Report of Anomaly Detection HW2

Name: 高煒軒

Student ID: 112062646

Implementation & explanation

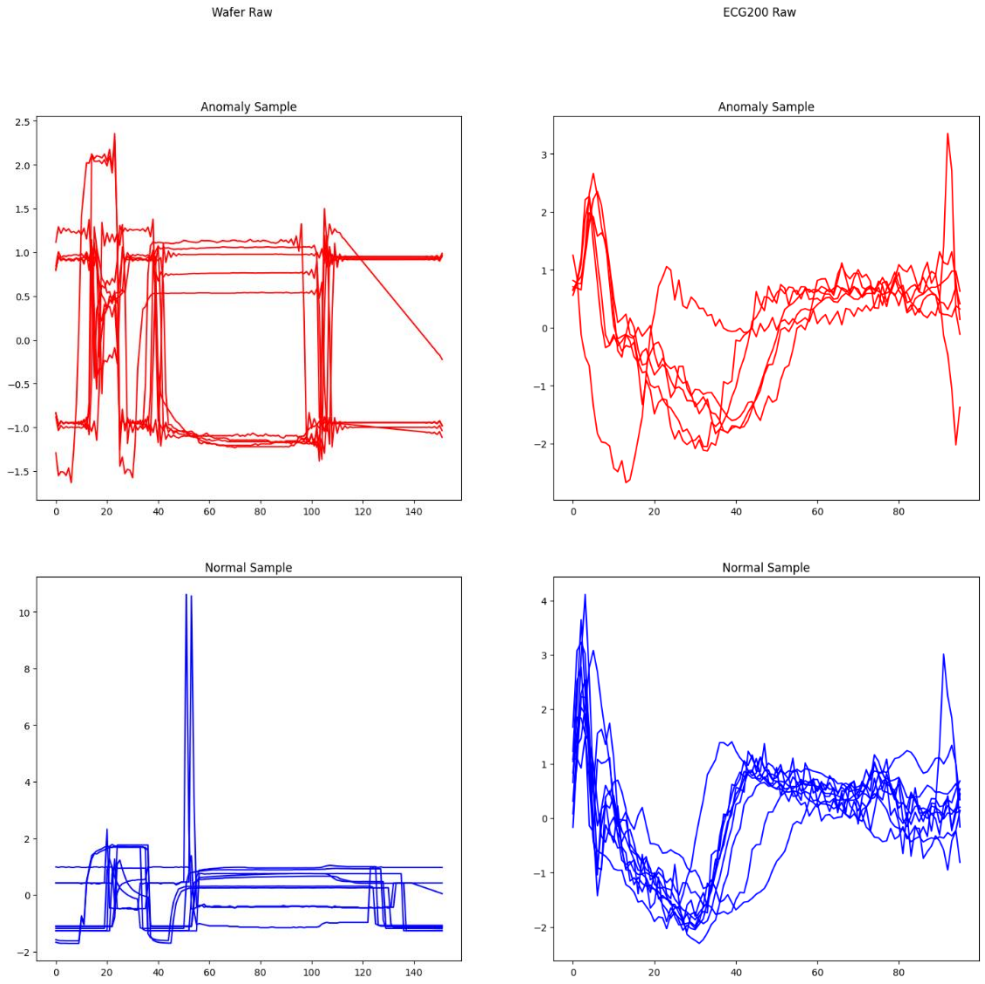
本次作業中的 **5** 個 **Problem implementation code & explanation**，會依據 **5** 個不同的 **Problem** 各自解釋。

(1) Visualization

Function 呼叫為:

```
data_plot(test_data, test_label, category+" Raw")
```

結果如下:



Implementation code

```python
def data_plot(data, label, title):
    normal_data  = data[label==0]
    anomaly_data = data[label==1]

    normal_idx = np.random.choice(normal_data.shape[0], size=10, replace=False)

    # Selecting "ALL" anomaly data, if the number of anomaly data <= 10
    if anomaly_data.shape[0] > 10:
        anomaly_idx = np.random.choice(anomaly_data.shape[0], size=10, replace=False)
    else:
        anomaly_idx = np.arange(anomaly_data.shape[0])


    fig, axs = plt.subplots(2,1, figsize=(8,16))
    x = np.arange(0, anomaly_data.shape[1])
    y = anomaly_data[anomaly_idx]
    for i in range(anomaly_idx.shape[0]):
        axs[0].plot(x, y[i], 'r')
    axs[0].set_title("Anomaly Sample")

    y = normal_data[normal_idx]
    for i in range(10):
        axs[1].plot(x, y[i], 'b')
    axs[1].set_title("Normal Sample")
    fig.suptitle(title)
    plt.show()
```

