Optimizing YouBike Redistribution via Machine Learning and Real-Time Forecasting

Hao-Chen, Lin Di-Ming, Yang

August 28, 2025

1 Introduction

Urban bike-sharing systems, such as YouBike in Taipei, play a significant role in reducing carbon emissions and alleviating traffic congestion. However, maintaining an optimal distribution of bikes across stations is a complex task due to fluctuating demand. This project proposes a machine learning-based system to forecast bike demand and assist in redistribution decisions using historical YouBike data.

2 Data Description

The dataset used in this study consists of YouBike station data from 2023, including temporal and station-specific information. The key features used are:

- **month** – Month of the observation
- **day** – Day of the month
- **hour** – Hour of the day
- **weekday** – Day of the week
- **is_holiday** – Whether the day is a holiday
- **station_id** – Encoded identifier for the station

The target variable is the **net_inflow**, defined as the number of bikes returned minus the number of bikes rented during a specific hour.

3 Methodology

3.1 Problem Formulation

This is a supervised regression task where the objective is to predict the net inflow of bikes at a given station and time, allowing us to detect which stations might be in surplus or deficit in the near future.

3.2 Models

Three types of models were explored:

- **XGBoost:** A gradient boosting algorithm suitable for structured data. Achieved the best performance with Test RMSE of 2.06.

- **DNN:** A feedforward deep neural network with multiple fully connected layers. Test RMSE was 2.78.
- **GRU:** A recurrent neural network using Gated Recurrent Units, implemented using TensorFlow. We structured the input as sequences of length 1 (shape: `(1, features)`), where each step represents the feature set at a given time. The model consists of:
  - One GRU layer with 32 units (default, tunable via hyperparameters)
  - A final dense layer with a single output neuron predicting the net inflow

  We used Mean Squared Error as the loss function and Adam as the optimizer. Training was performed with a batch size of 32 for 50 epochs.

Implementation Details

To maintain flexibility and scalability, we designed a modular training script that allows toggling between models such as XGBoost, DNN, or GRU via command-line flags.

Example usage:

```
Stations = predict_model('gru', '2024-06-15 18:00')
```

Instead of command-line arguments, the model type, prediction time, and station are specified directly within the Python script. The function `predict_model` returns a custom class instance containing the prediction results, which is subsequently used for the genetic algorithm (GA) optimization.

Each model is stored after training, allowing for consistent evaluation and future deployment. The architecture is abstracted into reusable components, enabling quick switching and comparison.

4 System Design

Command Line Operation

Users interact with the system directly through Python function calls rather than a terminal interface. By specifying the model type, target time, and station in the function arguments, the system loads the latest station status and generates a ranked list of dispatch priorities.

Example usage:

```
Stations = predict_model('gru', '2024-06-15 18:00')
```

The CLI will display:

- Forecasted net inflow per station
- Stations expected to experience shortage or surplus
- Suggested dispatch adjustments

This design enables seamless use within Python scripts and supports direct integration with the genetic algorithm (GA). For broader integration into automation pipelines or graphical user interfaces (GUI), additional wrappers such as command-line interfaces (CLI) or APIs can be implemented around the core function.

5 Evaluation

Model performance was evaluated using the Root Mean Squared Error (RMSE) metric on a held-out test set:

- XGBoost: RMSE = 2.06
- DNN: RMSE = 2.78
- GRU: RMSE = 2.95

XGBoost demonstrated the best performance, while GRU underperformed due to limited temporal signal and longer training times.

6 Discussion

6.1 Model Comparison

XGBoost outperformed both DNN and GRU, likely due to its robustness in handling structured data and categorical station identifiers. GRU, while theoretically strong in time-series tasks, struggled because the short window (24 hours) provided limited sequential information and training was computationally expensive.

6.2 Practical Considerations

We found that structuring input to the GRU as a full-day time series improved learning stability, even if long-term dependencies remained limited. The CLI-based dispatch simulation, while simple, proved effective in testing model suggestions and visualizing demand gaps in real-time.

To simulate realistic usage, we generated test inputs using real-world bike availability at various times. Forecasting errors were primarily due to model limitations in capturing temporal fluctuations and data sparsity at certain stations. Future work could explore the integration of external features (e.g., weather or events) to further improve robustness.

7 Conclusion

This project demonstrates the viability of using machine learning models for forecasting YouBike demand and supporting redistribution decisions. By offering real-time prediction and a command-line interface for generating dispatch suggestions, our system could aid city planners or operations staff in improving the balance of public bike availability.

8 Future Work

- Integrate weather and event data to enhance prediction
- Explore more advanced temporal models (e.g., TCNs, Transformers)
- Build a web-based dashboard for real-time monitoring

Appendix

GitHub Repository

Ubike-National-Carbon-Reduction-Plan (Private)