DBSCAN

Environment

OS: Mac OS, Windows 10

Language: Python 3.9

Run with the command

```
// In the DBSCAN folder.

python clustering.py [trainset_file_path] [#_of_clusters] [Eps] [MinPts]

Ex) python clustering.py ./data-2/input1.txt 8 15 22
```

n: number of clusters for the corresponding input data

Eps: maximum radius of the neighborhood

MinPts: minimum number of points in an Eps-neighborhood of a given point

Train set format

[object_id_1]\t[x_coordinate]\t[y_coordinate]\n

[object_id_2]\t[x_coordinate]\t[y_coordinate]\n

[object_id_3]\t[x_coordinate]\t[y_coordinate]\n

[object_id_4]\t[x_coordinate]\t[y_coordinate]\n

...

- Each row is an object.
- All the attributes are numerical.
- *object_id_i* is an identifier of the i-th object
- *(x coordinate, y coordinate)* is the location of the corresponding object in the 2-dimensional space.

Ex)

```
0 84.768997 33.368999
1 569.791016  55.458
2 657.622986  47.035
3 217.057007  362.065002
4 131.723999  353.368988
5 146.774994  77.421997
...
```

Output format

- input#_cluster_0.txt

[object_id]\n

[object_id]\n

…

- input#_cluster_1.txt

[object_id]\n

[object_id]\n

…

- input#_cluster_n-1.txt

[object_id]\n

[object_id]\n

…

- ‘input#_cluster_i.txt’ should contain all the ids belonging to cluster ‘i’ that were obtained by using the DBSCAN algorithm.

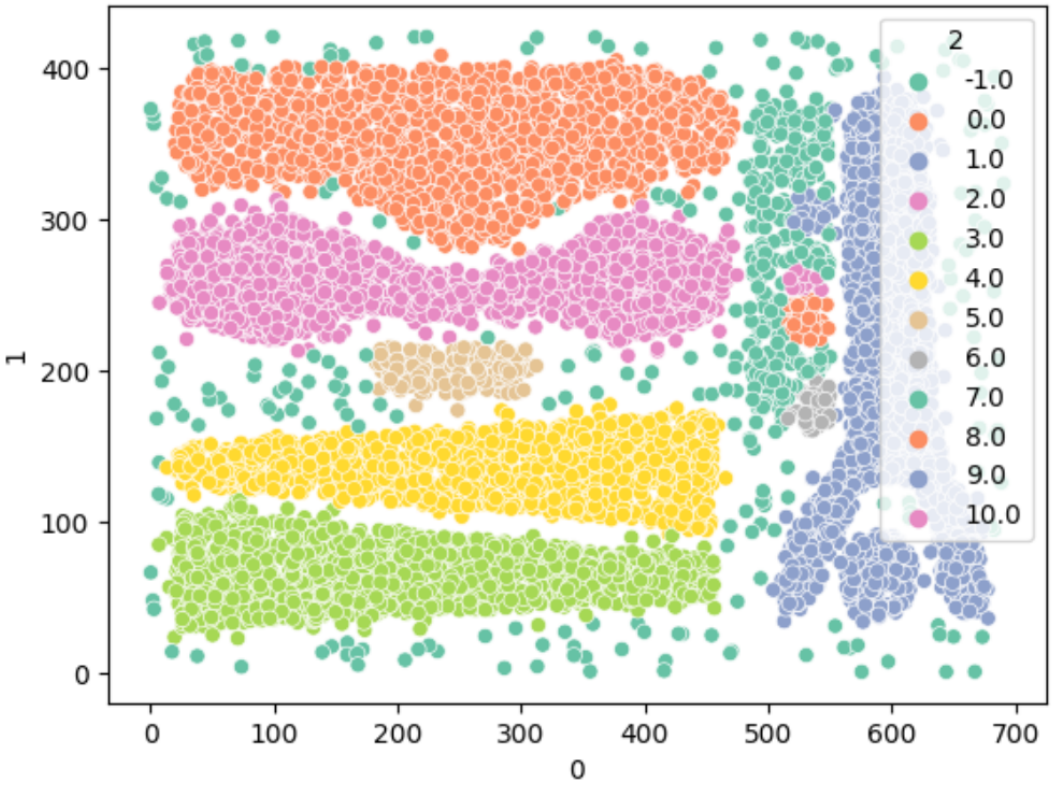
Ex) 'input#_cluster_i.txt'

```
1
2
15
17
22
30
47
...
```

