MODELING TEMPORAL STRUCTURES IN TIME-VARYING NETWORKS

A Dissertation Presented

by

KUN TU

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Feb 2019

College of Information and Computer Science

© Copyright by Kun Tu 2018

All Rights Reserved

MODELING TEMPORAL STRUCTURES IN TIME-VARYING NETWORKS

A Dissertation Presented

by

KUN TU

Approved as to style and content by:

Don Towsley, Chair

Ananthram Swami, Member

Weibo Gong, Member

David Jensen, Member

Ramesh Sitaraman, Member

James Allan, Chair of the Faculty
College of Information and Computer Science

ACKNOWLEDGMENTS

First and foremost, I feel lucky to have Prof. Don Towsley as my advisor. In the past few years, I gained a lot under Don's guidance and supervision. Don, thank you for inspiring me to work on so many interesting problems, for giving detailed suggestions on doing research and writing, for providing so many opportunities to work with different researchers in different fields.

I also thank my thesis committee, Dr. Ananthram Swami, Prof. Weibo Gong, Prof. David Jensen, and Prof. Ramesh Sitaraman. Ananthram has given so much help and advice during my stay in ARL. Weibo has raised many exciting topics in our group meetings. David has provided helpful advice in my research. I also gain a lot when working with Ramesh as a TA of the algorithm course. Most importantly, I would thank them for their feedback on my proposal.

It was a great pleasure to join the Computer Network Research Group at the College of Information and Computer Science. I want to thank Dr. Jian Li and Prof. Bruno Ribeiro for their help during our collaborations. I am grateful to other group members for giving enjoyable experience in UMass: Bo Jiang, Fabricio Ferreira, James Atwood, Yung-Chih Chen, Yeon-sup Lim, Sookhyun Yang, Amir Ramtin, Gayane Vardoyan, Stefan Dernbach, Arman Kabir, Nitish Panigrahy. I would also like to thank my other co-authors, collaborators and colleagues in different teams, who have made the research experience far more enjoyable.

Last but not least, I would like to express my deepest gratitude to my parents and my wife for their constant love and support.

ABSTRACT

MODELING TEMPORAL STRUCTURES IN TIME-VARYING NETWORKS

FEB 2019

KUN TU

B.E., SOUTH CHINA UNIVERSITY OF TECHNOLOGY

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Don Towsley

A dynamic network is a network whose structure changes because of the emergence and disappearance of node or edges. It can be used to study complex systems where individuals in a system are represented as nodes and their relations/interactions are represented as edges. Studying dynamic network structures helps to better understand changes in relationships. Considerable work has been conducted on learning network structure. However, due to the complexity of dynamic networks, there is considerable room for improvement to obtain better analysis results. This thesis studies different aspects of characteristic and dynamics of a network, focusing on their application in link prediction between nodes, temporal community detection and network representation.

In the first part of the thesis, we study bipartite networks constructed from online dating website data where nodes represent users and edges represent user interaction such as the exchange of invitation messages. We first formulate the prediction of future interaction between users as a link prediction problem, then propose a latent Dirichlet allocation (LDA) method to model user preferences and predict edges such that a recommendation system is built to recommend potential partners for a user. We find that user preferences changes over time and our method can adapt to these changes and outperforms baseline methods.

In the second part of the thesis, we consider more general dynamic networks and model the changes in similarities between nodes over time. We present network generative models using these similarities to detect communities and their lifetime. We present a low-rank tensor decomposition technique to learn the generative models. We show that our model is robust to the change in time granularity of network during analysis and has the best performance compared to baseline methods.

Finally, the last contribution of the thesis focuses on network graphlets, non-isomorphic subgraphs that represent node connection patterns in a network. We compute the significance of the graphlets by comparing the graphlet counts in an empirical network to random graphs and use this significance as feature representations for networks to analyze and characterize directed networks. Experiments show that our approach for network representation can significantly improve the accuracy on the-state-of-the-arts in network classification problem such as identifying departments in an email-exchange network or detect mobile users given their app-switching behavior represented as temporal networks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

A network, where nodes represent interacting individuals and edges represent relations or interactions between individuals, is a useful tool to study complex systems such as partnership in online dating websites, friendship in social networks, email communication between people or switching between mobile apps observed from user behaviors. These relations and interactions often change over time. These changes can modeled by temporal networks, which include timestamps on edges to store and represent temporal information for relations between individuals. This temporal information can be used to study the dynamics of complex systems.

Importantly, temporal networks representing real-world relations often display repeated patterns in the way that nodes are connected and suggest special network structures. Studying these network structures as well as changes in structure has become a fundamental and highly relevant endeavor in network science. First, it helps to understand structural properties of networks such as topologies and node degrees. This further allows inference of special relations among a group of nodes that may not be easily observed (e.g., communities). Second, it allows inference of the dynamics of these relations so that their changes are predictable. Application of the knowledge to real-world problems has triggered a wide range of studies on network structures, several of which are the focus of this thesis.

One topic of study focuses on relation or interaction between individuals in a complex system. The fundamental task of this topic is to understand and model the connectivity between nodes or the existence of edges. Researcher usually formalize this task as a link prediction problem: future links between nodes are predicted based on the assumption that node attributes (whether they are observable or hidden) or network structural information (such as node degree) have strong correlations with the existence of an edge. Real-world applications includes learning preferences of customers towards products or relations between users on social network to build recommendation systems.

Another topic focuses on properties of groups of individuals in complex systems, such as their composition, formation, and dissolution. When a temporal network is constructed to represent a real-world complex system, some individuals are observed to be more densely connected as a group than individuals across groups (if the edges are weighted, the weights of edges within groups are much high than across groups). These groups are called network communities. Many studies show that members of a network community usually share similarities in some of their attributes. For example, community members who have frequent email exchanges in an email network may be of the same team. There exists considerable work formalizing this problem as community detection (or clustering) and proposing different techniques to detect network communities[20, 60, 78].

A third topic related to network structure focuses on connectivity patterns of nodes in small non-isomorphic subgraphs, namely graphlets. Recent research has discovered that distributions of graphlets are similar for networks constructed from the same network domain (e.g., email networks, question answering networks). This has triggered considerable effort to study the relationship between graphlets and networks structures. For instance, the difference in graphlet counts between a real-world network and random graphs helps to characterize network structures and is useful for practical problems such as network embedding and network classification.

This thesis explores the three topics mentioned above based on various networks to deepen our understanding of temporal network structures and the application to real-world problems.

In Chapter 3, we learn the evolving topological structure of a bipartite network, under the assumption that the existence of an edge depends on node attributes. We formalize this as a link prediction problem. Solving this problem helps us to understand user behaviors in an online dating website, where we can observe users' profiles and their interactions. We represent users as nodes and their interactions as directed edges. We use "information gain ratio" from a decision tree to select node attributes that are relevant to the emergence of edges and construct a feature space from the attributes. We propose a latent Dirichlet allocation (LDA) method to model relations between edges and the feature space, and then use these relations to predict future edges. The LDA model is easily updated with new data.

In Chapter 4, we shift our focus from bipartite networks to more general complex networks. These networks are time-varying networks with a fixed set of nodes and edges changing over time. We present a temporal clustering framework to model these networks as a mixture of generative models and apply tensor decomposition (TD) to learn them. Further, we identify network communities from those models with a K-means algorithm and provide a bottom-up time series segmentation algorithm to obtain the lifespans of detected clusters.

In Chapter 5, we study the feature representation for directed networks (also called network embedding) using both static and temporal graphlets. More specifically, we use subgraph ratio profiles (SRPs), a measure of the difference in graphlet counts between a real-world network and random graphs, as a feature vector to represent a directed graph and investigate the effect of features created by static/temporal graphlets on network classification problem. Experiments on a wide range of real-world networks show that features from both static and temporal graphlets are important to characterize network structures because a combination of these features yields the best network classification accuracy.

1.1 Thesis Contribution

This thesis makes the following contributions.

- We propose a novel method using latent Dirichlet allocation (LDA) to compute the probability of node connectivity in a network to solve a link prediction problem. When this method is applied to online dating recommendation, the feature vector can be interpreted as a user preference to his/her partner. We show that user preferences learned by our method can be updated quickly with new incoming data and adapts to user changing behavior. Results from experiments on an online dating dataset show that our model has better performance than baseline methods such as K-nearest neighbor (a popular method in recommendation systems) in predicting user invitation sending and accepting. Compared to a model using stated preferences, our model adapts to changes in user behavior and improves the prediction of matching of users over time.

- We present a temporal clustering framework based on edge-generation between nodes in a time-varying network to detect evolving communities. We represent a temporal network as a three-way tensor with the similarity of nodes defined as a function of time base on the edge generation rate learned from tensor decomposition. K-means clustering and Silhouette criterion are applied to detect network communities. Experimental results show that our method, compared with baseline methods, is more robust to change in a hyper-parameter of clustering methods: the number of clusters in the dataset. We propose a similarity ordering score to improve temporal clustering methods on a precision-recall metric for community detection and community member detection. We also apply filtering methods for smoothing the edge generation rate in a cluster and propose a bottom-up segmentation algorithm for detecting formation, dissolution, and lifetimes of communities. We provide evidence that the performance of our model is robust to changes in time granularity and density of network snapshots.

- We propose *gl2vec*, a novel, flexible and scalable feature representation method for characterizing network structure in both static and temporal directed networks. This feature representation is based on relative differences in graphlet counts between a real-world network and random graphs. We compare *gl2vec* with state-of-the-art methods on network classification problems such as network type (email network, question answering network) classification and subgraph identification in several real-world static and temporal datasets. We find that *gl2vec* outperforms state-of-the-art methods in these two tasks. We further show that concatenating the feature vector computed by *gl2vec* with those from state-of-the-art methods, produce significant improvements in classification accuracy in real-world applications. This indicates that *gl2vec* provides important features not captured by state-of-the-art methods. We investigate the impact of different random graph models on the performance of network classification in static directed networks and results show that all models achieve equally good performance.

The rest of this thesis is organized as follows: Chapter 2 introduces definitions and concepts that are used in the rest of this thesis. Chapter 3 presents an LDA model for link prediction to make recommendation for an online dating website. Chapter 4 provides temporal clustering methods to track evolving network communities in a temporal network. Chapter 5 investigates the improvement obtained by network embedding with static and temporal graphlet in network classification problem. We conclude this thesis in Chapter 6 and provide directions to extend our work in future.

CHAPTER 2
PRELIMINARIES

In this chapter, we introduce networks, null models, graphlets and other concepts we use in the rest of this thesis.

2.1 Networks

A network (or a graph) is a useful tool to represent relations or interactions between individuals in a complex system. In this section, we introduce different concepts related to networks.

2.1.1 Static Networks

Definition 1. *A* static network $G(V, E)$ *is defined as a set of nodes, denoted as* V *and a set of edges, denoted as* E*.*

The concept of edge is important for a network. There are several type of edges according to their properties.

- An *undirected edge* is an unordered pair of nodes in a network, denoted as (u, v) (or (v, u), or e_{uv}), where u, v are nodes in a network G.

- A *directed edge* in a network is an ordered pair of nodes, denoted as (u, v) or e_{uv}, where u, v are nodes in a network G. The ordered pair (u, v) represents a direction from node u to node v.

A static network can be categorized as different types according to the attributes of an edge.

- A *static undirected network* $G(V, E)$ is a static network whose edges are *undirected*.
- A *static directed network* $G(V, E)$ is a static network whose edges are *directed*.

2.1.2 Representation of Static Networks

Common representations of a static network includes:

1. *Edge list.* A static network $G(V, E)$ can be represented as an edge list, denoted as $\{(u_i, v_i)\}_{i=1}^{|E|}$, where $u_i, v_i \in V$. For undirected network, an edge is an unordered node pair and can be represented as either (u_i, v_i) or (v_i, u_i).
2. *Adjacency Matrix* A static network $G(V, E)$ can also be represented as a $|V| \times |V|$ binary matrix, denoted as M, where an element $M_{u,v} = 1$ in M represents that there is an edge from node u to v, otherwise $M_{u,v} = 0$. For an undirected network, M is a symmetric matrix (that is, $M_{u,v} = M_{v,u}$).

An edge-list representation requires less space to store or represent a static network if the network is sparse. On the other hand, an adjacency-matrix representation is convenient in computation for the network. The adjacency matrix can be extend to a weighted matrix whose element represents a numeric attribute of an edge.

2.1.3 Temporal Network

A temporal network in discrete time is used to represent a complex system that evolves over time.

Definition 2. *A* temporal network *(or time-varying network) $G(V, E, \mathcal{T})$ is a set of nodes, denoted by V, and a collection of edges, denoted by E, with a timestamp $t \in \mathcal{T}$ on each edge. We refer to a tuple (u, v, t) as a* temporal edge*, where $u, v \in V$ and $t \in \mathcal{T}$, t is a timestamps representing the time steps when an edge exists. In this thesis, we focus on temporal networks represented in discrete time. The timestamp of an edge represent a time step when the edge exists.*

- A *temporal undirected edge* (u, v, t) is a temporal edge, where nodes $u, v \in V$ is an unordered pair.

- A *temporal directed edge* (u, v, t) is a temporal edge, where nodes $u, v \in V$ is an ordered pair.

Similar to static networks, temporal networks can be categorized as follows:

- A *temporal undirected network* (u, v, t) is a temporal edge, where nodes $u, v \in V$ is an unordered pair.

- A *temporal directed netowrk* (u, v, t) is a temporal edge, where nodes $u, v \in V$ is an ordered pair. A temporal edge $(u, v, t) \neq (v, u, t)$, if $u \neq v$.

2.1.4 Representation of Temporal Networks

There are two representations for a temporal network.The first representation is a temporal-edge list, denoted as $\{(u_i, v_i, t_i)\}_{i=1}^{|E|}$, where $u_i, v_i \in V$ and $t_i \in \mathcal{T}$ is a timestamp.

The second representation is a set of *Network snapshots* (a series of static networks), denoted as $\{G_t(V, E_t)\}_{t=1}^{T}$, where V is a set of nodes and E_t is a set of edges at time step t, T is the total number of time steps (timestamps of edges are represented as time step t). This representation is preferred in analyzing temporal network in discrete time at different time granularities.

We define *time granularity* w as the number of network snapshots to merge to construct a new static network. In this thesis, the t-th network snapshot in coarse time granularity is represented as a weighted matrix $M(w)$. The weight of an edge (u, v) is computed as

$$M_{uvt}(w) = \sum_{i=(t-1)w+1}^{tw} M_{uvi}(1)$$

2.1.5 Other Concepts Related to Networks

Definition 3. *A* self-loop *(or loop) is an edge that connects a node to itself.*

Definition 4. Multi-edges *in a network are two or more edges that are incident to the same pair of nodes.*

Definition 5. *A* simple graph *is a network that does not allow self-loops or multi-edges.*

Datasets studied in this thesis contain relations or interactions between different individuals of a complex system. As a result, networks created from these dataset do not have self-loops. We do not allow multi-edges and focus on simple graphs.

2.2 Null Models

A null model [72] is a generative model that generates random graphs that match the structural features of a specific graph, such as the degrees of nodes or number of nodes and edges. A null model is useful to determine if a network has a certain property value by chance by comparing this property to those in random graphs. In Chapter 5, the difference in network property are used to characterize a network.

For static networks, we consider three different null models, (i) ***NE*:** random graphs with the same number of nodes and edges; (ii) ***MAN*:** random graphs with the same numbers of (M)utual, (A)symmetric and (N)ull edges; and (iii) ***BDS*:** random graphs with the same in/out degree-pair sequence (also called bidegree sequence, BDS).

For temporal networks, since there is no equivalent null model, we consider ensembles of randomized time-shuffled data as a temporal null model [61]. To be more specific, we randomly permute the timestamps on the edges while keeping the node pairs fixed. This model breaks the temporal dependences between edges but preserves the network structure.

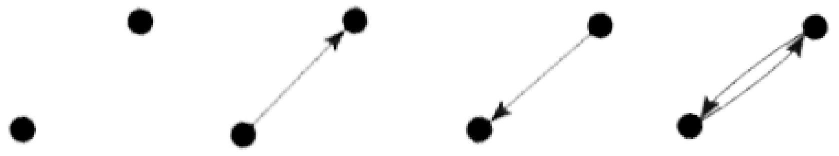


Figure 2.1: All four possible structures in a two-node static directed network.

2.3 Graphlets

Grahplets are small connected non-isomorphic induced subgraphs of a larger network. They are useful to study patterns of node connectivities in a network. Graphlets in static undirected networks have been applied in graph kernel [92] for network classifications. Recent research [62, 47] discovers that the distribution of graphlets in a directed network is strongly related to network domains.

In this thesis, we focus on graphlets in both static and temporal directed networks. In a static directed network, graphlets are categorized according to the number of nodes, such as dyad (graphlet of two nodes), triad (graphlet of three nodes), tetrad (graphlets of four nodes) and so on.

Let u, v be nodes in a dyad and $e_{u,v}$ be a directed edge from node u to v, there are four possible node connectivity types (Table 2.1), classified as three isomorphic dyad classes: null dyad (with no edges), asymmetric dyad (with only one directed edge) and mutual dyad (contain reciprocal edges).

Table 2.1: Four Possible Structures for node u, v and their edges

node connection type	1	2	3	4
directed edges	N/A	$e_{u,v}$	$e_{v,u}$	$e_{u,v}, e_{v,u}$
dyad class	null	asymmetric	asymmetric	mutual

Dyad motifs are basic structures to study other graphlet because properties of subgraphs can be derived from those of dyads.

In particular, we focus on triads in static networks (shown in Figure 2.2) because they provide more structural information than dyads and incur much less compu-

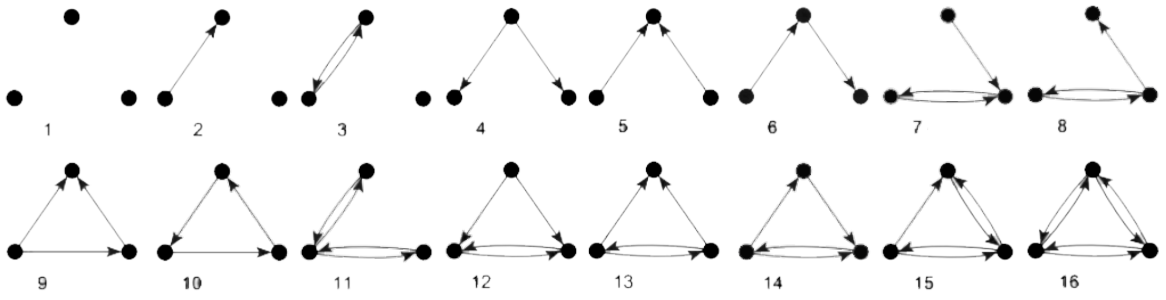


Figure 2.2: All 16 triads in static directed network.

tational complexity than tetrad. Note that the first three triads are not connected, hence do not satisfy the graphlet definition, but we argue that they should be included when constructing vectors for network feature representation.

Definition 6. *Temporal network graphlets are defined as induced subgraphs on sequences of temporal edges [79]. Formally, a k-node, l-edge, δ-temporal graphlet is a sequence of l edges, $M = (u_1, v_1, t_1), \cdots, (u_l, v_l, t_l)$ that are time-ordered within a duration δ, i.e., $t_1 < \cdots < t_l$ and $t_l - t_1 \leq \delta$, such that the induced static graph from edges is connected with k nodes.*

Note Paranjape et.al [79] refered to these as temporal motifs. However, *network motifs* are referred to as graphlets that occur significantly more frequently in a specific network than by chance. We rename the temporal motif to *temporal graphlets* for consistency. We consider all 2-node and 3-node, 3-edge, δ-temporal graphlets, as shown in Figure 2.3.

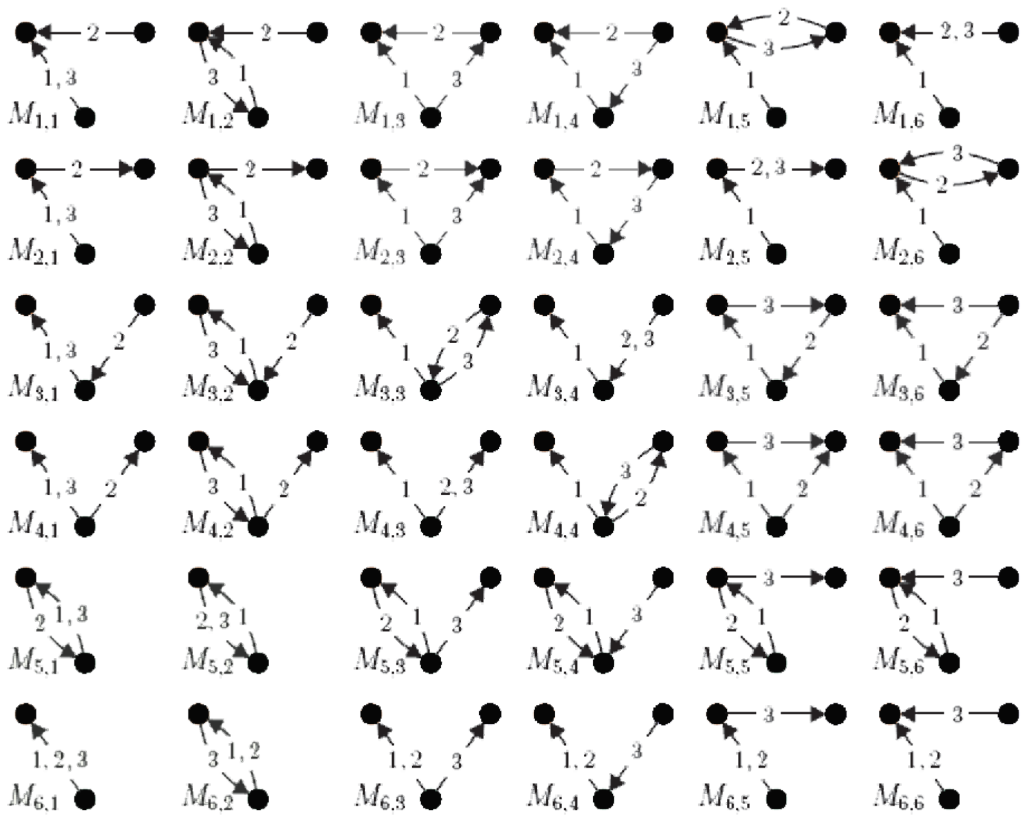


Figure 2.3: All 2-node and 3-node, 3-edge δ-temporal graphlets as defined in [79]. Edge labels correspond to the ordering of edges. All 36 graphlets are labeled with $M_{i,j}$ across 6 rows and 6 columns. The first edge in each graphlet is from the bottom to the left node. The second edge is the same along each row, and the third edge is the same along each column.

CHAPTER 3

LINK PREDICTION FOR RECOMMENDATION IN ONLINE DATING WEBSITE

3.1 Introduction

A bipartite network is a network whose nodes are split into two disjoint sets such that each edge in the network connects two nodes in both sets. Many relationships between real-world entities can be modeled with bipartite networks, e.g., relations between queries from users and documents retrieved by a search engine, partnerships between users from an online dating website and purchase behaviors of consumers at an online store. Studying a bipartite network and learning its evolving structure help us to not only understand but also predict future relationships. Unveiling the existence of edges between nodes or inferring edge weights is a fundamental problem in learning network structure.

We obtained a dataset from an heterosexual online dating website that records invitations sent/received by $200,000$ users registered in November 2011. The dataset contains $293,804$ user profiles (including users who did not register in November 2011) and two million invitations between November 2011 and January 31st 2012. We model the interactions of users by a directed bipartite network: male users and female users are considered nodes and divided into two sets; user profiles are node attributes and the behavior of sending/accepting an invitation is represented as a directed edge between a male and female user. Since accepting an invitation suggests a potential partnership, the problem of predicting a partnership between users according to their profiles is then equivalent to predicting future edges based on node attributes and existing edges in an evolving bipartite network.

Given the observable properties of a network and a problem objective, different approaches have been proposed for edge prediction. The first approach includes conventional classification methods such as logistic regression, decision trees, and random forest[13]. These methods label a directed edge as “exist” or “non-exist’ according to observable attributes of two nodes. However, collected node attributes may not be fully indicative of or directly related to the existence of an edge. As a result, accuracy of edge prediction by these methods is low.

To improve the prediction on the conventional method, a second approach creates new attributes from the observable ones. These new attributes are usually called hidden (latent) variables, and they are strongly correlated to the existence of edges. Neville et al.[68] proposed a latent group model to predict edges for nodes within the same group in a general network. Other works [106, 63, 83, 11] predict links within communities in social networks where edges represent exchanged text messages: Latent Dirichlet allocation (LDA) models [12] are used to learn the topics of those messages, and these topics serve as node attributes for community detection. Nodes exchanging messages of the same topic are likely to be in the same community and are expected to have links with one another in future. However, They rely on community detection to estimate the probabilities of links between nodes within or across communities. As a result, these methods do not apply to link prediction in bipartite networks, where nodes linked by an edge may be incomparable, and the relations between nodes are irrelevant to communities (it is less likely that a consumer belong to the same community as a product)

Another approach for predicting edges in a bipartite network is used in recommender systems, which are widely used in all kinds of online services such as shopping and dating [57, 53]. It considers one set of nodes as users and the other set as products. It uses “user preferences” (usually represented as a vector) to estimate the probability of an edge between nodes on two sides (interpreted as a user behavior such as buying a product or sending invitations for dating on an online dating website). User preferences are learned with the objective of maximizing the probability of all observed edges in the network. In machine learning, real-world data processed by recommender systems are usually imbalanced: only a small fraction of edges are observed in the bipartite network constructed from the data. This causes a problem in learning: there is an insufficient amount of data to learn a user's personal preference and some models are trained to simply classify all edges as "non-exist" so as to achieve high accuracy. A solution is "collaborative filtering" [90], which is widely applied and studied even nowadays. It assumes that some users have similar preferences on products and thus they have edges (representing behaviors such as "like" or "buy") linked to the same products. Users are grouped into clusters based on their preferences. Once some users in a cluster buy a product, other users in the same cluster are expected to perform similarly in future. In this case, an edge is predicted to exist from a user to a product. Xiang et al.[100] proposed a latent variable model for item recommendation based on user similarities. However, users within the same cluster in real-world data have unique preferences, and the difference between user preferences causes errors during prediction. Other methods [6, 15, 28, 51, 67, 81] model user preferences directly and learn them through the observed data.

These methods work well on datasets where user preferences remain unchanged and user profiles are accurate. According to our previous analysis of the data [99], there are several characteristics of user behaviors that make the edge prediction problem challenging and cause large errors in the previous methods:

- There is a significant discrepancy between a user's stated preference and his/her actual online dating behavior. Users send/reply to a large fraction of invitation messages that violate their stated preferences.
- User behavior changes over time. We investigate the fraction of reply messages that violate user stated preferences as a function of time (in units of weeks) and

find that this fraction increases each week. Most work on recommender systems assumes that user preferences are unchanged. This assumption is unsuitable in our scenario where users are observed to change their behavior. To adapt to this scenario, these systems need to frequently update their models with new data. However, updates of these models require time-consuming training processes.

- Our data is imbalanced. A user looks through profiles of other users before sending out invitations. We are unable to obtain information on the fraction of users who do not receive invitations but whose profiles were examined. Classification methods, such as logistic regression or random forest, need positive and negative instances during training. They are unable to learn and predict sender behavior in this dataset because there are no negative instances. Collaborative filtering methods for a recommendation system, such as K-nearest neighbor method, deal with imbalanced data using user ratings of items to learn user preferences. However, it is difficult to convert a sender behavior to receiver ratings.

- There is bias in user reply behavior. A large fraction of invitations are from senders to receivers where the sender profiles do not match receiver preferences. A receiver may choose to reply to invitations that are close to his/her true preference. In fact, invitation reply rate is low. The average reply rate of invitations fully matched to preferences is 10.82% for females and 20.17% for males.

- There is a considerable amount of missing data and inaccurate data in user profiles. For example, more than 50% of users provide no information on their city of residence and about 75% of females provide no information on their jobs. Moreover, some users set their ages below 10 or above 100. Some features in a user profile, such as age and Chinese zodiac, contain redundant information for edge prediction. This increases the computational complexity of a model, provides little improvement in prediction accuracy or even causes bias.

