $A$ (e.g., Metis or Graclus).
2 Compute top-$r$ eigenpairs $(\lambda_j^{(i)}, \mathbf{u}_j^{(i)})$ of $A_{ii}$ for $j = 1, \cdots, r$ using
    standard eigensolvers.
3 Select the top-$k$ eigenpairs from the $c$ clusters to generate
    $U_{k_1}^{(1)}, \cdots, U_{k_c}^{(c)}$.
4 Form block diagonal matrix $\Omega = U_{k_1}^{(1)} \oplus \cdots \oplus U_{k_c}^{(c)}$ $(\sum_i k_i = k)$.
5 Apply block Lanczos (Algorithm 1) with initialization $Q_1 = \Omega$.
```

We show that MSEIGS is robust to the quality of clustering by examining performance with varying cluster quality. To vary the clustering quality, we first cluster the CondMat graph into 4 clusters and then randomly perturb clusters by moving a portion of vertices from their original cluster to another random cluster, which reduces the number of within-cluster edges. Table 2.1 presents the performance of MSEIGS with different percentages of vertices shuffled. We can see that (1) the quality of clustering influences the performance of MSEIGS, i.e., better quality of clustering implies higher accuracy of MSEIGS; (2) even with poor clustering structure, MSEIGS can still obtain reasonably good approximations.

Table 2.1: Performance of MSEIGS with varying cluster quality.

Percent of vertices shuffled	0%	20%	40%	60%	80%	100%
Percent of within-cluster edges	86.31%	64.57%	47.08%	35.43%	27.42%	24.92%
Avg. cosine of principal angles	0.9980	0.9757	0.9668	0.9475	0.9375	0.9268

We can now analyze the approximation quality of Algorithm 2 by first examining the difference between the eigenvalues computed by Algorithm 2 and the exact eigenvalues of A.

Theorem 2.2.2. *Let $\bar{\lambda}_1 \geq \cdots \geq \bar{\lambda}_{kq}$ be the approximate eigenvalues obtained after q steps of block Lanczos. According to Kaniel-Paige Convergence Theory [87], we have*

$$\lambda_i \leq \bar{\lambda}_i \leq \lambda_i + \frac{(\lambda_1 - \lambda_i)\tan^2(\theta)}{T_{q-1}^2(\frac{1+\nu_i}{1-\nu_i})}.$$

Using Theorem 2.2.1, we further have

$$\lambda_i \leq \bar{\lambda}_i \leq \lambda_i + \frac{(\lambda_1 - \lambda_i)\|\Delta\|_2^2}{T_{q-1}^2(\frac{1+\nu_i}{1-\nu_i})(\eta^2 - \|\Delta\|_2^2)},$$

where $T_m(x)$ is the m-th Chebyshev polynomial of the first kind, θ is the largest principal angle of $\Theta(\Omega, U_k)$ and $\nu_i = \frac{\lambda_i - \lambda_{k+1}}{\lambda_i - \lambda_1}$.

Next we show the bound of Algorithm 1 in terms of rank-k approximation error.

Theorem 2.2.3. *Given a $n \times n$ symmetric matrix A, suppose by Algorithm 2, we can approximate its k dominant eigenpairs and form a rank-k approximation, i.e., $A \approx \bar{U}_k \bar{\Lambda}_k \bar{U}_k^T$ with $\bar{U}_k = [\bar{\boldsymbol{u}}_1, \cdots, \bar{\boldsymbol{u}}_k]$ and $\bar{\Lambda}_k = diag(\bar{\lambda}_1, \cdots, \bar{\lambda}_k)$. The approximation error can be bounded as*

$$\|A - \bar{U}_k \bar{\Lambda}_k \bar{U}_k^T\|_2 \leq 2\|A - A_k\|_2 \left(1 + \frac{\sin^2(\theta)}{1 - \sin^2(\theta)}\right)^{\frac{1}{2(q+1)}},$$

where q is the number of iterations for block Lanczos and A_k is the best rank-k approximation of A. Using Theorem 2.2.1, we further have

$$\|A - \bar{U}_k \bar{\Lambda}_k \bar{U}_k^T\|_2 \leq 2\|A - A_k\|_2 \left(\frac{\|\Delta\|_2^2}{\eta^2 - \|\Delta\|_2^2}\right)^{\frac{1}{2(q+1)}}.$$

The proof is given in Appendix 1.2. The above two theorems show that a good initialization is important for block Lanczos. Using Algorithm 2, we will expect a small $\|\Delta\|^2$ and θ (as shown in Figure 2.1(c)), and thus our algorithm can have faster convergence compared to block Lanczos with random initialization, because it embodies the clustering structure of A and constructs a good initialization. The time complexity for Algorithm 2 is $O(|\mathcal{E}|k + nk^2)$.

2.3 Multi-level Division: Multi-scale Spectral Decomposition

In this section, we describe our multi-scale spectral decomposition algorithm (MSEIGS). One challenge for Algorithm 2 is the trade-off in choosing the number of clusters c. If c is large, although computing the top-r eigenpairs of A_{ii} can be very efficient, it is likely to increase $\|\Delta\|$, which in turn will result in slower convergence of Algorithm 2. In contrast, larger clusters will emerge when c is small, increasing the time to compute the top-r eigendecomposition for each A_{ii}. However, $\|\Delta\|$ is likely to decrease in this case, resulting in faster convergence of Algorithm 2. To address this issue, we can further partition A_{ii} into c smaller clusters and construct a hierarchy until each cluster is small enough to be solved efficiently. After obtaining this hierarchical clustering, we can recursively apply Algorithm 2 as it moves from lower levels to upper levels in the hierarchy tree.

By constructing a hierarchy, we can choose a small c in order to obtain Ω with small $\Theta(\Omega, U_k)$ (we choose $c = 4$ in the experiments). Our MSEIGS algorithm with multiple levels is described in Algorithm 3. Figures 2.2(a) and 2.2(b) show a comparison between MSEIGS and MSEIGS with a single level. For the single level case, we use the top-r eigenpairs of the c child clusters computed up to machine precision. We can see that MSEIGS performs similarly well compared to the single level case showing the effectiveness of our multi-scale approach. To build the hierarchy, we can choose either top-down or bottom-up approaches using existing clustering algorithms. The overhead of clustering is very low, usually less than 10% of the total time. For example, MSEIGS takes 1,825 seconds, where clustering takes only 80 seconds, for the FriendsterSub dataset (in Table 2.2) with 10M nodes and 83M edges.

Algorithm 3: Multi-scale spectral decomposition (MSEIGS)

```
Input  : $n \times n$ symmetric sparse matrix $A$, target rank $k$, the
         number of levels $\ell$ of the hierarchy tree and the number
         of clusters $c$ at each node.
Output: The approximate dominant $k$ eigenpairs $(\lambda_i, \mathbf{u}_i)$ of $A$ for
         $i = 1, \cdots, k$.
1 Perform hierarchical clustering on $A$ (e.g., top-down or
   bottom-up).
2 Compute the top-$r$ eigenpairs of each leaf node, i.e., $A_{ii}^{(\ell)}$ for
   $i = 1, \cdots, c^{\ell}$, using block Lanczos.
3 for $i = \ell - 1, \cdots, 1$ do
4     for $j = 1, \cdots, c^i$ do
5         Form block diagonal matrix $\Omega_j^{(i)}$ by (2.2).
6         Compute the eigendecomposition of $A_{jj}^{(i)}$ using Algorithm 2
           with $\Omega_j^{(i)}$ as the initial block.
7     end
8 end
```

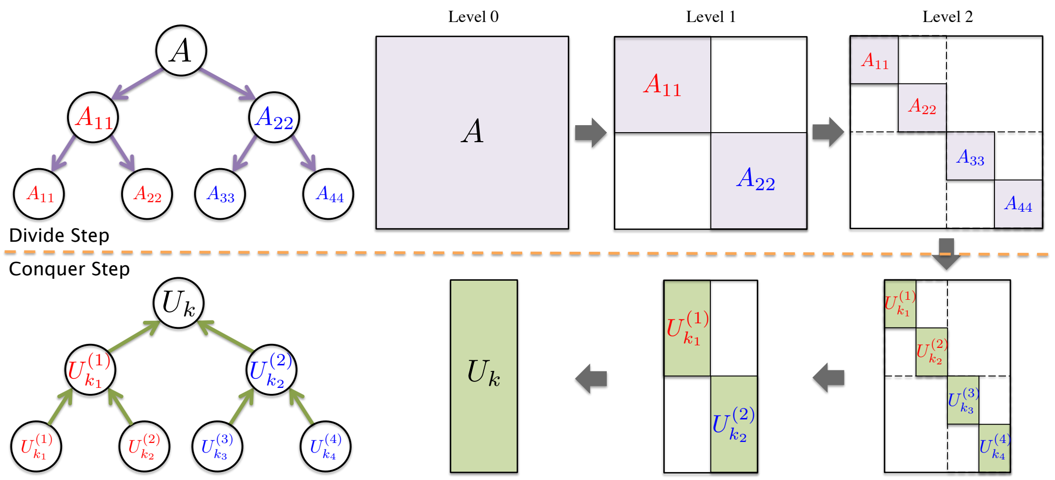


Figure 2.3: Illustration of MSEIGS framework.

2.3.1 Early Termination Strategy

Computing the exact spectral decomposition of A can be quite time consuming. Furthermore, highly accurate eigenvalues and eigenvectors are not essential for many applications. Thus we propose a fast early termination strategy (MSEIGS-Early) to approximate the eigenpairs of A by terminating MSEIGS at a certain level of the hierarchy tree. Suppose that we terminate MSEIGS at the ℓ-th level with c_ℓ clusters. From the top-r eigenpairs of each cluster, we can select the top-k eigenvalues and the corresponding eigenvectors from all c_ℓ clusters as an approximate eigendecomposition of A. As shown in Sections 2.4.2 and 2.4.3, we can significantly reduce the computation time while attaining comparable performance using the early termination strategy for two applications: label propagation and inductive matrix completion.

2.3.2 Multi-core Parallelization

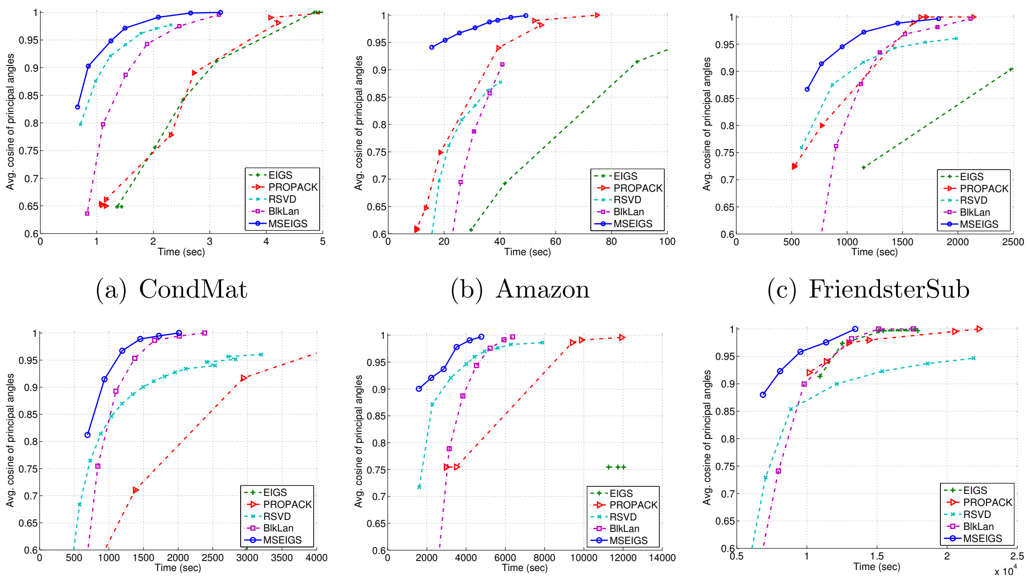
An important advantage of MSEIGS is that it can be easily parallelized, which is essential for large-scale eigendecomposition. There are two main aspects of parallelism in MSEIGS: (1) The eigendecomposition of clusters in the same level of the hierarchy tree can be computed independently; (2) Block Lanczos mainly involves matrix-matrix operations (Level 3 BLAS), thus efficient parallel linear algebra libraries (e.g., Intel MKL) can be used. We show in Section 2.4 that MSEIGS can achieve significant speedup in shared-memory multi-core settings.

2.4 Experiments

In this section we empirically demonstrate the benefits of our proposed method, MSEIGS. We compare MSEIGS with other popular eigensolvers including Matlab's `eigs` function (EIGS) [61], PROPACK [59], randomized SVD (RSVD) [36] and block Lanczos with random initialization (BlkLan) [84] on three different tasks: approximating the eigendecomposition, label propagation and inductive matrix completion. All experiments are conducted on computing nodes that have two Intel Xeon E5-2680 (v2) CPUs with either 256 GB or 1 TB of main memory. Our algorithms are implemented in C++ with OpenMP and all methods use Intel Math Kernel Library (MKL) as the underlying BLAS/LAPACK library.

Table 2.2: Datasets of increasing sizes.

dataset	CondMat	Amazon	RoadCA	LiveJournal	FriendsterSub	SDWeb
# of nodes	21,263	334,843	1,965,206	3,997,962	10.00M	82.29M
# of nonzeros	182,628	1,851,744	5,533,214	69,362,378	83.67M	3.68B
rank k	100	100	200	500	100	50



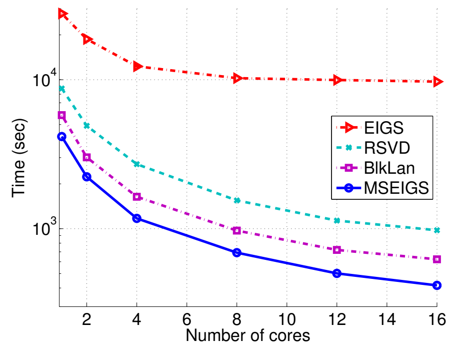
(a) CondMat (b) Amazon (c) FriendsterSub

(d) RoadCA (e) LiveJournal (f) SDWeb

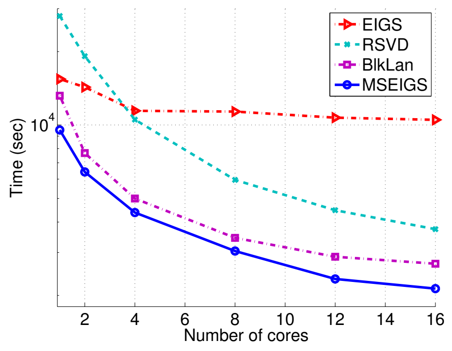
Figure 2.4: The k dominant eigenvectors approximation results showing time vs. average of the cosine of principal angles. For a given time, MSEIGS consistently yields better results than other methods.

2.4.1 Approximation Results

First, we show in Figure 2.4 the performance of MSEIGS for approximating the top-k eigenvectors for different types of real-world graphs including web graphs, social networks and road networks [107, 72]. Summary of the datasets is given in Table 2.2, where the largest graph contains more than 3.6 billion edges. We use the average of the cosine of principal angles $\cos(\Theta(\bar{U}_k, U_k))$ as the evaluation metric, where $\bar{U}_k$ consists of the computed



(a) LiveJournal



(b) SDWeb

Figure 2.5: Shared-memory multi-core results showing number of cores vs. time to compute similar approximation. MSEIGS achieves almost linear speedup and outperforms other methods.

top-k eigenvectors and U_k represents the "true" top-k eigenvectors computed up to machine precision using Matlab's eigs function. Larger values of the average $\cos(\Theta(\bar{U}_k, U_k))$ imply smaller principal angles between the subspace spanned by U_k and that of $\bar{U}_k$, i.e., better approximation. As shown in Figure 2.4, with the same amount of time, the eigenvectors computed by MSEIGS consistently yield better principal angles than other methods.

Since MSEIGS divides the problem into independent subproblems, it is naturally parallelizable. In Figure 2.5, we compare MSEIGS with other methods under the shared-memory multi-core setting on the LiveJournal and SDWeb datasets. We vary the number of cores from 1 to 16 and show the time to compute similar approximation of the eigenpairs. As shown in Figure 2.5, MSEIGS achieves almost linear speedup and outperforms other methods. For example, MSEIGS is the fastest method achieving a speedup of 10 using 16 cores for the LiveJournal dataset.

2.4.2 Label Propagation for Semi-supervised and Multi-label Learning

One application for MSEIGS is to speed up the label propagation algorithm, which is widely used for graph-based semi-supervised learning [113] and multi-label learning [101]. The basic idea for label propagation is to propagate the known labels over an affinity graph (represented as a weighted matrix W) constructed using both labeled and unlabeled examples. Mathematically, at the $(t+1)$-th iteration, $F(t+1) = \alpha SF(t) + (1-\alpha)Y$, where S is the normalized affinity matrix of W; Y is the $n \times l$ initial label matrix; F is the predicted label matrix; l is the number of labels; n is the total number of samples; $0 \leq \alpha < 1$. The optimal solution is $F^* = (1-\alpha)(I - \alpha S)^{-1}Y$. There are two standard approaches to approximate F^*: one is to iterate over $F(t)$ until convergence (truncated method); another is to solve F^* as a system of linear equations by using an iterative solver like conjugate gradient (CG). However, both methods suffer from slow convergence, especially when the number of labels, i.e., columns of Y, grows dramatically. As an alternative, we can apply MSEIGS to generate the top-k eigendecomposition of S such that $S \approx \bar{U}_k \bar{\Lambda}_k \bar{U}_k^T$ and approximate F^* as $F^* \approx \bar{F} = (1-\alpha)\bar{U}_k(I - \alpha\bar{\Lambda}_k)^{-1}\bar{U}_k^T Y$. Obviously, $\bar{F}$ is robust to large numbers of labels.

In Table 2.4, we compare MSEIGS and MSEIGS-Early with other methods for label propagation on two public datasets: Aloi for semi-supervised learning and Delicious for multi-label learning. In the experiment, we use the RBF kernel $W_{ij} = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$ to measure the similarity between samples i and j. Details of the datasets and parameters, which are chosen by cross-validation, are given in Table 2.3. As we can see in Table 2.4, MSEIGS and MSEIGS-Early significantly outperform other methods. To achieve similar accuracy, MSEIGS takes much less time. More interestingly, MSEIGS-Early is faster than MSEIGS and almost 10 times faster than other methods with little degradation of accuracy showing the efficiency of our early-termination strategy.

Table 2.3: Summary of datasets used for label propagation.

Dataset	# of training points	# of test points	# of classes/labels	dimension	γ	α
Delicious	12,920	3,185	983	500	10^{-1}	0.99
Aloi	10,000	98,000	1,000	128	10^{-7}	0.99

Table 2.4: The label propagation results on two real datasets including Aloi for semi-supervised classification and Delicious for multi-label learning. The graph is constructed using [69], which takes 87.9 seconds on Aloi and 16.1 seconds on Delicious.

Method	Aloi ($k = 1500$)		Delicious ($k = 1000$)		
	time(secs)	acc(%)	time(secs)	top3-acc(%)	top1-acc(%)
Truncated	1824.8	59.87	3385.1	45.12	48.89
CG	2921.6	60.01	1094.9	44.93	48.73
EIGS	3890.9	60.08	458.2	45.11	48.51
RSVD	964.1	59.62	359.8	44.11	46.91
BlkLan	1272.2	59.96	395.6	43.52	45.53
MSEIGS	767.1	60.03	235.6	44.84	49.23
MSEIGS-Early	176.2	58.98	61.36	44.71	48.22

2.4.3 Inductive Matrix Completion for Recommender Systems

In the context of recommender systems, Inductive Matrix Completion (IMC) [45] is another important application where MSEIGS can be applied. IMC incorporates side-information of users/items given in the form of feature vectors for matrix factorization, which has been shown to be effective for the gene-disease association problem [77]. Given a user-item ratings matrix $R \in \mathbb{R}^{m\times n}$, where R_{ij} is the known rating of item j by user i, IMC is formulated as follows:

$$\min_{W\in\mathbb{R}^{f_c\times r}, H\in\mathbb{R}^{f_d\times r}} \sum_{(i,j)\in\Omega} (R_{ij} - \mathbf{x}_i^T W H^T \mathbf{y}_j)^2 + \frac{\lambda}{2}(\|W\|_F^2 + \|H\|_F^2),$$

where Ω is the set of observed entries; λ is a regularization parameter; $\mathbf{x}_i \in \mathbb{R}^{f_c}$ and $\mathbf{y}_j \in \mathbb{R}^{f_d}$ are feature vectors for user i and item j, respectively. We evaluated MSEIGS combined with IMC for recommendation tasks where a social network among users is also available. It has been shown that exploiting these social networks improves the quality of recommendations [47, 100]. One way to obtain useful and robust features from the social network is to consider the k principal components, i.e., top-k eigenvectors, of the corresponding adjacency matrix A. We compare the recommendation performance of IMC using eigenvectors computed by MSEIGS, MSEIGS-Early and EIGS. We also report results for two baseline methods: standard matrix completion (MC) without user/item features and Katz[1] on the combined network $C = [A\ R; R^T\ 0]$ as in [100].

[1]The Katz measure is defined as $\sum_{i=1}^{t} \beta^t C^t$, and we set $\beta = 0.01$ and $t = 10$.

Table 2.5: Summary of datasets used for inductive matrix completion. Note that r is the rank of W and H in IMC and we set the regularization parameter $\lambda = 0.1$, which are chosen by cross-validation.

Dataset	# of users	# of items	# of ratings in R	# of links in A	r
Flixster	1.0M	48.8K	8.2M	11.8M	100
Amazon	334.8K	73.2K	2.7M	1.9M	200
LiveJournal	4.0M	2.0K	2.4M	69.4M	100

Table 2.6: Recall-at-20 (RCL@20) and top-k eigendecomposition time (eig-time, in seconds) results on three real-world datasets: Flixster, Amazon and LiveJournal. MSEIGS and MSEIGS-Early require much less time to compute the top-k eigenvectors (latent features) for IMC while achieving similar performance compared to other methods. Note that Katz and MC do not use eigenvectors.

