CS5658 Anomaly Detection

Homework1 – MNIST

Student ID: 109062211 Name: 張惇媛

1. K Nearest Neighbor

Implementation

```python
class KNN:
    def __init__(self, train, test, train_label, test_label):
        self.train_data = train
        self.train_label = train_label
        self.test_data = test
        self.test_label = test_label
        self.anomaly_score = np.zeros(len(test_label))
        self.distance = pairwise_distances(self.test_data, self.train_data, metric='euclidean')

    def fit(self, k):
        self.k = k
        return

    def classifier(self):
        self.anomaly_score = np.mean(np.sort(self.distance)[:,:self.k], axis=1)
        return

    def score(self):
        return roc_auc_score(self.test_label, self.anomaly_score)
```

首先，class KNN 中的 init function，定義會使用到的 training/testing data、初始化 anomaly score，並使用 pairwise_distances，計算 test_data 與 train_data 之間的 euclidean distance。接著，fit function 設定 k 值。classifier function 將每個 test_data 與其前 k 個鄰居(in training data)的距離取平均，作為各自的 anomaly score。最後，score function 以 test_label (ground_truth) 及 anomaly score，來取得 roc_auc_score。以下為實際於迴圈中的使用：

```python
# Method.1: KNN_classifier
record = []
knn = KNN(train_data, test_data, training_label, testing_label)

for k in [1, 5, 10]:
    knn.fit(k)
    knn.classifier()
    record.append(knn.score())

KNN_result.append(record)
```

Performance

```
======result of KNN======
K = 1: 0.9611
K = 5: 0.9663
K = 10: 0.9647
=========================
```

Analysis

K 屬於 hype-parameter，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。

而在此 dataset 中，K = 5 時有較好的 performance，推測是因 K = 1 時，容易受到 noise 影響，導致未見過的正常數據可能遭到誤判，而 K = 10 時，考慮到過於廣泛的區域，反而丟失了對於局部結構的區分能力，因此 K = 5 時，能夠獲得較好的 performance。

2. Cluster-based

Implementation

```
class KMeans:
    def __init__(self, train, test, train_label, test_label):
        self.train_data = train
        self.train_label = train_label
        self.test_data = test
        self.test_label = test_label
        self.anomaly_score = np.zeros(len(test_label))

    def fit(self, k):
        self.k = k
        index = np.random.choice(len(self.train_data), self.k, replace=False)
        self.centroids = np.array([self.train_data[index[i]] for i in range(self.k)])

        stop = False
        while not stop:
            stop = True
            dist = pairwise_distances(self.train_data, self.centroids, metric='euclidean')
            centroid_idx = np.argmin(dist, axis=1)

            # update centroid
            for i in np.unique(centroid_idx):
                new_centroid = np.mean(self.train_data[centroid_idx == i], axis=0)
                if not (new_centroid == self.centroids[i]).all():
                    self.centroids[i] = new_centroid
                    stop = False

    def classifier(self):
        dist = pairwise_distances(self.test_data, self.centroids, metric='euclidean')
        self.anomaly_score = np.sort(dist)[:,0]
        return

    def score(self):
        return roc_auc_score(self.test_label, self.anomaly_score)
```

首先，class KMeans 中的 init function，定義會使用到的 training/testing data、初始化 anomaly score，接著，fit function 設定 k 值，並隨機選擇 train_data 中的 k 個點，作為 centroids，根據 train_data 與 centroids 的 pairwise Euclidean distances 分成 k 個 cluster，再將各 cluster 的中心作為新的 centroids，不斷地更新 centroids 直到 centroids 不再移動，而取得 k 個 clusters 的 centroids。classifier function 將每個 test_data 與 clusters 之 centroids 取此 k 個中的最短距離，作為各自的 anomaly score。最後，score function 以 test_label (ground_truth) 及 anomaly score，來取得 roc_auc_score。

以下為實際於迴圈中的使用：

```
# Method.2: KMeans_clustering
record = []
kmeans = KMeans(train_data, test_data, training_label, testing_label)

for k in [1, 5, 10]:
    kmeans.fit(k)
    kmeans.classifier()
    record.append(kmeans.score())

KMeans_result.append(record)
```

Performance

```
=====result of KMeans====
K = 1: 0.8954
K = 5: 0.9464
K = 10: 0.9575
=========================
```

Analysis

K 同屬於 hype-parameter，當 K 太小時，會產生 underfitting，較無法代表真實分布；當 K 太大時，會產生 overfitting，對 noise 較為敏感，因此其最適值取決於 dataset。

而在此 dataset 中，K = 1 時有較差的 performance；K = 10 時有較好的 performance，推測是 K = 1 時，cluster 包含了所有的 training data，導致 cluster 內部結構較為複雜、差異大；K= 10 時，各 cluster 內部的差異小，而 cluster 之間的相似度低，所以 cluster 劃分的較為精細，獲得較好的 performance。

