UNIVERSITY OF CALIFORNIA

Los Angeles

# Graph Neural Networks Nodes Classcification of Recommendation Franchisee Location

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Applied Statistics and Data Science

by

Hao Zheng

2024

ABSTRACT OF THE THESIS

# Graph Neural Networks Nodes Classcification of Recommendation Franchisee Location

by

Hao Zheng
Master of Applied Statistics and Data Science
University of California, Los Angeles, 2024
Professor Yingnian Wu, Chair

This thesis presents a study on location optimization for franchisee restaurants using Graph Neural Network (GNN) models, namely GCN, GAT, and GraphSAGE. The research employs these models to analyze geographic coordinates and other relevant data to predict optimal franchise locations. By incorporating real-world data such as Yelp reviews, census information, and city demographics, the study attempts to model the significant factors that influence the success of franchise locations. The primary contribution of this work is the development of a tool that aids market research teams in making informed decisions about where to establish new restaurant outlets and optimizing location selection through advanced data analytics and machine learning techniques. Among the three models, the GraphSAGE model performed best. It achieved a loss of 0.48, an accuracy of 77%, and an ROC score of 0.78, outperforming the other models across various assessments.

The thesis of Hao Zheng is approved.

University of California, Los Angeles

2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

# CHAPTER 1

# Introduction

Franchising has emerged as a prevalent economic expansion strategy due to its consistent quality standards, reasonable pricing mechanisms, and established operational practices. [Mic03] This is particularly evident in the restaurant industry, where starting from scratch often entails significant expenditures, including rent, salaries for contractual staff, and time to fine-tune operational strategies. Critical considerations for a new restaurant include identifying peak business hours, balancing pricing strategies, and developing popular menu items. These elements represent just a few of the factors involved. For investors in the restaurant sector, initiating a new establishment carries considerable risks. At this juncture, franchising contracts start to capture investors' interest, as they mitigate many startup challenges. By adopting a franchising model, investors can avoid most of the complexities associated with launching a new restaurant. The franchise owner provides all necessary support and resources, leaving investors to focus primarily on selecting the optimal location for their new establishment.

Choosing the right location is arguably the most critical and challenging aspect of the investment process. Eugênio J. S. Bitti describes that location is a major entrepreneurial choice that has a huge effect on firm performance in retailing. [BFL19] An inappropriate location choice can lead to the quick closure of the establishment. Consequently, most franchises include a department dedicated to market research, tasked with analyzing various factors to determine the most advantageous site for a new outlet. The market research department often faces the daunting task of making multiple location decisions within a limited timeframe. This paper proposes the development of a location optimization tool to aid the market research department in this decision-making process.

To accurately model real-world scenarios, this study employs several Graph Neural Network models, including GCN, GAT, and GraphSAGE. These models use real-world geographic coordinates as positional information (POS) to define nodes and establish a spatial framework. The relationships between nodes are based on proximity, with physical distances calculated and represented in the models. Each node is weighted according to its proximity to others. After evaluating the trade-offs between efficiency and the potential loss of critical information, this paper selects the Nearest Neighbors (KNN) method to define the edge indices. Seokho Kang, validated the construction the kNN graph before passing it through the graph neural network. [Kan21] By comparing outcomes across various assessments of three models, to get the outstanding performance model. Then use that model to c Ideally, the examined model can predict whether a given location is likely to be recommended or not for a franchising restaurant, based on selected features.

# CHAPTER 2

# Data

## 2.1 Source

The dataset was compiled from three separate sources: Yelp,census.gov, and city-data.com. [Yel22a][Yel22b][Bur22][Dat22] The primary data consists of Yelp's top recommended restaurants in Los Angeles for 2022, alongside the lowest-rated restaurants as reported by Yelp for the same year. Due to Yelp's reporting limitations, this dataset comprises 354 entries, featuring columns such as Location (ZIP code), Rating (ranging from 0 to 5), and Number of Reviews. Results indicating whether a restaurant is recommended are recorded in the Recommendation column as 'Yes' or 'No'.

The secondary dataset, sourced from census.gov, includes 323 entries detailing ZIP Codes and Average Income for 2022. The third dataset, obtained from city-data.com, covers all ZIP codes in and near Los Angeles. It includes 328 entries with data on total population, median age, gender distribution, business profit or loss, and median gross rent. By aggregating data from all sources based on ZIP code, the final dataframe was constructed.

## 2.2 Data Processing

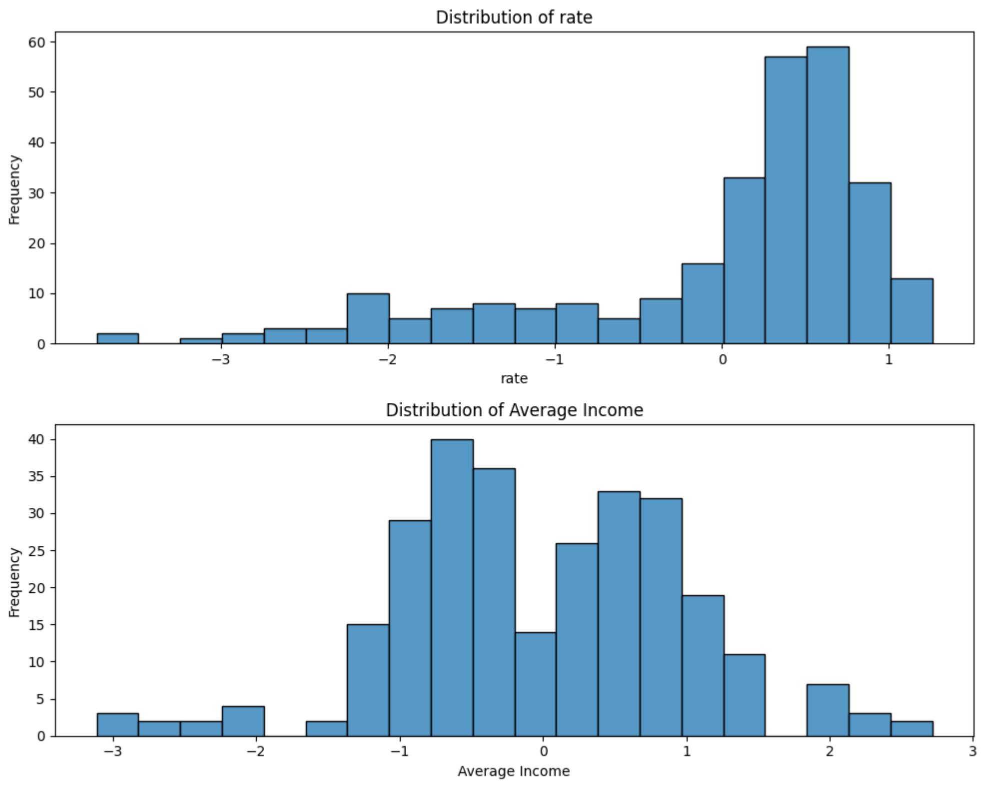


Figure 2.1: Histogram After Logarithmic Transformation

Initial data cleaning involved removing rows with missing or zero values to prevent potential issues with algorithm performance. The analysis began by addressing feature redundancy within ZIP codes, which could potentially skew the learning patterns of the Graph Neural Network (GNN) models. To minimize this, random noise with a scale of 0.2 was introduced into the dataset. Subsequent analysis involved plotting the distribution of each variable to identify columns requiring transformation; notably, the Rating and Average Income columns exhibited significant left skewness. A logarithmic transformation was applied to these columns to correct this issue, resulting in improved distributions.

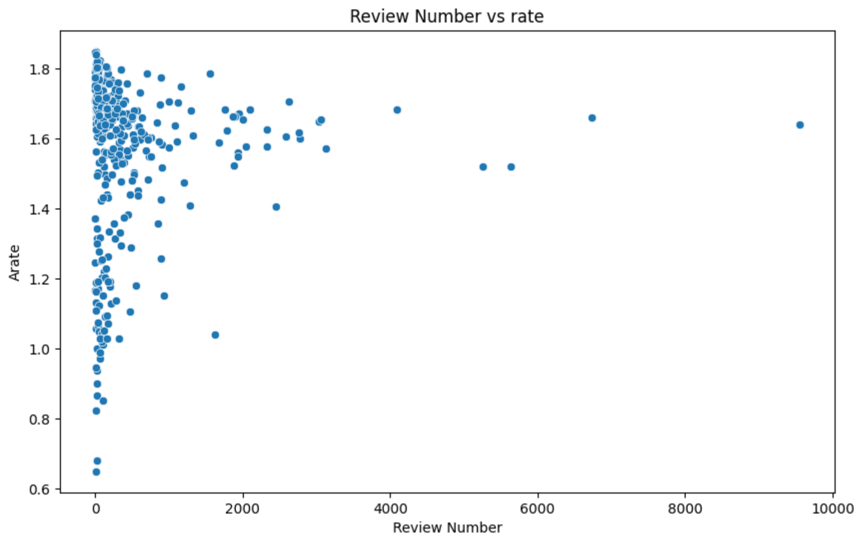


Figure 2.2: Testing the Relationship between Review Number and Rating

Further exploratory data analysis included plotting the relationship between Review Number and Rating, which indicated that higher ratings were not necessarily correlated with a higher number of reviews. Considering the challenges investors might face in obtaining review counts for franchised restaurants, and to simplify the model, the Review Number was subsequently excluded from the dataset. Additionally, after reviewing the distributions of the total population, total females, and total males, it was decided to retain only the total population to simplify the features.

