Predicting Traffic Flow in Google Maps: A Machine Learning Approach Using Graph Neural Networks on Weighted Directed Graphs

Mohammad Akmal Ramadan - 13522161[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*13522161@std.stei.itb.ac.id*

***Abstract*— The rapid growth of urban populations worldwide has led to increased traffic congestion, making efficient traffic prediction crucial. This paper presents a novel approach to predicting traffic flow in Google Maps using machine learning techniques, specifically Graph Neural Networks (GNNs), applied to weighted directed graphs. The primary objective of this research is to emphasize the significance role of weighted-directed graphs in the application of Graph Neural Networks (GNNs), aiming to achieve unprecedented accuracy in predicting traffic flow. This method involves representing the transportation network as a weighted directed graph, where nodes are locations, edges are roads, and weights are travel times. This paper employ GNNs, a powerful tool for learning on graph-structured data, to model and predict traffic conditions. The GNN is trained in historical traffic data and updates its predictions in real-time based on current traffic conditions. The results of this paper show a significant improvement in traffic prediction accuracy compared to traditional methods. Furthermore, the use of GNNs allowed for the capture of complex traffic patterns and dependencies that were not accounted for in previous models. In conclusion, this research demonstrates the potential of using machine learning, specifically GNNs, for traffic prediction in Google Maps. It opens up new possibilities for the application of these techniques in other areas of transportation planning and management. Future work will focus on refining the model and exploring its applicability to other types of transportation networks.**

***Keywords*—Graph Neural Networks, Prediction, Traffic, Weight-Directed Graph.**

I. INTRODUCTION

The exponential growth of urban populations across the globe has given rise to a pressing challenge, traffic congestion. As cities become more crowded, the need for efficient traffic prediction becomes increasingly critical. Accurate traffic prediction can enhance route planning, reduce travel time, and contribute to the overall efficiency of urban transportation systems.

This paper introduces a novel approach to tackle this issue by leveraging the power of machine learning techniques, specifically Graph Neural Networks (GNNs), applied to weighted directed graphs. The primary objective of this paper is not just to predict traffic flow, but to highlight the significant role that weighted-directed graphs play in improving the accuracy of these predictions.

This paper methodology involves a unique representation of the transportation network as a weighted directed graph, where each node corresponds to a location, each edge represents a road, and the weights signify travel times. This paper employ GNNs, known for their efficacy in learning on graph-structured data, to model and predict traffic conditions.

The GNN model is trained on historical traffic data, enabling it to update its predictions in real-time based on current traffic conditions. Our research results indicate a marked improvement in traffic prediction accuracy compared to traditional methods, demonstrating the potential of machine learning, and GNNs in particular, for traffic prediction in applications like Google Maps.

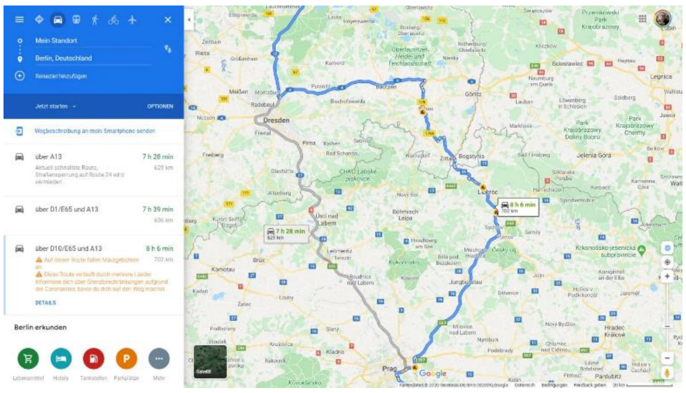


Fig. 1. Google Maps Traffic Prediction.
Source: *https://sakina-mooney.blogspot.com/2021/08/google-maps-route-google-maps-shortest.html*

Furthermore, this paper reveals how the use of GNNs allows for the capture of complex traffic patterns and dependencies, which were previously overlooked. This paper opens up new avenues for the application of these techniques in broader areas of transportation planning and management.

II. THEORETICAL FOUNDATIONS

A. Graph

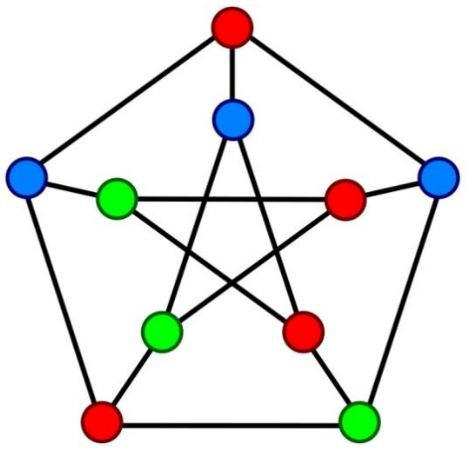


Fig. 2. Graph Illustrations
Source: Graf (itb.ac.id)

Graph is a discrete structure that represents a pairwise relationship between objects [1], [2]. A graph consists of a set of objects, called vertices or nodes, and a set of pairs of vertices, called edges or lines [1]–[4]. A graph can be denoted by an ordered pair G = (V, E), where V is the set of vertices and E is the set of edges [4]. Graph theory is the branch of discrete mathematics that studies the properties and applications of graphs.

A graph has two main components: First, a vertices or as known as nodes, these are points or entities within the graph. Each vertex represents an object or an entity, and it can be connected to other vertices via edges. The second one is edges; edges are the connections or links between pairs of vertices. They establish relationships or interactions between the vertices they connect. An edge can be directed or undirected, and it may or may not have a specific weight or value associated with it. These two components combined create a graph as shown in Fig. 2.