To handle these issues, we propose a latent Dirichlet allocation (LDA) method to explicitly model and learn T unique user preferences by using Gibbs sampling to cluster user behaviors. For each cluster of user behaviors, a categorical distribution is generated from these behaviors as a preference. We model each user as a mixture of T unique preferences. This mixture is considered as a user's "personal preference" and can differ from those of other users. A user's "personal preference" is learned by the probabilities of his/her past behaviors given the T preferences. Similar to collaborative filtering, our model can also deal with imbalanced data where users only send/reply to a small fraction of invitations. In summary, we make the following contribution in this chapter.

- We propose a latent Dirichlet allocation (LDA) model that learns user preferences that are unobservable in the datasets. Experiments with synthetic dataset show that user preferences learned from our model, measured by KL-divergence, is close to the ground truth user preferences.
- Our model updates evolving user preferences over time with new data. Experiments with real-world online dating dataset show that our model adapts to changes in user behavior and outperform baseline methods such as K-nearest neighbor method in predicting future user interactions with new data.

The rest of this chapter is organized as follows. We review related work in link prediction techniques for recommendation systems in Section 3.2. We then formulate recommendation for online dating as a link prediction problem in a bi-partite network in Section 3.3 and present a latent Dirichlet allocation model (LDA) to learn user preferences for link prediction in Section 3.4. We use both synthetic and real-world datasets to evaluate our model and compare it to baseline methods in Section 3.5 and then conclude this chapter in Section 3.6.

3.2 Related Work

Link prediction in a bipartite network can be formulated as a classification problem and conventional machine learning methods such as logistic regression [17] and random forest [13] have been applied to predict the existence of an edge based on observable node attributes. Although these models are fast and straightforward to implement, observed node attributes may not be directly correlated with the existence of an edge. This results in poor performances of these models. In order to improve the prediction performance, some models include new node attributes or hidden variables constructed from the observed attributes [100]. In recommendation systems, hidden variables called "user preferences" have been proposed to predict interactions between users and products [6, 15, 28, 51, 67, 81]. "Collaborative filtering" [90] assumes that users can be grouped according to the similarities in their behaviors and compute their preferences using linear regression [35]. However, these methods assume that user preferences remain unchanged. As a result, their predictions on user behaviors become less accurate as users change their preference over time.

It is worth noting that there is a large amount of work [106, 10, 63] using latent Dirichlet models (LDA) to predict links between nodes within a community in message exchange network, such as co-author network [59], Twitter [83], based on the content of the messages. However, these models rely on community detection [10] and estimate the probabilities of links between nodes within or across communities. As a result, they do not apply to applications in e-commerce (consumer-product) or two-sided matching problem (online dating) in bipartite networks, where nodes linked by an edge may be incomparable and the relations between nodes are irrelevant to communities (it is less likely that a consumer belongs to the same community as a product).

Our model differs from previously introduced LDA models in the following respects: First, our model focuses on bipartite networks. It is different from community-based link prediction methods using LDA. We do not predict links between nodes in the same community. More specifically, the two nodes linked by an edge in our problem belong to two disjoint sets and are allowed to be incomparable. Our LDA model learns user preferences from behaviors to make recommendations instead of learning topics from text messages to detect communities. Second, unlike previous recommender systems that assume that a cluster of users have the same preference, our model allows each user to have his/her own mixture of preferences, potentially resulting in a unique preference for each user. Third, previous methods model the similarity between users based on user profile features. This is inappropriate in our case because of the large fraction of missing data and inaccurate profiles in the dataset. Moreover, it is difficult to define a universal similarity function based on user profiles to fit in different user preferences. For example, many users have similar features in their profile, but they behave quite differently. To solve this issue, our model measures and learns the similarity between users based on their behaviors. For example, receivers who obtain invitations from the same group of senders are considered similar. We provide details on how to decide the group of senders in the next section.

3.3 Problem Formulation

In an online data website, there are D users interacting with each other by sending/replying dating invitations. Each user d has his/her profile, denoted by a feature vector $\vec{f_d} = [f_1, \cdots, f_K]$, where K is the number of features in a user profile, f_k is a categorical variable and n_k is the number of possible values for f_i for $1 \leq i \leq K$. Note that $\vec{f_d}$ is in a $n_1 \times n_2 \times \cdots \times n_K$ feature space. This feature space contains $W = n_1 n_2 \cdots n_K$ unique profiles. For simplicity, we use $w_d = k$ to represent that user d's profile $\vec{f_d}$ is the k-th profile, where $k \in \{1, \ldots, W\}$.

We define user preference as a categorical distribution $\phi_t = [\phi_{1|t}, \phi_{2|t}, \cdots, \phi_{W|t}]$, where $\phi_{k|t}$ is the probability of sending/accepting an invitation to/from a user of profile k, for $k = 1, \ldots, W$. We assume that there are T unique user preferences in the dataset and denote them as $\Phi = [\phi_1, \cdots, \phi_T]$. A user d is a mixture of the T preferences, denoted by a categorical distribution $\theta_d = [\theta_{1|d}, \cdots, \theta_{T|d}]$, where $\theta_{t|d} = P(\phi_t|d, G)$. We assume D users in the dataset and use $\Theta = [\theta_1, \ldots, \theta_D]$ to model user mixture preferences

We refer to a user who sends an invitation as a sender. A user who receives the invitation is a receiver. User d chooses preference ϕ_t with probability $\theta_{t|d}$. He then sends/accepts an invitation to/from a user (denoted as v) of the k-th profile with probability $\phi_{k|t}$. Thus the probability that sender d interact with receiver v is computed as

$$\sum_{t=1}^{T} \phi_{k|t}\theta_{t|d} \tag{3.1}$$

We model user interactions as a directed bipartite network, where nodes in the network represent users and directed edges represent that a user wants to date with another and send/accept an invitation (user interaction also called user behavior in recommendation systems). We want to predict a directed edge between existing nodes or new nodes in a growing directed bipartite network, denoted by $G(V_1, V_2, E)$, where V_1 and V_2 are two disjoint sets of nodes. $E \subseteq (V_1 \times V_2) \cup (V_2 \times V_1)$ is a set of directed edges connecting the nodes between V_1 and $V2$. Let $e_{dv} = 1$ if a directed edge exists from d to v and $e_{dv} = 0$ otherwise. We formulate link prediction as a problem of learning a probability model, denoted by $\mathcal{L} = \{\Phi, \Theta$, that maximize the probability of observed edges in G.

$$\begin{aligned}\max_{\mathcal{L}} \prod_{e_{uv} \in E} P(e_{uv}|\vec{f_u}, \vec{f_v}, G, \mathcal{L}) &= \max_{\mathcal{L}} \prod_{e_{uv} \in E} \sum_{t=1}^{T} \phi_{k|t} P(\phi_t|u = d, w_v = k, G, \mathcal{L}) \\ &= \max_{\mathcal{L}} \prod_{e_{uv} \in E} \sum_{t=1}^{T} \phi_{k|t}\theta_{t|d} \qquad (3.2)\end{aligned}$$

Since preference ϕ_t and receiver profile w_v are independent, $P(\phi_t|u = d, w_v = k, G, \mathcal{L}) = P(\phi_t|u = d, G, \mathcal{L}) = \theta_{t|d}$. In the rest of the chapter, we learn the probability model given observation from a specific bipartite network G, thus we remove G from the probability function for simplicity.

In the next section, we introduce a latent Dirichlet allocation (LDA) and apply Gibbs sampling to learn user preferences.

3.4 Latent Dirichlet Allocation for Link Prediction

In this section, we first introduce latent Dirichlet allocation (LDA) to model user preferncs and explain how to learn the preferences using Bayesian method. We then provide a formula for link prediction in the dataset.

3.4.1 Learning the Model

To learn Φ and Θ in the model, we assume that there are D users in the dataset. For simplicity, we only explain how LDA learns a sender's preference to send an invitation because the process is the same for modeling a receiver's preference to accept an invitation.

Figure 3.1 shows the graphical model of LDA for the online dataing dataset. There are N invitations from D senders in the dataset. $\mathcal{W} = [w_1, \ldots, w_N]$ in the shaded circle represents the identities of receiver profiles, where $w_i = k$ represents that the i-th invitation is sent to a receiver of profile k. Let $\mathbf{z} = [z_1, \cdots, z_N]$, where random variables z_i and z_j are independent for $i \neq j$. $z_i = t$ means that $w_i = k$ is generated with probability $\phi_{k|t}$. In another words, a user chooses preference $\phi_{z_i} = \phi_t$ to send an invitation. The prior of θ_d follows a Dirichilet distribution in (3.3). $\Phi = [\phi_1, \cdots, \phi_T]$ are the T preferences. The prior of ϕ_t is also a Dirichlet distribution, defined in (3.4). α and β are parameters of the two Dirichlet distributions in (3.3) and (3.4).

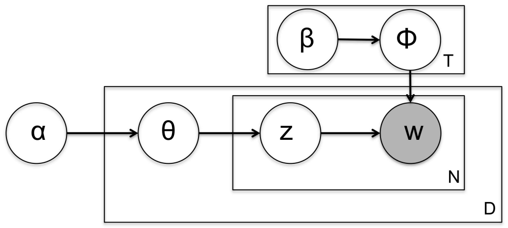


Figure 3.1: LDA Graphical Model.

$$P(\theta_d) = \mathrm{Dir}(\theta_d; \alpha\vec{m}) = \frac{\Gamma(\alpha)}{\prod_{t=1}^{T}\Gamma(\alpha m_t)}\prod_{t=1}^{T}\theta_{t|d}^{am_t-1}, \tag{3.3}$$

$$P(\phi_t) = \mathrm{Dir}(\phi_t; \beta\vec{n}) = \frac{\Gamma(\beta)}{\prod_{k=1}^{W}\Gamma(\beta n_k)}\prod_{k=1}^{W}\phi_{k|t}^{\beta n_k-1}. \tag{3.4}$$

where $\alpha > 0, \beta > 0, \vec{m} = [m_1, \ldots, m_T], \vec{n} = [n_1, \ldots, n_W], \sum_i m_i = 1, \sum_j n_j = 1$.

Prior Function. In LDA, ϕ_d and θ_t are independent for $d = 1, \ldots, D$ and $t = 1, \ldots, T$. Thus we have the prior of Φ and θ:

$$P(\phi_1, \ldots, \phi_T, \theta_1, \ldots, \theta_D) = \prod_{t=1}^{T}\mathrm{Dir}(\phi_t; \beta\vec{n})\prod_{d=1}^{D}\mathrm{Dir}(\theta_d; \alpha\vec{n}). \tag{3.5}$$

Likelihood Function. In the rest of the chapter, we use $w \sim \theta$ to represent that a random variable, denoted as w, follows a probability distribution, denoted as θ. Let $N_{k|t} = \sum_{i=1}^{N}\mathbf{1}(w_i = k, w_i \sim \phi_t)$ be the number of receivers of profile k that are chosen according to preference ϕ_t, where $\mathbf{1}(\mathcal{D}) = 1$ if $\mathcal{D}$ is true, otherwise $\mathbf{1}(\mathcal{D}) = 0$. The likelihood of $\mathcal{W}$ given $\mathbf{z}, \Phi$ is

$$P(\mathcal{W}|\mathbf{z}, \Phi) = \prod_{i=1}^{N}P(w_i|z_i, \Phi) = \prod_{t=1}^{T}\prod_{k=1}^{W}\phi_{k|t}^{N_{k|t}}. \tag{3.6}$$

Let $N_{t|d} = \sum_{i=1}^{N}\mathbf{1}(z_i = t, z_i \sim \theta_d)$ be the number of invitations that user d use preference ϕ_t to generate. The likelyhood of $\mathbf{z}$ given θ is:

$$P(\mathbf{z}|\theta) = \prod_{i=1}^{N} P(z_i|\theta) = \prod_{d=1}^{D}\prod_{t=1}^{T} \theta_{t|d}^{N_{t|d}}. \tag{3.7}$$

Evidence. Our LDA considers $\mathcal{W}$ and $\mathbf{z}$ as evidence. According to (3.3)- (3.7), their joint probabilities are:

$$\begin{aligned}
P(\mathcal{W},\mathbf{z}) &= \int \cdots \int P(\mathcal{W}|\mathbf{z})P(\mathbf{z}|\phi_1,\ldots,\phi_T,\theta_1,\ldots,\theta_D)P(\phi_1,\ldots,\phi_T,\theta_1,\ldots,\theta_D)d\phi_1\cdots d\theta_N,\\
&= \prod_{t=1}^{T}\left[\int \prod_{k=1}^{W} \phi_{k|t}^{N_{k|t}} \mathrm{Dir}(\phi_t;\beta\vec{n})d\phi_t\right] \prod_{d=1}^{D}\left[\int \prod_{t=1}^{T} \theta_{t|d}^{N_{t|d}} \mathrm{Dir}(\theta_d;\alpha\vec{m})d\theta_d\right],\\
&= \prod_{t=1}^{T} \frac{\Gamma(\beta)\prod_{k=1}^{W}\Gamma(N_{k|t}+\beta n_k)}{\Gamma(N_t+\beta)\prod_{k=1}^{W}\Gamma(\beta n_k)} \prod_{d=1}^{D} \frac{\Gamma(\alpha)\prod_{d=1}^{D}\Gamma(N_{t|d}+\alpha m_t)}{\Gamma(N_d+\alpha)\prod_{t=1}^{T}\Gamma(\alpha m_t)}. \qquad (3.8)
\end{aligned}$$

where $N_t = \sum_{k=1}^{W} N_{k|t}$ is the number of receivers chosen by preference ϕ_t. $N_d = \sum_{t=1}^{T} N_{t|d}$ is the number of invitations sent by sender d.

Note that $\mathbf{z}$ is unobservable in the data, we will explain how to use Gibbs sampling to obtain $P(\mathbf{z}|\mathcal{W})$ at the end of this section.

Posterior Function. According to (3.6) - (3.8), the posterior of Φ and θ given evidence $\mathcal{W}$ and $\mathbf{z}$ is obtained using Bayes rule:

$$P(\Phi,\theta|\mathcal{W},\mathbf{z}) = \frac{P(\mathcal{W}|\Phi,\mathbf{z})P(\mathbf{z}|\theta)P(\Phi,\theta)}{P(\mathcal{W},\mathbf{z})}, \tag{3.9}$$

$$= \prod_{t=1}^{T} \mathrm{Dir}\left(\phi_t;\left[\frac{\beta n_1 + N_{1|t}}{\beta+N_t},\ldots,\frac{\beta n_W + N_{W|t}}{\beta+N_t}\right]\right) \prod_{d=1}^{D} \mathrm{Dir}\left(\theta_d;\left[\frac{\alpha m_1 + N_{1|d}}{\alpha+N_d},\ldots,\frac{\alpha m_T + N_{T|d}}{\alpha+N_d}\right]\right).$$

The posterior function can be use to calculate $\mathbf{E}(\phi_{k|t}|\mathcal{W},\mathbf{z})$ and $\mathbf{E}(\theta_{t|d}|\mathcal{W},\mathbf{z})$:

$$\mathbf{E}(\phi_{k|t}|\mathcal{W},\mathbf{z}) = \frac{N_{k|t}+\beta n_k}{N_t+\beta}, \tag{3.10}$$

$$\mathbf{E}(\theta_{t|d}|\mathcal{W},\mathbf{z}) = \frac{N_{t|d}+\alpha m_t}{N_d+\alpha}. \tag{3.11}$$

Note that $\mathbf{E}(\phi_{k|t}|\mathcal{W}, \mathbf{z})$ only depends on $N_{k|t}$, suppose we observe new data $\mathcal{W}', \mathbf{z}'$ where $N'_{k|t}$ invitations are generated with ϕ_t to user of profile k, user d choose ϕ_t to send $N'_{t|d}$ invitations, then (3.10) and (3.11) can be updated as:

$$\begin{aligned}
\mathbf{E}(\phi_{k|t}|\mathcal{W}, \mathbf{z}) &= \frac{N_{k|t} + N'_{k|t} + \beta n_k}{N_t + N'_t + \beta}, & (3.12)\\
\mathbf{E}(\theta_{t|d}|\mathcal{W}, \mathbf{z}) &= \frac{N_{t|d} + N'_{t|d} + \alpha m_t}{N_d + N'_d + \alpha}. & (3.13)
\end{aligned}$$

where $N'_t = \sum_{k=1}^{W} N'_{k|t}, N'_d = \sum_{t=1}^{T} N'_{t|d}$.

Learning z Through Gibbs Sampling. When we compute the evidence function $P(\mathcal{W}, \mathbf{z})$, we want to estimate $P(\mathbf{z}|\mathcal{W})$. With Bayesian rule, $P(\mathbf{z}|\mathcal{W}) = \frac{P(\mathcal{W},\mathbf{z})}{\sum_{\mathbf{z}} P(\mathcal{W},\mathbf{z})}$. However, vector $\mathbf{z}$ has T^N possible values, causing the computation intractable. As a result, we use Gibbs sampling to obtain an approximated value of $\mathbf{z}$.

Gibbs sampling is a Markov chain Monte Carlo method that obtains a sequence of observation from a multivariate probability distribution. In our case, instead of sampling $\mathbf{z} = [z_1, \ldots, z_N]$ directly from $P(\mathbf{z}|\mathcal{W})$, we sample z_i from $P(z_i|\mathbf{z}_{(-z_i)})$ iteratively for $i = 1, \ldots, N$, where $\mathbf{z}_{(-z_i)} = z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_N$. The process is as follow:

- initialize $\mathbf{z} = [z_1^{(0)}, \ldots, z_N^{(0)}]$ with random values.
- For iteration $a+1$, sample $z_i^{(a+1)}$ from $P(z_i|z_1^{(a)}, \ldots, z_{i-1}^{(a)}, z_{i+1}^{(a)}, \ldots, z_N^{(a)})$ for $i = 1, \ldots, N$.
- repeat until $P(z_i|\mathbf{z}_{(-z_i)})$ converges or iteration step reaches a predefined threshold A.

With Gibbs sampling, $\mathbf{z}$ is approximated by $\hat{\mathbf{z}} = [\hat{z}_1, \ldots, \hat{z_{i-1}}] = [z_1^{(A)}, \ldots, z_N^{(A)}]$. We can count the approximate values of $N_{k|t}, N_t$ and $N_{t|d}$ in (3.6) - (3.8), denoted as $\hat{N}_{k|t}, \hat{N}_t$ and $\hat{N}_{t|d}$.

As for the distribution $P(z_i|\mathbf{z}_{(-z_i)})$ to sample $\mathbf{z}$, we have

$$P(z_i|\mathbf{z}_{(-z_i)}) = \frac{P(w_i, z_i|\mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)})}{\sum_{z_i} P(w_i, z_i|\mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)})} \propto P(w_i, z_i|\mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)}) \tag{3.14}$$

where $\mathcal{W}_{(-w_i)} = [w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_N]$. We have:

$$P(w_i = k, z_i = t|\mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)}) = P(w_i = k|z_i = t, \mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)})P(z_i = t|\mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)}) \tag{3.15}$$

To calculate right hand side of (3.15), let $N_{k|t}^{(-i)} = \sum_{j \neq i} \mathbf{1}(w_j = k, w_j \sim \phi_t), N_t^{(-i)} = \sum_{k=1}^{W} N_{k|t}^{(-i)}$ and $N_{t|d}^{(-i)} = \sum_{j \neq i} \mathbf{1}(z_j = t, z_j \sim \theta_d)$. Consider (3.10) and (3.11), we can compute $P(w_i = k, z_i = t|\mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)})$:

$$\begin{aligned} P(w_i = k|z_i = t, \mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)}) &= \int \cdots \int \phi_{k|t} \prod_{t=1}^{T} P(\phi_{k|t}|\mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)}) d\phi_1 \cdots d\phi_T, \\ &= \mathbf{E}(\phi_{k|t}|\mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)}), \\ &= \frac{N_{k|t}^{(-i)} + \beta n_k}{N_t^{(-i)} + \beta}. \end{aligned} \tag{3.16}$$

$$\begin{aligned} P(z_i = t|\mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)}) &= \int \cdots \int \theta_{t|d} \prod_{d=1}^{D} P(\theta_{t|d}|\mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)}) d\theta_1 \cdots d\theta_D, \\ &= \mathbf{E}(\theta_{t|d}|\mathcal{W}_{(-w_i)}, \mathbf{z}_{(-z_i)}), \\ &= \frac{N_{t|d}^{(-i)} + \alpha m_t}{N_d^{(-i)} + \alpha}. \end{aligned} \tag{3.17}$$

3.4.2 Prediction

After Gibbs sampling, we can approximate $\mathbf{E}(\phi_{k|t}|\mathcal{W}, \mathbf{z})$ and $\mathbf{E}(\theta_{t|d}|\mathcal{W}, \mathbf{z})$ by updating $N_{k|t}, N_t$ and $N_{t|d}$ in (3.10) and (3.11)

$$\mathbf{E}(\theta_{t|d}|\mathcal{W}, \hat{\mathbf{z}}) = \frac{\hat{N}_{t|d} + \alpha m_t}{N_d + \alpha} \tag{3.18}$$

$$\mathbf{E}(\phi_{k|t}|\mathcal{W}, \hat{\mathbf{z}}) = \frac{\hat{N}_{k|t} + \beta n_k}{\hat{N}_t + \beta} \tag{3.19}$$

Suppose we want to predict a new invitation from user u to v with profile k given data $\mathcal{W}, \hat{\mathbf{z}}$, the probability of a link between u and v in (3.1) can be computed as

$$P(e_{uv}|\vec{f_u}, \vec{f_v}, G) = \prod_{t=1}^{T} \frac{\hat{N}_{k|t} + \beta n_k}{\hat{N}_t + \beta} \frac{\hat{N}_{t|u} + \alpha m_t}{N_u + \alpha} \tag{3.20}$$

(3.20) shows that, to predict an edge e_{uv} from u to v, we only need to know N_u (the total number of invitations u sends/accept), $\hat{N}_{t|u}$ (number of times u chose ϕ_t previously), $\hat{N}_t$ (number of invitations sent with preference ϕ_t) and $\hat{N}_{k|t}$ (number of invitations sent to users of profile k with preference ϕ_t), which can be computed using Gibbs sampling method introduced previously.

3.5 Experiment

3.5.1 Experiment Setup, Metrics and Baseline Methods

Since an edge predicted by our model can be viewed as an online dating recommendation, we follow approaches that are widely used for offline evaluation of recommendation systems [91]: an online process is simulated for a model to learn user preferences and make prediction or recommendations to users. In this process, a subset of items from each user is chosen as training data and the rest are hidden for a recommendation system to predict. The simulated process is useful for testing models on a dataset where users tend to have unique preferences. This process suits our data because most stated preferences are different. We apply the so-called "all but n" protocol, where a fixed number of hidden items per user are chosen for testing. Since the fraction of reply behaviors that violate stated preferences increases with time, we also create another dataset by selecting users that have been active for at least two month to test if our methods can effectively update the user preferences. Baseline methods include logistic regression (LR), random forest (RF), K-nearest neighbor (KNN) and a model using stated preferences.

Baseline Methods Logistic regression and random forest learn user preferences only through their reply behaviors because we are unable to observe users that an invitation sender does not like. In other words, there are no negative instances to learn a sender's preferences for these two methods. In these two methods, the variable "reply" is set as a binary label and other variables such as user profiles are set as features. We use R packages for the two methods [13, 23] that deals with imbalance data with "out-of-bag" estimation. For random forest algorithm, we increase the number of trees from default value 100 to 400.

A stated preference of user d in our data is represented as $\mathcal{S}^d = [s_1^d, \ldots, s_K^d]$, where s_k^d is an interval(for numeric feature) or a set of integers(for categorical feature) that indicates the preferred value of the k-th user feature, for $k = 1, \ldots, K$, where K is the total number of features in a user profile. Let user v's profile be $\vec{f_v} = [f_1^v, \cdots, f_K^v]$ as defined in Section 3.3. The distance between a feature f_k^v value to the preference of s_k^d is defined as

$$a(f_k^v, s_k^d) = \begin{cases} 0, & \text{if } f_k^v \in s_k^d, \\ 1, & \text{if } f_k^v \notin s_k^d \text{ and } s_k^d \text{ is categorical,} \\ \sqrt{(f_k^v - \frac{b_k + c_k}{2})^2}, & \text{if } f_k^v \notin s_k^d \text{ and interval } s_k^d = [b_k, c_k]. \end{cases}$$

For an invitation i, we obtain a vector $x_i = [a(f_1^d, s_1^v), \cdots, a(f_K^d, s_K^v)]$ and a binary label representing reply behavior. we assume each feature in a profile affects a user's decision differently and assign a weight w_k to $a(f_k^v, s_k^d)$. The probability that receiver d replies to sender v in invitation i is modeled as a logistic function:

$$g(x_i) = \frac{1}{1 + e^{-Wx_i + w_0}} \tag{3.21}$$

where $W = [w_1, \cdots, w_K], w_0$ is a constant. We learn W and w_0 via logistic regression on the data, and decide a user d is interested in user v if $g(x) >= 0.5$.

Metrics In Section 3.5.2, we apply our model to synthetic data to learn ground truth user preferences and test the performance of our model using the number of retrieved ground truth preferences and the KL-divergence between the retrieved preference and the true preferences.

Section 3.5.4 evaluates the accuracy of our model in predicting edges that represent user behaviors. We use logistic regression (LR), random forests (RF) and collaborative filtering based on K-nearest neighbor (KNN) as baseline methods for comparison. A recommendation system usually faces the so-called "cold start" users whose behaviors have never been observed before. It is difficult to provide them with accurate recommendations in this case. A common approach is to recommend most popular items to users. Once the behaviors of these "cold start" users are observed, a recommendation system updates their preferences and provides better recommendations. A useful metric is to evaluate the improvement on prediction accuracy.

3.5.2 Evaluation of User Preferences with Synthetic Data

To understand how different datasets affect the performance of our LDA model, we simulate an online dating market and generate 20,000 male and female users with profiles, respectively. This data also helps us understand how our model performs when parameters, such as number of preferences T, change. Each simulated user has a profile of five categorical features representing the range of age, number of children, range of weight, range of income, and range of height. These five features consist of a feature vector v in a $6 \times 3 \times 6 \times 4 \times 6$ vector space, denoted as V.

Our simulator uses 40 distinct ground truth preferences per gender (and a total of 80 preferences for both genders). Each preference is a unique distribution over all possible feature vectors, denoted as $\phi_t = (\phi_{1|t}, \ldots, \phi_{|V||t})$, where $t = 1, \ldots, T\}$ and V is the feature space. We then randomly select 0.2% of the feature vectors in V that belong to the opposite gender as preference ϕ_t's favorite feature vectors, denoted as F. Then for each $v \in F$ we set $\phi_{v|t}$ with a value drawn uniformly from interval $(400, 500)$. For the remaining feature vectors, $v \in V \backslash F$, $\phi_{v|t}$ is sampled uniformly from the interval $[1, 2]$. This insures that favorite feature vectors will be chosen with a probability more than 60%. Finally, we normalize ϕ_t such that $\sum_{v \in V} \phi_{v|t} = 1$.

Each user d has a unique mixture of the 40 ground truth user preferences of his/her gender, denoted as $p_d = \sum_{t=1}^{40} \theta_{t|d} \phi_t$, such that he/she behaves differently from other users, where $\sum_{t=1}^{40} \theta_{t|d} = 1$. We refer to p_d as d's "*personal preference*" and distinguish it from user preference ϕ_t.

To simulate dating dynamics, each invitation sender d selects a feature vector v according to his/her own preference $\phi_{v|d}$. Then a user of the opposite gender with feature vector v is chosen without replacement as a receiver. If no such receiver matches v, a sender selects another feature vector v' until a total of k_d receivers are chosen, where k_d is uniformly sampled from interval $[100, 400]$. We also simulate a receiver r who receives K invitations. An invitation from sender d with feature vector v_d is assigned a weight $\phi_{v_d|r}$ from r's preference ϕ_d. Receiver r chooses an invitation to reply with probability proportional to its weight. This process is repeated until k_r invitations have been replied, where k_r is sampled uniformly from $[0, K]$.

