Intelligent Control Final Project

Prediction of YouBike station bike parking spaces based on Artificial Neural Networks (ANN)

Developer: Cheng Hao, Lin

I. Abstract:

In this project, we scraped the station information for YouBike in Taipei City and built an ANN to train the model to predict the number of parking spaces at each station after a specific number of minutes (set to 30 minutes in this project). Finally, we successfully trained an appropriate model that can be applied to datasets at any given time and developed a GUI interface to utilize the model.

II. Method:

a. Acquire the data: (figure 1 is the flowchart)

Data URL: https://tcgbusfs.blob.core.windows.net/dotapp/youbike/v2/youbike_immediate.json

1. Use the requests package in Python to make a request to the government's URL.
2. Wait for and receive the response, which is in JSON format.
3. Parse the data using the JSON package and save the data.

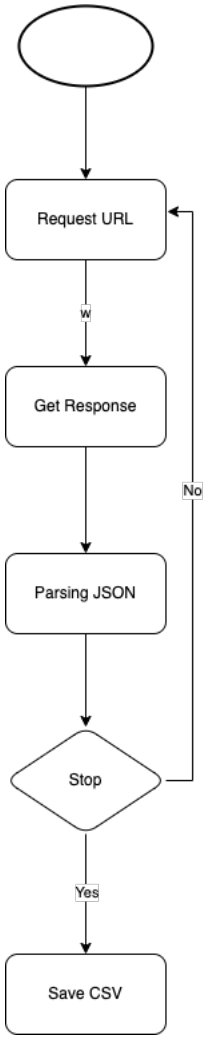

Figure 1 The flowchart for acquiring the data

b. Split the data: (figure 2 is the flowchart)

1. Read the CSV file from the previous step.
2. Splitting the required information, currently sno, tot, and sbi.

3. Shift the information of the specified minute to a new column sbi_xx_min, where "xx" represents the number of minutes. This column represents the "number of available spots after xx minutes.

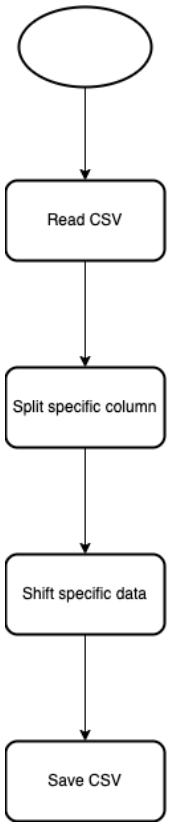

Figure 2 The flowchart for splitting the data

c. Training the data: (figure 3 is the flowchart)
   1. Splitting CSV file into input and output.
   2. An Artificial Neural Network (ANN) model is built using the PyTorch package.
   3. The hyperparameters of the model are adjusted to determine the appropriate architecture and values.
   4. The model is evaluated to assess its real-world application.

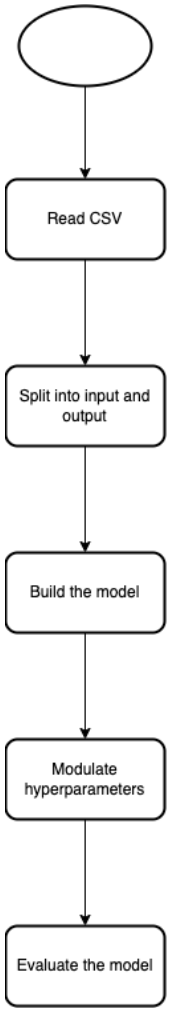

Figure 3 The flowchart for training the data

d. GUI: use pyqt6 to design a GUI and integrate the trained model.

III. Result and Discussion:

a. Figure 4 shows a code snippet that implements the acquisition of data. Since the JSON file on the Taipei City Government's official website is updated only once every minute, the code includes a sleep command of one minute at the end. Figure 5 displays the obtained results, with a total of 3785 data points collected.

```
url = 'https://tcgbusfs.blob.core.windows.net/dotapp/youbike/v2/youbike_immediate.json'
data = []  # save the json data

filename = 'data_test_gui.csv'
while True:
    data += json.loads(response.text)
    with open(filename, 'a', newline='') as csvfile:
        fieldnames = ['infoTime', 'tot', 'sbi', 'infoDate', 'aren', 'act', 'lng', 'srcUpdateTime', 'updateTime', 'sareaen', 'snaen', 'ar', 'bemp', 'mday', 'sarea', 'sna', 'lat', 'sno']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        for d in data:
            writer.writerow(d)
    # clear data to save new data
    data = []

    # wait a min
    time.sleep(60)
```

