Learning on Graphs

FULI FENG
(Bachelor of Computer Science and Engineering, Beihang University)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2019

Supervisor:
Professor Tat-Seng Chua

Examiners:
Associate Professor Min-Yen Kan
Dr Wei Wang
Professor Jimmy Xiangji Huang, York University

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

冯福利

FULI FENG

5 May 2019

ACKNOWLEDGEMENTS

First and foremost, I sincerely appreciate my supervisor, Prof Tat-Seng Chua, for his kind support and guidance in my Ph.D. journey. Since joining his research group in 2015, Prof Chua has been giving me precious thoughts and suggestions, teaching me all the skills of doing research. In addition to research, Prof Chua encouraged his students to regularly go jogging and join marathon competition, which would be a great lifelong habit benefiting my physical and mental health. Furthermore, Prof Chua has demonstrated how to be a perfect gentleman that is worth learning with the rest of my life. I have to say, it's way more than my honor to be his student.

Secondly, I would like to thank the NExT++ center, including all the responsible administrators, brilliant researchers, and other talented students. It's an enjoyable experience to be a member of NExT++ where everybody kindly supports each other in both research and daily life like a family. Within the NExT++ family, I want to especially express my heartfelt gratitude to my mentor, Dr. Liqiang Nie and Dr. Xiangnan He, who have been providing valuable suggestions to my research, including thinking, writing, and presenting. In addition, they have been influencing me in many other ways through countless details such as communication skills and time management.

Moreover, I would like to thank my thesis committee members: *To-Be-Updated* for their time and valuable comments on my thesis, which significantly help to complete the thesis.

Lastly, I have to sincerely thank my parents and my sister. Their selfless love and support are the greatest gifts in my world.

CONTENTS

ABSTRACT

Learning on graphs (*a.k.a.* graph-based learning) mainly aims to analyze the property of entities (*e.g.,* predict entity attributes) from graphs where entities and entity relations are represented as nodes[1] and edges, respectively. Graph-based learning plays a crucial role in a variety of emerging applications in disciplines including physics, biology, chemistry, social and information science. For instance, in social science, graph-based learning is a typical solution for social network applications such as targeted advertising and recommendation.

Most of the existing graph-based learning methods mainly model graph structure based on a *local smoothness assumption*, that is closely connected nodes have similar predictions. Specifically, this assumption is typically implemented via either a *graph Laplacian regularization* or node embedding diffussion over the graph structure. However, in addition to the graph structure, these implementations of local smoothness largely ignore the rich information in graph applications such as various edge attributes, dynamic vertex features, and rich domain knowledge, which contain additional clues for local smoothness.

In this thesis, we investigate techniques to thoroughly mine the graph data so as to enhance the modeling of local smoothness. In particular, 1) we devise a multi-relation learning framework which improves the modeling of local smoothness by jointly considering multiple types of relations between vertices. 2) We design a new regularization term to encode domain knowledge which could guide the local smoothness modeling. 3) We propose a new neural network operator which adaptively adjusts the strength of smoothness between vertices in a time-aware manner according to the dynamic features of vertices. 4) We develop a new adversarial training approach which aims to enhance the robustness of local smoothness modeling by additionally performing adversarial perturbations on vertex features.

The proposed methods leverage different types of additional information to enhance local smoothness modeling suitable for different kinds of graph applications. Therefore, we apply the methods on different applications to conduct experiments, in particular, 1) university ranking (multiple relations), 2) popularity prediction (domain knowledge), 3) stock ranking (dynamic vertex features), and 4) conventional node classification applications (adversarial perturbations). Extensive experiments demonstrate the effectiveness of the proposed methods and validate the necessity of jointly encoding graph structure along with additional information for local smoothness modeling.

[1]Node is also widely termed as vertex. In the following, we interchangeably use node and vertex.

LIST OF TABLES

LIST OF FIGURES

LIST OF PUBLICATIONS

- **Fuli Feng**, Liqiang Nie, Xiang Wang, Richang Hong, and Tat-Seng Chua. *Computational Social Indicators: a Case Study of Chinese University Ranking.* In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'17) pages 455–464, 2017

- **Fuli Feng**, Xiangnan He, Yiqun Liu, Liqiang Nie, and Tat-Seng Chua. *Learning on Partial-order Hypergraphs.* In Proceedings of the 2018 World Wide Web Conference on World Wide Web (WWW'18), pages 1523–1532, 2018

- **Fuli Feng**, Xiangnan He, Xiang Wang, Cheng Luo, Yiqun Liu, and Tat-Seng Chua. *Temporal Relational Ranking for Stock Prediction.* ACM Transactions on Information Systems (TOIS'19), volume 37 pages 2:1–2:27, 2019

- **Fuli Feng**, Xiangnan He, Jie Tang, and Tat-Seng Chua. *Graph Adversarial Training: Dynamically Regularizing Based on Graph Structure.* IEEE Transactions on Knowledge and Data Engineering (*Major revision*), 2019

Chapter 1

Introduction

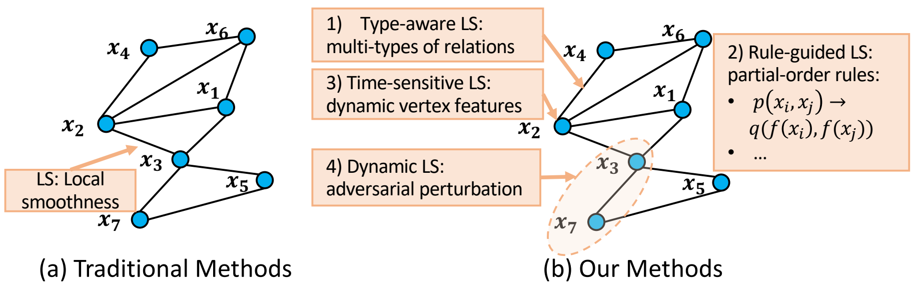
Graphs are natural representations of relational data where vertices and edges represent entities of interest and relations between them, respectively [39]. In the era of big data, graphs are playing more crucial roles in modeling the relations and processes in physical [12], biological [52], chemical [46], social [66], and information systems [89]. For instance, graphs can be used to represent atomic structures, molecule, protein-protein interactions, user friendships, and bank transaction flows. With graph representations of entities and their relations in these disciplines, analysis of graphs is crucial in a variety of associated applications. A large portion of these applications forms a main graph-oriented task, *vertex prediction* [89, 49, 48]. The target of vertex prediction is to forecast the attributes of the vertices which could be either categorical (*e.g.,* interests and demography of social network users) or numerical (*e.g.,* income and credit score of the users).

Learning on graphs (*a.k.a.* graph-based learning) tackles the vertex prediction task as a machine learning problem with graphs as inputs. As compared to standard classification solutions that mainly encode vertex features, graph-based learning additionally incorporates graph structure into the prediction. The existing graph-based learning approaches mainly encodes the graph structure based on a *local smoothness assumption*, *i.e.,* connected vertices tend to have similar predictions. Traditional approaches [195, 189] implement the assumption through a *graph Laplacian regularization* which encourages closely connected vertices to be predicted with similar labels. More recently, there has been a surge of *graph representation learning* approaches [127, 147, 63, 176, 37, 89, 154] that map vertices as points in a low-dimensional embedding space. Generally, the representation learning approaches achieve the local smoothness assumption via forcing connected vertices to be geometrically close in the embedding space.

We argue that the core of graph-based learning is the joint modeling of graph structure (*i.e.,* vertex connectedness) and additional information, *e.g.,* edge attributes, vertex features, and domain knowledge, to accurately model the local smoothness and comprehensively describe the entities. The existing methods mainly focus on vertex connectedness, but pay less attention on the additional information that indicates the property of connectedness. As such, the existing methods might not be suitable for graph applications with complex practical scenario and additional information from both edge and vertex perspectives. For instance:

- *Multiple types of relations* (edge perspective). In a real-world social media like Twitter and Facebook, users would interact with each other via multiple types of actions including follow, like, reply, and repost. Considering that different interactions reflect different kinds of connections, existing methods that intuitively focus on a single interaction or neglect the difference between interactions would not comprehensively model the local smoothness. As such, they would be suboptimal for predicting user attributes, which motivates the consideration of multiple relations in graph-based learning.

- *Graph-based domain knowledge* (edge perspective). Insufficient labeled data and frequent cold-start entities are common in many graph applications, which gives rise to requirement of models with strong generalization. In standard classification, incorporating domain knowledge into the data-driven learning framework could complement the training data and lead to models with better generalization [76, 80, 140]. Graph-based domain knowledge, which typically involves more than one vertex, widely exists in many graph applications such as product recommendation [165] and text retrieval [172]. Therefore, incorporating domain knowledge into graph-based learning methods would be desirable to enhance the modeling of local smoothness and lead to better



x_4
x_6
x_1
x_2
x_3
x_5
x_7
LS: Local smoothness
(a) Traditional Methods
1) Type-aware LS: multi-types of relations
3) Time-sensitive LS: dynamic vertex features
4) Dynamic LS: adversarial perturbation
2) Rule-guided LS: partial-order rules:
• $p(x_i, x_j) \rightarrow q(f(x_i), f(x_j))$
• …
(b) Our Methods

Figure 1.1: Intuitive illustration of the position of our methods as compared to general graph-based learning ones.

prediction performance.

- *Dynamic features* (vertex perspective). Dynamic features are the attributes of vertices which naturally varies over time. For instance, in a product graph with substitutable and complementary relations [166], the price of product is a dynamic feature. We argue that the strength of vertex connections might change along with the update of vertex features. For instance, a substitutable relation could become weak if the price of one connected product increases a lot. Obviously, lacking of accounting for the temporal property of vertex features by ignoring the changes on connection strength would lead to inaccurate modeling of local smoothness. As such, the existing approaches would be less effective for applications like sales prediction and advertising.

- *Adversarial perturbations* (vertex perspective). Adversarial perturbations are also performed on vertex features, resulting in feature changes. Unlike dynamic features which change according to the inherent status of the vertex, adversarial perturbations are intentionally performed to fool the graph-based learning model. In many graph applications, driven by profit, such adversarial perturbations could frequently be performed by crackers from the underground economy even end-users. For instance, in a recommender system, a merchant would intentionally change the description of the product so that it would be more frequently recommended to consumers than its competitor. Previous research has shown that graph-based learning methods, especially the recent graph representation learning ones, are vulnerable to such perturbations [197]. Hence, it is essential to incorporate adversarial perturbations in graph-based learning.

In this thesis, we investigate techniques to enhance the modeling of local smoothness and improve graph-based learning by considering information in addition to connectedness, including multiple types of relations, graph-based domain knowledge, dynamic vertex features, and adversarial vertex perturbations. In particular,

- *Type-aware local smoothness.* To better solve the graph-oriented task, we account for multiple types of relations in the modeling of local smoothness, that is, modeling local smoothness in a type-aware manner. In some graph applications, it is non-trivial to appropriately model the smoothness/similarity between entities with multiple types of relations. This is mainly because the various semantic meanings across different relations and the complex connections across different relations. For instance, in a social network, users have relations based on follow, like, reply and *etc.*, which could reflect different affinities between users. Moreover, different types of relations are connected with each other, *e.g.,* it is unlikely that a like relation exists between users without a follow relation. Therefore, the key to accurately modeling type-aware local smoothness is to appropriately model the property of different relations and the connections across relations. Towards this end, we propose a new graph Laplacian-based learning framework which encodes multiple relations between entities and captures the correlation between different types of relations.

- *Rule-guided local smoothness.* While accounting for graph-based domain knowledge in the modeling of local smoothness can result in a better model, it is insufficient to directly employ the existing methods in standard machine learning tasks [76, 80, 140]. The reasons are two-fold: 1) the knowledge in the vertex prediction task could be in a variety of formats involving single vertex, a pair of vertices, or even a group of vertices. It is more complex than standard learning tasks where knowledge typically involves only one entity. 2) Different graph-based learning approaches implement local smoothness in different manners, leading to challenges in devising a general solution to incorporate knowledge. By analyzing some typical vertex prediction applications, we find that domain knowledge in such applications can typically be represented as *partial-order rules.* One partial-order rule includes one kind of partial-order relation [44] on the vertex set and a pair-wise rule for making prediction. For instance, a video posted by a more influential user (*e.g.,* has more followers) tends to be more popular. To encode partial-order rules in a general manner, we design a regularization term which can be additionally applied to most existing graph-based learning approaches.

- *Time-sensitive local smoothness.* Despite that vertex connections (*i.e.,* with/without edges) are static in many applications, the strength of connections might vary over time. As such, it is desirable to encode the temporal property of vertex connections at different timestamps in local smoothness modeling, instead of treating them as static. The key challenge of dynamically adjusting the local smoothness in a time-sensitive manner is the implicitness of the strength of connection between vertices. Dynamic vertex features could reflect the status of the entities and potentially indicate the temporal property of the connection between entities. For instance, in a graph of a supplychain, the connection between a supplier and a consumer would become stronger when the consumer is launching its new product, which could be inferred from the dynamic cashflow feature of the consumer. As such, we incorporate dynamic vertex features into the modeling of local smoothness to capture the temporal property of vertex connections. In particular, we devise a new operator for graph neural network, named *Temporal Graph Convolution*, which learns vertex representations in a time-sensitive manner.

- *Dynamic local smoothness.* To defend against adversarial perturbations, there is a strong need to stabilize graph-based learning models, especially neural network-based models, *i.e.,* the model should not change the predictions much when perturbation happens. *Adversarial Training* (AT) has empirically shown to be able to stabilize neural networks, enhancing their robustness against perturbations in standard classification tasks [93, 111]. Specifically, it can be seen as a *dynamic regularization* technique that proactively simulates perturbations during the training phase [60]. Therefore, we believe that an equivalent of AT on a graph neural network model would also be helpful to the model's robustness. However, directly employing AT on graph neural network is insufficient, since it treats vertices as independent of each other and does not consider the impact from connected vertices. To tackle such limitation, we propose to explore graph adversarial training techniques which is a form of dynamic local smoothness that can resist perturbations from connected vertices.

In Figure 1.1, we intuitively depict the position of our techniques as compared to general graph-based learning approaches. As shown in Figure 1.1(a), most existing graph-based learning methods model local smoothness mainly via considering the connectedness between vertices. In contrast, as shown in Figure 1.1(b), our methods further leverages information from both edge perspective and vertex perspective in addition to the connectedness. On one hand, from the edge perspective, we enhance the modeling of local smoothness by additionally considering: 1) multi-type vertex relations; and 2) partial-order rules. On the other hand, we encode additional vertex information: 3) dynamic vertex features, and 4) adversarial perturbations to comprehensively model local smoothness.

Outline: The remainder of this thesis is organized as follows. We first review the literature on graph-based learning in Chapter 2. We describe our first work on multi-relation learning towards the computation of social indicators in Chapter 3. In Chapter 4, we present the second work of learning on partial-order hypergraphs, followed by introducing the third work on temporal graph-based learning for stock prediction in Chapter 5. Finally, we introduce the dynamic local smoothness modeling through graph adversarial training in Chapter 6, followed by the conclusion and future works for this thesis in Chapter 7.

Chapter 2

Literature Review

In this chapter, we first summarize and compare a variety of graphs with different properties to represent different kinds of relational data. We then review the prior efforts on graph-based learning, which has attracted a lot of attention in recent decades. We can roughly divide the methods into two groups: *graph Laplacian regularization* and *graph representation learning*. In this chapter, we mainly discuss graph-based learning approaches since this thesis focuses on the learning methods. For work on using these methods to different applications, we refer the readers to some recent surveys [34, 67, 182, 192, 186]. Moreover, we summarize the latest research on the applications used to test our proposed methods in the following four chapters, respectively.

We first introduce some notations used in this thesis. We use bold capital letters (*e.g.,* $\mathbf{X}$), bold lowercase letters (*e.g.,* $\mathbf{x}$), and capital script letters (*e.g.,* $\mathcal{X}$) to denote matrices, vectors, and tensors, respectively. Scalars and hyperparameters are respectively represented as normal lowercase letters (*e.g.,* x) and Greek letters (*e.g.,* λ). In addition, $tr(\mathbf{X})$ denotes the trace of $\mathbf{X}$. If not otherwise specified, all vectors are in a column form, and X_{ij} denotes the entry at the i-th row and the j-th column of $\mathbf{X}$. The symbols σ, $tanh$, and $\odot$ stand for the *sigmoid* function, hyperbolic tangent function, and element-wise product operation, respectively.

2.1 Graphs

If not otherwise specified, graphs mean simple graphs, detailed as $G = (\mathcal{V}, \mathcal{E})$, comprise of a set of N vertices ($\mathcal{V}$) representing entities of interested and a set of edges ($\mathcal{E}$) representing relations between entity pairs [39]. The edges could be either directed or undirected depending on whether the pairwise relationship between entities is symmetric. For example, a modulate graph representing its structure is an undirected graph. The World Wide Web is a well-known instance of directed graphs, where vertices represent webpages, and edges represent hyperlinks. Formally, a simple graph is typically represented as an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ with binary entries where $\mathbf{A}_{ij} = 1$ means there is an edge between i and j. Simple graph is the simplest format of graph data where only vertex connectedness is encoded.

To enhance representation ability, various generalizations of the simple graph have been proposed. One generalization of simple graph is hypergraph where edges (*a.k.a.* hyperedges) connect a set of vertices, enabling it to represent higher-order relations among vertices [190]. Another generalization is the attributed graph in which each vertex is associated with a vector $\mathbf{x}_i$ comprising of a set of vertex attributes [128]. By incorporating vertex features, an attributed graph naturally enhances the representation ability of simple graph and is more suitable for tasks in which vertex features play crucial roles. Naturally, another direction to extend simple graph is to associate edge attributes such as weights denoting the strength of connection between associated vertices (weighted graph) or type of connections (a well-known instance is a knowledge graph [20]). A special generalization of the simple graph is heterogeneous graph [144] which allows vertices to be heterogeneous. In other words, a heterogeneous graph explicitly represents different types of entities (*e.g.,* users and products) via different types of vertices rather than encode entity types as vertex attributes.

Different formats of graph contain different information. Although more complex graph contains more information, it also costs more storage and computation resources in practical applications. As such, suitable graph should be selected according to the requirement of an application. In Table 2.1, we summarize the

Table 2.1: Basic properties of different categories of graphs.

Graphs	Direction	Order	Vertex Attributes	Edge Attributes	Instance
Simple Graph	D/U	Pairwise	N	N	World Wide Web
Hypergraph	U	Higher Order	N	N	Modulate Reaction Graph
Attributed Graph	D/U	Pairwise	Y	Y/N	Citation Graph
Edge-attributed Graph	D/U	Pairwise	Y/N	Y	Knowledge Graph
Heterogeneous Graph	D/U	Pairwise	Y/N	Y/N	Bibliographic Graph

D and U means directed and undirected respectively. Y (yes) and N (no) indicates whether the corresponding graph contains vertex/edge attributes. For instance, World Wide Web is a simple graph of webpages connected by hyperlinks; modulate reaction graph is a hypergraph of modulates where hyperedges connect a set of modulates in a chemical formula. In the citation graph, vertices and edges are articles and references respectively. A knowledge graph consists of entities like earth and sun as well as edges representing the relations between entities in the real-world. In the bibliographic graph, the heterogeneous vertices include authors, articles, affiliations, topics, *etc.*.

properties of the aforementioned graphs to compare their differences. Among this graphs, attributed graph is the most widely used one in both practical applications and graph-based learning study [34, 67, 182, 192, 186]. In the following, if not otherwise specified, we refer to graphs as attributed graphs, *i.e.,* each vertex is associated with a feature vector. While designing new graphs with better representation ability is still remarkably attractive, this thesis focuses on learning methods on graphs.

2.2 Learning on Graphs

In recent decades, attention on analyzing graphs mainly focuses on three tasks:

- *Vertex prediction* [89, 49, 48]: As aforementioned, the target of vertex prediction is to predict the of interest attribute of vertices in a given graph.
- *Link prediction* [71, 17, 177]: Instead of focusing on vertex, link prediction aims to infer whether there would be an edge between a pair of vertices.
- *Clustering* [38, 29, 163]: The target of clustering is to identify group of vertices with similar property.

Among the three tasks, vertex prediction has been attracting the most attention on account of its wide range of applications, which is the focus of this thesis.

In the last decade, a surge of graph-based learning methods have been proposed to solve the vertex prediction task. Technically, the existing approaches fall into two main categories: *graph Laplacian regularization* and *graph representation learning*. Traditional approaches including label propagation [195] and graph spectral methods [189] rely mainly on *graph Laplacian regularization* which implements the assumption via encouraging closely connected vertices to be predicted with similar labels. More recently, there has been a surge of *representation learning* approaches that encode graph structure by mapping vertices as points in a low-dimensional embedding space. Generally, the representation learning approaches achieve local smoothness assumption by forcing the connected vertices to be geometrically close with a variety of techniques including skip-gram [127, 147, 63, 176] and graph convolution [37, 89, 154].

2.3 Graph Laplacian Regularization

The general problem setting of vertex prediction is to learn a prediction function $\hat{\mathbf{y}} = f(\mathbf{x})$, which maps a vertex from the feature space to the target label space [176]. Existing approaches under the graph Laplacian category usually solve the problem by minimizing an objective function abstracted as [189]:

$$\Gamma = \Omega + \lambda\Phi, \tag{2.1}$$

where $\Omega = \sum_{i=1}^{N} l(f(\mathbf{x}_i), \mathbf{y}_i)$ is a task-specific loss that measures the error between prediction $\hat{\mathbf{y}}$ and ground-truth $\mathbf{y}$. With categorical labels, l would be a classification loss such as cross-entropy loss and hinge loss. With numerical targets, a well-known instance of l could be mean square loss. Φ is a graph Laplacian regularization term implementing the *local smoothness* assumption and λ is a hyperparameter to balance the two terms. Formally, Φ is defined as:

$$\Phi = \sum_{i=1}^{N}\sum_{j=1}^{N} \underbrace{\mathbf{A}_{ij}}_{\text{strength of smoothness}} \underbrace{\left\| \frac{f(\mathbf{x_i})}{\sqrt{D_{ii}}} - \frac{f(\mathbf{x_j})}{\sqrt{D_{jj}}} \right\|^2}_{\text{smoothness}}, \tag{2.2}$$

where $\mathbf{A}_{ij}$ could be either the similarity score between a pair of vertices (weighted graph) or a binary value denoting whether vertices i and j are connected (simple graph); $D_{ii} = \sum_{j=1}^{N} \mathbf{A}_{ij}$ is the degree of vertex i [189]. The regularization term operates smoothness on each pair of entities, enforcing their predictions (after normalized by their degrees $\frac{1}{\sqrt{D_{ii}}}$) to be close to each other. The strength of smoothness is determined by the similarity or connectedness $\mathbf{A}_{ij}$. In other words, it encourages similar nodes or connected nodes to be predicted with similar predictions. It can be equivalently written in a more concise matrix form:

$$\mathcal{G} = trace(\hat{\mathbf{Y}}\mathbf{L}\hat{\mathbf{Y}}^T), \tag{2.3}$$

where $\hat{\mathbf{Y}} = [\hat{\mathbf{y_1}}, \hat{\mathbf{y_2}}, \cdots, \hat{\mathbf{y_N}}]$, $\mathbf{L}$ is defined as $\mathbf{L} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-1/2}$, also known as the *graph Laplacian matrix.*

With the aforementioned formulation, research on graph Laplacian methods has been mainly focusing on: *definition of prediction function* (f) and *calculation of Laplacian matrix* ($\mathbf{L}$).

- Various prediction functions have been proposed. Approaches like label propagation [195, 196], learning with local and global consistency [189] and Modified Adsorption [146], define f as a label lookup table for unlabeled instances in the graph. That is to say, the prediction function is non-parameterized, making predictions purely from graph structure without consideration of vertex features. In particular, label propagation [195, 196] forces f to agree with labeled instances. Other approaches [189, 146] allow the prediction on labeled instances to vary and incorporate vertex uncertainty. To account for vertex features, manifold regularization [14] parameterizes f in the Reproducing Kernel Hilbert Space (RKHS).

- An alternative line of research is to explore the calculation of Laplacian matrix, mainly focusing on the measure of vertex similarity (*i.e.,* definition of $\mathbf{A}_{ij}$). By comparing neighbors of a vertex pair, *Jaccard similarity* and its variations are first proposed to measure vertex similarity [121, 131, 57, 95]. Generally speaking, vertices with more common neighbors would obtain higher Jaccard similarity. Besides one-hop neighbors, vertex graph kernel measures incorporate multi-hop neighbors by considering the paths connecting the vertex pair [22, 5, 99]. In addition, Zhou *et al.* [190] extended the philosophy of Laplacian into hypergraph by defining $\mathbf{A}_{ij}$ as the number of hyperedges connecting vertices i and j.

Approaches based on graph Laplacian regularization have been applied on a wide range of applications and achieved significant success. However, inspired by the extraordinary representation ability of deep neural networks and their remarkable success in solving standard learning problems, the community has shifted the focus on deep graph-based learning where the graph Laplacian (*a.k.a.* Laplacian eigenmaps) techniques typically play auxiliary roles. For instance, some deep graph-based learning approaches use graph Laplacian regularization to enforce first-order similarity [13, 168]. Moreover, vertex graph kernels are applied to extract vertex features so as to encode higher-order similarity [27, 123].

2.4 Graph Representation Learning

Representation learning, especially the approaches based on deep neural networks owing to their extraordinary ability of non-linear modeling, has achieved success in various tasks [109, 75, 67]. The target of representation learning is to map the entities from either the feature space or IDs to points in a low-dimensional embedding space, *i.e.,* learning a low dimensional vector representation for each entity [75]. In graph representation learning (*a.k.a.* network embedding), the additional target is on preserving the graph structure in the embedding space [67]. In other words, the graph representation learning methods implement the local smoothness by forcing closely connected vertices to be geometrically close with a variety of techniques: 1) *matrix factorization*, 2) *skip-gram*, 3) *autoencoder*, and 4) *graph convolution.*

2.4.1 Matrix Factorization

Early methods for graph representation learning are largely based on matrix-factorization [13, 91], which learns vertex embeddings by factorizing a matrix $\hat{\mathbf{A}}$ corresponding to graph structure. The existing approaches typically solve the problem by minimizing an objective function abstracted as:

$$\Gamma = \sum_{(i,j)\in\mathcal{D}} \|g(\mathbf{z}_i, \mathbf{z}_j) - \hat{\mathbf{A}}_{ij}\|_2^2, \tag{2.4}$$

where g is a function to estimate the "similarity" of vertices i and j, $\mathbf{z}_i$ and $\mathbf{z}_j$ are the learned vertex embeddings. By minimizing the objective function, the embeddings of similar vertices are encouraged to be close. Note that $\mathcal{D}$ does not contain all vertex pairs since $\hat{\mathbf{A}}$ is highly sparse and would bias the embeddings to be zeros if optimized on the whole vertex pair set.

Various methods define different functions of g and the calculations of $\hat{\mathbf{A}}_{ij}$ as variants of Equation (2.4). There is a large number of recent embedding methodologies [4, 27, 123] implemented as inner-product:

$$g(\mathbf{z}_i, \mathbf{z}_j) = \mathbf{z}_i^T \mathbf{z}_j, \tag{2.5}$$

and they differ primarily in the calculation of $\hat{\mathbf{A}}_{ij}$. For example, the Graph Factorization algorithm defines $\hat{\mathbf{A}}_{ij}$ directly based on the adjacency matrix (*i.e.,* $\hat{\mathbf{A}}_{ij} = \mathbf{A}_{ij}$) [4]; GraRep considers various powers of the adjacency matrix (*e.g.,* $\hat{\mathbf{A}}_{ij} = \mathbf{A}_{ij}^2$) in order to capture higher-order vertex similarity [27]; and HOPE supports general similarity measures and vertex graph kernels (*e.g., Jaccard similarity*) [123]. While these different similarity definitions exploit the trade-off between modeling "first-order similarity" and "higher-order similarity", their similarities are highly correlated to the local structure of the graph, making them appropriate for modeling local smoothness.

Another line of research encourages the similarity with the idea of Laplacian regularization (*a.k.a.* Laplacian eigenmaps technique) [13, 168]. Similar as Equation (2.2), the objective function is defined as:

$$\Gamma = \sum_{(i,j)\in\mathcal{D}} \hat{\mathbf{A}}_{ij} \cdot \|\mathbf{z}_i - \mathbf{z}_j\|_2^2. \tag{2.6}$$

2.4.2 Skip-gram

Inspired by the skip-gram model [109], which is proposed to learn word embeddings from large-scale documents, many recent methods learn the vertex embeddings based on a large scale of vertex sequences generated by random walk. The key idea behind this approach is to encourage vertices co-occurring in short random walk sequences to have similar embeddings. Most of the existing methodologies in this line implement the idea by optimizing a classifier in which the aim is to predict the co-occurring vertex (positive/target vertex) of a given vertex (context vertex). Note that the skip-gram methods indirectly implement the local smoothness assumption since local vertices tend to co-occur in random walk sequences with higher probability and their embeddings are forced to be correlated by the classifier. DeepWalk [127] and node2vec [63] are the prior works of skip-gram methods.

To train the classifier efficiently, "negative sampling" is widely used during the training phase, which pairs negative vertices for a context vertex. As the learned embedding is highly sensitive to the sampling strategy, a line of work has been focusing on exploring new sampling techniques. In particular, recent work [19] uses vertex-anchored sampling as an alternative of the random sampler in vertex2vec, which incorporates the connectedness to the context vertex during the sampling. More recently, dynamic negative sampling scheme is adopted, which adaptively selects "hard" negative samples (*i.e.,* similar to the context vertex) to boost the training process [177, 23].

2.4.3 Autoencoder

Inspired by the success of Autoencoder (AE) in learning embedding of given entities from original features, various works use AE to learn vertex embedding [148, 157, 88]. By optimizing a re-construction error, the AE forces vertices with similar features to be close in the embedding space. Considering that the locally connected vertices tend to have similar features regarding graph structure, the Autoencoder-based representation learning approaches actually also implements the local smoothness assumption.

Sparse Autoencoder (SAE) [148] is the first work in this line of research, which takes each column of the adjacency matrix as vertex features. As vertices sharing more common neighbors tends to have similar vertex features, the embeddings learned by SAE mainly reserves the second-order similarity of vertices. Structure Deep Network Embedding SDNE [157] enhances SAE by incorporating the first-order similarity with the Laplacian eigenmaps technique. Furthermore, DNGR [28] encodes higher-order similarity via employing the positive pointwise mutual information between vertices as the features. As a natural extension of AE, recently, Variational Autoencoder (VAE) has also been introduced to learn vertex embeddings [88, 194], leading to more robust embeddings owing to the inherent generation model in VAE.

2.4.4 Graph Convolution

Convolution is a mathematical operation on two functions (*e.g.,* input features and a parameterized filter) to produce a third function that expresses how the shape of one is modified by the other. In computer vision, convolution neural networks have been proven to be remarkably effective to capture the local patterns in small windows. Inspired by the success, there has been attention on extending convolution to the non-Euclidean grid data, especially graphs. A line of research defines graph convolution as the propagation of vertex embeddings over the graph structure, which can be abstracted as:

$$\mathbf{H}^{o} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{in}\mathbf{W}), \tag{2.7}$$

where σ is a non-linear activation function such as the *sigmoid* function. $\mathbf{H}^{in} \in \mathbb{R}^{N\times D}$ is the input of the convolution layer which could be either the vertex features or the output of a previous convolution layer. $\mathbf{W}$ is the parameters of the convolution, modeling the interactions between different dimensions of the input. $\hat{\mathbf{A}}$ is the matrix representing vertex similarities where a non-zero entry $\hat{\mathbf{A}}_{ij}$ means a propagation from j to i.

Various approaches define different $\hat{\mathbf{A}}$ as specific variants of graph convolution. In particular, GCN [89] and GraphSAGE [66] defines $\hat{\mathbf{A}}$ as the normalized adjacency matrix with self-connections, emphasizing first-order similarity. Similarly, GAT [154] considers first-order similarity by modeling the similarity with a multi-head attention model. To encode higher-order similarity, ChebNet [37] and DCNN adopts k-th power of the adjacency matrix and a diffusion matrix denoting the transition probability of a length k diffusion process, respectively. Note that the existing methods largely propagate the embedding between connected vertices, affecting their embeddings to be similar and this implicitly implementing the local smoothness assumption.

