SAPIENZA
UNIVERSITÀ DI ROMA

Real Estate Value Prediction Based On Knowledge Graph

Facoltà di
INGEGNERIA DELL'INFORMAZIONE, INFORMATICA E STATISTICA
Dipartimento di COMPUTER SCIENCE - INFORMATICA
Corso di laurea in master's degree

Dongyuan Wu
ID number 1955113

Advisor
Prof. Giuseppe Pirro

Academic Year 2023/2024

Abstract

In recent years, knowledge graphs have emerged as powerful data structures for processing non-Euclidean data across various domains. These graphs integrate diverse information into a unified, structured knowledge base, mirroring human understanding of the world. Despite their widespread adoption in many fields, their application in real estate remains relatively limited. Real estate data is characterized by complex relationships, the need for cross-domain data integration, and the requirement for intelligent decision-making.

Knowledge graphs offer significant potential in the real estate industry, enabling enterprises to enhance data management, improve service quality, innovate business models, and gain competitive advantages. With the rapid advancements in graph neural networks (GNNs), another approach to addressing knowledge graph challenges has emerged. GNNs excel in structured modeling, and the emergence of deep learning and heterogeneous GNNs like Relational Graph Convolutional Networks (RGCN) and Heterogeneous Graph Transformer (HGT) has led to a growing trend of utilizing heterogeneous graph neural networks in knowledge graphs.

This paper comprises three main parts. Firstly, it leverages data from Russia's largest real estate advertising website to construct a knowledge graph. Secondly, it investigates the application of heterogeneous graph transformation neural networks in real estate value assessment. Finally, it compares the prediction results obtained from this method with those derived from traditional algorithms, ultimately drawing a positive conclusion regarding its efficacy.

Contents

Chapter 1

Introduction

In recent years, the application of knowledge graphs (KGs) has attracted a lot of attention in various fields, providing a powerful framework for organizing, integrating, and analyzing complex structured and unstructured data. Among these areas, the real estate industry has a lot to gain from the adoption of knowledge graph technology. With the large number of heterogeneous data sources available in the real estate space, including property listings, market trends, economic indicators, geographic information, and customer preferences, building a comprehensive knowledge graph can provide valuable insights and facilitate a variety of tasks such as real estate valuation, market analysis, and decision making. This paper focuses on building a knowledge graph for real estate and discusses its application in real estate price forecasting. By representing real estate entities and their relationships in a graph-based structure, we aim to capture the rich semantics and complex interdependencies inherent in the real estate ecosystem. In addition, we utilize the emerging heterogeneous graph Converter (HGT) technology for real estate price prediction in the constructed knowledge graph. HGT is a powerful deep learning framework designed to work with heterogeneous graphs, where nodes and edges can belong to different types and exhibit a variety of characteristics. By effectively capturing the structural and semantic information encoded in the knowledge graph, HGT enables us to model complex relationships between various real estate entities and derive accurate real estate price predictions. Through this research effort, we aim to demonstrate the effectiveness of knowledge graphs in addressing the unique challenges of the real estate industry and demonstrate the potential of advanced graph-based machine learning techniques like HGT to improve decision making and predictive analytics in the real estate market. By tapping into the wealth of data in the real estate sector and leveraging the latest technologies, we aim to empower stakeholders with actionable insights and enable smarter and more efficient real estate transactions and investments. HGT, known for its ability to accommodate the inherent diversity of real estate datasets, provides a sophisticated framework for representing various entities such as properties, locations, market trends, and demographic information. By encoding this wealth of information into a unified knowledge graph, HGT empowers analysts and stakeholders to identify complex relationships, spot emerging patterns, and make informed decisions. Until then, many different graph algorithms can be used to process graph data: from traditional techniques such as tag propagation and PageRank, to advanced graph neural network architectures such as DeepWalk, RGCN, and GAT, each algorithm has its own unique advantages. For example, tag propagation facilitates neighborhood detection and node classification in real estate maps, helping us identify clusters of properties with similar characteristics or market trends. At the same time, PageRank enables us to assess the centrality and influence of different entities in the real estate ecosystem, revealing key players, high-value attributes, and emerging market trends. DeepWalk, on the other hand, helps to learn low-dimensional representations of real estate entities by capturing structural and semantic relationships in diagrams. These embeddings serve as powerful capabilities for downstream tasks such as property recommendation, market segmentation, and anomaly detection. In addition, we delve into the field of graph neural networks, a class of deep learning models specifically designed for graph-structured data. Our research covers architectures such as GCN, DeepWalk and others designed to process graph data, as well as architectures such as GAT that utilize attention mechanisms to capture different levels of importance between neighboring nodes. Compared with traditional machine learning methods, such as catboost, lstm, etc., conclusions are drawn

The structure of the remaining sections of the thesis is as follows:
In Chapter 2, we will explore the current state of the art in the field of neural network and real estate knowledge graph.

Chapters 3, we will discuss knowledge graph

Chapters 4, 5,6,and 7 will provide a detailed discussion of the theoretical foundations and mathematical formulations of key components, including the Transformer, Graph Convolution Network(GCN), Heterogeneous Graph Transformer, and Traditional graph algorithm, respectively.

Chapter 8 will present a detailed account of the experiments carried out in this study.

Chapter 9 will present the forecast results and make a comparative analysis.

Finally,Chapter 10 will offer the conclusions drawn from the thesis.

Chapter 2

Related Work

2.1 Hetegeneous Graph mining:
Heterogeneous graphs [6] (a.k.a., heterogeneous information networks) are an important abstraction for modeling relational data for many real-world complex systems. Heterogeneous Graph refers to a graph structure that contains multiple types of nodes and edges. In contemporary data-driven research, heterogeneous graphs stand out as particularly rich representations capable of capturing complex relationships between different entities and attributes. It aims to extract meaningful insights, patterns, and knowledge from these complex and heterogeneous data structures.

The core of heterogeneous graph mining is to use the heterogeneous structure and semantics of graphs to discover hidden patterns, associations and embedded knowledge. This work involves developing and applying advanced algorithms, techniques, and methods to effectively deal with the complexity, diversity, and scale of heterogeneous graphical data.Heterogeneous graph mining encompasses a wide range of tasks, including but not limited to: Node classification[7] and entity link prediction[8], Community detection and clustering[9], Path analysis and knowledge Discovery[9],Graph representation learning and embedding[8].

2.2 Graph Neural Network
Graph neural networks(GNNs)[10], as a sub field of deep learning tailored for non-Euclidean spaces, exhibit exceptional performance in a wide array of tasks that involve the processing of data structured as graphs. GNNs extend the principles of deep learning to graph data by enabling the propagation of information across nodes in a graph. Unlike traditional neural networks that

operate on fixed-size vectors, GNNs operate directly on the graph structure, allowing them to incorporate both node features and graph topology into their learning process. By iteratively aggregating information from neighboring nodes, GNNs can effectively learn node representations that capture both local and global graph properties.

The core components of a GNN include node embedding, message passing, aggregation function, and output layer. Node embedding involves learning low-dimensional representations for each node in the graph, capturing both structural and feature-based information. Message passing enables information propagation across the graph, where nodes aggregate information from their neighbors and update their own representations. The aggregation function defines how information from neighboring nodes is combined, with common approaches including summation, averaging, or attention mechanisms. Finally, the output layer predicts node labels, edge labels, or graph-level properties based on the learned node representations.

GNNs have demonstrated remarkable success across a wide range of tasks, including node classification, link prediction, graph classification, and recommendation systems[30]. Their ability to capture both local and global graph properties makes them particularly effective for tasks where understanding the relationships between entities is crucial.

2.3 Real Estate Knowledge Graph

Knowledge graph is a semantic network of knowledge which is represented by a graph structure composed of nodes and edges/arcs. Nodes represent objects, and edges represent relationships between them. These networks are intuitive and have greater cognitive adequacy, which is the ability to solve problems in certain sociocultural contexts. Moreover, knowledge is organized in this semantic network, making it easy for machines to interpret it and extract domain-specific data. Knowledge graphs are a great solution for dealing with complex, multi-dimensional relationships, from the medical field, social media, to cybersecurity, the demand for knowledge graphs is increasing, and in the real estate field where data has complex relationships and diverse properties, knowledge graphs are a great option to better understand market trends and ultimately make more informed decisions

Chapter 3

Knowledge Graph

3.1 Preliminary Knowledge

In the era of big data, the knowledge graph has become a powerful tool for organizing, integrating, and querying large amounts of structured and unstructured information.Knowledge graphs, which encode entities, attributes, and relationships in a semantic graph structure, provide valuable context and background knowledge that can enhance the learning process of GNNs, in entity classification tasks, KGs provide valuable context and background information that can improve the classification accuracy of GNNs[4]. By leveraging the hierarchical structure and semantic relationships encoded in KGs, GNNs can effectively propagate information and make more informed predictions about the class labels of entities.. The synergy between knowledge graphs and graph neural networks is particularly advantageous in scenarios where data is heterogeneous[8], sparse, or noisy, such as in recommendation systems, question answering, entity classification, and link prediction. By incorporating knowledge from KGs into GNNs, these models can better capture the semantics of entities and relationships, generalize across domains, and handle data with missing or incomplete information.

The process of building a knowledge graph involves collecting and integrating data from different sources and transforming it into a graphical structure. Common sources of data include structured databases, semi-structured or unstructured text, web data, etc. The process of constructing knowledge graph includes entity recognition, relation extraction, knowledge representation, etc., aiming at capturing the relationship and semantic information among entitys

Knowledge graph can be divided into two kinds according to its construction knowledge system and coverage: (1) General knowledge graph, which is mainly based on general knowledge graph based on knowledge, mainly encyclopedia data source, extracted from structured encyclopedia data, emphasizing the coverage of knowledge. For example, in the Knowledge Graph open source library, DBpedia extracts entities from structured Wikipedia and displays them [34]. (2) Domain knowledge graph. Different from the general knowledge graph, the domain Knowledge graph is more specialized. It is mainly aimed at a certain professional field, such as medicine, film knowledge, etc. The knowledge graph is constructed by defining the entities in the professional field and the relationship between the entities, so as to exert its knowledge storage and application in related fields. At present, knowledge graph of vertical domain is mainly constructed through ontology fusion and database integration.

3.2 Elements of knowledge graph

The components of a knowledge graph are: Entities: In a knowledge graph, entities represent objects, concepts, or entities in the real world, such as people, places, events, etc. Each entity has a unique identifier and a set of attributes. Attributes: Attributes describe the characteristics or attributes of the entity, such as the entity's name, age, location, and so on. Attributes can be discrete or continuous and can contain multiple types of information. Relations: Relations represent semantic associations or connections between entities. For example, a "located" relationship can connect a place entity to a building entity, indicating that the building is located in that location.