3. Distance-based

Implementation

```
class Distance_Based:
    def __init__(self, train, test, train_label, test_label):
        self.train_data = train
        self.train_label = train_label
        self.test_data = test
        self.test_label = test_label
        self.anomaly_score = np.zeros(len(test_label))

    def fit(self, k, function):
        self.k = k
        self.func = function
        return

    def Cos(self, p1, p2):
        return 1 - np.dot(p1, p2) / (np.linalg.norm(p1) * np.linalg.norm(p2))
```

```python
def L1(self, p1, p2):
    return np.sum(np.abs(p1 - p2))

def L2(self, p1, p2):
    return np.sqrt(np.dot((p1 - p2), (p1 - p2).T))

def L_inf(self, p1, p2):
    return np.max(np.abs(p1 - p2))

def Mahalanobis(self, p1, p2):
    return np.sqrt(np.dot(np.dot((p1 - p2), self.cov_mat), (p1 - p2).T))

def classifier(self):
    match self.func:
        case "Cosine":
            self.anomaly_score = np.sort(pairwise_distances(self.test_data, metric=self.Cos))[:, self.k]
        case "L1":
            self.anomaly_score = np.sort(pairwise_distances(self.test_data, metric=self.L1))[:, self.k]
        case "L2":
            self.anomaly_score = np.sort(pairwise_distances(self.test_data, metric=self.L2))[:, self.k]
        case "L_inf":
            self.anomaly_score = np.sort(pairwise_distances(self.test_data, metric=self.L_inf))[:, self.k]
        case "Mahalanobis":
            self.cov_mat = np.linalg.inv(np.cov(self.train_data, rowvar=False))
            self.anomaly_score = np.sort(pairwise_distances(self.test_data, metric=self.Mahalanobis))[:, self.k]
    return

def score(self):
    return roc_auc_score(self.test_label, self.anomaly_score)
```

首先，class Distanced_Based 中的 init function，定義會使用到的 training/testing data、初始化 anomaly score，接著，fit function 設定 k 值、distance 計算時的 metric function。classifier function 根據不同的 distance-based，執行對應的 metric function，下面列點出參照講義之各 metric function 的公式：

(a). Cos function 為計算 Cosince Distance，Cosine Distance 為 $1 - \frac{p1 \cdot p2}{\|p1\|\|p2\|}$ (cosine similarity)。

(b). L1 function 為計算 Minkowski Distance(r = 1)，也就是 L^1 norm distance，Minkowski Distance(r = 1) 為 $\sum_{i=1}^{n} |p1_i - p2_i|$；L2 function 為計算 Minkowski Distance(r = 2)，也就是 L^2 norm distance，Minkowski Distance(r = 2) 為 $\sqrt{\sum_{i=1}^{n} |p1_i - p2_i|^2}$；L_inf function 為計算 Minkowski Distance(r = inf)，也就是 L^{inf} norm distance，Minkowski Distance(r = inf) 為 $\max_{i=1\sim n} |p1_i - p2_i|$。

(c). Mahalanobis function 為計算 Mahalanobis Distance，Mahalanobis Distance 為 $\sqrt{(p1 - p2)S^{-1}(p1 - p2)^T}$，其中 S 為 $covariance(training\_data)$。

計算完 pairwise distances 後，將每個 test_data 與其第 k 個鄰居(in testing data)的距離，作為各自的 anomaly score。

最後，score function 以 test_label (ground_truth) 及 anomaly score，來取得 roc_auc_score。

以下為實際於迴圈中的使用：

```
# Method.3: Distance-Based
record = []
distance = Distance_Based(train_data, test_data, training_label, testing_label)

# (a) Cosine Distance
distance.fit(5, "Cosine")
distance.classifier()
record.append(distance.score())

# (b) Minkowski Distance
distance.fit(5, "L1")
distance.classifier()
record.append(distance.score())

distance.fit(5, "L2")
distance.classifier()
record.append(distance.score())

distance.fit(5, "L_inf")
distance.classifier()
record.append(distance.score())

# (c) Mahalanobis Distance
distance.fit(5, "Mahalanobis")
distance.classifier()
record.append(distance.score())

Distance_result.append(record)
```

Performance

```
========result of Distance-Based========
Cosine distance: 0.9795
Minkowski(L1 norm) distance: 0.9520
Minkowski(L2 norm) distance: 0.9558
Minkowski(L-inf norm) distance: 0.9564
Mahalanobis distance: 0.9799
========================================
```

Analysis

在 Distance-based 中使用到五種不同的 distance metric function，其中 Mahalanobis Distance 獲得較好的 performance，推測是因 Mahalanobis Distance 考慮到了各 feature 之間的關聯性，即是使用到 training data features 間的 covariance matrix，並且具有尺度不變性(scale-invariance)，也就是不受 feature 受到單位縮放的影響，使得它能夠更好的反映數據的相關性。而 Cosine Distance 之 performance 稍低於前者，此做法是以餘弦相似度基於方向相似性作為考量，適合於高維空間下表示。最後，Minkowski Distance 獲得較差的 performance，推測是因僅考慮距離相似性，而沒有考慮 feature 間的相關性。