Method	Flixster ($k = 100$)		Amazon ($k = 500$)		LiveJournal ($k = 500$)	
	eig-time	RCL@20	eig-time	RCL@20	eig-time	RCL@20
Katz	-	0.1119	-	0.3224	-	0.2838
MC	-	0.0820	-	0.4497	-	0.2699
EIGS	120.51	0.1472	871.30	0.4999	12099.57	0.4259
RSVD	85.31	0.1491	369.82	0.4875	7617.98	0.4294
BlkLan	104.95	0.1465	882.58	0.4687	5099.79	0.4248
MSEIGS	36.27	0.1489	264.47	0.4911	2863.55	0.4253
MSEIGS-Early	21.88	0.1481	179.04	0.4644	1545.52	0.4246

We evaluated the recommendation performance on three publicly available datasets shown in Table 2.5 [47, 107]. For the Amazon and LiveJournal datasets, we randomly sampled items (affiliations) with at least 10 users. The Flixster dataset contains user-movie ratings information and the other two datasets are for the user-affiliation recommendation task. We report recall-at-N where $N = 20$ averaged over 5-fold cross-validation, which is a widely used evaluation metric for top-N recommendation tasks [21]. In Table 2.6, we can see that IMC outperforms the two baseline methods: Katz and MC. For IMC, both MSEIGS and MSEIGS-Early achieve comparable results compared to other methods, but require much less time to compute the user latent features (top-k eigenvectors). For the LiveJournal dataset, MSEIGS-Early is almost 8 times faster than EIGS while attaining similar performance as shown in Table 2.6.

2.5 Extension to Singular Value Decomposition

In this section, we show how to extend the multi-scale framework to computing the Singular Value Decomposition (SVD) of large-scale sparce matrices. The SVD is the generalization of the spectral decomposition of a square matrix to any $m \times n$ matrix. Formally, given a matrix A with m rows and n columns, if σ is a nonnegative scalar, and $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$ are nonzero vectors such that

$$A\mathbf{v} = \sigma\mathbf{u} \quad \text{and} \quad A^T\mathbf{u} = \sigma\mathbf{v},$$

then σ is a singular value of A and $\mathbf{u}$ and $\mathbf{v}$ are corresponding left and right singular vectors, respectively. As in the case of MSEIGS, our interest lies in the case where the top-k singular values and singular vectors are needed, where k is in the hundreds. Let Σ_k be a $k \times k$ diagonal matrix with the k largest singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$ and $U_k = \{\mathbf{u}_i\}_{i=1}^k$ and $V_k = \{\mathbf{v}_i\}_{i=1}^k$ be their corresponding orthonormal matrices of left and right singular vectors, respectively. Our goal is to efficiently compute

$$A \approx U_k \Sigma_k V_k^T.$$

2.5.1 Multi-scale Singular Value Decomposition

In general, algorithms for computing singular values are analogs of algorithms for computing eigenvalues of symmetric matrices [99, 20]. The key idea is to find square roots of eigenvalues of A^TA without actually forming A^TA:

$$A^TA = (U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma^2V^T.$$

Note that we assume $m \geq n$ and the algorithm may be applied to A^T when $m < n$. Specifically, the SVD of A is computed as follows:

1. Compute the spectral decomposition $A^TA = V\Lambda V^T$.
2. Let Σ be the nonnegative square root of diagonal matrix Λ.
3. Solve the system $U\Sigma = AV$ for orthogonal U.

For the block Lanczos algorithm, this can be achieved by replacing AQ_j with $A^T(AQ_j)$ as in Algorithm 4 (step 3). That is, we operate on the j-th Krylov subspace of A on V_0:

$$K_j(A, V_0) = \text{span}\{V_0, (A^TA)V_0, (A^TA)^2V_0, \cdots, (A^TA)^{j-1}V_0\}$$

to obtain the decomposition. One caveat of this approach is that it is numerically unstable for small singular values with $\sigma_i \ll \|A\|_2 = \sigma_1$ as we are squaring the condition number. However, this is not a problem in our case as we focus on the dominant singular values. Furthermore, we have seen that a reasonable approximation suffices for important machine learning applications as demonstrated with MSEIGS-Early in Section 2.4. In such case, the number of iterations j is small and the columns of $K_j(A, V_0)$ are unlikely to be dependent, and thus numerical stability is not a big issue.

Algorithm 4: Block Lanczos for SVD

```
Input  : $m \times n$ sparse matrix $A$ ($m \geq n$), rank $k$ and $n \times b$ initial
         matrix $V_0$.
Output: The approximate dominant $k$ singular values, left and
         right singular vectors $(\bar{\sigma}_i, \bar{\mathbf{u}}_i, \bar{\mathbf{v}}_i)$ of $A$ for $i = 1, \cdots, k$.

1  Initialize block Lanczos: $B_0 = 0$; $Q_0 = 0$; $Q_1 = V_0$
2  for $j = 1, 2, \cdots$ do
3      $R = A^T(AQ_j) - Q_{j-1}B_{j-1}^T$        // Let $R$ be orthogonal to
        $Q_{j-1}$
4      $D_j = Q_j^T R$        // Obtain $D_j$ by projecting $R$ onto $Q_j$
5      $R = R - Q_j D_j$                // Let $R$ be orthogonal to $Q_j$
6      $Q_{j+1}B_j = R$      // QR-factorization of $R$ to obtain $B_j$
        and $Q_{j+1}$
7      Form $\hat{T}_j$ and $\hat{Q}_j$ and compute the top-$k$ eigenpairs $(\hat{\lambda}_i, \hat{\mathbf{v}}_i)$ of $\hat{T}_j$
        to obtain the Ritz values $\bar{\lambda}_i = \hat{\lambda}_i$ and Ritz vectors $\bar{\mathbf{v}}_i = \hat{Q}_j \hat{\mathbf{v}}_i$
        of $A$.
8      If the residuals $\|A^T A\bar{\mathbf{v}}_i - \bar{\lambda}_i \bar{\mathbf{v}}_i\|$, $i = 1, \cdots, k$, are sufficiently
        small, then stop and output the approximate singular values
        $\bar{\sigma}_i = \sqrt{\bar{\lambda}_i}$ and right singular vectors $\bar{\mathbf{v}}_i$.
9  end
10 Solve $U\Sigma = AV$ for orthogonal $U$ as the left singular vectors.
    // e.g., via QR factorization
```

An alternative way to compute singular values without squaring the condition number is to consider the $(m + n) \times (m + n)$ matrix

$$C = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}. \tag{2.3}$$

Since $A = U\Sigma V^T$ implies $AV = U\Sigma$ and $A^T U = V\Sigma^T = V\Sigma$,

$$\begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} V & V \\ U & -U \end{bmatrix} = \begin{bmatrix} V & V \\ U & -U \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix},$$

which is equivalent to a spectral decomposition of C and extracting the corresponding components. However, we would need to explicitly form C for MSEIGS due to the divide step (clustering), which incurs unnecessary storage and computation costs as vectors will be of length $m + n$. Thus, we take the first approach of finding the square roots of eigenvalues of $A^T A$ outlined in Algorithm 4.

For the divide step, recall that we want the size of off-diagonal blocks $\|\Delta\|_F$ in Eq (2.1) to be small as possible by Theorem 2.2.1. To achieve this without explicitly computing $A^T A$, we employ the k-means clustering algorithm to cluster the columns of A into c clusters as $A = \begin{bmatrix} A_1 & A_2 & \cdots & A_c \end{bmatrix}$, where A_i denotes the i-th cluster of size $m \times n_i$ ($\sum_{i=1}^{c} n_i = n$). By doing so, we can partition $A^T A$ into c^2 submatrices similar to Eq (2.1) as

$$A^T A = D + \Delta = \begin{bmatrix} A_{11} & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & A_{cc} \end{bmatrix}, \; D = \begin{bmatrix} A_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A_{cc} \end{bmatrix}, \; \Delta = \begin{bmatrix} 0 & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & 0 \end{bmatrix},$$

where each diagonal block $A_{ij} = A_i^T A_j$ is an $n_i \times n_i$ matrix. The choice of distance metric for the k-means algorithm should depend on the application of interest. For the datasets used for evaluation (see Table 2.7), using cosine similarity as the distance metric (i.e., spherical k-means) gives good performance.

Once we obtained the top-k_i eigenvalues and corresponding eigenvectors $V_{k_i}^{(i)}$ for each A_{ii} ($i = 1, 2, \cdots, c$), we form an $n \times k$ orthonormal matrix $\Omega = V_{k_1}^{(1)} \oplus V_{k_2}^{(2)} \oplus \cdots \oplus V_{k_c}^{(c)}$ ($\sum_{i=1}^{c} k_i = k$) for the conquer step. Similar to MSEIGS with single-level (Algorithm 2), we use Ω to initialize a SVD solver for A as shown in Algorithm 5. By extending this single-level MSSVDS to the multi-level setting as described in Section 2.3, we arrive at the proposed multi-scale singular value decomposition (MSSVDS) method.

```
Algorithm 5: MSSVDS with single level
    Input  : $m \times n$ sparse matrix $A$, target rank $k$ and number of
             clusters $c$.
    Output: The approximate dominant $k$ singular values, left and
             right singular vectors $(\bar{\sigma}_i, \bar{\mathbf{u}}_i, \bar{\mathbf{v}}_i)$ of $A$ for $i = 1, \cdots, k$.
  1 Generate $c$ clusters $A_1, \cdots, A_c$ by clustering columns of $A$ (e.g.,
     k-means).
  2 Compute top-$r$ eigenpairs $(\lambda_j^{(i)}, \mathbf{v}_j^{(i)})$ of $A_{ii} = A_i^T A_i$ for $j = 1, \cdots, r$
     using standard eigensolvers.
  3 Select the top-$k$ eigenpairs from the $c$ clusters to generate
     $V_{k_1}^{(1)}, \cdots, V_{k_c}^{(c)}$.
  4 Form block diagonal matrix $\Omega = V_{k_1}^{(1)} \oplus \cdots \oplus V_{k_c}^{(c)}$ $(\sum_i k_i = k)$.
  5 Apply block Lanczos (Algorithm 4) with initialization $Q_1 = \Omega$.
```

2.5.2 Experiments

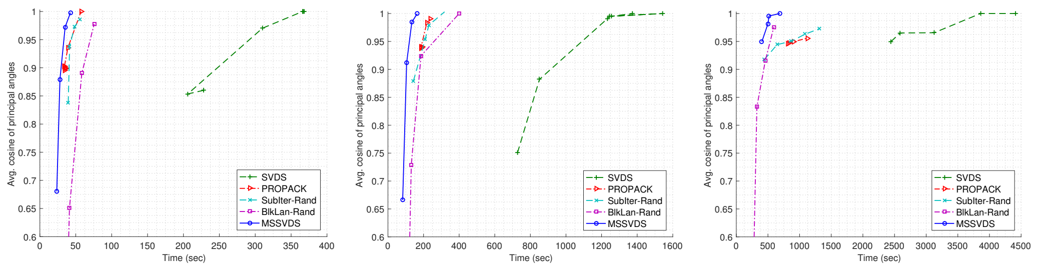
We give experimental results of the MSSVDS for approximating the top-k singular vectors compared to other solvers on real-world datasets including user-item ratings, text document-word counts and web graphs. Summary of the datasets is given in Table 2.7 [65, 71, 72]. We compare MSSVDS with other popular SVD solvers including Matlab's svds function (SVDS) [61], PROPACK [59], randomized SVD (RSVD) [36] and block Lanczos with random initialization (BlkLan) [84]. The average of the cosine of principal angles $\cos(\Theta(\bar{V}_k, V_k))$ is used as the evaluation metric, where $\bar{V}_k$ consists of the computed top-k singular vectors and V_k represents the "true" top-k singular vectors computed up to machine precision using Matlab's svds function. As shown in Figure 2.4, with the same amount of time, the singular vectors computed by MSSVDS consistently yield better principal angles (i.e., larger values of the average $\cos(\Theta(\bar{V}_k, V_k))$) than other compared methods. Matlab's svds performs the worst due to actually forming C in Eq (2.3) and computing its spectral decomposition for stability reasons. Lastly, we note that MSSVDS achieves similar speedups to that of MSEIGS in multi-core settings as shown in Section 2.4.1.

Table 2.7: Datasets of increasing sizes.

dataset	Epinions	RCV1	PLDWeb
# of rows (m)	72,119	677,399	27,370,613
# of columns (n)	755,137	42,735	36,165,034
# of nonzeros	13,430,330	49,556,258	623,056,312
rank k	100	200	20

2.6 Conclusions

In this chapter, we proposed a novel divide-and-conquer based framework, multi-scale spectral decomposition (MSEIGS), for approximating the top-k eigendecomposition of large-scale graphs. Our method exploits the clustering structure of the graph and converges faster than state-of-the-art meth-



(a) Epinions (b) RCV1 (c) PLDWeb

Figure 2.6: Approximation results of the k dominant left singular vectors showing time vs. average of the cosine of principal angles. For a given time, MSSVDS consistently yields better results than other methods.

ods. Moreover, our method can be easily parallelized, which makes it suitable for massive graphs. Empirically, MSEIGS consistently outperforms other popular eigensolvers in terms of convergence speed and approximation quality on real-world graphs with up to billions of edges. We show that MSEIGS is highly effective for two important applications: label propagation and inductive matrix completion. We also proposed multi-scale singular value decomposition (MSSVDS) that extends MSEIGS to computing the SVD of large-scale graphs and showed that it shares the advantages of MSEIGS resulting in better performance than other popular SVD solvers. Dealing with graphs that cannot fit into memory is one of our future research directions. We believe that MSEIGS can also be efficient in streaming and distributed settings with careful implementation.

Chapter 3

Link Prediction in Social Networks

A core problem for social network analysis is proximity estimation that infers the "closeness" of different users. Proximity measures quantify the interaction between users based on the structural properties of a graph such as the number of common friends. An important application of proximity estimation in social networks is link prediction, which is a key problem in social network analysis [67]. The task of link prediction refers to predicting which pairs of users in a social network will be connected in the future. It allows social network services to make better recommendations for potential new friends making it easier for users to expand their social neighborhood and consequently, to increase their activity.

Using proximity estimation for link prediction is based on the assumption that a pair of users with a high proximity score indicates they are close in terms of social relatedness and hence this pair of users will have a good chance to become friends in the future. Simple proximity measures such as neighborhood-based measures, e.g., common neighbors [79] and Adamic-Adar score [1], can be computed efficiently. However, they describe a very localized

The materials presented in this chapter have been published in [94]. Donghyuk Shin formulated the problems, developed the algorithms, and conducted experiments.

view of interaction. There are more comprehensive proximity measures that capture a broader perspective of social relationships by considering all paths between users. These path based methods, such as Katz [52] or rooted PageRank [67], are often more effective. Nonetheless, they are also well known for their high computational complexity and memory usage, which limits their applicability to massive graphs. For example, Facebook, one of the most popular social networking sites, now has more than 1.3 billion active users with 5 new users joining every second on average.[1]

To solve this problem, a great deal of work has been done on scalable proximity estimation [97, 13]. One basic idea is to perform dimensionality reduction on the original graph and then compute the proximity based on its low-rank approximation. Recently, clustered low rank approximation (CLRA) has been proposed, which develops a fast and memory efficient method [89, 98]. However, a single low-rank approximation may not be sufficient to represent the whole network. Furthermore, the approach in [89] uses only a single clustering structure making it sensitive to a particular clustering and biased against links that happen to be between clusters. More notably, a recent study has shown that large social networks tend to lack large well-defined clusters, which suggests that a single clustering structure can be problematic [63].

To address the above problems, we adapt the multi-scale framework from MSEIGS to the problem of link prediction. Recall that the goal of

[1]`http://newsroom.fb.com/company-info/`

MSEIGS is to efficiently compute the eigendecomposition of the entire matrix by utilizing approximations at lower levels as a good initialization, which is discarded afterwards. Instead of discarding such information, we treat each low-rank approximation at different levels as a representation of the entire network from different levels of granularity. As a results, we can obtain multiple granular views of the network based on its hierarchical clustering structure. Although we use a single hierarchical representation of the graph, it is important to note that we do not require it to be the optimal structure. The main purpose of using hierarchical clustering is not to detect the underlying community structure of the graph, but to use it as a tool for efficient multi-scale approximation as in MSEIGS. In the experimental section, we show that under clustering structures of varying quality, our proposed algorithm can still achieve better results compared to other link prediction algorithms (for example, in the Epinions network [86], which was also used in [63]).

Specifically, we propose a robust, flexible, and scalable framework for link prediction on social networks that we call *multi-scale link prediction* (MSLP). Our method exploits different scales of low-rank approximation of social networks by combining information from multiple levels in the hierarchy in an efficient manner. Higher levels in the hierarchy present a more global view, while lower levels focus on more localized information. MSLP works by first performing hierarchical clustering on the graph by utilizing a fast graph clustering algorithm, and then performing multi-scale approximation based on the produced hierarchy. Since different levels have different approximation, each level will give different approximated proximity scores. Afterwards, we combine approximated proximity scores from each level and makes the final prediction based on the combined scores. As a result, MSLP captures both local and global information of the network.

We list the benefits of our framework as follows:

- MSLP makes predictions based on information from multiple scales and thus can make more accurate and robust predictions.

- MSLP is fast and memory-efficient as it uses a simple and fast tree-structured subspace approximation method, which speeds up the computation of our multi-scale approximation while re-using memory across different levels. As a result, it can be applied to social networks with millions of users.

- MSLP is flexible in two aspects: (1) it can be used with any other reasonably good clustering algorithm to generate a multi-scale view of the graph as it does not depend on a particular hierarchical structure, (2) as a dimensionality reduction method, it is not tied down to a particular proximity measure, e.g., Katz and CN, and others can be used.

3.1 Proximity Measures and Link Prediction

Assume we are given a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \cdots, n\}$ is the set of vertices representing the users in a social network and $\mathcal{E} = \{e_{ij} | i, j \in \mathcal{V}\}$

is the set of weighted edges quantifying the connection between user i and user j. Let $A = [a_{ij}]$ be the corresponding $n \times n$ adjacency matrix of G such that $a_{ij} = e_{ij}$, if there is an edge between i and j and 0 otherwise. For simplicity, we assume G is an undirected graph, i.e., A is symmetric.

As shown in [67], various proximity measures can be computed from A. Many of these measures can be represented as a matrix function $f(A)$, where the (i, j)-th element represents the value of a proximity measure between user i and user j [29]. One popular measure is the number of common neighbors, which can be captured by $f_{cn}(A) = A^2$, describing a very localized view of interactions between vertices by considering only paths of length 2. A more extensive measure is the popular Katz measure. Such path-based proximity measures often achieve better accuracy at the cost of higher computational complexity. The Katz measure is defined as follows

$$f_{kz}(A) = \beta A + \beta^2 A^2 + \beta^3 A^3 + \cdots = \sum_{k=1}^{\infty} \beta^k A^k = (I - \beta A)^{-1} - I,$$

where I is the identity matrix and $\beta \leq 1/\|A\|_2$ is a damping parameter that ensures convergence of the series. As we can see, the Katz measure takes $O(n^3)$ time, which is computationally infeasible for large-scale networks with millions of nodes.

Here, dimensionality reduction methods such as the Singular Value Decomposition (SVD) play an important role. These methods are particularly useful, since it suffices to have a reasonably good estimation of a given proximity measure for most applications. Furthermore, low-rank approximation of the adjacency matrix serves as a useful conceptual and computational tool for the graph. Assume that we are given a rank-r approximation of the $n \times n$ matrix A as $A \approx \tilde{A} = USU^T$, where U is an $n \times r$ orthonormal matrix and S is an $r \times r$ matrix. Using this low-rank approximation $\tilde{A}$, the CN measure can be approximated as $f_{cn}(A) \approx US^2U^T$. Similarly, the Katz measure is approximated by

$$f_{kz}(A) \approx \sum_{k=1}^{\infty} \beta^k \tilde{A}^k = U(\sum_{k=1}^{\infty} \beta^k S^k)U^T = U((I - \beta S)^{-1} - I)U^T.$$

In general, $f(A) \approx Uf(S)U^T$, which requires less computational resources as the matrix function is only evaluated on the much smaller S matrix.

Based on the estimated proximity measures, we can perform link prediction on social networks. The problem of link prediction deals with networks that evolve over time. Given a "snapshot" $G_t = (\mathcal{V}_t, \mathcal{E}_t)$ of the network for time t, the task is to predict links that would form at a future time step $t+1$. For simplicity, we assume no links are removed and use $\mathcal{V}_t = \mathcal{V}_{t+1}$. In terms of adjacency matrices, this can be expressed as $A_{t+1} = A_t + \Delta_t$, where A_t is the adjacency matrix corresponding to G_t and Δ_t consists of links formed between time t and $t+1$. Then the link prediction problem is to find nonzero elements in Δ_t given A_t. A high proximity score between two users from A_t captures the high correlation between them and thus a high chance to form a new link in the future, i.e., in Δ_t.

3.2 Proposed Method: Multi-scale Link Prediction

In this section, we present our *multi-scale link prediction* (MSLP) framework for social networks. We adapt the multi-scale framework of MSEIGS presented in Chapter 2 to the problem of link prediction. Our method consists of three main phases: hierarchical clustering, subspace approximation and multi-scale prediction. Specifically, we first construct a hierarchy tree with a fast top-down hierarchical clustering approach. Then, a multi-scale low-rank approximation to the original graph is computed when traversing the hierarchy in a bottom-up fashion. Finally, we combine proximity measures, which are computed using the multi-scale low-rank approximation of the graph, and make our final predictions.

3.2.1 Hierarchical Clustering

The first step of our method is to hierarchically cluster or partition the graph A. The purpose of this is to efficiently generate a multi-scale approximation of the graph using the constructed hierarchical structure. This, in turn, makes predictions more accurate and robust as we combine predictions at each level of the hierarchy in the final step. Generally, there are two main approaches for hierarchical clustering: agglomerative (or bottom-up) approach and divisive (or top-down) approach. The agglomerative approach initially treats each vertex as one cluster and continually merges pairs of clusters as it moves up the hierarchy. The divisive approach takes the opposite direction, that is, all vertices are placed in a single cluster and recursively partitioned

Table 3.1: Percentage of within-cluster edges using Graclus. Numbers in brackets represent random clustering. It can be seen that Graclus is quite effective in finding good clustering structure. (these networks contain about 2 million nodes — details are given in Table 3.3.)

