

物件導向期末專題報告

第8組

主題：加密貨幣行情預測系統

我們之所以會選擇這個主題，是因為我們覺得這個主題可以讓我們學習到很多新的東西，並且可以讓我們在未來的工作上有所幫助。

介紹

這是使用pytorch實現的基於 GRU和MLP 的加密貨幣行情預測系統，使用的數據透過CCXT套件從Binance交易所獲取，並使用pandas套件進行數據處理，使用politly套件進行數據視覺化，使用Streamlit套件進行網頁化呈現。

如果你想要使用這個專案，請先安裝 python3.10 ，然後使用 pip 安裝 requirements.txt 中的套件，最後使用 streamlit run app.py 來啟動網頁。或是使用 docker 來啟動網頁，可以去pull這份 [dockerimage](#) 然後使用 docker run -p 8501:8501 hibana2077/oop_pj 來啟動網頁。

或是可以直接使用我們的網頁 [link](#)

程式碼講解

爬取數據

```
# Download data

binance = binance()
symbol = 'ETH/USDT'
timeframe = '15m'
file_name = f"../data/{symbol.replace('/', '_')}_{timeframe}.csv"
start = binance.parse8601('2022-01-01T00:00:00Z')
end = binance.parse8601('2023-01-01T00:00:00Z')
cnt_time = start
data = []
while cnt_time < end:
    ohlcv = binance.fetch_ohlcv(symbol, timeframe, cnt_time)
    data += ohlcv
    cnt_time = ohlcv[-1][0] + 900000
df = pd.DataFrame(data, columns=['time', 'open', 'high', 'low', 'close', 'volume'])
df['time'] = pd.to_datetime(df['time'], unit='ms')
df.to_csv(file_name, index=False)
```

數據處理

```
df['RSI'] = abstract.RSI(df, timeperiod=14)
df['MACD'] = abstract.MACD(df, fastperiod=12, slowperiod=26, signalperiod=9)['macd'] #只取MACD
df['OBV'] = abstract.OBV(df, timeperiod=14)
df['CCI'] = abstract.CCI(df, timeperiod=14)
df['ATR'] = abstract.ATR(df, timeperiod=14)
df['ADX'] = abstract.ADX(df, timeperiod=14)
df['MFI'] = abstract.MFI(df, timeperiod=14)
df['CLOSE_percent'] = df['close'].pct_change()
```

Python

```
df['UP'] = df['CLOSE_percent'].apply(lambda x: 1 if x > 0 else 0)
df['DOWN'] = df['CLOSE_percent'].apply(lambda x: 1 if x < 0 else 0)
df['UP'] = df['UP'].shift(-1) #shift UP-DOWN 一個單位，因為我們要預測的是下一個時間點的漲跌
df['DOWN'] = df['DOWN'].shift(-1)

df = df.dropna()
```

Python

正規化

#正規化

```
df['RSI'] = (df['RSI'] - df['RSI'].min()) / (df['RSI'].max() - df['RSI'].min())
df['MACD'] = (df['MACD'] - df['MACD'].min()) / (df['MACD'].max() - df['MACD'].min())
df['OBV'] = (df['OBV'] - df['OBV'].min()) / (df['OBV'].max() - df['OBV'].min())
df['CCI'] = (df['CCI'] - df['CCI'].min()) / (df['CCI'].max() - df['CCI'].min())
df['ATR'] = (df['ATR'] - df['ATR'].min()) / (df['ATR'].max() - df['ATR'].min())
df['ADX'] = (df['ADX'] - df['ADX'].min()) / (df['ADX'].max() - df['ADX'].min())
df['MFI'] = (df['MFI'] - df['MFI'].min()) / (df['MFI'].max() - df['MFI'].min())
df['open'] = (df['open'] - df['open'].min()) / (df['open'].max() - df['open'].min())
df['high'] = (df['high'] - df['high'].min()) / (df['high'].max() - df['high'].min())
df['low'] = (df['low'] - df['low'].min()) / (df['low'].max() - df['low'].min())
df['close'] = (df['close'] - df['close'].min()) / (df['close'].max() - df['close'].min())
df['volume'] = (df['volume'] - df['volume'].min()) / (df['volume'].max() - df['volume'].min())
```

分割數據



```
X,y = list(),list()
```

```
X = df.iloc[:,1:-2].values
```


```
y = df.iloc[:,-2:].values
```

建立資料集類別

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.75, random_state=42)
```

```
print(f"X_train shape: {X_train.shape} , y_train shape: {y_train.shape}")
```

```
X_train shape: torch.Size([21279, 12]) , y_train shape: torch.Size([21279, 2])
```

```
class TrainDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y
    def __len__(self):
        return len(self.X)
    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

class ValDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y
    def __len__(self):
        return len(self.X)
    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

class TestDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y
    def __len__(self):
        return len(self.X)
    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```



```
train_dataset = TrainDataset(X_train, y_train)
val_dataset = ValDataset(X_val, y_val)
test_dataset = TestDataset(X_test, y_test)
```

建立模型

SelectItem



```
class SelectItem(torch.nn.Module):  
    def __init__(self, item_index):  
        super(SelectItem, self).__init__()  
        self._name = 'selectitem'  
        self.item_index = item_index  
  
    def forward(self, inputs):  
        return inputs[self.item_index]
```

Ver1 (MLP+Dropout)

- 基底為MLP
- 使用Dropout避免過擬合

```
class CCV1(nn.Module):#MLP
    def __init__(self):
        super(CCV1, self).__init__()
        self.Linear1 = nn.Linear(12, 128)
        self.Linear2 = nn.Linear(128, 256)
        self.Linear3 = nn.Linear(256, 1024)
        self.Linear4 = nn.Linear(1024, 256)
        self.Linear5 = nn.Linear(256, 128)
        self.Linear6 = nn.Linear(128, 64)
        self.Linear7 = nn.Linear(64, 2)
        self.Dropout1 = nn.Dropout(0.3)

    def forward(self, x):
        x = F.relu(self.Linear1(x))
        x = F.relu(self.Linear2(x))
        x = self.Dropout1(x)
        x = F.relu(self.Linear3(x))
        x = F.relu(self.Linear4(x))
        x = self.Dropout1(x)
        x = F.relu(self.Linear5(x))
        x = F.relu(self.Linear6(x))
        x = self.Dropout1(x)
        x = self.Linear7(x)
        return x
```

