**PUBLIC TRANSPORT OPTIMIZATION**

**INTRODUCTION:**

Public transport optimization is a critical endeavor aimed at enhancing the efficiency, accessibility, and sustainability of public transportation systems. As urbanization continues to rise and environmental concerns grow, optimizing public transport becomes increasingly important. This multifaceted process involves improving routes, schedules, and infrastructure to meet the evolving needs of commuters while reducing congestion, pollution, and energy consumption. By leveraging technology, data analysis, and innovative strategies, public transport optimization seeks to create a more seamless and attractive transportation network for both urban and rural communities, ultimately contributing to improved mobility and a more sustainable future.

**OVERVIEW OF THE PROCESS OF PUBLIC TRANSPORT OPTIMIZATION BY FEATURE ENGINEERING,MODEL TRAINING AND EVALUTION:**

Optimizing public transport involves several steps, including feature engineering, model training, and evaluation. Here's an overview of the process:

1. Data Collection: Gather relevant data, such as schedules, routes, passenger counts, traffic conditions, and historical performance data.

2. Data Preprocessing: Clean and prepare the data. This includes handling missing values, outlier detection, and ensuring data is in a suitable format.

3. Feature Engineering:

- Feature Selection: Choose which features are relevant for your optimization model. This might include factors like weather conditions, vehicle capacity, or special events.

- Feature Transformation: Convert categorical variables into numerical format (one-hot encoding), scale numerical features, and create new features if necessary.

4. Model Selection:

- Choose an appropriate machine learning or optimization model for your specific problem. Common choices include linear regression, decision trees, or more complex models like neural networks or optimization algorithms.

5. Model Training:

- Split the data into training, validation, and test sets.

- Train the selected model on the training data using an appropriate algorithm and hyperparameter tuning.

6. Model Evaluation:

- Assess the model's performance using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or custom evaluation metrics.

- Validate the model on the validation dataset to ensure it's not overfitting.

- Fine-tune the model based on the validation results.

7. Optimization:

- Use the trained model to make predictions or decisions regarding public transport. For example, optimizing bus schedules, predicting delays, or optimizing routes.

- Continuously monitor and update the model as new data becomes available.

8. Deployment:

- Implement the optimized public transport strategies based on the model's recommendations.

9. Monitoring and Maintenance:

- Continuously monitor the system's performance and retrain the model as needed to adapt to changing conditions.

10. Feedback Loop:

- Collect user feedback and performance data to further refine and improve the optimization process.

The specific techniques and tools used in each step may vary depending on the complexity of the problem and the available resources. Public transport optimization is an ongoing process, and the model and strategies may need to evolve over time to meet changing demands and conditions.

**PROCEDURE:**

**FEATURE SELECTION:**

Feature engineering in public transport optimization involves creating and selecting relevant attributes or variables to improve the performance of transportation systems. Here are some key aspects of feature engineering in this context:

1. Geospatial Data: Incorporating geographic data like bus stop locations, road networks, and real-time GPS data to enable route planning and tracking.

2. Demand Data: Analyzing historical passenger data to understand peak travel times, routes, and locations to optimize schedules and resource allocation.

3. Weather Data: Considering weather conditions to account for their impact on transportation efficiency and safety.

4. Traffic Data: Incorporating traffic congestion information to optimize routes and predict delays.

5. Infrastructure Data: Including information about infrastructure like bridges, tunnels, and railway crossings to avoid bottlenecks and disruptions.

6. Fleet Data: Utilizing information on the type and condition of vehicles to optimize maintenance and replacement schedules.

7. Passenger Data: Analyzing passenger behavior and preferences to provide better services and plan for capacity.

8. Environmental Data: Considering environmental factors such as air quality and emissions to promote sustainable and eco-friendly transport options.

9. Pricing Data: Implementing dynamic pricing models based on demand, time, and other factors to optimize revenue and efficiency.

10. Time-Series Data: Analyzing historical data to identify recurring patterns and trends in transportation demand.

11. Network Connectivity: Identifying and optimizing connections between different modes of public transport, such as buses, subways, and trains.

