RELATIONAL LEARNING WITH HYPERGRAPHS

THIS IS A TEMPORARY TITLE PAGE
It will be replaced for the final print by a version
provided by the service academique.

Thèse n. 5807 (2013)
présenté le 24 Mai 2013
à la Faculté Informatique et Communications
Laboratoire d'intelligence artificielle
Programme doctoral en informatique, communications et information
École Polytechnique Fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences
par

Li Pu

acceptée sur proposition du jury:

Prof Matthais Seeger, président du jury
Prof Boi Faltings, directeur de thèse
Prof Kilian Stoffel, rapporteur
Prof Pierre Vandergheynst, rapporteur
Prof Qiang Yang, rapporteur

Lausanne, EPFL, 2013

Hike along the creek to the source,
and sit and watch times when the clouds rise.
— Wei Wang (701 – 761)

To my parents

Acknowledgements

First and foremost I would like to express my sincere gratitude to my supervisor, Prof. Boi Faltings, for his persistent guidance and insightful discussion. Prof. Faltings allowed me to explore the interesting problems in the field of AI, which gives me an enormous freedom in accessing the cutting-edge researches in multiple directions. His encouragement also inspired my curiosity and kept me going forward.

I would like to acknowledge the thesis jury members, Prof. Matthais Seeger, Prof. Kilian Stoffel, Prof. Pierre Vandergheynst, and Prof. Qiang Yang for the comments and suggestions on the thesis. I would especially thank Prof. Qiang Yang for his brief but significant advice on the principles of research, and the kind help on my internship.

Thanks also go to the colleagues in LIA, Florent Garcin for sharing a great office, Jason Li for pointing me the hypergraph, Christos Dimitrakakis for the helpful discussions, Radoslaw Szymanek for the educational tours in personal finance, Immanuel Trummer and Marina Boia for the pleasant conservations during after-dinner working hours, Thomas Léauté and Brammert Ottens for answering my questions on Europe culture, Ludek Cigler for showing me the ski lifts, and many others, Goran Radanovic, Paul Duetting, Claudiu Musat, for being great colleagues. I would like to thank Dr. Jean-Cédric Chappelier for the discussions and pointers to various topics in NLP, and the secretaries Sylvie Thomet and Marie Decrauzat for their kind help in LIA.

In the end, I would like to thank my parents for supporting me throughout my studies. Without their love and persistent encouragement, this thesis would not have been possible.

Lausanne, 6 June 2013

Abstract

Relational learning has received extensive attentions in recent years since a huge amount of data is generated every day in the cyber-space and most of them is organized by the relations between entities. The main tasks of the relational learning include discovering the communities of entities, classifying the entities, and make predictions of possible new relations. Since the graph is a natural representation of pairwise relations, these tasks have been widely studied using the graphs. In this work, we examine the relational learning tasks in the framework of hypergraph which is an extension of the graph. In a graph, an edge could connect exactly two vertices, while in a hypergraph a hyperedge could connect any number of vertices. This extension from graph to hypergraph allows us to represent the higher-order relations such as the co-occurrence relation.

The existing works of hypergraph learning mainly focus on the so-called "vertex expansion" where the hypergraph is transformed into a graph that shares the same set of vertices with the hypergraph. With different weighting functions used in the transformation, the resulting graph would have different structures. The spectral graph theory provides us a powerful tool to analyze the graph structures. It has been shown that one can use the eigenvectors of the graph Laplacian to discover clusters of vertices in the graph. Therefore, the spectral techniques are also adapted to the graph transformed from the hypergraph, which serves as the main ingredient of the clustering and classification algorithms. We show that a special vertex expansion called the normalized hypergraph cut (NHC) can be also used in the link prediction task to rank the possible relations that would appear in the future. In fact, the NHC expansion is able to produce a latent factor space where each entity is represented by a vector and all the vectors form approximately orthogonal clusters.

On the other hand, instead of taking the vertex-centric view in the vertex expansions, we turn to the hyperedge-centric view and develop the "hyperedge expansion" that reflects another category of objective functions defined on the hyperedges. We show that the hyperedge expansion objectives can be attained by computing the eigenvectors of the Laplacian of a directed auxiliary graph, and this eigen-decomposition is equivalent to a quadratic eigenvalue problem (QEP). Based on the analysis of the above eigen-decomposition, we present the clustering and classification algorithms with the hyperedge expansion.

All the approaches developed in this work are compared with state-of-the-art methods in

real applications. First the clustering algorithms are evaluated on the network traffic inspection problem where network traffic records are collected from real corporate networks. The resulting network traffic clusters could give the network administrators a better understanding of the services and traffic flows. Then the classification algorithms are tested in a semi-supervised setting with different relational datasets. We show that the hypergraph-based approaches perform significantly better than the feature-space algorithms when the data can be naturally organized with the co-occurrence relations. Finally the link prediction algorithms based on matrix factorization techniques are adapted into a recommender system and evaluated with datasets from the music and book domains. Evaluation results show that the hypergraph-based approach outperforms the other methods, such as singular value decomposition and non-negative matrix factorization, in terms of the accuracy and diversity of the recommendations.

Keywords: relational learning, hypergraph learning, spectral graph theory, recommender system, network traffic inspection

Résumé

L'apprentissage relationnel a reçu une attention particulière ces dernières années depuis qu'un gros volume de donnée est généré sur internet tout les jours, et la pluspart est organisé par des relations entre entités. Les tâches principales de l'apprentissage relationnel inclus la découverte de communautés d'entités, la classification d'entités, and faire des prédictions sur des relations possible. Comme le graphe est une représentation naturelle des relations par paires, ces tâches ont été largement étudiées à l'aide de graphes. Dans ce travail, nous examinons les tâches d'apprentissage relationel dans le cadre d'hypergraphe qui est une extension de graphe. Dans un graphe, un arc peut être connecté exactement à deux sommets, tandis que dans un hypergraphe un hyperarc peut être connecté à n'importe quel nombre d'arcs. Cette extension de graphe à hypergraphe nous permet de représenter des relations d'ordre supérieur telles que la relation de co-occurence.

Les travaux existant sur l'apprentissage d'hypergraphe se concentre principalement sur l'extension de sommets dont l'hypergraphe est transformé en un graphe partageant le même ensemble de sommets que l'hypergraphe. Avec des fonctions de poids différentes utilisées dans la transformation, le graphe résultant aurait des structures différentes. La théorie des graphes spectraux nous donne un outil puissant pour analyser les structures des graphes. Il a été démontré qu'il est possible d'utiliser les vecteurs propres du graphe Laplacien afin de découvrir des groupes de sommets dans un graphe. Ainsi, les techniques spectrales sont aussi adaptées au graphe résultant de la transformation de l'hypergraphe, nous donnant l'ingrédient principal pour les algorithmes de regroupement et de classification. Nous montrons que q'une extension spéciale de sommets appellée coupe d'hypergraphe normalisée (CHN) peut aussi être utilisée dans la täche de prédire des liens pour classer des relations possible qui pourraient apparaître dans le future. En fait, l'extension CHN est capable de produire un espace factoriel latent dans lequel chaque entité est représentée par un vecteur et tous les vecteurs forment un groupe orthogonal approximatif.

D'un autre côté, à la place de prendre une vue centrée sur les sommets dans l'extension de sommet, nous considérons une vue centrée sur les hyperarcs, et nous développons l'extension d'hyperarcs qui reflète une autre catégorie de fonctions objectives définie sur les hyperarcs. Nous montrons que les objectifs de l'extension d'hyperarc peuvent être atteint en calculant les vecteurs propres du Laplacien du graphe auxiliaire dirigé, et que cette décom-

position propre est équivalente au problème de valeur propre quadratique. Basé sur l'analyse de la décomposition propre, nous présentons des algorithmes de regroupement et classification avec une extension d'hypergraphe.
Toutes les approches développées dans ce travail sont comparées avec l'état de l'art pour des applications réelles. Premièrement, les algorithmes de regroupement sont évalués sur un problème d'inspection de trafic réseau dans lequel des rapports de trafic réseau sont collectés de réseaux d'entreprise réels. Les groupes de résultant peuvent donner aux administrateur de réseau une meilleure compréhension des services et flux de trafic. Ensuite les algorithmes de classification sont testés dans cadre semi-supervisé avec différent ensembles de données relationnelles. Nous montrons que les approches basées sur l'hypergraphe se comportent sensiblement mieux que les algorithmes d'espace caractéristique lorsque les données peuvent être naturellement organisée par des relations de co-occurence. Finallement, les algorithmes de prédiction de liens basés sur la factorisation matricielle sont adaptés aux systèmes de recommendation et évalués avec des ensembles de données dans le domaines des livres et musiques. Les résultats de l'évaluation montrent que l'approche basée sur l'hypergraphe est plus performante que que les autres méthodes pour ce qui est de la précision et la diversité des recommendations.

Mots-clés : apprentissage relationnel, apprentissage d'hypergraphe, théorie des graphes spectraux, systèmes de recommendation, inspection de trafic réseau

Contents

List of Figures

List of Tables

1 Introduction

People understand the world by recognizing the objects in the world and connecting them with relations. The genealogy tells us which family a person belongs to, an alliance shows us which countries are on the same side, and the follower-following relation decides who makes the biggest impact in an online social network. We have been using the relations for a long time to distinguish, categorize, and classify the objects, or even predict new relations that would appear in the future. As our world going quickly to the direction of digitization and mobile, an enormous amount of data is generated everyday on the Internet by the users, and many of them are recorded in terms of relations.

In this work, we study the relations and the learning problems with relational data. Traditionally the relational data is organized as high-dimensional data points, e.g. see [Roweis and Saul, 2000] for the relations between the words and documents, and special algorithms are designed for this type of inputs. However, as it turns out, relations are different from the data points in a vector space. The relations are essentially subsets of entities, and the correlations between entities are described in terms of intersections between the subsets. While in a vector space, the correlations between data points are usually defined by a distance function or a similarity function. In many applications, modelling the entities with relations would be more natural than modelling them with feature vectors. For example, the users in a social network can be easily linked by the follower-following relations, but it would be very factitious to create a feature vector for each user to describe these relations.

In recent years, many types of relational learning problems have been extensively studied [Getoor and Diehl, 2005, Getoor, 2007]. The clustering and classification problems require to learn a partition of entities so that each entity is attached with a label. For example, given a social network, we would like to discover the communities (clusters) of people who have stronger connectivity within a community than between the communities. The identified communities can be used to interpret the underlying social structures such as the emerging

political/cultural groups. Another example would involve the interactions in a cell and the gene expressions measured from a patient. We know that some diseases are caused by certain genes expressions through the signaling pathways. Thus it would help the doctor if we can classify the disease by measuring the gene expression profile of the patient and making inference through the relations in the pathway network. The third type of relational learning tasks tries to predict the future relations. In the million-dollar Netflix Prize, we are given a set of ratings for some movies from a set of users, and the goal is to recommend more movies, or discover more relations between the users and the movies, based on the given ratings [Bennett and Lanning, 2007]. This task is challenging because other than studying the correlations between the users and the movies, we need to create a model such that the existing correlations can be generalized to make predictions. The above three tasks, namely the clustering, classification, and link prediction, are the main learning problems in this work.

1.1 Pairwise and Co-occurrence Relations

There are mainly two types of relations discussed in this work, namely the *pairwise relation* and the *co-occurrence relation*. A pairwise relation is also called a binary relation. It involves in two entities, usually of the same type. For example, a friendship relation connects two persons. Normally the friendship is *symmetric*, which means that if Bob is a friend of Alice, Alice is also a friend of Bob. A binary square matrix $\boldsymbol{A}$ can be used to describe all the possible relations. The rows and columns of $\boldsymbol{A}$ are named after the entities, and the entry $\boldsymbol{A}(i,j)$ is set to 1 if a pairwise relation exists between the row entity i and the column entity j, otherwise $\boldsymbol{A}(i,j)$ is set to 0. For a symmetric relation, the matrix $\boldsymbol{A}$ is also symmetric. Another type of pairwise relation is the *asymmetric* relation, which allows only one direction of the relationship between any two entities. If Bob is the parent of Alice, Alice cannot be the parent of Bob. In this case, it is not possible to have $\boldsymbol{A}(i,j) = 1$ and $\boldsymbol{A}(j,i) = 1$ at the same time. But in many Internet applications, pairwise relations are neither symmetric nor asymmetric. For example, the twitter following-follower relation can be one-directional or bi-directional.

The pairwise relation has two limitations. Firstly, it could only describe homogeneous relations. Secondly, a pairwise relation could only relate exactly two entities. However, in many cases, there are entities of different types, and a relation could involve more than two entities. A co-occurrence relation is a natural extension of the pairwise relation. It could contain any number of entities (can be zero) that have co-occurred. For example, some students have participated in the same course, so we can create a co-occurrence relation that includes all these students. The co-occurrence relation can be named after the place or the occasion where the entities have co-occurred. In the above example, the course name could be the name of the co-occurrence relation. When there are many co-occurrence relations, the set of all entities that can co-occur is denoted as $Z = \{z_1, z_2, ..., z_n\}$ (e.g. the students), and the set

of co-occurrence relations is denoted as $Y = \{y_1, y_2, ..., y_m\}$ (e.g. the courses). A binary matrix (usually non-square) $\boldsymbol{X}$ of size $m \times n$ can be used to describe the co-occurrence relations. Each row of $\boldsymbol{X}$ represents a co-occurrence relation in Y, and each column of $\boldsymbol{X}$ represents an entity in Z. The entry $\boldsymbol{X}(i, j)$ is set to 1 if the entity z_j is observed in the co-occurrence relation y_i.

The roles of the entities in Z and the entities in Y can be swapped. Consider the courses taken by a student, we could include these courses in a co-occurrence relation which is named after the student. When performing this role swapping, the relation matrix $\boldsymbol{X}$ is simply transposed.

As the co-occurrence relations are represented by a 2-dimensional matrix, it could only display two types of entities. Many existing works have extended this representation to a multi-dimensional array (tensor) for more types of entities [Lin et al., 2009, Acar et al., 2009, Neubauer and Obermayer, 2009]. But in this work our discussion is restricted to the 2-dimensional case, because the tensor method often suffers from the sparsity of the data, and in most applications the multi-dimensional representation can be approximated by several 2-dimensional matrices.

It is possible to approximate a co-occurrence relation by a set of pairwise relations. This can be done by connecting each pair of entities in the co-occurrence relation with a pairwise relation. But some information is lost and some additional non-desirable information is added in this transformation. Consider the above example, the students from the CS department and the students from the EE department may attend the same course. This fact simply tells us that all the students in the course are likely to be in the same group because they share the same technical background, but the sub-group structures within this group are unknown. However, with the pairwise transformation, each pair of students in the course are connected, which implicitly suggests that a CS student is likely to be in the same group with an EE student. The meaning of the pairwise relations has been changed. Existing works have studied the differences between pairwise relations and co-occurrence relations, and suggested that the (higher-order) co-occurrence relations cannot be easily approximated by the pairwise relations [Agarwal et al., 2005, 2006, Bulò and Pelillo, 2009, Ladicky et al., 2010].

It is worth mentioning that both pairwise relation and co-occurrence relation can only describe the positive correlations between the entities, i.e. the information states that the entities are similar. A very fundamental assumption about these models is the *transitivity*. With the pairwise relations, if computer A is connected to computer B and computer B is connected to computer C, there must be a connection between computer A and computer C. And the similar property holds for the co-occurrence relations. In fact, almost all the existing works exploit this basic assumption to make inferences, and our learning algorithms in this work also rely on the transitivity. Therefore, dissimilar relations such as “these students come

from different cities" are not supported in this work.

1.2 Graph and Hypergraph

It is straightforward to model the pairwise relations as a graph. Let $G = \{V, E\}$ denote a graph where V is the vertex set and E is the edge set. Each entity can be represented by a vertex $v \in V$, and each pairwise relation can be represented by an edge $e \in E$ that connects two entities $v \in V$ and $u \in V$. The graph can be undirected if the pairwise relations are symmetric, and directed if the relations are non-symmetric (including the asymmetric case). The *adjacency matrix* of the graph is exactly the the matrix $\boldsymbol{A}$ introduced in the previous section. In this work, we assume that there is no edge from a vertex to itself, i.e. there is no relation from an entity to itself.

Similarly, a set of co-occurrence relations can be represented by a hypergraph $H = \{Z, Y\}$. The vertex set Z is the set of entities that could appear in the co-occurrence relations, and each element in the hyperedge set Y represents one co-occurrence relation. In fact, each hyperedge is a subset of Z, i.e. $y_i \in Y, y_i \subseteq Z$. The difference between a graph and a hypergraph is that an edge in a graph connects exactly two vertices, while a hyperedge could connect any number of vertices. We use the same notation y_i for a hyperedge and a co-occurrence relation, since they are referring to the same thing.

When the relational data is modeled by a graph or a hypergraph, the learning problem of clustering or classification can be expressed by a partition of the vertex set. Taking the friendship graph as an example, the people are represented by the vertices, and the friendship relations are represented by the edges. We would like to identify the communities of people, i.e. a partition of the vertex set, where people have stronger connections within a community than between the communities. To make the partition reasonable, one has to precisely define what is a good partition. In the context of graph, a straightforward rule would consider a partition that leaves the least number of edges between communities as the best partition. In other words, we would like to minimize the edge-cut when applying the partition [Wu and Leahy, 1993].

When it comes to the hypergraph, there are two choices. First, we could transform each co-occurrence relation to a set of pairwise relations and form a graph. Then we can simply borrow the rules from the graph to determine the best partition. Second, it is also possible to define new rules directly on the hypergraph, but all the techniques developed for the graphs have to be revised for the new rules. These two approaches are studied and compared in Chapter 3 and Chapter 4.

The main tool that we use to study the graph and the hypergraph is the spectral graph the-

ory [Chung, 1997]. The spectral graph theory tries to decompose a graph into orthogonal components such that different components can be studied separately. Since the discovery of the Fiedler vector (the second eigenvector of a graph Laplacian) in bi-partitioning a graph [Fiedler, 1973], the spectral techniques are extensively studied for clustering problems in graphs. In this work, we introduce these techniques as the main tools and extend them to the hypergraphs.

In the Internet applications, another very important observation is about the degree distribution of the vertices in the graphs or hypergraphs. Instead of an uniform or a normal distribution, there are usually a few very high-degree vertices and a lot of low degree-vertices. The actual distribution usually follows a power-law function [Faloutsos et al., 1999]. With this type of distribution, a good partition based on the edge-cut does not always exist. Large graphs tend to have an expander-like core and the community structures are hard to find [Abou-Rjeili and Karypis, 2006, Leskovec et al., 2009]. Generative models like preferential attachment [Newman, 2001] and "forest fire" model [Leskovec et al., 2005] provide possible explanations for the power-law distribution and the expander core. But if the partition is based on the vertex-separator instead of the edge-cut, balanced partition does exist by removing the high degree vertices [Albert et al., 2000]. Inspired by these works, we observe that a hypergraph constructed from a real world application often consists of a few hyperedges that include many vertices. The transformation from co-occurrence relations to pairwise relations would usually generate a power-law graph. Thus a partition can be better expressed in the original hypergraph, which inspires our work in Chapter 4.

1.3 Overview and Contributions

In Chapter 2, we formally define the graph, the hypergraph, and the learning tasks. Then the spectral graph theory is briefly introduced in the view of clustering the vertices in a graph. We also present a generative model based on the beta-Bernoulli process, which provides a foundation for the subsequent chapters. In Chapter 3, the spectral graph theory is extended to the hypergraph case, by the so called vertex expansions. The idea of vertex expansions is very similar to the above-mentioned transformation from the co-occurrence relations to the pairwise relations. But more sophisticated measures are taken to ensure that the transformation exhibits desired properties. We present the algorithms for the learning tasks based on the vertex expansions. And for the first time we show a justification of using the vertex expansion in an asymmetric manner in the link prediction task. In Chapter 4, we examine another category of hypergraph transformations that focus on the hyperedges rather than the vertices. This so called hyperedge expansion has drawn much less attentions in the literatures compared to the vertex expansions. For the first time we use a vertex-separator formulation in the transformation, and study the spectral properties of the resulting directed graph. Then

the algorithms for the clustering and classification tasks are presented. In Chapter 5, three experimental studies are carried out for the clustering, (semi-supervised) classification, and link prediction tasks. We show that our algorithms could produce good results in the real applications like network traffic inspection and recommender system. In Chapter 6, we conclude the thesis and propose several future research directions.

The contributions of this work include the following. We adapt the nonparametric Bayesian models to the "multi-class beta-Bernoulli process" which describes the generating process of a hypergraph (section 2.4). Based on this model, we could analyze the properties of a hypergraph, such as the sparsity of the hypergraph Laplacian (section 3.2 and 4.1) and the orthogonal structure of the vertex embeddings (section 3.2). To our best knowledge, this is the first work to combine such model with the spectral graph theory. Since the power-law distribution of vertex degrees is often observed in real applications, the link between the generating process that would produce the same type of distribution and the spectral graph theory is especially worth attentions.

The second main contribution of our work is the hyperedge expansion. We focus on the hyperedges, and take a different view in the transformation from a hypergraph to an auxiliary graph. The minimum hyperedge cut problem is converted to a minimum vertex separator problem, and then a min-cut problem in a directed graph (section 4.1). We show that the spectrum of the above directed graph is essentially the same as the spectrum of a quadratic eigenvalue problem of the half size, and thus can be used to approximately solve the original minimum hyperedge cut problem (section 4.2). To our best knowledge, this is the first work to analyze the spectral properties of the minimum hyperedge cut problem and link it to a quadratic eigenvalue solution.

We also propose a link prediction algorithm whose prediction rule is justified in the framework of multi-class beta-Bernoulli process (section 3.4), and the clustering/classification algorithms with the hyperedge expansion (section 4.3).

Finally, experiments with real applications and benchmark datasets are carried out to show the effectiveness of our proposed methods (chapter 5). It is worth mentioning that the hypergraph based relational learning algorithms have been successfully applied to a network traffic inspection tool which could help the network administrators to understand the traffic flows in the real corporate networks (section 5.1).

2 Relational Learning with Hypergraphs

The early studies of hypergraph focus on the partitioning [Lawler, 1973], matching and colouring problems [Berge, 1989] from a pure mathematical point of view. Then the hypergraphs have been used to describe the co-occurrence relations of logical formulas [Park and Van Gelder, 1996], association rules in databases [Han et al., 1997], and VLSI circuits [Karypis et al., 1997]. As soon as the volume of relational data has dramatically increased from popular Internet and mobile applications, the use of hypergraph or similar models has diversified into many areas, such as recommender system[Siersdorfer and Sizov, 2009, Bu et al., 2010], community detection[Lin et al., 2009, Neubauer and Obermayer, 2009], inference of graphical models[Kok and Domingos, 2009], and parallel computing[Devine et al., 2006, Gonzalez et al., 2012b].

The methodology of hypergraph research is also extended by the spectral graph theory [Chung, 1993, Zien et al., 1999, Zhou et al., 2007]. Just as the physicists study the spectrum of signals to reveal the composition of a material, spectral graph theory studies the principal properties and structures of a graph by decomposing the graph into a set of orthogonal bases. The spectral properties are closely connected to interesting phenomenons in a graph, e.g. the community structure, the diffusing time, and the fault tolerance property. The spectral graph theory plays a central role in our studies of hypergraph.

Another line of works zooms into the fundamental mechanisms with which the relations emerge [Newman, 2001, Chojnacki and Kłopotek, 2010, Broderick et al., 2012]. It is interesting to see how simple rules can be applied to individual entities and yet the aggregated behavior of a group follows certain patterns (e.g. the power-law distribution of vertex degrees). We propose a generative model based on existing works to create a hypergraph, which provides justifications for our usage of hypergraphs in various applications.

In this chapter we formally define the hypergraph and the problems to be studied, introduce the related techniques such as the spectral graph theory, and propose a generative model for

hypergraphs. These components form the foundations of our work.

2.1 Graph and Hypergraph

A *weighted graph* $G = \{V, E, w\}$ consists of a *vertex set* V, an *edge set* E, and a weighting function $w : E \to \mathbb{R}^+$. Each edge in the graph connects exactly two vertices $e = (v_i, v_j)$, and the weight of an edge $w(e)$ is assumed to take only positive real values. If the graph is *directed*, the edges connecting the same pair of vertices in opposite directions are different edges, i.e. $e = (v_i, v_j) \neq e' = (v_j, v_i)$. In this work, a graph is assumed to be undirected by default if the type is not specified. The *weighted degree* of a vertex is the sum of the weights of the edges which are connected to that vertex, i.e. $deg(v) = \sum_{e \in E, e=(v,v')} w(e)$. Similarly, when the graph is directed, the in-degree of a vertex is $indeg(v) = \sum_{e \in E, e=(v',v)} w(e)$, and the out-degree is $outdeg(v) = \sum_{e \in E, e=(v,v')} w(e)$.

Let $n = |V|$ denote the number of vertices in the graph. The *(weighted) adjacency matrix* is a $n \times n$ real matrix $\boldsymbol{A}$, where $\boldsymbol{A}(i, j) = w(e)$ if the edge $e = (v_i, v_j)$ exists. When the graph is undirected, $\boldsymbol{A}$ is symmetric (or real Hermitian). The *(weighted) degree matrix* is a $n \times n$ real diagonal matrix $\boldsymbol{D}$ whose main diagonal contains the (weighted) degrees of the vertices.

A *weighted hypergraph* $H = \{Z, Y, w\}$ consists of a *vertex set* Z, a *hyperedge set* Y, and a weighting function $w : Y \to \mathbb{R}^+$. Each hyperedge $y \in Y$ is a subset of Z, i.e. $y \subseteq Z$. A hyperedge y is *incident* with a vertex z if $z \in y$. The weight of a hyperedge $w(y)$ is assumed to take only positive real values. The weighted degree of a vertex is the sum of the weights of the hyperedges that are incident with that vertex, i.e. $deg(z) = \sum_{z \in y, y \in Y} w(y)$. The degree of a hyperedge is the number of vertices within that hyperedge, i.e. $deg(y) = |y|$.

Let $n = |Z|$ denote the number of vertices, and $m = |Y|$ denote the number of hyperedges. The *incident matrix* is a $m \times n$ binary matrix $\boldsymbol{X}$, where the entry $\boldsymbol{X}(i, k) = 1$ if the hyperedge y_i is incident with the vertex z_k, otherwise $\boldsymbol{X}(i, k) = 0$. The $m \times m$ diagonal matrix of hyperedge degrees is defined as $\boldsymbol{D}_y = \text{diag}(\boldsymbol{X}\boldsymbol{1})$, where $\boldsymbol{1}$ is a all-ones vector of proper length. Let $\boldsymbol{W}$ denote the $m \times m$ diagonal matrix of the hyperedge weights, i,e, $\boldsymbol{W} = \text{diag}\big(w(y_i)\big)$. The $n \times n$ diagonal matrix of vertex degrees is defined as $\boldsymbol{D}_z = \text{diag}(\boldsymbol{1}^\top \boldsymbol{W}\boldsymbol{X})$, where $(\cdot)^\top$ denotes the transpose of a matrix or a vector.

A hypergraph is an extension of a graph (the edges in a graph can be considered as hyperedges of fixed degree), so we could represent a graph either in the form of the adjacency matrix $\boldsymbol{A}$, or in the form of the incident matrix $\boldsymbol{X}$ and the weight matrix $\boldsymbol{W}$.

We say that a hypergraph (or similarly a graph) H is *connected* if for any pair of vertices $z_i, z_j \in Z$, there exists a hyperedge path $\{y_1, y_2, ..., y_p\}$ such that $z_i \in y_1$, $z_j \in y_p$ and $y_k \cap y_{k+1} \neq \emptyset$

($1 \leq k < p$). Without loss of generality, in this work we assume that a hypergraph (or a graph) is always connected.

The (undirected) *induced graph* $G_H = \{V_{G_H}, E_{G_H}, w_{G_H}\}$ derived from a hypergraph $H = \{Z, Y, w\}$ consists of the same vertex set $V_{G_H} = Z$. An edge $e = (v_i, v_j) \in E_{G_H}$ is placed between the vertices v_i and v_j in G_H if there exists a hyperedge y in the original hypergraph which is incident with both v_i and v_j. The weight of the edge is defined as $w_{G_H}(e) = \sum_{y \in Y, y \ni v_i, v_j} w(y)/deg(y)$. It is easy to show that the hypergraph is connected if and only if the induced graph is connected.

Another representation of the hypergraph or the co-occurrence relations resorts to a bi-partite graph G_b. The vertex set of G_b is the union of the set Z and Y in the original hypergraph. But an edge in G_b can only connect a member of Z to a member of Y. There is no edge connecting members within Z or within Y. The weights of the hyperedges can be attached to the vertices corresponding to Y such that the graph G_b is vertex-weighted (only some vertices have weights). Or we can assign weights to the edges of G_b so that G_b is edge-weighted. Figure 2.1 illustrates an example of a hypergraph, its induced graph, and the corresponding bi-partite graph.

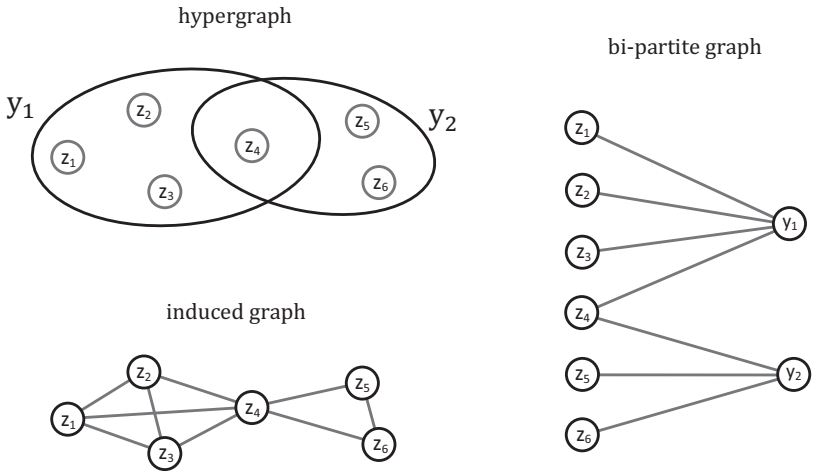

Figure 2.1: An example of a hypergraph, the induced graph, and the corresponding bi-partite graph.

There is an one-to-one mapping from a hypergraph to a bi-partite graph. But an induced graph can be mapped to multiple hypergraphs. In fact, the adjacency matrix of the induced graph G_H can be written as $\boldsymbol{A}_{G_H} = \boldsymbol{D}_z^{-1/2}\boldsymbol{X}^\top \boldsymbol{D}_y^{-1}\boldsymbol{W}\boldsymbol{X}\boldsymbol{D}_z^{-1/2}$, providing that the entries in the main diagonal of $\boldsymbol{A}_{G_H}$ are set to zeros. The induced graph G_H is uniquely determined by $\boldsymbol{A}_{G_H}$, while the combination of $\boldsymbol{X}$ and $\boldsymbol{W}$ that could generate $\boldsymbol{A}_{G_H}$ is usually not unique.

Below we list some examples of hypergraphs. They are constructed from real datasets and

used for illustrations through this work.

Example 2.1.1. [zoo] Consider some animals and their attributes. Some animals have 2 legs, some have 4 legs. And some animals live in the sea, while some others could fly. The animals that share the same attribute are considered to be similar. We can use the co-occurrence relations to model such similarities. To represent the zoo data with a hypergraph, each animal is represented by a vertex, and each hyperedge contains the animals who share the same attribute. For example, one hyperedge contains all the animals of 2 legs, and another hyperedge contains all the animals living in the sea. The total number of hyperedges depends on the number of attributes and the number of values that each attribute could take from.

Example 2.1.2. [amazonbooks] Consider the books available in Amazon.com and the users who purchase the books. The books purchased by the same user are usually similar because these books have similar features for attracting the user, although we do not know what are the features explicitly. For example, one user might only purchase comic books, and another user might only purchase love stories and biographies. To represent the amazonbooks data with a hypergraph, we can take each book as a vertex, and each user as a hyperedge that contains all the books (vertices) that he or she has purchased.

Example 2.1.3. [music] Consider the music artists in a music store and the users who listen to the musics. The users usually choose the artists by their personal preferences on the artist's music styles or genres. The artists visited by the same user would have similar styles. In the music data, we use a vertex to represent an artist, and a hyperedge to represent a user which contains all the artists who have been visited by this user.

2.2 Learning Tasks

The relational learning tasks on a hypergraph can be classified into several categories. Below we formally define three categories of tasks that are studied in this work.

2.2.1 Clustering

The goal of the clustering task is to find a partition of the vertex set such that similar vertices are grouped together. A *(vertex) partition* or a *clustering* of a hypergraph H consists of s non-overlapping subsets of the vertex set, i.e. $C = \{c_1, c_2, ..., c_s\}$ where $c_j \subseteq Z$, $c_j \neq \emptyset$, $c_j \cap c_{j'} = \emptyset$ for $j \neq j'$ and $\cup c_j = Z$. Each subset c_j is called a *cluster* or a *group*. When there is no confusion, we also denote the cluster that a vertex z belongs to as $C(z)$.

By the above definition, the clusters are mutually exclusive, which means that a vertex could belong to exactly one cluster. This seems to be a very strong assumption, since in many real applications a vertex could be involved in multiple clusters. For example, an artist could play

with more than one genres, and a book could have both romantic and SiFi styles. In fact, the mutual exclusion assumption is still valid in such cases because we could create a cluster for each combination of genres or styles. Then the groups of artists who play with different combinations of genres, or the groups of books that have distinct combinations of styles are mutually exclusive.

The optimal partition of the clustering task depends on how we define the similarities between the vertices. In general we have an objective $\Phi(H, C) \in \mathbb{R}$ that tells the goodness of the partition, and the optimal partition would simply minimize (or maximize) $\Phi(H, C)$. Notice that $\Phi(H, C)$ also depends on s (the number of clusters).

2.2.2 Semi-supervised Learning

Another task that is similar to the clustering task is the *semi-supervised learning*. In this task we also would like to find a partition C of the vertex set. But unlike the clustering, the number of clusters s is known and the clusters of some vertices are already given. In other words, we know a *partial clustering* $T = \{t_1, t_2, ..., t_s\}$ where $t_j \subseteq Z$, $t_j \neq \emptyset$ and $t_j \cap t_{j'} = \emptyset$ for $j \neq j'$, but the union $\cup t_j$ is not necessarily equal to Z. The goal of the task is to find a full clustering C that minimizes (or maximizes) $\Phi(H, C)$, where C has the same number of clusters as T and coincides with T on the given vertices $\cup t_j$. Taking the `zoo` dataset as an example, we know the biological classes of some animals (vertices), and we would like to classify the remaining animals by the co-occurrence relations encoded in the hypergraph. If the tiger is known to be in the mammal class, and the cat shares many hyperedges with tiger, cat is probably also in the mammal class.

