CS5658 Anomaly Detection

Homework2 – TimeSeries AD

Student ID: 109062211 Name: 張惇媛

---

0. 定義常用 function

Score(label, anomaly_score) – 用於計算 roc_auc_score

```
## roc_auc_score
def score(label, anomaly_score):
    return roc_auc_score(label, anomaly_score)
```

使用 scikit-learn.metric 中的 roc_auc_score 得出 roc_auc_score。

Plot(scores, metric, hyperparameter, category) – 用於找出最佳的 hyperparamter 與其 score、繪製不同 hyperparameter 數值下的 roc_auc_score 表現之折線圖。

```
## plot(x - hyperparameter; y - roc_auc_score)
def plot(scores, metric, hyperparameter, category):
    best_hyperparameter, best_score = max(scores, key=lambda x: x[1])
    print(f'{metric} anomaly detection ({category} Dataset): \n \
        {hyperparameter} = {best_hyperparameter}, score = {best_score:.4f}')

    plt.plot(*list(zip(*scores)), color = 'dodgerblue')
    plt.ylabel('score')
    plt.xlabel(hyperparameter)
    plt.title(f'{metric} Anomaly Detection\n({category})')
    plt.show()

    return best_hyperparameter, best_score
```

首先根據 hyperparamter 數值與對應的 roc_auc_score 之組合列表 scores，找出最佳的 hyperparamter 與其 score，並在最後回傳，可用於後面的最佳 hyperparamter 之 visualization 部分。接著，繪製在不同 hyperparameter 數值下的 roc_auc_score 表現之折線圖，x 軸為 hyperparameter、y 軸為 roc_auc_score。

1. Visualization

- Implementation

```
def random_choice(data, label, num_normal, num_anomaly, label_normal, label_anomaly):
    normal = data[label == label_normal]
    if num_normal < len(normal):
        idx = np.random.choice(len(normal), num_normal, replace=False)
        normal_data = np.array([normal[idx[i]] for i in range(num_normal)])
    else:
        normal_data = normal

    anomaly = data[label == label_anomaly]
    if num_anomaly < len(anomaly):
        idx = np.random.choice(len(anomaly), num_anomaly, replace=False)
        anomaly_data = np.array([anomaly[idx[i]] for i in range(num_anomaly)])
    else:
        anomaly_data = anomaly

    return len(normal_data), np.concatenate((normal_data, anomaly_data), axis=0)
```

將參數 data, label, num_normal, num_anomaly, label_normal, label_anomaly 用於 random_choice function，根據 label 從 data 中隨機挑選出 normal data. anomaly data(為使 function 能重複使用於後面部分，增加 if-else 判斷，若數量不足時，則使用類別中的全部 data)，之後回傳 num_normal, chosen_data。

```python
def visualization(data, num_normal, title):
    num, feat = data.shape

    plt.figure(figsize=(5, 8))
    plt.suptitle(f'Dataset {title}')

    x = np.linspace(1, feat, feat)

    plt.subplot(211)
    for i in range(num_normal):
        plt.plot(x, data[i], color = 'b')
    plt.title('Normal Sample')

    plt.subplot(212)
    for i in range(num_normal, num):
        plt.plot(x, data[i], color = 'r')
    plt.title('Anomaly Sample')

    plt.show()
    return
```