Ver3 (MLP+GRU)

- 基底為MLP
- 使用Dropout避免過擬合
- 使用GRU來加強過去的資訊

```
class CCV3(nn.Module):#MLP+GRU
    def __init__(self):
        super(CC3, self).__init__()
        self.Linear1 = nn.Linear(12, 128)
        self.Linear2 = nn.Linear(128, 256)
        self.Linear3 = nn.Linear(256, 2)
        self.Dropout1 = nn.Dropout(0.168)
        self.GRU1 = nn.GRU(256, 256, 1, batch_first=True)

    def forward(self, x):
        x = F.relu(self.Linear1(x))
        x = F.relu(self.Linear2(x))
        x = self.Dropout1(x)
        x, _ = self.GRU1(x)
        x = self.Linear3(x)
        return x
```

Ver5 (DUE channel MLP+Dropout)

- 基底為MLP
- 在第二層分成兩個通道
- 一個通道使用較多的Dropout
- 另一個通道使用較少的Dropout
- 並且在最後二層的輸出上使用 `torch.cat` 來合併

```
class CCV5(nn.Module):#DUE channel MLP
    def __init__(self):
        super(CCV5, self).__init__()
        self.pub1 = nn.Linear(12,32)
        self.allayer1 = nn.Linear(32, 256)
        self.allayer2 = nn.Linear(256, 512)
        self.allayer3 = nn.Linear(512, 256)
        self.allayer4 = nn.Linear(256, 64)#a1 end
        self.a2layer1 = nn.Linear(32, 256)
        self.a2layer2 = nn.Linear(256, 556)
        self.a2layer3 = nn.Linear(556, 256)
        self.a2layer4 = nn.Linear(256, 192)#a2 end
        self.concat = nn.Linear(256, 32)
        self.a3layer1 = nn.Linear(32, 16)
        self.a3layer2 = nn.Linear(16, 2)
        self.dropout = nn.Dropout(0.3)

    def forward(self, x):
        x = F.relu(self.pub1(x))
        a1 = F.relu(self.allayer1(x))
        a1 = F.relu(self.allayer2(a1))
        a1 = self.dropout(a1)
        a1 = F.relu(self.allayer3(a1))
        a1 = self.dropout(a1)
        a1 = F.relu(self.allayer4(a1))
        a2 = F.relu(self.a2layer1(x))
        a2 = F.relu(self.a2layer2(a2))
        a2 = self.dropout(a2)
        a2 = F.relu(self.a2layer3(a2))
        a2 = F.relu(self.a2layer4(a2))
        x = torch.cat((a1, a2), 1)
        x = F.relu(self.concat(x))
        x = F.relu(self.a3layer1(x))
        x = self.a3layer2(x)
        return x
```

訓練模型(函數)

```
def train(model, train_loader, optimizer, criterion, epoch, device):
    if epoch == 0: print("Training...")
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx == 0:
            print(f"data shape: {data.shape} , target shape: {target.shape} , output shape: {output.shape}")
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{}] ({:.0f}%) \t Loss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

測試模型(函數)


```
def test(model, test_loader, criterion, device):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:

            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += criterion(output, target).item() # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-probability
            correct += pred.eq(target.argmax(1, keepdim=True).view_as(pred)).sum().item() #

    test_loss /= len(test_loader.dataset)

    print('Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
    return (100. * correct / len(test_loader.dataset)), test_loss
```

訓練模型(model - ver1)



```
model = CCV1().to(DEVICE)
optimizer = optim.SGD(model.parameters(), lr=0.05,momentum=0.5)
criterion = F.cross_entropy

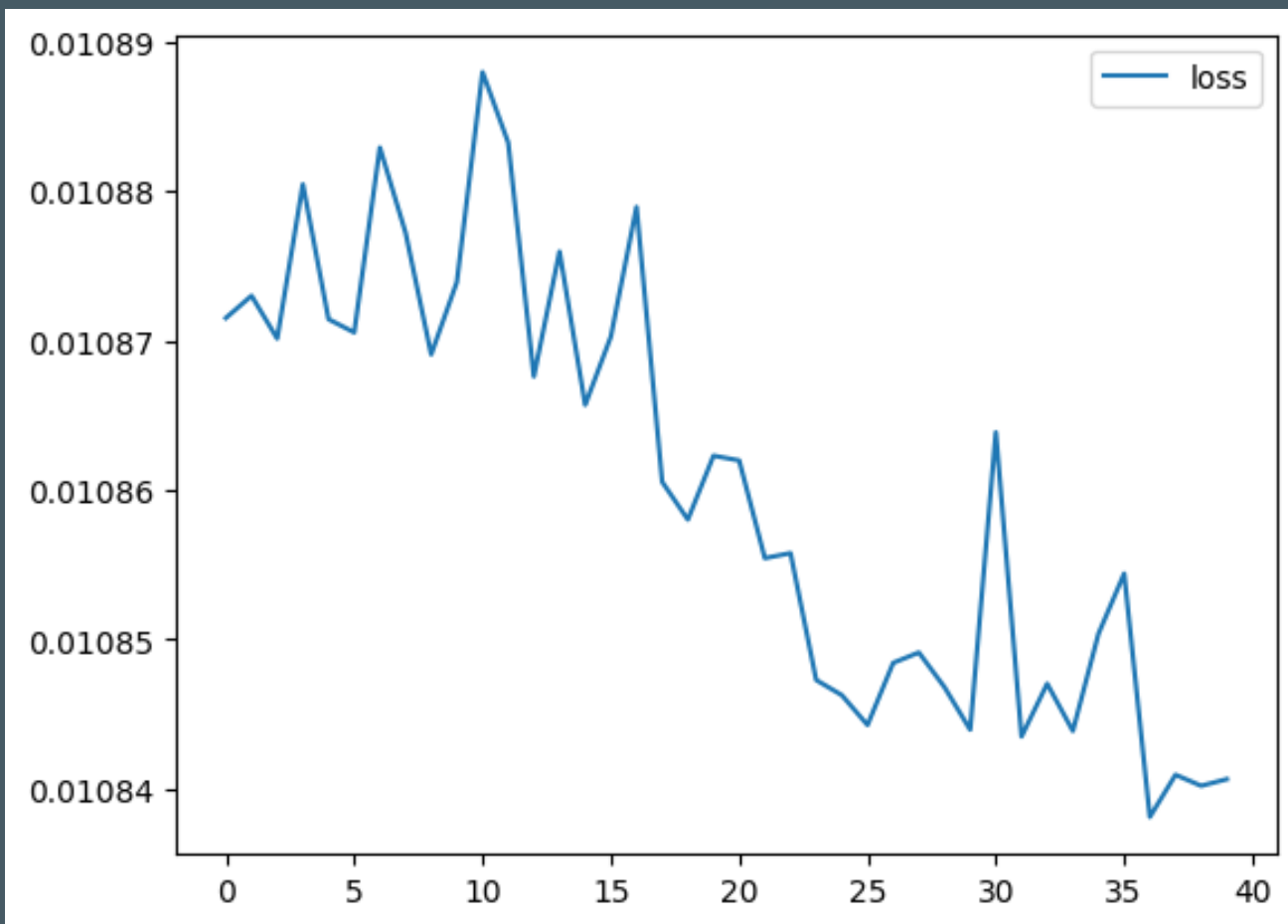
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=0)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False, num_workers=0)

