CE6023 Homework1 Report

學號:110201532　　姓名：范植緯

1.(5%) 具體說明你的資料前處理方式，並說明造成了的差異以及造成差異可能原因。

這份作業我在資料前的處理方式除了使用範例程式碼那樣將4天的資料合併成一個row外，我還設有了這些資料的次方項，因為原本的範例程式碼只為有那32項的一次方，而我處理的方式是將train_x的資料在創建時多乘了power項來放新的項次的空格，並在座堆疊時，利用for迴圈根據power的值來順便把項的nth_term也算出來並併入training data中，具體程式碼如下

```
train_x = np.empty([train_size, feature_size * input_date_data_size*power], dtype = float)#乘power讓資料集擴展到次方項
train_y = np.empty([train_size, feature_size], dtype = float)

for idx in range(train_size):
  temp_data = np.array([])
  for count in range(input_date_data_size):
    temp_data = np.hstack([temp_data, train[idx + count]])#把前四天的資料合併
    for nth_term in range(2,power+1):
        temp_data = np.hstack([temp_data, train[idx + count]**nth_term])#把前四天的資料合併
  train_x[idx, :] = temp_data
  train_y[idx, :] = train[idx + input_date_data_size]#應該要預測到的實際第五天資料

# y值只留下現鈔買入
filtered_columns = [train_df.columns.get_loc(col) for col in train_df.columns if '現鈔買入' in col]
train_y = train_y[:, filtered_columns]
```

造成的差異是在我輸入power值的時候，能夠產生相比於一次項還要更好的預測圖形，更能符合我們的期待，然而當power數設太高時卻反而導致我們的預測率下降，我想這跟overfitting有關，由於太過要求符合training_data而讓validation data反而下降。

2.(5%) 使用四種不同的 Learning Rate 進行 Training（方法參數需一致），作圖並討論其收斂過程（橫軸為 Iteration 次數，縱軸為 Loss 的大小，四種 Learning Rate 的收斂線請以不同顏色呈現在一張圖裡做比較）。

由下圖我們可以看出，在同樣的週期下，如果learning_rate越小，最終會收斂到更小的Loss，得到更好的module，但也會因為learning_rate太小，導致要到收斂的狀態必須經過很多次的周期才可以到達。