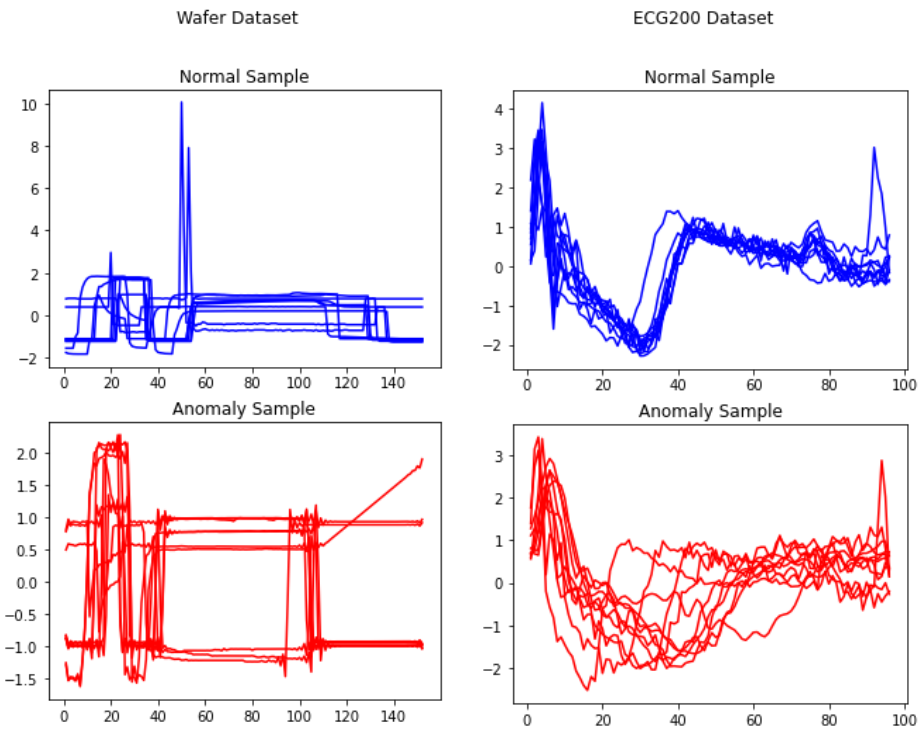
接著，visualization function 參考 HW spec 中的範例，將 chosen_data 分為 normal sample. anomaly sample 分別繪製成折線圖，x 軸為 features；y 軸為 value。

以下為 code 中的實際使用：

```python
## (1) Visualization
num_normal, data = random_choice(test_data, test_label, 10, 10, 1, -1)
visualization(data, num_normal, category)
```

Visualization 此部分將從 resample 前的 test data 中挑選，因此不會發生數量不足的現象。

- Result



2. Raw Data

- Implementation

```
def KNN_ad(training, testing, k):
    knn_model = NearestNeighbors(n_neighbors=k, metric='euclidean').fit(training)
    dist, _ = knn_model.kneighbors(testing)
    return np.mean(dist, axis=1)
```

使用 scikit-learn 中的 NearestNeighbors 進行 KNN anomaly detection，其中 metric ='eucliden'，首先將 KNN model fit training data，接著得出 testing data 的 k distances，最後取 k distances 之 mean 作為 anomaly score，用於 roc_auc_score 的計算。

以下為 code 中的實際使用：

```
## (2) KNN
k = 5
result = KNN_ad(train_data, test_data, k)
print(f'KNN anomaly detection ({category} Dataset): K = {k}, score = {score(test_label, result):.4f}')
```

- Performance
  - KNN anomaly detection (Wafer Dataset):
    K = 5, score = 0.9886
  - KNN anomaly detection (ECG200 Dataset):
    K = 5, score = 0.8620

3. PCA Reconstruction

- Implementation

```
def PCA_reconstruction(training, testing, N):
    pca_model = PCA(n_components = N).fit(training)
    data = pca_model.inverse_transform(pca_model.transform(testing))
    errors = pairwise_distances(testing, data, metric='euclidean')

    return errors.diagonal()

def PCA_visualization(training, testing, label, N, category):
    pca_model = PCA(n_components = N).fit(training)

    num_normal, data = random_choice(testing, label, 10, 10, 0, 1)
    chosen_data = pca_model.inverse_transform(pca_model.transform(data))
    visualization(chosen_data, num_normal, category + f'\nN={N}')
    return
```

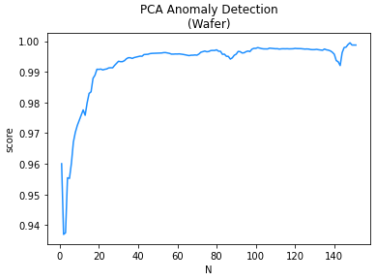
使用 scikit-learn 中的 PCA 進行 PCA reconstruction，其中 n_component 為 hyperparameter。首先將 PCA model fit training data，接著利用 PCA model 之 transform & inverse_transform 重建 testing data，最後計算 testing data 重建前後的 euclidean pairwise_distances 得出 errors，取 errors 之對角線(各 sapmple 的 reconstruction error)作為 anomaly score，用於 roc_auc_score 的計算。