Figure 4 Code snippet for the acquisition of data

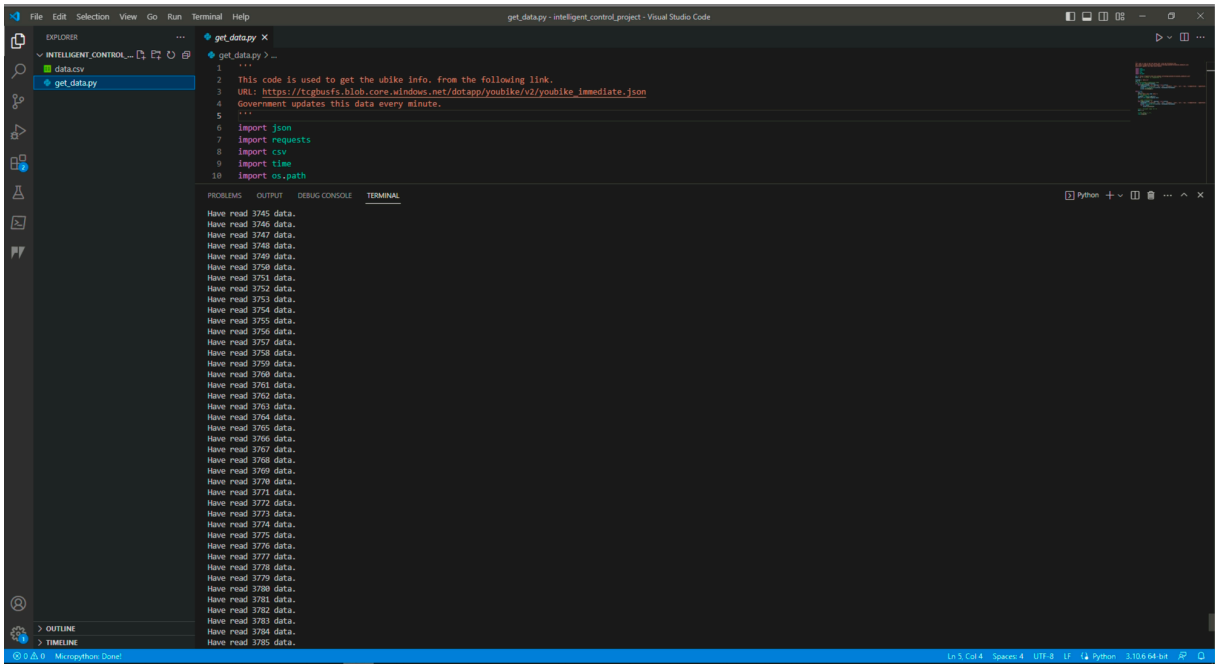

Figure 5 Obtained results

b. Figure 6 displayed the content of the data file from the previous step, while Figure 7 presented a code snippet for splitting the data. The goal was to organize the data into training datasets. In this project, we selected 'sno' (station number), 'tot' (total number), and 'sbi' (Number of available bicycles for rent.) as the training elements. The code that associated each vector with the number of parking spaces after 30 minutes (which could be defined by the user). Finally, Figure 8 showcased the output content after processing.

Figure 6 Obtained data

```python
# read data
data = pd.read_csv('data.csv')

# data transform
ann_data = data[['sno', 'tot', 'sbi']]
min = 30 # Customized data after a specific number of minutes.

sno_30min = (ann_data.iloc[0 + 1279 * min:]['sno'].values)
sbi_30min = (ann_data.iloc[0 + 1279 * min:]['sbi'].values)

datasets = pd.DataFrame()
datasets['sno'] = ann_data.iloc[:len(ann_data) - 1279 * min]['sno'].values
datasets['tot'] = ann_data.iloc[:len(ann_data) - 1279 * min]['tot'].values
datasets['sbi'] = ann_data.iloc[:len(ann_data) - 1279 * min]['sbi'].values
# add column
datasets['sno_30min'] = sno_30min
datasets['sbi_30min'] = sbi_30min
# save
datasets.to_csv('ann_datasets_test.csv', index=False)
```

Figure 7 Code snippet for splitting the data

```
sno,tot,sbi,sno_30min,sbi_30min
500101001,28,2,500101001,1
500101002,21,0,500101002,0
500101003,16,1,500101003,0
500101004,11,0,500101004,0
500101005,16,4,500101005,10
```