Hierarchy	`Flickr`	`LiveJournal`	`MySpace`
Level 1	96.2 (68.1)	99.3 (60.1)	98.6 (61.6)
Level 2	95.1 (61.7)	98.8 (51.7)	88.0 (35.2)
Level 3	88.1 (54.3)	85.0 (28.6)	69.5 (18.0)
Level 4	85.2 (51.4)	79.4 (15.2)	64.3 (13.0)
Level 5	66.7 (27.2)	70.0 (9.4)	56.3 (8.4)

into smaller clusters. Due to the large scale of the problem and the availability of efficient clustering software, such as Metis [51] or Graclus [24], we employ the divisive approach as we did for MSEIGS in Chapter 2.

As it is desirable to capture most of the links within clusters, we compare with random clustering in terms of the percentage of within-cluster links on three large-scale social networks in Table 3.1. For each level of the hierarchy tree, the within-cluster links are those that connect two vertices in the same cluster. As shown in Table 3.1, the percentage of within-cluster edges of random clustering is much smaller than the hierarchical clustering scheme, and the gap becomes much larger when going down the hierarchy. Even at the deepest level, the clustering scheme we use can still capture more than half of the edges compared with less than 10% in the `LiveJournal` and `MySpace` graphs when using random clustering. We note that the hierarchical clustering scheme is also very fast. Clustering the three networks of Table 3.1 into 5 levels with 2 clusters at each level can be completed in just 5 minutes on a 8-core 3.4GHz machine. In the next section, we show how to use the hierarchy structure to efficiently construct a multi-scale approximation of large-scale graphs.

3.2.2 Subspace Approximation

After constructing the hierarchy for a given graph, we can compute low-rank approximations of A at each level of the hierarchy to obtain a multi-scale approximation. The main idea of our approach is based on MSEIGS in Chapter 2. For notational brevity, we omit the superscripts denoting the level of the hierarchy. The cluster structure of A at level p with c clusters and its low-rank approximation $A \approx \tilde{A} = USU^T$ can be represented as:

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & A_{cc} \end{bmatrix} \approx \tilde{A} = \begin{bmatrix} U_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & U_c \end{bmatrix} \begin{bmatrix} S_{11} & \dots & S_{1c} \\ \vdots & \ddots & \vdots \\ S_{c1} & \dots & S_{cc} \end{bmatrix} \begin{bmatrix} U_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & U_c \end{bmatrix}^T,$$

where the columns of U_i are the set of orthonormal basis vectors forming the subspace for cluster i.

In MSEIGS, each S_{ii} is a diagonal matrix with the r dominant eigenvalues of A_{ii} and $S_{ij} = 0$ for $i \neq j$. However, such approximation does not contain any information about the off-diagonal matrices A_{ij} and would not be able to make predictions for links that appear between clusters. Therefore, we extend such cluster-wise approximation to the entire graph as in clustered low rank approximation (CLRA) [89]. That is, we set $S_{ij} = U_i^T A_{ij} U_j$ for $i \neq j$, which is optimal in the least squares sense. Note that level 0 in the hierarchy can be viewed as treating the entire graph as a single cluster, which yields a global view of the entire matrix A, while lower levels will preserve more local information within each cluster. Thus, each level of approximation concentrates on different levels of granularity, resulting in a multi-scale approximation of A.

An important issue here is how to compute each level's approximation of A efficiently. As shown in Section 2.2, we can use the union of all child cluster's subspace $\Omega = U_1 \oplus U_2 \oplus \cdots \oplus U_c$ as a good initializaiton to efficiently compute the subspace of the parent cluster. For the task of link prediction, we only require quality approximations to obtain a good estimate of the proximity measures. Thus, we employ subspace iteration as in Algorithm 6 to speed up the computation, whereas the block Lanczos algorithm was employed in MSEIGS for higher accuracy. The basic idea of Algorithm 6 is to construct an orthonormal basis Q for $A\Omega$ and then restrict A to this subspace to obtain the approximation. Specifically, we construct the Rayleigh quotient matrix $B = Q^T AQ$ and compute its eigendecomposition to compute the dominant subspace of the parent cluster.

3.2.3 Multi-scale Link Prediction

As mentioned in Section 3.1, many proximity measures $f(A)$ for link prediction are expensive to compute on large-scale networks because of their high complexity. One solution is to approximate A by a low-rank approximation $\tilde{A}$ and then compute approximated proximity measures with $f(\tilde{A})$ to make predictions. This stems from the idea that most of the action in A can be captured by a few latent factors, which can be extracted with low-rank

Algorithm 6: Tree-structured approximation of dominant subspace of parent cluster from child clusters

```
Input: $n \times n$ adjacency matrix of parent cluster $A = A^{(P)}$, child
       cluster's subspaces $U_1^{(C)}, \ldots, U_c^{(C)}$, target rank $r$.
Output: dominant subspace for parent cluster $A^{(P)}$, i.e., $U^{(P)}$.
1 $\Omega \leftarrow U_1^{(C)} \oplus U_2^{(C)} \oplus \cdots \oplus U_c^{(C)}$.
2 Compute $n \times cr$ matrix $Y = A\Omega$.
3 Compute $Q$ as an orthonormal basis for the range of $Y$.
4 Compute $B = Q^T AQ$.                    // $A \approx Q(Q^T AQ)Q^T$
5 Compute rank-$r$ eigen-decomposition of $B \approx V\Lambda V^T$.
6 Compute $U^{(P)} = QV$.
```

approximations of A. It has been shown that CLRA provides an accurate and scalable low-rank approximation, and can be used for efficient proximity estimation [98]. However, CLRA uses only a single clustering structure making it sensitive to a particular clustering and biased against links that appear between clusters.

Our proposed method alleviates such problem with a multi-scale approach. The main idea is that, under a hierarchical clustering, all links will eventually belong to at least one cluster. That is, even if we miss a between-cluster link at a certain level, it still has a good chance of getting corrected by upper levels as it will eventually become a within-cluster link. Moreover, links that lie within clusters at multiple levels, such as from the deepest level, get emphasized multiple times. Those links will have the propensity of being included in the final prediction, which aligns with the intuition that links are more likely to form within tight clusters.

Algorithm 7: Multi-Scale Link Prediction (MSLP)

```
Input: adjacency matrix $A$, number of levels $\ell$, number of clusters
       $c$ at each node, target rank $r$, weights $w_0, w_1, \ldots, w_\ell$.
Output: top-$k$ predictions.
/* Hierarchical clustering */
1  $A^{(0)}_{11} \leftarrow A$.
2  for $i = 0$ to $\ell$ do
3      for $j = 1$ to $c^i$ do
4          Cluster $A^{(i)}_{jj}$ into $c$ clusters.              // e.g. Graclus
5      end
6  end
/* Subspace approximation */
7  Compute $U^{(\ell)}$, $S^{(\ell)}$ using CLRA.           // approximation for
    deepest level
8  for $i = \ell - 1$ to $0$ do
9      Compute $U^{(i)}$ using Algorithm 6.       // approximation for
        intermediate levels
10     $S^{(i)} = {U^{(i)}}^T A U^{(i)}$.
11 end
/* Multi-scale prediction */
12 for $i = \ell$ to $0$ do
13     $K_i = f(A^{(i)}) = U^{(i)} f(S^{(i)}) {U^{(i)}}^T$.            // e.g. Katz
14 end
15 return top-$k$ predictions according to
    $P = w_0 K_0 + w_1 K_1 + \cdots + w_\ell K_\ell$.
```

Once the multi-scale low-rank approximation of A is obtained, we now perform multi-scale link prediction. From each low-rank approximation of the hierarchy, $\tilde{A}^{(i)}$, $i = 0, 1, \ldots, \ell$, the approximated proximity measure can be computed with $f(\tilde{A}^{(i)})$. This gives a total of $\ell + 1$ proximity measures for each link, which are combined to make final predictions. Formally, our multi-scale predictions are given by

$$P = g(w_0 f(\tilde{A}^{(0)}) + w_1 f(\tilde{A}^{(1)}) + \ldots + w_\ell f(\tilde{A}^{(\ell)})),$$

where w_i's are the weights for different levels and $g(\cdot)$ is the predictor (e.g., top-k scoring links). In the experiments, we use uniform weights for all levels, i.e., $w_i = 1/(\ell + 1)$.

The entire flow of our proposed method, Multi-Scale Link Prediction (MSLP), is listed in Algorithm 7 and illustrated in Figure 3.1. Next, we analyze the computation time and memory usage of MSLP.

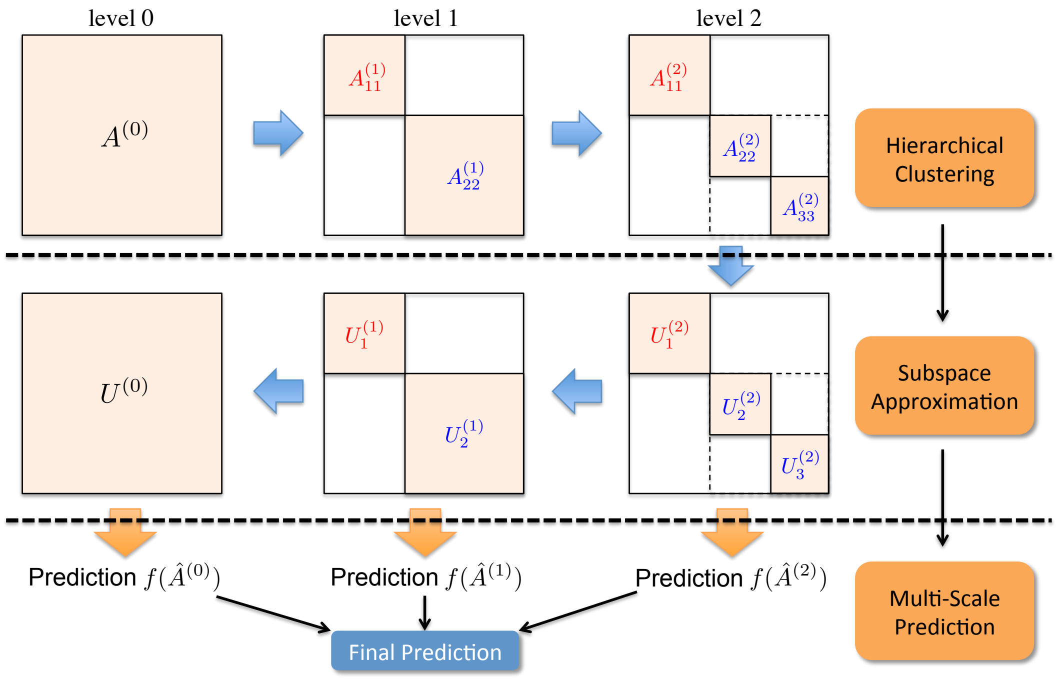


Figure 3.1: Illustration of MSLP framework.

Table 3.2: Computational time (in minutes) for subspace approximation by MSLP (Algorithm 6) and EIG on three large-scale social networks.

Network		`LiveJournal`	`Flickr`	`MySpace`
EIG		157.10	146.28	211.29
MSLP	Level 0	30.74	29.98	38.27
	Level 1	20.18	21.95	35.86
	Level 2	18.93	17.25	29.17
	Level 3	14.01	15.33	26.36
	Level 4	13.74	15.79	26.36
	Level 5	121.26	132.93	188.26
MSLP Total		218.88	233.33	344.02

Computation Time: As mentioned earlier, the hierarchical clustering is fast and linear in the number of edges in the network and can be finished in a few hundred seconds on networks with 2 million nodes. Computing the approximated proximity scores as a final step for a given user is simply a matrix multiplication of low-rank matrices and time complexity is $O(\ell n r^2)$. In general, we set the number of clusters c and the rank in each cluster r to be fairly small. Among the three phases of MSLP, the subspace approximation phase is the dominant part of the computation time. In Table 3.2, we compare the CPU time for subspace approximation by Algorithm 6 and EIG on three large-scale social networks with about 2 million users. We can see in Table 3.2 that for each intermediate level from 4 to 0, the subspace approximation in MSLP is up to 10 times faster than that of EIG, demonstrating the effectiveness of Algorithm 6. Furthermore, since we operate on each cluster independently, MSLP can be easily parallelized to gain greater speedups.

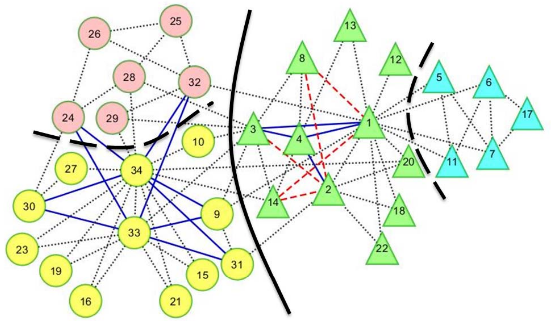
Memory Usage: For a rank-r approximation, EIG needs to store r eigenvectors and eigenvalues which takes $O(nr + r)$ memory. Compared with EIG, CLRA is memory efficient as it only takes $O(nr + c^{2\ell}r^2)$ memory for a larger rank-$c^\ell r$ approximation [89]. MSLP basically has the same memory usage as CLRA. While MSLP achieves a multi-scale approximation, it is not necessary to store the subspaces for all levels simultaneously. We can reuse the memory allocated for the child cluster's subspace to store the parent cluster's subspace using Algorithm 6.

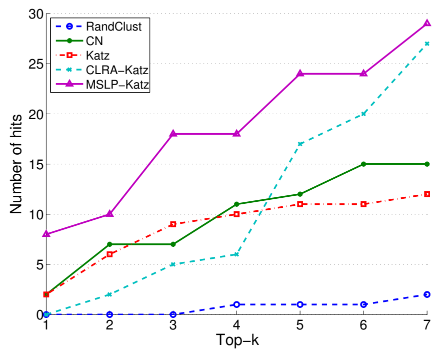
3.3 Experiments

In this section we present experimental results that evaluate both accuracy and scalability of our method, Multi-Scale Link Prediction (MSLP), for link prediction. First we present a detailed analysis of our method using the Karate club network as a case study. This will give a better understanding of our algorithm and illustrate where it succeeds. Next we provide results under different parameter settings on a large social network. Lastly, we compare MSLP to other popular methods on massive real-world social networks with millions of users and demonstrate its superior performance.

3.3.1 Case Study: Karate Club Network

We first start our performance analysis on a well-known small social network, Zachary's Karate club network [110]. The Karate club network represents a friendship network among 34 members of the club with 78 links.


(a) Hierarchy of the Karate club network



(b) Results of top-k hits

Figure 3.2: (a) Hierarchy of the Karate club network with 2 levels, two clusters on the first level and four clusters on the second level. (b) Number of top-k hits for different methods on the Karate club network.

The clustering structure of the Karate club network is a standard example for testing clustering algorithms. We adopt the clustering results from [4], where the clustering is found via modularity optimization. Figure 3.2(a) shows the hierarchy of the Karate club network. The first level has 2 clusters (circle and triangle) with 68 within-cluster links and the second level has 4 clusters (red, yellow, green and blue) with 50 within-cluster links. As the Karate club network is a small network, we apply the *leave-one-out* method to compare different methods. We first remove a single link from the network, treat the held out edge as 0 in A, and perform link prediction on the resulting network. For each leave-one-out experiment, we compute the rank of the removed link based on its proximity measure. If the rank of the removed link appears in the top-k list, we count it as a *hit*. The number of top-k hits is the number of hits out of all leave-one-out experiments.

We compare MSLP to four other methods: RandCluster, common neighbors (CN), Katz and CLRA. In RandCluster, we randomly partition the graph into 4 clusters and compute the Katz measure using CLRA with these clusters. Figure 3.2(b) shows the number of top-k hits for each method. Clearly, our method significantly outperforms other methods by achieving a much higher number of hits. This implies that MSLP makes more accurate predictions by considering the hierarchical structure of the network. RandCluster performs the worst, while CLRA has comparable performance with Katz indicating that the network's property can be captured by a few latent factors. For a better illustration of the advantage of our method, we annotate Figure 3.2(a) with the results of top-3 hits. The solid blue links correspond to hits made by MSLP and the dashed red links are hits made by both CLRA and MSLP, i.e. the set of links successfully predicted by CLRA is a subset of that of MSLP. We can see that all hits made by CLRA are within-cluster links (green cluster), showing that CLRA favors within-cluster links. In contrast, MSLP can predict not only more within-cluster links, but also links between clusters (red and yellow). The ability to correctly predict both within and between-cluster links is one of the main advantages of our multi-scale approach.

3.3.2 Results on Large-scale Datasets

In this section we present the results of link prediction on large real-world datasets. We start by examining how the parameters of MSLP affect performance. Particularly, we investigate how different hierarchical clustering structures impact the performance of MSLP. For this, we use a large real-world network: `Epinions`, which is an online social network from Epinions.com with 32,223 users and 684,026 links [86]. Next we use three real-world massive online social networks with millions of nodes: `Flickr` [74], `LiveJournal` and `MySpace` [97], and compare MSLP to other methods. These datasets have timestamps associated with them and we summarize each snapshot in Table 3.3. The adjacency matrix at the first timestamp, A_{t_1}, is used to compute proximity measures, and the adjacency matrix at the next timestamp, A_{t_2}, is used for testing and evaluation.

Evaluation Mmethodology

We evaluate the accuracy of different methods by computing the *true positive rat*e (TPR) and the *false positive rate* (FPR), defined by

$$\text{TPR} = \frac{\#\text{ of correctly predicted links}}{\#\text{ of actual links}},$$
$$\text{FPR} = \frac{\#\text{ of incorrectly predicted links}}{\#\text{ of non-friend pairs}},$$

Table 3.3: Summary of networks with timestamps.

Network	Date	# of nodes	# of links
`Flickr`	5/6/2007	1,994,422	42,890,114
	5/17/2007	1,994,422	43,681,874
`LiveJournal`	3/4/2009	1,757,326	84,366,676
	4/3/2009	1,757,326	85,666,494
`MySpace`	1/11/2009	2,086,141	90,918,158
	2/14/2009	2,086,141	91,587,516

for all links in a sampled test set. Our evaluation is based on *receiver operating characteristic* (ROC) curve and its *area under the ROC curve* (AUC) that present achievable TPR with respect to FPR. Predicting links with proximity measures involves some thresholding on the measures to produce top-k predictions. The ROC curves captures the full spectrum of prediction performance by varying the decision threshold. However, in a practical sense, a user is recommended only a small number of top-k predictions and the hope is that most of them are correct. Thus, we focus on the region of low FPR by plotting FPR along the x-axis in log-scale, since it reflects the quality of these top-k links. In the same spirit, we also use the *Precision at Top-k*, i.e., the number of correct predictions out of top-k recommendations, as our evaluation metric.

For the `Flickr`, `LiveJournal` and `MySpace` datasets, we randomly select 5,000 users and evaluate on these users as the networks are very large. Performance measures are averaged over 30 iterations of such sampling. As pointed out in [62], most of all newly formed links in social networks close a path of length two and form a triangle, i.e., appear in a user's 2-hop neighborhood. All three datasets show that this is the case for at least 90% of test links in the second timestamp. For similar reasons as in [6], we focus on predicting links to users that are within its 2-hop neighborhood.

Other Methods for Comparison: We have carefully chosen a variety of proximity measures to compare with: Preferential Attachment (PA), Adamic-Adar score (AA), Random Walk with Restarts (RWR), common neighbors

(CN) and Katz [67]. The actual values of Katz quickly becomes difficult to compute as scale increases due to its high computational cost. Therefore, we employ the Lanczos method [13] for its speed and good approximation of the real Katz values. We also consider a supervised machine learning method (LR) [6, 39]. For the latter, we extracted five network-based features: paths of lengths 3, 4 and 5, CN, and AA. Using these features, a logistic regression model is trained over a sampled set of positive and negative links from 10,000 users as in [6].

Varying Hierarchical Clustering Structure

We experiment with various hierarchical clustering structures by varying the number of levels in the hierarchy ℓ and the number of clusters at each node in the hierarchy c. We fix $r = 20$ while changing one parameter at a time and measure AUC and precision at top-20. The `Epinions` network does not have time information, thus we randomly sample a number of links and treat them as test links in A_{t_2}. The sampling is performed such that about 90% of test links appear in a user's 2-hop neighborhood. We compare the performance of our method to two other low-rank approximation methods: eigendecomposition (EIG) and clustered low rank approximation at the deepest level in the hierarchy tree (CLRA).

Table 3.4 shows how the performance changes as the hierarchical clustering structure changes. For a complete comparison, results of other methods are also given in Table 3.4. The second column in Tables 3.4(a) and 3.4(b)

Table 3.4: Varying hierarchical clustering structure by changing (a) the number of clusters per node at each level and (b) the number of levels on `Epinions` dataset. Results show that MSLP is not only more robust than CLRA to different clustering structures, but also outperforms other methods in most cases. Percentage is the percentage of within-cluster edges (Numbers in brackets represent percentage of within-cluster edges of random clustering).

(a) Changing the number of clusters per node at each level.

c	Percentage	CLRA-Katz		MSLP-Katz	
		AUC	Prec	AUC	Prec
2	53.41 (13.42)	0.7928	4.93	**0.8550**	**5.62**
3	41.91 (6.41)	0.7649	4.32	0.8520	5.48
4	37.33 (5.47)	0.7426	3.80	0.8463	5.27
5	34.03 (3.12)	0.7293	3.74	0.8276	4.80

(b) Changing the number of levels.

ℓ	Percentage	CLRA-Katz		MSLP-Katz	
		AUC	Prec	AUC	Prec
2	67.54 (26.51)	0.7970	5.02	0.8459	5.33
3	53.31 (13.42)	0.7928	4.93	**0.8550**	**5.62**
4	47.77 (8.59)	0.7825	4.60	0.8508	5.55
5	43.54 (5.39)	0.7633	4.15	0.8498	5.37

(c) Results of other methods.

Method	AUC	Prec
PA(Preferential Attachment)	0.7717	2.09
AA(Adamic-Adar)	0.8378	5.16
RWR(Random Walk /w Restarts)	0.8468	2.68
LR(Logistic Regression)	0.8227	4.60
CN(Common Neighbors)	0.8163	4.78
Katz(Katz)	0.8352	4.77

represents the percentage of within-cluster edges. It is clear that as the number of clusters at the bottom level increases the percentage decreases. While the accuracy of CLRA degrades as the percentage decreases, MSLP is still able to perform better than other methods in all cases with the only exception of Table 3.4(a) at $c = 5$. The results clearly show that MSLP is robust to different hierarchical structures.