Graph (G) is defined as a pair.

$$G = (V, E)$$

, where

$$V$$

is a set whose elements are called vertices, and

$$E$$

is a set of paired vertices, whose elements are called edges.
Graphs also can be classified based on various characteristics, directed and undirected graph also weighted and unweighted graph.

Directed Graph

A directed graph, also known as a digraph, is a graph in which the edges have a direction. This is usually indicated with an arrow on the edge. More formally, if v and w are vertices, an edge is an unordered pair {v, w}, while a directed edge, called an arc, is an ordered pair (v, w) or (w, v). The arc (v, w) is drawn as an arrow from v to w [5].

In a directed graph, the edge (i, j) is interpreted as going from vertex i into vertex j, and it is graphically represented by drawing an arrow from vertex i to vertex j [6].

The out-degree of a vertex can be described as the number of edges pointing from it, and the in-degree of a vertex can be described as the number of edges pointing to it.

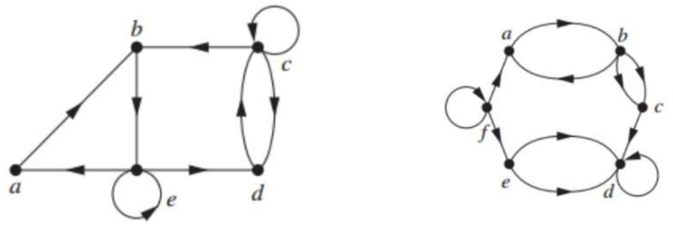


Fig. 3. Directed Graph
Source: Graf (itb.ac.id)

A directed path in a directed graph can be described as a sequence of vertices and a directed edge, where the edge is pointing from each vertex in the sequence to its successor in the sequence. The directed path will not contain repeated edges. If there are no repeated vertices, then the directed path will be simple. If the first and last vertices in the directed path are the same, and contain at least one edge, then the directed path will be known as the directed cycle. If there are no directed cycles, the directed graph will be known as the directed acyclic graph (DAG) [7].

Undirected Graph

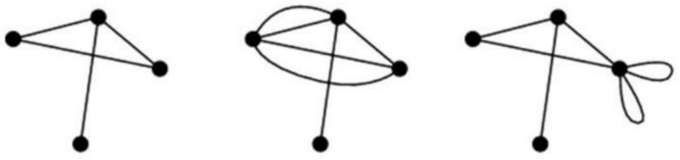


Fig. 4. Undirected Graph
Source: Graf (itb.ac.id)

As you can tell by the name and previous explanation, Undirected graph is a graph where all the edges are bidirectional. This means that there is no direction associated with the edges.

Weighted Graph

Weighted graph is a graph in which each edge is assigned a number, usually positive. This number is known as the weight of the edge. The weight can represent various concepts depending on the problem at hand. In this topic, the graph is a transportation network, where nodes are locations, edges are roads, and weights are travel.

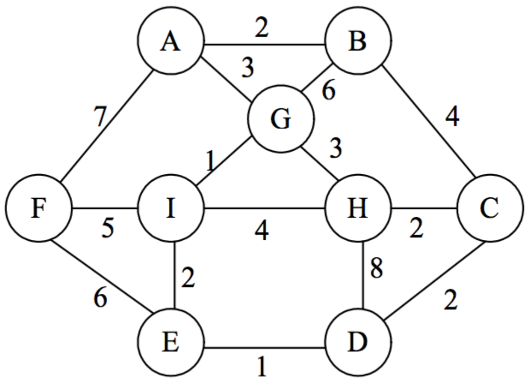


Fig. 5. Weighted Graph

Source: *https://www.chegg.com/homework-help/questions-and-answers/1-give-example-weighted-graph-minimum-spanning-tree-unique-indicate-minimum-spanning-tree--q9401981*

Weighted-Directed Graph

Weighted-Directed Graph is a graph resulted from combining weighted graph and directed graph. In a weighted-directed graph, the edge (i, j) is interpreted as going from vertex i into vertex j, and it is graphically represented by drawing an arrow from vertex i to vertex j.

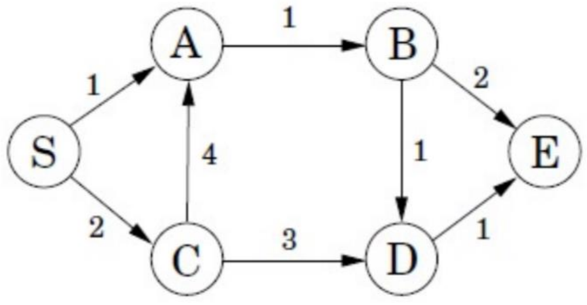


Fig. 5. Weighted-Directed Graph
Source: https://www.coursehigh.com/downloads/solved-given-weighted-directed-graph-g-v-vertices-e-edges-non-negative-edge-weights-write-pseudoc-q36047540/

B. Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) are a specific type of machine learning models that are designed to handle data in the form of graphs. These neural networks have the ability to learn and make predictions on graph-structured data, which makes them well-suited for tasks involving relationships and dependencies between entities in complex systems [8].

GNNs work by gathering information from neighboring nodes within a graph, effectively capturing the local structure and connectivity patterns of the data [9]. They update node representations by combining information from adjacent nodes in an iterative manner, typically through a message-passing scheme. This process enables GNNs to learn representations of nodes that encompass both their local neighborhood information and the overall structure of the entire graph.

