Statistical Analysis of Network Data Flows and Predictions Using Statistical and Machine Learning Regression Models

by

Albert Boateng
20216003

Maheen Mehjabeen Rahim
23216010

A thesis submitted to the Department of Mathematics and Natural Sciences
in partial fulfillment of the requirements for the degree of
B.Sc. in Mathematics

Department of Mathematics and Natural Sciences
Brac University
May 2024

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing degree at Brac University.

2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.

3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.

4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Albert Boateng
20216003

Maheen Mehjabeen Rahim
23216010

Approval

The thesis/project titled "Statistical Analysis of Network Data Flows and Predictions Using Statistical and Machine Learning Regression Models" submitted by

1. Albert Boateng (20216003)
2. Maheen Mehjabeen Rahim (23216003)

Of Spring, 2024 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Mathemarics on May 23, 2024.

Examining Committee:

Co-supervisor:
(Member)

Dr. Mohammad Rafiqul Islam
Professor
Department of Mathematics and Natural Sciences
Brac University

Program Coordinator:
(Member)

Dr. Syed Hasibul Hassan Chowdhury
Professor
Department of Mathematics and Natural Sciences
Brac University

Head of Department:
(Chair)

Professor A F M Yusuf Haider
Professor and Chairperson
Department of Mathematics and Natural Sciences
Brac University

Abstract

This paper presents a statistical analysis of measurements relating to network's data flows and predictions using statistical and machine learning regression models. The study's objective is to use statistical methods and machine learning regression models to analyze and make predictions on a spatio-temporal traffic volume dataset obtained by Dr. Liang Zhao (Emory University), from sensors along two major highways in Northern Virginia and Washington, D.C. This work aims to answer some fundamental questions related to the network such as: 1. What statistical inferences and descriptive analysis can be made on the network's data flow? 2. How can one obtain the Routine Matrix of the Network from the Adjacency Matrix? 3. How can one employ various techniques, such as Regularization and Singular Value Decomposition (SVD), to solve the singularity or ill posed nature of the network in the Traffic Matrix Estimation?, and 4. How can one apply Machine Learning regression models, such as Support Vector Regressor (SVR) and XGBoost Regressor, to make predictions on the Network's flow volume? Concepts in this work or paper can be practically applied on other real world networks to analyze and make predictions on the network's data flow.

Keywords: Statistical Analysis, Network, Data flow, Graph, Routing Matrix, Traffic Matrix, Adjacency Matrix, Machine Learning, Regression Models.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

λ	Regularization Parameter
μ	Estimated Traffic Matrix
Σ	Diagonal singular vectors matrix
A_{ij}	Adjacency matrix
B	Routing Matrix
B^+	Pseudo inverse of matrix B
c	Cost
c_{ij}	Origin-Destination cost
E	Edge
e	Link
EVS	Explained Variance Score
G	Network Graph
i	Origin
j	Destination
M_{+j}	Network's Inflow Routine Matrix
M_{i+}	Network's Outflow Routine Matrix
MAE	Mean Absolute Error
Max	Maximum Value
$MedAE$	Median Absolute Error
Min	Minimum Value
MLP	Deep Neural Multilayer Perceptron
MSE	Mean Squared Error

OD	Origin-Destination
$Q1$	First Quartile
$Q3$	Third Quartile
R^2	R-squared
RSS	Residual Sum of Squares
Std	Standard Deviation
SVD	Singular Value Decomposition
SVM	Support Vector Machine
SVR	Support Vector Regressor
TSS	Total Sum of Squares
U	Left singular vectors matrix
V	Right singular vectors matrix
V	vertice
w_i	Model Parameters
Xe	Total flow over a specific link $e \in E$
Z	Origin-Destination Matrix
Z_{+j}	Net In-flow corresponding to vertex j
Z_{i+}	Net Out-flow corresponding to vertex i
Z_{ij}	Origin-Destination (i, j) flow volume

Chapter 1

Introduction

Analysis of network flow data involves applying statistical and mathematical techniques to study the patterns, trends, and anomalies in a network's data. By identifying these patterns, trends, and anomalies within the data, network analysts can make informed decisions about the network which will improve network performance and optimize network resources. A network graph (G), consists of a set of vertices (V) and a set of edges (E) where the vertices represent the elements or nodes of the network, while the edges represent the connections or relationships between the vertices or nodes. In the context of network flows, these flows have specific directions, typically from a source or origin to a destination, implying that the edges of the graph have specific directions [1]. As such network graphs are directed graphs or digraphs.
The movement of data or traffic typically passes through multiple links as it travels between vertices or nodes in the graph. This movement is captured and represented by a matrix called the Routing Matrix [1]. The Routing Matrix (B) has the same number of rows and columns as the total number of links in the network graph (e) and the total number of origin-destination pairs (i, j) respectively. The entries of B indicate the amount of traffic or the proportion of traffic that flows through each link for a given origin-destination pair. These entries can be numerical values representing the fraction of traffic or they can be binary values indicating whether a link is used or not for a particular origin-destination pair [1]. That is,

$$B_{e,ij} = \begin{cases} 1 & \text{if link } e \text{ is traversed in going from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

By using the Routing Matrix B, we can analyze and optimize the flow of traffic in a network by understanding the paths taken by traffic and designing efficient routing algorithms which overall improves network performance.
Another matrix important in the study of Network data flow is the Traffic or Origin-Destination (OD) matrix, denoted as

$$Z = [Z_{ij}]. \tag{1.2}$$

The OD matrix provides information about the total volume of traffic flowing from an origin vertex i to a destination vertex j within a given period of time [2]. The OD matrix Z is a rectangular matrix with rows corresponding to the origin vertices and columns corresponding to the destination vertices [1]. Each element Z_{ij} of the

matrix represents the total volume of traffic flowing from the origin vertex i to the destination vertex j. The values in the OD matrix Z can be obtained through various data collection methods, such as traffic surveys, sensors, or statistical modeling techniques.
Other quantities related to the OD matrix Z include the net out-flow and net in-flow corresponding to vertices i and j, respectively, obtained from the sums of traffic volumes in the traffic matrix Z. The net out-flow corresponding to vertex i, denoted as Z_{i+}, represents the total volume of traffic flowing out from vertex i to all other vertices in the network. It can be calculated by summing up the traffic volumes Z_{ij} for all destination vertices j connected to vertex i [1]:

$$Z_{i+} = \sum_j Z_{ij} \tag{1.3}$$

On the other hand, the net in-flow corresponding to vertex j, denoted as Z_{+j}, represents the total volume of traffic flowing into vertex j from all other vertices in the network. It is calculated by summing up the traffic volumes Z_{ij} for all origin vertices i connected to vertex j [1]:

$$Z_{+j} = \sum_i Z_{ij} \tag{1.4}$$

These quantities, Z_{i+} and Z_{+j}, provide valuable information about the flow patterns at specific vertices in the network. By summing up the traffic volumes associated with each vertex, we can understand the total traffic entering or leaving a particular vertex.
Similarly, to quantify the flow of traffic on the link or edge, we can define Xe as the total flow over a specific link $e \in E$. Each X_e represents the volume or amount of flow passing through that particular link in the network. By considering all the links in the network, we can form a vector X, denoted as $X = (X_e)_{e \in E}$, representing the total flow across all links [1]. By multiplying the routing matrix B with the traffic matrix vector Z, we can obtain the link flow vector X expressed mathematically by [1] as:

$$X = BZ. \tag{1.5}$$

In addition, within the context of network flows, there exists a concept of "cost" (c), which is usually associated with the paths or links. The cost represents a measure of expense, effort, time, or other relevant factors required to traverse a specific path or link in the network [2]. It can be influenced by various factors such as distance, congestion, or resource utilization. The concept of cost plays a role in making routing decisions and optimizing network flows by minimizing the associated costs.
Nonetheless, in most network data, B and Z are not given and as such, one need to obtain these matrix so they can apply the various statistical and mathematical models involving these matrices to model their network. Usually, only the Adjacency matrix of the network is given in addition to the Traffic Flow matrices. The Adjacency Matrix (A) is a square matrix used to represent which vertices are adjacent to each other through edges in the network [3]. The entries of the Adjacency matrix (A_{ij}) is 1 if there is an edge between the vertices i and j and 0 otherwise [3]. That is:

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge between } i \text{ and } j \\ 0 & \text{otherwise} \end{cases} \tag{1.6}$$

The Traffic Flow Matrices are also usually given in addition to the Adjacency Matrix and gives the total flow volumes that was recorded on the vertices or edges of the network [4]. With, the traffic flow matrices, one can as well model the Origin-Destination Matrix as well as the different flow volumes or costs in the network. In addition to the adjacency matrix and the Traffic Flow Matrices, in some network data, are the Traffic Indices matrices which contain various network features and capture various aspects of traffic patterns and network topology [5].
Statistical analysis of Network Data Flow is classified by [1] into three main measurement and analysis goals:

1. **Observation of Traffic Matrix** Z where it's possible to observe the entire traffic matrix Z directly and our goal is to model these observations to understand how factors like costs affect flow volumes and make predictions about future flow volumes. Here, the Gravity models, such the General Gravity Model and the Stewart Gravity Model, are commonly used to achieve these goals. In depth knowledge and applications of the Gravity models can be found in [1].

2. **Traffic Matrix Estimation** where it's difficult or impossible to directly observe the traffic matrix entries (Z_{ij}) and instead, the link totals (X_e) are easier to measure. As such, the links totals (X_e) are used to estimate the traffic matrix entries (Z_{ij}).

3. **Modeling and Inference of Network Cost Parameters** where we have some Origin-Destination cost c_{ij} and our analysis goal is modeling and inferring other network cost parameters such as unobserved OD and link cost.

The type of statistical analysis performed on a network's data depends on these goals as well as the type of the network data, data availability, and also the parameters of interest in our analysis. Due to the nature of our dataset, our main measurement and analysis goal in this work is to estimate our traffic matrix which falls under goal 2: **Traffic Matrix Estimation**.

Chapter 2

Dataset

The dataset used for this analysis and prediction is a traffic flow forcasting dataset from [6] by Dr. Liang Zhao (Emory University) intended to forcast the spatio-temporal traffic volume based on the historical traffic volume and other features in neighboring locations. The dataset was obtained by measuring the traffic volume of 36 sensor along two major highways in Northern Virginia (Washington D.C. capital region) every 15 minutes.

The dataset has 5 variable names: `tra_X_te`, `tra_X_tr`, `tra_Y_te`, `tra_Y_tr`, `tra_adj_mat`. The `tra_X_te` variable is a $1 * 840$ array of matrices of input test set data with traffic indices for 840 contiguous quarter-hours where each element is a 36×48 matrix representing 36 spatial locations and 48 features. The features of the dataset include the historical sequence of traffic volume sensed during the 11 most recent sample points (11 features), week day (7 features), hour of day (24 features), road direction (4 features), number of lanes (1 feature), and name of the road (1 feature). The `tra_X_tr` variable is also a 1×1261 array of matrices of input training set data with traffic indices for 1261 contiguous quarter-hours where each element is a 36×48 matrix representing 36 spatial locations and 48 features.

The `tra_Y_te` variable is a 36×840 array of matrices of output test set data with traffic flow for 36 locations in 840 continuous quarter-hours from 2017-01-02 00:00. The `tra_Y_tr` variable is also a 36×1261 array of matrices of output test set data with traffic flow for 36 locations in 840 contiguous quarter-hours from 2017-01-02 00:00. Finally, the `tra_adj_mat` square matrix variable which is the adjacency matrix denoting the spatial connectivity of traffic network among the 36 locations.

Various statistical and machine learning models will be used to analyze and make predictions on this traffic flow forecasting dataset. We will first start by visualizing the network using the Traffic Adjacency Matrix. We will then make some Statistical inferences and descriptive analysis on the network then move to the traffic estimation section where we will obtain the Routine Matrix of the Network from the Adjacency Matrix as well as apply techniques, such as Regularization and SVD, to solve the singularity or ill posed nature of the network in the Traffic Matrix Estimation. Finally, we will move to the machine learning prediction section where we will use various machine learning regresion models to make predictions on the Network Traffice indices. All or most of our analysis here will be done in python using core libraries such as `networkx`, `scipy.io`, `matplotlib`, `pandas`, `seaborn`, `sklearn`, and so on.

Chapter 3

Visualizing the Network

There are various layouts for visualizing network graph. However, the choice of the best layout depends on the specific characteristics of the graph and the insights one wants to gain from the visualization. Different layouts or views provide different perspectives, and it's essential to perform some visualizations on a dataset to fully understand the dataset before selecting those that best suit the underlying data and visualization goals. With this network, we will obtain the matrix view, the radial or spring layout, the circular layout, the shell layout, and the planar layout of the traffic network graph. To visualize the graph or the connections between the 36 locations or nodes, the `tra_adj_mat` square adjacency matrix was plotted in python with the aid of the libraries: `networkx` and `matplotlib`.

3.1 Matrix View

The Matrix View visually represent the adjacency matrix of the network. In this representation, nodes are indexed by rows and columns, and the entries of the matrix indicate the presence or absence of edges between pairs of nodes, just like in the adjacency matrix [7]. The visual structure of the matrix provides insights into the connectivity and relationships within the graph, making it a useful tool in network analysis and visualization. Figure 3.1 shows the Matrix View or representation of our network.

3.2 Spring Layout

The spring layout, also known as the force-directed layout, visualize network graphs by simulating a physical system of springs and forces [8]. The main idea behind the spring layout is to position nodes in such a way that connected nodes are drawn closer together and disconnected nodes are pushed further apart [8]. The algorithm starts with an initial random placement of nodes in a 2D or 3D space. Then, it iteratively applies forces between nodes to adjust their positions. The forces can be attractive, pulling connected nodes closer together, and repulsive, pushing disconnected nodes apart [8].
According to [8], the attractive force between connected nodes is based on Hooke's law, which models springs. It pulls the nodes towards each other, making connected edges shorter. The repulsive force between disconnected nodes can be based on

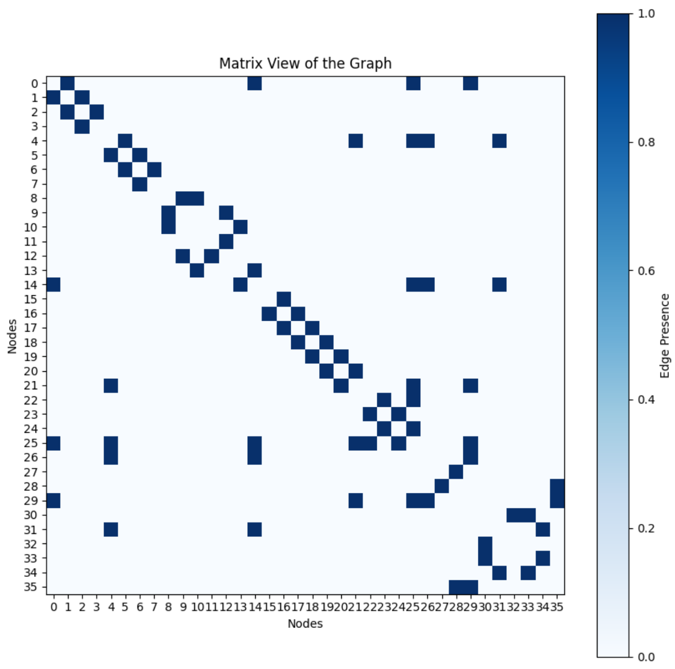


Figure 3.1: Matrix View of the Traffic Network graph

Coulomb's law, where nodes with a stronger repulsive force are farther apart [8]. The spring layout algorithm continues to adjust node positions iteratively until the system reaches a balanced state, where the forces between nodes are minimized or the total energy of the system is low [8]. Spring layout is a widely used algorithm for visualizing graphs and networks due to its simplicity and effectiveness. It works well for many types of graphs and can reveal underlying structures, clusters, and relationships in the network just as in figure 3.2 for our network graph.

3.3 Circular Layout

The circular layout is a graph layout algorithm that positions nodes in a circular arrangement. In this layout, nodes are placed along the circumference of a circle, and the edges are drawn as chords connecting the nodes [8]. The circular layout is useful when there is a notion of ordering among the nodes or when the graph has cyclic structures.
According to [8], nodes placed close together in the circular layout are typically more strongly connected, while nodes farther apart have weaker or no connections. The arrangement of nodes in a circle can make it easier to identify cyclic patterns, trace paths from one node to another, and observe the cyclic ordering of nodes. The circular layout is often used to visualize cyclic graphs, where the graph contains cycles or closed loops, or when there is a natural cyclic order among the nodes. It can reveal cyclic dependencies and help in understanding the cyclic relationships in the data just as in figure 3.3.

