데이터 사이언스

DBSCAN 구현.

2015004475 김태훈

1. 구동 환경

OS: Windows 10 64비트

Language: Python 3.8

2. 프로그램 구조

1. 데이터 읽어오기
2. DBSCAN 객체 생성 후 데이터 전달, 실행하여 라벨 배열 반환.
3. 클러스터들 크기로 정렬해 출력파일 n개 생성.

3. 코드 설명

clustering.py

```
def read_file(path):
    samples = []
    try:
        with open(path, 'r') as f:
            lines = f.readlines()
            lines = list(map(lambda s: s.rstrip('\n'), lines))
            for line in lines:
                samples.append(line.split('\t'))

    except FileNotFoundError as e:
        print(e)
        exit()
    except FileExistsError as e:
        print(e)
        exit()
    finally:
        f.close()

    data = np.array(samples)

    return data
```

인풋 파일 읽어 ndarray 형태로 반환

```
def sort_clusters_by_size(labels):
    cluster, counts = np.unique(labels, return_counts=True)
    cluster_count_dict = dict(zip(cluster, counts))
    try:
        del cluster_count_dict[-1]
    except KeyError:
        pass

    return sorted(cluster_count_dict.items(), key=lambda x: x[1], reverse=True)
```

클러스터 크기로 정렬.

```
def write_file(path, data):
    try:
        os.remove(path)
    except FileNotFoundError:
        pass

    with open(path, 'a') as f:
        f.writelines(data)


def write_result_files(input_file_name, n, result, labels):
    for i in range(0, min(n, len(result))):
        output_file_path = f'{input_file_name[:6]}_cluster_{i}.txt'
        output = []
        for idx, label in enumerate(labels):
            if label == result[i][0]:
                line = f'{idx}\n'
                output.append(line)

        write_file(output_file_path, output)
```

클러스터 별 파일 출력.

dbscan.py

```
UNVISITED = -2
OUTLIER = -1
```

UNVISITED로 라벨 초기화, OUTLIER 정의.

```python
class DBSCAN:
    def __init__(self, eps, min_pts):
        self.eps = eps
        self.min_pts = min_pts

        self.data = None
        self.labels = None
        self.distance_matrix = None

    def distance(self, p, q):
        return np.sum((p - q) ** 2) ** (1/2)

    def _make_distance_matrix(self):
        distance_matrix = np.zeros((self.data.shape[0], self.data.shape[0]), dtype=np.float32)
        for i in range(0, self.data.shape[0]):
            for j in range(i, self.data.shape[0]):
                distance_matrix[i][j] = distance_matrix[j][i] = np.sum((self.data[i] - self.data[j]) ** 2)

        return distance_matrix ** (1/2)
```

class DBSCAN

eps: 입실론

min_pts: MinPts, 코어 포인트 조건.

data: 좌표 인풋 데이터.

labels: 각 오브젝트의 클러스터 라벨 모음.

distance_matrix: 오브젝트 간 distance 배열.

distance(): 점 간 유클리드 거리 반환.

_make_distance_matrix(): 모든 점 간 유클리드 거리 연산해 배열에 저장.

```python
    def is_neighborhood(self, p, q):
        return self.distance_matrix[p][q] < self.eps

    # 모든 점에 대하여 타겟 포인트와 이웃인 것 찾기. 자신도 포함.
    def _find_neighbors(self, point_id):
        neighbors = []
        for i in range(0, self.data.shape[0]):
            if self.is_neighborhood(point_id, i):
                neighbors.append(i)

        return neighbors
```

is_neighborhood(): 주어진 점이 서로 이웃인지 거리표와 입실론을 비교하여 T/F 반환.

_find_neighbors(): 지정된 포인트의 이웃을 모두 찾아 반환.

```python
    def _make_cluster(self, point_id, cluster_id):
        # 타겟 포인트가 코어포인트가 아니면 일단 아웃라이어.
        neighbors = self._find_neighbors(point_id)
        if len(neighbors) < self.min_pts:
            self.labels[point_id] = OUTLIER

            return False

        # 고른게 코어면 클러스터 생성.
        else:
            print(f'{cluster_id}번 클러스터 생성 중')

            self.labels[point_id] = cluster_id
            # 코어 주변 이웃들 클러스터에 편입.
            for neighbor_id in neighbors:
                self.labels[neighbor_id] = cluster_id

            # 클러스터 확장 여부 판단.
            while len(neighbors) > 0:
                new_point = neighbors[0]
                new_neighbors = self._find_neighbors(new_point)  # _find_neighbors로 받아온 이웃 리스트는 이웃의 id로 이루어짐.

                # 이웃 포인트에 포커스, 코어 포인트면...
                if len(new_neighbors) >= self.min_pts:
                    for new_neighbor in new_neighbors:
                        # 새 포인트의 이웃이 미방문: 클러스터에 편입하며 코어 검사 (reachable 쭉 이어감).
                        # 아웃라이어: 이미 방문했는데 코어가 아니었던 포인트.
                        if self.labels[new_neighbor] == UNVISITED or self.labels[new_neighbor] == OUTLIER:
                            if self.labels[new_neighbor] == UNVISITED:
                                neighbors.append(new_neighbor)

                            self.labels[new_neighbor] = cluster_id

                neighbors = neighbors[1:]

            return True
```

_make_cluster(): 클러스터 생성

포인트 id를 받아 해당 포인트의 이웃을 찾고, 코어 포인트 기준을 만족하는지 확인.

코어가 아니라면 일단 아웃라이어로 라벨 후 종료.

주어진 포인트가 코어 포인트라면 주변 이웃과 자신을 주어진 클러스터 라벨로 표기.

이후 이웃 중 하나를 지정해 새 주시 포인트로 만들어 그 이웃을 구함.

이웃 수가 minpts를 만족해 코어 포인트가 되면 그 이웃들이 미방문 혹은 아웃라이어 일 때 클러스터에 편입 시킴. 이웃이 미방문 상태였다면 해당 포인트는 코어가 될지도 모르는 포인트이므로 검사를 위해 순회중인 리스트에 추가.

클러스터가 완성되면 True 반환.

```
    def get_labels(self):
        return self.labels

    def run(self, data: np.ndarray):
        # id 제거
        print('데이터 가공')
        self.data = data[:, 1:].astype(np.float32)

        # 오브젝트간 거리 행렬 연산.
        print('거리 연산')
        self.distance_matrix = self._make_distance_matrix()

        # 모든 점의 라벨 UNVISITED로 초기화.
        print('라벨 초기화')
        self.labels = np.full(self.data.shape[0], UNVISITED)

        cluster_id = 0
        # 모든 점을 순회
        for i in range(0, self.data.shape[0]):
            if self.labels[i] == UNVISITED:
                # 고른 포인트가 코어포인트라서 클러스터를 만들면 작업 후 다음 클러스터 아이디 증가.
                if self._make_cluster(i, cluster_id):
                    cluster_id = cluster_id + 1

        return self.get_labels()
```

get_labels(): labels accessor

run(): dbscan 작동. 인풋 데이터에서 id 부분 제거, distance matrix 생성, label 배열 생성.

모든 점 순회하며 미방문이면 _make_cluster 함수 호출. True를 받았으면(클러스터가 생성되었으면) 다음 클러스터 id를 +1. labels를 반환.

4. 소스 컴파일 가이드

python 3.8로 작성된 스크립트이며, 추가로 필요한 패키지는 프로젝트 패키지에 포함된 requirements.txt에 기록되어 있음.

clustering.py, dbscan.py와 인풋파일들은 같은 디렉토리에 존재해야함.