The connection between GNNs and weighted-directed graphs lies in how GNNs can effectively process and learn from such graph structures. Weighted-directed graphs provide additional information beyond simple connectivity by assigning weights and directions to edges, indicating the strength, distance, or other characteristics of relationships between nodes. GNNs can leverage this additional information encoded in the graph structure, utilizing the edge weights and directionalities during their learning process.

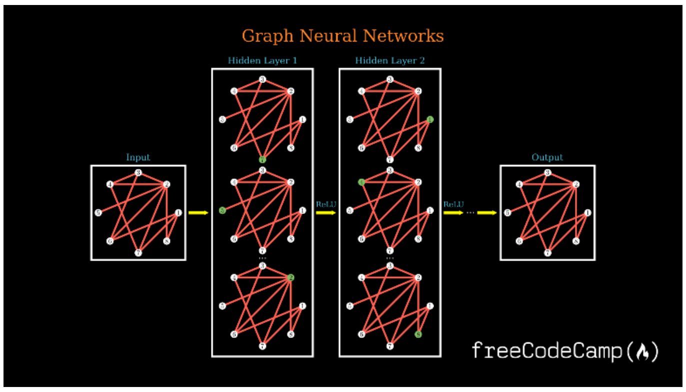


Fig. 6. Graph Neural Networks
Source: https://www.freecodecamp.org/news/graph-neural-networks-explained-with-examples/

In the context of traffic prediction or other applications where transportation networks are represented as weighted-directed graphs (with nodes representing locations, edges representing roads, and edge weights denoting travel times or distances), GNNs can be used to model and predict traffic conditions. By incorporating information about the directionality and weights of edges, GNNs have the potential to accurately capture intricate traffic patterns, dependencies, and variations in travel times between locations better than traditional methods.

The integration of GNNs with weighted-directed graphs allows these neural networks to exploit the richer information embedded in the graph structure.

C. Reinforcement Learning (RL)

RL is a subset of machine learning where an agent learns to behave in an environment, by performing certain actions and observing the rewards/results of those actions [10]. It's different from supervised learning, as it doesn't require a dataset with correct answers. Instead, the agent learns from the consequences of its actions.

Imagine an agent in a maze, where each state represents a position in the maze. The agent chooses actions (like moving north, south, east, or west), based on its current state, and receives rewards (like finding a piece of cheese or getting an electric shock). The action leads the agent to a new state and a new reward. The agent's aim is to find the best strategy or policy that maximizes the total rewards it receives.

There are different types of RL agents, which can be categorized based on their components: policy (the strategy that the agent uses to determine the next action), value function (a prediction of future rewards), and model (the agent's representation of the environment).

One common approach is to use the value function to choose actions. A popular algorithm in this category is Temporal Difference (TD) Q-learning. In this method, the agent maintains a table of action values (Q-values), which are estimates of the total reward that the agent will receive if it performs a particular action in a particular state. The agent updates these Q-values based on the rewards it receives and uses these updated Q-values to choose its actions.

$$Q(S,A) = Q(S,A) + \alpha(R + \gamma \, max_a \, Q(S',a) - Q(S,A) \quad (1)$$

In TD Q-learning, the Q-value of a state-action pair is updated using (1): learning rate (α) multiply with the accummulation of Reward(R) added by the difference between the maximum action value function of next state (maxa Q(S',a)) and the current state given action(Q(S,A)). The difference is multiplied first by a discount factor (γ). This process is repeated for many episodes (sequences of states and actions), and over time, the Q-values converge to the optimal values [11].

So, in essence, RL is about learning the best strategy or policy to maximize the total rewards in an environment, based on the agent's experiences.

D. Natural Language Processing (NLP)

Natural Language Processing (NLP) is a specialized field within artificial intelligence that focuses on the interaction between computers and human language. Its primary objective is to equip computers with the ability to comprehend, interpret, and handle extensive volumes of natural language data. NLP strives to enable systems to comprehend the intricacies embedded within textual content, encompassing nuanced aspects of vocabulary usage.

NLP technology empowers systems to accurately extract valuable information and insights from documents, facilitating their organization and categorization. Its significance is reflected across various industries, fueling applications like sentiment analysis, chatbots, virtual assistants, text classification, machine translation, text summarization, market intelligence, and more.

Within content-based recommendation systems, NLP assumes a pivotal role in computing document similarity scores. Utilizing techniques like the vector space model [11],documents are represented as vectors denoted by the document vector formula:

$$V_D[w_{1,d}, w_{2,d}, \dots, w_{n,d}]^T$$

Each vector V_D comprises word weights $[w_{1,d}, w_{2,d}, \dots, w_{n,d}]^T$, signifying the significance of words within each document. The representation generates sparse vectors due to the extensive vocabulary, resulting in numerous zero scores.

Various methods define word weight scores, including boolean scores indicating word presence, term frequency (TF) scores portraying word frequency, and TFxIDF scores [11] signifying word significance across the document collection.

The similarity score between two documents is calculated as a distance score between their respective document vectors. For instance, the widely used cosine similarity score measures the cosine of the angle between two vectors, represented by the formula:

$$\cos(d_j, d_k) = \frac{d_j . d_k}{\|d_j\| \|d_k\|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2}\sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

This computation captures the similarity in word usage between documents, enabling recommendation systems to suggest related documents based on their content similarity.

