# Query-Driven Graph-based User Recommender System

by

Yansong Li

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Master of Computer Science
in
Applied Artificial Intelligence

© Yansong Li, Ottawa, Canada, 2022

## Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University of Ottawa regulations concerning plagiarism, including those regarding consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

**Abstract**

Current Social Networking Systems (SNS) such as YouTube are creator-driven systems in which creators create content and users search among available content to find what they want. However, queries from user can be time-sensitive, such as some real-time hot topics, which are difficult to obtain at the very moment due to their timeliness and dynamically changing nature. To address this situation, we quest if the system can directly let a user input a query, match the most relevant users (receivers) based on the query and let the receivers decide whether to respond with the very content. In this way, the user can obtain the most relevant data through highly relevant receivers while reducing the reliance on the system's existing data in the recommendation process as an alternative, a new query-driven SNS paradigm.

The main objective is to target the most relevant receivers based on a query. In this case, we propose that by allowing users to provide their very moment ideas as queries, the system searches and ranks well-targeted users based on the semantic content of the query and existing user features. However, the user's feature might be incomplete or missing. To alleviate this issue, we propose a novel two-stage query-driven graph-based user recommender system (QDG) that supports query-to-user matching with dynamic update capabilities. In the first stage, we encode the query and item descriptions into attribute features and perform a similarity search to target the Top-N candidate items. In the second stage, we propose a temporal-based graph neural network (t-GNN), which combines the inductive learning-based GNN with the self-attention-based temporal analysis module to predict the most relevant user-item interaction by simultaneously extracting the existing Spatio-temporal features, where spatial feature represents user's relationship with items and temporal feature represents user's behaviour information. We conducted recommendation simulations on six million users and 150,000 merchants on North America YELP data. Experiments show that the QDG system can accurately target strongly relevant users in the North American population based on the query. To the best of our knowledge, we are the first to propose query-driven SNS and demonstrate its effectiveness in a million-scale Yelp dataset.

**Acknowledgements**

First of all, I would like to send my sincerest thanks to my supervisor, WonSook Lee for inviting me to join her lab and organizing weekly seminars to practice our literature reading, thinking and presentation skills. Professor Lee has always been patient in guiding my research content and providing key suggestions and corrections in the weekly meetings. I have gained a lot of practical and lifelong skills and knowledge from being in Professor Lee's group.

I am also grateful to the University of Ottawa for providing me with teaching resources and a good environment. During my three years at the University of Ottawa, I have not only gained academic progress, but also gained work experience and skills in the Co-op program. The mentors and friendships I have made during my studies are also a lifelong asset to me.

Finally I would also like to thank my family and friends for their unconditional and strong support and warmth during my studies, and with their encouragement I have been able to come this far and gain the benefits of working and further studying computer science.

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

**BERT** Bidirectional Encoder Representations from Transformers 21

**CPU** Central processing unit 26

**CV** Computer vision 20

**DistilBERT** A distilled version of BERT 6

**GNN** Graph neural network 4

**GPT** Generative Pre-trained Transformer 21

**GPU** Graphics processing unit 24

**IGMC** Inductive graph matrix completion 52

**LSTM** Long Short-term Memory 6

**NLP** Natural language processing 20

**PinSage** Random-walk-based GCN 52

**RGCN** Relational graph convolutional network 41

**sBERT** Sentence-BERT 24

**SNLI** Stanford Natural Language Inference x, 24

**sRGCNN** Recurrent Multi-Graph Neural Networks 52

# Chapter 1

# Introduction

With the development of user service-based systems, users implicitly or explicitly provide personal information to the system through their daily use of system services. Likewise, the system would utilize users' information to analyze user preferences and provide personal and accurate services. This so-called "use it first, recommend it later" paradigm has achieved great success and economic benefits on social media such as YouTube, Facebook and e-commerce platforms like Amazon and Alibaba [9]. This recommendation paradigm aims to acquire existing information to establish personal preference distribution (Such as user-item or user-user interaction matrix or feature representations) [8, 11, 19, 44] and produce corresponding recommendation algorithms (Such as collaborative-filtering or content-based approach) [2, 14, 25, 32, 35].

However, if users propose some novel requirements, the system will require a certain amount of time to collect enough information to initialize the corresponding recommendation distribution. Lack of information might lead to sub-optimal recommendations [16, 20]. For example, a visitor from abroad wants to know the most authentic cheeseburger in the current seasonal food fair in town, but the fact is that there is very little information on existing travel platforms or social media. By visiting the platform, users may end up with only the best burger joints from previous years.

To address this issue, we propose a query-to-user pipeline as a new alternative to the recommendation paradigm. Instead of entirely relying on existing information to provide recommendations, users themselves could be a solid recommendation source. We first redefine a user's role on a query, which is either a requester or a receiver. As shown in Figure 1.1, a requester generally acts as an initiator of a query, and a query receiver will choose whether to provide query-related information to the requester.

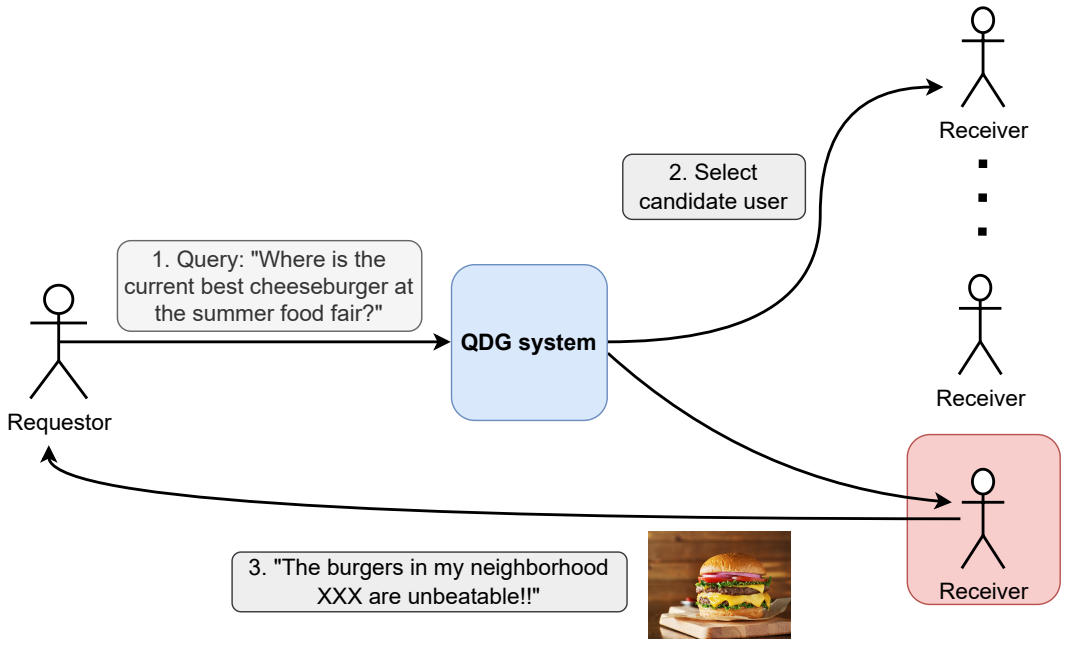



Figure 1.1: An illustration of the QDG work mechanism. The requester would send a query to QDG, and QDG would target the most relevant receiver based on the query and then let the receiver decide whether to respond to the requester.

## 1.1 Challenge

Nevertheless, our system faces the following challenges in the implementation process; 1. how to target the most relevant receivers to the query with limited user information, 2. how to maintain the quality of the recommendations while the user information is dynamically changing. (Note: User information includes both user's personal information and interaction information with new items or existing but unseen items.) For the receiver targeting challenge, one of the intuitive approaches is to directly transform the semantic information of the query into sentence representation and match the most similar receiver through content-based algorithms [25, 32] by calculating the similarity between the sentence representation and the receiver representations [29]. However, user feature might be incomplete or missing in many cases [21, 22, 24], especially for those newly registered users or users who deliberately fill in the wrong information for privacy preserving [40]. Therefore it is not optimal to directly match queries to users as one-stage recommendation

which is shown in figure 1.2. Without relying on subjective information filled in by the user, items with which the user has an objective interaction relationship, and the temporal behaviour sequence are the most comprehensive way to describe user preferences and characteristics [32,41,43]. Most item features are more authoritative and reliable than user features due to platform promotion and relatively strict policy [30].

## 1.2 Our System Structure

Consequently, as shown in Figure 1.2, we present a novel two-stage query-driven graph based recommender system that splits the query-to-user pipeline into query-to-item and item-to-user.

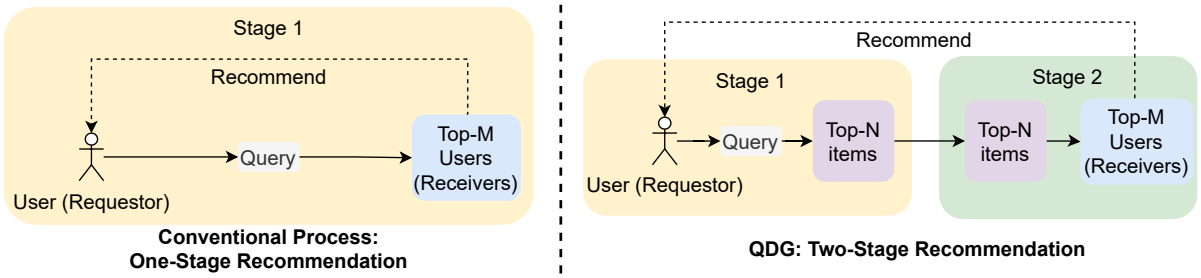



Figure 1.2: One-Stage Recommendation VS Two-Stage Recommendation. Our proposal is the two-stage QDG recommender system, in which the system utilizes items as a medium to target the Top-M most relevant users corresponding to the query.

### 1.2.1 Stage 1: Query-to-Item

In the first query-to-item matching stage,we implement the Faiss algorithm [6] to achieve Top-N matching items based on Faiss's nearest-neighbour distance between query's sentence representation and item features extracted by lightweight pre-trained DistilBERT [31]. It is also effective to achieve fast matching among million-scale data.

### 1.2.2 Stage 2: Item-to-User

In the second item-to-user stage, we need to address the dynamically changing user information challenge. However, most of the common recommendation algorithms are based

on matrix completion, which analyzes user preferences through matrix factorization based on existing user and item latent representations and interaction matrix [1, 3, 19, 35]. But matrix factorization is inherently transductive, and the model cannot be generalized to unseen data [47]. Therefore, we have adopted an inductive-learning strategy to adapt to the dynamically changing data. Many inductive learning-based Graph Neural Network (Graph neural network (GNN)) enables prediction of unseen data. Even when the user is anonymous, they can still predict relationships for new arrivals based on the user's existing relationships [13, 47]. But despite these models' apparent effectiveness, there remains a major drawback that these models do not utilize the user behaviour sequence, which might result in not capturing the user's most recent preference.

**t-GNN** To accurately locate the user preference distribution varying tendency and extract user behaviour information, we propose Temporal-based GNN (t-GNN), which combines the inductive learning-based GNN with the self-attention-based temporal analysis module to predict the most relevant user-item interaction by simultaneously extracting the existing Spatio-temporal features. Note: As shown in Figure 1.3 the spatio features here refer to the user-item relationship graph since the graph is a class of spatio structures. The temporal features refer to the sequence of the user behaviours in the graph.

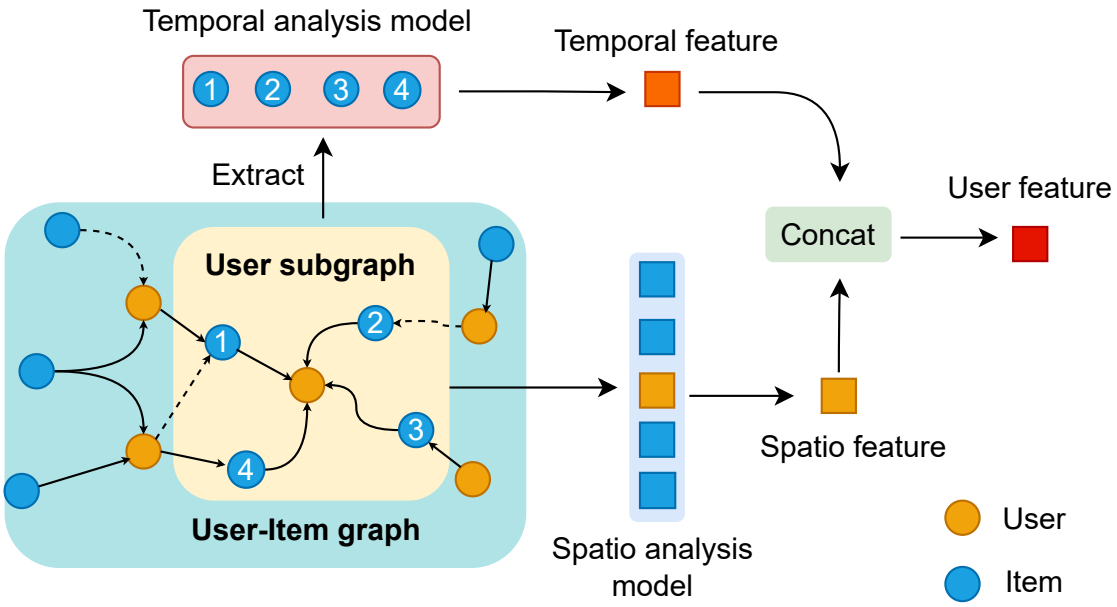



Figure 1.3: Overview of t-GNN structure, QDG utilize t-GNN in Item-to-User process.

By implementing the QDG recommender system, we can make a requester's query accurately match the most relevant receivers by users' spatio-temporal features in the absence of platform information or users' information. In that case, a user can be either a requester to get help or a receiver to be a source of recommendations and build connections with others. For the utility experiments, we investigate the QDG performance on million-scale YELP dataset, the final simulation test shows that our QDG can accurately match users within the dataset domain based on their queries on several set of simulations, and in the item-to-user module test, extensive empirical evaluation shows that the t-GNN with the combined Spatio-temporal module achieves the best AUC and accuracy score in user-item relationship prediction compared to the previous inductive-learning based GNN model.

## 1.3 Dataset Description

In our experiment, we selected data based on the following conditions: first, it should contain three essential elements: users, items, and their relationships. Then users and items need to have a certain size of features to satisfy the requirement of constructing a highly robust pre-training model. Finally, the user-centred association graph should have temporal attributes on its interaction with the items, which is necessary to construct a parsable temporal feature model. Also it needs to be publically available so that comparison can be done with others. We select the YELP open-source dataset as our experimental

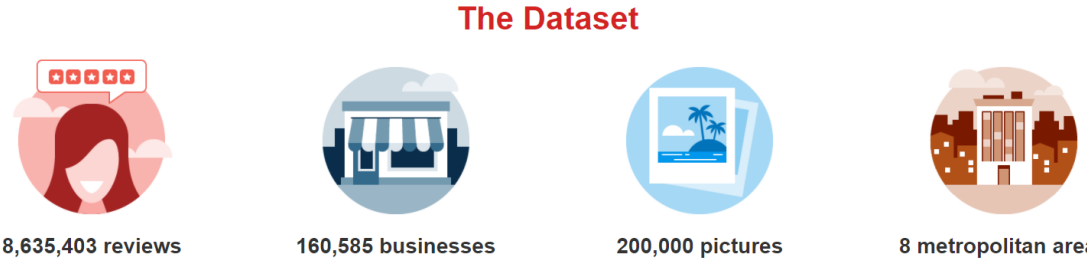



Figure 1.4: YELP dataset: 1,162,119 tips by 2,189,457 users, Over 1.2 million business attributes like hours, parking, availability, and ambience, aggregated check-ins over time for each of the 138,876 businesses

data. With unmatched native business data, photos and review content, YELP provides a one-stop native platform for shoppers to find, connect and interact with native businesses of all sizes by creating it straightforward to request a quote, be a part of a waitlist, and build a reservation, appointment or purchase.

As shown in the Figure 1.4, the YELP dataset contains interactions between 160,585 stores and 2189,457 users in 8 major metropolitan areas, with 8635,403 interactions.