Performance of LDA Given Different Datasets and Parameter Settings We investigate how different dataset and parameter setting of LDA model affect the performance. We first consider whether it is easy to distinguish a ground truth user preference from another and how differences between ground truth preferences affect the performance of our LDA model.

The difference between two preferences ϕ_t and $\phi_{t'}$ can be measured with KL-divergence, defined as

$$D_{KL}(\phi_t || \phi_{t'}) = \sum_{v=1}^{V} \log \left(\frac{\phi_{v|t}}{\phi_{v|t'}} \right) \phi_{v|t}$$

When $D_{KL}(\phi_t||\phi_{t'}) = 0$, the distribution ϕ_t and $\phi_{t'}$ can be considered to be nearly the same.

Considering that $D_{KL}(\phi_t||\phi_{t'})$ is asymmetric, the average difference of all true preferences in our data is measured by the mean of KL-divergence between all ground truth preference pairs, denoted as

$$\bar{D}_{KL}(\phi_t||\phi_k) = \frac{1}{T(T-1)} \sum_{t \neq k; t,k=1,\ldots,T} D_{KL}(\phi_t||\phi_k), \tag{3.22}$$

where T is the number of total ground truth preferences per gender. Small values of $\bar{D}_{KL}(\phi_t||\phi_k)$ indicate that the ground truth preferences are similar, potentially causing LDA model to retrieve a preference that is a mixture of several ground truth preferences. When $\bar{D}_{KL}(\phi_t||\phi_k) = 0$, all ground truth user preferences are identical. When $\bar{D}_{KL}(\phi_t||\phi_k)$ is large (e.g., $\bar{D}_{KL}(\phi_t||\phi_k) > 0.5$,) user preferences are considered to be different and easy to distinguish from each other.

We select different preferences to generate six datasets, where the average $\bar{D}_{KL}(\phi_t||\phi_k)$ of both genders are equal to $0.0019, 0.0047, 0.012, 0.053, 0.13, 0.56$.

We also investigate how the number of unique preferences for both genders, T, affects the performance of our model. We set $T = 10, 20, 30, 40, 50$ in our model and run it 10 times on each setting.

We implement 10-fold cross validation to generate ten training sets and ten test sets: invitation senders are randomly split into 10 groups. We choose behaviors of one sender group as a test set and the rest as a training set. We repeat this process ten times. Note that when a sender is chosen for training/testing, all his/her behaviors are used only as training/testing data. This setting is called "leave-one-batch-out" (LOBO), where all behaviors of a sender is called a "batch". LOBO is reported to avoid upward biases in accuracy[52]. On the other hand, a recommender system usually focuses on personalized recommendation and requires training data for each user to obtain better results. LOBO results in a so-called "cold start", where

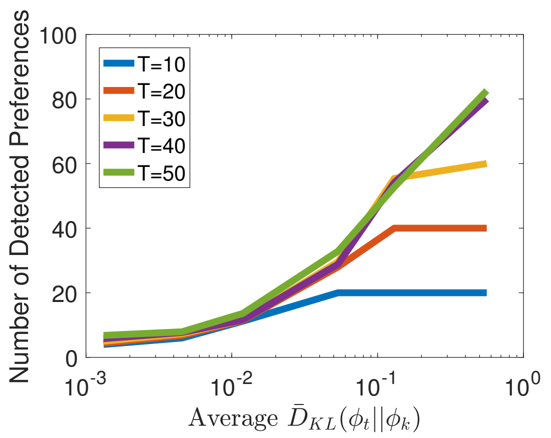


Figure 3.2: Number of Retrieved Ground Truth Preferences. X-axis is the log scale average $\bar{D}_{KL}(\phi_t||\phi_k)$ of both gender. The total number of user preferences in a dataset is 80(, 40 for each gender). We set $T = 10, 20, 30, 40, 50$ for each gender in our model. Small $\bar{D}_{KL}(\phi_t||\phi_k)$ indicates that ground truth preferences are similar and causes LDA model to retrieve less ground truth preferences.

users have no observed behaviors and the recommendation for them are less accurate. To study the effect of LOBO and how well our model improves recommendation performance with new data, we first learn user preference ϕ_t with LOBO for $t = 1, \ldots, 40$ in each gender and evaluate the results with KL-divergence. We then choose a fraction of behaviors of users from the test set to train our model to evaluate the improvement on "cold start".

Figure 3.2 illustrates how the average $\bar{D}_{KL}(\phi_t||\phi_k)$ affects the number of ground truth preferences retrieved by our model. When $\bar{D}_{KL}(\phi_t||\phi_k)$ is small and equal to 0.0019, users in the dataset have similar behaviors. It is difficult to distinguish one preference from another. Our LDA model, on average, retrieved less than 3 unique preferences for all genders even when T is large. As $\bar{D}_{KL}(\phi_t||\phi_k)$ increases, ground truth preferences are more distinguishable from one another, and our model retrievs more user preferences. All ground truth preferences are detected in our model when $\bar{D}_{KL}(\phi_t||\phi_k) = 0.56$.

We then evaluate how similar retrieved preferences and ground truth preferences are. A one-to-one mapping from a retrieved preference $\hat{\phi}_t$ to a ground truth preference ϕ_t (mapping learned distribution in LDA model) is a well-defined "label switching" problem that can be solved by choosing pairs of preferences with the smallest KL-divergence $D_{KL}(\hat{\phi}_k||\phi_t)$ [16], where $k = 1, \ldots, T; t = 1, \ldots, 40$. When T is larger than the number of ground truth preferences, the retrieved preferences that fail to match a ground truth preference usually contain few samples during Gibbs sampling and will be ignored. We simply follow an up-to-date approach in R package [86] and obtain K matched preference pairs for both genders from LDA model. For simplicity, we reorder the preferences and use $D_{KL}(\hat{\phi}_t||\phi_t)$ to represent the KL-divergence of the t-th matched preference pair, where $t = 1, \ldots, K$. A small value of $D_{KL}(\hat{\phi}_t||\phi_t)$ indicates that a retrieved preference is close to a ground truth preference.

Figure 3.3 shows the mean, maximum and minimum of $D_{KL}(\hat{\phi}_t||\phi_t)$, where $t = 1, \ldots, K$ and $K \leq T$ is the number of matched preferences from LDA model with $T = 40$.

Figure 3.4 shows the average difference between retrieved preferences and ground truth preferences, $\bar{D}_{KL}(\hat{\phi}_t||\phi_t)$, as a function of T given $\bar{D}_{KL}(\phi_t||\phi_k)$, the average difference between ground truth preferences. When ground truth preferences are similar ($\bar{D}_{KL}(\phi_t||\phi_k) = 0.0019$), the retrieved preferences are similar to the ground truth preferences and $\bar{D}_{KL}(\hat{\phi}_t||\phi_t)$ is less sensitive to the change in T, the number of preferences per gender in our model. On the other hand, if ground truth preferences are different ($\bar{D}_{KL}(\phi_t||\phi_k) = 0.56$), T significantly affects the difference between a matched preference-pair: a small T results in a large value in $\bar{D}_{KL}(\hat{\phi}_t||\phi_t)$; if T is greater or equal to the number of ground truth preferences, $\bar{D}_{KL}(\hat{\phi}_t||\phi_t)$ is small, indicating a good approximation of the ground truth.

A recommender system needs to update a users' mixture preference based on his new behaviors to provide better recommendations. Let $\hat{p}_d^{(i)}$ be d's retrieved preference

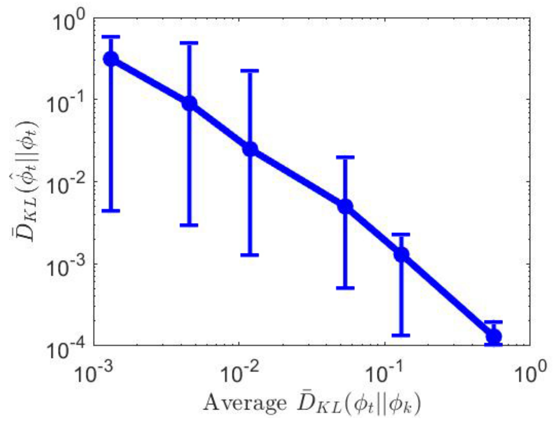


Figure 3.3: Y axis: errors in retrieving user preferences, measured by $\bar{D}_{KL}(\hat{\phi}_t||\phi_t)$ (average KL-divergence between retrieved preferences and ground truth preferences). X axis: diversity of ground truth user preferences, measured by average $\bar{D}_{KL}(\phi_t||\phi_k)$ (average KL-divergence between ground truth preferences). Retrieved preferences are obtained by LDA model with $T = 40$. Small $\bar{D}_{KL}(\phi_t||\phi_k)$ indicates ground truth preferences are similar with one another, resulting in larger preference retrieving error, $\bar{D}_{KL}(\hat{\phi}_t||\phi_t)$. As $\bar{D}_{KL}(\phi_t||\phi_k)$ increases, ground truth preferences are easier to distinguish, and LDA obtains the ground truth preferences with less error.

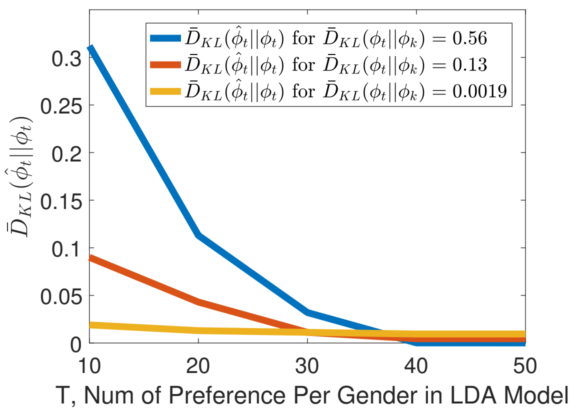


Figure 3.4: Y axis: error in retrieving user preferences, denoted as $\bar{D}_{KL}(\hat{\phi}_t||\phi_t)$ (KL-divergence measures the difference between retrieved prefererence $\hat{\phi}_t$ and ground truth prefererence ϕ_t). X axis: number of user preferences in LDA model, denoted by T. Our model was tested on synthetic datasets with different $\bar{D}_{KL}(\phi_t||\phi_k)$. Small $\bar{D}_{KL}(\phi_t||\phi_k)$ indicates that ground truth preferences are similar and difficult to be distinguished from one another. The results suggest that increasing T in our model can more significantly improve the approximation of ground truth preferences when $\bar{D}_{KL}(\phi_t||\phi_k)$ is larger.

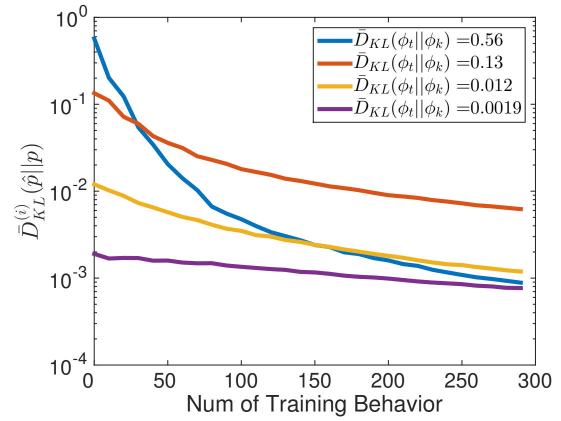


Figure 3.5: Log scale of $\bar{D}^{(i)}_{KL}(\hat{p}||p)$, the averge KL-divergence between retrieved personal preferences and ground truth preferences as a function of i, the number of training behavior per user for four synthetic datasets. Personal preferences for senders and receivers are retrieved with our LDA model for four datasets that differ in $\bar{D}_{KL}(\phi_t||\phi_k)$, the average difference between user preferences. The accuracy is measured with $\bar{D}^{(i)}_{KL}(\hat{p}||p)$. As the number of training behaviors for a user increases, $\bar{D}^{(i)}_{KL}(\hat{p}||p)$ decreases, implying that personal preferences are closer to ground truth.

learned from i of his behaviors, where $i = 1, 2, \ldots$ and $D_{KL}(\hat{p}^{(i)}_d||p_d)$ is the KL-divergence between $p^{(i)}_d$ and d's ground truth preference p_d. We use $\bar{D}^{(i)}_{KL}(\hat{p}||p) = \frac{1}{D^{(i)}}\sum_{d=1}^{D^{(i)}} D_{KL}(\hat{p}^{(i)}_d||p_d)$ to measure the performance of our model on learning users' mixture preferences, where $D^{(i)}$ is the number of users that send out more than i messages. Figure 3.5 shows $\bar{D}^{(i)}_{KL}(\hat{p}||p)$ as a function of i, the number of training behaviors per user. As i increases, $\bar{D}^{(i)}_{KL}(\hat{p}||p)$ decreases, indicating that the retrieved personal preference becomes more accurate. When ground truth user preferences are easy to distinguish (larger value of $\bar{D}^{(i)}_{KL}(\hat{p}||p)$), the approximation of personal preference is better if i is large enough. On the other hand, if ground truth preferences are almost identical ($\bar{D}^{(i)}_{KL}(\phi_t||\phi_k) = 0.0019$), all users behave in the same way. LDA model considers that those user behaviors come from the same preference and retrieves fewer preferences. Since the learned preferences are also similar to the ground truth, $\bar{D}^{(i)}_{KL}(\hat{p}||p)$ is small.

Next, we compare our LDA model to baseline methods for recommendation results using ROC curves. For sender behaviors, we use K-nearest neighbor (KNN) algorithm for comparison (logistic regression and random forest are unable to learn sender preferences because the lack of negative instances). Then we compare the performance of our model to all baseline methods on predicting receiver behaviors.

Our LDA model first retrieves user preferences ϕ_t from the training set for $t = 1, \ldots, 40$. To predict behaviors of a user d in the test set, Both LDA and KNN require that a fraction of d's behaviors be observed. Let i be the number of user d's behaviors that are randomly selected from the test set to update his personal preference. When $i = 0$, we begin a "cold start" prediction and the personal preference is an average of retrieved user preferences ϕ_t for $t = 1, \ldots, T$ (This means the training setup is LOBO). The i behaviors are used to obtain user d's K nearest neighbors in the training set.

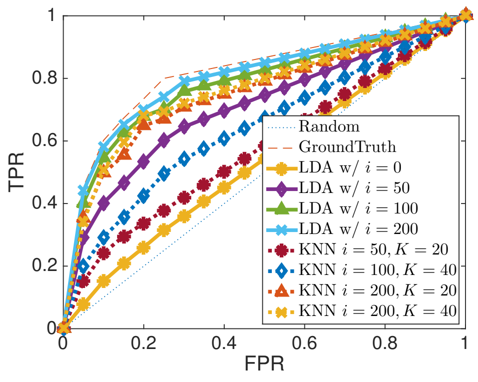
Note that our problem is to predict whether a sender sends an invitation to a user. This problem refers to a binary recommendation [91]. Howover, we do not observe users whose profiles are examined by a sender but do not receive an invitation. That is, there is no negative instances for our classification problem. As a result, we follow a common evaluation approach [91] suitable for our case: let d be a sender sending out invitations to $I > i$ receivers. A recommendation method generates "ranking scores" for all other users of the opposite gender (except for those i receivers in the training data) then ranks users in descending order.

Evaluation metric An ROC curve for the prediction of d's behavior is generated from the user ranking: Let $P = I - i$ be the number of receivers that receive invitations from d in the dataset, N is the number of users that do not receive invitations. J is the number of unique ranking scores, R_j is the j-th largest ranking score. Consider the users whose ranking scores are larger or equal to R_j, $P_j \leq P$ is the number of invitation receivers, N_j is the number of the rest users. A true positive rate (TPR) is calculated by $\frac{P_j}{P}$ and the corresponding false positive rate (FPR) is calculated by $\frac{N_j}{N}$. Each TPR-FPR pair is considered as a point in a 2D space. By repeating the above process for $j = 1, \ldots, J$, we obtain J points $\{(\frac{P_j}{P}, \frac{N_j}{N})\}_{j=1}^{J}$. An ROC curve is obtained by connecting $(0,0)$ to $(\frac{P_1}{P}, \frac{N_1}{N})$, and $(\frac{P_j}{P}, \frac{N_j}{N})$ to $(\frac{P_{j+1}}{P}, \frac{N_{j+1}}{N})$.

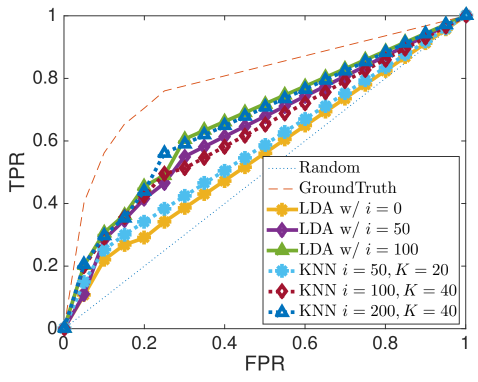
After obtaining a ROC curve for a sender sending $I > i$ invitations, we want to evaluate the overall performance of a method on all those senders in the ten test datasets. Fawcett [30] proposed a vertical averaging of ROC curves to obtain average TPR for each corresponding FPR: (1) choose a set of FPRs. In our case, we choose 20 FPR values: $\text{FPR}_j = \frac{N_j}{N} = 0.05, 0.1, 0.15, \ldots, 1$. (2) For each ROC curve for a user d, denoted as ROC_d, obtain the TPR values $\text{TPR}_j^d = \frac{P_j^d}{P}$ for $j = 1, \ldots, 20$ by interpolation on ROC_d. (3) calculate the average TPR for all users $\overline{\text{TPR}}_j = \frac{1}{D}\sum_{d=1}^{D} \frac{P_j^d}{P}$ so that for each j we obtain a TPR-FPR pair $(\overline{\text{TPR}}_j, \text{FPR}_j)$, where D is the number users sending more than i invitations. (4) connect $(0,0)$ and $(\overline{\text{TPR}}_j, \text{FPR}_j)$ and plot the ROC curve.

Figure 3.6 shows the ROC curves for our LDA model and KNN given senders' i observed behaviors on datasets with different average similarities in ground truth preferences (denoted by $\bar{D}_{KL}(\phi_t||\phi_k)$). Figure 3.6(a) corresponds to a data set that contains different user preferences ($\bar{D}_{KL}(\phi_t||\phi_k) = 0.56$). Figure 3.6(b) corresponds to a dataset with similar user preference ($\bar{D}_{KL}(\phi_t||\phi_k) = 0.13$). We select these two datasets to show that when preferences are easy to distinguish, all models performs better. However, we do not show the extreme case where $\bar{D}_{KL}(\phi_t||\phi_k) = 0.0019$ and all users have similar behaviors because this is not expected to occur in the real-world (In this case, all models have similar performances). When there is no training behavior for user d ($i = 0$), KNN is known to be equivalent to a random prediction. Our model assume user d's personal preference as $\frac{1}{40}\sum_{t=1}^{40} \phi_t$ according to (3.13).

A user's personal preference can be learned by his sending behaviors or replying behaviors. A receiver r's personal preference is difficult to learn because his behaviors



(a) $\bar{D}_{KL}(\phi_t||\phi_k) = 0.56$



(b) $\bar{D}_{KL}(\phi_t||\phi_k) = 0.13$

Figure 3.6: Average ROC curves of behavior prediction for invitation senders. (a) ROC curves for LDA model and KNN on a dataset with $\bar{D}_{KL}(\phi_t||\phi_k) = 0.56$, (b) ROC curves for LDA model and KNN on a dataset with $\bar{D}_{KL}(\phi_t||\phi_k) = 0.13$ where ground truth user preference ϕ_t is more difficult to distinguish than in (a). Senders' personal preferences are easier to learn in dataset in (a) than (b). For each sender d in test data, i of his/her behaviors are selected as observed data to train d's personal preference for our LDA model and to calculate his/her similarities to other sender to obtain K near neighbor for KNN. For both methods, performance improves with the increase in i, the number of training data for each sender. However, when i reaches 200 from 100, the improvement becomes less significant. Comparing the two figures (a) and (b), the improvement is more significant when $\bar{D}_{KL}(\phi_t||\phi_k)$ is larger, as shown in (a). Given the same i, LDA performs better than KNN, especially when i is small.

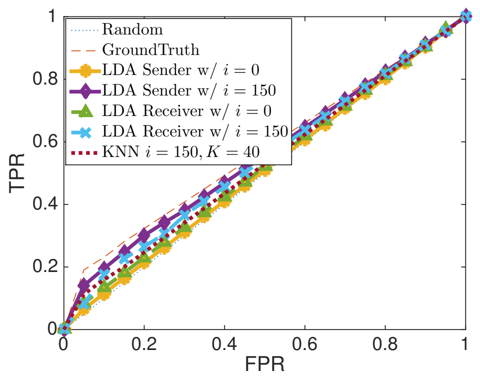


Figure 3.7: ROC curve for prediction of receivers' behaviors on dataset with $\bar{D}_{KL}(\phi_t||\phi_k) = 0.56$. Receivers' behaviors are more difficult to predict than senders' behavior. ROC curve for prediction shows that even ground truth preferences are unable to obtain high accuracy. Since a user uses the same personal preferences to send and respond to invitations. User preferences learned from senders behaviors have better performance than those learned from receiver behaviors.

are limited to replies to invitations. However, there is no guarantee that each of r's favorite type of user will be among those senders. As a result, we may not obtain all r's favorite types and r's personal preference may be biased.

In the following experiments, We test if personal preferences learned from sending behaviors have the same performance on prediction as those from replying behaviors.

We choose the dataset with $\bar{D}_{KL}(\phi_t||\phi_k) = 0.56$ for evaluation because user personal preferences are relatively easy to learn. Figure 3.7 shows the average ROC curve for predictions of receiver behaviors using sender personal preferences, receiver personal preferences and KNN. We also use the ground truth preferences for the prediction. The prediction for receiver behaviors is more difficult than that for sender behaviors: even ground truth preferences do not produce high accuracy. For "cold start" prediction (LOBO setting with $i = 0$), both sender and receiver preferences are just a little better than random binary classification. When $i = 150$, predictions using both type of preferences improves. Also note that prediction by receiver preferences are less accurate than prediction by sender preferences.

3.5.3 Real-world Dataset

We test our method on an online dating dataset collected from baihe.com. It consists of two million invitations between November 2011 and January 31st 2012. The invitations are sent/received by $200,000$ users registered in November 2011. The dataset contains $293,804$ user profiles (including users who did not register in November 2011), with each gender making up 69.7% and 30.3% of the users, respectively. Users come from all over China and also abroad [98]. When they register, they filled in requirements expected for their potential partner, such as age and income. These requirements are usually called "stated preferences". For each user, we obtain all invitations from the date that his/her account was created until January 31st, 2012. We also obtain the profile information of all users involved in these invitation, totaling 2 million unique pairs of users invitations during our observation period. The content of each invitation is removed for privacy concerns but other relevant information remains, such as the invitation timestamp, the sender's and receiver's profiles, which consists of 21 features including: gender, age, registration timestamp, blood type, weight, height, education, occupation, annual income level, housing situation (renting, home owner), body type, Western zodiac sign, Chinese zodiac sign, number of profile photos, whether the user owns a car, city of residence, and the whether user has a child and lives with the child, among other characteristics.

3.5.4 Experiment Setup

Setup for LDA model In our previous study [99], we discovered that some features in a user profile are redundant, some features may be irrelevant to a user's decision to send/reply an invitation. This increases computational complexity and provides little improvement in accuracy of prediction in our model. We apply feature selection with the importance scores of features generated from the gain ratio of decision trees in random forest [13].

Feature Selection in User Profiles The large number of features in user profiles and their values results in a huge feature space, making our LDA model infeasible to compute. Moreover, the large percentage of missing data and errors in some feature variables introduce biases in the data. To ameliorate these problems for our LDA model, we apply feature selection to choose the most relevant features to user behaviors and feature discretization to reduce feature space. We identify the five most relevant features to predicting sending/replying invitations using gain ratio obtained from decision trees and variable importance from in random forest: *age, weight, income difference, children information and height difference.*

Feature Discretization We apply the ChiMerge algorithm, a bottom-up Chi-square quantization algorithm [58] to discretize the feature values. Missing data is represented by a special categorical value "0". After the feature discretization, we have a $7 \times 9 \times 8 \times 5 \times 21$ feature space that represents $W = 52,920$ unique profiles.

Initial Mixture of "Cold Start" Users with Profile for LDA Model We assume that user profiles are related to user personal preferences. To test whether user profiles can improve prediction for "cold start" users, we also design a model to learn relations between user profiles and the personal preferences. We refer to this model as "*LDA model with feature in user profile*".

The process of designing this model is as follows: We define ϕ_t as a user preferences, where $t = 1, \ldots, T$. User r's personal preference is denoted as $p_r = \sum_{t=1}^{T} a_t^{(r)} \phi_t$, where $\sum_{t=1}^{T} a_t^{(r)} = 1$ and $A^{(r)} = [a_1^{(r)}, \ldots, a_T^{(r)}]$ represent a mixture. Given "cold start" user r's profile as a $1 \times K$ feature vector $\vec{f_r}$ in the test set, his mixture is initilized as

$$\hat{A}^{(r)} = \frac{\vec{f_r} M}{||\vec{f_r} M||}$$

where M is a $K \times T$ matrix and $||\cdot||$ represent an L2 norm. M can be easily learned as follows:

$$\arg\min_{M} \sum_{d=1}^{D} ||A^{(d)} - \hat{A}^{(d)}||$$

where d represents a user in the training sets and $A^{(d)}$ is d's mixture return by our LDA model.

Setup for baseline methods The baseline methods includes K-nearest-neighbor (KNN), logistic regression (LR) and random forest (RF). They are the most widely used methods applied in recommendation in industry, such as Netflix movie rating, online shopping recommendation and targeted advertisement. For all baseline methods, we did not perform feature selection or feature discretization on the data because of the methods themselves can decide the relations between features and class label.

We also propose a baseline model using user stated preferences. Recall, from the discussion at the beginning of Section 3.3, that each user has a profile and the profile contains a feature vector of K user features, denoted as $\vec{f} = [f_1, \ldots, f_K]$. In user d's stated preference, d specifies his preferred range of the value of f_k in a partner's profile, for $k = 1, \ldots, K$. However, it is unclear how a feature f_k affects a user's decision to send/reply an invitation. Our baseline model add a weight w_k to a feature f_k, where $w_k = 0$ represents that f_k is irrelevant to a user's decision. Suppose the range of value for f_k in d's stated preference is R_k, we define feature distance to a stated preference as:

$$g_k(f_k, R_k) = \begin{cases} 0, & \text{if } f_k \in R_k \\ \min((f_k - a_k)^2), & \text{if } f_k \notin R_k, \forall a_k \in R_k \end{cases}$$

where $a_k \in R_k$ is one of d's preferred values for feature f_k. The feature distance to a stated preference is 0 if the value of f_k is within the preference range R_k, otherwise, the distance is the minimum $L2$ norm of f_k and a value $a_k \in R_k$. The distance of a user r's profile $\vec{f_r}$ to d's preference p_d can be calculated as

$$d(\vec{f_r}, p_d) = \frac{1}{K}\sum_{k=1}^{K} w_k g_k(\hat{f}_k, R_k) \tag{3.23}$$

We use a logistic function to compute the probability that user d is interested in r:

$$P(e_{dr} = 1) = \frac{1}{1 + e^{d(\vec{f_r}, p_d) + w_0}} \tag{3.24}$$

We consider that user d sends/replies an invitation to/from r if $P(e_{dr} = 1) \geq 0.5$ and label this behavior as $y_{d,r} = 1$, otherwise, $y_{d,r} = 0$. To obtain w_k, for $k = 1, \ldots, K$, we train the baseline model to minimize square error:

$$\sum_{d}\sum_{r^{(d)}}(y_{d,r^{(d)}} - P(e_{dr} = 1))^2$$

where $r^{(d)}$ is a user that is observed to interact with d. We further rank all users for d in ascending order of $d(\vec{f_r}, p_d)$ for $r = 1, 2, \ldots,$ and generate a ranked list to obtain an ROC curve for comparison.