PCA Visualization 的部分，與前面 implementation 中 PCA 重建 testing data 的部分進行同樣的處理方式，其中使用到前方所定義之 random_choice function 挑選 samples 進行 PCA reconstruction、visualization function 將 PCA reconstruction 後的 samples 進行視覺化。

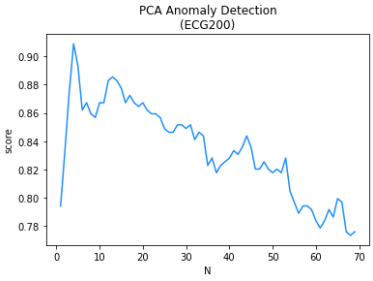
以下為 code 中的實際使用：

```
## (3) PCA
PCA_score = []
for n in range(1, min(n_samples, n_features)): # hyperparameter
    result = PCA_reconstruction(train_data, test_data, n)
    PCA_score.append([n, score(test_label, result)])

best_n, _ = plot(PCA_score, 'PCA', 'N', category)
PCA_visualization(train_data, test_data, test_label, best_n, category)
```

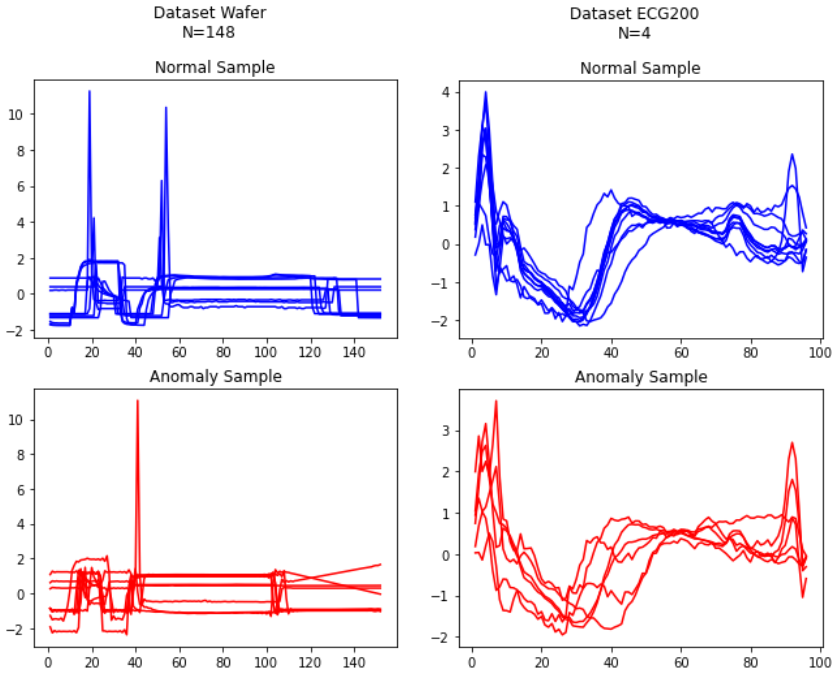
- Performance
  - PCA anomaly detection(Wafer Dataset): N = 148, score=0.9995



  - PCA anomaly detection(ECG200 Dataset): N = 4, score=0.9089



- Result



- Analysis

對於 Wafer Dataset，N = 148 時，有最好的 performance，roc_auc_score 為 0.9995；對於 ECG200 Dataset，N = 4 時，有最好的 performance，roc_auc_score 為 0.9089。

N 屬於 hyperparameter，當 N 太小時，會產生 underfitting，保留較少的 information 而不足以分辨出類別間的界線；當 N 太大時，會產生 overfitting，無法良好地分辨出沒見過的 testing data 之類別，因此其最適值取決於 dataset。推測 Wafer 在較大的 N，具有較好的 performance 之原因為捕捉到較細微、複雜的 information，可以分辨不同的類別。ECG200 在較小的 N，具有較好的 performance 之原因為沒有 overfitting 於 training data，而無法分辨出沒見過的 testing data 之類別。