A correlation analysis revealed a substantial relationship between Profit or Loss from Business and Average Income ($r = 0.62$), which is logical as salary often contributes significantly to business outcomes. However, given the data's broad representation of all business types within a ZIP code, and to avoid multicollinearity, the Profit or Loss from Business feature was excluded.

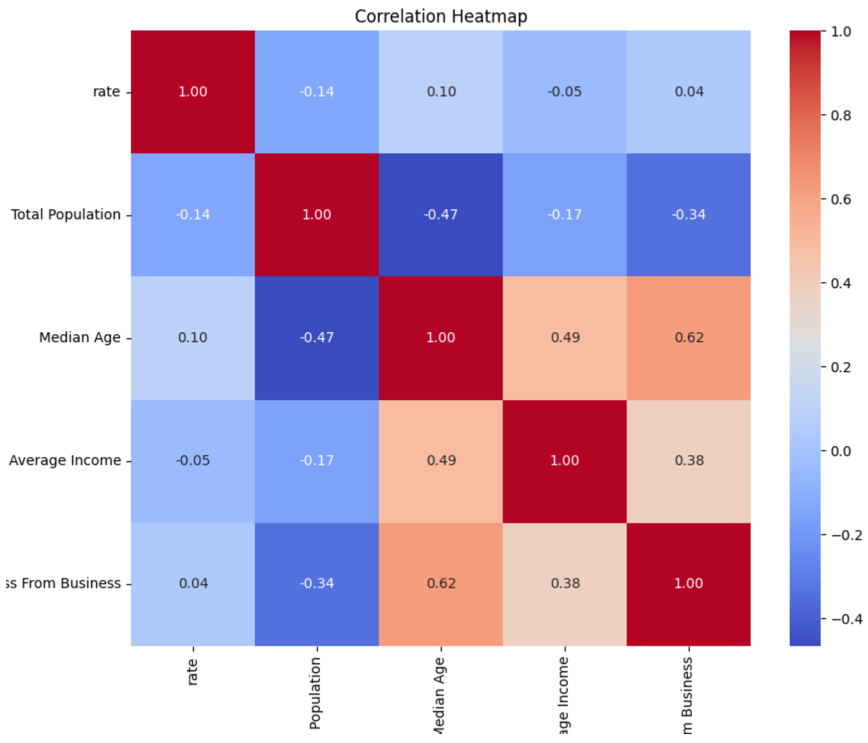


Figure 2.3: Correlation Test

Finally, the dataset was split into an 80% training set and a 20% testing set to prevent information leakage. To ensure feature values were on a comparable scale, standardization was applied to both the training and testing sets using the scaler fitted on the training set. The final distribution plot confirmed that all features were standardized and ready for modeling.

# CHAPTER 3

# Methology

## 3.1 Established the Space

### 3.1.1 Transofrmation into Tensors

In this project, we apply PyTorch tensors as the foundational data structure for Graph Neural Network (GNN) models. The method of tensor transformation generalizes the form of vectors through gradient calculations, allowing the automatic differentiation system to compute backward passes with high efficiency, which demonstrates its compatibility with GNN models. Here are the compositions and reasons behind the designed tensors:

Position Tensors: The latitude and longitude columns, which represent spatial location data points, are converted to torch.float tensors. These tensors help to locate each node in space. The use of real-world latitude and longitude allows for direct visualization in real-world space, providing clear visual insights to investors.

Feature Tensors: Selected features, including 'Rating', 'Total Population', 'Median Age', 'Average Income', and 'Median Gross House Rent', are converted into torch.float tensors. These features were chosen due to their acceptable correlation (r = 0.1 - 0.6) among them. This balance helps simplify the models to prevent overfitting problems and provides investors with a number of interesting inputs for investigation.

Target Tensors: The 'Recommendation' column, used as the target variable for model training, is transformed into a torch.long tensor, the required format for the loss function in classification tasks.

### 3.1.2 K-Nearest Neighbors Algorithm

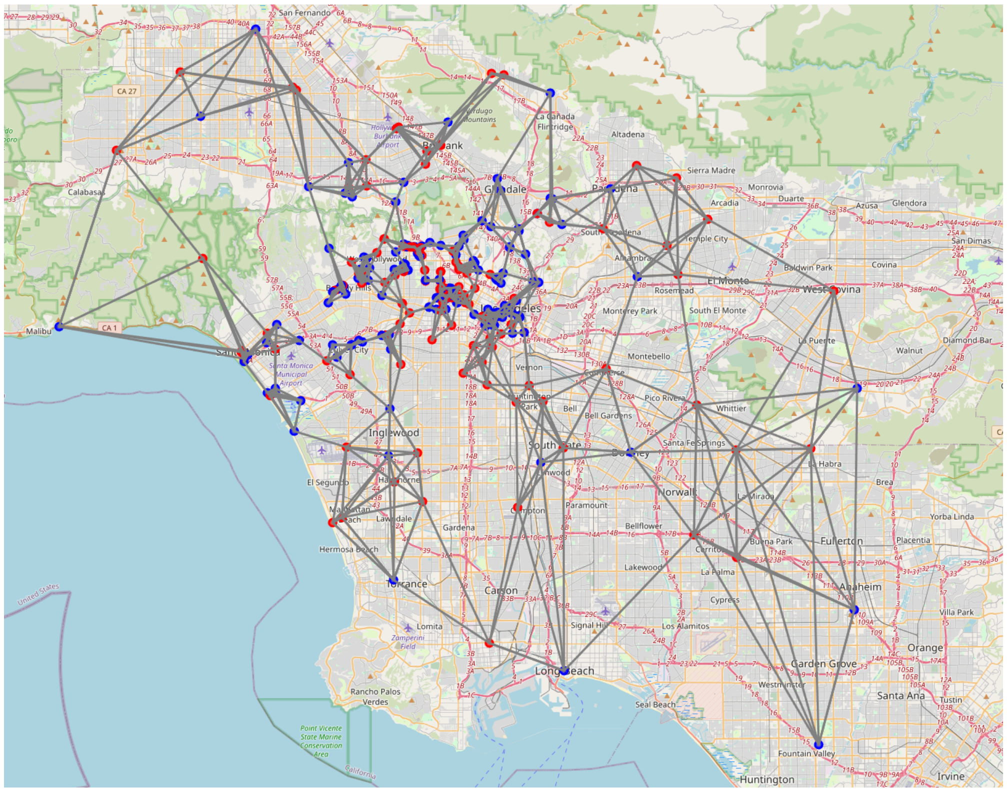


Figure 3.1: Connecting Nodes with K-NNs Method

This project applies the K-Nearest Neighbors (KNN) algorithm rather than directly connecting all the nodes with a certain threshold. [Kan21] Due to the clustering pattern of the point distribution, directly connecting nodes based on weight determined by the distance between two nodes results in either excessively numerous lines or sparse lines that focus on the clustering points. As a more optimized algorithm, KNN considers distance and establishes edges that reflect the data's clustering.

The K-NN algorithm typically employs non-parametric methods for classification. In this project, we primarily use this method to create the graph structure. K-NN evaluates an object's neighbors, classifies the object based on the most frequent class among its k nearest neighbors, and applies this understanding in both local and global contexts. This project opted for the 'ball_tree' implementation within the 'NearestNeighbors' function (k = 5) to query multidimensional spaces for a point's nearest neighbors. This approach helps avoid self-connections and prevents excessive edge connections.

To calculate edge weight, we used an exponential decay function based on pairwise distances. The function's negative exponent implies that interaction intensity between two points increases as their distance decreases. Therefore, nodes that are closer together, such as restaurants, likely exert greater mutual influence. The result of the K-NN edges looks like efficiently connects the nodes with the consideration numbers of edges and their weight depends on the length of distances. The Blue dots represent the recommended restaurants, while the red dots are not recommended.

## 3.2 Graph Convolutional Network (GCN)

### 3.2.1 Graph Theory

Graph theory examines the properties and interactions of graphs. In this theory, a graph G is defined as an ordered pair G = (V, E), where V represents a set of nodes in a dataset, and E is a set of edges that act as connections or relationships between entities. This concept is applied in real-world applications such as social networks, where individuals are represented by nodes and their social interactions by edges. [New10] This demonstrates the possibility of applying the same framework to individual restaurants, where the nodes are the restaurants and the edges represent their local interactions.