We applied the same evaluation process as on the synthetic dataset. The average KL-divergence for user preferences returned by our LDA model is $\bar{D}_{KL}(\phi_t || \phi_k) = 0.7544$, indicating that user preferences are easy to distinguish. Figure 3.8 shows the ROC curve for offline prediction of sender behaviors by KNN, a model with stated preference, LDA models with/without features in user profile to initilize personal preferences. From the ROC curve, we observe that stated preference is good for "cold start" users and have good performance in LOBO evaluation. For recommender systems such as KNN and our model, historical personal behaviors are important for prediction. Meanwhile features in user profiles can be applied to estimate personal preferences for "cold start" users and help to improve prediction.

We then investigate a receiver's behavior in replying/rejecting an invitation. Figure 3.9 shows the ROC for predictions on receiver behaviors by models with stated

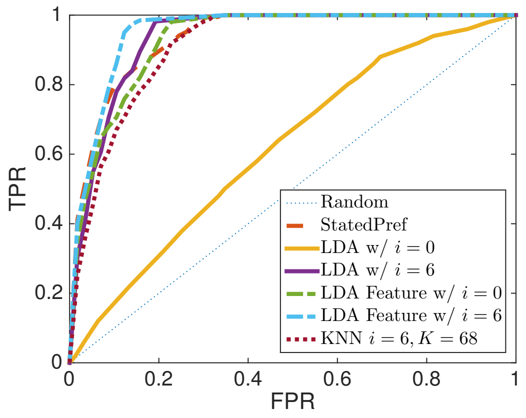


Figure 3.8: ROC Curve for Prediction of Sender Behaviors in Baihe Online Dating website. Model with stated Preferences has higher TPR compare to most of other models when FPR is low. An LDA model without features for personal preference estimation for "cold start" users (LDA w/ $i = 0$) performs poorly, just a little better than a random classification (small dot line). With observation of personal behaviors, our LDA model greatly improves its performance (LDA w/ $i = 6$). If features in user profiles are used to initialize user personal preferences, our model improves on predictions for "cold start" user (according to ROC curves for LDA feature w/$i = 0$). We also test KNN and plot the ROC curve for a KNN (The number of neighbors K ranges from 10 to 100, it has its best overall performance on the test sets when $K = 68$.)

preferences, LDA, KNN, logistic regression and random forest. Similar to the results obtained when examine the synthetic data, it is more difficult to predict receiver behaviors than sender behaviors when comparing ROC in figure 3.7 and 3.9. LDA models and KNN require observed behaviors of tested users to obtain ther personal preferences, we choose the number of observation $i = 1, \ldots, 6$, and only show the best result from $i = 6$ because users on average receive a small number of invitations. The LDA model using both user features and personal behaviors to learn personal preference has overall the best performance. The TPR in the LDA model using only personal behaviors is lower than in a model with stated preferences when FPR is low (below about 0.25), but becomes higher as FPR increases. Methods such as, logistic regression and random forest, use user features in profile without personal behavior for prediction have the lowest performances. This suggest that personalizing user preference can improve prediction.

3.5.5 Change of Stated Preferences

We discover that some users change their behaviors over time in our previous study [99]: on average in each week, there is an increase in the fraction of both sender and receiver behaviors that do not match stated preferences. To study the effect of this discrepancy between behaviors and preferences on prediction by the models, we investigate users who have been active as senders and receivers for at least two months. Those users provide enough behaviors representing changed preferences to test if models adapt to these changes.

Experiment setup and Metric We simulate an online process to compare our model to other methods with newly arriving data: We first adjust the time of some invitations such that all targeted users register in the first week. At the end of each week, we use all observed user behaviors to re-train each model and use it to predict user behavior for the next week. We generate ROC curve for prediction result by

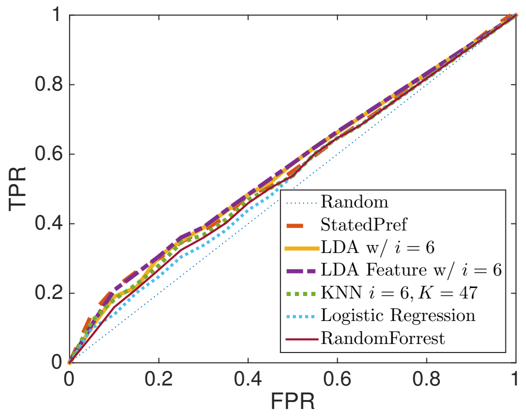


Figure 3.9: ROC Curve for Predictions of Receiver Behaviors. All ROC curves suggest that receiver behaviors are difficult to predict. The LDA model using both user features and observed behaviors (LDA feature w/ $i = 6$) has the best overall performance. A model with stated preferences (red dash line) has TPR a little higher than all other methods when FPR is low (below 0.1). This suggests that the model, on average, captures a receiver's the most preferred type of partner. However, its TPR is lower than LDA models using observed personal behaviors as FPR increases. This is because a receiver's stated preferences do not cover all his favorite type. KNN has its best performance when $K = 47$. Logistic regression and random forest use user features for classification without personalizing each receiver's own preference and have poor performances than others.

each model in each week. To better show the comparison of all models over time, we use the area under curve (AUC) as evaluation metric. AUC is computed as the area under ROC curve, ranging from 0 to 1. An AUC value of 1 means that a classifier makes a perfect classification. Random selection classifier has 0.5 AUC value.

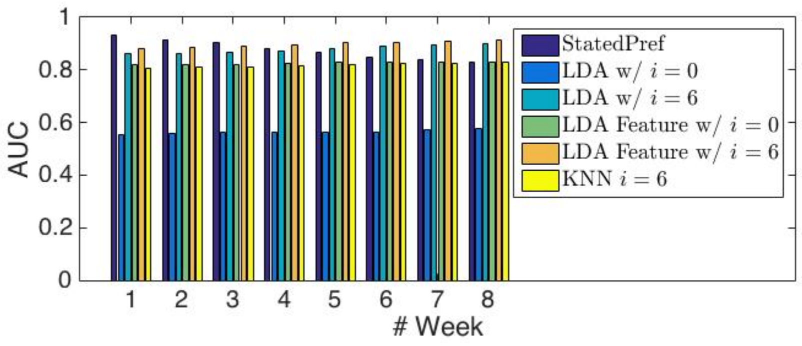


Figure 3.10: AUC for Sender Behavior Prediction by All Models. We adjust the time of some invitations such that all senders register in the first week. At the end of each week, models are trained with all previous data and predict sender behaviors for the next week.

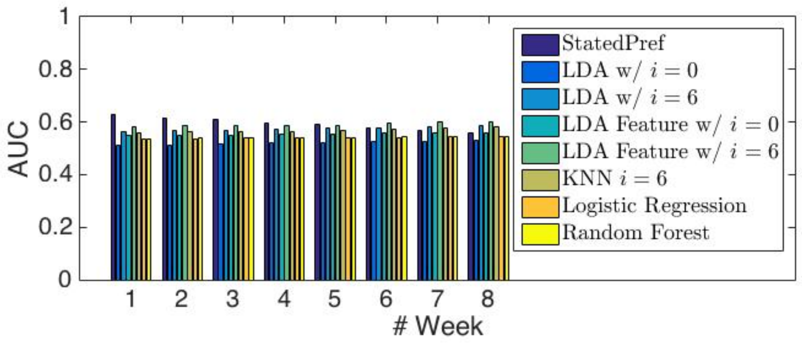


Figure 3.11: AUC for Receiver Behavior Prediction by All Models. We adjust the time of some invitations such that all receivers register in the first week. At the end of each week, models are trained with all previous data and predict receiver behaviors for the next week.

Figures 3.10 and 3.11 show AUCs for prediction results of both sender and receiver behavior. The model using stated preferences has the largest AUC for the first three weeks. This suggests that stated preferences are suitable for prediction for newly registered users. However, as users remain a customer of the website for a longer period, their preferences are likely to change, and the stated preference is less adaptive to these changes. On the other hand, other models exhibit robust performance as time increases.

3.6 Conclusion

In this Chapter, we propose an LDA model to predict edges in a bipartite network, where the node attributes can be observed and new edges emerge over time. We test our model by predicting users' behavior of sending/replying invitation in an online dating website.

In our experiments on multiple synthetic datasets, we show the performance of our model can be improved if the number of user preferences in our model is greater or equal to the ground truth. Our model also obtains a better approximation of ground truth user preferences if those preferences are easily distinguished using KL-divergence.

When applied to real-world online dating datasets where user behaviors change over time, we discovered that stated preferences have better performance in predicting newly registered users. However, as time increases, users change their behaviors, and stated preference is less effective in prediction. On the other hand, our model can capture the change in a user's personal behavior and has better performance than state-of-the-art methods such as KNN.

CHAPTER 4

TEMPORAL CLUSTERING IN DYNAMIC NETWORKS

4.1 Introduction

A dynamic network is a useful tool to study interactions between individuals in complex systems over time. Researchers usually represent it as a series of static network snapshots in discrete time. In each network snapshot, nodes represent individuals and edges represent relations or interactions between individuals in a time step. In practice, groups of nodes are observed to be densely connected with each other over time. These groups are called network communities. Clustering (community detection) in dynamic networks is an important task for understanding the dynamics of interactions within groups of individuals in complex systems. This is because clusters, with a correctly defined similarity metric for nodes, suggest "meaningful" network communities where nodes share common attributes in real world datasets. In social networks or communication networks, for example, clusters usually represent groups of close friends or frequent contacts. Tracking these clusters in a dynamic network helps us to understand not only the lifespan of friendships in a group of people but the dynamics of their formation and dissolution.

Current work mainly focuses on two different approaches to detect and track clusters. The first approach focuses on identifying clusters within each network snapshot and obtain their lifetimes based on the time steps when they are detected. Since there could be many clusters generated from all snapshots, previous work such as [20] only maintains the most frequently appearing clusters. Evolutionary clustering (EC) [18] applies K-means clustering on a similarity matrix generated from the current network snapshot and the clustering results of previous snapshots. Researchers have replaced the K-means clustering with different clustering methods [95, 56, 4, 45] for tracking the evolution of communities. However, these methods have a drawback: they use local information (more specifically, a similarity matrix generated from a subset of snapshots) to identify clusters. In practice, when dynamic networks are analyzed at small time granularities, their network snapshots are sparse and contain too few edges, the similarity matrices provide little information for clustering, and detected clusters are too small to provide enough information to understand relations between individuals in a group.

The other approach designs a model of a dynamic network with evolving latent structures and uses global information (all of the network snapshots) to learn those structures. It is proposed [32, 104] that a dynamic network consists of multiple stochastic block models (SBM) and each node has a mixed membership in those models. A Bayesian model is then applied to learn the mixture of memberships as well as the SMBs. However, since these methods assume network communities exist throughout time, they do not track community lifetimes. Tensor decomposition based methods such as [33, 60, 78] model a network as a three-mode tensor and apply low-rank tensor decomposition on it to obtain R components. Each component is considered as one of the latent structures and consists of three vectors termed "loading vectors". The loading vector related to nodes is used to generate communities and the loading vector related to time is used to track community lifetimes. However, previous analysis with tensor decomposition does not provide a well-defined physical interpretation of the vectors. More specifically, the meaning of the i-th element in the vector, considered to relate to a node or a time step i, is unclear. As a result, when analyzing at a very small granularity of time that causes snapshots to be sparse, they are unable to decide and verify the lifetime of a community because no snapshots contain a sufficient number of edges to indicate the time of existence of a community structure. The lack of physical interpretation of the vectors also makes it difficult to accurately calculate the lifetime. Previous methods present the plot of loading vector related to time, suggesting that the time steps of highest value are important but fail to provide a threshold for choosing those important time steps. Moreover, methods with latent structures require specifying the number of clusters. Current methods are not robust to changes in the community number and thus perform poorly.

To solve the problem that arises with sparse network snapshots, we propose a temporal clustering framework based on a set of latent network generative models, each of them consisting of several clusters. Given a time interval, edges between nodes in a cluster are generated at a specific rate. The time interval for the rate is independent of the time granularity of a dynamic network, so that the rate remains fixed with changes in time granularity. Nodes with higher edge-generation rates are considered more likely as a group. This enable our model to obtain the same clusters even when the time granularity of a network changes. We assume the rate changes at a speed slower than the network structure and use it to detect the formation, dissolution and lifetime of the clusters.

4.2 Contribution

The major contributions of this chapter include:

- We represent a temporal network as a three-way tensor with the similarity of nodes defined as a function of time base on the edge generation rate learned from tensor decomposition. K-means clustering and silhouette criterion are applied to detect network communities. Experimental results show that our method, compared with baseline methods, is more robust to change in the number of clusters.

- We propose a similarity ordering score to improve temporal clustering methods on a precision-recall metric for community detection and community member detection.

- We also apply filtering methods for smoothing the rate of edge generation in a cluster and propose a bottom-up segmentation algorithm for detecting community formation, dissolution and lifetime. We provide evidence that the performance of our model is robust to changes in time granularity and the density of network snapshots.

4.3 Model Description and Problem Fomulation

We analyze a dyanmic network in discrete time and represent it as a series of static network snapshots, denoted by $G = \{G_t(V, E_t)|t = 1, \cdots, T\}$, where G_t is the network snapshot at time t, V is a fixed set of nodes and E_t is the set of edges at time t, and T is the total number of network snapshots at the finest time granularity. We use $w = 1$ to represent the finest time granularity and $w = T$ for the coarsest time granularity (in which case, the network is represented as a static network). For simplicity, we explain our model at the finest time scale in the following sections and investigate the effect of time granularity on community detection in Section 4.5.

We propose a temporal clustering method (TC) such that edges in G are generated by R generative models, denoted by $\{X^{(r)}\}_{r=1}^{R} = \{A_r, \lambda^{(r)}(t)\}$, where $A_r = [a_{1r}, \ldots, a_{|V|r}]$ and a_{ir} is the probability that node i belongs to a community whose members and lifetime can be modeled in $\{X^{(r)}\}$, for $r = 1, \cdots, R$. $\lambda^{(r)}(t)$ is defined as edge-generation rate and represents the rate at which edges are generated between two nodes within a community from $X^{(r)}$. It can vary over time and can be represented as a time series or a function of time t (details are provided in Section 4.4.4). At time step t , the r-th generative model $X^{(r)}$ generates edges between two nodes i, j with rate

$$a_{ir}a_{jr}\lambda^{(r)}(t) \tag{4.1}$$

(In one time step, $a_{ir}a_{jr}\lambda^{(r)}(t)$ can be interpreted as the expected number edges generated from $X^{(r)}$). The total expected number of edges between node i and j at time t in G is then computed as

$$\sum_{r=1}^{R} a_{ir}a_{jr}\lambda^{(r)}(t). \tag{4.2}$$

Since a community is a group of nodes that are densely connected, we assume that a generative model $X^{(r)}$ tends to generate more edges within a community. We use $a_{ir}a_{jr}$ as the similarity between node i and j in $X^{(r)}$ to apply to a clustering algorithm for community detection. $\lambda^{(r)}(t)$ also affects the number of edges between nodes and can be used to detect the lifetime of a community: A small $\lambda^{(r)}(t)$ result in less edges in a group of nodes, suggesting that a community has dissolved (We provide more details in Section 4.4.4.)

Note that communities from different generative models can contain the same node. This allow a node to have multi-membership in different communities at the same time step.

4.3.1 Problem Formulation

We use a three-mode tensor $\mathcal{X}$ to represent a dynamic graph G, where $\mathcal{X}_{ijt}$ represents the number of edges observed between nodes i and j at time t. The problem of tracking evolving communities in a dynamic network can be formulated as learning R generative models, $\{X^{(r)}\}_{r=1}^{R}$, such that $\mathcal{X}_{ijt}$ can be approximated as $\sum_{r=1}^{R} a_{ir}a_{jr}\lambda^{(r)}(t)$ with minimum square error:

$$\begin{aligned} &\min \textstyle\sum_{i,j\in V, i\neq j} \sum_t (\mathcal{X}_{ijt} - \sum_{r=1}^{R} a_{ir}a_{jr}\lambda^{(r)}(t))^2 \\ \text{s.t}\quad & 0 \le a_{ir} \le 1;\ \text{for } i = 1,\dots,|V|, r = 1,\dots,R \\ & \lambda^{(r)}(t) \ge 0 \end{aligned} \tag{4.3}$$

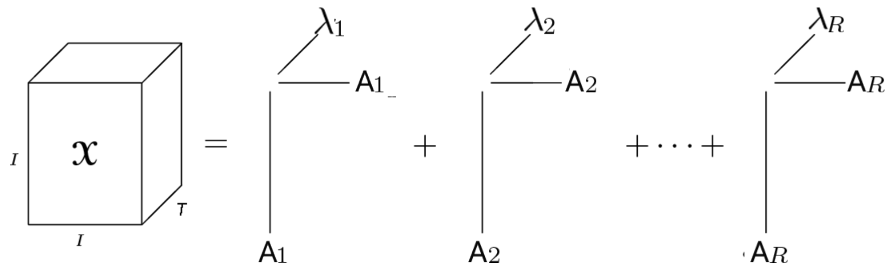


Figure 4.1: Learning temporal clustering model using PARAFAC decomposition. A dynamic network is represented as a three-mode $I \times I \times T$ tensor $\mathcal{X}$. PARAFAC decomposition decomposes $\mathcal{X}$ into R components that represents generative models $\{X^{(r)}\}_{r=1}^{R}$

We assume no self-loop in a dynamic network and add a constraint that $i \neq j$. If self-loops are allowed, we can extend the model by removing the constraint $i \neq j$ from the objective (4.3).

4.4 Temporal Clustering

Our approach of temporal clustering to detect evolving communities in a dynamic network consists of the following steps: (1) learning R generative models, (2) clustering in generativ models, (3) ranking cluster, and (4) identifying community lifetimes. We provide details on each step in this section.

4.4.1 Learning Generative Model with PARAFAC Decomposition

We apply parallel factorization (PARAFAC) decomposition to an $I \times I \times T$ tensor $\mathcal{X}$ that represents a dynamic network. Generative models that satisfy (4.3) can be obtained directly from the decomposition result (Figure 4.1). Alternative least squares (ALS) algorithm is applied to perform PARAFAC. We provide details about PARAFAC decompostion in Appendix A.

In the rest of the chapter, we use $\hat{X}^{(r)} = \{A_r, \vec{\lambda_r}\}$ to denote the r-th generative model learned from PARAFAC, where $a_{ir} \in A_r$ is the probability that node i belongs to $\hat{X}^{(r)}$, $\vec{\lambda_r} = [\lambda_{1,r}, \ldots, \lambda_{T,r}]$ is a sequence of T samples for the edge-generation rate

$\lambda^{(r)}(t)$. Note that all current methods for learning latent structures require setting the number of structures either manually or via criteria such as the elbow rule[44] and modularity gain[70]. For PARAFAC decomposition, we select R according to *core consistency* [14]. However, the rank R produced this way is not necessarily consistent with the data. Moreover, PARAFAC decomposition requires large computation power, so that R has to be small for large datasets. As a result, errors may occur in the components, making them unsuitable for analysis. For example, in an experiment by Gauvin et al. [33], one of the components from PARAFAC actually contains two independent communities, but the methods group the two communities as one. We deal with this issue by performing clustering on $X^{(r)}$ and propose a ranking score to retain the significant clusters.

4.4.2 Dynamic Community Detection with Generative Models

We allow $X^{(r)}$ to contain one or more clusters. The similarity between two nodes, i and j in $X^{(r)}$, is defined as:

$$s_{i,j}^{(r)} = a_{ir}a_{jr} \tag{4.4}$$

Clustering methods such as K-means clustering and spectral clustering return disjoint clusters. Let $C_m^{(r)}$ be the m-th cluster obtained from generative model $X^{(r)}$. When our model applies one of these methods, $|C_m^{(r)} \cap C_n^{(r)}|= 0$. However, clusters across different generative models may overlap: $|C_m^{(r)} \cap C_n^{(s)}|\geq 0$, for $r \neq s$. As a result, a node may have multi-membership in our temporal clustering method.

After obtaining R latent generative models, we perform clustering on each model $\hat{X}^{(r)}$ using the similarity defined in (4.4). There are many clustering algorithms such as K-means and spectral clustering[74], as well as community detection algorithm such as Girvan−Newman algorithm [71]. We choose K-means clustering with silhouette criterion to determine the best number of clusters. There are mainly two reasons for this. First, K-means clustering is fast, simple and easy to implement.

With silhouette score, we can automatically decide K and obtain a good clustering result. Second,other clustering algorithms do not improve performance over K-means clustering in our experiments.

Silhouette Clustering Criterion The silhouette score measures how well a point lies within its cluster [87]. Assume that we have K clusters $\{C_k\}, (k = 1, \cdots, K)$ and a similarity function $s(i, j)$ for data points i and j. The similarity of a node i to a cluster C_k is defined as

$$d(i, C_k) = \sum_{j \in C_k} s(i, j)/|C_k| \ .$$

$d(i, C_k)$ represents the average similarity of node i to all other nodes in cluster C_k. Suppose node i is in cluster C_k, then the *silhouette score* of i is

$$S(i) = \frac{d(i, C_k) - \max_{l \neq k} d(i, C_l)}{\max_m d(i, C_m)},$$

which lies in the interval $[-1, 1]$ Here, $S(i)$ measures how well i belongs to its own cluster, with positive values indicating good clustering.

Finally, the average silhouette score over all data points is a measure of the quality of the K clusters, computed as

$$\bar{s} = \sum_{i=1}^{|V|} S(i)/|V|.$$

The silhouette criterion is to choose K with the highest average silhouette score $\bar{s}$.

Note that in a generative model $X^{(r)}$, a set of nodes with low probability belonging to a community in $X^{(r)}$ are grouped into a cluster, denoted by $C = \{i | a_{ir} \approx 0\}$. These nodes generate almost no edges in our model and do not form a network community. We introduce a ranking score to filter these clusters in the next section.

4.4.3 Cluster Ranking

Clusters detected from our methods may include groups that have few edges, these groups are not network communities and should be filtered. Suppose K clusters are detected from generative model $X^{(r)}$ and denoted by $\{C_m^{(r)}\}_{m=1}^K$, we rank this clusters by their *average similarity ordering (SO)* score, which we define as:

$$\begin{aligned} SO_m^{(r)} &= \frac{\sum_{i,j \in C_m^{(r)}} a_{ir} a_{jr}}{|C_m^{(r)}|^2} \int_0^T \lambda^{(r)}(z) dz, & (4.5) \\ &\approx \frac{\sum_{i,j \in C_m^{(r)}} a_{ir} a_{jr}}{|C_m^{(r)}|^2} (\sum_{t=1}^{T} \hat{\lambda}_{t,r}), & (4.6) \end{aligned}$$

where $\hat{\lambda}_{t,r}$ is the edge-generation rate obtained from $\hat{X}^{(r)}$, as introduced in Section 4.4.1. The SO score represents, on average, the number of edges generated from a node-pair in a cluster over time. We generate a ranked list of clusters with SO score in decreasing order. Clusters at the top of the list are considered to be significant and provide useful information for analysis.

Moreover, the elbow rule [44] can be applied to the SO score to filter out insignificant clusters: clusters are ranked in a descending order of SO score, and we choose the rank at which the SO score drops significantly and remove clusters below it.

4.4.4 Lifetime Detection

We use $\vec{\lambda}_r$ from PARAFAC as an estimate for the edge-generation rate $\lambda^{(r)}(t)$ for generative model $X^{(r)}$ for $r = 1, \ldots, R$. In this section, we first provide an adaptive threshold to determine if a time step is in the lifetime of a community. We then model $\lambda^{(r)}(t)$ as a piecewise linear function computed by a time series segmentation algorithm. The formation or dissolution of a cluster can be defined as a time segment when $\lambda^{(r)}(t)$ increases or decreases.

4.4.4.1 Lifetime Threshold

We determine that a time step t belongs to the lifetime of a network community if the nodes within the community are densely connected . We first provide measures for the edge density in a network community.

Definition 7. *An* Average Edge-generation Rate for cluster $C_m^{(r)}$ at time t *is the averge number of expected edges generated between two nodes in a cluster, denoted by* $C_m^{(r)}$*, at time* t*:*

$$\lambda_m^{(r)}(t) = \frac{1}{|C_m^{(r)}|^2} \sum_{i,j \in C_m^{(r)}} a_{ir} a_{jr} \lambda^{(r)}(t) \tag{4.7}$$

Definition 8. *An* Average Edge-generation Rate for Network G at time t *is the average of expected number of edges generated between nodes in a dynamic network* G *at time* t*:*

$$\lambda(t) = \frac{1}{|V|^2} \sum_{r=1}^{R} \sum_{i,j \in V} a_{ir} a_{jr} \lambda^{(r)}(t) \tag{4.8}$$

Our model decides that a cluster $C_m^{(r)}$ exists at time t if a node pair in $C_m^{(r)}$, on average, generates more edges than a randomly selected node pair from the network. The average edge-generation rate of node pairs in the network, $\lambda(t)$ (4.8), serves as a threshold to determine the lifetime of a cluster. We define the lifetime of $C_m^{(r)}$ as:

$$L_m^{(r)} = \{t | \lambda_m^{(r)}(t) > \lambda(t)\} \tag{4.9}$$

where $\lambda_m^{(r)}(t)$ is the edge-generation rate of $C_m^{(r)}$ defined in (4.7), $\lambda(t)$ can be easily obtained by applying Sliding Window Filtering on the sequence $[\frac{\sum_{i,j=1}^{|V|} \mathcal{X}_{ijt}}{|V^2|}]$ for $t = 1, \cdots, T$.

4.4.4.2 Formation and Dissolution of a Community

Let $C_m^{(r)}$ be the m-th community detected from the r-th generative model, we define the formation of $C_m^{(r)}$ as a time period when its edge-generation function, denoted by $\lambda_m^{(r)}(t)$, is increasing. During the formation period, the expected number of edges between nodes increases according to (4.1). Similarly, the dissolution of $C_m^{(r)}$ is defined as a time period when $\lambda_m^{(r)}(t)$ decreases.

We learn $\lambda_m^{(r)}(t)$ by constructing a piecewise linear function, denoted by $f_D(t)$, from $\vec{\lambda_r}$ obtained from PARAFAC decomposition. A *piecewise linear function with D segments* is defined as:

$$f_D(t) = \begin{cases} b_1 t + c_1, & t \in Q_1, \\ b_2 t + c_2, & t \in Q_2, \\ \vdots & \vdots, \\ b_D t + c_D, & t \in Q_D. \end{cases} \tag{4.10}$$

where b_i, c_i are constants, $Q_i = [q_i, q_{i+1}]$ is an interval between time q_i and q_{i+1} and $q_i < q_{i+1}$ for $i = 1, \ldots, D$. We can determine that a time interval Q_i is a community formation period if the constant $b_i > 0$ for $1 \leq i \leq D$.

Constructing a piecewise linear function from a time series is a popular problem in time series study. Models for this problem requires setting hyper-parameters such as number of segments and maximum approximation error to obtain the best result. We design a bottom up algorithm for this problem without the requirement of setting hyper-parameters. Details are provided in Appendix B.

4.5 Evaluation

In this section, we evaluate our temporal clustering (TC) method on both synthetic and real-world datasets and compare it with baseline methods. We first use synthetic data to study the effect of R (the number of generative models in TC) on the result of PARAFAC decomposition. Then we compare the performance of TC to those of two baseline methods, evolutionary clustering (EC) and tensor decomposition with binary clustering (BC), in detecting network communities in both synthetic and real-world datasets.