Results on Large-scale Social Networks

In this section, we present results on real-world networks with millions of nodes presented in Table 3.3. We construct a hierarchical structure with $\ell = 5$ and $c = 2$ for all three networks, and use $r = 100$ for EIG, CLRA and MSLP. We set $\beta = 0.0005$ for the Katz measure, which yields the best results.

Table 3.5 gives AUC and precision at top-100 results for the various methods, respectively. MSLP-Katz gives a significant improvement over the Katz measure and outperforms all other methods. Specifically, it gains a relative improvement of up to 4% in AUC and 15% in precision over the next best performing method. We emphasize the superior performance in terms of precision at top-100 of MSLP-Katz as shown in Table 3.5. This is a very appealing aspect of MSLP as it reflects the quality of top recommendations. In contrast, MSLP-CN remains comparable to CN, but performs better than EIG and CLRA. Surprisingly, the supervised method LR does not perform well, which is consistent with results found in [6]. Note that we only use network-based features and no additional features for training. However, engineering for more features is a difficult task and constructing good features itself can be computationally expensive.

Figure 3.3 gives ROC curves focused on the low FPR region for the three large-scale networks. We note that only one representative method from methods that have similar performance is plotted for the sake of clarity. We

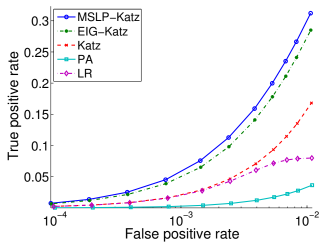
Table 3.5: Precision at top-100 and AUC results for `Flickr`, `LiveJournal` and `MySpace` datasets.

Method	`Flickr`		`LiveJournal`		`MySpace`	
	PRC@100	AUC	PRC@100	AUC	PRC@100	AUC
PA	1.02	0.6981	1.32	0.6075	4.57	0.8325
AA	7.29	0.8758	5.93	0.7709	7.44	0.8767
RWR	5.49	0.7872	3.46	0.7113	1.30	0.8357
LR	2.54	0.7115	2.23	0.7055	4.95	0.7487
CN	7.08	0.8649	5.94	0.7630	7.18	0.8801
EIG-CN	6.88	0.8583	5.34	0.7317	6.99	0.8732
CLRA-CN	6.91	0.8506	5.21	0.7390	6.88	0.8587
MSLP-CN	7.03	0.8621	5.59	0.7538	7.05	0.8743
Katz	7.17	0.8429	5.86	0.7651	6.18	0.8492
EIG-Katz	11.26	0.8887	5.62	0.7547	7.55	0.8638
CLRA-Katz	12.13	0.8611	6.11	0.7531	7.64	0.8646
MSLP-Katz	**13.34**	**0.8924**	**6.72**	**0.7890**	**8.38**	**0.8850**

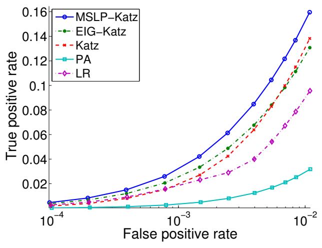
observe that MSLP-Katz performs the best in all three datasets with significant improvements over Katz. For a given TPR, MSLP reduces FPR by 10% on average and at most 20% compared to others in all datasets.

While dimensionality reduction methods, such as EIG and CLRA, tend to perform well in all three datasets, they are limited to a single low-rank representation of the network. Furthermore, CLRA has the largest drop in relative performance in terms of precision compared to MSLP in the `MySpace` dataset, where only 56% of the edges are within clusters, whereas MSLP achieves the best result. Overall, the superior performance of MSLP illustrates the effectiveness of our multi-scale approach.

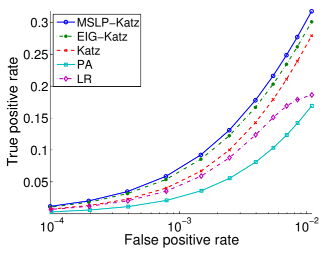
We note that the majority of time is taken by computing CLRA at the deepest level and thereafter low-rank approximations of upper levels can be obtained efficiently due to Algorithm 6. However, CLRA can be easily



(a) `Flickr`



(b) `LiveJournal`



(c) `MySpace`

Figure 3.3: ROC curve on low FPR region of different methods for `Flickr`, `LiveJournal` and `MySpace` datasets. MSLP performs the best on all three datasets.

parallelized as computing the subspace of each cluster is independent to other clusters. Thus, MSLP can achieve much more speedup by using a parallel implementation and serve as a highly scalable method for link prediction.

3.4 Conclusions

In this chapter, we have presented a general framework for multi-scale link prediction by combining predictions from multiple scales using hierarchical clustering. A novel tree-structured approximation method is proposed to achieve fast and scalable multi-scale approximations. Extensive experimental results on large real-world datasets have been presented to demonstrate the effectiveness of our method. This significantly widens the accessibility of state-of-the-art proximity measures for large-scale applications. For future work, we plan to investigate methods to learn the weights for various levels of the hierarchy, since some levels may have better predictions and deserve larger weights in the final prediction. In this work, we use a balanced hierarchical structure mainly for its simplicity in combining predictions. However, a more realistic setting would be to use an unbalanced hierarchical clustering structure. The issue here is how to combine predictions from different levels as some links may not receive predictions at certain levels. We also plan to develop a parallelized version of MSLP as each level of the hierarchy can be easily parallelized.

Chapter 4

Collaborative Filtering with Interactional Context

Recommender systems have become a vital tool for alleviating information overload for online users, due to their ability to model user preferences or interests of items at scale and to predict new associations between them in a personalized manner. For example, recommender systems are now ubiquitous in online shopping and media consumption supporting customers to find the most relevant items or contents (e.g., Netflix, Amazon, Twitter). The most prominent and successful technique for recommender systems is collaborative filtering (CF), which is based on the underlying assumption that users who have expressed similar interests are likely to share common interests in the future [31]. Throughout the past decade, remarkable progress has been made both theoretically and empirically in CF-based recommender systems [2, 56, 21, 15, 46], whose primary data source is a sparse user-item matrix representing user preferences for items. However, many recommendation settings have surfaced where various sources of information are accessible additional to the user-item matrix in recent years. In such scenarios, CF can be extended

The materials presented in this chapter have been published in [78]. All the co-authors contributed equally in the related publication.

to incorporate additional information in a way that alleviates existing issues (e.g., data sparsity, cold-start) and enhances recommendation quality. There are two main types of additional information: (1) interaction information associated with the interplay of users and items; (2) rich side-information of users and items as illustrated in Figure 4.1 [91]. We explore how the proposed multi-scale framework can be adapted to significantly improve recommendation performance in both cases. In this chapter, we focus on the first type of additional information, i.e., interaction information, and discuss the second type in Chapter 5.

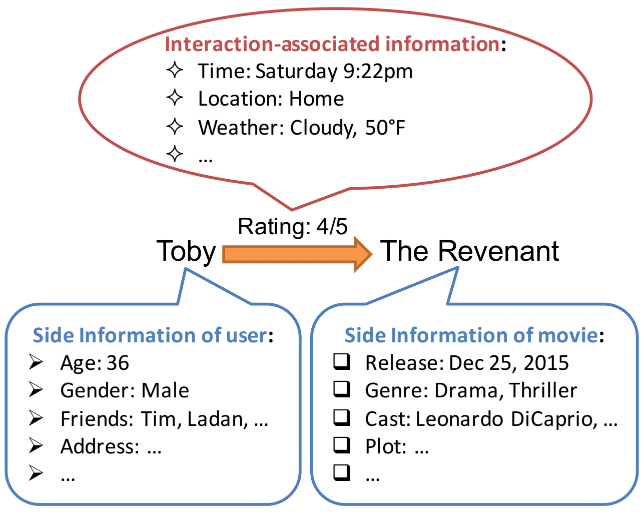


Figure 4.1: Information sources available in addition to the user-item matrix.

The interaction-associated or contextual information includes information sources that are directly related to the event of a user interacting (e.g., rating, clicking) with an item. The most common sources are timestamps and locations, which record the time and place at which a user interacted with an item. Based on the classification presented in [27], such contextual information is called *representational* context, which is usually defined *before* the user interacts and provided as attributes. That is, representational context often influences how the user interacts with the system. In contrast, the *interactional* context is defined dynamically and arises *from* user interaction, which is less explored in context-aware recommender systems. It is applicable in general settings where items are used repeatedly, such as listening to tracks from online music streaming services or browsing products in shopping websites. We propose a novel method for collaborative filtering with interactional context for mobile application recommendation and show that utilizing multiple scales of user behavior information enhances recommendation quality.

The problem of recommending an item to a user given a sequence of items that the user *recently* interacted with arises in many systems. The problem is best understood by considering a smart phone user navigating through various applications (apps) on the mobile device. Mobile apps are often used in conjunction with other relevant apps. For example, if a user launches the 'contacts' app, the next app is likely to be the 'mail' or the 'messages' app. The current context of the user can be characterized from recently launched apps. Recommending the "right" app to use next based on the context of the user's actions would improve user experience and multitasking efficiency.

Based on the classification presented in [27], we call the context that arises from user's activity within a session *interactional* context. In contrast, most of the existing context-aware recommender systems focus on *representational-*

tational context, usually defined *before* the user interacts and provided as attributes such as location, weather and interests [8].

The problem differs from traditional collaborative filtering (CF) settings, such as the Netflix rating prediction problem [11], in many respects. First, user interaction with items such as apps is *brief* and *repetitive* in nature, whereas items like movies are usually watched/rated once. Second, the user feedback is inherently implicit in the form of item clicks, as opposed to explicit feedback like ratings or comments. Additionally, we have a temporal ordering of clicks within user sessions. Third, recommendations must be made available *dynamically* as the user interacts with the system. That is, recommendations should be updated each time a user clicks an item. Finally, we have the notion of interactional context defined by the session in progress, to which the recommendations are targeted. It is desirable for a recommender system to use the context set by the user to update its recommendations. It might be tempting to relate the aspect to that of online CF systems [25], where systems could use newly available ratings to recompute predictions. However, there is a subtle difference. The interactional context should help better zero in on the apps that the user would launch next while she interacts with the system. It is therefore necessary to treat the current session differently from past sessions.

The problem is applicable in any setting where items are generally used repeatedly, such as listening to tracks from online music services or browsing products in shopping websites. If a user has been listening to rock music, she is more likely to prefer other rock music than tracks from a different genre. Note that in such general settings, the system should be able to recommend *new* or *unseen* items to the user. In the next app prediction setting, one could restrict the recommended apps to those that are already installed in the user device, or even recommend new apps that other users have used in similar context.[1] A less obvious but tangible benefit of predicting the next app is that the predicted apps can be pre-loaded to reduce both app launch latency and energy consumption [105].

In this section, we propose a novel method, iConRank, for collaborative filtering with interactional context. The fundamental observation is that we do not want to treat the sequence of item clicks as raw counts, but as ordered transitions. We model users' transition behavior between items as a Markov chain, where transition probabilities are empirically estimated. A single global Markov model would fall short of capturing diverse transition patterns. On the other hand, a fully personalized Markov model would suffer from extreme sparsity of observed transitions. So, we seek cluster-level Markov models, where the clusters themselves are *behavioral.* That is, we cluster users by their sparse one-step item transition probabilities and compute a representative Markov model per behavioral cluster. We develop our method by introducing a *context bias* to a classical neighborhood-based CF model. Our formulation essentially leads to a personalized PageRank [40] on a particular

[1]This choice is made when the recommender system is deployed. In experiments, we evaluate with the latter option.

Markov graph, where the so-called "personalization" vector is derived from interactional context.

Our contributions are as follows:

- The problem of incorporating interactional context in collaborative filtering is relatively unexplored (see Section 1.3). Though the setting is motivated from app launch patterns of smart phone users, it is applicable in many click-based interactive systems.

- We propose iConRank that makes personalized and dynamic recommendations given the current session. Recommendations are updated as the user interacts with the system. We show that the quality of recommendations made by our algorithm is superior to those of competitive methods on two real-life datasets.

- Behavioral clustering of users allows the system to make recommendations using past item transitions of a given user as well as transitions from users with similar navigational patterns. That is, we utilize app usage patterns at multiple scales from individual user level to cluster and global level.

- Our method is scalable and can handle large problems efficiently. The clustering stage of our algorithm is done offline. Personalized PageRank can be computed in a scalable manner, as detailed in Section 4.2.4, which enables implementation on devices with limited processing power.

4.1 Problem Setting

Our problem setting is motivated by smart phone users who exhibit patterns of interaction with the device through mobile apps. The sequence of apps launched by a user defines the *interactional* context of the user's actions. Interactional context arises from the user's activity within a session and is dynamic. The setting is applicable to any click-based recommender system, where recommendations are updated as the users *click* on an item (Youtube, Spotify, etc).

We refer to the sequence of items accessed by a user over a certain contiguous period as a *session*. In practice, sessions are defined based on the type of activity (e.g. articles read by an online user when she is signed in). Note that we do not consider any temporal aspects of the session, other than the ordering of clicks. Existing context-aware recommender systems focus on attribute-based representational context that is less dynamic and often fixed *before* the start of a session. A given user may have specified a set of preferences globally, but it is often the case that user preferences change between sessions. The goal of this chapter is to present recommendations to a given user based on her past sessions, sessions of other users in the system *and* the current session *in progress*. The key aspects of our problem setting are:

1. There are no explicit "likes" or "dislikes" of an item, unlike the case of Netflix ratings. We want to come up with a ranking of items that the user will click next in the current session. Following the recommender

systems literature, our setting relies on *implicit feedback* as against the Netflix prize setting that uses *explicit feedback*.

2. Users may be interested in multiple categories of items, but given the sequence of clicks made in the current session, there is an added *context bias* that needs to be accounted for. In general recommender systems, user bias and item bias are accounted for whereas context bias is either ignored or is not applicable.

We would like to emphasize here that though the first aspect in isolation is well-known in the recommender systems community [44, 106], the second aspect has received little attention [38].

The Problem Statement: The item recommendation problem in the *Collaborative Filtering with Interactional Context* setting is formally stated as follows. Given a history of sessions $\mathcal{S}$ of users $\mathcal{U} = \{u_1, u_2, \ldots, u_{|\mathcal{U}|}\}$ over a set of items $\mathcal{I} = \{a_1, a_2, \ldots, a_{|\mathcal{I}|}\}$, and a specific user $u \in \mathcal{U}$ with session *in progress* $s = \langle a_{i_1}, a_{i_2}, \ldots, a_{i_t} \rangle$, for some $t \geq 1$, we want to recommend the best candidate item $a_{i_{t+1}} \in \mathcal{I}$. Note that we want the recommendations to be (a) *personalized* to the user u, and (b) relevant to the context of the session s. The classical collaborative filtering systems (with implicit feedback) consider a specific case of the problem where the current session is ignored and the goal is to come up with a set of recommendations based on the click history.

4.2 Proposed Method: iConRank

In our problem setting, we work with implicit feedback in the form of click sequences. A widely-used collaborative filtering approach for implicit feedback data is to simply form a user-item count matrix, where an entry represents the number of times (or a monotonic function such as log of the count) a user has clicked on an item in the past. Any of the collaborative filtering approaches for explicit feedback such as matrix factorization or similarity-based methods can then be applied on the count matrix. However, such a naive approach is not appropriate for reasons manyfold. Most importantly, we do not want to predict any exact rating — we just need a ranking of relevant items. It is inherently hard to gauge user preferences with clicks — lack of an established scale like star ratings makes it tricky to compute similarities between users or items. For a detailed discussion of what prevents a direct use of algorithms designed for explicit feedback, see [44]. We will see later in the experiments (Section 4.3) that collaborative filtering methods on the count matrix perform poorly.

Some latent factor models for ratings data include biases due to attribute-based context variables such as location to appropriately learn the model parameters [9]. Other than the absence of explicit feedback, such context-aware models pose another immediate challenge — they rely on rating instances for different settings of context variables to learn the appropriate biases. In the case of interactional context, it is not obvious how to succinctly define context variables and obtain associated training examples. Also, it must be efficient to *update* model parameters as the current session progresses in order to provide dynamic recommendations.

In this section, we first describe how we model history in the form of click sequences. Then, we derive our PageRank-based method by incorporating interactional context in an existing collaborative filtering framework. Finally, we present our algorithm iConRank.

4.2.1 Modeling Implicit Feedback

The fundamental observation is that we do not want to treat the session history as counts but as sequences instead. A simple and effective way to model sequences is to use Markov models. Ideally, we would want to know the probability of user clicking on an item given the current session. To this end, we model the users as Markov. The set of items $\mathcal{I}$ corresponds to the state space of the Markov model, and the state transition probability M_{ij} is the probability that item j is clicked immediately after item i. From the sessions data, we can estimate M_{ij} as the fraction of times item j appears immediately after item i, whenever i appears.

Typically, a given user does not have enough training data to estimate $|\mathcal{I}| \times |\mathcal{I}|$ parameters of the Markov model. On the other hand, we do not need to determine a personalized Markov model for each user. The basic idea of collaborative filtering is to combine preferences from "similar" users in order to make recommendations for a given user. One naive way to combine preferences in our setting is to use session data of *all* users to determine a single *global* Markov model. While the training data may be rich enough to estimate a global model, it is less likely to be a good characterization of the diverse transition behavior of users. The aforementioned fully personalized and fully global models fall short, and our approach is to use cluster-level Markov models. It is reasonable to suppose that there are different clusters of users exhibiting a common navigational pattern. To discover behavioral clusters using session data, we need to find a good representation of users, where users who have similar transition patterns are "close" to each other than those that are not.

4.2.2 Behavioral Clustering

First, we note that the user-item count matrix itself cannot be used for clustering — we want the clusters to indicate *how* users click and not *what* users click. In particular, we want a feature map Φ_u that encodes the fraction of times a particular transition was made, rather than the number of transitions. Let $\Phi : \mathcal{U} \to \mathcal{M}$, where $\mathcal{M}$ denotes the set of row-stochastic matrices, i.e. $\mathcal{M} = \{M \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{I}|} : M_{ij} \geq 0, \sum_j M_{ij} = 1\}$. In particular, $\Phi(u) = M^{(u)}$ where $M^{(u)}$ denotes the one-step Markov transition probability matrix estimated from the session history of user u. Now, we need a distance measure to cluster users in the space of transition probability matrices. An appropriate measure of distance between two probability distributions is the KL-divergence or the relative entropy. The KL-divergence $d_{KL}(x, y)$ between two p-dimensional probability distributions x and y is defined as:

$$d_{KL}(x,y) = \sum_{i=1}^{p} x_i \log_2 \left(\frac{x_i}{y_i} \right).$$

The distance between two users u and v is in turn defined as:

$$d(u,v) = \frac{1}{|\mathcal{I}|} \sum_{i=1}^{|\mathcal{I}|} d_{KL}(M_{i\cdot}^{(u)}, M_{i\cdot}^{(v)}). \qquad (4.1)$$

We have $d(u,v) \geq 0$ since $d_{KL}(.,.) \geq 0$ and $d(u,v) = 0 \iff M^{(u)} = M^{(v)}$.[2] For the actual clustering step, we optimize the k-means objective. The centroid of the cluster π_k is computed as

$$M_k = \frac{1}{|\pi_k|} \sum_{u \in \pi_k} M^{(u)}, \qquad (4.2)$$

where $|\pi_k|$ denotes the number of users assigned to cluster k. Note that $M_k \in \mathcal{M}$ and we use M_k as the Markov model for the kth cluster.

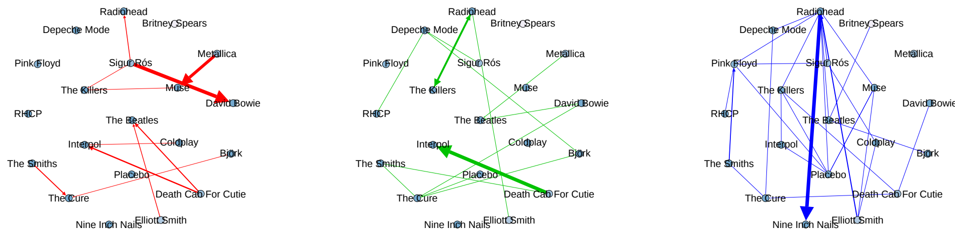
In Figure 4.2, we show three behavioral clusters discovered in one of our experimental datasets consisting of logs of artists played by users of an online radio station (see Section 4.3.1).[3] The clusters are computed using k-means with the KL-divergence measure (4.1). Observe that different clusters of users exhibit distinct navigational patterns among the top-20 artists.

4.2.3 Incorporating Interactional Context

We motivate our approach from a classical memory-based collaborative filtering model. Memory-based algorithms predict ratings for a given user

[2]In practice, many of M_{ij}'s are 0 and to have a well-defined $d_{KL}(\cdot,\cdot)$, we add a relatively tiny value to all the entries of the transition matrices.

[3]For clarity, we only show the top-20 artists with the highest play counts in the training data and omit edges whose transition probability is lower than a certain threshold.



(a) Cluster 1. (b) Cluster 2. (c) Cluster 3.

Figure 4.2: Behavioral clusters in `lastfm` dataset (see Section 4.3.1). We see how different clusters of users move between the top-20 artists with the highest play counts in the training data. Thicker edges represent higher transition probabilities.

based on past ratings of the user and other users in the system [88, 12]. The neighborhood models are a classical example, where the predicted rating of an item by a user is given by a weighted combination of "k-nearest neighbor" items (or users). The weights are proportional to the "similarity" between items, which is represented by the vector of observed user ratings. The Pearson correlation coefficient and cosine similarity are two commonly used similarity measures. The similarities between all item pairs are computed offline, and the predicted rating $\hat{r}_{u,i}$ for a user u and an item i is given by

$$\hat{r}_{u,i} = b_{u,i} + \sum_{j \in \mathcal{N}(i)} w(i,j)(r_{u,j} - b_{u,j}), \tag{4.3}$$

where $\mathcal{N}(i)$ denotes the "neighborhood" set of item i. For example, in the cosine similarity case, the top-k items with highest cosine similarity with item i constitute $\mathcal{N}(i)$. Typically, a rating bias term $b_{u,i}$ is included to account for the user bias (some users are predisposed to rate higher in general) and item bias (some movies get higher ratings than others). The weights are usually normalized for rating prediction tasks.