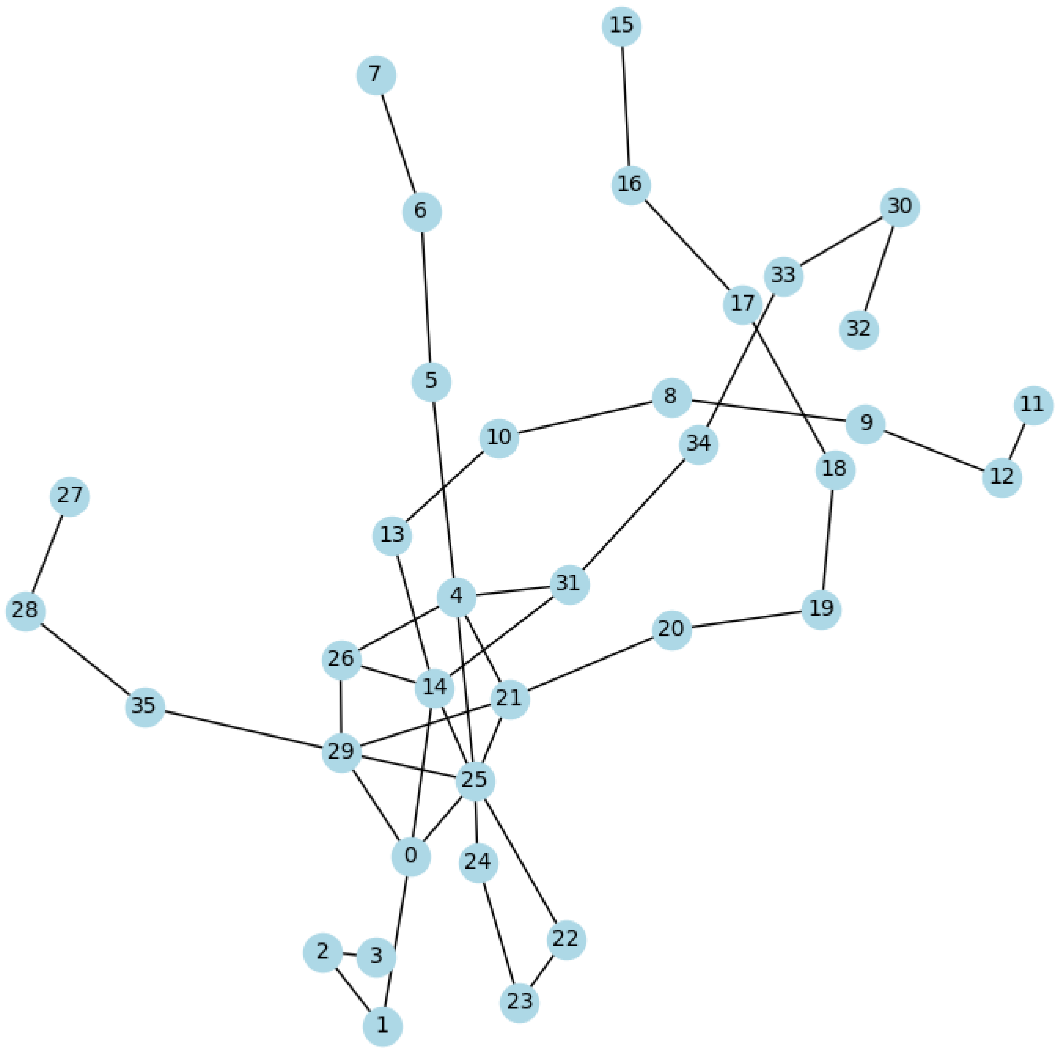


Figure 3.2: Spring layout of the Traffic Network graph

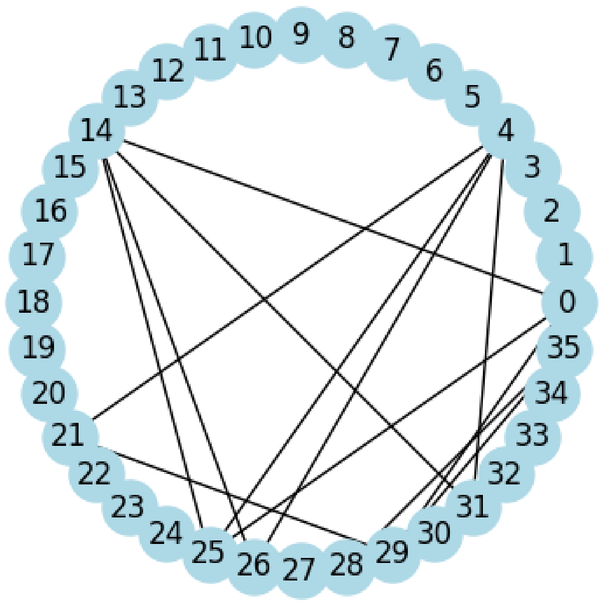


Figure 3.3: Circular layout of the Traffic Network graph

3.4 Shell Layout

The Shell Layout is similar to the Circular layout, but organizes nodes into concentric circles or “shells” [8]. This layout is often used for visualizing hierarchical or radial structures within a graph. The algorithm works by assigning nodes to different shells based on their distance from a specified center as seen in [8]. Nodes in the same shell are generally at the same level of hierarchy. The concentric circles formed by the shells can represent hierarchical levels or layers within the graph. Nodes in the same shell are considered to be at the same level of hierarchy, and edges typically connect nodes across adjacent shells to illustrate relationships [8]. Generally, the Shell Layout provides an effective means of representing and understanding the hierarchical relationships within a graph by organizing nodes into concentric circles around a central point [8]. Figure 3.4 show the Shell layout of our network.

3.5 Planar Layout

The planar layout is a graph layout algorithm used to visualize planar graphs. A planar graph is a graph that can be drawn on a 2D plane without any edge crossings [8]. The planar layout arranges the nodes in a way that preserves the planarity of the graph, where edges do not intersect with each other.
In the planar layout, nodes are positioned such that connected nodes are close together, and edges are drawn as straight lines between the nodes without any crossings [9]. The goal is to create a clear and easy-to-read visualization of the planar graph while maintaining the integrity of its planar structure [8]. Planar graphs often arise in various real-world applications, such as road networks, circuit layouts, and social networks. Their planar nature makes them suitable for certain types of visualizations, and the planar layout algorithm is designed to make the

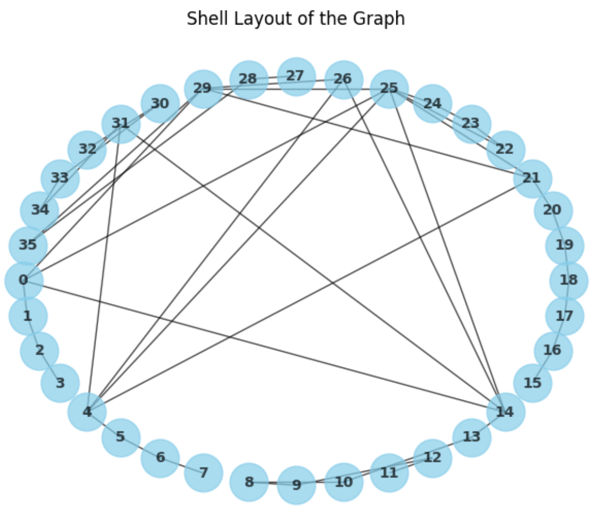


Figure 3.4: Shell layout of the Traffic Network graph

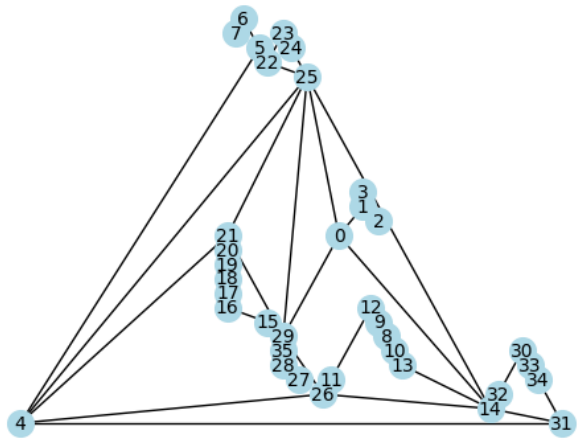


Figure 3.5: Planar layout of the Traffic Network graph

most of this property just as in figure 3.5.

Chapter 4

Descriptive Analysis of the Network

In this chapter, we will perform Descriptive Analysis on our network where we will look at Vertex Degree and Vertex centrality of our network. This analysis will contribute to a comprehensive understanding of the connectivity patterns and the relative influence of nodes within our network.

4.1 Vertex Degree

The degree of a vertex in a graph is the number of edges joined to the particular vertex in the graph [3]. Knowing the Vertices' Degree or Degree Distribution of the Vertices will help us gain insights into the structure, function, and behavior of our network. It will also be essential in characterizing the overall connectivity patterns in our graph which will be used in various graph algorithms and analysis [3].
Our network is composed of 36 vertices labelled from 0 to 35. The degree of the vertices in the graph is given in Table 4.2 and can be obtained from the Adjacency matrix shown in Table 4.1. Figure 4.1a also shows the plot of the Vertex degree and figure 4.1b, the Degree Distribution of the network.
From the table and the figures, we can infer that the network has a diverse range of degrees with vertex 25 having the largest degree (7). This imply that node 25 is the highly connected node or vertex in the network and it is one of the core vertices in the network. The presence of nodes with higher degrees suggests hubs or central nodes that play important roles in connecting different parts of the network [10]. Node 25 and also node 4, 14, 29, are key nodes in the network since they have the highest vertex degrees. Higher Vertex degree usually correspond to high flow volume [7] and as such this nodes will be closely observe during our network flow analysis in chapter 2.4.

4.2 Vertex Centrality

Vertex Centrality highlights the essential or most important vertices in the network [1]. Even though the vertex degree gave us a hint on key vertices in the network, other measure of Vertex Centrality, such as Closeness, Betweenness, and Eigenvector Centrality, can also be used to quantify the importance of the key nodes we

0	→	**[1, 14, 25, 29]**	**18**	→	**[17, 19]**
1	→	**[0, 2]**	**19**	→	**[18, 20]**
2	→	**[1, 3]**	**20**	→	**[19, 21]**
3	→	**[2]**	**21**	→	**[4, 20, 25, 29]**
4	→	**[5, 21, 25, 26, 31]**	**22**	→	**[23, 25]**
5	→	**[4, 6]**	**23**	→	**[22, 24]**
6	→	**[5, 7]**	**24**	→	**[23, 25]**
7	→	**[6]**	**25**	→	**[0, 4, 14, 21, 22, 24, 29]**
8	→	**[9, 10]**	**26**	→	**[4, 14, 29]**
9	→	**[8, 12]**	**27**	→	**[28]**
10	→	**[8, 13]**	**28**	→	**[27, 35]**
11	→	**[12]**	**29**	→	**[0, 21, 25, 26, 35]**
12	→	**[9, 11]**	**30**	→	**[32, 33]**
13	→	**[10, 14]**	**31**	→	**[4, 14, 34]**
14	→	**[0, 13, 25, 26, 31]**	**32**	→	**[30]**
15	→	**[16]**	**33**	→	**[30, 34]**
16	→	**[15, 17]**	**34**	→	**[31, 33]**
17	→	**[16, 18]**	**35**	→	**[28, 29]**

Table 4.1: Adjacency List

previously identified.

- Closeness captures the centrality of a node as it determines which node is the closest to the other nodes [10]. It is defined as the reciprocal of the average shortest path length from the node to all other nodes [10]. Nodes with high closeness centrality are, on average, closer to all other nodes.

- Betweenness captures the extent at which a node is found between two other nodes [2]. It also measures the extent to which a node lies on the shortest paths between other nodes in a network [10]. A node with high betweenness centrality plays a crucial role in connecting different parts of the network.

- Eigenvector Centrality captures the influence of a node in a network [10]. It assigns relative scores to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question[2].

There are other measures of Vertex Centrality in [10] such as:

- Katz Centrality which extends the idea of degree centrality by considering the total number of walks between nodes, with shorter walks weighted more.

- PageRank Centrality which measures the importance of a node based on the link structure of the network.

- Harmonic Centrality which takes into account the sum of the reciprocal of the shortest path lengths to all other nodes.

- Clustering Coefficient which measures the degree to which nodes in a network tend to cluster together.

0	**4**
1	**2**
2	**2**
3	**1**
4	**5**
5	**2**
6	**2**
7	**1**
8	**2**
9	**2**
10	**2**
11	**1**
12	**2**
13	**2**
14	**5**
15	**1**
16	**2**
17	**2**
18	**2**
19	**2**
20	**2**
21	**4**
22	**2**
23	**2**
24	**2**
25	**7**
26	**3**
27	**1**
28	**2**
29	**5**
30	**2**
31	**3**
32	**1**
33	**2**
34	**2**
35	**2**

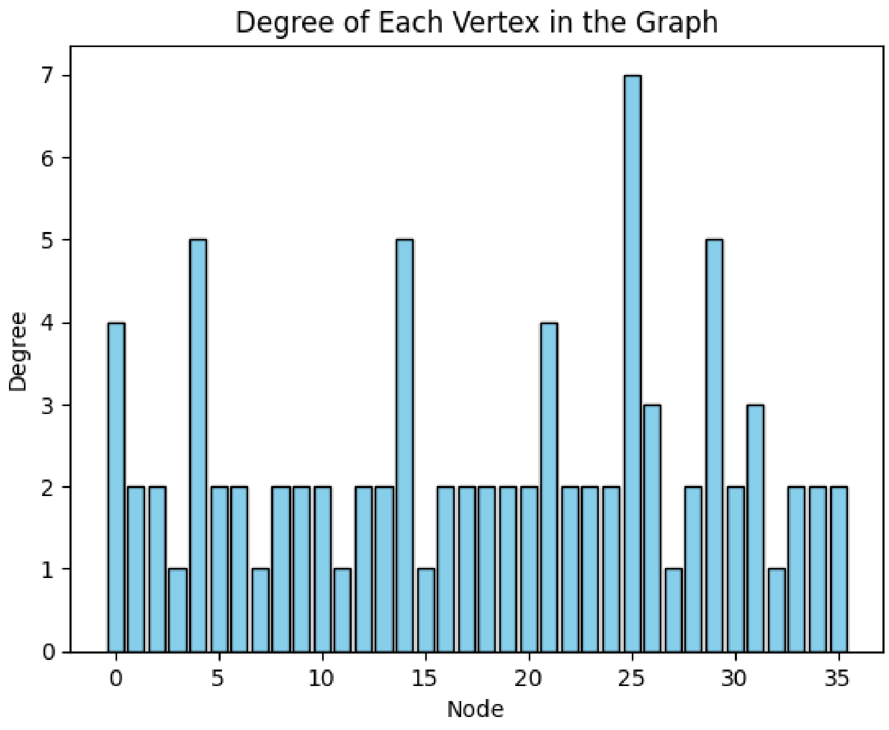
Table 4.2: Vertex Degrees (Vertex — Degree)

- Subgraph Centrality which measures the centrality of a node in a network based on the number of closed walks (cycles) that pass through that node.

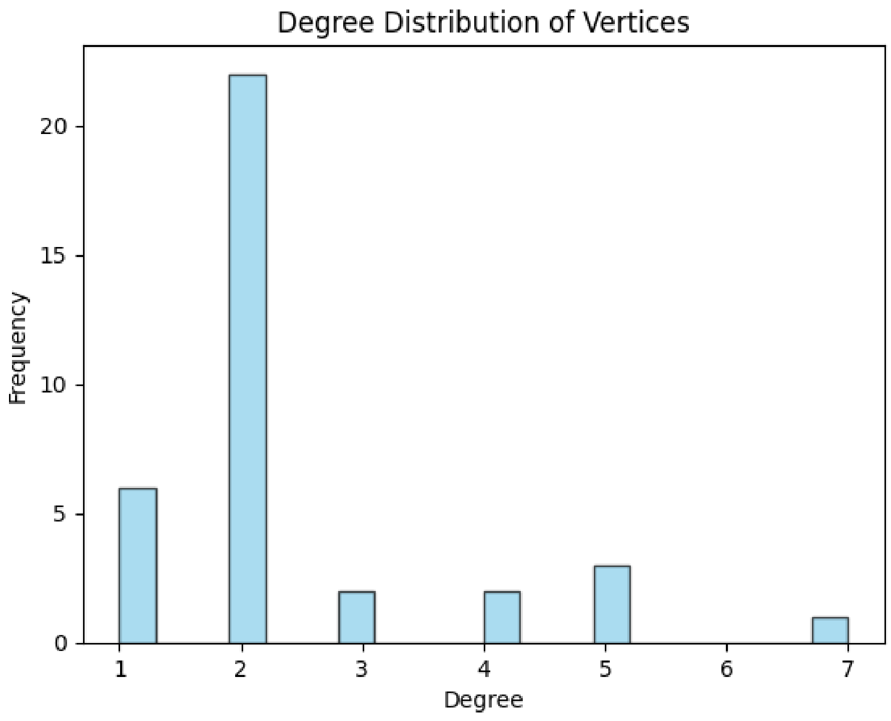
The various measure of centrality of our network is shown in table 4.3 and figure 4.2. The ranking of the nodes for each centrality measure is given below as:

1. **Degree Centrality:** 25 > 4, 14, 29 > 0, 21 > 26, 31 > 1, 2, 5, 6, 8, 9, 10, 12, 13, 16, 17, 18, 19, 20, 22, 23, 24, 28, 30, 33, 34, 35 > 3, 7, 11, 15, 27, 32
2. **Closeness Centrality:**

   25 > 14 > 21 > 4 > 0 > 29 > 26, 31 > 13 > 20, 22, 24 > 5 > 1 > 34, 35 > 10 > 19 > 23 > 6 > 2 > 33 > 28 > 8 > 18 > 7 > 3, 30 > 27 > 9 > 17 > 32 > 12 > 16 > 11 > 15
3. **Betweenness Centrality:** 14 > 25 > 21 > 4 > 13, 20 > 31 > 10, 19 > 29 > 0 > 8, 18, 34 > 1, 5, 9, 17, 33, 35 > 2, 6, 12, 16, 28, 30 > 26 > 22, 24 > 23 > 3, 7, 11, 15, 27, 32
4. **Eigenvector Centrality:** 25 > 29 > 14 > 4 > 21 > 0 > 26 > 31 > 22, 24 > 35 > 13 > 5 > 20 > 1 > 23 > 34 > 28 > 10 > 6 > 19 > 2 > 33 > 27 > 8 > 18 > 7 > 3 > 30 > 9 > 17 > 32 > 12 > 16 > 11 > 15
5. **Clustering Coefficient:** 0, 21 > 29 > 25 > 4, 14 > 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 22, 23, 24, 26, 27, 28, 30, 31, 32, 33, 34, 35
6. **Subgraph Centrality:** 25 > 29 > 14 > 4 > 21 > 0 > 26 > 31 > 22, 24 > 35 > 13 > 5 > 20 > 1 > 23 > 34 > 10 > 19 > 33 > 8 > 18 > 9 > 17 > 28 > 6 > 2 > 30 > 12 > 16 > 27 > 7 > 3 > 32 > 11 > 15
7. **Harmonic Centrality:** 25 > 14 > 4 > 29 > 21 > 0 > 26, 31 > 22, 24 > 13 > 20 > 5 > 1 > 35 > 34 > 10 > 19 > 23 > 6 > 33 > 2 > 28 > 8 > 18 > 9 > 17 > 30 > 7 > 3 > 27 > 12 > 16 > 32 > 11 > 15
8. **Katz Centrality:** 25 > 29 > 14 > 4 > 21 > 0 > 26 > 31 > 22, 24 > 35 > 13 > 5 > 20 > 1 > 34 > 23 > 10 > 19 > 33 > 8 > 18 > 9 > 17 > 28 > 6 > 2 > 30 > 12 > 16 > 27 > 7 > 3 > 32 > 11 > 15



(a) Vertex Degree



(b) Degree Distribution of the Vertices

9. **PageRank:** 25 > 4 > 14 > 29 > 0 > 21 > 16 > 12 > 30 > 31 > 17 > 9 > 2 > 6 > 28 > 18 > 8 > 33 > 26 > 19 > 10 > 34 > 1 > 5 > 35 > 20 > 13 > 23 > 22, 24 > 15 > 11 > 32 > 3 > 7 > 27

To create a combined ranking, we can assign weights to the normalized form of each centrality measure based on their importance and then compute a normalize weighted sum for each vertex.