4. Density-based

Implementation

```
class LOF:
    def __init__(self, test, test_label):
        self.test_data = test
        self.test_label = test_label
        self.num = len(test_label)
        self.lof_score = np.zeros(self.num)

    def fit(self, k):
        self.k = k
        return

    def classifier(self):
        dist = pairwise_distances(test_data, metric='euclidean')
        k_dist_mat = np.outer(np.ones(self.num), np.sort(dist, axis=0)[k])
        reach_dist = np.maximum(dist, k_dist_mat)
        sort_index = np.argsort(dist, axis=1)[:,1:k+1]

        LRD = np.zeros(self.num)
        for i in range(self.num):
            LRD[i] = 1 / np.mean(reach_dist[i, sort_index[i]])

        for i in range(self.num):
            self.lof_score[i] = np.mean(LRD[sort_index[i]] / LRD[i])
        return

    def score(self):
        return roc_auc_score(self.test_label, self.lof_score)

    def visualization(self):
        data = TSNE().fit_transform(self.test_data)

        plt.figure(figsize=(16, 5))

        plt.subplot(121)
        plt.scatter(*(data.T), c = self.lof_score, cmap = 'viridis', s = 1.8)
        plt.colorbar()
        plt.title('predicted LOF score for normal digit=0')

        plt.subplot(122)
        scatter = plt.scatter(*(data.T), c = self.test_label, s = 1.8)
        plt.legend(handles=scatter.legend_elements()[0], labels=['normal', 'anomaly'])
        plt.title('ground truth label for normal digit=0')

        plt.savefig('tsne.png')
        plt.show()

        return
```

首先，class LOF 中的 init function，定義會使用到的 testing data. num、初始化 lof score，接著，fit function 設定 k 值。classifier function 參照講義做法及公式，先計算 test_data 的 pairwise Euclidean distance，再經過 sort 得出 k_dist_mat(k-distance)，將兩者取 maximum 以獲得 reach_dist(reachability distance)，接著，使

用 np.argsort 得出 k neighbors' index，最後，取 neighbors 其 reach_dist 平均的倒數獲得 LRD (local reachability density)，lof_score 即是將 neighbors' LRD 值除以 self LRD 值的平均，作為其 anomaly score。最後，score function 以 test_label (ground_truth) 及 lof_score，來取得 roc_auc_score。

Visualization 的部分，將 test_data 經由 TSNE 降維成二維 data 來進行視覺化，左邊的 predicted LOF score 將 color 設為其 lof_score 繪製散布圖；右邊的 ground truth label 將 color 設為其 label 繪製散布圖，並標記圖例於右上角。

以下為實際於迴圈中的使用：

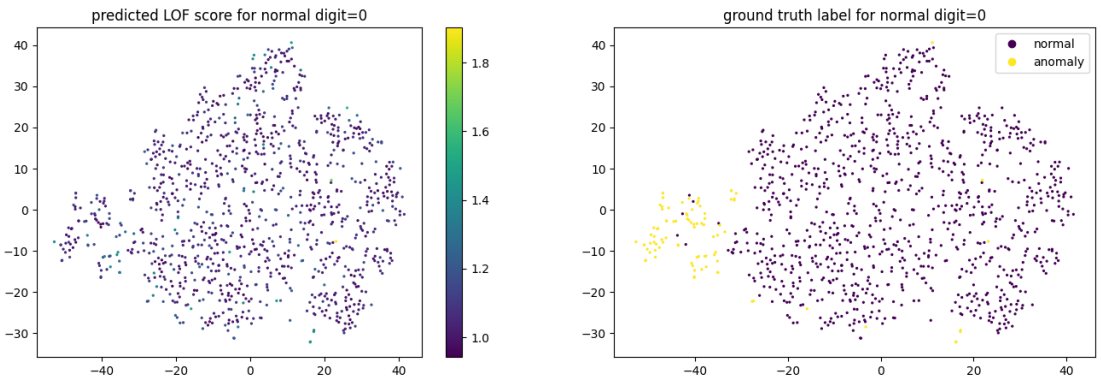
```
# Method.4: Density-Based AD
# (a) Local Outlier Factor
record = []
lof = LOF(test_data, testing_label)

lof.fit(5)
lof.classifier()
record.append(lof.score())

Density_result.append(record)

# (b) Visualize with t-SNE
if i == 0:
    lof.visualization()
```

Performance

```
========result of Density-Based========
Local Outlier Factor: 0.7930
=======================================
```

Visualization

以 normal digit = 0，作為下方 Analysis 部分的分析依據。



Analysis

在 Density-based 中使用 LOF 來進行異常檢測，其 performance 明顯低於前面的幾項做法，從 visualization result 可以看出在 ground truth 中異常數據呈現少部分聚集於一區域的現象，同時，正常數據中有部分 noise 存在較為分散的區域，也有聚集分布較為稀疏的區域，因此 LOF 以局部區域的密度作為 anomaly score，無法良好地分辨此 dataset 中的異常與正常數據。