ML Final Project Report

Ting-Chun Hung (D12922025)
Department of Computer Science and Information Engineering, National Taiwan University

1. Abstract

In this study, we applied machine learning methods, particularly Long Short-Term Memory networks (LSTMs), to predict the fluctuations in bicycle numbers at Taipei City's bike-sharing system stations. To achieve this, a comprehensive preprocessing of the bicycle usage data was undertaken, including handling missing values and feature engineering. During the data preprocessing phase, we employed forward and backward filling methods to address missing values, ensuring data integrity and continuity. This approach helps to minimize the impact of missing values on subsequent analyses and enhances the reliability of the prediction model.

In terms of feature engineering, we designed a set of time-series features. Specifically, we used data from every 20 minutes of the previous day as the training set to predict the bicycle count at the same time on the next day. Each sample included 72 time steps, with each step comprising 8 features: standardized station number, station bicycle capacity, date information (including day of the week and holiday status), standardized time information (hours and minutes), and the current bicycle count at the station (standardized relative to the station capacity). This multidimensional feature representation allowed us to capture key temporal and spatial factors affecting bicycle usage variations, without relying on additional data. Moreover, it enabled us to expand the training data by up to 20 times, due to the 72 time points every 20 minutes throughout a day, yielding 20 data sets per day.

We chose the Long Short-Term Memory network (LSTM) series as our predictive model, given its significant advantages in handling time-series data. The model was configured for 72 time steps and 8 features per sample. The strength of LSTM lies in its effective capture of temporal dependencies, crucial for understanding and predicting usage patterns in dynamic transportation systems. We experimented with variations of LSTM, including Dropout, Bidirectional-LSTM, and Finetuning sno-based weight, comparing them to the original LSTM. Ultimately, Finetuning sno-based weight showed the best performance. Our methodology combined feature engineering with a time-series deep learning model, finetuning each station with 50 epochs using a general-purpose parameter model as initial weight. The best-suited parameters for each station were saved using checkpoints. Through this approach, we achieved an error rate of 0.55474, using interpretable features and minimal training resources. 。

Keywords: Missing Value Imputation、Normalization、Feature Engineering、LSTM、Fine tune

2. Data Pre-processing:

2-1. Missing Value Imputation

In our dataset, there were some missing values, originally marked as -1. To better handle these missing values, we first replaced all -1s with NaN, enabling us to utilize Pandas' filling methods. Subsequently, we adopted the forward filling (ffill) method, where each NaN value is filled with the previous valid value. In cases where the dataset starts with NaN, we resorted to backward filling (bfill).

```python
def replace_missing_values(df, columns):
    for col in columns:
        df[col].replace(-1, np.nan).fillna(method='ffill').fillna(metho
```

2-2. Feature Organization

Feature organization is a crucial part of the preprocessing stage, involving the extraction of valuable information from raw data to be used as features. This includes the extraction of time-related features (such as hour, minute, accumulated time), date-based features (like day of the week, holiday status), and other station-related features (such as station number, total number of vehicles, vehicle status per minute). These features will be utilized in subsequent feature selection and model training. Given the large volume of data, threading was used to accelerate the process, employing important libraries including pandas, numpy, datetime, and ThreadPoolExecutor. Data from 1,327 stations, spanning from October 2, 2023, to December 24, 2023, was converted into feature information in approximately 30 minutes.

The initial feature set consists of 10 types of information: 'date', 'sno' (scaled down by dividing by 500101001), 'hour' (ranging from 0 to 23), 'minute', 'minute_accumulation' (ranging from 0 to 1439), 'day_code' (ranging from 1 to 7), 'holiday' (marked as +1 or -1), 'tot_acc', 'act_acc', and 'sbi_acc'.

	date	sno	hour	minuate	minuate_accumulation	day_code	holiday	tot_acc	act_acc	sbi_acc
Time:202310020	20231002.0	1.0	0.0	0.0	0.0	1.0	-1.0	28	1	1
Time:202310021	20231002.0	1.0	0.0	1.0	1.0	1.0	-1.0	21	1	0
Time:202310022	20231002.0	1.0	0.0	2.0	2.0	1.0	-1.0	21	1	0
Time:202310023	20231002.0	1.0	0.0	3.0	3.0	1.0	-1.0	21	1	0
Time:202310024	20231002.0	1.0	0.0	4.0	4.0	1.0	-1.0	21	1	0
...	...	...	...	...	...	...	...	...	...	...