2.5 Discussion

It is clear that graph-based learning has made noticeable progress in recent years. However, most of the existing methods focus on the simplest problem setting where vertices are connected with a homogeneous relation and associated with static features. That is to say the existing approaches mainly leverage the connection between vertices and ignore the other information in graph data and graph applications. While most of the graph-based learning applications can be simplified to this general setting, we argue that the existing approaches can be suboptimal for the applications with more complex inputs. In particular, such general solutions lack the ability to handle the more complex cases with multiple relations, dynamic vertex features, the need to incorporate domain knowledge and learning with adversarial perturbations, as discussed in the introduction. In this thesis, we will explore graph-based learning methods that can handle these complex cases.

Chapter 3

Multi-relation (channel) Learning towards Social Indicator Computation

In this chapter, we introduce a new graph-based learning framework that incorporates multiple types of entity relations into the modeling of local smoothness over graph. We demonstrate the framework with a social indicator computation application, university ranking. The target of university ranking is to learn a ranking list of the universities in an unsupervised manner by judging the quality of universities. The key of unsupervised university ranking is adjusting the relative rank of universities from a historical ranking list according to university similarities. Between universities, we have similarity relations from different perspectives, which are calculated with data from different channels, *e.g.,* official, academic, and social channels.

3.1 Introduction

Social indicators are defined as statistical measures and analytics that describe social trends and conditions that would impact social well-being [51]. A social indicator is usually in the form of a ranking list that orders the entities of interests according to some pre-defined rules. In the past few decades, several professional organizations, such as mass media, academic institutes, and government agencies, have calculated and released a wide variety of social indicators on different facets of our society, including cost of living [24], health expenditure [56], happiness index [153], and university quality [86]. Generally, social indicators have some key functions, spanning from providing information for decision-makers, monitoring and evaluating policies, to searching for a common good [8]. For instance, university ranking plays a pivotal role in assessing the quality of universities to help government in evaluating education and research policies, and potential students in selecting universities. Therefore, the accuracy and timely creation of these indicators are extremely useful to a wide variety of users and applications, including the formulation of government policies and planning of social services.

Most of the released social indicators are typically computed in two steps: given a set of entities to be ranked, they first calculate the scores of these entities according to several factors related to the desired social indicator and then fuse the scores using hand-crafted weights to rank the entities. Typically, such computation process suffers from the following problems: 1) Labor-intensive data collection. Data used to calculate social indicators usually rely heavily on user studies like questionnaire, especially, for subjective factors, such as the academic and employment reputation in the QS Ranking. It thus requires a lot of human resources and the collected data can hardly be applied to compute other social indicators. 2) Data insufficiency. Existing social indicators usually only cover a small fraction of target entities. For example, there are 2,553 universities in China[1], while most university rankings involve only less than 800 universities. This is because it is non-trivial to carry out a large-scale user study to gather comprehensive information for each target entity. And 3) expert-relied data fusion. Factor weighting policies rely heavily on experts and different weighting policies may lead to distinct social indicator results. Although we believe that we can find outstanding experts and generate reasonable social indicators, the process is extremely resource-consuming.

With the fast development of Internet, we are able to collect large-scale

[1]http://tinyurl.com/zcbumn3/.

and multi-facet data to describe almost any given entities from the Web, such as interactions and opinions shared in social networking services (SNSs), timely news reports in online mass media, and purchase history in e-commerce platforms. In a sense, the publicly accessible online data enable us to alleviate the aforementioned data collection bottleneck and the data scarcity problems, thus saving human labors. Considering the university ranking as an example, rich data from multiple channels can be gathered to comprehensively describe each university: 1) official statistics about students and teachers are available in platforms of the Ministry of Education (MOE) and various educational organizations; 2) important events related to universities are updated on the website of mass media in real time; 3) academic records are accessible through online bibliographic database like Microsoft Academic[2]; 4) employment status of graduate students are shared in business and employment-oriented SNSs, such as LinkedIn[3]; and 5) university-related comments and opinions from general users are shared in the mainstream social media like Twitter[4].

In this work, we propose to calculate social indicators in a data-driven manner by solving it as an unsupervised ranking task [2]. The target is to learn a model that adjusts the ranking score of the entities from a historical ranking by inferring their similarity relationships [193, 31, 26]. For instance, the model would ranks a pair of entities closer as compared to the historical ranking, if their similarity becomes high. In particular, we first collect multi-channel Web data corresponding to the given social indicator and extract a set of features from each channel to represent the candidates. For each channel, we construct graphs from its features to represent the similarities from the associated perspective, respectively. Note that graphs from different channels represent different types of similarity relations. With graph representations of the entities, graph-based learning methods [193, 31, 26] are the promising solutions for social indicator computation—predicting a ranking score for each vertex.

Empirical study [26, 47, 55] shows that properly fusing similarity relations in different types (from different channels) would be the key to success of such graph-based learning methods. Roughly, the existing methods fall into the

[2]https://academic.microsoft.com/.
[3]https://www.linkedin.com/.
[4]https://twitter.com/.

following two categories: 1) early fusion [26], which aggregates graphs from different channels into a general one before feeding it into graph-based learning models; 2) late fusion [47], which separately learns a ranking list from each channel and then aggregates them into a general one. Early fusion and late fusion are suitable for different scenarios. For instance, early fusion might perform well for graphs with noisy edges, which highlights the consistent parts across graphs to eliminate noises. On the contrary, late fusion might work well for graphs with complement structures. As such, neither early fusion nor late fusion would generate optimal results for social indicator computation because of: 1) Data heterogeneity. Data from different channels have different quality. For instance, an official channel might have less noise than a general user channel. 2) Complex channel relations. The correlation may be strong among some channels, while it may be very weak among others. Therefore, it will cause information loss if all channels are equivalently treated.

To leverage the advantages of both early fusion and late fusion, we propose a middle fusion method to aggregate the graphs in a type-aware manner. In particular, we cluster all the graphs with different types of similarity relations into groups based on the their correlations. With clustering, the involved graphs in each cluster are strongly correlated. In the light of this, we derive a common graph for each cluster and perform a graph-based ranking upon this common graph. We then fuse ranking results learned from different clusters to produce the final score. We apply the proposed graph-based multi-channel ranking scheme (GMR) to address the Chinese university ranking problem, as shown in Figure 3.1. In particular, this scheme first collects multi-channel Web data, ranging from official data, mass media reports, academic records, employment status of graduate students, to public comments. It then extracts a rich set of features from each channel to comprehensively represent the universities and then feeds the features into the model of GMR to generate the university ranking. Extensive experiments validate the effectiveness of the proposed middle fusion and the usability of the proposed GMR scheme.

The main contributions of this work are: 1) We present a novel graph-based multi-channel ranking scheme towards social indicator computation. It inherits the advantages of both early fusion and late fusion. 2) We

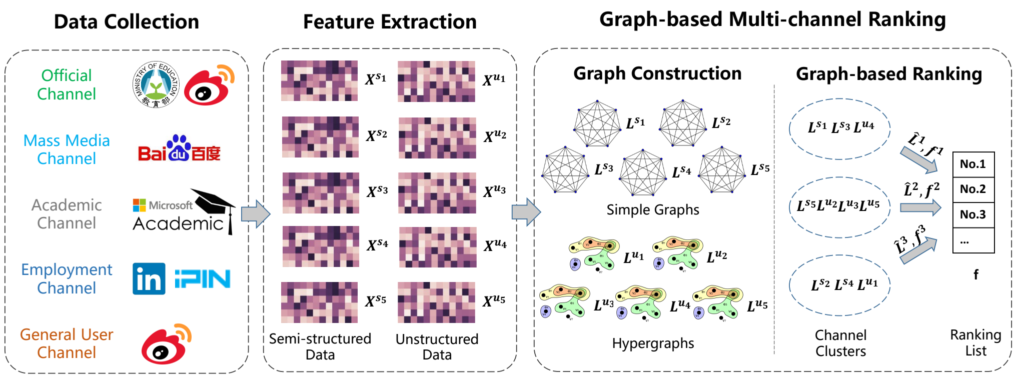


Figure 3.1: Schematic illustration of social indicator computing and a case study of Chinese university ranking.

successfully take the Chinese university ranking as a case study of social indicator computation. Experimental results validate the effectiveness of the proposed method (codes and our constructed data can be accessed through: https://github.com/hennande/cur/.

3.2 Related Work

Our work is related to recent studies on multi-view subspace learning, unsupervised ranking, and university ranking.

3.2.1 Multi-view Subspace learning

Subspace learning is a widely explored technique to analyze multi-view data. It aims to obtain compact latent representations by leveraging underlying structures and relations across multiple views. Typically, multiple views are mapped into a common space by different algorithms, including canonical correlation analysis [53], dictionary learning [10], matrix factorization [173, 105, 74], and joint learning [164]. In addition, the latent representations are further regularized to be sparse with different norms [138]. Apart from the shallow learning methods, subspace learning is also explored with deep learning models, such as deep restricted Boltzmann machines [117], deep feedforward networks [7, 183], and deep autoencoders [162]. In summary, although great success has been achieved by these models, few of them simultaneously consider the difference between unstructured and semi-structured data, let alone block-wise missing data.

3.2.2 Unsupervised Ranking

Unsupervised ranking is a popular technique to produce permutation of entities without labled data. Studies on unsupervised ranking are roughly separated into two categories based on whether the entities have direct linkages: 1) Linkage-based ranking. These methods infer rank of entities from the link structure information. For example, PageRank [124] and HITS [90] estimate the importance of webpages from the hyperlinks jumping to the given page. Standing on the shoulder of them, a couple of improvements have been presented. For instance, PopRank [120] further handles Web spam and heterogeneous graphs. BrowseRank [103] integrates the metadata of user behaviors. BiRank [72] expanded it to the bipartite graph. 2) Similarity-based ranking. Similarity-based ranking algorithms enforce that similar entities obtain close ranks. For example, Agarwal [2] constructed a graph, where vertices and edges respectively represent entities and similarity between them, and derived rankings from the Laplacian of the graph. Zhou et al. [193] replaced the conventional graph Laplacian with an iterated and unnormalized one to improve the robustness. Cheng et al. [31] further considered the entity redundancy with sink points in the Laplacian. In addition, Bu et al. [26] utilized the hypergraph instead of the simple one to represent the entities. Yet, most of the aforementioned methods are designed to process single view data.

3.2.3 University Ranking

Traditional university rankings, such as the U.S.News & World Report[5], Times Higher Education[6], and QS[7], usually measure the qualities of universities with a few pre-defined factors, such as the research reputation and academic reputation. These factors are then fused with human designated weights to obtain the final ranking scores. In China, several university rankings are calculated in a similar process by distinct organizations like the Chinese Universities Alumni

[5]http://www.usnews.com/rankings.
[6]https://www.timeshighereducation.com/.
[7]http://www.qs.com/.

Association (CUAA)[8], Research Center for China Science Evaluation (RCCSE)[9], and Chinese Academy of Management Science (CAMS)[10]. It is clear that the performance of these ranking systems highly depends on these pre-defined factors and their heuristic weights.

Instead of heuristic weights, some researchers attempted to fuse factors with statistical methods. Guarino et al. [64] applied the Bayesian latent variable analysis to learn the weights. Dobrota et al. [45] used I-distance values to estimate the weights based on data from previous years. In addition, some attempts have been done to rank universities with new factors. Lages et al. [94] ranked universities by the importance of their corresponding Wikipedia pages. Kapur et al. [86] utilized LinkedIn Economic Graph data to rank universities by employment of graduates. To sum up, these aforementioned ranking methods pay more attention to weight tuning or specific factors. With the multi-channel Web data, our ranking method explores multi-facets of universities and thus is able to rank the universities in a more comprehensive manner.

3.3 Methodology

The social indicator computation is formalized as: given a list of N entities, a historical ranking list of all the entities $\mathbf{y} \in \mathbb{R}^N$, and the latest entity descriptions from M channels, $\{[\mathbf{X^{s_1}}, \mathbf{X^{u_1}}], [\mathbf{X^{s_2}}, \mathbf{X^{u_2}}], \cdots, [\mathbf{X^{s_M}}, \mathbf{X^{u_M}}]\}$, the social indicator computation aims to learn a new ranking list $\mathbf{f} \in \mathbb{R}^N$ by harvesting the current data and the historical ranking list. $\mathbf{X^{s_m}} \in \mathbb{R}^{N \times D^{sm}}$ and $\mathbf{X^{u_m}} \in \mathbb{R}^{N \times D^{um}}$ are the features extracted from the semi-structured and unstructured data from the m-th channel; and $\mathbf{y}$ refers to the latest released social indicator by the various professional organizations. For example, if the desired social indicator is Chinese university ranking in 2017, $\mathbf{y}$ will be the ranking results in 2016.

We present a novel graph-based multi-channel ranking framework to compute social indicators: 1) We first construct a simple graph on the semi-structured data and a hypergraph on the unstructured data for each channel. 2) We then

[8]http://www.cuaa.net/cur/.
[9]http://www.nseac.com/html/168/.
[10]http://edu.sina.com.cn/gaokao/wushulian/.

cluster all the graphs into groups based on the correlations of their Laplacian matrices. 3) We ultimately learn a cluster-wise ranking list and fuse them together within a tailored objective function.

3.3.1 Graph Construction

In some channels, there indeed exist both semi-structured and unstructured data to describe the given entities. Semi-structured ones are of higher quality and thus more discriminative. On the contrary, the unstructured data are more noisy. Due to their distinct structures and features, we leverage two types of graph, simple graph and hypergraph, to represent the entities and their relations. The simple graph is sensitive to the noise in data; whereas the hypergraph is typically more robust but less discriminative than the simple one [53]. Thus, instead of naively merging the semi-structured and unstructured data, for each channel, we construct a simple graph over the semi-structured data and a hypergraph over the unstructured ones so that we neither sacrifice the discrimination of semi-structured data nor be affected by the noisy unstructured ones.

Simple Graph Construction. In a simple graph, vertices represent entities and edges refer to their pairwise relations. A simple graph with N vertices is represented by an incidence matrix, $\mathbf{W} \in \mathbb{R}^{N\times N}$, where W_{ij} is the weight of an edge linking the i-th and j-th vertices. The vertex degree is denoted by a diagonal matrix, $\mathbf{D} \in \mathbb{R}^{N\times N}$, where $D_{ii} = \sum_{j=1}^{N} W_{ij}$ is the degree of the i-th vertex. Given $\mathbf{X^{s_m}}$, $\mathbf{W}$ is estimated as,

$$W_{ij} = \begin{cases} exp(-\|\mathbf{x_i^{s_m}} - \mathbf{x_j^{s_m}}\|^2/2\sigma^2), & \text{if } i \neq j, \\ 0, & \text{otherwise,} \end{cases} \tag{3.1}$$

where the radius parameter σ is simply set as the median of the Euclidean distances of all pairs. Following [2], we normalize graph Laplacian matrix as,

$$\mathbf{L^{s_m}} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-1/2}. \tag{3.2}$$

Hypergraph Construction. Generalized from a simple graph where an edge links pairwise vertices, an edge in a hypergraph connects a set of vertices to represent the finitary relations. The incidence matrix of a hypergraph with

N vertices and P edges is $\mathbf{H} \in \mathbb{R}^{N\times P}$, where $H_{ij} = 1$ if the i-th vertex is connected by the j-th edge, otherwise $H_{ij} = 0$. The degree and weight of edges are respectively represented with two diagonal matrices, $\mathbf{E}$ and $\mathbf{W} \in \mathbb{R}^{P\times P}$, where $E_{jj} = \sum_{i=1}^{N} H_{ij}$. The vertex degree is represented by another diagonal matrix, $\mathbf{V} \in \mathbb{R}^{N\times N}$, where $V_{ii} = \sum_{j=1}^{P} W_{jj}H_{ij}$. Following [26], given $\mathbf{X^{u_m}}$, we calculate the hypergraph Laplacian matrix with,

$$\mathbf{L^{u_m}} = \mathbf{D^{v}}^{-1/2}(\mathbf{D^{v}} - \mathbf{HWD^{e}}^{-1}\mathbf{H}^{T})\mathbf{D^{v}}^{-1/2}. \tag{3.3}$$

In particular, the j-th edge connects the k-most similar vertices to the j-th vertex, $\mathcal{N}_j(\mathbf{x_j^{u_m}})$. The weight of the j-th edge is estimated by,

$$W_{jj} = \sum_{\mathbf{x_i^{u_m}} \in \mathcal{N}_j(\mathbf{x_j^{u_m}})} exp(-\|\mathbf{x_i^{u_m}} - \mathbf{x_j^{u_m}}\|^2/2\sigma^2). \tag{3.4}$$

3.3.2 Middle Fusion

After graph construction, the original multi-channel descriptions $\{[\mathbf{X^{s_1}}, \mathbf{X^{u_1}}]$, $[\mathbf{X^{s_2}}, \mathbf{X^{u_2}}]$, $\cdots, [\mathbf{X^{s_M}}, \mathbf{X^{u_M}}]\}$ are mapped to the Laplacian representations $\{[\mathbf{L^{s_1}}, \mathbf{L^{u_1}}]$, $[\mathbf{L^{s_2}}, \mathbf{L^{u_2}}]$, $\cdots, [\mathbf{L^{s_M}}, \mathbf{L^{u_M}}]\}$. Instead of fusing the graphs from different channels in either early fusion or late fusion, we propose to make a tradeoff between early and late fusion, named *middle fusion*. Middle fusion aims to leverage the advantages of both early and late fusion to handle the data heterogeneity and complex channel relations in social indicator computation. Towards this end, we first divide all the graphs into groups based on the correlations between their Laplacian matrices using spectral clustering [158]. During the clustering, the distance between two Laplacian matrices is estimated by Hilbert-Schmidt Independence Criterion (HSIC) [62],

$$dis(\mathbf{L^i}, \mathbf{L^j}) = HSIC(\mathbf{L^i}, \mathbf{L^j}, \phi, \varphi) = (N-1)^2/tr(\mathbf{PHQH}), \tag{3.5}$$

where ϕ and φ are the kernel functions of the i-th and j-th matrices; $\mathbf{P}$, $\mathbf{Q}$, and $\mathbf{H} \in \mathbb{R}^{N\times N}$. $\mathbf{P}$ and $\mathbf{Q}$ are the Gram matrices with,

$$\begin{cases} P_{mn} = \phi(\mathbf{l^i_m}, \mathbf{l^i_n}), \\ Q_{mn} = \varphi(\mathbf{l^j_m}, \mathbf{l^j_n}). \end{cases} \tag{3.6}$$

$\mathbf{H} = \mathbf{I} - N^{-2}\mathbf{I^1}$ centers the Gram matrix to have zero mean, where $\mathbf{I}$ and $\mathbf{I^1}$ respectively denote identity and all-one matrices.

3.3.3 Objective Function

Given the historical ranking list $\mathbf{y}$ and the clustered Laplacian matrices in K groups, $\{\{\mathbf{L^{1_1}}, \cdots, \mathbf{L^{1_{S^1}}}\}, \cdots, \{\mathbf{L^{K_1}}, \cdots, \mathbf{L^{K_{S^K}}}\}\}$, where S^k denotes the number of matrices in the k-th cluster. The desired ranking list $\mathbf{f}$ is learned via the following function:

$$\begin{aligned} \Gamma = \min_{\hat{\mathbf{L}}^\mathbf{k}, \mathbf{f}, \mathbf{f^k}} \frac{1}{2} \sum_{k=1}^{K} l_{intra}(\hat{\mathbf{L}}^\mathbf{k}, \{\mathbf{L^{k_1}}, \cdots, \mathbf{L^{k_{S^k}}}\}) + \\ \frac{\lambda_1}{2} \sum_{k=1}^{K} l_{man}(\hat{\mathbf{L}}^\mathbf{k}, \mathbf{f^k}) + \frac{\lambda_2}{2} l_{inter}(\mathbf{f}, \mathbf{y}, \{\mathbf{f^1}, \cdots, \mathbf{f^k}\}), \end{aligned} \tag{3.7}$$

where l_{intra}, l_{man}, and l_{inter} respectively denotes the loss of: 1) intra-group fusion, 2) manifold ranking, and 3) inter-group fusion. The intra-group fusion aims to learn a common Laplacian matrix $\hat{\mathbf{L}}^\mathbf{k} \in \mathbb{R}^{N\times N}$ to fuse the Laplacian matrices $\{\mathbf{L^{k_1}}, \cdots, \mathbf{L^{k_{S^k}}}\}$ in the k-th group. Based upon the k-th common Laplacian matrix, the manifold ranking learns a ranking list $\mathbf{f^k} \in \mathbb{R}^N$. Inter-group fusion combines rankings from different groups into the final ranking $\mathbf{f}$. $\mathbf{f}$ is further regularized by the historical ranking result $\mathbf{y}$ so that it satisfies the ranking smoothness. λ_1 and λ_2 balance the three terms.

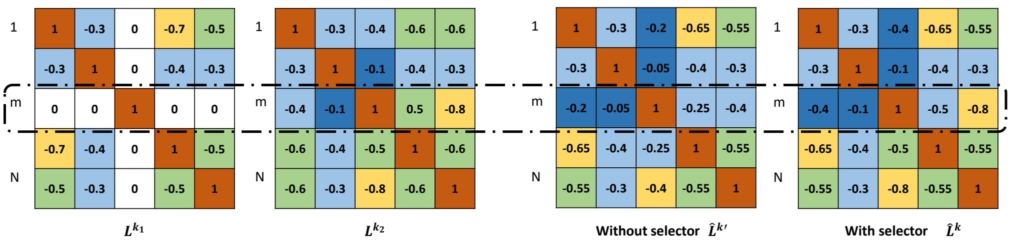
Intra-group Fusion. The intra-group fusion is an early fusion, which leans a common Laplacian matrix $\hat{\mathbf{L}}^\mathbf{k}$ to fuse the Laplacian matrices in the k-th group $\{\mathbf{L^{k_1}}, \cdots, \mathbf{L^{k_{S^k}}}\}$ by minimizing l_{intra},

$$\frac{1}{2} \sum_{i=1}^{S^k} tr\left((\hat{\mathbf{L}}^\mathbf{k} - \mathbf{L^{k_i}})^T \mathbf{S^{k_i}} (\hat{\mathbf{L}}^\mathbf{k} - \mathbf{L^{k_i}})\right). \tag{3.8}$$

Thereinto, $\mathbf{S^{k_i}} \in \mathbb{R}^{N\times N}$ is a diagonal matrix with,

$$S_{jj}^{k_i} = \begin{cases} 0, & \text{if the } j\text{-th entity misses the } i\text{-th channel,} \\ 1, & \text{otherwise.} \end{cases} \tag{3.9}$$

It is a selector to avoid the biases in the common Laplacian caused by missing data. Figure 3.2 presents a toy example with two graphs to illustrate the effects of the selector. In the example, data of the n-th entity in the k_1-th graph are missing. Thus, $\mathbf{S^{k_1}}$ and $\mathbf{S^{k_2}}$ are set as $\mathbf{I} \in \mathbb{R}^{N\times N}$ except $S_{nn}^{k_1} = 0$. So entries related to the n-th entity in the common Laplacian learned by the intra-group fusion are the same as those in $\mathbf{L^{k_2}}$. However, those entries could be bias towards zero if there is no selectors in the intra-group fusion. This is why we claim by integrating selectors, our intra-group fusion alleviates the impacts of data missing.



L^{k_1}

1	1	-0.3	0	-0.7	-0.5
	-0.3	1	0	-0.4	-0.3
m	0	0	1	0	0
	-0.7	-0.4	0	1	-0.5
N	-0.5	-0.3	0	-0.5	1

L^{k_2}

1	1	-0.3	-0.4	-0.6	-0.6
	-0.3	1	-0.1	-0.4	-0.3
m	-0.4	-0.1	1	0.5	-0.8
	-0.6	-0.4	-0.5	1	-0.6
N	-0.6	-0.3	-0.8	-0.6	1

Without selector $\hat{L}^{k'}$

1	1	-0.3	-0.2	-0.65	-0.55
	-0.3	1	-0.05	-0.4	-0.3
m	-0.2	-0.05	1	-0.25	-0.4
	-0.65	-0.4	-0.25	1	-0.55
N	-0.55	-0.3	-0.4	-0.55	1

With selector $\hat{L}^{k}$

1	1	-0.3	-0.4	-0.65	-0.55
	-0.3	1	-0.1	-0.4	-0.3
m	-0.4	-0.1	1	-0.5	-0.8
	-0.65	-0.4	-0.5	1	-0.55
N	-0.55	-0.3	-0.8	-0.55	1

Figure 3.2: A toy example to illustrate the impact of missing data. $\mathbf{L^k}$ and $\mathbf{L^{k'}}$ are the common Laplacian learned by the intra-group fusion with and without selectors, respectively.

Graph-based Ranking. Given a common Laplacian matrix $\mathbf{\hat{L}^k}$, manifold ranking learns a ranking list $\mathbf{f^k}$, where similar entities obtain close ranks, via,

$$\min_{\mathbf{f^k}} l_{man}(\mathbf{\hat{L}^k}, \mathbf{f^k}) = \mathbf{f^k}^T \mathbf{\hat{L}^k} \mathbf{f^k}, \tag{3.10}$$

where W_{ij} denotes the similarity between the i-th and j-th entities; D_{ii} and D_{jj} are the degree of the vertices representing the i-th and j-th entities.

Inter-group Fusion. As aforementioned, different local ranking lists are learned from different clusters, i.e., we have $\{\mathbf{f^1}, \mathbf{f^2}, \cdots, \mathbf{f^K}\}$. The inter-group fusion is a late fusion, which learns a set of weights $\mathbf{b} = [b_1, b_2, \cdots, b_K] \in \mathbb{R}^K$ to get the desired ranking $\mathbf{f} = \sum_{k=1}^{K} b_k \mathbf{f^k}$ and regulates the fused ranking to be smooth with the historical one by minimizing l_{inter},

$$(\sum_{k=1}^{K} b_k\mathbf{f^k} - \mathbf{y})^T\mathbf{C}(\sum_{k=1}^{K} b_k\mathbf{f^k} - \mathbf{y}),\ s.t.\ \sum_{k=1}^{K} b_k = 1, \tag{3.11}$$

where $\mathbf{C} \in \mathbb{R}^{N\times N}$ is diagonal matrix with $C_{jj} = c_j$. c_j is the pre-calculated weight of the j-th entity controlling the entity-aware ranking smoothness. Specifically, c_j should be set according to the confidence on its historical ranking y_j (detailed in Section 3.5.1).

3.3.4 Optimization

We adopt the alternating strategy to optimize the proposed model, until it converges.

Computing $\hat{\mathbf{L}}^{\mathbf{k}}$. To ease the optimization of $\hat{\mathbf{L}}^{\mathbf{k}}$, we set each common Laplacian as,

$$\hat{\mathbf{L}}^{\mathbf{k}} = \sum_{i=1}^{S^k} a_i^k\mathbf{L}^{\mathbf{k_i}},\ s.t.\ \sum_{i=1}^{S^k} a_i^k = 1, \tag{3.12}$$

and optimize each $\hat{\mathbf{L}}^{\mathbf{k}}$ independently keeping $\mathbf{f}$ and $\mathbf{b}$ fixed. After removing the fixed parts and substituting the constraint $\sum_{i=1}^{S^k} a_i = 1$ with Lagrange multipier δ, the objective function is rewritten as,

$$\begin{aligned}\min_{\mathbf{a^k}}\frac{1}{2}\sum_{i=1}^{S^k} tr\left((\sum_{j=1}^{S^k} a_j^k\mathbf{L}^{\mathbf{k_j}} - \mathbf{L}^{\mathbf{k_i}})^T\mathbf{S}^{\mathbf{k_i}}(\sum_{j=1}^{S^k} a_j^k\mathbf{L}^{\mathbf{k_j}} - \mathbf{L}^{\mathbf{k_i}})\right) + \\ \frac{\lambda_1}{2}\mathbf{f^k}^T\sum_{i=1}^{S^k} a_i^k\mathbf{L}^{\mathbf{k_i}}\mathbf{f^k} + \delta(1 - \mathbf{e}^T\mathbf{a^k}),\end{aligned} \tag{3.13}$$

where $\mathbf{e} = [1, 1, \cdots, 1]^T \in \mathbb{R}^{S^k}$. We then take the derivative of Equation (3.13) regarding $\mathbf{a_i^k}$, as follows,

$$\sum_{j=1}^{S^k}(a_j^k S^k - 1)tr\left(\mathbf{L}^{\mathbf{k_i}}\mathbf{S}^{\mathbf{k_i}}\mathbf{L}^{\mathbf{k_j}}\right) + \frac{\lambda_1}{2}\mathbf{f^k}^T\mathbf{L}^{\mathbf{k_i}}\mathbf{f^k} - \delta. \tag{3.14}$$

Setting it to zero and rearranging the terms, all a_i^k's and δ can be learned by

Table 3.1: Statistics of the collected multi-channel data.

Channels	Sources	#Universities	#Items	Duration
Official Channel	MOE	743	96,551	13.06-15.06
	Sina Weibo	721	10,912,234	15.01-16.05
Mass Media Channel	Baidu News	743	508,851	15.01-16.05
Academic Channel	Microsoft Academic	456	1,211,102	11.01-16.03
Employment Channel	LinkedIn	411	411	-
	iPIN	722	722	-
General User Channel	Sina Weibo	573	2,025,777	15.01-16.05

solving the following linear system,

$$\mathbf{M}\widehat{\mathbf{a^k}} = \mathbf{u}, \tag{3.15}$$

where $\widehat{\mathbf{a^k}} = [a_1^k, a_2^k, \cdots, a_{S^k}^k, \delta]^T \in \mathbb{R}^{S^k+1}$, $\mathbf{u} = [u_1, u_2, \cdots, u_{S^k}, 1]^T \in \mathbb{R}^{S^k+1}$, and $\mathbf{M} \in \mathbb{R}^{(S^k+1)\times(S^k+1)}$. M_{ij} and u_i are defined as follows,

$$\begin{cases} M_{ij} = S^k tr\left(\mathbf{L^{k_i}S^{k_i}L^{k_j}}\right), & i,j \neq S^k+1, \\ M_{ii} = 0, & i = S^k+1, \\ M_{ij} = 1, & \text{otherwise}, \\ u_i = \sum\limits_{j=1}^{S^k} tr\left(\mathbf{L^{k_i}S^{k_i}L^{k_j}}\right) - \frac{\lambda_1}{2}\mathbf{f^k}^T\mathbf{L^{k_i}f^k}. \end{cases} \tag{3.16}$$

Following a similar strategy, we compute **f** and **b**.

3.4 Chinese University Ranking

In this work, we take the Chinese university ranking as a case study of social indicator computation.

3.4.1 Data Collection

For each university, we collect five channel data from the Web. They are the official data, mass media data, academic records, employment data of graduate students, and public comments. In Table 3.1, we summarize the statistics of the collected data. In particular,

Official Channel. Official channel contains the primary information of the universities, such as student quality, official activities, and development plans, which plays a pivotal role in inferring university quality. Data in official channel are usually released by government agencies and university themselves. They includes: 1) MOE[11]. From the platform of MOE, we collect university profiles, such as location and category. Besides, we gather the enrollment scores of universities from 2013-2015[12]. As higher enrollment scores reflect the quality of the universities and potential of the students. And 2) Sina Weibo[13]. Sina Weibo is one of the most popular SNSs in China. Most Chinese universities publicize their official activities and announcements through their official Sina Weibo accounts. We thus crawl the historical posts from such accounts.

Mass Media Channel. Mass media channel contains insights of mass media which uncovers the hot topics, events, discoveries, and even criticisms related to universities. News reports from mass media are usually formalized by professional journalists with incisiveness of arguments, and hence their opinions are objective. To take full advantage of such opinions, we collect news reports mentioned the universities of interest from Baidu News[14].