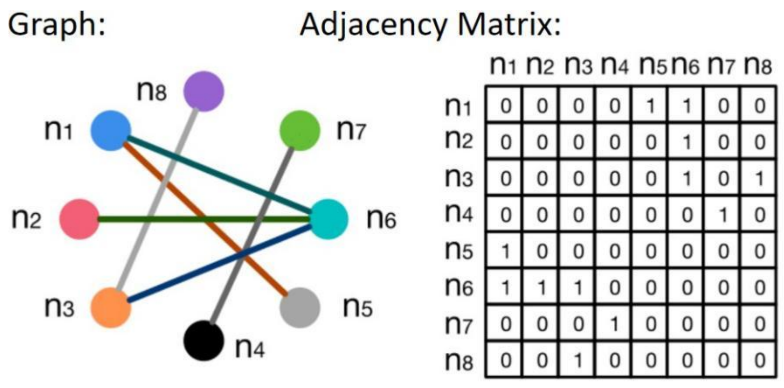
Chapter 4

Traditional Graph Algorithm

4.1 Graph

Graphs are fundamental mathematical structures that represent relationships between objects. In a graph, objects are represented as nodes or vertices, while the relationships between them are depicted as edges. Graphs are widely used in various fields, including computer science, social networks, transportation networks, and biological systems. Graphs provide a powerful framework for modeling and analyzing complex systems. They allow us to study the connectivity, dependencies, and interactions between entities. Graph-based algorithms and techniques enable us to extract meaningful insights[15][16][17], detect patterns[18][19], and solve problems such as graph clustering[20][21], community detection[22][23], shortest path finding[24][25], and recommendation systems[26][27].

Additionally, the graph can be visualized using a node-link diagram, where vertices are represented as nodes or circles, and edges are depicted as lines or arcs connecting the nodes. The layout and arrangement of the nodes and edges in the diagram provide insights into the connectivity and structure of the graph. An example of a graph, as illustrated in Figure 4.1



	n1	n2	n3	n4	n5	n6	n7	n8
n1	0	0	0	0	1	1	0	0
n2	0	0	0	0	0	1	0	0
n3	0	0	0	0	0	1	0	1
n4	0	0	0	0	0	0	1	0
n5	1	0	0	0	0	0	0	0
n6	1	1	1	0	0	0	0	0
n7	0	0	0	1	0	0	0	0
n8	0	0	1	0	0	0	0	0

Figure 4.1. A visual representation of the relationships between nodes and edges in a graph. In this example, we have a set of nodes ν = {v1, ..., v8} and a set of edges ε = {e1, ..., e6}. Each vertex represents a distinct entity, while the edges signify the connections or interactions between these entities

A graph is defined as $G = (v, \varepsilon)$, which includes n nodes $v = \{v_1, \dots, v_n\}$ and m edges $\varepsilon = \{e_1, \dots, e_m\}$, associated with features $X^{v} \epsilon R^{n \times C}$ and $X^{\varepsilon} \epsilon R^{m \times C}$. The graph G can be denoted by a $n \times n$ adjacency matrix A, with entries defined as

$$A(i,j) = \begin{cases} 1, & if\ (v_i, v_j) \in e \\ 0, & if\ (v_i, v_j) \notin e \end{cases} \qquad (4,1)$$

Additionally, in some cases, weighted graphs are considered, where the edges have associated weights. In such cases, the adjacency matrix can contain the weight values instead of just 0s and 1s, the adjacency matrix is defined as follows:

$$A(i,j) = \begin{cases} W_{I,J}, & if\ (v_i, v_j) \in e \\ 0, & if\ (v_i, v_j) \notin e \end{cases} \qquad (4,2)$$

The graph Laplacian is a mathematical operator that is derived from the adjacency matrix of a graph. It is a fundamental concept in graph theory and plays a crucial role in various graph-based algorithms and analyses. It is a powerful tool that aids in understanding and analyzing the properties and behavior of graphs, making it an essential concept in graph theory and its applications.

For the graph $G = (v, \varepsilon)$, the graph Laplacian matrix L of G is defined as $L = D - A$, where D is the degree matrix and A is the adjacency matrix of the graph. The graph Laplacian matrix L provides important information about the graph's structure and properties. It captures the relationships between vertices and their neighboring vertices, enabling various graph analysis techniques such as spectral graph theory[28], graph clustering, graph partitioning, and solving graph-based optimization problems.

The normalized Laplacian matrix eigen decomposition[29], also referred to as spectral decomposition, can be defined as follows:

$$\Delta = \mathrm{I} - \mathrm{D}^{\frac{1}{2}}\mathrm{A}\mathrm{D}^{-\frac{1}{2}} = \mathrm{U}^{\mathrm{T}}\Lambda\mathrm{U} \qquad (4.3)$$

Where I is identity matrix, Λ represents the eigenvalues, and U corresponds to the eigenvectors. The eigenvectors associated with the normalized Laplacian matrix are orthogonal to each other. The eigenvectors of graph capture unique structural patterns within the graph. These eigenvectors correspond to different frequencies or modes of oscillation, with the lower eigenvectors representing global structural information and the higher eigenvectors capturing more localized features[6]. The eigenvectors are orthogonal to each other, meaning they are linearly independent and do not share any common structural characteristics. By examining and analyzing these eigenvectors, we can gain insights into the underlying connectivity, community structure, and other important properties of the graph.

4.2 Label Propagation

LPA is a local community division based on label propagation. For each node in the network, in the initial phase, Label Propagation algorithm initializes a unique label for each node. Each iteration will change its own label according to the label of the node connected to it, and the principle of change is to choose the community label with the most labels in the node connected to it as its own community label, which is the meaning of tag propagation. As the community tag continues to spread. Eventually, tightly connected nodes will have a common label. LPA believes that the label of each node should be the same as that of most of its neighbors. It takes the label with the largest number among the labels of a node's neighbors as the label of the node itself, adds a label to each node to represent the community to which it belongs, and forms the same "community" with the same "label" through the "propagation" of labels. In the basic idea, LPA and Kmean are very similar in nature. In each iteration of LPA, the community to which a node belongs depends on the label with the largest cumulative weight among its neighbors, while Kmeans is the community that is "nearest" to the current node and the community to which the node belongs.
The biggest advantage of the LPA algorithm is that the logic of the algorithm is very simple, and the process of optimizing the modularity algorithm is very fast, without pylouvain's multiple iteration optimization process.Formula representation:

$$l_i^{(t+1)} = argmax \sum_{v_j \in N(v_i)} \delta(l_j^{(t)}, l_k)$$

The characteristics of the label propagation algorithm are:
Simple and efficient: LPA does not need to explicitly define the model, but realizes the clustering or classification of nodes through a simple label propagation

mechanism, so the algorithm is simple and efficient. Wide application range: LPA can be applied to various types of graphs, including social networks, biological networks, communication networks, etc., with a wide range of applications.
Nondeterminism: Since label propagation is decided by voting based on the labels of neighboring nodes, LPA has a certain degree of nondeterminism and may be affected by the initial label and graph structure.
Convergence: LPA can be guaranteed to converge under certain conditions, but it is not guaranteed to converge to the global optimal solution.

4.3 Page Rank

The PageRank algorithm was originally proposed by Larry Page and Sergey Brin in 1996, and although PageRank was originally designed for Internet pages, its core idea can be broadly applied to any digraph structure. In fact, over time, PageRank has been used for a variety of applications, such as social impact analysis, text summary generation, and more. The core idea of PageRank is based on a random walk model on a directed graph, which is a first-order Markov chain. The model describes how a random walker moves randomly along the edges of a graph, accessing from one node to another. If certain conditions are satisfied, the random walk process will eventually converge to a stationary distribution. In this stationary distribution, the probability of each node being visited is its PageRank value, which can be interpreted as the importance of the node. It is important to note that PageRank is recursively defined, which means that the PageRank value of a page depends in part on the PageRank value of other pages that link to it. Therefore, calculating the PageRank value usually requires an iterative approach.

In the early history of the Internet, the PageRank algorithm was introduced as a way to assess the importance of web pages. In short, PageRank is a function defined on the entire collection of web pages that assigns each web page a positive real number that represents the importance of that web page. These values form a vector where a higher PageRank value means that the page is more important and therefore likely to be shown in search results.

You can think of the entire Internet as a giant directed graph, where each web page is a node and hyperlinks are directed edges that lead from one page to another. Based on this perspective, we can construct a random walk model, which is a first-order Markov chain. In this model, we assume that a virtual web viewer will randomly follow any hyperlink on one page to another page with equal probability, and continue this random jump. Over a long period of time, the behavior of this random jump will form a stable pattern, that is, the smooth distribution of Markov chains. The PageRank value of each web page is actually the probability in this stationary distribution.

4.4 DeepWalk

DeppWalk is a kind of graph shallow encoder. In 2014, Bryan did Deepwalk work by randomly walking from one node to another, turning the network into a sequence[3]. It is a graph structure data mining algorithm that combines the random walk and word2vec algorithms. The algorithm can learn the hidden information of the network and can represent the nodes in the graph as a vector containing the underlying information

4.4.1 Random Walk

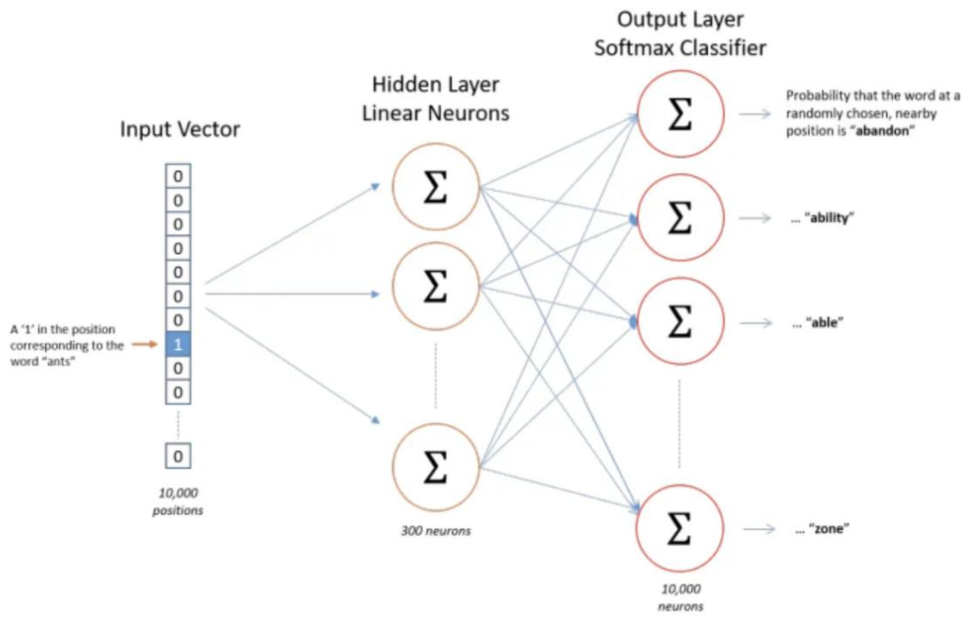
DeepWalk first performs a random walk in the graph, starting from each node and selecting the next node to move according to certain probability rules. These wandering paths connect nodes in the graph and capture structural information between nodes. The length and number of walks of each walk path can be set in advance, usually choosing a longer walk path and multiple walks to better capture the local information of the graph structure.