## 1.4 Contribution

Our contributions can be summarized as follows:

- We propose a new query-driven SNS recommendation paradigm in which users can either act as requesters, initiate queries to seed help from the matched users, or provide help as receivers.
- Our two-stage pipeline in the QDG enables accurate user targeting without relying on user data by adopting items as implicit user descriptors.
- Our t-GNN model address a user's dynamic changing information challenge by extracting the user's spatio and temporal features and establish the user's real-time preference.

## 1.5 Thesis Organization

In this thesis, chapter 2 will provide a detailed and comprehensive description of the various models and algorithms used in this system. First, we will discuss the reasons and details of using A distilled version of BERT (DistilBERT) as a Sentence-encoder, along with the advantages and reasons of transformer-related models in extracting sentence embedding vectors. After that, we will discuss in detail the advantages of using FAISS as a neighbour query algorithm and the reasons why it can still maintain high performance under large-scale data. We will then detail the main reasons for using inductive-learning-related models, in which we will compare the details of various transductive-learning and inductive-learning related models and further explain the advantages of using inductive-learning for large-scale, dynamically growing data. Finally, we will compare the details of the correlation models used to analyze time-series correlated events, such as RNN, Long Short-term Memory (LSTM) and multi-head attention. The comparison will explain in detail the advantages and disadvantages of each technique.

Subsequent sections 3 will compare and validate our various experimental combinations and effects. Also included are our methods for optimizing model performance. The final

results of the experiments confirm that by combining GNN and attention-based models, we can further improve the predictive power of the models for user-item interactions. Section 4 would demonstrate all experiments and our results.

Finally, Section 5 summarizes and categorizes all our current methods, summarizes the strengths and weaknesses of our models and methods, and suggests possible improvements.

# Chapter 2

# Literature Review

## 2.1 Recommender System

User recommendations are made using a recommender system to help guide them toward relevant items. Methods for achieving this goal fall into two categories: collaborative filtering methods and content-based methods. In order to get into more detail about particular algorithms, let's first talk about these two main paradigms.

### 2.1.1 Collaborative Filtering Methods

Collaboration methods for recommender systems are based exclusively on past interactions between users and items to generate new recommendations. The interaction matrix between the user and the item is stored in the so-called "user-item interaction matrix." Accordingly, these previous user-item interactions provide sufficient information for detecting similar users or similar items and making predictions based on these estimated proximities.

It is generally recognized that collaborative filtering algorithms can be divided into memory-based and model-based approaches. Memory-based approaches operate directly on the values of recorded interactions and assume no learn-able model participated. They are essentially based on nearest neighbour search (e.g., finding the nearest users from a user of interest and suggesting the most popular items). The model-based approach assumes that the user-item interactions are modelled by a "generative" model, which is used to make predictions.

Collaborative filtering suffers from the "cold start problem" since it only utilizes past interactions to make recommendations. It is impossible to recommend anything to newly registered users, nor to recommend a new item to any users, and many users and items have too few interactions to be considered for effective recommendation. There are several ways to address this drawback: either by recommending random items to new users or new items to new users (random strategy), or by recommending most popular items to most active users (maximum expectation strategy), or by recommending a set of different items to new users or a new item to a set of different users (exploratory strategy), or, finally, by using a non-collaborative strategy for the early lifecycle of both the user and the item.

We shall primarily present three classical collaborative filtering approaches in the following sections: two memory-based methods (user-users and item-items) and one model-based approach (matrix factorization).

### 2.1.2 Memory based collaborative approaches

User-user and item-item approaches share the characteristics that they rely solely on information from the user-item interactions matrix and do not assume any model to produce new recommendations.

**User-user:** The user-user method identifies the users with the most similar "interactions profiles" (nearest neighbours) and recommends items that are most popular in these neighbourhoods (and that are "new" to our user). As this approach represents users using their interactions with items and evaluates their distances to each other, it is said to be "user-centred." Consider the case where you wish to make a recommendation for a particular user. The interaction matrix would be composed of interactions between users and items. Afterwards, we can compute some kind of "similarity" based on our user of interest with other users. It would be considered similar if two users had similar interactions with the same set of items. After calculating the similarity between all users, we can implement the k-nearest-neighbor algorithm to identify the most similar users and then suggest the most popular items among them (for items that the target user has not yet interacted with).

**Item-item:** The item-item collaborative filtering is one type of recommendation method that searches for similar items based on the items that the users have already liked, clicked on, or interacted with positively in the past. Created by Amazon in 1998, it plays a vital role in the company's success. Item-based collaborative filtering suggests items based on what the user has consumed previously. After identifying which items have been consumed by the user, the system finds similar items and presents them to the user as recommendations. We will analyze an example to understand better how it works. For example, a user named

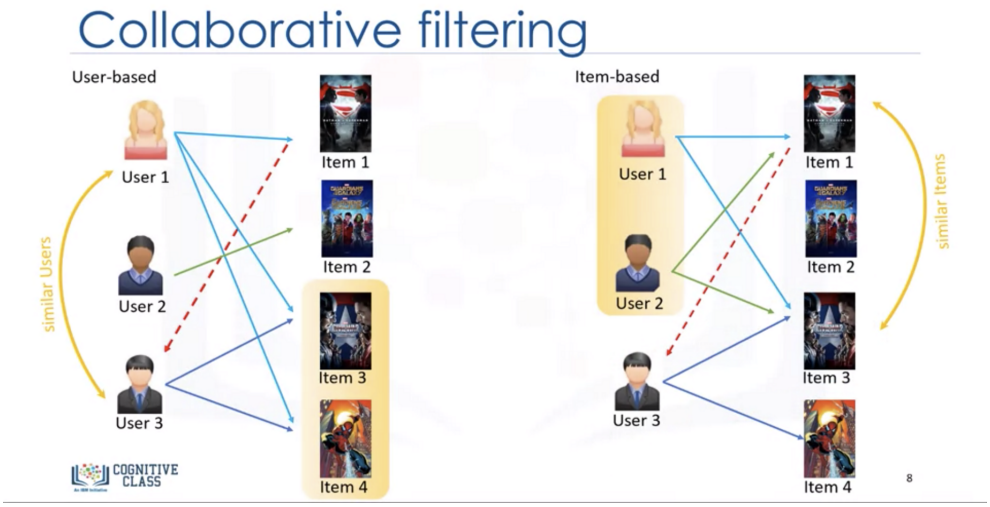



Figure 2.1: User-user vs Item-item. Note: user-user collaborative filtering is to recommend items based on the most similar users who share the analogous interest intersection area; item-item collaborative filtering is based on the most "popular" items that have the most interaction among user community [37].

Jhone is interested in purchasing a DVD. In this case, our task is to suggest a movie based on his past viewing preferences. We will first identify movies that Jhone has watched or liked. Let us call them "A," "B," and "C." Following that, we will search for other films similar to these three films. If we find out that movie 'D' is very similar to movie 'C,' then there is a high likelihood that Jhone will also like movie 'D' because it is a similar movie to one that he already likes. As a result, we will suggest the movie "D" to Jhone.

**User-user vs Item-item:** A user-to-user approach involves searching for users who have interacted with the same items in a similar manner. Due to the fact that most users interact with only a few items on a regular basis, the method has high variance (high sensitivity). Moreover, due to the fact that the final recommendation is based solely on interactions recorded for similar users to our target users, we obtain more tailored results (low bias). In contrast, the item-item method focuses on searching similar items in terms of user-item interactions. Since many users generally interact with a specific item, the neighbourhood search is less sensitive to single interactions (lower variance). Consequently, interactions originating from any user (including those who differ significantly from our reference user) are included in the recommendation, making it less personalized (more biased). It is, therefore, less personalized than the user-user approach but more robust.

**Memory based collaborative approaches shortage:** A significant disadvantage of memory-based collaborative filtering is that it does not scale easily: generating a new recommendation can take an extremely long time for extensive systems. If the nearest neighbours search step is not carefully designed, it can become intractable (the KNN algorithm has a complexity of $O(ndk)$ where n is the number of users, $d$ is the number of items, and $k$ is the number of neighbours to consider). We can use either approximate nearest neighbours (ANN) or take advantage of the sparsity of the interaction matrix when designing our algorithm to make computations more tractable for massive systems. To prevent what could be called an "information confinement area" for users, we need to be extremely careful in most recommendation algorithms so that the "rich get richer" effect does not occur for popular items. Alternatively, we do not want our system to tend to recommend only popular items, nor do we want to restrict users to only receiving recommendations for items remarkably similar to the one they already liked, moreover, without hearing about new items they might enjoy (because they are not "close enough" to be recommended). It is especially true of memory-based collaborative algorithms that can lead to these problems, as we mentioned earlier. As a matter of fact, without the model "to regularize," this kind of phenomenon can be reinforced and seen more frequently.

### 2.1.3 Model based collaborative approaches

The model based collaborative approaches rely solely on information about the interaction between users and items, assuming a latent model of them. Matrix factorization, for instance, allows for the reduction of a huge and sparse user-item interaction matrix into a product of two smaller and dense matrices: a user-factor matrix (representing users) multiplied by a factor-item matrix (representing items).

**Matrix factorization** Matrix factorization is primarily based on the assumption that there is a latent features space in which we can represent both users and items and such that interactions between users and items can be calculated from the dot product of the corresponding dense vectors in that space. Let us say we have a matrix for rating movies by users. We can assume the following regarding interactions between users and movies: A few features describe (and distinguish) movies reasonably well. Additionally, these features can also be used to describe user preferences (high values if the user enjoys the features, low values otherwise).

However, we do not want to incorporate these features into our model (as we could do so with content-based approaches, as discussed below). It is preferable to let the system discover these valuable features and make its representations of both items and users. In

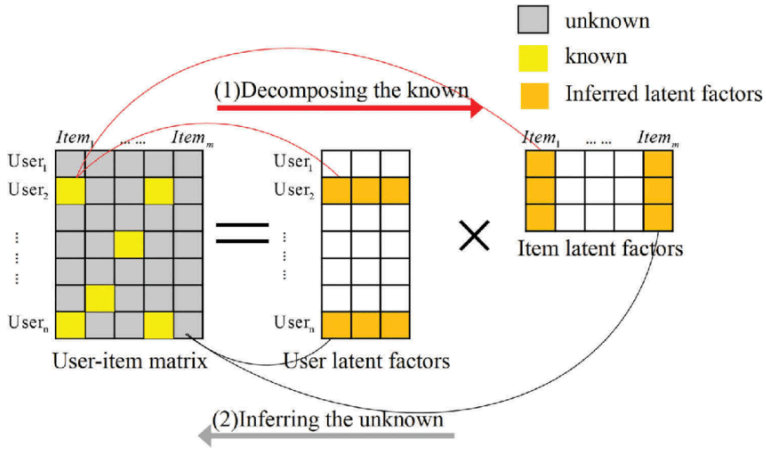



Figure 2.2: Matrix factorization illustration: In the absence of user-item relationships, an unknown user-item relationship can be predicted by decomposing the existing user-item matrix into latent user features and item features. [19]. Then the unknown user-item relationship can be predicted by the feature matrix dot product.

other words, since they are learned and not given, extracted features are a mathematical result but do not have an intuitive interpretation (and, thus, are hard to understand as a human). Despite this, it is not uncommon for structures resulting from algorithms of that sort to be very similar to intuitive structures conceivable to humans. As a result of this factorization, users with similar preferences and items with similar characteristics are represented in the latent space in close proximity. For example, consider an interaction matrix of ratings in which only a few items are rated by each user (most of the interactions are None to indicate no ratings). We seek to factorize this matrix such that each user is considered a distinct component.

$$M \approx X \cdot Y^T$$

$X$ is the "User Matrix" corresponding to the $n$ users. $Y$ is the "Item Matrix" corresponding to the $m$ items.

$$user_i \equiv X_i \; \forall i \in \{1, .., n\}$$

$$item_j \equiv Y_j \; \forall j \in \{1, .., m\}$$

Users and items will be represented within the latent space defined by $l$. Therefore, we seek those matrices $X$ and $Y$ whose dot product best approximates the existing interactions. We want to determine which X and Y sets minimize "rating reconstruction error" by

considering E the ensemble of pairs (i,j) such that $M_ij$ is set as ground truth (not None).

$$(X, Y) = argmin \sum_{(i,j)\in E} [(X_i)(Y_j)^T - M_{ij}]^2$$

We would add a regularization factor and divide it by 2.

$$(X, Y) = argmin \frac{1}{2} \sum_{(i,j)\in E} [(X_i)(Y_j)^T - M_{ij}]^2 + \frac{\lambda}{2}(\sum_{i,k}((X_{ik})^2 Y_{ik})^2))$$

Following the gradient descent optimization process, we can obtain matrices $X$ and $Y$, which we can notice two things. To begin with, the gradient does not necessarily have to be computed over all pairs in E at each step. Instead, we can consider only a subset of these pairs to optimize our objective function in batches. Secondly, $X$ and $Y$ do not have to be updated simultaneously, and the gradient descent can be performed alternatively on each matrix at each step (by doing so, we consider one matrix fixed and perform the optimization for the other before doing the opposite successive steps). Having factorized the matrix, we have fewer variables to manipulate to make a new recommendation: a user vector can be multiplied by any item vector to estimate a new rating. With these new representations of users and items, we should also use user-user and item-item methods: nearest neighbours searches would no longer be carried out over huge sparse vectors but over small dense ones, thus making some approximation techniques more tractable.

### 2.1.4 Content Based Approaches

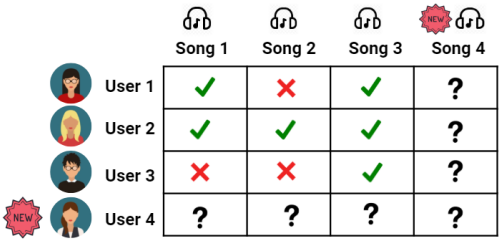



Figure 2.3: Content-based filtering, where each row represents an individual user, and each column represents one of the features categories.

Based on the user's previous actions or explicit feedback, content-based filtering recommends other items similar to what they like. Let us hand-engineer some features for the Google Play store to demonstrate content-based filtering. The following figure 2.3 represents a feature matrix in which each row represents an app, and each column represents a feature. For simplicity, assume this feature matrix is binary: a non-zero value indicates that the app contains the specified feature. A feature could include categories (such as Education, Casual, Health), the publisher of the application, and many others.

Additionally, we can represent the user within the same feature space. The user could explicitly specify certain user-related features. A user, for example, may select "Entertainment apps" in their profile. The presence of other features is implicit and can be determined by the prior apps they have installed. For instance, the user may have previously installed an app from Science R Us.

It is critical for the model to recommend relevant items to the user. In order to do so, we must first choose a similarity metric (such as dot product). In that case, you will have to configure the system to evaluate each candidate item using this similarity metric. The model's recommendations are specific to this user only since it did not consider any other information.

Let's say that both the user's embedding of $x$ and the app's embedding of $y$ are binary vectors. The number of features that are simultaneously active in both vectors is given by $< x, y >$. $< x, y >= \sum_{i=1}^{d} x_i y_i$ implies that features appearing in both x and y contribute 1 to the sum. It is then apparent that a high dot product has more similar features, thus indicating a higher similarity.

## 2.2 Graph Neural Network

In recent years, Graph Neural Networks (GNN) has gained increasing popularity in a variety of contexts, including social networks, knowledge graphs, recommendation systems, and even life sciences. GNN's ability to model node dependencies within a graph has enabled breakthroughs in graph analysis research. Here we will cover two advanced algorithms, DeepWalk and GraphSage, as well as the basics of Graph Neural Networks.

To begin the discussion of GNN, let us first understand what Graph is. Graphs are data structures consisting of vertex and edge components in computer science. The set of vertex $V$ and edges $E$ that makeup graph $G$ provides a concise description.

$$G = (V, E)$$

Depending on whether there are directional dependencies between vertices, edges can be either directed or undirected. In many instances, vertices are also called nodes, and these terms are interchangeable.