Academic Channel. This channel contains academic records of universities showing the academic contribution and influences of universities. Such records are available from online bibliographic databases such as Google Scholar. In this work, given a university, we collect papers whose authors' affiliation is the given university. Meanwhile, we gather the papers' citations from Microsoft Academic.

Employment Channel. Employment channel contains employment status of universities' graduate students. This is one of the key factors related to university quality, because most students pursue higher education for better employment. The employment data are accessible through employment-oriented SNSs and third party data analysis companies. They include: 1) iPIN. We collect

[11]http://gaokao.chsi.com.cn/.

[12]In China, final year high school students first take part in the National College Entrance Examination (NCEE). They then apply for universities based on their NCEE scores. Regarding applications from students, the university selects students by their scores from high to low. The lowest score of the selected students is released as the enrollment score of the university.

[13]weibo.com/.

[14]news.baidu.com/.

employment data of universities' graduate students from the homepage in iPIN[15], a data analysis company in China. iPin provides average salary, working location distribution, and male-female ratio information of graduate students. And 2) LinkedIn, from which we collect university homepages.

General User Channel. General user channel contains public impressions, attitudes, and sentiment polarities of universities shared in SNSs posts, signaling the reputation of universities. We hence collect posts mentioning the given university from Sina Weibo.

Historical Ranking Result. The historical ranking result $\mathbf{y}$ is estimated from the three most popular Chinese university rankings: CUAA, RCCSE, and CAMS (Wu Shulian). To generate a relatively objective historical ranking list, ranking results in 2015 of these three traditional rankings are averagely fused. It is worthwhile to highlight that the historical ranking results in future can be obtained from our previous release rather than the results of the traditional rankings.

3.4.2 Feature Extraction

Regarding the collected multi-channel data, we extract three types of features to describe each university: 1) Sentiment features. We notice that data in mass media and general user channels convey the attitude and sentiment of users. We thus utilize the Chinese microblog sentiment analysis tool [83] to judge the polarity of contents from the mass media and general user channels. For each given input, this tool generates a three dimension distribution to denote its probability to be negative, neutral, and positive. 2) Topic features. According to our observation, contents in the official, mass media, or general users about similar universities are likely to express similar topics. For instance, reports from mass media may have a higher probability to report the topics of "research achievements" and "technologies" for top universities. Inspired by this, we explore the topic distributions over official, mass media, and general user channel. In particular, we generate topic distributions using Latent Dirichlet Allocation [18], which has been widely used in topic modeling. And 3) Statistic features.

[15]www.ipin.com/.

Table 3.2: Features extracted from the multi-channel data.

Channels	Semi-structured Data	Dimension	Unstructured Data	Dimension
Official Channel	NCEE_enrollment_line, category, is_985, is_211, key_subjects_count, city, fans_count, followers_count, posts_count, comments_count, likes_count, etc.	78	topics	56
Mass Media Channel	monthly_reports_count	16	topics, sentiment	95
Academic Channel	papers_count, first_author_papers_count, cooperated_papers_count, authors_count, citations_count, citations_author, citations_paper	13	-	0
Employment Channel	average_sallary, average_sallary_top5_subjects, working_city, male_female_ratio, similar_universities	443	-	0
General User Channel	posts_count, reposted_count, likes_count, comments_count,	4	topics, sentiment	81

Quality of universities are directly reflected by the volume of statistics, for instance, the average salary of graduate students, the number of publications, and the NCEE enrollment scores. Together with the sentiment and topic features, the statistical features are summarized in Table 3.2.

3.5 Experiment

3.5.1 Experimental Settings

Entity-aware Ranking Smoothness. As our historical ranking y is estimated from CUAA, RCCSE, and CAMS, the ranking smoothness weight of the i-th univeristy C_{ii} in Equation (3.7) is assigned as the ratio of rankings including the given university among CUAA, RCCSE, and CAMS.

Ground Truth. Establishing the ground truth for university ranking from scratch by ourselves is extremely resource consuming and not reliable. We thus turn to justify the 2016 university ranking results by our model in a pair-wise fashion. In particular, although the traditional university ranking results of CUAA, RCCSE, and CAMS are time- and resource-consuming, they are generated by experts with sufficient domain knowledge. We view them as annotators and establish the pair-wise ground truth upon their 2016 results. Note that the selected rankings are the most popular university rankings in China, which focus on evaluating the comprehensive quality of universities. Given a pair of universities $< u_i, u_j >$, if all CUAA, RCCSE, and CAMS 2016 rank u_i as better or worse than u_j, then the pair is labeled as 1 or -1, respectively. Otherwise, it is labeled as 0, meaning u_i and u_j are not distinguishable. The statistics of the constructed ground truth are shown in Table 3.3.

Table 3.3: Statistics of the constructed ground truth.

Universities	University Pairs			
	Label 1	Label 0	Label -1	All
640	178,342	48,448	178,342	405,132

Evaluation Metrics. The performance of our model and the baselines are measured by Cohen's kappa coefficient (κ) [107], macro-averaged F1 score, and micro-averaged F1 score [16]. We also carry out the significance test and report the p-values, *e.g.,* p@F1.

University Pair Tagging. Regarding the learned ranking list $\mathbf{f}$, the label of the i-th and j-th universities is set as,

$$\begin{cases} 1, & \text{if } f_i - f_j > \theta, \\ -1, & \text{if } f_i - f_j < -\theta, \\ 0, & \text{otherwise.} \end{cases} \tag{3.17}$$

θ is a hyperparameter and empirically set as 0.004. Note that other values in $\{0.001, 0.002, \cdots, 0.01\}$ would lead to similar conclusions.

Compared Methods. We compare the **GMR** with the following baselines,

- **Historical Ranking (HR)**: It takes the historical ranking list $\mathbf{y}$ as current ranking.
- **NCEE Enrollment Scores (NES)**: It ranks universities upon the average NCEE enrollment scores of universities.
- **Concatenation (Con)**: It is an early fusion method that first concatenates features of all channels, and then performs graph-based ranking on graphs constructed from the concatenated features [82].
- **Voting (Vot)**: It is a late fusion method that separately performs graph-based ranking upon each channel, and then averagely fuses the generated ranking lists into a single one [25].
- **Joint Learning (JL)**: It belongs to early fusion, which learns a common ranking list by simultaneously performing regularization on Laplacian matrices of all channels [160].

- **Subspace Learning (SL)**: It first maps multi-channel data to subspaces with the same dimension by dictionary learning [10]. Regarding representations in subspaces, it then performs the ranking via **JL**. Note that this method also belong to early fusion.

It is worthwhile highlighting that **Con**, **Vot**, **JL**, and **SL** also encourage the final ranking results to be close to the initial ranking one.

3.5.2 Parameter Tuning

In the proposed **GMR**, we have two implicit parameters and two explicit parameters. They are the number of nearest neighbors k in hypergraph construction, the number of clusters K, λ_1 and λ_2. During the experiments, we heuristically set k to 5 based on our observation on the data. The optimal values of the remaining parameters are carefully tuned with a 5-fold cross-validation. In each round, we divide our dataset into two parts: 80% of the universities pairs are used for tuning, and 20% are used for testing. We employ grid search to select the optimal parameters with a small but adaptive step size. The search ranges for λ_1, λ_2, and K are $[0.1, 100]$, $[10, 10,000]$, and $[1, 8]$, respectively. The parameters corresponding to the largest micro-averaged F1 are used to report the final results. For other compared methods, the procedures of parameter tuning are the same to ensure fair comparison.

3.5.3 Performance Comparison

The comparison results between the proposed **GMR** and baselines are summarized in Table 3.4. From this table, we have the following observations: 1) **NES** and **HR** perform worse than the other methods that leverage multi-channel data. This demonstrates the importance of utilizing available multi-channel data to improve ranking performance. Moreover, this result shows the advantage of the graph-based learning solutions as compared to heuristic ranking methods. 2) **Vot** performs worse than the three early fusion methods: **Con**, **JL**, and **SL**. It indicates that late fusion methods are less suitable for the multi-channel university ranking application. The **Vot** is suspected to be affected by the noise in the multi-channel university data. 3) However, **GMR** shows superior to the early fusion methods, which performs middle fusion (late

Table 3.4: Performance comparison between GMR and baselines.

Methods	Macro Averaged		Micro Averaged		κ	p@κ
	F1	p@F1	F1	p@F1		
NES	0.540±3e-7	8e-10	0.761±7e-1	4e-9	0.572±2e-6	3e-9
HR	0.618±2e-7	1e-9	0.868±3e-7	2e-7	0.764±1e-6	8e-8
Con	0.801±3e-6	1e-5	0.896±5e-7	2e-6	0.823±1e-6	2e-6
Vot	0.684±6e-6	2e-8	0.878±7e-7	4e-7	0.784±2e-6	2e-7
JL	0.802±7e-6	2e-4	0.894±3e-6	1e-5	0.820±8e-6	1e-5
SL	0.800±4e-6	8e-5	0.893±2e-6	6e-6	0.818±6e-6	6e-6
GMR	**0.812±4e-6**	-	**0.906±2e-6**	-	**0.840±4e-6**	-

Table 3.5: Performance comparison among components in our GMR.

Methods	Macro Averaged		Micro Averaged		κ	p@κ
	F1	p@F1	F1	p@F1		
GMR-HRC	0.800±2e-6	2e-6	0.898±8e-7	2e-6	0.825±2e-6	2e-6
GMR-MD	0.809±5e-6	8e-4	0.902±1e-6	3e-5	0.834±3e-6	3e-5
GMR-DH	0.811±5e-6	9e-2	0.903±8e-7	6e-4	0.835±2e-6	1e-3
GMR-CC	0.802±2e-5	5e-4	0.903±3e-6	4e-3	0.834±8e-6	3e-3
GMR	**0.812±4e-6**	-	**0.906±2e-6**	-	**0.840±4e-6**	-

fusion after early fusion). This result validates the effectiveness of the proposed middle fusion method. 4) All the p-values of the pairwise significance t-test based on 5-fold evaluation are much smaller than 0.05. This demonstrates that the performance improvements achieved by our model over the baselines are statistically significant.

3.5.4 Component-wise Comparison

We also carry out experiments to justify the effectiveness of each component. In particular, we compare the following methods by disabling some terms of our objective function in Equation (3.7).

- **GMR-HRC**: We set $\mathbf{C}$ to an identity matrix to ignore the historical ranking confidence.
- **GMR-MD**: We set all $\mathbf{S^{k_i}}$'s to identity matrices to ignore the missing data problem.
- **GMR-DH**: In this method, the semi-structured and unstructured data from one channel are directly concatenated and used to construct the simple graph.
- **GMR-CC**: It learns a common space from all channels and then performs ranking on the common representations.

Table 3.5 displays the performance of the above methods. From this table, we observe that: 1) **GMR** performs better than the remaining methods. It confirms

Table 3.6: Performance comparison among channels with our GMR model. Official, Media, Academic, Employ, Crowd respectively denote the official, mass media, academic, employment, and general user channels. The best and worst performance *w.r.t.* each metric is highlighted with bold font size and underline, respectively.

Methods	Macro Averaged		Micro Averaged		κ	p@κ
	F1	p@F1	F1	p@F1		
No-Official	<u>0.791±6e-6</u>	7e-6	<u>0.867±4e-6</u>	9e-8	<u>0.783±9e-6</u>	1e-7
No-Media	0.805±2e-6	6e-5	0.893±6e-7	6e-7	0.820±2e-6	9e-7
No-Academic	**0.817±9e-6**	1e-3	0.902±3e-6	2e-4	0.835±9e-6	6e-4
No-Employ	0.795±4e-6	6e-5	0.900±1e-6	3e-4	0.829±3e-6	2e-4
No-Crowd	0.814±1e-5	8e-2	0.895±4e-6	1e-5	0.825±1e-5	3e-5
All	0.812±4e-6	-	**0.906±2e-6**	-	**0.840±4e-6**	-

Table 3.7: Performance comparison among our ranking and traditional Chinese university rankings.

Ranking Results	Ours	RCCSE	CAMS	CUAA
Average Scores	**8.12±0.99**	7.59±1.51	7.71±1.35	8.06±0.81
Highest Score Percentage	53%	18%	35%	**59%**

the effectiveness of jointly considering the block-wise data completion, cluster-wise ranking, and ranking results fusion. 2) **GMR-HRC** performs much worse than **GMR**. It shows the importance of carefully setting entity-aware ranking smoothness, and hence assigning identical ranking smoothness to all the entities may lead to suboptimal performance.

3.5.5 Channel Comparison

To measure the representation ability of each channel, we hold one channel out and feed the others into our **GMR** model. The experimental results are displayed in Table 3.6. We observe that: 1) The performance of **GMR** decreases more when the official channel is not used. This suggests that the official channel provides more informative and important cues for university ranking. 2) With all channels used, **GMR** performs the best, which indicates that universities can be comprehensively described by more channels. 3) All the p-values of the pairwise significance t-test are much smaller than 0.05, which verifies the significance of performance improvements.

3.5.6 User Study

To further investigate the effectiveness of our scheme, we invite 17 volunteers[16] to evaluate our generated ranking list and the ranking results of RCCSM, CAMS,

[16]The volunteers are Chinese graduate students, research fellows, and visiting professors in different majors of National University of Singapore. All of them have either studied or worked in one of the top universities in China.

and CUAA in 2016. Each of the volunteers is presented the top-30 of each ranking list and required to assign it with one of eleven scores (ranging from 0 to 10). These scores represent the strength of the volunteer's agreement with the given ranking list. If the volunteer assign score s, it means the number of universities whose ranks are consensus with the expectations of the volunteer belongs to the range $(3(s-1), 3s]$. For instance, if the volunteer thinks that 20 of the top-30 universities are ranked as exepected, then he/she will assign 7 to the given ranking list. The user study results are summarized in Table 3.7. As can be seen, our ranking achieves an average score as high as the best traditional ranking among RCCSM, CAMS, and CUAA. It shows that our ranking results are comparable to those traditional rankings and further validates the usability of our scheme.

3.6 Conclusion

In this work, we proposed a new graph-based learning method by incorporating multiple types of relations represented by different graphs. The proposed scheme clusters highly correlated graphs into groups during the fusion process to model the correlation across relations so that the local smoothness is performed in a type-aware manner. We applied the proposed scheme in a meaningful real-world social indicator computation application, university ranking, where data from different channels reflects the relation of universities from different perspectives. To justify the effectiveness of the proposed method, we collected a dataset of Chinese universities and conducted extensive experiments. The experimental results demonstrate that the proposed method can capture the importance of different relations (channels) while the official channel dominates the university ranking performance as expected. Moreover, the generated ranking results are comparable to the traditional Chinese university rankings, which demonstrates the effectiveness and rationality of our scheme.

As shown in the experiment results, the ranking performance is sensitive to the criteria to evaluate graph correlation and the technique to clustering graphs into groups. Therefore, it is worthwhile to explore techniques to enhance the stability of the proposed scheme.

Chapter 4

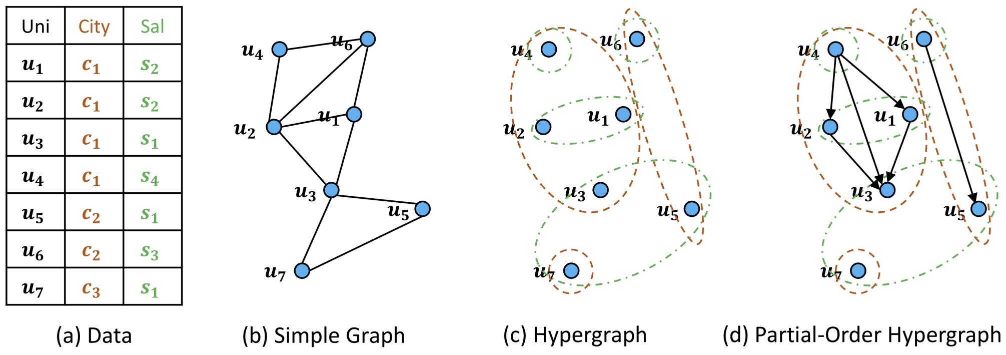
Learning on Partial-order Hypergraphs

In this chapter, we introduce a new graph-based learning framework which encodes task-specific knowledge in the format of partial-order rules.

4.1 Introduction

Hypergraph is a generalizaton of simple graph, in which an edge (*a.k.a.* hyperedge) can connect any number of vertices rather than just two. As such, it can model high-order relations among multiple entities that cannot be naturally represented by simple graphs. Figure 4.1 shows an illustrative example of using graph methods to tackle the university ranking task [49]. Each university has two features: the located city and the salary level of its graduates (Figure 4.1a). A simple graph can be constructed by connecting a university with its two nearest neighbors (Figure 4.1b). We can then perform a manifold ranking on the simple graph to obtain a ranked list of universities. Further, we can build a hypergraph by connecting universities with a same attribute (Figure 4.1c), *e.g.,* universities that are located in the same city, which is a high-order relation among universities missed by the simple graph.

In existing research, hyperedges in a hypergraph are typically formed by linking



Uni	City	Sal
u_1	c_1	s_2
u_2	c_1	s_2
u_3	c_1	s_1
u_4	c_1	s_4
u_5	c_2	s_1
u_6	c_2	s_3
u_7	c_3	s_1

(a) Data
(b) Simple Graph
(c) Hypergraph
(d) Partial-Order Hypergraph
u_1 u_2 u_3 u_4 u_5 u_6 u_7

Figure 4.1: An example of using graph methods to tackle the university ranking task. (a) Input data, where each row represents a university and its features: city and salary level; for salary level, smaller index indicates higher salary (*i.e.,* $s_1 > s_2 > s_3 > s_4$). (b) A simple graph, where an edge connects a vertex and its two-nearest vertices. (c) A hypergraph, where a hyperedge connects vertices with a same attribute: either in the same city or having the same salary level. (d) A partial-order hypergraph, where the directed edges within an hyperedge represent the partially ordering relationship between vertices on the salary level.

similar entities — either globally similar such as a cluster of entities that are close to each other [161, 110, 151], or locally similar such as sharing a same attribute [15, 150, 179]. However, we argue that many real-world applications need to deal with far more complex relations than similarities. One particular type is the ordering relationship among entities, which commonly exists in graded categorical features and numerical features. We take the university ranking task shown in Figure 4.1 as an example. Two universities u_5 and u_6 are located in the same city, while u_5 has a salary level much higher than that of u_6 — an evidence that u_5 might be ranked higher than u_6. Unfortunately, the hypergraph constructed in Figure 4.1c encodes the similarity information only, thus fails to capture the ordering information on salary. To address this limitation, an intuitive solution is to incorporate the ordering relations by adding directed edges between entities of a hyperedge, as shown in Figure 4.1d. It is worth noting that not every two entities within a hyperedge have an ordering relation; for example, two entities may have the same graded categorical feature (see u_1 and u_2 in Figure 4.1d) or the difference on the target numerical feature is not significant enough. As such, we term such generalized hypergraph with partial-order relations on vertices as a *Partial-Order Hypergraph* (POH), which is a new data structure that has never been explored before.

In this work, we formalize the concept of POHs and further develop regularization-based graph learning theories on them. We express the partial-order relations with logical rules, which can be used to encode prior domain knowledge. In the previous example of university ranking, one example of domain knowledge can be that for two universities u_i and u_j are in the same city, while u_i tends to be ranked higher than u_j if the salary level of u_i is higher than that of u_j. The corresponding logical rule can be written as:

$$city_{=}(u_i, u_j) \wedge salary_{>}(u_i, u_j) \rightarrow score_{>}(u_i, u_j). \tag{4.1}$$

We extend conventional hypergraph learning [190, 161] to incorporate such logical rules for an effective learning on POHs. Besides the improved accuracy, we can further enhance the interpretability of hypergraph learning. Specifically, we can interpret the learning results by verifying the logical rules, rather than relying on the *smoothness* factor only. To justify our proposed partial-order hypergraph and the learning method, we employ them to address two applications: university ranking [49] and popularity prediction [30, 70]. The two tasks are representatives of two machine learning tasks: unsupervised ranking and semi-supervised regression, respectively. Extensive results demonstrate the superiority of our proposed method, which significantly outperforms existing simple graph and hypergraph methods.

The key contributions of this work are summarized as follows.

- We propose a novel *partial-order hypergraph* to represent the partial-order relations among vertices, which are missing in the traditional hypergraph.
- We generalize existing graph-based learning methods to partial-order hypergraphs to encode domain knowledge in the form of logical rules.
- We empirically demonstrate the effectiveness of our POH and learning method on two machine learning tasks of unsupervised ranking and semi-supervised regression.

4.2 Related Work

Our work is directly related to the recent work on hypergraphs, which can be separated into two main categories: hypergraph construction and hypergraph utilization. Since proposed in [190], researchers have paid lots of attention on how to construct hypergraphs. For instances, Wang *et al.* [161] leveraged a sparse representation of the entity feature space to generate hyperedges and learn the relationship among hyperedges and the connected vertices. Instead of simply learning a sparse representation, Liu *et al.* [101] employed an elastic net to regulate the representation learning. Besides, Feng *et al.* [50] jointly learned hypergraph representations of multiple hypergraphs by further encouraging the consistency among different hypergraph representations. However, none of the aforementioned work is able to incorporate the partial-order relations among entities that exist in graded categorical features and numerical features during the construction of hypergraphs. They thus lead to severe information loss and limit the expressiveness of the constructed hypergraphs.

Besides, hypergraph and hypergraph-based learning have been widely applied on many machine learning tasks, including clustering, embedding, ranking, semi-supervised classification and regression [110, 151, 96, 15, 150, 179]. For instance, Li and Li [96] constructed a hypergraph to represent the correlations among news readers, news articles, topics and name entities and then ranked the news articles on the hypergraph to make recommendation for readers. Bellaachia and Al-Dhelaan [15] utilized a hypergraph to represent the relations among candidate sentences and made a graph-based extractive document summarization. Yoshida and Yuichi [178] used a hypergraph to estimate the betweenness centrality and importance of vertices. Huang *et al.* [81] constructed a hypergraph of images and predicted the attributes of the images with hypergraph-based label propagation. Tran *et al.* [150] built a hypergraph where vertices and hyperedges respectively represent features and training samples to represent the sparse pattern of the training data. Hmimida and Kanawati [77] depicted the relation among social users, resources, and the tags of resources assigned by the users. They then employed hypergraph-based ranking to recommend candidate tags for resources. These articles indicated the popularity and usability of hypergraphs and learning on hypergraphs. However they only utilized conventional hypergraphs instead of simultaneously improving the representation ability of the conventional hypergraphs and the corresponding learning methods.

4.3 Preliminaries

4.3.1 Hypergraph

As aforementioned, the vertices and hyperedges of a hypergraph represent the entities of interest and their relations, respectively. Given N entities with their features $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_N}]^T \in \mathbb{R}^{N \times D}$, we can construct a hypergraph with N vertices and M hyperedges, for which the structure can be represented as an incidence matrix $\mathbf{H} \in \mathbb{R}^{N \times M}$. Similar to the incidence matrix of a simple graph, $\mathbf{H}$ is a binary matrix, where $H_{ij} = 1$ if the i-th vertex is connected by the j-th hyperedge, otherwise $H_{ij} = 0$. There are two ways to define a hyperedge in existing work: attribute-based [96, 15, 150, 179] and neighbor-based [161, 110, 151]. An attribute-based hyperedge connects vertices with same value on one or multiple target attributes (*i.e.,* features). A neighbor-based hyperedge connects vertices nearby, based on these vertices with similarity larger than a threshold or simply using the k-nearest neighbors.

Moreover, we use the diagonal matrix $\mathbf{E} \in \mathbb{R}^{M \times M}$ to denote the degrees of hyperedges, *i.e.,* $E_{jj} = \sum_{i=1}^{N} H_{ij}$ denotes the number of vertices connected by the j-th hyperedge. Analogous to simple graph that an edge typically has a weight to model the strength of the relation, a hyperedge in hypergraphs also has a weight to denote the density of the vertices it is connected. Such weights are represented as a diagonal matrix $\mathbf{W} \in \mathbb{R}^{M \times M}$. To estimate the hyperedge weight, many methods have been proposed, among which the most popular one is to use the average pairwise similarity between vertices connected by the hyperedge:

$$W_{jj} = \frac{1}{E_{jj}} \sum_{H_{ij}=1} g(\mathbf{x_i}, \mathbf{x_j}), \tag{4.2}$$

where g denotes the similarity function on feature vectors. In graph-based methods, one common choice for g is the radial basis function (RBF) kernel, *i.e.,* $g(\mathbf{x_i}, \mathbf{x_j}) = \exp(\frac{-\|\mathbf{x_i}-\mathbf{x_j}\|^2}{2\sigma^2})$. Given the weights for hyperedges, we can further derive the degree of a vertex i: $V_{ii} = \sum_{j=1}^{M} W_{jj} H_{ij}$, *i.e.,* the sum of weights of hyperedges that are connected with i. We use the diagonal matrix $\mathbf{V} \in \mathbb{R}^{N \times N}$ to denote the vertex degree matrix.

4.3.2 Learning on Hypergraphs

Graph-based learning has been applied to various machine learning tasks such as manifold ranking, semi-supervised learning, and clustering [108, 180, 3]. The general problem setting is to learn a prediction function $\hat{\mathbf{y}} = f(\mathbf{x})$, which maps an entity from the feature space to the target label space. It is usually achieved by minimizing an objective function abstracted as:

$$\Gamma = \mathcal{G} + \lambda \mathcal{L}, \tag{4.3}$$

where $\mathcal{L}$ is a task-specific loss function that measures the error between prediction $\hat{\mathbf{y}}$ and ground-truth $\mathbf{y}$, $\mathcal{G}$ is a graph regularization term that smooths the prediction over the graph, and λ is a hyperparameter to balance the two terms. The regularization term typically implements the *smoothness* assumption that similar vertices tend to have similar predictions. For hypergraphs, a widely used $\mathcal{G}$ is the hypergraph Laplacian term, defined as:

$$\mathcal{G} = \sum_{i=1}^{N} \sum_{j=1}^{N} \underbrace{g(\mathbf{x_i}, \mathbf{x_j}) \sum_{k=1}^{M} H_{ik} W_{kk} H_{jk}}_{\text{strength of smoothness}} \underbrace{\left\| \frac{f(\mathbf{x_i})}{\sqrt{V_{ii}}} - \frac{f(\mathbf{x_j})}{\sqrt{V_{jj}}} \right\|^2}_{\text{smoothness}}. \tag{4.4}$$

The regularization term operates smoothness on each pair of entities, enforcing their predictions (after normalized by their degrees) to be close to each other. The strength of smoothness is determined by the similarity over their feature vectors $g(\mathbf{x_i}, \mathbf{x_j})$ and the hyperedges that connect the two vertices. It can be equivalently written in a more concise matrix form:

$$\mathcal{G} = trace(\hat{\mathbf{Y}}(\mathbf{S} \odot \mathbf{L})\hat{\mathbf{Y}}^T), \tag{4.5}$$

where $\hat{\mathbf{Y}} = [\hat{\mathbf{y_1}}, \hat{\mathbf{y_2}}, \cdots, \hat{\mathbf{y_N}}]$, each element of $\mathbf{S}$ is $S_{ij} = g(\mathbf{x_i}, \mathbf{x_j})$, and $\mathbf{L}$ is defined as $\mathbf{L} = \mathbf{V}^{-1/2}(\mathbf{V} - \mathbf{H}\mathbf{W}\mathbf{H}^T)\mathbf{V}^{-1/2}$, known as the *hypergraph Laplacian* *matrix.* Note that $\mathbf{L}$ is typically a sparse matrix, where an entry L_{ij} is nonzero only if vertex i and j are connected by at least one hyperedge. Thus, the time complexity of calculating $\mathcal{G}$ is linear *w.r.t.* the number of nonzero entries in $\mathbf{L}$, which is far smaller than N^2.

4.4 Partial-Order Hypergraph

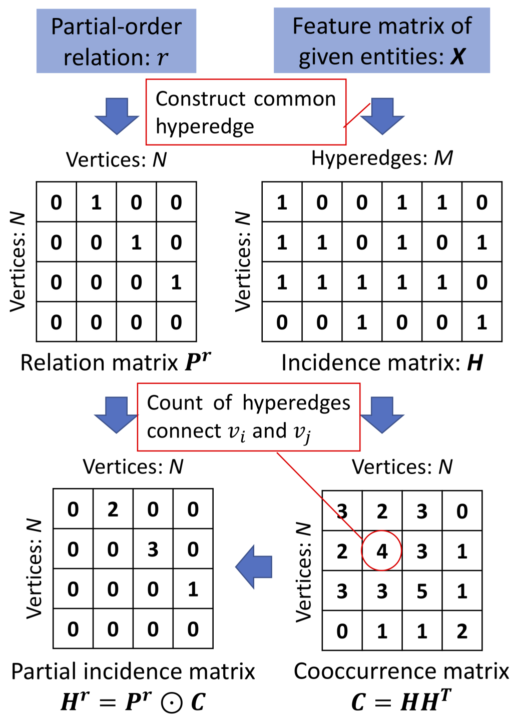
Distinct from the typical problem setting of hypergraph learning, we further associate the problem with a set of logic rules, which can be used to encode the partial-order relations between entities:

$$\{p^r(\mathbf{x_i}, \mathbf{x_j}) \rightarrow q^r(f(\mathbf{x_i}), f(\mathbf{x_j})) \mid r = 1, 2, \cdots, R\}, \tag{4.6}$$

where r denotes a partial-order relation, and there can be multiple relations (in total R) for a problem. For example, in the university ranking task, we can have a partial-order relation based on salary level, number of students, research grants, among other features. For each partial-order relation r, q^r is a binary function indicating whether the prediction of two entities satisfies a certain relation. For example, in a ranking/regression task, q^r can indicate whether $f(\mathbf{x}_i)$ is higher than $f(\mathbf{x}_j)$; in a classification task, q^r can indicate whether the probability of $\mathbf{x_i}$ being a class is higher than that of $\mathbf{x_j}$. The $p^r(\mathbf{x_i}, \mathbf{x_j})$ denotes whether $\mathbf{x}_i$ and $\mathbf{x}_j$ have the partial-order relation r. A partial-order relation is a pairwise relation satisfying the following basic properties on the entities connected by any hyperedge:

- Irreflexivity: not $p^r(\mathbf{x_i}, \mathbf{x_i})$.
- Asymmetry: if $p^r(\mathbf{x_i}, \mathbf{x_j})$ then not $p^r(\mathbf{x_j}, \mathbf{x_i})$.
- Transitivity: $p^r(\mathbf{x_i}, \mathbf{x_j})$ and $p^r(\mathbf{x_j}, \mathbf{x_k})$ implies $p^r(\mathbf{x_i}, \mathbf{x_k})$.

In what follows, we first present how to construct and represent a POH. We then elaborate our proposed regularized learning on POHs. Lastly, we analyze its time complexity.



Partial-order relation: r
Feature matrix of given entities: $\boldsymbol{X}$
Construct common hyperedge
Vertices: N
Hyperedges: M
Vertices: N
Relation matrix $\boldsymbol{P}^r$
Incidence matrix: $\boldsymbol{H}$
Count of hyperedges connect v_i and v_j
Partial incidence matrix $\boldsymbol{H}^r = \boldsymbol{P}^r \odot \boldsymbol{C}$
Cooccurrence matrix $\boldsymbol{C} = \boldsymbol{H}\boldsymbol{H}^T$

$$\boldsymbol{P}^r = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{H}^r = \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \boldsymbol{C} = \begin{bmatrix} 3 & 2 & 3 & 0 \\ 2 & 4 & 3 & 1 \\ 3 & 3 & 5 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}$$

Figure 4.2: A toy example to illustrate the construction of matrix representation of a POH. Given the feature matrix X and a partial-order relation r, we construct the incidence matrix H of the hypergraph and the binary relation matrix $\mathbf{P}^r$, respectively. We then generate the co-occurrence matrix C from H and apply element-wise product to C and $\mathbf{P}^r$ to get the partial incidence matrix $\mathbf{H}^r$.