4.3 Flow Volume Statistical Measures

To statistically analyze the traffic flow volumes of the vertices, various statistic measures will be employed. These statistical measures include Mean, Median, Standard Deviation (Std), Minimum Value, Maximum value, First Quartile (Q1), and Third Quartile (Q3).

Mean (Average)

The mean (or average) flow volume is a measure of central tendency that represents the arithmetic average flow volume across all vertices [7]. It is useful for understanding the average traffic flow volume across vertices. It is calculated by adding up all the flow volume values in a dataset and then dividing the sum by the total number of flow volume [7]. The Mean is given mathematically as:

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

where $\bar{X}$ is the mean flow volume, X_i represents each individual flow volumes, and n is the total number of flow volume values.

Median

The median flow volume is also measure of central tendency that represents the middle flow volume when it is ordered [11]. It gives a robust measure of central tendency, especially when dealing with skewed datasets or datasets with outliers [11]. It provides insight into the typical or central traffic flow volume.

Standard Deviation (Std)

Standard deviation measures the amount of variation or dispersion in the traffic flow volumes [11]. It also indicates the spread of traffic flow volumes around the mean. A vertice with higher standard deviation indicates greater variability in its flow volumes [11]. Std is given as:

$$\text{Std} = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n}}$$

where: X_i represents each flow volume, $\bar{X}$ is the mean (average) of the flow volumes, and n is the total number of values in the dataset [11].

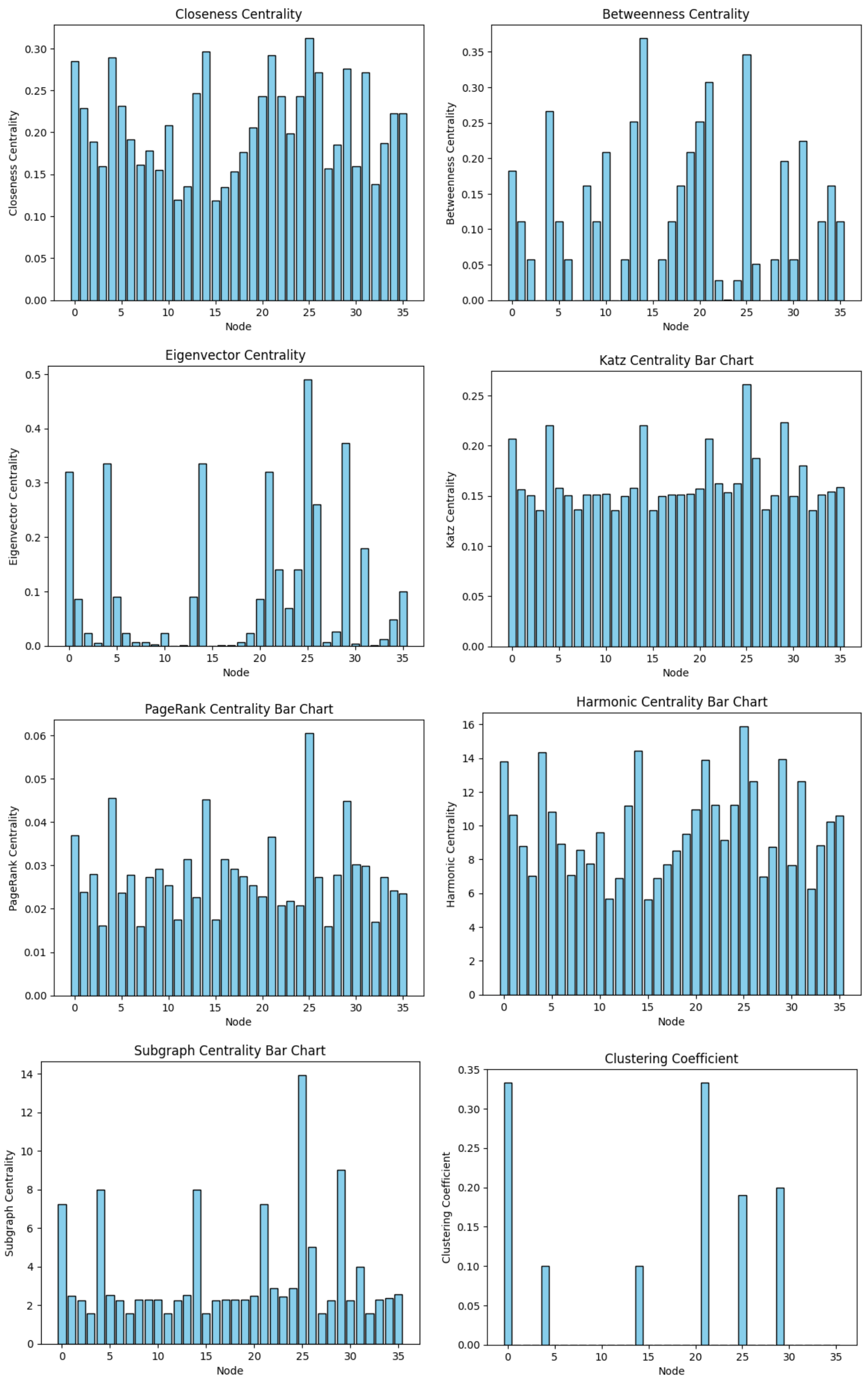
Minimum (Min) and Maximum (Max) Value

The Minimum flow volume is the smallest flow volume observed in the dataset while the Maximum flow volume is the largest flow volume observed in the dataset [11]. These values helps identify the range of values in the dataset. The range is given as $Max - Min$ and provides a quick overview of the data's spread from the minimum to the maximum value [11].

Vertex	Degree	Closeness Centrality	Betweenness Centrality	Eigenvector Centrality	Clustering Coefficient	Subgraph Centrality	Harmonic Centrality	Katz Centrality	PageRank Centrality
0	4	0.28455	0.18263	0.32073	0.33	7.22152	13.79405	0.20718	0.03701
1	2	0.22876	0.11092	0.08583	0.00	2.48385	10.62302	0.15676	0.02393
2	2	0.18919	0.05714	0.02286	0.00	2.23393	8.80198	0.15029	0.02799
3	1	0.15982	0.00000	0.00571	0.00	1.59071	7.02186	0.13604	0.01606
4	5	0.28926	0.26611	0.33487	0.10	7.97545	14.32738	0.22047	0.04559
5	2	0.23179	0.11092	0.08961	0.00	2.52956	10.81349	0.15810	0.02377
6	2	0.19126	0.05714	0.02387	0.00	2.23533	8.90317	0.15042	0.02788
7	1	0.16129	0.00000	0.00596	0.00	1.59074	7.08575	0.13605	0.01602
8	2	0.17857	0.16134	0.00643	0.00	2.27966	8.58297	0.15132	0.02738
9	2	0.15487	0.11092	0.00172	0.00	2.27799	7.75083	0.15111	0.02919
10	2	0.20833	0.20840	0.02400	0.00	2.28606	9.60754	0.15197	0.02543
11	1	0.11986	0.00000	0.00012	0.00	1.59064	5.65842	0.13598	0.01755
12	2	0.13566	0.05714	0.00046	0.00	2.22886	6.90672	0.14972	0.03149
13	2	0.24648	0.25210	0.08964	0.00	2.53116	11.16230	0.15825	0.02266
14	5	0.29661	0.36947	0.33488	0.10	7.97548	14.43452	0.22048	0.04521
15	1	0.11905	0.00000	0.00011	0.00	1.59064	5.63298	0.13598	0.01756
16	2	0.13462	0.05714	0.00044	0.00	2.22886	6.87382	0.14972	0.03150
17	2	0.15351	0.11092	0.00165	0.00	2.27799	7.70639	0.15111	0.02921
18	2	0.17677	0.16134	0.00615	0.00	2.27963	8.51908	0.15130	0.02742
19	2	0.20588	0.20840	0.02298	0.00	2.28466	9.50635	0.15183	0.02550
20	2	0.24306	0.25210	0.08586	0.00	2.48545	10.97183	0.15691	0.02278
21	4	0.29167	0.30700	0.32074	0.33	7.22155	13.90119	0.20720	0.03659
22	2	0.24306	0.02773	0.14001	0.00	2.89236	11.21190	0.16250	0.02081
23	2	0.19886	0.00084	0.06995	0.00	2.42959	9.14246	0.15351	0.02185
24	2	0.24306	0.02773	0.14001	0.00	2.89236	11.21190	0.16250	0.02081
25	7	0.31250	0.34650	0.49058	0.19	13.93815	15.88571	0.26136	0.06056
26	3	0.27132	0.05154	0.26040	0.00	5.01911	12.62738	0.18742	0.02724
27	1	0.15695	0.00000	0.00664	0.00	1.59074	6.97186	0.13605	0.01597
28	2	0.18519	0.05714	0.02657	0.00	2.23597	8.74643	0.15045	0.02777
29	5	0.27559	0.19636	0.37273	0.20	9.01005	13.93690	0.22316	0.04491
30	2	0.15982	0.05714	0.00342	0.00	2.22889	7.68139	0.14975	0.03026
31	3	0.27132	0.22409	0.17928	0.00	3.97484	12.62738	0.18052	0.02991
32	1	0.13834	0.00000	0.00086	0.00	1.59064	6.24293	0.13598	0.01703
33	2	0.18717	0.11092	0.01284	0.00	2.27999	8.83532	0.15140	0.02734
34	2	0.22293	0.16134	0.04799	0.00	2.35618	10.23968	0.15420	0.02426
35	2	0.22293	0.11092	0.09974	0.00	2.56274	10.58135	0.15837	0.02361

Table 4.3: Vertex Centrality

Figure 4.2: Vertex Centrality



First Quartile (Q1) and Third Quartile (Q3)

The Quartiles divide the dataset into four equal parts, providing information about the spread of data in different segments. Q1 is the value below which 25% of the data falls and represents the lower 25% of flow volumes. Q3 is the value below which 75% of the data falls and represents the lower 75% of flow volumes [11].

Vertex Centrality Vs. Flow Volume

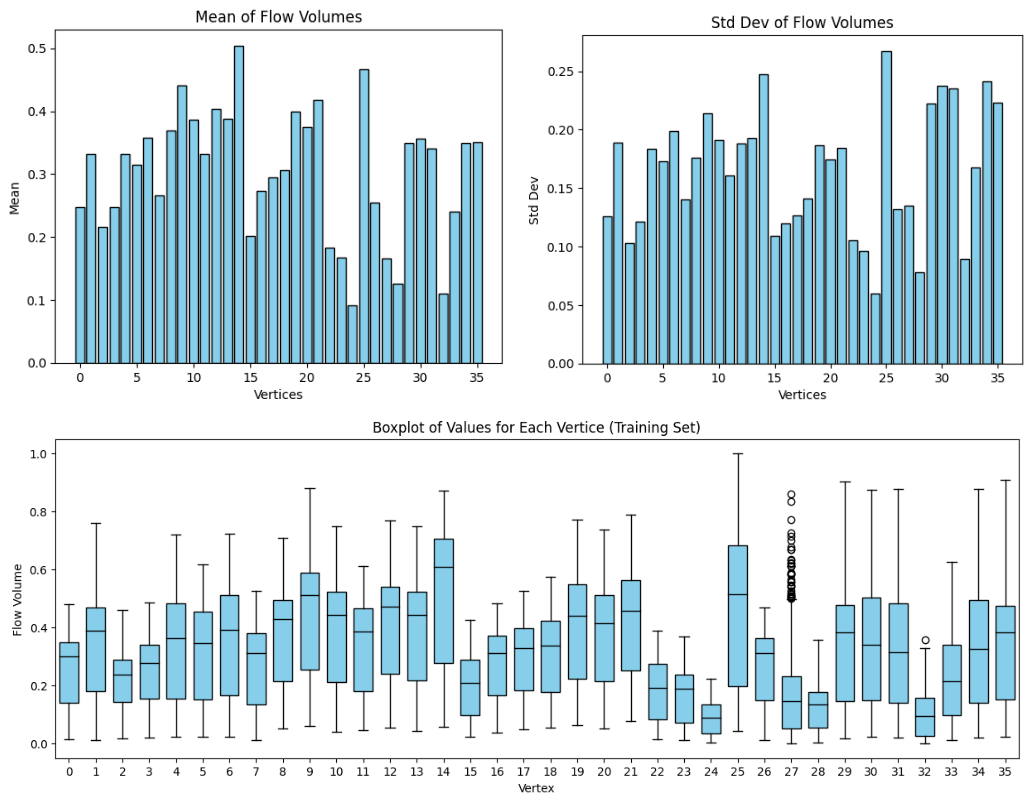
The statistical measures of all the 36 nodes are shown in Table 4.4 for the Training set and in Table 4.5 for the testing set. Figure 4.3b gives the Bar plot and Box plot of our flow volumes in both the Training set and the Testing set. Analyzing the graph shows that indeed the nodes with the highest Vertex Centrality $(25, 4, 12, 29)$ had the highest flow volumes in both the training and testing set.

Vertex	Mean	Median	Std	Min	Max	Q1	Q3
0	0.24792	0.30079	0.12596	0.01448	0.48015	0.13965	0.34844
1	0.33223	0.38860	0.18915	0.01214	0.75993	0.17982	0.46801
2	0.21522	0.23681	0.10315	0.01775	0.46007	0.14386	0.29005
3	0.24789	0.27837	0.12097	0.02149	0.48575	0.15367	0.34003
4	0.33210	0.36291	0.18360	0.02289	0.71976	0.15460	0.48342
5	0.31478	0.34563	0.17282	0.02429	0.61700	0.15227	0.45586
6	0.35713	0.39187	0.19902	0.02475	0.72209	0.16674	0.51191
7	0.26575	0.31200	0.14040	0.01168	0.52499	0.13498	0.37973
8	0.36958	0.42784	0.17625	0.05091	0.70808	0.21485	0.49510
9	0.44023	0.51238	0.21429	0.06119	0.88090	0.25362	0.58991
10	0.38605	0.44278	0.19111	0.04157	0.74872	0.21345	0.52452
11	0.33162	0.38767	0.16089	0.04577	0.61280	0.17936	0.46707
12	0.40392	0.47128	0.18805	0.05418	0.76880	0.24101	0.54087
13	0.38755	0.44418	0.19295	0.04297	0.74965	0.21859	0.52312
14	0.50429	0.61046	0.24712	0.05885	0.87342	0.27837	0.70761
15	0.20123	0.21018	0.10904	0.02242	0.42504	0.09902	0.29005
16	0.27334	0.31200	0.11981	0.03643	0.48342	0.16534	0.37179
17	0.29414	0.32882	0.12697	0.04998	0.52732	0.18216	0.39841
18	0.30596	0.33863	0.14071	0.05418	0.57403	0.17842	0.42270
19	0.39928	0.44092	0.18635	0.06212	0.77113	0.22373	0.54974
20	0.37551	0.41383	0.17473	0.05325	0.73657	0.21392	0.51191
21	0.41860	0.45680	0.18420	0.07707	0.78935	0.25315	0.56235
22	0.18306	0.19103	0.10563	0.01448	0.39000	0.08267	0.27370
23	0.16730	0.19010	0.09597	0.01308	0.36852	0.07193	0.23681
24	0.09135	0.08968	0.06006	0.00467	0.22466	0.03456	0.13545
25	0.46639	0.51425	0.26735	0.04250	1.00000	0.19710	0.68239
26	0.25451	0.31294	0.13228	0.01261	0.46847	0.14946	0.36338
27	0.16615	0.14619	0.13535	0.00140	0.86128	0.05231	0.23073
28	0.12522	0.13358	0.07805	0.00420	0.35778	0.05558	0.17702
29	0.34908	0.38440	0.22200	0.01728	0.90378	0.14666	0.47828
30	0.35618	0.34003	0.23719	0.02475	0.87623	0.14853	0.50304
31	0.34074	0.31387	0.23560	0.02055	0.87716	0.13965	0.48389
32	0.11002	0.09528	0.08978	0.00000	0.35638	0.02616	0.15740
33	0.23964	0.21625	0.16775	0.01168	0.62728	0.09668	0.33956
34	0.34971	0.32508	0.24165	0.02008	0.87669	0.13965	0.49369
35	0.35003	0.38393	0.22309	0.02382	0.91032	0.15086	0.47454