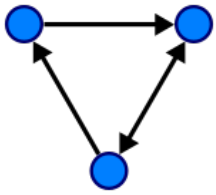


Figure 2.4: A directed graph

The Graph Neural Network is a type of Neural Network that operates directly on the graph structure. Node classification is one typical application of GNN. Essentially, every node in the graph has a label, and we wish to predict the label of each node without requiring a ground truth. The following section illustrates the algorithm described in the paper [33], which was the first proposal for GNN and thus often considered the original GNN.

Each node v in the node classification problem is characterized by its feature $x_v$ and assigned a ground-truth label $t_v$. In the context of a partially labelled graph $G$, we seek to utilize these labelled nodes to predict the unlabeled nodes' labels. Each node is represented by a d-dimensional vector (state) $h_v$, containing information about its neighbours.

$$h_v = f(x_v, x_{co[v]}, h_{ne[v]}, x_{ne[v]})$$

Specifically, the function $f$ is the transition function projecting these inputs onto a d-dimensional space. It specifies $X_{co[v]}$ as the features of the edges connected to v, $h_{ne[v]}$ as the embedding of the neighbouring nodes of $v$, and $x_{ne[v]}$ as the features of the neighbouring nodes of v. In order to find the unique solution of $h_v$, we should implement the Banach fixed point theory [12] as iterative updating process. Message passing or neighbourhood aggregation are terms used to describe such operations.

$$H^{t+1} = F(H^t, X)$$

Concatenation of all $h$ and $x$ is represented by $H$ and $X$, respectively. An output function $g$ computes the output of the GNN by passing the state $h_v$ and the feature $x_v$ to it.

$$o_v = g(h_v, x_v)$$

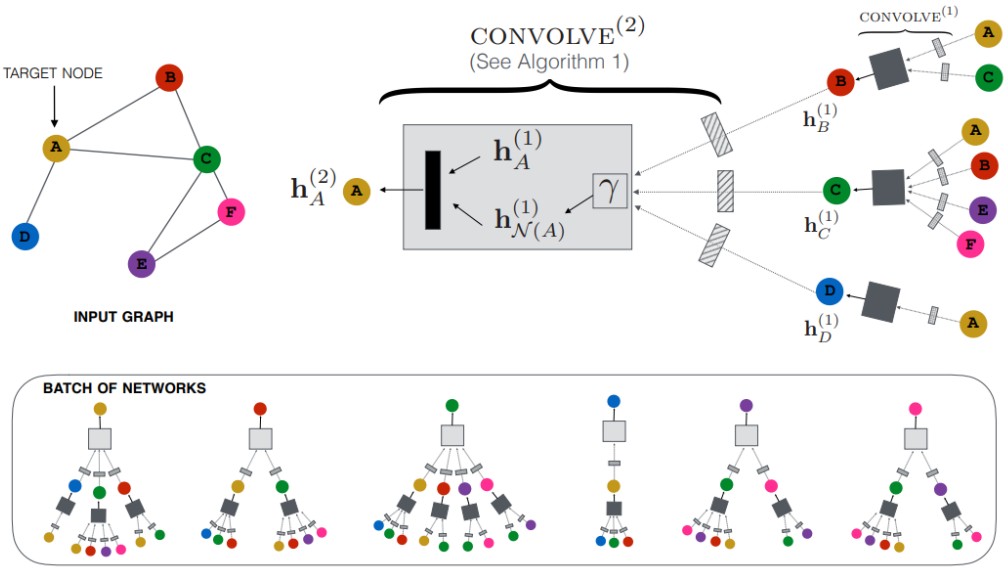



Figure 2.5: Graph neural network structure [10]

Feedforward fully-connected neural networks can be interpreted as both $f$ and $g$. It is straightforward to formulate the L1 loss as follows:

$$loss = \sum_{i=1}^{p} (t_i - o_i)$$

zhou2020graph point out three main limitations with the GNN proposal:

1. If the assumption of "fixed point" is flexible, Multi-layer Perceptrons can be used to build a more stable representation and eliminate the iterative updating process. According to the original proposal, different iterations used the same parameters for the transition function f, whereas the different parameters allowed for hierarchical feature extraction in different layers of MLP.

2. This method cannot handle edge information, such as how edges indicate different relationships between nodes in a knowledge graph.

3. The fixed point representation may discourage the diversification of node distribution and may not be suitable for learning to represent nodes.

Several GNN variant structures have been proposed to fix the above issues.

### 2.2.1 DeepWalk

In contrast to previous algorithms, DeepWalk [26] is the first algorithm that allows node embedding to be learned unsupervised. In terms of the training process, it is very similar to word embedding. As shown in the following figure, both nodes in a graph and words in a corpus follow power-law distributions:

Two steps are involved in this algorithm:

1. Generate node sequences by performing random walks on nodes in a graph.
2. Using the node sequences generated in step 1, run skip-gram to learn the embeddings of each node.

Random walk steps are performed uniformly to select the next node from its neighbor at each time step. Subsequences of each sequence are truncated to length $2|w| + 1$, where $w$ is the skip-gram window size.

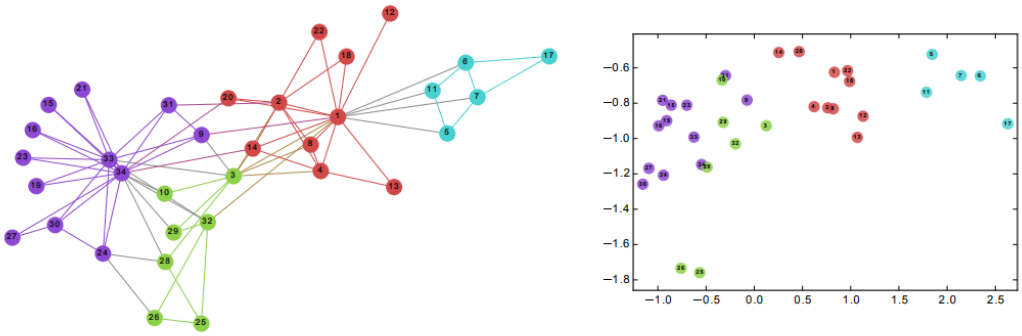
The paper presents hierarchical softmax as a solution for the computationally expensive computation of softmax due to the large number of nodes. In order to determine the softmax value of each output element, we must compute the $e^{x_k}$ of all $k$ elements.

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$$

For the original softmax, the computation time is $O(|V|)$, where $V$ is the set of vertices in the graph.

Binary trees are used to solve the problem in hierarchical softmax. A binary tree element can now be computed in $O(log(V))$ time due to the fact that the longest path can be expressed in $O(log(n))$, where $n$ is the number of leaves. The leaves of this binary tree ($v1, v2, ...v8$ in the above graph) are the graph's vertex points. Each of the inner nodes has a binary classifier to determine which path to choose. The probability of a given vertex $v_k$ is computed by adding the probabilities of each subpath from the root node to the leave vertex $v_k$. With each node's children adding up to 1, the sum of all vertices is one remains valid in the hierarchical softmax.

As shown in the following figure , when a DeepWalk GNN is trained, it learns a good representation of each node. A colour variation is indicative of a different label in the input graph. According to the output graph (embedding two dimensions), nodes with similar labels are clustered together, while most nodes with different labels are properly separated.



(a) Input: Karate Graph (b) Output: Representation

Figure 2.6: The DeepWalk GNN learns a latent space representation of community structure where input graph (a) can be converted into latent representation in (b) [46].

The DeepWalk GNN is applied to Zachary's Karate network to generate latent representations in L2-distance. It is notable how the community structure in the input graph matches the neighbourhood structure in the embedding. Colours of vertex edges indicate clustering based on modularity. Despite its impressive features, DeepWalk's main weakness is its inability to generalize. To represent a new node (transductive), the model must be re-trained whenever a new node comes in. This type of GNN is inappropriate for graphs in which nodes are constantly changing.

### 2.2.2 GraphSage

GraphSage addresses the problem mentioned above by inductively learning the embedding per node. To be more specific, nodes are represented by their neighbourhoods. The node can still be correctly represented by its neighbours even if a new node previously unknown appears in the graph during training. GraphSage's algorithm is shown below.

This outer loop indicates the number of update iterations, while $h_v^k$ is the latent vector's value at update iteration $k$. The $h_v^k$ value is updated at each update iteration based on the latent vectors of $v$ and $v$'s neighbourhood in the previous update iteration, along with the weight matrix $W^k$.

By aggregating nearby nodes, representable embedding can be generated for unseen nodes. Node embedding can be applied to dynamic graphs whose structure is ever-changing. As the core of their content discovery system, Pinterest uses an enhanced version

**Algorithm 1:** GraphSAGE embedding generation (i.e., forward propagation) algorithm

**Input :** Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth $K$; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity $\sigma$; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

**Output :** Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
**for** $k = 1...K$ **do**
  **for** $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  **end**
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
**end**
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Figure 2.7: GraphSAGE Algorithm [10].

of GraphSage called PinSage.

### 2.2.3 Inductive vs Transductive Learning

**Induction:** As a result of induction, observed training cases can be turned into general rules that can be applied to test cases. Inductive learning is the same thing as what is generally known as supervised learning. We construct and train a machine learning model using a labelled training dataset that we already have. Next, we apply this trained model to a testing dataset that we have never encountered before to predict the labels.

**Transduction:** Transduction involves reasoning from observed, specific cases (training) to specific cases (test). Comparatively to inductive learning, transductive learning has observed all the data in advance. The training dataset is used to infer the labels for the testing dataset based on the already observed training dataset. Although we do not know the labels of the testing datasets, we can still utilize the patterns and additional information in these datasets to aid in the learning process.

**Comparision:** There is a fundamental difference between transductive learning and inductive learning. The model has already encountered both the training and testing datasets during transductive learning training. As opposed to this, inductive learning encounters only the training data when training the model and then applies it to a dataset that it has never seen before.

The process of transduction does not generate a predictive model. In the case of adding new data points to the testing dataset, we would need to run the algorithm from scratch, train the model, and then use it to determine the labels. Alternatively, inductive learning creates an algorithm that builds a model that predicts data points. The algorithm does not have to be re-run when encountering new data points.

By definition, inductive learning seeks to build a generic model from an observed set of training data points to predict any future data points. In this case, the model can predict any point in the space of points, except those unlabelled. Transductive learning, on the other hand, constructs models based on data points already observed during training and testing. Utilizing the knowledge of labelled points and additional information, this approach predicts the labels of unlabeled points. Transductive learning can be expensive when a data stream contains new data points. New data points can be classified within a relatively short period and with a relatively small number of computations. It will be necessary to re-run the entire process every time a new data point is added. However, inductive learning initially generates a prediction model.

## 2.3 Transformer

The learning process of a transformer relies on the deep learning model, which uses the self-attention mechanism to differentiate the importance of every part of the input data. Most of the time, it is used in the fields of Natural language processing (NLP) [38] and Computer vision (CV) [42].

For tasks like translation and text summarization, transformers are intended to handle sequential input data, such as natural language. They are similar to recurrent neural networks (RNNs). The data is not always processed in the same sequence as it would be with RNNs. As a result, every place in the input sequence may be contextualized by the attention mechanism. The transformer does not need to handle the beginning of the sentence before the end of the sentence if the input data is a natural language sentence. Instead, it determines the context in which each word in the phrase acquires its meaning. In comparison to RNNs, this characteristic allows for more parallelization, which saves training time.

Developed by a team at Google Brain in 2017, transformers are quickly becoming the model of choice for natural language processing (NLP) problems [5] replacing RNN models such as long short-term memory (LSTM). Additionally, training on bigger datasets is now feasible due to the implementation of training parallelization. This research entails

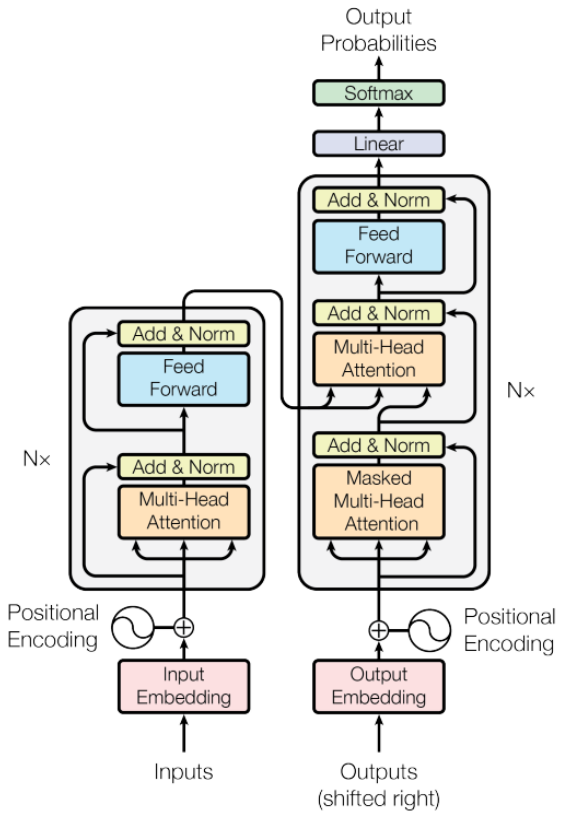



Figure 2.8: Transformer structure [38].

the development of pre-trained systems such as Bidirectional Encoder Representations from Transformers (BERT) and Generative Pre-trained Transformer (GPT) that have been trained with large language datasets, such as the Wikipedia Corpus and Common Crawl, and can be fine-tuned for specific tasks. [5] [27]

### 2.3.1 Attention

Attention methods were used to solve these issues. Models with attention mechanisms may access the state at any point in the sequence prior to the current point. Previous states are accessible to the attention layer so that relevant information about tokens that are far away may be provided.

The translation is an excellent illustration of the need to pay attention since context

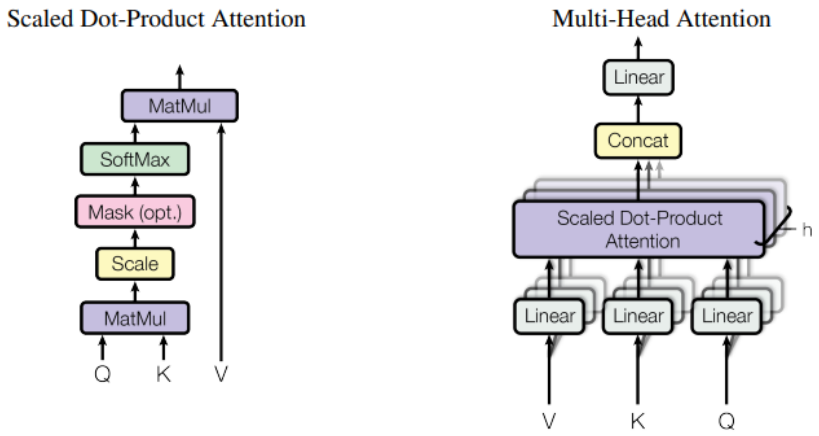



Figure 2.9: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel [38].

is critical in assigning a word's meaning to a phrase. The first word of the French output is likely to be substantially influenced by the first few words of the English input in a translation system from English to French. However, in a standard LSTM model, the model is simply given the state vector of the final English word in order to construct the first word of the French output. This vector may theoretically contain all of the required information about an English phrase. The LSTM, in reality, generally fails to retain this information. Additionally, the decoder is given access to the state vectors of every English input word, not just the latest one. This allows it to acquire attention weights that define how much to focus on each English input word.

Attention mechanisms enhance the performance of RNNs. Recurring recurrent processing of input is not required for RNNs with attention since attention mechanisms are strong in their own right, the Transformer design has shown. Without an RNN, Transformers analyze all tokens simultaneously and calculate attention weights between them in subsequent levels. Training speed is boosted by the fact that the attention mechanism only needs input from lower layers; thus, all tokens may be calculated simultaneously.

### 2.3.2 Scaled Dot-product Attention

Scaled dot-product attention units are implemented in the transformer building blocks, which are also known as attention units. Attention units generate embeddings for every

token in the context that includes information on the token itself as well a weighted combination of other relevant tokens, each of which is weighted by the attention weight of the token in question. When a sentence is routed through a transformer model, attention weights between each token are computed simultaneously.