4.4.2 Neural Network Embedding

Skip-gram model: Based on the sequence of nodes obtained by random walks, Skip-gram model is used to learn the embedded representation of nodes. In the Skip-gram model, the goal is to predict the node itself based on its context, that is, given a central node, predict the nodes around it. The skip gram model is like the following figure4.2.

Node Embedding: The Skip-gram model learns the node embedding vector by maximizing the joint probability of the node sequence so that nodes with similar contexts are closer together in the embedding space, while nodes with different contexts are further apart in the embedding space

After learning the embedded representation of nodes, the embedded vector can be used to calculate the similarity between nodes, for example, the similarity between nodes can be measured by calculating the cosine similarity between vectors, and the similarity between nodes can be used for node classification, link prediction, etc.



Figue 4.2 :the structure of skip gram,training examples are (input word, output word) ,and the output of the final model is a probability distribution

Chapter 5

Graph Convolutional Networks

5.1 Preliminary Knowledge

Addressing the task of categorizing nodes, like documents, within a graph, which may represent a network of citations, and when labels are accessible for only a limited portion of nodes, this issue can be conceptualized as semi-supervised learning within a graph-based context. In this framework, label data is diffused or spread throughout the graph through the incorporation of a specific graph-based regularization technique. One approach is to include a term in the loss function that leverages a graph Laplacian regularization:

$$\mathcal{L} = \mathcal{L}_0 + \lambda\mathcal{L}_{\text{reg}}, \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{i,j}\left\|f(X_i) + f(X_j)\right\|^2 = f(X)^T\Delta f(X) \tag{5.1}$$

Where $\mathcal{L}_0$ represents the supervised loss concerning the labeled portion of the graph. The function $f(\cdot)$ can take the form of a neural network or a differentiable function, and λ is a weighting factor. The matrix X contains node feature vectors denoted as X_i. The symbol $\Delta = D - A$ stands for the unnormalized graph Laplacian of an undirected graph denoted as $G = (V, \varepsilon)$, with a total of N nodes $v_i \epsilon V$, edges $(v_i, v_j)\epsilon\varepsilon$, an adjacency matrix $A\epsilon R^{N\times N}$ (which can be binary or weighted), and a degree matrix $D_{ii} = \sum_j A_{ij}$. The formulation in Equation 5.1 is based on the assumption that connected nodes in the graph are likely to share the same label.

By incorporating the graph structure directly through a neural network model denoted as f(X, A) and training it on a supervised target L0, which includes all nodes with labels, we can eliminate the need for explicit graph-based regularization in the loss function. By conditioning the function $f(\cdot)$ on the graph's adjacency matrix, this approach enables the model to propagate gradient information from the supervised loss $\mathcal{L}_0$ and facilitates the learning of node representations, whether they have labels or not.

To begin, let's present a straightforward and reliable layer-wise propagation rule for neural network models designed to work directly on graphs. This rule can be justified by drawing inspiration from a first-order approximation of spectral graph convolutions.
Subsequently, we'll illustrate how this type of graph-based neural network model can be effectively employed for rapid and scalable semi-supervised node classification within a graph.

5.2 Fast approximate convolutions on graphs

Offering the theoretical underpinning for a particular graph-based neural network model, denoted as f(X, A), we examine a multi-layer Graph Convolutional Network [11] (GCN) that follows the subsequent layer-wise propagation rule:

$$H^{l+1} = \sigma(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}H^{(l)}W^{l}) \tag{5.2}$$

Where $\widetilde{A} = A + I_N$ represents the adjacency matrix of the undirected graph G, with added self-connections. Here, I_N signifies the identity matrix, $\widetilde{D_{ii}} = \sum_j \widetilde{A_{ij}}$ and $W^{(l)}$ is a layer-specific trainable weight matrix. The function σ(·) denotes an activation function, for instance, ReLU(·) = max(0, ·). The matrix $H^l \epsilon R^{N \times D}$ comprises activations in the l^{th} layer, with $H^{(0)} = X$ denoting the initial features of the nodes. The subsequent section will demonstrate that this propagation rule's structure can be rationalized through a first-order approximation of localized spectral filters applied to graphs.

5.2.1 Spectral graph convolutions

When examining spectral convolutions on graphs, these are characterized by the multiplication of a signal, denoted as $x \in \mathrm{R}^N$ (representing a scalar value for each node), with a filter g_ϑ = diag(θ), which is parametrized by $\theta \in \mathrm{R}^N$. This convolution operation takes place in the Fourier domain and can be expressed as follows:

$$g_\theta * x = U g_\theta U^T x \tag{5.3}$$

Here U represents the matrix of eigenvectors of the normalized graph

Laplacian, denoted as $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, with $\wedge$ being a diagonal matrix containing its eigenvalues. The expression $U^T x$ corresponds to the graph Fourier transform of x. The function g_θ can be seen as a function of the eigenvalues of L ,i.e , $g_\theta(\wedge)$. Calculating Eq.5.3 can be computationally demanding, as it involves multiplying with the eigenvector matrix U which has a time complexity of $\vartheta(N^2)$. Additionally the initial computation of the eigendecomposition of L might be impractical for large graphs. To address these challenges, it has been suggested that $g_\theta(\wedge)$ can be effectively approximated using a truncated expansion in terms of Chebyshev polynomial s$T_K(x)$ up to the K^{th}order:

$$g_{\theta'}(\wedge) \approx \sum_{k=0}^{K} \theta'_k T_k(\widetilde{\wedge}) \tag{5.4}$$

In this modified approach, use a rescaled $\widetilde{\wedge} = \frac{2}{\lambda_{max}} \wedge - I_N$,where$\lambda_{max}$ represents the largest eigenvalue of $\mathcal{L}$. The parameter vector $\theta' \epsilon \mathbb{R}^K$ now comprises Chebyshev coefficients The Chebyshev polynomials are defined recursively as $T_k(x) = 2xT_{k-2}(x)$,
with initial values $T_0(x) = 1$ and $T_1(x) = x$ Returning to the definition of convolution between a signal x and a filter $g_{\theta'}$,now express it as follows:

$$g_{\theta'} * x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \tag{5.5}$$

This equation introduces $\tilde{L} = \frac{2}{\lambda_{max}} L - I_N$. Its validity can be readily confirmed by observing that $(U \wedge U^T)^k = U \wedge^k U^T$ It's important to note that this expression is now K-localized as it constitutes a polynomial of order in the Laplacian.
In other words, it solely relies on nodes that are, at most, K steps away from the central node (within a K^{th}-order neighborhood)he computational complexity of evaluating Equation 5.5 is $O(|\varepsilon|)$meaning it scales linearly with the number of edges in the graph.

5.2.2 Layer-Wise linear model

A neural network model rooted in graph convolutions can be constructed by stacking multiple convolutional layers, each following the form of Equation 5.5, with each layer subsequently applying a point-wise nonlinearity. Now, consider constraining the layer-wise convolution operation to K = 1 (as shown in Equation 5.5). In this case, the function becomes linear with respect to L and is thus a linear operation on the graph Laplacian spectrum.

In this linear formulation of a GCN, we can further approximate λ_{max} as approximately 2, as neural network parameters will adapt to this scaling change during training. Under these approximations, Equation 5.5 simplifies to:

$$g_{\theta'} * x \approx \theta'_0 x + \theta'_1 (L - I_N) x = \theta'_0 x - \theta'_1 \widetilde{D}^{-\frac{1}{2}} \widetilde{A} D^{-\frac{1}{2}} x \tag{5.6}$$

This equation represents a simplified filter with only two free parameters,θ'_0 andθ'_1 1

Importantly, these filter parameters can be shared across the entire graph. When these filters are successively applied in multiple layers, they effectively convolve the K^{th}-order neighborhood of a node, where k corresponds to the number of successive filtering operations or convolutional layers in the neural network model.

In practical applications, it can be advantageous to impose additional constraints on the number of parameters to combat overfitting and reduce the computational load, including matrix multiplications, in each layer. This leads us to the following expression:

$$g_\theta * x \approx \theta(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) x \tag{5.7}$$

In this refined formulation, we are left with a single parameter$\theta = \theta'_0 =- \theta'_1$. It's worth nothing than the matrix$I_N + \widetilde{D}^{-\frac{1}{2}} \widetilde{A} D^{-\frac{1}{2}}$ now possesses eigenvalues within the range of [0, 2]. Consequently, repeated application of this operation can potentially introduce numerical instabilities, as well as issues related to exploding or vanishing gradients when employed in a deep neural network model. To mitigate this issue, introduce the following renormalization trick

$$I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \rightarrow \widetilde{D}^{-\frac{1}{2}}\widetilde{A}D^{-\frac{1}{2}}$$

With $\widetilde{A} = A + I_N$ and $\widetilde{D_{ii}} = \sum_j \widetilde{A_{ij}}$

We can extend this definition to a single input matrix $X \in \mathbb{R}^{N\times C}$,where C represents the number of input channels (i.e., a C-dimensional feature vector for each node), and F denotes the number of filters or feature maps, in the following manner:

$$Z = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta \tag{5.8}$$

In this formulation, $\Theta \in R^{C\times F}$ represents a matrix of filter parameters, and $Z \in R^{N\times F}$ is the resulting convolved signal matrix. This filtering operation has a computational complexity of $O(|\varepsilon| F C)$, as the product $\tilde{A}X$ can be efficiently executed as the product of a sparse matrix with a dense matrix.