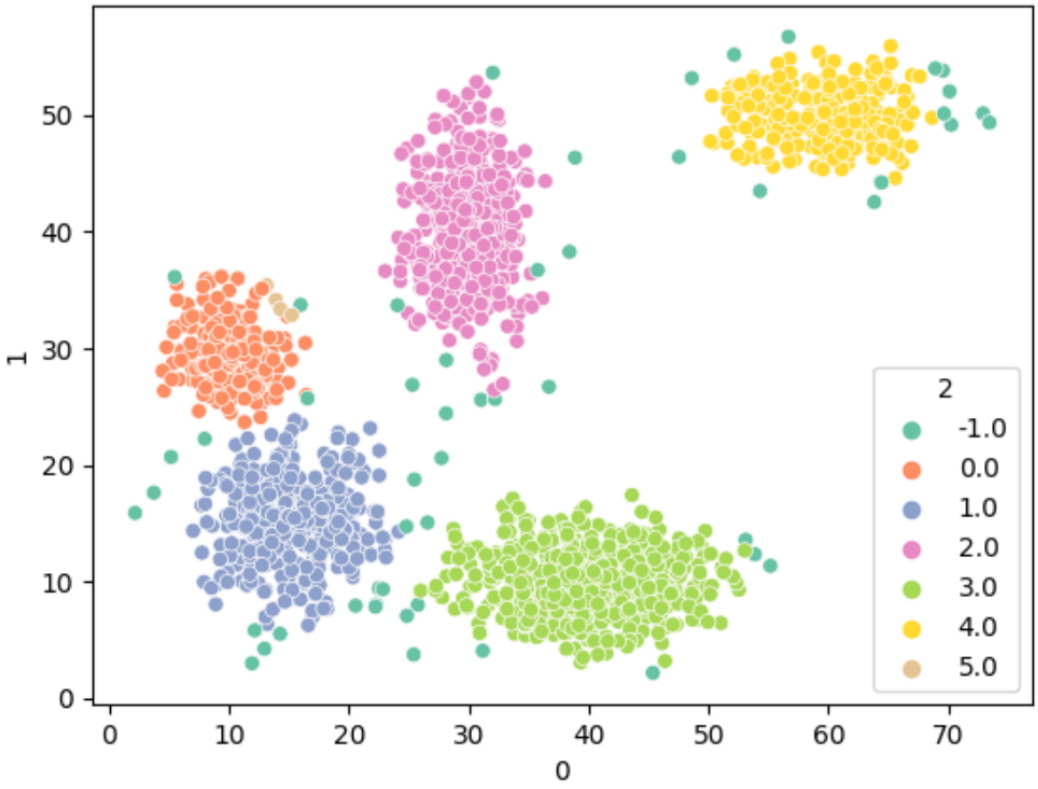
Desired output

NOTE: cluster named “-1” is an outlier cluster

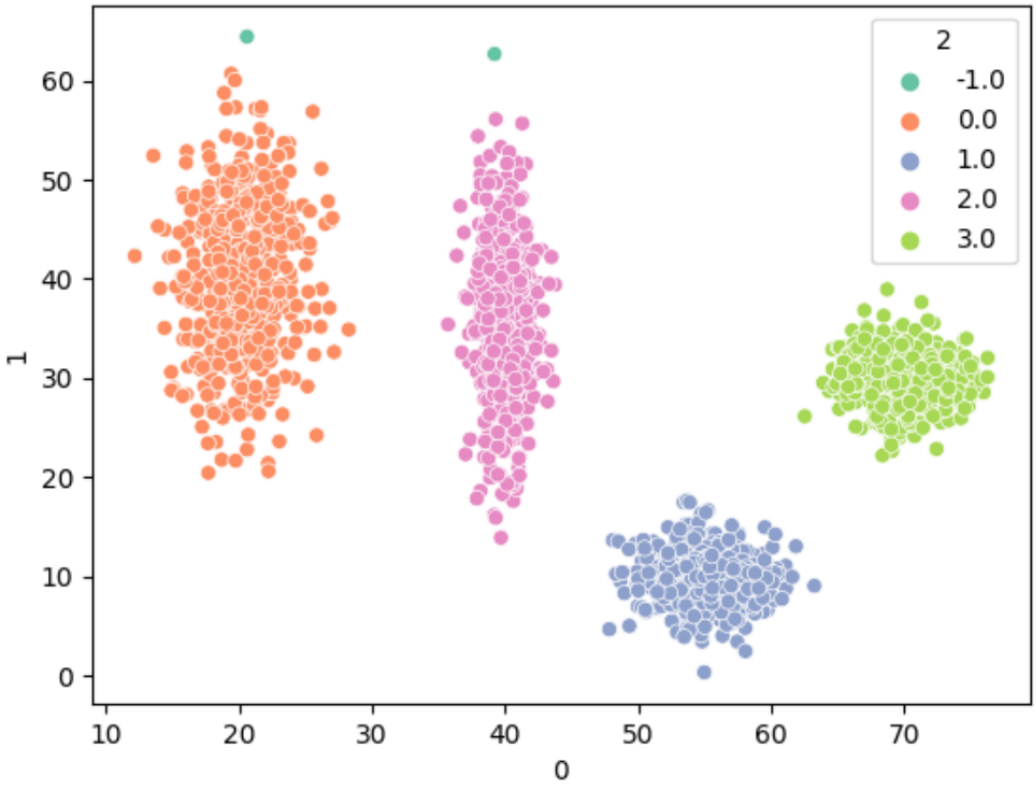
input1.txt, n=8, Eps=15, MinPts=22



input2.txt, n=5, Eps=2, MinPts=7



input3.txt, n=4, Eps=5, MinPts=22



How it works

Distance function

```
from sklearn.metrics import pairwise_distances

def _distance(a, b):
    distances = pairwise_distances(a, b)
    return distances
```

With given points, calculate euclidian distances.

Ex)

```
a = np.array([[1,2], [1,3]])
b = np.array([[1,2], [1,3], [4,6]])
_distance(a, b)

>> array([[0.        , 1.        , 5.        ],
          [1.        , 0.        , 4.24264069]])
```

Getting directly reachable points

```
def _get_directly_reachable_points(data, point, eps):
    '''
    :param data: pandas dataframe
    :param point: core point candidate
```

```
    :param eps: given eps(radius)
    :return: indices of all directly reachable point from candidate
    '''
    directly_reachable_idx = np.where(_distance(data, point) < eps)[0]
    #  eliminating self
    directly_reachable_idx = np.delete(directly_reachable_idx,
                                       np.argwhere(directly_reachable_idx == point.index.values))
    return directly_reachable_idx
```

With given data and core point candidate, get all possible directly reachable points.

After eliminating index of itself from the result, return it.

DBSCAN

```
def db_scan(data, eps, min_pts):
    current_cluster_number = 0

    label_vector = np.full(data.shape[0], np.NaN)
    label_vector = label_vector.reshape(-1, 1)

    while True:
        nan_mask = np.isnan(label_vector)
        nan_idx = np.where(nan_mask == True)[0]