The same mutually exclusive assumption on the final clustering is applied to the vertices in the semi-supervised learning task.

2.2.3 Link Prediction

The last task can be better described with the bi-partite graph representation. Suppose that the edges in the bi-partite graph G_b represent the co-occurrence relations that we have observed so far. In the *link prediction* task we would like to predict a new set of edges $\hat{E}_{G_b}$ in G_b that would appear in the near future. This prediction is usually based on the existing relations and a model of how the relations are generated.

In our model, we follow the common assumption that a clustering structure C does exist for the vertices on one side of the bi-partite graph, which is usually chosen to be the vertex set Z in the hypergraph. And we also assume that a vertex on the other side of the bi-partite graph (representing a hyperedge $y \in Y$) creates new edges to the vertices in Z according to a

distribution or a preference over the clusters $\{c_1, c_2, ..., c_s\}$. The best clustering over Z is also learned from the objective $\Phi(H, C)$.

For example, in the `amazonbooks` data, each book (vertex) belongs to a category (a cluster, can be SiFi, romance, comic, or a combination of these), and each user (hyperedge) purchases books by his or her preferences over the categories. Once we have identified the book categories and the users' preferences from existing relations, new relations can be predicted by matching the categories and preferences.

2.3 Spectral Graph Theory

All the three tasks listed above depend on the objective $\Phi(H, C)$. In this section, we discuss how the objective $\Phi(H, C)$ is defined and how to find the optimal clustering C by the spectral graph theory.

We start with the simple version of the objective where the hypergraph H is actually a graph G (each hyperedge contains exactly two vertices), so $\Phi(H, C)$ can be written as $\Phi(G, C)$. In this case, each edge in the graph represents a pairwise relation between two vertices. When making a clustering of the vertices, a possible objective is to minimize the number of edges that connect vertices from distinct clusters, which can be formulated as a min-cut problem on the graph. Let a vector $\boldsymbol{c}_j$ of length n denote the membership of vertices for cluster c_j. If a vertex v belongs to the cluster c_j, the v-th entry of $\boldsymbol{c}_j(v)$ is set to 1, otherwise the entry is 0. By the mutual exclusion assumption between the clusters, a 1 should appear only once in the v-th entries of all the membership vectors. Let $\boldsymbol{C} = [\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_s]$ denote the membership matrix. In a min-cut problem, the weighted sum of the edges connecting vertices from distinct clusters is

$$\Phi(G, C) = \frac{1}{2} \sum_{1 \le j \le s} \sum_{e=(v,u) \in E} w(e) \left(\boldsymbol{c}_j(v) - \boldsymbol{c}_j(u) \right)^2 \tag{2.1}$$

$$= \frac{1}{2} \sum_{1 \le j \le s} \boldsymbol{c}_j^\top (\boldsymbol{D} - \boldsymbol{A}) \boldsymbol{c}_j \tag{2.2}$$

$$= \frac{1}{2} \text{trace} \left(\boldsymbol{C}^\top (\boldsymbol{D} - \boldsymbol{A}) \boldsymbol{C} \right), \tag{2.3}$$

where the coefficient $1/2$ comes from the fact that each edge crossing different clusters is counted two times. The matrix $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ is commonly defined as the *combinatorial Laplacian* of the graph. Thus a clustering solution $\hat{\boldsymbol{C}}$ to the min-cut problem would minimize $\Phi(G, C)$

with the mutual exclusive constraint

$$\hat{\boldsymbol{C}} = \arg\min_{\boldsymbol{C}} \frac{1}{2}\text{trace}\left(\boldsymbol{C}^\top \boldsymbol{L}\boldsymbol{C}\right), \tag{2.4}$$

$$\text{subject to: } \boldsymbol{C}(k,j) \in \{0,1\},\ \boldsymbol{C}^\top \boldsymbol{C} = \boldsymbol{S}_d, \tag{2.5}$$

where $\boldsymbol{S}_d$ is a $s \times s$ diagonal matrix containing the cluster sizes. When there are only two clusters, this combinatorial optimization problem can be solved in polynomial time with a max-flow algorithm, but the result usually yields very unbalanced clusters. When some additional constraints on the cluster sizes or the connectivity of partitions are imposed, this problem is shown to be NP-hard [Johnson, 1979, Dyer and Frieze, 1985].

A common technique to tackle problem (2.4) is to relax the values in $\boldsymbol{C}$. Instead of taking value 0 or 1, the entries of $\boldsymbol{C}$ are allowed to take any real value. The last constraint $\boldsymbol{C}^\top \boldsymbol{C} = \boldsymbol{S}_d$ originally ensures that each vertex can be assigned to only one cluster, but now this constraint does not make sense anymore since the membership is a real value. The matrix $\boldsymbol{S}_d$ is a diagonal matrix, which implies that $\boldsymbol{c}_i^\top \boldsymbol{c}_j = 0$ for $\forall i \neq j$. Thus we can translate the original mutual exclusive constraint into a relaxed form

$$\hat{\boldsymbol{C}} = \arg\min_{\boldsymbol{C}} \frac{1}{2}\text{trace}\left(\boldsymbol{C}^\top \boldsymbol{L}\boldsymbol{C}\right), \tag{2.6}$$

$$\text{subject to: } \boldsymbol{C}(k,j) \in \mathbb{R},\ \forall i \neq j,\ \boldsymbol{c}_i^\top \boldsymbol{c}_j = 0,\ \forall j,\ \boldsymbol{1}^\top \boldsymbol{c}_j = 0, \boldsymbol{c}_j^\top \boldsymbol{c}_j = 1. \tag{2.7}$$

The last constraint prevents a relaxed membership from assigning all the vertices in a single cluster or forming an empty cluster. Since $\boldsymbol{L}$ is a real symmetric matrix, by the *Courant–Fischer min-max theorem* the column vectors in the optimal solution $\hat{\boldsymbol{C}}$ should be the eigenvectors of $\boldsymbol{L}$.

Theorem 2.3.1 (Courant–Fischer)**.** *Let $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$ and $\{\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_n\}$ denote the eigenvalues and the corresponding eigenvectors of a $n \times n$ Hermitian matrix $\boldsymbol{M}$. For any $1 \leq r \leq t \leq n$, and $\boldsymbol{g} \in span(\boldsymbol{f}_r, ..., \boldsymbol{f}_t)$ with the constraint $\boldsymbol{g}^\top \boldsymbol{g} = 1$, we have $\lambda_r \leq \boldsymbol{g}^\top \boldsymbol{M}\boldsymbol{g} \leq \lambda_s$.*

The set of eigenvalues and eigenvectors of the Laplacian matrix is also called the *spectrum* of the graph. It is easy to verify that if the graph is connected, the all-ones vector $\boldsymbol{1}$ is an eigenvector of $\boldsymbol{L}$ corresponding to the smallest eigenvalue 0. Then the optimal solution $\hat{\boldsymbol{C}}$ to (2.6) should contain the eigenvectors of $\boldsymbol{L}$ corresponding to the second smallest eigenvalue to the $s+1$ smallest eigenvalue (iteratively consider the subspaces span$(\boldsymbol{f}_r, ..., \boldsymbol{f}_n)$ in Theorem 2.3.1, $r = 2, 3, ..., s+1$). When only two clusters are requested, the eigenvector corresponding to the second smallest eigenvalue, often referred as the *Fiedler vector* [Fiedler, 1973], is the most useful one.

Theorem 2.3.2 (Cheeger)**.** *Let λ_2 denote the second smallest eigenvalue of the combinatorial*

Laplacian of a graph G, $\Phi_{Cheeger} = \min_C \frac{\Phi(G,C)}{\min(|c_1|,|c_2|)}$ *be the Cheeger constant of G in a partition of exactly two clusters. The following always holds [Cheeger, 1970, Chung, 1997]*

$$\frac{1}{2\max_{v\in V} deg(v)}\Phi^2_{Cheeger} \leq \lambda_2 \leq 2\Phi_{Cheeger}. \tag{2.8}$$

This result shows that the spectrum of the Laplacian matrix has a close connection to the clustering structure of the graph. We can bound the optimal min-cut by the eigenvalue of the Laplacian. Obviously the relaxed solution cannot be directly used as a clustering result, usually a post-process step is carried out to convert the real values in $\hat{\boldsymbol{C}}$ back to binary values. Some possible post-process steps are discussed later in this work.

Lemma 2.3.3. *For the combinatorial Laplacian* $\boldsymbol{L}$ *of a connected graph G, if the eigenvalues and the eigenvectors of* $\boldsymbol{L}$ *are* $\sigma_1 \leq \sigma_2 \leq ... \leq \sigma_n$ *and* $\{\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_n\}$*, we have [Chung, 1997]*

1. $\sigma_1 = 0$, *and* $\boldsymbol{f}_1 \propto \boldsymbol{1}$.

2. $\sigma_n = \|\boldsymbol{L}\|_2 \leq 2\max_v deg(v)$, *where* $\|\cdot\|_2$ *denotes the* l_2 *norm of a matrix.*

Proof. The first argument follows directly from the construction of $\boldsymbol{L}$. Let $\boldsymbol{F} = [\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_n]$ be the matrix of the eigenvectors of $\boldsymbol{L}$. The second argument follows the definition of $\|\cdot\|_2$ and the fact that

$$\sigma_n \|\boldsymbol{F}\|_1 = \|\sigma_n \boldsymbol{F}\|_1 = \|\boldsymbol{L}\boldsymbol{F}\|_1 \leq \|\boldsymbol{L}\|_1 \|\boldsymbol{F}\|_1, \tag{2.9}$$

where $\|\cdot\|_1$ denotes the l_1 norm of a matrix. Thus we have

$$\sigma_n \leq \|\boldsymbol{L}\|_1 = \max_j \sum_i |\boldsymbol{L}(i,j)| = 2\max_v deg(v). \tag{2.10}$$

□

In the formulation of the combinatorial Laplacian, we do not apply any control to the cluster sizes. But in many real applications, a balanced clustering is desired since it makes more sense than a trivial clustering where one vertex forms a cluster and all the other vertices form another cluster. One could enforce a hard constrain on the cluster size, while a more sophisticated approach would apply a normalization to the objective for a balanced result. Let $vol(c_j) = \sum_{v\in c_j} deg(v)$ denote the volume of the cluster c_j, and $\bar{c}_j = V \setminus c_j$ denote the compliment of c_j. The new objective normalized by the cluster volumes can be written as [Shi and

Malik, 2000]

$$\Phi_N(G,C) = \sum_{1\le j\le s}\left(\frac{1}{vol(c_j)}+\frac{1}{vol(\bar{c}_j)}\right)\sum_{e=(v,u)\in E} w(e)\left(\boldsymbol{c}_j(v)-\boldsymbol{c}_j(u)\right)^2 \tag{2.11}$$

$$= \sum_{1\le j\le s}\boldsymbol{c}_j^\top\left(\boldsymbol{I}-\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}\right)\boldsymbol{c}_j \tag{2.12}$$

$$= \text{trace}\left(\boldsymbol{C}^\top\left(\boldsymbol{I}-\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}\right)\boldsymbol{C}\right), \tag{2.13}$$

where $\boldsymbol{I}$ is an identity matrix. If a cluster creates a small edge-cut but has even smaller size, the objective would still have a large value. Therefore, the objective $\Phi_N(G,C)$ prefers a balanced clustering compared to the un-normalized version $\Phi(G,C)$. The matrix $\boldsymbol{L}_N = \boldsymbol{I}-\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ is denoted as the *normalized Laplacian*. Similar to Lemma 2.3.3, the following holds for the normalized Laplacian.

Lemma 2.3.4. *For the normalized Laplacian $\boldsymbol{L}_N$ of a connected graph G, if the eigenvalues and the eigenvectors of $\boldsymbol{L}_N$ are $\lambda_1 \le \lambda_2 \le ... \le \lambda_n$ and $\{\boldsymbol{f}_1,\boldsymbol{f}_2,...,\boldsymbol{f}_n\}$, we have [Chung, 1997]*

1. $\lambda_1 = 0$, *and* $\boldsymbol{f}_1 \propto \boldsymbol{D}^{-1/2}\boldsymbol{1}$.
2. $\lambda_n = \|\boldsymbol{L}_N\|_2 \le 2$.
3. $\lambda_1 \le 1$ *if G is not a complete graph.*

We show several examples in Figure 2.2 of the eigenvectors. In the special case where the graph contains some distinct connected components, it is easy to find out which component a vertex belongs to by looking at the corresponding entries of the eigenvectors. This transformation can be also regarded as a projection of the vertices to a lower dimensional space, e.g. 3-d space if the first 3 eigenvectors are used. Such projection is commonly referred as the *spectral embedding*. When the graph is connected but contains several clusters, the spectral embedding would project the vertices belonging to the same cluster to some points that are close to each other in the lower dimensional space. These points can be further processed to reconstruct the clusters.

Some graphs have very clear clustering structure, but some other graphs (especially real-world power-law graphs) does not possess balanced clusters. The differences between these situations can be reflected by the gap between the consecutive eigenvalues, or the *eigen-gap*, in the spectrum of the graph. If the graph consists of s connected components, one can show that the smallest s eigenvalues of $\boldsymbol{L}$ and $\boldsymbol{L}_N$ are all zeros, while all the other eigenvalues are strictly positive (see [Chung, 1997] Lemma 1.7). The multiplicative gap γ_s is defined as $\gamma_s = \frac{\lambda_n-\lambda_{s+1}}{\lambda_n-\lambda_s} \le 1$, where λ_n is the biggest eigenvalue of $\boldsymbol{L}$ or $\boldsymbol{L}_N$. The λ_n part in the formula comes from the fact that most numerical methods for computing the eigenvalues can only

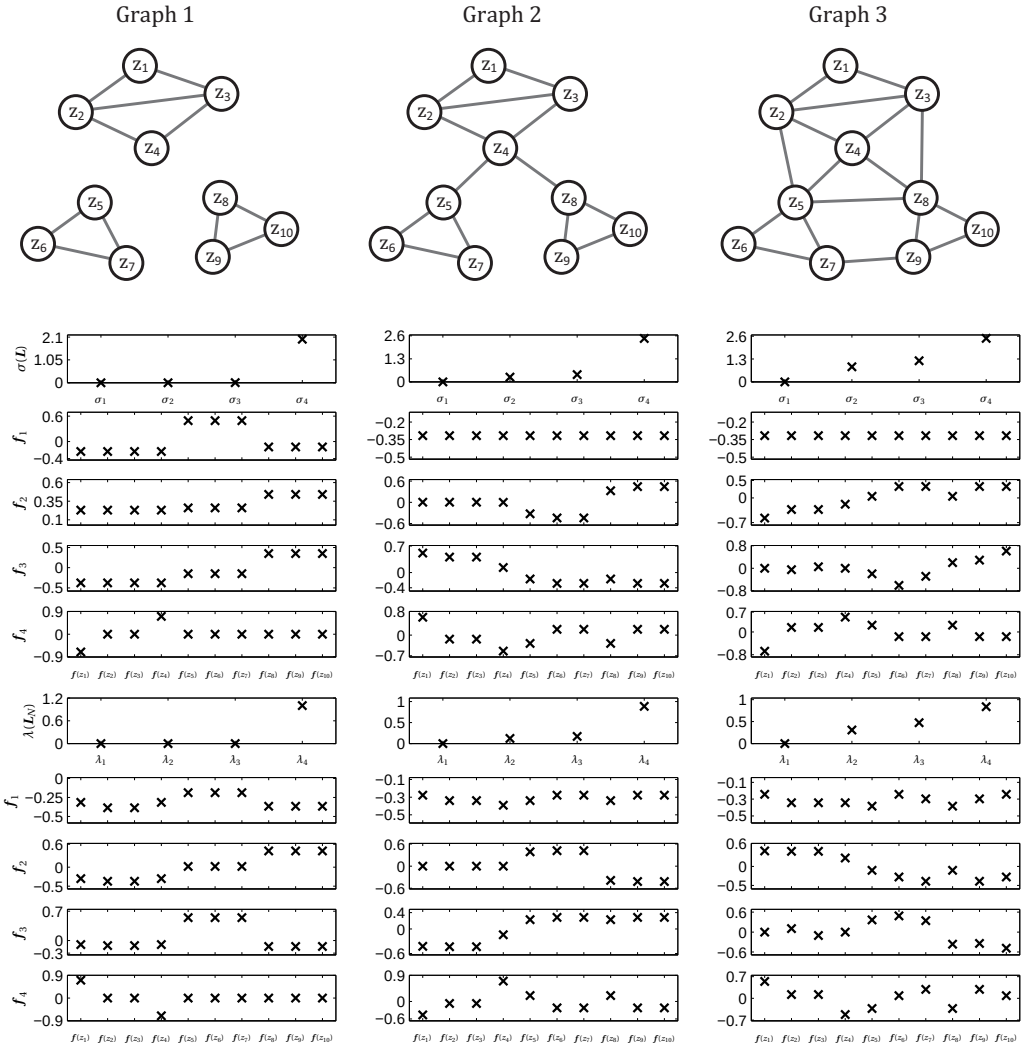

Figure 2.2: Examples of the eigenvectors and eigenvalues of different graphs. The eigenvalues and eigenvectors from both $\boldsymbol{L}$ and $\boldsymbol{L}_N$ of each graph are shown under the graph. There are roughly 3 clusters in each graph, and the first 4 eigenvalues and the corresponding eigenvectors are shown. We could generally distinguish vertices from different clusters by the first 3 eigenvectors $\{\boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{f}_3\}$. The eigen-gap between λ_3 and λ_4 becomes smaller from left to right as the clustering structure is less clear (but the multiplicative gap γ_3 becomes bigger from left to right).

estimate the eigenvalues of the largest absolute values. Thus we have to shift the spectrum of $\boldsymbol{L}$ or $\boldsymbol{L}_N$ to the negative side of the origin. It can be seen in Figure 2.2 that when the boundary between clusters are less clear, γ_s becomes bigger. When computing the first s eigenvalues and eigenvectors of a matrix, the numerical method usually operates in an iterative fashion. The number of iterations is determined by the eigen-gap, depending on the actual numerical method used, e.g. $O(\frac{1}{1-\gamma_s})$ iterations for an orthogonal iterations method, and $O(\frac{1}{\sqrt{1-\gamma_s}})$ iterations for a Lanczos method [Golub and Van Loan, 1996, Bach and Jordan, 2006, Mavroeidis, 2011]. Therefore, it would require less iterations in the eigenvalue computation for a graph with a clear clustering structure.

2.4 Multi-class Beta-Bernoulli Process

Various models have been proposed to describe the process of generating relational data. The nonparametric Bayesian approach provides a scheme in which data are assumed to emerge from some simple probabilistic rules, and yet follow certain distributions (e.g. power-law) in a global view. Since the power-law distribution is often observed in real applications, the nonparametric Bayesian approaches have been well-justified and gained great popularity in existing works [Broderick et al., 2012]. The well-known *Dirichlet process mixture model* includes a distribution over partitions (clusters) of entities, and this distribution can be generated incrementally in a *Chinese restaurant process* (CRP). But if we apply the CRP to generate a hypergraph, a vertex can only be associated with one hyperedge. The *beta-Bernoulli process* removes the above limitation to allow an arbitrary number of relations from the vertices to the hyperedges. This process could be also generated incrementally in the so-called *Indian buffet process* [Thibaux and Jordan, 2007, Griffiths and Ghahramani, 2011]. In this section, we start with a minor variation of the beta-Bernoulli process and model the generated data as a hypergraph.

Suppose we have two sets of entities Y and Z, and the entities in set Z belong to s clusters $\{c_1, c_2, ..., c_s\}$. This setting is exactly the same as in the definition of the hypergraph and the clustering task. Take the `music` data as an example, the set Y is the set of users and Z is the set of artists. Clusters or partitions over Z can be made according to the genres in which the artists play. For example, c_1 includes the rock music artists, while all the country music artists are in c_2. Each user in the set Y, on the other hand, is assumed to have some preferences over the music genres. If the user likes rock music, he would like the artists from c_1 with a high probability. Or the user would rarely choose artists from c_2 if he doesn't like country music. As explained in the previous section, the mutual exclusive assumption over clusters does not prohibit an artist playing in multiple genres, because we could create a cluster for a combination of genres when needed.

A (discrete) beta process $B \sim \mathrm{BP}(\xi, B_0)$ is a random process depending on two parameters: the *concentration function* ξ is a positive function defined on a finite (discrete) space $\Omega = \{\omega_i\}$, and B_0 is of the form $B_0 = \sum_i q_i \delta_{\omega_i}$ where $q_i \in [0,1]$ and δ_{ω_i} is a unit point mass at ω_i. Then the beta process B follows the similar form $B = \sum_i p_i \delta_{\omega_i}$, where p_i is drawn from a beta distribution $p_i \sim \mathrm{Beta}\big(\xi(\omega_i)q_i, \xi(\omega_i)(1-q_i)\big)$. Our model is a simplified version of the model in [Thibaux and Jordan, 2007] where the space Ω is infinite, while our Ω is finite.

Take the `music` example again, we can set $\Omega = Y$. Each q_i corresponds to a user in Y and indicates the user's overall preference to music, while p_i can be seen as the actual probability of observing that the user likes an artist. A q_i close to 1 means that the user generally likes music and thus p_i is also likely to be close to 1, and vice-versa.

The concentration function ξ can be different for each cluster, and for simplicity we assume that the function ξ takes the same value for each ω_i (or y_i), i.e. $\xi(y_i, c_j) = \xi(y_{i'}, c_j) = \xi(c_j)$. If $\xi(c_j)$ is very small (close to 0) for the cluster c_j, a user would either very much like an artist in c_j (p_i^j close to 1) or doesn't like her at all (p_i^j close to 0). Since ξ is different for each cluster, we use p_i^j to indicate the actual probability of observing that the user y_i likes an artist from the cluster c_j. The chance of drawing a p_i^j between 0 and 1 is very small, which means that the samples of p_i^j are *concentrated* on $\{0,1\}$. Figure 2.3 shows several examples of this process with different concentration functions.

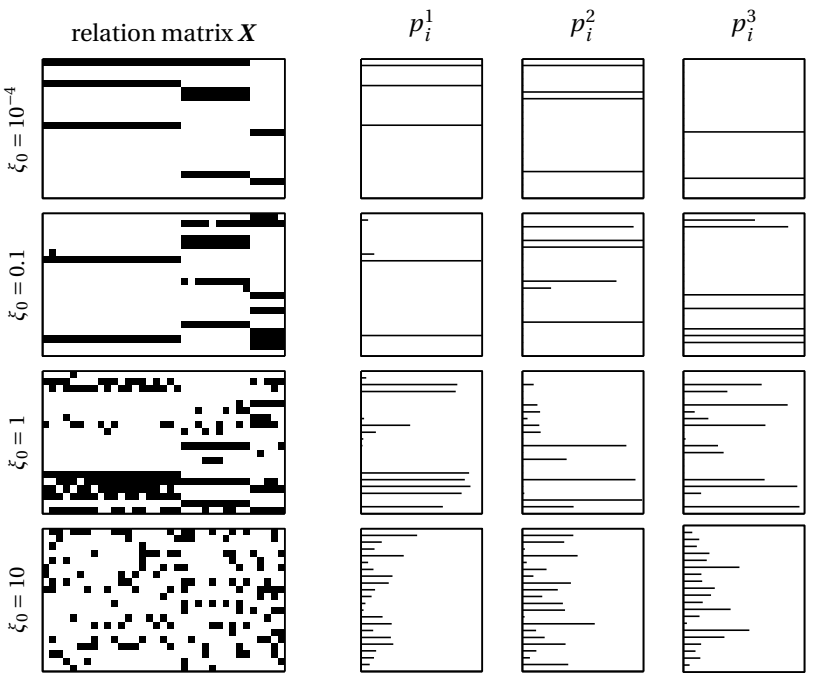

Figure 2.3: Samples drawn from the multi-class beta-Bernoulli process with $m = 10$, $n = 35$, and different concentration parameters ξ_0. There are 3 clusters of sizes $\{20, 10, 5\}$. The parameter $q_0 = 0.2$.

After we have the p_i^j's, the event of observing the user y_i likes an artist from cluster c_j is simply a Bernoulli trial with a success probability p_i^j. Since there are $|c_j|$ artists in the cluster c_j, we just repeat the Bernoulli trial $|c_j|$ times independently. By this second-step process, a binary matrix $\boldsymbol{X}$ of size $m \times n$ is generated (a success in the Bernoulli trial would write a 1 in $\boldsymbol{X}$, and 0 otherwise). This matrix can be also regarded as the incident matrix of a hypergraph. Samples of $\boldsymbol{X}$ are shown in Figure 2.3.

To further capture the difference between the clusters, a decaying parameter $\tau > 0$ is introduced to make the p_i^j smaller as j getting bigger. In summary, an entry $\boldsymbol{X}(i,k)$ of $\boldsymbol{X}$ is generated from the following distributions:

$$p_i^j \overset{i.i.d.}{\sim} \text{Beta}\left(\xi(c_j)q_i, \xi(c_j)(1-q_i) + (j-1)\tau\right), \tag{2.14}$$

$$\boldsymbol{X}(i,k) \overset{i.i.d.}{\sim} \text{Bernoulli}\left(p_i^j\right),\ z_k \in c_j. \tag{2.15}$$

For simplicity, we assume that the parameters are all the same if no special conditions are mentioned, i.e. $\forall i \in \{1,2,...,m\}$, $q_i = q_0$, and $\forall j \in \{1,2,...,s\}$, $\xi(c_j) = \xi_0$.

Broderick et al. show that with the beta-Bernoulli process the distribution of occurrences of entities in Z, i.e. the distribution of column sums of $\boldsymbol{X}$, is asymptotically a power-law when m and s approach infinity [Broderick et al., 2012]. Let K_k denote the number of entities in Z that are associated with exactly k entities in Y. The asymptotic property states that

$$K_k \overset{a.s.}{\sim} \frac{\alpha\Gamma(k-\alpha)}{k!}\beta m^\alpha,\ \ m, s \to \infty, \tag{2.16}$$

where $0 < \alpha < 1$ and $\beta > 0$ are constants, and $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$ is the gamma function. The symbol $\overset{a.s.}{\sim}$ means that the ratio of the left side over the right side is almost surely 1 as $m, s \to \infty$. The modification of adding multiple classes in our model does not change the outline of the proof in [Broderick et al., 2012], so this result can be directly applied to our model.

Given $\xi(c_j)$ and q_i, the rows of $\boldsymbol{X}$ are independently generated. For an entity y_i and a cluster c_j, the number of relations between y_i and the entities in c_j (denoted by $N_{i,j}$) follows a binomial distribution whose parameter p_i^j is drawn from a beta distribution. Let $f(\cdot)$ denote the probability density function (PDF), and p denote p_i^j for simplicity. Without loss of generality, we assume $j = 1$ in the following analysis. The probability of $N_{i,j} = k$ can be written as

$$\mathbb{P}\left(N_{i,j} = k\right) = \int_p f\left(N_{i,j} = k|p\right) f\left(p\right) dp \tag{2.17}$$

$$= \int_p \binom{|c_j|}{k} p^k (1-p)^{|c_j|-k} \frac{p^{\xi(q_i)q_i-1}(1-p)^{\xi(q_i)(1-q_i)-1}}{\mathrm{B}(\xi(q_i)q_i, \xi(q_i)(1-q_i))} dp \tag{2.18}$$

$$= \binom{|c_j|}{k} \frac{\mathrm{B}(\xi(q_i)q_i + k, \xi(q_i)(1-q_i) + |c_j| - k)}{\mathrm{B}(\xi(q_i)q_i, \xi(q_i)(1-q_i))} \int_p \frac{p^{\xi(q_i)q_i+k-1}(1-p)^{\xi(q_i)(1-q_i)+|c_j|-k-1}}{\mathrm{B}(\xi(q_i)q_i + k, \xi(q_i)(1-q_i) + |c_j| - k)} dp \tag{2.19}$$

$$= \binom{|c_j|}{k} \frac{\mathrm{B}(\xi(q_i)q_i + k, \xi(q_i)(1-q_i) + |c_j| - k)}{\mathrm{B}(\xi(q_i)q_i, \xi(q_i)(1-q_i))}, \tag{2.20}$$

where $\mathrm{B}(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$ is the beta function. The integral in (2.19) is the PDF of a beta distribution, so the result is 1. Figure 2.4 shows some examples of the distribution of $N_{i,j}$. We could observe that when $\xi(c_j)$ is small, $N_{i,j}$ is concentrated on 0 or $|c_j|$, and the distribution becomes flat as $\xi(c_j)$ getting bigger. The mass parameter q_i controls the overall density of relations in $\boldsymbol{X}$. A bigger q_i leads to more relations in $\boldsymbol{X}$.

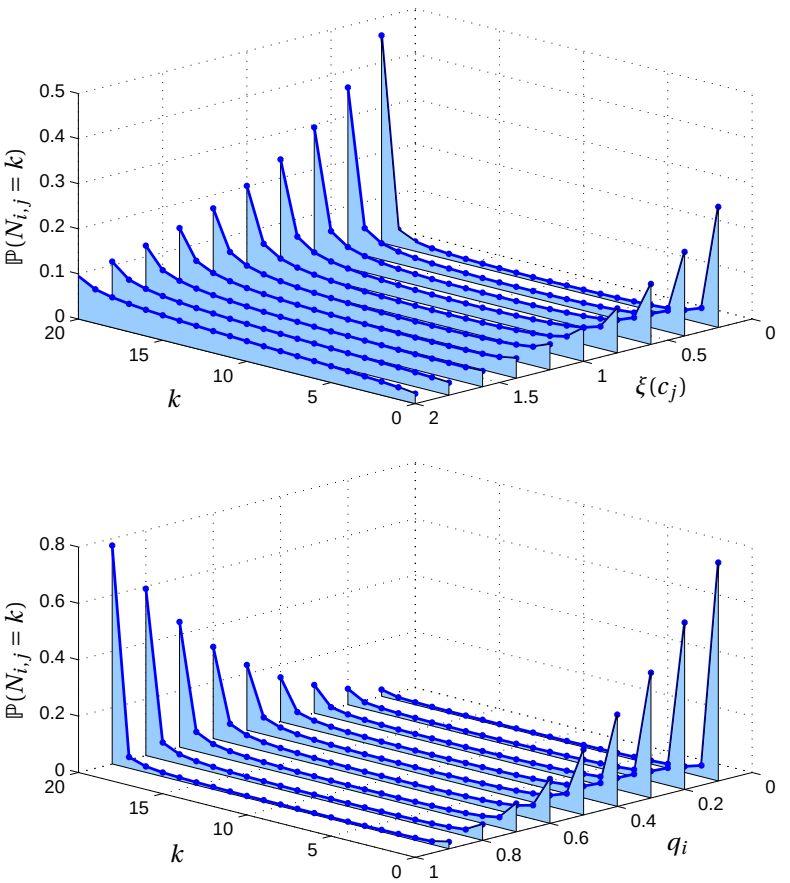

Figure 2.4: Distributions of $N_{i,j}$ with different concentration parameters $\xi(c_j)$ (top) and mass parameters q_i (bottom). For the top figure, $|c_j| = 20$ and $q_i = 0.6$. For the bottom figure, $|c_j| = 20$ and $\xi(q_i) = 0.5$. We assume $j = 1$ in this figure.

For any $z_k \in c_j$, the relation $\boldsymbol{X}(i,k)$ is generated independently, thus the expectation of $N_{i,j}$ can be written as

$$\mathbb{E}\left(N_{i,j}\right) = \mathbb{E}\left(\sum_{k=1}^{|c_j|} \boldsymbol{X}(i,k)\right) \tag{2.21}$$

$$= \sum_{z_k \in c_j} \mathbb{E}\left(\boldsymbol{X}(i,k)\right) \tag{2.22}$$

$$= |c_j| \mathbb{E}\left(\boldsymbol{X}(i,k)\right) \tag{2.23}$$

$$= |c_j| \frac{\mathrm{B}\left(\xi(q_i)q_i + 1, \xi(q_i)(1-q_i)\right)}{\mathrm{B}\left(\xi(q_i)q_i, \xi(q_i)(1-q_i)\right)} \tag{2.24}$$

$$= |c_j| q_i. \tag{2.25}$$

The line of (2.24) comes from a special case of (2.17) where a single entity z_k is considered as a cluster.

When there is more than one cluster, the number of relations between y_i and all the entities in Z is $N_i = \sum_j N_{i,j}$. The relations in each cluster are independently generated, thus we have

$$\mathbb{E}\left(N_i\right) = \sum_{j=1}^{s} \mathbb{E}\left(N_{i,j}\right) \tag{2.26}$$

$$= \sum_{j=1}^{s} \frac{\xi(q_i)q_i}{\xi(q_i) + (j-1)\tau} \tag{2.27}$$

Unfortunately, our model does not possess a power-law distribution of N_i.

In the subsequent sections, the multi-class beta-Bernoulli process is adopted in the learning tasks as the underlying model to generate hypergraphs. With the assumptions associated to the model, we are able to identify the structure of the vertex clusters in the hypergraphs that correspond to the clusters in the model, and the computational costs of the learning algorithms are estimated with the model parameters.

3 Vertex Expansion

The graph Laplacian and the normalized Laplacian, as introduced in the previous chapter, have some properties that are closely connected to the min-cut problem defined on a graph. We have shown how to use the eigenvalues and the eigenvectors of the Laplacian to discover the clusters in a graph. The hypergraph, however, as a higher order structure, cannot directly benefit from the extensively-studied results of graph Laplaicans. Since our learning tasks are defined based a clustering of the vertices, it is reasonable to transform the hypergraph into a graph such that the graph has the same vertex set as the hypergraph. By this transformation, we could make further operations on the transformed graph to reconstruct the clusters of vertices. One can imagine that there are many possible transformations from a hypergraph to a graph, but which one is better in a given situation? Would the induced graph introduced in section 2.1 be a good candidate? In this chapter we discuss the possible transformations and the corresponding algorithms for our learning tasks.

In this chapter, we focus on the transformations that keep the vertex set unchanged or hold all the original vertices in the new graph. Since the transformations are commonly referred as "expansions" in the literatures, the vertex-centric transformations that we discuss in this chapter are called the "vertex expansions". By the vertex expansion, all the original vertices that we would like to cluster or classify are still in the new graph, thus it is possible to directly use the graph Laplaicans in the algorithms. This is an advantage of the vertex expansion compared to the hyperedge expansion (discussed in the next chapter).