5.3 Semi-Supervised node classification

After introducing a straightforward yet adaptable model, denoted as f(X, A), designed for efficient information propagation within graph structures, we can revisit the challenge of semi-supervised node classification. By conditioning our model, f(X, A), on both the input data X and the adjacency matrix A of the underlying graph, we have the potential to relax certain assumptions that are conventionally made in graph-based semi-supervised learning. This approach is particularly promising in situations where the adjacency matrix carries information that is not present in the data X. For instance, this may be the case in scenarios involving citation links among documents in a citation network or relationships within a knowledge graph. The complete model, a multi-layer Graph Convolutional

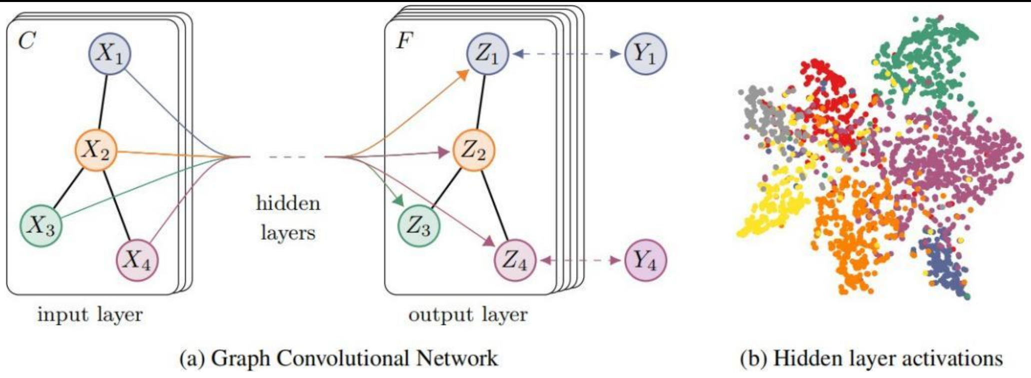


Figure 5.1. Lef t : Schematic illustration of a multi-layer Graph Convolutional Network (GCN) designed for semi-supervised learning. This GCN consists of C input channels and produces F feature maps in the output layer. The underlying graph structure, represented by black lines, is shared across different layers, and the labels are denoted as Y_i. Right : a t-SNE [31] visualization of the activations in the hidden layers of a two-layer GCN. This model was trained using only 5% of the available labels on the Cora dataset [32]. The colors in the visualization correspond to the document classes.

Network (GCN) for semi-supervised learning, is visually represented in Figure 5.1.

Chapter 6

Transformer

6.1 Arichitecture Overview

Ttransformer[35] is a powerful deep learning model that has revolutionized various areas of , including natural language processing (NLP) and computer vision. It was originally introduced in the context of machine translation, but one significant advantage of converter is its ability to handle remote dependencies more efficiently than traditional circular models. This makes it ideal for sequences of tasks involving long periods of time, such as machine translation, document summarization, and sentiment analysis. In addition, transformers introduce the concept of location coding, which injects location information into the input sequence. This helps the model differentiate elements based on their location, enabling it to understand the order and structure of the input data

Transformer can be regarded as a black box, whose structure is composed of several encoder and decoder. Encoder contains a Muti-Head Attention module, which is composed of multiple Self-Attention modules. The Decoder contains two Muti-Head Attention. There is also an Add & Norm layer after Muti-Head Attention, which is responsible for residual connections used to prevent network degradation. Norm is used to normalize the activation values of each layer.The structure is like the following figure6.1

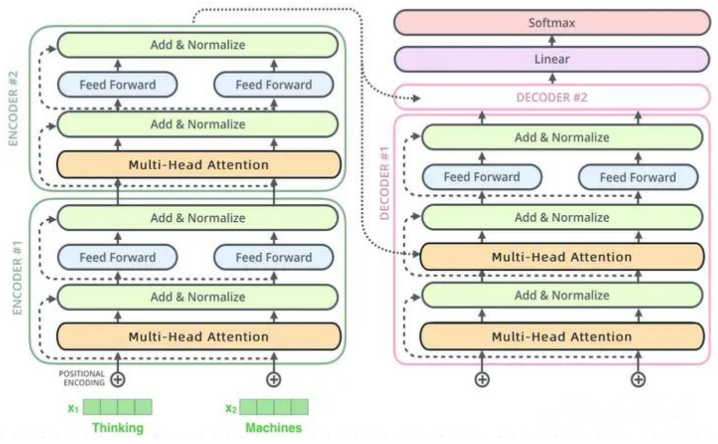


Figure 6.1 the Arichitecture of transformer

The transformer encoder accepts an input sequence and produces a representation of the sequence hidden. It consists of multiple layers, usually stacked on top of each other. Each layer of the encoder consists of two main sub-components: multi-head self-focused and feedforward neural networks.

6.2 Encoder

6.2.1 Self-Attention

Self-attention is used to calculate dependencies between different positions in the input sequence. The self-attention mechanism allows the model to take into account information from all other locations in the input sequence while generating a representation of each location, better capturing long-distance dependencies between sequences. In the self-attention mechanism, a given input sequence (x_1, x_2,... , x_n), where each x_i represents an element in the sequence (such as a word embedding vector), and our goal is to generate a representation vector for each position (y_1, y_2,... , y_n), where y_i represents the representation of position i.

6.2.2 Multi-Head

This mechanism allows the encoder to weigh the importance of different elements in the input sequence. It calculates the score of each element in the attention sequence, taking it into account with the other elements. The attention score determines how much each element should contribute to the representation of the other elements. Multi-headed self-attention uses multiple attention heads to capture different types of dependencies and provide rich representations. The multi-head attention architecture is shown in Figure 6.2

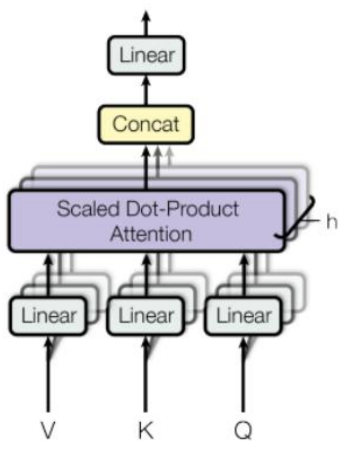


Figure 6.2 multi-head attention

6.2.3 Feedfoward network

After calculating self-attention, the encoder independently applies a feedforward neural network to each position in the sequence. The network processes the information from the attention step and applies non- linear transformations to

produce a more expressive representation. It helps to capture complex patterns and dependencies in sequences
The converter decoder gets the hidden representation from the encoder
And generate the output sequence. It is also composed of multiple layers, similar to Encoder. However, the decoder has an additional subcomponent called the encoder-decoder to focus on.

6.3 Decoder

6.3.1 Encoder-Decoder Attention

The decoder processes a hidden representation of the encoder to capture relevant information for generating the output sequence. It calculates the attention score between the decoder's current position and all positions in the encoder's hidden representation. This allows the decoder to focus on the relevant parts of the input sequence when generating the output.

Both encoders and decoders contain positional encoding, which is a mechanism for embedding positional information into an input sequence. This helps the model understand the order and structure of the inputs, whereas the transformer cannot have inherent position information, such as the cyclic model.

A crucial aspect of the transformer is the self-attention mechanism. It allows the model to pay attention to different parts of the input sequence, giving more weight to go to related elements and to go to unrelated elements. This mechanism enables Converters model complex relationships and dependencies in the data. The dependencies and relationships between the elements in them play a crucial role in acquiring dependencies

6.4 Mathematical Mechanism

6.4.1 K,Q,V

In the self-attention mechanism, each element in the input sequence is associated with three learned vectors: the key vector, the query vector, and the value vector. These vectors are derived from the input embeddings and serve different purposes.Matrix calculation is shown in Figure 6.3

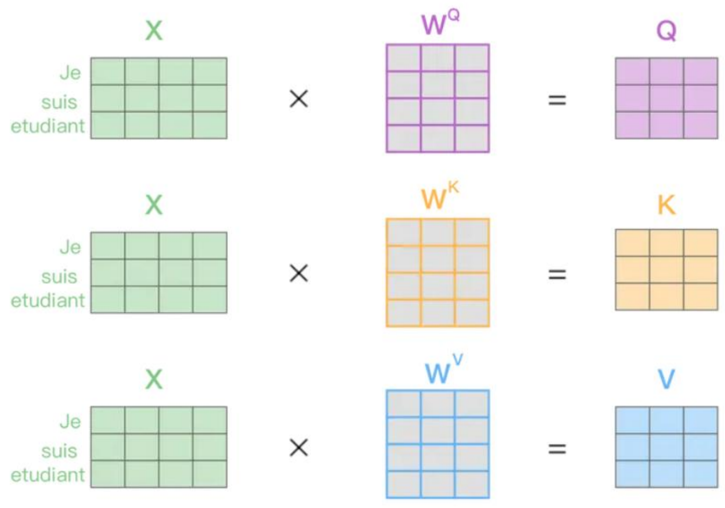


Figure 6.3:Q,K,V Matrix operations in self attention

6.4.2 Attention Scores

To calculate the attention score between elements, the self attention mechanism uses a similarity measure between the query vector of one element and the key vectors of all the other elements. This similarity measure is usually a dot product or a learned compatible function. The result is a set of attention scores that are used to quantify the relevance or importance of each element to the query element

6.4.3 Attention Weights and Context Vector

Attention score normalization uses the softmax function to obtain attention weights. These weights determine that each element contributes significantly to the representation of the query element. The higher the element attention weight, the greater the influence on the final representation. The attention weight is then used to calculate the weighted sum of the value vectors, producing the context vector

6.5 Transformer With Graph

In the context of graphical data, the researchers explored integrated transformers to enhance the representation and modeling capabilities of gnn. By combining the transformer's global context understanding with gnn's ability to capture local node information, these hybrid transformer-GNN models show better performance in tasks such as node classification, graph classification, and link prediction. The attention mechanism of the converter allows it to capture information about the relationships between nodes in the graph for a more efficient and expressive representation of the graph.

