#%%

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import LSTM

from tensorflow.keras.layers import Dropout

from tensorflow.keras.models import load\_model

from tensorflow.keras.optimizers import Adam

from sklearn.preprocessing import MinMaxScaler

import joblib

import numpy as np

import pandas as pd

import os

# 設定LSTM往前看的筆數和預測筆數

LookBackNum = 12 # LSTM往前看的筆數

ForecastNum = 48 # 預測題目當天的筆數 9:00至16:59

# 載入訓練資料

# DataName = os.getcwd() + '/L17\_0331\_9.csv'

DataName = os.getcwd() + '/TrainingData/L17\_0331\_11.csv'

SourceData = pd.read\_csv(DataName, encoding='utf-8')

#迴歸分析 選擇要留下來的資料欄位

#(風速,大氣壓力,溫度,濕度,光照度)

#(發電量)

Regression\_X\_train = SourceData[['WindSpeed(m/s)','Pressure(hpa)','Temperature(°C)','Humidity(%)','Sunlight(Lux)']].values

Regression\_y\_train = SourceData[['Power(mW)']].values

# 選擇要留下來的資料欄位

# (風速,大氣壓力,溫度,濕度,光照度)

AllOutPut = SourceData[['WindSpeed(m/s)','Pressure(hpa)','Temperature(°C)','Humidity(%)','Sunlight(Lux)']].values

# 正規化

LSTM\_MinMaxModel = MinMaxScaler().fit(AllOutPut)

AllOutPut\_MinMax = LSTM\_MinMaxModel.transform(AllOutPut)

X\_train = []

y\_train = []

#設定每i-12筆資料(X\_train)就對應到第i筆資料(y\_train)

for i in range(LookBackNum,len(AllOutPut\_MinMax)):

# 這裡的 i 起啟值為 LookBackNum，所以 i-LookBackNum:i 就會變成 0:LookBackNum 取前 LookBackNum 筆資料

X\_train.append(AllOutPut\_MinMax[i-LookBackNum:i, :])

# 將取下一筆資料當目標標籤

y\_train.append(AllOutPut\_MinMax[i, :])

X\_train = np.array(X\_train)

y\_train = np.array(y\_train)

# Reshaping

#(samples 是訓練樣本數量,timesteps 是每個樣本的時間步長,features 是每個時間步的特徵數量)

X\_train = np.reshape(X\_train,(X\_train.shape [0], X\_train.shape [1], 5))

print(X\_train.shape)

#%%

#============================建置&訓練模型============================

#建置LSTM模型

regressor = Sequential ()

# 1st LSTM Layer (increased units slightly)

regressor.add(LSTM(units=300, return\_sequences=True, input\_shape=(X\_train.shape[1], 5)))

regressor.add(Dropout(0.3))

# 2nd LSTM Layer (increased units)

regressor.add(LSTM(units=256, return\_sequences=True))

regressor.add(Dropout(0.4))

# 3rd LSTM Layer (additional layer for added depth)

regressor.add(LSTM(units=128, return\_sequences=False))

regressor.add(Dropout(0.3))

# Fully Connected Dense Layers

regressor.add(Dense(units=128, activation='relu'))

regressor.add(Dropout(0.2))

regressor.add(Dense(units=64, activation='relu'))

regressor.add(Dropout(0.2))

# output layer

regressor.add(Dense(units = 5))

learning\_rate = 0.01 # 學習率

optimizer = Adam(learning\_rate=learning\_rate)

regressor.compile(optimizer = optimizer, loss = 'mean\_squared\_error')

#開始訓練

regressor.fit(X\_train, y\_train, epochs = 1000, batch\_size = 100)

#保存模型

# from datetime import datetime

# NowDateTime = datetime.now().strftime("%Y-%m-%dT%H\_%M\_%SZ")

regressor.save('WheatherLSTM\_Model.h5')

print('Model Saved')

#%%

#============================建置&訓練「回歸模型」========================

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import MinMaxScaler

#開始迴歸分析(對發電量做迴歸)

RegressionModel = GradientBoostingRegressor(n\_estimators=100, learning\_rate=0.01, max\_depth=5)

RegressionModel.fit(LSTM\_MinMaxModel.transform(Regression\_X\_train), Regression\_y\_train)

#儲存回歸模型

# from datetime import datetime

# NowDateTime = datetime.now().strftime("%Y-%m-%dT%H\_%M\_%SZ")

joblib.dump(RegressionModel, 'WheatherRegression\_Model')

# 取得 R squared 分數

print('R squared: ', RegressionModel.score(LSTM\_MinMaxModel.transform(Regression\_X\_train), Regression\_y\_train))

# 取得特徵重要性

print('Feature importances: ', RegressionModel.feature\_importances\_)

'''

預測數據

'''

# %%

#============================預測數據============================

#載入模型

regressor = load\_model('WheatherLSTM\_Model.h5')

Regression = joblib.load('WheatherRegression\_Model')

inputs = [] # 存放參考資料

PredictOutput = [] # 存放預測值

PredictPower = [] #存放預測值(發電量)

# 取輸入資料的最後 LookBackNum 筆

TempData = AllOutPut[-LookBackNum:].reshape(LookBackNum, 5) # (12, 5)

TempData = LSTM\_MinMaxModel.transform(TempData) # 正規化

print("TempData.shape:", TempData.shape)

print("TempData.size:", TempData.size)

inputs = [TempData] # 初始化 inputs，形狀為 (1, 12, 5)

print("inputs.shape:", np.array(inputs).shape)

# 預測迴圈

for i in range(ForecastNum) :

#將新的預測值加入參考資料(用自己的預測值往前看)

if i > 0 :

# 將新的預測值加入 inputs

PredictValue = PredictOutput[i-1].reshape(1, 5) # 預測值形狀為 (1, 5)

NewInput = np.vstack((inputs[-1][1:], PredictValue)) # 拼接，保留最後 LookBackNum 筆

inputs.append(NewInput) # 確保形狀為 (12, 5)

print(f"inputs[{i}]: {[x.shape for x in inputs]}")

print(f"type(inputs[-1]): {type(inputs[-1])}")

# 從 inputs 提取新的參考資料12筆(往前看12筆)

X\_test = np.array(inputs[-1]) # 使用最新的 inputs，形狀為 (12, 5)

X\_test = np.reshape(X\_test, (1, LookBackNum, 5)) # 確保形狀為 (batch\_size, timesteps, features)

print(f"X\_test.shape: {X\_test.shape}")

# 預測

predicted = regressor.predict(X\_test) # 預測輸出形狀 (1, 5)

print('X\_test = ', X\_test)

print("predicted.shape: ", predicted.shape)

PredictOutput.append(predicted)

PredictPower.append(np.round(Regression.predict(predicted),2).flatten())

# 最後檢查輸出

print("Final PredictOutput.shape:", np.array(PredictOutput).shape)

print("Final PredictPower.shape:", np.array(PredictPower).shape)

#寫預測結果寫成新的CSV檔案

# 將陣列轉換為 DataFrame

df = pd.DataFrame(PredictPower, columns=['答案'])

# 將 DataFrame 寫入 CSV 檔案

df.to\_csv('output.csv', index=False, encoding='utf-8-sig')

print('Output CSV File Saved')

# %%