4. Discrete Fourier Transform

- Implementation

```
def DFT_ad(training, testing, n_features, M, k=5):
    freq = np.argsort(np.abs(np.fft.fftfreq(n_features)))[:M]

    train_dft = np.fft.fft(training)[:, freq]
    test_dft = np.fft.fft(testing)[:, freq]

    train_feat = train_dft.real
    test_feat = test_dft.real

    return KNN_ad(train_feat, test_feat, k) # anomaly_score
```

使用 numpy 中的 fft 進行 Discrete Fourier Transform，其中 M 為 hyperparameter。DFT_ad function 中首先使用 fft.freq 之絕對值得出 n_features 數量中頻率的絕對值最低之 M coefficients 的 index，接著將 training / testing data 分別進行 fft 再取出頻率最小的 M coefficients，並取個別的實數(real)部份作為 feature vector，並使用到前面的 KNN_ad function 來進行 KNN anomaly detection，最後得出 anomaly score。

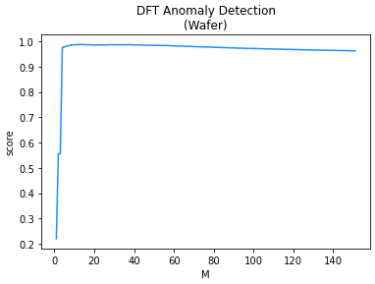
```
def DFT_visualization(data, label, n_features, M, category):
    num_normal, chosen_data = random_choice(data, label, 10, 10, 0, 1)

    dft = np.fft.fft(chosen_data)
    indices = np.argsort(np.abs(np.fft.fftfreq(n_features)))[:M]

    dft_mask = np.zeros_like(dft, dtype=np.complex128)
    rows = np.arange(chosen_data.shape[0])[:, np.newaxis]
    dft_mask[rows, indices] = dft[rows, indices]

    inverse_dft = np.fft.ifft(dft_mask).real
    visualization(inverse_dft, num_normal, category + f'\nM={M}')
    return
```

DFT Visualization 的部分，首先使用到前方所定義之 random_choice function 挑選 samples 進行 DFT / inverse DFT，接著與前面 implementation 中 DFT 挑選 feature vector 的部分進行同樣的處理方式，並且使用 mask 將其餘 index 的數值以零呈現，最後將 mask 後的 DFT data 進行 inverse DFT 後，取實數(real)部份，使用 visualization function 將 DFT / inverse DFT 後的實數部分之 samples 進行視覺化。

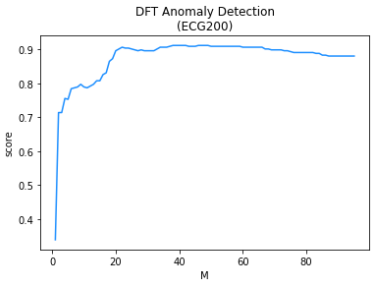
以下為 code 中的實際使用：

```
## (4) DFT
DFT_score = []
for m in range(1, n_features): # hyperparameter
    result = DFT_ad(train_data, test_data, n_features, m)
    DFT_score.append([m, score(test_label, result)])

best_m, _ = plot(DFT_score, 'DFT', 'M', category)
DFT_visualization(test_data, test_label, n_features, best_m, category)
```