We start with a survey of existing vertex expansions in an unified view. Then a special vertex expansion called normalized hypergraph cut is studied for its properties of the embeddings (projections to the lower dimensional space). Finally we present the algorithms for clustering, semi-supervised learning, and link prediction.

Our main contribution in this chapter is the spectral analysis of the embedding structures of

the normalized hypergraph cut based on the multi-class beta-Bernoulli process model. We also design a new link prediction algorithm based on the embedding structures. This algorithm is shown to be better than state-of-the-art methods in a recommender system experiment in terms of accuracy and diversity of recommendations. To our best knowledge, this is the first work to combine a generative hypergraph model with the spectral graph theory.

3.1 Clique Expansion and Star Expansion

Recall the optimization problem (2.6) defined for introducing the graph Laplacian. Essentially, we define a cluster membership indication vector $\boldsymbol{c}$ over the vertex set and try to minimize $\boldsymbol{c}^\top \boldsymbol{L}\boldsymbol{c}$ with the constraint $\boldsymbol{c}^\top \boldsymbol{c} = 1$. In other words, the purpose is to find the best $\boldsymbol{c}$ that agrees with the structure of the graph in the sense that the difference between $\boldsymbol{c}(v)$ and $\boldsymbol{c}(u)$ for any pair (v,u) is varying according to the weight of the edge $e = (v,u)$ (if the edge exists). If we generalize this idea to a hypergraph $H = \{Z, Y, w\}$, the vector $\boldsymbol{c}$ over Z is then supposed to agree with the hyperedges

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}} \sum_{\forall z_1, z_2 \in y,\ y \in Y} w(y) \left(\boldsymbol{c}(z_1) - \boldsymbol{c}(z_2)\right)^2. \tag{3.1}$$

This formula is very similar to (2.6). In fact, if we make a graph with the same vertex set $V = Z$, the indictor vector $\hat{\boldsymbol{c}}$ can be obtained in the new graph by the so called *clique expansion* [Zien et al., 1999, Agarwal et al., 2006]. The clique expansion, by its name, constructs a clique from each hyperedge in the hypergraph. The weight of the edges in the clique is simply the weight of the hyperedge. Then all the cliques from different hyperedges are combined together to form a new graph. The edge wight in the new graph can be written as

$$w(v,u) = \sum_{\forall v,u \in y,\ y \in Y} w(y). \tag{3.2}$$

So the adjacency matrix of the new graph $G_{H,\text{clique}}$ after the clique expansion is

$$\boldsymbol{A}_{H,\text{clique}} = \boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{X} - \boldsymbol{D}_z. \tag{3.3}$$

Then the combinatorial Laplacian or the normalized Laplacian can be obtained from $G_{H,\text{clique}}$ and the eigenvectors can be used for reconstructing clusters.

The clique expansion, however, does not consider the hyperedge sizes when performing the optimization in (3.1). If there is a big hyperedge that contains almost all the vertices, there would be a clique of very large size and all the clustering structures would be diminished in this clique. Since all the hyperedges are treated the same regardless of their sizes, the relative weighting portion of a clique in (3.1) would grow quadratically as the hyperedge size

increases. In extreme cases, although the hyperedge has a small weight, its importance would be inflated due to its large size.

To mitigate the problem of clique expansion, the *star expansion* is proposed with the normalization of the hyperedge size [Zien et al., 1999, Agarwal et al., 2006]. Star expansion works on the bi-partite graph representation of the hypergraph (see section 2.1). In the new graph $G_{H,\text{star}}$ the vertex set is the union $Z \cup Y$. An edge in $G_{H,\text{star}}$ can only connect a vertex $z_k \in Z$ to a vertex $y_i \in Y$. The weight of the edge $e = (z_k, y_i)$ is associated with the original hyperedge weight and normalized by the hyperedge size

$$w(z_k, y_i) = w(y_i)/|y_i|. \tag{3.4}$$

The corresponding adjacency matrix of $G_{H,\text{star}}$ is

$$\boldsymbol{A}_{H,\text{star}} = \begin{bmatrix} 0 & \boldsymbol{D}_y^{-1}\boldsymbol{W}\boldsymbol{X} \\ \boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{D}_y^{-1} & 0 \end{bmatrix}, \tag{3.5}$$

which is a block matrix with blocks on the anti-diagonal. In $G_{H,\text{star}}$ the original vertices are connected through the virtual vertices representing the hyperedges. In fact, the spectral properties of $G_{H,\text{star}}$ remain the same. Agarwal et al. show that the spectrum of the normalized Laplacian of $G_{H,\text{star}}$ is the same as the spectrum of the clique expansion, providing that the weighting function in (3.4) is scaled by $(|y_i|-1)/|y_i|$ [Agarwal et al., 2006].

Starting from the star expansion, it is possible to define more expansions. If we have a eigenvalue problem with a block matrix in the following form

$$\begin{bmatrix} \boldsymbol{I} & -\boldsymbol{M} \\ -\boldsymbol{M}^\top & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{g} \end{bmatrix} = \lambda \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{g} \end{bmatrix}, \tag{3.6}$$

where $\boldsymbol{M}$ is an arbitrary matrix, the partial eigenvector $\boldsymbol{f}$ and the value $(1-\lambda)^2$ is also an eigenpair

$$\boldsymbol{M}\boldsymbol{M}^\top \boldsymbol{f} = (1-\lambda)^2 \boldsymbol{f}. \tag{3.7}$$

It is easy to show that the Laplacian of the star expansion graph $G_{H,\text{star}}$ is in the form of (3.6). By taking the partial eigenvector corresponding the original vertices (the set Z), one can construct a vertex expansion called *normalized hypergraph cut* (NHC) [Zhou et al., 2005a, Agarwal et al., 2006], which is further studied later in this chapter. By taking the partial eigenvector corresponding to the hyperedges (the set Y), we obtain another category of expansions called the *hyperedge expansion*. The hyperedge expansions are studied in the next chapter.

Although the NHC can be derived from the normalized Laplacian of the star expansion with

weighting $w(z_k, y_i) = w(y_i)$, we explicitly show the underlying optimization problem of the NHC. Recall the induced graph defined in section 2.1. The only difference between the induced graph and the graph in the clique expansion is that the hyperedge weights are normalized by the hyperedge sizes in the induced graph. The optimization problem of NHC is in the same manner on the induced graph with the volume normalization (see (2.11))

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}} \left(\frac{1}{vol_{G_H}(\boldsymbol{c})} + \frac{1}{vol_{G_H}(\bar{\boldsymbol{c}})} \right) \sum_{e=(v,u)\in E_{G_H}} w_{G_H}(e) \left(\boldsymbol{c}(v) - \boldsymbol{c}(u)\right)^2, \tag{3.8}$$

where $\boldsymbol{c}$ is a clustering indicator vector. By relaxing $\boldsymbol{c}$ to take any real values, we obtain the relaxed optimization problem and the Laplacian of NHC can be defined as [Zhou et al., 2007]

$$\boldsymbol{L}_{\text{NHC}} = \boldsymbol{I} - \boldsymbol{D}_z^{-1/2} \boldsymbol{X}^\top \boldsymbol{D}_y^{-1} \boldsymbol{W} \boldsymbol{X} \boldsymbol{D}_z^{-1/2}. \tag{3.9}$$

The NHC Laplacian is the most commonly used vertex expansion. Applications such as metabolic pathway prediction and coreference resolution have demonstrated its effectiveness [Mithani et al., 2009, Lang et al., 2009]. However, there is no existing work to show the distribution of the NHC embeddings in the embedding space. We continue in this line of works and illustrate that the vertex clusters in the original hypergraph remain as clusters in the embedding space.

3.2 Normalized Hypergraph Embedding

We have shown in Figure 2.2 that the embeddings of the vertices can be used to find clusters in a graph. Assume that the adjacency matrix of a graph exhibits the following block structure

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 & & & \\ & \boldsymbol{A}_2 & & \\ & & \ddots & \\ & & & \boldsymbol{A}_s \end{bmatrix}, \tag{3.10}$$

which corresponds to s distinct connected components $\{c_1, c_2, ...c_s\}$, and any c_i and c_j are *comparable*, i.e. the second largest eigenvalue of $\boldsymbol{A}_i$ is smaller than the biggest eigenvalue of $\boldsymbol{A}_j$ and vice versa. Prakash et al. and Wu et al. show that the eigenvectors of the largest s eigenvalues of $\boldsymbol{A}$ would have the following structure [Prakash et al., 2010, Wu et al., 2011]

$$\begin{array}{lllllll} \boldsymbol{g}_1 & = [& g_1^1 \cdots g_1^{|c_1|} & 0 & \cdots & 0 &]^\top \\ \boldsymbol{g}_2 & = [& 0 & g_2^1 \cdots g_2^{|c_2|} & \cdots & 0 &]^\top \\ & & & & \cdots & & \\ \boldsymbol{g}_s & = [& 0 & 0 & \cdots & g_s^1 \cdots g_s^{|c_s|} &]^\top. \end{array} \tag{3.11}$$

This implies a set of axis-aligned embeddings of the vertices in the s-dimensional space. The vertices in c_1 are projected to some points on the first axis, the vertices in c_2 are projected to some points on the second axis, etc. Furthermore, it is shown that when the adjacency matrix is disturbed by a noise matrix $\boldsymbol{E}$, the spectral embeddings of the new adjacency matrix $\tilde{\boldsymbol{A}} = \boldsymbol{A}+\boldsymbol{E}$ would lie on s quasi-orthogonal lines in the s-dimensional space. The quasi-orthogonal lines $\{\boldsymbol{r}_1, ..., \boldsymbol{r}_s\}$ are in the following form ([Wu et al., 2011], Theorem 2)

$$\begin{array}{lllllll} \boldsymbol{r}_1 & = [& 1 & \frac{\beta_{1,2}}{\sigma_2-\sigma_1} & \cdots & \frac{\beta_{1,s}}{\sigma_s-\sigma_1} &]^\top \\ \boldsymbol{r}_2 & = [& \frac{\beta_{2,1}}{\sigma_1-\sigma_2} & 1 & \cdots & \frac{\beta_{2,s}}{\sigma_s-\sigma_2} &]^\top \\ & & & & \cdots & & \\ \boldsymbol{r}_s & = [& \frac{\beta_{s,1}}{\sigma_1-\sigma_s} & \frac{\beta_{s,2}}{\sigma_2-\sigma_s} & \cdots & 1 &]^\top, \end{array} \tag{3.12}$$

where $\{\sigma_1, ..., \sigma_s\}$ are the largest s eigenvalues of $\boldsymbol{A}$ and $\beta_{i,j} = \boldsymbol{g}_i^\top \boldsymbol{E}\boldsymbol{g}_j$. Figure 3.1 shows some example embeddings of a disconnected graph and a disturbed graph. This result provides a theoretical foundation by which the practice of using embeddings for clustering is justified. In the embedding space, clustering algorithm like k-means or k-nearest neighbor (kNN) can be further applied. By this result, the cosine distance or other angle-based distances should be chosen in the clustering algorithms, since clusters of points approximately distribute on some lines passing through the origin.

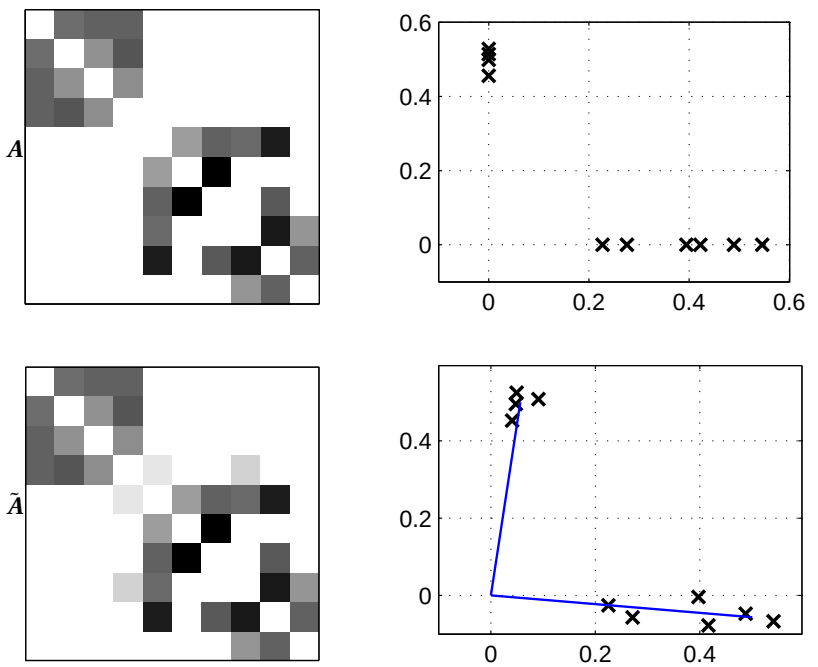

Figure 3.1: Embeddings of a disconnected graph and a disturbed graph. The quasi-orthogonal lines of the disturbed graph are also shown. The graph is weighted, so the gray scale in the adjacency matrix indicates the weights of the edges.

But is there a similar embedding structure when it comes to the hypergraph? In the remainder of this section, we first show that the NHC embeddings can be obtained by a singular value decomposition, then we show that in some cases the orthogonal structures can be observed with the NHC embeddings as well.

Recall the NHC Laplacian defined in (3.9) and denote the smallest l eigen-pairs of $\boldsymbol{L}_{\text{NHC}}$ as $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_l$ and $\{\boldsymbol{f}_1, ..., \boldsymbol{f}_l\}$. Also let $\boldsymbol{F} = [\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_l]$. If we define

$$\bar{\boldsymbol{X}} = \boldsymbol{D}_y^{-1/2} \boldsymbol{X} \boldsymbol{D}_z^{-1/2}, \tag{3.13}$$

the NHC Laplacian can be rewritten as

$$\boldsymbol{L}_{\text{NHC}} = \boldsymbol{I} - \bar{\boldsymbol{X}}^\top \bar{\boldsymbol{X}}, \tag{3.14}$$

and the biggest l eigenvalues of $\bar{\boldsymbol{X}}^\top \bar{\boldsymbol{X}}$ are exactly $\{1-\lambda_1, 1-\lambda_2, ..., 1-\lambda_l\}$, while the corresponding eigenvectors are still $\{\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_l\}$.

Then we decompose the matrix $\bar{\boldsymbol{X}}$ by the singular value decomposition (SVD)

$$\bar{\boldsymbol{X}} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}^\top, \tag{3.15}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are unitary matrices, and $\boldsymbol{\Sigma}$ is a rectangular diagonal matrix. The biggest l singular values in $\boldsymbol{\Sigma}$ are exactly $\{\sqrt{1-\lambda_1}, \sqrt{1-\lambda_2}, ..., \sqrt{1-\lambda_l}\}$, and the columns of $\boldsymbol{V}$ (right-singular vectors) are the eigenvectors of $\bar{\boldsymbol{X}}^\top \bar{\boldsymbol{X}}$. Therefore, instead of computing the eigen-decomposition of $\boldsymbol{L}_{\text{NHC}}$, the matrix $\boldsymbol{F}$ can be obtained by the SVD of $\bar{\boldsymbol{X}}$, i.e. $\boldsymbol{F}$ is the sub-matrix of the first l columns of $\boldsymbol{V}$. Computing $\boldsymbol{F}$ can be also done by the "truncated SVD"

$$\bar{\boldsymbol{X}} \approx \boldsymbol{U}_l \boldsymbol{\Sigma}_l \boldsymbol{V}_l^\top = \boldsymbol{U}_l \boldsymbol{\Sigma}_l \boldsymbol{F}^\top, \tag{3.16}$$

where only the l largest singular values $\boldsymbol{\Sigma}_l$ are calculated. This can be much quicker and more efficient than the full SVD if $l \ll \min(m, n)$.

There are many advantages of using the truncated SVD to compute the l-dimensional embeddings rather than the eigen-decomposition. First, in real applications the matrix $\bar{\boldsymbol{X}}$ is usually sparse, but $\bar{\boldsymbol{X}}^\top \bar{\boldsymbol{X}}$ might be non-sparse. Let $\bar{\boldsymbol{M}} = \bar{\boldsymbol{X}}^\top \bar{\boldsymbol{X}}$. The expected number of zero entries in $\bar{\boldsymbol{M}}$ from the multi-class beta-Bernoulli process (see section 2.4) can be written as

$$\mathbb{E}\left(\sum_{k=1}^{n} \sum_{t=1, t\neq k}^{n} \mathbb{1}(\bar{\boldsymbol{M}}(t,k) = 0) \right) = \sum_{k=1}^{n} \sum_{t=1, t\neq k}^{n} \mathbb{E}\left(\mathbb{1}(\bar{\boldsymbol{M}}(t,k) = 0) \right) \tag{3.17}$$

$$= \sum_{k=1}^{n} \sum_{t=1, t\neq k}^{n} \mathbb{E}\left(\mathbb{1}(\bar{\boldsymbol{X}}(\cdot, t)^\top \bar{\boldsymbol{X}}(\cdot, k) = 0) \right), \tag{3.18}$$

$$= \sum_{k=1}^{n} \sum_{t=1, t\neq k}^{n} \left(\bar{\boldsymbol{X}}(\cdot, t)^{\top} \bar{\boldsymbol{X}}(\cdot, k) = 0 \right), \tag{3.19}$$

where $\bar{\boldsymbol{X}}(\cdot, t)$ denote the t-th column of $\bar{\boldsymbol{X}}$ and $\mathbb{1}$ is the indication function. For some fixed t and k, denote $\bar{X}_i = \bar{\boldsymbol{X}}(i,t)\bar{\boldsymbol{X}}(i,k)$, $i \in \{1,2,...,m\}$. Recall that $\boldsymbol{X}$ is generated from the probabilities p_i^j, and p_i^j is drawn from a beta distribution. Let $f(\cdot)$ denote a probability density function (PDF). If the column $\boldsymbol{X}(\cdot, t)$ and column $\boldsymbol{X}(\cdot, k)$ are generated from the same cluster c_j, we have

$$\mathbb{P}\left(\bar{\boldsymbol{X}}(\cdot, i)^{\top} \bar{\boldsymbol{X}}(\cdot, k) = 0 \right) \tag{3.20}$$

$$= \int_{p_1^j, p_2^j, ..., p_m^j} f\left(\bar{X}_1 = 0, \bar{X}_2 = 0, ..., \bar{X}_m = 0 | p_1^j, p_2^j, ..., p_m^j \right) f\left(p_1^j, p_2^j, ..., p_m^j \right) d p_1^j p_2^j ... p_m^j \tag{3.21}$$

$$= \int_{p_1^j, p_2^j, ..., p_m^j} \prod_{i=1}^{m} f\left(\bar{X}_i = 0 | p_i^j \right) \prod_{i=1}^{m} f\left(p_i^j | \xi_0, q_0 \right) d p_1^j p_2^j ... p_m^j \tag{3.22}$$

$$= \prod_{i=1}^{m} \int_{p_i^j} \left(1 - \left(p_i^j \right)^2 \right) \frac{\left(p_i^j \right)^{\xi_0 q_0 - 1} \left(1 - p_i^j \right)^{\xi_0 (1-q_0) - 1 + (j-1)\tau}}{\mathrm{B}\left(\xi_0 q_0, \xi_0 (1-q_0) + (j-1)\tau \right)} d p_i^j \tag{3.23}$$

$$= \left(1 - \frac{\mathrm{B}\left(\xi_0 q_0 + 2, \xi_0 (1-q_0) + (j-1)\tau \right)}{\mathrm{B}\left(\xi_0 q_0, \xi_0 (1-q_0) + (j-1)\tau \right)} \right)^m \tag{3.24}$$

$$= \left(1 - \frac{\xi_0 q_0 (\xi_0 q_0 + 1)}{\left(\xi_0 + (j-1)\tau \right) \left(\xi_0 + (j-1)\tau + 1 \right)} \right)^m. \tag{3.25}$$

When the column $\boldsymbol{X}(\cdot, t)$ and column $\boldsymbol{X}(\cdot, k)$ are generated from different clusters c_j and $c_{j'}$, we have a similar result

$$\mathbb{P}\left(\bar{\boldsymbol{X}}(\cdot, i)^{\top} \bar{\boldsymbol{X}}(\cdot, k) = 0 \right) \tag{3.26}$$

$$= \prod_{i=1}^{m} \int_{p_i^j} \int_{p_i^{j'}} \left(1 - p_i^j p_i^{j'} \right) f\left(p_i^j | \xi_0, q_0 \right) f\left(p_i^{j'} | \xi_0, q_0 \right) d p_i^j p_i^{j'} \tag{3.27}$$

$$= \left(1 - \left(\frac{\mathrm{B}\left(\xi_0 q_0 + 1, \xi_0 (1-q_0) + (j-1)\tau \right)}{\mathrm{B}\left(\xi_0 q_0, \xi_0 (1-q_0) + (j-1)\tau \right)} \right) \left(\frac{\mathrm{B}\left(\xi_0 q_0 + 1, \xi_0 (1-q_0) + (j'-1)\tau \right)}{\mathrm{B}\left(\xi_0 q_0, \xi_0 (1-q_0) + (j'-1)\tau \right)} \right) \right)^m \tag{3.28}$$

$$= \left(1 - \frac{\xi_0^2 q_0^2}{\left(\xi_0 + (j-1)\tau \right) \left(\xi_0 + (j'-1)\tau \right)} \right)^m. \tag{3.29}$$

It can be seen from (3.25) and (3.29) that if the parameters τ, ξ_0 and q_0 are fixed, the number of non-zero entries in $\boldsymbol{L}_{\mathrm{NHC}} = \boldsymbol{I} - \bar{\boldsymbol{X}}^{\top} \bar{\boldsymbol{X}}$ is proportional to n^2.

When the vertex set size $n = |Z|$ is large, the computational cost and the storage cost of the eigen-decomposition might be impractical, while the truncated SVD can always directly benefit from the sparsity of $\bar{\boldsymbol{X}}$ and scale to a bigger dataset. Secondly, there are existing approaches to implement SVD incrementally, which allows us to just compute the minor

changes when modifying some entries of $\bar{\boldsymbol{X}}$ or including several more rows/columns into $\bar{\boldsymbol{X}}$. In many real applications, such as recommender systems, the ability of incremental computing is often desired.

To show the orthogonal structures with the NHC embeddings, we consider a special case of the multi-class beta-Bernoulli process: the parameters p_i^j are concentrated on $\{0,1\}$. In other words, $\xi(c_j)$ is always close to 0 and p_i^j can only take a value of either 0 or 1 (see the top-left example in Figure 2.3). This special case is known as the *Haldane's prior* in a beta distribution. With this assumption, the matrix $\bar{\boldsymbol{X}}$ can be written as

$$\bar{\boldsymbol{X}} = [\underbrace{\bar{\boldsymbol{x}}_1 \cdots \bar{\boldsymbol{x}}_1}_{|c_1|\text{vectors}} \underbrace{\bar{\boldsymbol{x}}_2 \cdots \bar{\boldsymbol{x}}_2}_{|c_2|\text{vectors}} \cdots \underbrace{\bar{\boldsymbol{x}}_s \cdots \bar{\boldsymbol{x}}_s}_{|c_s|\text{vectors}}], \tag{3.30}$$

where the column vectors in one cluster are all the same. The rank of $\bar{\boldsymbol{X}}$ is s or smaller than s. Thus the truncated SVD $\bar{\boldsymbol{X}} = \boldsymbol{U}_l \boldsymbol{\Sigma}_l \boldsymbol{V}_l^\top$ is not an approximation but an exact decomposition when $l = s$. Recall that the projections in $\boldsymbol{F}$ can be obtained from $\boldsymbol{F} = \boldsymbol{V}_l$. The following theorem shows the structure of $\boldsymbol{F}$.

Theorem 3.2.1. *If $\bar{\boldsymbol{X}}$ is generated from a multi-class beta-Bernoulli process of s clusters with the Haldane's prior, and we compute the embeddings by the truncated SVD $\bar{\boldsymbol{X}} = \boldsymbol{U}_s \boldsymbol{\Sigma}_s \boldsymbol{F}^\top$, rows of $\boldsymbol{F} = [\boldsymbol{\alpha}_1 \cdots \boldsymbol{\alpha}_n]^\top$ can be grouped by the clusters $\boldsymbol{\alpha}_k = \boldsymbol{\beta}_j$ for $\forall z_k \in c_j$. The $\boldsymbol{\alpha}_k$'s from the same cluster are identical (denoted as $\boldsymbol{\beta}_j$). Furthermore we have $\boldsymbol{\beta}_j^\top \boldsymbol{\beta}_{j'} = 0$ for $\forall j \neq j'$.*

Proof. It is easy to show that $\boldsymbol{\alpha}_k = \boldsymbol{\alpha}_{k'} = \boldsymbol{\beta}_j$ if $z_k, z_{k'} \in c_j$, because the k-th column and the k'-th column of $\boldsymbol{\Sigma}_s \boldsymbol{F}^\top$ must be the same to obtain the same $\bar{\boldsymbol{x}}_k$ and $\bar{\boldsymbol{x}}_{k'}$ in the truncated SVD, which implies that $\boldsymbol{\alpha}_k = \boldsymbol{\alpha}_{k'}$. Then we consider the full SVD of $\bar{\boldsymbol{X}}$. Since $\boldsymbol{\alpha}_k = \boldsymbol{\alpha}_{k'} = \boldsymbol{\beta}_j$ for $z_k, z_{k'} \in c_j$, we can list the rows of $\boldsymbol{V}$ corresponding to the entities in cluster c_j as

$$\boldsymbol{V}_{(j)}^\top = \begin{bmatrix} \boldsymbol{\beta}_j & \boldsymbol{\beta}_j & \cdots & \boldsymbol{\beta}_j \\ \boldsymbol{\gamma}_1 & \boldsymbol{\gamma}_2 & \cdots & \boldsymbol{\gamma}_{|c_j|} \end{bmatrix}. \tag{3.31}$$

The vectors $\{\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2, ..., \boldsymbol{\gamma}_{|c_j|}\}$ are not necessarily the same. On the other hand, it is always possible to find a linear combination of the first s columns of $\boldsymbol{V}$ (denoted as $[\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_s] = \boldsymbol{F}$) such that

$$\begin{aligned} \boldsymbol{F}\boldsymbol{t}_1 &= [\ 1\cdots1 \quad 0\cdots0 \quad \cdots \quad 0\cdots0\ ]^\top \\ \boldsymbol{F}\boldsymbol{t}_2 &= [\ 0\cdots0 \quad 1\cdots1 \quad \cdots \quad 0\cdots0\ ]^\top \\ &\qquad\qquad\qquad \cdots \\ \boldsymbol{F}\boldsymbol{t}_s &= [\ \underbrace{0\cdots0}_{|c_1|} \quad \underbrace{0\cdots0}_{|c_2|} \quad \cdots \quad \underbrace{1\cdots1}_{|c_s|}\ ]^\top \end{aligned} \tag{3.32}$$

where the 1's are in the entries corresponding to all the entities in c_j, and $\boldsymbol{t}_j$ are the coeffi-

cients. Because the rank of $\boldsymbol{F}$ is s. Since all the columns in $\boldsymbol{V}$ are orthogonal to each other, a column of $\boldsymbol{V}_{(j)}$ is also orthogonal to the linear combination $\boldsymbol{F}\boldsymbol{t}_j$ of some other columns. Therefore, the entries in each dimension of the vectors $\{\boldsymbol{\gamma}_1, ..., \boldsymbol{\gamma}_{|c_j|}\}$ sum up to 0. In other words, for $\forall c_j$ we have

$$\sum_{p=1}^{|c_j|} \boldsymbol{\gamma}_p = 0, \tag{3.33}$$

which implies that $\boldsymbol{V}_{(j)}^\top \boldsymbol{1} = |c_j| \left[\boldsymbol{\beta}_j^\top \boldsymbol{0}^\top\right]^\top$. Since all the rows in $\boldsymbol{V}$ are also orthogonal to each other, we have $\left(\boldsymbol{V}_{(j)}^\top \boldsymbol{1}\right)^\top \left(\boldsymbol{V}_{(j')}^\top \boldsymbol{1}'\right) = 0$ for $\forall j \neq j'$. This concludes to $\boldsymbol{\beta}_j^\top \boldsymbol{\beta}_{j'} = 0$. □

When $\xi_0 \to 0$, the embeddings of the vertices in the s-dimensional space form exactly s clusters and the vectors of the cluster centers are orthogonal to each other. Figure 3.2 shows several examples of the embeddings with different concentration parameter. This result extends the line orthogonality structure to hypergraphs. Unlike the graph case, the embeddings are not aligned to the axes. The rotation from the axes to the orthogonal projections can be arbitrary, depending on the sizes of clusters and the data distribution in the matrix.

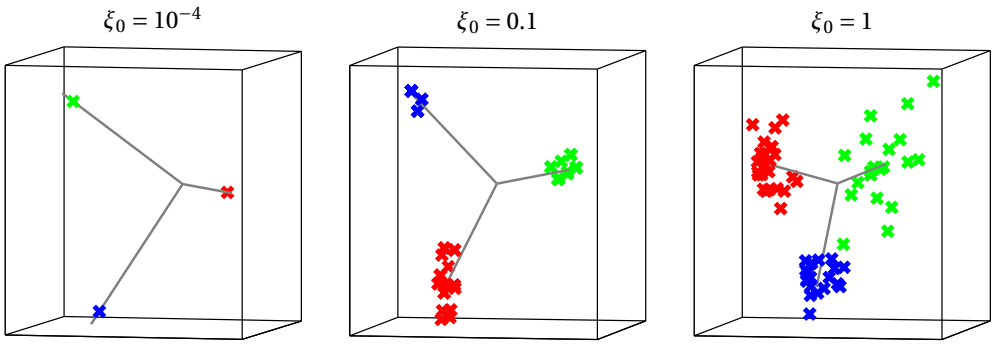

Figure 3.2: Hypergraph embeddings with different concentration parameters. The hypergraphs are randomly generated by the multi-class beta-Bernoulli process (3 clusters) with $q_0 = 0.3$ and concentration parameters shown in the figure. The outer boxes are the axes and the gray lines are lines from the origin to the cluster centers.

Although the embeddings are not directly aligned with the axes, we can make the possible alignments by a rotation. Then the embeddings would have a similar structure as in (3.11), and each dimension of the embedding space can be considered as a latent factor that corresponds to a cluster in the original data. This idea of latent factors has been widely applied in many other methods, either explicitly or implicitly.

In the *non-negative matrix factorization* (NMF), the data matrix $\boldsymbol{X}$ is directly decomposed into two low-rank, sparse and non-negative matrices $\boldsymbol{X} \approx \boldsymbol{G}\boldsymbol{H}^\top$ such that $\|\boldsymbol{X} - \boldsymbol{G}\boldsymbol{H}^\top\|_2$ is minimized [Hoyer, 2004, Chen et al., 2008]. The orthogonal latent factors are explicitly assumed

to control the values in $\boldsymbol{X}$, and each latent factor takes one axis in the low-rank approximation. In the `music` example, a row of $\boldsymbol{H}$ represents an artist, and each entry of the row vector indicates one associated latent attribute. A positive entry means that the artist has this latent attribute, and the corresponding entry in the user matrix $\boldsymbol{G}$ indicates if the user has the same latent attribute. A match of the latent attributes on the same dimension between a user and an artist would result in a positive entry in $\boldsymbol{X}$. The sparsity constraints on $\boldsymbol{G}$ and $\boldsymbol{H}$ ensure that each user and each artist could only be associated with a limited number of latent attributes.

Our model is similar to NMF in the sense that the latent factors are also orthogonal in the latent (embedding) space, although not aligned to the axes. But an entity in Z could be associated with multiple latent factors in NMF, while only one latent factor is allowed in our model. For the entities in Y, both approaches allow multiple associations to the latent factors. The non-negative constraint is unnecessary in our model since the embedding structure automatically implies it.

3.3 Clustering and Semi-supervised Learning

Once we have the embeddings of the vertices, the learning tasks can be carried out by additional procedures. We list below the algorithms for the tasks of clustering and semi-supervised learning using vertex expansions.

The clustering algorithm is denoted as `HCluster[VE]` where `VE` stands for the vertex expansion or the specific Laplacian to compute the embeddings.

Algorithm 1 The vertex expansion clustering algorithm

1: **procedure** `HCLUSTER[VE]` ($H = \{Z, Y, w\}$, l, s)
2: Compute the Laplacian $\boldsymbol{L}_{\texttt{[VE]}}$ from H with the selected vertex expansion `VE`
3: Compute the eigenvectors corresponding to the smallest l eigenvalues of $\boldsymbol{L}_{\texttt{[VE]}}$, and place the eigenvectors in the $|Z| \times l$ matrix $\boldsymbol{F}$
4: When $\boldsymbol{F}$ can be computed from the truncated SVD instead of the eigen-decomposition, replace the previous two steps with the truncated SVD
5: **return** a hard clustering $\hat{\boldsymbol{C}}$ from k-means or kNN that takes s (the desired number of clusters or the number of neighbors) and the rows of $\boldsymbol{F}$ as inputs
6: **end procedure**

As shown in the previous section, the cosine distance or another angle-based distance function would make most sense to compute the distances between the rows of $\boldsymbol{F}$ in the k-means or kNN algorithm. In theory, the embedding dimensionality l should be chosen by looking for the biggest eigen-gap. But a clear eigen-gap is usually absent in most real datasets that exhibit power-law degree distributions. A common practice is to use cross-validation to choose the best l.