The transformer model learns three weight matrices for each attention unit: the query weights $W_Q$, the key weights $W_K$, and the value weights $W_V$. For each attention unit, the transformer model learns three weight matrices. When a token $i$ is encountered, the input word embedding $x_i$ is multiplied by each of the three weight matrices to create three vectors: a query vector $q_i = x_i W_Q$, a key vector $k_i = x_i W_k$, and a value vector $v_i = x_i W_v$. We can compute the attention weights using the query and key vectors: the attention weight $a_{ij}$ from token $i$ to token $j$ is the dot product of the query and key vectors $q_i$ and $k_j$, respectively. When the attention weights are split by the square root of the key vector dimension, $\sqrt{d_k}$, the gradients become more stable throughout training. The attention weights are then processed through a softmax to make them consistent across trials. Because $W_Q$ and $W_K$ are separate matrices, attention may be non-symmetric: if token $i$ attends to token $j$ i.e. $q_i \cdot k_j$ is large is enormous), this does not imply that token $j$ will attend to token $i$ (i.e. $q_j \cdot k_i$ might be small). For each token $i$, the output of the attention unit is the weighted sum of the value vectors of all tokens, with each value vector being weighted by $a_{ij}$, which represents the attention from token $i$ to each token.

The attention calculation for all tokens may be written as a single massive matrix calculation using the softmax function, which is particularly beneficial for training owing to computational matrix operation optimizations that allow for the computation of matrix operations to be done fast. These are the matrices whose first and last rows are vectors $q_i$, $k_i$, and $v_i$, respectively, and whose first and last rows are matrices with the same $i$th row.

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

### 2.3.3 Multi-head Attention

An attention head is a collection of $W_Q$, $W_K$, and $W_V$ matrices, and each layer in a transformer model has many attention heads. While each attention head focuses on the tokens that are important to that token, the model may do so for several meanings of "relevance." Additionally, the impact field associated with relevance may be gradually dilated into successive layers. Numerous transformer attention heads encode relevant relevance links for humans. For example, some attention heads focus only on the next word, while others focus

exclusively on verbs and their immediate objects [4]. Each attention head's calculations may be conducted parallel, allowing for rapid processing. The attention layer's outputs are concatenated and fed into the feed-forward neural network layers.

### 2.3.4 Sentence Transformer

The Sentence encoder in our system is mainly based on DistilBERT, one of the sentence-BERT types pre-trained on the nli dataset. Sentence-BERT (sBERT) is a version of the BERT network that utilizes siamese and triplet networks to produce semantically relevant sentence embeddings. This allows BERT to be utilized for certain new activities that were previously inaccessible to BERT. These tasks involve comparing massive sets of semantic similarities, grouping, and retrieving information through semantic search.

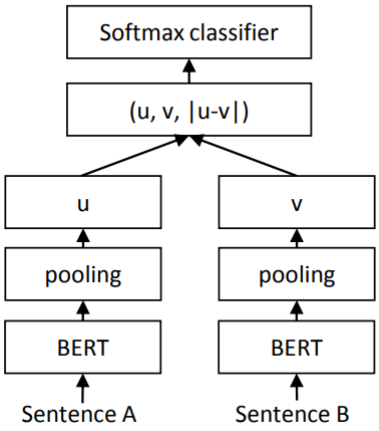



Figure 2.10: SBERT architecture (siamese network structure) [28] with classification objective function, e.g., for fine-tuning on SNLI dataset.

BERT established new benchmarks for performance on a variety of phrase categorization and pairwise regression problems. The BERT algorithm employs a cross-encoder: Two phrases are given to the transformer network, which predicts the goal value. This configuration, however, is unsatisfactory for a variety of pair regression tasks because of the large number of potential combinations. Finding the pair with the highest similarity level in a batch of $n = 10000$ phrases involves BERT $n \cdot (n-1)/2 = 49995000$ inference operations. This takes around 65 hours on a contemporary V100 Graphics processing unit (GPU). Similarly, determining which of Quora's over 40 million existing questions is the

most comparable to a new question might be treated as a pairwise comparison using BERT, but addressing a single inquiry would take over 50 hours.

Clustering and semantic search are often addressed by mapping each phrase to a vector space in such a way that semantically comparable sentences are near. Individual sentences were entered into BERT, and fixed-size sentence embeddings were produced. The most often used method is to average the BERT output layer (also known as BERT embeddings) or to utilize the first token's output (the [CLS] token). As we shall see, this popular technique produces relatively poor sentence embeddings, often worse than those obtained by averaging GloVe embeddings.

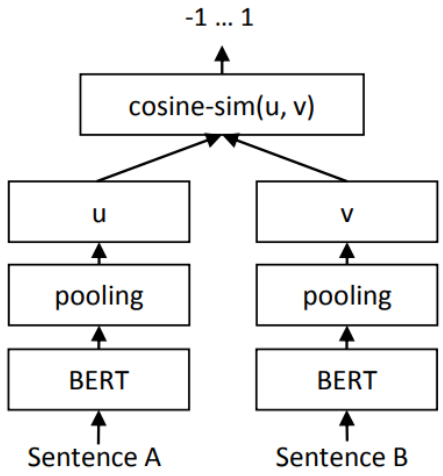



Figure 2.11: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

SBERT was created to address this problem. The siamese network design permits the derivation of fixed-sized vectors for the input texts. Semantically related phrases may be identified using a similarity metric such as cosine similarity or Manhatten-Euclidean distance. Due to the high efficiency with which these similarity measures can be computed on contemporary technology, SBERT may be used for both semantic similarity search and clustering. The difficulty of identifying the most similar sentence pair in a collection of 10,000 sentences is lowered from 65 hours with BERT to the calculation of 10,000 sentence embeddings (5 seconds with SBERT) and cosine similarity computation (0.01 second). Finding the most comparable Quora topic may be shortened from 50 hours to a few moments by employing efficient index structures.

The author fine-tunes SBERT using NLI data, resulting in sentence embeddings that outperform state-of-the-art approaches such as InferSent and Universal Sentence Encoder. SBERT outperforms InferSent by 11.7 points and Universal Sentence Encoder by 5.5 points on seven Semantic Textual Similarity (STS) tests. The paper gain an increase of 2.1 and 2.6 points, respectively, on SentEval, a toolbox for evaluating sentence embeddings.

## 2.4 FAISS

FAISS is a package for searching for and grouping dense vectors efficiently. It includes methods for searching through collections of vectors of any size, up to those that may not fit in RAM. Additionally, it offers accompanying code for parameter assessment and adjustment.

FAISS has numerous ways of doing similarity searches. It is assumed that the instances are vectors with integer identifiers and that the vectors may be compared using L2 (Euclidean) distances or dot products. A query vector has a low L2 distance as well as a high dot product. Additionally, since this is a dot product on normalized vectors, it allows cosine similarity. The majority of approaches, such as those based on binary vectors and compact quantization codes, rely entirely on compressed representations of the vectors and do not need the original vectors to be retained. While this often results in a less exact search, these approaches are capable of scaling to billions of vectors in main memory on a single server.

The GPU implementation can read data from either the Central processing unit (CPU) or the GPU's memory. GPU indexes may be used in lieu of CPU indexes (e.g., IndexFlatL2 can be replaced with GpuIndexFlatL2), and copies to and from GPU memory are handled automatically. However, results will be quicker if both input and output stay on the GPU. The application supports both single-and multi-GPU configurations.

### 2.4.1 Similarity Search

Traditionally, databases are composed of organized tables that hold symbolic data. For instance, an image collection may be represented as a table with one row for each indexed photograph. Each row provides information on the picture, such as an identifier for the image and descriptive text. Rows may also be connected to entries in other tables. For example, a picture containing individuals can be linked to a database of names. AI methods such as text embedding (word2vec) or deep learning-trained convolutional neural network

(CNN) can create high-dimensional vectors. These representations are far more powerful and adaptable than static symbolic representations. Nonetheless, standard databases that support SQL queries have not been modified to these new forms. To begin, the massive influx of new multimedia products results in the creation of billions of vectors. Second, and maybe more crucially, comparable entries need equivalent high-dimensional vectors, which is expensive, if not impossible, using standard query languages.

Consider the following scenario: the city hall of a midsize city whose name users have forgotten — and the user want to locate all other photos of the same building in the image collection. A key/value query, which is often used in SQL, is useless since the user forgotten the city's name.

This is the point at which similarity search becomes useful. The vector representation of pictures is intended to provide comparable vectors for similar images, where similar vectors are defined as those in Euclidean space that are close together.

Classification is another use of vector representation. Considering the situation in which the user want a classifier to identify which photographs in a collection depict a daisy. The classifier's training procedure is well-known: The algorithm accepts as input photos of daisies and non-daisy objects (cars, sheep, roses, cornflowers). If the classifier is linear, it produces a classification vector whose dot product with the image vector indicates the probability that the picture includes a daisy. Then, the dot product may be calculated using all of the collection's elements, and the photos with the highest values are returned. This is a "maximum inner-product" query.

Thus, we need the following procedures for similarity search and classification:

1. Return a list of database items that are closest to a given query vector in terms of Euclidean distance.

2. Return a list of database items that have the highest dot product with a given query vector.

Currently present software tools are insufficient for performing the database search procedures outlined above. Traditional SQL database systems are unsuitable for this purpose since they are geared for hash-based or 1D interval searches. Similarity search functions included in packages such as OpenCV, as well as other similarity search libraries that examine "small" data sets, are severely constrained in terms of scalability (for example, only 1 million vectors). Other packages include research artifacts created for the purpose of demonstrating performance in certain environments. In that case, Facebook proposed FAISS with following features.

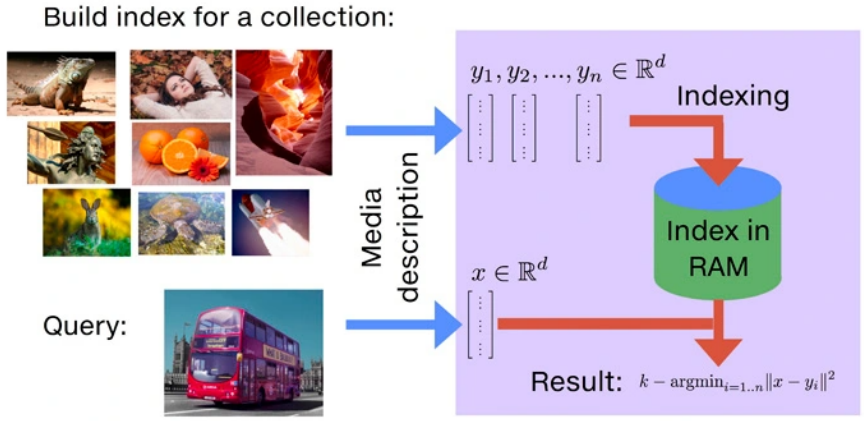



Figure 2.12: Facebook introduce FAISS that addresses the conventional SQL's limitation [17].

1. FAISS has a variety of similarity search algorithms that cover a broad range of use trade-offs.
2. FAISS's memory use and performance are optimized.
3. FAISS implements the most important indexing algorithms using state-of-the-art GPU technology.

After learning machinery extracts vectors (from photos, videos, and text documents, among other sources), they are ready to be fed into the similarity search library.

A reference brute-force technique computes all the similarities precisely and exhaustively and delivers the list of items with the highest degree of similarity, generating the "gold standard" list of reference results. Notably, successfully implementing the brute-force approach is not trivial, and it often affects the performance of other components.

Similarity searches may be significantly accelerated if we are prepared to sacrifice some accuracy; that is, depart somewhat from the reference result. For instance, it may be irrelevant whether the first and second results of an image similarity search are switched since both are undoubtedly right for a given query. Accelerating the search process requires some pre-processing of the data collection, which can refer to as indexing. In that case, there are three major measure can be implemented as metrics:

1. **Speed**: How long does it take to identify the ten (or other number) vectors that are the most similar to the query? Hopefully less time than the brute-force technique requires

2. **Memory consumption**: How much RAM is required for this method? More or fewer vectors than the originals? FAISS allows only memory-based searches, while disk-based databases are orders of magnitude slower.

3. **Accuracy**: How closely does the provided list of results match the results of the brute-force search? Accuracy may be determined by counting the number of queries for which the true closest neighbor is returned first in the result list (a metric called 1-recall@1) or by calculating the average percentage of 10 nearest neighbors returned in the first ten results (a metric called "10-intersection").

### 2.4.2 FAISS GPU Implementation

The GPU implementation took considerable work, resulting in astounding single-machine speed with native multi-GPU compatibility. Additionally, the GPU implementations are drop-in replacements for their CPU counterparts, and you do not need to be familiar with the CUDA API to take use of the GPUs. GPU FAISS is backwards compatible with any Nvidia GPUs released after 2012 (Kepler, compute capability 3.5+).

Facebook follows the roofline model, which suggests that one should aim to fully use the memory bandwidth or floating-point units. On a single GPU, FAISS GPU is often 5-10x quicker than FAISS CPU implementations. This is increased to 20x+ with new Pascal-class hardware, such as the P100.

1. With approximate indexing, a k =10 brute-force k-nearest-neighbour graph can be formed in 35 minutes on four Maxwell Titan X GPUs using 128D CNN descriptors of 95 million pictures from the YFCC100M data set with a 10-intersection of 0.8.

2. Now, billion-vector k-nearest-neighbor graphs are readily accessible. On four Maxwell Titan X GPUs, a brute-force k-NN graph (k = 10) of the Deep1B data set with a 10-intersection of 0.65 takes less than 12 hours; on eight, Pascal P100-PCIe GPUs, it takes less than 12 hours. On the Titan X setup, lower-quality graphs may be generated in less than 5 hours.

3. Other components operate well. For example, creating the Deep1B index above involves clustering 67.1 million 120-dim vectors to 262,144 centroids using k-means,

which takes 139 minutes on four Titan X GPUs (12.6 tflop/s of computation) or 43.8 minutes on eight P100 GPUs (40 tflop/s of computing) for 25 E-M iterations. Notably, the training set for the clustering algorithm does not need to fit in GPU memory since the data is streamed to the GPU as required without compromising speed.

# Chapter 3

# Query-Driven Graph-based User Recommender System

Existing recommender systems are still mainly about user-user or user-item recommendation, and there is no recommendation algorithm directly about request-user. This chapter will describe the components of a query-driven recommender system and the corresponding mechanism for its implementation.

Since matching user embedding directly with query embedding lacks some realistic feasibility due to the incomplete user features. In that case, we split the whole query-item process into two stages. The first stage matches the nearest item embeddings with the query embedding. In the second stage, Faiss and pre-trained t-GNN are implemeneted to match the selected items from stage one with the user embedding. The main functional module of our system consists of three parts as follows:

1. The first part is to encode all the items in the item database into sentence embeddings by pre-trained DistilBERT, and DistilBERT is also responsible for converting the query into sentence embedding.

2. **Query-item**: The second part uses the Faiss algorithm to select the TopN nearest items in the item database as the most similar items to the query.

3. **Item-user**: In the third part, we use our pre-trained t-GNN model to calculate the association probability from each user embedding to the TopN items and select the most likely users related to the TopN items. In this way, we can implement **query-user** recommendations by converting from query to item and item to user via the second-order transformation.