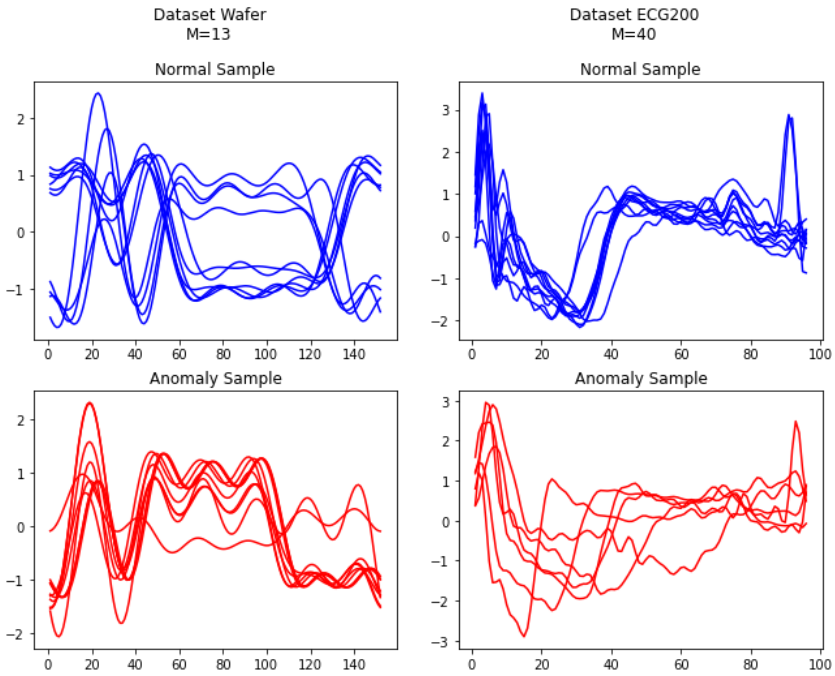
- Performance
  - DFT anomaly detection(Wafer Dataset): M=13, score = 0.9872



  - DFT anomaly detection(ECG200 Dataset): M=40, score = 0.9115



- Result



- Analysis

對於 Wafer Dataset，M = 13 時，有最好的 performance，roc_auc_score 為 0.9872；對於 ECG200 Dataset，M = 40 時，有最好的 performance，roc_auc_score 為 0.9115。

M 屬於 hyperparameter，當 M 太小時，會產生 underfitting，損失較多 information，使類別間的界線變得模糊；當 M 太大時，會產生 overfitting，對 noise 較為敏感，因此其最適值取決於 dataset。由 performance 可以看出在兩個 dataset 中，兩者 score 皆是在一開始急遽上升，而後呈現平緩或下降之趨勢，符合分析的推論。

5. Discrete Wavelet Transform

- Implementation

```
def padding(training, testing, n_features):
    levels = int(np.ceil(np.log2(n_features)))
    padding_shape = ((0, 0) , (0, int(2**levels) - n_features))
    train_padding = np.pad(training, padding_shape)
    test_padding = np.pad(testing, padding_shape)

    return levels, train_padding, test_padding
```

```
def Discrete_Wavelet_Transform(training, testing, levels):
    n, feat = training.shape
    m, feat = testing.shape

    train_avg = np.zeros((levels+1, feat))
    test_avg = np.zeros((levels+1, feat))

    train_diff = np.zeros((levels+1, feat))
    test_diff = np.zeros((levels+1, feat))

    train_feat = np.zeros((n, feat))
    test_feat = np.zeros((m, feat))

    for i in range(n):
        train_avg[0] = training[i]
        for row in range(1, levels+1):
            for col in range(0, feat, 2):
                train_avg[row, col//2] = (train_avg[row-1, col] + train_avg[row-1, col+1])/2
                train_diff[row, col//2] = (train_avg[row-1, col+1] - train_avg[row-1, col])/2

        feat_vector = [train_avg[levels, 0]]
        for row in range(levels):
            feat_vector = np.append(feat_vector, train_diff[levels-row, :int(2**row)])

        train_feat[i] = feat_vector

    for i in range(m):
        test_avg[0] = testing[i]
        for row in range(1, levels+1):
            for col in range(0, feat, 2):
                test_avg[row, col//2] = (test_avg[row-1, col] + test_avg[row-1, col+1])/2
                test_diff[row, col//2] = (test_avg[row-1, col+1] - test_avg[row-1, col])/2

        feat_vector = [test_avg[levels, 0]]
        for row in range(levels):
            feat_vector = np.append(feat_vector, test_diff[levels-row, :int(2**row)])

        test_feat[i] = feat_vector

    return train_feat, test_feat
```