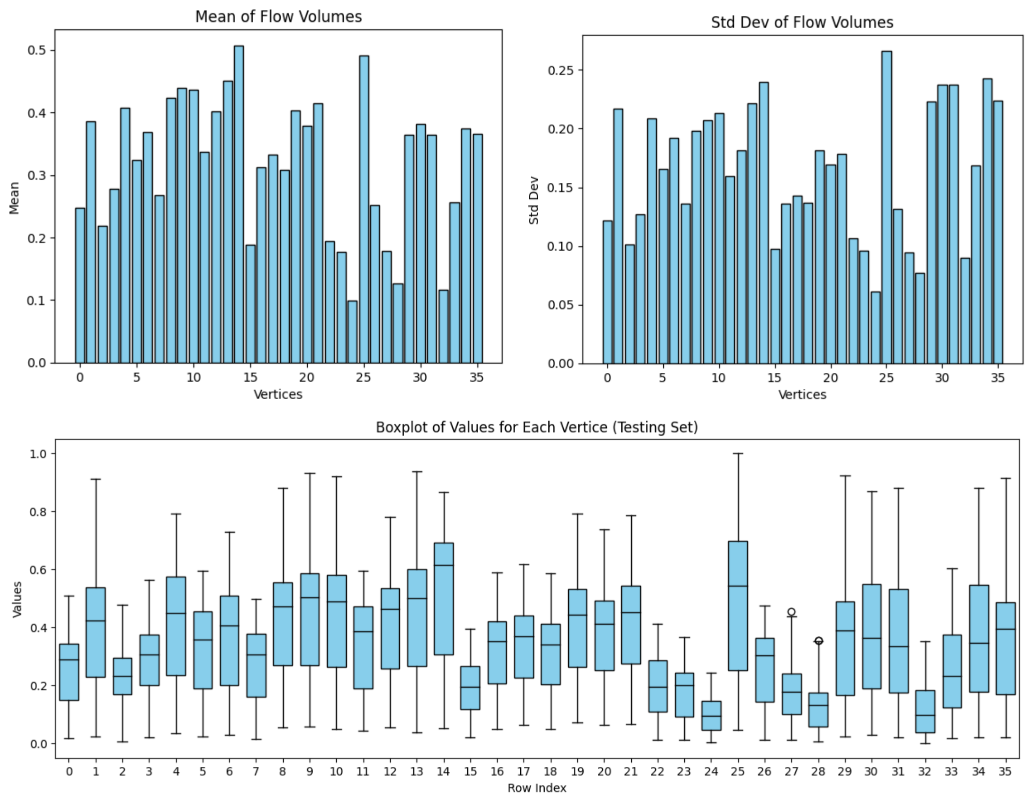
Table 4.4: Training Set

Vertex	Mean	Median	Std	Min	Max	Q1	Q3
0	0.24721	0.28748	0.12146	0.01728	0.50911	0.15016	0.34283
1	0.38609	0.42363	0.21694	0.02475	0.90986	0.22992	0.53631
2	0.21909	0.23190	0.10151	0.00747	0.47735	0.16803	0.29332
3	0.27698	0.30570	0.12671	0.01962	0.56282	0.20002	0.37553
4	0.40710	0.44886	0.20828	0.03596	0.79122	0.23412	0.57251
5	0.32397	0.35801	0.16538	0.02429	0.59411	0.18940	0.45504
6	0.36780	0.40448	0.19173	0.02896	0.72723	0.19956	0.50911
7	0.26773	0.30476	0.13572	0.01588	0.49743	0.15927	0.37669
8	0.42333	0.47244	0.19790	0.05465	0.87856	0.26763	0.55441
9	0.43962	0.50163	0.20731	0.05885	0.93087	0.26892	0.58547
10	0.43610	0.48879	0.21342	0.04811	0.91826	0.26296	0.58034
11	0.33690	0.38440	0.15944	0.04344	0.59505	0.18940	0.47011
12	0.40203	0.46147	0.18165	0.05371	0.78001	0.25619	0.53386
13	0.45022	0.50000	0.22126	0.03737	0.93601	0.26623	0.59879
14	0.50715	0.61420	0.23943	0.05184	0.86595	0.30441	0.69185
15	0.18892	0.19477	0.09722	0.02149	0.39281	0.11770	0.26716
16	0.31278	0.35030	0.13588	0.05044	0.58758	0.20645	0.41850
17	0.33311	0.36899	0.14267	0.06212	0.61794	0.22606	0.43998
18	0.30805	0.34050	0.13689	0.04998	0.58524	0.20189	0.41114
19	0.40340	0.44208	0.18137	0.07099	0.78935	0.26436	0.53223
20	0.37797	0.41102	0.16968	0.06259	0.73610	0.25175	0.49113
21	0.41475	0.45002	0.17827	0.06632	0.78515	0.27417	0.54332
22	0.19430	0.19337	0.10615	0.01121	0.41009	0.10766	0.28468
23	0.17687	0.20014	0.09561	0.01074	0.36572	0.09096	0.24346
24	0.09849	0.09435	0.06108	0.00467	0.24241	0.04577	0.14573
25	0.49046	0.54367	0.26638	0.04577	0.99813	0.25140	0.69687
26	0.25213	0.30220	0.13142	0.01121	0.47314	0.14433	0.36385
27	0.17825	0.17609	0.09468	0.01168	0.45306	0.09995	0.23972
28	0.12635	0.13078	0.07729	0.00607	0.35497	0.05733	0.17515
29	0.36405	0.38954	0.22276	0.02289	0.92247	0.16604	0.48739
30	0.38151	0.36245	0.23726	0.02802	0.86735	0.18846	0.54764
31	0.36346	0.33489	0.23718	0.02149	0.87763	0.17398	0.53094
32	0.11609	0.09762	0.08971	0.00000	0.35030	0.03865	0.18309
33	0.25646	0.23073	0.16876	0.01681	0.60206	0.12436	0.37553
34	0.37391	0.34493	0.24235	0.02008	0.87763	0.17830	0.54507
35	0.36577	0.39351	0.22379	0.02149	0.91219	0.16908	0.48587

Table 4.5: Testing Set



(a) Training Set



(b) Testing Set

Figure 4.3: Training and Testing Set

Chapter 5

Traffic Matrix Estimation

As stated earlier in chapter 1, the type of statistical measurement and analysis performed on a network data flow depend on data availability and parameter of interest in our analysis. With this dataset, we only have our adjacency matrix and our Traffic flow matrix at each time interval. As such we are task here to first find the Routing Matrix (B) which will then be used in the traffic Matrix estimation. After obtaining our Routing Matrix, we can then apply various techniques needed to Estimate the Traffic Matrix.

5.1 Generating Routing Matrix B

We can find the Routing Matrix from the Adjacency Matrix by first assuming that the flow edges in the network are bi-direction. That is, given two edge i and j, data can flow bidirectional from i to j and also from j to i. As such for all i and j in our network G, we will have 86 flow directions for all the 43 edges in the network. These 86 flow directions are given in the adjacency list of the flow given in Table 4.1.

The flow volume given in the Traffic Flow matrix gives the total flow volumes at the sensor of the node. These flow volumes in fact represent the Net In-flow volume (Z_{+j}) at a node j or the Net Out-flow volume (Z_{i+}) at a node i. From Eqn 1.3, we can extrapolate from the nodes in the Adjacency list that

$$\begin{aligned}
z_{0+} &= z_{(0,1)} + z_{(0,14)} + z_{(0,25)} + z_{(0,29)} \\
z_{1+} &= z_{(1,0)} + z_{(1,2)} \\
z_{2+} &= z_{(2,1)} + z_{(2,3)} \\
&\vdots \\
z_{33+} &= z_{(33,30)} + z_{(33,34)} \\
z_{34+} &= z_{(34,31)} + z_{(34,33)} \\
z_{35+} &= z_{(35,28)} + z_{(35,29)}
\end{aligned}$$

$$\begin{bmatrix} z_{0+} \\ z_{1+} \\ z_{2+} \\ \vdots \\ z_{33+} \\ z_{34+} \\ z_{35+} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} z_{(0,1)} \\ z_{(0,14)} \\ z_{(0,25)} \\ \vdots \\ z_{(34,33)} \\ z_{(35,28)} \\ z_{(35,29)} \end{bmatrix}$$

$$[z_{i+}] = M_{i+} * [z_{(i,j)}]_{\{i+\}} \tag{5.1}$$

Applying this same method using Eqn 1.4 gives

$$[z_{+j}] = M_{+j} * [z_{(i,j)}]_{\{+j\}} \tag{5.2}$$

Infact, both equations (5.1 and 5.2) are equivalent to equation 1.5. Therefore, for this network, $[z_{i+}], [z_{+j}] \equiv X$, $M_{i+}, M_{+j} \equiv B$, and $[z_{(i,j)}]_{\{i+\}}, [z_{(i,j)}]_{\{+j\}} \equiv Z$. Even though $M_{i+} \neq M_{+j}$ and $[z_{(i,j)}]_{\{i+\}} \neq [z_{(i,j)}]_{\{+j\}}$, $[z_{i+}] = [z_{+j}]$ which implies that both equations (5.1 and 5.2) can be use for any estimation involving networks of this form. Nonetheless, for the convenience sake we will be using equation 5.1 for this analysis.
With these correspondence, we can obtain our Routing Matrix (B) from our Adjancency matrix using this algorithm:

1. Calculate the total number of links in the adjacency matrix.
2. Get the number of origin-destination pairs (nodes). Assume that all the edges are bidirectional.
3. Initialize a binary matrix with zeros to represent the routing matrix. The dimension of the binary matrix will be the Number of Vertices by the number of origin-destination pairs.
4. Create a list of all the links in the adjacency matrix.
5. Create a dictionary to map links to indice.
6. Set entries to 1 where links are traversed for each origin-destination pair.

With our Routing Matrix we can then move on to our Traffic Matrix Estimation where we will estimate the OD Matrix with Least Square Approaches.

5.2 Least Square Approaches

The Least Square Approaches minimizes the sum of the squared differences between the observed and predicted values in a linear regression model [1]. Given the linear equation,

$$X = BZ$$

the objective function in the Least Square Approaches is formulated to minimize the difference between the observed counts (X) and the predicted flow volumes (Z_{ij}) weighted by some measure of uncertainty or error (ϵ) [1], [12]. This is often done by assuming that the errors follow a Gaussian distribution [12]. The optimization problem aims to find the traffic matrix that best fits the observed link counts while considering the uncertainty. In our analysis, the observed counts X on the network links, at each time, are modeled using a simple form, represented by the equation:

$$X = B\mu + \epsilon \tag{5.3}$$

where X is the observed counts $[z_{i+}]$, B is the routing matrix M_{i+}, μ is the expected flow volumes for all (i, j) origin-destination pairs, and ϵ is the vector representing the errors.
To estimate the expected flow volumes μ, the Ordinary Least Squares (OLS) approach is often employed. The OLS method minimizes the squared difference between the observed counts X and the estimated counts $B\mu$ [12]. The objective of OLS is to find the values of μ that minimize the sum of squared errors. Given by [1] as

$$\min_{\mu}(x - B\mu)^T(x - B\mu)$$

The traffic flow for the 36 locations in the network where recorded in continuous quarter-hours. Therefore with the OLS method, given the time dependent equation,

$$\begin{gathered} X^{(t)} = B^{(t)}Z^{(t)} \\ Z^{(t)} = \left(\left(B^{(t)}\right)^T B^{(t)}\right)^{-1} \left(B^{(t)}\right)^T X^{(t)} \\ or \\ [z_{(i,j)}]^{(t)}_{\{i+\}} = \left(\left(M^{(t)}_{i+}\right)^T M^{(t)}_{i+}\right)^{-1} \left(M^{(t)}_{i+}\right)^T [z_{i+}]^{(t)} \end{gathered} \tag{5.4}$$

Nonetheless, because our network is static implies that the Routing Matrix B or M_{i+} will be the same for all t. As such, the equations in 5.4 can be simplified as:

$$\begin{gathered} X^{(t)} = BZ^{(t)} \\ Z^{(t)} = \left(B^T B\right)^{-1} B^T X^{(t)} \\ or \\ [z_{(i,j)}]^{(t)}_{\{i+\}} = \left(M^T_{i+} M_{i+}\right)^{-1} M^T_{i+}[z_{i+}]^{(t)} \end{gathered} \tag{5.5}$$

Applying equation 5.5 to the network data flow gives the Traffic Matrix values $Z_{(i,j)}$ in $[z_{(i,j)}]_{\{i+\}}$.
However, in this network data, $B^T B$ is invertible due to its singularity. As such, various techniques, such as incorporating additional data sources, imposing regularization, such as Ridge Regression or Lasso Regression, or priors on the flow volumes, using Singular Value Decomposition (SVD) or considering statistical assumptions about the error term ϵ, can be employed to address this issue.
An alternative to OLS approach involves using generalized least-squares to incorporate these measurements into the estimation process. The generalized least-squares formulation involves the equation

$$\begin{bmatrix} Z^{(0)} \\ X \end{bmatrix} = \begin{bmatrix} I \\ B \end{bmatrix} \mu + \begin{bmatrix} \xi \\ \epsilon \end{bmatrix} \tag{5.6}$$

where I is the identity matrix, B is a matrix that relates the traffic matrix μ to the measurements X, ξ and ϵ are independent error vectors, and Ψ and Σ are the covariance matrices of ξ and ϵ, respectively [1]. The objective is to minimize the squared difference between the measurements and the estimated values. Given by [1] as

$$\min_{\mu} \begin{bmatrix} Z^{(0)} - \mu \\ X - B\mu \end{bmatrix}^T \begin{bmatrix} \Psi^{-1} & 0 \\ 0 & \Sigma^{-1} \end{bmatrix} \begin{bmatrix} Z^{(0)} - \mu \\ X - B\mu \end{bmatrix}$$

The solution to this generalized least-squares problem is also given by [1] as

$$\hat{\mu} = \left(\Psi^{-1} + B^T\Sigma^{-1}B\right)^{-1} \left(\Psi^{-1}z^{(0)} + B^T\Sigma^{-1}x\right). \tag{5.7}$$

Here, $\hat{\mu}$ represents the estimated traffic matrix, $z^{(0)}$ is the initial set of measurements, x is the observed link count vector, and Ψ^{-1} and Σ^{-1} are the inverse covariance matrices of the error vectors ξ and ϵ [1]. The estimated traffic matrix $\hat{\mu}$ is a linear combination of the initial measurements $z^{(0)}$ and the observed link counts x. According to standard linear model theory and under the assumed model conditions, the mean of the estimator $\mathbb{E}(\hat{\mu})$ is equal to the true traffic matrix μ, and the covariance $\mathbb{V}(\hat{\mu}) = \left(\Psi^{-1} + B^T\Sigma^{-1}B\right)^{-1}$ represents the minimum covariance among all unbiased estimators [1]. The covariance can be used to quantify the uncertainty associated with the estimation of μ when combined with the estimator $\hat{\mu}$ [1]. The specific type of uncertainty depends on the error covariances Ψ and Σ. Typically, Σ is considered to be a matrix with diagonal entries, while the structure of Ψ depends on the sampling method used for the initial measurements $z^{(0)}$ [1]. The values of the components within Ψ and Σ can be determined using previous measurements or estimations of $\hat{\mu}$ from past instances.

5.3 Regularization Techniques

Regularization Techniques can be used here to deal with the singularity or ill-posed problems in our dataset. It involves adding a penalty term to the objective function to constrain the solution and narrow down the set of potential solutions in a meaningful way [13]. Two common regularization techniques used especially in the field of machine learning are L1 Regularization (Lasso Regression) and L2 Regularization (Ridge Regression).

In L1 regularization, also known as Lasso (Least Absolute Shrinkage and Selection Operator) regression, a penalty term is added to the loss function based on the absolute values of the model's coefficients [13]. The objective function for Lasso regression is given by:

$$\text{Loss}(y, \hat{y}) + \lambda \sum_{i=1}^{n} |w_i|$$

Here, $\text{Loss}(y, \hat{y})$ is the original loss function without regularization, measuring the difference between the predicted ($\hat{y}$) and actual (y) values, n is the number of features in the model, w_i represents the weight (coefficient) associated with the i-th feature, and λ is the regularization parameter, controlling the strength of the regularization [13]. The optimization problem associated with Lasso regression is to minimize this modified loss function with respect to the model parameters (w_i). The choice of λ

determines the trade-off between fitting the data well and keeping the model simple by shrinking some coefficients towards zero [13]. Cross-validation is often used to find an appropriate value for λ.
In L2 regularization, also known as Ridge regression, a penalty term is added to the loss function based on the squared values of the model's coefficients [13]. The objective function for Ridge regression is given by:

$$\text{Loss}(y, \hat{y}) + \lambda \sum_{i=1}^{n} w_i^2$$

Here, $\text{Loss}(y, \hat{y})$ is the original loss function without regularization, measuring the difference between the predicted ($\hat{y}$) and actual (y) values, n is the number of features in the model, w_i represents the weight (coefficient) associated with the i-th feature, and λ is the regularization parameter, controlling the strength of the regularization [13]. It's a hyperparameter that needs to be tuned. The optimization problem associated with Ridge regression is to minimize this modified loss function with respect to the model parameters (w_i). The choice of λ determines the trade-off between fitting the data well and keeping the model simple by controlling the size of the coefficients [13]. Like L1 regularization, the value of λ is often determined through cross-validation.
In the context of traffic matrix estimation, the tomogravity method utilizes regularization by replacing the least-squares criterion in the objective function with a penalized least-squares criterion [1]. The modified objective function takes the form:

$$(x - B\mu)^T (x - B\mu) + \lambda D(\mu || \mu^{(0)}) \tag{5.8}$$

Here, x represents the observed link count vector, B is a matrix that relates the traffic matrix μ to the link counts, μ is the estimated traffic matrix, λ is a regularization parameter, and $D(\mu||\mu^{(0)})$ is the penalty term [1].
The penalty term, $D(\mu||\mu^{(0)})$, measures the relative entropy "distance" between the estimated traffic matrix μ and a pre-specified vector $\mu^{(0)}$. Relative entropy, also known as Kullback-Leibler divergence, is a measure of dissimilarity between probability distributions [1]. It is defined as:

$$D(\mu||\mu^{(0)}) = \sum_{ij} \frac{\mu_{ij}}{\mu_{++}} log \frac{\mu_{ij}}{\mu_0} \tag{5.9}$$

where μ_{ij} represents the elements of the estimated traffic matrix μ, μ_{++} represents the total traffic within the network, and μ_0 represents the pre-specified vector with a specific multiplicative structure resembling a basic gravity model [1]. The tomogravity method incorporates the concept of a gravity model by defining $\mu^{(0)}$ with a multiplicative structure similar to a basic gravity model. The gravity model assumes that the logarithm of the expected flow volume (μ_{ij}) can be expressed as a linear combination of factors related to the origin (α_i), the destination (β_j), and a pre-specified gravity constant (γ_{ij}) [1]. The equation takes the form $\log \mu_{ij} = \alpha_i + \beta_j + \log \gamma_{ij}$. To estimate the values of α_i and β_j, a nonlinear least-squares algorithm is used with the constraint that the estimated origin and destination volumes (μ_{i+} and μ_{+j}) match the given values ($\hat{\mu}_{i+}$ and $\hat{\mu}_{+j}$) [1]. The resulting estimated traffic matrix ($\hat{\mu}$) will be unique if the given values and the gravity constant (γ_{ij}) are all positive.

The net out-flow and in-flow at vertices i and j, respectively, are multiplied to obtain the initial traffic matrix [1] $\mu^{(0)}$:

$$\mu_{ij}^{(0)} = Z_{i+}^{(0)} \times Z_{+j}^{(0)} \tag{5.10}$$

Additionally, there is a requirement that μ_{++} is equal to $Z_{++}^{(0)}$, which represents the total traffic within the network. These quantities can be calculated by summing the elements in the observed link count vector x in an appropriate manner. By incorporating the penalty term and the gravity model structure in the tomogravity method, the objective function aims to find an estimated traffic matrix μ that minimizes the penalized least-squares criterion while adhering to the constraints imposed by the gravity model [1]. The regularization parameter λ controls the trade-off between the fit to the observed data and the adherence to the gravity model structure. According to [2], $\lambda = (0.01)^2$ works well in practice and as such will be used in this work.