In the semi-supervised learning setting, the clusters of some vertices are already known. This known part can be represented by the partial indicator matrix $\boldsymbol{T}$ of size $|Z| \times s$ where s is the given number of clusters. An entry $\boldsymbol{T}(k, j) = 1$ if the vertex z_k is known to be in the cluster c_j, otherwise $\boldsymbol{T}(k, j) = 0$. We assume that at least one vertex from each cluster is labeled in $\boldsymbol{T}$, i.e. the column sums of $\boldsymbol{T}$ are all greater than 0. Then the goal of semi-supervised learning is to find a full clustering $\hat{\boldsymbol{C}}$ that minimizes both $\Phi(H, \hat{\boldsymbol{C}}) = \text{trace}(\hat{\boldsymbol{C}}^\top \boldsymbol{L}_{\texttt{[VE]}} \hat{\boldsymbol{C}})$ and the empirical loss between $\hat{\boldsymbol{C}}$ and $\boldsymbol{T}$. The first part can be solved by a relaxation, and the second part (empirical loss) can be defined as the norm of $\hat{\boldsymbol{C}} - \boldsymbol{T}$. Putting them together, we have the relaxed solution as follows

$$\hat{\boldsymbol{F}} = \arg\min_{\boldsymbol{F}} \text{trace}\left(\boldsymbol{F}^\top \boldsymbol{L}_{\texttt{[VE]}} \boldsymbol{F}\right) + \mu \text{trace}\left((\boldsymbol{F} - \boldsymbol{T})^\top (\boldsymbol{F} - \boldsymbol{T})\right), \tag{3.34}$$

$$\text{subject to: } \boldsymbol{F}(k, j) \in \mathbb{R}, \tag{3.35}$$

where $\boldsymbol{L}_{\texttt{[VE]}}$ is the selected Laplacian of a vertex expansion, and $\mu > 0$ is a parameter to specify the balance between the two terms. By taking the partial derivative with respect to $\boldsymbol{F}$ and setting the formula to zero, it can be shown that the optimal $\hat{\boldsymbol{F}}$ is

$$\hat{\boldsymbol{F}} = \left(\boldsymbol{L}_{\texttt{[VE]}} + \mu \boldsymbol{I}\right)^{-1} \boldsymbol{T}. \tag{3.36}$$

The $\left(\boldsymbol{L}_{\texttt{[VE]}} + \mu \boldsymbol{I}\right)$ part is positive definite since the smallest eigenvalue of $\boldsymbol{L}_{\texttt{[VE]}}$ is 0 (see Lemma 2.3.4). The algorithm `HSS[VE]` is summarized as below (the `SS` stands for "semi-supervised").

Algorithm 2 The vertex expansion semi-supervised learning algorithm

1: **procedure** `HSS[VE]` ($H = \{Z, Y, w\}$, $\boldsymbol{T}$, μ)
2: Initialize $\hat{\boldsymbol{C}}$
3: Compute the Laplacian $\boldsymbol{L}_{\texttt{[VE]}}$ from H with the selected vertex expansion `VE`
4: Compute the relaxed solution $\hat{\boldsymbol{F}} = \left(\boldsymbol{L}_{\texttt{[VE]}} + \mu \boldsymbol{I}\right)^{-1} \boldsymbol{T}$
5: For each vertex z_k which is not labeled in $\boldsymbol{T}$, let $\hat{\boldsymbol{C}}(k, j) = 1$ where $j = \arg\max_p \hat{\boldsymbol{F}}(k, p)$, and set all the other entries of the k-th row of $\hat{\boldsymbol{C}}$ to 0
6: **return** a hard clustering that combines $\hat{\boldsymbol{C}}$ and $\boldsymbol{T}$
7: **end procedure**

This algorithm actually takes the common one-over-all scheme for the semi-supervised learning, i.e. each cluster is chosen to make a bi-clustering with respect to all the other clusters. This is done implicitly in each dimension of $\boldsymbol{F}$ in (3.36). The values in $\hat{\boldsymbol{F}}$ are then treated as scores, and an unlabeled vertex is assigned to the cluster which has the highest score.

The running time of Algorithm 1 mainly depends on the eigen-decomposition of the Laplacian. The common routines for computing a few eigen-pairs are all iterative algorithms. For example, the Lanczos method for Hermitian matrices computes a set of vectors over the iterations and reconstruct the eigenvectors in the end. The basic operation in each iteration is the

matrix-vector multiplication where the matrix is usually sparse and the vector is dense. This operation can be done in $O(M)$ time if M is the number of non-zeros in the matrix (e.g. the non-zeros in $\boldsymbol{L}_{[\texttt{VE}]}$). On the other hand, the number of iterations to converge depends on the eigen-gap (see the end of section 2.3). There are various ways from the numerical computing research to accelerate the convergence, but most approaches rely on certain structures of the matrix. In the case where the eigen-decomposition can be replaced by a SVD of matrix $\bar{\boldsymbol{X}}$ (see equation 3.16), the truncated SVD routine usually takes the eigen-decomposition approach on the matrix $\bar{\boldsymbol{X}}^\top \bar{\boldsymbol{X}}$, i.e. the matrix-vector multiplication is replaced by two consecutive matrix-vector multiplications (first $\bar{\boldsymbol{X}}$ then $\bar{\boldsymbol{X}}^\top$). The two multiplications could take less time because $\bar{\boldsymbol{X}}$ is sparse while $\bar{\boldsymbol{X}}^\top \bar{\boldsymbol{X}}$ is not (see the results of (3.25) and (3.29)). Furthermore, by the two multiplications there is no need for the large space to store $\bar{\boldsymbol{X}}^\top \bar{\boldsymbol{X}}$.

The Algorithm 2 involves in solving a linear system of size n. In general one can compute the matrix inverse $\left(\boldsymbol{L}_{[\texttt{VE}]} + \mu \boldsymbol{I}\right)^{-1}$ and multiply it with $\boldsymbol{T}$, but more efficient algorithm does exist since $\boldsymbol{L}_{[\texttt{VE}]} + \mu \boldsymbol{I}$ is sparse, symmetric, and diagonally-dominate. Spielman and Teng show that approximately solving such linear system can be done in $O(M^{1.31})$ time [Spielman and Teng, 2003].

3.4 Link Prediction

The third learning task, link prediction, works on the bi-partite graph representation. Predicting more edges in the bi-partite graph is the same as predicting more 1's in the relation matrix $\boldsymbol{X}$. We make the predictions based on the multi-class beta-Bernoulli process. Once the entities in Z are projected into some clusters of embeddings in the l-dimensional embedding space, we model an entity in Y as another vector in the same space. A vector $\boldsymbol{\theta}_i$ of length l is assigned to each $y_i \in Y$. Then a prediction score is computed for the possible relation between y_i and z_k

$$P(y_i, z_k) = \boldsymbol{\theta}_i^\top \boldsymbol{\alpha}_k, \tag{3.37}$$

where $\boldsymbol{\alpha}_k$ is the embedding of z_k. The vector $\boldsymbol{\theta}_i$ should take the values such that the predictor P coincides with the existing relations (or as close as possible).

In many applications, the relation matrix contains not only binary relations but also weights of the relations (e.g. ratings in a recommender system). In this case, we have an edge-weighted bi-partite graph and a weighted relation matrix $\boldsymbol{R}$ where $\boldsymbol{R}(i,k) \in \mathbb{R}^+ \cup \{0\}$. But the embeddings are still learned from the unweighted relations $\boldsymbol{X}$ which is extracted from $\boldsymbol{R}$, i.e. $\boldsymbol{X} = \text{sign}(\boldsymbol{R})$ where $\text{sign}(x) = 1$ if $x > 0$, $\text{sign}(x) = -1$ if $x < 0$, and $\text{sign}(x) = 0$ when $x = 0$.

The link prediction algorithm first takes $\boldsymbol{X}$ to compute the vertex embeddings $\boldsymbol{F}$ with the NHC

Laplacian (let $\boldsymbol{\alpha}_k$'s denote the rows of $\boldsymbol{F}$). The points in Figure 3.3 are examples of $\boldsymbol{\alpha}_k$. Then the vector $\boldsymbol{\theta}_i$ for the entity y_i is calculated based on the original weighted relation matrix $\boldsymbol{R}$ and the embeddings $\boldsymbol{F}$. Let $\boldsymbol{R}(i,\cdot)$ denote the i-th row of $\boldsymbol{R}$. The vector $\boldsymbol{\theta}_i$ is determined by

$$\boldsymbol{\theta}_i = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{R}(i,\cdot) - \boldsymbol{\theta}^\top \boldsymbol{F}^\top\|_2. \tag{3.38}$$

Figure 3.3 explains equation (3.38) in an illustrative way. The red points in Figure 3.3 are the $\boldsymbol{\alpha}_k$'s that correspond to the entities related to y_i. For example, they could be the embeddings of the books that the user y_i has read. We would like to choose a $\boldsymbol{\theta}_i$ that is close to these selected $\boldsymbol{\alpha}_k$'s. When the $\boldsymbol{R}(i,\cdot)$ is weighted, $\boldsymbol{\theta}_i$ should be even closer to the $\boldsymbol{\alpha}_k$'s with higher weights, which can be written as

$$\boldsymbol{\theta}_i = \arg\min_{\boldsymbol{\theta}} \sum_{z_k \in y_i} \left(\boldsymbol{R}(i,k) - \boldsymbol{\theta}_i^\top \boldsymbol{\alpha}_k\right)^2. \tag{3.39}$$

For those $\boldsymbol{\alpha}_k$'s that are not related to y_i (the gray points in Figure 3.3), they should stay as far from $\boldsymbol{\theta}_i$ as possible, ideally orthogonal to $\boldsymbol{\theta}_i$. This can be written as

$$\boldsymbol{\theta}_i = \arg\min_{\boldsymbol{\theta}} \sum_{z_k \notin y_i} \left(\boldsymbol{\theta}_i^\top \boldsymbol{\alpha}_k\right)^2. \tag{3.40}$$

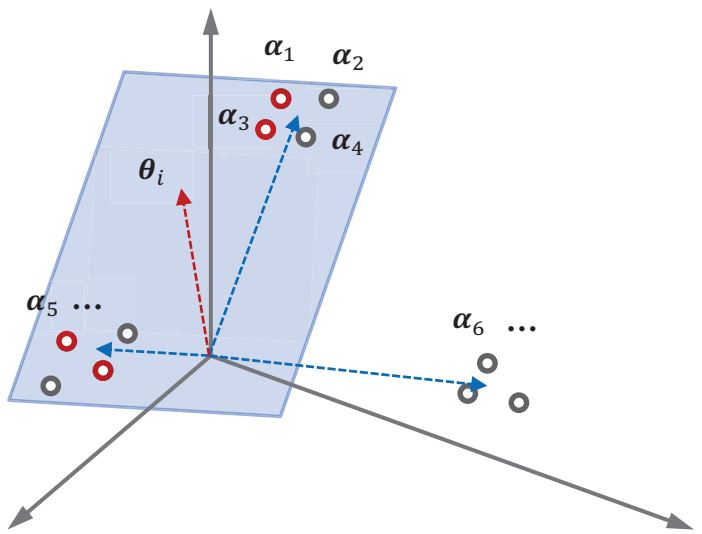

Figure 3.3: An illustration of the link prediction algorithm.

Putting (3.39) and (3.40) together, we obtain (3.38). By this formulation, the vector $\boldsymbol{\theta}_i$ should be approximately in the subspace spanned by the related $\boldsymbol{\alpha}_k$'s (the hyperplane in Figure 3.3). In the `music` example, a user's latent attributes $\boldsymbol{\theta}_i$ is roughly represented by a vector in the subspace spanned by the embedding vectors of the artists that the user has visited, and approximately orthogonal to the subspace spanned by the embedding vectors of the unrelated

artists. Finally a prediction score $P(y_i, z_k)$ is computed by (3.37), and we could make predictions by selecting the unseen relations of the highest scores.

The full link prediction algorithm is presented in Algorithm 3. Since our approach works with the hypergraph embedding by the NHC Laplacian and the truncated SVD, we call it `PredictNHC`. The main operation in Algorithm 3 is the truncated SVD. Thus the computational cost is similar to Algorithm 1 with the NHC Laplacian.

Algorithm 3 The hypergraph link prediction algorithm

1: **procedure** PREDICTNHC($G_b = \{Z \cup Y, E_b, w\}$, l) ▷ G_b is a weighted bi-partite graph representation
2: Convert G_b to a weighted relation matrix $\boldsymbol{R}$ of size $|Y| \times |Z|$. $\boldsymbol{R}(i,k) = w\big((y_i, z_k)\big)$ for $\forall (y_i, z_k) \in E_b$
3: Let $\boldsymbol{X} = \text{sign}(\boldsymbol{R})$
4: Compute $\bar{\boldsymbol{X}}$ from $\boldsymbol{X}$ by (3.13)
5: Compute the l-dimensional hypergraph embeddings $\boldsymbol{F}$ by the truncated SVD $\bar{\boldsymbol{X}} \approx \boldsymbol{U}_l \boldsymbol{\Sigma}_l \boldsymbol{F}^\top$
6: Obtain $\boldsymbol{\Theta}$ by solving $\boldsymbol{F}\boldsymbol{\Theta} = \boldsymbol{R}^\top$ in a least-squares sense
7: Compute prediction score matrix $\boldsymbol{P} = \boldsymbol{\Theta}^\top \boldsymbol{F}^\top$
8: **return** a set of relations of the highest scores in $\boldsymbol{P}$ that are unseen in G_b (possibly ordered by the scores)
9: **end procedure**

Our algorithm takes into account only the right singular vectors, so the basic assumption must be verified when using this approach. In the multi-class beta-Bernoulli process, it assumes that the set Z can be partitioned into some non-overlapping clusters. Therefore, it makes more sense to fill the set Z with the artists in the `music` example. Normally an artist performs in a limited number of genres, while a user could be in fond of a wider range of genres. This assumption can be a limitation with a dataset where clustering structures are hard to find. But if the intuition behind the clusters follows some categorical attributes, e.g. genres, languages, professions, etc., or a combination of categorical attributes, our basic assumption would always hold.

In Algorithm 3 the embeddings in $\boldsymbol{F}$ is usually stationary over time, because the embeddings are determined by all the relations in the dataset and the overall distribution of relations would not change very often. On the other hand, for a specific entity y_i, the $\boldsymbol{\theta}_i$ could change very rapidly. For example, a user would suddenly start to like new a genre. In practice, the values in $\boldsymbol{F}$ can be stored in the system and refreshed periodically (e.g. once every week), while the $\boldsymbol{\theta}_i$'s should be computed on the fly. In the `music` example, when a user y_i asks for more artists, $\boldsymbol{\theta}_i$ is calculated online and destroyed after the prediction scores $\boldsymbol{P}(i, \cdot)$ are obtained. If $|Z|$ is much smaller than $|Y|$, e.g. there is often much less artists than users, we could save a lot of storage space for only saving $\boldsymbol{F}$.

3.4.1 Related Works

Algorithm 3 falls into the category of matrix factorization approaches, since the main ingredient of our proposed method is the SVD. The use of matrix factorization for link prediction has been studied for a long time. For the purpose of comparison, we list some other existing approaches for the link prediction task.

Sarwar et al. proposed one of the early works that adopt SVD to predict more relations [Sarwar et al., 2000]. The method is straightforward: use the low-rank SVD approximation to fill the missing values in the original relation matrix. If the existing relations (weighted) are provided in the matrix $\boldsymbol{R}$, the matrix of prediction scores is simply

$$\boldsymbol{P} = \boldsymbol{U}_l \boldsymbol{\Sigma}_l \boldsymbol{V}_l^\top, \text{ from truncated SVD: } \boldsymbol{R} \approx \boldsymbol{U}_l \boldsymbol{\Sigma}_l \boldsymbol{V}_l^\top. \tag{3.41}$$

We denote this method as `PredictSVDR` since it works directly on the weighted relation matrix $\boldsymbol{R}$. Imputation of missing values is implicitly required in this method when performing SVD, and all the three resulting matrices of the SVD are used in the prediction scores.

To make a better baseline, we adapt our setting to the `PredictSVDR` to first compute SVD and then predict with only the right singular vectors in an asymmetric manner, i.e. follow the same procedures in Algorithm 3 but replace $\bar{\boldsymbol{X}}$ with $\boldsymbol{R}$ in step 5. This modified algorithm is called `PredictSVDR2`.

The NMF is another matrix factorization technique that could replace SVD in many relational learning problems. As explained above, the NMF explicitly assumes that there are l latent dimensions in the latent space and each entity in Y or Z can take a few non-zero values along the l dimensions. The non-negative constraint implies that an entity can only be positively associated with a latent factor. For example, a user could like a genre of artists or stand in a neutral position, but the user is not allowed to hate a genre. The prediction scores $\boldsymbol{P} = \hat{\boldsymbol{G}}\hat{\boldsymbol{H}}^\top$ are computed from the low-rank approximation of NMF [Hoyer, 2004, Chen et al., 2008]

$$\hat{\boldsymbol{G}}, \hat{\boldsymbol{H}} = \arg\min_{\boldsymbol{G},\boldsymbol{H}} \|\boldsymbol{R} - \boldsymbol{G}\boldsymbol{H}^\top\|_2, \tag{3.42}$$

$$\text{subject to: } \boldsymbol{G}(i,j) \geq 0, \boldsymbol{H}(i,j) \geq 0. \tag{3.43}$$

Since the optimization problem of NMF is not convex, an alternating algorithm is often adopted to optimize either $\boldsymbol{G}$ or $\boldsymbol{H}$ in each iteration while fixing another one [Lin, 2007]. Although this algorithm would eventually converge, the final solution would depend on the initial values and may not be the global optimal. The randomness with the final solutions makes it hard to interpret the latent attributes. We denote this method as `PredictNMFR`.

Similar to the `PredictSVDR2`, the NMF decomposition is also adapted into our scheme. The

modified algorithm `PredictNMFR2` follows the same procedures in Algorithm 3, but in step 5 the embeddings are replaced by $\boldsymbol{F} = \hat{\boldsymbol{H}}$ where $\hat{\boldsymbol{H}}$ comes from (3.42).

The SVD and NMF techniques have been proven effective in many applications, but the rapid increase of data sizes requires more efficient algorithms. In the million-dollar Netflix Prize, a competitor needs to predict which movies a user would like to watch based on the ratings that the user has given to the system. The Netflix dataset contains more than 100 million ratings (weighted relations), thus an efficient algorithm to process all the data in a reasonable time would be more important than the accuracy. Funk propose an alternative way of computing a matrix factorization on a very big matrix that works only with the non-empty entries in the matrix [Funk, 2006], which became very popular in later years. The decomposition is similar to the NMF, i.e. $\boldsymbol{R} \approx \boldsymbol{G}\boldsymbol{H}^\top$. But there is no constraint on the values in $\boldsymbol{G}$ or $\boldsymbol{H}$. The desired parameters should minimize the weighted loss on the non-zero entries in $\boldsymbol{R}$

$$\hat{\boldsymbol{G}}, \hat{\boldsymbol{H}} = \arg\min_{\boldsymbol{G},\boldsymbol{H}} \sum_{\boldsymbol{R}(i,k)>0} \left(\boldsymbol{R}(i,k) - \boldsymbol{G}(i,\cdot)\boldsymbol{H}(k,\cdot)^\top\right)^2. \tag{3.44}$$

In [Funk, 2006] an iterative algorithm (Algorithm 4) is proposed to obtain the optimal solution. The parameter γ_r is the learning rate, which is suggested to be 0.001 in practice. The parameter γ_K is introduced into the algorithm as a regularization parameter because without the γ_K this method could suffer from over-fitting on the non-empty entries. $\gamma_K = 0.015$ is suggested to be a good choice in practice.

Algorithm 4 The Simon Funk algorithm

```
1:  procedure SimonFunk(R, l, γ_r, γ_K)
2:      Initialize G of size m × l and H of size n × l (suppose R is of size m × n)
3:      while overall error does not meet some condition do
4:          for p = 1 to l do
5:              for all (i, k) such that R(i, k) > 0 do
6:                  err = R(i, k) − G(i, ·)H(k, ·)^⊤
7:                  G(i, p) = G(i, p) + γ_r (err · H(k, p) − γ_K · G(i, p))
8:                  H(k, p) = H(k, p) + γ_r (err · G(i, p) − γ_K · H(k, p))
9:              end for
10:         end for
11:     end while
12:     return G and H
13: end procedure
```

Once we have $\boldsymbol{G}$ and $\boldsymbol{H}$, the prediction scores are simply $\boldsymbol{P} = \boldsymbol{G}\boldsymbol{H}^\top$. The whole algorithm is denoted as `PredictNZR` which calls Algorithm 4 as a subroutine. Here `NZR` means that the algorithm works only on the non-zero entries of $\boldsymbol{R}$. Notice that the `PredictNZR` is exactly the same as `PredictSVDR` except the matrix factorization subroutine.

In PredictNZR we need to keep two matrices before making the prediction, which requires a lot of storage space if $\boldsymbol{R}$ is large. Based on the work of [Funk, 2006], Paterek suggested a new model to reduce the number of parameters in storage [Paterek, 2007]. In this approach, a decomposition is computed by Algorithm 4, but only the matrix $\boldsymbol{H}$ is taken into the next step. Let $\boldsymbol{\alpha}_k$ denote the rows of $\boldsymbol{H}$. An entity $y_i \in Y$ is represented by all the incident entities in Z: $\boldsymbol{\theta}_i = \sum_{k,\boldsymbol{R}(i,k)>0} \boldsymbol{\alpha}_k$. Then a prediction score is computed with the following rule

$$\boldsymbol{P}(i,k) = a_k + \boldsymbol{\theta}_i^\top \boldsymbol{\alpha}_k, \tag{3.45}$$

where $a_k = \text{mean}(\{\boldsymbol{R}(i,k) | i = 1,2,...,m, \boldsymbol{R}(i,k) > 0\})$ is the average weight (average rating) of the entity z_k. We call this method PredictANZR where the A stands for asymmetric.

It has been shown that the asymmetric method that takes partial result from the matrix factorization would actually outperform the full SVD methods [Koren, 2008]. But no one has provided an explanation of why the asymmetric method should work better. Our proposed methods PredictNHC has the similar asymmetric style, and the quasi-orthogonal embedding structures presented in the previous section give us an explanation of the good performance of the asymmetric method. Besides the benefits of fewer parameters to store, Koren pointed out several other advantages of the asymmetric methods, including easy handling of new relations and the good explainability of predictions [Koren, 2008].

4 Hyperedge Expansion

With a vertex expansion, the set of co-occurrence relations is transformed into a set of pairwise relations between the vertices, where evaluating the goodness of the clustering is done on the induced graph. If we have a hyperedge of k vertices, by the clique expansion, a partition that separates the hyperedge into 1 and $k-1$ vertices would cut $k-1$ pairwise edges, while a cut that splits the vertices in two equal halves would have $k^2/4$ cut edges. So the clique expansion would prefer an unbalanced clustering. To mitigate the problem of unbalanced clustering, the star expansion adopts the normalizer of the hyperedge size to minimize the impact of large hyperedges. Furthermore, the NHC takes an additional normalizer of the cluster volume to balance the cluster sizes. But it can be seen that the cost of the NHC still depends on how the vertices distribute among the clusters, because a balanced separation of a hyperedge would cut more edges in the induced graph anyway.

We have explained in section 3.1 that the star expansion can be further reduced into two categories of expansions, depending on which partial eigenvector is taken in the spectrum of $\boldsymbol{L}_{H,\text{star}}$. In many applications, the exact objective for a clustering should be designed to depend on the number of hyperedges that are separated in the clustering, regardless the distribution of vertices across the partition boundary. For example, in the `zoo` data, we would like to classify the animals according to their attributes (the hyperedges) in such a way that all the animals that share the same attribute should stay in one cluster. If the animals that share the same attribute appear in different clusters, it creates a violation. And our goal, or the objective function, is to minimize the total number of violations. A violation should be counted once whether it involves ten animals or only one animal. The optimal clustering by this rule can be formulated as follows

$$\hat{C} = \arg\min_{C} \Phi_{HE0}(H, C) = \arg\min_{C} \sum_{y \in Y} w(y) \mathbb{1}\left(|\{c_j | y \cap c_j \neq \emptyset\}| > 1\right), \tag{4.1}$$

$$\hat{C} = \arg\min_{C} \Phi_{HE}(H,C) = \arg\min_{C} \sum_{y \in Y} w(y) \left(|\{c_j | y \cap c_j \neq \emptyset\}| - 1 \right). \tag{4.2}$$

The first objective $\Phi_{HE0}(H,C)$ counts the (weighted) number of violations. The second objective $\Phi_{HE}(H,C)$ further considers different degrees of the violations. If the vertices in a hyperedge appear in three clusters, the cost would be higher than the case where the vertices in a hyperedge appear in two clusters. When there are only two clusters in C, $\Phi_{HE}(H,C)$ reduces to $\Phi_{HE0}(H,C)$. The `HE` in the subscript of the objectives stands for the "hyperedge expansion", and we call the objective defined above "hyperedge cut".

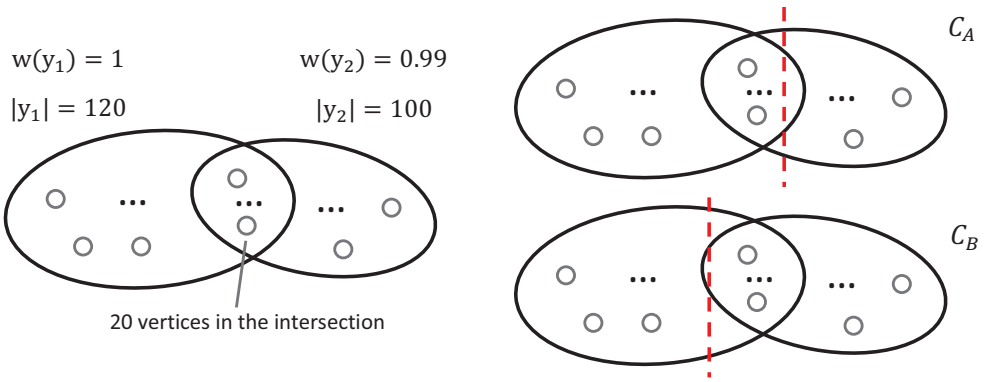

Figure 4.1: A hypergraph with two hyperedges for bi-clustering. The first hyperedge has weight 1 and 120 vertices. The second hyperedge has weight 0.99 and 100 vertices. There are 20 vertices in the intersection part. The two possible ways of partitioning the hypergraph into two clusters are shown as C_A and C_B.

The following example shows why a vertex expansion like NHC cannot fulfill the requirement in $\Phi_{HE0}(H,C)$. Suppose we have a hypergraph as in Figure 4.1, and we would like to find two clusters in this hypergraph. Obviously there are only two ways of making the clustering, either assign the intersection part to the left side or to the right side. By the objective $\Phi_{HE0}(H,C)$, the intersection part should be assigned to the left side (the C_A case in Figure 4.1), because the weight of y_2 is smaller than the weight of y_1. But if we compute the NHC cost, i.e. the normalized objective on the induced graph (see equation (2.11)), the C_B case would be preferred because the volume normalizer tries to make a balanced clustering. Even with the help of volume normalizer, a vertex expansion cannot completely eliminate the influence of the vertex distribution as long as the expansion is vertex-centric.

Many works have shown that the hyperedge-centric objective $\Phi_{HE0}(H,C)$ or $\Phi_{HE}(H,C)$ is more suitable for some applications [Singla and Domingos, 2006, Ladicky et al., 2010, Pu and Faltings, 2011]. In this chapter, we start from the objective $\Phi_{HE0}(H,C)$ and focus on the hyperedge-centric view. We show how a solution for the objective $\Phi_{HE0}(H,C)$ can be found by transforming it into a vertex separator problem. Then the vertex separator problem is studied with the spectral techniques. The algorithms for the clustering and semi-supervised

learning with hyperedge expansion are also presented in this chapter.

Our main contribution in this chapter is the spectral analysis of the minimum hyperedge cut problem. We show that the minimum hyperedge cut problem is essentially equivalent to a minimum vertex separator problem and a min-cut problem in a directed auxiliary graph (notice that the NHC expansion is based on a min-cut problem in an undirected graph, which is different from the directed auxiliary graph in this chapter). Based on the directed graph, a new Laplacian that encodes the minimum hyperedge cut is proposed. Then the new Laplacian is linked to a quadratic eigenvalue problem, and the spectrum structure of the new Laplacian is presented. We also show that (under some conditions) the minimum hyperedge cut can be lower bounded by the second smallest eigenvalue of the new Laplacian. To our best knowledge, this is the first work to analyze the spectral properties of the minimum hyperedge cut problem and link it to a quadratic eigenvalue solution.

4.1 The Hyperedge Expansion Transformation

As in Algorithm 2, our analysis takes the one-over-all scheme when there are multiple clusters in the problem, i.e. choose one cluster in each step and consider all the other clusters as one virtual cluster. Thus our analysis would focus on the bi-clustering case. For the hyperedge expansion problem, the objective $\Phi_{HE}(H,C)$ reduces to $\Phi_{HE0}(H,C)$ and we only need to cover the later.

Given a hypergraph H, consider an auxiliary graph $G_{B,H}$ constructed from H. The vertices of $G_{B,H}$ correspond to the hyperedges of H, and an edge is placed between a vertex y_i and an other vertex y_j (in graph $G_{B,H}$) if the two original hyperedges intersect in the hypergraph, i.e. $y_i \cap y_j \neq \emptyset$. The graph $G_{B,H}$ is vertex-weighted. The weight of a vertex is the same as the weight of the original hyperedge. Figure 4.2 shows an example of the auxiliary graph $G_{B,H}$. Denote the adjacency matrix of $G_{B,H}$ as

$$\boldsymbol{B} = \left[\boldsymbol{B}(i,j)\right], \text{ where } \begin{cases} \boldsymbol{B}(i,j) = 1, \text{ if } y_i \cap y_j \neq \emptyset \\ \boldsymbol{B}(i,j) = 0, \text{ otherwise} \end{cases}. \tag{4.3}$$

When we have a minimum hyperedge cut solution to (4.1), it can be mapped to a vertex separator on $G_{B,H}$, i.e. a set of vertices in $G_{B,H}$ whose removal would separate $G_{B,H}$ into at least two distinct connected components (see Figure 4.2 (a) and (b) for an example). On the other hand, if we have a vertex separator on $G_{B,H}$, it can be mapped back to a minimum hyperedge cut solution. We say that the mapping from a bi-clustering solution of (4.1) to a vertex separator on $G_{B,H}$ is valid in most cases, because it could be the case that several vertices that only appear in one hyperedge are assigned to one cluster and all the other vertices form another cluster (Figure 4.2 (c)), which cannot be represented by a vertex separator in $G_{B,H}$. But this

is a trivial case which is not very useful in practice and we normally would like to avoid it. Without loss of generality, a non-trivial optimal solution to (4.1) can be always established by finding a minimum vertex separator in $G_{B,H}$.

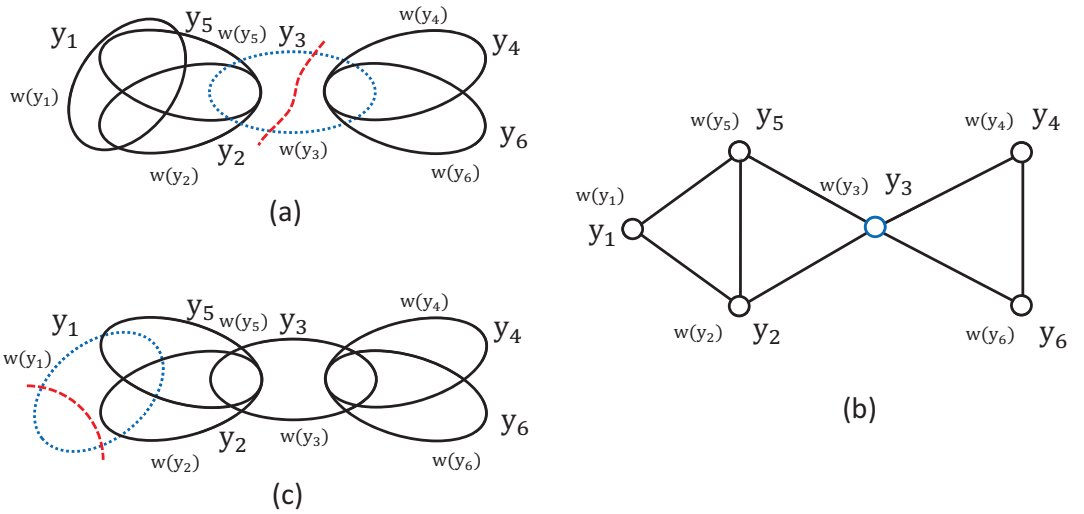

Figure 4.2: A hypergraph H (a) and its undirected auxiliary graph $G_{B,H}$ (b). The vertices in the hypergraph are not shown for simplicity. If a bi-clustering solution cuts the hyperedge y_3 into two parts in (a), it corresponds to a vertex separator of $G_{B,H}$ in (b). (c) A counterexample when a trivial bi-clustering cannot be mapped to a vertex separator.

The minimum vertex separator problem is shown to be NP-hard [Bui and Jones, 1992]. But the problem itself has a close connection with the max-flow/min-cut problem in a directed graph, which leads to various approximation algorithms, e.g. see [Leighton and Rao, 1999, Feige et al., 2008]. Lawler proposed one of the early works to find an optimal clustering in a hypergraph with the transformation to a max-flow/min-cut problem in a directed graph [Lawler, 1973]. We use a slightly different transformation as in [Acid and Campos, 1996]. In this transformation, there is an one-to-one correspondence between the minimum hyperedge cut and the min-cut in the auxiliary directed graph.

Starting from the undirected auxiliary graph $G_{B,H}$, another directed auxiliary graph $G_{\vec{B},H} = \{V_{\vec{B}}, E_{\vec{B}}, w_{\vec{B}}\}$ is constructed as follows. Firstly, a vertex in the undirected graph $G_{B,H}$ is copied to two vertices in the directed graph $G_{\vec{B},H}$, one with a + sign and another one with a − sign (notice that a vertex in $G_{B,H}$ corresponds to a hyperedge in the original hypergraph). Then a directed edge is placed between each pair of vertices in $G_{\vec{B},H}$ from the + sign to the − sign. The weight of the directed edge is the same as the weight of the original hyperedge (Figure 4.3 (b)). In the next step, for each pair of vertices y_i and y_j which are connected by an edge in $G_{B,H}$, we place two directed edges in $G_{\vec{B},H}$ from the − sign to the + sign: (y_i^-, y_j^+) and (y_j^-, y_i^+). The weight of the edge (y_i^-, y_j^+) is $w(y_j)$, and the weight of the edge (y_j^-, y_i^+) is $w(y_i)$ (Figure 4.3 (c) and (d)). Now the construction of $G_{\vec{B},H}$ completes.

If we look at a vertex in $G_{\vec{B},H}$ with a + sign, e.g. the vertex y_i^+, it has exactly one out-going edge and all the edges attached to vertex y_i^+ have the same weight. This implies that if the min-cut of $G_{\vec{B},H}$ contains an edge e from a − sign to a + sign (any in-coming edge of vertex y_i^+), one can always find another min-cut of the same cost where e is replaced by another edge e' from a + sign to a − sign (the out-going edge of vertex y_i^+). See Figure 4.3 (e) for an example. The blue edges are possible candidates of e, and the red edge is the e'. Therefore, a min-cut of $G_{\vec{B},H}$ could only contain the edges from the + sign to the − sign, which implies an one-to-one mapping from a min-cut in $G_{\vec{B},H}$ to a minimum vertex separator in $G_{B,H}$ (Figure 4.3 (f)). With the removal of the min-cut edges, the graph $G_{\vec{B},H}$ would be separated into at least two distinct strongly connected components.