In graph representation, the adjacency matrix $A$ indicates the graph's connectivity. In an unweighted graph, $A_{ij} = 1$ if an edge exists between nodes and 0 otherwise. In a weighted graph, $A_{ij}$ represents the weight of the connection between nodes i and j, typically based on the distance between the two points. [Wes01] After constructing the adjacency matrix, the feature matrix X captures the attributes of each node. Each row in matrix X corresponds to a node, and the columns represent the node's features. Translating graphs into structured forms allows them to be processed by Graph Convolutional Networks (GCNs).

### 3.2.2 Convolutional Neural Networks(CNNs)

Convolutional Neural Networks (CNNs) are capable of processing Euclidean data, such as images and audio, by utilizing convolutional layers that operate on local patches of an image. CNNs can apply edge detection learned at one location to other areas in the image, enabling them to aggregate simple patterns detected by earlier layers into more complex structures. [KSH12]

However, CNNs may encounter challenges when applied directly to graph data, primarily because the unfixed shape of graphs results in a variable number of neighbors, which diverges from the fixed pixel neighborhoods required by CNNs. [BBL17] Additionally, translational invariance does not always hold in graph structures. [13] These limitations have led to the development of techniques that more effectively process the data's intrinsic properties, including the rise of graph-specific neural network architectures such as Graph Convolutional Networks (GCNs).

### 3.2.3 Spectral Graph Theory

Spectral Graph Theory explains the properties of graphs in relation to the eigenvalues and eigenvectors of matrices such as the graph Laplacian. The graph Laplacian, $L$, can be defined as $L = D - A$, where $D$ represents the diagonal matrix containing the degree of each vertex, and $A$ is the adjacency matrix of the graph. Typically, in the Laplacian matrix, the eigenvalues form the spectrum of the graph, and the eigenvectors reveal the graph's structure. [Chu97]

By applying eigen decomposition to the graph Laplacian, expressed as $L = U\Lambda U^T$, where $U$ is the matrix of eigenvectors and $\Lambda$ is the diagonal matrix of eigenvalues, we obtain the Graph Fourier Transform. In this transform, a signal $x$ defined on the vertices of a graph can be transformed using the equation $\hat{x} = U^T x$, which simplifies convolution operations into straightforward element-wise multiplications.Graph convolution can then be computed as $x * y = U((U^T x) \odot (U^T y))$, where $x$ and $y$ represent the graph signals, $*$ denotes the convolution, and $\odot$ represents the element-wise multiplication. Utilizing full spectral convolution to interpret the data forms the foundation of Graph Convolutional Networks (GCNs). [SNF13]

### 3.2.4 GCNs Backgounrd

Graph Convolutional Networks shift from spectral-based to spatial graph convolutions. Spatial graph convolutions work directly in the spatial domain by aggregating features from the local neighborhoods of each node. They operate similarly to CNNs but leverage the sparsity of graphs, gathering information without the overhead of spectral transformations. [BZS14]

The operation of Graph Convolutional Networks is represented by the equation:

$$h_i^{(l+1)} = \sigma\left(W^{(l)} \cdot \sum_{j \in N(i)} \frac{1}{c_{ij}} h_j^{(l)} + b^{(l)}\right).$$

Here, $h_i^{(l)}$ is the feature vector of node $i$ at layer $l$. $N(i)$ denotes the neighbors of node $i$, $W^{(l)}$ and $b^{(l)}$ are parameters of the layer, $c_{ij}$ is a normalization constant, and $\sigma$ is a non-linear activation function such as ReLU. The use of edge weights allows the model to capture the importance of different nodes within a neighborhood, enhancing the model's ability to learn the complex structures of the input data. [HYL17]

By eliminating the necessity for the Laplacian's eigendecomposition, spatial convolutions enable the application of deep learning across multiple domains, such as network analysis and recommendation systems. A weighted Graph Convolutional Network (GCN) model could be an effective approach for classifying nodes in restaurant recommendation systems.

Table 3.1: GCNs Layer Configuration

| **Type** | **Input Features** | **Output Features** | **Parameters** |
|---|---|---|---|
| GCNConv | 5 | 32 | Uses edge weights |
| BatchNorm1d | - | - | Normalizes 32 features |
| Activation | - | - | - |
| Dropout | - | - | 0.65 |
| GCNConv | 32 | 16 | Uses edge weights |
| BatchNorm1d | - | - | Normalizes 16 features |
| GCNConv | 16 | 2 | Uses edge weights |

### 3.2.5 GCNs Model Architecture

During fine-tune the GCNs model, the GCNs weighted model is designed as a total of three convolution layers. The model transforms the input from five selected features into a higher-dimensional space of 32 dimensions to construct the representation of each node's attributes in the network. Subsequently, the model reduces the dimensionality first to 16 and then to 2, allowing it to learn the relevant patterns for the classification task. During each convolutional layer in the GCN, batch normalization was applied to stabilize the learning process. Theoretically, if the feature distribution has a mean of zero, batch normalization can reduce the internal covariate shift, thus saving processing time. ReLU activation, added after each batch normalization, enables the model to learn complex patterns. This approach may also reduce the likelihood of vanishing gradient issues. To prevent the model from overfitting, a dropout rate of 0.65 was implemented after each ReLU activation. This randomly deactivates some subsets of neurons, enabling the network to learn more robust features, thus enhancing the model's generalization capabilities.

To achieve optimal performance, selecting an appropriate optimizer is critical to efficiently reducing loss and ensuring stable training. As a final decision, the Adam optimizer (r = 0.01) was used due to its adaptive learning rate capabilities. The Adam optimizer maintains a per-parameter learning rate, potentially improving the handling of sparse data from the restaurant network and automatically adjusting the learning rate based on the averages of the gradients. Additionally, we integrated L2 regularization ($Value = 5 * 10^{-4}$) into the optimization process. This adds a penalty equal to the square of the magnitude of weights to the loss, aiming to prevent potential overfitting. Providing a balance between the strength of the regularization and the model's ability to comprehend complex patterns.

We chose CrossEntropyLoss as the loss function, which is likely suitable for node classification problems with multiple classes. This function combines LogSoftmax and Negative Log Likelihood Loss, making it appropriate for binary classification—in this case, classifying restaurants as recommended or not recommended.

### 3.2.6 GCNs Result

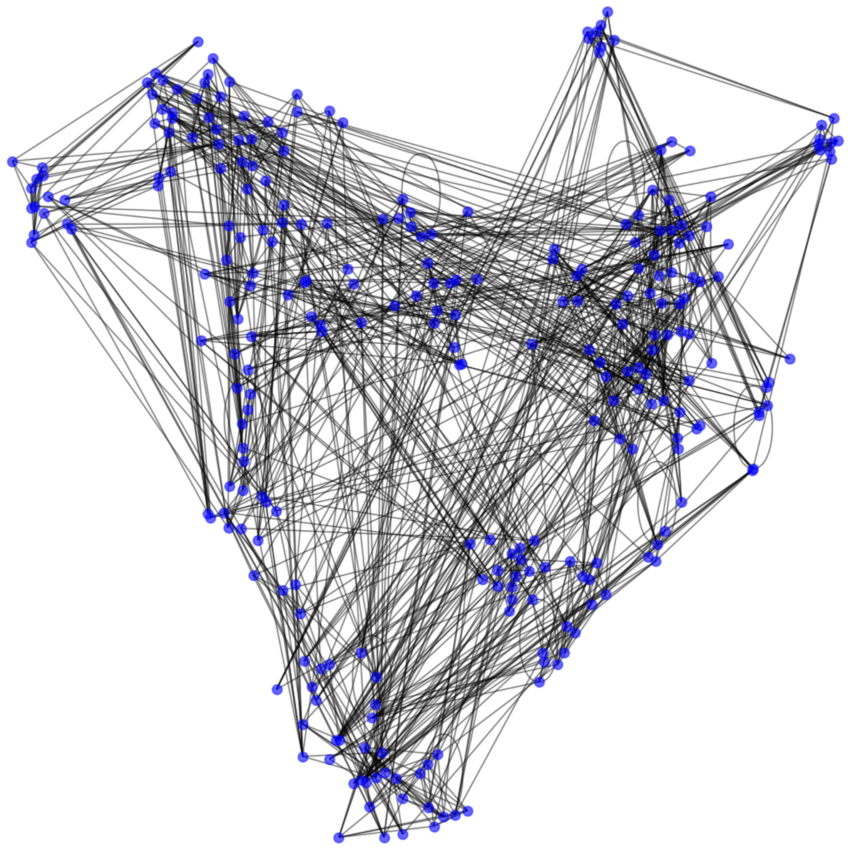


Figure 3.2: t-SNE visualization for GCNs model

The t-SNE visualization plot shows distinct clustering patterns of the nodes, suggesting that the GCN model can effectively group similar data nodes based on proximity. Additionally, we observed dense areas that highlight differences in feature similarity among the nodes, indicating that the GCN model captures complex interrelations. Some nodes appear to be outliers, which the GCN model struggles to group due to their extreme values. Overall, the lack of high overlap in the t-SNE plot suggests that the parameter settings of the GCN models are appropriately fine-tuned.