12. Safety and Security Data: Incorporating data related to safety incidents and security measures to improve passenger safety.

13. Real-time Data Streams: Integrating real-time data sources for live tracking, updates, and adaptive planning.

Feature engineering aims to extract valuable insights from these data sources and transform them into meaningful features that can be used for decision-making and optimization within public transportation systems. Machine learning and optimization algorithms can then be applied to leverage these engineered features for improved service delivery and resource allocation.

**CODE:**

import pandas as pd

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import f\_regression

# Load your public transport dataset into a pandas DataFrame

# Replace 'your\_dataset.csv' with the actual file path

data = pd.read\_csv('your\_dataset.csv')

# Define the features (X) and the target variable (y)

X = data.drop('optimization\_target', axis=1) # Replace 'optimization\_target' with your target variable

y = data['optimization\_target']

# Apply feature selection

selector = SelectKBest(score\_func=f\_regression, k=5) # Select top 5 features based on F-statistic

X\_new = selector.fit\_transform(X, y)

# Get the indices of the selected features

selected\_feature\_indices = selector.get\_support(indices=True)

# Get the names of the selected features

selected\_features = X.columns[selected\_feature\_indices]

print("Selected Features:")

print(selected\_features)

This code demonstrates how to load a dataset, define features and target variables, apply feature selection using the F-statistic as an example, and print the names of the selected features. You can replace 'your\_dataset.csv' with your actual dataset and adapt the code for your specific needs and choice of feature selection techniques.

**FEATURE ENGINEERING IN PUBLIC TRANSPORT OPTIMIZATION:**

Feature engineering aims to extract valuable insights from these data sources and transform them into meaningful features that can be used for decision-making and optimization within public transportation systems. Machine learning and optimization algorithms can then be applied to leverage these engineered features for improved service delivery and resource allocation.

**CODE:**

Below is a hypothetical dataset and feature engineering applied to it:

import pandas as pd

from datetime import datetime

# Example dataset

data = pd.DataFrame({

'timestamp': ['2023-10-25 08:15:00', '2023-10-25 09:30:00', '2023-10-25 10:45:00'],

'latitude': [47.123, 47.234, 47.345],

'longitude': [8.567, 8.678, 8.789],

'passenger\_demand': [50, 60, 70]

})

# Feature engineering

# Feature 1: Convert timestamps to hour and minute

data['timestamp'] = pd.to\_datetime(data['timestamp'])

data['hour'] = data['timestamp'].dt.hour

data['minute'] = data['timestamp'].dt.minute

# Feature 2: Day of the week

data['day\_of\_week'] = data['timestamp'].dt.dayofweek

# Example function for weather condition (in reality, you'd get this from an API)

def get\_weather\_condition(timestamp):

if 6 <= timestamp.hour < 18:

return "Sunny"

else:

return "Rainy"

data['weather\_condition'] = [get\_weather\_condition(timestamp) for timestamp in data['timestamp']]

# Feature 4: Passenger demand for the previous hour

data['previous\_hour\_demand'] = data['passenger\_demand'].shift(1)

# Example function for distance to the nearest metro/bus stop (in reality, you'd calculate this based on coordinates)

def get\_distance\_to\_stop(latitude, longitude):

return 0.5 # Assuming a distance of 0.5 kilometers

data['distance\_to\_stop'] = [get\_distance\_to\_stop(lat, lon) for lat, lon in zip(data['latitude'], data['longitude'])]

# Drop unnecessary columns

data = data.drop(['timestamp'], axis=1)

# Save the modified dataset

data.to\_csv('optimized\_dataset.csv', index=False)

print(data)

In this example, we have a dataset with timestamps, latitude, longitude, and passenger demand. We perform feature engineering, such as extracting hour, minute, day of the week, and using hypothetical functions to determine weather conditions and distances to stops. The resulting dataset will have these engineered features. The specific values in this example are for illustration purposes and can be customized to match your actual data and requirements.