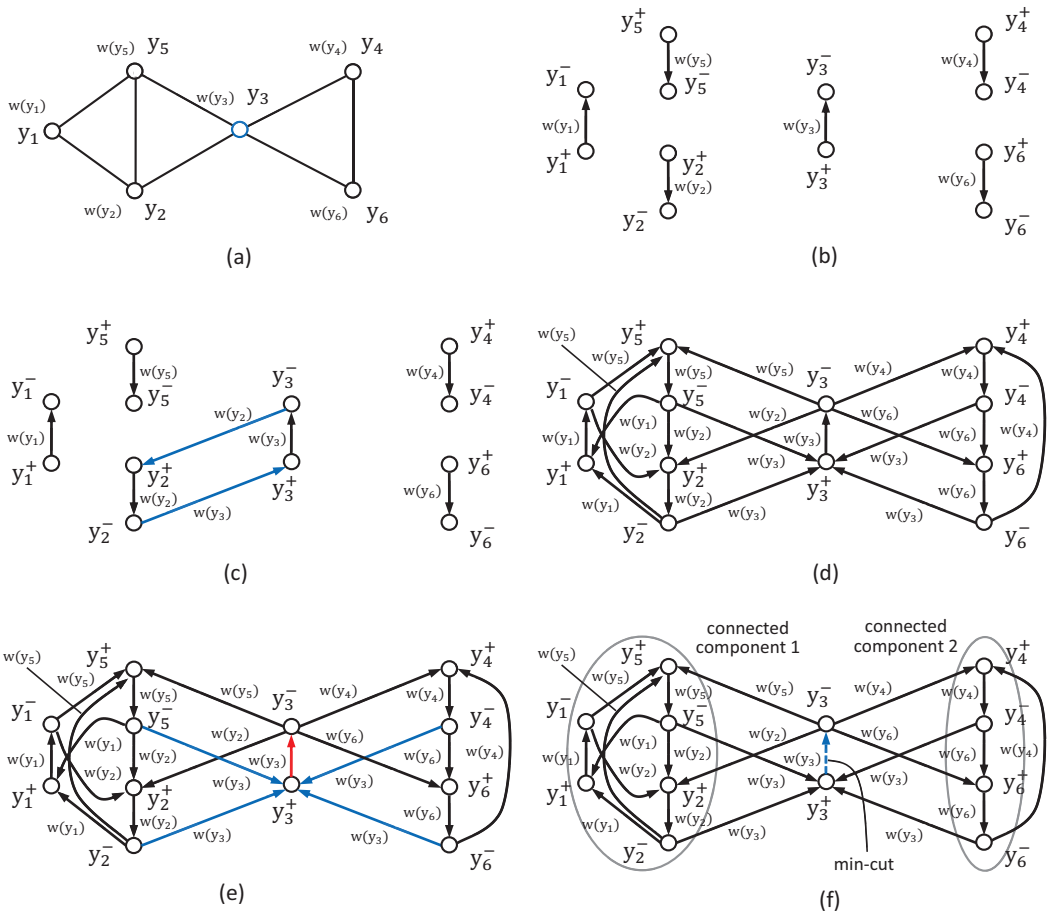

Figure 4.3: The detailed steps to construct $G_{\vec{B},H}$ from $G_{B,H}$. (a) The undirected auxiliary graph $G_{B,H}$. (b) - (f) The directed auxiliary graph $G_{\vec{B},H}$.

By linking the minimum hyperedge cut to the min-cut in $G_{\vec{B},H}$ with the minimum vertex separator in $G_{B,H}$ as an intermediator, we obtain the hyperedge expansion transformation. It is worth mentioning that the trivial solutions of the minimum hyperedge cut as in Figure

4.2 (c) are not covered by our transformation. But in a normal application, from a min-cut solution in $G_{\vec{B},H}$ it is sufficient to reconstruct a minimum hyperedge cut solution.

Let the vector $\boldsymbol{c}$ of length $2m$ denote the membership indicator of vertices in $G_{\vec{B},H}$, and S denote the set of vertices that are on one side of the min-cut in $G_{\vec{B},H}$ (consequently $\bar{S} = V_{\vec{B}} \setminus S$ is the compliment set). An entry $\boldsymbol{c}(i)$ can take values from $\{1/\sqrt{|S|}, 0\}$, which implies that $\boldsymbol{c}^\top \boldsymbol{c} = 1$ always holds. We sort the entries in $\boldsymbol{c}$ with the order corresponding to the vertices $\{y_1^-, y_2^-, ..., y_1^+, y_2^+, ...\}$ (the vertices with the $-$ sign come fist, then the vertices with the $+$ sign). The optimization problem of finding the min-cut can be formulated as

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}} \Phi_{HE0}(H, \boldsymbol{c}) \tag{4.4}$$

$$= \arg\min_{\boldsymbol{c}} \sum_{(i,j)\in E_{\vec{B}}} |S| w_{\vec{B}}(i,j)\big(\boldsymbol{c}(i) - \boldsymbol{c}(j)\big)\boldsymbol{c}(i) \tag{4.5}$$

$$\text{subject to: } \boldsymbol{c}(i) \in \{1/\sqrt{|S|}, 0\},\ \boldsymbol{c}^\top \boldsymbol{c} = 1. \tag{4.6}$$

The second $\boldsymbol{c}(i)$ in (4.5) ensures that only the edges from S to $\bar{S}$ are counted when $\boldsymbol{c}(i) = 1/\sqrt{|S|}$ and $\boldsymbol{c}(j) = 0$. It is easy to show that there is an one-to-one mapping from a solution $\hat{\boldsymbol{c}}$ of (4.4) to a min-cut of $G_{\vec{B},H}$. In the next section, we start from this optimization problem to define a hyperedge expansion Laplacian, and present several properties of the new defined Laplacian.

By the construction, the number of edges in $G_{\vec{B},H}$ is two times the number of edges in $G_{B,H}$ plus the number of hyperedges. The former can be calculated as the number of non-zero entries in $\boldsymbol{B}$. Let $\boldsymbol{M}_B = \boldsymbol{X}\boldsymbol{X}^\top$ and set the main diagonal of $\boldsymbol{M}_B$ to zeros. If the hypergraph is generated from a multi-class beta-Bernoulli process, the expected number of zeros in $\boldsymbol{B}$ is

$$\mathbb{E}\left(\sum_{i=1}^{m}\sum_{t=1}^{m} \mathbb{1}(\boldsymbol{M}_B(t,i) = 0)\right) = \sum_{i=1}^{m}\sum_{t=1}^{m} \mathbb{E}\left(\mathbb{1}(\boldsymbol{M}_B(t,i) = 0)\right) \tag{4.7}$$

$$= \sum_{i=1}^{m}\sum_{t=1,t\neq i}^{m} \mathbb{E}\left(\mathbb{1}(\boldsymbol{X}(t,\cdot)\boldsymbol{X}(i,\cdot)^\top = 0)\right) \tag{4.8}$$

$$= \sum_{i=1}^{m}\sum_{t=1,t\neq i}^{m} \mathbb{P}\left(\sum_{z_k\in c_1} \boldsymbol{X}(t,k)\boldsymbol{X}(i,k) = 0, \ ..., \ \sum_{z_k\in c_s} \boldsymbol{X}(t,k)\boldsymbol{X}(i,k) = 0\right) \tag{4.9}$$

$$= \sum_{i=1}^{m}\sum_{t=1,t\neq i}^{m}\prod_{j=1}^{s} \mathbb{P}\left(\sum_{z_k\in c_j} \boldsymbol{X}(t,k)\boldsymbol{X}(i,k) = 0\right). \tag{4.10}$$

Let $f(x, \xi_0, q_0, j, \tau)$ denote the PDF of the beta distribution with parameters $\{\xi_0, q_0, j, \tau\}$ (see (2.14)), and p_1, p_2, p denote some intermediate variables. By the similar derivation as in (3.26), we have

$$\mathbb{E}\left(\sum_{i=1}^{m}\sum_{t=1}^{m} \mathbb{1}(\boldsymbol{M}_B(t,i) = 0)\right) \tag{4.11}$$

$$= \sum_{i=1}^{m} \sum_{t=1,t\neq i}^{m} \prod_{j=1}^{s} \int_{p_1,p_2} \sum_{h=0}^{|c_j|} (-1)^h \binom{|c_j|}{h} p_1^h p_2^h f(p_1,\xi_0,q_0,j,\tau) f(p_2,\xi_0,q_0,j,\tau) dp_1 p_2 \tag{4.12}$$

$$= \sum_{i=1}^{m} \sum_{t=1,t\neq i}^{m} \prod_{j=1}^{s} \sum_{h=0}^{|c_j|} (-1)^h \binom{|c_j|}{h} \left(\int_p p^h f(p,\xi_0,q_0,j,\tau) dp \right)^2 \tag{4.13}$$

$$= m(m-1) \prod_{j=1}^{s} \sum_{h=0}^{|c_j|} (-1)^h \binom{|c_j|}{h} \left(\frac{(\xi_0 + (j-1)\tau)^{(h)}}{(\xi_0 q_0)^{(h)}} \right)^2, \tag{4.14}$$

where $x^{(h)} = x(x+1)...(x+h-1)$ is the Pochhammer symbol. This result contains a ${}_3F_2$ hypergeometric series that makes it hard to simplify. We show an example of the expected percentage of non-zeros in $\boldsymbol{B}$ in Figure 4.4.

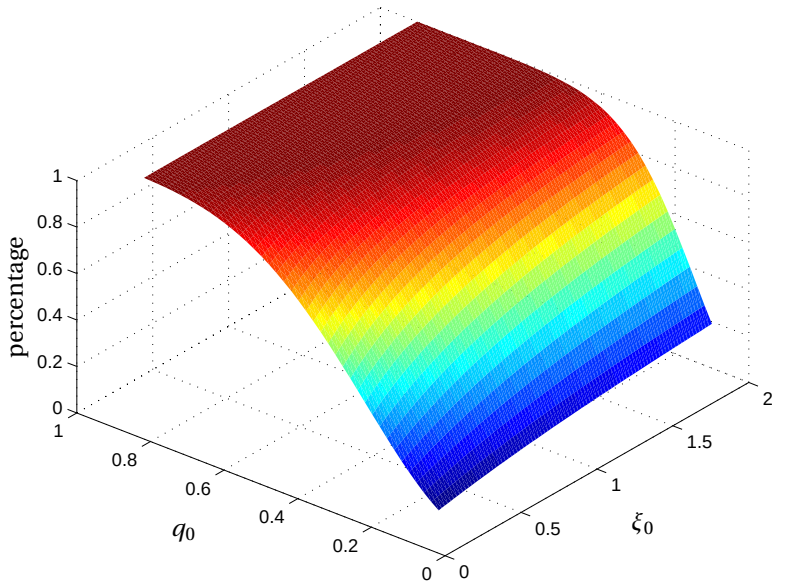

Figure 4.4: The expected percentage of non-zeros in $\boldsymbol{B}$ with respect to ξ_0 and q_0. We use three clusters in this example of sizes $\{20, 10, 8\}$, and set $\tau = 0$.

4.2 The Hyperedge Expansion Laplacian

The standard spectral graph theory starts with the min-cut problem with binary indicator variables. Then the binary variables are relaxed to take real values so that an approximation can be established between the eigen-decomposition and the min-cut. We use the same technique to relax $\boldsymbol{c}$ to take positive real values and find the relaxed $\boldsymbol{c}$ which minimizes $\Phi_{HE0}(H,\boldsymbol{c})$ (see (4.4)) by the Lagrange multiplier method with the constraint $\boldsymbol{c}^\top \boldsymbol{c} = 1$. The Lagrange func-

tion is defined as

$$\mathscr{L}(\boldsymbol{c},\lambda) = \Phi_{HE0}(H,\boldsymbol{c}) - \lambda\left(\boldsymbol{c}^\top\boldsymbol{c}-1\right), \tag{4.15}$$

where λ is a Lagrange multiplier. When taking the partial derivatives, we drop the contributions from $\boldsymbol{c}(j)$ which is close to zero. This implies the following approximation

$$\frac{\partial w_{\vec{B}}(i,j)\left(\boldsymbol{c}(i)-\boldsymbol{c}(j)\right)\boldsymbol{c}(i)}{\partial \boldsymbol{c}(i)} = 2w_{\vec{B}}(i,j)\boldsymbol{c}(i) - w_{\vec{B}}(i,j)\boldsymbol{c}(j) \tag{4.16}$$

$$\approx 2w_{\vec{B}}(i,j)\boldsymbol{c}(i), \tag{4.17}$$

$$\frac{\partial w_{\vec{B}}(i,j)\left(\boldsymbol{c}(i)-\boldsymbol{c}(j)\right)\boldsymbol{c}(i)}{\partial \boldsymbol{c}(j)} = -w_{\vec{B}}(i,j)\boldsymbol{c}(i). \tag{4.18}$$

Setting $\frac{\partial\mathscr{L}(\boldsymbol{c},\lambda)}{\partial\boldsymbol{c}(i)} = 0$, we obtain the following $2m$ equations

$$\begin{cases} w(y)deg(y)\boldsymbol{c}(y_i^-) - \sum_{y_i\cap y_j\neq\emptyset} w(y)\boldsymbol{c}(y_j^+) = \lambda\boldsymbol{c}(y_i^-) & i\in\{1,2,...,m\} \\ w(y)\left(\boldsymbol{c}(y_i^+)-\boldsymbol{c}(y_i^-)\right) = \lambda\boldsymbol{c}(y_i^+) & i\in\{1,2,...,m\} \end{cases} \tag{4.19}$$

In a matrix form, the above equations are equivalent to

$$\boldsymbol{c}^\top\boldsymbol{L}_{\mathrm{HE}} = \lambda\boldsymbol{c}^\top, \tag{4.20}$$

where

$$\boldsymbol{L}_{\mathrm{HE}} = \begin{bmatrix} \boldsymbol{D}_B & -\boldsymbol{WB} \\ -\boldsymbol{W} & \boldsymbol{W} \end{bmatrix}, \tag{4.21}$$

and $\boldsymbol{D}_B = \mathrm{diag}(\boldsymbol{1}^\top\boldsymbol{WB})$ is the weighted degree matrix of the vertices in $G_{B,H}$. The matrix $\boldsymbol{L}_{\mathrm{HE}}$ is called the *hyperedge expansion Laplacian*. And the Lagrange multiplier λ can be interpreted as an eigenvalue of $\boldsymbol{L}_{\mathrm{HE}}$ and $\boldsymbol{c}$ as a left eigenvector. Notice that the left eigenvector and the right eigenvector of $\boldsymbol{L}_{\mathrm{HE}}$ are not the same, because the graph $G_{\vec{B},H}$ is a directed graph and the induced Laplacian $\boldsymbol{L}_{\mathrm{HE}}$ is a non-Hermitian matrix.

For a non-Hermitian matrix like $\boldsymbol{L}_{\mathrm{HE}}$, the Courant–Fischer min-max theorem does not hold anymore. The field of values of the non-Hermitian matrix is a superset of the convex hull of the eigenvalues [Horn and Johnson, 1991, da Silva, 2010], and there is no guarantee that all the eigenvalues are real, even if all the entries of $\boldsymbol{L}_{\mathrm{HE}}$ are real values.

There are existing theories about the spectral properties of a directed graph based on the symmetrization of $\boldsymbol{L}_{\mathrm{HE}}$ [Wu, 2005, Chung, 2005], and the corresponding learning problems for the directed graph [Zhou et al., 2005b]. It is shown that a Cheeger inequality can be established with the first non-trivial eigenvalue of $\tilde{\boldsymbol{L}} = \frac{1}{2}\left(\vec{\boldsymbol{V}}\vec{\boldsymbol{P}} + \vec{\boldsymbol{P}}^\top\vec{\boldsymbol{V}}\right)$, where $\vec{\boldsymbol{P}}$ is the (non-Hermitian) tran-

sition probability matrix of the directed graph and $\vec{\boldsymbol{V}}$ is the diagonal matrix of the first non-trivial eigenvector of $\vec{\boldsymbol{P}}$. In the transition probability matrix $\vec{\boldsymbol{P}}$, the weights of the out-going edges are normalized by the out-degrees. For the special structure of our directed graph $G_{\vec{B},H}$, the out-degree normalization could be problematic because the correct mapping from the min-cut of $G_{\vec{B},H}$ to the original minimum hyperedge cut relies on the special weighting of the directed edges. If the edge weights are changed, it is possible that the min-cut of $G_{\vec{B},H}$ also contains edges from the − sign vertices to the + sign vertices, which is undesirable in our case. This is the reason why we use the unnormalized non-Hermitian Laplacian $\boldsymbol{L}_{\text{HE}}$ instead of the Hermitian Laplacian $\tilde{\boldsymbol{L}}$.

Although $\boldsymbol{L}_{\text{HE}}$ is a non-Hermitian matrix, we show that the special structure of the directed auxiliary graph $G_{\vec{B},H}$ leads to some special properties of $\boldsymbol{L}_{\text{HE}}$. These special properties would allow us to carry out the learning tasks with $\boldsymbol{L}_{\text{HE}}$. Denote the eigenvalues of $\boldsymbol{L}_{\text{HE}}$ as $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_{2m}$, and the left eigenvectors as $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_{2m}\}$.

Lemma 4.2.1. *For the hyperedge expansion Laplacian $\boldsymbol{L}_{HE}$, we have $\lambda_1 = 0$ and all the entries of $\boldsymbol{f}_1$ are real positive values.*

Proof. It is easy to verify that $\lambda_1 = 0$ is an eigenvalue of $\boldsymbol{L}_{\text{HE}}$ with an all-ones right eigenvector. Consider the matrix $\tilde{\boldsymbol{L}}_{\text{HE}} = \alpha \boldsymbol{I} - \boldsymbol{L}_{\text{HE}}$ where α is a real positive value. For any eigenvalue λ of $\boldsymbol{L}_{\text{HE}}$, $\alpha - \lambda$ is an eigenvalue of $\tilde{\boldsymbol{L}}_{\text{HE}}$, and vise versa. In the underlying graph $G_{\vec{B},H_T}$, we can find a directed path between any two vertices if the hypergraph is connected. Thus the graph $G_{\vec{B},H_T}$ is strongly connected, and the matrix $\tilde{\boldsymbol{L}}_{\text{HE}}$ is irreducible and has non-negative entries for some $\alpha > 0$. By the Perron-Frobenius theorem [Berman and Plemmons, 1979], there exists a left eigenvector $\boldsymbol{f}_1$ whose entries are real positive, and a real eigenvalue $\alpha - \lambda_1$ for $\tilde{\boldsymbol{L}}_{\text{HE}}$, where $\alpha - \lambda_1$ has the biggest magnitude. Thus $\boldsymbol{f}_1$ is a left eigenvector of $\boldsymbol{L}_{\text{HE}}$ to the eigenvalue λ_1. □

For each eigenvector, let $\boldsymbol{f}_k^-$ and $\boldsymbol{f}_k^+$ denote the first and the second halves of $\boldsymbol{f}_k$. The following theorem states that we could still expect real entries in the other eigenvectors.

Theorem 4.2.2. *All the eigenvalues of $\boldsymbol{L}_{HE}$ are non-negative real numbers and the left eigenvectors of $\boldsymbol{L}_{HE}$ are real if and only if there exists $\gamma \in \mathbb{R}$ such that the matrix $\boldsymbol{Q}(\gamma) = \gamma^2 \boldsymbol{W}^{-2} - \gamma \boldsymbol{W}^{-1}(\boldsymbol{I} + \boldsymbol{W}^{-1}\boldsymbol{D}_B) + (\boldsymbol{W}^{-1}\boldsymbol{D}_B - \boldsymbol{B})$ is negative definite.*

Proof. Take an eigenvalue λ of $\boldsymbol{L}_{\text{HE}}$ and a left eigenvector $\boldsymbol{f} = [\boldsymbol{f}^-, \boldsymbol{f}^+]$. The eigenvalue problem $\boldsymbol{f}^\top \boldsymbol{L} = \lambda \boldsymbol{f}^\top$ can be reformulated as

$$\boldsymbol{D}_B \boldsymbol{f}^- - \boldsymbol{W}\boldsymbol{f}^+ = \lambda \boldsymbol{f}^-, \tag{4.22}$$

$$-\boldsymbol{W}\boldsymbol{B}\boldsymbol{f}^- + \boldsymbol{W}\boldsymbol{f}^+ = \lambda \boldsymbol{f}^+. \tag{4.23}$$

By substituting $\boldsymbol{f}^+ = \boldsymbol{W}^{-1}(\boldsymbol{D}_B - \lambda \boldsymbol{I})\boldsymbol{f}^-$ in the second equation, we obtain a *quadratic eigenvalue problem* (QEP)

$$\boldsymbol{Q}(\lambda)\boldsymbol{f}^- = \left(\lambda^2 \boldsymbol{W}^{-2} - \lambda \boldsymbol{W}^{-1}(\boldsymbol{I} + \boldsymbol{W}^{-1}\boldsymbol{D}_B) + (\boldsymbol{W}^{-1}\boldsymbol{D}_B - \boldsymbol{B})\right)\boldsymbol{f}^- = 0. \tag{4.24}$$

Notice that the coefficient matrices of λ^2 and λ are positive definite (omitting the minus sign before λ), because $\boldsymbol{W}$ and $\boldsymbol{D}_B$ are diagonal matrices with positive entries in the main diagonal. It is known that the QEP in (4.24) is overdamped if and only if there exists $\gamma \in \mathbb{R}$ such that the matrix $\boldsymbol{Q}(\gamma)$ is negative definite and $(\boldsymbol{W}^{-1}\boldsymbol{D}_B - \boldsymbol{B})$ is positive semi-definite (see Theorem 2 and Definition 4 of [Guo and Lancaster, 2005]). The second condition can be always satisfied because for any vector $\boldsymbol{x} \in \mathbb{R}^m$

$$\boldsymbol{x}^\top(\boldsymbol{W}^{-1}\boldsymbol{D}_B - \boldsymbol{B})\boldsymbol{x} = \sum_{(i,j)\in E_{B,H}} \left(\sqrt{\frac{w_j}{w_i}}\boldsymbol{x}(i) - \sqrt{\frac{w_i}{w_j}}\boldsymbol{x}(j)\right)^2. \tag{4.25}$$

It is also known that the overdamped QEP $\boldsymbol{Q}(\lambda)\boldsymbol{f}^- = 0$ has $2m$ non-negative real eigenvalues and $2m$ real eigenvectors of length m. Thus $\boldsymbol{L}_{\text{HE}}$ has $2m$ non-negative real eigenvalues and $2m$ real left eigenvectors of length $2m$. □

The condition stated in Theorem 4.2.2 is hard to verify in practice. The state-of-the-art techniques usually require to actually compute all the eigenvalues of a QEP to test if it is overdamped [Higham et al., 2002]. We give a sufficient condition which is easier to verify.

Theorem 4.2.3. *All the eigenvalues of $\boldsymbol{L}_{HE}$ are non-negative real numbers and the left eigenvectors of $\boldsymbol{L}_{HE}$ are real if $d(\boldsymbol{D}_B(i,i) + \boldsymbol{W}(i,i)) > 8\boldsymbol{D}_B(i,i)\boldsymbol{W}(i,i)$ for all $i \in \{1,2,...,m\}$, where $d = \min_i(\boldsymbol{D}_B(i,i) + \boldsymbol{W}(i,i))$.*

Proof. As shown in Definition 1 of [Guo and Lancaster, 2005], the conclusion of the theorem holds if

$$\left((\boldsymbol{f}^-)^*\boldsymbol{W}^{-1}(\boldsymbol{I} + \boldsymbol{W}^{-1}\boldsymbol{D}_B)\boldsymbol{f}^-\right)^2 > 4\left((\boldsymbol{f}^-)^*\boldsymbol{W}^{-2}\boldsymbol{f}^-\right)\left((\boldsymbol{f}^-)^*(\boldsymbol{W}^{-1}\boldsymbol{D}_B - \boldsymbol{B})\boldsymbol{f}^-\right) \tag{4.26}$$

for all non-zero $\boldsymbol{f}^- \in \mathbb{C}^m$, where $(\boldsymbol{f}^-)^*$ denotes the conjugate transpose of $\boldsymbol{f}^-$. Let $\boldsymbol{g} = \boldsymbol{W}^{-1}\boldsymbol{f}^-$. Because $\boldsymbol{W}^{-1}$ is a diagonal matrix with positive main diagonal, we can transform the above condition into

$$\left(\frac{\boldsymbol{g}^*(\boldsymbol{W} + \boldsymbol{D}_B)\boldsymbol{g}}{\boldsymbol{g}^*\boldsymbol{g}}\right)^2 > \frac{\boldsymbol{g}^*\boldsymbol{W}(4\boldsymbol{W}^{-1}\boldsymbol{D}_B - 4\boldsymbol{B})\boldsymbol{W}\boldsymbol{g}}{\boldsymbol{g}^*\boldsymbol{g}} \tag{4.27}$$

for all non-zero $\boldsymbol{g} \in \mathbb{C}^m$. Both sides of the inequality contain a Rayleigh quotient. Since $(\boldsymbol{W} +$

$\boldsymbol{D}_B$) is a diagonal matrix, by the Courant–Fischer min-max theorem (Theorem 2.3.1), we have

$$\min_{\boldsymbol{g}} \frac{\boldsymbol{g}^*(\boldsymbol{W}+\boldsymbol{D}_B)\boldsymbol{g}}{\boldsymbol{g}^*\boldsymbol{g}} = \min_{i\in\{1,2,...,m\}} (\boldsymbol{D}_B(i,i)+\boldsymbol{W}(i,i)) \tag{4.28}$$

$$= d > 0. \tag{4.29}$$

Therefore another sufficient condition can be written as

$$d\frac{\boldsymbol{g}^*(\boldsymbol{W}+\boldsymbol{D}_B)\boldsymbol{g}}{\boldsymbol{g}^*\boldsymbol{g}} > \frac{\boldsymbol{g}^*\boldsymbol{W}(4\boldsymbol{W}^{-1}\boldsymbol{D}_B-4\boldsymbol{B})\boldsymbol{W}\boldsymbol{g}}{\boldsymbol{g}^*\boldsymbol{g}}, \tag{4.30}$$

or

$$\frac{\boldsymbol{g}^*(d(\boldsymbol{W}+\boldsymbol{D}_B)-4\boldsymbol{D}_B\boldsymbol{W}+4\boldsymbol{W}\boldsymbol{B}\boldsymbol{W})\boldsymbol{g}}{\boldsymbol{g}^*\boldsymbol{g}} > 0 \tag{4.31}$$

for all non-zero $\boldsymbol{g}\in\mathbb{C}^m$, which means that the Hermitian matrix $d(\boldsymbol{W}+\boldsymbol{D}_B)-4\boldsymbol{D}_B\boldsymbol{W}+4\boldsymbol{W}\boldsymbol{B}\boldsymbol{W}$ must be positive definite. We know that a Hermitian matrix is positive definite if it is strictly diagonally dominant and has all positive diagonal entries. Noticing that each row of ($\boldsymbol{W}\boldsymbol{B}\boldsymbol{W}-\boldsymbol{D}_B\boldsymbol{W}$) sums up to 0, which implies the conclusion in the theorem. □

In fact we find that many hypergraphs tested in our experiments satisfy this sufficient condition, and the first 6 eigenvalues (smallest magnitude) of almost all the hypergraphs are real non-negative. Experiments in the next chapter show that the hyperedge expansion Laplacian works well in general.

The eigen-pairs of an overdamped QEP fall into two distinct classes, the primary $\{\lambda_1 \le \lambda_2 \le ... \le \lambda_m\}$ and the secondary $\{\lambda_{m+1} \le \lambda_{m+2} \le ... \le \lambda_{2m}\}$ [Tisseur and Meerbergen, 2001]. The corresponding eigenvectors also fall into two classes, the primary $\{\boldsymbol{f}_1,...,\boldsymbol{f}_m\}$ and the secondary $\{\boldsymbol{f}_{m+1},...,\boldsymbol{f}_{2m}\}$. Unlike the spectrum of a Hermitian matrix, the eigenvectors of an overdamped QEP do not necessarily orthogonal to each other. Instead, we know that the primary eigenvectors are linearly independent, and the secondary eigenvectors are linearly independent. This means that span($\boldsymbol{f}_1,...,\boldsymbol{f}_m$) = $\mathbb{R}^m$ and span($\boldsymbol{f}_{m+1},...,\boldsymbol{f}_{2m}$) = $\mathbb{R}^m$ [Duffin, 1955]. Figure 4.5 shows an example of the primary eigenvalues and the secondary eigenvalues of the overdamped `zoo` dataset (see section 5.2.1 for a detailed description of the dataset).

It has been shown that the min-cut in a graph (the Cheeger constant) can be bounded by the second smallest eigenvalue of the Laplacian (Theorem 2.3.2). We present a similar result for the hyperedge cut. Recall that the minimum vertex separator of the undirected auxiliary graph $G_{B,H} = \{V_{B,H}, E_{B,H}\}$ can be mapped to the minimum hyperedge cut in the original hypergraph. Let $\hat{\Phi}_{HE0} = \min_C \Phi_{HE0}(H,C)$ represent the minimum hyperedge cut when dividing the hypergraph into two parts, and $w_i = \boldsymbol{W}(i,i)$ denote the weight of the vertex i in $G_{B,H}$ or

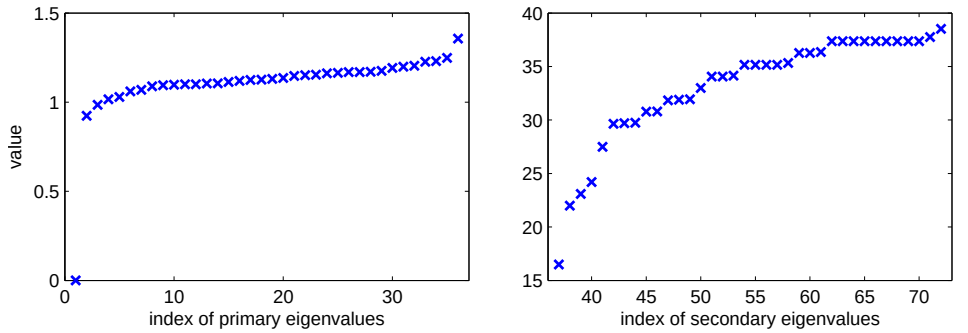

Figure 4.5: The primary eigenvalues and the secondary eigenvalues of the `zoo` dataset.

the weight of the original hyperedge y_i.

Theorem 4.2.4. *Suppose the QEP of the hyperedge expansion Laplacian $\boldsymbol{L}_{HE}$ is overdamped. Let $w_{max} = \max_{i \in V_{B,H}} w_i$ and $\varepsilon = \max_{i,j \in V_{B,H}} |w_i - w_j|$, we have*

$$\lambda_2^2 \leq w_{max} \hat{\Phi}_{HE0} + 2\varepsilon^2 |E_{B,H}|, \tag{4.32}$$

where λ_2 is the second smallest eigenvalue of $\boldsymbol{L}_{HE}$.

Proof. We use the same notations as in the proof of Theorem 4.2.2 except that replace $\boldsymbol{f}^-$ with $\boldsymbol{u}$ for simplicity. As in [Rogers, 1964], denote the roots of the quadratic $\boldsymbol{u}^\top \boldsymbol{Q}(x)\boldsymbol{u} = 0$ as $p(\boldsymbol{u})$ where $\boldsymbol{u} \in \mathbb{R}^m$. Let $a(\boldsymbol{u}) = \boldsymbol{u}^\top \boldsymbol{W}^{-2}\boldsymbol{u}$, $b(\boldsymbol{u}) = \boldsymbol{u}^\top \left(\boldsymbol{W}^{-1}(\boldsymbol{I} + \boldsymbol{W}^{-1}\boldsymbol{D}_B)\right)\boldsymbol{u}$, and $c(\boldsymbol{u}) = \boldsymbol{u}^\top \left(\boldsymbol{W}^{-1}\boldsymbol{D}_B - \boldsymbol{B}\right)\boldsymbol{u}$. By the proof of Theorem 4.2.2, we know that $a(\boldsymbol{u})$, $b(\boldsymbol{u})$ and $c(\boldsymbol{u})$ are all non-negative. The smaller root of the quadratic can be written as

$$p(\boldsymbol{u}) = \frac{b(\boldsymbol{u}) - \sqrt{b(\boldsymbol{u})^2 - 4a(\boldsymbol{u})c(\boldsymbol{u})}}{2a(\boldsymbol{u})} \tag{4.33}$$

$$\leq \frac{\sqrt{4a(\boldsymbol{u})c(\boldsymbol{u})}}{2a(\boldsymbol{u})} \tag{4.34}$$

$$= \sqrt{\frac{c(\boldsymbol{u})}{a(\boldsymbol{u})}}. \tag{4.35}$$

The inequality (4.34) comes from the fact that $b(\boldsymbol{u})^2 - 4a(\boldsymbol{u})c(\boldsymbol{u})$ is always positive since the QEP is overdamped. The term $\sqrt{c(\boldsymbol{u})/a(\boldsymbol{u})}$ is called a (generalized) Rayleigh quotient in [Duffin, 1955]. Clearly scaling the vector $\boldsymbol{u}$ does not change $p(\boldsymbol{u})$, i.e. $p(\alpha\boldsymbol{u}) = p(\boldsymbol{u})$ for any scaler α.