test_acc_list = []
test_loss_list = []

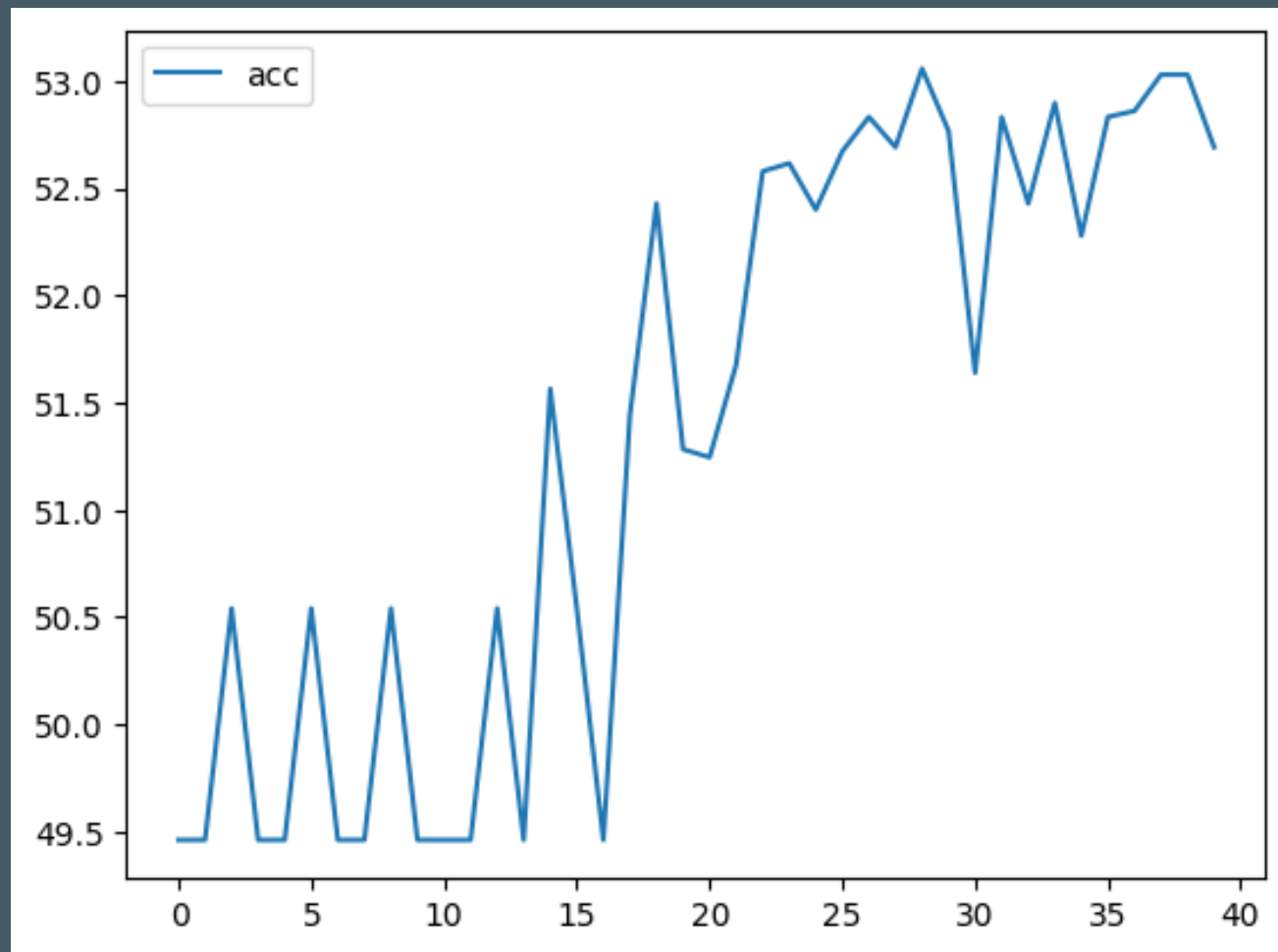
for epoch in range(1, EPOCHS + 1):
    train(model, train_loader, optimizer, criterion, epoch, DEVICE)
    test_acc, test_loss = test(model, test_loader, criterion, DEVICE)
    test_acc_list.append(test_acc)
    test_loss_list.append(test_loss)
```


訓練結果(model - ver1)

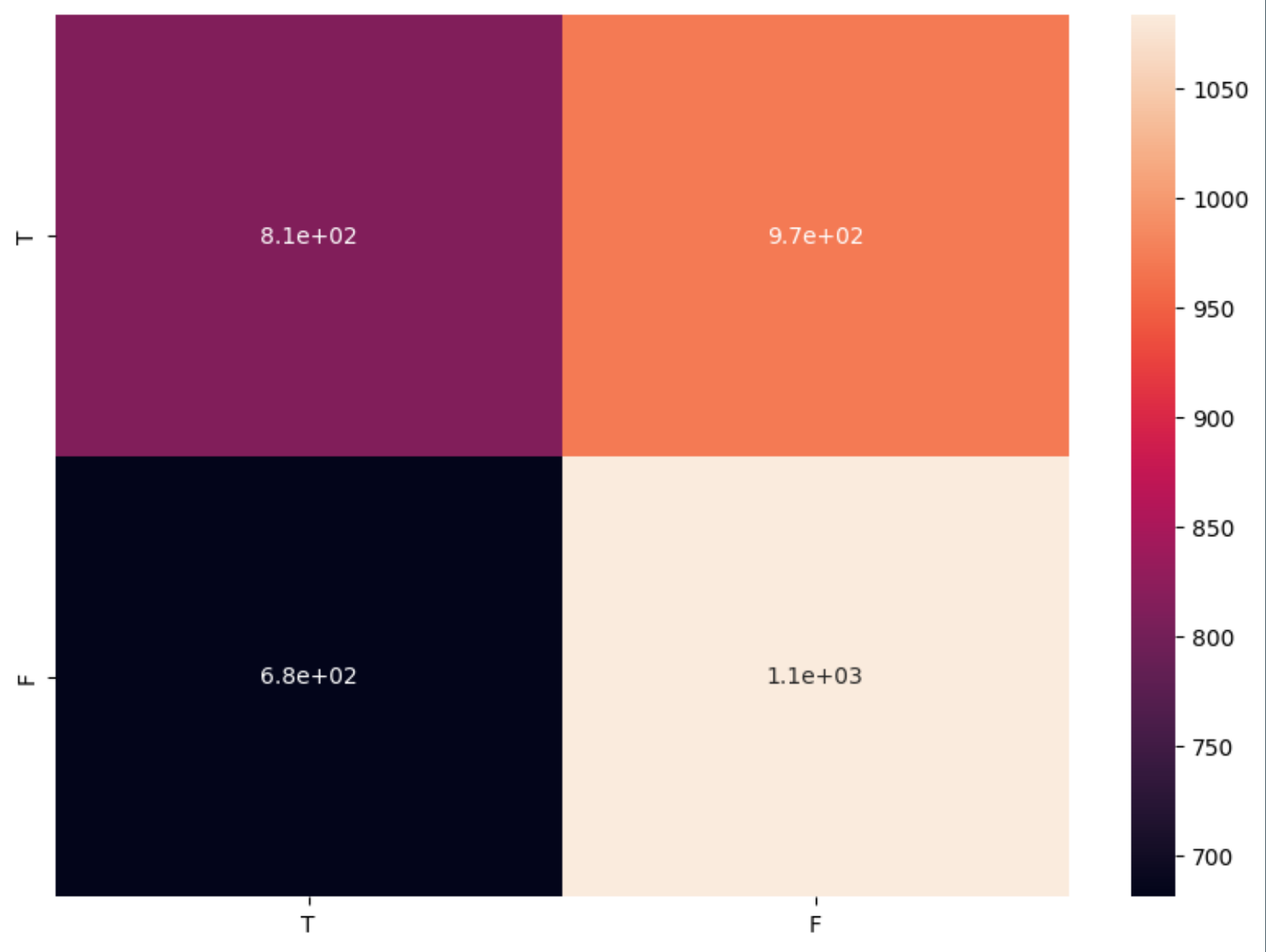
Loss



Accuracy



confusion
matrix



訓練模型(model - ver3)