Let us focus on the item recommendation problem in the interactional context setting, where the knowledge of the current session is available. First, we want to use the cluster Markov models computed using (4.2) to learn a ranking of items given the user u and the current session s. Second, we want to incorporate interactional context in (4.3). In particular, we want to introduce a *context bias* term $c_{u,i}$ in addition to the rating bias $b_{u,i}$. In the implicit feedback setting, we interpret $b_{u,i}$ as accounting for *click bias* rather than rating bias. To this end, we want a scoring function $f_{u,i}$ using both past sessions and current session s for the user u. Suitably modifying (4.3), we have,

$$f_{u,i} = b_{u,i} + \alpha \sum_{j \in \mathcal{N}(i)} w(i,j)(f_{u,j} - b_{u,j}) + (1-\alpha)c_{u,i}, \tag{4.4}$$

where the nonnegative weight α controls the tradeoff between the current context and information from the past sessions. Notice the recurrence nature of the above equation — we want to estimate all user-item scores *as the session progresses*, as against the context-aware model suggested in [9] based on attribute-based context variables.

Define $z_{u,i} = f_{u,i} - b_{u,i}$ for all $u \in \mathcal{U}$ and $i \in \mathcal{I}$, so that we can remove the click bias from the equation resulting in a simpler model:

$$z_{u,i} = \alpha \sum_{j \in \mathcal{N}(i)} w(i,j) z_{u,j} + (1-\alpha)c_{u,i}. \tag{4.5}$$

From the graph corresponding to the Markov model with transition probability matrix M, we set $w(i,j) = M_{ji}$. This gives an intuitive interpretation that items tend to transition to i more often should receive higher similarity than items that do not. Given the choice of $w(i,j)$, $\mathcal{N}(i)$ corresponds to items that are adjacent to i in the Markov graph. Estimating $c_{u,i}$ using the click sequences can be hard and expensive. However, it is easy to specify what items appear in the current session s. We let $c_{u,i} = 1$ if the item a_i appears in s or 0 otherwise. Let $\mathbf{c}_u$ denote the indicator vector with ith entry equal to $c_{u,i}$, and let the vector of item scores for a user u be $\mathbf{z}_u$. Rewriting (4.5) in matrix form: $\mathbf{z}_u = \alpha M^T \mathbf{z}_u + (1-\alpha)\mathbf{c}_u$. Letting $\mathbf{1}$ denote vector of all ones, and normalizing $\mathbf{z}_u$ to sum to 1, the above equation can be rewritten as:

$$\mathbf{z}_u = (\alpha M + (1-\alpha)\mathbf{1}\mathbf{c}_u^T)^T \mathbf{z}_u. \tag{4.6}$$

The quantity $\mathbf{z}_u$ is nothing but the personalized PageRank vector for user u, using the graph M and personalization vector $\mathbf{c}_u$. Thus we have an efficient way to estimate $z_{u,i}$. Recall that the click bias $b_{u,i}$ was obviated to compute $\mathbf{c}_u$. So, the final score is given by $f_{u,i} = z_{u,i} + b_{u,i}$. A standard way to estimate click bias is to average the launch count of item a_i over users.[4] However, a better choice would be to use transition probabilities of item a_{i_t} to other items, where i_t is the index of the last item in s, since we focus on item transitions. We find in our experiments that $M^{(u)}_{i_t,i}$ is an effective choice for click bias. The final score for user-item pair (u,i), given the current session $s = \langle a_{i_1}, \cdots, a_{i_t} \rangle$ is:

$$f_{u,i} = z_{u,i} + M^{(u)}_{i_t,i}, \tag{4.7}$$

[4]Note that user bias need not be added as we only need to rank items.

where $z_{u,i}$ is the solution to (4.6).

4.2.4 iConRank Algorithm

We are now ready to give our algorithm iConRank for recommending items in the collaborative filtering with interactional context setting. Given a current session $s = \langle a_{i_1}, a_{i_2}, \ldots, a_{i_t} \rangle$ executed by a given user u, history of sessions $\mathcal{S}$ for all users in $\mathcal{U}$, and number of behavioral clusters K, the iConRank algorithm is specified as follows:

1. (**Offline step**) Cluster $\mathcal{U}$ using the sessions history $\mathcal{S}$ into K clusters by first forming the per-user transition matrices and then using the k-means algorithm as described in Section 4.2.2. Compute the corresponding Markov transition probabilities M_k for each cluster $k \in [K]$ using (4.2).
2. Let $\pi(u)$ denote the cluster to which user u belongs. Set $\mathbf{c}_u$ to be the normalized indicator vector of items appearing in s. Compute the personalized PageRank $\mathbf{z}_u$ by (4.6) with $M = M_{\pi(u)}$ and current $\mathbf{c}_u$.
3. Compute scores $f_{u,i}$, for all $i \neq i_t$ using (4.7).
4. Rank items using the computed scores and return the top-N items as recommendations for $a_{i_{t+1}}$.

A few remarks on the iConRank algorithm are in order:

Efficiency: Note that the clustering step is done offline as it is independent of current session s. Steps 2 through 4 are executed *every time* the user clicks, i.e., as session s progresses, so that the recommendations can be updated. Computing the personalized PageRank in Step 2 can be expensive if $|\mathcal{I}|$ is of the order of tens of thousands, even if the matrices M_k are sparse. Scalability is even more of a concern if the algorithm is to be implemented in real-time systems with small processing capabilities like mobile phones. The linearity property of personalized PageRank [40] stated below can be exploited to implement Step 2 in a scalable way. Let $\mathbf{z}_u(\mathbf{v})$ denote the solution to (4.6) computed with the personalization vector $\mathbf{c}_u = \mathbf{v}$. Then:

$$\mathbf{z}_u(\beta\mathbf{v}_1 + (1-\beta)\mathbf{v}_2) = \beta\mathbf{z}_u(\mathbf{v}_1) + (1-\beta)\mathbf{z}_u(\mathbf{v}_2)$$

for nonnegative β and distributions $\mathbf{v}_1$ and $\mathbf{v}_2$. We can pre-compute $\mathbf{z}_u(\mathbf{e}_i)$ for $i = 1, 2, \ldots, |\mathcal{I}|$, where $\mathbf{e}_i$ denotes the ith column of the identity matrix. Then, for a given session $s = \langle a_{i_1}, a_{i_2}, \ldots, a_{i_t}\rangle$, and chosen $\mathbf{c}_u = \mathbf{v}$ we can compute $\mathbf{z}_u$ efficiently as:

$$\mathbf{z}_\mathbf{u}(\mathbf{v}) = \sum_{j=1}^{t} v_{i_j} \ . \ \mathbf{z}_\mathbf{u}(\mathbf{e}_{i_j})$$

It is easy to see why the above equality holds: Our choice of $\mathbf{c}_u = \mathbf{v}$ is such that it is non-zero only in the positions corresponding to items that appear in s, and is normalized to sum to 1.

Cold Start: The algorithm can seamlessly deal with new users. In particular, we can choose $\mathbf{c}_u$ to be the uniform distribution over items and use the global Markov model instead of $M_{\pi(u)}$.

4.3 Experiments

In this section we present experimental results of our proposed algorithm, iConRank, on two real-life datasets: `Apps` and `LastFM`. We first give a detailed description of the two datasets and the experimental setting. Next we compare the recommendation performance of iConRank to a number of other successful methods. We also evaluate the methods on recommending existing and new items and study how performance is influenced by current session length.

For testing, we measure the accuracy of recommending the next item $a_{i_{t+1}}$, given the current session $s = \langle a_{i_1}, a_{i_2}, \dots, a_{i_t} \rangle$ of a user. Thus, there is only one correct item for each test case. A user is usually presented with a small list of recommended items. Therefore, we measure *recall at top-N* (Recall@N) for $N = 5, 10, 15, 20$. For the sake of completeness, we also report the *ROC curve* (recall vs. precision plot up to $N = 1,000$), which is the standard measure for comparing the recommendation qualities of different methods for recommender systems. Recall and precision are measured in the standard way for top-N recommendation tasks [21].

4.3.1 Dataset Description

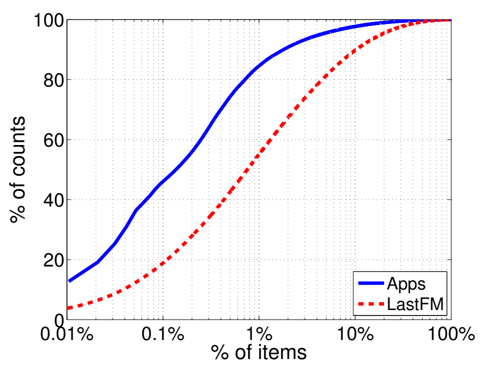
The datasets used in this study consist of user event logs with timestamps, where the events are launching mobile apps or streaming tracks of an artist. Sessions are formed from the user logs. We note that consecutive uses of the same item are considered as a single interaction for methods using sequences, including ours. However, launch counts are retained for use by methods based on the user-item matrix. Training data consists of all sessions before a chosen date.[5]

Apps Dataset: The Apps dataset is a proprietary dataset obtained from a manufacturer of mobile devices. It consists of mobile app usage patterns of 17,062 smart phone users over 9,583 apps.[6] The dataset spans over a one year period and each log record consists of user, timestamp information and one of the two events, app launch and screen on/off. The screen on/off event is used as an indicator of a new session, i.e., a new session is started when the screen is on and the current session is ended when the screen is off *and* at least one minute has elapsed.[7] In the end, we have 1,167,171 training and 459,899 test sessions. Figure 4.3(a) plots the distribution of item count, which corresponds to launch count of apps. Apps in the horizontal axis are ordered by their launch counts, with the most frequently launched app on the left. We can see in Figure 4.3(a) that top 1% of the most frequently launched apps cover about 85% of total launch counts. Figure 4.3(b) shows the CDF of the length of sessions. About 87% of sessions have at most length 10. Our statistics align with the findings in [90] that interactions with smart phones are mostly brief.

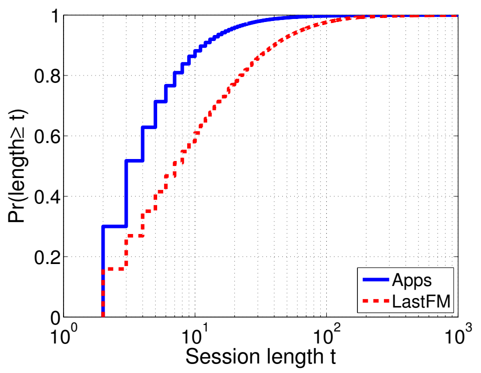
[5]Date was chosen to maximize the number of users appearing in both training and testing sets.

[6]We emphasize that the data provided to us was highly anonymized and contained only generic identifiers that cannot be correlated or traced back to actual users.

[7] Time lapses were tuned to achieve the most meaningful session statistics.



(a) Item count distribution (most frequent item on the left).



(b) CDF of the length of sessions.

Figure 4.3: Item count and session length distributions for `LastFM` and `Apps` datasets.

LastFM Dataset: The `LastFM` (`last.fm`) dataset consists of listening habits of about 1,000 users, where each log contains user, track, artist and timestamp (98,412 artists appear in the dataset). It has been used in many studies [41, 106]. Here, the task is to recommend the next artist to listen to. Unlike the `Apps` dataset, the `LastFM` dataset lacks any explicit indication of start or end of a session. So, we mark sessions by periods of inactivity. We end a session if there is no other artist streamed within an hour from the last artist and start a new one with the next artist resulting in 644,001 training and 95,038 test sessions.[7] Listening to music is a more time consuming activity than using a smart phone. Furthermore, preference over artists is more diverse than the apps case, where pre-installed apps are more commonly used. These differences show up in Figure 4.3. Sessions in `LastFM` tend to be longer and more diverse — about 50% of sessions have at least length 10 and top 1% of the most frequent artists cover only about 55% of total counts.

4.3.2 Experimental Results

We compare iConRank to the following collaborative filtering (CF) methods:

- NNCosNgbr: Non-Normalized Cosine Neighborhood [21] is a neighborhood-based model using cosine similarity between items (represented as vectors of launch counts). The score for item a_i for a given user is calculated as a weighted average of the k-nearest neighbor items of a_i as in (4.3), where we set $k = 5$ and use cosine similarity as the weights.

- SVD(r): The scores are computed as $\hat{R} = U\Sigma V^T \approx A$, where $A \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{I}|}$ is the user-item count matrix, r is the number of latent factors, $U \in \mathbb{R}^{|\mathcal{U}|\times r}$, $V \in \mathbb{R}^{|\mathcal{I}|\times r}$ are orthonormal matrices and $\Sigma \in \mathbb{R}^{r\times r}$ is a diagonal matrix of the top r singular values. Missing values of A are set to zero.

- Markov: The recommended item a_{t^*} depends on the last item a_t in the current session via the global Markov model. Formally, $t^* = \text{argmax}_j M_{tj}$, where M is the global transition probability matrix estimated from the sessions of all users.

- ContextNgbr: Identical to NNCosNgbr except that the k-nearest neighbors are computed from the set of items in the current session. We use $k = 5$ (include all items in the current session if its length is less than k).

- SeqPattern: Sequence mining algorithm used in [38].[8] We use the last 10 items as the user's active session.

The last three methods are context-aware approaches that consider user's current session. We note that ranking items by popularity (i.e. number of times an item is accessed in the training phase), which is known to perform reasonably well in recommendation tasks, does significantly worse in our setting than the methods we compare here.

Table 4.1 reports the Recall@N of the methods on both datasets for $N = 5, 10, 15, 20$. Our algorithm, iConRank, significantly outperforms all other methods with impressive recall rates. It achieves the highest recall of 0.8632 for the `Apps` dataset, which means the app $a_{i_{t+1}}$ that a user will use has a probability of about 86% to be ranked in the top-20 results returned by iConRank. The next best performing method for the `Apps` dataset is SeqPattern followed by ContextNgbr, both of which use current context. However, they are the two worst performing methods for the `LastFM` dataset. This can be expected as the number of items is much larger than in the `Apps` dataset. SeqPattern suffers from the sheer diversity of artists and lack of adequate representative patterns in the training data. Among the two CF based techniques, SVD performs better than NNCosNgbr in both the datasets confirming the findings of [21].[9] Surprisingly, the simple Markov model performs better than CF based

[8][38] applies sequence mining on latent topics discovered from playlists, but we mine sequence of items directly as we don't have item features.

[9]We tested SVD with various ranks r, and chose $r = 200$ for the `Apps` dataset and $r = 300$ for the `LastFM` dataset. Larger ranks yielded only marginal performance improvements.

Method	`Apps` dataset				`LastFM` dataset			
	$N = 5$	$N = 10$	$N = 15$	$N = 20$	$N = 5$	$N = 10$	$N = 15$	$N = 20$
NNCosNgbr	0.4301	0.5478	0.6167	0.6636	0.0691	0.1044	0.1328	0.1560
SVD	0.4574	0.5853	0.6480	0.6851	0.0810	0.1286	0.1633	0.1922
Markov	0.4592	0.5744	0.6370	0.6754	0.0631	0.0905	0.1113	0.1285
ContextNgbr	0.5266	0.6248	0.6739	0.7045	0.0597	0.0775	0.0884	0.0971
SeqPattern	0.5517	0.6451	0.6899	0.7223	0.0371	0.0536	0.0656	0.0748
iConRank	**0.6701**	**0.7927**	**0.8386**	**0.8632**	**0.1277**	**0.1882**	**0.2304**	**0.2633**

Table 4.1: Recall@N results on `Apps` and `LastFM` datasets for $N = 5, 10, 15, 20$. iConRank outperforms the other methods in all cases, achieving 20% and 37% improvement in Recall@20 over the next best method on `Apps` and `LastFM` datasets, respectively.

methods for the `Apps` dataset, but fails in the `LastFM` dataset conforming to our intuition that a single global Markov model is not enough. In evaluating the methods on the `Apps` dataset, the recommended candidate apps for a given user can contain apps that are *not* installed in the user's device, which is consistent with the evaluation on the `LastFM` dataset.[10]

We further study the performance in different categories: existing and new items. Specifically, evaluation is restricted to items that appeared at least once in the training data ("existing") or those that were not seen in the training data ("new") for each user. The results are presented in Figure 4.4. We can see that iConRank not only performs well on existing items, but is also capable of recommending new items to users significantly better than the rest.

A desirable property of our algorithm is that it is able to make more accurate recommendations as more items are observed in a given session. We

[10]The `Apps` dataset did not contain information about which apps are installed in the user device.

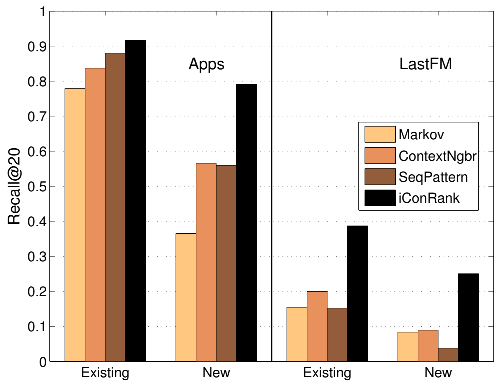


Figure 4.4: Recall@20 for existing and new items. iConRank performs the best in all cases.

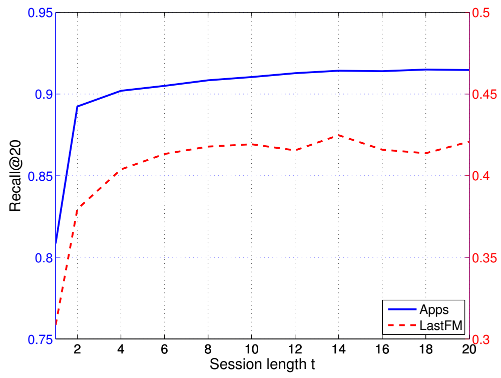


Figure 4.5: Recall@20 of different session lengths. Performance increases as the session length increases showing that iConRank effectively captures the current context.

examine how iConRank performs as more of the current session is revealed. In particular, we measure Recall@20 when t (length of s) varies as shown in Figure 4.5. As the session length increases, Recall@20 also increases for both datasets, illustrating that iConRank effectively captures the current context of the session. Especially, the performance greatly increases after observing just two items in a given session. However, we also find that recall rates can drop when t becomes very large (not shown in the figure) — the personalization vector tends to a uniform distribution and context ceases to have relevance. In practice, most of the sessions are short as shown in Figure 4.3(b).

4.4 Conclusions

We have considered the problem of collaborative filtering with interactional context to recommend potential items of interest to a user already engaged in a session, using past sessions of the user and of other users. We are given implicit feedback in the form of clicks and a session that establishes the current context of a user. Our problem setting differs from the traditional setting of collaborative filtering in crucial respects. We propose iConRank, a novel method that is motivated by introducing 'context bias' in the neighborhood-based model. Our formulation essentially leads to the personalized PageRank, where context is captured by the personalization vector. Experimental results on real-life datasets demonstrate that our algorithm achieves superior recommendation performance illustrating its ability to capture the context of a given session.

Chapter 5

Collaborative Filtering with Side Information

The range of side-information sources of users and items stretching beyond the user-item matrix is quite broad and varied. One of the most common side information sources is attribute information, which reflects properties of users and items (e.g., user's age and gender, item's category) [3]. More recently, two sources of information that have increased in importance in recommender systems are *social networks* and *user-generated* information. Social networks introduce information in the form of user-user relationships such as trust or friendship. Recommender systems that attempt to leverage social networks, both directed or undirected, apply the assumption that users who stand in a positive relationship with each other also share similar interests [47, 70]. User-generated information has also become widely available in recommender systems and its volume has grown exponentially. Examples include tags, geotags, multimedia content and reviews or comments, which has been shown to be highly beneficial for improving the recommendation performance [35, 76, 64]. Here, we examine the problem of recommending blogs to follow, which naturally entails rich side information as blog posts contain user

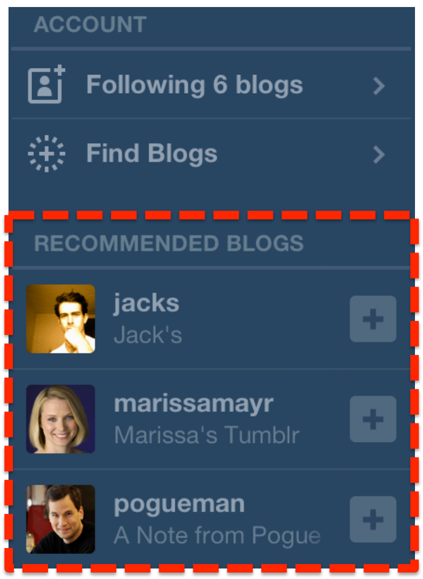
The materials presented in this chapter have been published in [92, 93]. Donghyuk Shin formulated the problems, developed the algorithms, and conducted experiments.

generated text and multimedia content.

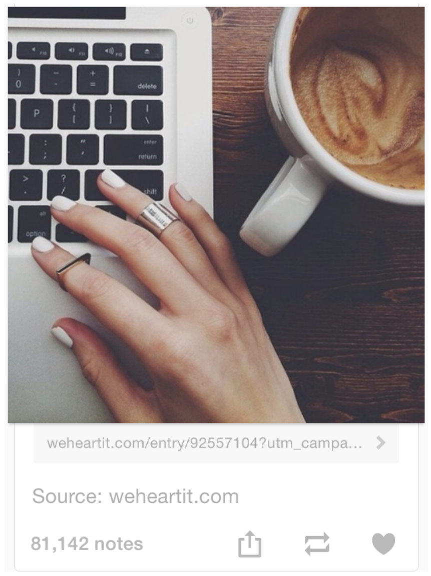
In this chapter, we explore rich side-information of users and items additional to the user-item matrix in the context of blog recommendation. Microblogging services have emerged as a leading content sharing and communication platform combining both traditional blogging and social networking characteristics. Tumblr[1] is one of the most popular microblogging services with more than 200 million users, where users can create and share posts with the followers of their blogs. Conversely, users consume shared content by following blogs of interest, which has become an overwhelming task due to the sheer number of options. Thus, one of the core problems in microblogging sites is predicting whether a user will follow a blog or not. Improved blog recommendations would not only lead to higher user engagement by assisting users to discover interesting content, but also attract more appealing followers for sponsored or advertisers blogs. Figure 5.1(a) shows the blog recommendation module in Tumblr.