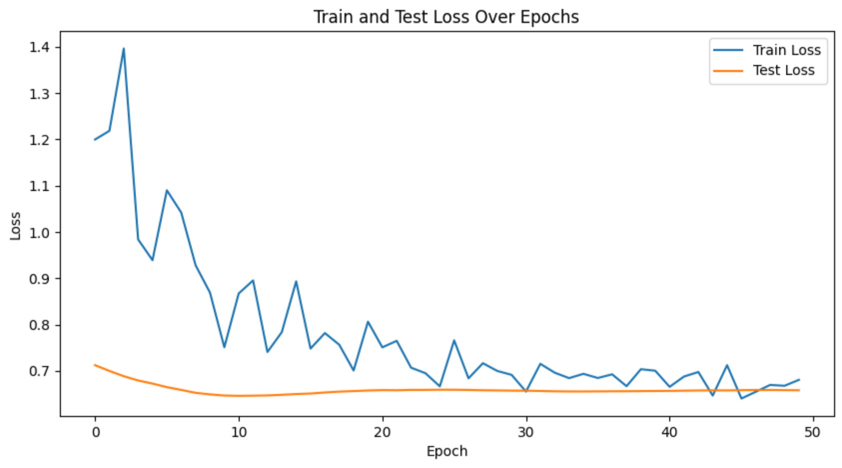


Figure 3.3: GCNs Loss

During the 50 epochs, the training loss began with a high value but rapidly decreased after a few epochs. This indicates that the model was quickly learning the patterns from the input features. After this initial rapid decline, the training loss continued to decrease, albeit at a slower rate, with some minor peaks. It might show the model is improving over time. The testing loss started at a lower value compared to the initial training loss and maintained a decreasing trend throughout the epochs. The testing loss exhibited a period of stability with small variance throughout the training process, suggesting that the model is well-trained on unseen data. This also suggests that the model may have reached its optimal performance on the test dataset relatively early. Since neither the testing loss nor the training loss diverged, and both decreased simultaneously, this suggests that there may not be a serious overfitting problem. The converging losses indicate that the model is stable, and additional training epochs may not further improve the model's performance.

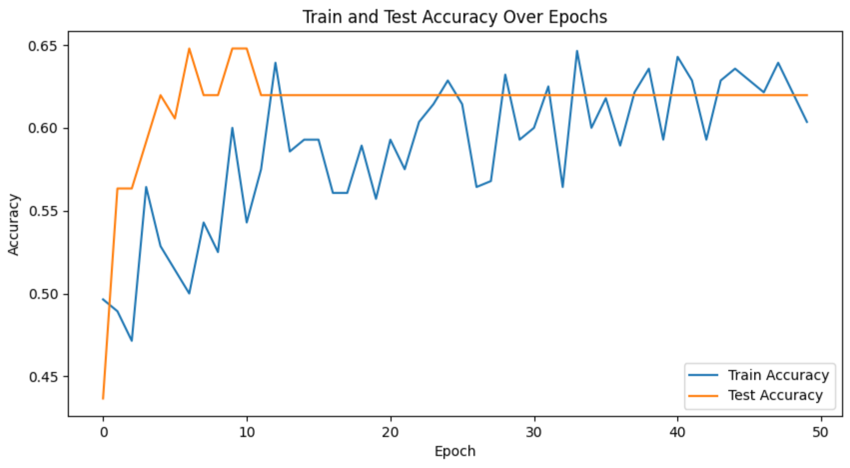


Figure 3.4: GCNs Accuracy

The training accuracy starts around 50% and quickly increases to above 60% within the first 10 epochs. This corresponds with the training loss, indicating that the model is learning patterns during the first 10 epochs. However, following this peak, the training accuracy exhibits volatility without a clear trend, ultimately stabilizing around 60%. Similarly, the testing accuracy initially starts at around 45% and experiences a sharp rise within the first few epochs. It peaks around epoch 10 and then stabilizes at 62% thereafter. This might indicate that the model has likely reached its capacity to handle unseen data. Therefore, it appears that both the testing and training data achieve stable performance after certain epochs. Additionally, the lack of a downward trend in testing accuracy suggests that the model might not have serious overfitting issues.

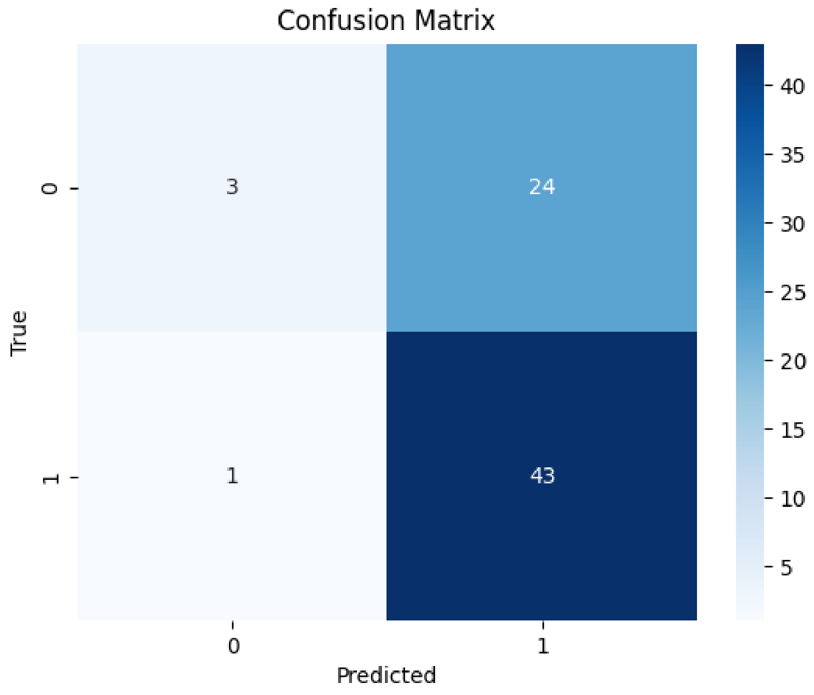


Figure 3.5: GCNs Confusion matrix

The confusion matrix indicates a preference in the model to predict restaurants as recommendable, with only three true negatives compared to 24 false positives, suggesting a potential bias towards positive predictions. Additionally, the AUC plot shows a score of 0.55, which is slightly better than random guessing. The trend continues to move closely following the diagonal line. This indicates that with the weighted GCN models tuned there is still a struggle, in effectively grasping the patterns of both categories.

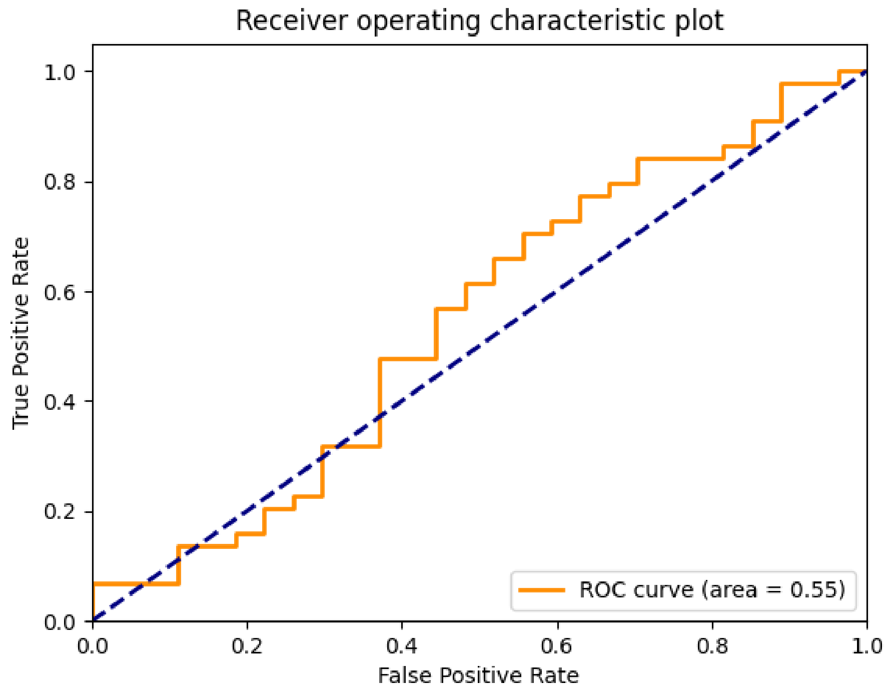


Figure 3.6: GCNs ROC Curve

GCNs may face challenges in treating all neighboring nodes uniformly, as they differentiate them based on fixed weights such as node degrees. However, in restaurant scenarios, node characteristics could be more complex and may not be accurately represented by fixed weights.