Figure2. Feature Organization DataFrame

3. Experiment:

The experiments were based on two types of predictive goals, with the primary focus on (2), and a brief explanation of the reasons for not choosing the other. (3-1) Predicting the next minute's 'sbi_acc' (bicycle count). This was part of the initial research phase, where the direction of the experiment was adjusted through trials. This aspect will be primarily described. (3-2) Predicting 'sbi_acc' for every 20 minutes over a 72-minute period for the following day. This approach was chosen for preserving the original data information and is the main method adopted by our team. This report will primarily focus on introducing this method.

3-1. Predicting the next minute's 'sbi_acc':

3-1-1. Data:

(1) The training data is composed of the aforementioned feature information tracked back for 20 minutes, with the current minute's 'sbi_acc' serving as the label. (2) During model training, it was found that training took too long, so for ML, 112 main stations were used as training data.

3-1-2. Features:

A total of 21 time steps, each containing 10 features. (1) The data from the previous 20 minutes is used as time steps, each containing the 10 features mentioned above. (2) Statistical analysis of the 'sbi_acc' from the previous 20 minutes is conducted, including methods such as mean, median, standard deviation, variance, minimum value, maximum value, range value, skewness, kurtosis, and interquartile range (IQR).

3-1-3. Model Selection:

The models chosen were Sklearn LinearRegression, Sklearn SVR, RNN, and LSTM.

- The ML methods employed the following procedures for training and parameter selection:
  - Pipeline: Normalizer、PCA、LinearRegression / SVR
  - GridSearchCV & RandomSearchCV (CV=5)