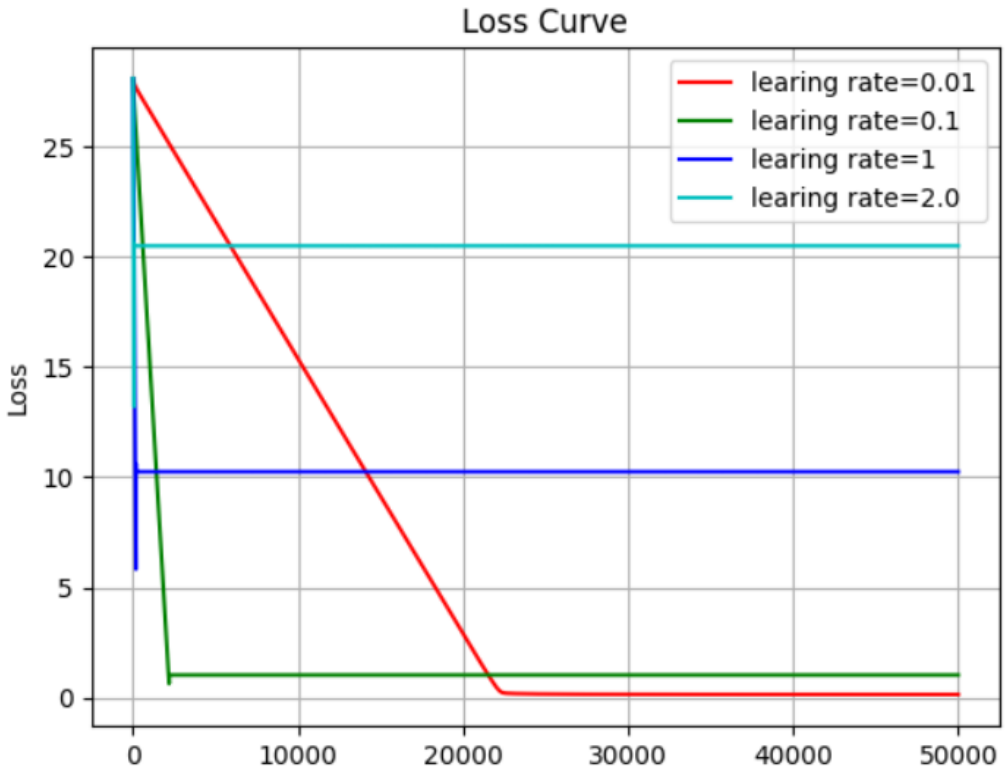
會得到這樣的結果其原因是在做gradient_decent時，會因為給的learning_rate來決定下一步是要走的多遠，learning_rate越大走得越大步，也就是因為這樣，當我們要逼近一個極小值時，learning_rate越小我們越可接近極小值，反而如果lerning_rate越大，他會在極小值之前就開始反覆恆跳，始終到達不了更低的loss，因此得到這樣的結果。程式碼如下

```
epochs = 50000
train_loss_history=[]
for lr in learning_rates:
    model = LinearRegression(feature_size * input_date_data_size*power, 8)
    train_loss_data_history = []
    optimizer = torch.optim.SGD(model.parameters(), lr=lr)
    for epoch in range(epochs):
        model.train()
        # forward pass and loss
        train_y_predicted = model(train_x)
        loss = criterion(train_y_predicted, train_y)
        train_loss_data_history.append(loss.item())
        # backward pass
        loss.backward()
        # update
        optimizer.step()
        # init optimizer
        optimizer.zero_grad()

        # 验证资料集
        model.eval()
        val_y_predicted = model(val_x)
        val_loss = criterion(val_y_predicted, val_y)
        val_loss_history.append(val_loss.item())
        if (epoch + 1) % 10 == 0:
            print(f'epoch: {epoch+1}, train_loss = {loss.item(): .4f}, val_loss = {val_loss.item(): .4f}')
    train_loss_history.append(train_loss_data_history)
```

```
epochs_range = range(1, epochs + 1)
colors = ['r', 'g', 'b', 'c']

for i in range(4):
    plt.plot(epochs_range, train_loss_history[i], colors[i], label='learing rate='+str(learning_rates[i]))
plt.title('Loss curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```



3. (5%) 比較取前 2 天和前 4 天的資料的情況下，於 Validation data 上預測的結果，並說明造成的可能原因。

以下為我取兩天時，跑10000次iteration後所得出的loss

```
epoch: 10000, train_loss =  0.2548, val_loss =  0.2503
```

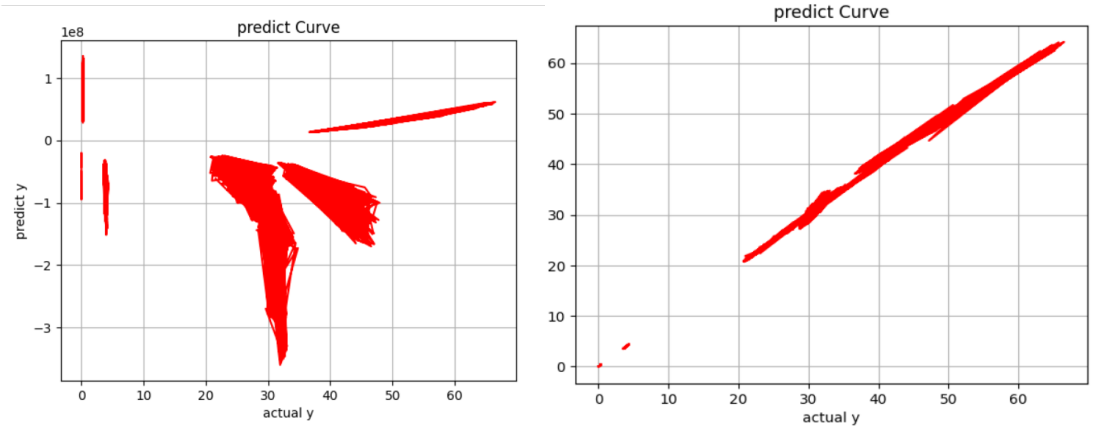
另外一張圖是取四天時，跑10000次iteration後所得出的loss