6.5.1 GAT

GAT is a neural network model for graphical data designed to learn node representations and the relationships between nodes. The core idea of GAT is to use attention mechanisms to capture complex relationships between nodes and dynamically assign weights according to the importance of the nodes. As with all attention mechanisms, the computation of GAT is divided into two steps

1 compute attention coefficient For vertex i, calculate its neighbors one by one

$$e_{i,j} = a([Wh_i][Wh_j]), j \epsilon N_i$$

With the correlation coefficient, the attention coefficient is normalized by softmax

$$\boldsymbol{\alpha}_{ij} = \frac{exp(LeakyReLU(e_{ij}))}{\sum_{k\boldsymbol{\epsilon}\boldsymbol{N_i}} exp(LeakyReLU(e_{ik}))}$$

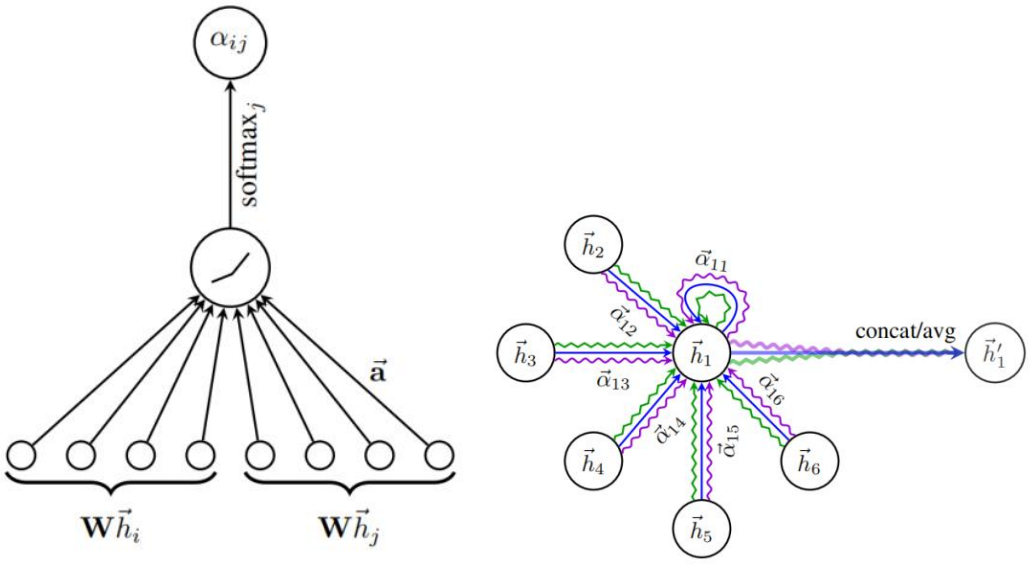
2 aggregate

According to the calculated attention coefficient, the feature weighted sum

$$h_i{}' = \sigma(\sum_{j\epsilon\boldsymbol{N_i}} \boldsymbol{\alpha}_{ij}\, Wh_j)$$

Multi-head version:

$$h_i{}' = \prod_{k=1}^{K} \sigma(\sum_{j\epsilon\boldsymbol{N_i}} \boldsymbol{\alpha}_{ij}\, Wh_j)$$



(a) **(b)**

Figure 6.1. **Left** : The attention mechanism a($\mathbf{W}\vec{h}_i$, $\mathbf{W}\vec{h}_j$) employed by the model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathrm{R}^{2F'}$, applying a LeakyReLU activation. **Right** : An illustration of multi-head attention(with K=3 heads)by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

Chapter 7

Heterogenesou Graph Transformer

7.1 Overall HGT Architecture

Figure 7.1 shows the overall architecture of Heterogeneous Graph.It Is a deep learning model for processing heterogeneous graph data, designed to learn the representation of nodes and relationships, as well as perform various tasks on heterogeneous graphs, such as node classification, link prediction, etc.Given a sampled heterogeneous subgraph, HGT extracts all linked node pairs, where the target node t is connected by the source node s via edge e. The goal of HGT is to aggregate information from the source node to get a contextual representation of the target node t. These processes can be broken down into three parts: heterogeneous mutual attention, heterogeneous messaging, and target-specific aggregation.[2]

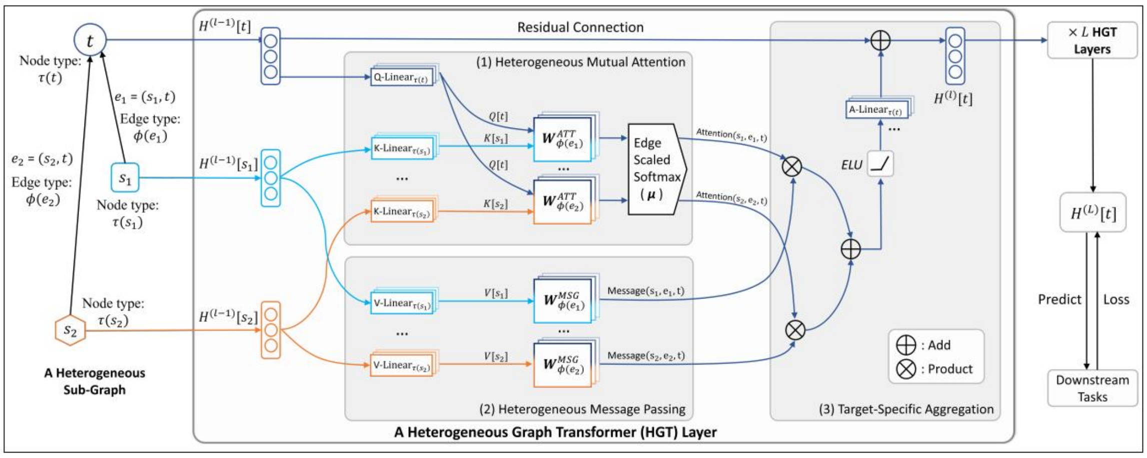


Figure7.1 The Overall Architecture of Heterogeneous Graph Transformer[2]: Given a sampled heterogeneous sub-graph with t as the target node, s1 & s2 as source nodes, the HGT model takes its edges e1 = (s1,t) & e2 = (s2,t) and their corresponding meta relations < τ (s1),ϕ(e1), τ (t) > & < τ (s2),ϕ(e2), τ (t) > as input to learn a contextualized representation H (L) for each node, which can be used for downstream tasks. Color decodes the node type. HGT includes three components: (1) meta relation-aware heterogeneous mutual attention, (2) heterogeneous message passing from source nodes, and (3) target-specific heterogeneous message aggregation.

7.2 Heterogeneous Mutual Attention

Heterogeneous Mutual Attention is a heterogeneous attention mechanism used to process heterogeneous graph data. It allows a model to consider the interaction between different types of nodes when learning node representation. Its core idea is to dynamically calculate the attention weights between different types of nodes using the attention mechanism. To guide the propagation and aggregation of information in the graph. It allows the model to adaptively adjust the node representation based on the similarity and importance between nodes. In Heterogeneous Mutual Attention, attention weight is obtained by calculating the similarity between different types of nodes. Specifically, for each pair of nodes, we calculate its attention score and obtain the attention weight through the normalization of softmax function. According to the attention weight, the features of adjacent nodes are weighted and summed to obtain the aggregated feature representation of each node. In this way, each node dynamically updates its own representation based on the importance of its neighbor nodes. In order to enhance the expressive ability of the model, Heterogeneous Mutual Attention mechanism is usually used. In each head, a different set of attention weights is learned, allowing the model to learn relationships between nodes in different attention subspaces.
General attention-based GNNs as follows

$$H^l[t] \leftarrow \underset{\forall s \epsilon N(t), \forall e \epsilon E(s,t)}{AGGREGATE} (ATTENTION(s,t) \cdot Message(s)) \tag{7.1}$$

For example, the Graph Attention Network (GAT) [5] adopts an additive mechanism as Attention, uses the same weight for calculating Message, and leverages the simple average followed by a nonlinear activation for the Aggregate step.Though GAT is effective to give high attention values to important nodes, it assumes thats and t have the same feature distributions by using one weight matrixW ., Which is usually incorrect for heterogeneous graphs, where each type of nodes can have its own feature distribution

In view of this limitation HGT design Heterogeneous Mutual Attention mechanism. Given a target node t, and all its neighbors $s\epsilon N(t)$, which might belong to different distributions, we want to calculate their mutual attention grounded by their meta relations, i.e., the $\langle\tau(s),\varphi(e),\tau(t)\rangle$ triplets

HGT map target node t into a Query vector, and source node s into a Key vector, and calculate their dot product as attention. The key difference
is that the vanilla Transformer uses a single set of projections for all words, while in our case each meta relation should have a distinct set of projection weights. To maximize parameter sharing while still maintaining the specific characteristics of different relations, we propose to parameterize the weight matrices of the interaction operators into a source node projection, an edge projection, and a target node projection. Specifically, we calculate the h-head attention for each edge e = (s,t)

$$Attention_{HGT}(s,e,t)=\underset{\forall s\epsilon N(t)}{Softmax}\left(\underset{i\in[1,h]}{\|}ATT-head^{i}(s,e,t)\right) \qquad (7.3)$$

$$\mathrm{ATT}-\mathrm{head}^{\mathrm{i}}(\mathrm{s},\mathrm{e},\mathrm{t})=(\mathrm{K}^{\mathrm{i}}(\mathrm{s})\mathrm{W}_{\varphi(\mathrm{e})}^{\mathrm{ATT}}\mathrm{Q}^{\mathrm{i}}(\mathrm{t})^{\mathrm{T}})\cdot\frac{\mu\langle\tau(s),\varphi(e),\tau(t)\rangle}{\sqrt{\mathrm{d}}}$$

$$K^{i}(s)=K-Linear_{\tau(s)}^{i}(H^{(l-1)}[s])$$

$$Q^{i}(t)=Q-Linear_{\tau(s)}^{i}(H^{(l-1)}[t])$$

The specific workflow is as follows: First, HGT maps the node features with dimension d of the source node to each head, and takes the node type of the source node as the index. A similar method is used to calculate Q. After that, we calculate the similarity of Q vector and K vector. Since there are many types of edges in heterogeneous graphs, HGT use a completely different WAT matrix for each type of edge to obtain different semantics for different types of edges (even if they have the same node type). In addition, a prior tensor μm is defined to define the meta relation triplet. Finally, the muti head results are stitched together

7.3 Heterogeneous Message Passing

While calculating mutual attention, we want to incorporate edge meta-relationships into the messaging process to ease the distribution differences between different types of nodes and edges. For a pair of nodes e = (s,t), HGT calculate its multi-head Message by:

$$Message_{HGT}(s,e,t) = \underset{i\in[1,h]}{\|} MSG - head^{i}(s,e,t) \qquad (7.4)$$

$$\text{MSG} - head^{i}(s,e,t) = M - Linear^{i}_{\boldsymbol{\tau(s)}}(H^{(l-1)})W^{MSG}_{\boldsymbol{\varphi(e)}}$$

To get the i-th message head $MSG - head^{i}(s,e,t)$, first,

Project the source node into the message vecor, followed by $W^{MSG}_{\boldsymbol{\varphi(e)}}$, to integrate the edge relationships. Finally, all h heads are concat

7.4 Target-specific Aggregation

With the heterogeneous multi-head attention and message calculated, we need to aggregate them from the source nodes to the target node,to average the corresponding messages from the source nodes and get the updated vector $\widetilde{H}^{l}[t]$,followed by residual connection [8] as:

$$\widetilde{H}^{l}[t] = A - Linear_{\boldsymbol{\tau(t)}}(\boldsymbol{\sigma}(\widetilde{H}^{l}[t])) + H^{(l-1)}[t] \qquad \mathbf{(7.5)}$$

In this way, we get the l-th HGT layer's output $H^{l}[t]$, for the target node t Due to the "small-world" property of real-world graphs, stacking the HGT blocks for L layers (L being a small value) can enable each node reaching a large proportion of nodes with different types and relations—in the full graph. That is, HGT generates a highly contextualized representation H^{L} for each node

7.5 Relative Temporal Encoding

The traditional way to incorporate temporal information is to construct a separate graph for each time slot. However, such a procedure may lose a large portion of structural dependencies across different time slots. Meanwhile, the representation of a node at time t might rely on edges that happen at other time slots. Therefore, a proper way to model dynamic graphs is to maintain all the edges happening at different times and allow nodes and edges with different timestamps to interact with each other. In light of this, HGT propose the Relative Temporal Encoding (RTE) mechanism(Figure 7.2) to model the dynamic dependencies in heterogeneous graphs. RTE is inspired by Transformer's positional encoding method [15, 21], which has been shown successful to capture the sequential dependencies of words in long texts.