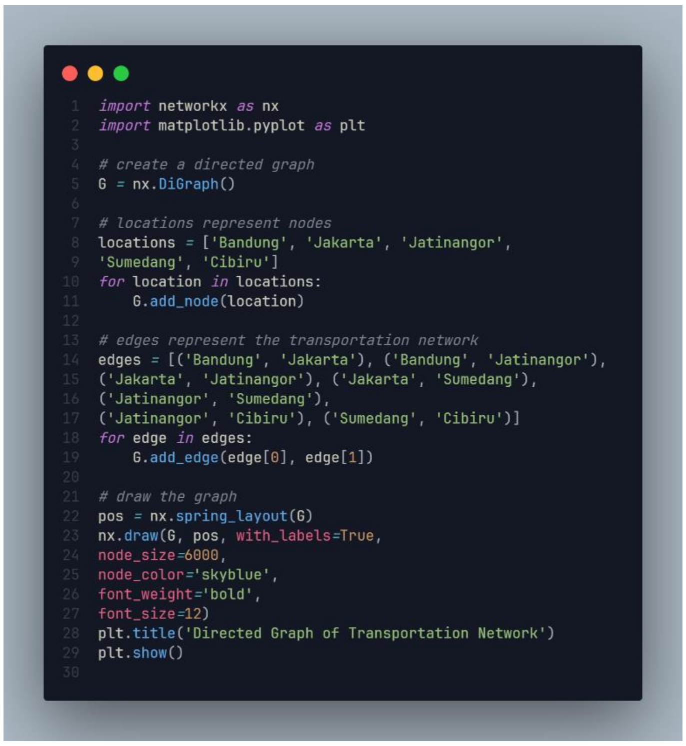
III. Analysis

The first step to applying Graph Neural Networks (GNNs) is to build the weight-directed graph. There are three main components in this weight-directed graph, which will be explained below.

1. Graph Initialization

the first thing is the nodes, which represents a location; the initial and destination location that the user wants to travel to, the second one is the edges, this edges will show all the route possibilities from one location to another location, the last component of this weight-directed graph is the weight, weight consist of all the aspects that affect traffic, such as rush hour, road quality, speed limits, accidents, and closure.

To make this directed graph, we can use a simple python code with the help of networkx and matplotlib library:



```python
import networkx as nx
import matplotlib.pyplot as plt

# create a directed graph
G = nx.DiGraph()

# locations represent nodes
locations = ['Bandung', 'Jakarta', 'Jatinangor',
'Sumedang', 'Cibiru']
for location in locations:
    G.add_node(location)

# edges represent the transportation network
edges = [('Bandung', 'Jakarta'), ('Bandung', 'Jatinangor'),
('Jakarta', 'Jatinangor'), ('Jakarta', 'Sumedang'),
('Jatinangor', 'Sumedang'),
('Jatinangor', 'Cibiru'), ('Sumedang', 'Cibiru')]
for edge in edges:
    G.add_edge(edge[0], edge[1])

# draw the graph
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True,
node_size=6000,
node_color='skyblue',
font_weight='bold',
font_size=12)
plt.title('Directed Graph of Transportation Network')
plt.show()
```

Fig. 7. Directed Graph in Python Code

From Fig. 7, the result of the directed graph would look like this:

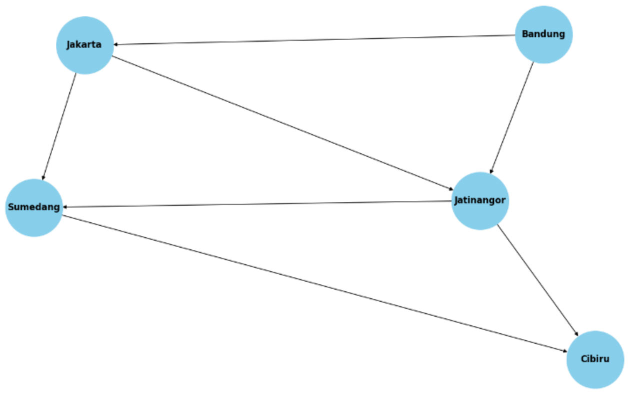


Fig. 8. Visualization from Fig. 7 Python Code

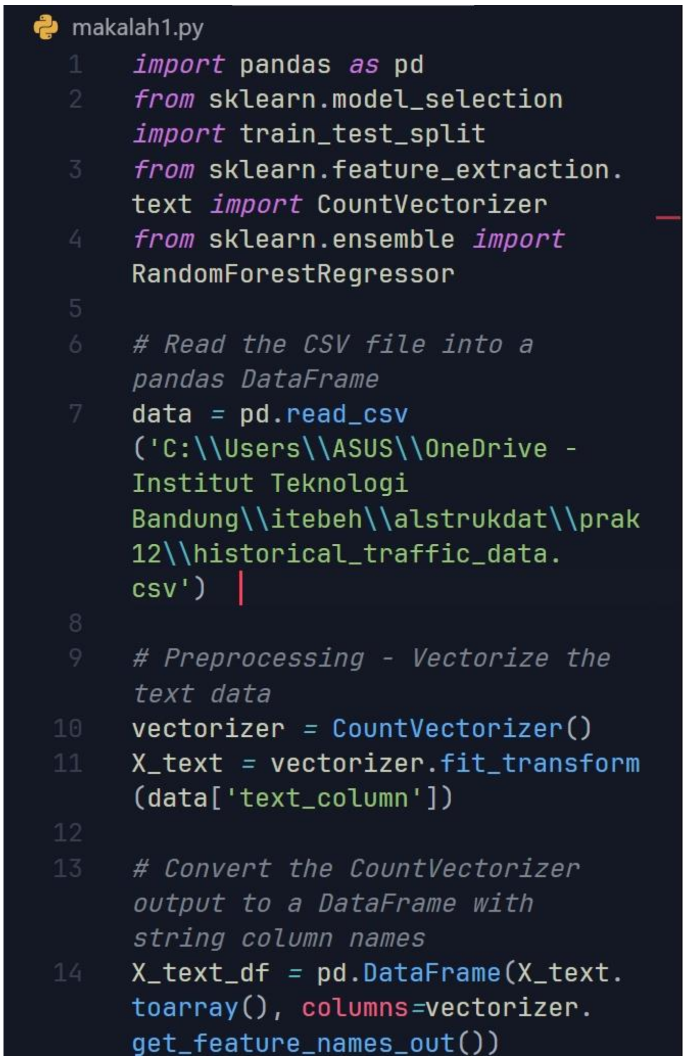
Note that there is a one-way road which allows users to travel from point A to point B, but cannot travel to point A from point B, hence the edges are not always bidirectional. For example, in Fig 8, users can travel from Jatinangor to Cibiru, but users will not be able to travel from Cibiru to Jatinangor.