Figure 8 Training datasets

c. Figure 9 displayed the training process, where the input corresponded to the current 'sbi' (number of available parking spaces), and the prediction target was the 'sbi_30min' (number of parking spaces after 30 minutes). Figures 10 to 13 presented the results of various parameter experiments conducted in this project. Among them, Figure 13 illustrated the optimal configuration of model architecture and hyperparameters. Additionally, data from different days were collected for testing purposes. The loss obtained using the L1 loss function was 4.12.

```python
# build a list to save the loss data
train_losses = []
val_losses = []

for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()

    # forward propagration
    output = model(train_input_tensor)

    # calculate the training loss
    train_loss = criterion(output, train_output_tensor)

    # back propagation and update the weight
    train_loss.backward()
    optimizer.step()

    # # update learning rate
    # scheduler.step()

    # evaluate the model and calculate the validation loss
    model.eval()
    with torch.no_grad():
        val_output_pred = model(val_input_tensor)
        val_loss = criterion(val_output_pred, val_output_tensor)

    # save the value of loss
    train_losses.append(train_loss.item())
    val_losses.append(val_loss.item())

    # print the info.
    print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss.item()}, Val Loss: {val_loss.item()}")


```

Figure 9 The code snippet for training process

```
# define the hyperparameters
model_idx = 1
input_dim = train_input.shape[1]    # 1279
output_dim = train_output.shape[1]    # 1279
num_epochs = 200
learning_rate = 0.001
save_model_or_not = 0
MyModel(
  (model): Sequential(
    (0): Linear(in_features=1279, out_features=1279, bias=True)
  )
)
```

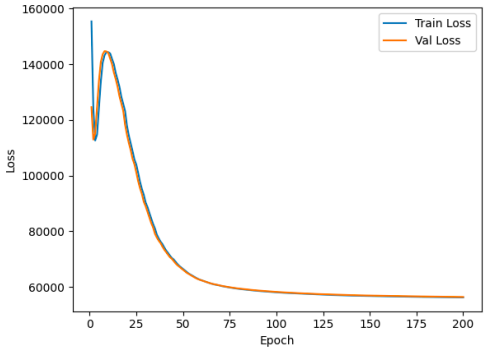

Figure 10 Model architecture, hyperparameter and result 1

```
# define the hyperparameters
model_idx = 1
input_dim = train_input.shape[1]    # 1279
output_dim = train_output.shape[1]    # 1279
num_epochs = 200
learning_rate = 0.001
save_model_or_not = 0
MyModel(
  (model): Sequential(
    (0): Linear(in_features=1279, out_features=512, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=512, out_features=512, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.2, inplace=False)
    (6): Linear(in_features=512, out_features=1279, bias=True)
  )
)
```

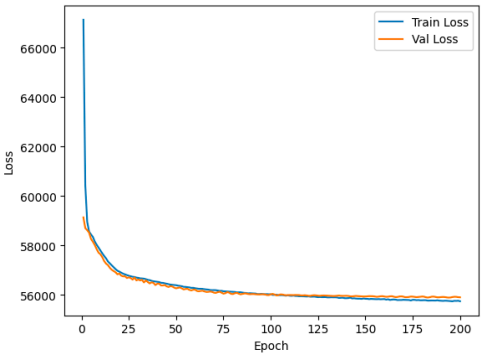

Figure 11 Model architecture, hyperparameter and result 2

```
# define the hyperparameters
model_idx = 3
input_dim = train_input.shape[1]    # 1279
output_dim = train_output.shape[1]    # 1279
num_epochs = 200
learning_rate = 0.001
save_model_or_not = 0
MyModel(
  (model): Sequential(
    (0): Linear(in_features=1279, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.01)
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): LeakyReLU(negative_slope=0.01)
    (4): Linear(in_features=512, out_features=1279, bias=True)
  )
)
```

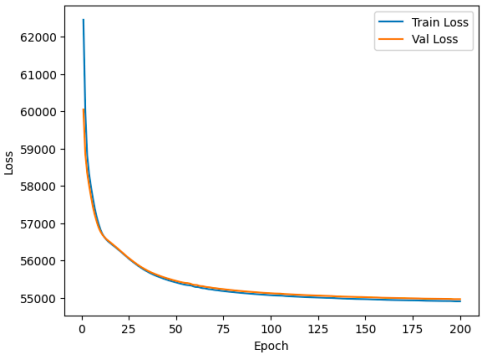

Figure 12 Model architecture, hyperparameter and result 3

```
# define the hyperparameters
model_idx = 4
input_dim = train_input.shape[1]    # 1279
output_dim = train_output.shape[1]    # 1279
num_epochs = 200
learning_rate = 0.001
save_model_or_not = 0
MyModel(
  (model): Sequential(
    (0): Linear(in_features=1279, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.01)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.01)
    (4): Linear(in_features=256, out_features=128, bias=True)
    (5): LeakyReLU(negative_slope=0.01)
    (6): Linear(in_features=128, out_features=64, bias=True)
    (7): LeakyReLU(negative_slope=0.01)
    (8): Linear(in_features=64, out_features=1279, bias=True)
  )
)
```