Given a source node s and a destination node t, and their corresponding timestamps T(s) and T(t), HGT take the relative time gap ΔT (t, s) =T(t)–T (s) as an index to get a relative time code RTE (ΔT (t, s)). For the training set, not all time gaps will be included, so RTE should automatically generate time gaps that have not been seen before. Therefore, we adopt a fixed set of sinusoid functions as basis,with a tunable T-linear as RTE.Finally, the temporal encoding relative to the target node t is added to the source node s' representation as follows

$$\widetilde{H}^{(l-1)}[s] = H^{(l-1)}[s] + RTE(\Delta \boldsymbol{T}(\boldsymbol{t}, \boldsymbol{s}))$$

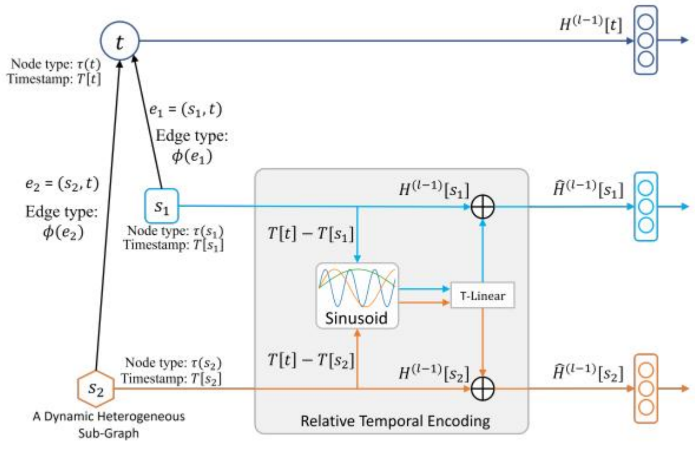


Figure 7.2 : Relative Temporal Encoding (RTE) to model graph dynamic. Nodes are associated with timestamps T (·). After the RTE process, the temporal augmented representations are fed to the HGT model.

Chapter 8

Experiments

8.1 General Structure

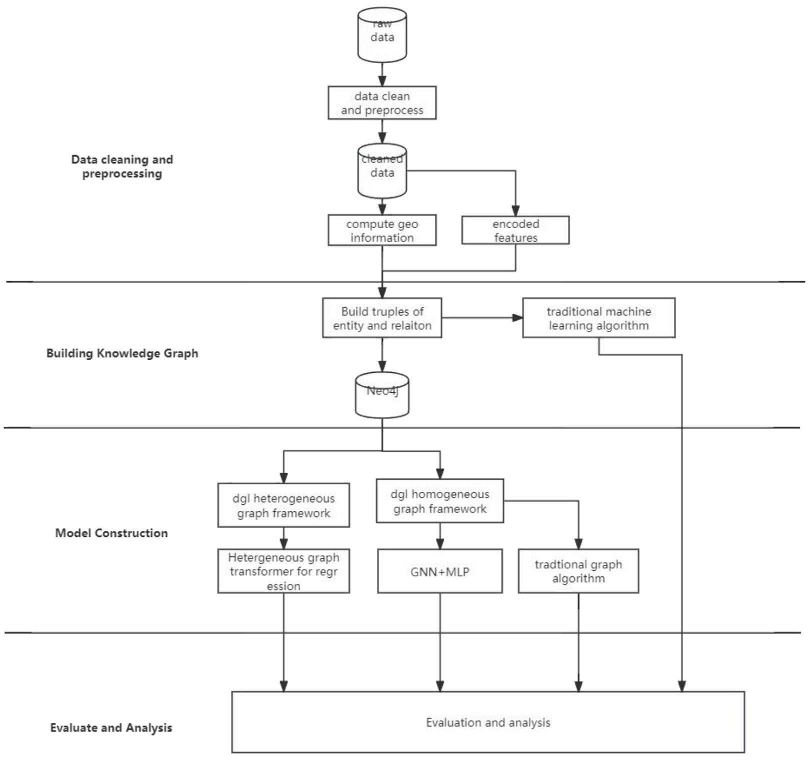

raw data
data clean and preprocess
cleaned data
Data cleaning and preprocessing
compute geo information
encoded features
Build truples of entity and relaiton
traditional machine learning algorithm
Building Knowledge Graph
Neo4j
dgl heterogeneous graph framework
dgl homogeneous graph framework
Model Construction
Hetergeneous graph transformer for regr ession
GNN+MLP
tradtional graph algorithm
Evaluate and Analysis
Evaluation and analysis

Figure8.1: General Schema

In my experiment, my overall process is shown in Figure 8.1. The overall process is divided into four stages: data cleaning and preprocessing, knowledge graph construction, model construction and final evaluation and analysis.This will be discussed in detail below

8.2 Dataset

Russia Real Estate 2021 : Real estate ads in Russia are published on the websites avito.ru, realty.yandex.ru, cian.ru, sob.ru, youla.ru, n1.ru, moyareklama.ru. These websites are the largest real estate websites in Russia, providing real estate information in various cities in Russia they provide a wealth of real estate information for users in Russia, and users can find housing information that suits their needs on

In the Russian real estate market, there are two main types, represented in the dataset as object types 0 - secondary real estate market; 2 - new construction. Each advertisement includes the geographical location and area of the property. Additionally, the dataset synchronizes with the federal public database through the Federal Tax Service (FIAS) to obtain house numbers.

8.3 Implementation details

8.3.1 Data cleaning and process

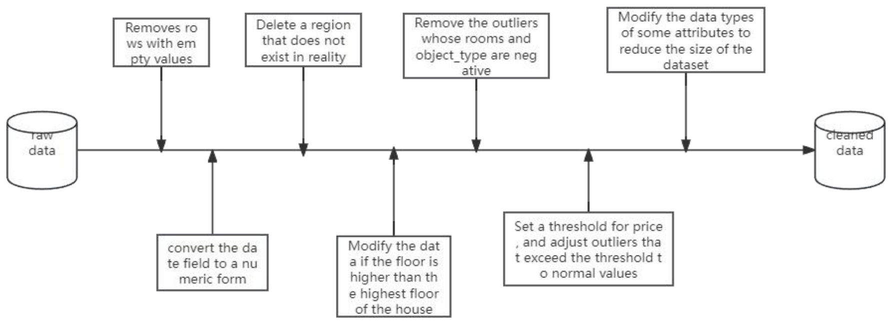

raw data
Removes ro ws with em pty values
convert the da te field to a nu meric form
Delete a region that does not exist in reality
Modify the dat a if the floor is higher than th e highest floor of the house
Remove the outliers whose rooms and object_type are neg ative
Set a threshold for price , and adjust outliers tha t exceed the threshold t o normal values
Modify the data types of some attributes to reduce the size of the dataset
cleaned data

Figure 8.5 Data Cleaning Process

This real estate dataset originates from multiple real estate advertising websites, including avito.ru, realty.yandex.ru, qian.ru, sob.ru, youla.ru, n1.ru, and moyareklama.ru. In Russia, some real estate agents may propose very high prices to purchase certain properties. Additionally, advertising websites allow authors to manually edit property information, leading to frequent occurrences of typographical errors and logical inconsistencies in property listings. Due to various reasons, data cleaning and processing becomes particularly crucial. In this experiment, we set thresholds for the price attribute and adjusted outliers to prevent them from affecting property price predictions. We also conducted logical checks on various attributes in the dataset. For example, if some properties have floor levels higher than the total number of floors in the building, we swapped the floor height and floor level. Furthermore, if the kitchen area exceeds the total area of the property, we swapped the kitchen area and total area, as it is likely due to user input errors. We also removed non-existent area codes and performed more refined cleaning operations on the dataset. Finally, we formatted dates for ease of processing and converted IDs such as street_id from floating-point to integer to reduce the dataset's storage space. Ultimately, we obtained a clean dataset.

8.3.2 Building Knowledge Graph

Real Estate Knowledge graph :

The structure of my knowledge graph is like the following Fagure 8.1

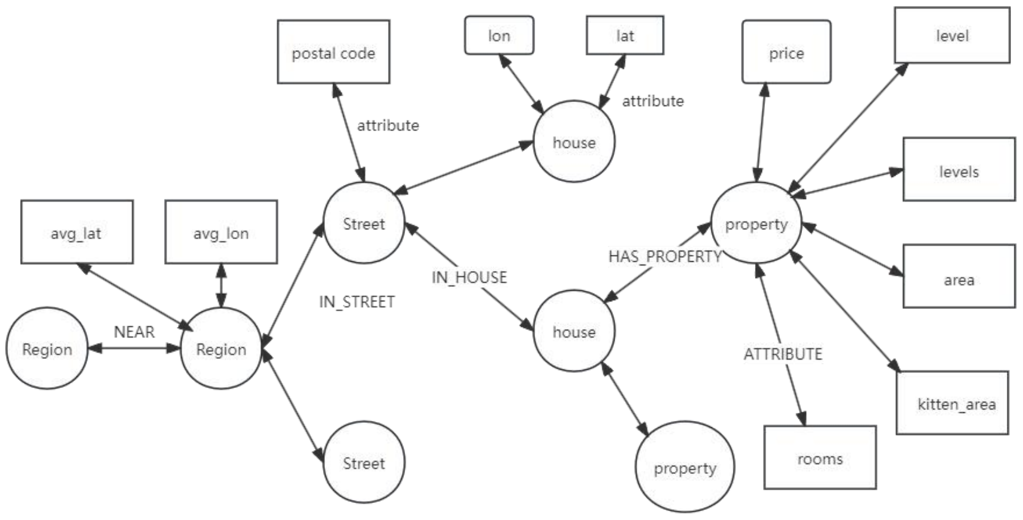


Figure 8.1. Real estate knowledge graph structure. The circles represent the entities of the real estate knowledge graph, and the boxes represent the attributes of each entity in the real estate knowledge graph

Entity : In the knowledge graph, entities are objects, concepts, or things in the real world that can be clearly identified, distinguished, and described. Entities usually represent nodes in the knowledge graph and are one of the basic components of the knowledge graph. In the knowledge graph I built, I defined the types of four entities, namely: Property, House, Street, Region.