## 3.3 Graph Attention Networks (GATs)

### 3.3.1 GATs Backgounrd

To find the solution of fixed weight in Graph Convolutional Networks (GCNs), one solution is the attention mechanism used in Graph Attention Networks (GATs). It begins by calculating the attention coefficient, which can be defined as:

$$e_{ij} = a(\mathbf{W}h_i, \mathbf{W}h_j)$$

Here, $e_{ij}$ represents the raw attention score between nodes $i$ and $j$, and $h_i$ and $h_j$ denote the features of those nodes. $\mathbf{W}$ is the weight matrix which is applied to the nodes.

After the calculation of attention coefficients, they are normalized using the softmax function across all nodes connected to a particular node. The formula for this normalization is:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}(i)} \exp(e_{ik})}$$

The updated feature $h_i'$ of node $i$ is a blend of its neighbors' features, defined by:

$$h_i' = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}\mathbf{W}h_j\right)$$

Here, $\sigma$ represents a non-linear function such as LeakyReLU, with each neighbor's contribution weighted by its attention coefficient. [Vel18] This entire process of the attention mechanism allows the network to focus more on important neighbors.

Based on the previous formula, when adding the attention heads, the new equation will be revised as:

$$h_i' = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{k} W^k h_j\right) \tag{3.1}$$

Here, $K$ denotes the number of heads, and $\alpha_{ij}^k$ and $W^k$ represent the attention coefficients and weight matrices for the $k$-th attention head. Dropout was used during the calculation of the weights $\alpha_{ij}$, which helps introduce variation and prevents the model from becoming overly fixated on specific patterns, thereby reducing the risk of overfitting.

### 3.3.2 GATs Model Architecture

In the GAT model the initial GATConv layer uses 8 attention heads enabling the model to explore patterns within the input features. Similar to the GCNs model Batch Normalization

Table 3.2: GATs Layer Configuration

| **Type** | **Input Features** | **Output Features** | **Parameters** |
|---|---|---|---|
| GATConv | 5 | 8 | heads=8,dropout=0.6 |
| BatchNorm1d | - | - | Normalizes features from 64 features |
| Activation | - | - | - |
| Dropout | - | - | 0.3 |
| GATConv | 32 | 16 | heads=1, concat=False, dropout=0.6 |

and ReLU activation functions are included in the layers to enhance learning stability and facilitate the identification of patterns. Moreover the GAT model employs two dropout rates for managing overfitting; a dropout rate of 0.6 is implemented before the first GATConv layer and, after the second one while a dropout rate of 0.3 is utilized throughout these two layers. This strategy helps in mitigating overfitting by randomly omitting different portions of the features during training.

In the GAT model, we continue to use the Adam Optimizer with a learning rate of 0.005 and weight decay set at $1X10^{-4}$ to address sparse gradients and to regularize the model. The CrossEntropyLoss function is also applied as the loss criterion for the binary classification task.

### 3.3.3 GATs Result

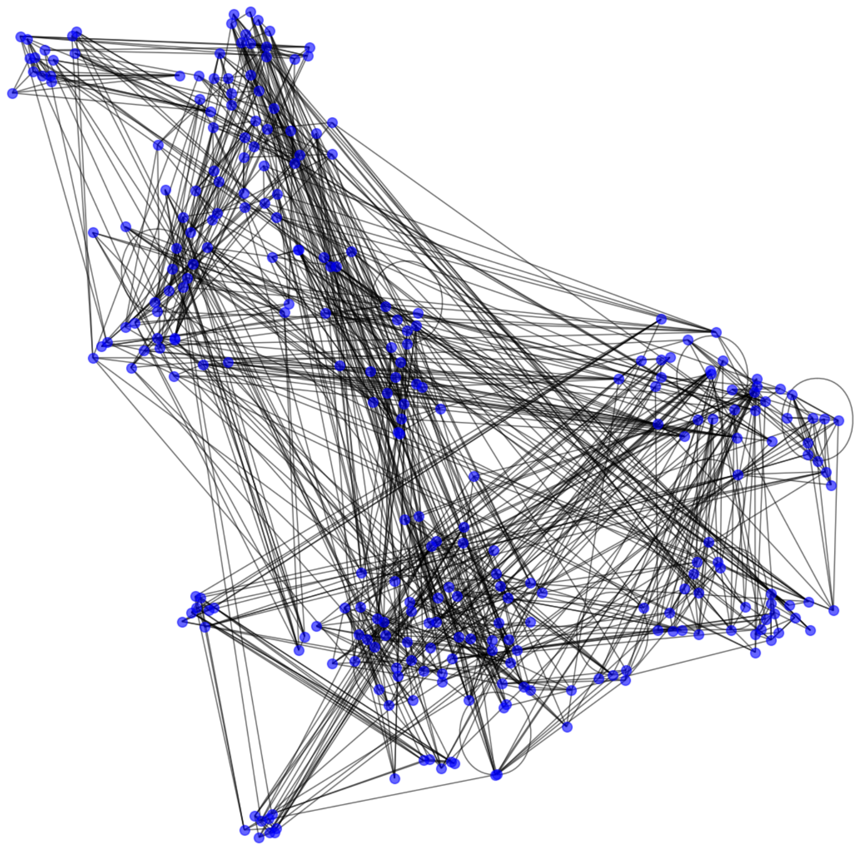


Figure 3.7: t-SNE visualization for GATs model

The t-SNE visualization displays some clusters that are dispersed, with overlapping regions indicating that the GAT model might effectively group similar nodes. However, the boundaries between the categories in the binary classification are not clearly defined. Additionally, a few nodes appear to be separated from the main clusters, which might represent unique data points that do not share strong similarities with the majority of nodes. Despite this, the limited amount of overlap among points suggests that the GAT model has likely achieved a good level of parameter tuning.

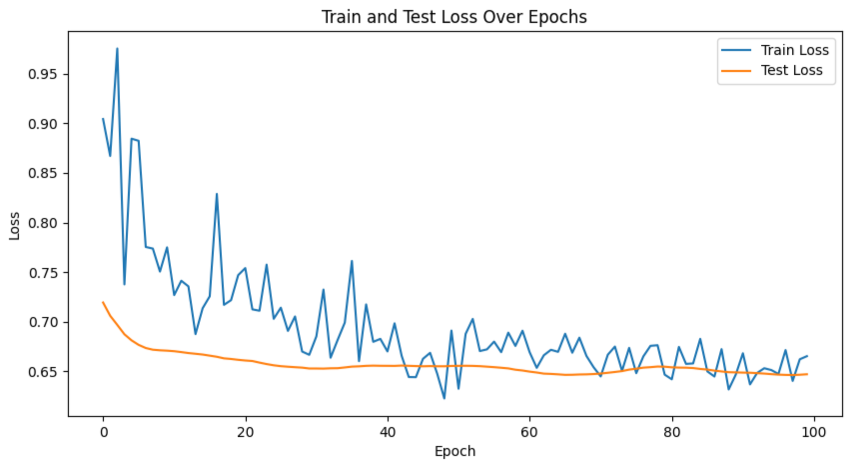


Figure 3.8: GATs Loss

The training loss starts at approximately 0.90, and quickly decreases to around 0.75 within the first 10 epochs, indicating that the GAT model has rapidly adapted to the patterns from the training features. Subsequently, it shows fluctuations but generally follows a downward trend until it stabilizes around 0.70. In contrast, the test loss begins at a lower value compared to the training loss, around 0.73, and unlike the training loss, it appears to be more stable and becomes consistent after epoch 20. This suggests that the model is both learning the training features and generalizing well to unseen data. Furthermore, the narrow gap between the training and testing losses indicates that the L2 regularization and dropout may have effectively mitigated potential overfitting problems.

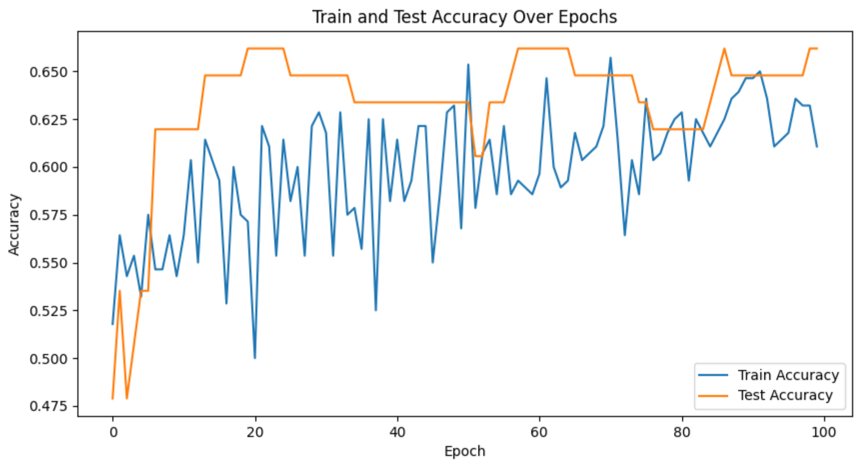


Figure 3.9: GATs Accuracy

From the accuracy plot, both training and testing accuracies increase from the beginning during the first few epochs. The training accuracy increases to around 58%, with the testing accuracy rising to approximately the same level. After the initial epochs, the training accuracy fluctuates, peaking up and down around 63%, suggesting that the training data may contain some complex structures. Observing the testing accuracy, it demonstrates a more stable improvement after the first few epochs and continuously appears to follow the trend of the training accuracy. Ultimately, the testing accuracy stabilizes at around 66%.