The problem of recommending blogs differs from traditional collaborative filtering settings, such as the Netflix rating prediction problem [11], in two main aspects. First, interactions between users and blogs are *binary* in the form of follows and there is no explicit rating information available about user preferences. The "follow" information can be represented as a unidirectional unweighted graph and popular proximity measures based on the structural

[1]`www.tumblr.com`



(a) Blog recommendation module in Tumblr



(b) Example post with high note count.

Figure 5.1: The blog recommendation module (a) and an example post (b) with high note count (i.e., like and reblog count) in Tumblr.

properties of the graph can then be applied to the problem [108]. Secondly, an important but beneficial difference is that blog recommendation inherently entails rich *side information* in addition to the conventional user-item matrix (i.e., follower graph). There are two main categories of side information: (1) *user generated content* such as images, tags and text (e.g., Figure 5.1(b)) and (2) *user activity* including likes and reblogs. In the case of Tumblr, incorporating image features is crucial as majority of posts contain photos. Text data is also rich in Tumblr, since posts have no limitation in length, compared to other microblogging sites such as Twitter[2]. While such user generated con-

[2]`www.twitter.com`; posts (or tweets) are restricted to 140 characters.

tent characterizes various blogs, user activity is a more direct and informative signal of user preference as users can explicitly express their interests by liking and reblogging a post. This implies that users who liked or reblogged the same posts are likely to follow similar blogs. In fact, as shown in many existing studies, such side information not only improves recommendation quality, but also alleviates sparsity issues in the user-item matrix [70, 42, 91].

On the other hand, rigorous approaches for incorporating side-information in a recommender system setting are lacking. Consider the standard matrix completion (MC), one of the most widely used and theoretically well-studied method for recommendation tasks, for which there have been several rigorous guarantees established in the recent past [53, 15, 46, 18]. However, MC is exposed to data sparsity issues and restricted to the transductive setting, i.e., predictions can only be made for existing users/items, as it only considers observations from the user-item matrix. More recently, the inductive matrix completion (IMC) was proposed and theoretically analyzed by [45] motivated by settings where side information of users/items is available in the form of feature vectors. However, IMC assumes that observed entries are fully explained by such features, which is not always the case especially with noisy features that do not support the user-item matrix. Furthermore, IMC cannot make meaningful recommendations for users or items without any features, which is often the case in Tumblr (see Section 5.1).

To this end, we propose a novel Boosted Inductive Matrix Completion (BIMC) model for blog recommendation that combines the power of an inductive matrix completion model together with a standard matrix completion model via boosting. Specifically, BIMC first applies the MC model to smooth the input matrix and reduce the noise level by low-rank approximation, and then further models the residual of the approximation with the IMC model. That is, BIMC captures both the low-rank structure of follow relationships as well as the latent structure using side-information of users and items in an additive manner capturing entries in the follower graph where MC fails to learn.

By incorporating user/blog features, BIMC is also capable of making recommendations in the *inductive* setting, i.e., make predictions for users or blogs *not seen at training time*, which includes cold-start[3] cases. This is particularly important for Tumblr as users and blogs often have very few or no links in the follower graph as shown in Section 5.1. Experiments on large-scale real-world proprietary data from Tumblr show that our proposed BIMC significantly outperforms MC, IMC and several other standard methods for the blog recommendation task.

Lastly, an important issue is how to effectively represent the three side-information sources (image, text and activity) as features. Recently, deep learning approaches have emerged as a powerful class of models that understand semantic content of images, giving state-of-the-art performance on image recognition tasks [57, 26, 50]. This is also the case for text data, where

[3]Cold-start refers to users or items without any known entries in the user-item matrix.

vectorial representations of words capturing semantic relations between them are learned from neural networks [73, 103]. Encouraged by these results, we employ deep learning features for both images and tags/text as a useful and robust representation of users and blogs. For activity features, we represent likes and reblogs as a weighted graph similar to the follower graph and we compute principal components of the activity graph as features. To our knowledge, we are the first to consider image as well as activity features. Furthermore, adopting deep learned features for recommender systems is still unexplored.

Our contributions are summarized as follows:

- We propose a Boosted Inductive Matrix Completion based blog recommendation system that combines the power of an inductive matrix completion model together with a standard matrix completion model.
- We represent users and blogs with an extensive set of side information sources such as the user activity, text/tags, and images; and extract a comprehensive set of features using state-of-the-art deep learning methods.
- We show that the proposed BIMC model effectively combines heterogeneous user and blog features from multiple sources for more accurate recommendations.
- We conduct extensive experiments as well as detailed analysis on large-scale real-world data from Tumblr, and demonstrate the superiority of the proposed BIMC method over several state-of-the-art baselines.

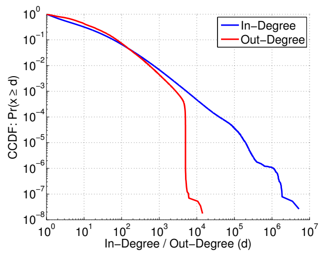
5.1 Tumblr Dataset

In this section, we analyze some important characteristics of different aspects of the Tumblr data.[4] As a social network service, Tumblr users can follow blogs of interest without mutual confirmation similar to Twitter, but different from Facebook[5]. The follow information can be represented as a directed bipartite graph where nodes correspond to users and blogs and an edge from node i to j represents user i following blog j. We use a snapshot of the follower graph sampled from June 2014, which consists of 76.86 million nodes with 2.27 billion edges. We find similar characteristics as in [16] including the in/out degree distributions shown in Figure 5.2(a). The in-degree follows a power-law distribution, while the out-degree does not and shows a sharp drop when the out-degree is around 5,000, which is the maximum number of blogs a user can follow in Tumblr. About 50% of nodes are without any followers (i.e., 0 in-degree) and the maximum in-degree is 5.22 million, while about 25% of nodes are not following any blogs (i.e., 0 out-degree) and the maximum out-degree is 14,208.

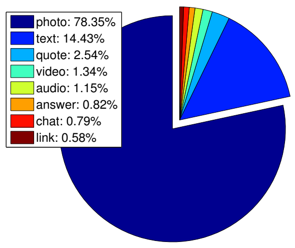
As a microblogging platform, Tumblr provides useful tools close to that of traditional blogging sites for creating longer, richer and higher quality content. Specifically, it allows users to create 8 different types of posts: *photo*, *text*, *answer*, *link*, *quote*, *video*, *audio* and *chat*. Furthermore, posts in Tumblr

[4]The reported datasets and results are deliberately incomplete and subject to anonymization, and thus do not necessarily reflect the real portfolio at any particular time.

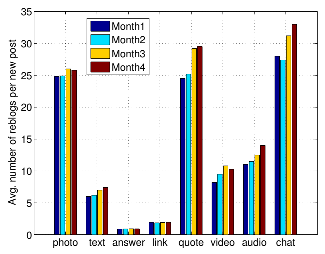
[5]`www.facebook.com`



(a) In/out degree distribution of the follower graph.



(b) Distribution of different post types.



(c) Average number of reblogs per new post for each post type.

Figure 5.2: Statistics of Tumblr data.

have no limitation in length unlike other microblogging sites such as Twitter, which is restricted to 140 characters per post. It also supports the use of *tags* for each post, which are separate from the post content. Lastly, users can *like* a post or re-broadcast the post to its own followers by *reblogging*. While these two activities have different intentions to the user, both directly reflect the user's interest which should be utilized for better recommendation quality.

We processed 5 months of Tumblr data, where each month contains about 1.5 TB of sampled records of posts created, reblogged and liked. Note that we restrict to users with at least 5 records in each month. On average, there are more than 150 million newly created posts, 2.5 billion reblogged posts and 2 billion likes per month. We show the distribution of each post type in Figure 5.2(b). Almost 80% of posts are *photo* posts suggesting image features are a crucial component for analyzing posts. Figure 5.2(c) reports the average number of reblogs a new post gets for each post type. We can see in the figure that photo, quote and chat posts are reblogged significantly more than other types of posts. Overall, a new post gets reblogged more than 15 times on average illustrating the high sharing activity in Tumblr. We have also found that about 8.3% of users do not have any posts and about 12.2% of users do not have any activity information. More detailed analysis of the Tumblr data can be found in [16].

5.2 Methods

In this section, we describe a natural way of combining various user/blog features and the follower graph to enable the inductive setting, i.e., recommendations for new users and blogs. We first describe the Inductive Matrix Completion method for blog recommendation, which is based on the proposition that user-blog follow behavior arises from applying a low-rank matrix to user and blog features. Next we motivate and present our proposed Boosted Inductive Matrix Completion method. We briefly establish the notation used before describing our proposed approaches.

Notation: We denote the follower graph by $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$, where $\mathcal{V}_1$ ($m = |\mathcal{V}_1|$) and $\mathcal{V}_2$ ($n = |\mathcal{V}_2|$) is the set of users and blogs, respectively; $\mathcal{E} = \{e_{ij} | i \in \mathcal{V}_1, j \in \mathcal{V}_2\}$ is the set of edges indicating user i follows blog j. Let $A \in \mathbb{R}^{m \times n}$ be the adjacency matrix of $\mathcal{G}$, where each row corresponds to a user and each column corresponds to a blog, such that $A_{ij} = 1$, if user i is following blog j and 0 otherwise. That is, we treat missing values as zeros. Note that $\mathcal{G}$ is a directed graph, i.e., A is non-symmetric. Let $X \in \mathbb{R}^{m \times f_u}$ and $Y \in \mathbb{R}^{n \times f_b}$

denote the user and blog feature matrices, respectively.

5.2.1 Matrix Completion

The low rank matrix completion (MC) approach is one of the most popular and successful collaborative filtering methods for recommender systems [56]. The goal is to recover the underlying low rank matrix by using the observed entries of A, which is typically formulated as follows:

$$\min_{U,V} \sum_{(i,j)\in\Omega} (A_{ij} - (UV^T)_{ij})^2 + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2), \tag{5.1}$$

where $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$ with r being the dimension of the latent feature space; $\Omega \in [m] \times [n]$ is the set of observed entries; λ is a regularization parameter. Note that matrix completion only utilizes samples from the follower graph A and ignores the side information that might be present in the system.

5.2.2 Inductive Matrix Completion

The standard matrix completion formulation is restricted to the transductive setting, i.e., predictions can only be made for existing users and items without re-training for latent factors of new users or items. Furthermore, the standard formulation suffers performance with extreme sparsity in the data, which is the case for Tumblr as about 50% of users do not have any followers and about 25% of users are not following any blogs. One simple way to make predictions for such users is to use a popularity based global ranking of blogs and recommend the top ranked ones. In order to make meaningful predictions, one would need more information about users and blogs. For Tumblr, such information can be obtained from rich content (photos, text) and activity (reblog, like) information.

Recently, a novel inductive matrix completion (IMC) approach was proposed and theoretically analyzed by [45] to alleviate data sparsity issues as well as enable predictions for new users and items by incorporating side information of users and items given in the form of feature vectors. The main idea is to model A_{ij} using user i's feature vector $\mathbf{x}_i \in \mathbb{R}^{f_u}$, item j's feature vector $\mathbf{y}_j \in \mathbb{R}^{f_b}$ and a low-rank matrix $Z \in \mathbb{R}^{f_u \times f_b}$ as

$$A_{ij} = \mathbf{x}_i^T Z \mathbf{y}_j. \tag{5.2}$$

That is, the interaction between user i and item j is generated by applying their respective feature vectors to Z. For a new item b, the predictions A_{ib} for each user i can be calculated with the feature vector $\mathbf{y}_b$ available.

By factoring $Z = WH^T$, the goal of IMC is to recover $W \in \mathbb{R}^{f_u \times r}$ and $H \in \mathbb{R}^{f_b \times r}$ using the observed entries in A. The IMC objective is given as

$$\min_{W,H} \sum_{(i,j)\in\Omega} \ell(A_{ij}, \mathbf{x}_i^T W H^T \mathbf{y}_j) + \frac{\lambda}{2}(\|W\|_F^2 + \|H\|_F^2),$$

for some loss function ℓ that measures the difference between the observations and predictions, e.g., squared loss $\ell_s(a,b) = (a-b)^2$ or logistic loss $\ell_l(a,b) = \log(1+e^{-ab})$. The number of parameters to learn is $(f_u + f_b) \times r$ depending only on the number of user and item features, whereas there are $(m+n) \times r$

parameters in the standard matrix completion. Note that matrix completion is a special case of IMC when $X = I$ and $Y = I$.

For a convex loss function ℓ, the above IMC objective becomes a convex function when either W or H is fixed (similar to the standard matrix completion case). The computational cost to solve the optimization problem differs based on the choice of the loss function ℓ. In our experiments, we use the squared loss in the objective and employ the alternative minimization approach in [109]. Under this setting, the computational cost for each step is $O((nnz(A) + mf_u + nf_b)r^2c)$, where $nnz(A)$ is the number of non-zeros in A and c is a small constant. In our experiments, f_u, f_b and r are very small (few hundreds) and the solution converges in less than 10 iterations.

5.2.3 Boosted Inductive Matrix Completion

Next we present our method called Boosted Inductive Matrix Completion (BIMC). One issue with IMC is that the model is too rigid as it heavily depends on the user features X and item features Y. That is, user and item features from different sources should support the underlying structure of the follower graph A in order to make good predictions. Let $X = U_X \Sigma_X V_X^T$, where $U_X \Sigma_X V_X^T$ is the SVD of X. Similarly, let $Y = U_Y \Sigma_Y V_Y^T$ be the SVD of Y. From the IMC formulation, we have

$$A = XZY^T = U_X(\Sigma_X V_X^T Z V_Y \Sigma_Y)U_Y^T = U_X \hat{Z} U_Y^T,$$

where $\hat{Z} = \Sigma_X V_X^T Z V_Y \Sigma_Y$. Thus, the subspace spanned by U_X must have significant overlap with that of A to achieve small error. For example, like and reblog activity features can be quite helpful as a direct reflection of user interest. Similar arguments can be made for Y as well.

However, features from various sources may not always support the matrix A and IMC can suffer performance significantly in such case. For instance, text data in Tumblr is extremely sparse and noisy, and thus may not directly reveal user preference. Moreover, it is not always the case that all users and items have features (as shown in Section 5.1), in which IMC would not be able to make any predictions.

To address these problems, we propose to combine both standard matrix completion and inductive matrix completion, and thereby better utilize the power of both approaches. That is, we combine the power of MC to reduce the noise level in the input data as well as the advantage of IMC to incorporate side information of users and items. Our idea is to model A_{ij} as

$$A_{ij} = (UV^T)_{ij} + \alpha \mathbf{x}_i^T Z \mathbf{y}_j, \tag{5.3}$$

where the parameter α adjusts the contribution of features in the final prediction. Choosing a good α is crucial for both performance and solving the optimization problem, which can be difficult to tune. Furthermore, simultaneously solving for all four latent factor matrices U, V, W and H will lead to slower convergence due to the increased number of parameters.

Thus, our strategy is to first learn the latent factor matrices U and V of the MC model as in (5.1). The resulting approximation error or residual matrix $R = A - UV^T$ represents links in the follower graph that MC could not fully capture. Then we model R_{ij} with IMC as

$$R_{ij} = A_{ij} - (UV^T)_{ij} = \mathbf{x}_i^T Z \mathbf{y}_j. \tag{5.4}$$

In other words, we first try to find the support of the follower graph A with the latent factors U and V and focus on the part that it can not accurately model using IMC. This is especially useful when the norm of the residual from (5.1) is large, which suggests a significant deviation from low rank structure in A. Our objective is

$$\min_{W,H} \sum_{(i,j)\in\Omega} \ell(A_{ij} - (UV^T)_{ij}, \mathbf{x}_i^T W H^T \mathbf{y}_j) + \frac{\lambda}{2}(\|W\|_F^2 + \|H\|_F^2).$$

With ℓ being the squared loss and fixing H, the gradient of the above objective in matrix form is given as

$$X^T(A - UV^T - XWH^TY)Y^TH + \lambda W.$$

Note that we do not have to explicitly form $A - UV^T$, which would be a dense matrix and infeasible to store in memory.

Our approach eliminates the need to fine tune the parameter α. Furthermore, existing efficient solvers for MC and IMC with theoretical guarantees [46, 109] can be directly applied. An overview of our proposed BIMC model can be found in Figure 5.3, where we can see that BIMC can handle both sparsity in A as well as users/items without features. Given a user i with features $\mathbf{x}_i$ and blog j with features $\mathbf{y}_j$, we use (5.3) to obtain predictions with the learned factors and α set to 1. Finally, we note that a converse approach

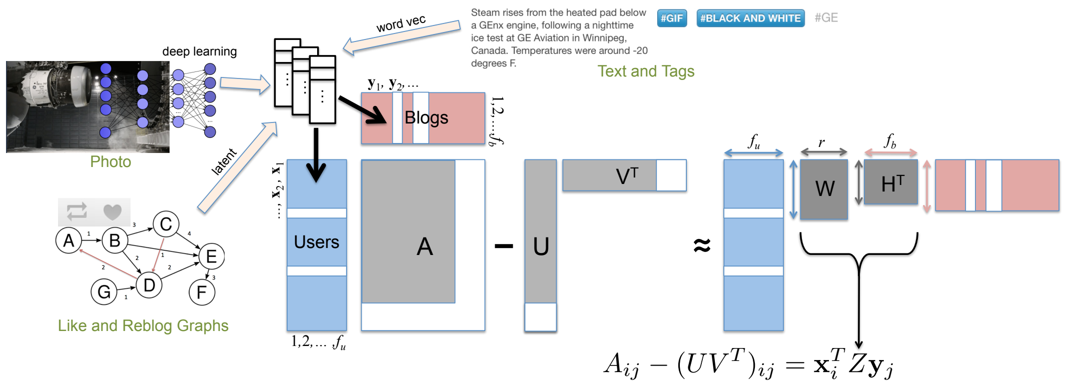

Steam rises from the heated pad below a GEnx engine, following a nighttime ice test at GE Aviation in Winnipeg, Canada. Temperatures were around -20 degrees F.
#GIF
#BLACK AND WHITE
#GE
Text and Tags
word vec
deep learning
Photo
latent
Like and Reblog Graphs
Blogs
Users
A
U
V^T
W
H^T
$\mathbf{y}_1, \mathbf{y}_2, \ldots$
$\ldots \mathbf{x}_2, \mathbf{x}_1$
$1,2,\ldots f_u$
$1,2,\ldots f_b$
f_u
r
f_b
$A_{ij} - (UV^T)_{ij} = \mathbf{x}_i^T Z \mathbf{y}_j$

Figure 5.3: Boosted Inductive Matrix Completion model. Shaded areas represent available information.

can also be used, i.e., learn the IMC model first, and then train the MC model on the resulting residual matrix; we found the results to be comparable, so we only present results for the former.

We illustrate the advantages of BIMC compared to IMC when features are noisy with the following experiment on the MovieLens-100K[6] dataset. We compute the rank-20 SVD of the user-movie matrix $A = U_A \Sigma_A V_A^T$ and set $X = U_A$ and $Y = V_A$, i.e., the left and right singular vectors of A to be the user and movie features, respectively. Then we perturb the columns of X by adding noise and measure the relative approximation error for IMC: $\|A - XWH^TY^T\|_F / \|A\|_F$ and for BIMC: $\|A - UV^T - XWH^TY^T\|_F / \|A\|_F$. Results are given in Figure 5.4 confirming that BIMC achieves lower approximation error rates than IMC due to modeling the residual matrix R as in (5.4). This

[6] 100,000 ratings from 1,000 users on 1,700 movies; `https://grouplens.org/datasets/movielens/`

shows that BIMC is robust to noisy features whereas IMC suffers performance as the noise level increases in the user features X.

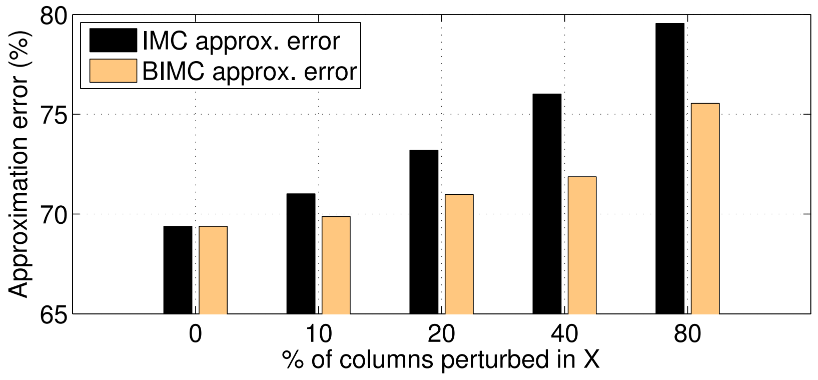


Figure 5.4: Relative approximation error for IMC and BIMC with different levels of noise in features.

5.3 Feature Extraction

As described in Section 5.1, Tumblr data contains three main sources of side information of users and blogs: (1) likes and reblogs, (2) tags and text, and (3) images. Extracting useful features from these sources is a crucial step of the recommender system. We discuss the details of how we extract these user and blog features as follows.

User Activity: For user activity information, we use likes and reblogs from Tumblr of the 1 million sampled users and blogs. Both like and reblog activities can be represented as a weighted graph similar to the follower graph A, where the edge weight between a user i and blog j is set to be the number of liked and reblogged posts of blog j by user i. The edge weights in both cases follow a power-law distribution. Each user likes or reblogs about 340 posts from 27 blogs and each blog gets a total of 410 likes or reblogs on average. In the experiments, we aggregate like and reblog graphs to a single activity graph from the training data and use a log-scale of the edge weights. One way to obtain useful and robust features is to consider the principal components of the adjacency matrix corresponding to the activity graph. That is, we compute p principal components and use them as latent user and blog features for IMC. Thus, we have $f_u = f_b = p$ user activity features for users and blogs, where we empirically set $p = 500$ in the experiments.