Now, we can move on to create the weight-directed graph, to make this, we will need to take account of the variables that are contained in the weight. These variables then will be processed into machine learning algorithms called Natural Language Processing or shortened as NLP, to count the travel time of each node as the base for building the weight-directed graph.

1.1 Weight Initialization with NLP Algorithm

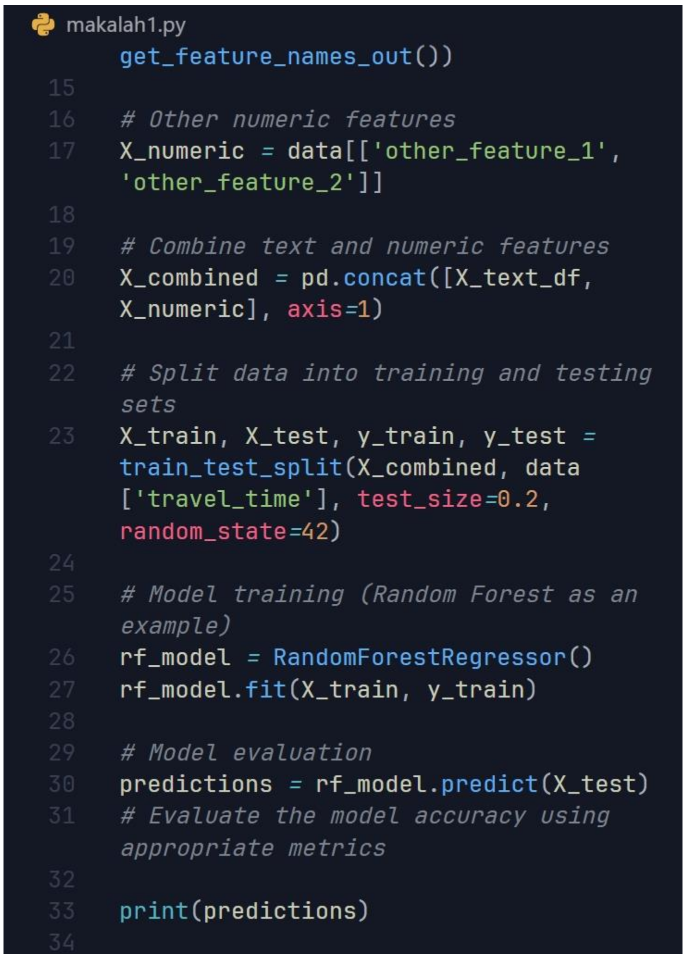
As already mentioned, NLP helps with calculating weight of the graph that consists of a lot of variables, in this method the weight will be calculated with one of the component of Natural Language Processing (NLP), which is text vectorization through the CountVectorizer from the sklearn.feature_extraction.text module. CountVectorizer is a technique commonly used in NLP to convert text data into a numerical format that machine learning algorithms can process.

In the provided code, the text data is converted into a numeric representation by creating a matrix where each row corresponds to a text sample, and each column represents a word/token in the text samples, with the cell values indicating the count of occurrences of each word in each text sample. Here's how it looks:



makalah1.py

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestRegressor

# Read the CSV file into a pandas DataFrame
data = pd.read_csv('C:\\Users\\ASUS\\OneDrive - Institut Teknologi Bandung\\itebeh\\alstrukdat\\prak12\\historical_traffic_data.csv')

# Preprocessing - Vectorize the text data
vectorizer = CountVectorizer()
X_text = vectorizer.fit_transform(data['text_column'])

# Convert the CountVectorizer output to a DataFrame with string column names
X_text_df = pd.DataFrame(X_text.toarray(), columns=vectorizer.get_feature_names_out())
```



makalah1.py

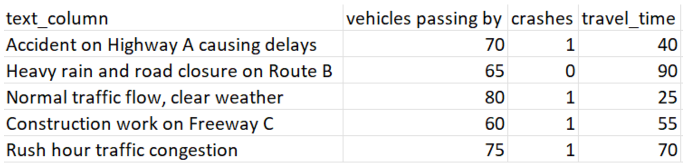
```python
get_feature_names_out())

# Other numeric features
X_numeric = data[['other_feature_1', 'other_feature_2']]

# Combine text and numeric features
X_combined = pd.concat([X_text_df, X_numeric], axis=1)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_combined, data['travel_time'], test_size=0.2, random_state=42)