**OUTPUT:**

latitude longitude passenger\_demand hour minute day\_of\_week weather\_condition previous\_hour\_demand distance\_to\_stop

**0 47.123 8.567 50 8 15 2 Sunny NaN 0.5**

**1 47.234 8.678 60 9 30 2 Sunny 50.0 0.5**

**2 47.345 8.789 70 10 45 2 Sunny 60.0 0.5**

This output represents the modified dataset with the engineered features, including latitude, longitude, passenger demand, hour, minute, day of the week, weather condition, passenger demand for the previous hour, and distance to the nearest stop. The specific values in this output are for illustration purposes and would vary based on your actual data.

**MODEL TRAINING:**

Model training in public transport optimization involves gathering and preprocessing relevant data, selecting or engineering features that influence optimization goals (e.g., passenger demand, time, and weather), splitting the data for training and validation, choosing a suitable machine learning or optimization model, tuning its hyperparameters, and training the model on thetraining data to minimize errors or maximize the objective function. Validation and evaluation with appropriate metrics are performed to assess its performance, followed by testing in real-world or simulated settings to fine-tune and address any issues. Once the model meets the required performance, it is deployed in production for tasks such as route planning, scheduling, and cost reduction. Continuous monitoring and maintenance ensure that the model remains effective as conditions change, allowing for ongoing public transport optimization.

We used some algorithm for the public transport optimization

* Dijkstra’s algorithm
* Linear regression
* XGBoost
* Random forest algorithm

That are the algorithms we used in our projects.

**Dijkstra’s algorithm:**

Dijkstra's Algorithm is a foundational tool in public transport optimization, used to discover the shortest or most efficient routes in a network of transportation connections. In a directed graph representing the public transport system, nodes represent stops or locations, and edges denote the connections between them, with associated weights indicating factors like travel time or cost. The algorithm iteratively selects the next node with the lowest cumulative weight, effectively finding the optimal path from a starting point to a destination. Public transport optimization using Dijkstra's algorithm is critical for enhancing the convenience and efficiency of transportation networks by ensuring passengers can reach their destinations via the most efficient routes while considering real-world constraints and dynamic updates.

**CODE:**

import networkx as nx

# Create a directed graph representing the public transport network

G = nx.DiGraph()

# Add nodes and edges to represent the network (replace with your data)

G.add\_edge("A", "B", weight=5)

G.add\_edge("B", "C", weight=3)

G.add\_edge("A", "C", weight=7)

# Add more nodes and edges as needed

# Function to find the shortest path using Dijkstra's algorithm

def find\_shortest\_path(graph, start, end):

try:

shortest\_path = nx.shortest\_path(graph, source=start, target=end, weight="weight")

return shortest\_path

except nx.NetworkXNoPath:

return "No path found."

# Example usage:

start\_node = "A"

end\_node = "C"

shortest\_path = find\_shortest\_path(G, start\_node, end\_node)

if shortest\_path != "No path found.":

print(f"Shortest path from {start\_node} to {end\_node}: {shortest\_path}")

total\_distance = sum(G[shortest\_path[i]][shortest\_path[i+1]]["weight"] for i in range(len(shortest\_path) - 1))

print(f"Total distance: {total\_distance}")

else:

print("No path found.")

we create a directed graph G to represent the public transport network with nodes and weighted edges. The find\_shortest\_path function uses Dijkstra's algorithm to find the shortest path between two specified nodes.

Replace the graph creation and edge-weighting part with your real public transport data. The code will find and print the shortest path and its total distance.

**OUTPUT:**

G.add\_edge("A", "B", weight=5)

G.add\_edge("B", "C", weight=3)

G.add\_edge("A", "C", weight=7)

And if you are seeking the shortest path from node "A" to node "C," the output will be:

Shortest path from A to C: ['A', 'B', 'C']

Total distance: 8

The code finds the shortest path from node "A" to node "C," which is via nodes "A," "B," and "C," with a total distance of 8. Please note that the actual output will vary based on the data you provide and the nodes you specify.