Recall that $\boldsymbol{D}_B = \mathrm{diag}(\boldsymbol{1}^\top \boldsymbol{W}\boldsymbol{B})$ and $\boldsymbol{B}$ is the adjacency matrix of $G_{B,H} = \{V_{B,H}, E_{B,H}\}$. The i-th

diagonal entry of $\left(\boldsymbol{W}^{-1}\boldsymbol{D}_B - \boldsymbol{B}\right)$ is $\frac{1}{w_i}\sum_{(i,j)\in E_{B,H}} w_j$. Thus (4.35) can be written as

$$p(\boldsymbol{u})^2 \leq \frac{c(\boldsymbol{u})}{a(\boldsymbol{u})} = \frac{\sum_{(i,j)\in E_{B,H}} \left(\sqrt{\frac{w_j}{w_i}}\boldsymbol{u}(i) - \sqrt{\frac{w_i}{w_j}}\boldsymbol{u}(j)\right)^2}{\sum_i \frac{\boldsymbol{u}(i)^2}{w_i^2}}. \tag{4.36}$$

Suppose the minimum vertex separator of $G_{B,H}$ is the subset of vertices $S_m \subset V_{B,H}$, and the removal of S_m would make two disjoint subsets of vertices $S_l \subset V_{B,H}$ and $S_r \subset V_{B,H}$ which are not connected by any edge in $G_{B,H}$. These three subsets are disjoint and $S_l \cup S_m \cup S_r = V_{B,H}$. If a special vector $\boldsymbol{g}$ is taken such that $\boldsymbol{g}(i) = g_l \in \mathbb{R}$ for $i \in S_l$, $\boldsymbol{g}(i) = 0$ for $i \in S_m$ and $\boldsymbol{g}(i) = g_r \in \mathbb{R}$ for $i \in S_r$, then (4.36) can be written as

$$p(\boldsymbol{g})^2 \leq \frac{\sum_{i\in S_m}\left(\sum_{\substack{j\in S_l \\ (i,j)\in E_{B,H}}} g_l^2\frac{w_i}{w_j} + \sum_{\substack{j\in S_r \\ (i,j)\in E_{B,H}}} g_r^2\frac{w_i}{w_j}\right) + \sum_{\substack{i,j\in S_l \\ (i,j)\in E_{B,H}}} g_l^2\frac{(w_i-w_j)^2}{w_i w_j} + \sum_{\substack{i,j\in S_r \\ (i,j)\in E_{B,H}}} g_r^2\frac{(w_i-w_j)^2}{w_i w_j}}{\sum_{j\in S_l}\frac{g_l^2}{w_j^2} + \sum_{j\in S_r}\frac{g_r^2}{w_j^2}} \tag{4.37}$$

$$\leq \frac{\sum_{i\in S_m}\left(\sum_{\substack{j\in S_l \\ (i,j)\in E_{B,H}}} g_l^2\frac{w_i}{w_j} + \sum_{\substack{j\in S_r \\ (i,j)\in E_{B,H}}} g_r^2\frac{w_i}{w_j}\right) + \varepsilon^2\sum_{\substack{i\in S_l \\ (i,j)\in E_{B,H}}} g_l^2\frac{deg(i)}{w_i^2} + \varepsilon^2\sum_{\substack{i\in S_r \\ (i,j)\in E_{B,H}}} g_r^2\frac{deg(i)}{w_i^2}}{\sum_{j\in S_l}\frac{g_l^2}{w_j^2} + \sum_{j\in S_r}\frac{g_r^2}{w_j^2}} \tag{4.38}$$

$$\leq w_{\max}\frac{\sum_{i\in S_m} w_i\left(\sum_{\substack{j\in S_l \\ (i,j)\in E_{B,H}}}\frac{g_l^2}{w_j} + \sum_{\substack{j\in S_r \\ (i,j)\in E_{B,H}}}\frac{g_r^2}{w_j}\right)}{\sum_{j\in S_l}\frac{g_l^2}{w_j} + \sum_{j\in S_r}\frac{g_r^2}{w_j}} + \varepsilon^2\sum_{i\in S_l\cup S_r} deg(i) \tag{4.39}$$

$$\leq w_{\max}\sum_{i\in S_m} w_i + 2\varepsilon^2|E_{B,H}| \tag{4.40}$$

$$= w_{\max}\Phi_{HE0}(H,\boldsymbol{g}) + 2\varepsilon^2|E_{B,H}|. \tag{4.41}$$

The $\Phi_{HE0}(H,\boldsymbol{g})$ on the right side is exactly the vertex separator cost given by $\boldsymbol{g}$. The inequality (4.38) is obtained by assigning an edge to the vertex with smaller weight in S_l or S_r, then each vertex is processed iteratively. The $deg(i)$ is the unweighted degree of the vertex i. Then (4.39) and (4.40) are obtained by omitting some vertices of S_l and S_r in the denominator. There might be a tricky situation where a vertex $i \in S_m$ is not directly connected to any vertex in $S_l \cup S_r$. This could happen when a vertex is connected by only one edge to a vertex in the minimum vertex separator. We can work around it by assigning such vertex to either S_l or S_r.

It is shown that there exists a $(m-1)$-dimensional subspace U such that $\lambda_2 = \min_{\boldsymbol{u}\in U} p(\boldsymbol{u})$ for an overdamped QEP ([Duffin, 1955], Lemma 6). The special vector $\boldsymbol{g}$ could take values

from a 2-dimensional subspace G spanned by the two indicator vectors of S_l and S_r. Thus the intersection of U and G is always non-trivial (non-zero) in $\mathbb{R}^m$. Let $\hat{\boldsymbol{g}}$ be a vector in the intersection. We have $\lambda_2 \leq p(\hat{\boldsymbol{g}})$. Combing it with (4.41), we obtain the conclusion of the theorem. □

The above eigenvectors from the hyperedge expansion Laplacian are the embeddings of the hyperedges, but the required clusters or classifications in our learning tasks are about the vertices. In order to utilize the hyperedge expansion, we need to find a mapping from the hyperedge embeddings to the vertex embeddings. The idea is to use the incident matrix of the hypergraph to find the mapping, i.e. the embedding of a vertex is the linear combination of the embeddings of the hyperedges that are incident with the vertex. If the first l eigenvectors of $\boldsymbol{L}_{\text{HE}}$ are used, the vertex embeddings can be constructed by

$$\boldsymbol{\alpha}_z = \left[\boldsymbol{f}_1^-, \boldsymbol{f}_2^-, ..., \boldsymbol{f}_l^-\right]^\top \boldsymbol{X}(\cdot, z), \ \forall \ z \in Z. \tag{4.42}$$

Notice that the first halves of the $\boldsymbol{f}_k$'s are used in (4.42). The reason is that we use the vertices with the + sign in $G_{\vec{B},H}$ to enforce additional constraints in semi-supervised learning. With the constraints applied to the vertices with the + sign, it makes more sense to construct the embeddings from the vertices with the − sign.

4.3 Clustering and Semi-supervised Learning

The clustering algorithm with the hyperedge expansion is similar to those with the vertex expansions. Besides the different Laplacian for computing the eigenvectors, we also need to project the hyperedge embeddings back to the vertex embeddings. We denote the hyperedge expansion clustering algorithm as `HCluster[HE]`.

Algorithm 5 The hyperedge expansion clustering algorithm

1: **procedure** `HCLUSTER[HE]` ($H = \{Z, Y, w\}$, l, s)
2: Compute the $|Y|\times|Z|$ incident matrix $\boldsymbol{X}$ and the $2|Y|\times 2|Y|$ hyperedge expansion Laplacian $\boldsymbol{L}_{\text{HE}}$ from H, see (4.3) and (4.21)
3: Compute the left eigenvectors corresponding to the smallest l eigenvalues of $\boldsymbol{L}_{\text{HE}}$, and place the eigenvectors in the $2|Y| \times l$ matrix $\boldsymbol{F}_y$
4: Take the upper half matrix $\boldsymbol{F}_y^-$ of $\boldsymbol{F}_y$
5: Let $\boldsymbol{F}_z = \boldsymbol{X}^\top \boldsymbol{F}_y^-$
6: **return** a hard clustering $\hat{\boldsymbol{C}}$ from k-means or kNN that takes s (the desired number of clusters or the number of neighbors) and the rows of $\boldsymbol{F}_z$ as inputs
7: **end procedure**

Unlike the vertex expansion case, we do not have a theory with the hyperedge expansion to support a specific type of distance in the k-means or k-NN algorithm. But in practice the co-

sine distance works well in most applications. We can also use cross-validation for choosing a good parameter l. Algorithm 5 involves in the computation of the leading l eigenvectors of a matrix of size $2|Y| \times 2|Y|$. In the iterative eigen-decomposition routines, the running time per iteration is proportional to the non-zeros in the adjacency matrix $\boldsymbol{B}$ of graph $G_{B,H}$, which is estimated in (4.14). The total number of iterations depends on the eigen-gap of graph $G_{B,H}$ and can be variant for different datasets.

In the semi-supervised learning, some vertices are already classified into some clusters, but our auxiliary graphs for the hyperedge expansion are constructed with the hyperedges. Therefore, we need to first convert the given partial clustering to the hyperedge setting, which is done by adding some auxiliary hyperedges into the original hypergraph.

Suppose that the known partial clustering is represented by the partial indicator matrix $\boldsymbol{T}$ of size $|Z| \times s$ where s is the given number of clusters. An entry $\boldsymbol{T}(k, j) = 1$ if the vertex z_k is known to be in the cluster c_j, otherwise $\boldsymbol{T}(k, j) = 0$. For each cluster c_j, a *label hyperedge* is created by including all the vertices in the j-th column of $\boldsymbol{T}$. In other words, a new hyperedge $y_{c_j} = \{z_k | \boldsymbol{T}(k, j) = 1\}$ is added to the original hypergraph. We assume that at least one vertex from each cluster is labeled in $\boldsymbol{T}$, so the label hyperedge can be always constructed. The weights of all the label hyperedges are set to a pre-defined value w_T, i.e. $w(y_{c_1}) = w(y_{c_2}) = ... = w_T$. Let H_T denote the extended hypergraph with label hyperedges. Then a directed auxiliary graph $G_{\vec{B},H_T}$ is constructed from H_T by the steps described in section 4.1.

Our learning algorithm is in the one-over-all style. In each step, one cluster is selected (suppose it is c_p), and a modified hyperedge expansion Laplacian is defined as

$$\boldsymbol{L}_{\mathrm{HET},c_p} = \boldsymbol{L}_{\mathrm{HET}} - \mu \boldsymbol{M}_{c_p} \tag{4.43}$$

where $\boldsymbol{L}_{\mathrm{HET}}$ is the hyperedge Laplacian of the extended hypergraph H_T, μ is a parameter and

$$\boldsymbol{M}_{c_p} = \begin{bmatrix} \ddots & & \\ & w(y_{c_p}) & \\ & & \ddots \end{bmatrix}. \tag{4.44}$$

The matrix $\boldsymbol{M}_{c_p}$ has only one non-zero entry $w(y_{c_p})$ in the bottom-right half diagonal corresponding to the position of the vertex $y^+_{c_p}$ in the directed graph $G_{\vec{B},H_T}$. The modification of the Laplacian can be seen as an additional constraint to (4.5) on the vertex $y^+_{c_p}$ such that the entry $\boldsymbol{c}(y^+_{c_p})$ is forced to take the positive value $1/\sqrt{|S|}$. The degree of the constraint is controlled by the parameter μ. A larger μ would have a bigger influence on guiding the direction of the hyperedge partition around $y^+_{c_p}$, while a smaller μ would let the partition follow the intrinsic principle direction of $G_{\vec{B},H_T}$. It is worth mentioning that $\boldsymbol{L}_{\mathrm{HET},c_p}$ is defined for a

specific cluster c_p. Thus for each cluster the matrix $\boldsymbol{L}_{\text{HET},c_p}$ has to be recomputed.

With the Hermitian Laplacian for graphs, the eigenvector associated with the second smallest eigenvalue, namely the Fiedler vector, is often taken to make a bi-clustering. In our case, if the modified Laplacian $\boldsymbol{L}_{\text{HET},c_p}$ is computed with a positive parameter $\mu > 0$, the eigenvector corresponding to the smallest real eigenvalue is more useful.

Theorem 4.3.1. *If $\mu > 0$, there smallest eigenvalue λ_{c_p} of $\boldsymbol{L}_{HET,c_p}$ is real and has the smallest real part among all the eigenvalues of $\boldsymbol{L}_{HET,c_p}$. The left eigenvector $\boldsymbol{f}_{c_p}$ corresponding to λ_{c_p} is also real and has all positive entries. Furthermore we have $\lambda_{c_p} \geq -\mu w(y_{c_p})$.*

Proof. The proof follows the same steps as in Lemma 4.2.1. The bound of λ_{c_p} directly comes from the spectral radius bound of the Perron-Frobenius theorem. □

Similar to the semi-supervised learning algorithm with vertex expansions, the values in $\boldsymbol{f}_{c_p}$ are treated as scores for the final assignment of an unlabeled vertex. But again we need to first convert the scores on the hyperedges to the scores on the vertices. The score of a vertex z to a cluster c_j is

$$\text{score}(z, c_j) = \sum_{z \in y} \boldsymbol{f}_{c_j}(y^-). \tag{4.45}$$

Since the additional constraint is applied to a + sign vertex in the modified Laplacian, only the entries with the − sign are taken into consideration in the above formula. Putting everything together, we obtain the semi-supervised learning algorithm with hyperedge expansion (denoted as `HSS[HE]`).

Algorithm 6 The hyperedge expansion semi-supervised learning algorithm

1: **procedure** `HSS[HE]` ($H = \{Z, Y, w\}$, $\boldsymbol{T}$, w_T, μ)
2: Initialize a $|Z| \times s$ matrix $\hat{\boldsymbol{C}}$ and a $|Z| \times s$ matrix $\hat{\boldsymbol{F}}$, where s is the number of clusters specified in $\boldsymbol{T}$
3: Construct an extended hypergraph H_T from the input hypergraph H, the partial clustering $\boldsymbol{T}$ and the label hyperedge weight w_T
4: **for all** cluster c_p in partial clustering $\boldsymbol{T}$ **do**
5: Compute the modified hyperedge expansion Laplacian $\boldsymbol{L}_{\text{HET},c_p}$ from H_T, see (4.43)
6: Compute the left eigenvector $\boldsymbol{f}_{c_p}$ corresponding to the smallest real eigenvalue of $\boldsymbol{L}_{\text{HET},c_p}$
7: Fill the entry $\hat{\boldsymbol{F}}(k, p)$ with the score(z_k, c_p) defined in (4.45)
8: **end for**
9: For each vertex z_k which is not labeled in $\boldsymbol{T}$, let $\hat{\boldsymbol{C}}(k, j) = 1$ where $j = \arg\max_p \hat{\boldsymbol{F}}(k, p)$, and set all the other entries of the k-th row of $\hat{\boldsymbol{C}}$ to 0
10: **return** a hard clustering that combines $\hat{\boldsymbol{C}}$ and $\boldsymbol{T}$
11: **end procedure**

Algorithm 6 involves in the computation of the leading eigenvector of a matrix of size $2N \times 2N$ ($N = |Y| + s$), and this procedure has to be repeated s times. Since we are only interested in the eigenvector of the smallest real eigenvalue, it can be solved by the power iteration methods. The basic operation in each iteration is a matrix-vector multiplication where the matrix is essentially $\boldsymbol{L}_{\mathrm{HET},c_p}$. If M is the number of non-zero entries in $\boldsymbol{L}_{\mathrm{HET},c_p}$, the time complexity of each iteration is $O(M)$. The convergence rate depends on the eigen-gap of $\boldsymbol{L}_{\mathrm{HET},c_p}$ and the initial value of the iterations. In fact, the lower bound in Theorem 4.3.1 can be used as a good initial guess of the eigenvalue.

4.3.1 Related Works

The clustering and semi-supervised learning algorithms proposed in this chapter and in the previous chapter take relational data (hypergraph) as input and output a hard clustering of vertices. For the purpose of comparison, we list some other existing approaches for clustering or semi-supervised learning on relational data.

The hMETIS toolkit is a commonly used tool for hypergraph partitioning [Karypis et al., 1997]. It tries to optimize the $\Phi_{HE}(H, C)$ objective with a heuristic algorithm. Although hMETIS is mainly designed for VLSI applications, reports show that this toolkit can be applied to general classification/clustering problems [Strehl and Ghosh, 2003]. There is a "pre-assignment of vertices" function in hMETIS that can be used for assigning some vertices to known clusters in the semi-supervised learning task.

The *rendezvous algorithm* (denoted as `Rend`) is a semi-supervised learning approach based on a random walk on a graph [Azran, 2007]. The algorithm first constructs a directed graph from the kNN algorithm where all the labeled vertices have only in-coming edges and thus act as absorbing states of the random walk. Then the algorithm simulates a set of particles that start a random walk from each unlabeled vertex and stop at some labeled vertices. Intuitively, a particle from an unlabeled vertex will stop at a labeled vertex of its true label with higher probability. Thus the algorithm determines the labels based on the outcome of the random walk. Since our experiments are carried out on the hypergraphs, we take the induced graph of the hypergraph as the input of the rendezvous algorithm. A Gaussian kernel function is also adopted when constructing the kNN graph as suggested by the author.

The *semi-supervised kernel k-means* (denoted as `SSKKmeans`) is an extension of the kernel k-means method where the kernel function is a linear combination of the graph kernel and the label-induced modifier [Kulis et al., 2005]. The label-induced component includes both same-class rewards and different-classes penalties. Again we use the induced graph from the hypergraph to compute the graph kernel. For the kernel k-means step, the result of 1-nearest labeled neighbor algorithm (with cosine distance) are used to initialize the partition in our

implementation, which is far better than a random initialization in practice.

The *AnchorGraph* algorithm focuses on the scalability of semi-supervised learning [Liu et al., 2010]. Instead of constructing a kNN graph from the original data, AnchorGraph chooses a small set of anchors which connect to the s-nearest neighbors in the original data, and represents each data point with a linear combination of these anchors. The semi-supervised algorithm is faster because the values to learn are only the weights of the anchors rather than the labels of the original data, and the number of anchors is far less than the number of data points. The AnchorGraph algorithm is designed as a feature-based approach. To adapt it into the hypergraph scheme, we take the incident vector of each vertex as the feature vector for the AnchorGraph algorithm. And the cluster centers from a k-means algorithm is pre-computed as good anchors, as instructed by the authors.

5 Applications

In this chapter, we present the applications of the hypergraph learning tasks: clustering, semi-supervised learning, and link prediction. Each learning task is associated with one specific application. The section of *network traffic inspection* presents a clustering tool to help the network administrators to monitor the network activities. The section of *semi-supervised learning* shows a comparison between the hypergraph algorithms and some other semi-supervised learning methods on benchmark datasets. Finally in the section of *recommender system* we demonstrate the link prediction algorithm in the recommender systems with book and music datasets.

In each section, we first introduce the motivation and background of the application. Then our algorithms are adapted to serve the goal of the application. After presenting the datasets, we show the comparison between our approaches and state-of-the-art methods.

5.1 Network Traffic Inspection

5.1.1 Introduction

Networked computing environments are subject to configuration errors, undesired activities and attacks by malicious software. These can be detected by monitoring network traffic, but the network administrators are usually overwhelmed by the amount of data that needs to be inspected. Filtering techniques are widely used by the administrators to pick out the unusual traffic [Chandola et al., 2009]. But this approach requires the pre-defined filters, so it might miss some unknown anomalies. To deal with this problem, more intelligent techniques need to be adopted. We aim to develop a clustering strategy based on the hypergraph model for the administrator so that the network traffic can be efficiently inspected with a concise list.

The real-time enterprise desktop monitoring software developed by Nexthink [1] continuously monitors the network traffic and application activities (processes) on all the desktops in an enterprise network (Figure 5.1). Unlike the traditional system logs, the monitoring software called Nexthink Collector is installed on the client side rather than the server side. This feature allows the complete records of TCP/UDP connections, which includes not only the common 5-tuple (source IP, destination IP, source port, destination port, and protocol) but also the information of the application that has initialized the connection (e.g. application name, application version, and the user ID who is using the application). We study two problems with the Nexthink dataset, namely the network service identification and the user affiliation identification. Each problem is modeled with a hypergraph as a clustering task.

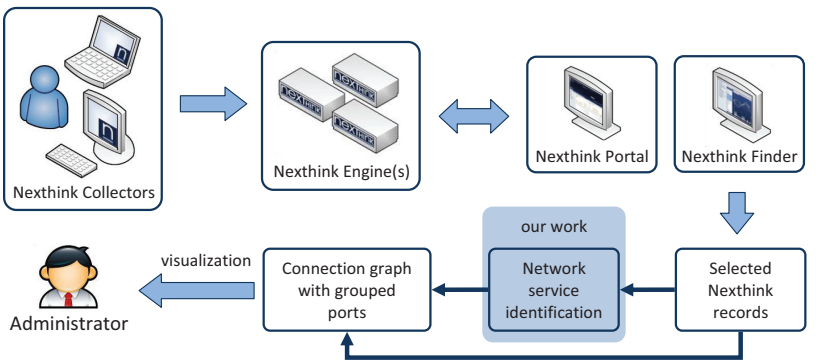

Figure 5.1: The architecture of Nexthink tools.

With the Nexthink Finder tool, an administrator can select the part of records of particular interest (by smart filters) and generate a connection graph for inspection. In this tool, connections are grouped by the port number range (e.g. TCP1-TCP1000, TCP1001-TCP2000, etc.), which is obviously not a good grouping strategy because in each port number range there could be ports of different functionalities. In the *network service identification problem*, we aim to provide a better grouping strategy of the ports based on the network service, i.e. a set of ports that serve for the same functionality. The functionalities can be the email service, the printing service, the anti-virus service, etc.

The similar idea is applied to identify the affiliation of a user. In a corporate network, users from the same department tend to use the same set of applications to access similar destinations, since they do similar works. The affiliation information can be directly obtained from the profile of the user, but there might exist latent "affiliations" that reflect some unknown work patterns. In the *user affiliation identification problem*, we try to identify some user groups by the user behaviors which are particularly interesting from the perspective of network management. The identified user groups can be used to improve the work flows or the security rules.

[1] http://www.nexthink.com

5.1.2 Constructing the Hypergraph

Five types of entities are recorded with the Nexthink dataset: the *connections* represent the TCP/UDP sessions; the *applications* are the processes on the client machines with the same executable filename; the *destinations* are the servers of an unique IP address which open some ports to the applications; the *ports* are the combinations of port types (TCP or UDP) and port numbers; and the *users* are the unique IDs which login to the client machines and access the destinations through the applications. Each record is simply a tuple <connection, application, destination, port, user>, and we have a list of records in the database collected from a period of time. In fact, the connection field is only used as an identifier of the records, which is not taken into the clustering algorithm. The key of using the hypergraph for relational clustering is to construct the co-occurrence relations (or the hyperedges) from the raw records.

In the network service identification problem, the vertices of the hypergraph are the ports since we would like to find clusters of ports. Each cluster is called a *network service*. Consider the set of applications and destinations that are associated with a network service, e.g. the email service. There might be different applications acting as email client, and there might exist multiple email servers (destinations) in the corporate network. But normally these applications and destinations should follow the common protocols for the email service, which would specify some special ports to be reserved only for the emails. These specified ports are not unique. For example, the ports TCP25, TCP110, TCP465 and TCP995 can all be used in one application for the email service, or opened in a server. Ideally a network service (or a cluster) should include all these ports. The co-occurrence relation that we extract from raw records is the application-destination pair, because all the ports associated with the same application-destination pair should serve for similar functionalities. For example, there are 4 records with the same application "outlook.exe" and the same destination "mail.epfl.ch", and the ports in those 4 records are TCP25, TCP110, TCP465 and TCP995. Then we could create a co-occurrence to include the 4 ports. The underlying assumption is that "outlook.exe" connects to "mail.epfl.ch" for a very specific reason – exchanging emails. Although in reality we do not know the reason explicitly, but we assume that a latent reason does exist.

In some cases, a malware might mimic the behavior of a normal software such that the malicious ports are identified as normal. But in order to hide itself, the malware has to know the ports of other services. We offset the impact of this adversarial behavior by excluding the well known ports to make it harder to hide the malicious ports within a known network service. Therefore the records with the common ports, e.g. TCP80, are filtered out in the pre-processing.

It can be observed in the records that the ports are not always used for the registered purpose

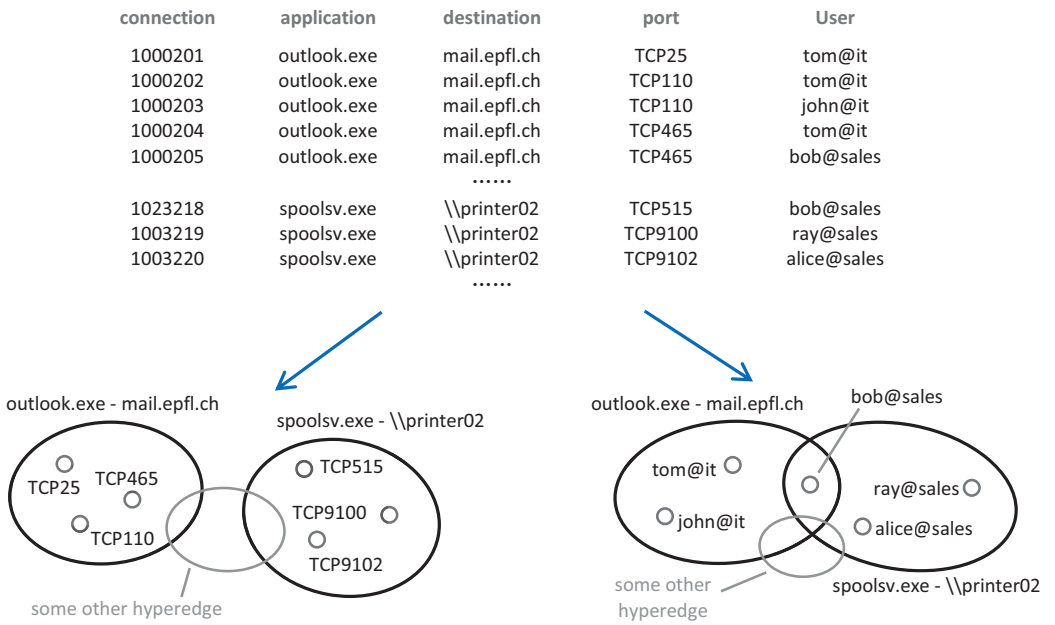

Figure 5.2: Constructing a hypergraph from the raw records of Nexthink dataset. The bottom left hypergraph is for the network service identification, and the bottom right hypergraph is for the user affiliation identification.

as shown in the IANA (Internet Assigned Numbers Authority) list, but we find that the functionality of one port is quite stable within the enterprise network. Thus it is reasonable to assume that one port belongs to exactly one network service (cluster), which is also the basic assumption of the clustering task defined in section 2.2.1.

Apparently the above two assumptions do not always hold, thus the constructed hypergraph might be noisy. We leave the hypergraph to the clustering algorithm to filter out the noises and extract the clusters. Notice that in network service identification the connection and user information is discarded from the raw records. We denote this hypergraph as H_{port}.

In the user affiliation identification problem, we adopt the same idea to construct the hyperedges, but replace the ports with the users. The destinations used in this problem are not restricted in the corporate network. Any IP address that a user has connected to is included in the raw records. We also reduce the number of destinations by replacing the IP addresses with its corresponding net-names and/or domain names. For example, the IP address "173.252.110.27" is converted to "facebook.com". This pre-processing procedure usually reduces the raw IP addresses to less than 10% destinations. We denote this hypergraph as H_{user}. Figure 5.2 shows an example of the hypergraph H_{port} and H_{user}.

The first hypergraph H_{port} consists of hyperedges that are generated by the softwares. Since the behavior of the software is rather deterministic, one could observe that H_{port} exhibits a

relation distribution that is very close to the $\xi_0 \to 0$ case in the multi-class beta-Bernoulli process. On the other hand, because the hyperedges in H_{user} are generated by the user activities, the relation distribution of H_{user} is less deterministic, and conveys more randomness.

5.1.3 Experiments

We take datasets collected from different corporate networks for the clustering tasks. Since there is no ground-truth about the clustering, an internal metric that only depends on the hypergraph and the clustering result needs to be measured for evaluating the goodness of the result.

The modularity is the most commonly used metric for community detection [Newman and Girvan, 2004, Newman, 2006, Brandes et al., 2007]. It is defined on a graph and measures the actual edge weights in a cluster minus the expected edge weights in the same cluster as if the edges are randomly placed between the vertices. Given an undirected graph $G = \{V, E, w\}$ and a vertex clustering $C = \{c_1, c_2, ..., c_s\}$, define $E(c_j) = \{e | e = (v, u) \in E, v \in c_j, u \in c_j\}$ as the set of edges that connect vertices within cluster c_j, and $E(c_i, c_j) = \{e | e = (v, u) \in E, v \in c_i, u \in c_j\}$ as the set of edges that connect vertices between cluster c_i and c_j. The modularity is defined as [Brandes et al., 2007]

$$Q(G, C) = \sum_{j=1}^{|C|} \frac{w\left(E(c_j)\right)}{w\left(E\right)} - \left(\frac{w\left(E(c_j)\right) + \sum_{i=1, i\neq j}^{|C|} w\left(E(c_i, c_j)\right)}{2w\left(E\right)} \right)^2, \tag{5.1}$$

where $w\left(E(\cdot)\right)$ denote the weight sum of the edges in the edge set $E(\cdot)$. The $Q(G, C)$ value can range from -1 to 1, and a larger value suggests a better clustering or community structure. The modularity measures the edges between distinct clusters like the min-cut, and also considers the average density of edges within a cluster, which penalizes the trivial partitions that contain single-vertex clusters. Notice that the modularity can be also used to compare the partitions with different numbers of clusters.

Because the modularity is defined on a graph, we take the induced graph G_H (section 2.1) of the hypergraph H to compute the value $Q(G_H, C)$. Some other metrics like the hyperedge cut $\Phi_{HE0}(H, C)$, $\Phi_{HE}(H, C)$ (see (4.1)), and the normalized cut on the induced graph $\Phi_N(G_H, C)$ (see (2.11)) are also calculated.

We run experiments on Algorithm 1 with the NHC Laplacian, Algorithm 5 with the hyperedge expansion Laplacian, the hMETIS toolkit (see section 4.3.1), and the agglomerative hierarchical clustering algorithm. Algorithm 1 and Algorithm 5 are tested with both k-means and kNN subroutines. We apply a maximum cluster number $k_{\max}$ to the k-means algorithm, but the final result could include less number of clusters since some clusters might become empty

over the iterations of the k-means algorithm. The kNN algorithm takes the parameter k_N as input, constructs an auxiliary graph with k_N-nearest neighbor search, and outputs the distinct (weakly) connected components in the auxiliary graph as clusters. As a baseline, we also use the columns of the incident matrix $\boldsymbol{X}$ as (binary) feature vectors and take them into the k-mean/kNN algorithm. The single-linkage algorithm, one of the agglomerative hierarchical clustering algorithms, is reported to be the best clustering approach for the network traffic or other signatures generated by the softwares [Bayer et al., 2009, Perdisci et al., 2010]. Thus we also include it in the experiments. The distance function used in the experiments is the cosine distance, and the dimensionality of the embeddings is set to $l = 10$.

In the network service identification problem, two datasets collected in a period of 10 days are used. The first dataset `Nexthink92` contains 6.1 million records, 426 applications, and 5975 destinations. After pre-processing, there are 10061 ports and 4324 application-destination pairs (hyperedges) in the constructed hypergraph. The second dataset `Nexthink93` contains 5.4 million records, 184 applications, and 2089 destinations. After pre-processing, there are 6291 ports and 2099 application-destination pairs (hyperedges) in the constructed hypergraph.

	$\Phi_{HE0}(H,C)$	$\Phi_{HE}(H,C)$	$\Phi_N(G_H,C)$	$Q(G_H,C)$	#clusters
`SingleLinkage` ($k_{\max}=20$)	20	20	1.042	0.020	20
`hMETIS` ($k_{\max}=20$)	1342	1394	4.459	0.468	20
k-means ($k_{\max}=20$)	2433	2550	1.518	0.534	12
`HCluster[NHC]` ($k_{\max}=20$)	270	378	2.712	0.641	19
`HCluster[HE]` ($k_{\max}=20$)	332	433	1.374	0.655	13
kNN ($k_N=8$)	7	7	0.210	0.009	7
`HCluster[NHC]` ($k_N=4$)	1928	2397	14.339	0.519	73
`HCluster[NHC]` ($k_N=6$)	1899	2209	7.898	0.533	46
`HCluster[NHC]` ($k_N=8$)	1517	1598	5.989	0.546	37
`HCluster[NHC]` ($k_N=10$)	1335	1365	2.782	0.384	25
`HCluster[HE]` ($k_N=4$)	3958	7228	11.201	0.301	61
`HCluster[HE]` ($k_N=6$)	326	386	5.392	0.529	33
`HCluster[HE]` ($k_N=8$)	2909	3002	5.039	0.294	33
`HCluster[HE]` ($k_N=10$)	3292	3705	3.463	0.316	23

Table 5.1: Results of the clustering algorithms on dataset `Nexthink92`.

Table 5.1 and 5.2 show the results of the experiments. The modularity $Q(G_H,C)$ values indicate that our proposed approaches outperform the baselines and the other methods. In fact, we are able to identify some clusters that can be easily mapped to a network service. In Table 5.3, some detected network services are listed. There are two main types of network services: ports range and ports function. Ports range is a service that contains some consecutive or random ports uniquely used by a specific application. Service 1 is an example of ports range.

	$\Phi_{HE0}(H,C)$	$\Phi_{HE}(H,C)$	$\Phi_N(G_H,C)$	$Q(G_H,C)$	#clusters
`SingleLinkage` ($k_{\max}=20$)	19	19	4.145	0.008	20
`hMETIS` ($k_{\max}=20$)	137	181	5.807	0.415	20
k-means ($k_{\max}=20$)	598	658	0.551	0.693	10
`HCluster[NHC]` ($k_{\max}=20$)	511	534	1.614	0.710	19
`HCluster[HE]` ($k_{\max}=20$)	154	157	0.119	0.749	6
kNN ($k_N=8$)	12	15	1.474	0.088	13
`HCluster[NHC]` ($k_N=4$)	282	386	9.092	0.760	49
`HCluster[NHC]` ($k_N=6$)	265	329	6.794	0.764	38
`HCluster[NHC]` ($k_N=8$)	170	198	4.367	0.671	27
`HCluster[NHC]` ($k_N=10$)	92	106	2.194	0.685	18
`HCluster[HE]` ($k_N=4$)	239	286	6.728	0.748	33
`HCluster[HE]` ($k_N=6$)	206	241	3.810	0.752	23
`HCluster[HE]` ($k_N=8$)	148	169	2.720	0.728	16
`HCluster[HE]` ($k_N=10$)	119	137	1.909	0.735	14

Table 5.2: Results of the clustering algorithms on dataset `Nexthink93`.

It contains all the ports used by the process that supports the "Simple Network Management Protocol". Service 2 to 5 are examples of ports functions. Service 2 contains the two common file-sharing ports in Windows system. It provides the list of all file-sharing servers without any prior knowledge about the functionalities of ports in Windows system. The applications in service 3 actually belong to the same software package "Novadigm Radia software delivery and management tools". With this information, we can also find out which servers are hosting the Radia service, which could help the administrator to manage the software. Applications in service 5 are the products of "Symantec AntiVirus Suite". These clusters could largely reduce the amount of work for the administrators to sift through the dataset.