(2) Raw Data

Function 呼叫順序為:

```
Raw_AD = AD()
Raw_AD(train_data, train_label, test_data, test_label
```

KNN_AD(): based on K-neighbors classification 的 Anomaly Detection algorithm。回傳 roc-auc 值。

AD():

__inti__: 定義特別的 feature processing function e.g. DFT(), DWT()。

__call__: 使用定義好的 function 處理資料、取得 features 後，將處理後的 data 丟入 KNN_AD。

■ Implementation code

```
48  def KNN_AD(train_X, train_y, test_X, test_y, k=5):
49      KNN = KNeighborsClassifier(n_neighbors=k)
50      KNN.fit(train_X, train_y)
51      dist, _ = KNN.kneighbors(test_X)
52      avg_dist = np.mean(dist, axis=1)
53
54      return roc_auc_score(y_true=test_y, y_score=avg_dist)
```

```
115 class AD:
116     def __init__(self, func=None) -> None:
117         self.process_func = func
118
119     def __call__(self, train_X, train_y, test_X, test_y, k=5, arg=None) -> float:
120         if self.process_func == None:
121             train_proc = train_X
122             test_proc  = test_X
123         else:
124             train_proc = self.process_func(train_X, arg)
125             test_proc  = self.process_func(test_X, arg)
126
127         return KNN_AD(train_proc, train_y, test_proc, test_y, k=k)
```

(3) PCA

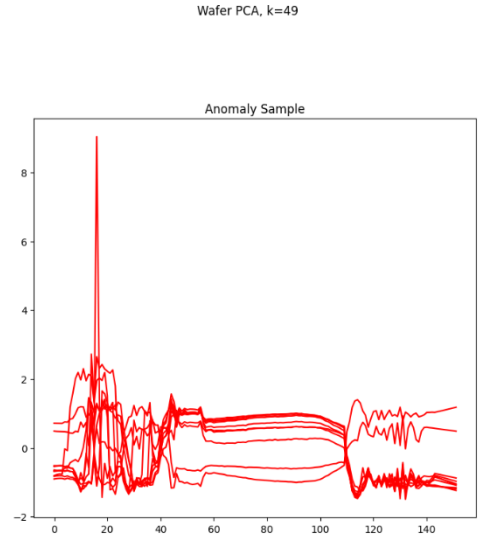
Function 呼叫順序為:

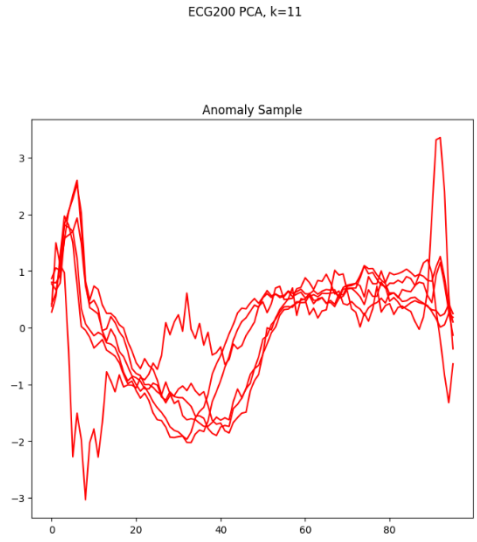
```
PCA_AD(train_data, test_data, test_label, k=i)
```

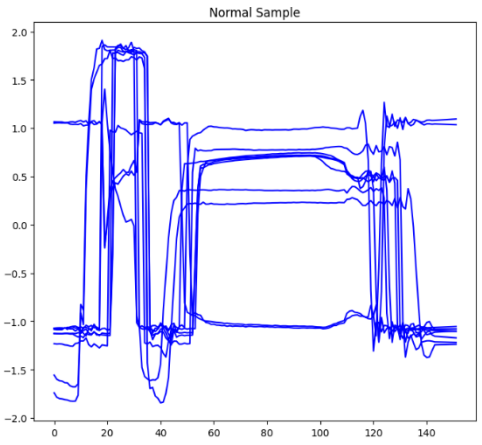
此處 **N** 值嘗試 **1** 到 **data** 的 **attribute** 數的 **1/4**。

