

113-1 大數據分析與智慧運算 12/06 作業 2

指導老師：吳政瑋 助理教授

學生：資工三 B1143015 林宣佑

題目

台股或美股，1. 預測收盤價(回歸)，2. 預測漲跌(分類)

Github 開源 – Apache 2.0 (可商用)

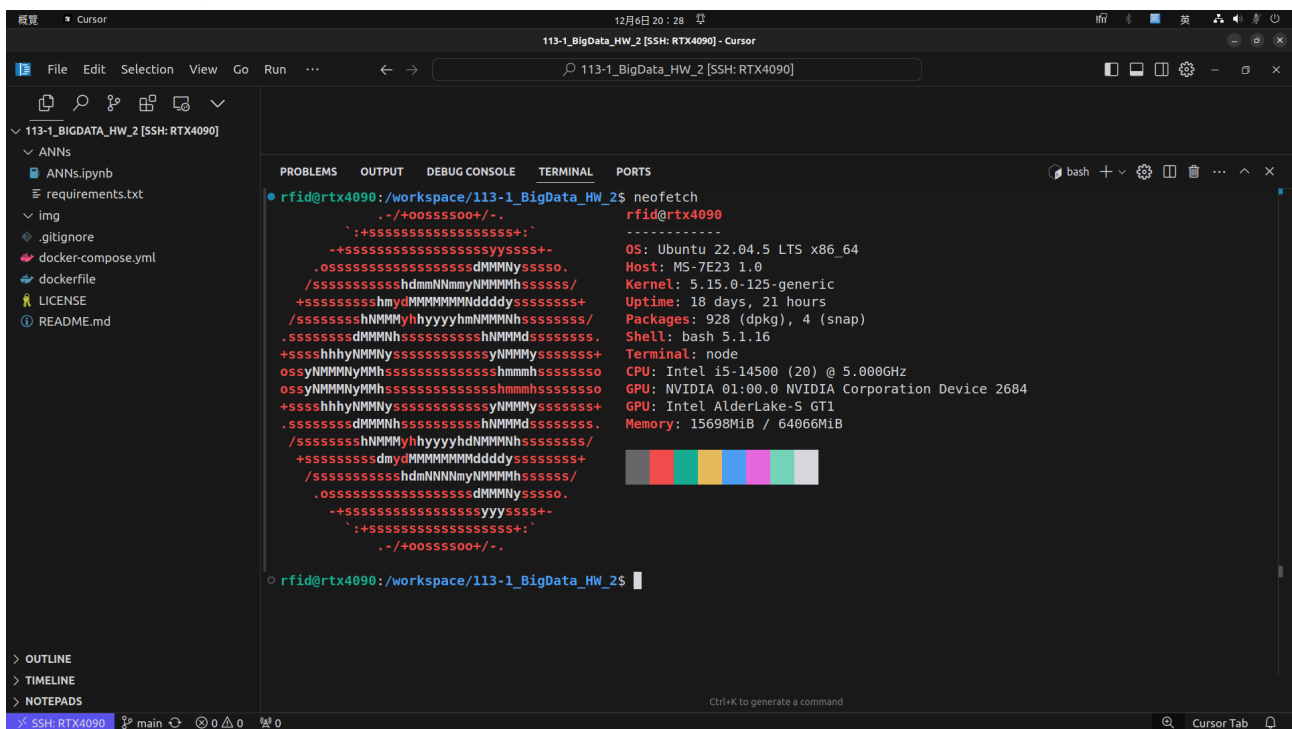
https://github.com/TsukiSama9292/113-1_BigData_HW_2

程式碼參考源也是原先我的開源儲存庫(同為 Apache 2.0)

程式碼

於 ANNs 資料夾中的 ANNs.ipynb

開發環境

A screenshot of a Cursor IDE terminal window. The terminal shows the output of the 'neofetch' command, which displays system information in a stylized ASCII art format. The information includes the OS (Ubuntu 22.04.5 LTS), host (MS-7E23), kernel (5.15.0-125-generic), uptime (18 days, 21 hours), packages (928 dpkg, 4 snap), shell (bash 5.1.16), terminal (node), CPU (Intel i5-14500), GPU (NVIDIA 01:00.0 NVIDIA Corporation Device 2684), and memory (15698MiB / 64066MiB). The terminal prompt is 'rfid@rtx4090: /workspace/113-1_BigData_HW_2\$'.