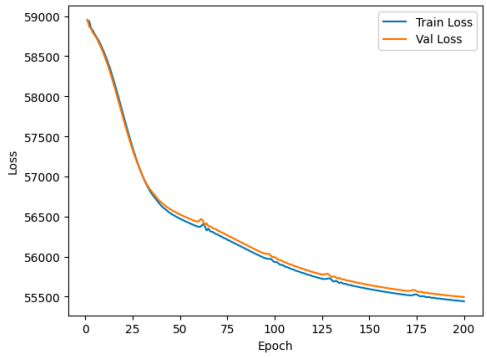

Figure 13 Model architecture, hyperparameter and result 4

d. Figure 14 showcased the GUI developed using the pyqt6 package.

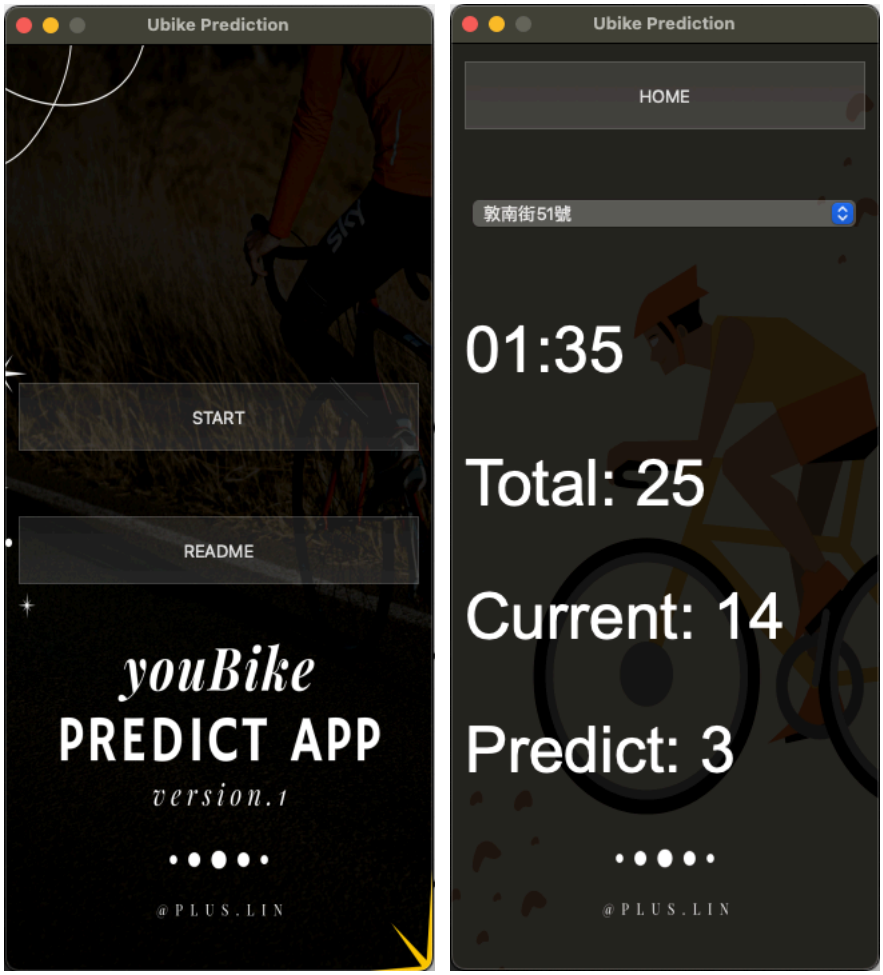

Figure 14 GUI

IV. Conclusion:

In this project, we developed a web scraping code to collect YouBike information from the Taipei City Government website. Then, we split this information for subsequent training of an Artificial Neural Network (ANN). During the ANN training, we experimented with four different model architectures and explored the effectiveness of ensemble learning. Finally, we developed a GUI that provides real-time predictions of the available parking spaces after 30 minutes.

V. Future development:

- Improve model accuracy
- Enhance data collection stability
- Add quantity prediction at any given time point

VI. Appendix:

YouTube link: https://youtu.be/f4is5F48LG0