```
model3 = CCV3().to(DEVICE)
optimizer = optim.SGD(model3.parameters(), lr=0.05, momentum=0.5)
criterion = F.cross_entropy

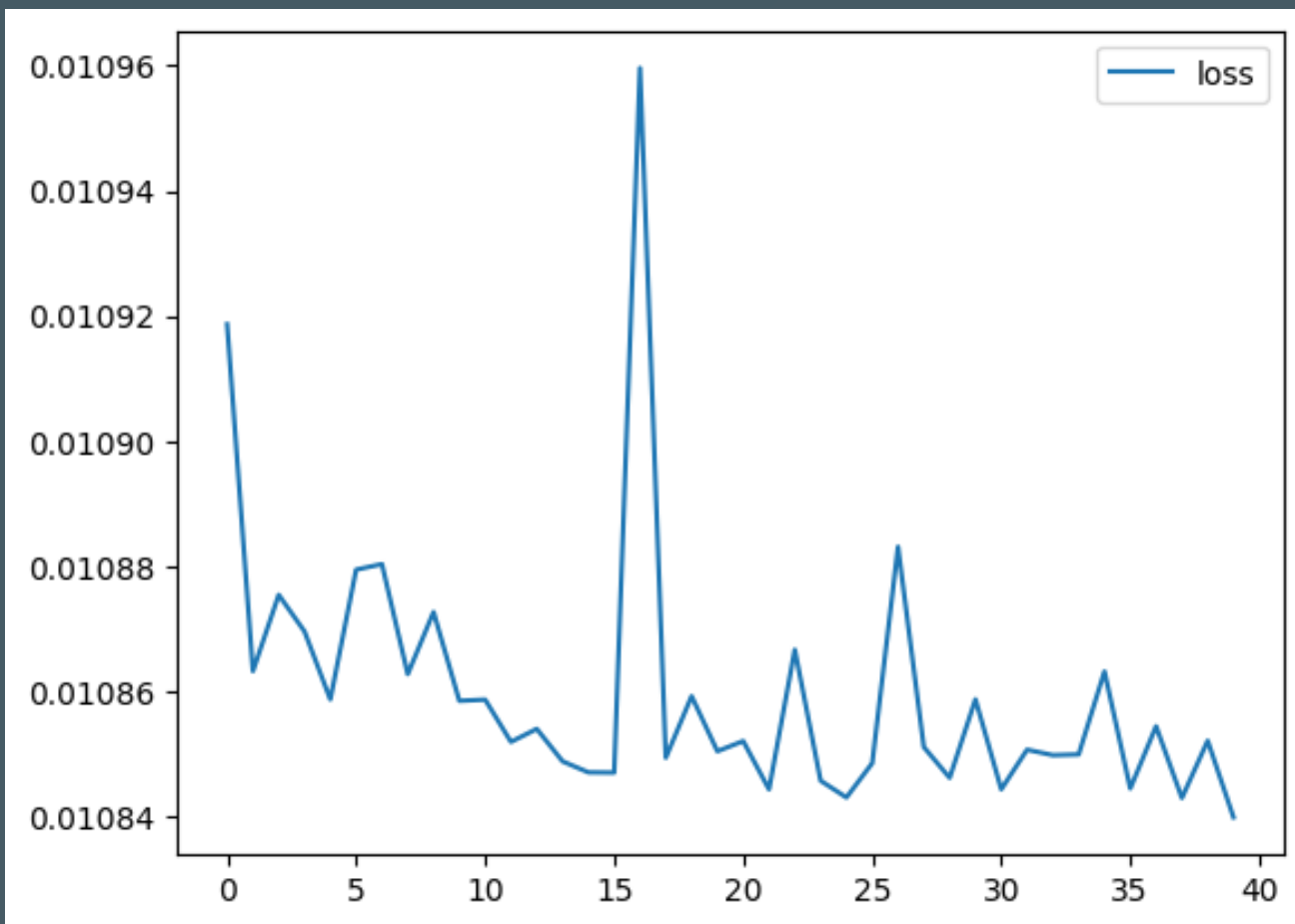
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=0)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False, num_workers=0)

test_acc_list = []
test_loss_list = []