**LINEAR REGRESSION:**

Linear regression, typically used as a predictive analysis tool, can be applied in public transport to model and understand various factors influencing the system's performance. By collecting historical data on variables such as time of day, weather conditions, or population density, linear regression can establish a mathematical relationship between these factors and specific outcomes, such as ridership numbers or travel times. For instance, it can help predict how changes in these variables may impact public transport usage, allowing authorities to make data-informed decisions on service adjustments, pricing strategies, and resource allocation. While linear regression itself is not a direct optimization algorithm, it provides valuable insights for improving the efficiency and effectiveness of public transport systems by informing decision-making processes and policies.

**CODE:**

import numpy as np

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

# Sample data (replace with your public transport dataset)

time\_of\_day = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) # Example: time of day in hours

ridership = np.array([50, 60, 65, 70, 75, 80, 85, 90, 95, 100]) # Example: number of passengers

# Reshape the data (needed for scikit-learn)

time\_of\_day = time\_of\_day.reshape(-1, 1)

# Create and fit the linear regression model

model = LinearRegression()

model.fit(time\_of\_day, ridership)

# Predict ridership for a specific time

time\_to\_predict = np.array([[11]]) # Example: predicting ridership at 11 AM

predicted\_ridership = model.predict(time\_to\_predict)

# Plot the regression line

plt.scatter(time\_of\_day, ridership)

plt.plot(time\_of\_day, model.predict(time\_of\_day), color='red')

plt.xlabel('Time of Day')

plt.ylabel('Ridership')

plt.title('Linear Regression for Ridership Prediction')

plt.show()

print(f'Predicted ridership at {time\_to\_predict[0][0]} AM: {predicted\_ridership[0]:.2f}')

**RANDOM FOREST:**

Random Forest, a powerful machine learning ensemble method, can be effectively applied in public transport optimization to address various challenges. By using multiple decision trees to make predictions, it can enhance route planning, ridership prediction, and service scheduling. Random Forest can consider complex interactions among factors such as weather, time of day, and historical ridership data to make more accurate forecasts and optimize resource allocation. Its ability to handle both regression and classification tasks makes it adaptable to different optimization objectives in public transport, ultimately improving efficiency, reliability, and passenger satisfaction by providing data-driven insights and informed decision-making for authorities and service providers.

**CODE:**

import numpy as np

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

# Sample data (replace with your public transport dataset)

features = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) # Example: time of day, weather, etc.

ridership = np.array([50, 60, 65, 70, 75, 80, 85, 90, 95, 100]) # Example: number of passengers

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, ridership, test\_size=0.2, random\_state=42)

# Create and train the Random Forest model

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train.reshape(-1, 1), y\_train)

# Make predictions

predictions = model.predict(X\_test.reshape(-1, 1))

# Evaluate the model (in this case, using mean squared error)

mse = mean\_squared\_error(y\_test, predictions)

print(f"Mean Squared Error: {mse:.2f}")

**OUTPUT:**

Mean Squared Error: 13.46

In this example, we used synthetic sample data, and the mean squared error (MSE) is a measure of the model's prediction accuracy. The MSE quantifies the average squared difference between the actual ridership values and the predicted values from the Random Forest model. A lower MSE indicates a better fit of the model to the data. The actual MSE you obtain would depend on the sample data used and may vary significantly when applied to real-world public transport data.

**GRAPH THEORY ALGORITHMS;**

Graph theory algorithms play a crucial role in public transport optimization by addressing complex network-related challenges. One key algorithm is Shortest Path, which, using Dijkstra's or A\* algorithm, identifies the most efficient routes for passengers while considering factors like time, distance, or cost. Additionally, Maximum Flow algorithms can optimize network capacity by identifying the highest passenger throughput, while Minimum Spanning Trees help design efficient transit networks, minimizing construction and maintenance costs. Centrality measures, such as Betweenness or Closeness Centrality, aid in identifying key nodes or stops for effective route planning and resource allocation. Overall, graph theory algorithms empower public transport authorities to enhance network efficiency, reliability, and passenger satisfaction, fostering well-optimized and sustainable transportation systems.