Docker 版本：24.0.7

GPU：Nvidia GeForce RTX 4090 24G

程式碼介紹

第一步，取得股票資訊

我使用了 twstock 套件，取得台股過去股票資訊，作為本次作業的資料。

套件安裝與載入

安裝其餘套件

```
# 台股資訊套件
!pip install -U -qqq twstock
!pip install -U -qqq lxml
# 數據處理套件
!pip install -U -qqq pandas numpy scikit-learn matplotlib
```

載入需求套件

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import lr_scheduler
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
import datetime
import time
import pickle
import csv
from twstock import Stock
import os
from huggingface_hub import HfApi
from safetensors.torch import save_file
import os
from sklearn.model_selection import train_test_split
```

第二步，進行資料處理

我將資料進行正規化，並且將資料分為訓練組及測試組，作為本次作業的訓練資料及測試資料。

由於預測收盤價和預測漲跌的資料處理不同，因此分為不同資料處理。

1. 預測收盤價(回歸) - Pytorch 深度學習 - B1143015 林宣佑

將 CSV 檔案讀取並處理成訓練和測試資料

```
# Get the current time
current_time = datetime.datetime.now()

# Format the time
formatted_time = current_time.strftime("%Y_%m_%d_%H_%M_%S")

print("Current time:", formatted_time)

# Read the data
data = pd.read_csv(filename)

# Select the necessary features and target variable
features = data[['Open', 'High', 'Low', 'Close', 'Volume']]
target = data['Change']

new_target = []
for value in target:
    new_target.append(value)

# Normalize the data using a dictionary of scalers
scalers = {}
scaled_features = []

for feature in features.columns:
    scaler = MinMaxScaler()
    scaled_feature = scaler.fit_transform(features[[feature]])
    scalers[feature] = scaler # Save the scaler for each feature
    scaled_features.append(scaled_feature)

# Convert the list of scaled features back to a DataFrame
scaled_features = np.hstack(scaled_features)
print("scaled_features : ", scaled_features[0])

# 保存 scaler 字典到檔案
with open('scalers.pkl', 'wb') as f:
    pickle.dump(scalers, f)
```

第三步，建立模型

模型最前面有一層 ReLU 激活函數，中間有 LSTM 雙層，最後有一層全連接層輸出。

```
class Stock_ANNs(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim, dropout):
        super(Stock_ANNs, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.relu = nn.Linear(input_dim, hidden_dim)
        self.lstm = nn.LSTM(hidden_dim, hidden_dim, num_layers, batch_first=True, dropout=dropout)
        self.output = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = F.relu(self.relu(x)) # ReLU
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_().to(device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_().to(device)
        out, (hn, cn) = self.lstm(x.to(device), (h0.detach(), c0.detach()))
        out = self.output(out[:, -1, :])
        return out.to(device)
```

第四步，訓練模型

我使用 Adam 優化器，對於回歸模型使用 MSE 損失函數，對於分類模型使用 CrossEntropy 損失函數。

```
# 設定模型各層維度
input_dim = X_train.shape[2] # 輸入
hidden_dim = 200 # 隱藏層維度
num_layers = 2 # LSTM 層數
output_dim = 1 # 輸出
dropout = 0.001 # 防止過擬合

# 訓練參數設置
epochs = 150 # 訓練次數
initial_lr = 5e-4 # 初始學習率
batch_size = 64 # 批次大小
criterion = nn.MSELoss() # 損失函數改為均方誤差

# 模型建立並移動到 device (CPU 或 GPU)
model = Stock_ANNs(input_dim=input_dim, hidden_dim=hidden_dim, num_layers=num_layers, output_dim=output_dim, dropout=dropout).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=initial_lr) # 優化器
scheduler = lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.975) # 調節學習率工具