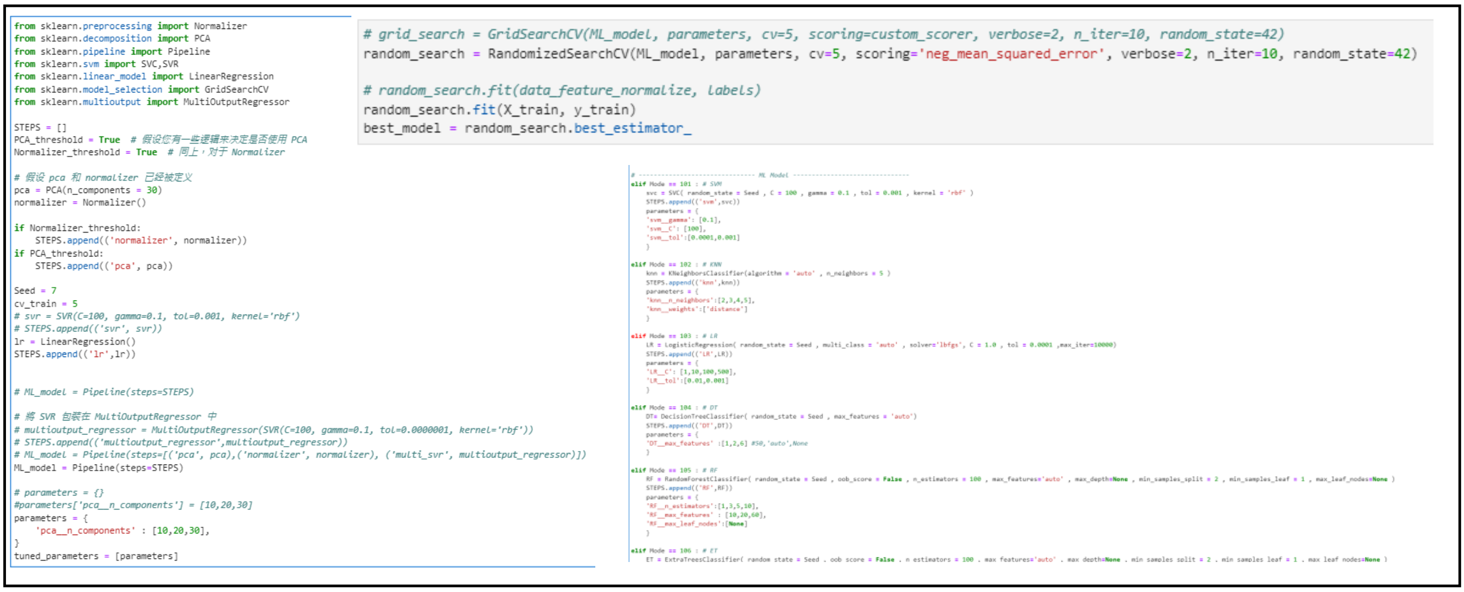


Figure3. Sklearn Implementation Tools

Additional Note: We also attempted to use classification models, treating 'sbi_acc' as 100 different classes (ranging from 0 to 100), but the results were similarly below expectations. The models tried included Sklearn's SVM (SVC), KNN, Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Extra Trees Classifier, Gradient Boosting Classifier, and AdaBoost Classifier. It's possible that ML models required more time for parameter tuning, but due to the excessively long training time, we decided to abandon training this data on ML models.

- The DL methods employed the following procedures for training and parameter selection:
  - Backbone: RNN、LSTM

```python
from keras import models, layers, losses, optimizers
from tensorflow.keras.callbacks import ModelCheckpoint

time_steps = 21  # 根据您的数据设置时间步
features = 10    # 特征数量
num_outputs = 1  # 回归模型的输出维度通常是 1

model = models.Sequential()
model.add(layers.SimpleRNN(
    batch_input_shape=(None, time_steps, features),  # time_steps, features
    units=50,
    unroll=True,
))
model.add(layers.Dense(units=num_outputs, kernel_initializer='normal'))  # 无激活函数

print(model.summary())

# 设置训练
model.compile(loss=losses.mean_squared_error,  # 回归任务常用的损失函数
              optimizer='adam',  # 优化器
              metrics=['mean_squared_error']  # 评估指标
              )
```

```python
from keras import models, layers, losses, optimizers

Mode = 'LSTM'
time_steps = 21  # 根据您的数据设置时间步
features = 10    # 特征数量
num_outputs = 1  # 回归模型的输出维度通常是 1

model = models.Sequential()
model.add(layers.LSTM(
    units=50,
    input_shape=(time_steps, features),  # time_steps, features
    unroll=True,
))
model.add(layers.Dense(units=num_outputs, kernel_initializer='normal'))  # 无激活函数

print(model.summary())

# 设置训练
model.compile(loss=losses.mean_squared_error,  # 回归任务常用的损失函数
              optimizer='adam',  # 优化器
              metrics=['mean_squared_error']  # 评估指标
              )
```

  - Using checkpoint to save the best epoch.

```python
Batch_size = 128
Epochs = 5
filepath = save_path_phase2+"model/LSTM_weights.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_mean_squared_error', verbose=1, save_best_only=True, save_weights_only=True, mode='auto', save_freq='epoch')
callbacks_list = [checkpoint]
logs = model.fit(X_train_train, y_train_train, batch_size=Batch_size, epochs=Epochs, validation_data=(X_train_val, y_train_val), callbacks=callbacks_list)
```

3-1-4. Results:

Findings from Usage: The method proved infeasible, with training results falling short of expectations, and the complete prediction of a week taking too long, leading to its abandonment. However, this attempt led to several discoveries: (1) Refocusing the Goal: A more efficient model prediction scenario would be on a daily basis. Additionally, since the sample submission includes a prediction target of less than 7 days, predicting for one day might be most appropriate for this final report. (2) After experimenting with a small dataset, we adjusted the Features: (i) removing 'date' & 'act_acc', (ii) normalizing the remaining features. (3) ML methods struggled with the load of 1316 stations. When using 112 stations as training data, Linear Regression performed better than SVR on this dataset, but the effectiveness was still poor, likely due to insufficient training data. DL methods could handle the load more efficiently, with LSTM showing better results, hence LSTM was chosen as the basis for this study.

3-2. Previous Day's 20 Minutes, Predicting Every 20 Minutes of the Next Day's 72 Minutes 'sbi_acc':

3-2-1. Features:

A total of 72 time steps, each with 8 features. Data from every 20 minutes of the previous day is used as training data, to predict every 20 minutes of the next day. Since a day has 1440 minutes, there are 20 training and label data pairs per day. Each set has 72 time steps, and each time step contains 8 features. The features I used are: (1) Station number, normalized to approximately between 1 and 1.1, (2) Station bicycle capacity, kept as original data, such as 17, 28, etc., (3) Date information - day of the week, normalized from 1/7 to 1, (4) Date information - holiday status, with 1 representing a holiday and -1 a non-holiday, (5) Time information - hour, originally from 0 to 23 hours, normalized to between 0 and 1, (6) Time information - minute, originally from 0 to 59 minutes, normalized to between 0 and 1, (7) Time information - accumulated time, originally from 0 to 1439, normalized to between 0 and 1, (8) Current moment's bicycle count, normalized to between 0 and 1. The normalization method involves dividing by the bicycle capacity of the station.