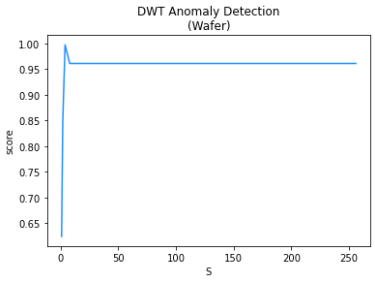
```python
def DWT_ad(training, testing, Si, k=5):
    S = int(2**Si)
    train_feat = training[:,:S]
    test_feat = testing[:,:S]

    return KNN_ad(train_feat, test_feat, k) # anomaly_score
```

參考 HW spec 中的 process. figure 進行 Discrete Wavelet Transform，首先 padding function 將 training / testing data 之 features 數量 padding 至 2 的冪次方，接著 Discrete Wavelet Transform function 中分別計算出 training / testing data 之每層 level 之 avg. diff 後，取出 avg 之最後一 level 的 value 與 diff 之所有 level 的 value，將所有 values 回傳，最後 DWT_ad function 再取出 S(power of 2) the most significant coefficients 作為 feature vector，其中 S 為 hyperparameter，並且使用到前面的 KNN_ad function 來進行 KNN anomaly detection，最後得出 anomaly score。

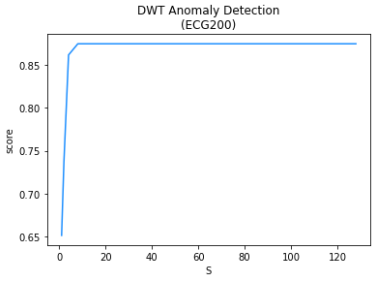
以下為 code 中的實際使用：

```python
## (5) DWT
levels, train_padding, test_padding = padding(train_data, test_data, n_features)
train_feat, test_feat = Discrete_Wavelet_Transform(train_padding, test_padding, levels)

DWT_score = []
for si in range(levels+1): # hyperparameter
    s = int(2**si)
    result = DWT_ad(train_feat, test_feat, s)
    DWT_score.append([s, score(test_label, result)])

best_s, _ = plot(DWT_score, 'DWT', 'S', category)
```

- Performance
  - DWT anomaly detection (Wafer Dataset): S=4, score = 0.9976



  - DWT anomaly detection (ECG200 Dataset): S=8, score = 0.8750



- Analysis

對於 Wafer Dataset，S = 4 時，有最好的 performance，roc_auc_score 為 0.9976；
對於 ECG200 Dataset，S = 8 時，有最好的 performance，roc_auc_score 為 0.8750。

S 屬於 hyperparameter，當 S 太小時，會產生 underfitting，損失較多 information，使類別間的界線變得模糊；當 S 太大時，會產生 overfitting，對 noise 較為敏感，因此其最適值取決於 dataset。
由 performance 可以看出在兩個 dataset 中，兩者 score 皆是在一開始急遽上升，而後呈現平緩或下降之趨勢，符合上方之分析。