5.4 Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) can also be used to deal with the singularity or ill-posed problems in our dataset. SVD is a technique that decomposes a matrix into three other matrices, providing a useful tool for various numerical computations and data analysis [14]. It can also be applied to solve linear systems of equations, especially when the coefficient matrix is ill-conditioned or singular. For the singular routing matrix B, the SVD is expressed by [15] as:

$$B = U\Sigma V^T$$

Here, U, the left singular vectors matrix, contains the eigenvectors of BB^T. Also, the columns of U are orthogonal unit vectors, and they form a basis for the column space of B. Σ, the diagonal singular values matrix, is also a diagonal matrix with singular values on the diagonal . The singular values are related to the eigenvalues of BB^T and B^TB, providing information about the scaling and orientation of the transformation represented by B. V^T, the right singular vectors matrix, contains the eigenvectors of B^TB. Similar to U, the columns of V are orthogonal unit vectors and form a basis for the row space of B.[16]
SVD has various applications, and one of its key uses is in solving linear regression problems, especially when dealing with multicollinearity or when the matrix B is not full rank. For linear regression, SVD can be applied to find the Pseudo Inverse of singular matrix B. So let B^+ be the pseudo inverse of matrix B. B^+ can be obtain in the following way [16]:
SVD of Matrix B:

$$B = U\Sigma V^T$$

Multiply both sides by B^{-1}:

$$I = B^{-1}U\Sigma V^T$$

Multiply both sides by V:

$$V = B^{-1}U\Sigma$$

Multiply byΣ^{-1}:

$$V\Sigma^{-1} = B^{-1}U$$

Multiply by U^T:

$$V\Sigma^{-1}U^T = B^{-1}UU^T$$

$$B^{-1} = V\Sigma^{-1}U^T$$

$$\therefore B^+ = B^{-1} = V\Sigma^{-1}U^T$$

$$So\ for\ X = BZ$$

$$Z = B^{-1}X = B^+X$$

$$\therefore Z = V\Sigma^{-1}U^TX$$

where Z is the coefficient matrix, U, Σ, and V are the matrices obtained from the SVD of B, and X is the vector of observed values.
When dealing with singularity issues, SVD allows us to identify and handle multi-collinearity by examining the singular values. If some singular values are close to zero, it indicates that the corresponding columns (or rows) in U or V are nearly collinear and may be causing the singularity [16]. Regularization techniques, such as Ridge Regression, can also be combined with SVD to stabilize the solution and handle multi-collinearity effectively.
The `np.linalg.pinv()` function from numPy can be also simply used to calculate the pseudoinverse of matrix **B**, and then solves for Z using the formula $Z = B^+X$, where B^+ is the pseudoinverse of B.

5.5 Dataset Traffic Flow Estimates

To find the traffic flow volume estimate of the dataset, the tomogravity method will be used here, where will first calculate the tomographic parameters ($Z^{(0)}_{i+}$, $Z(0)_{j+}$, μ_{++}), define the pre-specified vector $\mu^{(0)}$ with gravity model structure, and then initialize an array to store estimated traffic matrices for each time interval. Next, we will use Ridge regression for the traffic matrix estimation with regularization for each time interval with $\lambda = (0.01)^2$, calculate the penalty term and the objective function value, and then stack the estimated traffic matrices along the third axis to get a 3D array of the traffic matrix estimate which is partially shown in table for the training set and table for the testing set. The flow estimate for all the 1,261 and 840 flow intervals can not be fully shown here and as such the first three and last three values have been shown in Table 5.1 for the Training set and Table 5.2 for the Testing set.
Some of the traffic volume estimates are negative and there is a need to impose the constraint $\mu \geq 0$ for all (i, j) pairs of flow volumes. To address this, we can modify the traffic matrix estimation method to enforce non-negativity constraints. One way to achieve this is to use non-negative least squares (NNLS) optimization. The NNLS algorithm in Python is available in the `scipy.optimize` module. The NNLS alogorithm can be used within the context of the tomogravity method by just replacing the Ridge regression with NNLS for the traffic matrix estimation. The `nnls` function from `scipy.optimize` enforces non-negativity constraints on the estimated traffic matrix for each time interval. The traffic volume estimates using the NNLS algorithm is show in Table 5.3 for the Training set and Table 5.4 for the Testing set.

(i,j)	t1	t2	t3		t1259	t1260	t1261
(0, 1)	-0.0015371	-0.0025414	-0.0034310	···	0.0152137	0.0205882	0.0152723
(0, 14)	-0.0015371	-0.0025414	-0.0034310	···	0.0152137	0.0205882	0.0152723
(0, 25)	-0.0015371	-0.0025414	-0.0034310	···	0.0152137	0.0205882	0.0152723
(0, 29)	-0.0015371	-0.0025414	-0.0034310	···	0.0152137	0.0205882	0.0152723
(1, 0)	-0.0054093	-0.0041485	-0.0031255	···	0.0301931	0.0479475	0.0697759
(1, 2)	-0.0054093	-0.0041485	-0.0031255	···	0.0301931	0.0479475	0.0697759
(2, 1)	-0.0033076	-0.0067173	-0.0040596	···	-0.0277209	0.0297326	0.0088262
(2, 3)	-0.0033076	-0.0067173	-0.0040596	···	-0.0277209	0.0297326	0.0088262
(3, 2)	-0.0122192	-0.0195053	-0.0071847	···	-0.0082694	0.0613304	0.0316622
(4, 5)	-0.0024440	-0.0009122	-0.0004095	···	-0.0316394	-0.0290212	-0.0213171
(4, 21)	-0.0024440	-0.0009122	-0.0004095	···	-0.0316394	-0.0290212	-0.0213171
(4, 25)	-0.0024440	-0.0009122	-0.0004095	···	-0.0316394	-0.0290212	-0.0213171
(4, 26)	-0.0024440	-0.0009122	-0.0004095	···	-0.0316394	-0.0290212	-0.0213171
(4, 31)	-0.0024440	-0.0009122	-0.0004095	···	-0.0316394	-0.0290212	-0.0213171
(5, 4)	-0.0058764	-0.0041485	-0.0045266	···	-0.0959099	-0.0863288	-0.0771107
(5, 6)	-0.0058764	-0.0041485	-0.0045266	···	-0.0959099	-0.0863288	-0.0771107
(6, 5)	-0.0000383	0.0035578	-0.0012573	···	-0.0821320	-0.0790896	-0.0701050
(6, 7)	-0.0000383	0.0035578	-0.0012573	···	-0.0821320	-0.0790896	-0.0701050
(7, 6)	-0.0136203	-0.0087637	-0.0160581	···	-0.1348331	-0.0666344	-0.0351224
(8, 9)	0.0130391	0.0210721	0.0150894	···	0.1703075	0.1780204	0.1624850
(8, 10)	0.0130391	0.0210721	0.0150894	···	0.1703075	0.1780204	0.1624850
(9, 8)	0.0240147	0.0364846	0.0244304	···	0.2506398	0.2550833	0.2386138
(9, 12)	0.0240147	0.0364846	0.0244304	···	0.2506398	0.2550833	0.2386138
(10, 8)	0.0170090	0.0199044	0.0211610	···	0.1817502	0.1838585	0.1867715
(10, 13)	0.0170090	0.0199044	0.0211610	···	0.1817502	0.1838585	0.1867715
(11, 12)	0.0480270	0.0542846	0.0656712	···	0.2457920	0.1864930	0.1899836
(12, 9)	0.0228471	0.0334488	0.0239633	···	0.2100066	0.1943671	0.1804663
(12, 11)	0.0228471	0.0334488	0.0239633	···	0.2100066	0.1943671	0.1804663
(13, 10)	0.0188772	0.0196709	0.0218616	···	0.1789479	0.1960018	0.1774305
(13, 14)	0.0188772	0.0196709	0.0218616	···	0.1789479	0.1960018	0.1774305
(14, 0)	0.0106337	0.0103907	0.0137892	···	0.0936267	0.1031573	0.0871347
(14, 13)	0.0106337	0.0103907	0.0137892	···	0.0936267	0.1031573	0.0871347
(14, 25)	0.0106337	0.0103907	0.0137892	···	0.0936267	0.1031573	0.0871347
(14, 26)	0.0106337	0.0103907	0.0137892	···	0.0936267	0.1031573	0.0871347
(14, 31)	0.0106337	0.0103907	0.0137892	···	0.0936267	0.1031573	0.0871347
(15, 16)	-0.0126862	-0.0223074	-0.0239975	···	-0.1226905	-0.1217433	-0.1303954
(16, 15)	0.0111709	0.0140663	0.0132212	···	-0.0802638	-0.0905323	-0.0855176
(16, 17)	0.0111709	0.0140663	0.0132212	···	-0.0802638	-0.0905323	-0.0855176
(17, 16)	0.0163084	0.0175692	0.0155565	···	-0.0765274	-0.0870294	-0.0820147
(17, 18)	0.0163084	0.0175692	0.0155565	···	-0.0765274	-0.0870294	-0.0820147
(18, 17)	0.0184101	0.0206050	0.0178917	···	-0.0774615	-0.0919334	-0.0796795
(18, 19)	0.0184101	0.0206050	0.0178917	···	-0.0774615	-0.0919334	-0.0796795
(19, 18)	0.0331221	0.0297124	0.0272327	···	-0.0578455	-0.0601741	-0.0486208
(19, 20)	0.0331221	0.0297124	0.0272327	···	-0.0578455	-0.0601741	-0.0486208
(20, 19)	0.0328886	0.0264431	0.0269991	···	-0.0685876	-0.0648446	-0.0553930
(20, 21)	0.0328886	0.0264431	0.0269991	···	-0.0685876	-0.0648446	-0.0553930
(21, 4)	0.0208818	0.0199943	0.0137334	···	-0.0226181	-0.0230819	-0.0232601
(21, 20)	0.0208818	0.0199943	0.0137334	···	-0.0226181	-0.0230819	-0.0232601
(21, 25)	0.0208818	0.0199943	0.0137334	···	-0.0226181	-0.0230819	-0.0232601
(21, 29)	0.0208818	0.0199943	0.0137334	···	-0.0226181	-0.0230819	-0.0232601
(22, 23)	-0.0152173	-0.0141901	-0.0171369	···	-0.0676535	-0.0648446	-0.0509560
(22, 25)	-0.0152173	-0.0141901	-0.0171369	···	-0.0676535	-0.0648446	-0.0509560
(23, 22)	-0.0149838	-0.0174594	-0.0199392	···	-0.0905388	-0.0900652	-0.0885534
(23, 24)	-0.0149838	-0.0174594	-0.0199392	···	-0.0905388	-0.0900652	-0.0885534
(24, 23)	-0.0236242	-0.0237646	-0.0232085	···	-0.1164600	-0.1162199	-0.1086364
(24, 25)	-0.0236242	-0.0237646	-0.0232085	···	-0.1164600	-0.1162199	-0.1086364
(25, 0)	0.0031918	0.0025512	0.0013756	···	0.0110290	0.0234414	0.0335483
(25, 4)	0.0031918	0.0025512	0.0013756	···	0.0110290	0.0234414	0.0335483
(25, 14)	0.0031918	0.0025512	0.0013756	···	0.0110290	0.0234414	0.0335483
(25, 21)	0.0031918	0.0025512	0.0013756	···	0.0110290	0.0234414	0.0335483
(25, 22)	0.0031918	0.0025512	0.0013756	···	0.0110290	0.0234414	0.0335483
(25, 24)	0.0031918	0.0025512	0.0013756	···	0.0110290	0.0234414	0.0335483
(25, 29)	0.0031918	0.0025512	0.0013756	···	0.0110290	0.0234414	0.0335483
(26, 4)	-0.0129474	-0.0139751	-0.0128260	···	-0.0055590	0.0023853	0.0010580
(26, 14)	-0.0129474	-0.0139751	-0.0128260	···	-0.0055590	0.0023853	0.0010580
(26, 29)	-0.0129474	-0.0139751	-0.0128260	···	-0.0055590	0.0023853	0.0010580
(27, 28)	-0.0351034	-0.0367852	-0.0380083	···	-0.0470325	-0.0605631	-0.0570725
(28, 27)	-0.0212890	-0.0183935	-0.0201727	···	0.0180499	-0.0274808	-0.0570276
(28, 35)	-0.0212890	-0.0183935	-0.0201727	···	0.0180499	-0.0274808	-0.0570276
(29, 0)	-0.0019770	-0.0060499	-0.0008766	···	0.1127762	0.0835407	0.0834916
(29, 21)	-0.0019770	-0.0060499	-0.0008766	···	0.1127762	0.0835407	0.0834916
(29, 25)	-0.0019770	-0.0060499	-0.0008766	···	0.1127762	0.0835407	0.0834916
(29, 26)	-0.0019770	-0.0060499	-0.0008766	···	0.1127762	0.0835407	0.0834916
(29, 35)	-0.0019770	-0.0060499	-0.0008766	···	0.1127762	0.0835407	0.0834916
(30, 32)	-0.0107804	-0.0146571	-0.0089636	···	-0.0797967	-0.0720839	-0.0717396
(30, 33)	-0.0107804	-0.0146571	-0.0089636	···	-0.0797967	-0.0720839	-0.0717396
(31, 4)	-0.0084325	-0.0085261	-0.0092452	···	-0.0563124	-0.0483681	-0.0591923
(31, 14)	-0.0084325	-0.0085261	-0.0092452	···	-0.0563124	-0.0483681	-0.0591923
(31, 34)	-0.0084325	-0.0085261	-0.0092452	···	-0.0563124	-0.0483681	-0.0591923
(32, 30)	-0.0542514	-0.0554662	-0.0585574	···	-0.2884843	-0.3197618	-0.3181394
(33, 30)	-0.0149838	-0.0195611	-0.0164363	···	-0.1036162	-0.1017414	-0.1095705
(33, 34)	-0.0149838	-0.0195611	-0.0164363	···	-0.1036162	-0.1017414	-0.1095705
(34, 31)	-0.0114810	-0.0179265	-0.0115323	···	-0.0762939	-0.0692816	-0.0754760
(34, 33)	-0.0114810	-0.0179265	-0.0115323	···	-0.0762939	-0.0692816	-0.0754760
(35, 28)	-0.0082116	-0.0139566	-0.0010237	···	0.2999134	0.2048757	0.2383803
(35, 29)	-0.0082116	-0.0139566	-0.0010237	···	0.2999134	0.2048757	0.2383803