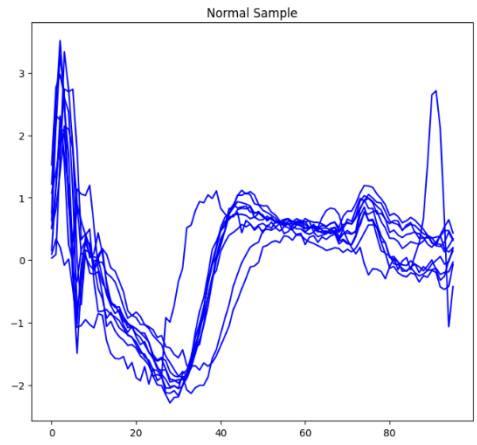
PCA_AD():

1. 使用 **training data** 找出 **PCA** 的 **N** 個 **components**。
2. 使用這 **N** 個 **components** 將 **testing data** 轉換成 **feature**，再將 **feature** 重新 **reconstruct** 回 **testing data**。
3. **Reconstruction** 與原 **testing data** 之間的距離即為 **anomaly score**









Implementation code

```python
def PCA_AD(train_X, test_X, test_y, k, return_recons=False):
    pca = PCA(n_components=k)
    pca.fit(train_X)
    test_recons = pca.inverse_transform(pca.transform(test_X))
    score = np.linalg.norm(test_X-test_recons, axis=1)
    if return_recons:
        return test_recons, roc_auc_score(y_true=test_y, y_score=score)

    return roc_auc_score(y_true=test_y, y_score=score)
```

(4) Discrete Fourier Transform

Function 呼叫順序為:

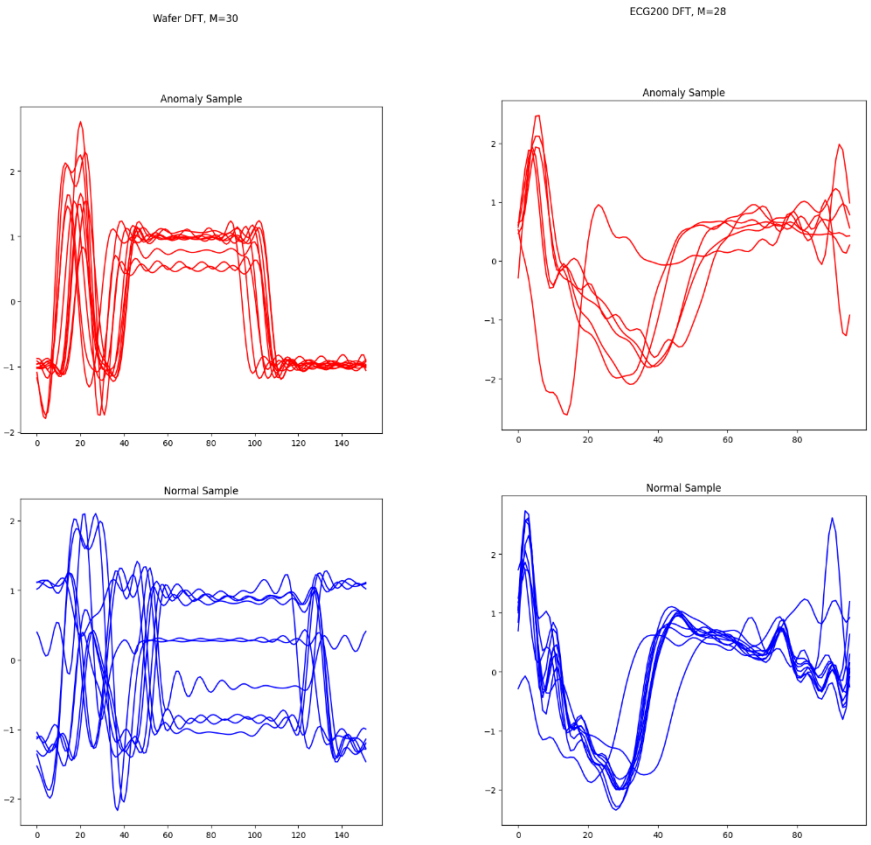
```
DFT_AD = AD(DFT)
DFT_AD(train_data, train_label, test_data, test_label, arg=i)
```

此處 **M** 值嘗試 **1** 到 **data** 的 **attribute** 數的 **1/2**。

AD(): 同上。

DFT():

1. 對 **data** 做 **discrete Fourier Transform** 取得 **coefficients**。選擇 **coefficients** 中對應最小的 **M** 個 **frequency** 的 **coefficients** 作為 **features**
2. 用 **features** 計算 **magnitudes** 及 **reconstruction**。
3. **Magnitudes** 丟給 **KNN** 計算 **anomaly score**，**reconstruction** 用來 **visualization**。