4.4.1 Construction and Representation

To jointly represent the partial-order relations and the higher-order relations among entities, we first construct a normal hypergraph, and then use the directed edges to connect vertices that have any partial-order relation. Note that it is possible that there are multiple edges between two vertices, since multiple partial-order relations are considered. Concerning the efficiency of downtream graph-based learning applications, we constrain the directed edges to exist only on vertices connected by at least one hyperedge. Such a constraint guarantees that the number of directed edges constructed from a partial-order relation is no larger than the number of nonzero entries in the hypergraph Laplacian matrix. As such, a learning algorithm that enumerates all directed edges will not increase the time complexity of calculating the hypergraph Laplacian term.

As described in Section 4.3.1, after constructing a hypergraph, we use several matrices to represent it, such as the incidence matrix $\mathbf{H}$ and hypergraph Laplacian matrix $\mathbf{L}$. In addition to these matrices, we further introduce a partial incidence matrix $\mathbf{H^r} \in \mathbb{R}^{N \times N}$ to represent the directed edges of a partial-order relation r. As shown in Figure 4.2, given a partial-order relation r, we first construct a binary relation matrix $\mathbf{P^r} \in \mathbb{R}^{N \times N}$, where $P^r_{ij} = 1$ if $p^r(\mathbf{x_i}, \mathbf{x_j})$ is true. Based on the incidence matrix $\mathbf{H}$ of the hypergraph, we further build a co-occurrence matrix $\mathbf{C} \in \mathbb{R}^{N \times N}$, where each element C_{ij} denotes the number of hyperedges connecting vertex i and j. Then the partial incidence matrix $\mathbf{H}^r$ can be derived,

$$\mathbf{H^r} = \mathbf{P^r} \odot \mathbf{C}, \tag{4.7}$$

where $\odot$ is the element-wise matrix multiplication. In the partial incidence matrix, a non-zero entry H^r_{ij} means that the i-th and j-th vertices have the r-th partial-order relation, and they are simultaneously connected by H^r_{ij} hyperedges. In other words, we assign higher weights to vertex pairs that are connected by more hyperedges, accounting for the effect that vertex pairs with higher co-occurrence are more important.

4.4.2 Learning on Partial-Order Hypergraphs

After constructing a POH, we have several matrices to represent it, including the general ones describing a conventional hypergraph (*e.g.,* the incidence matrix $\mathbf{H}$ and edge weight matrix $\mathbf{W}$), and the specific partial incidence matrices $\{\mathbf{H^r} | r = 1, 2, \cdots, R\}$ to model partial-order relations. We now consider how to extend the conventional hypergraph learning methods for POHs.

The key problem is the encoding of the partial-order relations and the corresponding logical rules into the learning framework of Equation (4.3). Generally speaking, there are two solutions — either injecting the rules into the predictive model (*i.e.,* the definition of $f(\mathbf{x})$), or using the rules to regularize the learning (*i.e.,* augmenting the objective function Γ). The first solution may need different ways to encode the rules for different predictive models, such as the logistic regression, factorization machines [69], and neural networks [68]. As such, we opt for the second solution, aiming to provide a generic solution for POH learning. Specifically, we append an additional regularization term $\mathcal{P}$ that encodes partial-order rules to the objective function:

$$\Gamma = \mathcal{G} + \lambda\mathcal{L} + \beta\mathcal{P}, \tag{4.8}$$

where β is a hyperparameter to balance $\mathcal{P}$ and the other two terms. Similar to the smoothness regularizer $\mathcal{G}$, $\mathcal{P}$ should also operate on the predicted label space to regularize the learning process. We define $\mathcal{P}$ as:

$$\mathcal{P}_0 = \sum_{r=1}^{R} a_r \mathbb{E}_{(i,j)\sim H_{ij}^r \neq 0}\left[1 - q^r(\hat{\mathbf{y}}_\mathbf{i}, \hat{\mathbf{y}}_\mathbf{j})\right] = \sum_{r=1}^{R} \frac{a_r}{|\mathbf{H^r}|} \sum_{\{i,j|H_{ij}^r \neq 0\}} 1 - q^r(\hat{\mathbf{y}}_\mathbf{i}, \hat{\mathbf{y}}_\mathbf{j}), \tag{4.9}$$

where $\hat{\mathbf{y}}_\mathbf{i} = f(\mathbf{x_i})$ is the prediction of $\mathbf{x_i}$, $|\mathbf{H^r}|$ denotes the number of nonzero entries in $\mathbf{H^r}$, and a_r is the hyperparameter to control the importance of the logical rule of the r-th partial-order relation. The core idea of this regularization term is to enforce the predictions of vertices that have a partial-order relation to be consistent with the corresponding rule. To be more specific, small values of $\mathcal{P}_0$ can be achieved if $q^r(\hat{\mathbf{y}}_\mathbf{i}, \hat{\mathbf{y}}_\mathbf{j})$ is 1, meaning that $p^r(\mathbf{x_i}, \mathbf{x_j})$ is true (evidenced by $H_{ij}^r \neq 0$) and the rule $p^r(\mathbf{x_i}, \mathbf{x_j}) \rightarrow q^r(f(\mathbf{x_i}), f(\mathbf{x_j}))$ is satisfied. However, this definition treats all vertex pairs of a partial-order relation equally, without considering their relative strengths. This assumption may decrease modelling fidelity for practical applications. To address this problem, we revise the regularizer as:

$$\mathcal{P}_1 = \sum_{r=1}^{R} \frac{a_r}{|\mathbf{H^r}|} \sum_{\{i,j|H_{ij}^r \neq 0\}} (1 - q^r(\hat{\mathbf{y}}_\mathbf{i}, \hat{\mathbf{y}}_\mathbf{j}))H_{ij}^r, \tag{4.10}$$

which incorporates H_{ij}^r as a coefficient to rescale the regularization strength of a vertex pair. With such a formulation, we enforce stronger partial-order regularization for vertex pairs that are connected by more hyperedges. Lastly, to get rid of the difficulties in discrete optimization, we replace the binary logic function q^r with a continuous function s^r that approximates it. Such approximation trick allows stable optimization and has been widely used in probabilistic soft logics [9]. This leads to the final version of our proposed partial-order regularizer:

$$\mathcal{P} = \sum_{r=1}^{R} \frac{a_r}{|\mathbf{H^r}|} \sum_{\{i,j|H_{ij}^r \neq 0\}} s^r(\mathbf{\hat{y}_i}, \mathbf{\hat{y}_j}) H_{ij}^r, \tag{4.11}$$

where s^r is a self-defined function adjustable for different problems. For instance, s^r might be the subtraction between the predicted ranks $\mathbf{\hat{y}_i} - \mathbf{\hat{y}_j}$ in a ranking problem, while in a classification problem, s^r could be the gap between the predicted probabilities on a specific class.

Optimization. By minimizing the objective function of Equation (4.8), we can achieve the prediction function that is smooth over the hypergraph and satisfies the logical rules on partial-order relations. Note that the regularization terms $\mathcal{L}$ and $\mathcal{P}$ operate on the predicted label space only and do not introduce extra model parameters. As such, the only model parameters to learn come from the predictive model $f(\mathbf{x})$. Given that f is a differentiable function (*e.g.,* logistic regression and neural networks), we can optimize the objective function with standard gradient-based methods, such as the stochastic (or batch) gradient descent [59]. Moreover, one can also directly learn $f(\mathbf{x})$ without specifying an explicit form of the predictive model. This will make the prediction function comply with the regularization to the maximum degree. We will empirically study how would this affect the prediction performance for downstream applications in Section 4.6.

4.4.3 Time Complexity Analysis

In this subsection, we analyze the time complexity of POH learning by comparing it with the conventional hypergraphs. As discussed in [89] and Section 4.3.2, the computational complexity of conventional hypergraph learning methods are $O(J)$, where J denotes the number of nonzero entries in the sparse hypergraph Laplacian matrix $\mathbf{L}$. In contrast, the additional computational cost of our POH learning comes from the construction of the partial incidence matrices $\{\mathbf{H^r}|r = 1, 2, \cdots, R\}$ and the partial-order regularization term $\mathcal{P}$. To compute $\mathbf{H^r}$, we need to obtain the co-occurrence matrix $\mathbf{C}$ first, and then evaluate the element-wise product $\mathbf{C} \odot \mathbf{P^r}$. For $\mathbf{C}$, we can achieve it by traversing all the nonzero entries on the incidence matrix $\mathbf{H}$, for which the complexity is $O(J)$. Then for each nonzero element C_{ij} in $\mathbf{C}$, we check whether $\mathbf{p}^r(\mathbf{x}_i, \mathbf{x}_j)$ is true or not to obtain $\mathbf{C} \odot \mathbf{P}^\mathbf{r}$. As such, the complexity of constructing a $\mathbf{H}^\mathbf{r}$ is $O(J)$, and the overall complexity of constructing all R partial incidence matrices is $O(RJ)$. Similarly, the computation of $\mathcal{P}$ can be done in $O(RJ)$ time. In a real-world application, the number of partial-order relations R is typically small, since we need to account for the most prominent numerical or graded categorical features only. As such, the overall time complexity of our POH learning is essentially $O(J)$, which is the same as that of conventional hypergraph learning.

4.5 University Ranking

Following the work described in Chapter 3, we formulate university ranking as an unsupervised ranking (*i.e.,* re-ranking) problem. Given N universities with a feature matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$ and a historical ranking result $\mathbf{y} \in \mathbb{R}^N$, the target is to learn a new ranking $\mathbf{f} \in \mathbb{R}^N$. To solve the problem, we manually select several partial-order relations and construct a partial-order hypergraph (POH) to represent the given universities. Upon the constructed POH, we learn $\mathbf{f}$ by minimizing a ranking instantiation of the POH learning objective function. Specifically, we set the loss term in Equation (4.8) as $\mathcal{L} = \|\mathbf{y} - \mathbf{f}\|_F^2$, which encourages the learned ranking to be consistent and smooth with the historical one. Besides, we set the soft logic functions $\{s^r | r = 1, 2, \cdots, R\}$ as $s^r(f_i, f_j) = f_i - f_j$, *i.e.,* university i is encouraged to be ranked ahead of university j if the two universities have the selected partial-order relations. By specifying the above application-specific terms, we derive the objective function for the task,

$$\Gamma = \mathbf{f}^T \mathbf{L} \mathbf{f} + \lambda \|\mathbf{f} - \mathbf{y}\|_F^2 + \beta \sum_{r=1}^{R} \frac{a_r}{|\mathbf{H}^\mathbf{r}|} \sum_{\{i,j | H_{ij}^r \neq 0\}} ReLU((f_i - f_j) H_{ij}^r), \tag{4.12}$$

where we further use the rectifier function (ReLU) [65] on the partial-order regularization term, so as to guarantee the objective function to be non-negative for stable optimization.

4.5.1 Experiment Settings

Dataset. To investigate the effectiveness of the proposed method, we employ the dataset of Chinese university ranking collected by [49]. This dataset contains

743 Chinese universities with data collected between January 1st, 2015 and May 1st, 2016. Among the 743 universities, we select 438 top-tier ones in China[1] since they have more academic and research activities. Towards the historical ranking result $\mathbf{y}$, we merge the ranking results in 2015 from four well-known ranking systems of Chinese universities, namely, CUAA, WSL, WH, and iPIN[2]. The ranking ground-truth is generated in the same way, but based on the rankings of the four systems in the year 2016. As such, the task can be understood as using the past year's ranking and this year's features to predict the ranking of universities in this year. We normalize the constructed historical ranking result and the ground-truth into range $[0, 1]$ by scaling them with $1/N$.

Evaluation. We perform 5-fold cross-validation, employing three metrics to evaluate the ranking performance: mean absolute error (MAE) [170], Kendall's tau (Tau) [116], and Spearman's rank (Rho) [142]. The three metrics have been widely used to evaluate pointwise, pairwise, and listwise ranking methods [102]. Note that better performance is evidenced by smaller MAE, larger Tau and Rho scores. Moreover, we carry out the Student's t-test and reported the p-values where necessary.

Methods. We compare with the following baselines:

- **Simple Graph** [191]: It first constructs a simple graph to represent the universities, where the edge weight between two vertices is evaluated using the RBF kernel. We set the radius parameter σ as the median of the Euclidean distances of all pairs. The method then calculates the Laplacian matrix $\mathbf{L}$, learning $\mathbf{f}$ by minimizing the objective function $\mathbf{f}^T\mathbf{L}\mathbf{f} + \lambda\|\mathbf{y} - \mathbf{f}\|^2$. We experiment with different values of λ and report the best performance.

- **Hypergraph** [15]: It first calculates the similarities between universities, and then constructs the hypergraph using neighbor-based methods. Specifically, the i-th hyperedge connects the k universities that are most similar to university i. The learning of $\mathbf{f}$ is performed by minimizing Equation (4.3). We tune the two hyperparameters k and λ.

[1]In China, universities are officially separated into three tiers by Ministry of Education (https://tinyurl.com/moe-univ-list/.).

[2] CUAA: http://www.cuaa.net/cur/. WSL: http://edu.sina.com.cn/gaokao/wushulian/. WH: http://www.nseac.com/html/168/. iPIN: https://www.wmzy.com/api/rank/schList/.

- **GMR** [49]: This is the state-of-the-art method for the university ranking task. It builds a simple graph from the features of each channel, modelling the relations between channels to reach a consensus ranking on all simple graphs. We use the same hyperparameter settings as report in their paper.

Note that we omit the comparison with **TMALL** and **GCN** mentioned in Section 4.6.1. Because [49] has shown that **TMALL** (similar to the **JL** baseline in their paper) is less effective than **GMR**; **GCN** is not fit for this unsupervised ranking task as it is designed for semi-supervised and supervised tasks [89].

We evaluate several POH methods that incorporate different partial-order relations on the same hypergraph structure of **Hypergraph**:

- **POH-Salary**: This method considers the partial-order relation on the salary feature. We encode the logical rule $salary_{>}(\mathbf{x_i}, \mathbf{x_j}) \rightarrow rank_{<}(\mathbf{x_i}, \mathbf{x_j})$, meaning that $\mathbf{x_i}$ tends to be ranked higher than $\mathbf{x_j}$ if the salary feature of $\mathbf{x_i}$ is higher than that of $\mathbf{x_j}$.

- **POH-NCEE**: This method considers the partial-order relation on the NCEE feature, which stands for a university's admission requirement on the score of National College Entrance Examination. The logical rule to be encoded is naturally $NCEE_{>}(\mathbf{x_i}, \mathbf{x_j}) \rightarrow rank_{<}(\mathbf{x_i}, \mathbf{x_j})$, meaning universities with a higher NCEE score tend to have a better quality.

- **POH-All**: In this method, we model both partial-order relations as encoded in **POH-Salary** and **POH-NCEE**. We set the importance hyperparameters for the regularizers of the two rules as a_1 and $1 - a_1$, respectively. Tuning procedure of a_1 will be detailed in the following section.

Hyperparameter Tuning. We employ grid search to select the optimal hyperparameters for POH methods based on the results of Kendall's tau (τ). The optimal hyperparameter setting and implementation of the compared methods can be publicly accessed[3]. For **POH-Salary** and **POH-NCEE**, we tune one implicit (k) and two explicit hyperparameters (λ and β). To validate the strength of the proposed POH over traditional hypergraph, we set k and λ as the optimal ones of the baseline **Hypergraph**, and then search β in the range of

[3]https://github.com/hennande/Partial_Order_Hypergraph

Table 4.1: Performance comparison on university ranking. $*$ and $**$ denote that the corresponding performance is significantly better (p-value < 0.05) than all baselines and all other methods, respectively.

Methods	MAE	Tau	Rho
Simple Graph	$0.074 \pm$ 9e-3	$0.870 \pm$ 2e-2	$0.970 \pm$ 8e-3
Hypergraph	$0.067 \pm$ 7e-3	$0.876 \pm$ 9e-3	$0.974 \pm$ 5e-3
GMR	$0.065 \pm$ 7e-3	$0.871 \pm$ 3e-2	$0.970 \pm$ 1e-2
POH-Salary	$0.054 \pm$ 1e-2*	$0.892 \pm$ 1e-2*	$0.979 \pm$ 5e-3*
POH-NCEE	$0.055 \pm$ 1e-2*	$0.893 \pm$ 9e-3*	$0.978 \pm$ 5e-3*
POH-All	**$0.053 \pm$ 1e-2***	**$0.898 \pm$ 1e-2****	**$0.980 \pm$ 6e-3****

$[1e\text{-}4, 1e1]$. For **POH-All**, we tune one more hyperparameter a_1, which controls the importance of logical rules and is in the range of $[0, 1]$.

Note that we have intentionally fix k and λ to the optimal ones of **Hypergraph**, which also simplifies the tuning process. Further tuning k and λ based on the performance of POH methods can lead to even better performance (see Figure 4.4). Figure 4.3 shows the performance of **POH-All** *w.r.t.* β and a_1. This is accomplished by varying one parameter and fixing the other to the optimal value. As can be seen, our method is rather insensitive to hyperparameters around their optimal settings.

4.5.2 Experiment Results

Method Comparison. Table 4.1 summarizes the performance comparison on university ranking, from which we have the following observations: (1) **Hypergraph** performs better than **Simple Graph**, which verifies that considering the higher-order relations among universities is effective for the ranking task. (2) All POH-based methods outperform baselines by a large margin (*e.g.,* **POH-All** ourperforms **GMR** with an improvement of 18.46%, 3.10%, and 1.03% *w.r.t.* MAE, Tau, and Rho, respectively). It demonstrates the effectiveness of our proposed POH and regularized learning in integrating partial-order relations. (3) **POH-All** outperforms both **POH-Salary** and **POH-NCEE**. It further verifies the advantage of POH-based learning methods and reflects that jointly modelling multiple partial-order relations and rules is helpful. (4) The p-values of student's t-test between POH-based methods and all the other methods are smaller than 0.05, indicating the significance of the performance improvements.

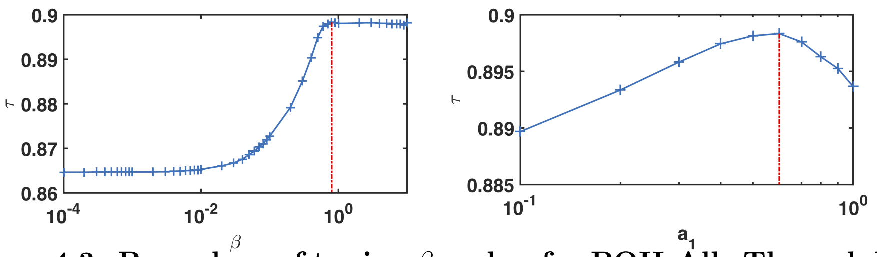


Figure 4.3: Procedure of tuning β and a_1 for POH-All. The red dotted line marked the optimal settings

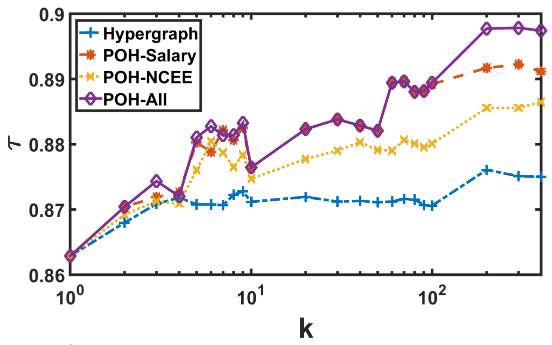


Figure 4.4: Performance comparison on Kendall's tau (τ) of Hypergraph and POH-based methods *w.r.t.* different k.

As we construct hypergraphs by connecting a vertex with its k-nearest vertices, the larger k makes the POH-based methods to consider more vertex pairs with the given partial-order relations (these pairs would be eliminated if the vertex pair is not connected by any hyperedge). It is thus interesting to see how does the setting of k impact the performance of POH learning. Figure 4.4 shows the performance of Tau of **Hypergraph** and our POH-based methods on different k. Note that other hyperparameters have been fairly tuned for each setting of k. As can be seen, all POH-based methods outperform the **Hypergraph** on all settings. It demonstrates that the proposed POH learning consistently outperforms the conventional hypergraph, regardless of the underlying hypergraph structure. Moreover, all POH-based methods achieve performances better than those reported in Table 4.1, which shows the performance of POH methods on the optimal k of **Hypergraph** only. It reveals the potential of POH-based methods to further improvements if a better hyperparameter tuning strategy is applied.

Result Analysis. To understand the results better, we perform finer-grained error analysis. Given the result generated by a method, we generate an array of rank positions (integers from 1 to N) for universities, computing the absolute

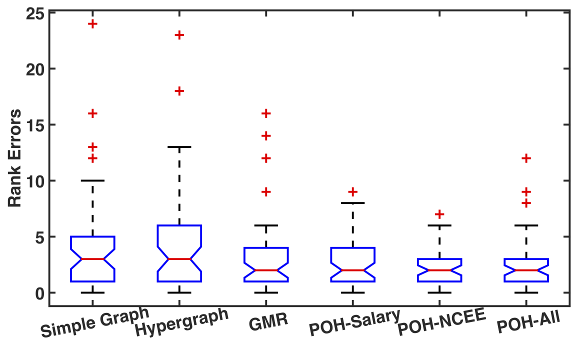


Figure 4.5: Distribution of absolute rank errors.

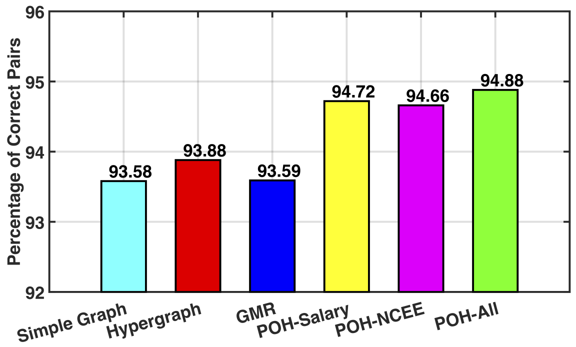


Figure 4.6: Percentage of correctly ranked university pairs.

error on the rank position on each university.

Figure 4.5 depicts the distribution of the absolute rank errors as a boxplot. As can be seen, the rank error distribution of POH-based methods is more dense and centralizes at smaller medians than that of **Simple Graph** and **Hypergraph**. It provides sufficient evidence on the better ranking generated by the POH-based methods. Moreover, we find that **Simple Graph** and **Hypergraph** make errors larger than 5 on about 25% of the universities, which is rarely seen from POH-based methods. Meanwhile, the largest error made by **Simple Graph** and **Hypergraph** is almost two times that of POH-based methods. These results demonstrate that POH-based methods are more robust, thus more applicable to real-world applications. Among the baselines, **GMR** achieves the smallest rank error, which is comparable with that of **POH-Salary**. This signifies the usefulness of modelling the relations among data from different channels, which could be a future direction to be explored on POH-based methods.

Besides the investigation on pointwise rank errors, we further performed an analysis on pairwise ranks. For each method, we count the number of university pairs that are ranked correctly, and drew the percentage of correct pairs in Figure

4.6. As shown, POH-based methods manage to generate ranks with correct order on 1% more university pairs than those on **Simple Graph**, **Hypergraph**, and **GMR**. This further demonstrates the accuracy and advantage of POH-based methods. Considering that there are more than 80,000 university pairs, an improvement of 1% (correctly ranking 800+ pairs) is a significant improvement.

4.6 Popularity Prediction

Predicting the popularity of online content is a hot research topic in social media mining and has varying problem statements [70, 132]. Following the recent work on micro-video popularity prediction [30], we formulate the task as a semi-supervised regression problem. Given $N + U$ items with a feature matrix $\mathbf{X} \in \mathbb{R}^{(N+U)\times M}$ and the ground-truth popularity of the N items $\mathbf{y} \in \mathbb{R}^N$, the objective is to learn a function $\hat{y_i} = f(\mathbf{x_i})$ that maps an item from the feature space to the popularity space. To solve the problem, we first construct a POH with partial-order relations on some important numerical features (to be detailed later in experiments). We then derive an instantiation of the general framework Equation (4.8) for the semi-supervised regression task as follows:

$$\Gamma = \hat{\mathbf{y}}^T \mathbf{L} \hat{\mathbf{y}} + \lambda \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 + \beta \sum_{r=1}^{R} \frac{a_r}{|\mathbf{H^r}|} \sum_{\{i,j|H_{ij}^r \neq 0\}} ReLU((\hat{y}_j - \hat{y}_i) H_{ij}^r), \qquad (4.13)$$

where $\hat{\mathbf{y}} = [\hat{y}_1, \cdots, \hat{y}_N, \hat{y}_{N+1}, \cdots, \hat{y}_{N+U}] \in \mathbb{R}^{N+U}$, denoting the prediction of all items (both with labels and without labels).

4.6.1 Experiment Settings

We test the proposed method on a micro-video popularity prediction dataset [30]. The dataset contains 9,719 micro-videos collected from Vine[4]. Following the original paper [30], we use the same features to describe the videos and perform 10-fold cross-validation. From the regression perspective, we follow the previous work [30] and employ normalized mean square error (nMSE). Meanwhile, we utilize two ranking-oriented metrics, Tau and Rho correlation coefficients. Besides, we carry out the Student's t-test and report the p-values where necessary.

[4]https://vine.co/.

Methods. We compare with the following baselines:

- **Simple Graph** [191]: We apply the same setting as the **Simple Graph** described in Section 4.5.1.

- **Hypergraph** [15]: We also adopt the same setting as the **Hypergraph** in Section 4.5.1.

- **TMALL** [30]: This method first calculates a simple graph Laplacian matrix with features from each modality (visual, audio, *etc.*). It then learns a common space Laplacian matrix by considering the relations among different modalities and fusing the corresponding graph Laplacian matrices. It finally performs a simple graph learning like **Simple Graph** on the common Laplacian matrix. We follow the settings as reported in their paper.

- **GCN** [89]: This is the state-of-the-art graph learning method by using graph convolutional neural networks. We replace the log loss term in their implementation with the same mean squared loss in Equation (4.13) for a fair comparison. We carefully tune the four hyperparameters, namely, learning rate, dropout ratio, l_2-norm weight and hidden layer size.

- **LR-HG**: This method is similar to **Hypergraph**. Instead of directly learning $\hat{y}$, we parameterize it as a linear regression (LR) model on features. The optimization process learns the parameters of LR, which is used to predict $\hat{y}$.

We evaluate several POH methods on the same hypergraph structure of **Hypergraph**:

- **POH-Follow**: This method considers a partial-order relation on the follower feature (*i.e.,* the number of followers of the user who posted the video). It encodes the logical rule $followers_{>}(\mathbf{x_i}, \mathbf{x_j}) \rightarrow \mathit{popularity}_{>}(\mathbf{x_i}, \mathbf{x_j})$, meaning that $\mathbf{x_i}$ would be more popular than $\mathbf{x_j}$ if the user of $\mathbf{x_i}$ has more followers than that of $\mathbf{x_j}$.

- **POH-Loop**: This method has the same setting as **POH-Follow**, besides that it encodes another partial-order relation on the loop feature (*i.e.,* total number of views of all videos posted by a user).

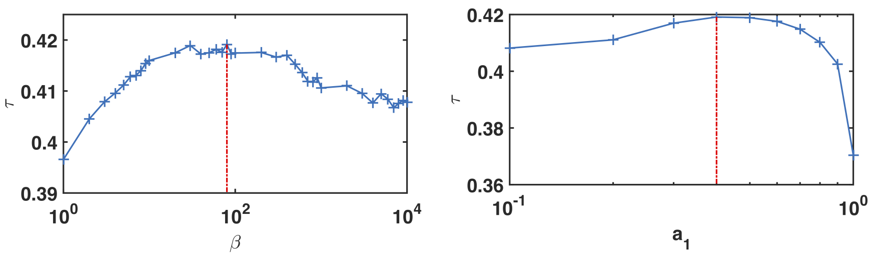


Figure 4.7: Procedure of tuning β and a_1 for POH-All. The red dotted line marks the optimal settings.

Table 4.2: Performance comparison on popularity prediction. $*$ ($**$) denote that the corresponding performance is significantly better (p-value < 0.05) than all baselines (all other methods).

Methods	nMSE	Tau	Rho
Simple Graph	$0.999 \pm$ 1e-3	$0.137 \pm$ 2e-2	$0.200 \pm$ 2e-2
Hypergraph	$1.000 \pm$ 4e-5	$0.165 \pm$ 3e-2	$0.240 \pm$ 4e-2
TMALL[5]	$0.979 \pm$ 9e-3	-	-
POH-Follow	$1.000 \pm$ 4e-4	$0.393 \pm$ 3e-2*	$0.562 \pm$ 3e-2*
POH-Loop	$0.997 \pm$ 2e-3	$0.376 \pm$ 2e-2*	$0.540 \pm$ 3e-2*
POH-All	$0.989 \pm$ 9e-3	**0.419 $\pm$ 2e-2****	**0.592 $\pm$ 3e-2****
GCN	$0.919 \pm$ 6e-2	$0.171 \pm$ 2e-2	$0.252 \pm$ 3e-2
LR-HG	$0.846 \pm$ 1e-1*	$0.117 \pm$ 2e-2	$0.182 \pm$ 3e-2
LR-POH	**0.724 $\pm$ 2e-1****	$0.350 \pm$ 2e-2*	$0.496 \pm$ 3e-2*

- **POH-All**: This method jointly encodes the two partial-order relations in **POH-Follow** and **POH-Loop**. We set the corresponding rule importance hyperparameters as a_1 and $1 - a_1$, respectively.

- **LR-POH**: Similar to **LR-HG**, this method parameterizes the $\hat{y}$ of **POH-All** as a linear regression model on input features.

Hyperparameter Tuning. We employ the same procedure as described in Section 4.5.1 to tune the hyperparameters of POH methods. Optimal hyperparameter settings of each compared method will be released together with their codes. We investigate the sensitivity of our proposed POH-based methods by taking **POH-All** as an example. Figure 4.7 illustrates the performance of **POH-All** while varying one hyperparameter and fixing the others with optimal values. Again, the results demonstrate that our model is not sensitive to the parameters around their optimal settings.

4.6.2 Experiment Results

Method Comparison. We first investigate the effectiveness of the proposed methods. Table 4.2 shows the performance of all the compared methods. We have the following findings:

(1) **Hypergraph** outperforms **Simple Graph** *w.r.t.* Tau and Rho, although they achieve the same performance level on nMSE. It verifies that considering the higher-order relations among videos leads to more accurate popularity prediction with relative orders.

(2) **POH-Follow** and **POH-Loop** further surpass **Hypergraph** with an average improvement of 133.03% and 129.58% on the pairwise and listwise ranking metrics; meanwhile, slight improvement is obtained on the pointwise regression metric nMSE. This indicates that considering meaningful partial-order relations is particularly helpful for better predicting the relative order of the videos.

(3) **POH-All** outperforms **POH-Follow** and **POH-Loop** with a significant average improvement on Tau (+8.97%) and Rho (+7.44%) as well as a slight improvement on nMSE. It validates that jointly considering multiple partial-order relations is useful.

(4) Comparing **Hypergraph** with **LR-HG**, we can see that better nMSE can be achieved by using LR as the predictive model, but the two ranking metrics become worse. The same situation can be observed for **POH-All** and **LR-POH**. This provides evidence that using a sophisticated model can better fit the labels and help to minimize the regression loss, however, the ranking performance may not be necessarily improved. The same finding has been observed before in popularity prediction [70] and another orthogonal application of item recommendation [33]. In our case of graph-based learning, the regularizers (for smoothness and partial-order rules) carry strong signals for learning the relative orders between vertices. However, the regularization effects might be weakened when a specialized model is used to fit the label in the meantime. (5) **GCN** outperforms **POH-All** *w.r.t.* nMSE, while Tau and Rho indicate that its ranking performance is worse. The lower nMSE of **GCN** can be credited to the strong representation power of the underlying neural network, which can fit the labels well. However, **GCN** may overfit the data and fail to predict the popularity ranking well without regularization on the relative orders of vertices.