Figure 5.3 shows another concrete example of network services. It draws the incident matrix of the hypergraph with reordered rows and columns by the learned clusters. From this view one can easily identify the network services and the associated applications/servers.

For the user affiliation identification problem, we use a dataset collected by Nexthink from the computer rooms where the users (students) from different departments could access local and Internet contents. The affiliation of users (the ground truth) is directly taken from the online user profile and used as labels. In the label list, there are 12 affiliations that correspond to 12 combinations of department and class year, e.g. CS2010, EE2011, etc. The smallest and largest affiliations contain 7 users and 45 users respectively. Because the data from the small affiliations is too sparse to make any reasonable clustering, we only keep the users from the 4 largest affiliations in the final hypergraph, which contains 146 users and 452 hyperedges. The dataset was collected from a period of two weeks.

service		Ports, applications, destinations
1	port:	UDP50758, UDP50760, UDP50767, UDP50769, UDP50770, UDP50771, ...
	app:	snmp.exe
	dest:	10.0.0.20
2	port:	TCP139, TCP445
	app:	system
	dest:	10.130.10.111,10.130.10.107,10.130.10.226,10.130.10.98,10.130.10.222, ...
3	port:	TCP3464, TCP3466
	app:	nvdkit.exe, radconct.exe, radstgms.exe, radstgrq.exe
	dest:	10.130.10.94,10.144.0.5,10.136.0.5,10.60.15.5,10.140.1.5,10.20.3.8,10.100.0.5, ...
4	port:	TCP5008, TCP5009, TCP5011
	app:	vau.exe, workstation.exe
	dest:	10.21.49.176
5	port:	TCP2967, UDP1281, UDP2967, UDP38293
	app:	rtvscan.exe, savroam.exe
	dest:	10.130.10.98,10.144.0.5,10.136.0.5,10.2.0.5,10.60.15.5,10.140.1.5,10.20.3.8, ...

Table 5.3: Examples of network services (clusters) extracted from the Nexthink dataset.

Besides the internal metric like the modularity, we can also evaluate the results with some external metrics since now the ground truth is available. Let $\bar{C}$ and C denote the ground truth clustering and the computed clustering respectively. The true positive (TP), false positive (FP), and false negative (FN) for the cluster $\bar{c}_j \in \bar{C}$ are

$$\mathrm{TP}_j(\bar{c}_j, c_j) = |\bar{c}_j \cap c_j|, \tag{5.2}$$

$$\mathrm{FP}_j(\bar{c}_j, c_j) = |c_j \setminus \bar{c}_j|, \tag{5.3}$$

$$\mathrm{FN}_j(\bar{c}_j, c_j) = |\bar{c}_j \setminus c_j|. \tag{5.4}$$

When there are more than two clusters, the averaged F-score can be defined in two ways [Yang, 1999]: the micro-averaged F-score

$$F_{\mathrm{micro}}(\bar{C}, C) = 2\frac{\mathrm{pre}_{\mathrm{micro}} \cdot \mathrm{rec}_{\mathrm{micro}}}{\mathrm{pre}_{\mathrm{micro}} + \mathrm{rec}_{\mathrm{micro}}} \tag{5.5}$$

$$\text{where} \quad \mathrm{pre}_{\mathrm{micro}} = \frac{\sum_j \mathrm{TP}_j}{\sum_j \mathrm{TP}_j + \sum_j \mathrm{FP}_j}, \quad \mathrm{rec}_{\mathrm{micro}} = \frac{\sum_j \mathrm{TP}_j}{\sum_j \mathrm{TP}_j + \sum_j \mathrm{FN}_j}, \tag{5.6}$$

and the macro-averaged F-score

$$F_{\mathrm{macro}}(\bar{C}, C) = 2\frac{\mathrm{pre}_{\mathrm{macro}} \cdot \mathrm{rec}_{\mathrm{macro}}}{\mathrm{pre}_{\mathrm{macro}} + \mathrm{rec}_{\mathrm{macro}}} \tag{5.7}$$

$$\text{where} \quad \mathrm{pre}_{\mathrm{macro}} = \frac{1}{|\bar{C}|}\sum_j \frac{\mathrm{TP}_j}{\mathrm{TP}_j + \mathrm{FP}_j}, \quad \mathrm{rec}_{\mathrm{macro}} = \frac{1}{|\bar{C}|}\sum_j \frac{\mathrm{TP}_j}{\mathrm{TP}_j + \mathrm{FN}_j}. \tag{5.8}$$

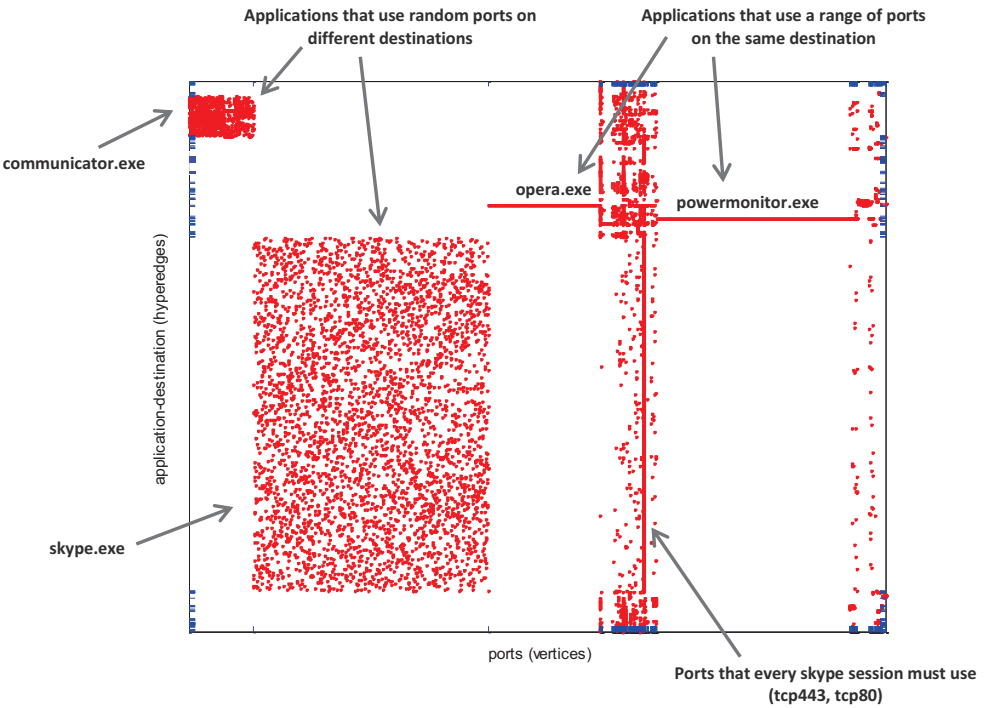

Figure 5.3: A spy plot of the incident matrix for the dataset `Nexthink92`. The rows and columns of the incident matrix are reordered by the learned clusters, and some identified applications are shown in the figure. The clustering is generated by the `HCluster[HE]` with a kNN algorithm in the second step.

Both F-scores range from 0 to 1, and a larger value indicates a better matching between $\bar{C}$ and C. But in order to compute the F-score, one cluster in C has to be mapped to one cluster in $\bar{C}$. We use the matching that maximizes the F-score over all possible matchings. On the other hand, if the clustering is treated as a distribution of vertices, the above matching step can be avoided by computing the mutual information between distributions. Let the $|\bar{C}| \times |C|$ matrix $\boldsymbol{N}$ denote the confusion matrix where $\boldsymbol{N}(i,j) = |\bar{c}_i \cap c_j|$. The normalized mutual information (NMI) to measure the consistency between $\bar{C}$ and C is defined as [Danon et al., 2005]

$$NMI(\bar{C},C) = \frac{-2\sum_{i=1}^{|\bar{C}|}\sum_{j=1}^{|C|}\boldsymbol{N}(i,j)\log\frac{N\cdot\boldsymbol{N}(i,j)}{(\sum_t \boldsymbol{N}(i,t))(\sum_k \boldsymbol{N}(k,j))}}{\sum_{i=1}^{|\bar{C}|}\left(\sum_t \boldsymbol{N}(i,t)\right)\log\frac{\sum_t \boldsymbol{N}(i,t)}{N} + \sum_{j=1}^{|C|}\left(\sum_k \boldsymbol{N}(k,j)\right)\log\frac{\sum_k \boldsymbol{N}(k,j)}{N}}, \tag{5.9}$$

where N is the sum of all entries in $\boldsymbol{N}$. The NMI also ranges from 0 to 1. A value 1 appears if $\bar{C}$ and C are identical and 0 appears if $\bar{C}$ and C are totally independent.

Table 5.4 shows the results of the user affiliation identification problem. Since we know the number of clusters in the ground truth, the k-means algorithm is tested with a fixed parameter $k_{\max} = 4$. The clustering algorithm with the NHC Laplacian works the best in this experi-

ment. It is able to correctly classify the affiliations for over 50% of the users. We can see that the results indicated by the internal metric (the modularity $Q(G_H, C)$) are not consistent with those indicated by the external metrics (the F-scores and the NMI), which suggests that the underlying clusters do not completely follow the the structures established by the affiliations.

	hMETIS	SingleLinkage	k-means	HCluster[NHC]	HCluster[HE]
$\Phi_{HE0}(H,C)$	152	347	317	339	311
$\Phi_{HE}(H,C)$	298	568	529	617	556
$\Phi_N(G_H,C)$	1.362	1.116	1.166	1.171	1.130
$Q(G_H,C)$	0.061	0.143	0.132	0.130	0.146
$F_{\text{micro}}(\bar{C},C)$	0.430	0.397	0.470	0.511	0.441
$F_{\text{macro}}(\bar{C},C)$	0.402	0.392	0.471	0.507	0.436
$NMI(\bar{C},C)$	0.130	0.173	0.353	0.403	0.309
accuracy	0.430	0.397	0.470	0.511	0.441
#clusters	4	4	4	4	4

Table 5.4: Results of the clustering algorithms for the user affiliation problem. The second step clustering algorithm for `HCluster[NHC]` and `HCluster[HE]` is k-means. The parameter $k_{\max}$ is set to 4 for all the approaches. The shown values with a k-means algorithm are averaged over 50 runs since the initialization of k-means is random.

5.2 Semi-supervised Learning

In this section, we compare the semi-supervised learning algorithms on benchmark datasets. In these datasets, the ground truth partition $\bar{C}$ over the vertices is known. All the classification experiments are carried out in a transductive manner: we first create a hypergraph from the raw data. Then the hypergraph and some vertex labels (a partial clustering T) are taken as inputs and the algorithm predicts the labels of the vertices which are not in T.

5.2.1 Experiments Setting

The tested algorithms include Algorithm 2 with the NHC Laplacian (`HSS[NHC]`), Algorithm 5 with the hyperedge expansion Laplacian (`HSS[HE]`), the hMETIS toolkit (`hMETIS`), the rendezvous algorithm (`Rend`), the semi-supervised kernel k-means algorithm (`SSKKmeans`), and the AnchorGraph algorithm (`AnchorGraph`). Please refer to section 4.3.1 for an introduction of the last four algorithms.

The performance of the algorithms are evaluated in repeated runs. In each run, the labeled vertices are randomly chosen from the vertex set such that every class has at least one labeled vertex, and the same set of labeled vertices is applied to all the algorithms in each run to ensure a fair comparison. We use the macro-averaged F-score (see (5.7)) as the evaluation

metric.

In the experiments, we use 13 relational datasets from three different domains. The *AmazonBook co-purchase dataset* contains the books in Amazon.com and the list of other books that are co-purchased [2]. We take three subsets of the book products to construct the hypergraphs. In each hypergraph, a book is represented by a vertex, and a hyperedge represents a co-purchase list of books. The size of a hyperedge is no more than 6 because at most 5 other products can be listed in an Amazon product page. The label of each vertex is simply the category of the corresponding book. We also have excluded the vertices which have no label information from the dataset. The classes (labels) of the constructed hypergraphs are: {Arts and Photography, Biographies and Memoirs, History} for the hypergraph `ABook3`, {Computers and Internet, Engineering, Science Fiction and Fantasy, Science} for the hypergraph `ABook4`, and {Accessories, Bargain Books, Entertainment, GayLesbian, Law} for the hypergraph `ABook5`. The parameter μ in Algorithm 2 and Algorithm 5 is set to 1 for the above three hypergraphs.

We also construct co-citation hypergraphs from the commonly-used `Cora`, `citeseer`, and `WebKB` data [Sen et al., 2008]. They all have a citation structure where a paper is cited by another paper or a webpage is linked by another webpage. For `Cora` and `citeseer`, a vertex in the hypergraph represents a paper, and a hyperedge contains all the papers that cite the same paper (thus one can name a hyperedge by the name of the paper that has been cited). The papers in one hyperedge are assumed to be similar because they refer to the same topic by citing the same paper. For the `WebKB` data (`cornell` and `texas`), besides the link information, word-based content information is also available. Thus we create some additional hyperedges that include all the webpages that contain the same word. The set of words that are included in the dataset is selected by a pre-processing procedure so that the stop words and less important words are excluded. In order to show how the link information could help in semi-supervised learning, the hypergraphs using only contents (denoted by C) and contents plus links (denoted by CL) are constructed respectively for each `WebKB` dataset. The parameter μ in Algorithm 2 and Algorithm 5 is set to 50 for the co-citation hypergraphs.

In the last domain, categorical dataset, every entity has a set of nominal attributes which could take values from a finite set. We use four labeled categorical datasets: `zoo`, `letter`, `20newsgroups`, and `covertype` from the UCI repository [3]. For each dataset, a hypergraph is constructed by taking the entities as vertices and creating a hyperedge for each value of the attributes. Then every hyperedge should contain the entities that share the same attribute value. For example, in the hypergraph `zoo`, a vertex represents an animal, and the hyperedge "legs=2" should include all the animals which have 2 legs. We discretize those attributes

[2] Available at http://snap.stanford.edu/data/amazon-meta.html.
[3] Available at http://archive.ics.uci.edu/ml/.

whose value is an integer with a cardinality larger than 10 into 10 sections. Some tested algorithms do not scale very well on the datasets `letter`, `20newsgroups`, and `covertype`, so only a subset is tested for each of them. We set the parameter $\mu = 1$ for `20newsgroups` and $\mu = 100$ for all the other hypergraphs. Weighting the hyperedges usually depends on the domain knowledge. For simplicity, we assign the same weight to all the hyperedges in the experiments. Table 5.5 shows the statistics of the hypergraphs.

hypergraph	#vertices	#hyperedges	#clusters	#labeled vertices in T
ABook3	24500	29709	3	100
ABook4	18120	20219	4	100
ABook5	6965	8152	5	80
Cora	1961	875	7	40
citeseer	1318	638	6	40
cornell-CL	195	1254	5	20
cornell-C	195	1209	5	20
texas-CL	187	1125	5	20
texas-C	187	1074	5	20
zoo	100	36	7	15
letterAE	1022	189	5	50
20newsgroups	1067	100	4	50
covertype	6344	140	7	50

Table 5.5: The statistics of the hypergraphs in the semi-supervised learning experiments. The number of labeled vertices in the partial clustering T is also shown for each hypergraph.

5.2.2 Main Results

As shown in Table 5.6, the algorithm `HSS[HE]` performs significantly better than other methods in most cases. In the hypergraph `ABook5`, the difference between `HSS[HE]` and `HSS[NHC]` becomes indistinguishable mainly due to the large variance of the F-score. In fact, the ratio of the maximum class size over the minimum class size is much larger in `ABook5` (835.4) than the ratio in `ABook3` (1.5) and `ABook4` (10.3). With the unbalanced class sizes, the F-score of small classes would be more sensitive to the choice of labeled vertices in T, which leads to the high variance of the macro averaged F-scores over all the runs. For some datasets, the `hMETIS` algorithm does not work very well, partially because it is originally designed for the VLSI applications, but not general classification tasks. From the results of `cornell` and `texas` with the `HSS[HE]` algorithm, we can observe an improvement from the content only case (C) to the content+link case (CL). It confirms that the link information does help in classifying the webpages.

hypergraph	hMETIS	SSKKmeans	AnchorGraph	Rend	HSS[NHC]	HSS[HE]
ABook3	0.565(0.022)	0.446(0.015)	0.519(0.025)	—	0.645(0.019)	**0.657**(0.023)
ABook4	0.517(0.107)	0.376(0.016)	0.561(0.058)	—	0.765(0.046)	**0.798**(0.023)
ABook5	0.525(0.040)	0.357(0.067)	0.472(0.046)	—	0.724(0.087)	0.716(0.064)
Cora	0.477(0.054)	0.449(0.048)	0.500(0.050)	—	0.613(0.046)	**0.637**(0.040)
citeseer	0.492(0.046)	0.361(0.030)	0.401(0.038)	—	**0.518**(0.046)	0.509(0.046)
cornell-CL	0.275(0.055)	0.411(0.091)	0.417(0.058)	0.304(0.068)	0.320(0.091)	**0.497**(0.047)
cornell-C	0.279(0.057)	0.427(0.092)	0.425(0.059)	0.299(0.056)	0.346(0.069)	**0.480**(0.050)
texas-CL	0.238(0.028)	0.362(0.050)	0.317(0.066)	0.249(0.042)	0.268(0.089)	**0.425**(0.045)
texas-C	0.236(0.039)	0.350(0.047)	0.338(0.050)	0.254(0.047)	0.267(0.098)	**0.410**(0.068)
zoo	0.467(0.066)	0.822(0.058)	0.803(0.075)	0.571(0.088)	0.359(0.147)	**0.832**(0.052)
letterAE	0.379(0.049)	0.629(0.023)	**0.664**(0.039)	0.543(0.039)	0.606(0.047)	0.627(0.028)
20newsgroups	0.489(0.080)	0.480(0.041)	0.552(0.042)	0.482(0.069)	**0.642**(0.033)	0.628(0.042)
covertype	0.164(0.017)	0.285(0.019)	0.238(0.016)	0.268(0.022)	0.254(0.064)	**0.307**(0.028)

Table 5.6: The macro-averaged F-scores (and the standard deviation in the parentheses) on 13 hypergraphs. The algorithms are tested on ABook3, ABook4, ABook5, and covertype with 10 runs, on the co-citation data with 50 runs, and on the other categorical data with 100 runs. The bold number indicates the algorithm that performs significantly better than all the others (p-value < 0.05 in a paired t-test). The Rendezvous algorithm (Rend) cannot return a result in a reasonable time for some hypergraphs, so some F-scores are missing.

Nevertheless, the algorithms that are designed for hypergraphs (e.g. hMETIS, HSS[NHC] and HSS[HE]) generally perform significantly better than those based on the graphs (SSKKmeans) or feature vectors (AnchorGraph). It suggests that the hypergraph approaches would be better choices when the data is naturally organized as co-occurrence relations. For the hypergraph letterAE, the AnchorGraph algorithm actually works best, because the original attributes of letterAE are all integer values on the real line (such as the mean of the x-positions of the pixels) rather than nominal variables. When we construct the hypergraph for letterAE, these integer values are discretized into intervals to form hyperedges, which might cause information loss. If the raw data is represented as vectors in a continuous vector space, methods like the AnchorGraph could be better for capturing the underlying patterns.

The running time of the algorithms is tested with different subsets of covertype such that the sizes of the vertex sets range from 583 to 27056. These subsets are randomly extracted from the original data. Figure 5.4 shows the measured times in log scale. We have shown that the complexity of the HSS[HE] algorithm mainly depends on the number of hyperedges. The running time of HSS[HE] does not change too much when increasing the number of vertices. This is because the number of hyperedges in the subsets remains in the same level (from 122 to 143). Therefore the algorithm based on the hyperedge expansion can be orders of magnitude faster than the approaches based on the vertex expansions if the number of hyperedges is much smaller than the number of vertices. By increasing the parameter μ,

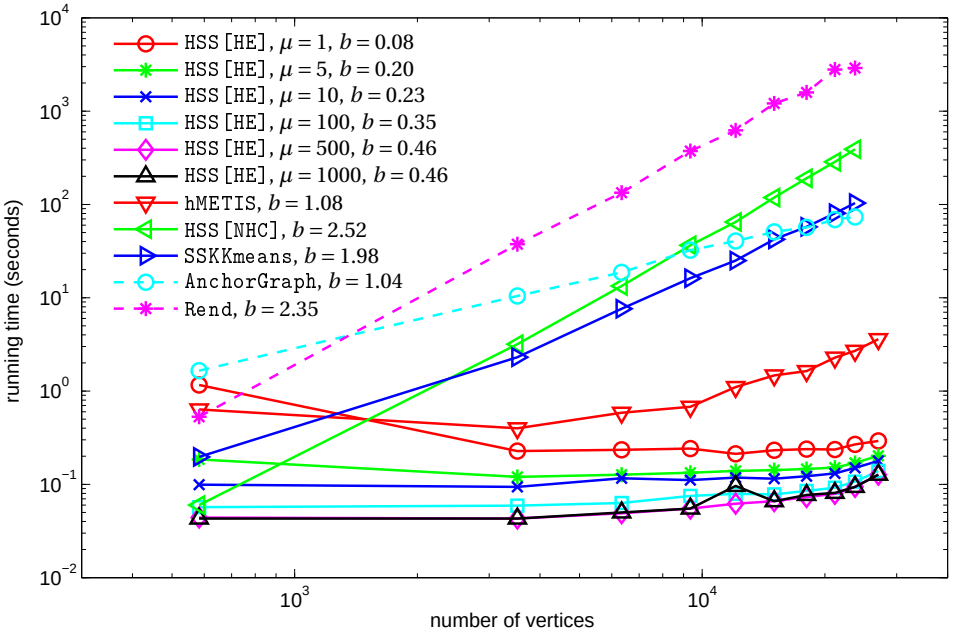

Figure 5.4: The measured running time of the algorithms with 100 labeled vertices on different subsets of `covertype`. The slope b of each curve is shown in the legend, which is computed by the least square fitting. Notice that both axes are in log-scale. The `HSS[HE]` algorithm is tested with different parameters μ.

we can observe that `HSS[HE]` runs faster due to the higher convergence rate of the eigenvector computation, which confirms the conclusion in [Mavroeidis, 2011]. In practice, the choice of μ also depends on the classification performance, but the running time would not change by more than an order of magnitude when tuning μ. The running times of `hMETIS` and `AnchorGraph` grow linearly with respect to the number of vertices, while for `HSS[NHC]`, `SSKKmeans` and `Rend` the running time grows quadratically. We have shown that the time complexity of `HSS[NHC]` is proportional to the number of non-zeros in the Laplacian $\boldsymbol{L}_{\text{NHC}}$, and the sparsity ($\frac{\#\text{non-zeros}}{\#\text{all entries}}$) of $\boldsymbol{L}_{\text{NHC}}$ would be a constant. Thus the running time of `HSS[NHC]` is proportional to n^2 where n is the number of vertices and $n \times n$ is the size of $\boldsymbol{L}_{\text{NHC}}$.

Figure 5.5 shows a comparison of the embeddings using the vertex expansion and the hyperedge expansion. The vertices (animals) of the hypergraph `zoo` are projected into a 2-dimensional space. It can be seen that the hyperedge expansion generates a different picture from that of the vertex expansion. The hyperedge expansion shows a better separation between different classes in the 2-dimensional space, but for some animals (e.g. `seasnake` and `platypus`) both embeddings fail to assign them to a cluster, mainly due to their special attributes.

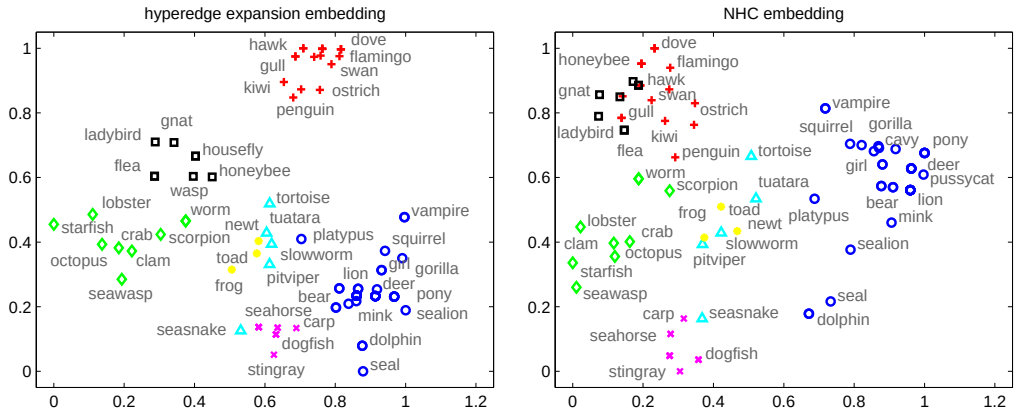

Figure 5.5: The vertex embeddings of `zoo` with the (scaled) eigenvectors corresponding to the 2nd and 3rd smallest eigenvalues. The clusters of animals are annotated with different markers. Left: the embeddings from the hyperedge expansion Laplacian $\boldsymbol{L}_{\text{HE}}$. Right: the embeddings from the NHC Laplacian $\boldsymbol{L}_{\text{NHC}}$.

5.3 Recommender System

5.3.1 Introduction

The recommender systems have been studied since the rise of web-services and gained great success in recent years [Resnick and Varian, 1997, Adomavicius and Tuzhilin, 2005]. Such system tries to predict the possible preferences or ratings from a user to an item (movie, music, social elements, etc.) by looking into the attributes of the item or the history of user behaviors. Then the items can be recommended to a user with an order ranked by some personalized relevance scores.

In this work we mainly focus on the relation-based approaches that make predictions according to the user behaviors. Obviously, the relations between the users and the items can be modeled as a hypergraph with the bi-partite graph representation, or as a weighted relation matrix $\boldsymbol{R}$ where $\boldsymbol{R}(i, j)$ is the rating from the row i (a user) to the column j (an item). The memory-based collaborative filtering (CF) systems first find a neighborhood for each user (or item) whose behavior is similar to all the entities in the neighborhood [Schafer et al., 2007]. By the assumption that all the users (or items) in the neighborhood have the similar tastes or properties, the recommendations are made from the collective preferences of the whole neighborhood.

On the other hand, instead of assuming that the users would behave similarly in a neighborhood, one can create a model to describe the underlying mechanism from which the relations are generated. The model-based systems usually involve the so-called latent factors that can

be shared and matched between the users and the items. For example, to represent the ratings in a music website, the rows of $\boldsymbol{R}$ are the users , and the columns of $\boldsymbol{R}$ are the artists. A typical recommender system would try to guess the user's interests and generate a list of unseen artists that meets the user's interests. Normally there is already a set of old users with many seen artists (many non-zero entries in $\boldsymbol{R}$). A model-based system would try to extract the latent factors, e.g. preferences over music genres, from the matrix $\boldsymbol{R}$. When a new user comes into the system, based on a few relations from the new user to the artists, the system would try to align the new user's latent factors (interests) to the existing ones learned from the previous step. Then the predictions of the most probable relations can be made from the aligned latent factors.

Among many other model-based approaches such as probabilistic model or latent semantic model, the matrix factorization technique is an ideal tool for extracting latent factors from the rating matrix $\boldsymbol{R}$. Early works (e.g. [Sarwar et al., 2000]) adopt the singular value decomposition (SVD) to make a low-rank approximation of a matrix, and produce predictions based on the generalization of the low-rank SVD to the missing entries. Along this line of works, Funk [Funk, 2006] has developed a SVD algorithm that ignores missing values in the matrix, which could scale up to bigger datasets. Paterek [Paterek, 2007] and Koren [Koren, 2008] have pointed out that better predictions can be obtained from an asymmetric model. In the asymmetric model, only the right singular vectors are used in the prediction step.

Since the rating matrix can be modeled as a hypergraph, we propose to use the link prediction algorithm developed in section 3.4 to serve as a recommender system. In the hypergraph representation, a vertex usually represents an item, and a user is denoted by a hyperedge. The model behind our algorithm is the multi-class beta-Bernoulli process (see section 2.4), which assumes that the items in the recommender system can be organized into some clusters and the users choose to engage with the items in each cluster by some probabilities generated from a beta distribution. For example, in the `music` data, a cluster of items would include all the artists who play a music genre or a combination of genres, where the latter case allows us to handle the artists who play in multiple genres in our model.

Figure 5.6 shows the setting of our system. The rating matrix $\boldsymbol{R}$ is split into three parts. The sub-matrix $\boldsymbol{R}_{\text{m}}$ contains most rows of $\boldsymbol{R}$ which represents the old users in the system. The remaining rows of $\boldsymbol{R}$ are further split into $\boldsymbol{R}_{\text{tr}}$ and $\boldsymbol{R}_{\text{te}}$, where a few ratings are available in $\boldsymbol{R}_{\text{tr}}$ and our goal is to predict more relations in the $\boldsymbol{R}_{\text{te}}$ part. This setting simulates a recommender system where we have a rich set of existing ratings and some new users would give only a few ratings. In fact, this setting has the flexibility to allow us to learn the latent factors from any sub-matrix of $\boldsymbol{R}$ (if the sub-matrix has all the columns of $\boldsymbol{R}$), and to predict new relations for any rows. The three matrices $\boldsymbol{R}_{\text{m}}$, $\boldsymbol{R}_{\text{tr}}$ and $\boldsymbol{R}_{\text{te}}$ do not have to be disjoint. Notice that our setting does not consider the cold-start problem when there are not enough ratings in the

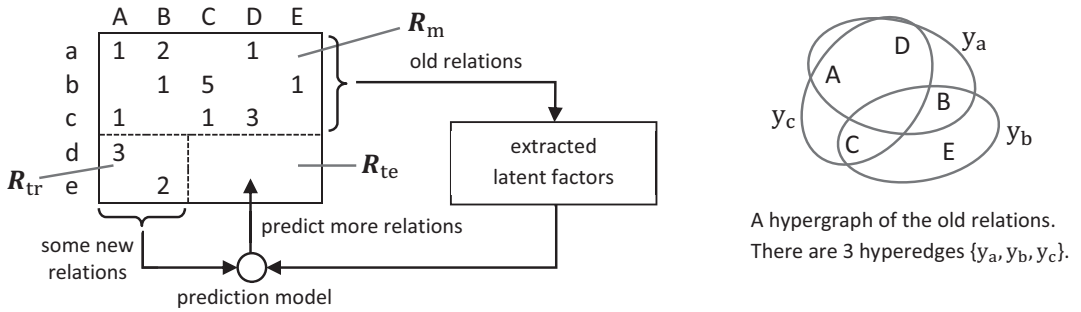

Figure 5.6: An illustration of recommender system with a hypergraph representation.

sub-matrix $\boldsymbol{R}_{\mathrm{m}}$ to learn a good set of latent factors. Usually one has to resort to a hybrid approach that combines the contents of the items and the relations [Lam et al., 2008].

We have shown in Algorithm 3 that the latent factors can be obtained by the truncated SVD and the predictions can be made by computing a vector in the latent space for each user. In section 3.2, it is also shown that the right singular vectors of the truncated SVD approximately form some clusters which have an orthogonal structure. This justifies our algorithm which has a similar asymmetric model to those in [Paterek, 2007] and [Koren, 2008].

5.3.2 Experiments Setting

Three datasets from the music domain and the book domain are used to test the performances of the algorithms. We choose these two domains because they both have a categorical structure on the items (songs, artists, books) and both domains possess an enormous number of items so that recommendations are desired by the users. The first dataset `Lastfm` [4] contains the relations between users and artists, and the rating associated with the relation is the actual count of plays with which a user has listened to an artist. A subset of 5848 artists and 19810 users are selected to construct the hypergraph. The second dataset `YahooMusic` [5] is also in the music domain, but includes the relations between users and tracks. The ratings in `YahooMusic` are in the range from 0 to 100. We also take a subset of the raw data and include 5644 tracks and 6020 users to construct the hypergraph. The last dataset `BookCrossing` contains the ratings from the users in `bookcrossing.com` on books (see [Ziegler et al., 2005] for more details). The range of ratings is from 1 to 10. We take a subset of 5557 books and 1864 users in the hypergraph.

The algorithms are evaluated in two scenarios. Firstly we test the algorithm's ability to dis-

[4] Available at `http://mtg.upf.edu/node/1671`, collected from `last.fm`.

[5] Available at `http://labs.yahoo.com/Academic_Relations`, the Yahoo! Webscope dataset ydata-ymusic-kddcup-2011-track2.

cover unseen relations. The rating matrix is split into three parts as shown in Figure 5.6. The matrix $\boldsymbol{R}_{\mathrm{m}}$ contains all the rows that correspond to the users who have rated more than 60 items. Let Y_{m} denote this set of users, and Y_{t} denote the set of other users. Then we randomly select $\boldsymbol{R}_{\mathrm{tr}}$ to have 5 ratings for each row (each user) that is in Y_{t}, and the remaining ratings are all assigned into $\boldsymbol{R}_{\mathrm{te}}$. This setting simulates the situation where new users just come into the system with only a few ratings, which is denoted as `ON-5`. Similarly, we create another setting called `ON-20` where each row of $\boldsymbol{R}_{\mathrm{tr}}$ has 20 ratings to simulate the users with more available ratings. In all the experiments, $\boldsymbol{R}_{\mathrm{tr}}$ and $\boldsymbol{R}_{\mathrm{te}}$ are randomly selected in 5 different runs. Since all the items which are not in $\boldsymbol{R}_{\mathrm{tr}}$ can be recommended in the final result, this scenario is denoted as `PREDICT-ALL`.

All the tested algorithms have to make predictions for the users in Y_{t}, based on the $\boldsymbol{R}_{\mathrm{m}}$ and $\boldsymbol{R}_{\mathrm{tr}}$ parts. In Algorithm 3, the $\boldsymbol{R}_{\mathrm{m}}$ part is used to compute the item embeddings, i.e. let $\boldsymbol{X} = \mathrm{sign}(\boldsymbol{R}_{\mathrm{m}})$ in step 3 of Algorithm 3. And the $\boldsymbol{R}_{\mathrm{tr}}$ part is used to calculate the user vector in step 6, i.e. obtain $\boldsymbol{\Theta}$ by solving $\boldsymbol{F}\boldsymbol{\Theta} = \boldsymbol{R}_{\mathrm{tr}}^{\top}$. Finally the top k_{r} unseen items with the highest prediction scores are selected for each user in Y_{t}. The other two algorithms introduced in section 3.4.1, the `PredictSVDR2` and `PredictNMFR2`, are also adapted into the above setting.

