Report of Anomaly Detection HW1

Name: 高煒軒

Student ID: 112062646

Implementation & explanation

本次作業中的四個 **algorithm's implementation code & explanation**，會依據四個不同的 **algorithm** 各自解釋。

(1) KNN

Function 呼叫順序為**:**

```
knns.fit(train_data)
knns.prediction(test_data)
```

fit(): 記住 **input training data**。

score(): 計算 **data** 之間的距離。

prediction(): 計算出距離後選出前 **k** 個近的 **data**，並且這前 **k** 個近的 **data** 距離平均即為 **anomaly score**。

Implementation code

```python
from sklearn.metrics import pairwise_distances

def euclidean(point, data, axis=None):
    return np.sqrt(np.sum((point - data)**2, axis=axis))

class KNN_AD:
    def __init__(self, k=1):
        '''
        Implement of Anomaly Detection Algorithm with KNN-classifier
        :param k: "K" neariset neighbor
        '''
        self.k = k

    def fit(self, X):
        self.X = X

    def score(self, Y):
        return pairwise_distances(X=Y, Y=self.X)

    def prediction(self, Y, normal=False):
        # Get sorted pairwise distances training and testing data
        pair_dist = self.score(Y)
        sorted_idx = np.argsort(pair_dist)
        sorted_idx = sorted_idx[:, :self.k]

        selected = [self.X[index] for index in sorted_idx]
        selected = np.array(selected)

        output = [np.sum(euclidean(point=Y[idx], data=selected[idx], axis=1)) for idx in range(Y.shape[0])]
        output = np.array(output)
        output = output / self.k

        # Mapping output -> [0,1]
        if normal:
            Min = min(output)
            Max = max(output)
            output = (output - Min)/(Max - Min)

        return output
```

(2) K-means

Function 呼叫順序為**:**

```
kms.fit(train_data)
kms.prediction(test_data)
```

fit():

1. 選出 **k** 個 **centroids**
2. 將 **data** 分配到距離最近的 **cluster**，再重新計算 **centorids(**取 **cluster** 中的所有 **data** 的平均**)**
3. 以上迭代至 **centroids** 不再更新或到達最大迭代次數。

score(): 計算 **data** 與 **centroids** 之間的距離

prediction(): 計算出 **data** 與 **centroids** 之間的距離後，將這些距離平均即為該 **data** 的 **anomaly score**。

Implementation code

```python
class KMeans_AD:
    def __init__(self, n_clusters=1, max_iter=300):
        self.n_clusters = n_clusters
        self.max_iter = max_iter

    def fit(self, X):
        idx = np.random.choice(X.shape[0] , self.n_clusters, replace=False)
        self.centroids = np.array([X[i] for i in idx])

        iteration = 0
        prev_centroids = np.array(None)
        while (self.centroids != prev_centroids).any() and iteration < self.max_iter:

            dists = pairwise_distances(X, self.centroids)
            centroids_id = np.argmin(dists, axis=1)

            buckets = []
            for i in range(self.n_clusters):
                buckets.append(np.where(centroids_id==i))

            prev_centroids = np.copy(self.centroids)
            for i in range(self.n_clusters):
                self.centroids[i] = np.mean(X[buckets[i]], axis=0)

            for i, centroid in enumerate(self.centroids):
                if np.isnan(centroid).any():
                    self.centroids[i] = prev_centroids[i]
            iteration += 1

    def score(self, Y):
        return pairwise_distances(X=Y, Y=self.centroids)

    def prediction(self, Y, normal=False):
        pair_dists = self.score(Y)
        output = np.min(pair_dists, axis=1)

        if normal:
            Min = min(output)
            Max = max(output)
            output = (output - Min)/(Max - Min)

        return output
```

(3) Distance Base

Function 呼叫順序為:

```
dist_ad.prediction(X=test_data, ref=train_data, metric=metrics[i])
```

prediction(): 計算出 data 之間的距離後，將最近的五個 data 距離平均即為該 data 的 anomaly score。

ref: input parameter, 用來計算 Mahalanobis Distance 所需之 covariance matrix。

metric: input parameter, 用來選擇使用哪個 distance metric。

CosineDistance (X, Y):

i.e. (1 – Consine similarity(X, Y))

Consine similarity definition:

$$\cos(X, Y) = \frac{X \cdot Y}{\|X\|\|Y\|}$$

L1Distance (X, Y):

Minkowski Distance, if r=1. Also called Manhattan distance.

Definition:

$$d(X, Y) = \sum_i |X_i - Y_i|$$

L2Distance (X, Y):

Minkowski Distance, if r=2. Also called Euclidean distance.

Definition:

$$d(X,Y) = \sqrt[2]{\sum_i (X_i - Y_i)^2}$$

ChebyshevDistance (X, Y):

Minkowski Distance, if r=∞.