Table 5.1: Traffic Estimates for the Training Set

(i,j)	t1	t2	t3		t838	t839	t840
(0, 1)	0.0114785	-0.0015830	-0.0075746	· · ·	-0.0137085	-0.0081229	-0.0108750
(0, 14)	0.0114785	-0.0015830	-0.0075746	· · ·	-0.0137085	-0.0081229	-0.0108750
(0, 25)	0.0114785	-0.0015830	-0.0075746	· · ·	-0.0137085	-0.0081229	-0.0108750
(0, 29)	0.0114785	-0.0015830	-0.0075746	· · ·	-0.0137085	-0.0081229	-0.0108750
(1, 0)	0.1086598	0.1163984	0.1020802	· · ·	-0.0080338	-0.0080721	-0.0065704
(1, 2)	0.1086598	0.1163984	0.1020802	· · ·	-0.0080338	-0.0080721	-0.0065704
(2, 1)	0.0412404	0.0235581	-0.0218898	· · ·	-0.0277425	-0.0273521	-0.0262169
(2, 3)	0.0000000	0.0000000	0.0000000	· · ·	0.0000000	0.0000000	0.0000000
(3, 2)	0.1075579	0.0427062	0.0430266	· · ·	-0.0048582	-0.0049349	-0.0005306
(4, 5)	-0.0141702	-0.0019203	0.0018803	· · ·	0.0148150	0.0105961	0.0115704
(4, 21)	-0.0141702	-0.0019203	0.0018803	· · ·	0.0148150	0.0105961	0.0115704
(4, 25)	-0.0141702	-0.0019203	0.0018803	· · ·	0.0148150	0.0105961	0.0115704
(4, 26)	-0.0141702	-0.0019203	0.0018803	· · ·	0.0148150	0.0105961	0.0115704
(4, 31)	-0.0141702	-0.0019203	0.0018803	· · ·	0.0148150	0.0105961	0.0115704
(5, 4)	-0.0615792	-0.0477690	-0.0326632	· · ·	0.0176538	0.0152803	0.0188837
(5, 6)	-0.0615792	-0.0477690	-0.0326632	· · ·	0.0176538	0.0152803	0.0188837
(6, 5)	-0.0797190	-0.0492977	-0.0018078	· · ·	0.0278335	0.0618497	0.0499081
(6, 7)	0.0000000	0.0000000	0.0000000	· · ·	0.0000000	0.0000000	0.0000000
(7, 6)	-0.0605710	-0.0231443	0.0037965	· · ·	-0.0202701	0.0132791	-0.0023987
(8, 9)	0.1544305	0.1248053	0.1025472	· · ·	0.0097140	0.0068734	0.0048722
(8, 10)	0.1544305	0.1248053	0.1025472	· · ·	0.0097140	0.0068734	0.0048722
(9, 8)	0.2268230	0.1810846	0.1660658	· · ·	0.0052770	0.0024364	0.0009023
(9, 12)	0.2268230	0.1810846	0.1660658	· · ·	0.0052770	0.0024364	0.0009023
(10, 8)	0.1791841	0.1469901	0.1219297	· · ·	0.0055106	0.0089751	0.0016029
(10, 13)	0.1791841	0.1469901	0.1219297	· · ·	0.0055106	0.0089751	0.0016029
(11, 12)	0.1234368	0.1080896	0.1593158	· · ·	-0.0300776	-0.0180116	-0.0234148
(12, 9)	0.2981040	0.2612737	0.2289025	· · ·	0.0077514	0.0044056	-0.0061348
(12, 11)	0.0000000	0.0000000	0.0000000	· · ·	0.0000000	0.0000000	0.0000000
(13, 10)	0.1584004	0.1682408	0.1137564	· · ·	0.0059776	0.0043046	-0.0011994
(13, 14)	0.1584004	0.1682408	0.1137564	· · ·	0.0059776	0.0043046	-0.0011994
(14, 0)	0.0848469	0.0810300	0.0810006	· · ·	0.0042594	0.0063925	0.0035370
(14, 13)	0.0848469	0.0810300	0.0810006	· · ·	0.0042594	0.0063925	0.0035370
(14, 25)	0.0848469	0.0810300	0.0810006	· · ·	0.0042594	0.0063925	0.0035370
(14, 26)	0.0848469	0.0810300	0.0810006	· · ·	0.0042594	0.0063925	0.0035370
(14, 31)	0.0848469	0.0810300	0.0810006	· · ·	0.0042594	0.0063925	0.0035370
(15, 16)	-0.1222183	-0.1146812	-0.1283715	· · ·	-0.0347478	-0.0315553	-0.0402277
(16, 15)	0.0000000	0.0000000	0.0000000	· · ·	0.0000000	0.0000000	0.0000000
(16, 17)	-0.1535089	-0.1202855	-0.1064213	· · ·	0.0997553	0.0655859	0.0760615
(17, 16)	-0.0769918	-0.0437991	-0.0529798	· · ·	0.0638916	0.0428361	0.0490083
(17, 18)	-0.0769918	-0.0437991	-0.0529798	· · ·	0.0638916	0.0428361	0.0490083
(18, 17)	-0.0706867	-0.0501042	-0.0469082	· · ·	0.0365693	0.0267229	0.0328951
(18, 19)	-0.0706867	-0.0501042	-0.0469082	· · ·	0.0365693	0.0267229	0.0328951
(19, 18)	-0.0391609	-0.0066687	-0.0018380	· · ·	0.0578200	0.0512430	0.0564811
(19, 20)	-0.0391609	-0.0066687	-0.0018380	· · ·	0.0578200	0.0512430	0.0564811
(20, 19)	-0.0428973	-0.0178779	-0.0198194	· · ·	0.0566523	0.0458719	0.0529782
(20, 21)	-0.0428973	-0.0178779	-0.0198194	· · ·	0.0566523	0.0458719	0.0529782
(21, 4)	-0.0102398	-0.0041518	0.0030510	· · ·	0.0356831	0.0278407	0.0366483
(21, 20)	-0.0102398	-0.0041518	0.0030510	· · ·	0.0356831	0.0278407	0.0366483
(21, 25)	-0.0102398	-0.0041518	0.0030510	· · ·	0.0356831	0.0278407	0.0366483
(21, 29)	-0.0102398	-0.0041518	0.0030510	· · ·	0.0356831	0.0278407	0.0366483
(22, 23)	-0.0438314	-0.0374939	-0.0256575	· · ·	-0.0302186	-0.0202154	-0.0191807
(22, 25)	-0.0438314	-0.0374939	-0.0256575	· · ·	-0.0302186	-0.0202154	-0.0191807
(23, 22)	-0.0744231	-0.0732231	-0.0534468	· · ·	-0.0311527	-0.0216165	-0.0222166
(23, 24)	-0.0744231	-0.0732231	-0.0534468	· · ·	-0.0311527	-0.0216165	-0.0222166
(24, 23)	-0.1017454	-0.1101199	-0.1006187	· · ·	-0.0442300	-0.0363286	-0.0352939
(24, 25)	-0.1017454	-0.1101199	-0.1006187	· · ·	-0.0442300	-0.0363286	-0.0352939
(25, 0)	0.0392538	0.0429328	0.0597262	· · ·	0.0057114	0.0027646	0.0026599
(25, 4)	0.0392538	0.0429328	0.0597262	· · ·	0.0057114	0.0027646	0.0026599
(25, 14)	0.0392538	0.0429328	0.0597262	· · ·	0.0057114	0.0027646	0.0026599
(25, 21)	0.0392538	0.0429328	0.0597262	· · ·	0.0057114	0.0027646	0.0026599
(25, 22)	0.0392538	0.0429328	0.0597262	· · ·	0.0057114	0.0027646	0.0026599
(25, 24)	0.0392538	0.0429328	0.0597262	· · ·	0.0057114	0.0027646	0.0026599
(25, 29)	0.0392538	0.0429328	0.0597262	· · ·	0.0057114	0.0027646	0.0026599
(26, 4)	-0.0194132	-0.0343375	-0.0647450	· · ·	-0.0198347	-0.0117646	-0.0151227
(26, 14)	-0.0194132	-0.0343375	-0.0647450	· · ·	-0.0198347	-0.0117646	-0.0151227
(26, 29)	-0.0194132	-0.0343375	-0.0647450	· · ·	-0.0198347	-0.0117646	-0.0151227
(27, 28)	-0.0638402	-0.1356973	-0.1367779	· · ·	0.0072844	-0.0619119	-0.0388266
(28, 27)	0.0000000	0.0000000	0.0000000	· · ·	0.0000000	0.0000000	0.0000000
(28, 35)	-0.1156799	-0.1342962	-0.1414482	· · ·	-0.0926589	-0.0712524	-0.0752545
(29, 0)	0.0870888	0.0864479	0.0843635	· · ·	-0.0076974	-0.0066852	-0.0072055
(29, 21)	0.0870888	0.0864479	0.0843635	· · ·	-0.0076974	-0.0066852	-0.0072055
(29, 25)	0.0870888	0.0864479	0.0843635	· · ·	-0.0076974	-0.0066852	-0.0072055
(29, 26)	0.0870888	0.0864479	0.0843635	· · ·	-0.0076974	-0.0066852	-0.0072055
(29, 35)	0.0870888	0.0864479	0.0843635	· · ·	-0.0076974	-0.0066852	-0.0072055
(30, 32)	0.0000000	0.0000000	0.0000000	· · ·	0.0000000	0.0000000	0.0000000
(30, 33)	-0.1324928	-0.0978683	-0.0854052	· · ·	0.0142898	0.0146802	0.0167494
(31, 4)	-0.0465025	-0.0408763	-0.0211531	· · ·	0.0018056	0.0022471	0.0034039
(31, 14)	-0.0465025	-0.0408763	-0.0211531	· · ·	0.0018056	0.0022471	0.0034039
(31, 34)	-0.0465025	-0.0408763	-0.0211531	· · ·	0.0018056	0.0022471	0.0034039
(32, 30)	-0.3094952	-0.3103646	-0.2885610	· · ·	-0.0907908	-0.0665821	-0.0635789
(33, 30)	-0.1113199	-0.0949408	-0.0971158	· · ·	-0.0108361	-0.0050363	-0.0091392
(33, 34)	-0.1113199	-0.0949408	-0.0971158	· · ·	-0.0108361	-0.0050363	-0.0091392
(34, 31)	-0.0746566	-0.0466014	-0.0408365	· · ·	0.0080793	0.0029035	0.0058063
(34, 33)	-0.0746566	-0.0466014	-0.0408365	· · ·	0.0080793	0.0029035	0.0058063
(35, 28)	0.2046382	0.2135444	0.2045972	· · ·	-0.0194765	-0.0176466	-0.0205819
(35, 29)	0.2046382	0.2135444	0.2045972	· · ·	-0.0194765	-0.0176466	-0.0205819

Table 5.2: Traffic Estimates for the Testing Set

(i,j)	t1	t2	t3		t1259	t1260	t1261
(0, 1)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.4226997
(0, 14)	0.0000000	0.0476413	0.0448389	···	0.0000000	0.4245680	0.0000000
(0, 25)	0.0000000	0.0000000	0.0000000	···	0.3713218	0.0000000	0.0000000
(0, 29)	0.0509108	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(1, 0)	0.0462401	0.0495096	0.0000000	···	0.0000000	0.0000000	0.5011677
(1, 2)	0.0000000	0.0000000	0.0523120	···	0.3708547	0.4381130	0.0000000
(2, 1)	0.0504437	0.0443718	0.0504437	···	0.2550210	0.4016815	0.3792620
(2, 3)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(3, 2)	0.0448389	0.0382999	0.0513779	···	0.3021952	0.4035497	0.3932742
(4, 5)	0.0448389	0.0532461	0.0000000	···	0.0000000	0.0000000	0.0000000
(4, 21)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(4, 25)	0.0000000	0.0000000	0.0565156	···	0.0000000	0.0000000	0.2550210
(4, 26)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.1971042	0.0000000
(4, 31)	0.0000000	0.0000000	0.0000000	···	0.1522653	0.0000000	0.0000000
(5, 4)	0.0453059	0.0495096	0.0495096	···	0.1186362	0.1695469	0.0000000
(5, 6)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.2073797
(6, 5)	0.0569827	0.0649229	0.0560486	···	0.1461934	0.1840262	0.2213919
(6, 7)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(7, 6)	0.0434376	0.0490425	0.0425035	···	0.1756189	0.2755722	0.3264830
(8, 9)	0.0831387	0.0999533	0.0887436	···	0.6510976	0.6982718	0.6865950
(8, 10)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(9, 8)	0.1050911	0.1307800	0.1074264	···	0.8117702	0.8524054	0.8388603
(9, 12)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(10, 8)	0.0910789	0.0976179	0.1008874	···	0.6739841	0.7099486	0.7351705
(10, 13)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(11, 12)	0.1050911	0.1120972	0.1242410	···	0.5562821	0.5287249	0.5516114
(12, 9)	0.1027557	0.1247081	0.1064923	···	0.7304998	0.7309668	0.7225596
(12, 11)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(13, 10)	0.0948155	0.0971509	0.1022887	···	0.6683793	0.7342363	0.7164876
(13, 14)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(14, 0)	0.1102289	0.1097618	0.1275105	···	0.7786081	0.8580103	0.7972910
(14, 13)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(14, 25)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(14, 26)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(14, 31)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(15, 16)	0.0443718	0.0354974	0.0345633	···	0.1877627	0.2204577	0.2312004
(16, 15)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(16, 17)	0.0794021	0.0859411	0.0850070	···	0.1499299	0.1611397	0.1905652
(17, 16)	0.0896777	0.0929472	0.0896777	···	0.1574031	0.1681457	0.1975712
(17, 18)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(18, 17)	0.0938814	0.0990191	0.0943484	···	0.1555348	0.1583372	0.2022419
(18, 19)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(19, 18)	0.1233069	0.1172349	0.1130313	···	0.1947688	0.2218589	0.2643624
(19, 20)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(20, 19)	0.1228398	0.1106959	0.1125642	···	0.1732835	0.2125175	0.2508174
(20, 21)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(21, 4)	0.1405885	0.1377861	0.1134984	···	0.2199907	0.2498832	0.2685661
(21, 20)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(21, 25)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(21, 29)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(22, 23)	0.0266231	0.0294255	0.0242877	···	0.1751518	0.2125175	0.2596917
(22, 25)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(23, 22)	0.0270901	0.0228865	0.0186829	···	0.1293788	0.1620738	0.1844932
(23, 24)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(24, 23)	0.0098085	0.0102756	0.0121439	···	0.0775339	0.1097618	0.1443251
(24, 25)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 0)	0.0794021	0.0756656	0.0681924	···	0.3876693	0.5063055	0.5964503
(25, 4)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 14)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 21)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 22)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 24)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 29)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(26, 4)	0.0182158	0.0158804	0.0200841	···	0.2937879	0.3493695	0.3647828
(26, 14)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(26, 29)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(27, 28)	0.0219524	0.0210182	0.0205511	···	0.2634283	0.2816441	0.3045306
(28, 27)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(28, 35)	0.0144792	0.0210182	0.0182158	···	0.3465670	0.2872489	0.2475479
(29, 0)	0.0471742	0.0275572	0.0541803	···	0.8743578	0.7599253	0.7790752
(29, 21)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(29, 25)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(29, 26)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(29, 35)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(30, 32)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(30, 33)	0.0354974	0.0284914	0.0406352	···	0.1508641	0.1980383	0.2181224
(31, 4)	0.0317609	0.0322279	0.0308267	···	0.1415227	0.1971042	0.1840262
(31, 14)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(31, 34)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(32, 30)	0.0028024	0.0023354	0.0000000	···	0.0219524	0.0224194	0.0434376
(33, 30)	0.0270901	0.0186829	0.0256889	···	0.1032228	0.1387202	0.1424568
(33, 34)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(34, 31)	0.0340962	0.0219524	0.0354974	···	0.1578702	0.2036432	0.2106492
(34, 33)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(35, 28)	0.0406352	0.0298926	0.0565156	···	0.9103223	0.7519851	0.8383933
(35, 29)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000

Table 5.3: NNLS Traffic Estimates for the Training Set

(i,j)	t1	t2	t3		t838	t839	t840
(0, 1)	0.4016815	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(0, 14)	0.0000000	0.0000000	0.0000000	···	0.0523120	0.0000000	0.0000000
(0, 25)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(0, 29)	0.0000000	0.3801962	0.3554414	···	0.0000000	0.0555815	0.0420364
(1, 0)	0.5730967	0.6193368	0.5899113	···	0.0000000	0.0719290	0.0723961
(1, 2)	0.0000000	0.0000000	0.0000000	···	0.0910789	0.0000000	0.0000000
(2, 1)	0.3970107	0.4100887	0.3638487	···	0.0794021	0.0607193	0.0593181
(2, 3)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(3, 2)	0.4633349	0.4292387	0.4287716	···	0.1022887	0.0831387	0.0850070
(4, 5)	0.0000000	0.0000000	0.0000000	···	0.1812237	0.1410556	0.1433909
(4, 21)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(4, 25)	0.0000000	0.3769267	0.0000000	···	0.0000000	0.0000000	0.0000000
(4, 26)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(4, 31)	0.2849136	0.0000000	0.3951425	···	0.0000000	0.0000000	0.0000000
(5, 4)	0.2326016	0.0000000	0.3204110	···	0.1424568	0.1186362	0.1233069
(5, 6)	0.0000000	0.2909855	0.0000000	···	0.0000000	0.0000000	0.0000000
(6, 5)	0.2760392	0.3372256	0.3839327	···	0.1349837	0.1499299	0.1354507
(6, 7)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(7, 6)	0.2951892	0.3633816	0.3895376	···	0.0868753	0.1013545	0.0831387
(8, 9)	0.6646427	0.6361513	0.5908454	···	0.1265764	0.1018216	0.0952826
(8, 10)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(9, 8)	0.8094348	0.7487156	0.7178888	···	0.1177020	0.0929472	0.0873424
(9, 12)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(10, 8)	0.7141523	0.6805231	0.6296123	···	0.1181691	0.1060252	0.0887436
(10, 13)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(11, 12)	0.4792153	0.4946287	0.5450724	···	0.0770668	0.0700607	0.0621205
(12, 9)	0.6539000	0.6478281	0.6146660	···	0.1148996	0.0924801	0.0794021
(12, 11)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(13, 10)	0.6725829	0.7230266	0.6132648	···	0.1191032	0.0966838	0.0831387
(13, 14)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(14, 0)	0.7800093	0.7916861	0.7907520	···	0.1284447	0.1200374	0.1032228
(14, 13)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(14, 25)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(14, 26)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(14, 31)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(15, 16)	0.2335357	0.2718356	0.2573564	···	0.0723961	0.0565156	0.0453059
(16, 15)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(16, 17)	0.2022419	0.2662307	0.2793087	···	0.2069127	0.1536665	0.1616067
(17, 16)	0.2017749	0.2989257	0.2797758	···	0.2349369	0.1737506	0.1835591
(17, 18)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(18, 17)	0.2143858	0.2863148	0.2919197	···	0.1802896	0.1415227	0.1513312
(18, 19)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(19, 18)	0.2774404	0.3731901	0.3820645	···	0.2227931	0.1905652	0.1985054
(19, 20)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(20, 19)	0.2699673	0.3507707	0.3461000	···	0.2204577	0.1798225	0.1914993
(20, 21)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(21, 4)	0.3148062	0.3699206	0.3979449	···	0.2498832	0.1994395	0.2321345
(21, 20)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(21, 25)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(21, 29)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(22, 23)	0.2680990	0.3115367	0.3344232	···	0.0467071	0.0476413	0.0471742
(22, 25)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(23, 22)	0.2069127	0.2400747	0.2788417	···	0.0448389	0.0448389	0.0411023
(23, 24)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(24, 23)	0.1522653	0.1662774	0.1844932	···	0.0186829	0.0154134	0.0149463
(24, 25)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 0)	0.6305465	0.6870621	0.8038300	···	0.1471275	0.1074264	0.1041569
(25, 4)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 14)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 21)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 22)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 24)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(25, 29)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(26, 4)	0.2975245	0.2835124	0.1914993	···	0.0476413	0.0527791	0.0401681
(26, 14)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(26, 29)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(27, 28)	0.2919197	0.2508174	0.2489491	···	0.1144325	0.0261560	0.0467071
(28, 27)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(28, 35)	0.2400747	0.2522186	0.2442784	···	0.0144792	0.0168146	0.0102756
(29, 0)	0.7912191	0.8187763	0.8075666	···	0.0686595	0.0546474	0.0495096
(29, 21)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(29, 25)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(29, 26)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(29, 35)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(30, 32)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(30, 33)	0.2232602	0.2886502	0.3003270	···	0.1214386	0.1027557	0.1022887
(31, 4)	0.2162541	0.2638954	0.3222793	···	0.1125642	0.0948155	0.0957496
(31, 14)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(31, 34)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(32, 30)	0.0462401	0.0761326	0.0971509	···	0.0163475	0.0214853	0.0219524
(33, 30)	0.1331154	0.1966371	0.1914993	···	0.0854741	0.0780009	0.0672583
(33, 34)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(34, 31)	0.2064456	0.2933209	0.3040635	···	0.1233069	0.0938814	0.0971509
(34, 33)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000
(35, 28)	0.7650631	0.8136385	0.7949556	···	0.0681924	0.0527791	0.0443718
(35, 29)	0.0000000	0.0000000	0.0000000	···	0.0000000	0.0000000	0.0000000

Table 5.4: NNLS Traffic Estimates for the Testing Set

Chapter 6

Predictions Using Machine Algorithms

The traffic indices have 48 features which can be used to make predictions on the data using machine learning models. This can be achieved by training the models with the `tra_X_tr` and `tra_Y_tr` variables and then making the predictions using the `tra_X_te` and `tra_Y_te` variables of the datasets. However, these variables are composed of Sparse Matrices which cannot be fitted directly into our models. As such, these Sparse Matrices will be converted into a DataFrame and the respective features of each node in the `tra_X_tr` and `tra_X_te` variable traffic indices will then be matched to their corresponding flow values in the `tra_Y_tr` and `tra_Y_te` traffic flows. We will then train machine learning algorithms with the Dataframe data corresponding to the `tra_X_tr` and `tra_Y_tr` variables and then test the algorithms with the Dataframe data corresponding to the `tra_X_te` and `tra_Y_te` variables.

6.1 Machine Learning Regression Models

The traffic flow data in the `y_train` and `y_test` sets are all continuous values and as such, machine learning regression models will be used to make predictions on the data. The regression models which will be used here include: Support Vector Regressor (SVR), Linear Regressor, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, K-Neighbors Regressor, XGBoost Regressor, and Deep Neural Multilayer Perceptron (MLP) regressor. After training and testing, performance evaluation metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), R-squared (R^2), Explained Variance Score (EVS), Median Absolute Error (MedAE), Max Error metrics were then used to evaluate the performance of the individual models.

6.1.1 Support Vector Regressor (SVR)

SVR is a machine learning algorithm used for regression tasks. It is a variation of the Support Vector Machine (SVM) algorithm, which is primarily used for classification. In SVR, the goal is to find a function that can approximate the mapping from input features (x) to the target variable (y) while minimizing the prediction error [17]. The main idea is to find a hyperplane in the feature space that best fits the training data while keeping the errors (also called residuals) within a certain margin.