Implementation code

```python
def DFT(X, M=20, use_first_m = False, return_recons = False):
    ceo     = np.fft.fft(X)
    mag_idx = np.zeros(X.shape[-1], dtype=bool)

    if use_first_m:       # use first M coes as features
        mag_idx[:M] = True
    else:                 # use first M/2 and last M/2 coes as featrues
        mag_idx[:int(np.ceil(M/2))]   = True
        mag_idx[-int(np.floor(M/2)):] = True

    magnitudes               = np.abs(ceo[:, mag_idx])
    ceo_selected             = np.zeros_like(ceo)
    ceo_selected[:, mag_idx] = ceo[:, mag_idx]
    recons                   = np.fft.ifft(ceo_selected)

    if return_recons:
        return magnitudes, recons
    else:
        return magnitudes
```

(5) Discrete Wavelet Transform

Function 呼叫順序為:

```
DWT_AD = AD(DWT)
DWT_AD(train_data, train_label, test_data, test_label, arg=i)
```

此處 **S** 值嘗試 **1** 到 **data** 的 **level** 值**+1**。

另外，**level=range([1,ceiling(log2(feature_dim))])**

AD(): 同上。

DWT():

1. 對 **Data** 做 **Discrete Wavelet Transform**，並取 **last level** 的 **A coefficient** 及倒數 **S-1** 個 **D coefficients** 作為 **features**。
2. 將 **features** 丟入 **KNN** 中計算 **anomaly score**。

Implementation code

```python
def DWT(X, S, return_coes=False):
    S        = 2**S
    levels   = int(np.ceil(np.ma.log2(X.shape[-1])))
    size     = 2**(levels) - 1
    X_padded = np.pad(X, pad_width=((0,0),(0,2**levels-X.shape[-1])), mode='constant',constant_values=0)
    A_coes   = np.pad(X_padded, ((0,0), (0, size)), mode='constant', constant_values=0)
    D_coes   = np.zeros((X.shape[0], size))

    prev_begin = 0
    begin      = X_padded.shape[-1]
    D_begin    = 0

    for i in range(1,levels+1):
        len = levels - i

        A_coes[:, begin:begin+2**len]     = (A_coes[:,prev_begin+1:begin:2] + A_coes[:,prev_begin:begin:2]) / 2
        D_coes[:, D_begin:D_begin+2**len] = (A_coes[:,prev_begin+1:begin:2] - A_coes[:,prev_begin:begin:2]) / 2

        prev_begin  = begin
        begin      += 2**len
        D_begin    += 2**len

    out_A_coes = np.expand_dims(A_coes[:, -1], axis=1)
    out_D_coes = D_coes[:, -1:-S:-1]
    if return_coes:
        return np.concatenate([out_A_coes, out_D_coes], axis=1),  A_coes[:, -size:], D_coes

    return np.concatenate([out_A_coes, out_D_coes], axis=1)
```

Performance

1. Raw Data:

Wafer 及 ECG200 在 raw data 的 knn anomaly detection 上 performance 分別為: Wafer: 0.9884 , ECG200: 0.8750

2. PCA:

Wafer 及 ECG200 在經由 PCA 處理後，performance 最大值分別為:

Wafer: 0.9961, N=49

ECG200: 0.8568, N=11

另外此處 N 值的範圍為 1~ feature_dim/4(實際取到 25 的倍數)

3. DFT:

Wafer 及 ECG200 在經由 DFT 處理後，performance 最大值分別為:

Wafer: 0.9984, M=30

ECG200: 0.8568, M=11

另外此處 M 值的範圍為 1~ feature_dim/2(實際取到 50 的倍數)

4. DWT:

Wafer 及 ECG200 在經由 DWT 處理後，performance 最大值分別為:

Wafer: 0.9986, S=8

ECG200: 0.9479, S=4

Bonus:

此處針對 **Raw Data, DFT, DWT** 進行以下混合實驗:

KNN : K ->[1,10]

DFT : M -> [1, feature_dim/4](實際上無條件進位到 25 的倍數)

DWT : S -> 2^[1, ceiling(log2(feature_dim))+1]

最終各個 **methods** 在兩個 **datasets** 的最佳 **performance** 如下:

1. **Wafer:**

   Raw Data, k=1, ROC-AUC : 0.9913

   DFT, k=2, M=29, ROC-AUC : 0.9985

   DWT, k=10, S=8, ROC-AUC : 0.9987

2. **ECG200:**

   Raw Data, k=5, ROC-AUC : 0.8750

   DFT, k=4, M=45, ROC-AUC : 0.9297

   DWT, k=3, S=4, ROC-AUC : 0.9531