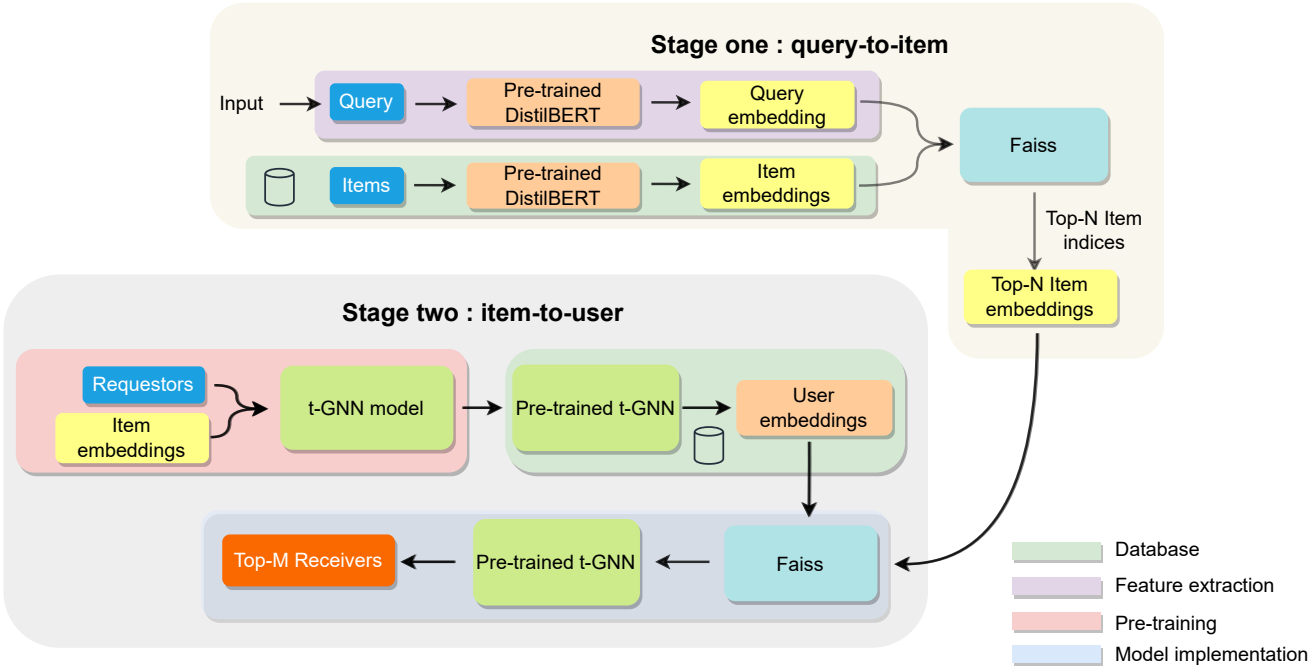



Figure 3.1: **System structure**: Accordingly, the entire system can be divided into the following components, including obtaining embedding of items using Distil-BERT, then pre-training t-GNN using the extracted embedding, items, and user-item interaction. As a result of these steps, the whole model can be utilized to predict query-user relationships. The testing phase begins with the DistilBERT algorithm generating the query embedding; then a Faiss algorithm matches the N most similar items based on the query embedding. Faiss identifies some clusters of users with similar characteristics during the application phase. Following that, pretrained t-GNN selects the top-M users from the matched user clusters.

As shown in the Figure 3.1, we have embedded all the components of the model together for a macroscopic presentation. However, in terms of the order of code application, the entire model is applied in three phases. In the next sections, we will describe in detail the functionality of the different models and the specific implementation methods.

## 3.1 Feature Extraction

During the feature extraction stage, we primarily construct pre-processing schemes based on two types of features. In addition to the introduction to our pre-processing pipeline for user and item data, we will describe how we built DistilBERT's pre-trained model to extract sentence embeddings from the data and how we incorporated the user and business features.

### 3.1.1 Data Preprocessing

**Items**: In the Yelp dataset, items mainly refer to a variety of businesses, most of which are restaurant-based, and each business is composed of several basic attributes, as shown in the following Table:

| Feature | Datatype | Description |
|---|---|---|
| business_id | String | 22 character unique string business id |
| name | String | The business's name |
| address | String | The full address of the business |
| city | String | The city |
| state | String | 2 character state code, if applicable |
| postal code | String | The postal code |
| latitude | Float | Latitude |
| longitude | Float | Longitude |
| stars | Float | Star rating, rounded to half-stars |
| review_count | Integer | Number of reviews |
| is_open | Boolean | 0 or 1 for closed or open, respectively |
| attributes | Boolean, String | Business attributes to values |
| categories | String | An array of strings of business categories |
| hours | String | An object of key day to value hours |

Table 3.1: Business attributes

As we can observe from the table above, the characteristics of business can be divided into the following parts: geographical information, business service information, business operation information, and the corresponding social media information.

Based on the data Table 3.1 above, we can see the Business data is composed of features of different data types, among which attributes fall into a special category. As shown in below examples:

| | |
|---|---|
| Garaje resturant | RestaurantsTakeOut: true, RestaurantsDelievery: true<br>BusinessParking:(garage: false, street: true, validated: false) |
| Steak house | BusinessAcceptsCreditCards:True, ByAppointmentOnly:False<br>BusinessParking:(validated: False, lot: False, valet: False) |

Table 3.2: Business Attribures Samples

According to Table 3.2, various businesses provide a range of different services, so they may have features of varying lengths. In this case, we cannot directly input the business features to the fixed-size feature extractor. Therefore, as shown in figure 3.2, we propose a dynamic transformation of features that transforms the features of the business into a descriptive sentence, which is then passed to a transformer-based encoder. As a result, we can embed sentences of different lengths according to a uniform specification. (Note: We only utilize Name, Address, State, Postal code, Categories, Attributes as major item features)


Figure 3.2: Dynamic Feature Transformation: We transform each business's feature into a descriptive sentence.

**Users**: As shown in the above Table 3.3, Next, we will discuss the most critical features related to users, as shown in the table below.

Apart from basic user personal information, most of the user characteristics are based on social media characteristics on Yelp, where the types of reviews and ratings are an important indicator of user characteristics. These indicators include whether the user's comments are available, interesting, and accompanied by relevant photos.

As the vast majority of user features are primarily numeric in nature, and the data associated with each of the users is consistent, no special data pre-processing was required. We also compared the 0-1 normalization method to the features of the original data and were not able to observe any significant differences between the two. Consequentially, the user features remain as they were. The final number of valid features is 19.

**Reviews**: The following diagram illustrates a sample graph depicting the direct relationship between users and business.

| Feature | Data Type | Description |
|---|---|---|
| user_id | String | 22 character unique user id String |
| name | String | The user's first name |
| review_count | Integer | The number of reviews they've written |
| yelping_since | String | When the user joined Yelp |
| friends | Array | An array of the user's friend as $user_{ids}$ |
| useful | Integer | Number of useful votes sent by the user |
| funny | Integer | Number of funny votes sent by the user |
| cool | Integer | Number of cool votes sent by the user |
| fans | Integer | Number of fans the user has |
| elite | Array | The years the user was elite |
| average_stars | Float | Average rating of all reviews |
| compliments | List | Multi-categories |

Table 3.3: User Attributes

In the user-business relationship, the point set is composed of either the user or the business, and the relationship between them represents the review given by the user to the business. As shown in the following Table 3.4, we will highlight the specific features of the Review as well as the pre-processing steps involved.

| Feature | Data Type | Description |
|---|---|---|
| review_id | String | 22 character unique review id. |
| user_id | String | 22 character unique user id |
| business_id | String | 22 character business id |
| stars | Integer | Star rating |
| date | String | Date formatted YYYY-MM-DD |
| text | String | The review itself |
| useful | Integer | Number of useful votes received |
| funny | Integer | Number of funny votes received |
| cool | Integer | Number of cool votes received |

Table 3.4: Review Feature

From the above Table, we can observe that the specific content of the review is composed of the user's subjective evaluation of the business, so the review is represented as an edge in our graph data. In our subsequent application, we mainly use the review "stars" and their corresponding dates as the main features to participate in the training process of the subsequent model.

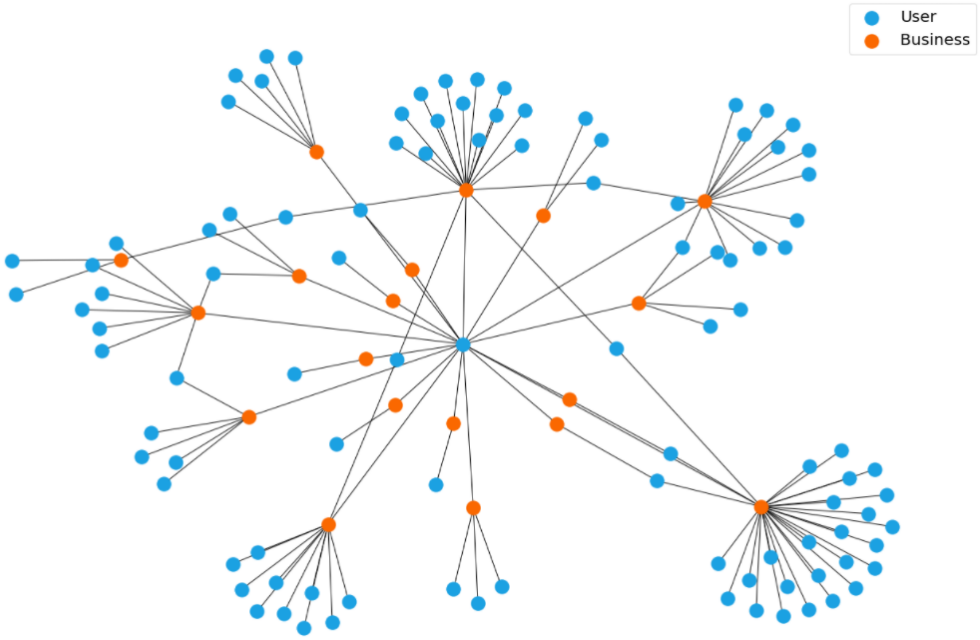



Figure 3.3: User-Business Graph

### 3.1.2 Sentence Embedding

Following feature extraction, the number of user features categories and the range of values of each user feature are fixed, and 'stars' and the review date are added as side attributes as part of the graph neural network training. However, business cannot be directly involved in the model training process due to its diversity and the corresponding variety of attributes and data types. Consequently, we present how the business description statements are encoded into fixed-length vectors, as well as the composition and rationale for the encoder.

Presented in the Figure 3.4 are the main elements of our sentence encoder, which include a tokenizer to tokenize the sentence and a sinusoidal function to encode the elements in various positions into the embedded sentence vector. The positional encoding functions are shown in below equation, where i is the desired position in an input sentence, PE is the corresponding encoding, d is the encoding dimension.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

As a result of the above location encoding, we can accomplish the following functions.

- In every time-step (word position), it should produce a unique encoding.
- The distance between any two time-steps should remain constant regardless of the length of a sentence.
- Accordingly, the model should be applicable to longer sentences without difficulty. In addition, the model's values should be bound.
- This would ensure deterministic position encoding.

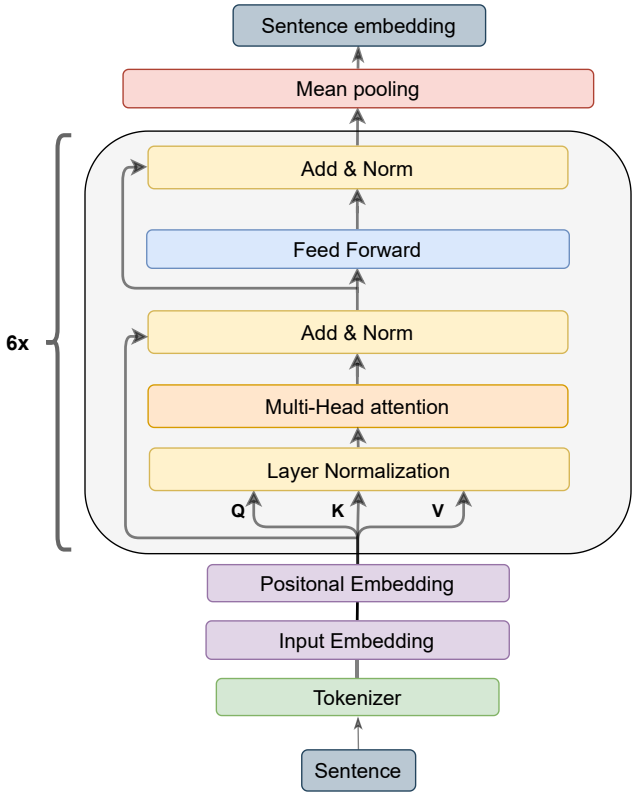



Figure 3.4: DistilBERT encoder structure

With the implementation of sentence vector embedding and position vector embedding, sentence embedding updates the original sentence vectors through feed-forward layer and

multi-head attention layer. For DistilBERT, the vectors are extracted from six layers of identical encoder layers with mean pooling layer. A 60% speed increase over the BERT model, as well as a 40% reduction in the number of parameters, was achieved, which met our requirement for efficient large-scale feature extraction. Lastly, using DistilBERT encoder, we shall extract query embedding and item embeddings.

## 3.2 t-GNN

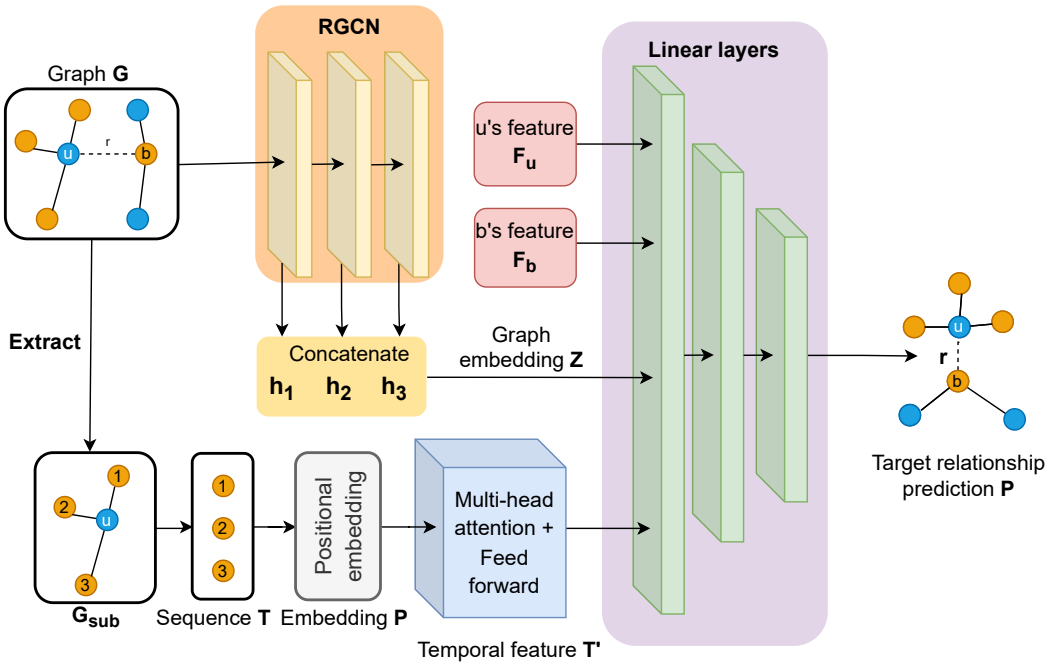



Figure 3.5: The figure shows the key steps in our model. The first step is to extract graph embedding $Z$ by RGCN from graph $G$. Multi-head attention + feed-forward is used to encode the user-centric subgraph $G_{sub}$ to temporal features $T'$. After combining spatial features, temporal features, target user features $F_u$, and target business features $F_b$, it is then possible to predict the relationship between u and b through linear layers.

In the following section, we will discuss the structure and mechanism of our t-GNN model, which can be divided into two phases from the perspective of the entire system, based on pre-training and actual use testing. Additionally, the pre-training process includes a series of pre-processing methods for training data, which are described in detail in the following section.

In order to implement the pre-training process, our approach comprises three stages, namely the pre-processing of graph data, the development of the model, and the testing of the model.