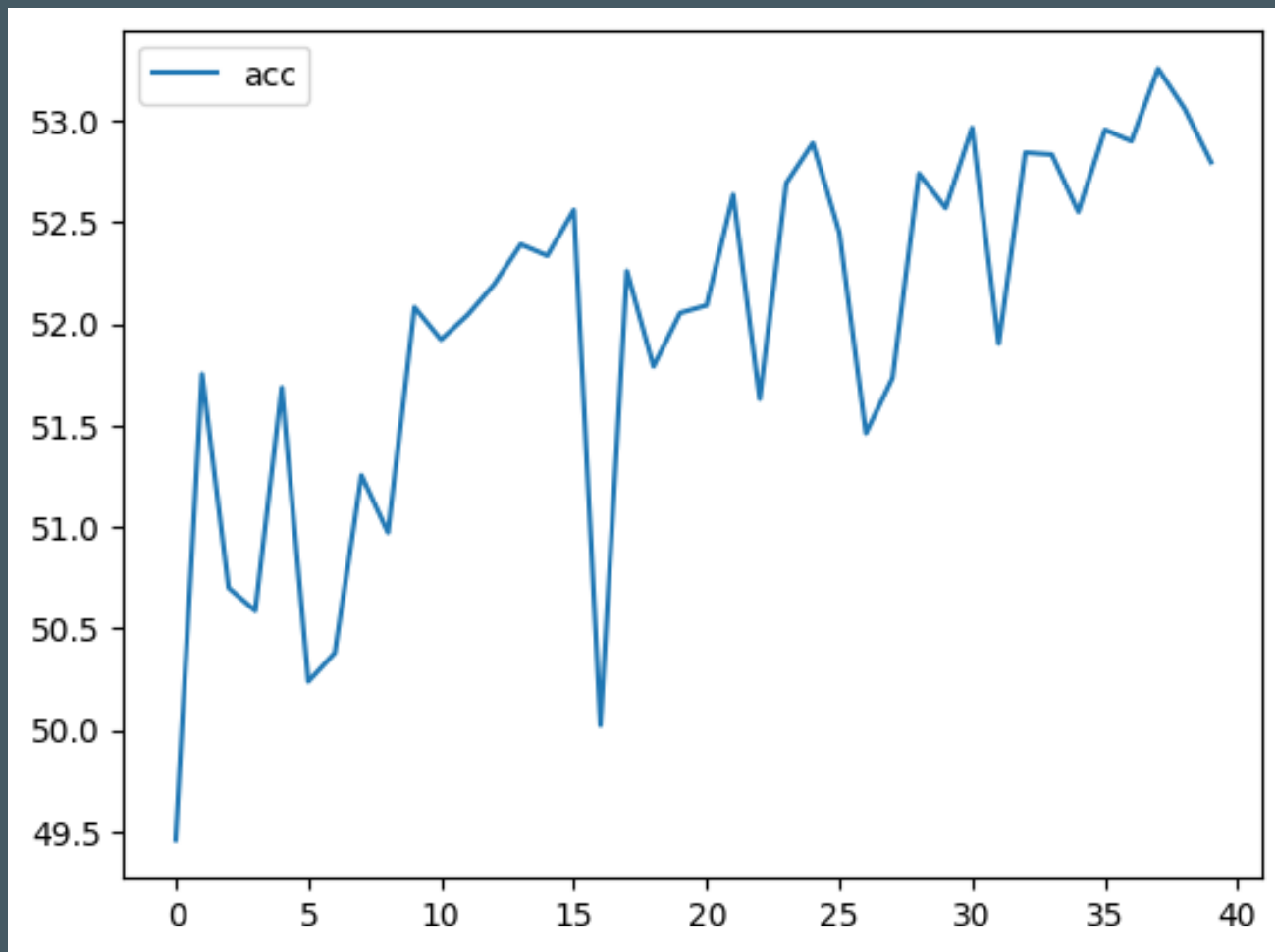
for epoch in range(1, EPOCHS + 1):
    train(model3, train_loader, optimizer, criterion, epoch, DEVICE)
    test_acc, test_loss = test(model3, test_loader, criterion, DEVICE)
    test_acc_list.append(test_acc)
    test_loss_list.append(test_loss)
```

訓練結果(model - ver3)

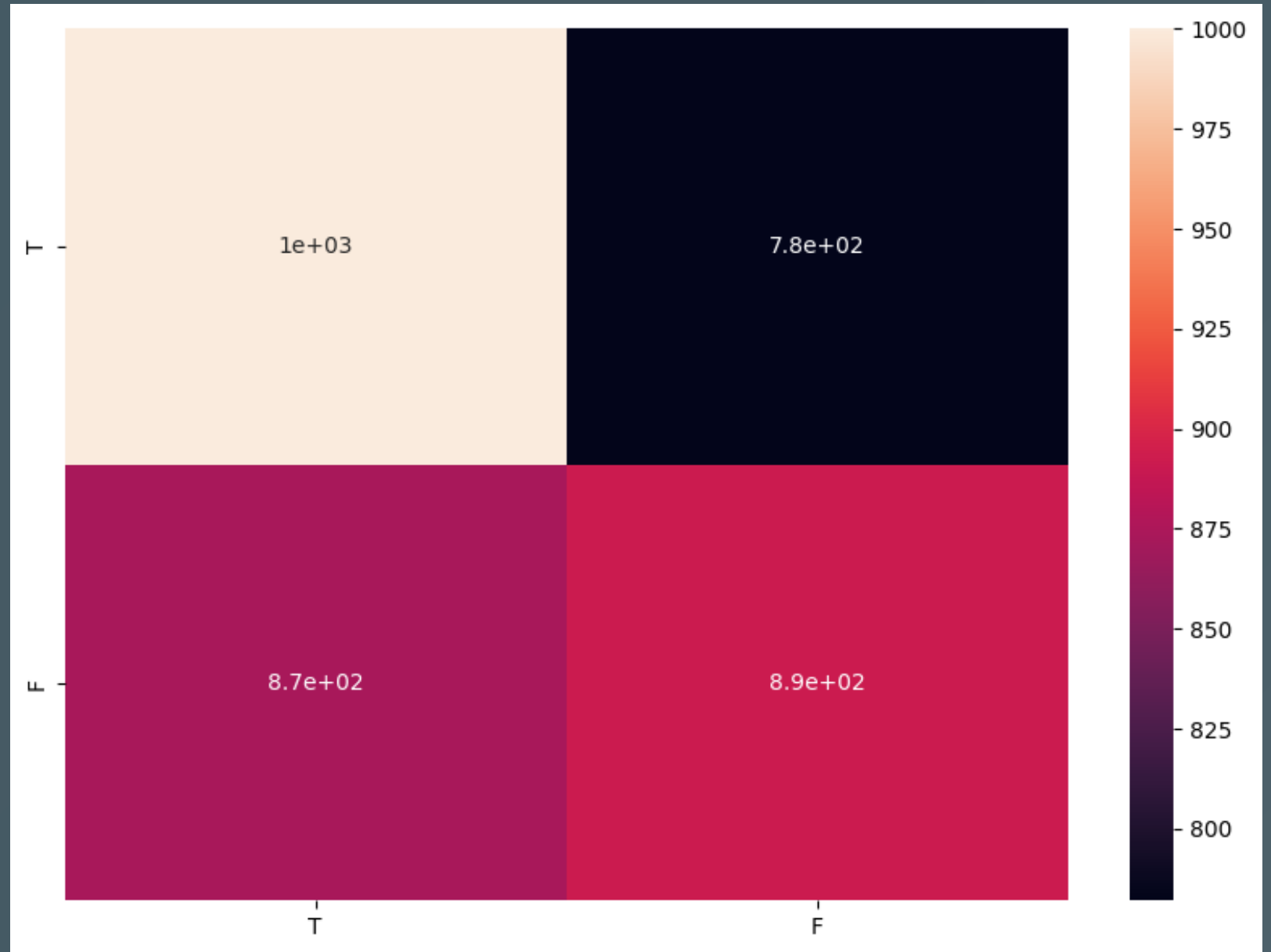
Loss




Accuracy



confusion matrix



訓練模型(model - ver5)



```
model5 = CCV5().to(DEVICE)
optimizer = optim.SGD(model5.parameters(), lr=0.05, momentum=0.5)
criterion = F.cross_entropy