```
epoch: 10000, train_loss =  0.5150, val_loss =  0.4968
```

若以val_loss來評估兩個model的性能時，我發現取前兩天得出的結果較好，我認為可能的原因在於時間的相關性，相對於四天來說，我們使用兩天的數據更好捕捉市場的快速變化，此外，隨著數據量的增加，四天的數據變得更為複雜，導致更難將模型收斂，反而兩天的數據集更容易實現整體的訓練收斂。

4. (5%) 比較資料在有無 Normalization 或 Standardization的情況下，於 Validation data 上預測的結果，並說明造成的可能原因。

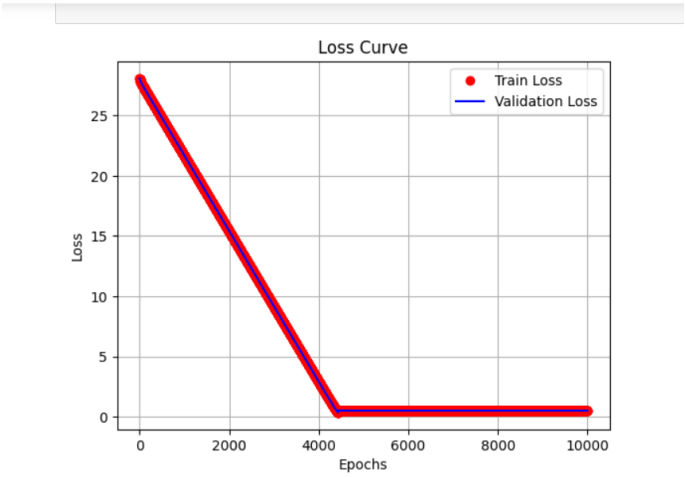
得到無normal和Standardization的圖如左下，右下則為有normal和Standardization的圖



我認為會造成這樣的原因主要是因為在無 Normalization 或 Standardization的情況時，我們的Xi會因為有平方、次方項而導致數值變很大，相比之下xi的一次項比起其他次項就會顯得較無影響力，這樣將會導致預測的數值會特別側重於多次方項的係數，因而導致預測不準，然而，如果我們有用Normalization和Standardization，將資料都縮放到0~1之間的範圍內，使得feature的權重差異變得不敏感，每個feature的影響都差不多，就較能預測出我們正確的值。

5. (5%) 請作圖並說明你的Training Loss 和 Validation Loss 間的關係，並說明可能造成的原因。

我的Training Loss 和 Validation Loss 的關係如下圖，基本上當Training Loss下降時，我的Validation Loss也會一起下降，此時是因為模型正在有效學習數據，而到後面Training Loss平穩時Validation Loss也跟著平穩，此時代表已經收斂了，而如果Training Loss在下降而Validation Loss上升則代表已經overfittin了，可能是因為複雜度過高，在我把次方項設很高時就很常發生這個原因，因為太過於要fit Training Loss而讓實際數據上性能下降



6. (5%) 請說明你超越 Baseline 的 Model（最後選擇在 Kaggle 上提交的）是如何實作的（若你有額外實作其他 Model，也請分享是如何實作的）。

在Baseline的model中，所使用的model只有一次項，我藉由在前面的資料輸入training data時，讓資料集擴充到次方項，

```
train_x = np.empty([train_size, feature_size * input_date_data_size[x]*power], dtype = float)#乘power讓資料集擴展到次方項
train_y = np.empty([train_size, feature_size], dtype = float)
```

並在把資料輸入進去時，順便把feature的平方項等，用for迴圈輸入進去

```
for count in range(input_date_data_size[x]):
  temp_data = np.hstack([temp_data, train[idx + count]])#把前幾天的資料合併
  for nth_term in range(2,power+1):
      temp_data = np.hstack([temp_data, train[idx + count]**nth_term])#把前幾天的資料合併
```

得到這些資料後，這樣就能擴展到次方項了，接下來做feature scaling，分割訓練集還有調整learning rate，再用RMSE來計算loss，並趨近於gradient-decent的minimum，所得出這個model。

** 因為 Testing data 預測結果要上傳 Kaggle 後才能得知，所以在報告中並不要求同學們呈現 Testing data 的結果， **Validation data 部分，請自行從training data中隨機取20%做Validation data**，如果不知道甚麼是Validation data 請參考：https://youtu.be/D_S6y0Jm6dQ?t=1949