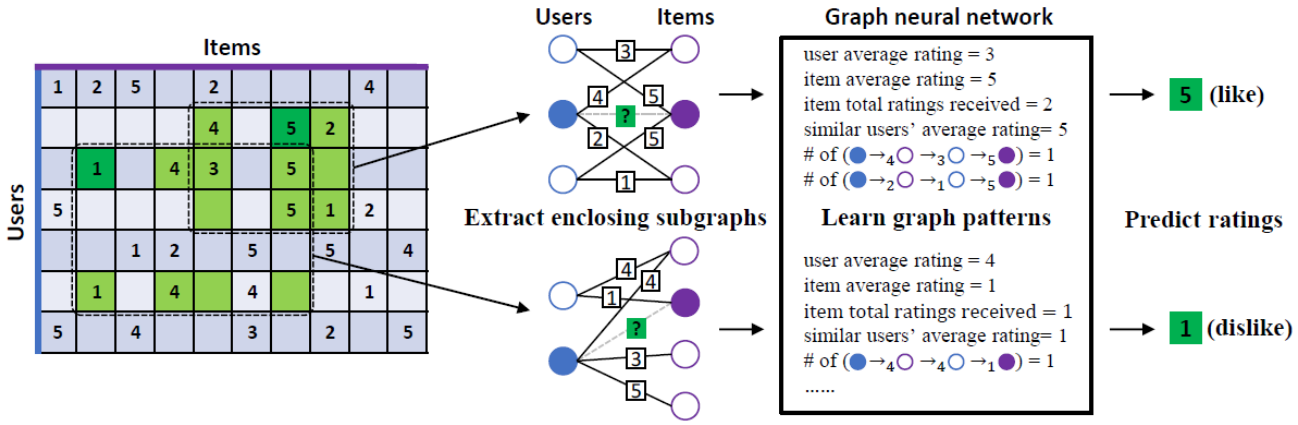



Figure 3.6: A GNN is trained to map subgraphs to ratings by extracting the local enclosing subgraph around each rating (dark green). The enclosing subgraphs are induced by the user and items associated with the target rating as well as their H-hop neighbors (where H=1). GNNs can learn mixed network patterns (such as average ratings, paths, etc.) to predict ratings from subgraphs. In the GNN box, examples of possible patterns are illustrated. The missing entries are completed by the trained GNN. [**?**]

### 3.2.1 Graph Data Preprocessing

Within this section, As shwon in Figure 3.6, we describe how a subgraph should be initiated. A subgraph is constructed around each item-business pair in the review data. Following this, the edge between item and business is masked, and the graph embedding provides the prediction of the relationship between item and business. In the following algorithm we will describe in detail the process and method of subgraph partitioning.

**Algorithm 1:** ENCLOSING SUBGRAPH EXTRACTION

**Data:** $h$, target user-business pair $(u, b)$, the graph $G$
**Result:** the h-hop enclosing subgraph $G^h_{u,b}$
$U = U_{neighbor} = \{u\}$, $B = B_{neighbor} = \{b\}$ ;
**while** $i <= h$ **do**
  $U\prime_{neighbor} = \{u_i : u_i \sim B_{neighbor}\} \setminus U$ ;
  $B\prime_{neighbor} = \{b_i : b_i \sim U_{neighbor}\} \setminus B$;
  $U\_neighbor = U \cup U\prime_{neighbor}, B = B \cup B\prime_{neighbor}$ ;
**end**
Remove edge $(u, b)$ from graph $G^h_{u,b}$ ;
return $G^h_{u,b}$

Additionally, the original purpose of our t-GNN model was to predict relationships based on local subgraphs rather than global features. Accordingly, the features of each point are not determined by the features of the original point, but rather by its role within the subgraph. In the following section, we present a one-hot feature encoding approach based on subgraphs.

### 3.2.2 Node Labelling

Node labelling is applied to an enclosing subgraph before it is fed to the GNN, which gives every node in the subgraph an integer label. In this way, different labels can be applied to nodes indicating their specific roles within a subgraph. A node label should be able to both: 1) distinguish between the target user and target item between which the target rating lies, and 2) distinguish between nodes belonging to the user type and those belonging to the item type. Without those details, the GNN cannot determine which item belongs to which user and may also lose node-type information.

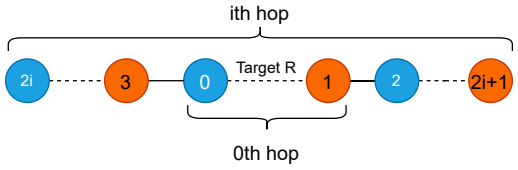



Figure 3.7: Node labelling

As shown in figure 3.7, the following labelling would fulfill these conditions: We assign 0 and 1 labels, respectively, to the target user and target item. Based on which hop they occur in the subgraph, we determine the labels of other nodes in Algorithm 1. We will assign 2i to a user-type node included at the ith hop. The node will receive 2i + 1 if it is an item-type node at the ith hop. By distinguishing nodes in this manner, one can distinguish 1) target nodes from "context" nodes, 2) users from items (users are always labelled the same), and 3) nodes whose distance from the target is different.

### 3.2.3 Spatial Analysis Module

As shown in Figure 3.5. Specifically, we implement two independent modules for both spatial and temporal feature extraction. First, we implement the three-layer Relational graph convolutional network (RGCN) to extract the hidden state $\{h_1, h_2, h_3\}$ from the graph $G$, and combine those three hidden states together as final graph embeddings $Z$. As another temporal analysis component, we would first extract the user-centric subgraph $G_{sub}$, and then construct the sequence $T$ based on the interaction sequences of the business to target users. Positional embedding $P$ and business embeddings would be combined in the following procedure. The temporal feature $T'$ is then extracted from the One-layer Multi-head attention + Feed-forward encoder. In the final step, we would feed all $T', Z$ and target user $u$'s feature $F_u$ and target business $b$'s feature $F_b$ together into linear layers to get the prediction $P$ for their relation $r$.

In the next section we will present the main principles of the spatial and temporal extraction modules respectively.

**RGCN** models are primarily motivated by an extension of the GCN concept to large-scale relational data that operates on local graph neighborhoods ( [7] [18]). These methods, as well as related methods such as graph neural networks [34], fall within the framework of a simple differentiable message-passing system:

$$h_u^{l+1} = \sigma(\sum_{m \in M_i} g_m(h_i^l, h_j^l))$$

This equation represents the hidden state of node $v_i$ in the $l$-th layer of the neural network, where $h_i^{l+1} \in R^{d^l}$ is the dimensionality of the representations in the $l$-th layer. An element-wise activation function $\sigma(\cdot)$ is applied to incoming messages of the form $g_m(\cdot, \cdot)$, such as $ReLU(\cdot) = max(0, \cdot)$. In node $v_i$, the number $M_i$ designates the set of incoming messages. This set is often chosen to be identical to the number of incoming edges. As

per Kipf and Welling (2017), $g_m(\cdot,\cdot)$ is typically chosen as a function similar to a neural network depending on the message or simply as a linear transformation $g_m(h_i, h_j) = Wh_j$ with a weight matrix $W$.

In the domain of graph classification [7] as well as graph-based semi-supervised learning [18], this type of transformation has been shown to be very useful for accumulating and encoding features from local, structured neighborhoods.

We derive the following propagation model from these architectures, in order to compute the forward pass update of an entity or node marked by $v_i$ in a multi-graph that is relational (directed and labeled):

$$h_i^{l+1} = \sigma(W_0^l h_i^l + \sum_{r\in\mathcal{R}} \sum_{j\in\mathcal{N}_i^r} \frac{1}{c_{i,r}} W_r^l h_j^l) \tag{3.1}$$

In this case, $N_i^r$ represents the set of neighboring indices under the relationship $r$. The normalization constant $c_{i,r}$ is either learned or pre-determined depending on the problem.

A normalized sum of the transformed feature vectors of neighbouring nodes explains how this equation accumulates transformed feature vectors. As opposed to regular GCNs, we introduce transformations specific to a given relationship, for example, depending on the type and direction of the edge. For each node in the data, we add one self-connection of a particular relationship type to ensure that the representation at layer l + 1 is informed by the representation at layer l. One could also use more flexible multi-layer neural networks (at the expense of computation efficiency) instead of simple linear message transformations.

For every node in the graph, RGCN is evaluated in parallel as part of an update of the neural network layer. By using sparse matrix multiplications, the RGCN can be implemented efficiently, avoiding explicit neighborhood summation. It is possible to stack multiple layers to accommodate dependencies across several steps of the relation. As the final graph embedding in our experiments, we combine features from different RGCN layers. In essence, this is done to reduce the likelihood of over-smoothing.

$$Z_h = concat(h_1, h_2, h_3) \tag{3.2}$$

**Over-smoothing**: Generally, we believe that the over-smoothing problem results from the mixing of information with noise, which is greatly influenced by the quality of the messages received by the nodes. A GNN operation may generate an interaction message from other nodes that either provide helpful information or cause interference. For instance, in the

node classification task, the interaction between nodes of the same class provides valuable information that makes their representations more similar, which increases the likelihood of the nodes being classified together. Conversely, the interaction of nodes from other classes causes noise. Therefore, GNNs are effective because the received useful information exceeds the noise. As opposed to this, if there is more noise than information, the learned graph representation will become over-smoothed.

### 3.2.4 Temporal Analysis Module

LSTM and Multi-head attention layer are the two main functional modules to be compared in the temporal analysis module, since the target users interact with the business arranged based on time. In this section, we will discuss the application steps in detail and the actual comparison relationship.

**LSTM:** In deep learning, long short-term memory (LSTM) [48] is an artificial recurrent neural network (RNN) architecture. This network has feedback connections, as opposed to conventional feedforward neural networks. In addition to processing single data points (e.g. images), it is also capable of processing entire sequences of data (e.g. speech or video). There are four components in a common LSTM unit: a cell, an input gate, an output gate, and a forget gate. The cell stores values over arbitrary time intervals, while the three gates control information flow inside and outside the cell.

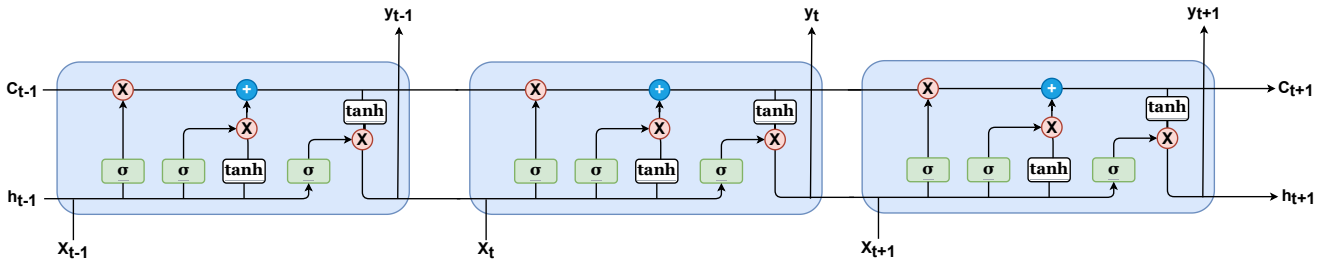



Figure 3.8: LSTM structure

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i)$$
$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$
$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$$

$$c'_t = \sigma(w_c[h_{t-1}, x_t] + b_c)$$
$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$
$$h_t^{LSTM} = o_t \cdot \sigma(c_t) \tag{3.3}$$

where $i_t$ represents input gate, $f_t$ represents forget gate, $o_t$ represents output gate, $\sigma$ represents Sigmoid function, $w_x$ weight for the respective gate(x) neurons. $h_{t-1}$ is the output of the previous LSTM block(at timestamp t-1). $x_t$ is the input at current timestamp. $b_x$ are biases for the respective gates(x).

The objective of this experiment is to input user-related events sequentially in temporal order into an LSTM and extract user-related temporal data using a single-layer LSTM. Afterwards, we insert the last hidden state of LSTM $h_t^{LSTM}$ directly into RGCN extracted spatial data and then merge the user and business data together to input to the final linear layer, where $t$ is the max time step of our LSTM cell.

**Multi-head attention:** During the comparison with LSTM, we first perform position embedding on the words in the sentence, following which the Attention mechanism is used to determine the importance of each component. Finally, the temporal features extracted from Attention are added to the linear classification layer.

Firstly, we will start with positional embedding, basically positional embedding is a D-dimensional vector containing information about a specific position within a sentence. Furthermore, the encoding is not integrated into the model. By contrast, this vector is used to provide each word with information about its position in a sentence. As a consequence, we enhance the model's input by inserting the words' order.

We will further illustrate the mechanism behind position embedding, we assume $t$ is the position in an input sentence, where $\vec{p_t} \in \mathbb{R}^d$ is the corresponding encoding, $d$ is the encoding dimension. In that case $f : \mathbb{N} \to \mathbb{R}^d$ would be the function that produces the final output vector $\vec{p_t}$ and it is defined as follow equation:

$$\vec{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k.t), & \text{if } i = 2k \\ \cos(\omega_k.t), & \text{if } i = 2k+1 \end{cases}$$

where

$$\omega_k = \frac{1}{10000^{2k/d}}$$

Since the original frequencies appears decreasing trendency along the vector dimension.

It forms a geometric progression from $2\pi$ tp $10000 \cdot 2\pi$ on the wavelengths. $\vec{p_t}$ is the vector contain pairs of sin and cos function for each frequency:

$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1.t) \\ \cos(\omega_1.t) \\ \\ \sin(\omega_2.t) \\ \cos(\omega_2.t) \\ \\ \vdots \\ \\ \sin\big(\omega_{d/2}.t\big) \\ \cos\big(\omega_{d/2}.t\big) \end{bmatrix}_{d \times 1}$$

After initializing the positional encoding for all input business based on sequential position. We start feeding the concatenation of positional embedding and business embedding into the multi-head attention layer.

As shown in Figure 3.9, the Attention module of the Transformer repeats its computation several times in parallel. Each repeat is referred to as an Attention Head. Query, Key, and Value parameters are split N-ways by Attention and each split passes independently through a separate Head. After combining these similar calculations, a final Attention score is calculated. Transformers with Multi-head attention are capable of encoding multiple relationships and nuances for each word.

In our experiment, we set the max-sequence length to $t$, the dimensionality of each business is $N_b$. In order to construct $Q, K, V$ pairs to generate the Query, Key, and Value, we would initialize three corresponding weight matrices $W_Q$, $W_K$ and $W_V$ to produce dot production with input data X. Assume we have $H$ heads for multi-head attetnion, as shown in the following equation, the final temporal embedding would be:

$$h_t^{Attention} = (\|_{h=1}^{H} Softmax(\frac{QK^T}{\sqrt{d_W}})V_k)W_{H \cdot N_b} \tag{3.4}$$

where $W_{H \cdot N_b}$ is the initialized weight matrix for the multi-head production and

$$Q = XW_Q$$

$$K = XW_K$$

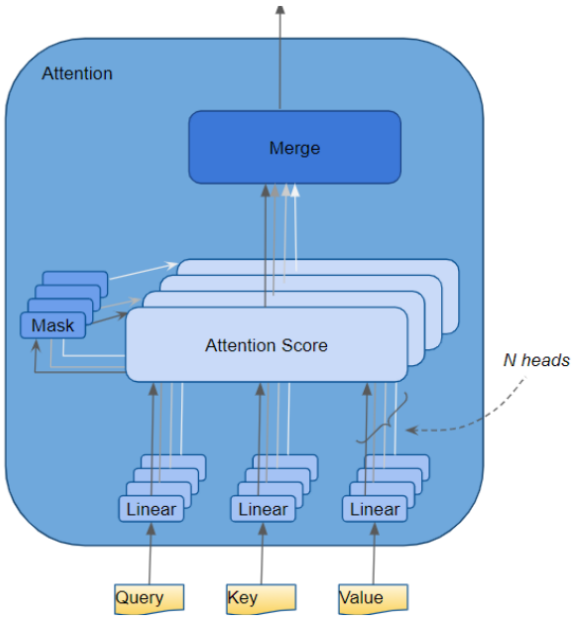



Figure 3.9: Multi-head Attention

$$V = XW_V$$

### 3.2.5 Pre-training: Loss Function

Based on our spatial and temporal module, we conclude our final embedding into following components:

1. **Spatio Component**: RGCN: $Z_h$ from equation 3.2.
2. **Temporal Components**: LSTM:$h_t^{LSTM}$ from equation 3.3. or Attention: $h_t^{Attention}$ from equation 3.4.
3. **User embedding**: $N_{User}$.
4. **Business embedding** $N_{Business}$ from DistilBERT encoder.

Base on the above components, we define our objective function into the following equation.

**Regression Objective Function**:

$$r_{u,b} = \lambda W_{linear} \text{concat}(N_{User}, N_{Business}, Z_h, h_t^{LSTM} | h_t^{Attention})$$

**Classification Objective Function**:

$$r_{u,b} = Argmax(Softmax(W_{linear} \text{concat}(N_{User}, N_{Business}, Z_h, h_t^{LSTM} | h_t^{Attention})))$$

Where $W_{linear}$ is the learnable learning parameter from final linear layer, $\lambda$ is the regularization parameter. The trainable parameters of our system are from RGCN, LSTM or Attention, final linear layers. As a result of the direct interaction between users and businesses, we performed both classification and regression tests, where classification refers to classifying each class according to the degree of division of the score from 1 to 5. The regression model, on the other hand, is capable of accurately predicting the exact score of the user assessment. we would describe both target function in the following equation respectively.