4.5.1 Clustering Error Caused by Low Rank Decomposition

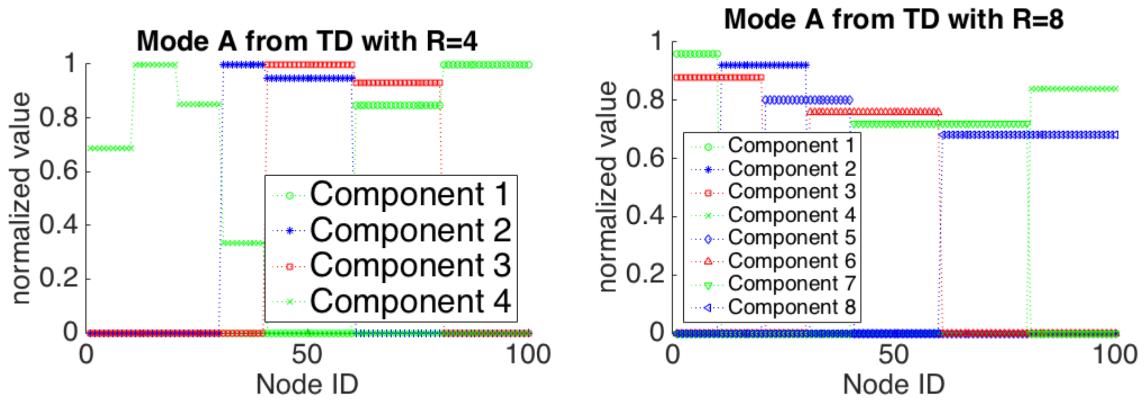
Previous work with PARAFAC decomposition, such as [33, 60], uses a component to represent a community. However, PARAFAC decomposition is NP-hard and current algorithms only provide approximate solutions. This causes error in a generative model $X^{(r)}$, especially when the number of components, R, is improperly chosen. However, little is known on how that error affects clustering result.

We begin by applying our method to a synthetic dataset to understand how R, the number of generative models causes errors in $X^{(r)}$ and affects the community detection result. The synthetic dataset is a dynamic graph with 100 nodes. For the first 50 time steps, nodes 1-10, 11-30, 31-60 and 61-100 form four disjoint cliques (denoted by $C_1^* - C_4^*$). Then for the next 50 steps, nodes 1-20, 21-40, 41-80 and 81-100 form another set of disjoint cliques (denoted $C_5^* - C_8^*$). $\mathcal{X}_{ijk} = 1$ if node v_i, v_j belongs to the same clique at time step k, otherwise $\mathcal{X}_{ijk} = 0$.

We apply our method on the dataset with $R = 2, \ldots, 8$. Generative models are learned from the components of the PARAFAC decompostion results. Figure 4.2 shows loading vectors A_r from selective decomposition rank $R = 4, 8$, for $r = 1, \ldots, R$. When $R = 4$ (Fig 4.2(a)), PARAFAC has a large approximation error in objective (A.1). It causes a generative model to include multiple cliques. For example, loading vector from component 4 plotted with "x" represents a mixture of C_1^*, C_2^*, C_5^* andC_6^*. Applying BC in [33] results in a cluster consisting of nodes 1-30, which do not map to any ground truth clique in the data. When $R = 8$, PARAFAC has no error as is shown in Fig 4.2(b) where the value of A_r is normalized to $[0, 1 - 0.04r]$ for clear illustration, for $r = 1, \ldots, R$. It is clear that a cluster generated from nodes with high value in A_r can be mapped to a ground truth clique C_r, for $r = 1, \ldots, 8$.

In summary, the effects of R on clustering on TD components are:

- If R is less than the number of ground truth clusters, a component from PARAFAC may include multiple different clusters of nodes that are densely



(a) R=4 (b) R=8

Figure 4.2: Plot of loading vector A_r: (a) PARAFAC with small R generates loading vectors A_r that fail to represent ground truth communities; (b) PARAFAC with proper R provides loading vectors that represent ground truth communities.

connected. However, the elements a_{ir} in A_r for the nodes v_i that belong to the union of those cluster, have similar values. In this case, those clusters are difficult to distinguish and it is not suitable to use models (such as BC) that consider a component as a single community.

- As R increases, a component tends to contain fewer clusters of nodes that are densely connected and may map to ground truth communities.

4.5.2 Baseline and Metrics

4.5.2.1 Baseline methods

EC is a commonly used method to cluster nodes in dynamic networks. Researchers have proposed multiple methods to improve its performance [95, 56, 4, 45, 101]. It operates as follows: Let $\mathcal{X}_t = \mathcal{X}_{(:,:,t)}$ be a similarity matrix for graph G_t of N nodes. EC cluster nodes based on the similarity matrix $\mathcal{R}_t = (1-\beta)\mathcal{X}_t + \beta\mathcal{X}_{t-1}$ at time t to obtain a set of clusters $\mathcal{C}_t = [\mathcal{C}_{1,t}, \ldots, \mathcal{C}_{N,t}]$, where $\beta \in [0,1]$ and $\mathcal{C}_{i,t}$ is the index of the cluster that node i belongs to at time t. Since $\mathcal{R}_t$ differs from the current similarity matrix $\mathcal{X}_t$, $\mathcal{C}_t$ is not necessarily the best set of clusters for $\mathcal{X}_t$, a snapshot quality function $sq(\mathcal{C}_t, \mathcal{X}_t)$ is used to evaluate the clusters. EC includes a historical cost function $hc(\mathcal{C}_t, \mathcal{X}_{t-1})$ to calculate the difference between the current and previous clustering results. The objective is to minimize

$$\sum_{t=1}^{T} sq(\mathcal{C}_t, \mathcal{X}_t) - c\sum_{t=2}^{T} hc(\mathcal{C}_t, \mathcal{X}_{t-1})$$

where c is a scalar.

BC is another method to detect network structures in dynamic networks. Its applications include anomaly network event detection [60], community detection in networks from sensor data[33] and social networks [78]. It considers each component $X^{(r)}$ as a community. It chooses a threshold for binary classification: For a node i, if a_{ir} from the loading vector A_r is larger than the threshold, then i belongs to that community.

4.5.2.2 Metrics for Clustering and Lifetime Detection

In this subsection, we introduce multiple metrics for the performance in community detection, community member detection and community lifetime detection in dynamic networks. These metric includes precision, recall, F1 score, and precision-recall (PR) curve.

Community detection metric For community detection, we map a detected community to a ground truth community using the Jaccard index (refer to detailed steps in Appendix C).

If a method detects P out of N ground truth communities, the *community recall* is $\mathcal{R} = \frac{P}{N}$. Suppose M communities are identified as meaningful by the method, then the *community precision* is $\mathcal{P} = \frac{P}{M}$.

Community member detection metric Suppose a detected community $\hat{C}_k$ is mapped to ground truth community C_n^*, the *community member precision* is $\mathcal{P} = \frac{|\hat{C}_k \cap C_n^*|}{|\hat{C}_k|}$, the *member recall* is $\mathcal{R} = \frac{|\hat{C}_k \cap C_n^*|}{|C_n^*|}$.

Community lifetime detection metric For community lifetime detection, we consider the finest time granularity. Let L_n^* be a set of time steps that represents the lifetime of a ground truth community C_n^* and $\hat{L}_k$ (computed from (4.9)) is the lifetime of a detected community mapped to C_n^*. the *lifetime precision* of C_n^* is $\mathcal{P} = \frac{|\hat{L}_k \cap L_n^*|}{|\hat{L}_k|}$, the *lifetime recall* is $\mathcal{R} = \frac{|\hat{L}_k \cap L_n^*|}{|L_n^*|}$

F1 score If some methods have high precision but low recall in their results while other methods have high recall but low precision, the *F1 score* is used in this situation to give an overall evaluation. It is defined as the harmonic mean of $\mathcal{R}$ and $\mathcal{P}$: $\mathcal{F}_1 = \frac{2\mathcal{P}\mathcal{R}}{\mathcal{P}+\mathcal{R}}$.

PR curve A PR curve is usually applied when the compared methods have similar recall values. We mainly use PR curve to evaluate community member detection. A PR curve requires a clustering method to return a ranked list of clusters. Suppose the ranked list has K clusters, $\{\hat{C}_i\}_{i=1}^k$ are top k clusters in the list and are mapped to k out of N ground truth clusters $\{C_{n_i}^*\}_{i=1}^k, 1 \leq n_i \leq N$. For $k = 1, \cdots, K$, we define precision of the top k clusters as $\mathcal{P}_k = \frac{\sum_{i=1}^k |\hat{C}_i \cap C_{n_i}^*|}{\sum_{i=1}^k |\hat{C}_i|}$ and recall as $\mathcal{R}_k = \frac{\sum_{i=1}^k |\hat{C}_i \cap C_{n_i}^*|}{\sum_{j=1}^N |C_j^*|}$. We generate a PR curve by plotting $(\mathcal{P}_k,\ \mathcal{R}_k)$ in the precision-recall space. $\mathcal{R}_K$ is defined as the *community member recall for a dynamic network.*

Our method can return a ranked list of communities using SO score. To generate a ranked list from BC, let $A_r = [a_{1r}, \ldots, a_{|V|r}]$ denote the loading vector in the rth component $X^{(r)}$. BC classifies a node i as a member in a community of interest, $\hat{C}_1^{(r)}$, if a_{ir} is larger than a threshold. Otherwise i is put in another community $\hat{C}_2^{(r)}$ that can be ignored. Components $X^{(r)}$ are placed in descending order of their Frobenius norm, for $r = 1, \ldots, R$. $\hat{C}_1^{(r)}$ is believed to have stronger structures than $\hat{C}_1^{(r')}$ if $r < r'$ [33]. A ranked list from BC can be generated as: $[\hat{C}_1^{(1)}, \ldots, \hat{C}_1^{(R)}, \hat{C}_2^{(1)}, \ldots, \hat{C}_2^{(R)}]$

4.5.3 Evaluation Results

We use randomly generated dynamic networks with multiple clusters to show that TC is more robust to changes in R, sparsity of network and time granularities when compared to baselines such as EC and BC. We also test our method in a mobility network generated from the Lakehurst mobility trace dataset from [20]. Experimental results show that our SO score improve precision of a method given the same recall when detecting community members. Finally, we applied TC to a dynamic network generated from Enron email dataset[82] and discover differences in email-exchange behavior between communities.

4.5.3.1 Synthetic networks

We generate 5040 synthetic dynamic networks with sizes ranging from 100 to 500, and with number of snapshots $T \in [1000, 4000]$. Clusters in a network may have common nodes and their sizes vary from 8 to 80. The number of clusters in a network is between 10 and 40.

For each cluster, we split a time interval $[1, T]$ into multiple segments and choose a fraction of those segments as the lifetime of the cluster. During each lifetime segment, we set the edge-generation rate as either a constant or a linear function, with its value varying from 0.0015 to 1. We set the lifetime of a cluster to be at least $0.3T$ time steps such that there are enough edges generated from the cluster for clustering. We also set up periodic lifetimes for some clusters. The periods range from 20 time steps to $\frac{T}{2}$.

We also add noise to each network: each node pair in a network generates an edge with rate e during a period of one time step, where e is sampled uniformly from $[0, 0.01]$.

To examine the effect of time granularity on the performance of TC, we represent those dynamic networks under different time granularities. In particular, we consider networks where w consecutive snapshots are aggregated together. In such case, an original network consisting of T snapshots is transformed into one consisting of T/w snapshots. At the smallest granularity, let $\mathcal{X}_{ijt}$ be the weight or number of edges between node i, and j at time step t. At time granularity w, the weight of an edge between node i and j is defined as:

$$\mathcal{X}_{ijt'}(w) = \sum_{t=(t'-1)w}^{t'w} \mathcal{X}_{ijt}$$

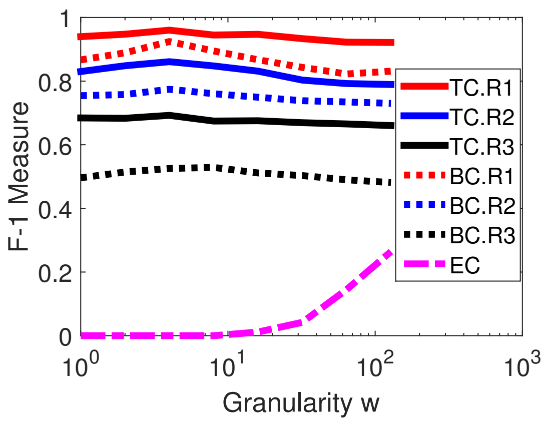
.

We define the density of a community C_k at t as $d_k(t) = \frac{2\sum_{i,j\in C_k}(a_{ir}a_{jr}\lambda^{(r)}(t))}{|C_k|^2}$, and use d_k to denote the average density throughout lifetime. Average community densities in our synthetic data vary from 0.0008 to 0.9341. We split networks into 10 groups according to their average community density. Networks in the i-th group have average community density within $[(i-1)\times 0.1, i\times 0.1]$.

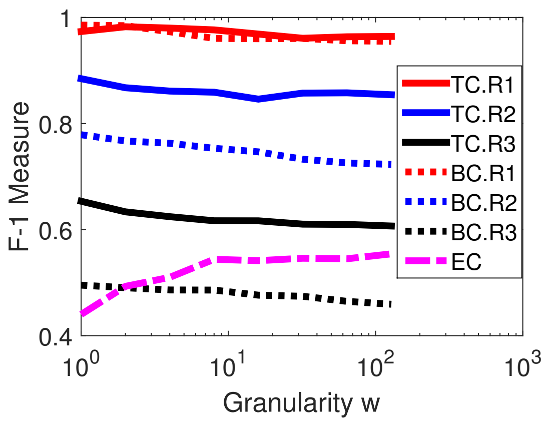
Experiment Setup. Let K denote the number of ground truth clusters in a synthetic network, we set the rank $R = 0.3K, 0.4K, \ldots, K$ for BC and our temporal clustering method. For time granularity, we set $w = 2^q$ for $q = 0, \ldots, 7$.

We run EC algorithm using publicly available software [101], and use the tensor toolbox [9] for PARAFAC decompositon on a cluster of 32 Intel Xeon E5-2670@2.60GHz CPUs with 256Gb RAM.

Community Detection. We first use F1 score to evaluate the performance of methods that retrieve communities in networks given different granularity w. Figure 4.3 shows the change in F1 score of clusters as a function of time granularity. In Figure 4.3(a), the three methods identify clusters in networks whose average community densities falls in interval $[0.3, 0.4]$. The F1 scores for both BC and our temporal lustering method (TC) change little with the increase of granularity. EC retrieves no ground truth clusters when the granularity is small, but begin to identify clusters as granularity increases. BC and TC perform better when the rank R is set to K.



(a) Average Community Density in $(0.3, 0.4)$

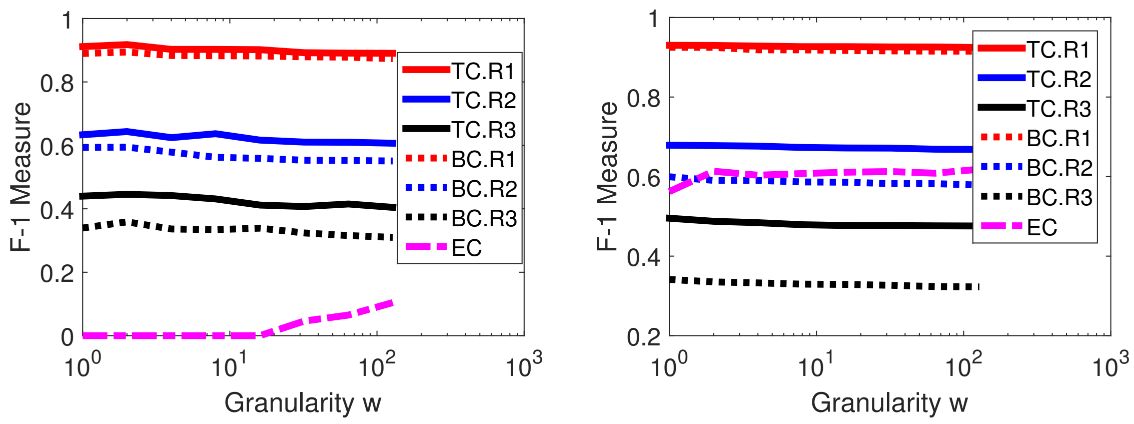


(b) Average Community Density in $(0.7, 0.8)$

Figure 4.3: F1 Score for Community Detection. K is the number of ground truth communities in a network, BC and TC apply PARAFAC with rank $R_1 = K$, $R_2 = 0.8K$, and $R_3 = 0.6K$. Both BC and TC are robust to the changes in w. TC is more robust to the change in R when compared to BC.

As R decreases to $0.6K$ and $0.8K$, the F1 scores of both methods drop significantly. BC is more sensitive to change in R. Figure 4.3(b) illustrates the result on networks whose average community density ranges from 0.7 to 0.8. Compared to Figure 4.3(a), the performances of BC and TC are similar, but the F1 score for EC improves as community densities increase. Figure 4.3 suggests that EC is sensitive to the change in granularity and the community density when detecting communities in dynamic network, while both BC and TC are more robust than EC. TC performance similar to that of BC when R is chosen properly, and is more robust in performance than BC when R decreases.

Community Member Detection. Figure 4.4 illustrates the F1 scores for community member detection by EC, BC and TC. TC and BC both have F1 scores larger than that of EC. TC has better F1 score than BC and is more robust to the changes in R. In Figure 4.4(a), F1 scores are computed for communities with community density $d_k \in [0.3, 0.4]$ across all synthetic networks. Figure 4.4(b) shows the results for communities with densities within $[0.9, 1]$. EC performs much better on communities



(a) Average Community Density in $(0.7, 0.8)$ (b) Average Community Density in $(0.9, 1)$

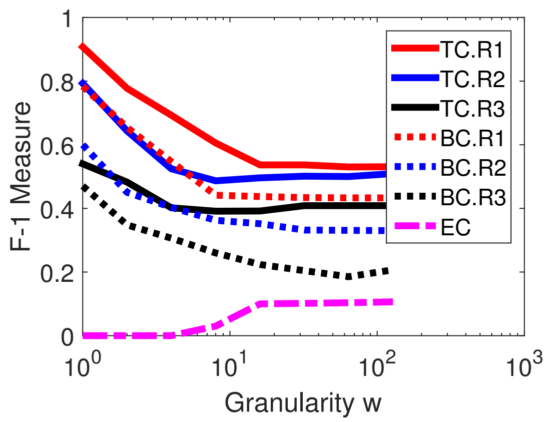
Figure 4.4: F1 Measure for Community Member Identification. BC and TC apply PARAFAC with rank $R_1 = K$, $R_2 = 0.8K$, and $R_3 = 0.6K$

with high densities than at low densities. Since we allow clusters to share common nodes in the network, EC only retrieves disjoint clusters, its F1 score is lower than that of TC and BC.

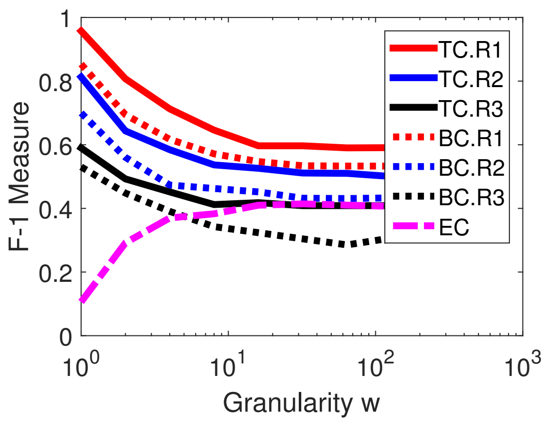
Lifetime Detection

Figure 4.5 shows the F1 score of lifetime detection for clusters with densities within $[0.7, 0.8]$ and $[0.9, 1]$. The F1 score of both BC and TC decreases as granularity increases from 1 to about 10, suggesting that lifetime detection by TC and BC is sensitive to changes in time granularity. When granularity continues to increase, the performances of all methods remain stable. Lifetime detection by EC benefits from the increase in time granularity, but accuracy stops improving when granularity becomes large.

In order to understand why time granularity affects the performance in community lifetime detection, we compare clusters with high lifetime detection accuracies to those with low accuracies at coarse granularities. Figure 4.6 shows the detection of periodicity in lifetime for two ground truth communities C_1^* and C_2^* from the same dynamic network under granularities $w = 1$ and $w = 128$. Both communities



(a) Average Community Density in $(0.7, 0.8)$



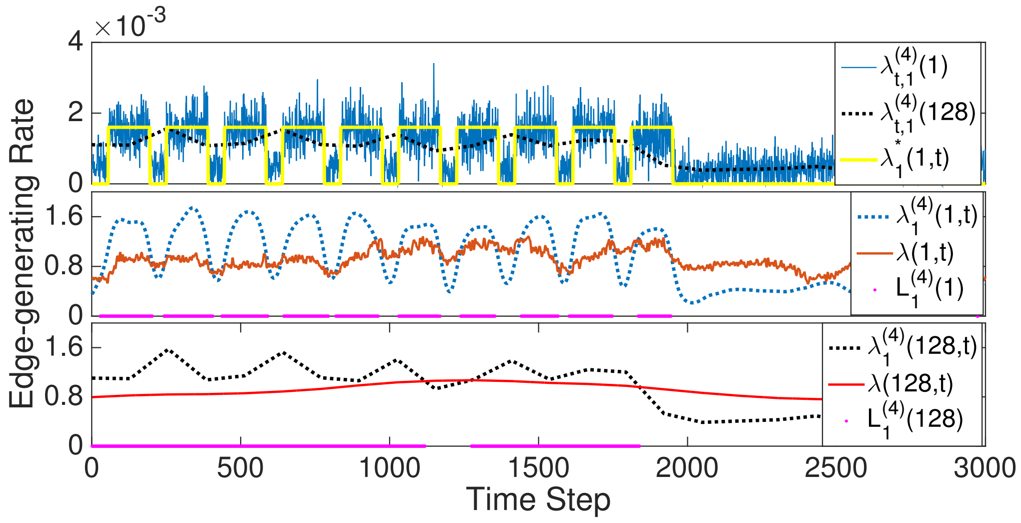
(b) Average Community Density in $(0.9, 1)$

Figure 4.5: F1 Measure for Lifetime Detection. BC and TC apply PARAFAC with rank $R_1 = K$, $R_2 = 0.8K$, and $R_3 = 0.6K$

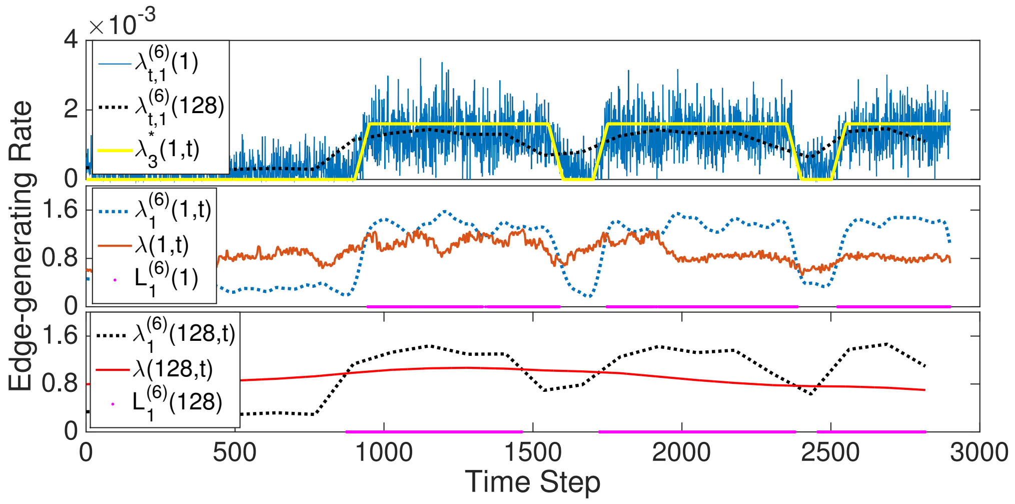
have the same edge-generation rates. C_1^* forms and dissolves frequently throughout time (Figure 4.6(a)). The ground truth edge-generation rate is denoted as $\lambda_1^*(1, t)$. The time series obtained from TC at granularity w is $\lambda_{1,t}(w)$, for $w = 1, 128$. At granularity w, we reconstruct the edge-generation rate as $\lambda_1(w, t)$ using $\lambda_{1,t}(w)$. $\lambda(t)$ in Eq(4.8) is denoted as $\lambda(w, t)$. The lifetime of C_1^*, denoted as $L_1(w)$, is obtained with Eq(4.9). Note that the period of the lifetime of C_1^* is small and we can maintain the periodicity at $w = 1$. At $w = 128$, however, lifetime $L_1(128)$ is not accurate and the periodicity is lost. Figure 4.6(b) illustrates the lifetime of community C_2^*, which forms and dissolves less frequently (with period larger than that of C_1^*). The periodicity at both granularities $w = 1$ and 128 can be maintained.

4.5.3.2 Lakehurst Data

The Lakehurst dataset from the US Army Research Lab contains a three hour (10800 seconds) trace of 70 vehicles (ground and airborne) [20]. 64 vehicles are split into 9 platoons and the other six move separately. The platoons moves from one checkpoint to another. All vehicles start from the same origin and reach the same destination at the end. There are two paths from the start location to the destination



(a) Lifetime Detection of C_1^* at $w = 1, 128$



(b) Lifetime Detection of C_2^* at $w = 1, 128$

Figure 4.6: Periodic Lifetime Detection for Two Clusters C_1^* and C_2^* at Granularity $w = 1, 128$. (a) Periodic lifetime with short period loses periodicsity at large time granularity $w = 128$. (b) Lifetime with long period maintains its periodicity at $w = 128$.

and each path has five checkpoints. A platoon chooses one path and stops for a while at each checkpoint until it reaches the destination. The other six vehicles sometimes intersect with one of those platoons.

The dataset contains the location of each vehicle at every second. Vehicles in a platoon move together and two vehicles in the same platoon are within 150 meters of each other for 99% of the time. We create a dynamic graph by adding an edge between two vehicles if they are within 150 meters of each other at each time step. In each snapshot of the graph, nodes in the same platoon tend to have an edge. The average density of the adjacency matrix of a snapshot is 0.235 and the maximum is 0.688. The network structure changes slowly, sometimes it remains unchanged for 3 minutes. We construct a $70 \times 70 \times 10800$ tensor. Some platoons meet one another and periodically form a larger community for a period of time. There are 10 such clusters. As a result, there are 19 ground truth clusters appearing in the dynamic network. The size of clusters ranges from 5 to 25.

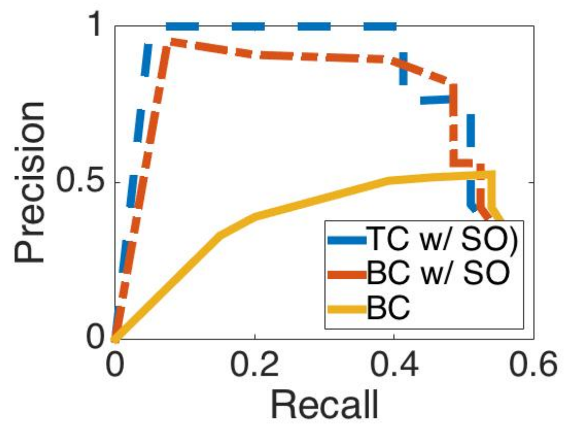
Since platoons have intersecting paths and sometimes forms larger clusters only for seconds, we are interested in the number of ground truth clusters that a method obtains. We use community recall and community member recall to evaluate the performance of EC, BC and TC. We applied PARAFAC decomposition with $R = 10, 15, 20, 25$ and 30. The recall of ground truth clusters and nodes are illustrated in Table 4.1.

The snapshots in this dataset are dense. EC detects clusters and community members even at the smallest time granularity. However, it only retrieves the nine ground truth platoons and fails to obtain the larger clusters formed by multiple platoons.