Relation : In the knowledge graph, relations are used to describe the semantic associations or connections between entities. Relationships are the sides that connect entities in a knowledge graph, and they represent various relationships, connections, or interactions between entities.In my knowledge graph propery entity has a relation 'LOCATE_IN 'with house entity, house entity has relation 'LOCATE' with property entity, house entity has relation 'IN_STREET' and 'STREET' with street entity， street entity has relation 'IN_REGION' and 'REGION' with region entity. For each region entity, I use the average geographical location of each real estate in the region as the coordinates of the region, and calculate the distance between each region. If the distance between two region is less than the average, I add a "NEAR" relationship between these two regions

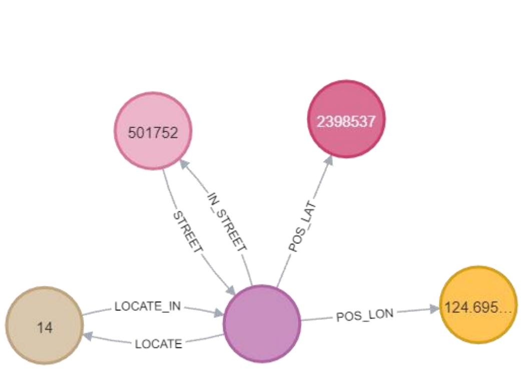
Attribute :In a knowledge graph, an attribute is a data item that describes the characteristics, attributes, or other relevant information of an entity. Each entity can have multiple properties, which can help us better understand and describe the meaning and characteristics of the entity in the knowledge graph.In my real estate knowledge graph,for each property entity we have 'price','level','levels',rooms','area,'kitchen_area', and for each house entity, we have 'geo_lat','geo_lon', and for each street entity, we have 'avg_geo_lat' and 'avg_geo_lon'

Knowledge Graph Model Represented by triples

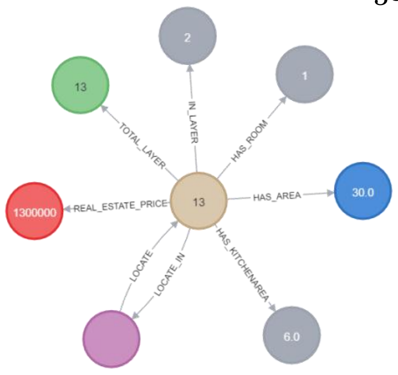
subject	predicate	object
house	IN_STREET	street
street	STREET	house
house	LOCATE	property
property	LOCATE_IN	house
region	NEAR	region
region	REGION	street
street	IN_REGION	Region
property	REAL_ESTATE_PRICE	Price
property	IN_LAYER	Level
property	TOTAL_LAYER	Levels
property	HAS_ROOM	Rooms
property	HAS_AREA	Area
property	HAS_KITCHEN_AREA	Kitchen_Area
house	POS_LAT	Lat
house	POS_LON	Lon
street	HAS_POSTCODE	postcode

Table 1: triples of real estate knowledge graph

The knowledge graph is saved in triples and is later used to build the dgl dataset, and The structural features of the data can be obtained from the knowledge graph



(a)



(b)

Figure 8.2. (a)a house entity and it's attributes and the property entitys linked by it ; (b)a property entity and it's attributes

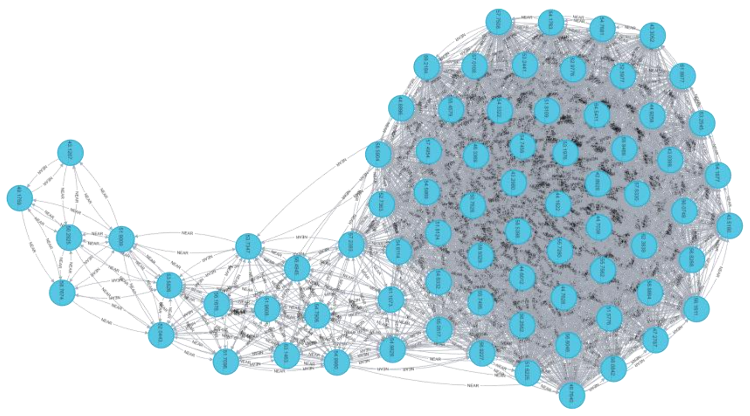


Figure 8.3. all the region entitys in real estate knowledge graph and 'NEAR' relations between them

The knowledge graph constructed in this paper is stored in Neo4j, a popular graph database management system, which is specially used to store, query and analyze graph data. Neo4j uses graphical data model to represent data and supports the storage and query of nodes, relationships and attributes. neo4j can use Cypher language to query, flexibility and expression ability. Due to the characteristics of its graphical data model, neo4j has high flexibility and expression ability. It is also able to naturally represent and process complex entity relationship structures, with community support and an ecosystem, as well as a separate visual interface

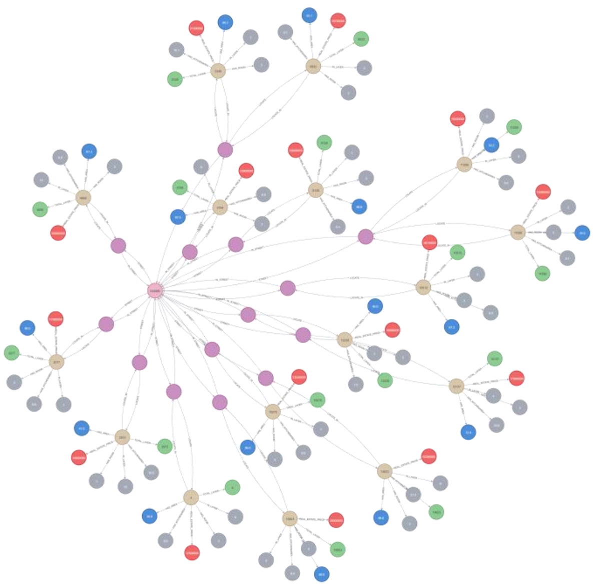


Figure 8.4. part of my real estate knowledge ,one street entity with all the entity and attributes associated with it

Dataset	#Property	#House	#Street	#Region
Russia 2012 Real Estate Knowledge graph	19057	14785	8446	76

Table2: Russia real estate 2021 knowledge graph entity counts

Dataset	#IN_STREET	#STREET	#LOCATE_IN	#LOCATE	#NEAR	#REGION	#IN_REGION
Russia 2012 Real Estate Knowledge graph	14785	14785	19057	19057	2898	8446	8446

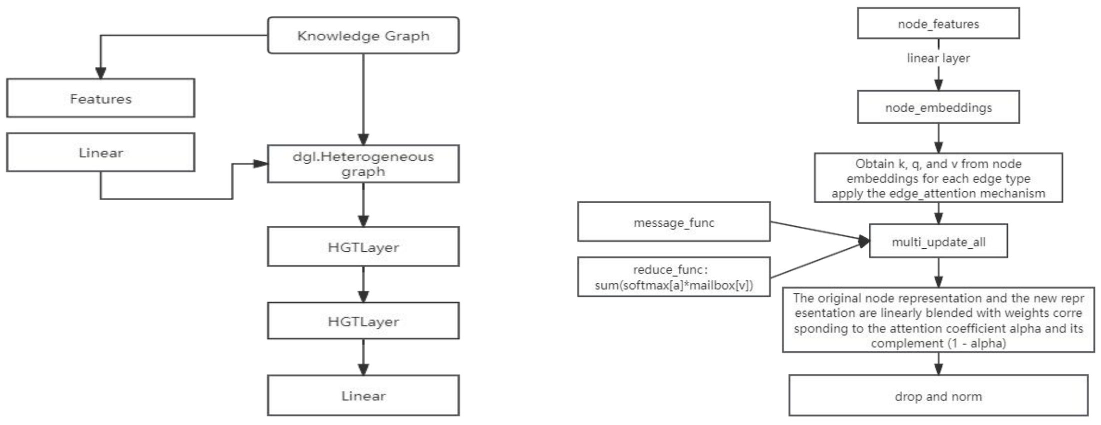
Table3: Russia real estate 2021 knowledge graph relation counts

Dataset	#Has_area	#Has_kitchen_area	#Has_room	#Pos_lat	#Pos_lon	#Real_estate_price	#Total_layer	#Has_postcode
Russia 2012 Real Estate Knowledge grap	19057	19057	19057	14785	14785	19057	19057	14785

Table4: Russia real estate 2021 knowledge graph attribute counts

8.3.3 Building Models

I built four graph neural network models. The first one is deepwalk+ MLP for prediction, the second one is GCN model, and the third one is GAT model. The fourth is HGT model using self-attention mechanism. I used these three models to model and process the above knowledge graph data. And then get the prediction of real estate price .In addition, I used some machine learning models for comparison to see how HGT improves predictive performance. Following is my implementation of my HGT model.