3-2-2. Model Selection:

- The DL methods were chosen based on the 'val_mean_squared_error' observed during the training process.:
  - Selection of LSTM Model Backbone (Whether to Include Dropout), During the training process, it was observed from the validation set carved out of the training data that the backbone I designed performed better on this dataset without dropout.
    - Non-Dropout

```python
from keras import models, layers, losses, optimizers

num_outputs = 72  # 回归模型的输出維度通常是

model = models.Sequential()
model.add(layers.LSTM(
    units=50,
    input_shape=(time_steps, features),  # time_steps, features
    unroll=True,
))
model.add(layers.Dense(units=num_outputs, kernel_initializer='normal'))  # 无激活函数

print(model.summary())

# 设置训练
model.compile(loss=losses.mean_squared_error,  # 回归任务常用的损失函数
              optimizer='adam',  # 优化器
              metrics=['mean_squared_error']  # 评估指标
              )
```

```
Epoch 1/5
13771/13771 [==============================] - 495s 35ms/step - loss: 0.0493 - mean_squared_error: 0.0493 - val_loss: 0.0464 - val_mean_squared_error: 0.0464

Epoch 00001: val_mean_squared_error improved from inf to 0.04641, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day/model\LSTM_weights.hdf5
Epoch 2/5
13771/13771 [==============================] - 442s 32ms/step - loss: 0.0470 - mean_squared_error: 0.0470 - val_loss: 0.0454 - val_mean_squared_error: 0.0454

Epoch 00002: val_mean_squared_error improved from 0.04641 to 0.04541, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day/model\LSTM_weights.hdf5
Epoch 3/5
13771/13771 [==============================] - 436s 32ms/step - loss: 0.0463 - mean_squared_error: 0.0463 - val_loss: 0.0448 - val_mean_squared_error: 0.0448

Epoch 00003: val_mean_squared_error improved from 0.04541 to 0.04481, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day/model\LSTM_weights.hdf5
Epoch 4/5
13771/13771 [==============================] - 459s 33ms/step - loss: 0.0458 - mean_squared_error: 0.0458 - val_loss: 0.0446 - val_mean_squared_error: 0.0446

Epoch 00004: val_mean_squared_error improved from 0.04481 to 0.04455, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day/model\LSTM_weights.hdf5
Epoch 5/5
13771/13771 [==============================] - 458s 33ms/step - loss: 0.0455 - mean_squared_error: 0.0455 - val_loss: 0.0438 - val_mean_squared_error: 0.0438

Epoch 00005: val_mean_squared_error improved from 0.04455 to 0.04377, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day/model\LSTM_weights.hdf5
```

    - Dropout

```python
from keras import models, layers, losses, optimizers, regularizers

time_steps = 72  # 假設您的時間步長為72
features = 8  # 假設您有8個特徵
num_outputs = 72  # 輸出維度

model = models.Sequential()
model.add(layers.LSTM(
    units=50,
    input_shape=(time_steps, features),
    unroll=True,
    dropout=0.25,  # 加入 dropout
    recurrent_dropout=0.25  # 加入 recurrent dropout
))
model.add(layers.Dense(
    units=num_outputs,
    kernel_initializer='normal',
    kernel_regularizer=regularizers.l2(0.01)  # 加入 L2 正則化
))
model.add(layers.Dropout(0.5))  # 在 Dense 層後加入額外的 Dropout 層

# 設定模型訓練
model.compile(
    loss=losses.mean_squared_error,  # 使用均方誤差作為損失函數
    optimizer='adam',
    metrics=['mean_squared_error']
)
```

```
Epoch 1/5
13771/13771 [==============================] - 427s 28ms/step - loss: 0.1281 - mean_squared_error: 0.1266 - val_loss: 0.0976 - val_mean_squared_error: 0.0963

Epoch 00001: val_mean_squared_error improved from inf to 0.09630, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day/mod
arizers_weights.hdf5
Epoch 2/5
13771/13771 [==============================] - 352s 26ms/step - loss: 0.1263 - mean_squared_error: 0.1251 - val_loss: 0.0965 - val_mean_squared_error: 0.0953

Epoch 00002: val_mean_squared_error improved from 0.09630 to 0.09528, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day
egularizers_weights.hdf5
Epoch 3/5
13771/13771 [==============================] - 353s 26ms/step - loss: 0.1260 - mean_squared_error: 0.1249 - val_loss: 0.0985 - val_mean_squared_error: 0.0974

Epoch 00003: val_mean_squared_error did not improve from 0.09528
Epoch 4/5
13771/13771 [==============================] - 354s 26ms/step - loss: 0.1259 - mean_squared_error: 0.1248 - val_loss: 0.0974 - val_mean_squared_error: 0.0962

Epoch 00004: val_mean_squared_error did not improve from 0.09528
Epoch 5/5
13771/13771 [==============================] - 352s 26ms/step - loss: 0.1258 - mean_squared_error: 0.1247 - val_loss: 0.0985 - val_mean_squared_error: 0.0974

Epoch 00005: val_mean_squared_error did not improve from 0.09528
```