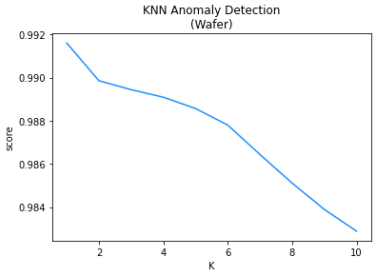
Bonus

- Implementation

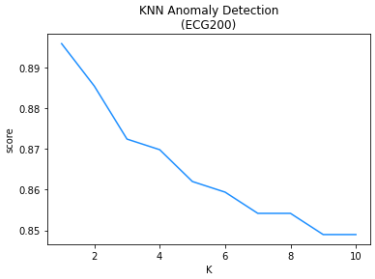
利用前面已完成的 function(KNN_ad. DFT_ad. DWT_ad)以及在 DFT. DWT 的部分中找到的最佳 hyperparamer(best_m. best_s)，在設定不同的 k 值下，進行 KNN / DFT / DWT anomaly detection。
以下為 code 中的實際使用：

```
## Bonus
### KNN for different k
KNN_score = []
for k in range(1, 11): ## hyperparameter
    result = KNN_ad(train_data, test_data, k)
    KNN_score.append([k, score(test_label, result)])

best_k, _ = plot(KNN_score, 'KNN', 'K', category)

### DFT for different k
KNN_score = []
for k in range(1, 11): ## hyperparameter
    result = DFT_ad(train_data, test_data, n_features, best_m, k)
    KNN_score.append([k, score(test_label, result)])

best_k, _ = plot(KNN_score, 'DFT', 'K', category)

### DWT for different k
KNN_score = []
for k in range(1, 11): ## hyperparameter
    result = DWT_ad(train_feat, test_feat, best_s, k)
    KNN_score.append([k, score(test_label, result)])

best_k, _ = plot(KNN_score, 'DWT', 'K', category)
```

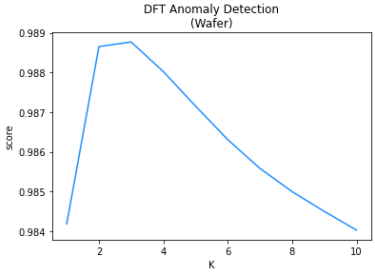
- Performance
  - KNN anomaly detection (Wafer Dataset): K = 1, score = 0.9916



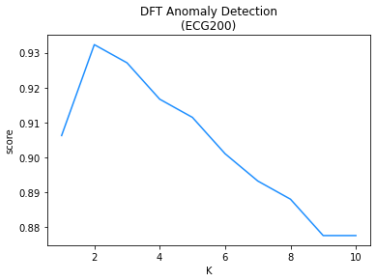
- KNN anomaly detection(ECG200 Dataset): K = 1, score = 0.8958



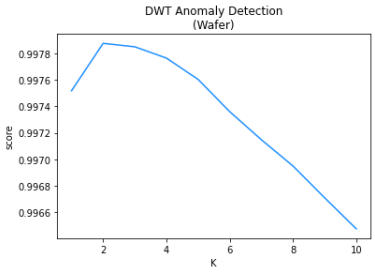
- DFT anomaly detection (Wafer Dataset): K = 3, score = 0.9888



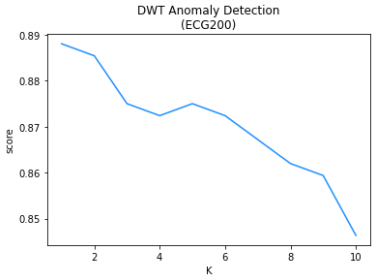
- DFT anomaly detection(ECG200 Dataset): K = 2, score = 0.9323



- DWT anomaly detection (Wafer Dataset): K = 2, score = 0.9979



- DWT anomaly detection(ECG200 Dataset): K = 1, score = 0.8880



- Analysis

KNN

對於 Wafer Dataset，K = 1 時，有最好的 performance，roc_auc_score 為 0.9916；
對於 ECG200 Dataset，K = 1 時，有最好的 performance，roc_auc_score 為 0.8958。

DFT

對於 Wafer Dataset，M=13, K=3 時，有最好的 performance，roc_auc_score 為 0.9888；
對於 ECG200 Dataset，M=40, K=2 時，有最好的 performance，roc_auc_score 為 0.9323。

DWT

對於 Wafer Dataset，S=4, K=2 時，有最好的 performance，roc_auc_score 為 0.9979；
對於 ECG200 Dataset，S=8, K=1 時，有最好的 performance，roc_auc_score 為 0.8880。

K 屬於 hyperparameter，當 K 太小時，會較容易遭受 noise 的影響；當 K 太大時，會使類別間的界線變得模糊，因此其最適值取決於 dataset。

Conclusion

對於 Wafer Dataset，使用 Discrete Wavelet Transform anomaly detection，且 K = 2, S = 4 時，有最好的 performance，roc_auc_score 為 0.9979。
對於 ECG200 Dataset，使用 Discrete Fourier Transform anomaly detection，且 K = 2, M = 40 時，有最好的 performance，roc_auc_score 為 0.9323；