Mathematically, the SVR aims to solve the following optimization problem as shown in [17]:

$$\text{Minimize: } \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

subject to the following constraints for each training sample:

$$y_i - (w^T \cdot x_i + b) \leq \epsilon + \xi_i$$

$$(w^T \cdot x_i + b) - y_i \leq \epsilon + \xi_i^*$$

where:

- w is a weight vector that defines the hyperplane in the feature space
- b is the bias term (also called the intercept)
- C is a hyperparameter that controls the trade-off between maximizing the margin and minimizing the error. It is a regularization parameter.
- ξ_i and ξ_i^* are slack variables that allow for some training samples to violate the margin constraint by a certain amount (ξ_i and ξ_i^* must be non-negative).
- N is the number of training samples.
- ϵ is the size of the margin (also called the tube) within which errors are tolerated.

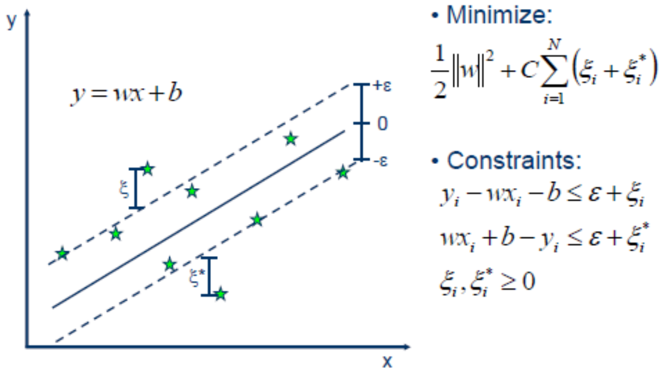

y
x
$y = wx + b$
+ε
0
-ε
ξ_i
ξ_i^*
• Minimize:
$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$
• Constraints:
$y_i - wx_i - b \leq \varepsilon + \xi_i$
$wx_i + b - y_i \leq \varepsilon + \xi_i^*$
$\xi_i, \xi_i^* \geq 0$

Figure 6.1: Support Vector Regressor

The optimization problem aims to find the best values for w, b, ξ_i, and ξ_i^* such that the training errors are minimized, and the margin is maximized [17]. The training samples that fall within the margin or violate the margin constraint will have non-zero values for ξ_i and ξ_i^*, whereas samples outside the margin will have $\xi_i = \xi_i^* = 0$ [17]. Once the optimization problem is solved, the SVR model can be used to predict the target variable for new input data by computing $y = w^T \cdot x + b$, where w is the weight vector and b is the bias term. In practice, the SVR can be further extended to handle non-linear relationships by using kernel methods, where the data is implicitly mapped to a higher-dimensional space [17]. This allows the SVR to handle more complex data distributions and achieve better performance in regression tasks.

6.1.2 Linear Regression

Linear regression is a simple and widely used supervised learning algorithm used for regression tasks. It aims to establish a linear relationship between the input features and the target output. The basic idea behind linear regression is to find the best-fitting line that minimizes the sum of the squared differences between the predicted values and the actual target values [18].
Mathematically, the linear regression model can be represented as: Given a training dataset with input feature vectors $X = \{x_1, x_2, ..., x_n\}$ and corresponding target output values $y = \{y_1, y_2, ..., y_n\}$, where x_i is the input feature vector of dimension d and y_i is the target output for the i-th sample [18]. The linear regression model is represented in [18] by the equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_d x_d$$

Here, y is the predicted target output for a given input feature vector x, β_0 is the y-intercept (the value of y when all input features are zero), and $\beta_1, \beta_2, ..., \beta_d$ are the coefficients corresponding to each input feature [18]. The goal of linear regression is to find the optimal values for $\beta_0, \beta_1, \beta_2, ..., \beta_d$ that minimize the sum of the squared differences between the predicted values and the actual target values [18]. This optimization process is typically done using the method of least squares. Once the coefficients are determined, the linear regression model can be used to make predictions on new, unseen data by plugging the input feature values into the equation.

6.1.3 Decision Tree Regressor

A Decision Tree Regressor is a supervised learning algorithm used for regression tasks. It works by recursively partitioning the input feature space into subsets and fitting a simple model (usually a constant value) to each subset [19]. This creates a tree-like structure where each internal node represents a decision based on a specific feature, and each leaf node represents a predicted output value.
Given a training dataset with input feature vectors $X = \{x_1, x_2, ..., x_n\}$ and corresponding target output values $y = \{y_1, y_2, ..., y_n\}$, where x_i is the input feature vector and y_i is the target output for the i-th sample. The Decision Tree Regressor recursively partitions the feature space by selecting the best feature and a corresponding split point that maximizes the homogeneity (reduction in variance) of the target output values within each subset [20]. At each internal node, a decision is made based on a specific feature and its split point. If a data point satisfies the condition, it moves to the left child node; otherwise, it moves to the right child node.
The recursive partitioning process continues until a stopping criterion is met, such as reaching a maximum tree depth or having a minimum number of samples in each leaf node [20]. At each leaf node, the predicted output value is determined based on the target output values of the training samples within that node. For regression tasks, the predicted value is typically the mean (or another summary statistic) of the target output values in the leaf node [20]. To make predictions on new, unseen data, the input feature vector is traversed down the decision tree, and the final predicted value is obtained from the corresponding leaf node.

Decision Tree Regressors are interpretable, easy to visualize, and can handle both numerical and categorical features. However, they can be sensitive to small changes in the data and may lead to overfitting, especially if the tree becomes too deep [19]. To address this, ensemble methods like Random Forest and Gradient Boosting are often used to combine multiple decision trees and improve predictive performance while reducing overfitting.

6.1.4 Random Forest Regressor

The Random Forest Regressor is an ensemble learning algorithm used for regression tasks. It builds multiple decision trees and combines their predictions to create a more accurate and robust regression model [21]. The mathematical formulation of the Random Forest Regressor involves the aggregation of individual decision tree predictions to make the final prediction.
Given a training dataset with input feature vectors $X = \{x_1, x_2, ..., x_n\}$ and corresponding target output values $y = \{y_1, y_2, ..., y_n\}$, where x_i is the input feature vector and y_i is the target output for the i-th sample. Random Forest creates an ensemble of decision trees. To build each tree in the forest, it uses a random subset of the training data (sampling with replacement) and a random subset of the features (sampling without replacement) [21]. These random selections introduce diversity among the trees.
Each decision tree in the Random Forest is grown using the recursive partitioning process of the Decision Tree Regressor. At each internal node of a tree, a decision is made based on a randomly selected feature and split point that maximizes the homogeneity of the target output values within each subset [21]. The predictions from each tree in the forest are combined to obtain the final prediction. For regression tasks, the most common method is to take the average (or another summary statistic) of the predicted values from all the trees.
Mathematically, the final prediction of the Random Forest Regressor can be represented as in [21] by:

$$F_{\text{final}}(x) = \frac{1}{M} \sum_{m=1}^{M} F_m(x)$$

Where $F_{\text{final}}(x)$ is the final predicted value for the input feature vector x, $F_m(x)$ is the prediction from the m-th decision tree in the forest for the input feature vector x, and M is the total number of decision trees in the Random Forest.
The averaging of predictions from multiple trees helps in reducing the variance and improving the generalization of the model. Random Forest Regressors are robust to outliers and noisy data, and they can handle both numerical and categorical features [21]. They are also less prone to overfitting compared to a single decision tree. However, Random Forests can be computationally expensive and may require tuning of hyperparameters, such as the number of trees in the forest, to achieve optimal performance [21]. Despite this, Random Forest Regressors are widely used and are known for their high accuracy and versatility in various regression tasks.

6.1.5 Gradient Boosting Regressor

Gradient Boosting Regressor is an ensemble learning algorithm used for regression tasks. It combines multiple weak learners sequentially to create a strong predic-

tive model. The mathematical formulation of Gradient Boosting Regressor involves optimization of a loss function with respect to the ensemble of weak learners [22]. Given a training dataset with input feature vectors $X = \{x_1, x_2, ..., x_n\}$ and corresponding target output values $y = \{y_1, y_2, ..., y_n\}$, where x_i is the input feature vector and y_i is the target output for the i-th sample. The initial prediction, $F_0(x)$, is usually set to the average of all target output values in the training dataset, i.e., $F_0(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$ [22]. In each iteration m, a new weak learner, $h_m(x)$, is trained to fit the negative gradient of the loss function, $L(y, F_{m-1}(x))$, with respect to the current prediction, $F_{m-1}(x)$. This is known as the residual error [22].
The objective is to find $h_m(x)$ that minimizes the following loss function [22]:

$$h_m(x) = \arg\min_h \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h(x_i))$$

Common loss functions used in Gradient Boosting Regressor are Mean Squared Error (MSE) and Mean Absolute Error (MAE). Once $h_m(x)$ is determined, the weak learner's output is added to the current prediction, and the model is updated [22]:

$$F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x)$$

Here, η is the learning rate, which is a hyperparameter that controls the step size of each update. It prevents overfitting and helps in better convergence. The above steps are repeated for a fixed number of iterations (trees) or until a stopping criterion is met. The final prediction of the Gradient Boosting Regressor is the sum of all the weak learners' predictions [22]:

$$F_{\text{final}}(x) = F_0(x) + \sum_{m=1}^{M} \eta \cdot h_m(x)$$

Gradient Boosting Regressor effectively combines the predictions of multiple weak learners, where each weak learner corrects the errors made by its predecessors. This iterative process leads to a more accurate and powerful predictive model, which is capable of capturing complex relationships in the data. However, the main challenge with Gradient Boosting Regressor lies in hyperparameter tuning, as an improper choice of parameters can lead to overfitting or slow convergence [22]. Nonetheless, Gradient Boosting Regressor is widely used and is considered one of the most powerful and effective regression algorithms.

6.1.6 K-Neighbors Regressor

The K-Neighbors Regressor is a supervised learning algorithm used for regression tasks. It works based on the principle of finding the k nearest neighbors to a given data point in the feature space and then using their target output values to predict the target output value for the new data point [20]. The mathematical formulation of the K-Neighbors Regressor involves calculating the average (or weighted average) of the target output values of the k nearest neighbors.
Given a training dataset with input feature vectors $X = \{x_1, x_2, ..., x_n\}$ and corresponding target output values $y = \{y_1, y_2, ..., y_n\}$, where x_i is the input feature vector and y_i is the target output for the i-th sample. For a new data point x_{new},

the K-Neighbors Regressor finds the k nearest neighbors to x_{new} from the training dataset [20]. The distance metric used (e.g., Euclidean distance) determines the notion of "closeness." The predicted target output value for x_{new}, denoted as y_{pred}, is calculated by taking the average (or weighted average) of the target output values of the k nearest neighbors [20].
Mathematically, the prediction for x_{new} using the K-Neighbors Regressor can be represented in [20] as:

$$y_{\text{pred}} = \frac{1}{k}\sum_{i=1}^{k} y_i$$

Alternatively, if weighted averaging is used, the prediction can be represented as:

$$y_{\text{pred}} = \frac{\sum_{i=1}^{k} w_i \cdot y_i}{\sum_{i=1}^{k} w_i}$$

where y_{pred} is the predicted target output value for the new data point x_{new}, y_i is the target output value of the i-th neighbor, k is the number of neighbors (a hyperparameter) used in the prediction, and w_i is the weight assigned to the i-th neighbor (used in weighted averaging, typically based on the distance or similarity) [20].
The K-Neighbors Regressor is a non-parametric algorithm and can be sensitive to the choice of k. A small k value may lead to noisy predictions, while a large k value may result in overly smooth predictions, especially in regions with varying densities of training data [20]. Properly choosing k and selecting an appropriate distance metric are important aspects of using the K-Neighbors Regressor effectively. Additionally, it is worth noting that the K-Neighbors Regressor can become computationally expensive as the size of the training dataset grows.

6.1.7 XGBoost Regressor

XGBoost (Extreme Gradient Boosting) Regressor is an ensemble learning algorithm used for regression tasks. It is an advanced version of Gradient Boosting that introduces regularization and parallel processing, making it highly efficient and effective [23]. The mathematical formulation of the XGBoost Regressor involves optimizing a loss function with respect to an ensemble of weak learners and employing regularization terms to control model complexity.
Given a training dataset with input feature vectors $X = \{x_1, x_2, ..., x_n\}$ and corresponding target output values $y = \{y_1, y_2, ..., y_n\}$, where x_i is the input feature vector and y_i is the target output for the i-th sample. XGBoost starts with an initial prediction, $F_0(x)$, typically set to the average of all target output values in the training dataset, i.e., $F_0(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$ [23].
XGBoost builds multiple weak learners sequentially. At each iteration m, it adds a new decision tree $h_m(x)$ to correct the errors made by the previous ensemble of decision trees [23]. The goal is to minimize a loss function, $L(y, F_{m-1}(x) + h_m(x))$, which measures the discrepancy between the actual target output y and the current prediction $F_{m-1}(x) + h_m(x)$ [23]. XGBoost uses gradient descent optimization to find the best weak learner $h_m(x)$ that reduces the loss function [23]. Additionally, it employs a regularization term to control the complexity of the model and prevent

overfitting. The regularization term consists of two parts: a L1 regularization term (Lasso regularization) and a L2 regularization term (Ridge regularization) [23]. The objective function to be minimized at each iteration m is given by [23] as:

$$\text{Objective}_m = \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h_m(x_i)) + \Omega(h_m)$$

Here, $\Omega(h_m)$ represents the regularization term for the m-th weak learner. The weak learner $h_m(x)$ is added to the ensemble with a learning rate η, which controls the contribution of each tree to the final prediction. The update rule is as follows [23]:

$$F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x)$$

The above steps are repeated for a fixed number of iterations (trees) or until a stopping criterion is met. The final prediction of the XGBoost Regressor is the sum of all the weak learners' predictions [23]:

$$F_{\text{final}}(x) = F_0(x) + \sum_{m=1}^{M} \eta \cdot h_m(x)$$

Where:

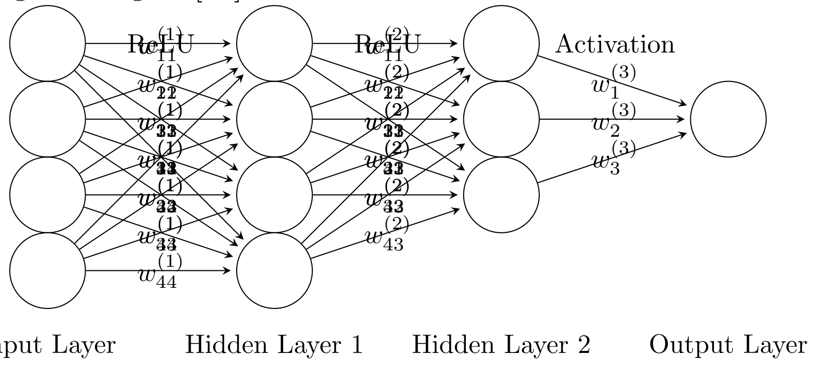
- $F_{\text{final}}(x)$ is the final predicted value for the input feature vector x.
- $F_m(x)$ is the prediction from the m-th weak learner (decision tree) for the input feature vector x.
- M is the total number of weak learners (trees) in the XGBoost Regressor.
- η is the learning rate, a hyperparameter that controls the contribution of each tree to the final prediction.
- $\Omega(h_m)$ is the regularization term for the m-th weak learner.

XGBoost Regressor is known for its high accuracy, efficiency, and robustness in various regression tasks. Its ability to handle large-scale datasets and incorporate regularization techniques makes it a popular choice for many real-world applications.

6.1.8 Deep Neural Multilayer Perceptron (MLP) regressor

The Deep Neural Multilayer Perceptron (MLP) Regressor is a supervised learning algorithm used for regression tasks. It is a type of artificial neural network that consists of multiple layers of interconnected neurons. The mathematical formulation of the Deep Neural MLP Regressor involves the forward pass, where the input data is propagated through the network to make predictions, and the backpropagation algorithm, which is used to update the model parameters during the training process [24].
Given a training dataset with input feature vectors $X = \{x_1, x_2, ..., x_n\}$ and corresponding target output values $y = \{y_1, y_2, ..., y_n\}$, where x_i is the input feature vector and y_i is the target output for the i-th sample. The Deep Neural MLP Regressor consists of an input layer, one or more hidden layers, and an output layer.

Each layer contains a set of neurons (nodes), and the neurons are connected through weighted edges. [24]



During the forward pass, the input feature vector x_i is fed into the input layer, and its values are propagated through the network layer by layer. At each neuron, the input is linearly combined with the neuron's weights and bias, and then passed through an activation function to introduce non-linearity. The forward pass of the l-th hidden layer can be represented mathematically in [24] as:

$$z^{(l)} = W^{(l)} \cdot a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = \text{activation}(z^{(l)})$$

Where:

- $z^{(l)}$ is the linear combination of inputs in the l-th layer.
- $W^{(l)}$ is the weight matrix for the l-th layer.
- $a^{(l-1)}$ is the output (activation) of the previous layer.
- $b^{(l)}$ is the bias vector for the l-th layer.
- $a^{(l)}$ is the output (activation) of the l-th layer after passing through the activation function.

The forward pass continues until the output layer is reached, and the final predicted value y_{pred} for the input feature vector x_i is obtained from the output layer. During the training process, the model's parameters (weights and biases) are initialized randomly. The backpropagation algorithm is used to update these parameters iteratively to minimize a loss function (e.g., Mean Squared Error or Mean Absolute Error) that measures the discrepancy between the predicted values and the actual target output values [24]. The backpropagation algorithm involves computing the gradients of the loss function with respect to the model parameters and using these gradients to update the parameters using optimization techniques like stochastic gradient descent (SGD) or its variants [24]. The training process continues for a fixed number of iterations (epochs), or until the model converges to a satisfactory level.
The Deep Neural MLP Regressor is capable of learning complex relationships in the data and can handle both numerical and categorical features. However, it requires careful tuning of hyperparameters, such as the number of hidden layers, the number of neurons in each layer, the activation functions, and the learning rate, to

achieve optimal performance and avoid overfitting [24]. Deep Neural MLP Regressors are widely used in various applications, especially when dealing with large and complex datasets that require high representational power and non-linear mapping capabilities.

6.2 Performance Evaluation

After training and testing our flow data, it is necessary to evaluate the performance of the models on the dataset. Because the target variable is continuous, metrics for continuous data points will be used to evaluate the performance of the regressor models. These metrics include: Mean Squared Error (MSE), Mean Absolute Error (MAE), R-squared (R^2), Explained Variance Score (EVS), Median Absolute Error (MedAE), and Max Error metrics.