Tags and Text: Tags and text used in posts of Tumblr are extremely sparse and noisy. There are an average of 28.7 words and 4.8 tags per post. Furthermore, tags are unconstrained in Tumblr, where a user can put in any arbitrary text. Existing models based on bag-of-words (e.g., LSA, LDA) can suffer from such issues. Therefore, we utilize word2vec, which is a recent neural network inspired method that learns word embeddings in the vector space [73]. word2vec utilizes the technique called skip-gram with negative samples, which tries to represent each of the words by a vector such that words in similar contexts are close to each other. This representation is accomplished by maximizing the predicted probability of words co-occurring in the training corpus. In our work, we first compute d-dimensional vector representations of each word using word2vec, and then cluster these words into c clusters by the k-means algorithm. Using the cluster information, we finally create a histogram of word clusters for each post as a compact representation of tags and text used in that blog. We set $d = 300$ and $c = 1,000$ and processed both textual features for each month in the training data.

Images: Images are an important part of Tumblr data as shown in Section 5.1. We randomly sampled about 800K images per month from blogs that appear in the training data. We trained a convolutional neural network (CNN) [57, 26] on 1.5M Flickr images with labels due to the unavailability of image labels for the Tumblr dataset. The CNN is composed of seven hidden layers, which consist of five successive convolutional layers followed by two fully connected layers, plus a final soft-max layer. The nonlinearity of each neuron in this CNN is modeled by Rectified Linear Units (ReLUs) $f(x) = \max(0, x)$, which accelerates learning compared with saturating nonlinearity such as tanh units. The CNN takes a 224×224 pixel RGB image as input. Each convolutional layer convolves the output of its previous layer with a set of learned kernels, followed by ReLU non-linearity, and two optional layers, local response normalization and max pooling. The local response normalization layer is applied across feature channels, and the max pooling layer is applied over neighboring neurons. The output of the 7th layer is fed into the last soft-max layer, which outputs confidence scores over the pre-defined 958 categories for a given input image. Using the neural network, we extracted deep learning features from the sampled Tumblr images. For users, we averaged the resulting feature vector over all images that the user posted, liked and reblogged. For blogs, only posted and reblogged images were considered as reblogged posts also become a post of the blog.

5.4 Experiments

In this section, we present the experimental setup used to evaluate our proposed method BIMC in comparison to IMC as well as several other baseline methods on the Tumblr dataset for blog recommendation with additional side information of users and blogs. From the Tumblr follower graph, we randomly sampled 1 million users and blogs resulting in about 12 million follows, i.e., nonzero elements in A. Both user activity and user generated content information were collected over a 5 month period from Tumblr post data.

5.4.1 Baselines and Evaluation Metrics

We perform both offline and temporal evaluations. For the offline evaluation, we use 10-fold cross-validation. Temporal evaluation is used to simulate online evaluations, where we use data from preceding 4 months as training and the remaining month as testing.

In both cases, we compare BIMC against the standard matrix completion formulation (MC) and the Singular Value Decomposition (SVD), which has been shown to perform well for top-N recommendation tasks [21]. We also compare with methods that incorporate side information including the inductive matrix completion (IMC). Another popular approach is the collective matrix completion (CMC) [96, 14]. The goal of CMC is to jointly recover a collection of matrices with shared low rank structure, which is different from IMC. Specifically, given user-item matrix A, user features X and item features Y, CMC finds a joint factorization as:

$$A = UV^T, \quad X = UP^T \text{ and } Y = VQ^T.$$

That is, the shared user latent factor matrix U is obtained from both A and X (similarly V is obtained from A and Y). Recent work by [33] provides consistency guarantees for CMC, thus we use the algorithm presented in [33]. The Katz[7] measure [100, 94], which is one of the most successful proximity measures for link prediction, is also included in the comparison as a graph-based approach. We compute Katz scores between users and blogs on the combined (symmetric) matrix $C = \begin{bmatrix} S_u & A \\ A^T & S_b \end{bmatrix}$, where S_u and S_b are similarity matrices between users and blogs, respectively, computed from their features. Lastly, we report results of using a simple global popularity ranking (Global) for recommendation as a baseline, where blogs are ranked by the number of followers. We use rank $r = 10$ for MC and SVD, rank $r = 100$ for CMC, IMC and BIMC, and set $\lambda = 0.1$ for all methods, which are determined using cross-validation.

We measure the recommendation performance using precision (PRC@10) and recall (RCL@10) at top-10 generated by each method, which is the region

[7]The Katz measure is defined as $\sum_{i=1}^{\infty} \beta^i C^i$, and we set $\beta = 10^{-6}$.

of practical interest for recommender systems. We also report the AUC (area under the ROC curve) of each method for completeness.

5.4.2 Experimental Results

Results of the proposed BIMC method with user and blog features are shown in comparison to the baselines: Global, SVD, MC, Katz, CMC, and IMC for PRC@10, RCL@10 and AUC in Table 5.1 for the offline evaluation and Table 5.2 for the temporal evaluation (that simulated A/B testing conditions).

Table 5.1: Offline evaluation results of the proposed Boosted Inductive Matrix Completion method (BIMC) in comparison to several baselines.

Method	PRC@10	RCL@10	AUC
Global	1.03%	4.80%	0.8687
SVD	1.28%	5.10%	0.8530
MC	1.28%	5.07%	0.8515
Katz	1.90%	8.15%	0.9209
CMC	0.49%	2.41%	0.8996
IMC	2.93%	11.33%	0.9075
BIMC	**3.21%**	**12.28%**	**0.9221**

Table 5.2: Temporal evaluation results of the proposed Boosted Inductive Matrix Completion method (BIMC) in comparison to several baselines.

Method	PRC@10	RCL@10	AUC
Global	1.01%	4.70%	0.8626
SVD	1.24%	4.82%	0.8479
MC	1.19%	4.53%	0.8464
Katz	1.33%	5.69%	0.9125
CMC	0.46%	1.81%	0.8932
IMC	2.85%	10.38%	0.8953
BIMC	**3.12%**	**11.32%**	**0.9129**

Performance Comparison

It is very interesting to see in Table 5.1 that the simple Global method outperforms both SVD and MC baselines in terms of AUC. This can be explained by the facts that most users follow highly popular blogs such as institutions or celebrities [16] and that both SVD and MC suffer from data sparsity. Yet, it is important to note that Global is outperformed by both SVD and MC for precision and recall results. Table 5.1 also shows that our proposed method, BIMC, achieves the best performance in all three evaluation metrics out of all methods. Note that the two best performing methods, BIMC and IMC, both utilize side information of users and blogs. This implies that such information is crucial to improve the recommendation quality.

In contrast, CMC performs the worst in the top-k list as seen in Table 5.1 showing that there does not exist significant shared low-rank structure between the follower graph and user/item features. Moreover, textual features in Tumblr are extremely sparse and noisy, which makes the CMC formulation more problematic. This clearly demonstrates that BIMC and IMC incorporate user/item features more effectively in a robust manner. Nonetheless, CMC still achieves similar AUC results compared with IMC. Katz performs comparably to BIMC in terms of AUC, but not in the top-k list, which can also be explained by the fact that similarities in S_u and S_b are affected by noisy features. In sum, we can see that BIMC achieves superior performance than other methods by successfully incorporating rich user and blog features.

Temporal Evaluation

Although cross-validation is a widely accepted evaluation methodology, it can produce biased results when temporal effects are not considered when splitting the data into training and testing sets. Thus, we evaluated all methods using another dataset divided into training and testing using a fixed date and time. Specifically, we use data from a 4-month period as training and the following 5th month data as testing. This evaluation is more similar than the offline-evaluation to A/B testing, which is broadly used in industry. Results for the temporal evaluation is given in Table 5.2, in which we can observe very similar results to the offline case in Table 5.1. This suggests that our proposed methods would also perform well in production settings.

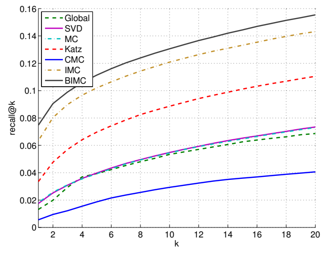
Performance for Users and Items with Different Levels of Sparsity

In order to better understand the effect of utilizing rich information about users and items in BIMC, we divide users as well as blogs into different categories based on the number of people they follow and the number of people who follow them. In other words, user and blog segmentation is done based on the number of nonzero elements in A, to examine how the methods perform under different levels of sparsity in the data. Specifically, users are partitioned into three groups based on the number of followees n_f: $n_f \leq 40$ (Low), $40 < n_f \leq 100$ (Medium) and $n_f > 100$ (High), where each group consists of about 89.36%, 7.81% and 2.83% of users. Similarly, we also partition the blog dimension (with the same thresholds), where each group consists of about 95.12%, 3.26% and 1.63% blogs respectively.

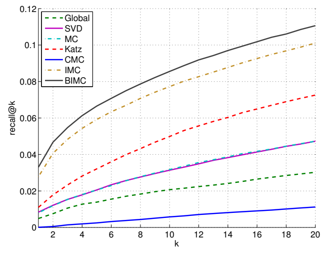
Table 5.3: AUC results for different groups of users of the proposed Boosted Inductive Matrix Completion method (BIMC) in comparison to several baselines.

Method	Low	Medium	High
Global	0.8639	0.8623	0.8403
SVD	0.8488	0.8782	0.8715
MC	0.8473	0.8767	0.8709
Katz	0.9199	0.9304	0.9194
CMC	0.8978	0.9063	0.9058
IMC	0.9040	0.9299	0.9162
BIMC	**0.9209**	**0.9316**	**0.9198**

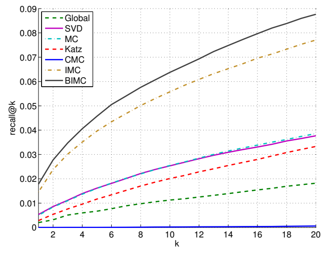
For each user category, we present Recall@k in Figure 5.5 for $k = 1, 2, \cdots, 20$. As shown in Figure 5.5, BIMC outperforms all other baselines for all user groups in terms of Recall@k. The second best method is IMC followed by Katz. This explicitly shows that utilizing both user and item features significantly helps in dealing with different sparsity conditions including cold-start. For the Low user group in Figure 5.5, it is interesting to see that SVD and MC suffers from severe sparsity and therefore perform comparably with the Global baseline. Note that the performance decreases for all methods as we move from Low to High user groups. This can be explained by the facts that users who already follow many popular blogs would need to be recommended more diverse blogs in the long-tail, which is generally a much harder task. Table 5.3 presents AUC results on all three user groups, where we can see that BIMC achieves the largest AUC values across all user categories. As discussed in Section 5.4.2, CMC is not able to make any good predictions in the top-k list, but still achieves reasonable AUC levels as shown in Table 5.3. This set of



(a) Users with low activity

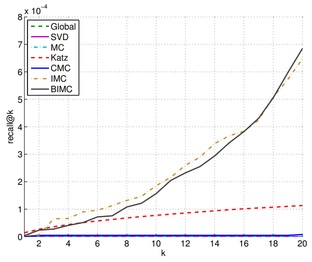


(b) Users with medium activity

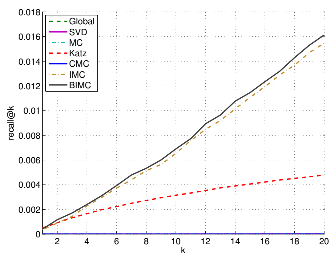


(c) Users with high activity

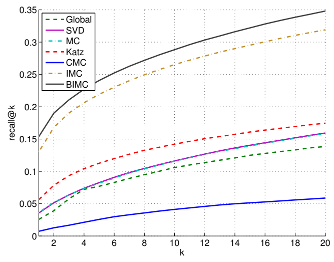
Figure 5.5: Recall@k results for user groups with different activity levels (low/medium/high) the proposed Boosted Inductive Matrix Completion method (i.e., BIMC) in comparison to several baselines ($k = 1, 2, \cdots, 20$).



(a) Blogs with low popularity



(b) Blogs with medium popularity



(c) Blogs with high popularity

Figure 5.6: Recall@k results for blog groups with different popularity levels (low/medium/high) of the proposed Boosted Inductive Matrix Completion method (i.e., BIMC) in comparison to several baselines ($k = 1, 2, \cdots, 20$).

results shows that BIMC successfully handles data sparsity by incorporating rich user and blog features, and significantly improves over other methods.

Next, we analyze the performance of all methods for blog groups with different popularity levels as shown in Figure 5.6, where we can observe similar trends as found in Figure 5.5 with different user groups. It can be seen in Figure 5.6 that BIMC and IMC outperforms all other baselines for all blog groups in terms of Recall@k, while all other baselines suffer from data sparsity, and cannot make any correct retrieval. For blogs with high popularity, their performances are much better. For this group, SVD and MC perform slightly better than the Global, but still much worse than BIMC and IMC. Another set of interesting results is the fact that IMC performs comparably with BIMC for items with low and medium popularity. This can be explained by the fact that the MC step in BIMC is suffering from data sparsity in both of these cases, and is not helping the BIMC as much as in the case of items with high popularity.

Finally, we analyze the performance of all methods for user groups and item groups jointly. Specifically, Figures 5.7 and 5.8 show the performances of all methods for all user and item groups jointly in terms of Precision@k and Recall@k respectively. It can be observed that both precision and recall increases significantly for users and items with high activity/popularity. For users with low and medium activity, all methods other than BIMC and IMC severely suffer from data sparsity. For users with high activity, BIMC outperforms IMC, both of which significantly outperform other baselines. Another interesting result is the fact that IMC outperforms or performs comparably with BIMC for user and items with low activity/popularity, showing that the the MC step in BIMC suffers from data sparsity and cannot effectively help IMC, in which case all prediction from BIMC depends on IMC step alone. Overall, this set of experiments clearly demonstrate the power of BIMC over IMC, as well as utilizing rich set of user and item features in BIMC and IMC over other baselines.

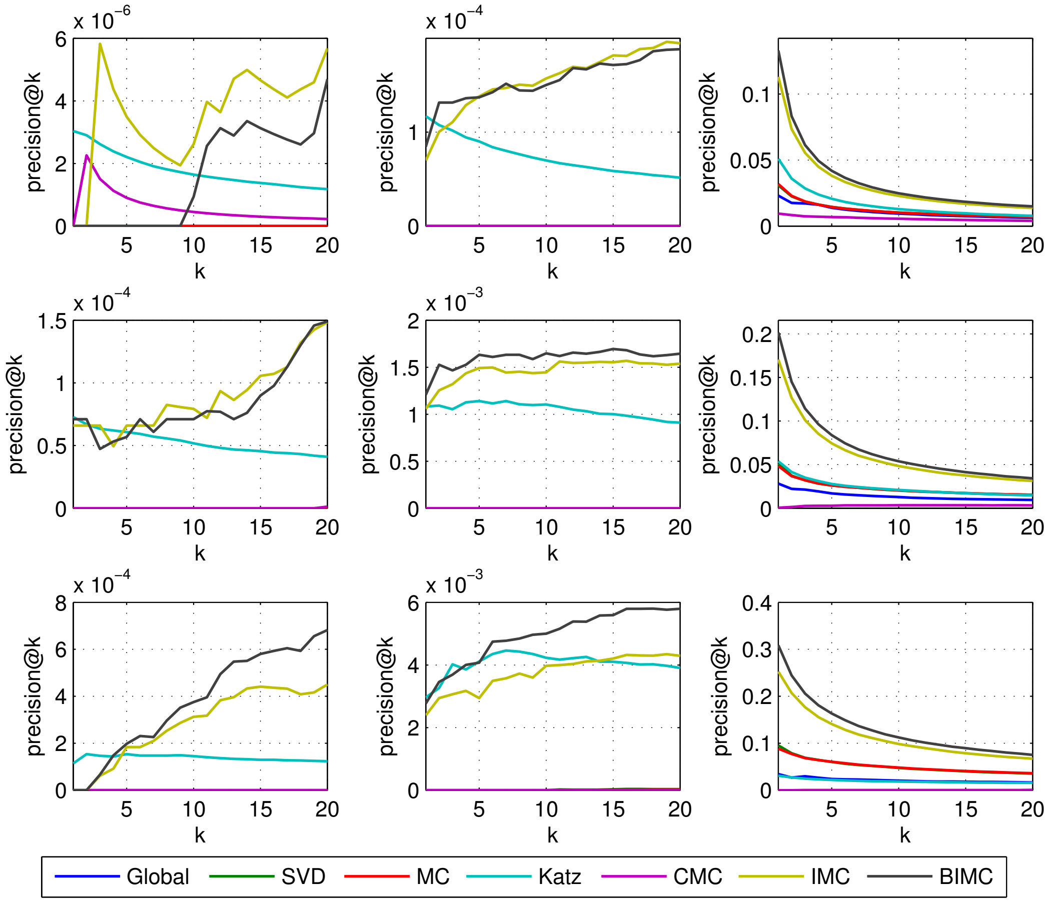


Figure 5.7: Precision at top-k results for different user and blog groups. Both users and blogs are divided into three groups (n_d: the number of links): (1) $n_d \leq 40$, (2) $40 < n_d \leq 100$, and (3) $n_d > 100$. From left to right are blog groups 1, 2 and 3, and from top to bottom are user groups 1, 2 and 3. BIMC outperforms other methods in most user-blog group combinations.

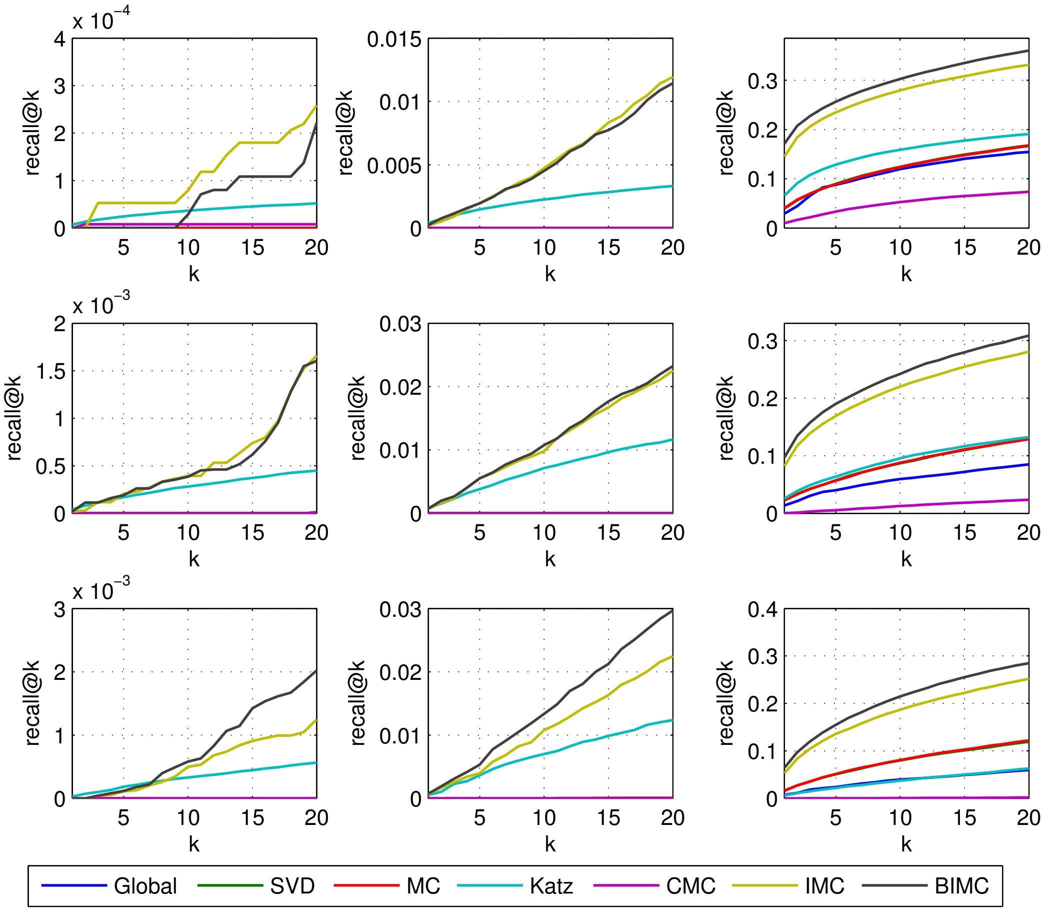


Figure 5.8: Recall at top-k results for different user and blog groups. Both users and blogs are divided into three groups (n_d: the number of links): (1) $n_d \leq 40$, (2) $40 < n_d \leq 100$, and (3) $n_d > 100$. From left to right are blog groups 1, 2 and 3, and from top to bottom are user groups 1, 2 and 3. BIMC outperforms other methods in most user-blog group combinations.

5.5 Conclusions

Recommending blogs to follow is one of the core tasks for online microblogging sites such as Tumblr for improving user engagement as well as advertising revenue. In this chapter, we propose a novel boosted inductive matrix completion (BIMC) model for the task that combines the power of an inductive matrix completion model together with a standard matrix completion model. The proposed BIMC model focuses on the residual matrix that is calculated from the approximation matrix of a standard matrix completion (MC) model, and learns an inductive matrix completion model (IMC) to effectively utilize the rich side information of users and blogs to learn the missing links in the follower graph where a standard MC fails to learn. We utilize state-of-the art deep learning methods such as word2vec and convolutional neural networks to extract a comprehensive set of features. An extensive set of experiments conducted on large-scale real-world data from Tumblr demonstrate the effectiveness of the proposed BIMC over MC and IMC methods as well as several other baselines.

Appendix

Appendix 1

Appendix

1.1 Proof of Theorem 2.2.1

Proof. The proof is based on the $\sin\theta$ theorem in [23]. Let the eigenvectors of the $n \times n$ symmetric matrices D and $D+\Delta$ be $E = [E_0|E_1]$ and $F = [F_0|F_1]$, respectively, where $E_0, F_0 \in \mathbb{R}^{n\times k}$ and $E_1, F_1 \in \mathbb{R}^{n\times(n-k)}$. Then

$$\begin{aligned}
D &= E \begin{bmatrix} \Lambda_0^D & 0 \\ 0 & \Lambda_1^D \end{bmatrix} E^T, \\
D+\Delta &= F \begin{bmatrix} \Lambda_0^{D+\Delta} & 0 \\ 0 & \Lambda_1^{D+\Delta} \end{bmatrix} F^T, \\
E_0^T F_0 &= U \cos\Theta V^T,
\end{aligned}$$

where Θ are the principal angles between E_0 and F_0. Assume that $\Lambda_0^D \subseteq [a,b]$ and $\Lambda_1^{D+\Delta} \subseteq (-\infty, a-\delta) \cup (b+\delta, \infty)$. Then

$$\|F_1^T E_0\| = \|F_0^T E_1\| = \|\sin\Theta\|.$$

Assume that the eigenvalues Λ_0^D lie in some interval, while the eigenvalues $\Lambda_1^{D+\Delta}$ lie in some distance $\eta \geq 0$ from that interval (possibly on both sides of it). Then

$$\begin{aligned}
\|\sin(\Theta(E_0,F_0))\|_2 &\leq \frac{\|\Delta E_0\|_2}{\eta} \leq \frac{\|\Delta\|_2}{\eta}, \\
\|\sin(\Theta(E_0,F_0))\|_F &\leq \frac{\|\Delta E_0\|_F}{\eta} \leq \sqrt{k}\frac{\|\Delta\|_F}{\eta}.