  - Choosing between LSTM and BiLSTM. Found that training took twice as long, but the results were not better.

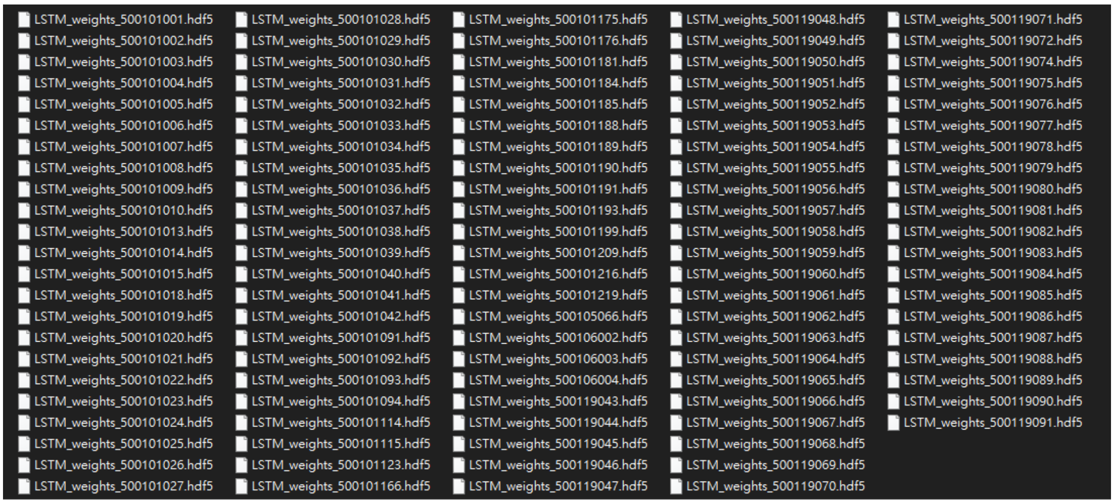
```python
from keras import models, layers, losses, optimizers

num_outputs = 72  # 回归模型的输出维度

# 定义模型
model = models.Sequential()
model.add(layers.Bidirectional(layers.LSTM(
    units=50,
    input_shape=(time_steps, features),  # time_steps, features
    unroll=True
)))
model.add(layers.Dense(units=num_outputs, kernel_initializer='normal'))  # 无激活函数


# 设置训练
model.compile(loss=losses.mean_squared_error,  # 回归任务常用的损失函数
              optimizer='adam',  # 优化器
              metrics=['mean_squared_error']  # 评估指标
              )
```

```
Epoch 1/5
13771/13771 [==============================] - 898s 63ms/step - loss: 0.0487 - mean_squared_error: 0.0487 - val_loss: 0.0463 - val_mean_squared_error: 0.0463

Epoch 00001: val_mean_squared_error improved from inf to 0.04630, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day/model\BiLSTM_weights.hdf5
Epoch 2/5
13771/13771 [==============================] - 907s 66ms/step - loss: 0.0465 - mean_squared_error: 0.0465 - val_loss: 0.0452 - val_mean_squared_error: 0.0452

Epoch 00002: val_mean_squared_error improved from 0.04630 to 0.04520, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day/model\BiLSTM_weights.hdf5
Epoch 3/5
13771/13771 [==============================] - 900s 65ms/step - loss: 0.0458 - mean_squared_error: 0.0458 - val_loss: 0.0445 - val_mean_squared_error: 0.0445

Epoch 00003: val_mean_squared_error improved from 0.04520 to 0.04452, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day/model\BiLSTM_weights.hdf5
Epoch 4/5
13771/13771 [==============================] - 896s 65ms/step - loss: 0.0453 - mean_squared_error: 0.0453 - val_loss: 0.0440 - val_mean_squared_error: 0.0440

Epoch 00004: val_mean_squared_error improved from 0.04452 to 0.04399, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day/model\BiLSTM_weights.hdf5
Epoch 5/5
13771/13771 [==============================] - 892s 65ms/step - loss: 0.0449 - mean_squared_error: 0.0449 - val_loss: 0.0440 - val_mean_squared_error: 0.0440

Epoch 00005: val_mean_squared_error improved from 0.04399 to 0.04389, saving model to D:/在職進修/修課/機器學習/Final_project/data_science_phase2_feature/feature1_day/model\BiLSTM_weights.hdf5
```