Knowledge Graph
Features
Linear
dgl.Heterogeneous graph
HGTLayer
HGTLayer
Linear
node_features
linear layer
node_embeddings
Obtain k, q, and v from node embeddings for each edge type apply the edge_attention mechanism
message_func
reduce_func: sum(softmax[a]*mailbox[v])
multi_update_all
The original node representation and the new repr esentation are linearly blended with weights corre sponding to the attention coefficient alpha and its complement (1 - alpha)
drop and norm

(a) (b)

Figure 8.3 : (a) Figure is the overall architecture of the HGT model, including two HGTLayers and the processing of features in the heterogeneous graph; (b) Figure is the internal implementation logic of HGTLayer

All my experiments are based on python3.11.5,cuda version of torch 2.1.2,cuda version of dgl 2.0.0+cu121, geopy package to calculate distance by latitude and longitude, py2neo package to implement interaction with Neo4 Figure j database. For the MLP model, I only used two layers of the middle layer with 128 hidden layers, and a linear layer to do the output of regression, training the batch size of 32 and epoch of 200. For the DeepWalk model, I transformed the heterogeneous graph into the same graph. Then, deepwalk with epoch 3 is carried out on the graph to obtain the embedding of the graph. The embedding feature and node feature are fused and put into MLP for training to obtain the prediction result. For GAT, I design a two layer GAT with num_head = 4. For GCN, I also converted the heterogeneous graph into the same graph, and I used two layers of graphconv as my model, and relu was selected as the activation function to solve the problem of gradient disappearance. For HGT, I raised the dimension of all types of node attributes in the heterogeneous graph to 128 through linear layer. For each node type in the heterogeneous graph, k,q and v matrices were set respectively. After linear transformation, the input features of the model were passed into HGTLayer, and after multi-head attention calculation, skip connection and residual link, norm normalization was finally used. In the HGT layer, messages from neighbor nodes are calculated according to edge characteristics in message_func and sent to the corresponding target node. In the neighbor node, reduce_func is used to aggregate the neighbor nodes. softmax is used to normalize neighbor nodes, and then the messages received by each node are multiplied by the corresponding weight. Finally, the message received by all nodes is summed to get a new node representation, after two HGTLayer, after an MLP, the final prediction result is obtained

Meanwhile, for other experiments, I designed catboost with depth 3 for the machine learning model, using the original data set, LSTM with hidden layer of 128 neurons, using the original data set.

8.4 Evaluation metrics

Mean Squared Error (MSE) is a common metric used to measure the difference between a model's predictions and the true label, and is often used in regression tasks. The calculation is as follows

Mean Square Error (MSE):

$$\mathrm{MSE} = \frac{1}{n}\sum_{1}^{n} |y_i - x_i|$$

Where:
n is the total number of data points
$y_i$is the predicted value for data point i
$x_i$is the observed value for data point i

The smaller the value of MSE, the better, because this means that the smaller the difference between the model's predictions and the real label, the better the model's predictions. Among the commonly used evaluation indexes of regression tasks, MSE is one of the important indexes.In this experiment, MSE was used to evaluate the output of the model on the test set as the basis for measuring the quality of the model.

Chapter 9

Result and Disscussion

epoch	MLP	LSTM	CatbBoost	DeepWalk	GCN	GAT	HGT
40	46112.4648	38.1557	39.7656	56.6198	1317994.3750	415168.5625	29.9561
80	408.9117	35.7952	31.3040	36.8715	28593.0449	96222.1484	25.9842
120	35.9775	35.5150	28.398	32.9485	460.2014	415.3784	24.1938
160	33.9191	34.4391	26.4600	32.1692	34.8478	72.2393	22.9170
200	33.9072	34.5967	24.9310	32.0144	27.3689	23.4113	22.7329

Table5: learning loss in the training ,using different models.LSTM is based on data set , DeepWalk, GCN, HGT are based on my real estate knowledge graph

	MLP	LSTM	CatBoost	DeepWalk	GCN	GAT	HGT
MSE	33.9072	34.5891	24.9319	33.2619	25.7054	22.4687	22.3666

Table6: The mean square error of the model on the test set with the epoch is 200

Table 5 shows the training loss obtained by different epoch during the training process, and Table 6 shows the mean square error of each model on the test set when the epoch of the model is 200. It can be found from the experiment that HGT has obvious advantages for graph mining of heterogeneous graphs, and it is inappropriate to treat data with heterogeneous graph attributes as the same graph, because heterogeneous graphs contain multiple relationships between nodes, and weight matrices should be used separately instead of sharing the same weight matrix. Similarly, we can find that there are also effective models in traditional machine learning models, which may be caused by the fact that graph neural network pays too much attention to the structural features of data and pays too much attention to the statistical features and sequence features of data. Similarly, due to the existence of a large number of false data in the data set, the model is also greatly affected.

Chapter 10

Conclusions

In conclusion, this study constructed a real estate domain knowledge Graph using a real estate value prediction task based on knowledge graph, and discussed the performance of HGT (Heterogeneous Graph Transformer) in processing real estate data and knowledge graph. The experimental results show that the HGT model has excellent performance for multi-node and multilateral heterogeneous graph data. By comparing it with traditional graph neural networks (GCN) and traditional graph algorithms, we find that HGT achieves the best results in the real estate value prediction task with a test set mean square error (MSE) of 22.7329. This shows that HGT model has significant advantages in processing heterogeneous graphical data, and has important value and potential in real estate applications.

Further analysis shows that the HGT model is not only robust and universal in performance, but also can achieve consistent excellent performance under different experimental Settings. This makes HGT an important tool in real estate research and practice, which can be widely used in real estate value assessment, home sales forecasting, rent pricing and other tasks. Using HGT's powerful modeling capabilities and understanding of complex relationships, we can more accurately predict the value of real estate and provide reliable support for relevant decisions.

However, while HGT has achieved remarkable results in the real estate value forecasting task, there are still some potential challenges and room for improvement. The problem that a large number of false and wrong data in real estate data affect the forecast results remains to be solved. At the same time, future research can further explore the performance of HGT when processing real estate datasets of different sizes and complexity, optimize model design and training strategies, improve model performance and efficiency, and combine HGT with traditional machine learning algorithms to improve the expression of its features.

Bibliography

[1] Brin, S., & Page, L. (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine. In Proceedings of the 7th International World Wide Web Conference (WWW 1998).

[2] Qu, M., Shen, Y., Gao, C., Zhang, Z., & Han, J. (2020). Heterogeneous Graph Transformer. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining

[3] Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). DeepWalk: Online Learning of Social Representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

[4] Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. V. D., Titov, I., & Welling, M. (2018). Modeling Relational Data with Graph Convolutional Networks. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence

[5] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph Attention Networks. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence

[6] Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., & Eliassi-Rad, T. (2008). Collective classification in network data.

[7] Yang, Y., Yoo, J., Kim, J., & Chawla, N. V. (2016). Node classification in heterogeneous information networks with unseen edges. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

[8] Wang, D., Cui, P., Zhu, W., & Yang, S. (2019). Heterogeneous graph attention network. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 2022-2030).

[9] Dong, Y., Chawla, N. V., & Swami, A. (2017). metapath2vec: Scalable representation learning for heterogeneous networks. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

[10] Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations (ICLR).

[11] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V.Chawla. 2019. Heterogeneous Graph Neural Network

[12] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S.Yu. 2019. Heterogeneous Graph Attention Network. In KDD ' 19. 2022–2032.

[13] Li, Z., Yang, R., Zhou, B., Hu, X., & Deng, X. (2021). Graph Transformer Networks. In Proceedings of the AAAI Conference on Artificial Intelligence

[14] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alexander A Alemi. Watch your step: Learning node embeddings via graph attention. Advances in neural information processing systems, 31, 2018.

[15] Hao Wang, Fanjiang Xu, Xiaohui Hu, and Yukio Ohsawa. Ideagraph: a graphbased algorithm of mining latent information for human cognition. In 2013 IEEE International Conference on Systems, Man, and Cybernetics, pages 952–957. IEEE, 2013.

[16] Anastasios Drosou, Ilias Kalamaras, Stavros Papadopoulos, and Dimitrios Tzovaras. An enhanced graph analytics platform (gap) providing insight in big network data. Journal of Innovation in Digital Ecosystems, 3(2):83–97, 2016.

[17] Ronky Francis Doh, Conghua Zhou, John Kingsley Arthur, Isaac Tawiah, and Benjamin Doh. A systematic review of deep knowledge graph-based recommender systems, with focus on explainable embeddings. Data

[18] Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H Pham, Jafar M Al-Kofahi, and Tien N Nguyen. Graph-based mining of multiple object usage patterns. In Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering, pages 383–392, 2009.

[19] Frauke Heinzle, Karl-Heinrich Anders, and Monika Sester. Graph based ap proaches for recognition of patterns and implicit information in road networks. In Proceedings of the 22nd international cartographic conference, pages 11–16. ICA Washington, DC, 2005

[20] Brian Kulis, Sugato Basu, Inderjit Dhillon, and Raymond Mooney. Semisupervised graph clustering: a kernel approach. In Proceedings of the 22nd international conference on machine learning, pages 457–464, 2005.

[21] Laurent Galluccio, Olivier Michel, Pierre Comon, and Alfred O Hero III. Graph based k-means clustering. Signal Processing, 92(9):1970–1984, 2012.

[22] Gui Yang, Wenping Zheng, Chenhao Che, and Wenjian Wang. Graph-based label propagation algorithm for community detection. International Journal of Machine Learning and Cybernetics, 11:1319–1329, 2020.

[23] Gema Bello-Orgaz and David Camacho. Evolutionary clustering algorithm for community detection using graph-based information. In 2014 IEEE

congress on evolutionary computation (CEC), pages 930–937. IEEE, 2014.

[24] Khushboo S Thakkar, Rajiv V Dharaskar, and MB Chandak. Graph-based algorithms for text summarization. In 2010 3rd International Conference on Emerging Trends in Engineering and Technology, pages 516 – 519. IEEE, 2010

[25] Guang Tan, Marin Bertier, and A-M Kermarrec. Visibility-graph-based shortestpath geographic routing in sensor networks. In IEEE INFOCOM 2009, pages 1719–1727. IEEE, 2009

[26] Zan Huang, Wingyan Chung, Thian-Huat Ong, and Hsinchun Chen. A graph based recommender system for digital library. In Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries, pages 65–73, 2002.

[27] Zahra Zamanzadeh Darban and Mohammad Hadi Valipour. Ghrs: Graph-based hybrid recommendation system with application to movie recommendation. Expert Systems with Applications, 200:116850, 2022

[28] Daniel Spielman. Spectral graph theory. Combinatorial scientific computing, 18:18, 2012.

[29] Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. Advances in Neural Information Processing Systems, 35:14582 – 14595, 2022.

[30] Chen Gao, Xiang Wang, Xiangnan He, and Yong Li. Graph neural networks for recommender system. In Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, pages 1623–1625, 2022

[31] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. Journal of Machine Learning Research, 9(86):2579–2605, 2008.

[32] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. AI magazine, 29(3):93–93, 2008.

[33] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). "Attention is All You Need". Advances in Neural Information Processing Systems, 30.

[34] BIZER C, LEHMANN J, KOBILAROV G, et al. Dbpedia-a crystallization point for the web of data[J]. Journal of web semantics, 2009, 7(3): 154-165.