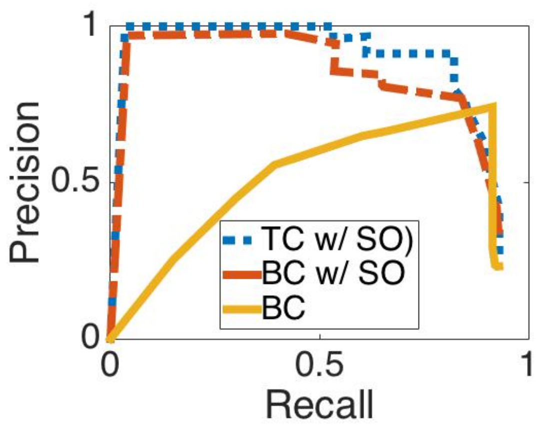
Table 4.1: Clusters Recall and Member Recall for Lakehurst Data

	Temporal Clustering					PARAFAC w/ BC					EC
R	10	15	20	25	30	10	15	20	25	30	N/A
Cluster Recall	0.368	0.421	0.579	0.895	1.0	0.319	0.421	0.579	0.895	1.0	0.474
Member Recall	0.430	0.541	0.841	0.875	0.904	0.403	0.532	0.841	0.862	0.904	0.275

Since both TC and PARAFAC with BC have similar community recall and member recall, we use PR curves to compare their performance. We also want to test how SO score affects performances. Fig 4.7 illustrates the PR curves generated from the results of TC with SO ranking and BC with rank $R = 15, 30$. Both sub-figures show that SO ranking in TC has better precision than BC when the recall is less than 0.5, suggesting the ordered list from TC is more accurate than BC. By applying SO score with BC, the performance of BC also improves significantly. The reason is that in a component with larger norm, some nodes i do not belong to any community, but have positive values a_{ir} larger than the threshold in BC. BC simply considers that a component only contains one meaningful community and mistakenly classifies i as a community member. Our method separates nodes into multiple communities and gives low rankings to communities with small SO scores.



(a) $R = 15$



(b) $R = 30$

Figure 4.7: PR Curves for BC and SO Ranking Performance. SO score improves the performance of BC.

4.5.4 Application to Enron Email Data

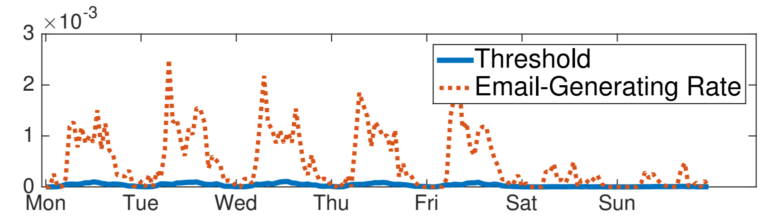
We apply our method to a dynamic network constructed from the Enron email dataset by Priebe et al. [82]. The Enron dataset contains 184 unique email addresses and 125,409 messages dated from November 1998 to June 2002. We represents the data as a time varying undirected graph with time granularity of one second, resulting in a total of $113,733,036$ time steps. This produces a tensor that exceeds the capacity of our computational resource. Hence we construct a $184 \times 184 \times 31,592$ tensor with a time granularity of one hour $w = 3600$. We denote the tensor as $\mathcal{X}$, where $\mathcal{X}_{ijt}$ is the number of emails exchanged between person i and person j at the t-th hour. The average density of a matrix of a snapshot is 3.22×10^{-5} and the maximum density is 0.0019. The edge-generation rate here corresponds to the email-exchanging rate.

We apply PARAFAC with rank $R = 60$ to $\mathcal{X}$ and generate a ranked list of 120 clusters using SO score. There is a large gap between the SO score of the 60-th community and 61-th cluster; hence we apply the elbow rule to decide that the first 60 clusters are worth investigating.

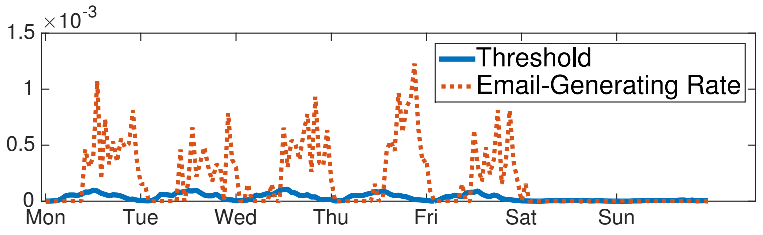
Since the ground truth is not known for the Enron dataset, we refer to results stated in [29, 97] to validate our results. We also check if our method provides new insights into behaviors produced by email exchanges.

One interesting discovery from our method is the difference in the email exchange behavior of different communities. Figure 4.8 shows the average weekly email exchange rate of two groups. The first community in our ranked list consists of two CEOs, three presidents and one employee. These people exchange emails even on weekends. According to Diesner et. al [29], people in this community are "key players" with respect to "centrality measure". On the other hand, the fifth community detected in our model consists of a president and other employees. They only communicate during working hours on weekdays.

We discover that some people belong to several different communities. For example, Richard Shapiro, the vice president of regulatory affair, and Jeff Dasovich appear together in different groups. All identified communities exhibit weekly periodic lifetime and their edge-generation rates have similar patterns every weekday. Some communities have lifetimes across a three-year period (e.g., a community including presidents (Kevin Presto, Richard Shapiro, Shelley Corman) and several employees (Jeff Dasovich, Kay Mann, Susan Scott)), while some last for less than a year (e.g., a community including a manager(Philip Allen) and employees(Daron Giron, phillip Love, and Eric Bass)).



(a) Average Weekly Email exchange Rate of Community 1



(b) Average Weekly Email exchange Rate of Community 5

Figure 4.8: Different Email Exchanging Behavior between Identified Clusters.(a) Email exchange rate of a community consisting of two CEOs (David Delainey and John Lavorato), three vice presidents(James Steffs, Richard Sanders and Richard Shapiro) and an Employee (Jeff Dasovich) (b) Email exchange rate of a community consisting of a president (Louise Kitchen), director(Jonathon Mckay), Employees (Cara, Kate) and traders (Diana, Ryan). The first community exchanges emails even on weekends while the second only exchange emails on weekdays. Time steps belong to a community lifetime when email-generatig rate is larger than theshold.

4.6 Conclusion

We proposed a temporal clustering method based on a network generative model to detect clusters and obtain their temporal information in a sparse dynamic network. PARAFAC decomposition is applied to learn our model and K-mean algorithm with silhouette criterion is used to obtain clusters. We propose a time series segmentation algorithm to analyze the change in clusters and provide a way to obtain community lifetimes. The experiments show that our method has advantages over snapshot-clustering based methods such as evolutionary clustering and other PARAFAC decomposition methods when snapshots are too sparse to provide community structures or lifetime information.

Several discoveries in our experiments and analysis could be helpful to the research on community detection in dynamic networks:

- A snapshot-based clustering model uses local information from limited number of network snapshots. According to experiments on the Lakehurst dataset, it can be applied only if a snapshot contains enough edges to construct a similarity matrix that is consistent with the ground truth information. Models with latent structures takes advantage of global information of all snapshots and usually obtain better clustering results.

- Processing data at coarse scale of time granularity helps snapshot-based clustering, but is relatively less effective for models like PARAFAC decomposition that use global information. However, the temporal information such as the periodicity of a community could be lost at large granularity.

- When determining the lifespan of a community with its average edge-generation rate, it is better to use a possibly different threshold at each time step.

CHAPTER 5

CHARACTERIZING NETWORKS WITH NETWORK MOTIFS

5.1 Introduction

Networks, where interacting elements are denoted as nodes and interactions are denoted as edges, are fundamental to the study of complex systems [5, 69], including social, communication, biological, and economic networks. Analysis of such networks includes network classification, community detection and so on. There exists a rich literature applying machine learning techniques to solve these problems, which requires the network to be represented as a feature vector. However, representing a network is usually difficult to do due to the high dimensionality and network structure.

Various ways of learning *feature representations* of nodes in networks have been recently proposed to exploit their relationships to vector representations [92, 36, 84, 3, 66]. However, most have been applied to node and edge predictions and fail to capture network topological structures. It is unclear if network classification using such node embedding methods can be improved since network structure also plays a significant role. Furthermore, typical analyses usually model these systems as static undirected graphs when describing relations between nodes. However, in many applications, relations are directional and evolve over time [39, 48, 79]. Modeling these *directed and temporal* properties is of additional interest as it can provide a richer characterization of relations between nodes in the network.

In this chapter, we address the aforementioned issues by proposing a novel and flexible network embedding methodology, *gl2vec*, for network classification in both static and temporal directed networks. *gl2vec* constructs vectors for *feature representations* by comparing *static or temporal network graphlet statistics in a network to random graphs generated from different null models.* Graphlets are small non-isomorphic induced subgraphs representing connected patterns in a network and their statistics measure network structures. Null models generate random graphs with specific structural features to decide the significance of graphlets. We show that subgraph ratio profiles (SRP), measures of the ratios for occurrences of graphlets in a network to those in random graphs, can be used as a fixed length feature representation to classify and compare dynamic networks of varying sizes and periods of time with high accuracy. We apply various well-known machine learning models along with our graph feature representation for network classifications, and make a comparison with state-of-the-art methods, such as different graph kernels [92], *node2vec* [36], *struc2vec* [84], *sub2vec* [3], *graph2vec* [66].

In particular, we consider two classification problems. First, we study how static and temporal network graphlets can be used to classify *network domain.* A network domain [79] (or superfamily [62]) represents the type of relation or interaction between nodes that a network describes, e.g., email networks, social networks such as Google+ or Twitter, question answering networks, or even networks that represent switching between mobile apps. Graphs or subgraphs from the same network domain tend to exhibit similar structures [62]. Identifying their network domains further allows us to study interactions between nodes, and predict unobserved network structures. Second, we consider the problem of identifying a particular (sub)network within the same network domain from its static or temporal topological structure. For example, we predict community ID for (sub)graphs within a network, such as identifying a department based on the temporal email-exchange pattern or detecting a mobile phone user given his app switching behaviors represented as static or temporal networks.

Given a network topological structure, identifying the network domain or a network community ID in a network can be viewed as a (sub)graph classification problem. Many existing methods use different graph embedding techniques to represent graphs in a vector space and apply machine learning methods for classification. Yet, little work has applied network graphlets to real-world application in directed (temporal) network classifications. In this chapter, we find that there exists a strong relation between a network domain, graphlet distribution, and subgraph ratio profile (SRP). Here SRP is a measure used to determine if a graphlet is a significant pattern in a network by comparing graphlet counts between an empirical network and random graphs.

Highlights of our contributions include:

1. We propose *gl2vec*, a novel, flexible and scalable feature representation method for network classification in both static and temporal directed networks.

2. We empirically evaluate *gl2vec* against state-of-the-art methods on tasks such as network domain classification and subgraph identification in several real-world static and temporal datasets. We find that *gl2vec* is competitive with state-of-the-art methods in these two tasks.

3. We further show that when *gl2vec* is concatenated with state-of-the-art methods, it significantly improve classification accuracy in real-world applications from several domains. This indicates that *gl2vec* provides important features not captured by state-of-the-art methods.

4. We numerically characterize the impact of different null models on the performance of network classification in static directed networks and results show that all models achieve equally good performance.

The rest of the chapter is organized as follows. First, we present explicit formulation of the problem in Section 5.2. From this, we present *gl2vec* in Section 5.3 and evaluate it in Section 5.4, followed by a discussion on related work in Section 5.5. We conclude in Section 5.6.

5.2 Problem Formulation

In this chapter, we represent a temporal directed network $G(V, E, \mathcal{T})$ as a *temporal edge set* (refer to Definition 2 in Chapter 2). To strictly order the temporal directed edges, denoted by $\{(u_i, v_i, t_i)\}_{i=1}^{|\mathcal{T}|}$, we assume timestamps t_i are unique and $t_{i-1} < t_i$, where u_i, v_i are nodes and $t_i \in \mathcal{T}$ is a timestamp . This assumption is easily extended to cases where temporal directed edges are not unique at the cost of complex notation.

5.2.1 Problem Formulation

Next, we formulate our problem, which applies to the tasks of network domain classification and subgraph identification.

Denote $\{G_i(V_i, E_i, L_i)\}_{i=1}^{N}$ as (sub)graphs in different static or temporal networks, where V_i is a set of nodes and E_i is a set of edges in G_i. If G_i is a temporal network, E_i is then a temporal edge with a timestamp as defined in Definition 2, otherwise, E_i is a directed edge. We assume that a graph belongs to one of D classes of graphs, where $D < N$. We associate each graph G_i with a class label $L_i \in \{1, \cdots, D\}$.

Let $f : \{G_i\} \to \mathbb{R}^m$ be a *mapping function* (also called graph embedding function) from G_i to a $1 \times m$ *feature representation vector* defined using SRPs of static or temporal graphlets. We formally define SRP in Section 5.3.

Let $g : \mathbb{R}^m \to P \in \mathbb{R}^D$ be a *classifier* that maps a feature representation to a categorical distribution P for D labels. We represent probability distribution of G_i's label as $P_i = [p_{i,1}, \ldots, p_{i,D}] = g(f(G_i))$.

Our goal is to solve this classification problem by designing an embedding function f and selecting a machine learning model g that minimizes *the sum of cross entropy*

[25] for all graphs

$$\arg\min_{g,f}\left(-\sum_i\sum_{j=1}^{D}\mathbf{1}_{L_i=j}\log(p_{i,j})\right)=\arg\min_{g,f}\left(-\sum_i\log(p_{i,L_i})\right). \tag{5.1}$$

We obtain g by training machine learning models. In the next section, we discuss how to design an embedding function f for static and temporal networks using graphlets.

5.3 Network Embedding Using Graphlets

Network embedding has received considerable attention due to its effect on the performance of network classification, see Section 5.5. However, previous works have primarily focused on undirected static networks [41, 37, 3, 65, 66]. Applying these techniques directly to directed static networks may lose network structure information while applying them to temporal networks loses temporal information and both may result in poor accuracy. Therefore, we introduce a new directed network embedding technique based on static (temporal) network graphlets.

Graph embeddings need to be independent of network size and if temporal, the time period the network covers. While previous works have shown that the graphlet counts and the probability distribution of graphlets are strongly related to network domains [79], graphlet counts may differ across networks. Instead, we use subgraph ratio profiles (SRPs) for network embedding, which are computed using graphlet counts from both the network in question and random graphs produced using null models (refer to Section 2.2 in Chapter 2).

For static networks, we consider three different null models, ***NE*** (random graphs with the same number of nodes and edges), ***MAN*** (random graphs with the same number of reciprocal edges) and ***BDS*** (random graphs with the same in/out degree sequence). ***NE*** has been widely used in previous studies since it is easy to generate random graphs [50] from this model. The node degree distribution is accurately approximated by a Poisson distribution for large graphs [73]. Thus network features and graphlet statistics can be easily modeled. Recent studies [7, 73] show that node degrees in a wide range of real-world networks do not necessarily follow a Poisson distribution and suggested a null model with controlled node degree sequence for network study. We extend this to ***BDS*** for our study. One study [40] discovered that reciprocity between nodes in different social networks tends to reach maximal reciprocity constraint given in/out degree sequence. Since reciprocities can be computed by the numbers of mutual, asymmetric and null edges, the effect of graphlets statistics in ***MAN*** on network classification is worth investigating. We numerically characterize the impact of null models on the performance of network classification in Section 5.4.

For temporal networks, since there is no equivalent null model, we consider ensembles of randomized time-shuffled data as a temporal null model [61]. To be more specific, we randomly permute the timestamps on the edges while keep the node pairs fixed. This model breaks the temporal dependences between edges but preserves the network structure.

Definition 9. *Subgraph ratio profile (SRP) [62] for a graphlet i is defined as*

$$SRP_i = \frac{\Delta_i}{\sqrt{\sum \Delta_i^2}}, \tag{5.2}$$

where Δ_i is a normalization term that measures the difference between the count of graphlet i observed in an empirical network (denoted as N_{ob_i}) and the average count in random networks in a null model (denoted as $\langle N_{rand_i} \rangle$):

$$\Delta_i = \frac{N_{ob_i} - \langle N_{rand_i} \rangle}{N_{ob_i} + \langle N_{rand_i} \rangle + \epsilon}, \tag{5.3}$$

where ϵ (usually set to four) is an error term to make sure that Δ_i is not too large when a graphlet i rarely appears in both the empirical and random graphs.

In statistics, SRPs are used to identify significant graphlets as network motifs. A large positive value of an SRP indicates that a graphlet occurs much more frequently in a network than would be expected by random chance. Since an SRP for a graphlet is a normalized term, it can be used to compare different size networks. The network embedding is a vector containing 16 SRPs for static triads. For null models of temporal directed networks, we further randomize the order of temporal edges. The embedding contains the SRPs for 36 temporal graphlets illustrated in Figure 2.3.

5.3.1 Algorithm

Our algorithm *gl2vec* works as follows: given the topological structure of a directed static or temporal network, we first compute its graphlet counts. For static networks, we applied JMotif [96] to compute the triad counts for networks and random graphs in different null models. We refer interested readers to [96] for more details. For temporal networks, we use SNAP package [79] to compute 3-edge, $\delta-$temporal graphlet counts.

Then we compute the average graphlet count for different null models. For static networks, there are two approaches: simulation based and probability based. The simulation based approach generates a large set of random graphs with the same structure as the given network and graphlet counts are computed for each random graph. The probability based approach computes the probability of occurance for each type of graphlet given the in/out degree of the nodes involved. We apply the probability based approach to **NE** and **MAN** null models due to their fast computation speed and high accuracy. We apply the simulation based approach to the **BDS** null model because it has lower computational complexity. It also avoids approximation error that occurs when the network size is small (less than 1000 nodes). For temporal networks, we generate random graphs by shuffling timestamps on edges and then compute their average temporal graphlet counts. Finally, we compute SRPs for corresponding graphlets using (5.2). The complete pseudocode is presented in Algorithm 1.

```
Algorithm 1: gl2vec
  Data: Static or temporal graph edges list $E$,
        Null model $M$
  Result: Graph feature vector $\vec{f}$
1 $\vec{N}_{ob}$ = getGraphletCounts($E$) ;
2 $\vec{N}_{rand}$ = getAvgGraphletCountsInNullModel($E, M$)
3 for $i = 1 : |\vec{N}_{obs}|$ do
4     $\vec{f}_i$ = getSRP($\vec{N}_{obs_i}, \vec{N}_{rand_i}$)
5 return $\vec{f}$
```

5.4 Experiments

In this section, we conduct network classification test on several real-world static and temporal directed networks that contain only topological structure information. Experiments focus on two tasks: network domain classification and subgraph identification. In network domain classification, we use *gl2vec* to predict the most likely relation and interaction between nodes, e.g., email communication, question answering or friendship in social networks. In subgraph identification, we predict the community ID for (sub)graphs within the same network. Examples include identifying a department based on email-exchange patterns or a mobile phone user based on his app switching behavior represented as static or temporal networks.

We compare *gl2vec* with state-of-the-art methods on the aforementioned tasks. We also characterize the impact of different null models on the performance of *gl2vec* in network classification. Highlights of our experimental findings include:

1. *gl2vec*, constructing vectors for feature representations using static or temporal graphlet SRPs, is competitive with state-of-the-art methods in network domain classification.

2. For static directed networks, we find that all three null models (*NE*, *MAN* and *NE*) achieves similar performance in network classification in most cases. Compared to other two null models, *BDS* and *MAN*, *NE* has the least control on the network structure and requires less time complexity.

3. Adding graphlet features from *gl2vec* to state-of-the-art-methods significantly improves their performance. This suggests that graphlet patterns from SRPs provide substantial information about network domains that do not exist in state-of-the-art-methods.

4. Both static and temporal graphlets play important roles in temporal network classification.

5.4.1 Datasets

We use a wide range of real-world network datasets, which only contain topological structure. Attributes of nodes and edges are unknown, except for labels for classification and timestamps of edges in temporal networks. These datasets challenge some current state-of-the-art methods that require attributes of nodes or edges.

5.4.1.1 Static directed network datasets

We use static directed networks in different domain and perform network classification using their topological structures in our experiments.

SNAP datasets [55]: For social Networks, *Twitter dataset* contains 1000 ego-networks with $81,306$ nodes and $1,768,149$ edges. *Google+ dataset* contains 133 ego-networks with $106,674$ nodes and $13,673,453$ edges. A directed edge from u to v represents that user u follows v. The size of ego-networks range from 10 to $4,964$ nodes.

Askubuntu and *Mathoverflow* datasets are question-answering networks that store interactions between users. The interactions include posting answers to question (a2q), comments to questions (c2q) and comments to answers (c2a). Both datasets contain four directed networks: an a2q network, a c2q network, a c2a network and a network containing all interactions. *Askubuntu* contains $159,316$ nodes and $596,933$ static edges. *Mathoverflow* contains $24,818$ nodes and $239,978$ static edges.

p2p-Gnutella dataset contains 9 directed peer-to-peer file sharing networks with $6,301$ to $62,586$ nodes and $20,777$ to $147,892$ edges. Nodes represent hosts and edges represent topological connections between hosts.

Cit-HepPh and *Cit-HepTh* are two physics paper citation networks. The former contains $34,546$ nodes and $421,578$ edges, while the later contains $27,770$ nodes and $352,807$ edges.

Slashdot is a friendship network with $77,360$ nodes and $905,468$ edges, where users tag each other as friends. *WikiVote dataset* contains votes from users in Wikipedia to promote other users to become administrators. There are a total of $7,115$ nodes and $103,689$ edges. *Bitcoin OTC trust weighted signed network* contains ratings from Bitcoin users to other users and contains 5881 nodes and $35,592$ edges.

Other Network Types: *Epinion social network* [85] is a who-trust-whom network from a consumer review site Epinions.com containing $75,879$ nodes and $508,837$ edges. A directed edge represents a user "trusting" another user. Advice dataset contains advice-seeking between employees in four different companies [54, 49, 22]. Network sizes ranges from 30 to 60 with edge number ranging from 200 to 500. Co-sponsorship networks [31] contains US Senate co-sponsorship patterns during the 1995, 2000, 2005, and 2010 congressional terms. Nodes represent senators and a directed edge from u to v represents that senator u cosponsored at least one piece of legislation for which senator v was the primary sponsor.

5.4.1.2 Temporal directed network datasets

We also collect temporal directed networks to test feature representation using temporal graphlets.

Email Networks: EmailEU [105] is a directed temporal network constructed from email exchanges in a large European research institution over a 803-day period. It contains 986 email addresses as nodes and $332,334$ emails as edges with timestamps. There are 42 ground truth departments in the dataset and we choose 26 departments containing more than ten users. EmailTraffic [77] is a temporal directed network storing email interactions of 819 staff in 23 different departments in BBN for about 7 months. Edges with integer timestamps represent emails sent out at a certain time.

We apply 10-fold cross validation to construct temporal subgraphs: the 803-day period is split into 10 consecutive time periods, each containing about 80 days. We construct temporal subgraphs in one 80-day period as a test set. Each subgraph lasting 12 weeks for departments in EmailEU networks. This insures that each subgraph is connected when represented as to a static graph. Temporal subgraphs constructed from the other time periods are used as training sets. We create these graphs at the beginning of every four weeks to avoid too much overlap of edges between graphs. Each department has up to 28 subgraphs as a result. For departments in EmailTraffic, we create subgraphs at the beginning of every week and each subgraph covers four weeks.

SwitchApp: (from the Tymer project [76]) contains application switching data for 53 Android users over a 42-day period. We construct a directed temporal network for each user on each day, where a directed edge (denoted as e_{uv}) with an integer timestamp t represents a user switching from an app u to another v at time t.

5.4.2 Experiment Setup

We compute SRPs for static and temporal graphlets for corresponding static and temporal networks in our datasets. In order to get the best classification results, we consider three widely used machine learning models that provide good performance using small amounts of training data in multi-class classification: XGBoosting [19], SVM [21], random forest [93]. XGBoosting is usually superior to other classifiers when the dataset is of medium size. SVM is suitable for a small amount of training data. Random forest not only works well for imbalanced data, but also performs feature selection during training which can help us investigate the usefulness of our feature representation, especially when used in conjunction with other approaches by concatenating the feature vectors.

We use a grid search method to search the best hyper-parameters for these models. For XGBoosting algorithm, the learning rate ranges from 0.001 to 1, maximal tree depth ranges from 4 to 32, minimal child weight is 1 and the subsample ratio of training instances ranges from 0.4 to 1. The regularization weight in SVM ranges from 1 to 8. In random forest, the number of trees ranges from 50 to 400 and the minimal number of samples required to split a tree node is 2 to 10. 10-fold cross-validation is adopted to split the data for select the best parameters. All experiments are conducted using a cluster with 32 Xeon CPUs, 256Gb RAM and one Tesla K40 GPU.

We compare the network classification accuracy of *gl2vec* to state-of-the-art methods, including graphlet and Weisfeiler-Lehman kernels [92], and recently developed node and graph embedding methods *node2vec* [36], *struc2vec* [84], *sub2vec* [3], *graph2vec* [66].

For node embedding methods such as *node2vec* and *struc2vec*, we apply the sum-based approach [24] to aggregate node embedding vectors to construct a graph embedding. We refer the interested reader to [38] for more detail. The length of the network embedding (ranging from 50 to 500) is decided using grid search and 10-fold cross-validation. We modify state-of-the-art methods to apply them to directed graphs: we run a random walk on directed graphs in *sub2vec* instead of undirected graphs. Some state-of-the-art methods also require node attributes for network embeddings and node degree are suggested for computing undirected graph embedding [38]. For directed network in our case, we use NetworkX [27] to compute the in/out degree and centralities such as betweenness, closeness and in/out degree centrality for each node.

5.4.3 Network Domain Classification

In network domain classification, we are given the topological structure of a subgraph in a network. Our goal is to predict the type of interaction that an edge represents, e.g. email exchange or question answering.

Among all datasets introduced in Section 5.4.1, *EmailEU*, *EmailTraffic* and *SwitchApp* datasets have ground truth labels (department ID or user ID) available for each subgraph, which is created from email exchanges in a department or app switch behaviors of a user within a period of time. Hence, we can obtain all subgraphs for these communities in these three networks. For the other datasets, there is no ground truth information on network communities; we detect network communities using modularity [71] to obtain subgraphs. These subgraphs are converted into feature vectors using the previously introduced embedding methods and assigned with labels according to network domains. Finally, we collect about $10,000$ (sub)graphs from $2,355$ real-world networks taken from 15 network domains introduced above, which include Google+ and Twitter in social networks, high energy physics theory citation networks, Gnutella P2P networks, SwichApp and so on.

5.4.3.1 Static Directed Network

We use all datasets to evaluate embedding methods on static networks. Note that we convert temporal networks into unweighted static networks by removing the timestamps on the edges. Baseline methods include graphlet graph kernel (GK graphlet), Weisfeiler-Lehman graph kernel (GK WL), feature vector with triad distribution (MotifDist), *node2vec, graph2vec, sub2Vec* and *struc2vec.* Our proposed graph representation approach *gl2vec* include SRPs computed using three different null models, i.e., (i) *gl2vec(BDS)*: random graphs with the same bidegree sequence; (ii) *gl2vec(MAN)*: random graphs with the same number of mutual, asymetric and null edges; and (iii) *gl2vec(NE)*: random graphs with the same number of nodes and edges.

The accuracies of different embedding methods for network domain classification are presented in Table 5.1. We make the following observations:

1. The graph-based network embedding methods, GK Graphlet, *sub2vec* and *gl2vec* are on average more accurate than other node based network embedding methods. This further validates the importance of characterizing network structure into feature representations for tasks like network classification in which network structure plays a significant role.

2. The machine learning methods also impact of the results. For this task, XGBoost provides the best performance on average in network domain classification. Although *sub2vec* is robust across all three machine learning models, *gl2vec* achieve the highest accuracy and we can always choose the trained model with the highest accuracy for prediction.

3. All null models to compute SRP have similar impact on classification, with *NE* performing slightly better than the others. *NE* is thus preferable among the three because of it lower computational complexity.