(6) **LR-POH** achieves the best performance with significantly better nMSE than all the other compared methods as well as tremendously better Tau and Rho than all the baseline methods. This further demonstrates the effectiveness

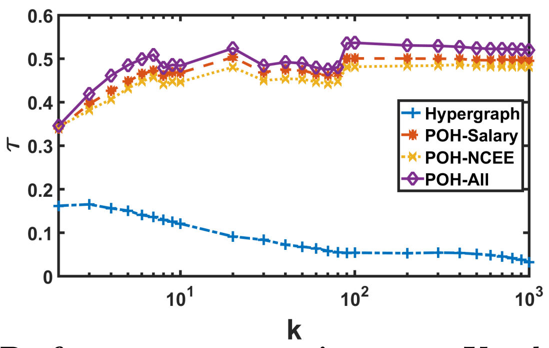


Figure 4.8: Performance comparison on Kendall's tau (τ) of Hypergraph and POH-based methods *w.r.t.* different k.

of our proposed POH learning.

We further study whether the performance improvements of the proposed POH-based methods are consistent under different hypergraph settings. We compare the optimal performance of **Hypergraph**, **POH-Follow**, **POH-Loop**, and **POH-All** under different values of k, which controls the number of videos connected by a hyperedge. As illustrated in Figure 4.8, all POH-based methods outperform the **Hypergraph** under all the values of k by a large margin. It is worth noting that the optimal performance of POH methods are better than that shown in Table 4.2 (Table 4.2 shows the results of POH on the optimal setting of **Hypergraph**). This is consistent with the university ranking task, which implies the potential of further improving POH learning with a better hyperparameter tuning strategy.

4.7 Conclusions

In this work, we proposed a novel partial-order hypergraph that improves conventional hypergraphs by encoding the partial-order relations among vertices. We then generalized existing graph-based learning methods to partial-order hypergraphs by integrating the second-order logic rules that encode the partial-order relations. Moreover, the time complexity of learning remains unchanged. Experimental results on university ranking and video popularity prediction demonstrate the effectiveness of our proposed methods.

In addition, the results on popularity prediction show the merit of representation learning operations in fitting the data. Recall that **LR-POH** (**GCN**) outperforms **POH-All** (**Simple Graph**) *w.r.t.* nMSE (lower nMSE means closer predictions to real values). As such, in the following, we shift our attention from graph Laplacian regularization to graph representation learning.

Chapter 5

Temporal Graph-based Learning for Stock Prediction

In this chapter, we introduce our approach for graph-based learning with dynamic vertex features. We demonstrate the framework with a stock prediction task where we have relations between stocks and historical prices (time-series) as the dynamic stock features.

5.1 Introduction

According to the statistics reported by the World Bank in 2017, the overall capitalization of stock markets worldwide has exceeded 64 trillion U.S. dollars[1]. With the continual increase in stock market capitalization, trading of stocks has become an attractive investment instrument for many investors. However, whether an investor could earn or lose money depends heavily on whether he/she can make the right stock selection. Stock prediction, which aims to predict the future trend and price of stocks, is one of the most popular techniques to make profitable stock investment [129], although there are still debates about whether the stock market is predictable (*a.k.a.* the Efficient Markets Hypothesis) among the financial economists [114, 104]. Some recent evidences indicate the

[1]https://data.worldbank.org/indicator/CM.MKT.LCAP.CD/.

Table 5.1: An intuitive example of one method that predicts the price change of stocks more accurately (*i.e.,* smaller MSE) but leads to a less profitable stock selection (*i.e.,* smaller profit). Method 1 selects stock A (30) while Method 2 selects stock B (10).

Ground Truth			Method 1					Method 2				
			Prediction			Performance		Prediction			Performance	
A	B	C	**A**	B	C	MSE	Profit	A	**B**	C	MSE	Profit
+30	+10	-50	**+50**	-10	-50	266	30	+20	**+30**	-40	200	10

A, B, C denote three stocks; numbers (+20) are the true/predicted price change of stocks; values in bold correspond to suggested selections. MSE is calculated between the prediction and ground-truth over A, B, and C. Profit is the true movement (ground truth) of the selected stock (*e.g.,* A for Method 1).

predictability of stock markets, which motivates further exploration of stock prediction techniques [152, 97, 184, 79, 135].

Traditional solutions for stock prediction are based on time-series analysis models, such as Kalman Filters [174], Autoregressive Models and their extensions [1]. Given an indicator of a stock (*e.g.,* stock price), this kind of model represents it as a stochastic process and takes the historical data of the indicator to fit the process. We argue that such mainstream solutions for stock prediction have three main drawbacks: 1) The models heavily rely on the selection of indicators, which is usually done manually and is hard to optimize without special knowledge of finance. 2) The hypothesized stochastic processes are not always compatible with the volatility of stock prices in the real world. 3) These models can only consider a few indicators since their inference complexity typically increases exponentially with the number of indicators. As such, they lack the capability to comprehensively describe a stock that could be influenced by a plethora of factors. To tackle these drawbacks, advanced techniques like deep neural networks, especially the recurrent neural networks (RNNs), have become a promising solution to substitute the traditional time-series models to predict the future trend or exact price of a stock [11, 187, 185, 184].

A state-of-the-art neural network-based solution is the State Frequency Memory (SFM) network [184], which models the historical data in a recurrent fashion and captures the temporal patterns in different frequencies. This method achieves promising performance of predicting the daily opening price of fifty U.S. stocks one day ahead with a mean square error (MSE) of less than six dollars. However, we argue that such prediction methods are suboptimal to guide stock selection, since their optimization target is not at selecting the top stocks with the highest expected revenue. To be specific, they typically address stock prediction as either a classification (on price movement direction) or a regression (on price value) task, which would cause a large discrepancy on the investment revenue. Table 5.1 gives an intuitive example, where a method with a better prediction performance (measured by regression MSE) suggests a less profitable stock. This implies the possible discrepancy between the actual target of stock selection and the optimized target of regression (classification), such that an optimal method of regression (classification) does not necessarily select the optimal stock to trade.

Another limitation of existing neural network-based solutions is that they typically treat stocks as independent of each other and ignore the relations between stocks. However, the rich relations between stocks and the corresponding companies may contain valuable clues for stock prediction. For example, stocks under the same sector or industry like GOOGL (Alphabet Inc.) and FB (Facebook Inc.) might have similar long-term trends. Besides, the stock of a supplier company might impact the stock of its consumer companies especially when a scandal of the supplier company is reported, such as the falsification of product quality data. To integrate stock relations into prediction, an intuitive solution is to represent the stock relations as a graph and regularize the prediction of stocks based on the graph (*i.e.,* graph-based learning) [49, 122, 89, 84]. However, conventional graph learning techniques cannot capture the temporal evolution property of stock markets (*e.g.,* the strength of influence between two given stocks may vary quickly), since the graph is fixed at a particular time.

To address the aforementioned limitations of existing solutions, we formulate stock prediction as a ranking task, for which the target is to directly predict a stock list ranked by a desired criteria like return ratio. We then propose an end-to-end framework, named *Relational Stock Ranking* (RSR), to solve the stock ranking problem. An illustration of our framework can be found in Figure 5.1. Specifically, we first feed the historical time series data of each stock to a Long Short-Term Memory (LSTM) network to capture the sequential dependencies and learn a stock-wise sequential embedding. By devising a new *Temporal Graph Convolution* (TGC), we next revise the sequential embeddings by accounting for

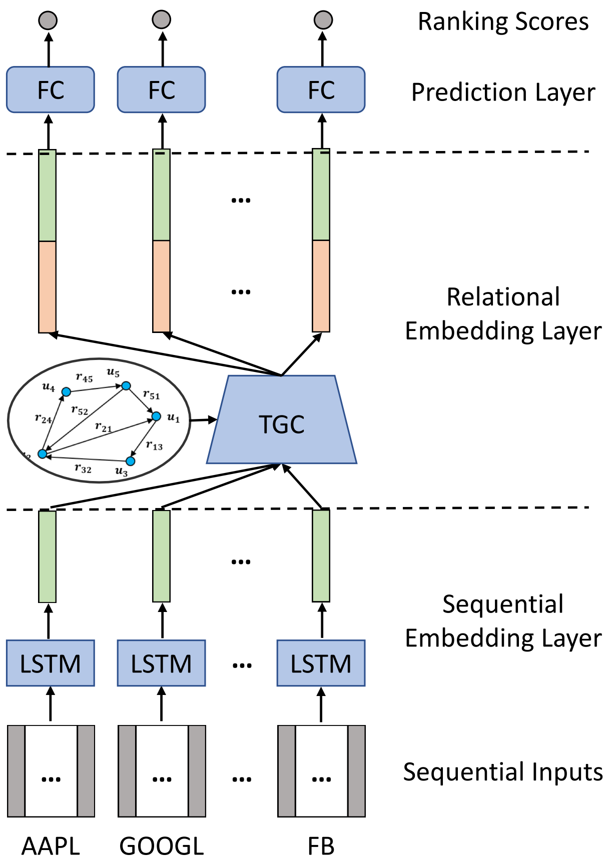


Figure 5.1: Relational stock ranking framework. It should be noted that the LSTM cells and FC units (Fully Connected layer) depicted in the same layer share the same parameters.

stock relations in a time-sensitive way. Finally, we feed the concatenation of sequential embeddings and relational embeddings to a fully connected layer to obtain the ranking score of stocks. To justify our proposed method, we employ it on two real-world markets, New York Stock Exchange (NYSE) and NASDAQ Stock Market (NASDAQ). Extensive back-testing results demonstrate that our RSR significantly outperforms SFM [184] with more than 115% improvements in return ratio.

The key contributions of this work are as follows.

- We propose a novel neural network-based framework, named *Relational Stock Ranking*, to solve the stock prediction problem in a learning-to-rank fashion.
- We devise a new component in neural network modeling, named *Temporal Graph Convolution*, to explicitly capture the domain knowledge of stock relations in a time-sensitive manner.

- We empirically demonstrate the effectiveness of our proposals on two real-world stock markets, NYSE and NASDAQ.

5.2 Related Work

In addition to *graph-based learning*, our work is directly related to the recent works on *stock prediction* and *knowledge graph embedding.*

5.2.1 Stock Prediction

Recent work on stock prediction can be separated into two main categories: *stock price regression* and *stock trend classification.* On one hand, Bao *et al.* [11] predicted the 1-day ahead closing price of stocks with historical prices as input. The authors viewed the historical price as a signal and decomposed the historical price into multiple frequencies with a Wavelet Transform. They then filtered out noises in the frequency domain with a Stacked Autoencoder (SAE) and fed the output of SAE to an LSTM to make the prediction. Zhang *et al.* [184] devised an extension of LSTM, which decomposes the historical prices into frequency domain with a Discrete Fourier Transform and equips each frequency with a memory state to capture the patterns in different frequencies. Instead of directly modeling the stock prices, Alberg and Lipton [6] used a combination of an LSTM and Multi-Layer Perception to predict the future trend of fundamental indicators of a company and trade the corresponding stock based on the predicted indicators.

On the other hand, Nguyen and Shirai [118] proposed a stock trend classification solution, which learns a topic distribution representation of each stock from posts mentioning it on stock message boards, and fed the topic representation into a Support Vector Machine to make the trend classification. Under a similar classification setting, another line of research is in classifying the trend of a stock from relevant financial news reports [135, 42, 41, 187, 79]. For instance, Zhao *et al.* [187] achieved it by constructing a event causality network of news reports and learning news embeddings from the causality networks, which is fed into a classification layer. Taking financial news as input as well, Hu *et al.* [79] devised a neural network-based solution, named *Hybrid Attention Networks*, which leverages a hybrid attention mechanism to attentively fuse multiple news reports mentioning a stock into a joint representation. In addition, textual contents mentioning stocks in social medial are also used to forecast the movement of stocks [98].

However, none of the existing work is able to incorporate the rank/relative order among stocks regarding the expected revenue. This may lead to suboptimal stock selections. Moreover, the existing work either totally ignores stock relations or heuristically models such relations. For instance, an intuitive consideration of sector-industry relation is to separately train a predictor for stocks under each sector [134]. To the best of our knowledge, our work is the first to leverage techniques of learning-to-rank to solve the stock prediction task and inject the stock relations into the learning framework with a new neural network component.

5.2.2 Knowledge Graph Embedding

In a similar line, modeling the relations of two entities (a.k.a. knowledge graph embedding) has been extensively studied in the literature of knowledge graphs. Bordes, *et al.* [21] represented entities and relations with embedding vectors and transferred entity embeddings through adding relation embedding. Similarly, Socher *et al.* [139] represented relations as matrices and transferred entity embeddings by matrix multiplication. Such techniques mainly focus on solving knowledge graph-oriented problems such as knowledge graph completion, while we target on a different problem setting of stock ranking. Though the idea of embedding propagation that revises stock sequential embeddings through stock relations is partially inspired by TransE, our TGC is more generic in terms of jointly capturing temporal properties and topologies of relations.

5.3 Preliminaries

5.3.1 Long Short-Term Memory

LSTM [78] networks have been widely used to process sequential data, such as the natural language [175], voice [61], and video [143]. LSTM is a special kind of Recurrent Neural Networks (RNNs) [58] that evolve hidden states through time to capture the sequential pattern of input data, *e.g.,* the dependency between words in a sentence. Compared to the vanilla RNN, which is known to suffer from vanishing gradients while trained with Back-Propagation Through Time (BPTT), LSTM adds cell states to store the long-term memory and capture the long-term dependency in a sequence.

Before providing the specific formulation of LSTM, we first describe the terms associated with LSTM. At each time-step t, $\mathbf{x^t} \in \mathbb{R}^D$ denotes an input vector (*e.g.,* embedding vector of the t-th word in a given sentence), where D is the input dimension. Vectors $\mathbf{c^t}$ and $\mathbf{h^t} \in \mathbb{R}^U$ denote the cell (memory) state vector and the hidden state vector, respectively, where U is the number of hidden units. Vector $\mathbf{z^t} \in \mathbb{R}^U$ is an information transformation module. Vectors $\mathbf{i^t}$, $\mathbf{o^t}$, and $\mathbf{f^t} \in \mathbb{R}^U$ denote the input, output, and forget gate, respectively. Formally, the transformation module, state vectors, and controlling gates are defined via the following equations:

$$
\begin{aligned}
\mathbf{z^t} &= tanh(\mathbf{W_z x^t} + \mathbf{Q_z h^{t-1}} + \mathbf{b_z}) \\
\mathbf{i^t} &= \sigma(\mathbf{W_i x^t} + \mathbf{Q_i h^{t-1}} + \mathbf{b_i}) \\
\mathbf{f^t} &= \sigma(\mathbf{W_f x^t} + \mathbf{Q_f h^{t-1}} + \mathbf{b_f}) \\
\mathbf{c^t} &= \mathbf{f^t} \odot \mathbf{c^{t-1}} + \mathbf{i^t} \odot \mathbf{z^t} \\
\mathbf{o^t} &= \sigma(\mathbf{W_o x^t} + \mathbf{W_h h^{t-1}} + \mathbf{b_o}) \\
\mathbf{h^t} &= \mathbf{o^t} \odot tanh(\mathbf{c^t}),
\end{aligned}
\tag{5.1}
$$

where $\mathbf{W_z}$, $\mathbf{W_i}$, $\mathbf{W_f}$, $\mathbf{W_o} \in \mathbb{R}^{U \times D}$, and $\mathbf{Q_z}$, $\mathbf{Q_i}$, $\mathbf{Q_f} \in \mathbb{R}^{U \times U}$ are mapping matrices; $\mathbf{b_z}$, $\mathbf{b_i}$, $\mathbf{b_f}$, and $\mathbf{b_o} \in \mathbb{R}^U$ are the bias vectors. The updating formulation can be understood as performing the following procedures: (1) calculate the information to be transformed from the input $\mathbf{x^t}$ to the memory states $\mathbf{c^t}$ by updating $\mathbf{z^t}$; (2) update the input gate $\mathbf{i^t}$ to control the information from $\mathbf{z^t}$ to $\mathbf{c^t}$; (3) update the forget gate $\mathbf{f^t}$ to decide how much information should be kept in the memory state; (4) refresh the memory state $\mathbf{c^t}$ by fusing the information flows from the input gate and memory gate; (5) update the output gate $\mathbf{o^t}$ to regulate the amount of information that can be outputted; and (6) update the hidden state $\mathbf{h^t}$. As can be seen, the memory state $\mathbf{h^t}$ only has linear adding interactions, which allows information to be unchanged during BPTT. Benefiting from the memory state, LSTM is capable of capturing the long-term dependency in the sequential data.

5.4 Relational Stock Ranking (RSR)

The typical problem setting of stock prediction (*i.e.,* price movement classification and price regression) is to learn a prediction function $\hat{y}^{t+1} = f(\mathbf{X^t})$ which maps a stock from the feature space to the target label space at time-step t. Matrix $\mathbf{X^t} = [\mathbf{x^{t-S+1}}, \cdots, \mathbf{x^t}]^T \in \mathbb{R}^{S\times D}$ represents the sequential input features, where D is the dimension of features at each time-step and S is the length of the sequence. Distinct from the typical problem setting of stock prediction, which treats different stocks as independent sequences, our target is to learn a ranking function $\mathbf{\hat{r}^{t+1}} = f(\mathcal{X}^t)$, which simultaneously maps a bunch of stocks to a ranking list. In the learned ranking list, stocks with higher ranking scores are expected to achieve higher investment revenue at time-step $t+1$. Assuming that we have N stocks, then $\mathcal{X}^t \in \mathbb{R}^{N\times S\times D} = [\mathbf{X^t_1}, \cdots, \mathbf{X^t_N}]^T$ is the collected features. In addition, we further associate the problem with a set of explicit stock relations (*e.g.,* supplier-consumer relations), which reflect the potential influence between different stocks. Given K types of relations, we encode the pairwise relation between two stocks as a multi-hot binary vector $\mathbf{a_{ij}} \in \mathbb{R}^K$ and represent the relation of all stocks as a tensor $\mathcal{A} \in \mathbb{R}^{N\times N\times K}$, of which the entry at the i-th row and j-th column is $\mathbf{a_{ij}}$.

In what follows, we first present the overall solution. We then elaborate our proposed *Temporal Graph Convolution* for handling stock relations, followed by discussing its connections to existing graph-based learning methods.

5.4.1 Framework

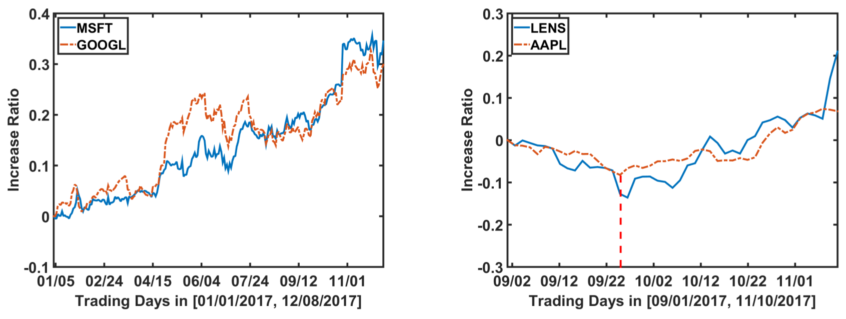
As illustrated in Figure 5.1, RSR contains three layers, named a sequential embedding layer, a relational embedding layer, and a prediction layer, which are elaborated as follows.

Sequential Embedding Layer. Considering the strong temporal dynamics of stock markets, it is intuitive to regard the historical status of a stock as the most influential factor to predict its future trend. As such, we first apply a sequential embedding layer to capture the sequential dependencies in the historical data. Since RNN has achieved significant performance to process sequential data [175, 143, 61] and demonstrated to be effective in recent stock prediction research [11, 184], we opt for RNN to learn the sequential embeddings. Among the various RNN models, such as vanilla RNN, LSTM, and Gated Recurrent Unit (GRU) [32], we choose LSTM owing to its ability to capture long-term dependency, which is of great importance to stock prediction. This is because many factors have long-term effects on a stock, such as the rise of interest rates, the release of annual reports, a rapid drop in its price, among the others. For example, if a stock has experienced a very rapid drop in its price, after that, the stock's price tends to exhibit an upward trend in the following days or weeks (*a.k.a.* the mean reversion phenomenon). As such, we feed the historical time series data of stock i at time-step t ($\mathbf{X_i^t}$) to a LSTM network and take the last hidden state ($\mathbf{h_i^t}$) as the sequential embedding ($\mathbf{e_i^t}$) of a stock (note that $\mathbf{e_i^t} = \mathbf{h_i^t}$), *i.e.,* we have,

$$\mathbf{E^t} = LSTM(\mathcal{X}^t), \tag{5.2}$$

where $\mathbf{E^t} = [\mathbf{e_1^t}, \cdots, \mathbf{e_N^t}]^T \in \mathbb{R}^{N \times U}$ denotes the sequential embeddings of all stocks, and U denotes the embedding size (*i.e.,* U is the number of hidden units in LSTM).

Relational Embedding Layer. We now consider how to model the influence between different stocks, especially the ones with explicit relations. Note that it can be seen as an injection of explicit domain knowledge (*i.e.,* stock relations) into the data-driven approach for sequential embedding learning. Here we



(a) Sector-industry relation (b) Supplier-consumer relation

Figure 5.2: Two examples of stock price history (normalized as increase ratio as compared to the first depicted trading day) to illustrate the impact of company relations on the stock price.

present two cases for illustration:

- If two companies are in the same sector or industry, they may exhibit similar trends in their stock prices, since they tend to be influenced by similar external events. Figure 5.2(a) shows two example stocks, MSFT (Microsoft Inc.) and GOOGL (Alphabet Inc.), both of which are in the same sector (Technology) and industry (Computer Software)[2]. As can be seen in Figure 5.2(a), the two stocks exhibit quite similar trends in terms of the change on price in 2017. Note that the depicted value on each day is the increase ratio at each trading day as compared to the price on the first observed day (*i.e.*, 09/02/2017).

- If two companies are partners in a supply chain, then the events of the upstream company may affect the stock price of the downstream company. Figure 5.2(b) shows an example to demonstrate the impact of such supplier-consumer relation, which shows the stock price change of Lens Technology Co Ltd after the release of iPhone 8 (09/22/2017)[3]. Since the Lens Technology Co Ltd is the supplier of the screen of iPhone, which was expected to be selling well, its stock price kept increasing in the following several weeks of 09/22/2017.

To capture such patterns in stock historical data, we devise a new component of neural network modeling, named *Temporal Graph Convolution*, to revise the sequential embeddings according to stock relations. It generates the relational embeddings $\overline{\mathbf{E^t}} \in \mathbb{R}^{N\times U}$ in a time-sensitive (dynamic) way, which is a key

[2]http://www.nasdaq.com/screening/companies-by-industry.aspx
[3]https://www.techradar.com/reviews/iphone-8-review

technical contribution of this work; it will be elaborated later in Section 5.4.2.

Prediction Layer. Lastly, we feed the sequential embeddings and revised relational embeddings to a fully connected layer to predict the ranking score of each stock. The ranked list of stocks recommended to buy is then generated based on the prediction scores.

To optimize the model, we propose an objective function that combines both pointwise regression loss and pairwise ranking-aware loss:

$$l(\mathbf{\hat{r}^{t+1}}, \mathbf{r^{t+1}}) = \left\|\mathbf{\hat{r}^{t+1}} - \mathbf{r^{t+1}}\right\|^2 + \alpha \sum_{i=0}^{N}\sum_{j=0}^{N} max(0, -(\hat{r_i}^{t+1} - \hat{r_j}^{t+1})(r_i^{t+1} - r_j^{t+1})), \tag{5.3}$$

where $\mathbf{r^{t+1}} = [r_1^{t+1}, \cdots r_N^{t+1}]$ and $\mathbf{\hat{r}^{t+1}} = [\hat{r}_1^{t+1}, \cdots, \hat{r}_N^{t+1}] \in \mathbb{R}^N$ are ground-truth and predicted ranking scores, respectively, and α is a hyperparameter to balance the two loss terms. Since we focus on identifying the most profitable stock to trade, we use the 1-day return ratio of a stock as the ground-truth rather than the normalized price used in previous work [184]. We will provide more details on computing the ground-truth in Section 5.5.1 in our data collection.

The first regression term punishes the difference between the scores of ground-truth and prediction. The second term is pair-wise max-margin loss [188], which encourages the predicted ranking scores of a stock pair to have the same relative order as the ground-truth. The similar max-margin loss has been used in several applications such as recommendation [167] and knowledge based completion [139], and has been demonstrated to have good performance in ranking tasks. Minimizing our proposed combined loss will force the prediction ranking scores to be close to: 1) the return ratios of stocks in terms of absolute values, and 2) the relative orders of return ratios among stocks, so as to facilitate investors making better investment decisions. On one hand, the correct relative order of stocks could help to select the investment targets (*i.e.,* the top ranked stocks). On the other hand, the accurate prediction of return ratio would facilitate decision on the timing of investment since the top ranked stocks are valuable targets only when the return ratios would largely increase.

5.4.2 Temporal Graph Convolution

Given N stocks with their sequential embeddings $\mathbf{E^t} \in \mathbb{R}^{N \times U}$ (*i.e.,* the output of sequential embedding layer) and their multi-hot binary relation encodings $\mathcal{A} \in \mathbb{R}^{N \times N \times K}$, the aim of *Temporal Graph Convolution* is to learn the revised embeddings $\overline{\mathbf{E^t}} \in \mathbb{R}^{N \times U}$ that encode the relation information. Instead of directly presenting the formulation of TGC, we detail how we design the component to shed some lights on its rationale. Lastly, we discuss its connection with existing graph-based learning methods.

a) Uniform Embedding Propagation. Our first inspiration comes from link analysis research, where in a graph, the impact of a vertex on another one can be captured by propagating information on the graph. A well-known example is the PageRank [125] method that propagates the importance score of a vertex to its connected vertices. Since a stock relation encodes certain similarity information between two connected stocks, we consider relating their embeddings through a similar propagation process as in link analysis:

$$\overline{\mathbf{e_i^t}} = \sum_{\{j|sum(\mathbf{a_{ji}})>0\}} \frac{1}{d_j} \mathbf{e_j^t}, \tag{5.4}$$

where $sum(\mathbf{a_{ji}})$ is the sum of all elements in the relation vector $\mathbf{a_{ji}}$ (recall that $\mathbf{a_{ji}}$ is a multi-hot binary vector where each element denotes whether the corresponding type of relation exists between j and i). The condition $sum(\mathbf{a_{ji}}) > 0$ ensures that only stocks have at least one relation will be considered. d_j is the number of stocks satisfying the condition $sum(\mathbf{a_{ji}}) > 0$. After such a propagation in the embedding space, the relational embedding $\overline{\mathbf{e_i^t}}$ encodes the impacts coming from other stocks that have relations with stock i at time t.

b) Weighted Embedding Propagation. Considering that different relations between two stocks may have varying impacts on their prices, we apply a non-uniform coefficient when propagating the embeddings:

$$\overline{\mathbf{e_i^t}} = \sum_{\{j|sum(\mathbf{a_{ji}})>0\}} \frac{g(\mathbf{a_{ji}})}{d_j} \mathbf{e_j^t}, \tag{5.5}$$

where $g(\mathbf{a_{ji}})$ is a mapping function that aims to learn the impact strength of the relations in $\mathbf{a_{ji}}$, and we term it as the *relation-strength function*. As an example, suppose we have two relations named *supplier_customer* and *same_industry*, and three stocks j, i, and k. Given that stock j is a supplier of stock i while stock k is in the same industry as stock i, we can encode their relations as two different vectors: $\mathbf{a_{ji}} = [1, 0]$ and $\mathbf{a_{ki}} = [0, 1]$. We can see that by feeding different relation vectors into a learnable relation-strength function for different stock pairs, we allow the embedding propagation process to account for both the topology of relation graph and the semantics of relations.

c) Time-aware Embedding Propagation. A limitation of the above weighted propagation process is that the *relation-strength function* returns a fixed weight for a given relation vector $\mathbf{a_{ji}}$ regardless the evolution across different time-steps. As stock market is highly dynamic such that the status of a stock and the strength of a relation are continuously evolving, the assumption that a relation vector has a static weight limits its modeling fidelity. For instance, in the previous example of Figure 5.2(b), the *supplier_customer* relation between Apple Inc. and Lens Technology Co Ltd has a larger impact on Lens's stock price in the period of releasing new version of iPhone than usual. To address this limitation, we propose to encode the temporal information into the relation-strength function and define the *Time-aware Embedding Propagation* process as follows:

$$\overline{\mathbf{e_i^t}} = \sum_{\{j|sum(\mathbf{a_{ji}})>0\}} \frac{g(\mathbf{a_{ji}}, \mathbf{e_i^t}, \mathbf{e_j^t})}{d_j} \mathbf{e_j^t}, \tag{5.6}$$

which takes the sequential embeddings (note that they are time-sensitive) into account to estimate the strength of a relation. Besides encoding the temporal information, another benefit of such a design is that the sequential embedding also encodes the stock information. This allows the relation-strength function to estimate the impact of a relation vector based on the stocks of concern, which is very desirable.

Next we describe two designs of the time-sensitive relation-strength function, which differ in whether to model the interaction between two stocks in an explicit or implicit manner.

- **Explicit Modeling**. For the explicit model, we define the relation strength function as:

$$g(\mathbf{a_{ji}}, \mathbf{e_i^t}, \mathbf{e_j^t}) = \underbrace{\mathbf{e_i^t}^T \mathbf{e_j^t}}_{\text{similarity}} \times \underbrace{\phi(\mathbf{w}^T \mathbf{a_{ji}} + b)}_{\text{relation importance}}, \tag{5.7}$$

  where $\mathbf{w} \in \mathbb{R}^K$ and b are model parameters to be learned; and ϕ is an activation function[4]. The relation strength of $\mathbf{a_{ji}}$ is determined by two terms – *similarity* and *relation importance*. Specifically, the first term measures the similarity between the two stocks at the current time-step. The intuition is that the more similar the two stocks are at the current time, it is more likely that their relations will impact their prices in the near future. We use inner product to estimate the similarity, inspired by its effectiveness in modeling the similarity (interaction) between two entities (embeddings) in Collaborative Filtering [75]. The second term is a nonlinear regression model on the relations, where each element in $\mathbf{w}$ denotes the weight of a relation in general and b is a bias term. Since both terms of this function are directly interpretable, we call it *Explicit Modeling*.

- **Implicit Modeling**. In this design, we feed the sequential embeddings and the relation vector into a fully connected layer to estimate the relation strength:

$$g(\mathbf{a_{ji}}, \mathbf{e_i^t}, \mathbf{e_j^t}) = \phi(\mathbf{w}^T [\mathbf{e_i^t}^T, \mathbf{e_j^t}^T, \mathbf{a_{ji}}^T]^T + b), \tag{5.8}$$

  where $\mathbf{w} \in \mathbb{R}^{2U+K}$ and b are model parameters to be learned; ϕ is an activation function same as the one in Equation (5.8). Then we normalize the outputs using a softmax function, which also endows it with more non-linearities. Since this way of interaction is implicitly captured by the parameters, we call it *Implicit Modeling*.

[4]Note that we employ the *leaky rectifier* [106] with a slope of 0.2 as the activation function in our implementation.

5.5 Data Collection

Most existing works evaluate stock prediction on dozens of stocks, and therefore lacks a large-scale stock dataset for extensive evaluations. As such, we decide to construct a large-scale data by ourselves, which is accessible through: https://github.com/hennande/Temporal_Relational_Stock_Ranking. Specifically, we collect the stocks from the NASDAQ and NYSE markets that have transaction records between 01/02/2013 and 12/08/2017, obtaining $3,274$ and $3,163$ stocks respectively. Note that we select these two markets for their representative properties: NASDAQ is more volatile, whereas NYSE is more stable [136]. Furthermore, we perform a filtering on the stocks by retaining the stocks satisfying the two conditions that: 1) they have been traded on more than 98% of trading days since 01/02/2013; and 2) they have never been traded at less than five dollars per share during the collection period. It should be noted that the first condition is based on concerns that intermittent sequences may bring in abnormal patterns; while the second condition ensures that the selected stocks are not penny stocks[5], which are too risky for general investors as suggested by the U.S. Securities and Exchange Commission. This results in $1,026$ NASDAQ and $1,737$ NYSE stocks for our experiments. For these stocks, we collect three kinds of data: 1) historical price data and 2) Wiki relations between their companies such as supplier-consumer relation and ownership relation. Next, we present the details of these data.