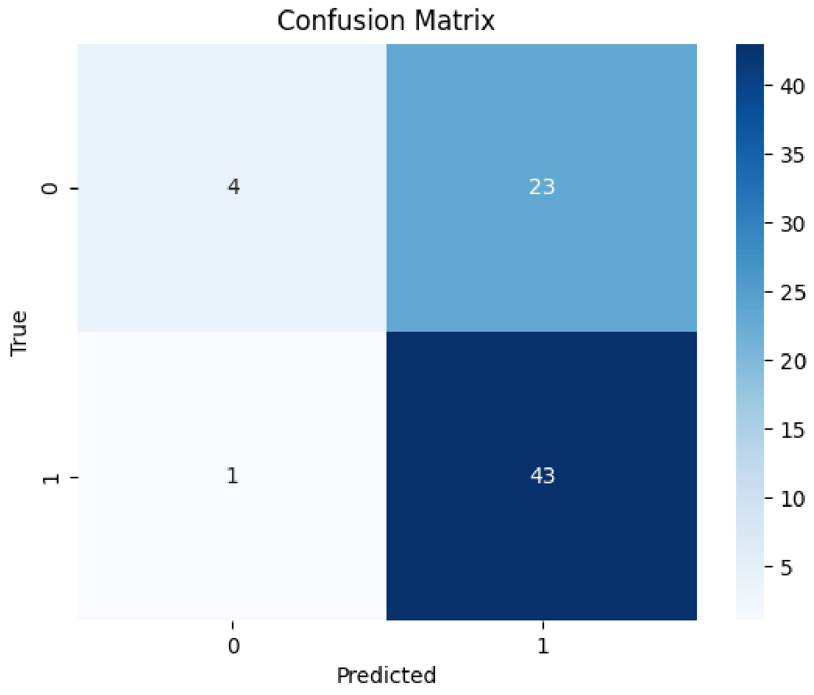


Figure 3.10: GATs Confusion Matrix

The confusion matrix for the GAT model still shows a tendency to predict nodes as recommendable restaurants. However, the presence of 4 true negatives and 23 false positives indicates that the GAT model, with its adjustable weights, has improved the bias towards positive classifications seen in the GCNs model. From the ROC plot, the ROC curve, which progresses from the bottom left to the top right, appears to perform better compared to the GCNs model. Combined with an area under the AUC of 0.61, this suggests that in approximately 61% of cases, the GATs model can correctly differentiate between recommendable and non-recommendable restaurants.

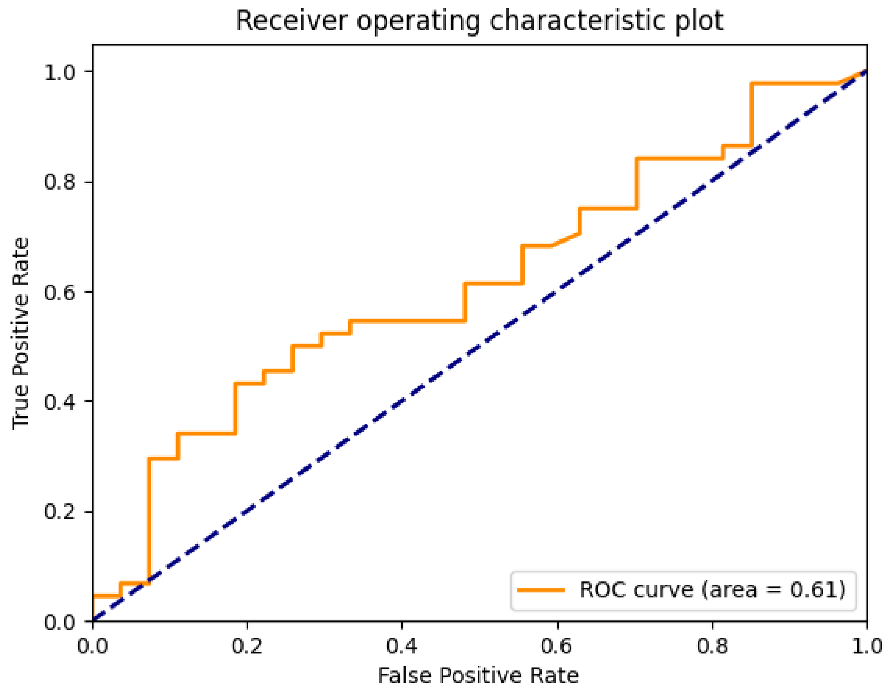


Figure 3.11: GATs ROC Curve

We find that the use of flexible weights with attention heads appears to successfully enhance the model's ability to learn complex patterns from the selected input features. However, there are still limitations observed in the GATs model's performance in classification tasks. These limitations may stem from the requirement of the GATs model for sufficient data to effectively train these sophisticated models with multiple attention heads.

## 3.4 Graph Sample and Aggregate(GraphSAGE)

### 3.4.1 GraphSAGE Background

GraphSAGE introduces a novel neighbor sampling method that selects a constant number of neighbors for each node during the process. For a given node $v$, a fixed number $S$ of its neighbors is randomly chosen, and the node's new feature vector $h_v^{(k+1)}$ is computed by aggregating features from these sampled neighbors $\mathcal{N}_S(v)$ at layer $k$:

$$h_v^{(k+1)} = \sigma\left(W^{(k)} \cdot \text{AGGREGATE}_k\left(\left\{h_u^{(k)} \mid \forall u \in \mathcal{N}_S(v)\right\}\right)\right) \tag{3.2}$$

where $\sigma$ is a non-linear activation function, $W^{(k)}$ is the weight matrix at layer $k$, and AGGREGATE is a function that combines the features of the neighbors. [HYL17]

Unlike GCNs and GATs, GraphSAGE provides customized aggregation methods for various graph data and tasks, such as the aggregator, LSTM aggregator, and pooling aggregator, each serving different purposes. The GraphSAGE model can dynamically select the appropriate aggregation method based on the context. It also has the ability to generate node embeddings that can potentially enrich the training data. The feature vector for a node, $h_{\text{new}}$, is calculated by aggregating the feature vectors of its neighboring nodes, expressed as $h_{\text{new}} = \sigma(W \cdot \text{AGGREGATE}(\{h_u \mid u \in N(\text{new})\}))$. This aggregation uses a weighted sum with a trainable matrix $W$ and a non-linear activation function $\sigma$. [Zho20]

### 3.4.2 GraphSAGEs Model Architecture

Table 3.3: GraphSAGEs Layer Configuration

| **Type** | **Input Features** | **Output Features** | **Parameters** |
|---|---|---|---|
| SAGEConv | 5 | 64 | - |
| BatchNorm1d | - | - | Normalizes features 64 features |
| Dropout | - | - | p=0.5 |
| SAGEConv | 64 | 32 | - |
| BatchNorm1d | - | - | Normalizes features 32 features |
| SAGEConv | 32 | 2 | - |
| Log Softmax | - | - | Applied to output, dim=1 |

Within the first layer in the GraphSAGE model increases the input features into 64 dimensions. Then, Subsequent layers reduce the dimensions further—first to 32 and then to 2 for binary classification. This structure is designed to aggregate neighborhood features and capture local structural information around each restaurant's representatives.

Batch normalization is applied after the first and second layers to accelerate the learning process. Dropout, with a probability of 0.5, is also applied after the first and second SAGEConv layers to prevent overfitting. A ReLU function is used following each batch normalization to facilitate the learning of complex patterns from the input features.

The final output of the layers in the GraphSAGE model is processed by the log softmax function, designed to transform the output node features into log probabilities. This transformation facilitates the calculation of the CrossEntropyLoss. Similarly to the GAT model, the GraphSAGE model employs the Adam Optimizer ($r = 0.001$) with a weight decay ($value = 5X10^{-4}$), which is aimed at addressing sparse gradients and regularizing the model. The CrossEntropyLoss function is applied as the loss criterion for the binary classification task.