The loss function with respect to the above objective function is shown below:

**Regression Loss Function**: $N$ is the batch size.

$$L_{loss} = l(r_{u,b}, r_{true}) = Mean(\{l_1, ..., l_N\}^T), l_n = (r_{u,b} - r_{true})^2.$$

**Classification Loss Function**:The negative log likelihood loss.

$$L_{loss} = -log(r_{u,b})$$

Following are detailed discussions regarding the impact of regression and classification on the final model's performance, as well as a discussion of the impact of replacing the various components on the final model's performance.

### 3.2.6 Implementation

As shown in Figure 3.10, after we have pre-trained the t-GNN with all the user-business pairs in the YELP dataset, we will then show how to construct the system with the pre-trained t-GNNs. As part of the application phase, we will describe in the following steps:

1. As shown in Figure 3.11, we will implement pretrained t-GNN and pretrained DistilBERT to extract both user and business spatio and temporal features and combine them with the existing identical feature as the final user and item embeddings.

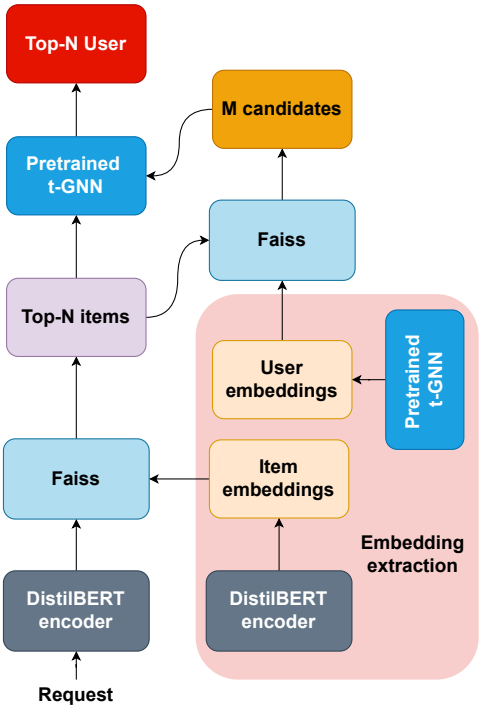



Figure 3.10: Query-driven Graph-based Recommender System

2. Given that a user initiates a query to the system, the pretrained DistilBERT will extract the sentence embedding from the query.

3. Faiss then matches the query embedding with all the item embedding in the database based on the threshold N.

4. Afterward, FAISS compares the matched Top-N business embedding directly with the user embedding. We match 1000 nearest users for each Top-N Items, primarily because: 1. It is time-consuming to compare each top-N item directly with all users using the pre-trained t-GNN. 2. The pre-trained t-GNN has an unavoidable bias that causes many popular users and items to always appear on the recommendation list, which will result in the system losing its diversity.

5. Finally, as the shown in figure 3.12. Following validation of proximity users, we forward Top-M candidate users and Top-N items to pre-trained t-GNN's linear layers for final evaluation to filter out the Top-N users that are the most appropriate to the

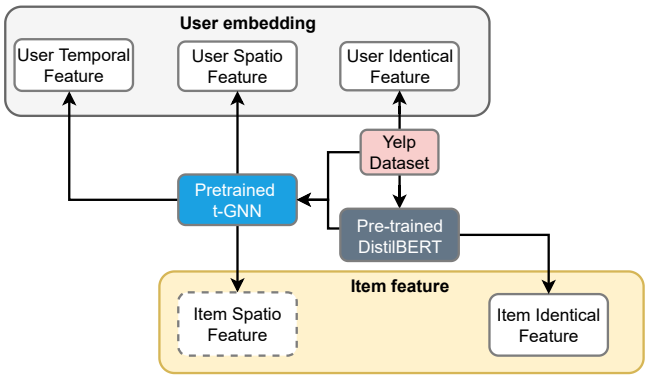



Figure 3.11: Components of user and item embeddings

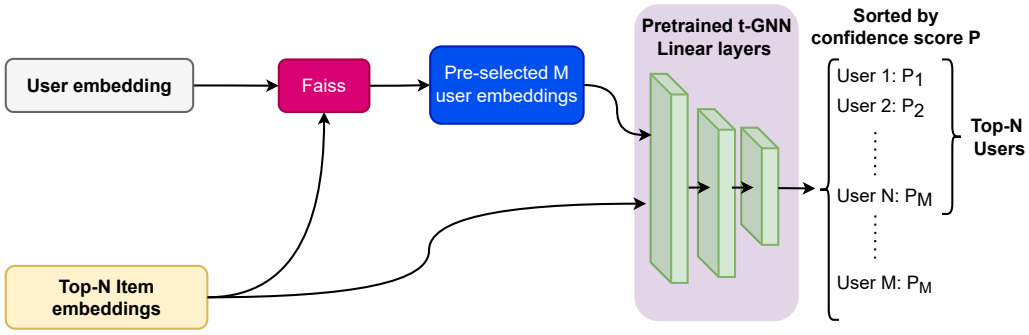



Figure 3.12: Two-stage user filtering based on Faiss and pretrained t-GNN linear layers

request based on the confidence score of each user-item pairs. (Note: The essence of this step is to use the t-GNN linear layer parameterized by the relational prediction task to evaluate the confidence value of the direct association between the user and the item. Our empirical experiment in section 4.2.3 demonstrates that the relevance of users further filtered by t-GNN is better than the effect of Faiss or t-GNN alone.)

Next we will demonstrate the functionality of the system in a demo. Let's just say we send the following request to the system: **"I like Pizza"**. The system would respond the following user based on the request.

Based on above results, most of the users met the condition that the keyword "pizza" was included in all the businesses they viewed or visited.

User name: Gene
---
The resturant that the user visited:
American (New), American (Traditional), Restaurants, Burgers
Restaurants, Event Planning & Services, Modern European, Pizza, Personal Chefs, Italian, American (New)

Figure 3.13: Sample1

User name: Martin
---
The resturant that the user visited:
Restaurants, Steakhouses, American (New), American (Traditional)
Restaurants, Burgers, Sandwiches, Pizza, Cafes, Wine Bars, Breakfast & Brunch, Nightlife, Pubs, Bars

Figure 3.14: Sample2

User name: James
---
The resturant that the user visited:
Barbers, Beauty & Spas, Hair Salons, Hair Stylists
Nightlife, Pubs, Restaurants, Bars
Nightlife, Bars, Pubs, Food, Breweries
Restaurants, Seafood, Sushi Bars, Steakhouses
Restaurants, Food, Beer, Wine & Spirits, American (Traditional)
Mexican, Lounges, Restaurants, Bars, Nightlife
Restaurants, Nightlife, American (Traditional), Cocktail Bars, Gastropubs, Bars
French, American (Traditional), Restaurants
Modern European, Wine Bars, Mediterranean, Tapas Bars, Moroccan, Bars, Nightlife, Restaurants
Pizza, Restaurants, Food, Italian

Figure 3.15: Sample3

# Chapter 4

# Evaluation

The goal of the QDG system is to provide the very content at the very moment. Therefore necessitates storing comprehensive features for each user. However, the fact is that most users' information could be absent or incomplete due to various obstructions. To achieve accurate user targeting without relying on existing user data. We proposed the t-GNN model to separately analyze user interest distribution and interest change tendency from spatial and temporal perspectives. To validate the rationality of our strategy, we investigate and analyze the impact of the t-GNN model on QDG. In the experimental session, we first demonstrate all evaluation metrics for the experiments. After investigating the t-GNN model with the corresponding ablation study, an evaluation of the QDG system is carried out through simulation.

## 4.1 t-GNN

### 4.1.1 Evaluation Metrics

To evaluate the t-GNN model, we used accuracy as the evaluation criterion for user-item relationship prediction and AUC score as the criterion for correct positive and negative class classification. The number of parameters is used to measure the model cost-effectiveness (the larger the number of parameters, the more costly the model computation)

### 4.1.2 Performance Comparison

In this section, we compare our t-GNN moedl with the following inductuve learning-based models with t-GNN on Yelp test data. Recurrent Multi-Graph Neural Networks (sRGCNN) [23],Inductive graph matrix completion (IGMC) [47] and Random-walk-based GCN (PinSage) [45] are compared with our t-GNN as shown in Table 4.1.

| model | Accuracy | AUC score | Parameters |
|---|---|---|---|
| sRGCNN | 0.44±0.03 | 0.68±0.03 | $11.8 * 10^5$ |
| PinSage | 0.39±0.02 | 0.57±0.02 | $35.1 * 10^5$ |
| IGMC | 0.45±0.01 | 0.69±0.01 | $11.8 * 10^5$ |
| t-GNN(ours) | **0.47±0.01** | **0.71±0.01** | $11.8 * 10^5$ |
| — *w/o* spatio component | 0.42±0.01 | 0.64±0.01 | $11.7 * 10^5$ |
| — *w/o* temporal component | 0.40±0.01 | 0.58±0.01 | $3.2 * 10^3$ |

Table 4.1: Model comparison

Our t-GNN model essential consists of a spatial component and temporal component. We independently analyze the effect of both components under the training parameter set. The experimental result demonstrates that either component can not perform well without a comprehensive analysis of the user's behaviour. As shown in Fig. 4.1, the experimental results of t-GNN overscore other models in the Yelp dataset in terms of parameter size, accuracy and AUC score.

### 4.1.3 Regression VS Classification

After comparing the temporal and spatial analysis modules. We investigated how the model should be trained, since the user-business relationships themselves can be directly categorized into specific scores by the classification task, or directly predicted to the exact score by the regression task itself.

We selected 100 test user-business pairs to observe the difference between classification and regression. In the above Figure 4.1 , we can observe that the prediction values of the model trained by regression are generally distributed around the plural 3-4 (because the user ratings in our data are mostly perfect), but the model loses the ability to identify low-rated businesses and lacks diversity in order to fit the data.

However, when the classification task is used to train the model, the model prediction scores are uniformly distributed, which greatly improves the robustness of the model and the ability to judge the degree of business recommendation.

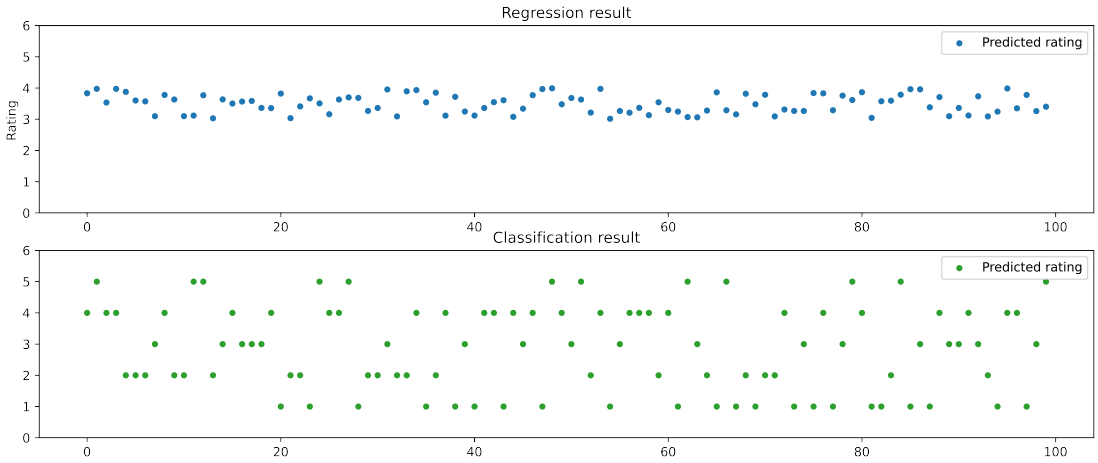



Figure 4.1: Regression VS Classification

### 4.1.4 Ablation Study: Spatio Component Comparison

In this section, we investigate different spatio component's impact on the t-GNN model. As shown in Table 4.2, We compared the GCN [33], GAT [39], SAGE [10], RGCN [36] spatio analysis component on Yelp validation dataset over 5 runs. We use relationship rating prediction accuracy and AUC scores as our measuring metrics. By comparing inductive learning-based GNN-style models, RGCN slightly outperforms the other models with the same parameter setting on the Yelp dataset.

| Model | Accuracy | AUC score | Parameters |
|---|---|---|---|
| GCN | 0.36±0.01 | 0.55±0.01 | $3.2 * 10^3$ |
| SAGE | 0.38±0.0 | 0.57±0.01 | $3.2 * 10^3$ |
| GAT | 0.39±0.01 | 0.57±0.01 | $3.2 * 10^3$ |
| RGCN | **0.40±0.02** | **0.58±0.01** | $3.2 * 10^3$ |

Table 4.2: Spatio component comparison

### 4.1.5 Ablation Study: Temporal Component Comparison

To investigate additional temporal analysis component's effectiveness on t-GNN model, based on Table 4.3, We design and compare the MLP, Bi-LSTM [15] temporal analysis module, as well as with the multi-head attention related temporal analysis module on Yelp validation dataset over 5 runs. We use relationship rating prediction accuracy and AUC score as our measure. Experiments show that multi-head attention is more applicable to extract user temporal features at the same parametric scale.

| Model | Accuracy | AUC score | Parameters |
|---|---|---|---|
| RGCN (Baseline) | 0.40±0.01 | 0.58±0.01 | $3.2 * 10^3$ |
| RGCN + 1 layer MLP | 0.42±0.02 | 0.60±0.01 | $5.9 * 10^5$ |
| RGCN + 2 layer MLP | 0.41±0.01 | 0.61±0.01 | $11.8 * 10^5$ |
| RGCN + 1 layer Bi-LSTM | 0.45±0.01 | 0.69±0.01 | $5.9 * 10^5$ |
| RGCN + 2 layer Bi-LSTM | 0.44±0.01 | 0.68±0.01 | $11.8 * 10^5$ |
| RGCN + 1 layer Attention | 0.46±0.02 | 0.70±0.02 | $5.9 * 10^5$ |
| RGCN + 2 layer Attention | **0.47±0.01** | **0.71±0.01** | $11.8 * 10^5$ |

Table 4.3: Temporal component comparison

## 4.2 QDG

### 4.2.1 Evaluation Metrics

To measure whether QDG can match the most relevant users, we designed two query templates, which are "I like $\{item\}$?" as sample1 and "Can I see the picture of $\{item\}$?" as sample2.

Here we will use RRS (receiver relevance score) (Query $\cap$ Receivers) to measure whether the key items appearing in the query overlap with the items that the recommended user has viewed. The intersection score (IS) represents the mean value of the intersection of users matched by the current Item and users matched by other items, and the response time (RT) represents the system practicality.

### 4.2.2 Simulation Test: Two-Stage QDG

As shown in Fig. 4.2. We set threshold N to 20 in the query-to-item process, meaning we would match the 20 most similar items to the query. To filter users with the pre-trained t-GNN in the item-to-user process, we first pre-screen 1000 feasible users with Faiss for each Top-20 item (Our empirical experiments show that pre-screening of user embedding with Faiss can improve the overall speed and efficiency of the system twice as much and has no significant impact on the final results). Then the pre-train t-GNN filters the top-100

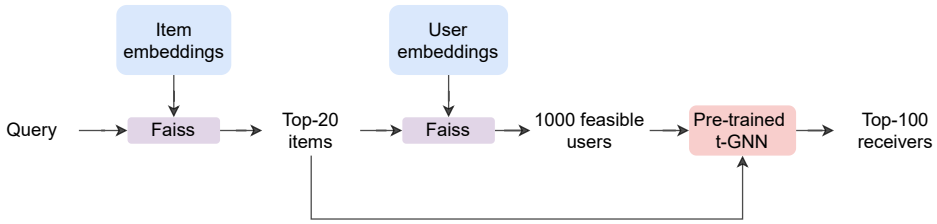



Figure 4.2: Two-stage recommendation simulation test process

receivers from the 1000 feasible users as the final recommended receivers. In order to test the degree of association between the recommended Top-M users and requests, we did the following simulation tests based on the YELP food dataset.