5.5.1 Sequential Data

Following [184], we set the prediction frequency at daily-level. Under our problem formulation, we aim to predict a ranking list of stocks for the following trading day, based on the daily historical data of the last S trading days. As the return ratio of a stock indicates the expected revenue of the stock, we set the ground-truth ranking score of stock i as its 1-day return ratio $r_i^{t+1} = (p_i^{t+1} - p_i^t)/p_i^t$ where p_i^t is the closing price at day t. To calculate the ground-truth, we first collect the daily closing price of each stock ranging from 01/02/2013 and 12/08/2017. After the collection, we normalize the price of

[5]https://www.sec.gov/fast-answers/answerspennyhtm.html

Table 5.2: **Statistics of the sequential data.**

Market	Stocks#	Training Days# 01/02/2013 12/31/2015	Validation Days# 01/04/2016 12/30/2016	Testing Days# 01/03/2017 12/08/2017
NASDAQ	1,026	756	252	237
NYSE	1,737	756	252	237

Table 5.3: **Statistics of Wiki relation data in the NASDAQ and NYSE datasets.**

	Wiki Relation	
	Relation Types#	Relation Ratio (Pairwise)
NASDAQ	42	0.21%
NYSE	32	0.30%

each stock by dividing it by its maximum value throughout the entire 2013-2017 dataset. In addition to the normalized closing price, we calculate four more sequential features, *i.e.,* the 5, 10, 20, and 30 days moving averages which represent the weekly and monthly trends. Following existing works of stock prediction [184], we chronologically separate the sequential data into three time periods for training (2013-2015), validation (2016), and evaluation (2017), respectively, and summarize the basic statistics in Table 5.2. As can be seen, there are 756, 252, and 237 trading days in the training, validation, and evaluation set, respectively.

5.5.2 Wiki Company-based Relations

As rich sources of entity relations, knowledge bases contain company entities and company relations, which might reflect the impact across stocks. As such, we extract the first-order and second-order company relations from Wikidata [156], one of the biggest and most active open domain knowledge bases with more than 42 million items (*e.g., Alphabet Inc.*) and 367 million statements (*e.g., Alphabet Inc.; founded by; Larry Page*) in the format of (*subject; predicate; object*)[6]. As shown in Figure 5.3, company i has a first-order relation with j if there is a statement that has i and j as the subject and object, respectively. Companies i and j have a second-order relation if they have statements sharing the same object, such as *Boeing Inc.* and *United Airlines, Inc.* have different statements linking to *Boeing 747*. After an exhausted exploration of a recent dump of Wikidata (01/05/2018), we obtain 5 and 53 types of first-order and second-order relations, respectively[7]. The detailed description of these relations is elaborated

[6]https://www.mediawiki.org/wiki/Wikibase/DataModel/JSON

[7]We manually filter out less informative relations such as *located at the same timezone.*

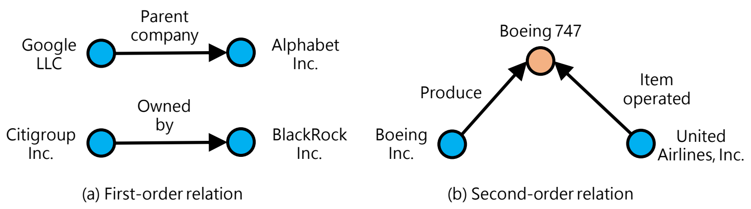


Figure 5.3: Examples of the first-order and second-order company relations extracted from Wikidata.

in Section A.1 at the end of this thesis proposal. We then summarize the count of relation types and the ratio of stock pairs with at least one Wiki company-based relation in Table 5.3. As can be seen, there are 42 and 32 types of company relations occurring between stock pairs in NASDAQ and NYSE, respectively.

5.6 Experiment

To the best of our knowledge, our work is the first study to incorporate stock relations into the models for stock prediction, especially neural network-based ones. As such, in this section, we conduct experiments with the aim of answering the following research questions:

1. **RQ1**: How is the utility of formulating stock prediction as a ranking task as compared to the state-of-the-art stock prediction solutions in regression formulation?

2. **RQ2**: Do stock relations enhance the neural network-based solution for stock prediction? How is the effectiveness of our proposed TGC component compared to conventional graph-based learning?

3. **RQ3**: How does our proposed RSR solution perform under different back-testing strategies?

5.6.1 Experimental Setting

Evaluation Protocols. Following [43], we adopt a daily buy-hold-sell trading strategy to evaluate the performance of stock prediction methods regarding the revenue. On each trading day $t + 1$ during the testing period (from 01/03/2017 to 12/08/2017), we simulate a trader using a stock prediction method to trade in the following way:

1. **When the market closes on day** t: The trader uses the method to get the prediction, a ranking list with predicted return ratio of each stock. The trader buys the stock with the highest expected revenue (*i.e.,* top-1).

2. **When the market closes on trading day** $t+1$: The trader sells the stock purchased on day t.

In calculating the cumulative investment return ratio, we follow several simple assumptions: (1) The trader spends the same amount of money (*e.g.,* 50 thousand dollars) on every trading day. We make this assumption to eliminate the temporal dependency of the testing procedure for a fair comparison. (2) The market is always sufficiently liquid such that the buying order gets filled at the closing price of day t and the selling price is the closing price of day $t+1$. (3) The transaction costs are ignored since the costs for trading US stocks through brokers are quite cheap no matter charging by trades or shares. For instance, Fidelity Investments and Interactive Brokers charge only 4.95 dollars per trade and 0.005 dollar per share, respectively[8].

Since the target is to accurately predict the return ratio of stocks and appropriately rank the relative order of stocks, we employ three metrics, Mean Square Error (MSE), Mean Reciprocal Rank (MRR), and the cumulative investment return ratio (IRR), to report the model performance. MSE has been widely used for evaluating regression tasks such as stock price prediction [92, 115, 184]. We thus calculate the MSE over all stocks on every trading day within the testing period. MRR [141] is a widely used metric for ranking performance evaluation. Here, we calculate the average reciprocal rank of the selected stock over the testing days. Since directly reflecting the effect of stock investment, IRR is our main metric, which is calculated by summing over the return ratios of the selected stock on each testing day. Smaller value of MSE (≥ 0) and larger value of MRR ($[0,1]$) and IRR indicate better performance. For each method, we repeat the testing procedure five times and report the average performance to eliminate the fluctuations caused by different initializations.

[8]https://www.stockbrokers.com/guides/commissions-fees

Methods. We compare with the following stock price prediction baselines with *regression* formulation:

- **SFM** [184]: This method is the state-of-the-art stock price prediction method. It takes the historical closing prices as input and decomposes the prices into signals of different frequencies with a Discrete Fourier Transform (DFT). It then feeds the DFT coefficients into an extended LSTM with separate memory states for different frequencies to learn the frequency-aware sequential embeddings, which are fed into a FC layer to make the prediction.

- **LSTM** [11]: This method is the vanilla LSTM, which operates on the sequential data including closing prices and moving averages of 5, 10, 20, and 30 days, to obtain a sequential embedding; and then a FC layer is used to make prediction of the return ratio.

It should be noted that we ignore the potential baselines based on time-series models and shallow machine learning models, since they have been reported to be less effective than **SFM** and **LSTM** in several previous works [184, 11, 79]. Moreover, we also compare with several methods with *ranking* formulation:

- **Rank_LSTM**: We remove the relational embedding layer of the proposed RSR to obtain this method, *i.e.,* this method ignores stock relations.

- Graph-based ranking (**GBR**): We add the graph regularization term to the loss function of **Rank_LSTM**, which smooths predicted return ratios over the graph of *stock relations*. In the graph, we connect a pair of vertices (*i.e.,* stocks) having at least one type of relations.

- **GCN** [89]: GCN is the state-of-the-art graph-based learning method. We obtain this method by replacing the TGC layer of our proposed RSR with a GCN layer. The graph of *stock relations* in **GBR** is fed into the **GCN** layer.

- **RSR_E**: One case of our proposed RSR with explicit modeling of stock relation (Equation (5.7)) in the TGC layer.

- **RSR_I**: Another case of RSR with implicit modeling of stock relation (Equation (5.8)) in the TGC layer.

Table 5.4: Performance comparison between the solutions with regression formulation (SFM and LSTM) and ranking formulation (Rank_LSTM) on the NASDAQ and NYSE datasets.

	NASDAQ			NYSE		
	MSE	MRR	IRR	MSE	MRR	IRR
SFM	5.20e-4±5.77e-5	2.33e-2±1.07e-2	-0.25±0.52	3.81e-4±9.30e-5	**4.82e-2±4.95e-3**	0.49±0.47
LSTM	3.81e-4±2.20e-6	3.64e-2±1.04e-2	0.13±0.62	2.31e-4±1.43e-6	2.75e-2±1.09e-2	-0.90±0.73
Rank_LSTM	**3.79e-4±1.11e-6**	**4.17e-2±7.50e-3**	**0.68±0.60**	**2.28e-4±1.16e-6**	3.79e-2±8.82e-3	**0.56±0.68**

Rank_LSTM, **RSR_E**, and **RSR_I** are different variants of our method. **Rank_LSTM** is used to compare with existing stock prediction methods to answer **RQ1**. **RSR_E** and **RSR_I** are compared with the existing graph Laplacian regularization method (**GBR**) and the graph representation learning method (**GCN**) to answer **RQ2**.

Parameter Settings. We implement the models with TensorFlow[9] except **SFM** of which we use the original implementation[10]. We employ grid search to select the optimal hyperparameters regarding IRR for all methods. For **SFM**, we follow the original setting in [184], optimizing it by RMSProp with a learning rate of 0.5, and tuning the number of frequencies and hidden units within $\{5, 10, 15\}$ and $\{10, 20, 30\}$, respectively. For all other methods, we apply the Adam [87] optimizer with a learning rate of 0.001. We tune two hyperparameters for **LSTM**, the length of sequential input S and the number of hidden units U, within $\{2, 4, 8, 16\}$ and $\{16, 32, 64, 128\}$, respectively. Besides S and U, we further tune α in Equation (5.3), which balances the point-wise and pair-wise terms; specifically, we tune α within $\{0.1, 1, 10\}$ for **Rank_LSTM**, **GCN**, **RSR_E**, and **RSR_I**. We further tune the λ of the regularization term in **GBR** within $\{0.1, 1, 10\}$. Note that we report the average performance of five different runs.

5.6.2 Study of Stock Ranking Formulation (RQ1)

We first investigate whether stock ranking is a promising formulation for stock prediction by comparing the performance of methods that only consider sequential features (*i.e.,* without consideration of stock relations). Table 5.4 summarizes the performance of the baselines in *regression* fashion and our basic **Rank_LSTM** model for stock ranking *w.r.t.* MSE, MRR, and IRR. From the Table, we have the following observations:

[9]`https://www.tensorflow.org/`
[10]`https://github.com/z331565360/State-Frequency-Memory-stock-prediction`

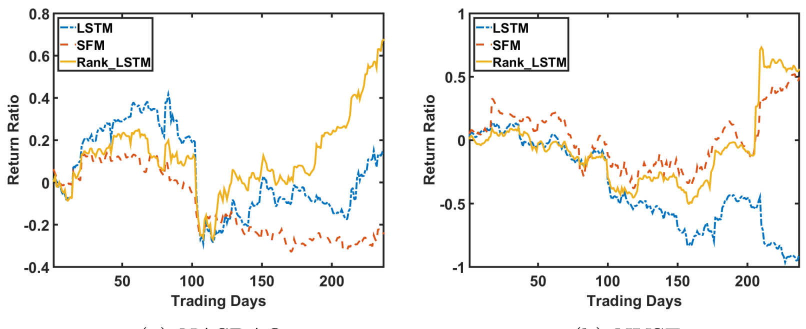


Figure 5.4: Performance comparison of Rank_LSTM, SFM, and LSTM regarding IRR.

- **Rank_LSTM** outperforms both **SFM** and **LSTM** on the two markets with great improvement *w.r.t.* IRR (>14%). It verifies the advantage of the stock ranking solutions and answers **RQ1** that stock ranking is a promising formulation for stock prediction. Moreover, it indicates the potential of advanced learning-to-rank techniques in solving the stock prediction task.

- However, **Rank_LSTM** fails to consistently beat **SFM** and **LSTM** regarding all evaluation measures, its performance on NYSE *w.r.t.* MRR is worse than **SFM**. The reason could be attributed to minimizing the combination of point-wise and pair-wise losses, which would lead to a tradeoff between accurately predicting absolute value of return ratios and their relative order.

- The performance *w.r.t.* IRR varies a lot under different runs of a method. It is reasonable since the absolute value of daily return ratio varies from 0 to 0.98 in our dataset, which means that a tiny switch of the top 2 ranked stocks may lead to a huge change of the IRR. Such results also indicate that there is a need for further study on the learning to rank techniques that emphasizes the top-ranked stocks

- The performance of **LSTM** on the NYSE market *w.r.t.* IRR is unexpectedly bad. We repeat the parameter tuning and testing procedure several times and find that **LSTM** could achieve better performance (with IRR value between 0.1 and 0.2) with other settings of hyperparameters. However, the selected setting always beats the others on the validation. This result indicates the potential difference between the validation and testing.

Figure 5.4 illustrates the procedure of back-testing regarding the cumulative return ratios. As can be seen, in all cases, the curves are volatile, which indicates that selecting only one stock from more than 1,000 is a highly risky operation. Consequently, it also suggests the importance of introducing risk-oriented criteria into stock ranking tasks in the future.

5.6.3 Impact of Stock Relations (RQ2)

We next study the whether stock relations are helpful for stock ranking. Table 5.5 shows the results of considering the Wiki relation of stocks. From the Table, we observe that:

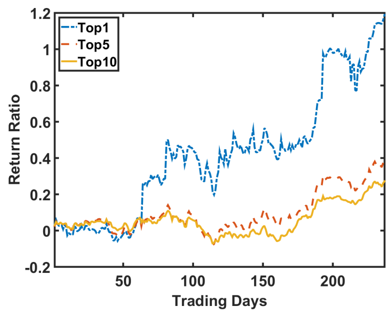
- In most cases, methods that incorporate stock relations, *i.e.,* **GBR**, **GCN**, **RSR_E**, and **RSR_I**, outperform **Rank_LSTM** *w.r.t.* IRR. Since **Rank_LSTM** is the building block of the other compared methods, this result indicates that encoding stock relations would lead to more accurate ranking of stocks and facilitate profitable investment.

- In all cases, our proposed framework RSR with either explicit modeling or implicit modeling of stock relations (*i.e.,* **RSR_E** and **RSR_I**) achieve better performance than **GBR** and **GCN** *w.r.t.* IRR. Note that the key difference is that **RSR_E** and **RSR_I** encode stock relations with the proposed *TGC* component instead of conventional graph-based learning techniques used in **GBR** and **GCN**. As such, this result validates the effectiveness of the *TGC* component, which models local smoothness in a time-sensitive manner.

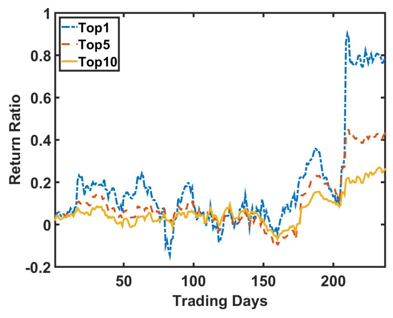
- We observe that the performance of different methods *w.r.t.* different evaluation measures is inconsistent. In most cases, the performance of different methods is comparable to each other *w.r.t.* MSE and MRR while various in a wide range *w.r.t.* IRR. We speculate the reason as tuning the hyperparameters *w.r.t.* IRR, which focuses more on correct ranking on testing days with high return ratios. For instance, the correct prediction on a trading day with ground truth return ratio of 0.5 would lead to higher IRR than the correct predictions in ten trading days with a return ratio of 0.01. As such, a model that achieves better IRR could achieve suboptimal MSE and MRR.

Table 5.5: Performance comparison among relational ranking methods with Wiki relations on the NASDAQ and NYSE datasets.

	NASDAQ			NYSE		
	MSE	MRR	IRR	MSE	MRR	IRR
Rank_LSTM	**3.79e-4±1.11e-6**	**4.17e-2±7.50e-3**	0.68±0.60	2.28e-4±1.16e-6	3.79e-2±8.82e-3	0.56±0.68
GBR	3.80e-4±2.40e-7	3.32e-2±4.50e-3	0.33±0.34	**2.26e-4±4.20e-7**	3.64e-2±5.35e-3	0.65±0.27
GCN	**3.79e-4±9.70e-7**	3.24e-2±3.21e-3	0.11±0.06	**2.26e-4±6.60e-7**	3.99e-2±1.03e-2	0.74±0.30
RSR_E	3.80e-4±7.20e-7	3.94e-2±8.15e-3	0.81±0.85	2.29e-4±2.77e-6	4.28e-2±6.18e-3	**0.96±0.47**
RSR_I	**3.79e-4±6.60e-7**	4.09e-2±5.18e-3	**1.19±0.55**	**2.26e-4±1.37e-6**	**4.58e-2±5.55e-3**	0.79±0.34



(a) NASDAQ



(b) NYSE

Figure 5.5: Comparison on back-testing strategies (Top1, Top5, and Top10) *w.r.t.* IRR based on the prediction of RSR_I.

5.6.4 Study on Back-testing Strategies (RQ3)

We next investigate the performance of our proposed methods under three different back-testing strategies, named **Top1**, **Top5**, and **Top10**, buying stocks with top-1, 5, 10 highest expected revenue, respectively. For instance, with the back-testing strategy of **Top10**, we equally split our budget to trade the top-10 ranked stocks on each testing day. Note that we accordingly calculate the IRR by averaging the return ratio of the 10 selected stocks on each testing day. Figure 5.5 illustrates the performance comparison of these strategies using the predictions of **RSR_I** only, which implicitly models stock relations. We select **RSR_I** instead of **RSR_E** for the reason that the result of **RSR_I** is more stable (smaller standard deviation as shown in Table 5.5). From Figure 5.5, we have the following observations:

- In the other cases, the performance of **Top1**, **Top5**, and **Top10** on most testing days follows the order of **Top1** > **Top5** > **Top10**, *i.e.,* **Top1** and **Top10** achieve the highest and lowest IRR, respectively. The reason could be that the ranking algorithm could accurately rank the relative order of stocks regarding their future return ratios. Once the order is accurate, buying and selling the stock with higher expected profit (*e.g.,* the top-1 ranked one) would achieve higher cumulative return ratio.

- Under the back-testing strategy **Top1**, the curve of IRR has sharper fluctuations that those of **Top5** and **Top10**. This result indicates that **Top1** would be a risky investment strategy, *i.e.,* it would lead to large changes *w.r.t.* return ratios. In other words, for risk-sensitive stock investors, it is suggested to trade on the Top-5 and Top-10 ranked stocks to distribute the risk. Furthermore, this result suggests the need for further study on the joint modeling of return ratio and risk in stock prediction.

To further investigate the effectiveness of the proposed method in facilitating stock trading, we further compare the back-testing performance of our method with the following baselines:

- **S&P500**: The Standard & Poor's 500 Index is a composite market index of 500 U.S. stocks with the largest capitalization. Note that following market index is a strong investment strategy in U.S. stock market[11].

- **DJI**: Dow Jones Industrial Average Index is another widely referred composite market index in the U.S. stock market.

- **Market**: It is one of the ideal investment strategies, which trades stocks with the highest return ratio (*e.g.,* Top10) in the testing period from the whole market. It should be noted that we rank stocks by return ratios calculated from the prices on the last day and the first day of the testing period.

- **Selected**: It is another ideal investment strategy that selects the stocks with the highest return ratio among the stocks traded by the proposed method in the testing period.

Table 5.6 shows the performance of the compared investment strategies *w.r.t.* IRR. From the results, we have the following observations:

- In the testing period, **S&P500** and **DJI** increase about twenty percent which indicates that the stock market is bullish. Considering that it would be easier to achieve better performance *w.r.t.* IRR in a bullish market, it suggests the future exploration of the proposed method in a bearish market.

[11]From 2008 to 2017, the S&P 500 achieved a return ratio of 125.8%, beating most of the portfolios of hedge funds.

Table 5.6: Performance of RSR_I as compared to market indices and ideal investment strategies *w.r.t.* IRR.

	NASDAQ			NYSE		
	Top1	Top5	Top10	Top1	Top5	Top10
RSR_I	1.19	0.40	0.27	1.06	0.18	0.26
Market	**3.40**	**2.36**	**1.99**	**2.42**	**1.90**	**1.47**
Selected	1.63	0.81	1.10	2.24	1.78	1.39
S&P 500	0.17					
DJI	0.22					

- Our proposed method achieves a higher IRR as compared to **S&P500** and **DJI** in all the cases. Since following market indices are competitive investment strategies, this result validates the effectiveness of the proposed method.

- The performance of the proposed method presents a significant gap below **Market** and **Selected**. This result is acceptable since to accurately select the stocks that perform the best in a testing of one year is a very difficult problem. Furthermore, it also presents a great opportunity for research of stock prediction methods.

5.7 Conclusions

In this work, we formulated stock prediction as a ranking task and demonstrated the potential of employing the learning-to-rank methods for predicting stocks. To tackle the problem, we proposed a *Relational Stock Ranking* framework. The core of the framework is a neural network modeling component, named *Temporal Graph Convolution*, which can handle the impact among different stocks by encoding stock relations in a time-sensitive way. Note that the TGC extends existing graph convolution models by enabling the modeling of temporal vertex features. Experimental results on NASDAQ and NYSE demonstrate the effectiveness of our solution — the RSR framework outperforms two well known market indices S&P 500 and DJI with significantly higher return ratio.

Chapter 6

Dynamic Local Smoothness via Graph Adversarial Training

In this chapter, we introduce our graph adversarial training method, which stabilize the promising graph neural networks by encouraging the model to defense adversarial perturbations propagated over graph.

6.1 Introduction

Owing to the extraordinary representation ability, deep neural networks become prevalent models for graph-based learning [176, 157, 89, 119, 155]. Despite promising performance, we argue that graph neural networks are vulnerable to small but intentional perturbations on the input features [197], and this could be even more serious than the standard neural networks that do not model graph structure. The reasons are twofold: 1) graph neural networks also optimize the supervised loss on labeled data, thus it will face the same vulnerability issue as the standard neural networks [60], and 2) the additional smoothness constraint will exacerbate the impact of perturbations, since smoothing across connected nodes[1] would aggregate the impact of perturbations from nodes connected to the target node (i.e., the node that we apply perturbations with the aim of changing its prediction). Figure 6.1 illustrates the impact of perturbations on

[1]In the following sections, we interchangeably use node and example.

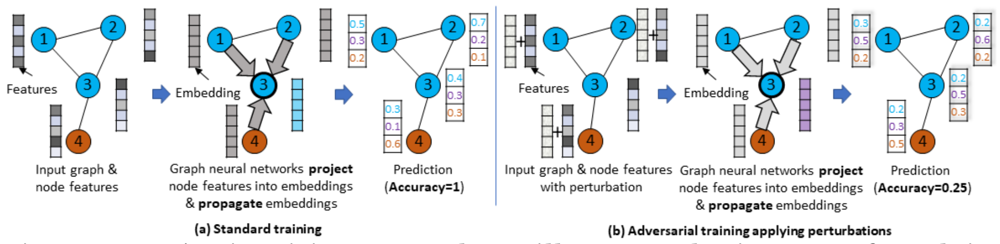

Features
Input graph & node features
Embedding
Graph neural networks project node features into embeddings & propagate embeddings
Prediction (Accuracy=1)
(a) Standard training
Input graph & node features with perturbation
Graph neural networks project node features into embeddings & propagate embeddings
Prediction (Accuracy=0.25)
(b) Adversarial training applying perturbations

Figure 6.1: An intuitive example to illustrate the impact of applying perturbations in the input node features to the prediction of graph neural networks. Here the model implements the graph smoothness constraint via propagating node embeddings over the graph. On the right, the model propagates the applied perturbations on the connected nodes of the target node 3, leading to a wrong prediction. Moreover, the perturbations on node 1 and 2 directly lead to the wrong associated predictions like in the standard neural networks.

node features with an intuitive example of a graph with 4 nodes. A graph neural network model predicts node labels (3 in total) for both the clean input features and features with applied perturbations, respectively. Here perturbations are intentionally applied to the features of nodes 1, 2, 4. Consequently, the graph neural network model is fooled to make the wrong predictions on nodes 1 and 2 as with standard neural networks. Moreover, by propagating the node embeddings, the model aggregates the perturbations to node 3, from which its prediction is also affected. In real-world applications, small perturbations like the update of node features may frequently happen, but should not change the predictions much. As such, we believe that there is a strong need to stabilize the graph neural network models during training.

Adversarial Training (AT) is a dynamic regularization technique that proactively simulates the perturbations during the training phase [60]. It has been empirically shown to be able to stabilize neural networks, and enhance their robustness against perturbations in standard classification tasks [93, 111]. Therefore, employing a similar approach to that of AT on a graph neural network model would also be helpful to the model's robustness. However, directly employing AT on graph neural network is insufficient, since it treats examples as independent of each other and does not consider the impacts from connected examples. As such, we propose to investigate a new adversarial training method, named *Graph Adversarial Training* (GAT), which learns to construct and resist perturbations by taking the graph structure into account.

The key idea of GAT is that, when generating perturbations on a target example, it maximizes the divergence between the prediction of the target example and its connected examples. That is, the adversarial perturbations would attack the graph smoothness constraint as much as possible. Then, GAT updates model parameters by additionally minimizing a *graph adversarial regularizer*, aiming to reduce the prediction divergence between the perturbed target example and its connected examples. Through this way, GAT can resist the worst-case perturbations on graph-based learning and enhance model robustness. To efficiently calculate the adversarial perturbations, we further devise a linear approximation method based on back-propagation.

To demonstrate GAT, we employ it on a well-established graph neural network model, *Graph Convolutional Network* (GCN) [89], which implements the smoothness constraint by performing embedding propagation. We study the method's performance on node classification, one of the most popular tasks on graph-based learning. Extensive experiments on three public benchmarks (two citation graphs and a knowledge graph) verify the strengths of GAT. As compared to normal training on GCN, GAT leads to 4.51% improvement in accuracy. Moreover, the improvements on less popular nodes (with a small degree) are more significant, highlighting the necessity of performing AT with the graph structure considered.

The main contributions of this work are summarized as:

- We formulate *Graph Adversarial Training*, a new optimization method for graph neural networks that can enhance the model's robustness against perturbations on node input features.

- We devise a *graph adversarial regularizer* that encourages the model to generate similar predictions on the perturbed target example and its connected examples, and develop an efficient algorithm to construct perturbations.

- We demonstrate the effectiveness of GAT on GCN, conducting experiments on three datasets which show that our method could achieve state-of-the-art performance for node classification.

6.2 Related Work

In addition to graph-based learning, the existing research on adversarial learning is closely related to this work.

6.2.1 Adversarial Learning

Adversarial Training. In order to tackle the vulnerability to intentional perturbations of deep neural networks, researchers have proposed adversarial training which is an alternative minimax process [145]. The adversarial training methods augment the training process by dynamically generating adversarial examples from clean examples with perturbations to maximally attack the training objective, and then learn over these adversarial examples by minimizing an additional regularization term [113, 171, 60, 111, 112, 100, 149, 130]. The adversarial training methods mainly fall into two categories: *supervised* and *semi-supervised* ones, regarding the target of the training objective. In the supervised learning tasks such as visual recognition [60], supervised loss [113, 171, 60] and its surrogates [100, 149, 130] over adversarial examples are designed as the target of the maximization and minimization. For semi-supervised learning where partial examples are labeled, divergence of predictions for inputs around each examples is adopted as the target. Generally speaking, the philosophy of adversarial training methods is to smooth the prediction around individual inputs in a dynamical fashion.

Our work is inspired by these adversarial training methods. In addition to the local smoothness of individual examples, our method further accounts for relation between examples (*i.e.,* the graph structure) in the target of the minimax process so as to learn robust classifiers that is able to predict smoothly over the graph structure. To the best of our knowledge, this is the first attempt to incorporate graph structure in adversarial training.

Another emerging research topic related to our work is the generation of adversarial perturbations to attack the neural graph-based learning models where [35] and [197] are the only published work. However, methods in [35] and [197] are not suitable for constructing adversarial examples in graph adversarial training. This is because these methods generate a new graph as the adversarial example for each individual node, *i.e.,* they would generate N graphs when the number of nodes is N leading to unaffordable memory overhead. In this work, we devise an efficient method to generate adversarial examples for graph adversarial training.

Generative Adversarial Networks. Generative adversarial networks (GAN) is a machine learning framework with two different networks as a generator and a discriminator playing minimax game on generating and detecting fake examples. Recently, several GAN-based models are proposed to learn graph embeddings, which either generate fake nodes and edges to augment embedding learning [159, 40] or smooth the leaned embeddings to follow a prior distribution [133, 181, 126, 36]. However, using two different networks inevitably doubles the loads of computation of model training and the tuning of parameters of GAN-based methods. Moreover, for different applications, one may need to build GAN from scratch, whereas our method is a generic solution that can be seamlessly applied to enhance the existing graph neural network models with less computing and tuning overhead.

6.3 Methodology

In this section, we first introduce the formulation of *graph adversarial training*, followed by the introduction of *GATV*, an extension of GAT, which incorporates the virtual adversarial regularization [112]. We then present two solutions for the node classification task, *GCN-GAT* and *GCN-GATV*, which employ GAT and GATV to train GCN [89], respectively. Finally, we analyze the time complexity of the two solutions and present the important implementation details.

6.3.1 Graph Adversarial Training

Recent advances of *adversarial training* (AT) has been successful in learning deep neural network-based classifiers, making them robust against perturbations for a wide range of standard classification tasks such as visual recognition [60, 93, 112] and text classification [111]. Generally, applying AT would regulate the model parameters to smooth the output distribution. Specifically, for each clean example in the dataset, adversarial training encourages the model to assign similar outputs to the artificial input (*i.e.,* the *adversarial example*) derived from the clean example. Inspired by the philosophy of standard AT, we develop graph adversarial training, which trains graph neural network modules in the manner of generating adversarial examples and optimizing additional regularization terms over the adversarial examples, so as to prevent the adverse effects of perturbations. Here the focus is to prevent the perturbations propagated through the node connections (as illustrated in Figure 6.1), *i.e.,* accounting for graph structure in adversarial training.

Generally, the formulation of graph adversarial training is:

$$\begin{aligned}
&\textbf{min: } \Gamma_{GAT} = \Gamma + \beta \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_i} D(f(\boldsymbol{x}_i + \boldsymbol{r}_i^g, G|\boldsymbol{\Theta}), f(\boldsymbol{x}_j, G|\boldsymbol{\Theta})), \\
&\textbf{max: } \boldsymbol{r}_i^g = \arg \max_{\boldsymbol{r}_i, \|\boldsymbol{r}_i\| \leq \epsilon} \sum_{j \in \mathcal{N}_i} D(f(\boldsymbol{x}_i + \boldsymbol{r}_i, G|\hat{\boldsymbol{\Theta}}), f(\boldsymbol{x}_j, G|\hat{\boldsymbol{\Theta}})), \qquad (6.1)
\end{aligned}$$

where Γ_{GAT} is the training objective function with two terms: the standard objective function of the origin graph-based learning model (*e.g.,* a classification loss) and *graph adversarial regularizer*. The second term encourages the graph adversarial examples to be classified similarly as the connected examples where $\boldsymbol{\Theta}$ denotes the parameters to be learned, and D is a nonnegative function that measures the divergence (*e.g.,* Kullback-Leibler divergence [85]) between the two predictions. $\boldsymbol{r}_i^g$ denotes the graph adversarial perturbation, which is applied to the input feature of the clean example i to construct a graph adversarial example.