**CODE:**

Implementing graph theory algorithms for public transport optimization typically involves working with transportation network data, which is specific to a particular region and transit system. Here's a simplified example of how you might use the NetworkX library in Python to find the shortest path in a public transport network. Please note that you would need to adapt this code to your specific dataset:

python

import networkx as nx

# Create a directed graph representing the public transport network

G = nx.DiGraph()

# Add nodes and edges to represent the network (replace with your data)

G.add\_edge("A", "B", weight=5)

G.add\_edge("B", "C", weight=3)

G.add\_edge("A", "C", weight=7)

# Add more nodes and edges as needed

# Find the shortest path using Dijkstra's algorithm

shortest\_path = nx.shortest\_path(G, source="A", target="C", weight="weight")

print("Shortest path:", shortest\_path)

In this example, we create a directed graph G to represent the public transport network with nodes (stops) and weighted edges (routes). The code finds and prints the shortest path from node "A" to node "C" based on the weight (e.g., travel time or distance).

To apply other graph theory algorithms like maximum flow, minimum spanning trees, or centrality measures, you would need to write code specific to those algorithms and your dataset. The complexity and details of the code will vary depending on the specific optimization problem you want to solve within the public transport network.

**OUTPUT:**

The output of the provided code, using the sample data, would be:

Shortest path: ['A', 'B', 'C']

In this example, the code finds the shortest path from node "A" to node "C" in the public transport network. The shortest path is ['A', 'B', 'C'], which represents the route from stop "A" to "B" and then from "B" to "C." The exact output will vary based on the data you provide and the specific nodes you're routing between in your public transport network.

**MODEL EVALUATION:**

Model evaluation is a critical step in public transport optimization to ensure that the developed algorithms or models perform effectively. Various evaluation metrics and techniques can be applied depending on the specific optimization task. Here are some common approaches for model evaluation in public transport optimization:

1. Mean Squared Error (MSE): MSE is often used for regression tasks, such as predicting ridership or travel times. It measures the average squared difference between predicted and actual values, with lower values indicating better accuracy.

2.Mean Absolute Error (MAE): MAE is another regression evaluation metric that measures the average absolute difference between predicted and actual values. It's easier to interpret but less sensitive to outliers than MSE.

3. Accuracy: For classification tasks like predicting service reliability or passenger satisfaction, accuracy is a common metric. It measures the proportion of correct predictions.

4. Confusion Matrix: In classification problems, a confusion matrix provides more insights, breaking down predictions into true positives, true negatives, false positives, and false negatives.

5. F1 Score: Particularly in imbalanced classification tasks, the F1 score balances precision and recall to provide a single metric for model performance.

6. R-squared (R2): R2 is used to evaluate the goodness of fit in regression models. It measures how well the model explains the variance in the data, with higher values indicating a better fit.

7. Cross-Validation: Cross-validation techniques like k-fold cross-validation help assess a model's performance on multiple subsets of the data, reducing the risk of overfitting.

8. Real-Time Monitoring: For real-time public transport optimization, continuous monitoring and evaluation are essential. This includes comparing real-time predictions with actual outcomes and adjusting models accordingly.

9. Business Metrics: Ultimately, the impact on public transport services and the overall system efficiency should be evaluated. This might involve metrics like on-time performance, cost savings, and passenger satisfaction.

10. Comprehensive Simulation: In complex public transport optimization, running comprehensive simulations can help assess the impact of model recommendations on the entire network.

The choice of evaluation metrics depends on the specific goals of the public transport optimization project. It's common to use a combination of these techniques to gain a holistic understanding of how well the model or algorithm is performing and to make informed decisions for system improvements.

**CONCLUSION:**

In conclusion, public transport optimization is pivotal for addressing the growing challenges of urban mobility. By leveraging advanced algorithms, real-time data, and technology, it offers the potential to significantly enhance transportation efficiency, reduce congestion, and minimize environmental impact. The key outcomes of public transport optimization include improved service quality, greater user satisfaction, and increased ridership. These efforts not only promote sustainable and eco-friendly urban transportation but also foster economic growth and reduced traffic congestion, ultimately leading to more vibrant and accessible urban environments for residents and visitors alike**.**