### 3.4.3 GraphSAGE Result

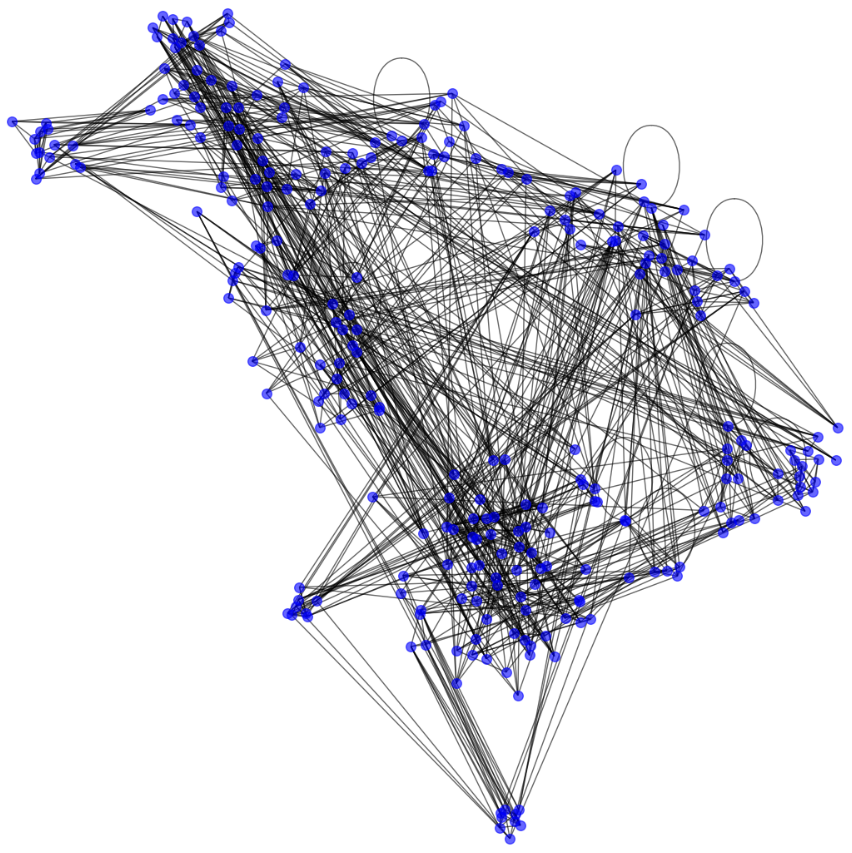


Figure 3.12: t-SNE visualization for GraphSAGE model

After investigating the t-SNE plot for the GraphSAGE model, it appears to depict a complex network structure with several connected clusters. Unlike with GCNs and GATs, the cluster points in the GraphSAGE model seem to be more clearly separated, which might indicate an improved capability of the model to classify the positive and negative outcomes for recommended restaurants. These results could suggest that the parameters and architecture of the GraphSAGE model are well-tuned. Although most nodes are densely packed, a few outstanding outliers might be caused by unusual features from the restaurants, warranting further investigation.

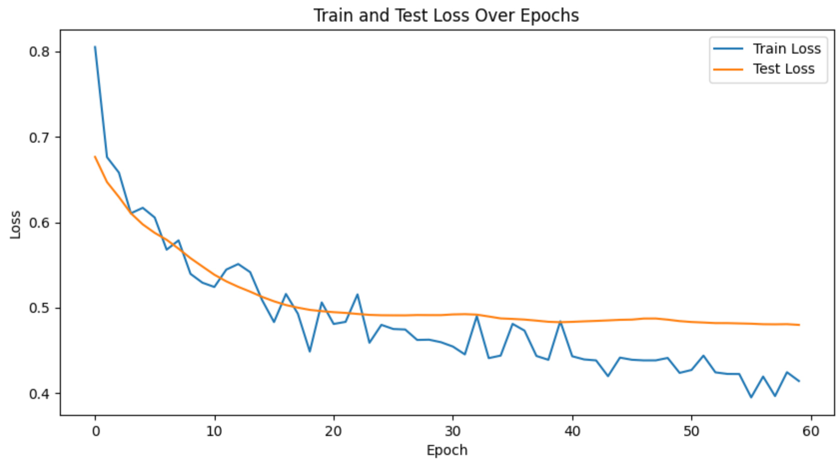


Figure 3.13: GraphSAGE Loss

Examining the loss change plot, there is a clear decline in both training and testing loss during the first 10 epochs. The training loss starts at around 0.8, and the testing loss starts at 0.7. While the training loss shows some variability, it generally displays a downward trend and seems to converge after epoch 55, reaching around 0.45. The testing loss mirrors the major changes in the training loss, exhibiting a more stable movement overall, with no upward trend. This indicates that the current parameter settings might successfully avoid serious overfitting issues.

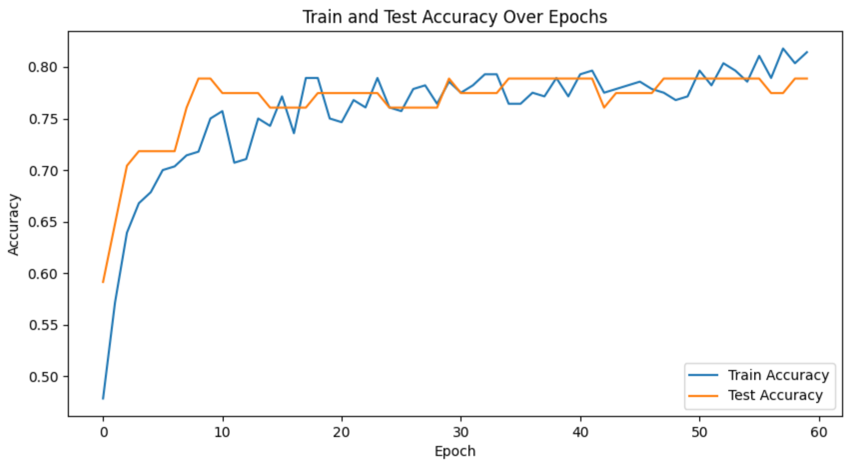


Figure 3.14: GraphSAGE Accuracy

From the accuracy plot, the training accuracy presents a sudden increase in the initial epochs, rising from approximately 55% to nearly 78% before the 10th epoch. It then maintains a relatively small magnitude of fluctuations compared to the GCNs and GATs models, with an overall upward direction, and finally stabilizes at around 80%. The testing accuracy also rapidly increases to approximately 75% within the first 10 epochs. After the 10th epoch, the testing accuracy shows more stable variability compared to the training accuracy, ultimately converging around 77%. The small gap between training and testing accuracies overall indicates that the model handles unseen data well and seems to avoid overfitting problems.

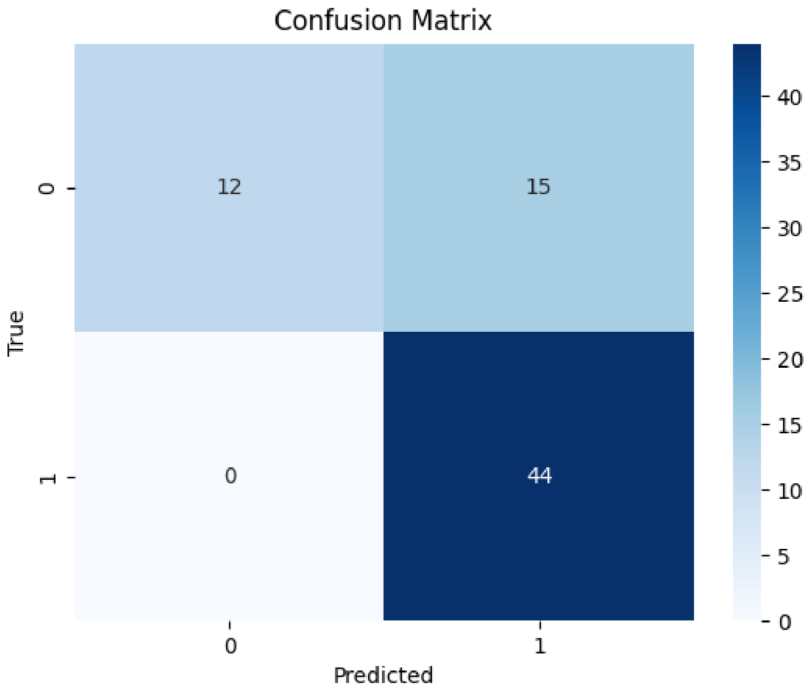


Figure 3.15: GraphSAGE Confusion Matrix

The GraphSAGE model appears to have improved its performance over the GCNs and GATs models, as evidenced in the ROC plot. The plot reveals that the model correctly predicted 12 non-recommendable restaurants, although 15 non-recommendable restaurants were incorrectly predicted. Despite more incorrect than correct predictions in this category, this still represents a significant improvement compared to the GCNs and GATs models. Notably, the GraphSAGE model achieved 44 correct predictions for recommendable restaurants, with zero incorrect predictions. This shows that GraphSAGE has not only gotten better at spotting negative outcomes but has also kept up its knack for predicting positive outcomes.