# 訓練模型
print("\n*****Training Status*****")
train_start = time.time()
for epoch in range(1, epochs + 1):
    model.train() # 將模型設置為訓練模式
    total_loss = 0 # 記錄 LOSS
    start_time = time.time() # 記錄當前epoch的開始時間
    for i in range(0, len(X_train_tensor), batch_size):
        batch_X = X_train_tensor[i:i + batch_size] # 從訓練集中取出一個批次的特徵數據
        batch_y = y_train_tensor[i:i + batch_size] # 從訓練集中取出一個批次的目標數據
        outputs = model(batch_X) # 獲取模型的預測結果
        loss = criterion(outputs, batch_y.unsqueeze(1)) # 計算模型的損失，目標數據需要改為 (batch_size, 1)
        optimizer.zero_grad() # 梯度反向傳播和參數更新
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    # 每十次輸出訓練狀態
    if epoch % 10 == 0:
        print('| epoch {:3d} | lr {:.10f} | {:.2f} ms | loss {:.5f}'.format(
            epoch, scheduler.get_last_lr()[0], (time.time() - start_time) * 1000, total_loss))
        scheduler.step() # 每個epoch結束後調整學習率(記數)

train_times = time.time() - train_start # 計算整個訓練過程的總執行時間
print(f"Training cost: {train_times:.2f} seconds") # 打印整個訓練過程的執行時間
```

第五步，評估模型

我將模型進行評估，並且將結果進行視覺化。

```
# 評估模型
print("\n*****Eval Status*****")
model.eval()
test_outputs = model(X_test_tensor)

predicted = test_outputs.detach().cpu().numpy()
actual = y_test_tensor.detach().cpu().numpy()

# 評估指標
mae = np.mean(np.abs(predicted - actual)) # 平均絕對誤差
mse = np.mean((predicted - actual) ** 2) # 均方誤差
rmse = np.sqrt(mse) # 均方根誤差
r2 = 1 - (np.sum((predicted - actual) ** 2) / np.sum((actual - np.mean(actual)) ** 2)) # R平方

print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"R-Squared: {r2}")

# 預測結果與實際結果的折線圖
plt.figure(figsize=(20, 8))
plt.plot(range(len(actual)), actual, color='blue', label='Actual Values', marker='o')
plt.plot(range(len(predicted)), predicted, color='red', label='Predicted Values', marker='x')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.show()