6.2.1 Mean Squared Error (MSE):

Measures the average squared difference between the predicted values and the actual target values in the dataset. Mathematically, the Mean Squared Error is calculated as:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - F(x_i))^2$$

where y_i represents the true target output value, and $F(x_i)$ represents the predicted output value for the i-th sample [20]. The squared difference $(y_i - F(x_i))^2$ is computed for each sample, and then the average of these squared differences is taken by summing them up and dividing by the total number of samples n. The MSE penalizes larger prediction errors more heavily, as it squares the differences. This makes it sensitive to outliers, meaning that large errors have a significant impact on the overall score. [20]

6.2.2 Mean Absolute Error (MAE):

Like MSE, also measures the performance of a regression model by quantifying the difference between the predicted values and the actual target values in the dataset. However, instead of squaring the differences, MAE takes the absolute value of the differences. Mathematically, the Mean Absolute Error is calculated as follows: Given a dataset with n samples, where y_i is the actual target output value and $F(x_i)$ is the predicted output value for the i-th sample:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - F(x_i)|$$

where y_i represents the true target output value, and $F(x_i)$ represents the predicted output value for the i-th sample [20]. The absolute difference $|y_i - F(x_i)|$ is computed for each sample, and then the average of these absolute differences is taken by summing them up and dividing by the total number of samples n. The MAE is less sensitive to outliers compared to the MSE because it does not square the differences.

As a result, large prediction errors do not have as significant an impact on the overall score as they do in the MSE. [20]

6.2.3 R-squared ($\mathbf{R^2}$):

Also known as the coefficient of determination, is a statistical measure used to evaluate the performance of a regression model. It provides information about how well the model fits the data points and explains the variability of the target variable around its mean. Mathematically, given a dataset with n samples, where y_i is the actual target output value, and $\hat{y}_i$ is the predicted output value by the model for the i-th sample, the mean of the actual target values is denoted as $\bar{y}$:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Then, the total sum of squares (TSS) is calculated, representing the total variability of the target variable around its mean [18]:

$$\text{TSS} = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

Next, the residual sum of squares (RSS) is computed, representing the variability that is not explained by the model [18]:

$$\text{RSS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Finally, the R-squared value is obtained as the ratio of explained variability to total variability [18]:

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$$

The R-squared value ranges from 0 to 1. A value closer to 1 indicates a better fit of the model to the data, meaning the model can explain a larger proportion of the variability in the target variable. Conversely, a value closer to 0 suggests that the model is not a good fit to the data, and it fails to explain much of the variability [18].

6.2.4 Explained Variance Score (EVS):

It is a performance metric used to evaluate the goodness of fit of a regression model, particularly in cases where the target variable has significant variance [25]. EVS quantifies the proportion of variance in the target variable that is explained by the regression model [25]. Given a dataset with n samples, where y_i is the actual target output value and $\hat{y}_i$ is the predicted output value by the model for the i-th sample, the mean of the actual target values is denoted as $\bar{y}$:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

The total sum of squares (TSS) represents the total variance of the target variable around its mean:

$$\text{TSS} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

The residual sum of squares (RSS) represents the unexplained variance of the target variable by the model:

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Finally, the Explained Variance Score is obtained as the ratio of explained variance to total variance [25]:

$$\text{EVS} = 1 - \frac{\text{RSS}}{\text{TSS}}$$

EVS is just like R^2. However, EVS ranges from negative infinity to 1. A value of 1 indicates that the model perfectly explains all the variance in the target variable, while values close to 0 or negative values suggest that the model provides little or no improvement over a constant baseline model. [25] The Explained Variance Score can be useful in situations where the variance of the target variable is significant, as it directly measures how much of that variance is captured by the model.

6.2.5 Median Absolute Error (MedAE):

is a statistical measure used to evaluate the performance of regression models, particularly in cases where the data may contain outliers. It is a robust metric that is less sensitive to extreme values compared to mean-based metrics such as Mean Squared Error (MSE) or Mean Absolute Error (MAE). MedAE is calculated as the median of the absolute differences between the predicted values and the actual target values. Given a dataset with n samples, where y_i is the actual target output value, and $\hat{y}_i$ is the predicted output value by the model for the i-th sample:

$$\text{MedAE} = \text{median}(|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, ..., |y_n - \hat{y}_n|)$$

The Median Absolute Error measures the typical magnitude of the prediction errors. It provides a more robust estimate of the error compared to the mean-based metrics, as it is less influenced by the presence of outliers or extreme errors. When comparing regression models, a lower value of MedAE indicates a better fit, meaning the model's predictions are, on average, closer to the true target values.
The choice of using MedAE as an evaluation metric depends on the specific characteristics of the data and the requirements of the problem. If the data contains outliers or extreme values, or if robustness against outliers is a priority, MedAE can be a more appropriate choice than MSE or MAE. However, MedAE may not be as widely used or as easy to interpret as MSE and MAE in some contexts.

6.2.6 Max Error:

is a statistical metric used to evaluate the performance of a regression model. It measures the maximum absolute difference between the predicted values and the actual target values in the dataset. Max Error provides an indication of the worst-case prediction error made by the model. Mathematically, given a dataset with n samples, where y_i is the actual target output value and $F(x_i)$ is the predicted output value by the model for the i-th sample, the absolute difference $|y_i - F(x_i)|$ is computed for each sample. The Max Error is then obtained as the maximum of these absolute differences:

$$\text{Max Error} = \max(|y_1 - F(x_1)|, |y_2 - F(x_2)|, ..., |y_n - F(x_n)|)$$

In other words, the Max Error represents the largest absolute difference between any predicted value and the corresponding actual target value in the dataset. It identifies the most significant prediction error made by the model.

6.2.7 Metric Results

The metric results obtained from the dataset is shown in the table below (table 6.1).

Regression Models	**MSE**	**MAE**	**R^2**	**EVS**	**MedAE**	**Max Error**
Support Vector	0.00227	0.03715	0.94562	0.94642	0.03092	0.34232
Linear Regressor	0.00206	0.03086	0.95060	0.95061	0.02127	0.48601
Decision Tree	0.00282	0.03528	0.93249	0.93250	0.02242	0.47968
Random Forest	0.00144	0.02514	0.96566	0.96568	0.01641	0.37741
Gradient Boosting	0.00174	0.02852	0.95834	0.95835	0.01931	0.45196
K-Neighbors	0.00227	0.03097	0.94579	0.94611	0.01887	0.40112
XGBoost	0.00140	0.02540	0.96650	0.96651	0.01717	0.34567
MLP regressor	0.00142	0.02630	0.96594	0.96603	0.01859	0.33641

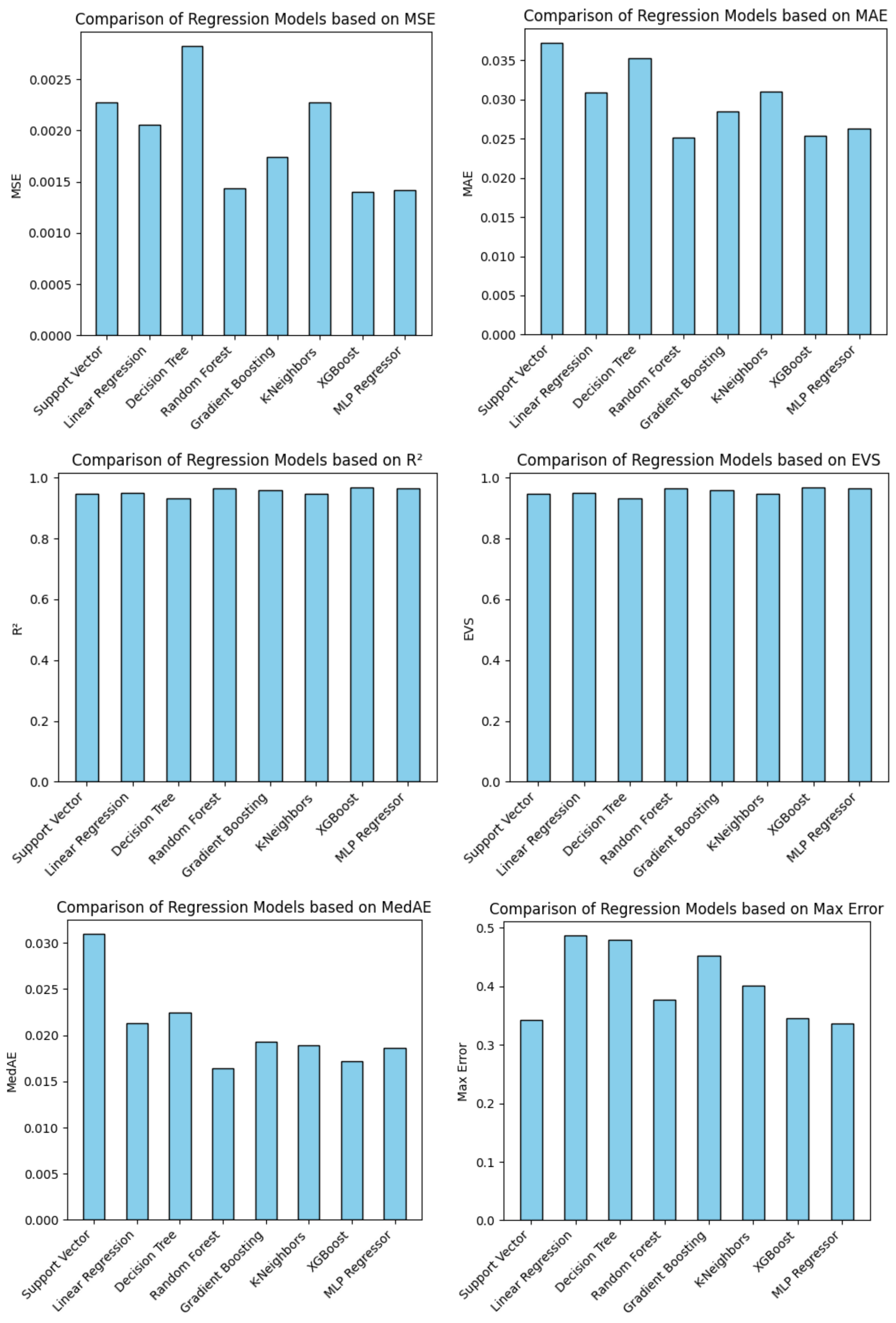
Table 6.1: Performance of the Regression Models

Plotting the performance of the models gives the graphs shown below (6.2).
Analyzing the results in table 6.1 and figure 6.2 shows that XGBoost regressor had the highest performance since it had the lowest MSE, MAE, MedAE, and Max Error and the highest R^2 and EVS. Even though Random Forest regressor had an equivalent performance to XGBoost Regressor, Random Forest regressor had slightly lower Max Error. Nonetheless, both had comparable similar performance and as such is the highest performing models in all the models. Decision Tree, SVR, and Linear Regressor were the lowest performing models with Decision having the highest MSE, SVR having the highest MAE and MedAE.
To create an overall ranking, we might consider assigning a weighted score to each model based on their performance across all metrics. We can then sum up these weighted scores to obtain a comprehensive ranking. The weights will reflect the relative importance of each metric.
For example, if you consider all six metrics (MSE, MAE, R^2, EVS, MedAE, Max Error), you could assign equal weights for simplicity:

$$\text{Overall Score} = w_{\text{MSE}} \times \text{Rank}_{\text{MSE}} + w_{\text{MAE}} \times \text{Rank}_{\text{MAE}} + w_{\text{R}^2} \times \text{Rank}_{\text{R}^2} + w_{\text{EVS}} \times \text{Rank}_{\text{EVS}}$$
$$+ w_{\text{MedAE}} \times \text{Rank}_{\text{MedAE}} + w_{\text{Max Error}} \times \text{Rank}_{\text{Max Error}}$$

However, assuming equal weights for all the metrics (i.e., $w_{\text{MSE}} = w_{\text{MAE}} = w_{\text{R}^2} = w_{\text{EVS}} = w_{\text{MedAE}} = w_{\text{Max Error}} = 1$), the overall ranking can be calculated by summing

Figure 6.2: Models Performance



up the individual ranks for each model.

Support Vector: $7 + 8 + 7 + 6 + 8 + 2 = 38$
Linear Regressor: $5 + 5 + 5 + 5 + 6 + 8 = 32$
Decision Tree: $8 + 7 + 8 + 8 + 7 + 7 = 42$

Random Forest: $3 + 1 + 3 + 3 + 1 + 4 = 15$
Gradient Boosting: $4 + 4 + 4 + 4 + 5 + 6 = 27$
K-Neighbors: $6 + 6 + 5 + 7 + 4 + 5 = 33$
XGBoost: $1 + 2 + 1 + 1 + 2 + 3 = 10$
MLP Regressor: $2 + 3 + 2 + 2 + 3 + 1 = 13$

The models are ranked based on their total sum of ranks. Lower total sum indicates a better overall ranking and a higher total sum indicate worst overall ranking. Therefore, based on these assumptions, the overall ranking from best to worst with respect to this network flow data is:

1. **XGBoost**
2. **MLP Regressor**
3. **Random Forest**
4. **Gradient Boosting**
5. **Linear Regressor**
6. **K-Neighbors**
7. **Support Vector**
8. **Decision Tree**

The actual weights might differ based on the importance of each metric to the specific use case. One can therefore adjust the weights accordingly in other scenarios if certain metrics are more critical than others for their application.
Overall, all the algorithms performed very well which shows that machine learning regression models are very ideal for network flow volume predictions.

Chapter 7

Conclusion

This works gives a comprehensive analysis of the spatio-temporal traffic volume dataset. All the fundamental questions that was posed earlier on in the abstract has been successfully answered. Statistical inferences and descriptive analysis have been made on the network's data flow; the Routine Matrix of the Network was successfully obtained from the Adjacency Matrix; Regularization Techniques were employed to solve the singularity or ill posed nature of the network in the Traffic Matrix Estimation; and finally, Machine Learning regression models were applied to make predictions on the Network's flow volume and it was realised that XGBoost, MLP Regressor, and Random Forest Regressor performed best.
Knowledge or concepts in this paper can be practically applied to analyze other network flow data. Future works include applying Methods Based on Poisson Models and Methods Based on Entropy Minimization to estimating the network's traffic flow volume and then comparing and contrasting the results we will obtain to the results obtained here using Methods Based on Least-Squares and Gaussian Models. Nonetheless, we acknowledge the original source of the dataset [6], Dr. Liang Zhao (Emory University), and we do cherish the time and energy used to obtain this dataset.

Bibliography

[1] E. D. Kolaczyk, *Statistical Analysis of Network Data: Methods and Models* (Springer Series in Statistics), 1st ed. Springer New York, NY, 2009, pp. XII, 386, Published: 19 March 2009, ISBN: 978-0-387-88146-1 (eBook). DOI: 10.1007/978-0-387-88146-1.

[2] E. D. Kolaczyk and G. Csárdi, *Statistical Analysis of Network Data with R* (Use R!), 1st ed. Springer New York, NY, 2014, pp. XIII, 207, DOI: https://doi.org/10.1007/978-1-4939-0983-4, ISBN: 978-1-4939-0983-4.

[3] R. Wilson, *Introduction to Graph Theory.* Longman, 2010, ISBN: 9780273728894.

[4] D. Easley and J. Kleinberg, *Networks, Crowds, and Markets: Reasoning about a Highly Connected World.* Cambridge University Press, 2010, ISBN: 9781139490306.

[5] R. G. Thompson and D. M. Levinson, "Transportation network analysis," in *The Oxford Handbook of Urban Planning*, R. Weber and R. Crane, Eds., Oxford University Press, 2012.

[6] L. Zhao, O. Gkountouna, and D. Pfoser, "Spatial auto-regressive dependency interpretable learning based on spatial topological constraints," *ACM Trans. Spatial Algorithms Syst.*, vol. 5, no. 3, 19:1–19:28, Aug. 2019, DOI: https://doi.org/10.1145/3339823. [Online]. Available: https://doi.org/10.1145/3339823.

[7] A. Barabási and M. PÃ3sfai, *Network Science.* Cambridge University Press, 2016, ISBN: 9781107076266.

[8] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, 1st. USA: Prentice Hall PTR, 1998, ISBN: 0133016153.

[9] T. Nishizeki and M. Rahman, *Planar Graph Drawing* (Lecture notes series on computing). World Scientific, 2004, ISBN: 9789812560339.

[10] M. Newman, *Networks: An Introduction.* OUP Oxford, 2010, ISBN: 9780199206650.

[11] W. Navidi, *Statistics for Engineers and Scientists.* McGraw-Hill, 2011, ISBN: 9780071327565.

[12] D. Montgomery, E. Peck, and G. Vining, *Introduction to Linear Regression Analysis* (Wiley Series in Probability and Statistics). Wiley, 2013, ISBN: 9781118627365.

[13] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Springer Series in Statistics). Springer New York, 2013, ISBN: 9780387216065.

[14] L. Trefethen and D. Bau, *Numerical Linear Algebra: Twenty-Fifth Anniversary Edition* (Other Titles in Applied Mathematics). SIAM, Society for Industrial and Applied Mathematics, 2022, ISBN: 9781611977165.

[15] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*, 1st. USA: Cambridge University Press, 2019, ISBN: 1108422098.

[16] G. Golub and C. Van Loan, *Matrix Computations* (Johns Hopkins Studies in the Mathematical Sciences). Johns Hopkins University Press, 2013, ISBN: 9781421407944.

[17] A. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, pp. 199–222, Aug. 2004. DOI: 10.1023/B%3ASTCO.0000035301.49549.88.

[18] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R* (Springer Texts in Statistics). Springer New York, 2013, ISBN: 9781461471387.

[19] A. Müller and S. Guido, *Introduction to Machine Learning with Python: A Guide for Data Scientists.* O'Reilly Media, 2016, ISBN: 9781449369903.

[20] C. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics). Springer New York, 2016, ISBN: 9781493938438.

[21] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, Oct. 2001. DOI: 10.1023/A:1010950718922.

[22] J. H. Friedman, "Greedy function approximation: A gradient boosting machine.," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. DOI: 10.1214/aos/1013203451. [Online]. Available: https://doi.org/10.1214/aos/1013203451.

[23] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794, ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. [Online]. Available: https://doi.org/10.1145/2939672.2939785.

[24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (Adaptive Computation and Machine Learning series). MIT Press, 2016, ISBN: 9780262035613.

[25] S. Raschka and V. Mirjalili, *Python Machine Learning.* Packt Publishing, 2017, ISBN: 9781787126022.