Definition:

$$d(X,Y) = \max_i |X_i - Y_i|$$

MahalanobisDistance (X, Y, VI):

Mahalanobis distance, 其中 VI 為 inverse covariance matrix 以 training data 計算求得。

Definition:

$$VI = V^{-1}$$

$$d(X,Y,V^{-1}) = \sqrt{(X-Y)^T V^{-1} (X-Y)}$$

Implementation code

```python
def CosineDistance(X, Y):
        X_dot_Y = np.dot(X,Y)
        X_norm = np.linalg.norm(X)
        Y_norm = np.linalg.norm(Y)
        return 1 - (X_dot_Y / (X_norm * Y_norm))

def L1Distance(X, Y):
    return np.sum(np.abs(X - Y))

def L2Distance(X, Y):
    return np.linalg.norm(X - Y)

def ChebyshevDistance(X, Y):
    return np.max(np.abs(X-Y))

def MahalanobisDistance(X, Y, VI):
    diff = X - Y
    return np.sqrt(np.matmul(np.matmul(diff, VI), diff.T))

class Distance_AD:
    def __init__(self, k=5):
        self.k = k
        self.metrics = {"CosineDistance": CosineDistance,
                        "L1Distance": L1Distance,
                        "L2Distance": L2Distance,
                        "ChebyshevDistance": ChebyshevDistance,
                        "MahalanobisDistance": MahalanobisDistance}

    def predict(self, X, ref=None, metric="L2Distance"):
        if (ref != None).any() and metric=="MahalanobisDistance":
            self.VI = np.linalg.inv(np.cov(ref.T))
            dists = pairwise_distances(X=X, metric=self.metrics[metric], VI=self.VI)
        else:
            dists = pairwise_distances(X=X, metric=self.metrics[metric])
        output = np.sort(dists, axis=1)[:, self.k]
        return output
```

(4) LOF

Function 呼叫順序為:

```
Lof.predict(test_data)
```

prediction(): 依序呼叫 **_KNN (), _reach_distance (), _LRD (), _LOF_score ()**後輸出 **data** 的 **anomaly score**。

_KNN (): 計算出 **data** 之間的距離**(dist)**，並以此記錄各個 **data** 的 **k near neighbor(**即 **k_near)**和第 **k** 個 **neighbor** 的距離**(**即 **k_dist)**。

k_near 及 **k_dist** 請參考 **implementation code**。

_reach_distance (): 計算各個 **data** 的 **reachable distance**。

※為加速 **algorithm**，我不使用 **loop** 來計算 **reachable distance**，而是將 **k_dist** 擴展到與 **dists** 相同維度並比較大小。

_LRD (): 計算 **local reachable density of data**，此處 **output** 為 **lrd** 的 **inverse**。

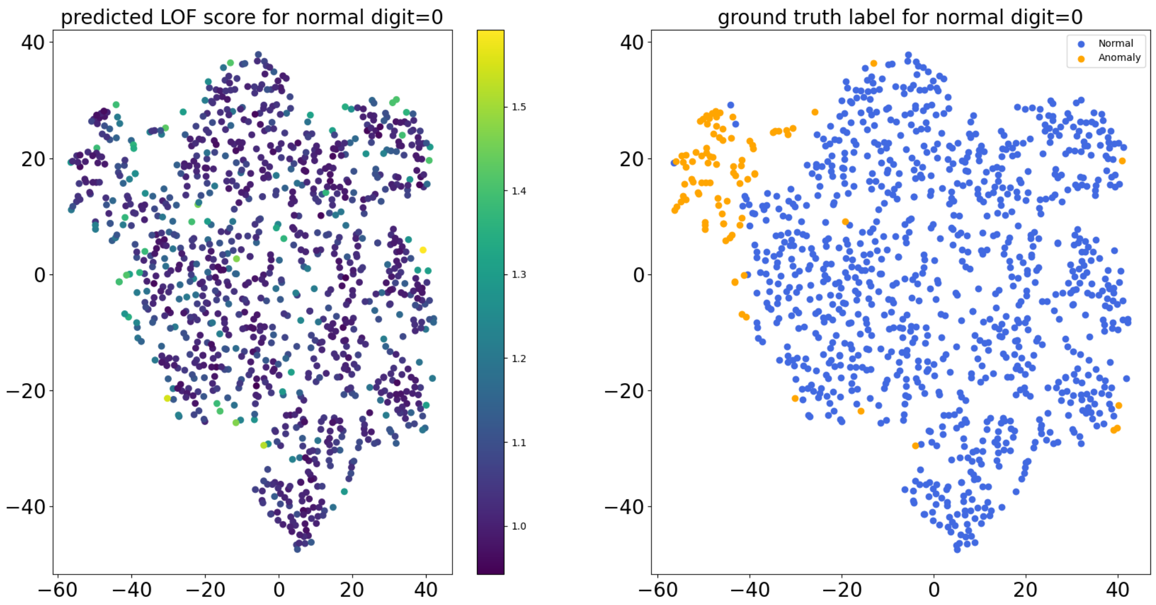
_LOF_score (): 計算 **local outlier factor of data**。