# 最後50筆資料的折線圖
plt.figure(figsize=(20, 8))
plt.plot(range(len(actual[-50:])), actual[-50:], color='blue', label='Actual Values (Last 50)', marker='o')
plt.plot(range(len(predicted[-50:])), predicted[-50:], color='red', label='Predicted Values (Last 50)', marker='x')
plt.title('Actual vs Predicted Values (Last 50 Data Points)')
plt.legend()
plt.show()
```

結語

本次作業讓我整合過去與本堂課期中專案產出的 Code，
雖然我對深度學習的認知與 Pytorch 有比較熟練，
分類模型最高準確率有 49.72%(過去是 49.55%)，

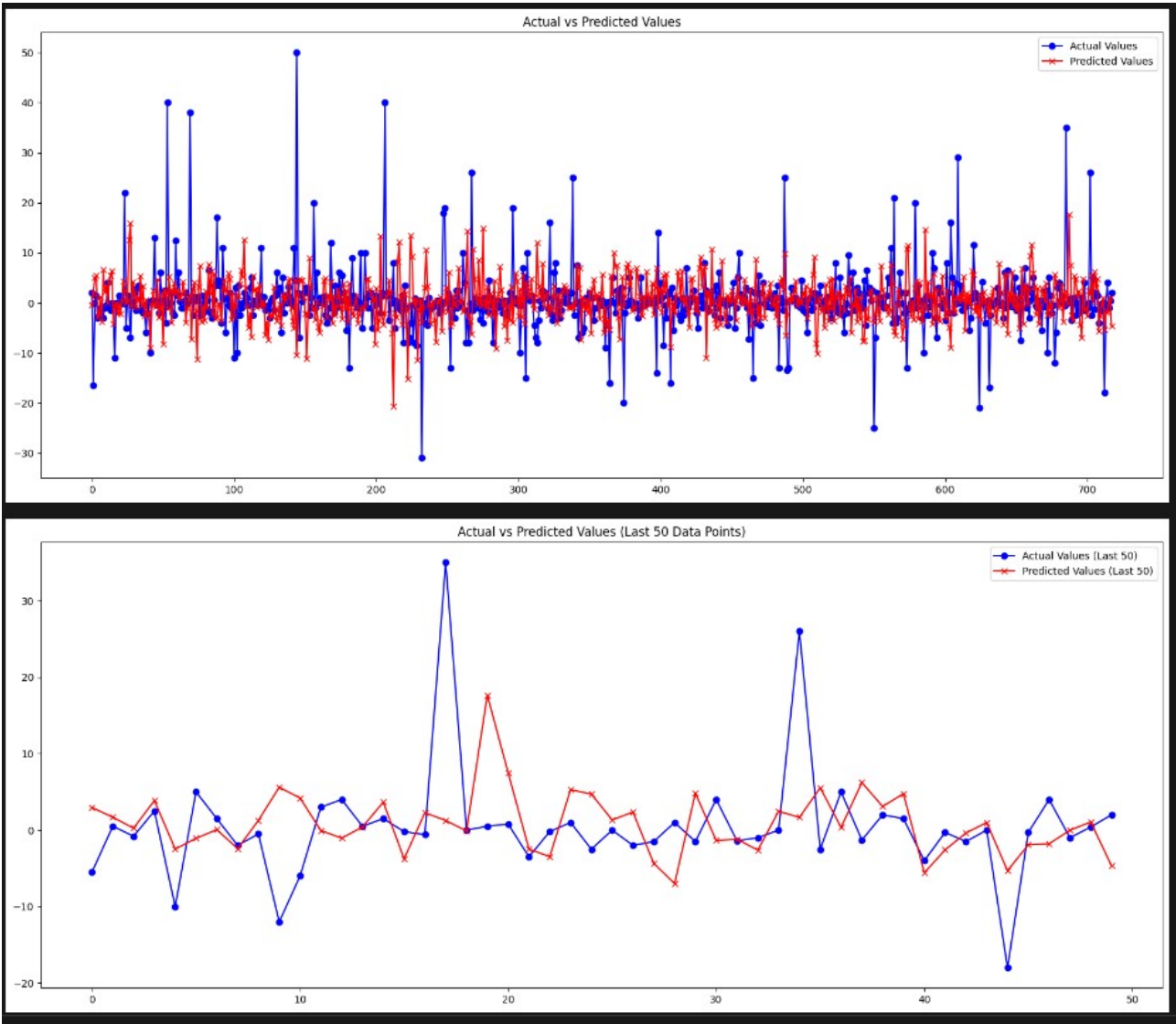
![最高模型準確率](./img/d5fca5c9-886b-4760-a820-fb64f36d3aab.jpeg)

但似乎預測股票不能僅看這些股票的基本資訊，
可能還需要看新聞正負面評價、大盤趨勢、國際事件...等，
將它們量化成可參考的訓練資料，
說不定能提升模型的準確率。

這次模型訓練過程中，
我調整了許多的參數，
發現 Batch Size 對於模型訓練的影響很大，
Batch Size 小，模型訓練越慢、更新頻率高，訓練時間長，模型較不易發生欠擬合，
Batch Size 大，模型訓練越快、更新頻率低、損失平均化、需長序列處理...等，導致欠擬合發生。

預測測試資料圖

回歸



分類