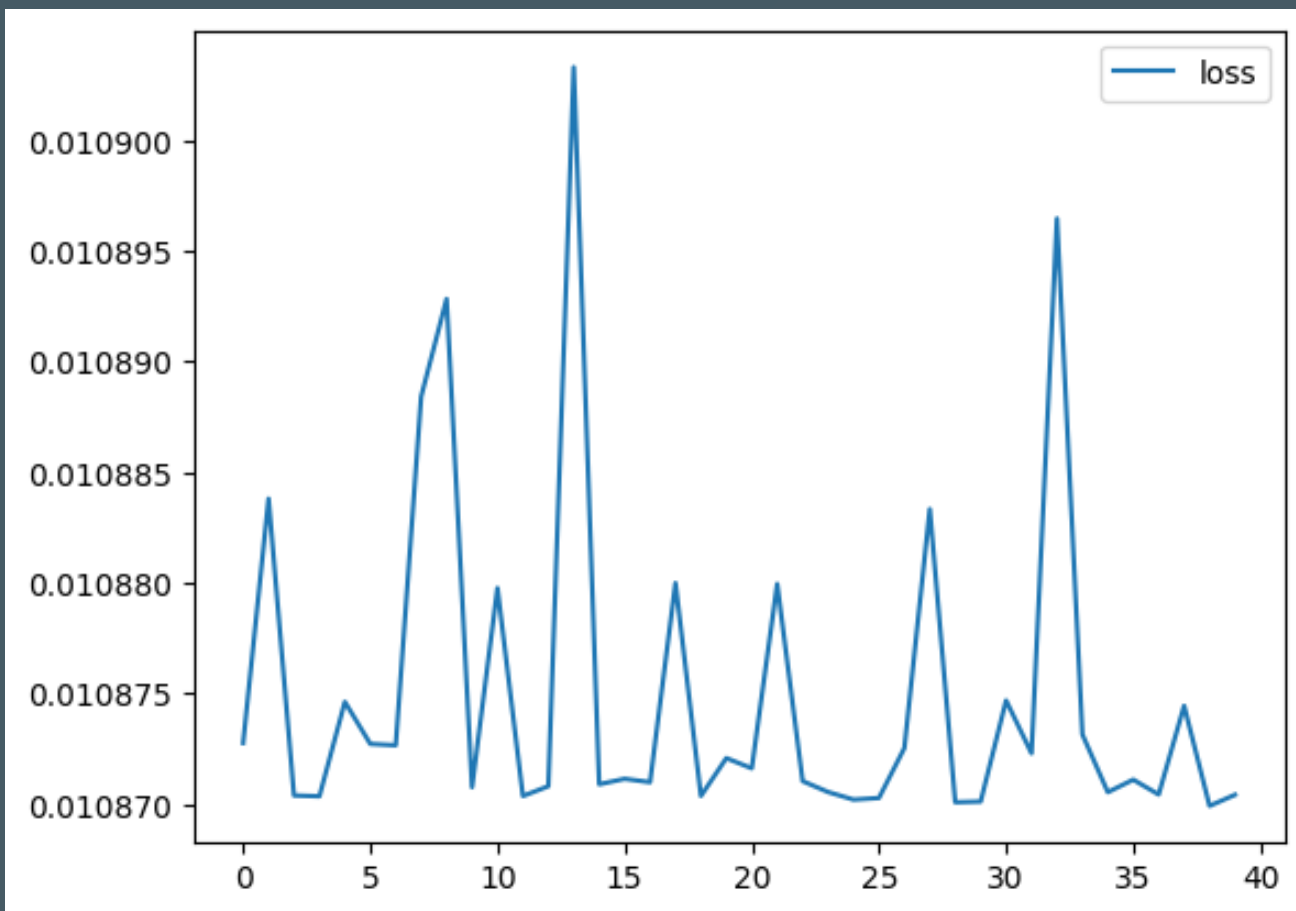
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=0)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False, num_workers=0)

test_acc_list = []
test_loss_list = []