The graph adversarial perturbation is calculated by maximizing the graph adversarial regularizer under current value of model parameters. That is to say, the graph adversarial perturbation is the direction of change on the input feature, which can maximally attack the graph adversarial regularizer, *i.e.,* the worst case of perturbations propagated from neighbor nodes. ϵ is a hyperparameter controling the magnitude of perturbations, which is typically set to small values so that the feature distribution of adversarial examples is close to that of the clean examples.

Similar to the standard adversarial training, each iteration of GAT can also be viewed as playing a *minimax game*:

- **Maximization**: GAT generates graph adversarial perturbations from clean examples, which break the smoothness between the connected nodes to the maximum extent, and then constructs graph adversarial examples by adding the perturbations to the input of associated clean examples.
- **Minimization**: GAT minimizes the objective function of the graph neural network with an additional regularizer over graph adversarial examples, by encouraging smoothness between predictions of adversarial examples and connected examples. As such, the model becomes robust against perturbations propagated through the graph.

While the traditional graph-based regularizations (*e.g.,* the graph Laplacian term) also encourage the smoothness of predictions over the graph structure, GAT is believed to be a more advanced regulation for two reasons: 1) the regularization performed by GAT is dynamic since the adversarial examples are adaptively generated according to the current parameters and predictions of the model whereas the standard graph-based regularizations are static; and 2) GAT to some extent augments the training data, since the generated adversarial examples would not have occurred in the training data, which would be beneficial to model generalization.

Approximation. It is non-trivial to obtain the closed-form solution of $\boldsymbol{r}_i^g$. Inspired by the *linear approximation* method proposed in [60] for standard adversarial training, we also design a linear approximation method to calculate the graph adversarial perturbations in GAT, of which the formulation is:

$$\boldsymbol{r}_i^g \approx \epsilon \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}, \text{ where } \boldsymbol{g} = \nabla_{\boldsymbol{x}_i} \sum_{j \in \mathcal{N}_i} D(f(\boldsymbol{x}_i, G|\hat{\boldsymbol{\Theta}}), f(\boldsymbol{x}_j, G|\hat{\boldsymbol{\Theta}})), \tag{6.2}$$

where $\boldsymbol{g}$ is the gradient *w.r.t.* the input $\boldsymbol{x}_i$. For graph neural network models, the gradient can be efficiently calculated by one backpropagation. Note that $\hat{\boldsymbol{\Theta}}$ is a constant set denoting the current model parameters.

6.3.2 Virtual Adversarial Training

Considering that node classification is a task of semi-supervised learning by nature, we further devise an extended version of GAT (GATV), which additionally smooths the distribution of predictions around each clean example to further enhance the model robustness. Inspired by the idea of virtual adversarial training [112], we further add a virtual adversarial regularizer into the training objective function and construct virtual adversarial examples to attack the local smoothness of predictions. The formulation of GATV is:

$$\begin{aligned}
\textbf{min: } \Gamma_{GATV} &= \Gamma + \underbrace{\alpha \sum_{i=1}^{N} D(f(\boldsymbol{x}_i + \boldsymbol{r}_i^v, G|\boldsymbol{\Theta}), \tilde{\boldsymbol{y}}_i) +}_{\text{virtual adversarial regularizer}} \\
&\quad \underbrace{\beta \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_i} D(f(\boldsymbol{x}_i + \boldsymbol{r}_i^g, G|\boldsymbol{\Theta}), f(\boldsymbol{x}_j, G|\boldsymbol{\Theta})),}_{\text{graph adversarial regularizer}} \\
\textbf{max: } \boldsymbol{r}_i^v &= \arg \max_{\boldsymbol{r}_i', \|\boldsymbol{r}_i'\| \le \epsilon'} D(f(\boldsymbol{x}_i + \boldsymbol{r}_i', G|\hat{\boldsymbol{\Theta}}), \tilde{\boldsymbol{y}}_i),
\end{aligned} \tag{6.3}$$

where $\boldsymbol{r}_i'$ denotes the virtual adversarial perturbation, the direction that leads to the largest change on the model prediction of $\boldsymbol{x}_i$. For labeled nodes and unlabeled nodes, $\tilde{\boldsymbol{y}}_i$ denotes both the ground truth label and model prediction, as follows:

$$\tilde{\boldsymbol{y}}_i = \begin{cases} \hat{\boldsymbol{y}}_i, & i \le M \text{ (labeled node)}, \\ f(\boldsymbol{x}_i, G|\hat{\boldsymbol{\Theta}}), & M < i \le N \text{ (unlabeled node)}. \end{cases}$$

Note that GATV can be seen as jointly playing two minimax games with three players, where the two maximum players generate virtual adversarial examples and graph adversarial examples, respectively. That is, in each iteration, two types of perturbations and the associated adversarial examples are generated to attack: 1) the smoothness of prediction around individual clean example; and 2) the smoothness of connected examples. By minimizing the additional regularizers over these adversarial examples, the learned model is encouraged to be more smooth and robust, thus achieving good generalization.

Approximation. For labeled nodes, $\boldsymbol{r}_i'$ can be easily evaluated via linear approximation [60], *i.e.,* calculating the gradient of $D(f(\boldsymbol{x}_i, G|\hat{\boldsymbol{\Theta}}), \tilde{\boldsymbol{y}}_i)$ *w.r.t.* $\boldsymbol{x}_i$. For unlabeled nodes, such approximation is infeasible since the gradient will always be zero. This is because $D(f(\boldsymbol{x}_i, G|\hat{\boldsymbol{\Theta}}), \tilde{\boldsymbol{y}}_i)$ achieves the minimum value (0) at $\boldsymbol{x}_i$ (note that $\tilde{\boldsymbol{y}}_i = f(\boldsymbol{x}_i, G|\hat{\boldsymbol{\Theta}})$ for unlabeled data). Realizing that the first-order gradient is always zero, we estimate $\boldsymbol{r}'_i$ from the second-order Taylor approximation of $D(f(\boldsymbol{x}_i + \boldsymbol{r}'_i, G|\hat{\boldsymbol{\Theta}}), \tilde{\boldsymbol{y}}_i)$. That is, $\boldsymbol{r}^v_i \approx \arg\max_{\boldsymbol{r}'_i, \|\boldsymbol{r}'_i\| \leq \epsilon'} \frac{1}{2} \boldsymbol{r}'^T_i \boldsymbol{H} \boldsymbol{r}'_i$ where $\boldsymbol{H}$ is the Hessian matrix of $D(f(\boldsymbol{x}_i + \boldsymbol{r}'_i, G|\hat{\boldsymbol{\Theta}}), \tilde{\boldsymbol{y}}_i)$. For efficiency consideration, we calculate $\boldsymbol{r}^v_i$ via the power iteration approximation [112]:

$$\boldsymbol{r}^v_i \approx \epsilon' \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}, \text{ where } \boldsymbol{g} = \nabla_{\boldsymbol{r}_i} D(f(\boldsymbol{x}_i + \boldsymbol{r}_i, G|\hat{\boldsymbol{\Theta}}, \tilde{\boldsymbol{y}}_i)) \,|_{\boldsymbol{r}_i = \xi \boldsymbol{d}} \,, \tag{6.4}$$

where $\boldsymbol{d}$ is a random vector. The detailed derivation of the method is referred to [112].

6.3.3 Graph Convolution Network

Inspired by the extraordinary representation ability, many neural networks have been used as the predictive model $f(\boldsymbol{x}_i, G|\boldsymbol{\Theta})$ [157, 89, 119, 155]. Under the transductive setting, Graph Convolutional Network [89] is a state-of-the-art model. Specifically, GCN stacks multiple graph convolution layers, which is formulated:

$$\boldsymbol{H}^l = \sigma\left(\widetilde{\boldsymbol{D}}^{-\frac{1}{2}} \widetilde{\boldsymbol{A}} \widetilde{\boldsymbol{D}}^{-\frac{1}{2}} \left(\boldsymbol{H}^{l-1} \boldsymbol{W}^l + \boldsymbol{b}^l\right)\right). \tag{6.5}$$

Here, the l-th graph convolution layer conducts three operations to project $\boldsymbol{H}^{l-1} \in \mathbb{R}^{N \times D^{l-1}}$ (the output of the $(l-1)$-th layer or the node features $\boldsymbol{X}$) into $\boldsymbol{H}^l \in \mathbb{R}^{N \times D^l}$, where D^{l-1} and D^l are the output dimension of layer $l-1$ and l, respectively.

- Similar as the fully connected layer, the graph convolution layer first *projects* the input ($\boldsymbol{H}^{l-1}$) into latent representations with $\boldsymbol{W}^l \in \mathbb{R}^{D^{l-1} \times D^l}$ and $\boldsymbol{b}^l \in \mathbb{R}^{D^l}$.

- It then *propagates* the latent representations ($\boldsymbol{H}^{l-1}\boldsymbol{W}^l + \boldsymbol{b}^l$) through the normalized adjacency matrix $\widetilde{\boldsymbol{D}}^{-\frac{1}{2}} \widetilde{\boldsymbol{A}} \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$ with self-connections, where $\widetilde{\boldsymbol{D}} = \boldsymbol{D} + \boldsymbol{I}$ and $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ ($\boldsymbol{I} \in \mathbb{R}^{N \times N}$ is an identity matrix). Here, the representation of node i in $\boldsymbol{H}$ is the aggregation of latent representations in $(\boldsymbol{H}^{l-1}\boldsymbol{W}^{l} + \boldsymbol{b}^{l})$ of nodes connected to i (including itself due to the self-connection).

- Finally, a non-linear activation function σ (*e.g.,* the sigmoid, hyperbolic tangent, and rectifier functions) is applied to allow for non-linearity.

The original objective function of GCN is,

$$\sum_{i=1}^{M} \textit{cross-entropy}(f(\boldsymbol{x}_i, G|\boldsymbol{\Theta}), \boldsymbol{y}_i) + \lambda\|\boldsymbol{\Theta}\|_F^2, \tag{6.6}$$

where the second term is L_2-norm to prevent overfitting. To train GCN with our proposed GAT and GATV, we set the Γ term in Equations (6.1) and (6.3) as the cross-entropy loss in Equation (6.6).

6.3.4 Time Complexity and Implementation

Time Complexity. As compared to GCN with standard training, the additional computation of GCN-GAT is twofold: 1) generating graph adversarial perturbations ($\{\boldsymbol{r}_i^g, i < N\}$) with Equation (6.2); and, 2) calculating the value of graph adversarial regularizer ($\sum_{i=1}^{N}\sum_{j\in\mathcal{N}_i} D(f(\boldsymbol{x}_i + \boldsymbol{r}_i^g, G|\boldsymbol{\Theta}), f(\boldsymbol{x}_j, G|\boldsymbol{\Theta}))$). Considering that they can be accomplished with a back-propagation and a forward-propagation (to calculate $f(\boldsymbol{x}_i + \boldsymbol{r}_i^g, G|\boldsymbol{\Theta})$), the computation overhead of GCN-GAT is acceptable. Additionally, GCN-GATV computes virtual adversarial perturbations and virtual adversarial regularizer, which can be performed with one back-propagation and one forward-propagation, respectively [112]. It indicates that the overhead of GCN-GATV is still acceptable [112].

Implementation. Note that the number of connected nodes varies a lot across the nodes in the graph, we sample K neighbors for each node to generate the adversarial examples and calculate the graph adversarial regularizer to facilitate the calculation. Here, the following sampling strategies are considered:

- **Uniform:** Neighbors are selected uniformly.

- **Degree:** The probability of selecting a node is proportional to the normalized node degree.

- **Degree-Reverse:** On the contrary, the probability is the reciprocal of node degree (also normalized to sum to unity).

- **PageRank:** It performs PageRank [125] on the graph and takes the normalized pagerank score as the sampling probability.

Note that the other advanced but complex sampling strategies (*e.g.,* the one in [177]) are not considered due to efficiency consideration.

6.4 Experiments

6.4.1 Experimental Settings

Datasets. We follow the same experimental settings as in [89] and conduct experiments on two types of node classification datasets: two citation network datasets (Citeseer and Cora [137]) and a knowledge graph dataset (NELL [176])[2]. The statistics of the datasets are summarized in Table 6.1. Note that we adopt the exactly same data processing and data split as in [89] for fair comparison.

- In the citation networks, nodes and edges represent documents and citation links between documents, respectively. Note that the direction of edge is omitted since a citation is assumed to have equal impacts on the prediction of the two associated documents. Each document is associated with a normalized bag-of-words feature vector and a class label. During training, we use features of all nodes, but only 20 labels per class. Among the remaining nodes, 500 and 1,000 of them are used as validation and testing, respectively.

- NELL is a bipartite graph of 55,864 relation nodes and 9,891 entity nodes, extracted from a knowledge graph which is a set of triplets in the format of (e_1, r, e_2). Here e_1 and e_2 are entities, and r is the connected relation between them. Following [89], each relation r is split into two relation nodes (r_1 and r_2), from which two edges (e_1, r_1) and (e_2, r_2) are constructed. Entity nodes and relation nodes are described by bag-of-words feature vectors (normalized)

[2]`https://github.com/kimiyoung/planetoid`.

Table 6.1: Statistics of the experiment datasets.

Dataset	#Nodes	#Edges	#Classes	#Features	Label rate
Citeseer	3,312	4,732	6	3,703	0.036
Cora	2,708	5,429	7	1,433	0.052
NELL	65,755	266,144	210	5,414	0.001

and one-hot encodings, respectively. Note that we pad zero values to align the feature vectors of entity and relation nodes. Here only labels of entity nodes are available and only 0.1% of entities under each class are labeled during training.

Baselines. We compare our proposed method with the following baselines:

- **LP** [196]: It performs label propagation that ignores the node features and propagates only the labels over the graph structure.
- **DeepWalk** [127]: It is a skip-gram based graph embedding method, which learns the embedding of a node by predicting its contexts that are generated by performing random walk on the graph.
- **SemiEmb** [169]: It learns node embeddings from node features and leverages Laplacian regularization to encourage connected nodes to have close embeddings.
- **Planetoid** [176]: Similar as DeepWalk, this method learns node embeddings by predicting the node context, while additionally accounts for node features.
- **GCN** [89]: It stacks two graph convolution layers to project node features into labels. It propagates node representations and predictions over the graph structure to smooth the output.
- **GraphSGAN** [40]: This is a semi-supervised generative adversarial network which encodes the density signal of the graph structure during the generation of fake nodes.

Note that **LP**, **DeepWalk**, **SemiEmb**, and **Planetoid** are also baselines in the paper of **GCN**, and we follow exactly their settings in [89]. In addition, the setting of **GraphSGAN** is same as that in the original paper.

Parameter Settings. We implement our proposed GCN-GAT and GCN-

Table 6.2: Performance of the compared methods on the three datasets *w.r.t.* accuracy.

Category	Method	Citeseer	Cora	NELL
Graph	LP	45.3	68.0	26.5
	DeepWalk	43.2	67.2	58.1
+Node Features	SemiEmb	59.6	59.0	26.7
	Planetoid	64.7	75.7	61.9
	GCN	69.3	81.4	61.2
+Adversarial	GraphSGAN	73.1	**83.0**	—
	GCN-GATV	**73.7**	82.6	**64.7**

GATV based on Tensorflow. GCN-GAT has six hyperparameters in total: 1) D^1, the size of hidden layer (GCN); 2) λ, the weight for L_2-norm (GCN); 3) dropout ratio (GCN); 4) ϵ, the scale of graph adversarial perturbations (GAT); 5) β, the weight for graph adversarial regularizer (GAT); and 6) K, the number of sampled neighbors (GAT). For fair comparison, we set the D^1 and λ as the optimal values of standard GCN. But we set dropout ratio as zero in GCN-GAT for stable training. For the remaining three parameters, ϵ, β, and K, we performed grid-search within the ranges of [0.01, 0.05, 0.1, 0.5, 1], [0.01, 0.05, 0.1, 0.5, 1, 5], [1, 2, 3], respectively.

For GCN-GATV, for simplicity, we set the six hyperparameters common to that of GCN-GAT using the optimal values found for GCN-GAT. Here we only tune its three additional hyperparameters: 1) ϵ', the scale of virtual adversarial perturbations; 2) α, the weight for virtual adversarial regularizer; and 3) ξ, the scale to calculate approximation. In particular, we perform grid-search to tune the parameters ϵ', α, and ξ within the ranges of [0.01, 0.05, 0.1, 0.5, 1], [0.001, 0.005, 0.01, 0.05, 0.1, 0.5], [1e-6, 1e-5, 1e-4], respectively. It should be noted that the *Uniform* strategy is adopted to sample the neighbor nodes if not otherwise specified.

6.4.2 Performance Comparison

Model Comparison. We first investigate the effectiveness of the proposed *graph adversarial training* via comparing the performance of GCN-GATV with the state-of-the-art node classification methods. Table 6.2 shows the classification performance of the compared methods on the three datasets *w.r.t.* accuracy. The performance of LP, DeepWalk, SemiEmb, and Planetoid are taken from the GCN paper [89] since we follow its settings exactly. From the results, we have the following observations:

- GCN-GATV significantly outperforms the standard GCN, exhibiting relative improvements of 6.35%, 1.47%, and 5.72% on the Citeseer, Cora, and NELL datasets, respectively. As the only difference between GCN-GATV and GCN is the use of the proposed graph adversarial training, the improvements are attributed to the proposed training method which would enhance the stabilization and generalization of GCN. Besides, the results validate that GCN-GATV is effective in tackling the node classification task.

- GCN-GATV achieves comparable performance as that of GraphSGAN, which is the state-of-the-art method of node classification. It demonstrates the efficacy of the proposed method. However, our method could offer a more feasible solution for two reasons: 1) GraphSGAN is based on the standard generative adversarial network, which explicitly plays a mini-max game between a discriminator and a generator (two different networks). This, inevitably, will lead to doubling of the computation of model training and the labor of parameter tuning. 2) For different node classification applications, GraphSGAN needs to be built from scratch, whereas our GCN-GATV is a generic solution that can be seamlessly applied to enhance the existing models of the applications.

- GCN-GATV and GraphSGAN achieve better results in all the cases as compared to the other baselines. On the Citeseer, Cora, and NELL datasets, the relative improvements are at least 6.35%, 1.97%, and 4.52%, respectively. This indicates the effectiveness of adversarial learning, *i.e.,* dynamically playing a mini-max game either implicitly (GCN-GATV) or explicitly (GraphSGAN) in the training phase. Moreover, the results are consistent with findings in previous works [60, 181, 73, 112].

- Among the baselines, 1) the methods that jointly account for the graph structure and node features (in the category of *+Node Features*) outperform LP and DeepWalk that only consider graph structure. This suggests further exploration of how to combine the connection patterns and node features more appropriately. 2) As compared to SemiEmb that is a shallow model, Planetoid and GCN achieves significant improvements (from 8.56% to 131.8%) in all cases. The improvement is reasonable and attributed to the strong representation ability of neural networks. As such, methods targeting to enhance the graph neural network models, such as the graph adversarial training, will be meaningful and influential in future.

Performance *w.r.t.* Node Degree. We next study how the graph adversarial training performs on nodes with different densities of connections so as to understand where this regularization technique can be more suitably applied. We empirically split the nodes into three groups according to node degree (*i.e.,* the number of neighbors), where node degrees are in ranges of $[1, 2]$, $[3, 5]$, $[6, N]$. Figure 6.2 illustrates the distribution of nodes over the three groups. As can be seen, in all the three datasets, a great number of nodes are sparsely connected (with degrees smaller than three), and only about ten percent of the nodes are densely connected with degrees bigger than five. By separately computing the

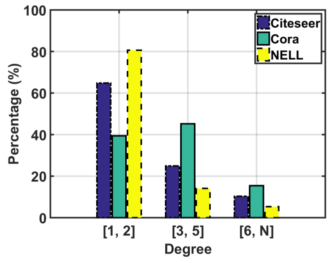


Figure 6.2: Percentage of nodes with different groups of node degree in the three datasets.

accuracy of GCN and GCN-GATV over nodes in different groups, we obtain the group-oriented performance on the three datasets, as depicted in Figure 6.3. From the results, we observe that:

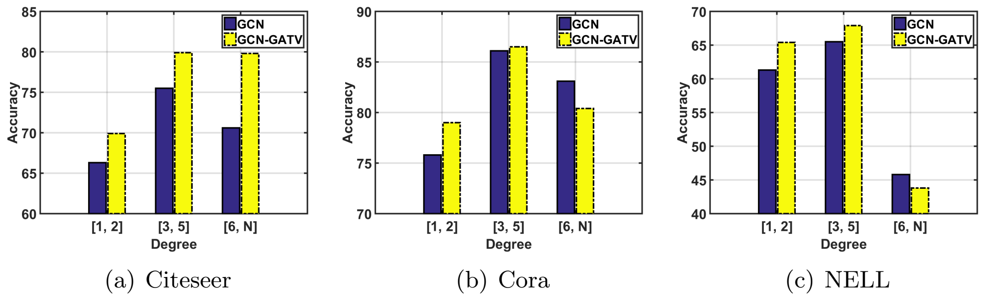


Figure 6.3: Performance of GCN and GCN-GATV on nodes with different degrees in Citeseer (a), Cora (b), and NELL (c).

Table 6.3: Effect of graph adversarial regularization and virtual adversarial regularization.

Category	Method	Citeseer	Cora
Standard Training	GCN	69.3	81.4
Adversarial Training	GCN-VAT	72.4	79.3
	GCN-GAT	73.4	82.5
	GCN-GATV	**73.7**	**82.6**

- In all the three datasets, both GCN and GCN-GATV achieves the best performance on the group of $[3, 5]$. The relatively worse performance on the group of $[1, 2]$ could be attributed to that the nodes in that group are sparsely connected and lacks sufficient signals propagated from the neighbors, which are helpful for the classification [196, 89, 54]. In addition, we postulate the reason for the worse performance over nodes with degrees in $[6, N]$ as such nodes are harder to classify. This is because such nodes typically represent more general entities, such as those having connections to other entities with different types of relations and are thus harder to be accurately classified into a specific category.

- In most cases (except the $[6, N]$ group of Cora and NELL), GCN-GATV outperforms GCN, which indicates that graph adversarial training would benefit the prediction of nodes with different degrees and is roughly not sensitive to the density of graph. For one of the exceptions (the $[6, N]$ group of NELL), we speculate that the reason is the under-fitting of standard GCN on such nodes (note that the performance of GCN on $[6, N]$ is 27.7% on average worse than the other two groups), where additional regularization performed by graph adversarial training worsens the under-fitting problem.

- GCN-GATV significantly and consistently outperforms GCN on the group of $[1, 2]$, with an average improvement of 5.45%. The result indicates that the graph adversarial training would be more effective on the sparse part of the graph. It should be noted that most of the graphs are sparse in real world applications [34]. As such this result further demonstrates the potential of the proposed methods in real world applications.

6.4.3 Method Ablation

Recall that we design two versions of graph adversarial training: 1) basic GAT (Equation (6.1)); and 2) incorporating virtual adversarial training (Equation (6.3)). To evaluate the contribution of these two types of regularizations, we compare the performance of the following solutions built upon GCN:

- **GCN**: It learns the parameters of GCN with standard training, *i.e.,* it optimizes Equation (6.6).
- **GCN-VAT**: GCN with virtual adversarial training, which performs perturbations by considering node features only. It is employed to train GCN, *i.e.,* optimizing Equation (6.3) with $\beta = 0$.
- **GCN-GAT**: It trains GCN by the basic GAT, of which the perturbations only focus on graph structure, *i.e.,* optimizing Equation (6.3) with $\alpha = 0$.
- **GCN-GATV**: It accounts for both the virtual and graph adversarial regularizations during the training of GCN.

It should be noted that, in the following, we focus on the citation graphs and omit results on NELL, which is a bipartite graph rather than a standard simple graph. Table 6.3 shows the performance of the compared methods on the citation networks *w.r.t.* accuracy. As can be seen:

- In most of the cases, GCN performs worse than the other approaches. This indicates that adversarial training could enhance the node classification model as compared to the standard training; *i.e.,* by intentionally and dynamically generating perturbations and optimizing additional regularizers, the trained model could by more accurate.
- GCN-GATV achieves the best performance in all cases. It shows that perturbations targeting at both the individual nodes (virtual adversarial perturbations) and neighbor nodes (graph adversarial perturbations) benefit the training of graph neural network model. Moreover, it suggests that it is beneficial to jointly consider both the node features and graph structure in

Table 6.4: Performance of GCN-GAT as tuning all hyperparameters (*i.e.,* β, ϵ, and k) and tuning ϵ with fixed $\beta = 1.0$ and $k = 1$.

Hyperparameter	Citeseer	Cora
$\{\beta,\ \epsilon,\ k\}$	73.4	**82.5**
$\{\epsilon\}$	**73.6**	**82.5**

Table 6.5: Performance comparison of GCN-GAT with different sampling strategies of neighbors during adversarial example generation.

Sampling Strategy	Citeseer	Cora
Uniform	73.4	82.5
Degree	73.0	82.9
Degree-Reverse	**73.8**	82.4
PageRank	72.6	**83.1**

adversarial training of graph neural networks.

- Compared with GCN-VAT, GCN-GAT achieves improvements of 1.38% and 4.04% on the Citeseer and Cora datasets, respectively. This again signifies the benefit of accounting for the graph structure in adversarial training of graph neural networks.

6.4.4 Effect of Parameter Settings

Tuning ϵ Only. Consider that the number of candidate combinations increases exponentially with the number of hyperparameters, we explore whether comparable performance could be achieved when we tune only one hyperparameter while fixing the others with empirical values. It should be noted that previous work [112] has shown that tuning ϵ' alone is suffice for achieving satisfactory performance of VAT. Similarly, we tune ϵ with $\beta = 1$ and $k = 1$ and summarize the performance of GCN-GAT in Table 6.4. As can be seen, tuning ϵ alone achieves satisfactory performance which indicates that tuning ϵ suffices for achieving satisfactory performance. As such, the overhead of the additional hyperparameters of the proposed method could be ignored.

Sensitivity to Sampling Strategies. As mentioned in Section 6.3.4, different strategies could be adopted to sample neighbor nodes for the generation of graph adversarial perturbations and the calculation of graph adversarial regularizer. Here, we investigate the effect of sampling strategies by comparing the results of GCN-GAT performing different samplings. Table 6.5 shows the corresponding performance, from which we can observe that the performance of different sampling strategies are comparable to each other. As such, *Uniform* would be a suitable selection since it will not bring any additional computation as compared to the other approaches.

6.5 Conclusion

In this work, we proposed a new learning method, named *graph adversarial training*, which additional accounts for relations between examples as compared to standard adversarial training. By iteratively generating adversarial examples to attack the graph smoothness constraint and learn over adversarial examples, the proposed method encourages the smoothness of predictions over the given graph, a property indicating good generalization of the model. As can be seen as a dynamic regularization technique, our method is generic and can be applied to train most graph neural network models. We trained one well-established model, GCN, with the proposed method to solve the node classification task. By conducting experiments on three benchmark datasets, we demonstrated that training GCN with our method is remarkably effective, achieving an average improvement of 4.51%. Moreover, it also beats GCN trained with VAT, indicating the necessity of performing AT with consideration of graph structure.

Chapter 7

Conclusion and Future Work

We found that most existing graph-based learning methods mainly focus on modeling local smoothness by considering the connection properties between vertices. However, the rich information within graph data and graph applications are ignored, which results in the suboptimal modeling of local smoothness. In this thesis, we explored techniques to enhance the local smoothness by additionally incorporating the ignored information from both edge and vertex perspectives. Essentially, we focused on various edge attributes, domain knowledge, dynamic vertex features, and adversarial perturbations. Specifically,

- We modeled *relation-aware local smoothness* by a multi-relation learning framework which jointly considers multiple types of entity relations.

- We designed a new regularization term to encode domain knowledge which can be represented by partial-order rules in the graph-based learning framework in order to model local smoothness in a rule-guided manner (*rule-guided local smoothness*).

- We captured the *temporal* property of *local smoothness* by a new neural network operator which adaptively adjusts the strength of smoothness between vertices according to the temporal features of vertices.

- We developed a new training approach for the advanced graph neural networks, resulting in robust models that can defend adversarial perturbations on vertex features.

We applied the proposed methods on different applications including university ranking, stock ranking, popularity prediction and conventional node classifications. Experimental results on benchmark datasets validated the effectiveness of the proposed methods. For instance, our method that models temporal local smoothness significantly outperforms conventional graph-based learning methods. In addition, in a testing year, a simulation of the proposed method achieves exciting revenues on both NYSE and NASDAQ with return ratios higher than 100%. Moreover, we found that domain knowledge plays a crucial role in graph-based learning.

In the future, we would like to explore the following directions:

- We would apply the proposed methods on more graph applications to further test their effectiveness and investigate the scope of scenarios suitable for each method. Furthermore, such test would suggest what additional information should be considered according to the property of the graph application.

- We are interested in exploring more techniques on leveraging domain knowledge in graph-based learning, including 1) incorporating knowledge in different formats into the learning framework; and 2) inferencing over knowledge to enhance model generalization and explainability.

- The proposed graph adversarial training introduces a new direction for implementing local smoothness, which is in a dynamic manner along the training procedure, as compared to the static implementations (*i.e.,* graph Laplacian regularization and conventional representation learning). We would follow this line of research to further explore techniques on dynamic local smoothness.

- While graph neural networks achieved exciting performance in many graph applications, to improve their performance and usability, it is worthwhile further developing such models in the following directions: 1) Elaborate neighbor aggregation. Different neighbors might have various connections with a center node and reflect different properties of the center node. Aggregating information of neighbors in an elaborate manner, *e.g.,* by capturing the various connections with the center node and encoding the influence among neighbors, would be an opportunity. 2) Explainability. Like standard machine learning models, explainability is one of the key factors of using graph neural networks in practical applications. Therefore, it is desired to design explainable graph neural networks which can infer explanation of a prediction by also incorporating the graph structure. 3) Efficient training. A graph in practical applications might contain millions even billions nodes, which is a great challenge for applying graph neural networks, especially when the application has strict constraint on response time. It thus requires research of efficient and scalable training methods such as distributed training for these models. 4) Robustness training. As discussed in Chapter 6, graph neural networks are vulnerable to adversarial attacks which commonly occur in practical graph applications. Further research along this line would be theoretical analysis of the impacts of attacks, certification of prediction robustness, and exploration of strategies to better defend attacks.

Appendices

Appendix A

Appendices

A.1 Wiki Company-based Relations

In this appendix, we describe the details of *Wiki company-based relations*) in our collected data (Section 5.5).

From Wikidata, one of the biggest and most active open domain knowledge bases, we obtain 5 and 53 types of first-order (in the format of $\textcircled{A} \xrightarrow{R} \textcircled{B}$) and second-order relations (in the format of $\textcircled{A} \xrightarrow{R_1} \textcircled{C} \xleftarrow{R_2} \textcircled{B}$) between companies corresponding to the selected stocks in NASDAQ and NYSE markets, respectively. Note that A and B denote entities in Wikidata corresponding to two companies; C denotes another entity bridging two company-entities in a second-order relation; R, R_1, and R_2 denotes different types of entity relation defined in Wikidata[1]. In Table A.1 and A.2, we summarize the extracted first-order and second-order relations, respectively.

Table A.1: First-order Wiki company-based relations in the format of $\textcircled{A} \xrightarrow{R} \textcircled{B}$.

	Wikidata Relation (R)	Relation Description
1	P127	*Owned by*: owner of the subject.
2	P155	*Follows*: immediately prior item in a series of which the subject is a part.
3	P156	*Followed by*: immediately following item in a series of which the subject is a part.
4	P355	*Subsidiary*: subsidiary of a company or organization.
5	P749	*Parent organization*: parent organization of an organisation, opposite of subsidiaries.

[1]https://www.wikidata.org/wiki/Wikidata:List_of_properties/all

Table A.2: Second-order Wiki company-based relations in the format of $\textcircled{A} \xrightarrow{R_1} \textcircled{C} \xleftarrow{R_2} \textcircled{B}$.