# Model training (Random Forest as an example)
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)

# Model evaluation
predictions = rf_model.predict(X_test)
# Evaluate the model accuracy using appropriate metrics

print(predictions)
```

Fig. 9. Simplified NLP Algorithm in Python Code

Fig. 9 employs a Random Forest Regressor, a type of ensemble learning model known for its robustness and ability to handle complex relationships within data. This model is used for regression tasks, particularly in predicting travel times. The code utilizes the CountVectorizer from the scikit-learn library to transform textual data into numerical representations, making it feasible for the machine learning model to interpret. By converting text into numerical features (word counts), CountVectorizer allows the inclusion of textual information alongside other numeric features within the dataset.

The script preprocesses the text data, combines it with additional numeric features, and splits the data into training and testing sets. Subsequently, it trains the Random Forest Regressor on the combined dataset, enabling the model to learn patterns and relationships among the features. Once trained, the model predicts travel times based on the test dataset's features. Although the code doesn't explicitly demonstrate detailed evaluation metrics, after predictions are generated, typical evaluation metrics such as Mean Absolute Error or R-squared could be employed to assess the model's accuracy and performance in estimating travel times. This approach, through combining text and numeric features within a Random Forest Regressor, aims to accurately predict travel times by leveraging both textual information and other relevant numeric data.

The numeric data used in this scenario is stored in an Excel file. This data includes various features or attributes relevant to the prediction of travel times. These features could encompass diverse aspects related to transportation networks, such as weather conditions, road types, traffic incidents, time of day, geographical details, or any other factors that might influence travel duration or speed.



text_column	vehicles passing by	crashes	travel_time
Accident on Highway A causing delays	70	1	40
Heavy rain and road closure on Route B	65	0	90
Normal traffic flow, clear weather	80	1	25
Construction work on Freeway C	60	1	55
Rush hour traffic congestion	75	1	70

Fig. 10. Numeric Data Mockup

Each row in the Excel file represents an instance or observation, such as a particular traffic event or situation. Columns in the Excel file represent different attributes or features associated with these instances. For instance, columns might contain data about the presence of road closures, the level of traffic congestion, weather conditions, specific road names, time of day, or any other factors that could impact travel time prediction.

The data in these columns provides the necessary information for training a machine learning model. However, in the provided code snippet, the text_column attribute from a CSV file represents the textual information, which is preprocessed using NLP techniques. The rest of the attributes are assumed to be already numeric in nature or preprocessed separately.

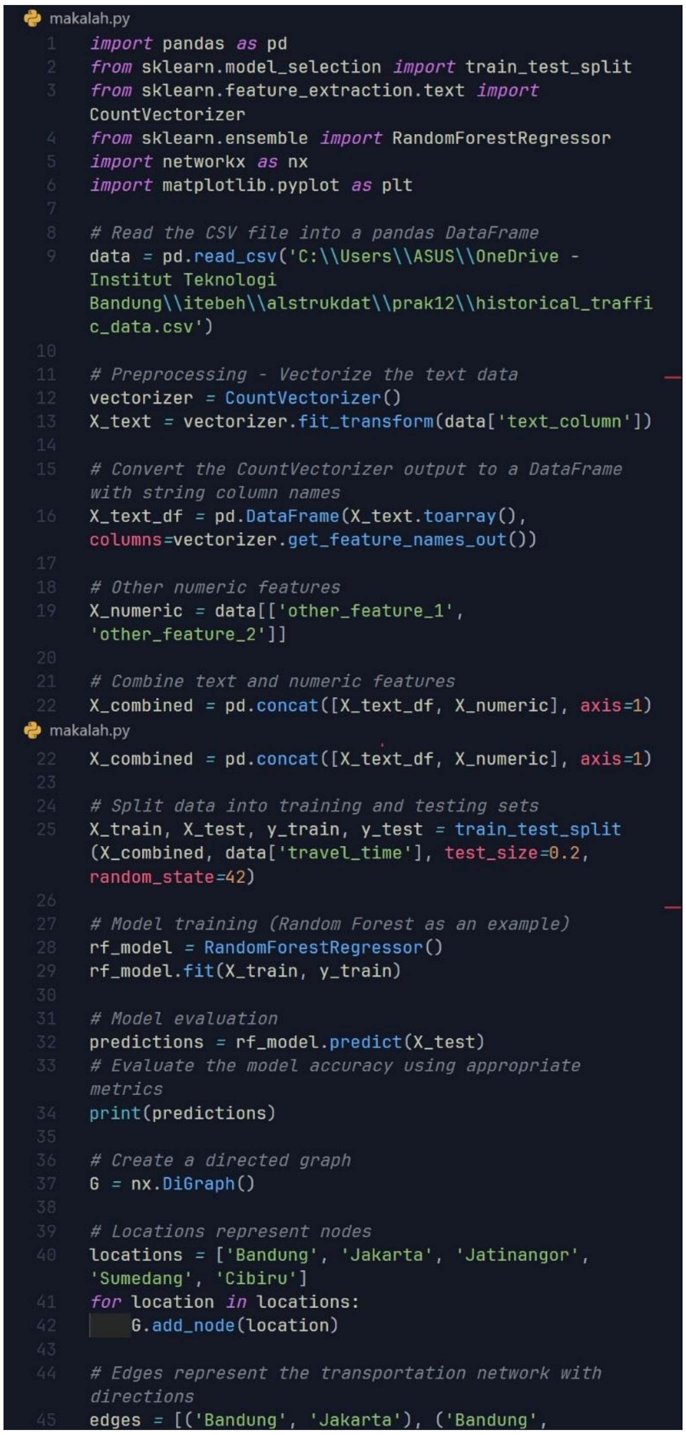
When reading the Excel file into a pandas DataFrame (data), it would contain columns representing different features related to transportation networks, and one of these columns (text_column) might contain textual descriptions or information about traffic situations, incidents, or conditions.

Combining these numeric features with the processed textual information aims to create a comprehensive dataset that integrates both types of data to improve the accuracy and efficacy of the machine learning model in predicting travel times accurately based on historical data and relevant attributes.

2. Combine the Directed Graph and Weight

The first part of the code, Fig. 7, processes the data, builds a directed graph representing various locations and their connections in a transportation network, and visualizes it using NetworkX and Matplotlib. This graph serves as the foundation for displaying travel time predictions.

After training the predictive model and obtaining travel time predictions, the code, Fig. 9, checks for available predictions. It then takes the first prediction and places it as the weight on the edge between 'Bandung' and 'Jakarta'. The weight is then displayed on the edge when the graph is visualized.

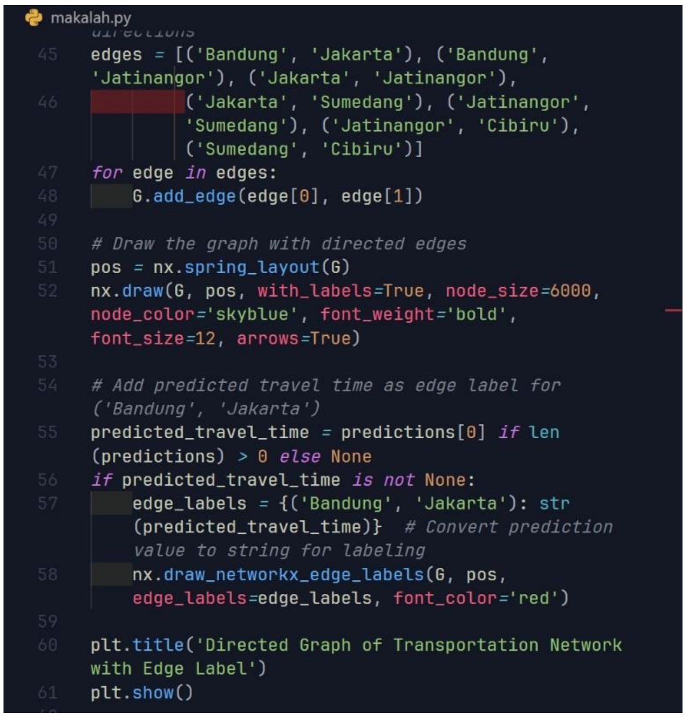


makalah.py

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestRegressor
import networkx as nx
import matplotlib.pyplot as plt

# Read the CSV file into a pandas DataFrame
data = pd.read_csv('C:\\Users\\ASUS\\OneDrive - Institut Teknologi Bandung\\itebeh\\alstrukdat\\prak12\\historical_traffic_data.csv')

# Preprocessing - Vectorize the text data
vectorizer = CountVectorizer()
X_text = vectorizer.fit_transform(data['text_column'])

# Convert the CountVectorizer output to a DataFrame with string column names
X_text_df = pd.DataFrame(X_text.toarray(), columns=vectorizer.get_feature_names_out())

# Other numeric features
X_numeric = data[['other_feature_1', 'other_feature_2']]

# Combine text and numeric features
X_combined = pd.concat([X_text_df, X_numeric], axis=1)
```