  - Finetuning, The LSTM weights trained on 1317 stations were used as initial weights, followed by further fine-tuning for the targeted 112 stations, with models being saved individually for each station.

```python
from IPython.display import clear_output
for sno in tqdm(target_sno):
    filepath = save_path_phase2+"model/finetune/LSTM_weights_"+sno+".hdf5"
    try:
        model.load_weights(filepath)
    except:
        model.load_weights(save_path_phase2+"model/LSTM_weights.hdf5")

    X_train_one = np.load(save_path_phase2+sno+'_X_train.npy')
    y_train_one = np.load(save_path_phase2+sno+'_y_train.npy')

    time_steps = 72 # 根据您的数据设置时间步
    features = 8 # 特征数量
    X_train_one_reshaped = X_train_one.reshape((X_train_one.shape[0], time_steps, features))
    y_train_one_reshaped = y_train_one.reshape((y_train_one.shape[0], time_steps, features))
    y_train_one_reshaped = y_train_one_reshaped[:, :, -1:]

    X_train_one_reshaped_train, X_train_one_reshaped_val, y_train_one_reshaped_train, y_train_one_reshaped_val =  X_train_one_reshaped[:60] , X_train_one_reshaped[60:], y_train_one_reshaped[:60] , y_trai

    Batch_size = 128
    Epochs = 50

    checkpoint = ModelCheckpoint(filepath, monitor='val_mean_squared_error', verbose=1, save_best_only=True, save_weights_only=True, mode='auto', save_freq='epoch')
    callbacks_list = [checkpoint]
    logs = model.fit(X_train_one_reshaped_train, y_train_one_reshaped_train, batch_size=Batch_size, epochs=Epochs, validation_data=(X_train_one_reshaped_val, y_train_one_reshaped_val), callbacks=callbacks
    clear_output()
```

```
100%|██████████████████████████████████████████████████| 112/112 [20:53<00:00, 11.19s/it]
```



3-2-3. Results evaluate:

前一天資料	LSTM	Dropout LSTM	BiLSTM	**Finetuned-LSTM**
MSE	0.06326	0.14708	0.06348	**0.06191**
Class_error	0.40388	0.55984	0.40457	**0.39295**

Table 1. Using data from the previous day to predict the situation for the following day.

連續預測 7 天	LSTM	Dropout LSTM	BiLSTM	**Finetuned-LSTM**
MSE	0.12144	0.22959	0.12395	**0.09276**
Class_error	0.65571	0.72913	0.65598	**0.55474**

Table 2. Using the data from the previous day to predict the next day's scenario, and then using that prediction as input to continue forecasting each subsequent day, until predictions for the entire week are completed.

Discussion

Due to time constraints, I have yet to explore the correlation between stations and the potential enhancement of results by randomly shuffling the dataset for training. Additionally, due to hardware limitations, attempting to train SVR using MultiOutputRegressor led to insufficient memory issues, preventing the training. Exploring ensemble learning and other methods remains a potential direction for future attempts. All this work was completed independently, without any division of tasks. This summarizes the content of my final report.

[class/ML at main · jeffhong824/class (github.com)](https://github.com/jeffhong824/class)