To evaluate the recommendations, for each user, we check if the set of recommended items (the k_{r} unseen items with the highest prediction scores) coincides with the set of liked items. The set of liked items for a user is defined as the items with ratings higher than the median in $\boldsymbol{R}_{\mathrm{te}}$. For the user y_i, let $H_i^{k_{\mathrm{r}}}$ denote the set of top k_{r} recommended items, and H_i^{like} denote the set of liked items. We measure the average precision

$$\mathrm{pre}_{\mathrm{avg}} = \frac{1}{|Y_{\mathrm{t}}|} \sum_{y_i \in Y_{\mathrm{t}}} \frac{|H_i^{k_{\mathrm{r}}} \cap H_i^{\mathrm{like}}|}{|H_i^{k_{\mathrm{r}}}|}. \tag{5.10}$$

Sometimes we might already know a set of candidate relations that the user has provided implicitly. For example, a user has viewed a list of pages of artists, but did not make any further actions like purchase a CD or download a track. In this case, we can limit our predictions on the candidate relations and only recommend the items that are truly liked by the user. In other words, the predictions are made on the items already seen in $\boldsymbol{R}_{\mathrm{te}}$, and we only try to rank them such that the liked ones are ranked higher. This scenario is denoted as `RANK-CANDIDATES`. The average precision defined in (5.10) is also used to evaluate `RANK-CANDIDATES`.

To summarize, there are two scenarios: `PREDICT-ALL` and `RANK-CANDIDATES`. And within each scenario there are two settings: `ON-5` and `ON-20`. When evaluating the results, we select the top $k_{\mathrm{r}} = 20$ recommendations for each user. Besides our proposed Algorithm 3, the other 6 algorithms introduced in section 3.4.1 are also tested in the experiments.

5.3.3 Main Results

Table 5.7 and Table 5.8 show the performances of PREDICT-ALL and RANK-CANDIDATES respectively. In the first scenario, our proposed algorithm performs best in the music domain, and shows results as good as the best methods in BookCrossing. The algorithms ignoring the empty values (PredictNZR and PredictANZR) do not perform very well, mainly because these methods are originally designed to minimize the RMSE (root mean squared error) on a testing set of known ratings, which is more similar to the RANK-CANDIDATES scenario. Thus we observe poor performances from PredictNZR and PredictANZR for the PREDICT-ALL scenario. The difference between the SVD-based methods and the NMF-based methods is not distinguishable with the ON-5 setting. But with more ratings available for each user in the ON-20 setting, the SVD-based methods perform slightly better than the NMF-based methods. Since the only different between PredictNHC and PredictSVDR2 is the matrix used to compute the embeddings, the better performance of PredictNHC suggests that the normalizations in the NHC Laplacian do help in constructing a better latent space. If we compare the performances of ON-5 and ON-20 in the same algorithm, the average precision of ON-20 is slightly worse than that of ON-5. This result is a bit counter-intuitive and mainly due to the fact that we measure the precision by comparing the recommendation set with the set of highly-rated items in $\boldsymbol{R}_{\text{te}}$. In the ON-20 setting, there are more rated items in $\boldsymbol{R}_{\text{tr}}$ but less in $\boldsymbol{R}_{\text{te}}$, so it would be harder to find good recommendations in $\boldsymbol{R}_{\text{te}}$.

	Lastfm		YahooMusic		BookCrossing	
	ON-5	ON-20	ON-5	ON-20	ON-5	ON-20
PredictSVDR	0.113	0.083	0.237	0.207	0.078	0.063
PredictSVDR2	0.114	0.087	0.237	0.210	0.078	0.062
PredictNMFR	0.114	0.090	0.218	0.189	0.073	0.054
PredictNMFR2	0.110	0.089	0.211	0.190	0.071	0.056
PredictNZR	0.002	0.003	0.001	0.001	0.005	0.003
PredictANZR	0.003	0.002	0.007	0.005	0.005	0.003
PredictNHC	**0.180**	**0.158**	**0.258**	**0.227**	0.075	0.065

Table 5.7: The average precisions in the PREDICT-ALL scenario. The bold number indicates a method that performs significantly better than the others (p-value < 0.05 in a paired t-test).

In the RANK-CANDIDATES scenario, no method performs significantly better than the others over all datasets. But the NMF-based method PredictNMFR, the asymmetric method PredictANZR and our proposed algorithm PredictNHC are constantly as good as the best one. For the comparison between ON-5 and ON-20 in the same algorithm, we do not observe a clear performance drop from the ON-20 setting to the ON-5 setting. Because in this scenario, an algorithm could only recommend the rated items in $\boldsymbol{R}_{\text{te}}$.

	Lastfm		YahooMusic		BookCrossing	
	ON-5	ON-20	ON-5	ON-20	ON-5	ON-20
PredictSVDR	0.198	0.140	0.490	0.418	0.743	0.744
PredictSVDR2	0.198	0.139	0.496	0.435	0.743	0.745
PredictNMFR	**0.541**	**0.545**	0.763	0.774	0.731	0.740
PredictNMFR2	0.279	0.222	0.579	0.496	0.743	0.745
PredictNZR	0.177	0.244	0.249	0.224	0.389	0.165
PredictANZR	0.537	0.531	0.755	**0.783**	0.696	0.714
PredictNHC	0.515	0.524	0.765	0.766	**0.745**	**0.747**

Table 5.8: The average precisions in the RANK-CANDIDATES scenario. The bold number indicates a method that performs significantly better than the others (p-value < 0.05 in a paired t-test).

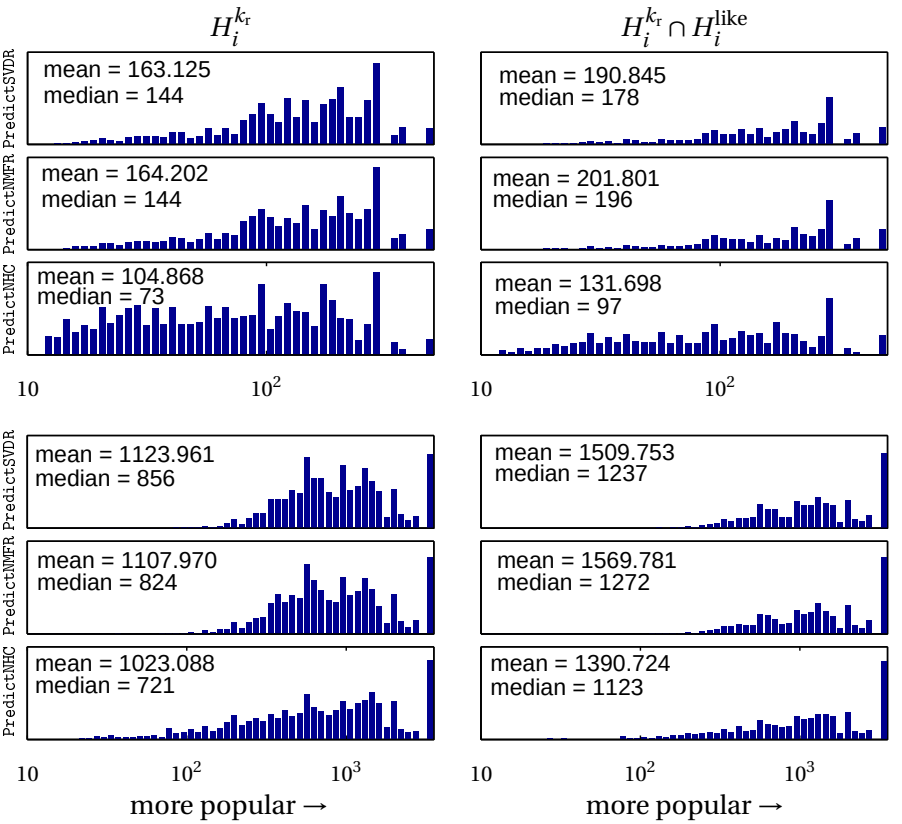

Figure 5.7: The popularity distribution of the recommended items in the PREDICT-ALL scenario. The x-axis (popularity) is in log-scale. The mean and median of the popularity are also shown in the figure. Top: BookCrossing. Bottom: YahooMusic.

In the recommender systems, people are also interested in recommending less-popular items, because the users are usually already aware of the popular items from other sources. For

the purpose of diversifying the recommendations, we measure the popularity distribution of items in $H_i^{k_r}$ and $H_i^{k_r} \cap H_i^{like}$, i.e. in the set of recommended items and in the set of successfully recommended items. The popularity of an item z_k in Z is calculated as the number of ratings in the k-th column of the rating matrix $\boldsymbol{R}$. Figure 5.7 shows the distributions in the PREDICT-ALL scenario. We can observe that PredictNHC produces more diverse recommendations with the same (or better) level of precision. Especially with the BookCrossing data, there are much more less-popular recommendations from PredictNHC compared to those from PredictSVDR and PredictNMFR.

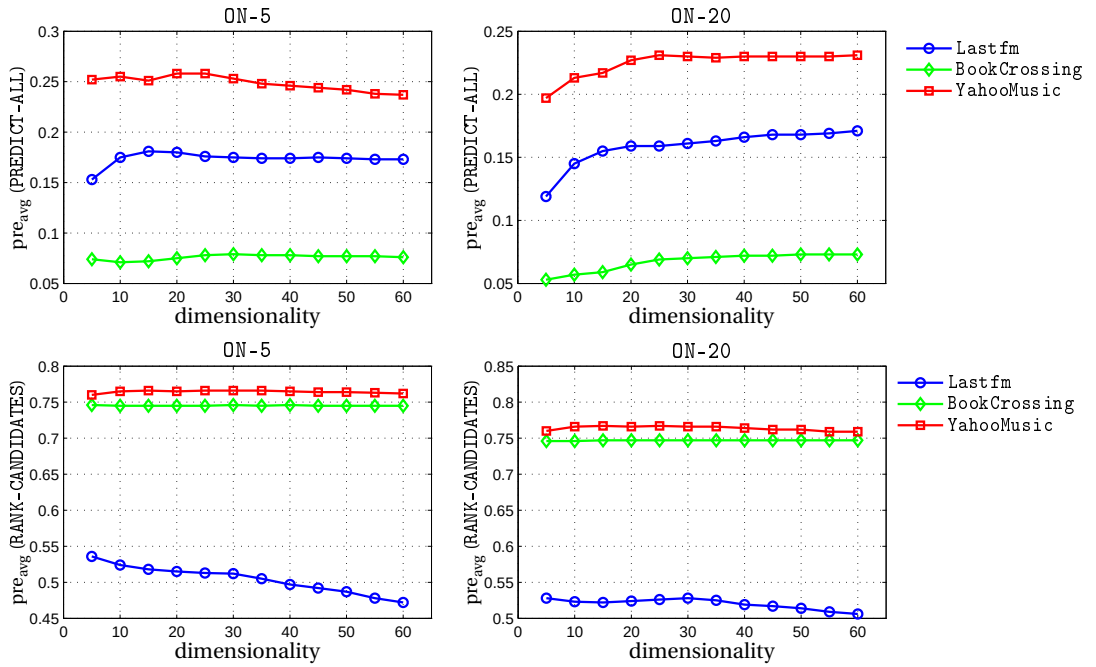

Figure 5.8: The average precisions with different dimensionality parameters l in the algorithm PredictNHC.

To examine the influence of the dimensionality parameter l in the algorithm PredictNHC, the average precisions are measured with changing l values, as shown in Figure 5.8. In the PREDICT-ALL scenario, the average precision stays in the same level when $l > 20$. It means that we cannot achieve better predictions by using larger l beyond some threshold, though the threshold slightly varies for different datasets. In the RANK-CANDIDATES scenario, the average precision does not change when using different dimensionalities. An exception is the Lastfm dataset with the ON-5 setting. The performance actually decreases as the dimensionality getting bigger. This result might suggest that the embeddings are crucial for discovering possible new relations (PREDICT-ALL), but less important for ranking the possible relations (RANK-CANDIDATES).

6 Conclusion

In this work, we have proposed to use the hypergraph model to represent the co-occurrence relations. Compared to the graph model, the hypergraph is more expressive and can be applied to many real applications. We have discussed the generating process of a hypergraph, the vertex expansions and the hyperedge expansion, as well as the experiments on three relational learning tasks.

6.1 Hypergraph-based Relational Learning

For the hypergraph problems, a generative model is proposed to simulate the mechanism from which the relations in the hypergraph are created. The multi-class beta-Bernoulli process is extended from the beta process, a nonparametric Bayesian model, and we have shown that the frequency of the vertices which have the same degree follows a power-law distribution. In most hypergraphs in our experiments, we do observe such power-law distribution. Although the basic assumption requires the mutual exclusion between the vertices from different clusters, this generative model can be applied to many cases where the vertices have a set of categorical attributes.

Starting from the multi-class beta-Bernoulli process, we have examined the sparsity of the hypergraph Laplacians. It has been shown that the sparsity is a fixed value which depends on the parameters of the multi-class beta-Bernoulli process and is independent of the size of the hypergraph. Because the computational complexity of the hypergraph learning algorithms is linear to the number of non-zeros in the Laplacian, a fixed sparsity implies that the running time would increase quadratically with respect to the hypergraph size. This result is also observed in the experiments.

Based on the multi-class beta-Bernoulli process, the vertex embeddings from the NHC Laplacian are shown to have a special structure in the embeddings space. In fact, the embedding

vectors from s clusters of vertices also aggregate into s clusters in the embeddings space, and the cluster centers are approximately orthogonal to each other. This result extends the similar result in the graphs to the hypergraphs. Unlike the graph case, we show that the embeddings are no longer aligned to the axes of the embedding space, but up to a rotation that depends on the distribution of relations. This rotation could be very sensitive to modifications of the hypergraph, if the second smallest eigenvalue of the NHC Laplacian is not far away from the rest of the spectrum.

We utilize the special structure of the NHC embeddings to develop a link prediction algorithm, which has an asymmetric style in using only the right singular vectors. In the experiments of recommender systems, it has been shown that our algorithm outperforms the existing approaches based on matrix factorization techniques. Since the only difference from our algorithm to one of the baselines (`PredictSVDR2`) is the step of embedding computation. The empirical results suggest that hypergraph-based models can better handle the relations in a recommender system by means like hyperedge size normalization and special designed weighting functions.

Instead of the vertex-centric view in the vertex expansions, we have also examined the other possibility of the hyperedge-centric view. The hyperedge expansion Laplacian is proposed from a transformation from the minimum hyperedge cut problem to a min-cut problem in a directed auxiliary graph. We have shown the conditions with which the spectrum of the hyperedge expansion Laplacian are all real, although the hyperedge expansion Laplacian is a non-Hermitian matrix and thus could have a complex spectrum. The eigenvalue problem of the hyperedge expansion Laplacian is also linked to a quadratic eigenvalue problem, which has a different spectrum structure from that of an undirected graph. We have shown that if the corresponding quadratic eigenvalue problem is overdamped, the minimum hyperedge cut can be lower bounded by the second smallest eigenvalue of the hyperedge expansion Laplacian. With the hyperedge expansion, the proposed algorithms for clustering and semi-supervised learning tasks are tested on some benchmark datasets. The empirical results show that our approach is significantly better than the state-of-the-art methods when the data can be naturally organized by some co-occurrence relations.

6.2 Vertex Expansion vs. Hyperedge Expansion

The vertex expansion and the hyperedge expansion can be seen as different parts of a star expansion (although the weights cannot be directly derived from the star expansion). But given a dataset, which expansion should be applied is more than a simple rule. Generally we should start from the objectives defined for the expansions and find the objective (and the expansion) that would better formulate the problem.

One should also examine the statistics of the co-occurrence relations in the hypergraph. If the hyperedges are all small compared to the whole vertex set, the vertex expansion could be better in capturing the clusters in the hypergraph. Because if the hyperedge sizes are small, the information loss in the transformation to the induced graph is negligible. On the other hand, if there are many large hyperedges, a vertex expansion would produce big cliques that include almost all the vertices in the induced graph and thus have difficulties in finding clusters. In this case, the hyperedge expansion would be a better choice, because the objective of the hyperedge expansion does not depend on the hyperedge sizes.

Computational cost is another criterion to choose the expansion. We have shown that the running time of a vertex expansion algorithm increases quadratically with respect to the number of vertices, and the running time of a hyperedge expansion algorithm has the similar relation to the number of hyperedges. Therefore, if the hypergraph has much more vertices than hyperedges, the hyperedge expansion would run faster, and vice versa.

6.3 The Hypergraph Analysis Toolkit

Along with the study of hypergraphs, we have developed a MATLAB toolkit called the hypergraph analysis toolkit, or `HAT` [1]. The `HAT` includes 8 sets of functions from data pre-processing, algorithms, evaluation to visualization.

The `preprocessing` folder contains the functions to import a csv file or a excel file into a hypergraph structure, the functions to manipulate a hypergraph (e.g. taking a sub-hypergraph by labels, removing small hyperedges, etc.), and the functions to add additional attributes to the vertices or the hyperedges. The `graph` and `hypergraph` folders include the functions for computing graph-based or hypergraph-based metrics (e.g. the normalized edge cut, graph modularity, degree distributions, etc.), and the functions for transforming a hypergraph into the induced graph. The `clustering`, `classification` and `task` folders provide the hypergraph learning algorithms based on the vertex expansions and hyperedge expansion, and a set of functions to run the algorithms in batch mode. In the `evaluation` folder, various external and internal metrics are supported to evaluate the results of the algorithms. Finally the `visual` folder offers different ways of presenting a graph or a hypergraph together with the algorithm outputs. We also provide a function to export the graph structure to the state-of-the-art visualization software *Gephi* such that the results can be presented with a professional visualization.

[1] Available at http://lia.epfl.ch/index.php/research/relational-learning.

6.4 Future Directions

The hypergraph has been successfully applied to many applications, but the theoretical aspect of the hypergraph learning is far from perfect. We list several future research directions as follows.

In the research of complex networks, various models are proposed to simulate the power-law distributions, or more general distributions like the log-normal distribution or the double Pareto log-normal distribution, which are observed in the real world datasets. Some well-known models include the preferential attachment model [Newman, 2003] and the forest fire model [Leskovec et al., 2005]. Some nonparametric Bayesian models also show the similar power-law behaviors [Paisley et al., 2010, Broderick et al., 2012]. Our model, the multi-class beta-Bernoulli process, only shows a power-law distribution on the vertex degrees. But none of the existing models can be applied to the hypergraph so that the hypergraph could exhibit power-law distributions on both vertex degrees and hyperedge degrees, which we do observe in real datasets (e.g. `Lastfm`, `YahooMusic` and `BookCrossing`). Some existing works have pointed out the hints for creating such models [Reed and Hughes, 2003, Chakrabarti et al., 2004, Seshadri et al., 2008, Clauset et al., 2009], and the key component is a multiplicative process that adds a relation to an entity with probability proportional to the number of relations already attached to that entity, or a sequence of decaying probabilities that simulates the multiplicative process in a reversed order. An interesting future work is to develop a new model with the desired distributions, and utilize this model for improving the clustering/classification algorithms. Since we already have clusters in the current model, the hierarchical clustering structure proposed in the forest fire model could be a possible extension for further studies.

As the volume of data created everyday exceeds 2.5 quintillion (2.5×10^{18}) bytes [2], processing them becomes an important issue in the big-data age. It generally requires the algorithms to have linear or at most quasilinear complexity for obtaining results in a reasonable amount of time, and approximate solutions could be accepted for large-scale problems. On the other hand, the computational infrastructure needs to flexibly handle tasks of different sizes, unexpected data loss, and skewed data distributions. The graph or hypergraph datasets in real applications usually exhibit the properties like large size and power-law distribution of the vertex (or hyperedge) degrees. It has been shown that the commonly used `Map-Reduce` model is not able to handle such datasets, and even the systems specially designed for graphs, like Google's Pregel [Malewicz et al., 2010], are not efficient enough [Gonzalez et al., 2012a]. The main obstacle is that distributing the computations for the high degree notes over the machines leads to a huge communication overhead. In fact, the optimal strategy to split the

[2] http://www-01.ibm.com/software/data/bigdata/

graph (or hypergraph) over the machines is equivalent to a minimum hyperedge cut problem. For the large-scale eigen-decomposition and SVD problems, we need to design new efficient algorithms which take the computational infrastructure into considerations.

When we transform the minimum hyperedge cut to the min-cut of the directed auxiliary graph, the minimum vertex separator is introduced as an intermediate step. There are many other applications whose solution can be found from a minimum vertex separator [Kayaaslan et al., 2012, Zhu et al., 2012]. For example, in the inference problem in a probabilistic graphical model and the SAT solver, a minimum vertex separator is often desired to divide the original problem into smaller ones by recursive conditioning. We have shown that the minimum vertex separator can be approximately obtained from a QEP. More works can be done along this direction to enrich the understanding of the vertex separator problem. For example, we could develop an algorithm that recursively produces bi-clusterings with minimum vertex separators.

Bibliography

A. Abou-Rjeili and G. Karypis. Multilevel algorithms for partitioning power-law graphs. In *20th International Parallel and Distributed Processing Symposium*, 2006.

E. Acar, D. M. Dunlavy, and T. G. Kolda. Link prediction on evolving data using matrix and tensor factorizations. In *IEEE International Conference on Data Mining Workshops*, 2009.

S. Acid and L. Campos. An algorithm for finding minimum d-separating sets in belief networks. In *Proceedings of the 12th UAI*, 1996.

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

S. Agarwal, J. Lim, L. Zelnik-Manor, P. Perona, D. Kriegman, and S. Belongie. Beyond pairwise clustering. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.

S. Agarwal, K. Branson, and S. Belongie. Higher order learning with graphs. *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, 2000.

A. Azran. The rendezvous algorithm: Multiclass semi-supervised learning with markov random walks. In *Proceedings of the 24th ICML*, 2007.

F. R. Bach and M. I. Jordan. Learning spectral clustering, with application to speech separation. *The Journal of Machine Learning Research*, 7:1963–2001, 2006.

U. Bayer, P. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering. In *Network and Distributed System Security Symposium (NDSS)*, 2009.

J. Bennett and S. Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, page 35, 2007.

C. Berge. *Hypergraphs: combinatorics of finite sets*. North holland, 1989.

A. Berman and R. J. Plemmons. *Nonnegative matrices*. SIAM, 1979.

U. Brandes, D. Delling, M. Gaertler, et al. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, 2007.

T. Broderick, M. Jordan, and J. Pitman. Beta processes, stick-breaking and power laws. *Bayesian analysis*, 7(2):439–476, 2012.

J. Bu, S. Tan, C. Chen, C. Wang, H. Wu, L. Zhang, and X. He. Music recommendation by unified hypergraph: combining social media information and music content. In *Proceedings of the international conference on Multimedia*, 2010.

T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is np-hard. *Information Processing Letters*, 42(3):153–159, 1992.

S. R. Bulò and M. Pelillo. A game-theoretic approach to hypergraph clustering. *Advances in Neural Information Processing Systems*, 2009.

D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. *Computer Science Department, CMU*, 2004.

V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

J. Cheeger. A lower bound for the smallest eigenvalue of the laplacian. *Problems in analysis*, 625:195–199, 1970.

Y. Chen, M. Rege, M. Dong, and J. Hua. Non-negative matrix factorization for semi-supervised data clustering. *Knowledge and Information Systems*, 17(3):355–379, 2008.

S. Chojnacki and M. Kłopotek. Random graph generative model for folksonomy network structure approximation. *Procedia Computer Science*, 1(1):1683–1688, 2010.

F. Chung. The laplacian of a hypergraph. *Expanding graphs (DIMACS series)*, pages 21–36, 1993.

F. Chung. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92)*. American Mathematical Society, 1997.

F. Chung. Laplacians and the cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 2005.

A. Clauset, C. R. Shalizi, and M. E. Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

R. R. da Silva. A new generalized field of values. *arXiv preprint arXiv:1012.3835*, 2010.

L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(09), 2005.

K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek. Parallel hypergraph partitioning for scientific computing. In *20th International Parallel and Distributed Processing Symposium*, 2006.

R. Duffin. A minimax theory for overdamped networks. *Journal of Retional Mechanics and Analysis*, 4:221–233, 1955.

M. Dyer and A. Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10(2):139–153, 1985.

M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *ACM SIGCOMM Computer Communication Review*, 29(4):251–262, 1999.

U. Feige, M. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.

M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(2):298–305, 1973.

S. Funk. Netflix update: Try this at home. *http://sifter.org/~simon/journal/20061211.html*, 2006.

L. Getoor. *Introduction to statistical relational learning*. The MIT press, 2007.

L. Getoor and C. P. Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2): 3–12, 2005.

G. H. Golub and C. F. Van Loan. *Matrix computations*. Johns Hopkins University Press, 1996.

J. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proc. of the 10th USENIX conference on Operating Systems Design and Implementation*, 2012a.

J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proc. of the 10th USENIX conference on Operating systems design and implementation*, 2012b.

T. Griffiths and Z. Ghahramani. The indian buffet process: An introduction and review. *Journal of Machine Learning Research*, 12:1185–1224, 2011.

C. Guo and P. Lancaster. Algorithms for hyperbolic quadratic eigenvalue problems. *Mathematics of Computation*, pages 1777–1791, 2005.

E.-H. Han, G. Karypis, V. Kumar, and B. Mobasher. *Clustering based on association rule hypergraphs*. University of Minnesota, Department of Computer Science, 1997.

N. J. Higham, F. Tisseur, and P. M. Van Dooren. Detecting a definite hermitian pair and a hyperbolic or elliptic quadratic eigenvalue problem, and associated nearness problems. *Linear Algebra and its applications*, 351:455–474, 2002.

R. Horn and C. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991.

P. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

D. Johnson. *Computers and Intractability-A Guide to the Theory of NP-Completeness*. Freeman, San Fransisco, CA, 1979.

G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. In *Proceedings of the 34th annual Design Automation Conference*, 1997.

E. Kayaaslan, A. Pinar, Ü. Çatalyürek, and C. Aykanat. Partitioning hypergraphs in scientific computing applications through vertex separators on graphs. *SIAM Journal on Scientific Computing*, 34(2):A970–A992, 2012.

S. Kok and P. Domingos. Learning markov logic network structure via hypergraph lifting. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. of SIGKDD*, 2008.

B. Kulis, S. Basu, I. Dhillon, and R. Mooney. Semi-supervised graph clustering: a kernel approach. In *Proceedings of the 22nd ICML*, 2005.

L. Ladicky, C. Russell, P. Kohli, and P. Torr. Graph cut based inference with co-occurrence statistics. *Proceedings of ECCV*, 2010.

X. N. Lam, T. Vu, T. D. Le, and A. D. Duong. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, 2008.

J. Lang, B. Qin, T. Liu, and S. Li. Unsupervised coreference resolution with hypergraph partitioning. *Computer and Information Science*, 2(4), 2009.

E. L. Lawler. Cutsets and partitions of hypergraphs. *Networks*, 3(3):275–285, 1973.

T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005.

J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

C. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007.

Y.-R. Lin, J. Sun, P. Castro, R. Konuru, H. Sundaram, and A. Kelliher. Metafac: community discovery via relational hypergraph factorization. In *Proceedings of the 15th ACM SIGKDD*, 2009.

W. Liu, J. He, and S. Chang. Large graph construction for scalable semi-supervised learning. In *Proceedings of the 27th ICML*, 2010.

G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146, 2010.

D. Mavroeidis. Mind the eigen-gap, or how to accelerate semi-supervised spectral learning algorithms. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2011.

A. Mithani, G. M. Preston, and J. Hein. Rahnuma: hypergraph-based tool for metabolic pathway prediction and network comparison. *Bioinformatics*, 25(14):1831–1832, 2009.

N. Neubauer and K. Obermayer. Towards community detection in k-partite k-uniform hypergraphs. In *Proceedings of the NIPS 2009 Workshop on Analyzing Networks and Learning with Graphs*, 2009.

M. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23), 2006.

M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2), 2004.

M. E. Newman. Clustering and preferential attachment in growing networks. *Physical Review E*, 64(2), 2001.

M. E. Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

J. Paisley, A. Zaas, C. W. Woods, G. S. Ginsburg, and L. Carin. A stick-breaking construction of the beta process. In *International Conference on Machine Learning*, 2010.

T. Park and A. Van Gelder. Partitioning methods for satisfiability testing on large formulas. *Automated Deduction—Cade-13*, pages 748–762, 1996.

A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. of KDD Cup and Workshop*, 2007.

R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *USENIX conference on NSDI*, 2010.

B. Prakash, A. Sridharan, M. Seshadri, S. Machiraju, and C. Faloutsos. Eigenspokes: Surprising patterns and scalable community chipping in large graphs. *Advances in Knowledge Discovery and Data Mining*, pages 435–448, 2010.

L. Pu and B. Faltings. Hypergraph clustering for better network traffic inspection. In *IJCAI Workshop on Intelligent Security (SecArt)*, 2011.

W. Reed and B. Hughes. Power-law distributions from exponential processes: an explanation for the occurrence of long-tailed distributions in biology and elsewhere. *Scientiae Mathematicae Japonicae*, 58(2):473–483, 2003.

P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3): 56–58, 1997.

E. Rogers. A mimmax theory for overdamped systems. *Archive for Rational Mechanics and Analysis*, 16(2):89–96, 1964.

S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system-a case study. In *Proc. of WebKDD'00*, 2000.

J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. *The Adaptive Web*, pages 291–324, 2007.

P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

M. Seshadri, S. Machiraju, A. Sridharan, J. Bolot, C. Faloutsos, and J. Leskove. Mobile call graphs: beyond power-law and lognormal distributions. In *Proceedings of the 14th ACM SIGKDD*, pages 596–604, 2008.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

S. Siersdorfer and S. Sizov. Social recommender systems for web 2.0 folksonomies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, 2009.

P. Singla and P. Domingos. Entity Resolution with Markov Logic. In *Proceedings of the 6th ICDM*, 2006.

D. A. Spielman and S.-H. Teng. Solving sparse, symmetric, diagonally-dominant linear systems in time o (m< sup> 1.31</sup>. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003.

A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2003.

R. Thibaux and M. Jordan. Hierarchical beta processes and the indian buffet process. In *Proc. of AISTATS*, 2007.

F. Tisseur and K. Meerbergen. The quadratic eigenvalue problem. *Siam Review*, 43(2):235–286, 2001.

C. Wu. On rayleigh-ritz ratios of a generalized laplacian matrix of directed graphs. *Linear algebra and its applications*, 402:207–227, 2005.

L. Wu, X. Ying, X. Wu, and Z. Zhou. Line orthogonality in adjacency eigenspace with application to community partition. In *Proc. of IJCAI*, 2011.

Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

Y. Yang. An evaluation of statistical approaches to text categorization. *Information retrieval*, 1(1-2):69–90, 1999.

D. Zhou, J. Huang, and B. Scholkopf. Beyond pairwise classification and clustering using hypergraphs. *Technical Report*, 2005a.

D. Zhou, J. Huang, and B. Schölkopf. Learning from labeled and unlabeled data on a directed graph. In *Proceedings of the 22nd ICML*, 2005b.

D. Zhou, J. Huang, and B. Scholkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in Neural Information Processing Systems*, 2007.

Z. Zhu, G. Cao, S. Zhu, S. Ranjan, and A. Nucci. A social network based patching scheme for worm containment in cellular networks. *Handbook of Optimization in Complex Networks*, pages 505–533, 2012.

C. Ziegler, S. McNee, J. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proc. of WWW*, 2005.

J. Y. Zien, M. D. Schlag, and P. K. Chan. Multilevel spectral hypergraph partitioning with arbitrary vertex sizes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(9):1389–1399, 1999.

Li PU

Email: li.pu@epfl.ch Tel: (0041) 21 693 6711 Url: http://people.epfl.ch/li.pu
Address: INR240 (Bâtiment INR), Station 14, CH-1015 Lausanne, Switzerland

SUMMARY

My main research interests are relational learning with graph or hypergraph, hypergraph spectral learning, and applications such as recommender system, network traffic inspection, and data visualization.

EDUCATION AND EXPERIENCES

09.2008 - present **PhD candidate**, in the Artificial Intelligence Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

During the PhD, I supervised 6 semester/master projects in EPFL from 2009 to 2012. I developed a Matlab toolbox called HAT for analyzing hypergraphs.

03.2012 – 06.2012 **Research intern**, in Microsoft Research Redmond and bing.com, Redmond

I developed a full-scale prototype of a new search scenario which was integrated into the browser and took the relations between entities into account (like a knowledge graph). This was an award-winning project in an internal event.

02.2010 – 05.2010 **Visiting student**, in the CS department of Hong Kong University of Science and Technology (HKUST), Hong Kong

09.2006 – 07.2008 **M.Sc**, in State Key Laboratory of Intelligent Technology and Systems, Department of CS, Tsinghua University, Beijing, China

During the Master, I studied the parameter estimation problem of hybrid systems.

09.2001 – 06.2005 **B.Eng**, in the School of Mechanical Engineering and Automation, Beihang University, Beijing, China

I was leading a team of 5 members which developed an autopilot system for flying robots. The follow-up project became the first Chinese UAV (unmanned air vehicle) task in the Antarctic expedition.

EMPLOYMENT AND PUBLIC SERVICE

02.2009 – 01.2010 President of the Association of Chinese Scholars and Students in Lausanne

08.2008 – 12.2008 Developed the open-source project DSP_INS

This project is an inertial navigation system on a DSP with MEMS inertial sensors and GPS module. The core of the project is a real-time extended Kalman filter implemented in C.

PUBLICATIONS

Li Pu and Boi Faltings, Hypergraph Learning with Hyperedge Expansion, ECML-PKDD, 2012

Li Pu and Boi Faltings, Hypergraph Clustering for Better Network Traffic Inspection, the 3rd Workshop on Intelligent Security at IJCAI, 2011

Li Pu, Boi Faltings, Qiang Yang and Derek H. Hu, Relational Network-Service Clustering Analysis with Set Evidences, the 3rd Workshop on Artificial Intelligence and Security, 2010

Li Pu, Jinchun Hu, and Badong Chen, Information Theoretical Approach to Identification of Hybrid Systems, the 11th International Conference on Hybrid Systems: Computation and Control, 2008

Badong Chen, Jinchun Hu, Li Pu, and Zengqi Sun, Stochastic Gradient Algorithm under (h, φ)-entropy, *Circuits Systems and Signal Processing*, 26(6), 941-960, 2007

Li Pu, Tianmiao Wang, Jianhong Liang, and Song Wang, An Attitude Estimate Approach using MEMS Sensors for Small UAVs, the 4th IEEE International Conference on Industrial Informatics, 2006

LANGUAGES

English: fluent
Chinese (mandarin): native speaker
French: basic knowledge (A2/B1 level of EU reference card)