Implementation code

```
class LOF:
    def __init__(self, k=5):
        self.k = k

    def _KNN(self, X):
        dists  = pairwise_distances(X=X)
        k_near = np.argsort(dists, axis=1)[:, 1:self.k+1]
        k_dist = np.sort(dists, axis=1)[:, self.k]

        return dists, k_near, k_dist

    def _reach_distance(self, dists, k_dist):
        k_dist_ex  = np.expand_dims(k_dist, axis=0)
        k_dist_rep = np.repeat(k_dist_ex, self.size, axis=0)
        reach_dist = np.maximum(k_dist_rep, dists)
        assert (reach_dist > 0).all(), "Error:: Reachable Distance expect to be > 0, but receive <= 0 value"

        return reach_dist

    def _LRD(self, reach_dist, k_n (variable) idx: int
        # Local reachablility dist
        k_reachDist = [reach_dist[idx, k_near[idx]] for idx in range(self.size)]
        lrd = np.array([len(k_near[i])/(k_reachDist[i].mean()) for i in range(self.size)])

        return lrd

    def _LOF_score(self, lrd, k_near):
        k_lrds = [lrd[k_near[idx]] for idx in range(self.size)]
        lof = np.array([k_lrds[idx].sum()/lrd[idx] for idx in range(self.size)])

        return lof / self.k

    def predict(self, X):
        self.size = X.shape[0]
        dists, k_near, k_dist = self._KNN(X)
        reach_dist = self._reach_distance(dists, k_dist)
        lrd = self._LRD(reach_dist, k_near)
        output = self._LOF_score(lrd, k_near)

        return output
```

Visualization

```python
# The TSNE.png of LOF
test_data,test_label = resample(orig_test_data,orig_test_label,target_label=0,outlier_ratio=0.1)
test_label = np.where(test_label==0, 0, 1)
Lof_score = Lof.predict(test_data)

t_sne       = TSNE()
t_sne_out   = t_sne.fit_transform(test_data)
normal_idx  = test_label == 0
anomaly_idx = test_label == 1

fig, axs = plt.subplots(1,2, figsize=(20,10))

im = axs[0].scatter(t_sne_out[:,0], t_sne_out[:,1], c=Lof_score)
axs[0].set_title("predicted LOF score for normal digit=0", fontsize=20)
axs[0].tick_params(axis='both', which='major', labelsize=20)

axs[1].scatter(t_sne_out[normal_idx,0],
            t_sne_out[normal_idx,1],
            c='royalblue',
            label='Normal')
axs[1].scatter(t_sne_out[anomaly_idx,0],
            t_sne_out[anomaly_idx,1],
            c='orange',
            label='Anomaly')
axs[1].legend()
axs[1].set_title("ground truth label for normal digit=0", fontsize=20)
axs[1].tick_params(axis='both', which='major', labelsize=20)

plt.colorbar(im, ax=axs[0])
plt.savefig("TSNE.png", format="png")
plt.show()
```



Performance

```
Average ROC-AUC of KNN:           (k=1)    0.9658, (k=5) 0.9683, (k=10) 0.9669
Average ROC-AUC of K-means:       (k=1)    0.9032, (k=5) 0.9487, (k=10) 0.9611
Average ROC-AUC of Distance-Base: (Cosine) 0.9744, (r=1) 0.9483, (r=2)  0.9517, (r=inf) 0.9532, (mahalanobis) 0.9794
Average ROC-AUC of LOF:  0.7644
```

1. KNN:

根據 **ROC-AUC**，可以看到 **KNN** 在這次的 **anomaly detection task** 的效能不會因為 **k** 值的改變而有太多的效能增減。此現象可能與 **MNIST** 的不同的 **label** 對應的圖像經過 **PCA** 降維後分布差距較大有關。

2. K-means:

相對於上述之 **KNN**，我們可以看到 **K-means** 的效能根據 **k** 值而有所改變。這可能是因為當 **k** 值設定過小時，會導致過多的 **normal data** 被判斷為 **anomaly**。

3. Distance Base:

在我的作業撰寫過程中，我使用了 **sklearn** 的 **api : pairwise_distance** 來幫助計算距離，並使用自己寫的 **metrics function** 作為此 **api** 的 **callback function**。

並且我發現不同的 **metrics function** 除了在 **ROC-AUC** 的結果上有差異之外，使用自定義的 **metrics function** 還會降低 **pairwise_distance** 的執行速度(相較於直接使用 **sklearn** 提供的 **metric** 選項)，我猜測這可能與 **sklearn** 的執行速度優化有關。

4. LOF:

在這項作業中 **LOF** 的效能是最差的，並且根據 **visualize** 的結果，我感覺 **density** 與是否為 **anomaly** 並非完全相關。