\end{aligned}$$

We can set $\eta = \min|\lambda - \hat{\lambda}|$ with $\lambda \in \Lambda_0^D$ and $\hat{\lambda} \in \Lambda_1^{D+\Delta}$. $\square$

1.2 Proof of Theorem 2.2.3

Proof. According to Lemma 1.2.1,

$$\|A - \bar{U}_k \bar{\Lambda}_k \bar{U}_k^T\|_2 = \|A - QQ^T AQQ^T\|_2 \leq 2\|A - QQ^T A\|_2.$$

where Q is an orthogonal basis for the Krylov subspace $[\Omega, A\Omega, \cdots, A^q A\Omega]$.

First, let $Z = [\Omega, A\Omega, \cdots, A^q A\Omega]$, $Q_B Q_B^T$ is a projector for $B = A^{q+1}$ and $Q_Z Q_Z^T$ is a projector for Z. Since we know that

$$\text{range}(A^q A\Omega) \subset \text{range}([\Omega, A\Omega, \cdots, A^q A\Omega]),$$

by Lemma 1.2.4, we have

$$\|(I - Q_Z Q_Z^T)A\| \leq \|(I - Q_B Q_B^T)A\|.$$

Furthermore, by Lemma 1.2.2, we have

$$\|(I - Q_B Q_B^T)A\| \leq \|(I - Q_B Q_B^T)B\|^{\frac{1}{(q+1)}}.$$

Next, we need to bound $\|(I - Q_B Q_B^T)B\|^{\frac{1}{(q+1)}}$. According to Lemma 1.2.3, we have

$$\begin{aligned}
\|(I - Q_B Q_B^T)B\|^{\frac{1}{q+1}} &\leq (\|\Lambda_2^B\|^2 + \|\Lambda_2^B \Omega_2^B (\Omega_1^B)^\dagger\|^2)^{\frac{1}{2(q+1)}} \\
&\leq (\|\Lambda_2^B\|^2 (1 + \|\Omega_2^B\|^2 \|(\Omega_1^B)^\dagger\|^2))^{\frac{1}{2(q+1)}} \\
&= \sigma_{k+1}(1 + \|\Omega_2^B\|^2 \|(\Omega_1^B)^\dagger\|^2))^{\frac{1}{2(q+1)}},
\end{aligned}$$

where σ_{k+1} is the $(k+1)$-th largest singular value of A.

Next, we show how to bound the error for both $\|\Omega_2^B\|$ and $\|(\Omega_1^B)^\dagger\|$. We already know $\Omega_1 = U_1^T\Omega$, $\Omega_2 = U_2^T\Omega$ and $\Omega \leftarrow \mathrm{diag}(U_{k_1}^{(1)}, U_{k_2}^{(2)}, \ldots, U_{k_c}^{(c)})$, which shows that Ω is the top-k eigenvectors for the (unperturbed) matrix D. With Theorem 2.2.1, we can now bound $\|\Omega_2^B\|$ and $\|(\Omega_1^B)^\dagger\|$ as

$$\begin{aligned}
\|\Omega_2^B\| &= \|\sin\Theta\| \leq \frac{\|\Delta\|}{\eta},\\
\|(\Omega_1^B)^\dagger\| &= \|\frac{1}{\cos\Theta}\| = \frac{1}{\sqrt{1-\|\sin\Theta\|^2}} \leq \frac{1}{\sqrt{1-\frac{\|\Delta\|^2}{\eta^2}}}.
\end{aligned}$$

As a consequence, we have

$$\begin{aligned}
\|A - \bar{U}_k\bar{\Lambda}_k\bar{U}_k^T\|_2 &\leq 2\|A - Q_ZQ_Z^TA\|\\
&\leq 2\|(I - Q_BQ_B^T)A\|\\
&\leq 2\|(I - Q_BQ_B^T)B\|^{\frac{1}{(q+1)}}\\
&\leq 2\sigma_{k+1}\left(1 + \frac{\sin^2\theta}{1-\sin^2\theta}\right)^{\frac{1}{2(q+1)}},
\end{aligned}$$

where θ is the largest principal angle of Θ.

□

Lemmas used in the above proof are listed as follows [36]:

Lemma 1.2.1. *Suppose A is Hermitian and Q is an orthogonal basis, then*

$$\|A - QQ^*AQQ^*\| \leq 2\|A - QQ^*A\| = 2\|(I - QQ^*)A\|.$$

Lemma 1.2.2. *Let A be an $m \times m$ matrix and Ω be an $m \times \ell$ matrix. Fix a non-negative integer q, from $B = A^q A$, and compute the sample matrix $Z = B\Omega$. For an orthogonal basis Q for Z,*

$$\|(I - QQ^*)A\| \leq \|(I - QQ^*)B\|^{\frac{1}{q+1}},$$

where $\|\cdot\|$ represents unitary-invariant norm including the spectral norm and the Frobenius norm.

Lemma 1.2.3. *Let A be an $m \times n$ matrix with singular value decomposition*

$$A = U\Lambda V^* = U \begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix} \begin{bmatrix} V_1^* \\ V_2^* \end{bmatrix}$$

and fix $k \geq 0$, where the size of V_1 and V_2 are $n \times k$ and $n \times (n-k)$, respectively. Choose a test matrix Ω and construct the sample matrix $Y = A\Omega$. Let $\Omega_1 = V_1^*\Omega$ and $\Omega_2 = V_2^*\Omega$. Assuming Ω_1 and Ω_2 has full column rank, the approximation error satisfies

$$\|(I - P_Y)A\|^2 \leq \|\Lambda_2\|^2 + \|\Lambda_2\Omega_2\Omega_1^{\dagger}\|^2,$$

where $\|\cdot\|$ denotes either the spectral norm or the Frobenius norm and P_Y is the projector for Y.

Lemma 1.2.4. *Suppose range$(N) \subset$ range(M). Then, for a matrix A, it holds that $\|P_N A\| \leq \|P_M A\|$ and $\|(I - P_M)A\| \leq \|(I - P_N)A\|$, where P_N and P_M are projectors for range(N) and range(M), respectively.*

Bibliography

[1] Lada A. Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[3] 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*, pages 19–28, 2009.

[4] Gaurav Agarwal and David Kempe. Modularity-maximizing graph communities via mathematical programming. *European Physical Journal B*, 66(3):409–418, 2008.

[5] Marcelo G. Armentano, Daniela Godoy, and Analía A. Amandi. Followee recommendation based on text analysis of micro-blogging activity. *Information Systems*, 38(8):1116–1127, 2013.

[6] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the 4th*

ACM International Conference on Web Search and Data Mining, pages 635–644, 2011.

[7] James Baglama, Daniela Calvetti, and Lothar Reichel. IRBL: An implicitly restarted block-Lanczos method for large-scale hermitian eigenproblems. *SIAM Journal on Scientific Computing*, 24(5):1650–1677, 2003.

[8] Linas Baltrunas. *Context-aware collaborative filtering recommender systems*. PhD thesis, Free University of Bozen-Bolzano, 2011.

[9] Linas Baltrunas, Bernd Ludwig, and Francesco Ricci. Matrix factorization techniques for context aware recommendation. In *Proceedings of the 5th ACM Conference on Recommender Systems*, pages 301–304, 2011.

[10] Linas Baltrunas and Francesco Ricci. Context-based splitting of item ratings in collaborative filtering. In *Proceedings of the 3rd ACM Conference on Recommender Systems*, pages 245–248, 2009.

[11] Robert M. Bell and Yehuda Koren. Lessons from the Netflix Prize Challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75–79, 2007.

[12] Robert M. Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proceedings of the 7th IEEE International Conference on Data Mining*, pages 43–52, 2007.

[13] Francesco Bonchi, Pooya Esfandiar, David F. Gleich, Chen Greif, and Laks V. S. Lakshmanan. Fast matrix computations for pair-wise and column-wise commute times and Katz scores. *Internet Mathematics*, 8(1-2):73–112, 2012.

[14] Guillaume Bouchard, Dawei Yin, and Shengbo Guo. Convex collective matrix factorization. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*, pages 144–152, 2013.

[15] Emmanuel Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119, 2012.

[16] Yi Chang, Lei Tang, Yoshiyuki Inagaki, and Yan Liu. What is Tumblr: A statistical overview and comparison. *ACM SIGKDD Explorations Newsletter*, 16(1):21–29, 2014.

[17] Annie Chen. Context-aware collaborative filtering system: Predicting the user's preference in the ubiquitous computing environment. In *Proceedings of the 1st International Conference on Location and Context-Awareness*, pages 1110–1111, 2005.

[18] Yudong Chen, Srinadh Bhojanapalli, Sujay Sanghavi, and Rachel Ward. Coherent matrix completion. In *Proceedings of the 31st International Conference on Machine Learning*, pages 674–682, 2014.

[19] Aaron Clauset, Cristopher Moore, and Mark E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, 2008.

[20] Alan Kaylor Cline and Inderjit S. Dhillon. Computation of the singular value decomposition. In *Handbook of Linear Algebra*, chapter 45, pages 1–13. CRC Press, 2006.

[21] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-N recommendation tasks. In *Proceedings of the 4th ACM Conference on Recommender Systems*, pages 39–46, 2010.

[22] J. J. M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numerische Mathematik*, 36(2):177–195, 1980.

[23] Chandler Davis and William M. Kahan. The rotation of eigenvectors by a perturbation. III. *SIAM Journal on Numerical Analysis*, 7(1):1–46, 1970.

[24] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

[25] Ernesto Diaz-Aviles, Lucas Drumond, Lars Schmidt-Thieme, and Wolfgang Nejdl. Real-time top-N recommendation in social streams. In

Proceedings of the 6th ACM Conference on Recommender Systems, pages 59–66, 2012.

[26] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the 31st International Conference on Machine Learning*, pages 647–655, 2014.

[27] Paul Dourish. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1):19–30, 2004.

[28] Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data*, 5(2):10:1–10:27, 2011.

[29] Ernesto Estrada and Desmond J. Higham. Network properties revealed through matrix functions. *SIAM Review*, 52(4):696–714, 2010.

[30] Omar U. Florez and Lama Nachman. Deep learning of semantic word representations to implement a content-based recommender for the RecSys Challenge'14. In *Proceedings of the Semantic Web Evaluation Challenge: SemWebEval at ESWC'14*, 2014.

[31] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[32] Roger G. Grimes, John G. Lewis, and Horst D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 15(1):228–272, 1994.

[33] Suriya Gunasekar, Makoto Yamada, Dawei Yin, and Yi Chang. Consistent collective matrix completion under joint low rank structure. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, pages 306–314, 2015.

[34] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. WTF: The who to follow service at twitter. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 505–514, 2013.

[35] Ido Guy, Naama Zwerdling, Inbal Ronen, David Carmel, and Erel Uziel. Social media recommendation based on people and tags. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 194–201, 2010.

[36] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[37] John Hannon, Mike Bennett, and Barry Smyth. Recommending twitter users to follow using content and collaborative filtering approaches. In

Proceedings of the 4th ACM Conference on Recommender Systems, pages 199–206, 2010.

[38] Negar Hariri, Bamshad Mobasher, and Robin Burke. Context-aware music recommendation based on latent topic sequential patterns. In *Proceedings of the 6th ACM Conference on Recommender Systems*, pages 131–138, 2012.

[39] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *Proceedings of SDM'06 workshop on Link Analysis, Counter-terrorism and Security*, pages 556–559, 2006.

[40] Taher H. Haveliwala. Topic-sensitive PageRank: a context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):784–796, 2003.

[41] Balázs Hidasi and Domonkos Tikk. Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback. In *Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 67–82, 2012.

[42] Liangjie Hong, Aziz S. Doumith, and Brian D. Davison. Co-factorization machines: Modeling user interests and predicting individual decisions in twitter. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining*, pages 557–566, 2013.

[43] Rong Hu, Wanchun Dou, and Jianxun Liu. A context-aware collaborative filtering approach for service recommendation. In *Proceedings of the 2012 International Conference on Cloud and Service Computing*, pages 148–155, 2012.

[44] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining*, pages 263–272, 2008.

[45] Prateek Jain and Inderjit S. Dhillon. Provable inductive matrix completion. *CoRR*, abs/1306.0626, 2013.

[46] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing*, pages 665–674, 2013.

[47] Mohsen Jamali and Martin Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the 4th ACM Conference on Recommender Systems*, pages 135–142, 2010.

[48] Masayuki Karasuyama and Hiroshi Mamitsuka. Manifold-based similarity adaptation for label propagation. In *Advances in Neural Information Processing Systems 26*, pages 1547–1555, 2013.

[49] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the 4th ACM Conference on Recommender Systems*, pages 79–86, 2010.

[50] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[51] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[52] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[53] Raghunandan H. Keshavan, Andrea Montanari, and Sewoong Oh. Matrix completion from noisy entries. *Journal of Machine Learning Research*, 11:2057–2078, 2010.

[54] Younghoon Kim and Kyuseok Shim. TWITOBI: A recommendation system for twitter using probabilistic modeling. In *Proceedings of the 11th IEEE International Conference on Data Mining*, pages 340–349, 2011.

[55] Ioannis Konstas, Vassilios Stathopoulos, and Joemon M. Jose. On social networks and collaborative recommendation. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 195–202, 2009.

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

[57] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105, 2012.

[58] Cornelius Lanczos. An iterative method for the solution of the eigenvalue problem of linear differential and integral. *Journal of Research of the National Bureau of Standards*, 45:255–282, 1950.

[59] Rasmus M. Larsen. Lanczos bidiagonalization with partial reorthogonalization. Technical Report DAIMI PB-357, Aarhus University, 1998.

[60] Dominique LaSalle and George Karypis. Multi-threaded modularity based graph clustering using the multilevel paradigm. *Journal of Parallel and Distributed Computing*, 2014. To appear.

[61] Richard B. Lehoucq, Danny C. Sorensen, and Chao Yang. *ARPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1998.

[62] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *Proceedings of the 14th*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 462–470, 2008.

[63] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large letworks: natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[64] Asher Levi, Osnat Mokryn, Christophe Diot, and Nina Taft. Finding a needle in a haystack of reviews: Cold start context-based hotel recommender system. In *Proceedings of the 6th ACM Conference on Recommender Systems*, pages 115–122, 2012.

[65] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[66] Ren-Cang Li. Relative perturbation theory: II. eigenspace and singular subspace variations. *SIAM Journal on Matrix Analysis and Applications*, 20(2):471–492, 1998.

[67] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the 2003 ACM International Conference on Information and Knowledge Management*, pages 556–559, 2003.

[68] Ryan N. Lichtenwalter, Jake T. Lussier, and Nitesh V. Chawla. New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 243–252, 2010.

[69] Wei Liu, Junfeng He, and Shih-Fu Chang. Large graph construction for scalable semi-supervised learning. In *Proceedings of the 27th International Conference on Machine Learning*, pages 679–686, 2010.

[70] Hao Ma, Dengyong Zhou, Chao Liu, Michael R. Lyu, and Irwin King. Recommender systems with social regularization. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining*, pages 287–296, 2011.

[71] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *Proceedings of the 1st ACM Conference on Recommender Systems*, pages 17–24, 2007.

[72] Robert Meusel, Sebastiano Vigna, Oliver Lehmberg, and Christian Bizer. Graph structure in the web — revisited: A trick of the heavy tail. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, pages 427–432, 2014.

[73] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. 2013.

[74] Alan Mislove, Hema Swetha Koppula, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Growth of the Flickr social network. In *Proceedings of the 1st Workshop on Online Social Networks*, pages 25–30, 2008.

[75] Peter L. Montgomery. A block Lanczos algorithm for finding dependencies over GF(2). In *Proceedings of the 14th Annual International Conference on Theory and Application of Cryptographic Techniques*, pages 106–120, 1995.

[76] Yashar Moshfeghi, Benjamin Piwowarski, and Joemon M. Jose. Handling data sparsity in collaborative filtering using emotion and semantic based features. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 625–634, 2011.

[77] Nagarajan Natarajan and Inderjit S. Dhillon. Inductive matrix completion for predicting gene-disease associations. *Bioinformatics*, 30(12):i60–i68, 2014.

[78] Nagarajan Natarajan, Donghyuk Shin, and Inderjit S. Dhillon. Which app will you use next? collaborative filtering with interactional context. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 201–208, 2013.

[79] Mark E. J. Newman. Clustering and preferential attachment in growing networks. *Physical Review E*, 64(2):025102, 2001.

[80] Mark E. J. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46(5):323–351, 2005.

[81] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856, 2001.

[82] Aaron Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems 26*, pages 2643–2651, 2013.

[83] Dimitris Papailiopoulos, Ioannis Mitliagkas, Alexandros Dimakis, and Constantine Caramanis. Finding dense subgraphs via low-rank bilinear optimization. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1890–1898, 2014.

[84] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, 1980.

[85] Rob Patro and Carl Kingsford. Global network alignment using multi-scale spectral signatures. *Bioinformatics*, 28(23):3105–3114, 2012.

[86] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. Trust management for the semantic web. In *Proceedings of the 2nd International Semantic Web Conference*, pages 351–368, 2003.

[87] Yousef Saad. On the rates of convergence of the Lanczos and the block-Lanczos methods. *SIAM Journal on Numerical Analysis*, 17(5):687–706, 1980.

[88] 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*, pages 285–295, 2001.

[89] Berkant Savas and Inderjit S. Dhillon. Clustered low rank approximation of graphs in information science applications. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, pages 164–175, 2011.

[90] Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. LiveLab: Measuring wireless networks and smartphone users in the field. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):15–20, 2011.

[91] Yue Shi, Martha Larson, and Alan Hanjalic. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys*, 47(1):3:1–3:45, 2014.

[92] Donghyuk Shin, Suleyman Cetintas, and Kuang-Chih Lee. Recommending Tumblr blogs to follow with inductive matrix completion. In *Poster Proceedings of the 8th ACM Conference on Recommender Systems*, 2014.

[93] Donghyuk Shin, Suleyman Cetintas, Kuang-Chih Lee, and Inderjit S. Dhillon. Tumblr blog recommendation with boosted inductive matrix completion. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 203–212, 2015.

[94] Donghyuk Shin, Si Si, and Inderjit S. Dhillon. Multi-scale link prediction. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, pages 215–224, 2012.

[95] Si Si, Donghyuk Shin, Inderjit S. Dhillon, and Beresford N. Parlett. Multi-scale spectral decomposition of massive graphs. In *Advances in Neural Information Processing Systems 27*, pages 2798–2806, 2014.

[96] Ajit P Singh and Geoffrey J Gordon. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 650–658, 2008.

[97] Han H. Song, Tae W. Cho, Vacha Dave, Yin Zhang, and Lili Qiu. Scalable proximity estimation and link prediction in online social networks. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, pages 322–335, 2009.

[98] Han H. Song, Berkant Savas, Tae W. Cho, Vacha Dave, Zhengdong Lu, Inderjit S. Dhillon, Yin Zhang, and Lili Qiu. Clustered embedding of massive social networks. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):331–342, 2012.

[99] Lloyd N. Trefethen and David Bau, III. *Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, 1997.

[100] Vishvas Vasuki, Nagarajan Natarajan, Zhengdong Lu, Berkant Savas, and Inderjit S. Dhillon. Scalable affiliation recommendation using auxiliary networks. *ACM Transactions on Intelligent Systems and Technology*, 3(1):1–20, 2011.

[101] Bo Wang, Zhuowen Tu, and John K. Tsotsos. Dynamic label propagation for semi-supervised multi-class multi-label classification. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, pages 425–432, 2013.

[102] Chao Wang, Venu Satuluri, and Srinivasan Parthasarathy. Local probabilistic models for link prediction. In *Proceedings of the 7th IEEE International Conference on Data Mining*, pages 322–331, 2007.

[103] Jason Weston, Sumit Chopra, and Keith Adams. #TagSpace: Semantic embeddings from hashtags. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1822–1827, 2014.

[104] Joyce J. Whang, Xin Sui, and Inderjit S. Dhillon. Scalable and memory-efficient clustering of large-scale social networks. In *Proceedings of the 12th IEEE International Conference on Data Mining*, pages 705–714, 2012.

[105] Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, pages 113–126, 2012.

[106] Diyi Yang, Tianqi Chen, Weinan Zhang, Qiuxia Lu, and Yong Yu. Local implicit feedback mining for music recommendation. In *Proceedings of the 6th ACM Conference on Recommender Systems*, pages 91–98, 2012.

[107] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the 12th IEEE International Conference on Data Mining*, pages 745–754, 2012.

[108] Dawei Yin, Liangjie Hong, and Brian D. Davison. Structural link analysis and prediction in microblogs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 1163–1168, 2011.

[109] Hsiang-Fu Yu, Prateek Jain, Purushottam Kar, and Inderjit S. Dhillon. Large-scale multi-label learning with missing labels. In *Proceedings of the 31st International Conference on Machine Learning*, pages 593–601, 2014.

[110] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

[111] Gang Zhao, Mong Li Lee, Wynne Hsu, Wei Chen, and Haoji Hu. Community-based user recommendation in uni-directional social networks. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pages 189–198, 2013.

[112] Yi Zhen, Wu-Jun Li, and Dit-Yan Yeung. TagiCoFi: Tag informed collaborative filtering. In *Proceedings of the 3rd ACM Conference on Recommender Systems*, pages 69–76, 2009.

[113] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems 17*, pages 321–328, 2004.