	XGBoost (%)	SVM (%)	RF (%)
GK Graphlet	78.94 ± 3.18	72.66 ±2.79	78.72 ±3.01
+gl2vec(BDS)	81.62 ± 3.05	68.38 ± 2.78	81.16 ± 3.33
+gl2vec(MAN)	81.92 ± 2.87	67.29 ± 3.14	81.01 ± 3.29
+gl2vec(NE)	82.18 ± 2.86	69.01 ± 2.27	81.39 ± 3.36
GK WL	78.26 ±2.65	72.81 ± 2.74	78.41 ±3.02
+gl2vec(BDS)	81.87 ± 2.48	68.08 ± 3.51	81.66 ± 2.86
+gl2vec(MAN)	81.71 ± 2.82	66.87 ± 4.13	81.88 ± 3.14
+gl2vec(NE)	82.54 ± 2.85	68.59 ± 2.75	82.26 ± 3.43
MotifDist	78.08 ± 3.34	71.40 ±2.29	78.01 ± 3.56
+gl2vec(BDS)	81.44 ± 3.13	67.93 ± 2.98	80.57 ± 3.71
+gl2vec(MAN)	81.90 ± 3.33	67.60 ± 2.85	80.73 ± 3.12
+gl2vec(NE)	81.75 ± 3.48	69.70 ± 3.64	80.95 ± 3.63
node2vec	74.25 ±3.07	69.03 ±1.23	72.24 ±1.67
+gl2vec(BDS)	88.62 ± 0.89	70.18 ± 3.60	85.77 ± 2.03
+gl2vec(MAN)	89.04 ± 1.17	68.62 ± 3.90	86.54 ± 1.88
+gl2vec(NE)	88.76 ± 1.26	73.24 ± 2.92	86.14 ± 1.71
graph2vec	72.48 ± 3.99	70.81 ± 3.84	72.61 ±3.36
+gl2vec(BDS)	79.81 ± 4.02	66.23 ± 3.91	80.17 ± 4.58
+gl2vec(MAN)	80.09 ± 4.87	66.20 ± 3.83	80.34 ± 4.60
+gl2vec(NE)	79.83 ± 4.59	66.70 ± 4.04	80.03 ± 4.38
sub2vec	81.39 ± 1.70	79.69± 1.41	78.44 ±2.26
+gl2vec(BDS)	92.38 ± 2.38	83.23 ± 2.27	90.46 ± 2.45
+gl2vec(MAN)	92.57 ± 2.40	81.53 ± 2.56	90.57 ± 1.90
+gl2vec(NE)	92.30 ± 2.29	83.16 ± 2.62	90.01 ± 2.16
struc2vec	79.15 ± 3.42	78.22 ±3.15	78.94 ±3.31
+gl2vec(BDS)	92.99 ± 1.56	84.37 ± 1.46	93.10 ± 1.78
+gl2vec(MAN)	93.38 ± 1.50	82.97 ± 2.40	93.22 ± 1.75
+gl2vec(NE)	93.38 ± 1.51	84.25 ± 0.82	93.48 ± 1.42
gl2vec(BDS)	80.92 ± 3.20	72.69 ±3.38	80.17 ±4.07
gl2vec(MAN)	81.49 ± 3.33	71.39 ± 3.98	79.78 ±3.46
gl2vec(NE)	81.58 ±3.07	71.64 ±2.13	79.42 ±3.69

Table 5.1: Network type classification accuracy. We use "+" to denote an embedding generated by combining two embedding methods. Bold indicated best performance machine learning model for each embedding.

4. We also combine *gl2vec* with state-of-the-art methods by directly concatenating their feature representation vectors. We observe a significant improvement over state-of-the-art methods, especially for sub2vec and struc2vec. This suggests that both our approach and state-of-the-art methods capture important but different features for network domain classification. The best approach for the problem is to combine those features. Furthermore, there are also improvements on MotifDist and GK Graphlet. This indicates that adding null models to construct feature representation does help to improve performance. Since representations from *gl2vec*, MotifDist and GK Graphlet construct features from graphlets, the improvement is not as significant as other methods.

5.4.3.2 Temporal directed network

We consider the temporal datasets discussed in Section 5.4.1. We explore if temporal graphlets provide more information for classification than static graphlets in temporal networks. We investigate their effect on predicting whether a temporal (sub)graph is an email exchange network or the app switching behavior of a mobile user. Since the-state-of-the-art methods work only on static networks, we choose *gl2vec(NE)* as a baseline for comparison due to its good performance in static network domain classification. The results are shown in Figure 5.1. From Figure 5.1, we observe that temporal information improves network domain classification in all models considered here. Therefore, it is critical to use temporal graphlets for constructing vectors for feature representations of temporal networks, since temporal graphlets provide more network structure information than static graphlets.

5.4.4 Subgraph Identification

In subgraph identification, we are interested in classifying subgraphs within the same network given their topological structure. For example, we would like to identify

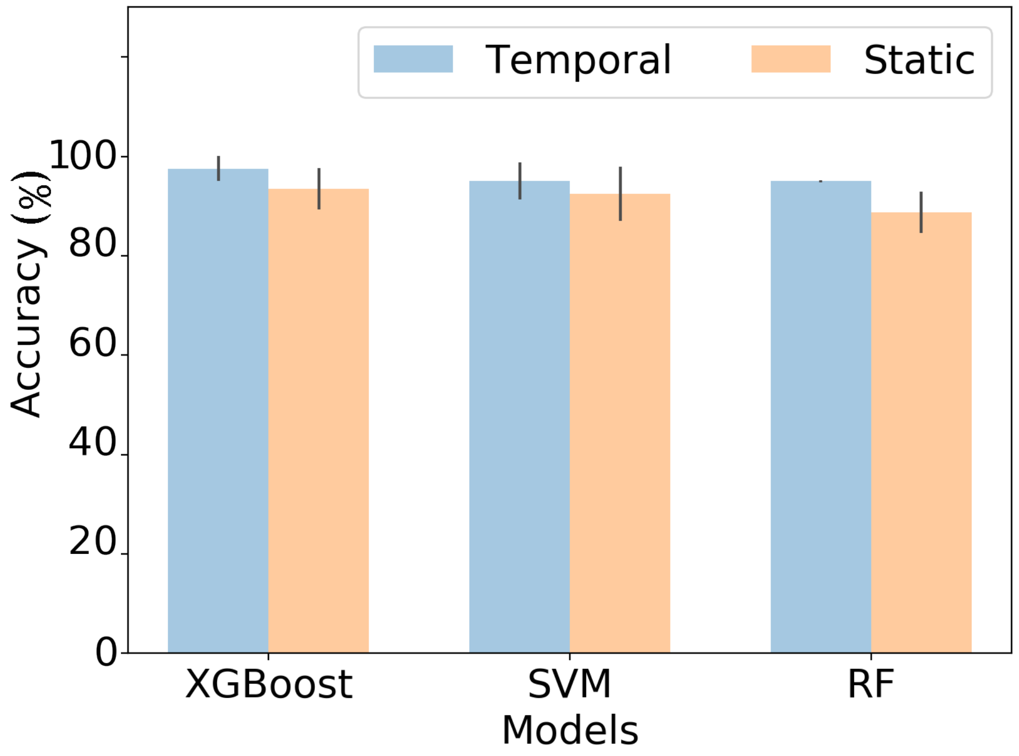


Figure 5.1: Classifying email datasets and SwitchApp Temporal Networks.

which department an email exchange subgraph belongs to or detect a mobile phone user given his app switching behaviors in a day.

We use EmailEU, EmailTraffic and SwitchApp datasets since ground truth labels (department ID or user ID) are available for each subgraph, which is created from email exchanges in a department or app switch behavior of a user over a period of time. We first solve this problem using static graph embedding methods. Then we investigate whether the timestamp information of edges can help improve identification accuracy.

5.4.4.1 Static directed networks

Results on the accuracy of department identification in emailEu, emailTraffic networks and user ID in app switch network using different methods are illustrated in Tables 5.2, 5.3 and 5.4, respectively. We cannot obtain results from graph2vec due to the insufficient memory in GPU. We also consider a combination of feature vector embedding between the state-of-the-art methods and our proposed *gl2vec*. We use "+" to denote the combination of features from two methods. For example, MotifDistr +graphlet2vec(BDS) means a combination of feature vectors from MotifDistr and *gl2vec(BDS)* for feature representation.

We observe that the addition of graphlet SRP features to the state-of-the-art network embeddings can significantly improve their performances. This indicates that *gl2vec* provides new information not present in state-of-the-art methods.

5.4.4.2 Algorithm performance with graphlet features

One observes from Table 5.2, 5.3 and 5.4 that random forest (RF) is usually more accurate for graph embeddings that include our SRP feature vectors in baselines. This is because RF automatically performs feature selection during training and adapts to the change in number of features. As a result, it is easier for RF to achieve better results given similar amount of efforts for fine-tuning the hyper-parameters. Finally, the improvements on all machine learning models confirm that it is worth combining our graph embedding with other methods to achieve better performance.

5.4.4.3 Temporal directed networks

In temporal networks from EmailEU and EmailTraffic, we attempt to identify which department emails belong to. For the SwitchApp dataset, we attempt to identify a particular user based on his daily app switching behavior represented as a temporal network.

For the EmailEU and EmailTraffic dataset, multiple temporal and static networks are constructed for each department from email exchanges as described in Section 5.4.1.2. For the SwitchApp dataset, 42 temporal and static networks are generated for each person from his app switching behaviors every day. XGBoosting, SVM and random forest are implemented using different network feature representations: subgraph ratio profile (SRP) with temporal ("Temporal") and with static ("Static")

	XGBoost (%)	SVM (%)	RF (%)
MotifDistr	56.59 ± 6.63	46.03 ± 7.45	61.13 ± 9.77
+gl2vec(BDS)	63.86 ± 5.83	54.11 ± 8.13	63.45 ± 5.04
+gl2vec(MAN)	62.33 ± 6.42	49.24 ± 8.07	61.98 ± 7.67
+gl2vec(NE)	64.18 ± 6.72	51.96 ± 4.54	63.66 ± 8.04
GK WL	51.54 ± 8.05	48.11 ± 6.56	56.86 ± 6.45
+gl2vec(BDS)	65.85 ± 5.10	52.48 ± 6.40	67.34 ± 5.82
+gl2vec(MAN)	64.75± 5.30	50.33 ± 7.32	65.30 ± 4.14
+gl2vec(NE)	64.12 ± 4.73	50.97 ± 4.84	64.89 ± 7.99
GK Graphlet	60.99 ± 4.01	52.32 ± 4.94	61.70 ± 3.91
+gl2vec(BDS)	63.92 ± 6.33	53.63 ± 7.04	66.01 ± 6.85
+gl2vec(MAN)	63.64 ± 7.04	49.20 ± 9.13	61.71 ± 6.99
+gl2vec(NE)	63.13 ± 6.09	53.37 ± 5.81	63.79 ± 7.82
node2vec	56.28 ± 3.38	55.84 ± 4.04	57.92 ± 3.05
+gl2vec(BDS)	63.84 ± 3.01	60.42 ± 6.06	63.93 ± 4.27
+gl2vec(MAN)	63.94 ± 2.91	58.13 ± 6.70	61.19 ± 4.13
+gl2vec(NE)	62.32 ± 3.48	58.29 ± 5.48	62.32 ± 2.42
sub2vec	54.34 ± 3.74	51.32 ± 3.82	58.37 ± 3.32
+gl2vec(BDS)	72.65 ± 10.92	63.38 ± 9.48	78.75 ± 11.88
+gl2vec(MAN)	73.76 ± 8.77	54.99 ± 8.30	78.33 ± 9.47
+gl2vec(NE)	72.31 ± 8.33	59.84 ± 8.25	77.91 ± 7.31
struc2vec	60.14 ± 8.84	55.38 ± 10.23	63.78 ± 12.93
+gl2vec(BDS)	68.23 ± 4.99	53.84 ± 6.42	71.48 ± 6.93
+gl2vec(MAN)	69.42 ± 5.87	52.83 ± 6.67	70.86 ± 6.05
+gl2vec(NE)	68.43 ± 5.97	52.94 ± 8.04	68.66 ± 7.74
gl2vec(BDS)	**61.66** ± 3.44	52.92 ± 3.62	60.01 ± 3.02
gl2vec(MAN)	57.84 ± 3.83	45.49 ± 3.23	59.24 ± 3.48
gl2vec(NE)	60.01 ± 2.92	52.43 ± 3.19	61.22 ± 3.21

Table 5.2: Accuracy in correctly identifying 26 EmailEU department in static directed networks.

	XGBoost (%)	SVM(%)	RF (%)
MotifDistr	67.18 ± 6.60	59.77 ± 7.23	69.43 ± 6.11
+gl2vec(BDS)	78.10 ± 10.22	68.22 ± 6.55	82.17 ± 7.04
+gl2vec(MAN)	77.32 ± 9.55	68.74 ± 6.74	80.99 ± 6.48
+gl2vec(NE)	76.81 ± 8.42	69.54 ± 6.33	78.43 ± 8.25
GK WL	70.18 ± 5.43	66.23 ± 6.47	72.03 ± 6.09
+gl2vec(BDS)	78.21 ± 8.93	67.21 ± 7.10	79.84 ± 7.26
+gl2vec(MAN)	77.99 ± 6.98	64.91 ± 6.11	78.37 ± 6.36
+gl2vec(NE)	76.34 ± 7.31	70.88 ± 6.33	79.64 ± 6.32
GK graphlet	70.66 ± 10.71	66.12 ± 11.21	74.43 ± 7.11
+gl2vec(BDS)	74.30 ± 10.23	67.04 ± 7.43	79.11 ± 6.28
+gl2vec(MAN)	75.01 ± 9.34	68.88 ± 6.23	78.05 ± 7.01
+gl2vec(NE)	74.67 ± 10.77	68.52 ± 7.01	79.22 ± 7.34
node2vec	72.03 ± 8.13	67.93 ± 12.64	74.06 ± 7.80
+gl2vec(BDS)	82.30 ± 6.77	69.12 ± 6.99	85.06 ± 4.12
+gl2vec(MAN)	82.77 ± 6.90	70.32 ± 6.65	84.72 ± 3.99
+gl2vec(NE)	82.38 ± 6.42	72.44 ± 7.17	84.64 ± 4.09
sub2vec	73.23 ± 3.11	73.12 ± 3.09	73.22 ± 3.83
+gl2vec(BDS)	81.52 ± 5.13	81.33 ± 6.48	85.05 ± 3.93
+gl2vec(MAN)	80.61 ± 5.29	81.49 ± 5.98	84.27 ± 3.53
+gl2vec(NE)	81.94 ± 4.97	82.02 ± 5.25	84.93 ± 4.00
struc2vec	70.78 ± 8.83	65.14 ± 8.43	68.16 ± 8.38
+gl2vec(BDS)	77.11 ± 11.33	54.23 ± 12.39	80.96 ± 13.32
+gl2vec(MAN)	77.84 ± 10.48	54.34 ± 11.43	79.43 ± 10.48
+gl2vec(NE)	76.15 ± 10.46	54.98 ± 13.39	76.43 ± 12.66
gl2vec(BDS)	74.32 ± 5.98	67.13 ± 6.35	74.99 ± 7.02
gl2vec(MAN)	73.84 ± 5.93	68.82 ± 6.10	74.22 ± 6.10
gl2vec(NE)	73.32 ± 6.04	67.51 ± 6.41	75.74 ± 6.43

Table 5.3: Accuracy in correctly identifying EmailTraffic department in static directed networks.

	XGBoost (%)	SVM (%)	RF (%)
MotifDistr	11.82 ± 2.03	11.62 ± 2.02	12.33 ± 2.28
+gl2vec(BDS)	16.31 ± 2.71	13.43 ± 2.05	8.06 ± 2.43
+gl2vec(MAN)	16.53 ± 1.97	12.80 ± 1.64	15.87 ± 2.08
+gl2vec(NE)	16.16 ± 1.85	12.95 ± 2.31	15.34 ± 1.45
GK WL	11.50 ± 1.65	14.59 ± 0.97	13.43 ± 2.26
+gl2vec(BDS)	16.42 ± 2.22	13.56 ± 1.98	17.30 ± 2.66
+gl2vec(MAN)	16.16 ± 1.86	13.00 ± 2.02	16.51 ± 2.32
+gl2vec(NE)	16.01 ± 2.43	13.15 ± 1.44	17.31 ± 1.81
GK graphlet	13.89 ± 1.26	15.24 ± 1.67	15.29 ± 2.28
+gl2vec(BDS)	17.26 ± 2.53	13.43 ± 2.43	18.13 ± 1.72
+gl2vec(MAN)	16.67 ± 1.65	13.15 ± 2.08	16.25 ± 1.78
+gl2vec(NE)	16.52 ± 1.77	13.98 ± 2.51	15.95 ± 1.61
node2vec	10.15 ± 1.50	7.91 ± 1.32	9.98 ± 1.66
+gl2vec(BDS)	17.33 ± 2.62	13.10 ± 2.31	16.93 ± 3.42
+gl2vec(MAN)	15.93 ± 3.04	13.06 ± 2.63	17.21 ± 2.61
+gl2vec(NE)	16.33 ± 2.04	12.94 ± 2.71	16.21 ± 1.97
sub2vec	16.27 ± 2.20	16.19 ± 4.37	16.54 ± 1.64
+gl2vec(BDS)	33.21 ± 3.91	22.73 ± 2.61	33.09 ± 4.24
+gl2vec(MAN)	32.52 ± 3.78	22.18 ± 2.46	32.26 ± 3.57
+gl2vec(NE)	31.74 ± 3.58	23.43 ± 2.33	33.94 ± 4.58
struc2vec	14.18 ± 2.21	9.75 ± 2.49	12.17 ± 2.64
+gl2vec(BDS)	21.77 ± 3.47	10.05 ± 3.14	20.95 ± 2.49
+gl2vec(MAN)	20.58 ± 2.29	9.91 ± 2.29	19.03 ± 2.43
+gl2vec(NE)	19.53 ± 3.13	9.30 ± 1.60	20.70 ± 3.07
gl2vec(BDS)	15.94 ± 2.52	13.36 ± 2.60	16.80 ± 2.16
gl2vec(MAN)	15.58 ± 1.85	12.27 ± 2.75	14.54 ± 1.96
gl2vec(NE)	16.17 ± 1.80	13.56 ± 1.60	16.82 ± 1.31

Table 5.4: Accuracy in correctly identifying 53 SwitchApp user in static directed networks.

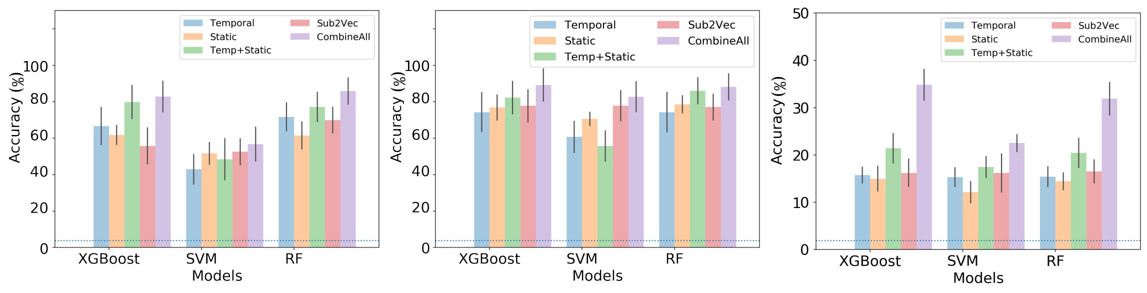


Figure 5.2: *(Left):* Department identification in EmailEU dataset. Five graph embedding methods (with Sub2Vec, SRPs of temporal graphlets, static graphlets, both temporal and static graphlets, and concatenation of Sub2vec and all SRPs) are applied in three machine learning models (XGBoost, SVM and random forest) *(Middle):* BBN department identification in EmailTraffic; *(Right):* User identification in SwitchApp. Dash line represents the accuracy of a random selection model.

graphlets, combined SRPs with both temporal and static graphlet ("Temp+Static"). We illustrate the result from sub2vec representation ("Sub2Vec") because it performs best among the baseline methods. Finally, we create a combination of all three representations ("CombineAll").

The results for EmailEU, EmailTraffic and SwitchApp are shown in Figure 5.2. The dashed line is the accuracy of a random selection model. Accuracy achieved by temporal graphlet embedding is slightly better than that of static graphlet embedding in both emailEU and SwitchApp datasets. However, static graphlet embedding performs better than temporal graphlets in EmailTraffic dataset. This shows that static graphlets are still useful for temporal network classification and can capture useful features even better than temporal graphlets in some datasets. Hence, we combine both static and temporal graphlet features ("Temp+Static") and observe that this achieves significant improvements in accuracy, which suggests that both temporal and static graphlets are useful for network identification (of departments or personal app switching behavior). Furthermore, our graphlet-based network embeddings are competitive with the state-of-the-art method, *sub2vec*. Finally, combining all three graph embedding vectors for classification yield the best accuracy. This suggests that both our static and temporal embedding approaches capture useful features to boost the performances of state-of-the-art methods.

5.5 Related Work

The primary focus of related work in classifying networks involves examining the topological structure of the graph. Most related work to our method is graph kernel based, which has been used to calculate similarities between static undirected graphs [34, 41, 103]. However, the corresponding computational complexity grows significantly with the increase in network size. Moreover, studies in graph kernel do not consider features generated by comparing graphlet count between an empirical network and random graphs from different null models, which turn out to lead to a significant improvement in network classification in our experiments.

Different node embedding techniques have been proposed in the past years, such as *node2Vec* [37], DeepWalk [80], Line [94] and Local Linear Embedding [88] that use feature vectors to embed nodes into high-dimensional space and empirically perform well. However, these methods can only be applied to node classification but not networks. Graph neural network (GCN) [46, 26, 8] recently obtain competitive results against kernel-based methods and graph-based regularization techniques, but are computationally expensive and useful for small scale tasks.

Additionally, several approaches have been proposed to aggregate node embedding to a feature vector for networks. For example, the graph-coarsening approach [26] computes a hierarchical structure containing multiple layers, nodes in lower layers are clustered and combined as node in upper layers using element-wise max-pooling. However, this has high computational complexity. Some approaches [75, 42] define an order of nodes and concatenate their feature vectors for a convolutional neural network for classification, however, this can only be applied to undirected static networks. Recently, some subgraph embedding based approaches were proposed. *struc2vec* [84] applied sum-based approach such as mean-field [24] and loopy belief propagation [64] to aggregate node embedding to graph representation. *sub2vec* [3] embedded subgraphs with arbitrary structure, while *graph2vec* [66] was proposed based on a doc2vec framework to learn data-driven distributed representations of arbitrary sized graphs. But these embedding do not fully capture network structures to acheive the best performance.

5.6 Conclusion

We proposed *gl2vec* to classify static and temporal directed networks based on their topological structure. Experiments with real-world datasets showed that both temporal and static graphlets are important for network type classification and subgraph identification. Furthermore, we illustrated that concatenating these two embedding with many state-of-the-art methods yield the best accuracy for real-world applications such as identifying network types, predicting community ID for subgraphs and detecting mobile phone users based on their app-switching behaviors. Going further, we will investigate if graphlet census information can serve as features for nodes in a network. Specifically, we will check if embedding nodes with the numbers of graphlets that it belongs to in a network, can improve node classification and network classification.

CHAPTER 6
CONCLUSION AND FUTURE WORK

This thesis studies various aspects of dynamic network structure and their application, with focuses on link prediction, community detection, network embedding and network classification.

Chapter 3 investigates relation between node attributes and existence of edges in a growing bipartite network. We model a link prediction problem using a Bayesian method and apply it to recommendation on an online dating website. We use latent Dirichlet allocation model to learn latent variables and interpret them as "user preferences". We show with experiments that our model adapts to changes in user behavior and outperform baseline methods when users change their behavior over time.

There is one possible future direction related to Chapter 2. One popular topic in network science is node embedding in a network [37], which requires attributes of a node as input and produces an embedding vector that is more predictive for existence of edges. Given that latent variables learn in our LDA model can be used for link prediction. It is worthy of investigate if the latent variables can be used as input of the node embedding methods to gain improvement in link prediction.

Chapter 4 proposes a temporal clustering (TC) framework for time-varying network with PARAFAC decomposition, where the time-varying network is represented as a series of static symmetric networks. TC can be applied to track evolving communities. Experiments show that TC is more robust to changes in its hyper-parameters than another tensor decomposition based method and outperform an evolutionary clustering in detecting communities, community members and community lifetimes.

Future study is needed for the application of TC to directed and asymmetric network. This is because many real-world networks can be represented as directed networks. Understanding the

Chapter 5 study graphlets to understand the connectivity patterns of nodes in both static and temporal directed networks. For static network, we compute SRPs for graphlets using three different null models to measure the significances of graphlet occurrences in a network. We propose gl2vec, a network embedding method using graphlet SRPs, for network classification given only topological structures. We show with experiments that network embeddings by gl2vec can be combined to state-of-the-art methods to improve the classification performance. For temporal network, we compute SRPs using a null model that breaks the dependence of edges and discover that both static and temporal graphlets are important to identify a (sub)network. We also show that there is no significant differences in network classification when using SRPs computed by different null models. As a result, we recommend a null model with the same number of nodes and edges to embed a network because of the low time complexity in computing SRPs.

For future work, it is of interest to characterize nodes with the SRPs of graphlets. Structural information such as node degree has proven to be useful in node embedding for classification tasks in real-wold practice [84]. However, it is unclear whether graphlets can be used as node attribute to improve accuracy in link prediction, node classification and network classification.

APPENDIX A
PARAFAC DECOMPOSITION

Tensor decomposition is a useful tool to analyze high dimension matrices. In this section, we introduce a tensor decomposition method: parallel factorization decompostion (PARAFAC decomposition). A PARAFAC model decomposes an tensor into multiple components via a high-order singular value decomposition (HOSVD). It has been applied in many fields and a good reference is the survey from [9]. For simplicity, we describe a PARAFAC decomposition on a three-mode tensor $\mathcal{X}$. A rank-R PARAFAC decomposes $\mathcal{X}$ into R components, each of which consists of a scalar λ_r and a rank-1 tensor produced by the outer product of three column vectors $A_r \otimes B_r \otimes \mathcal{T}_r$(Fig A.1), with the objective of minimizing the Frobenius norm of the error:

$$f(\{\lambda_r, A_r, B_r, \mathcal{T}_r\}_{r=1}^{R}) = |\,\mathcal{X} - \hat{\mathcal{X}}\,|_{\mathbf{F}} = \sqrt{\sum_{i,j,k} (\mathcal{X}_{ijk} - \hat{\mathcal{X}}_{ijk})^2} \tag{A.1}$$

where $\hat{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r A_r \otimes B_r \otimes \mathcal{T}_r$, $A = [A_1, \cdots, A_R], B = [B_1, \cdots, B_R]$ and $\mathcal{T} = [\mathcal{T}_1, \cdots, \mathcal{T}_R]$ are call loading matrices. The PARAFAC decomposition model can be written as:

$$\mathcal{X}_{ijk} \approx \sum_{r=1}^{R} \lambda_r a_{ir} b_{jr} t_{kr} \tag{A.2}$$

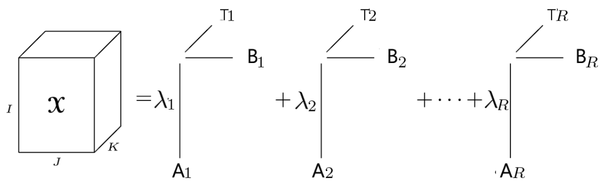


Figure A.1: PARAFAC decomposition of a 3 mode tensor $\mathcal{X}$

where a_{ir} is the i-th element of A_r, b_{jr} is the j-th element of B_r and t_{kr} is the k-th element of $\mathcal{T}_r$.

There are many approximate algorithms for PARAFAC decomposition [2, 1, 102, 89]. We apply the alternating least squares algorithm (ALS) [89] because of its good performance and fast computation speed on average. It applies a gradient descent method to optimize (A.1) iteratively. At each iteration, each loading matrix is updated while other matrices are fixed.

In Chapter 4, since the objective (4.3) of temporal clustering is similar to that of PARAFAC (please refer to (A.1)), we add two constraints to (A.1) and apply PARAFAC decomposition to learn our generative models: (1) the non-negative constraint $\lambda_r, a_{ir}, b_{jr}, t_{kr} \geq 0$ and (2) the symmetric constraint $A_r = B_r$ since network snapshots are symmetric. We also normalize a_{ir} in the interval $[0, 1]$, so that $\max_i(a_{ir}) = 1$, then represent the edge-generation rate $\lambda^{(r)}(k)$ as a piecewise linear segmentation on the sequence of T samples $\vec{\lambda}_r = [\lambda_r t_{kr}]$ for $k = 1, \cdots, T$.