As shown in Table 4.4, here we have selected some common cuisines as evaluation criteria, from which we can see that the model can maintain a fairly high relevance rate for North American native cuisines among their corresponding populations. We also observe that for each item, the Intersection score of matched users is maintained in a certain range, reflecting users' diversity for different queries. However, for some Asian or Italian cuisines, these options are not completely accurate. The most likely reason for this is that the Yelp dataset only samples four major cities in the United States and does not have a rich sampling of businesses in other countries or regions, resulting in a single structured business and a high repetition rate of users' historical business browsing. We also separately test pre-trained t-GNN, sRGCN, PinSage and IGMC embedded in QDG on the final simulation test with sample1 and sample2. As shown in Fig. 4.4, t-GNN outperforms other models in our simulation test.

### 4.2.3 Ablation Study: QDG Recommendation Process

We also implement the ablation study on the final user filtering pipeline by separately removing the Faiss or pretrained t-GNN to test the system reaction time and RRS value. As shown in table 4.4, we can observe that t-GNN-only can seriously slow down the response time of the system and return a higher overlap-rate user group. However, the effect of Faiss alone will significantly reduce the RRS score since the distance-based measurement lack comprehension of user-business relationships. Therefore, our system adopts a combination of t-GNN and Faiss in the item-to-business mapping stage to achieve a more balanced effect.

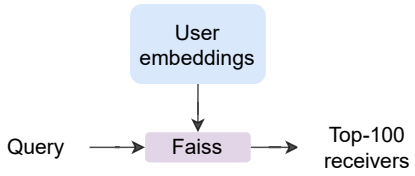



Figure 4.3: One-stage recommendation simulation test process

### 4.2.4 Two-Stage QDG VS One-Stage QDG

Finally, to further validate the rationality of the two-stage recommendation in the application process. After extracting user embedding from pertrained t-GNN. As shown in Figure 4.3, we modify the recommendation process into the one-stage pipeline and compare it with the two-stage recommendation in our final simulation test. Our empirical experiment in Table 4.5 demonstrate the effectiveness of two-stage of recommendation on RRS score.

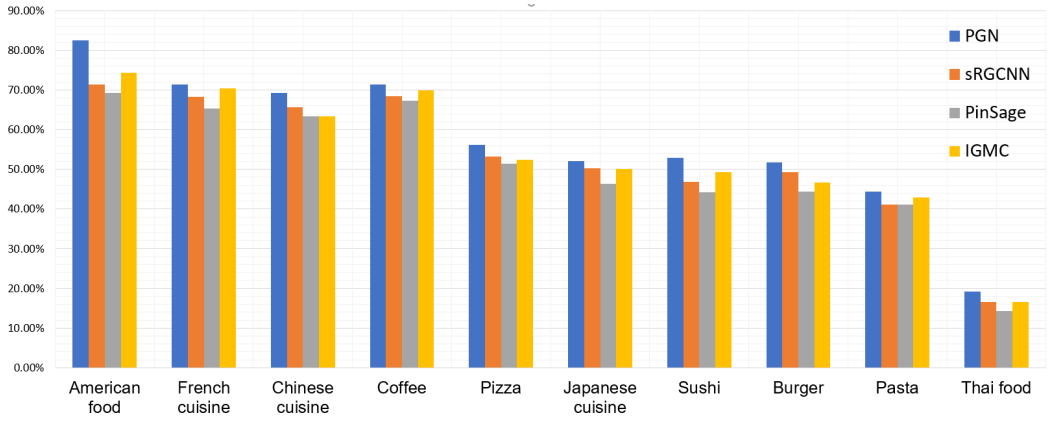



Figure 4.4: The mean RRS of t-GNN, sRGCN, PinSage and IGMC in Sample1 and Sample2 respectively.

| Item | Sample1 RRS | Sample2 RRS | IS | RT |
|---|---|---|---|---|
| American food | 83.54% | 82.47% | 34.44% | 19s |
| - *w/o* t-GNN | 72.34% | 70.58% | **25.31%** | **2s** |
| - *w/o* Faiss | **86.75%** | **85.68%** | 46.23% | 75s |
| French cuisine | 72.54% | 71.36% | 21.93% | 19s |
| - *w/o* t-GNN | 62.18% | 59.34% | **16.28%** | **3s** |
| - *w/o* Faiss | **78.36%** | **76.57%** | 38.43% | 77s |
| Chinese cuisine | 71.54% | 69.23% | 26.14% | 19s |
| - *w/o* t-GNN | 63.67% | 61.28% | **18.94%** | **2s** |
| - *w/o* Faiss | **75.68%** | **74.97%** | 39.34% | 76s |
| Coffee | 70.42% | 71.34% | 57.28% | 19s |
| - *w/o* t-GNN | 64.35% | 63.12% | **45.31%** | **2s** |
| - *w/o* Faiss | **75.26%** | **74.39%** | 67.28% | 74s |
| Pizza | 58.45% | 56.24% | 31.23% | 19s |
| - *w/o* t-GNN | 52.13% | 50.37% | **24.69%** | **2s** |
| - *w/o* Faiss | **64.91%** | **62.78%** | 41.55% | 79s |
| Japanese cuisine | 56.24% | 52.13% | 24.67% | 19s |
| - *w/o* t-GNN | 51.23% | 49.83% | **29.68%** | **2s** |
| - *w/o* Faiss | **65.35%** | **63.28%** | 46.23% | 73s |
| Sushi | 54.63% | 52.89% | 21.58% | 19s |
| - *w/o* t-GNN | 44.58% | 41.42% | **30.03%** | **2s** |
| - *w/o* Faiss | **58.64%** | **55.22%** | 36.15% | 74s |
| Burger | 53.76% | 51.74% | 32.23% | 19s |
| - *w/o* t-GNN | 42.39% | 41.99% | **25.32%** | **2s** |
| - *w/o* Faiss | **56.34%** | **53.79%** | 41.67% | 71s |
| Pasta | 48.84% | 44.36% | 15.37% | 19s |
| - *w/o* t-GNN | 42.28% | 40.13% | **10.72%** | **2s** |
| - *w/o* Faiss | **52.87%** | **51.34%** | 23.15% | 77s |
| Thai food | 21.43% | 19.28% | 3.12% | 19s |
| - *w/o* t-GNN | 14.37% | 16.15% | **2.69%** | **2s** |
| - *w/o* Faiss | **29.54%** | **25.63%** | 7.89% | 75s |

Table 4.4: Model simulation test.

| Item | Strategy | Sample1 RRS | Sample2 RRS | IS | RT |
|---|---|---|---|---|---|
| American food | - *Two-Stage* | **83.54%** | **82.47%** | 34.44% | 19s |
| | - *One-Stage* | 71.28% | 69.88% | **23.36%** | **2s** |
| French cuisine | - *Two-Stage* | **72.54%** | **71.36%** | 21.93% | 19s |
| | - *One-Stage* | 61.55% | 58.39% | **15.21%** | **3s** |
| Chinese cuisine | - *Two-Stage* | 71.54% | 69.23% | 26.14% | 19s |
| | - *One-Stage* | 61.76% | 59.57% | **16.22%** | **2s** |
| Coffee | - *Two-Stage* | **70.42%** | **71.34%** | 57.28% | 19s |
| | - *One-Stage* | 62.87% | 60.09% | **43.75%** | **3s** |
| Pizza | - *Two-Stage* | **58.45%** | **56.24%** | 31.23% | 19s |
| | - *One-Stage* | 50.18% | 51.46% | **21.44%** | **2s** |
| Japanese cuisine | - *Two-Stage* | **56.24%** | **52.13%** | 24.67% | 19s |
| | - *One-Stage* | 49.89% | 48.65% | **27.14%** | **2s** |
| Sushi | - *Two-Stage* | **54.63%** | **52.89%** | 21.58% | 19s |
| | - *One-Stage* | 40.16% | 39.17% | **27.51%** | **2s** |
| Burger | - *Two-Stage* | **53.76%** | **51.74%** | 32.23% | 19s |
| | - *One-Stage* | 44.58% | 44.37% | **24.17%** | **3s** |
| Pasta | - *Two-Stage* | **48.84%** | **44.36%** | 15.37% | 19s |
| | - *One-Stage* | 47.32% | 44.59% | **8.31%** | **2s** |
| Thai food | - *Two-Stage* | **21.43%** | **19.28%** | 3.12% | 19s |
| | - *One-Stage* | 16.87% | 17.53% | **3.42%** | **2s** |

Table 4.5: Two-Stage recommendation vs One-Stage recommendation.

# Chapter 5

# Conclusion

## 5.1 Conclusion

As a recommendation system serving user experience, its measurement criteria not only exist in offline model metrics, but the user's experience as a subjective measure has become an essential metric for evaluating recommendation systems. On this premise, we believe that since users' subjective evaluations can be used to measure the system, users' subjective preferences can also be used as essential indicators for system optimization.

We represent the user's subjective preferences in the form of a text like request, whose semantics will be used as important features to guide the system's recommended results. As the main objective of this system, we design and implement a query-driven recommendation system based on the request-to-user process. Our preliminary analysis found that most common recommendation systems are built on either user-item or user-user, with very few directly built on the user's request. Consequently, we developed a two-stage Query-centric personalized graph-based recommendation system, which breaks down the request-to-user process into two steps, request-to-item and item-to-user.

In the first step of the request-to-item phase, the semantic features described by request and item are extracted sequentially by DistilBERT as Sentence encoder, and the Faiss algorithm achieves fast matching to locate the most relevant Top-N items with the request.

In the second stage of item-to-user mapping, we analyze the user-item interactions and the interaction order, taking into account the missing and biased user information and the dynamic growth of information. The general idea of this problem can be divided into two dimensions: temporal and spatial. First, the spatial dimension represents the explicit and

implicit association between users and items in the relationship graph, representing the distribution of users' preferences. And the temporal dimension is to analyze the user's behaviour and the trend of preference change according to the temporal relationship between the user and item interaction. Through both temporal and spatial requirements, we design the t-GNN model, which combines the inductive-learning-based RGCN and multi-head attention modules to analyze both temporal and spatial dimensions respectively, and at the same time can effectively explore the implicit relationships and potential behavioural trends of users in the relationship graph.

The whole model construction phase can be divided into two parts, the pre-training and application phases, respectively. The first one is pre-training the t-GNN model with the already YELP data by predicting the relationship task for each item-user pair. In the following application phase, the pre-trained t-GNN can evaluate the users interested in the item.

We compared different models from temporal and spatial perspectives in the first pre-training stage. For the first temporal analysis model, we mainly compared the performance of LSTM and multi-head Attention model in user temporal sequence prediction, and the experiments found that multi-head Attention was better than LSTM as a temporal analysis module. Finally, we compare the t-GNN model with some typical relational prediction GNN-style models, and our t-GNN model outperforms other models in the relational prediction task of the YELP dataset.

Finally, we performed some tests using information from the YELP dataset in the food and entertainment area in the application phase. The experimental results show that our query-centric recommendation system can successfully return the corresponding user based on the input requests.

## 5.2 Future work

Here we believe that there is still a lot of room for system improvement, including seasonality, timeliness and policy friendliness as a recommendation system. For these features we need to create additional interfaces to perform a series of filters on our existing information. The system should also be able to use the user's geographical and age information for some filtering. In addition, the system itself should have the ability to extract some key information from the request. For example, additional extraction is needed for geographic information, notes, etc., so that we can better capture the information expected by users when we recommend them based on the request.

For the algorithm side, we need more comprehensive relational database in pre-training the model. If we want to expand the model completely into the field of food, only the information of North America is not enough, we also need a lot of geographical features of other countries, so that the model can better resolve the correlation between user requests and different businesses. If our goal is only social software, diverse relationships are also helpful for model pre-training, because the spatial analysis module of t-GNN, RGCN, is better than the common GAT and SAGEConv for analyzing diverse relationships.

# References


[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[2] Chumki Basu, Haym Hirsh, William Cohen, et al. Recommendation as classification: Using social and content-based information in recommendation. In *Aaai/iaai*, pages 714–720, 1998.

[3] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.

[4] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

[5] Jacob Devlin and Ming-Wei Chang. Open sourcing bert: State-of-the-art pre-training for natural language processing. *Google AI Blog*, 2, 2018.

[6] Wenshuang Du, Zhongxian Wang, and Junjie Ai. Fast search of massive high-dimensional vectors similarity. In *2020 3rd International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)*, pages 67–70. IEEE, 2020.

[7] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

[8] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pages 417–426, 2019.

[9] J Grau. Personalized product recommendations: Predicting shoppers' needs, 2009.

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[11] Eui-Hong Han and George Karypis. Feature-based recommendation system. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 446–452, 2005.

[12] G Ei Hardy and TD Rogers. A generalization of a fixed point theorem of reich. *Canadian Mathematical Bulletin*, 16(2):201–206, 1973.

[13] Jason Hartford, Devon Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh. Deep models of interactions across sets. In *International Conference on Machine Learning*, pages 1909–1918. PMLR, 2018.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

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

[16] Fatima EL Jamiy, Abderrahmane Daif, Mohamed Azouazi, and Abdelaziz Marzak. The potential and challenges of big data-recommendation systems next level application. *arXiv preprint arXiv:1501.03424*, 2015.

[17] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

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

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

[20] Rebecca Lengnick-Hall, Donald R Gerke, Enola K Proctor, Alicia C Bunger, Rebecca J Phillips, Jared K Martin, and Julia C Swanson. Six practical recommendations for improved implementation outcomes reporting. *Implementation Science*, 17(1):1–8, 2022.

[21] Chao Ma, Wenbo Gong, José Miguel Hernández-Lobato, Noam Koenigstein, Sebastian Nowozin, and Cheng Zhang. Partial vae for hybrid recommender system. In *NIPS Workshop on Bayesian Deep Learning*, volume 2018, 2018.

[22] Benjamin Marlin, Richard S Zemel, Sam Roweis, and Malcolm Slaney. Collaborative filtering and the missing at random assumption. *arXiv preprint arXiv:1206.5267*, 2012.

[23] Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. *Advances in neural information processing systems*, 30, 2017.

[24] Hyun-ju Noh, Min-jung Kwak, and In-goo Han. Handling incomplete data problem in collaborative filtering system. *Journal of Intelligence and Information Systems*, 9(2):51–63, 2003.

[25] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[26] Bryan Perozzi, Rami Al-Rfou, and Steven 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, 2014.

[27] Alec Radford, Jeffrey Wu, Dario Amodei, Daniela Amodei, Jack Clark, Miles Brundage, and Ilya Sutskever. Better language models and their implications. *OpenAI blog*, 1:2, 2019.

[28] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[29] Nils Reimers, Iryna Gurevych, Nils Reimers, Iryna Gurevych, Nandan Thakur, Nils Reimers, Johannes Daxenberger, Iryna Gurevych, Nils Reimers, Iryna Gurevych, et al. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 671–688. Association for Computational Linguistics, 2019.

[30] NI Reshetko, VL Belozerov, SP Vakulenko, PV Kurenkov, EA Zmeškal, EA Chebotareva, IA Solop, EL Kuzina, MA Vasilenko, and VY Barashyan. Company product policy for the production and sales of electric vehicles. *Transportation Research Procedia*, 55:356–361, 2021.

[31] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[32] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, 2001.

[33] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[34] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[35] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[36] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[37] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009, 2009.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[40] Alexandre Viejo, Jordi Castella-Roca, and Guillem Rufián. Preserving the user's privacy in social networking sites. In *International Conference on Trust, Privacy and Security in Digital Business*, pages 62–73. Springer, 2013.

[41] Jun Wang, Arjen P De Vries, and Marcel JT Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 501–508, 2006.

[42] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[43] Feng Xue, Xiangnan He, Xiang Wang, Jiandong Xu, Kai Liu, and Richang Hong. Deep item-based collaborative filtering for top-n recommendation. *ACM Transactions on Information Systems (TOIS)*, 37(3):1–25, 2019.

[44] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In *IJCAI*, volume 17, pages 3203–3209. Melbourne, Australia, 2017.

[45] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

[46] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

[47] Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. *arXiv preprint arXiv:1904.12058*, 2019.

[48] Tehseen Zia and Usman Zahid. Long short-term memory recurrent neural network architectures for urdu acoustic modeling. *International Journal of Speech Technology*, 22(1):21–30, 2019.