	Wikidata Relations	Relation Descriptions
1	R_1 = P31	*Instance of*: that class of which this subject is a particular example and member.
	R_2 = P366	*Use*: main use of the subject.
2	R_1 = P31	*Instance of*: that class of which this subject is a particular example and member.
	R_2 = P452	*Industry*: industry of company or organization.
3	R_1 = P31	*Instance of*: that class of which this subject is a particular example and member.
	R_2 = P1056	*Product or material produced*: material or product produced by an agency.
4	R_1 = P112	*Founded by*: founder or co-founder of this organization.
	R_2 = P112	*Founded by*: founder or co-founder of this organization.
5	R_1 = P112	*Founded by*: founder or co-founder of this organization.
	R_2 = P127	*Owned by*: owner of the subject.
6	R_1 = P112	*Founded by*: founder or co-founder of this organization.
	R_2 = P169	*Chief executive officer*: the CEO within an organization.
7	R_1 = P113	*Airline hub*: airport that serves as a hub for an airline.
	R_2 = P113	*Airline hub*: airport that serves as a hub for an airline.
8	R_1 = P114	*Airline alliance*: alliance the airline belongs to.
	R_2 = P114	*Airline alliance*: alliance the airline belongs to.
9	R_1 = P121	*Item operated*: equipment, installation or service operated by the subject.
	R_2 = P1056	*Product or material produced*: material or product produced by an agency.
10	R_1 = P121	*Item operated*: equipment, installation or service operated by the subject.
	R_2 = P121	*Item operated*: equipment, installation or service operated by the subject.
11	R_1 = P127	*Owned by*: owner of the subject.
	R_2 = P112	*Founded by*: founder or co-founder of this organization.
12	R_1 = P127	*Owned by*: owner of the subject.
	R_2 = P127	*Owned by*: owner of the subject.
13	R_1 = P127	*Owned by*: owner of the subject.
	R_2 = P169	*Chief executive officer*: the CEO within an organization.
14	R_1 = P127	*Owned by*: owner of the subject.
	R_2 = P355	*Subsidiary*: subsidiary of a company or organization.
15	R_1 = P127	*Owned by*: owner of the subject.
	R_2 = P749	*Parent organization*: parent organization of an organisation.
16	R_1 = P127	*Owned by*: owner of the subject.
	R_2 = P1830	*Owner of*: entities owned by the subject.
17	R_1 = P127	*Owned by*: owner of the subject.
	R_2 = P3320	*Board member*: member(s) of the board for the organization.
18	R_1 = P155	*Follows*: immediately prior item in a series of which the subject is a part.
	R_2 = P155	*Follows*: immediately prior item in a series of which the subject is a part.
19	R_1 = P155	*Follows*: immediately prior item in a series of which the subject is a part.
	R_2 = P355	*Subsidiary*: subsidiary of a company or organization.
20	R_1 = P166	*Award received*: award or recognition received by a person, organisation.
	R_2 = P166	*Award received*: award or recognition received by a person, organisation.
21	R_1 = P169	*Chief executive officer*: the CEO within an organization.
	R_2 = P112	*Founded by*: founder or co-founder of this organization.
22	R_1 = P169	*Chief executive officer*: the CEO within an organization.

	R_2 = P127	*Owned by*: owner of the subject.
23	R_1 = P169	*Chief executive officer*: the CEO within an organization.
	R_2 = P169	*Chief executive officer*: the CEO within an organization.
24	R_1 = P169	*Chief executive officer*: the CEO within an organization.
	R_2 = P3320	*Board member*: member(s) of the board for the organization.
25	R_1 = P199	*Business division*: divisions of this organization.
	R_2 = P355	*Subsidiary*: subsidiary of a company or organization.
26	R_1 = P306	*Operating system*: operating system (OS) on which a software works.
	R_2 = P1056	*Product or material produced*: material or product produced by an agency.
27	R_1 = P355	*Subsidiary*: subsidiary of a company or organization.
	R_2 = P127	*Owned by*: owner of the subject.
28	R_1 = P355	*Subsidiary*: subsidiary of a company or organization.
	R_2 = P155	*Follows*: immediately prior item in a series of which the subject is a part.
29	R_1 = P355	*Subsidiary*: subsidiary of a company or organization.
	R_2 = P199	*Business division*: divisions of this organization.
30	R_1 = P355	*Subsidiary*: subsidiary of a company or organization.
	R_2 = P355	*Subsidiary*: subsidiary of a company or organization.
31	R_1 = P361	*Part of*: object of which the subject is a part.
	R_2 = P361	*Part of*: object of which the subject is a part.
32	R_1 = P366	*Use*: main use of the subject.
	R_2 = P31	*Instance of*: that class of which this subject is a particular example and member.
33	R_1 = P400	*Platform*: platform for which a work was developed or released.
	R_2 = P1056	*Product or material produced*: material or product produced by an agency.
34	R_1 = P452	*Industry*: industry of company or organization.
	R_2 = P31	*Instance of*: that class of which this subject is a particular example and member.
35	R_1 = P452	*Industry*: industry of company or organization.
	R_2 = P452	*Industry*: industry of company or organization.
36	R_1 = P452	*Industry*: industry of company or organization.
	R_2 = P1056	*Product or material produced*: material or product produced by an agency.
37	R_1 = P452	*Industry*: industry of company or organization.
	R_2 = P2770	*Source of income*: source of income of an organization or person.
38	R_1 = P463	*Member of*: organization or club to which the subject belongs.
	R_2 = P463	*Member of*: organization or club to which the subject belongs.
39	R_1 = P749	*Parent organization*: parent organization of an organisation.
	R_2 = P127	*Owned by*: owner of the subject.
40	R_1 = P749	*Parent organization*: parent organization of an organisation.
	R_2 = P1830	*Owner of*: entities owned by the subject.
41	R_1 = P1056	*Product or material produced*: material or product produced by an agency.
	R_2 = P31	*Instance of*: that class of which this subject is a particular example and member.
42	R_1 = P1056	*Product or material produced*: material or product produced by an agency.
	R_2 = P121	*Item operated*: equipment, installation or service operated by the subject.
43	R_1 = P1056	*Product or material produced*: material or product produced by an agency.
	R_2 = P306	*Operating system*: operating system (OS) on which a software works.
44	R_1 = P1056	*Product or material produced*: material or product produced by an agency.
	R_2 = P400	*Platform*: platform for which a work was developed or released.
45	R_1 = P1056	*Product or material produced*: material or product produced by an agency.

	R_2 = P452	*Industry*: industry of company or organization.
46	R_1 = P1056	*Product or material produced*: material or product produced by an agency.
	R_2 = P1056	*Product or material produced*: material or product produced by an agency.
47	R_1 = P1344	*Participant of*: event a person or an organization was a participant in.
	R_2 = P1344	*Participant of*: event a person or an organization was a participant in.
48	R_1 = P1830	*Owner of*: entities owned by the subject.
	R_2 = P127	*Owned by*: owner of the subject.
49	R_1 = P1830	*Owner of*: entities owned by the subject.
	R_2 = P749	*Parent organization*: parent organization of an organisation.
50	R_1 = P2770	*Source of income*: source of income of an organization or person.
	R_2 = P452	*Industry*: industry of company or organization.
51	R_1 = P3320	*Board member*: member(s) of the board for the organization.
	R_2 = P127	*Owned by*: owner of the subject.
52	R_1 = P3320	*Board member*: member(s) of the board for the organization.
	R_2 = P169	*Chief executive officer*: the CEO within an organization.

Bibliography

[1] A. A. Adebiyi, A. O. Adewumi, and C. K. Ayo. Comparison of arima and artificial neural networks models for stock price prediction. *Journal of Applied Mathematics*, 2014, 2014.

[2] S. Agarwal. Ranking on graph data. In *International Conference on Machine Learning*, pages 25–32. ACM, 2006.

[3] C. C. Aggarwal and C. K. Reddy. *Data clustering: algorithms and applications.* CRC press, 2013.

[4] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48. ACM, 2013.

[5] A. Airola, S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, and T. Salakoski. A graph kernel for protein-protein interaction extraction. In *Proceedings of the workshop on current trends in biomedical natural language processing*, pages 1–9. Association for Computational Linguistics, 2008.

[6] J. Alberg and Z. C. Lipton. Improving factor-based quantitative investing by forecasting company fundamentals. *arXiv preprint arXiv:1711.04837*, 2017.

[7] G. Andrew, R. Arora, J. A. Bilmes, and K. Livescu. Deep canonical correlation analysis. In *International Conference on Machine Learning*, pages 1247–1255, 2013.

[8] A. Armstrong, R. Francis, M. Bourne, and I. Dussuyer. *Difficulties of developing and using social indicators to evaluate government programs: a critical review.* PhD thesis, 2002.

[9] S. H. Bach, M. Broecheler, B. Huang, and L. Getoor. Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research*, 2017.

[10] S. Bahrampour, N. M. Nasrabadi, A. Ray, and W. K. Jenkins. Multimodal task-driven dictionary learning for image classification. *TIP*, 25(1):24–38, 2016.

[11] W. Bao, J. Yue, and Y. Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.

[12] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[13] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.

[14] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, pages 2399–2434, 2006.

[15] A. Bellaachia and M. Al-Dhelaan. Multi-document hyperedge-based ranking for text summarization. In *Proceedings of the 23rd ACM International on Conference on Information and Knowledge Management*, pages 1919–1922, 2014.

[16] P. N. Bennett and N. Nguyen. Refined experts: improving classification in large taxonomies. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 11–18. ACM, 2009.

[17] R. v. d. Berg, T. N. Kipf, and M. Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[18] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[19] A. Bojchevski and S. Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. 2018.

[20] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008.

[21] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, pages 2787–2795, 2013.

[22] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[23] A. Bose, H. Ling, and Y. Cao. Adversarial contrastive estimation. *ACL*, 2018.

[24] M. J. Boskin. Consumer price indexes. *Concise Encyclopedia of Economics*, 2008.

[25] E. Bruno and M. M. Stephane. Multiview clustering: a late fusion approach using latent models. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 736–737. ACM, 2009.

[26] 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 18th ACM international conference on Multimedia*, pages 391–400. ACM, 2010.

[27] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 891–900. ACM, 2015.

[28] S. Cao, W. Lu, and Q. Xu. Deep neural networks for learning graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1145–1152, 2016.

[29] C. Chen, C. Lu, Q. Huang, Q. Yang, D. Gunopulos, and L. Guibas. City-scale map creation and updating using gps collections. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1465–1474, 2016.

[30] J. Chen, X. Song, L. Nie, X. Wang, H. Zhang, and T.-S. Chua. Micro tells macro: predicting the popularity of micro-videos via a transductive model. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 898–907, 2016.

[31] X. Q. Cheng, P. Du, J. Guo, X. Zhu, and Y. Chen. Ranking on data manifold with sink points. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):177–191, Jan 2013.

[32] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *The Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734. ACL, Oct. 2014.

[33] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*, pages 39–46, 2010.

[34] P. Cui, X. Wang, J. Pei, and W. Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[35] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song. Adversarial attack on graph structured data. In *International Conference on Machine Learning*, volume 80, pages 1115–1124. PMLR, 2018.

[36] Q. Dai, Q. Li, J. Tang, and D. Wang. Adversarial network embedding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[37] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[38] I. S. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556, 2004.

[39] R. Diestel. *Graph theory.* Springer Publishing Company, Incorporated, 2018.

[40] M. Ding, J. Tang, and J. Zhang. Semi-supervised learning on graphs with generative adversarial nets. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 913–922. ACM, 2018.

[41] X. Ding, Y. Zhang, T. Liu, and J. Duan. Using structured events to predict stock price movement: An empirical investigation. In *The Conference on Empirical Methods in Natural Language Processing*, pages 1415–1425, 2014.

[42] X. Ding, Y. Zhang, T. Liu, and J. Duan. Deep learning for event-driven stock prediction. 24th international joint conference on artificial intelligence, pages 2327–2333. AAAI Press, 2015.

[43] M. Dixon, D. Klabjan, and J. H. Bang. Classification-based financial markets prediction using deep neural networks. *Algorithmic Finance*, (Preprint):1–11, 2016.

[44] C. Djeraba. *Mathematical Tools For Data Mining: Set Theory, Partial Orders, Combinatorics. Advanced Information and Knowledge Processing.* Springer, 2008.

[45] M. Dobrota, M. Bulajic, L. Bornmann, and V. Jeremic. A new approach to the qs university ranking using the composite i-distance indicator: Uncertainty and sensitivity analyses. *Journal of the Association for Information Science and Technology*, 67(1):200–211, 2016.

[46] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[47] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the tenth international conference on World Wide Web*, pages 613–622. ACM, 2001.

[48] F. Feng, X. He, Y. Liu, L. Nie, and T.-S. Chua. Learning on partial-order hypergraphs. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1523–1532. International World Wide Web Conferences Steering Committee, 2018.

[49] F. Feng, L. Nie, X. Wang, R. Hong, and T.-S. Chua. Computational social indicators: a case study of chinese university ranking. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 455–464. ACM, 2017.

[50] X. Feng, S. Wu, and W. Zhou. Multi-hypergraph consistent sparse coding. *Transactions on Intelligent Systems and Technology*, 8(6):75, 2017.

[51] O. for Economic Co-operation and Development. *Measuring social well-being: a progress report on the development of social indicators.* OECD Publications Center, 1976.

[52] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 6530–6539, 2017.

[53] Y. Fu, T. M. Hospedales, T. Xiang, and S. Gong. Transductive multi-view zero-shot learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(11):2332–2345, 2015.

[54] H. Gao, Z. Wang, and S. Ji. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1416–1424. ACM, 2018.

[55] W. Gao and P. Yang. Democracy is good for ranking: towards multi-view rank learning and adaptation in web search. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 63–72. ACM, 2014.

[56] U. Gerdtham and B. Jönsson. International comparisons of health expenditure: theory, data and econometric analysis. *Handbook of Health Economics*, 1:11–53, 2000.

[57] D. S. Goldberg and F. P. Roth. Assessing experimentally derived interactions in a small world. *Proceedings of the National Academy of Sciences*, 100(8):4372–4376, 2003.

[58] C. Goller and A. Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *International Conference on Neural Networks*, volume 1, pages 347–352. IEEE, 1996.

[59] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning.* MIT press, 2016.

[60] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *Proceedings of the International Conference on Learning Representations*, 2015.

[61] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[62] A. Gretton, O. Bousquet, A. Smola, and B. Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *International Conference on Algorithmic Learning Theory*, pages 63–77. Springer, 2005.

[63] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.

[64] C. Guarino, G. Ridgeway, M. Chun, and R. Buddin. Latent variable analysis: a new approach to university ranking. *Higher Education in Europe*, 30(2):147–165, 2005.

[65] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000.

[66] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[67] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[68] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[69] X. He and T.-S. Chua. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 355–364, 2017.

[70] X. He, M. Gao, M.-Y. Kan, Y. Liu, and K. Sugiyama. Predicting the popularity of web 2.0 items based on user comments. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 233–242, 2014.

[71] X. He, M. Gao, M.-Y. Kan, and D. Wang. Birank: Towards ranking on bipartite graphs. *Transactions on Knowledge and Data Engineering*, 29(1):57–71, 2017.

[72] X. He, M. Gao, M. Y. Kan, and D. Wang. Birank: Towards ranking on bipartite graphs. *IEEE Transactions on Knowledge and Data Engineering*, 29:57–71, 2017.

[73] X. He, Z. He, X. Du, and T.-S. Chua. Adversarial personalized ranking for recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 355–364. ACM, 2018.

[74] X. He, M. Kan, P. Xie, and X. Chen. Comment-based multi-view clustering of web 2.0 items. In *Proceedings of the 23rd international conference on World Wide Web*, pages 771–782. ACM, 2014.

[75] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[76] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[77] M. Hmimida and R. Kanawati. A graph-coarsening approach for tag recommendation. In *Proceedings of the 25th international conference on World Wide Web*, pages 43–44, 2016.

[78] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[79] Z. Hu, W. Liu, J. Bian, X. Liu, and T.-Y. Liu. Listening to chaotic whispers: A deep learning framework for news-oriented stock trend prediction. In *Proceedings of the 11th ACM international conference on Web search and data mining*, pages 261–269. ACM, 2018.

[80] Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. Xing. Harnessing deep neural networks with logic rules. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 2410–2420, 2016.

[81] S. Huang, M. Elhoseiny, A. Elgammal, and D. Yang. Learning hypergraph-regularized attribute predictors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 409–417, 2015.

[82] J. Jeon, V. Lavrenko, and R. Manmatha. Automatic image annotation and retrieval using cross-media relevance models. In *Proceedings of the 26th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 119–126. ACM, 2003.

[83] F. Jiang, Y. Liu, H. Luan, M. Zhang, and S. Ma. Microblog sentiment analysis with emoticon space model. In *Chinese National Conference on Social Media Processing*, pages 76–87. Springer, 2014.

[84] S. Jiang, Y. Hu, C. Kang, T. Daly Jr, D. Yin, Y. Chang, and C. Zhai. Learning query and document relevance from a web-scale click graph. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 185–194. ACM, 2016.

[85] P. Joyce. *The state of freedom: A social history of the British state since 1800*. Cambridge University Press, 2013.

[86] N. Kapur, N. Lytkin, B. Chen, D. Agarwal, and I. Perisic. Ranking universities based on career outcomes of graduates. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–144. ACM, 2016.

[87] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations*, 2015.

[88] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[89] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *Proceedings of the International Conference on Learning Representations*, 2017.

[90] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[91] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[92] B. S. Kumar and V. Ravi. A survey of the applications of text mining in financial domain. *Knowledge-Based Systems*, 114:128–147, 2016.

[93] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale. *Proceedings of the International Conference on Learning Representations*, 2017.

[94] J. Lages, A. Patt, and D. L. Shepelyansky. Wikipedia ranking of world universities. *The European Physical Journal B*, 89(3):1–12, 2016.

[95] E. A. Leicht, P. Holme, and M. E. Newman. Vertex similarity in networks. *Physical Review E*, 73(2):026120, 2006.

[96] L. Li and T. Li. News recommendation via hypergraph learning: encapsulation of user behavior and news content. In *Proceedings of the 6th ACM international conference on Web search and data mining*, pages 305–314, 2013.

[97] Q. Li, Y. Chen, L. L. Jiang, P. Li, and H. Chen. A tensor-based information framework for predicting the stock market. *ACM Transactions on Information Systems*, 34(2):11, 2016.

[98] Q. Li, Y. Chen, J. Wang, Y. Chen, and H. Chen. Web media and stock markets: A survey and future directions from a big data perspective. *IEEE Transactions on Knowledge and Data Engineering*, 30(2):381–399, 2018.

[99] X. Li, H. Chen, J. Li, and Z. Zhang. Gene function prediction with gene interaction networks: a context graph kernel approach. *IEEE Transactions on Information Technology in Biomedicine*, 14(1):119–128, 2010.

[100] F. Liao, M. Liang, Y. Dong, and T. Pang. Defense against adversarial attacks using high-level representation guided denoiser. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[101] Q. Liu, Y. Sun, C. Wang, T. Liu, and D. Tao. Elastic net hypergraph learning for image clustering and semi-supervised classification. *Transactions on Image Processing*, 26(1):452–463, 2017.

[102] T.-Y. Liu. *Learning to rank for information retrieval.* Springer Science & Business Media, 2011.

[103] Y. Liu, B. Gao, T. Liu, Y. Zhang, Z. Ma, S. He, and H. Li. Browserank: letting web users vote for page importance. In *Proceedings of the 31st International ACM*

SIGIR Conference on Research and Development in Information Retrieval, pages 451–458. ACM, 2008.

[104] A. W. Lo and A. C. MacKinlay. *A non-random walk down Wall Street.* Princeton University Press, 2002.

[105] H. Ma, T. C. Zhou, M. R. Lyu, and I. King. Improving recommender systems by incorporating social contextual information. *ACM Transactions on Information Systems*, 29(2):9:1–9:23, 2011.

[106] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, volume 30, page 3, 2013.

[107] O. Megorskaya, V. Kukushkin, and P. Serdyukov. On the relation between assessor's agreement and accuracy in gamified relevance assessment. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 605–614. ACM, 2015.

[108] T. Mei, Y. Rui, S. Li, and Q. Tian. Multimedia search reranking: A literature survey. *ACM Computing Surveys*, 46(3):38, 2014.

[109] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[110] M. Mitzenmacher, J. Pachocki, R. Peng, C. Tsourakakis, and S. C. Xu. Scalable large near-clique detection in large-scale networks via sampling. In *Proceedings of the 21st ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 815–824, 2015.

[111] T. Miyato, A. M. Dai, and I. Goodfellow. Adversarial training methods for semi-supervised text classification. *Proceedings of the International Conference on Learning Representations*, 2017.

[112] T. Miyato, S.-i. Maeda, S. Ishii, and M. Koyama. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[113] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *The IEEE Conference on Computer Vision and Pattern Recognition*, July 2017.

[114] G. L. Musgrave. A random walk down wall street. *Business Economics*, 32(2):74–76, 1997.

[115] A. K. Nassirtoussi, S. Aghabozorgi, T. Y. Wah, and D. C. L. Ngo. Text mining for market prediction: A systematic review. *Expert Systems with Applications*, 41(16):7653–7670, 2014.

[116] R. Nelsen. Kendall tau metric. *Encyclopaedia of Mathematics*, 3:226–227, 2001.

[117] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *International Conference on Machine Learning*, pages 689–696, 2011.

[118] T. H. Nguyen and K. Shirai. Topic modeling based sentiment analysis on social media for stock market prediction. In *ACL*, volume 1, pages 1354–1364, 2015.

[119] J. Ni, S. Chang, X. Liu, W. Cheng, H. Chen, D. Xu, and X. Zhang. Co-regularized deep multi-network embedding. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 469–478. International World Wide Web Conferences Steering Committee, 2018.

[120] Z. Nie, Y. Zhang, J. Wen, and W. Ma. Object-level ranking: bringing order to web objects. In *Proceedings of the 14th international conference on World Wide Web*, pages 567–574. ACM, 2005.

[121] K. Nowicki and T. A. B. Snijders. Estimation and prediction for stochastic blockstructures. *Journal of the American statistical association*, 96(455):1077–1087, 2001.

[122] A. Omari, D. Carmel, O. Rokhlenko, and I. Szpektor. Novelty based ranking of human answers for community questions. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 215–224. ACM, 2016.

[123] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114. ACM, 2016.

[124] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. Technical Report 66, 1999.

[125] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[126] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang. Adversarially regularized graph autoencoder for graph embedding. In *The international joint conference on artificial intelligence*, pages 2609–2615, 2018.

[127] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[128] J. J. Pfeiffer III, S. Moreno, T. La Fond, J. Neville, and B. Gallagher. Attributed graph models: Modeling network structure with correlated attributes. In *Proceedings of the 23rd international conference on World wide web*, pages 831–842. ACM, 2014.

[129] G. Preethi and B. Santhi. Stock market forecasting techniques: A survey. *Journal of Theoretical & Applied Information Technology*, 46(1), 2012.

[130] A. Raghunathan, J. Steinhardt, and P. Liang. Certified defenses against adversarial examples. *Proceedings of the International Conference on Learning Representations*, 2019.

[131] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabási. Hierarchical organization of modularity in metabolic networks. *science*, 297(5586):1551–1555, 2002.

[132] M.-A. Rizoiu, L. Xie, S. Sanner, M. Cebrian, H. Yu, and P. Van Hentenryck. Expecting to be hip: Hawkes intensity processes for social media popularity. In *Proceedings of the 26th international conference on World Wide Web*, pages 735–744, 2017.

[133] L. Sang, M. Xu, S. Qian, and X. Wu. Aaane: Attention-based adversarial autoencoder for multi-scale network embedding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[134] R. P. Schumaker and H. Chen. Evaluating a news-aware quantitative trader: The effect of momentum and contrarian stock selection strategies. *Journal of the American Society for Information Science and technology*, 59(2):247–255, 2008.

[135] R. P. Schumaker and H. Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. *ACM Transactions on Information Systems*, 27(2):12, 2009.

[136] G. W. Schwert. Stock volatility in the new millennium: how wacky is nasdaq? *Journal of Monetary Economics*, 49(1):3–26, 2002.

[137] 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.

[138] S. Shekhar, V. M. Patel, N. M. Nasrabadi, and R. Chellappa. Joint sparse representation for robust multimodal biometrics recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(1):113–126, 2014.

[139] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems*, pages 926–934, 2013.

[140] X. Song, F. Feng, X. Han, X. Yang, W. Liu, and L. Nie. Neural compatibility modeling with attentive knowledge distillation. In *The 41st International ACM SIGIR Conference on Research &#38; Development in Information Retrieval*, 2018.

[141] X. Song, F. Feng, J. Liu, Z. Li, L. Nie, and J. Ma. Neurostylist: Neural compatibility modeling for clothing matching. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 753–761. ACM, 2017.

[142] C. Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 100:441–471, 1987.

[143] N. Srivastava, E. Mansimov, and R. Salakhudinov. Unsupervised learning of video representations using lstms. In *International Conference on Machine Learning*, pages 843–852, 2015.

[144] Y. Sun and J. Han. Mining heterogeneous information networks: a structural analysis approach. *Acm Sigkdd Explorations Newsletter*, 14(2):20–28, 2013.

[145] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *Proceedings of the International Conference on Learning Representations*, 2014.

[146] P. P. Talukdar and K. Crammer. New regularized algorithms for transductive learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 442–457. Springer, 2009.

[147] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[148] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu. Learning deep representations for graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1293–1299, 2014.

[149] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. Ensemble adversarial training: Attacks and defenses. *Proceedings of the International Conference on Learning Representations*, 2018.

[150] K. Tran, S. Hosseini, L. Xiao, T. Finley, and M. Bilenko. Scaling up stochastic dual coordinate ascent. In *Proceedings of the 21st ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1185–1194, 2015.

[151] C. E. Tsourakakis, J. Pachocki, and M. Mitzenmacher. Scalable motif-aware graph clustering. In *Proceedings of the 27th international conference on World Wide Web*, pages 1451–1460, 2017.

[152] W. Tu, D. W. Cheung, N. Mamoulis, M. Yang, and Z. Lu. Investment recommendation using investor opinions in social media. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 881–884. ACM, 2016.

[153] K. Ura, S. Alkire, and T. Zangmo. A short guide to gross national happiness index. 2012.

[154] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *Proceedings of the International Conference on Learning Representations*, 1(2), 2018.

[155] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *Proceedings of the International Conference on Learning Representations*, 1(2), 2018.

[156] D. Vrandečić and M. Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[157] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.

[158] D. Wang, T. Li, S. Zhu, and C. Ding. Multi-document summarization via sentence-level semantic analysis and symmetric matrix factorization. In *Proceedings of the 31st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 307–314. ACM, 2008.

[159] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo. Graphgan: Graph representation learning with generative adversarial nets. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[160] M. Wang, H. Li, D. Tao, K. Lu, and X. Wu. Multimodal graph-based reranking for web image search. *Transaction on Image Processing*, 21(11):4649–4661, 2012.

[161] M. Wang, X. Liu, and X. Wu. Visual classification by ℓ_1 -hypergraph modeling. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2564–2574, 2015.

[162] W. Wang, R. Arora, K. Livescu, and J. Bilmes. On deep multi-view representation learning. In *International Conference on Machine Learning*, pages 1083–1092, 2015.

[163] X. Wang, F. Nie, and H. Huang. Structured doubly stochastic matrix for graph based clustering: Structured doubly stochastic matrix. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1245–1254, 2016.

[164] X. Wang, L. Nie, X. Song, D. Zhang, and T. Chua. Unifying virtual and physical worlds: Learning toward local and global consistency. *ACM Transactions on Information Systems*, 36:4:1–4:26, 2017.

[165] X. Wang, D. Wang, C. Xu, X. He, Y. Cao, and T.-S. Chua. Explainable reasoning over knowledge graphs for recommendation. *arXiv preprint arXiv:1811.04540*, 2018.

[166] Z. Wang, Z. Jiang, Z. Ren, J. Tang, and D. Yin. A path-constrained framework for discriminating substitutable and complementary products in e-commerce. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 619–627. ACM, 2018.

[167] M. Weimer, A. Karatzoglou, Q. V. Le, and A. J. Smola. Cofirank maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems*, pages 1593–1600, 2007.

[168] J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the 25th international conference on Machine learning*, pages 1168–1175. ACM, 2008.

[169] J. Weston, F. Ratle, H. Mobahi, and R. Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[170] C. J. Willmott and K. Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[171] Y. Wu, D. Bamman, and S. Russell. Adversarial training for relation extraction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1778–1783, 2017.

[172] C. Xiong, R. Power, and J. Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the 26th international conference on world wide web*, pages 1271–1279. International World Wide Web Conferences Steering Committee, 2017.

[173] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–273. ACM, 2003.

[174] Y. Xu and G. Zhang. Application of kalman filter in the prediction of stock price. In *International Symposium on Knowledge Acquisition and Modeling (KAM). Atlantis press*, pages 197–198, 2015.

[175] R. Yan, Y. Song, and H. Wu. Learning to respond with deep neural networks for retrieval-based human-computer conversation system. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 55–64. ACM, 2016.

[176] Z. Yang, W. Cohen, and R. Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, pages 40–48, 2016.

[177] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &#38; Data Mining*, pages 974–983, 2018.

[178] Y. Yoshida. Almost linear-time algorithms for adaptive betweenness centrality using hypergraph sketches. In *Proceedings of the 20th ACM SIGKDD international*

conference on Knowledge discovery and data mining, pages 1416–1425, 2014.

[179] H.-F. Yu, C.-J. Hsieh, H. Yun, S. Vishwanathan, and I. S. Dhillon. A scalable asynchronous distributed algorithm for topic modeling. In *Proceedings of the 25th international conference on World Wide Web*, pages 1340–1350, 2015.

[180] R. Yu, H. Qiu, Z. Wen, C. Lin, and Y. Liu. A survey on social media anomaly detection. *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 18(1):1–14, 2016.

[181] W. Yu, C. Zheng, W. Cheng, C. C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang. Learning deep network representations with adversarially regularized autoencoders. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2663–2671. ACM, 2018.

[182] D. Zhang, J. Yin, X. Zhu, and C. Zhang. Network representation learning: a survey. *IEEE Transactions on Big Data*, 2018.

[183] H. Zhang, X. Shang, H. Luan, M. Wang, and T. Chua. Learning from collective intelligence: Feature learning using social images and tags. *ACM transactions on multimedia computing, communications, and applications*, 13:1:1–1:23, 2017.

[184] L. Zhang, C. Aggarwal, and G.-J. Qi. Stock price prediction via discovering multi-frequency trading patterns. In *Proceedings of the 23rd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 2141–2149. ACM, 2017.

[185] Y. Zhang, Y. Xiong, X. Kong, and Y. Zhu. Learning node embeddings in interaction graphs. In *Proceedings of the 26th ACM International on Conference on Information and Knowledge Management*, pages 397–406. ACM, 2017.

[186] Z. Zhang, P. Cui, and W. Zhu. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202*, 2018.

[187] S. Zhao, Q. Wang, S. Massung, B. Qin, T. Liu, B. Wang, and C. Zhai. Constructing and embedding abstract event causality networks from text snippets. In *Proceedings of the tenth ACM international conference on Web search and data mining*, pages 335–344. ACM, 2017.

[188] Z. Zheng, K. Chen, G. Sun, and H. Zha. A regression framework for learning ranking functions using relative relevance judgments. In *Proceedings of the 30th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 287–294. ACM, 2007.

[189] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328, 2004.

[190] D. Zhou, J. Huang, and B. Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in neural information processing systems*, pages 1601–1608, 2007.

[191] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. Ranking on data manifolds. In *Advances in neural information processing systems*, pages 169–176, 2004.

[192] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[193] X. Zhou, M. Belkin, and N. Srebro. An iterated graph laplacian approach for ranking on manifolds. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 877–885. ACM, 2011.

[194] D. Zhu, P. Cui, D. Wang, and W. Zhu. Deep variational network embedding in wasserstein space. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2827–2836. ACM, 2018.

[195] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.

[196] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning*, pages 912–919, 2003.

[197] D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 2847–2856. ACM, 2018.