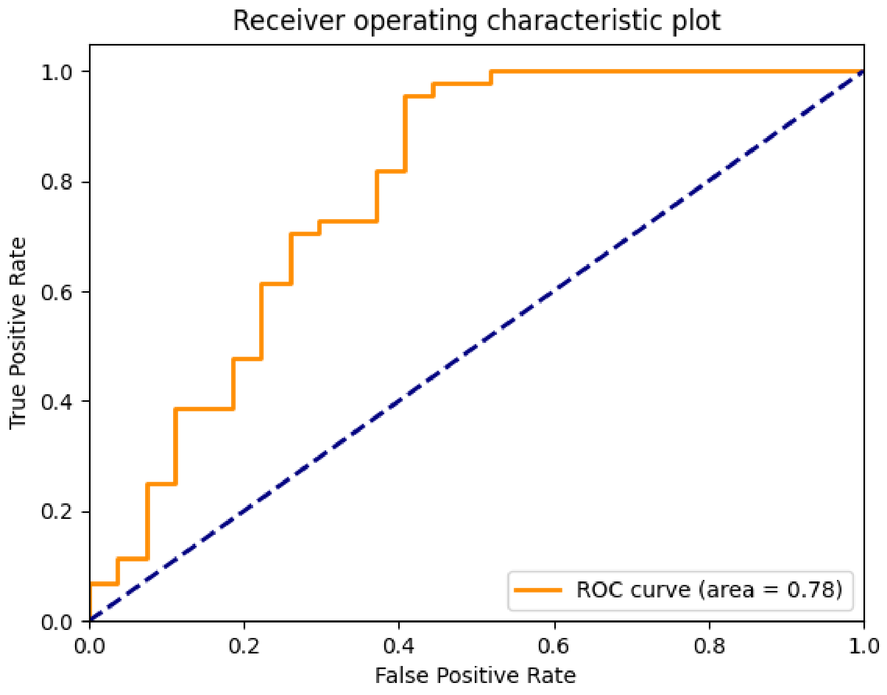


Figure 3.16: GraphSAGE ROC Curve

Moreover, the ROC curve ponits strongly towards the top right corner, with an AUC score of 0.78, highlighting the GraphSAGE model's high probability of correctly classifying restaurant recommendations.

# CHAPTER 4

# Disscusion

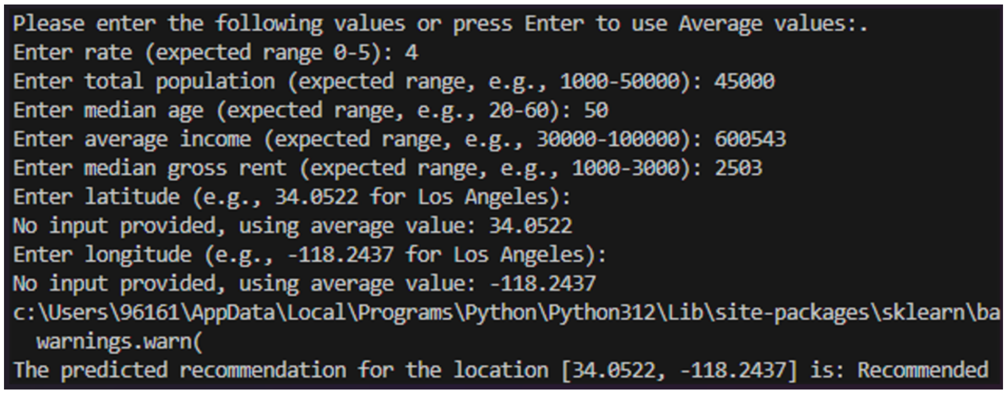
## 4.1 Result & Disscusion

| **Metric** | **GCNs** | **GATs** | **GraphSAGE** |
|---|---|---|---|
| Loss | 0.6761 | 0.6469 | 0.48 |
| Accuracy | 0.62 | 0.66 | 0.77 |
| ROC Score | 0.55 | 0.61 | 0.78 |

Table 4.1: Performance of different Graph Neural Networks

Among the GCNs, GATs, and GraphSAGE models, GraphSAGE seems to yield the most promising outcomes across various assessments, such as testing accuracy, t-SNE plots, and AUC scores. GraphSAGE's sampling strategy, randomly selecting a set number of neighbors for each node, could mitigate overfitting in smaller datasets. Conversely, GCN and GAT models' utilization of all neighbors may heighten overfitting risks, despite fine-tuning efforts. Moreover, GraphSAGE benefits from its capability to employ different aggregation functions, which enables it to adapt more flexibly to diverse characteristics of graph data. Given the apparent influence of geography in the distribution of nodes, the mean aggregator was applied to capture the average influence of neighboring restaurants. Moreover, the GraphSAGE model has the capability to generate embeddings for new nodes by collecting features from their neighbors, which potentially helps utilize limited data more effectively and improve the sparsity of the node representation. This feature of GraphSAGE is particularly relevant in scenarios where data scarcity is prevalent, such as in the restaurant industry where businesses often compete without sharing critical data. Maximizing the efficiency of using limited data to yield accurate classification results is one of the primary goals outlined in this report.

## 4.2 Applicable Tools



```
Please enter the following values or press Enter to use Average values:.
Enter rate (expected range 0-5): 4
Enter total population (expected range, e.g., 1000-50000): 45000
Enter median age (expected range, e.g., 20-60): 50
Enter average income (expected range, e.g., 30000-100000): 600543
Enter median gross rent (expected range, e.g., 1000-3000): 2503
Enter latitude (e.g., 34.0522 for Los Angeles):
No input provided, using average value: 34.0522
Enter longitude (e.g., -118.2437 for Los Angeles):
No input provided, using average value: -118.2437
c:\Users\96161\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\ba
  warnings.warn(
The predicted recommendation for the location [34.0522, -118.2437] is: Recommended
```

Figure 4.1: Demo Tool Exhibition

Users have the opportunity to test the trained model using interactive tools. The tool requests users to input geolocations via latitude and longitude, along with other optional features such as average population, median income, average age, gross median rent, and ratings. If the users do not input these optional features, the tool will automatically populate them with average values for the specified geolocation. Users need not worry about logarithmic transformations and standardization; the program will automatically process the real-world data into a form suitable for model input. After entering all the features, the model will process the data and deliver a result, recommending or not recommending the location. As a prototype, this tool has the potential to assist marketing departments and franchising investors in selecting geographically advantageous locations.

## 4.3 Limitation & Feature works

One of the primary limitations highlighted in this paper is the sparse and limited amount of data available. Restaurant data often constitutes a business secret, potentially dangerous if competitors access it. Currently, the most accessible sources are Yelp and citizen-generated data, which is why we initially chose to classify restaurants as recommended or not. Looking ahead, should we gain access to more comprehensive data concerning restaurant expansion and earnings, we could refine the response variables of the GNN models to distinguish between profitable and unprofitable ventures, offering more direct insights for investors.

Additionally, in this paper, input features are primarily based on ZIP codes. However, even for geographically close restaurants, actual customer flow can differ. In the future, we might integrate population flow data from Google APIs, assuming such functionalities become available. Another improvement could involve using specific building prices for the gross median renting fee instead of relying on ZIP code averages. With more detailed data, we could address the issue of sparse nodes, potentially reducing overfitting risks and enhancing model performance. This would not only improve the effectiveness of GCNs and GATs models but also provide alternative models to handle datasets with different structures.

## REFERENCES

[BBL17] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. "Geometric deep learning: going beyond Euclidean data." *IEEE Signal Processing Magazine*, **34**(4):18–42, 2017.

[BFL19] Eugênio J. S. Bitti, Muriel Fadairo, Cintya Lanchimba, and Vivian Lara dos Santos Silva. "Should I Stay or Should I Go? Geographic Entrepreneurial Choices in Brazilian Franchising." *Journal of Small Business Management*, **57**(S2):244–267, 2019.

[Bur22] U.S. Census Bureau. "Median Household Income in Los Angeles." https://data.census.gov/profile?q=los

[BZS14] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. "Spectral Networks and Deep Locally Connected Networks on Graphs." In *ICLR*, 2014.

[Chu97] Fan R. K. Chung. *Spectral Graph Theory.* American Mathematical Society, 1997.

[Dat22] City Data. "ZIP Code Directory." https://www.city-data.com/zips/zipdir/dir111.html, 2022.

[HYL17] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive representation learning on large graphs." In *NeurIPS*, 2017.

[Kan21] Seokho Kang. "k-Nearest Neighbor Learning with Graph Neural Networks." *Mathematics*, **9**(8):830, 2021.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks." In *NeurIPS*, 2012.

[Mic03] Steven C. Michael. "First mover advantage through franchising." *Journal of Business Venturing*, **18**(1):61–80, 2003.

[New10] M. E. J. Newman. *Networks: An Introduction.* Oxford University Press, 2010.

[SNF13] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains." *IEEE Signal Processing Magazine*, **30**(3):83–98, 2013.

[Vel18] Petar Veličković et al. "Graph attention networks." In *ICLR*, 2018.

[Wes01] D. B. West. *Introduction to Graph Theory.* Prentice Hall, 2 edition, 2001.

[Yel22a] Yelp. "Top Restaurants in Los Angeles, CA." https://www.yelp.com/search?find$_{d}esc = Top + Restaurantsfind_{l}oc = Los + Angeles$

[Yel22b] Yelp. "Worst Restaurants in Los Angeles, CA." https://www.yelp.com/search?find$_{d}esc = Worst + Restaurantfind_{l}oc = Los + Angeles$

[Zho20] Jie Zhou et al. "Graph Neural Networks: A Review of Methods and Applications." *AI Open*, 2020.