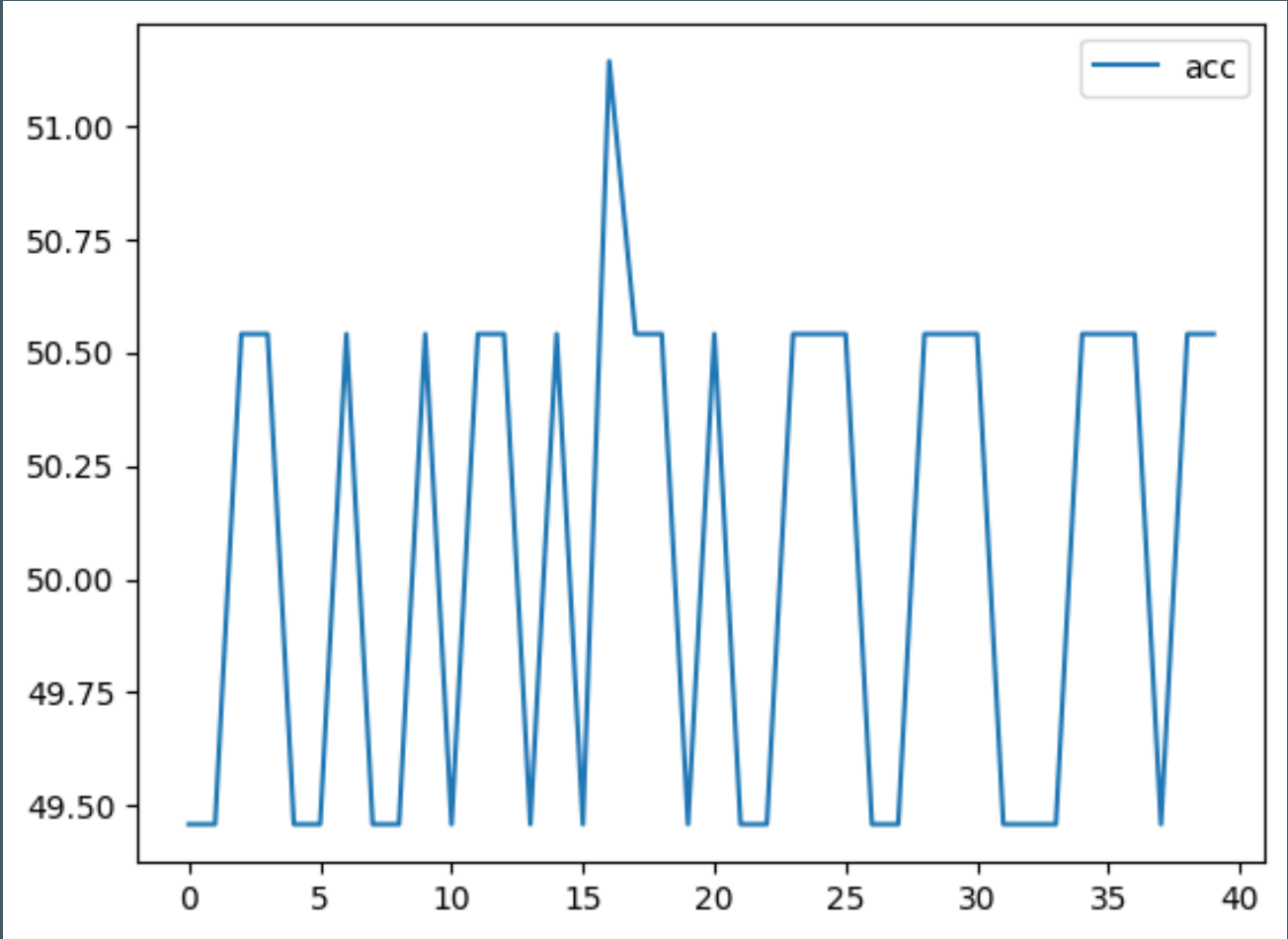
for epoch in range(1, EPOCHS + 1):
    train(model5, train_loader, optimizer, criterion, epoch, DEVICE)
    test_acc, test_loss = test(model5, test_loader, criterion, DEVICE)
    test_acc_list.append(test_acc)
    test_loss_list.append(test_loss)
```


訓練結果(model - ver5)

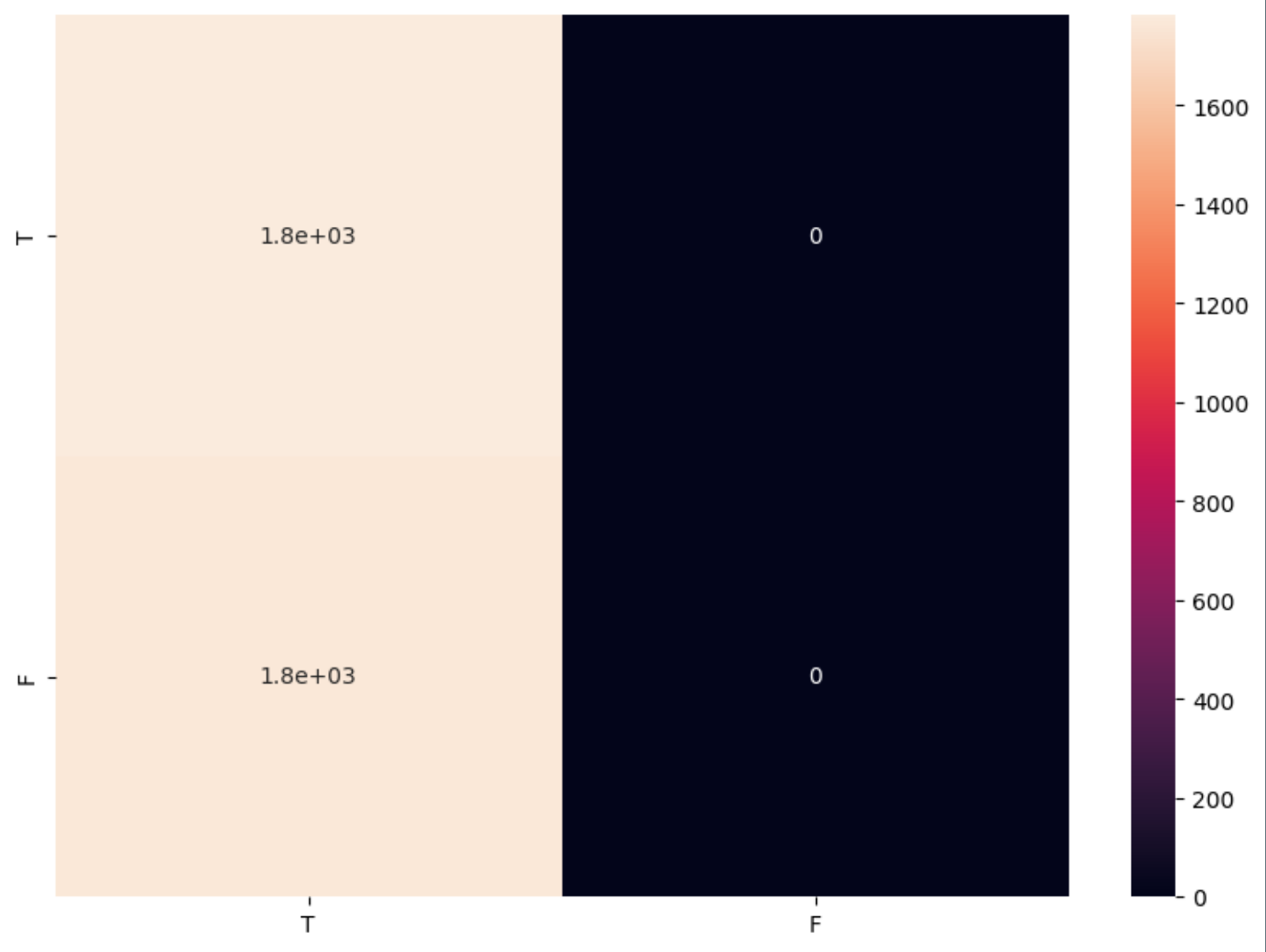
Loss



Accuracy



**confusion
matrix**



結論

透過以上實驗，我們可以發現，要預測加密貨幣價格走勢是一件不太容易的事情，因為加密貨幣的價格走勢是一個非常不穩定的東西，而且價格走勢的變化也是非常快速的，因此我們在訓練模型的時候，需要將資料集的時間間隔設定得越短越好，這樣才能讓模型更好的去預測價格走勢，我們也發現，模型的深度太大的話資料必須要增加，才能提升模型的準確度。

部屬模型

模型訓練完成後，我們就可以將模型部屬到網站上，網站透過streamlit框架完成，並且使用firebase平台部屬，網站的功能是可以輸入加密貨幣的名稱，然後輸入想要預測的時間，就可以預測該加密貨幣的價格走勢。

網站介紹

可以掃描下方的QRcode，或是點擊旁邊的網址，就可以進入 [網站](#)。



使用方式

- 選擇 **模型使用** 頁面
- 選擇想要預測的加密貨幣
- 輸入想要預測的時間
- 點擊 **預測** 按鈕



預測結果

開始預測...

下載數據中...

K線圖



預測結果

會上漲 

參考資料

- [1] <https://www.kaggle.com/>
- [2] [PyTorch 中文手冊\(pytorch handbook\)](#).
- [3] [PyTorch 官方文件](#)
- [4] [LSTM-Classification-pytorch](#)
- [5] [CCXT 官方文件](#)

謝謝大家

powered by [marp](#)