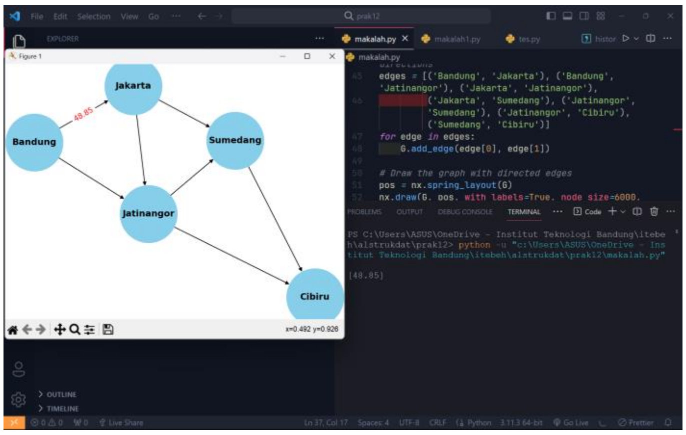
makalah.py

```python
X_combined = pd.concat([X_text_df, X_numeric], axis=1)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_combined, data['travel_time'], test_size=0.2, random_state=42)

# Model training (Random Forest as an example)
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)

# Model evaluation
predictions = rf_model.predict(X_test)
# Evaluate the model accuracy using appropriate metrics
print(predictions)

# Create a directed graph
G = nx.DiGraph()

# Locations represent nodes
locations = ['Bandung', 'Jakarta', 'Jatinangor', 'Sumedang', 'Cibiru']
for location in locations:
    G.add_node(location)

# Edges represent the transportation network with directions
edges = [('Bandung', 'Jakarta'), ('Bandung',
```



```
makalah.py
edges = [('Bandung', 'Jakarta'), ('Bandung',
'Jatinangor'), ('Jakarta', 'Jatinangor'),
                ('Jakarta', 'Sumedang'), ('Jatinangor',
                'Sumedang'), ('Jatinangor', 'Cibiru'),
                ('Sumedang', 'Cibiru')]
for edge in edges:
    G.add_edge(edge[0], edge[1])

# Draw the graph with directed edges
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_size=6000,
node_color='skyblue', font_weight='bold',
font_size=12, arrows=True)

# Add predicted travel time as edge label for
('Bandung', 'Jakarta')
predicted_travel_time = predictions[0] if len
(predictions) > 0 else None
if predicted_travel_time is not None:
    edge_labels = {('Bandung', 'Jakarta'): str
    (predicted_travel_time)}  # Convert prediction
    value to string for labeling
    nx.draw_networkx_edge_labels(G, pos,
    edge_labels=edge_labels, font_color='red')

plt.title('Directed Graph of Transportation Network
with Edge Label')
plt.show()
```