APPENDIX B

MODEL TIME SERIES WITH A PIECEWISE LINEAR FUNCTION

A piecewise linear function is a simple and useful tool to study the trend of the time series. Its application includes learning a time period when time series increases or decrease so as to make predictions on the values of the time series. In this thesis, we represent edge-generation rate, denoted by $\lambda^{(r)}(t)$, between nodes in a network community as a time series and use a piecewise linear function to detect periods of formation or dissolution of the community.

A *piecewise linear function with D segments* is defined as:

$$f_D(t) = \begin{cases} b_1 t + c_1, & t \in Q_1, \\ b_2 t + c_2, & t \in Q_2, \\ \vdots & \vdots, \\ b_D t + c_D, & t \in Q_D. \end{cases} \tag{B.1}$$

where b_i, c_i are constants, $Q_i = [q_i, q_{i+1}]$ is an interval between time q_i and q_{i+1} and $q_i < q_{i+1}$ for $i = 1, \ldots, D$.

Let $\vec{\lambda} = [\lambda_{1r}, \cdots, \lambda_{Tr}]$ be a time series, $f_D(t)$ be a piecewise linear function, the problem of constructing $f_D(t)$ from $\vec{\lambda}$ can formalized as: given integer D, find a segmentation, denoted by $\pi_D = \{Q_i\}_{i=1}^{D}$, and a piecewise linear function $f_D(t)$ to minimize the squared error

$$e(\vec{\lambda}, \{\pi_D, f_D(t)\}) = \sum_{j=1}^{T} (\lambda_{jr} - f_D(j))^2 \tag{B.2}$$

This problem requires manually determining D,the number of segments, and is not suitable for applications where D is unknown.

Note that $e(\vec{\lambda}, \{\pi_D, f_D(t)\})$ is sensitive to D: in the extreme case, $e(\vec{\lambda}, \{\pi_D, f_D(t)\}) = 0$ if $D = T - 1$ where each time step is a segment; at the other extreme, when $D = 1$, $e(\vec{\lambda}, \{\pi_D, f_D(t)\})$ equals to the error of a linear regression on the entire time series. To automatically determine D, Keogh et.al [43] re-formalize the problem as

$$\begin{aligned} & \min D \\ s.t. \quad & \max_{1 \leq d \leq D}(E_d) \leq E_{\max} \quad \text{for } d = 1, \ldots, D, \end{aligned} \tag{B.3}$$

where E_d is the squared error of using $f_D(t)$ to approximate $\vec{\lambda}$ in time segment Q_d, for $d = 1, \ldots, D$, and $E_{\max}$ is a constant hyper-parameter that need to determine manually .

To solve this problem, previous work [43] propose a bottom up algorithm with : the algorithm begins with $T-1$ segments and obtains a solution $M^*_{T-1} = \{\pi_{T-1}, f_{T-1}(t)\}$. At each iteration, the algorithm merges two neighboring segments so as to minimize the increase in cost (denoted as $\Delta C(D) = e(\vec{\lambda}, M^*_{D-1}) - e(\vec{\lambda}, M^*_D)$). The algorithm stops when $\max_{1 \leq d \leq D}(E_d) \leq E_{\max}$. However, it is difficult to choose a threshold $E_{\max}$ for a given time series.

To automatically determine the threshold $E_{\max}$, we assume $\vec{\lambda_r}$ contains noise, δ, that comes from a normal distribution $N(\bar{\delta}, \sigma)$, where $\bar{\delta}$ is the mean of the distribution, and σ is the standard deviation. The error E_d is caused by δ. We use $\bar{\delta} + 3\sigma$ as threshold for $E_{\max}$ because 99.7% of δ's values lie within three standard deviations of $\bar{\delta}$. To learn the distribution $N(\bar{\delta}, \sigma)$, we apply a Sliding Window Filter to $\vec{\lambda_r}$ to compute $\hat{\lambda}_r(t)$, then obtain noise samples $\delta_j = \hat{\lambda}_r(j) - \lambda_{jr}$ for $j = 1, \cdots, T$, and calculate the mean $\bar{\delta} = \frac{1}{T}\sum_{j=1}^{T} \delta_j$ and standard error $\sigma = \sqrt{\frac{1}{T-1}\sum_{j=1}^{T}(\delta_j - \bar{\delta})^2}$.

Finally, the *time series segmentation problem* can be formalized as:

$$\min D \tag{B.4}$$
$$s.t. \quad E_d \leq (\bar{\delta} + 3\sigma)^2 \quad \text{for } d = 1, \ldots, D$$

Algorithm 2: Time Mode Segmentation Algorithm

```
Data: Time Series $Y = [y_1, \cdots, y_T]$
Result: Segmentation $Q = [Q(1), \cdots, Q(D)]$, $f_D(x)$
1  $Q = [[1], [2], \cdots, [T]]$; $\hat{Y} = \text{SlideWindowFilter}(Y)$;
2  $\Delta = Y - \hat{Y}$; $\bar{\delta} = \text{mean}(\Delta)$; $\hat{\sigma} = \text{standardError}(\Delta)$;
3  for $i = 1 : T - 1$ do
4      merge_cost(i) = LinearRegressionError(Y[i:i+1]);
5  while min(merge_cost)$\leq (\bar{\delta} + 3\hat{\sigma})^2$ do
6      i = indexOf(min(merger_cost));
7      Q(i) = merge(Q(i),Q(i+1));
8      delete(Q(i+1);
9      merge_cost(i) = LinearRegressionError(merge(Q(i), Q(i+1));
10     merge_cost(i-1) = LinearRegressionError(merge(Q(i-1), Q(i));
```

Algorithm 2 is a bottom-up algorithm for segmentation of time series and use a piecewise linear function to approximate the time series. Unlike other segmentation algorithm such as [43], our algorithm automatically obtains a piecewise linear function without any hyper-parameters. The function LinearRegressionError($Y[i : j]$) calculates the linear regression $f_D(x)$ for a segment $Q_d = Y[i : j]$ and returns the error. The edge-generation rate $\lambda^{(r)}(t)$ can be constructed as $f_D(x)$ from Algorithm 2 for $d = 1, \ldots, D$.

APPENDIX C
MAPPING CLUSTERS TO GROUND TRUTH CLUSTERS

We need to map the retrieved clusters to ground truth clusters before evaluation. A distance function between two clusters is required for the mapping. For evolving clulsters, the distance function should consider the dynamics of clusters.

Suppose that a method retrieves a ranked list of K clusters $\{\hat{C}_i\}_{i=1}^K$, where $\hat{C}_i$ has higher rank than $\hat{C}_j$ if $i < j$. $\{C_n^*\}_{n=1}^N$ is the set of ground truth clusters. We use x_{ijt}^* to denote the number of edges generated between $i, j \in C_n^*$ at t. $x_{ijt}^* = 0$ if i, j belong to different ground truth clusters. We define the distance function between $\hat{C}_i$ and C_n^* as the approximation error:

$$E(\hat{C}_i, C_n^*) = \frac{\sum_{t=1}^T \sum_{m,o \in \hat{C}_k \cup C_n^*} (b_{mr} b_{or} \hat{\lambda}^{(r)}(t) - x_{mot}^*)^2}{\sum_{t=1}^T \sum_{j,p \in C_n^*} (x_{jpt}^*)^2}$$

where $b_{mi} = a_{mi}$ if node $m \in \hat{C}_i$ and $b_{mi} = 0$ otherwise. Note that $E(\hat{C}_k, C_n^*) = 0$ if $\hat{C}_i = C_n^*$ and $E(\hat{C}_i, C_n^*) = 1$ if $\hat{C}_i$ is the empty set. We map $\hat{C}_i$ to the empty set ϕ if $E(\hat{C}_i, C_n^*) \geq 1$ for any C_n^*. The mapping is performed via the following steps:

(1) Begin with $i = 1$ and map $\hat{C}_i$ to $C_n^* = \arg\min_{C_j^*} E(\hat{C}_i, C_j^*)$ for $j = 1, \ldots, N$.

(2) Increase i by 1 and repeat the previous step. Note that two clusters, $\hat{C}_i$ and $\hat{C}_j$, may be mapped to the same ground truth cluster C_n^*, for $i < j$. We allow these two mappings only if $\hat{C}_i$ and $\hat{C}_j$ are generated from the same generative model $X^{(r)}$. In this case, $\hat{C}_i$ and $\hat{C}_j$ are likely to be sub-clusters of C_n^*. Otherwise, we consider $E(\hat{C}_j, C_n^*) = 1$ and choose another $C_{n'}^*$ for C_j.

BIBLIOGRAPHY

[1] Acar, Evrim, Dunlavy, Daniel M., and Kolda, Tamara G. A scalable optimization approach for fitting canonical tensor decompositions. *Journal of Chemometrics 25*, 2 (February 2011), 67–86.

[2] Acar, Evrim, Dunlavy, Daniel M, Kolda, Tamara G, and Mørup, Morten. Scalable tensor factorizations with missing data. In *SDM* (2010), SIAM, pp. 701–712.

[3] Adhikari, Bijaya, Zhang, Yao, Ramakrishnan, Naren, and Prakash, B Aditya. Sub2vec: Feature Learning for Subgraphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2018), Springer, pp. 170–182.

[4] Ahmed, Amr, and Xing, Eric P. Dynamic non-parametric mixture models and the recurrent chinese restaurant process: with applications to evolutionary clustering. In *SDM* (2008), SIAM, pp. 219–230.

[5] Albert, Réka, and Barabási, Albert-László. Statistical Mechanics of Complex Networks. *Reviews of modern physics 74*, 1 (2002), 47.

[6] Alsaleh, Slah, Nayak, Richi, Xu, Yue, and Chen, Lin. Improving Matching Process in Social Network Using Implicit and Explicit User Information. In *APWeb* (2011).

[7] Amaral, L. A. N., Scala, A., Barthélémy, M., and Stanley, H. E. Classes of small-world networks. *Proceedings of the National Academy of Sciences 97*, 21 (2000), 11149–11152.

[8] Atwood, James, and Towsley, Don. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 1993–2001.

[9] Bader, Brett W., and Kolda, Tamara G. Matlab tensor toolbox version 2.6. `http://www.sandia.gov/~tgkolda/TensorToolbox/`, February 2015.

[10] Balasubramanyan, Ramnath, and Cohen, William W. Block-lda: Jointly modeling entity-annotated text and entity-entity links. In *Proceedings of the 2011 SIAM International Conference on Data Mining* (2011), SIAM, pp. 450–461.

[11] Biswas, Anupam, and Biswas, Bhaskar. Community-based link prediction. *Multimedia Tools and Applications* (2017), 1–21.

[12] Blei, David M, Ng, Andrew Y, and Jordan, Michael I. Latent dirichlet allocation. *JMLR 3* (2003), 993–1022.

[13] Breiman, Leo. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[14] Bro, Rasmus, and Kiers, Henk AL. A new efficient method for determining the number of components in parafac models. *Journal of chemometrics 17*, 5 (2003), 274–286.

[15] Brozovsky, Lukas, and Petricek, Vaclav. Recommender system for online dating service. *arXiv preprint cs/0703042* (2007).

[16] Celeux, Gilles. Bayesian inference for mixture: The label switching problem. In *Compstat* (1998), Springer, pp. 227–232.

[17] Cen, Hao, Koedinger, Kenneth, and Junker, Brian. Learning factors analysis–a general method for cognitive model evaluation and improvement. In *International Conference on Intelligent Tutoring Systems* (2006), Springer, pp. 164–175.

[18] Chakrabarti, Deepayan, Kumar, Ravi, and Tomkins, Andrew. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), ACM, pp. 554–560.

[19] Chen, Tianqi, and Guestrin, Carlos. Xgboost: A Scalable Tree Boosting System. In *ACM SIGKDD* (2016), pp. 785–794.

[20] Chen, Yung-Chih, Rosensweig, Elisha, Kurose, Jim, and Towsley, Don. Group detection in mobility traces. In *Proceedings of the 6th international wireless communications and mobile computing conference* (2010), ACM, pp. 875–879.

[21] Cortes, Corinna, and Vapnik, Vladimir. Support-vector networks. *Machine learning 20*, 3 (1995), 273–297.

[22] Cross, Robert Lee, Cross, Robert L, and Parker, Andrew. *The hidden power of social networks: Understanding how work really gets done in organizations.* Harvard Business Press, 2004.

[23] Culp, Mark, Johnson, Kjell, and Michailidis, George. ada: An r package for stochastic boosting. *Journal of Statistical Software 17*, 2 (2006), 9.

[24] Dai, Hanjun, Dai, Bo, and Song, Le. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning* (2016), pp. 2702–2711.

[25] De Boer, Pieter-Tjerk, Kroese, Dirk P, Mannor, Shie, and Rubinstein, Reuven Y. A Tutorial on the Cross-Entropy Method. *Annals of operations research 134*, 1 (2005), 19–67.

[26] Defferrard, Michaël, Bresson, Xavier, and Vandergheynst, Pierre. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS* (2016), pp. 3844–3852.

[27] Developers, NetworkX. Networkx. *networkx. lanl. gov* (2010).

[28] Diaz, Fernando, Metzler, Donald, and Amer-Yahia, Sihem. Relevance and ranking in online dating systems. In *SIGIR* (2010).

[29] Diesner, Jana, and Carley, Kathleen M. Exploration of communication networks from the enron email corpus. In *SIAM International Conference on Data Mining: Workshop on Link Analysis, Counterterrorism and Security, Newport Beach, CA* (2005).

[30] Fawcett, Tom. Roc graphs: Notes and practical considerations for researchers. *Machine learning 31*, 1 (2004), 1–38.

[31] Fowler, James H. Connecting the congress: A study of cosponsorship networks. *Political Analysis 14*, 4 (2006), 456–487.

[32] Fu, Wenjie, Song, Le, and Xing, Eric P. Dynamic mixed membership blockmodel for evolving networks. In *Proceedings of the 26th Annual International Conference on Machine Learning* (New York, NY, USA, 2009), ICML '09, ACM, pp. 329–336.

[33] Gauvin, Laetitia, Panisson, André, and Cattuto, Ciro. Detecting the community structure and activity patterns of temporal networks: a non-negative tensor factorization approach. *PloS one 9*, 1 (2014), e86028.

[34] Gaüzère, Benoit and Grenier, Pierre-Anthony and Brun, Luc and Villemin, Didier. Treelet Kernel Incorporating Cyclic, Stereo and Inter Pattern Information in Chemoinformatics. *Pattern Recognition 48*, 2 (2015), 356–367.

[35] Ge, Xinyang, Liu, Jia, Qi, Qi, and Chen, Zhenyu. A new prediction approach based on linear regression for collaborative filtering. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on* (2011), vol. 4, IEEE, pp. 2586–2590.

[36] Grover, Aditya, and Leskovec, Jure. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (2016), ACM, pp. 855–864.

[37] Grover, Aditya, and Leskovec, Jure. node2vec: Scalable Feature Learning for Networks. In *ACM SIGKDD* (2016), pp. 855–864.

[38] Hamilton, William L, Ying, Rex, and Leskovec, Jure. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[39] Holme, Petter, and Saramäki, Jari. Temporal Networks. *Physics reports 519*, 3 (2012), 97–125.

[40] Jiang, Bo, Zhang, Zhi-Li, and Towsley, Don. Reciprocity in social networks with capacity constraints. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015), ACM, pp. 457–466.

[41] Kaspar, Riesen, and Horst, Bunke. *Graph Classification and Clustering based on Vector Space Embedding*, vol. 77. World Scientific, 2010.

[42] Kearnes, Steven, McCloskey, Kevin, Berndl, Marc, Pande, Vijay, and Riley, Patrick. Molecular Graph Convolutions: Moving beyond Fingerprints. *Journal of computer-aided molecular design 30*, 8 (2016), 595–608.

[43] Keogh, Eamonn, Chu, Selina, Hart, David, and Pazzani, Michael. Segmenting time series: A survey and novel approach. *Data mining in time series databases 57* (2004), 1–22.

[44] Ketchen, David J, and Shook, Christopher L. The application of cluster analysis in strategic management research: an analysis and critique. *Strategic management journal 17*, 6 (1996), 441–458.

[45] Kim, Min-Soo, and Han, Jiawei. A particle-and-density based evolutionary clustering method for dynamic networks. *Proceedings of the VLDB Endowment 2*, 1 (2009), 622–633.

[46] Kipf, Thomas N, and Welling, Max. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[47] Kovanen, Lauri, Karsai, Márton, Kaski, Kimmo, Kertész, János, and Saramäki, Jari. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment 2011*, 11 (2011), P11005.

[48] Kovanen, Lauri, Karsai, Márton, Kaski, Kimmo, Kertész, János, and Saramäki, Jari. Temporal Motifs in Time-Dependent Networks. *Journal of Statistical Mechanics: Theory and Experiment 2011*, 11 (2011), P11005.

[49] Krackhardt, David. Cognitive social structures. *Social networks 9*, 2 (1987), 109–134.

[50] Kretzschmar, Mirjam, and Morris, Martina. Measures of concurrency in networks and the spread of infectious disease. *Mathematical biosciences 133*, 2 (1996), 165–195.

[51] Krzywicki, Alfred, Wobcke, Wayne, Cai, Xiongcai, Mahidadia, Ashesh, Bain, Michael, Compton, Paul, and Kim, Yang Sok. Interaction-based collaborative filtering methods for recommendation in online dating. In *WISE*. 2010.

[52] Kubat, Miroslav, Holte, Robert C, and Matwin, Stan. Machine learning for the detection of oil spills in satellite radar images. *Machine learning 30*, 2-3 (1998), 195–215.

[53] Kunegis, Jérôme, Gröner, Gerd, and Gottron, Thomas. Online dating recommender systems: The split-complex number approach. In *Proceedings of the 4th ACM RecSys Workshop on Recommender Systems and the Social Web* (New York, NY, USA, 2012), RSWeb '12, ACM, pp. 37–44.

[54] Lazega, Emmanuel, et al. *The collegial phenomenon: The social mechanisms of cooperation among peers in a corporate law partnership.* Oxford University Press on Demand, 2001.

[55] Leskovec, Jure, and Krevl, Andrej. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[56] Lin, Yu-Ru, Chi, Yun, Zhu, Shenghuo, Sundaram, Hari, and Tseng, Belle L. Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In *Proceedings of the 17th international conference on World Wide Web* (2008), ACM, pp. 685–694.

[57] Linden, Greg, Smith, Brent, and York, Jeremy. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing 7*, 1 (2003), 76–80.

[58] Liu, Huan, and Setiono, Rudy. Chi2: Feature selection and discretization of numeric attributes. In *ICTAI* (1995).

[59] Liu, Yan, Niculescu-Mizil, Alexandru, and Gryc, Wojciech. Topic-link lda: Joint models of topic and author community. In *Proceedings of the 26th Annual International Conference on Machine Learning* (New York, NY, USA, 2009), ICML '09, ACM, pp. 665–672.

[60] Mao, Hing-Hao, Wu, Chung-Jung, Papalexakis, Evangelos E, Faloutsos, Christos, Lee, Kuo-Chen, and Kao, Tien-Cheu. Malspot: Multi2 malicious network behavior patterns analysis. In *Advances in Knowledge Discovery and Data Mining.* Springer, 2014, pp. 1–14.

[61] Mellor, A. Classifying Conversation in Digital Communication. *Arxiv preprint arXiv:1801.10527* (2018).

[62] Milo, Ron, Itzkovitz, Shalev, Kashtan, Nadav, Levitt, Reuven, Shen-Orr, Shai, Ayzenshtat, Inbal, Sheffer, Michal, and Alon, Uri. Superfamilies of Evolved and Designed Networks. *Science 303*, 5663 (2004), 1538–1542.

[63] Mimno, David, Wallach, Hanna M, and McCallum, Andrew. Community-based link prediction with text.

[64] Murphy, Kevin P, Weiss, Yair, and Jordan, Michael I. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *UAI* (1999), pp. 467–475.

[65] Narayanan, Annamalai, Chandramohan, Mahinthan, Chen, Lihui, Liu, Yang, and Saminathan, Santhoshkumar. subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs. *arXiv preprint arXiv:1606.08928* (2016).

[66] Narayanan, Annamalai, Chandramohan, Mahinthan, Venkatesan, Rajasekar, Chen, Lihui, Liu, Yang, and Jaiswal, Shantanu. graph2vec: Learning Distributed Representations of Graphs. *Arxiv preprint arXiv:1707.05005* (2018).

[67] Nayak, Richi, Zhang, Meng, and Chen, Lin. A Social Matching System for an Online Dating Network: A Preliminary Study. In *ICDMW* (2010).

[68] Neville, Jennifer, and Jensen, David. Leveraging relational autocorrelation with latent group models. In *Proceedings of the 4th international workshop on Multi-relational mining* (2005), ACM, pp. 49–55.

[69] Newman, Mark. *Networks: an Introduction.* Oxford university press, 2010.

[70] Newman, Mark EJ. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences 103*, 23 (2006), 8577–8582.

[71] Newman, Mark EJ. Modularity and community structure in networks. *Proceedings of the national academy of sciences 103*, 23 (2006), 8577–8582.

[72] Newman, Mark EJ, and Girvan, Michelle. Finding and Evaluating Community Structure in Networks. *Physical review E 69*, 2 (2004), 026113.

[73] Newman, Mark EJ, Strogatz, Steven H, and Watts, Duncan J. Random graphs with arbitrary degree distributions and their applications. *Physical review E 64*, 2 (2001), 026118.

[74] Ng, Andrew Y, Jordan, Michael I, Weiss, Yair, et al. On spectral clustering: Analysis and an algorithm. In *NIPS* (2001), vol. 14, pp. 849–856.

[75] Niepert, Mathias, Ahmed, Mohamed, and Kutzkov, Konstantin. Learning Convolutional Neural Networks for Graphs. In *ICML* (2016), pp. 2014–2023.

[76] Noë, Beryl, Turner, Liam D, Linden, David EJ, Allen, Stuart M, Maio, Gregory R, and Whitaker, Roger M. Timing Rather than User Traits Mediates Mood Sampling on Smartphones. *BMC research notes 10*, 1 (2017), 481.

[77] Olsson, Catherine, Petrov, Plamen, Sherman, Jeff, and Perez-Lopez, Andrew. Finding and explaining similarities in linked data. In *STIDS* (2011), pp. 52–59.

[78] Papalexakis, Evangelos E, Pelechrinis, Konstantinos, and Faloutsos, Christos. Location based social network analysis using tensors and signal processing tools. In *Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2015 IEEE 6th International Workshop on* (2015), IEEE, pp. 93–96.

[79] Paranjape, Ashwin, Benson, Austin R, and Leskovec, Jure. Motifs in temporal networks. In *ACM WSDM* (2017), pp. 601–610.

[80] Perozzi, Bryan, Al-Rfou, Rami, and Skiena, Steven. Deepwalk: Online Learning of Social Representations. In *ACM SIGKDD* (2014), pp. 701–710.

[81] Pizzato, Luiz, Rej, Tomek, Chung, Thomas, Koprinska, Irena, and Kay, Judy. RECON: A reciprocal recommender for online dating. In *RecSys* (2010).

[82] Priebe, Carey E, Conroy, John M, Marchette, David J, and Park, Youngser. Scan statistics on enron graphs. *Computational and Mathematical Organization Theory 11*, 3 (2005), 229–247.

[83] Quercia, Daniele, Askham, Harry, and Crowcroft, Jon. Tweetlda: supervised topic classification and link prediction in twitter. In *Proceedings of the 4th Annual ACM Web Science Conference* (2012), ACM, pp. 247–250.

[84] Ribeiro, Leonardo FR, Saverese, Pedro HP, and Figueiredo, Daniel R. struc2vec: Learning Node Representations from Structural Identity. In *ACM SIGKDD* (2017), pp. 385–394.

[85] Richardson, Matthew, Agrawal, Rakesh, and Domingos, Pedro. Trust management for the semantic web. In *International semantic Web conference* (2003), Springer, pp. 351–368.

[86] Rodríguez, Carlos E, and Walker, Stephen G. Label switching in bayesian mixture models: Deterministic relabeling strategies. *Journal of Computational and Graphical Statistics 23*, 1 (2014), 25–45.

[87] Rousseeuw, Peter J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics 20* (1987), 53–65.

[88] Roweis, Sam T, and Saul, Lawrence K. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science 290*, 5500 (2000), 2323–2326.

[89] Sands, Richard, and Young, Forrest W. Component models for three-way data: An alternating least squares algorithm with optimal scaling features. *Psychometrika 45*, 1 (1980), 39–67.

[90] Schafer, J Ben, Frankowski, Dan, Herlocker, Jon, and Sen, Shilad. *Collaborative filtering recommender systems.* Springer, 2007, pp. 291–324.

[91] Shani, Guy, and Gunawardana, Asela. *Evaluating recommendation systems.* Springer, 2011, pp. 257–297.

[92] Shervashidze, Nino, Schweitzer, Pascal, Leeuwen, Erik Jan van, Mehlhorn, Kurt, and Borgwardt, Karsten M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research 12*, Sep (2011), 2539–2561.

[93] Svetnik, Vladimir, Liaw, Andy, Tong, Christopher, Culberson, J Christopher, Sheridan, Robert P, and Feuston, Bradley P. Random Forest: a Classification and Regression Tool for Compound Classification and QSAR Modeling. *Journal of chemical information and computer sciences 43*, 6 (2003), 1947–1958.

[94] Tang, Jian, Qu, Meng, Wang, Mingzhe, Zhang, Ming, Yan, Jun, and Mei, Qiaozhu. Line: Large-Scale Information Network Embedding. In *WWW* (2015), pp. 1067–1077.

[95] Tang, Lei, Liu, Huan, Zhang, Jianping, and Nazeri, Zohreh. Community evolution in dynamic multi-mode networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), ACM, pp. 677–685.

[96] Tu, Kun. Jmotif, 2018.

[97] URC, PC. Network analysis with the enron email corpus. *Journal of Statistics Education 23*, 2 (2015).

[98] Xia, Peng, Ribeiro, Bruno, Chen, Cindy, Liu, Benyuan, and Towsley, Don. A study of user behavior on an online dating site. In *ASONAM* (2013).

[99] Xia, Peng, Tu, Kun, Ribeiro, Bruno, Jiang, Hua, Wang, Xiaodong, Chen, Cindy, Liu, Benyuan, and Towsley, Don. Who is dating whom: Characterizing user behaviors of a large online dating site. *arXiv preprint arXiv:1401.5710* (2014).

[100] Xiang, Rongjing, Neville, Jennifer, and Rogati, Monica. Modeling relationship strength in online social networks. In *Proceedings of the 19th international conference on World wide web* (2010), ACM, pp. 981–990.

[101] Xu, Kevin S, Kliger, Mark, and Hero, Alfred O. Evolutionary spectral clustering with adaptive forgetting factor. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on* (2010), IEEE, pp. 2174–2177.

[102] Xu, Yangyang, and Yin, Wotao. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on imaging sciences 6*, 3 (2013), 1758–1789.

[103] Yanardag, Pinar, and Vishwanathan, SVN. Deep Graph kernels. In *ACM SIGKDD* (2015), pp. 1365–1374.

[104] Yang, Tianbao, Chi, Yun, Zhu, Shenghuo, Gong, Yihong, and Jin, Rong. Detecting communities and their evolutions in dynamic social networks: a bayesian approach. *Machine learning 82*, 2 (2011), 157–189.

[105] Yin, Hao, Benson, Austin R, Leskovec, Jure, and Gleich, David F. Local Higher-Order Graph Clustering. In *ACM SIGKDD* (2017), pp. 555–564.

[106] Zhou, Ding, Manavoglu, Eren, Li, Jia, Giles, C Lee, and Zha, Hongyuan. Probabilistic models for discovering e-communities. In *Proceedings of the 15th international conference on World Wide Web* (2006), ACM, pp. 173–182.