Fig. 10. Final Simplified Python Code of Using GNNs and NLP


File Edit Selection View Go
prak12
EXPLORER
makalah.py
makalah1.py
tes.py
Figure 1
Jakarta
Bandung
Sumedang
Jatinangor
Cibiru
48.85
x=0.492 y=0.926
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
OUTLINE
TIMELINE
Ln 37, Col 17 Spaces: 4 UTF-8 CRLF Python 3.11.3 64-bit Go Live Prettier

```
edges = [('Bandung', 'Jakarta'), ('Bandung',
'Jatinangor'), ('Jakarta', 'Jatinangor'),
                ('Jakarta', 'Sumedang'), ('Jatinangor',
                'Sumedang'), ('Jatinangor', 'Cibiru'),
                ('Sumedang', 'Cibiru')]
for edge in edges:
    G.add_edge(edge[0], edge[1])

# Draw the graph with directed edges
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_size=6000,
```

```
PS C:\Users\ASUS\OneDrive - Institut Teknologi Bandung\itebe
h\alstrukdat\prak12> python -u "c:\Users\ASUS\OneDrive - Ins
titut Teknologi Bandung\itebeh\alstrukdat\prak12\makalah.py"

[48.85]
```

Fig. 11. Final Result

From Fig. 11, the prediction result from Fig.10 shown in the directed graph, making it successfully created a desired weight-directed graph, the weight shown in the graph is only one because the data of the description is only one (Fig. 10), hence the weight that is shown in the graph is only the edge that has the data.

IV. Conclusion

In conclusion, this paper highlights the successful implementation of simplified Graph Neural Networks (GNNs) and Natural Language Processing (NLP) techniques to predict traffic conditions. By leveraging machine learning models like RandomForestRegressor and incorporating textual data preprocessing alongside network graph representations, achieved a predictive framework for estimating travel times in a transportation network.

The application of GNNs allowed us to model the transportation network as a graph, representing locations as nodes and road connections as edges. Incorporating travel time predictions from the machine learning model into this graph provided an intuitive visualization of estimated travel durations between specific locations. This fusion of NLP and GNNs enabled us to grasp traffic dynamics affected by various conditions, such as accidents, road closures, weather, and rush hour, contributing to more informed decision-making regarding travel routes and conditions.

Moreover, the experiment showcased the superiority of the GNN and NLP-based approach over conventional methods in traffic prediction. The integration of machine learning models with graph representations captured the complex interactions between locations and traffic conditions more effectively than traditional methods. Conventional approaches often overlook the intricate relationships and textual data that significantly influence traffic, while our GNN and NLP-based approach incorporated these nuances, leading to more accurate traffic predictions.

This study underlines the potential of advanced machine learning techniques like GNNs and NLP in enhancing traffic prediction systems by leveraging network structures and textual information. It suggests that these methodologies hold promise for developing more robust and accurate traffic forecasting systems compared to conventional approaches.

References

[1] "Graph Theory-Discrete Mathematics (Types of Graphs)." Accessed: Dec. 09, 2023. [Online]. Available: https://byjus.com/maths/graph-theory/

[2] "Graph & Graph Models." Accessed: Dec. 09, 2023. [Online]. Available: https://www.tutorialspoint.com/discrete_mathematics/graph_and_graph_models.htm

[3] "Digraphs, theory, algorithms, applications." Accessed: Dec. 09, 2023. [Online]. Available: https://www.cs.rhul.ac.uk/books/dbook/

[4] "Definitions." Accessed: Dec. 09, 2023. [Online]. Available: https://discrete.openmathbooks.org/dmoi3/sec_gt-intro.html

[5] "5.11: Directed Graphs - Mathematics LibreTexts." Accessed: Dec. 09, 2023. [Online]. Available: https://math.libretexts.org/Bookshelves/Combinatorics_and_Discrete_Mathematics/Combinatorics_and_Graph_Theory_%28Guichard%29/05%3A_Graph_Theory/5.11%3A_Directed_Graphs

[6] S. M. Ross, "Directed Graphs," *Topics in Finite and Discrete Mathematics*, pp. 150–179, Jul. 2000, doi: 10.1017/CBO9780511755354.007.

[7] "Directed and Undirected graph in Discrete Mathematics - javatpoint." Accessed: Dec. 09, 2023. [Online]. Available: https://www.javatpoint.com/directed-and-undirected-graph-in-discrete-mathematics

[8] "A Comprehensive Introduction to Graph Neural Networks (GNNs) | DataCamp." Accessed: Dec. 09, 2023. [Online]. Available: https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial

[9] X. Li and J. Saúde, "Explain Graph Neural Networks to Understand Weighted Graph Features in Node Classification," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12279 LNCS, pp. 57–76, 2020, doi: 10.1007/978-3-030-57321-8_4/COVER.

[10] "Reinforcement learning - GeeksforGeeks." Accessed: Dec. 09, 2023. [Online]. Available: https://www.geeksforgeeks.org/what-is-reinforcement-learning/

[11] P. D. Turney, "From Frequency to Meaning: Vector Space Models of Semantics," *Journal of Artificial Intelligence Research*, vol. 37, pp. 141–188, 2010, Accessed: Dec. 10, 2023. [Online]. Available: http://www.natcorp.ox.ac.uk/.

STATEMENT

In this statement, I declare that this paper I have written is my own work, not a duplication or translation of someone else's paper, and is not plagiarized.

Bandung, 10 December 2023

[illegible]

Mohammad Akmal Ramadan - 13522161