        if len(nan_idx) == 0:
            break

        core_candidate_idx = np.random.choice(nan_idx, 1)
        core_candidate = data.iloc[core_candidate_idx]

        directly_reachable_points_idx = _get_directly_reachable_points(data, core_candidate, eps)

        if len(directly_reachable_points_idx) < min_pts:
            label_vector[core_candidate_idx] = -1.0  # means noise / outlier
            continue
        else:
            label_vector[core_candidate_idx] = current_cluster_number

        while len(directly_reachable_points_idx) > 0:
            next_candidate_idx = np.random.choice(directly_reachable_points_idx, 1)
            label_vector[next_candidate_idx] = current_cluster_number

            next_candidate = data.iloc[next_candidate_idx]

            next_candidate_directly_reachable_points_idx = _get_directly_reachable_points(data, next_candidate, eps)
            if len(next_candidate_directly_reachable_points_idx) >= min_pts:
                directly_reachable_points_idx = np.union1d(directly_reachable_points_idx,
                                                           next_candidate_directly_reachable_points_idx)

            condition1 = np.isnan(label_vector[directly_reachable_points_idx])
            condition1 = condition1.squeeze()
            condition2 = label_vector[directly_reachable_points_idx] == -1
            condition2 = condition2.squeeze()

            directly_reachable_points_idx = directly_reachable_points_idx[condition1 | condition2]

        current_cluster_number += 1

    data_labeled = pd.DataFrame(np.concatenate((data.values, label_vector), axis=1))
    return data_labeled
```

1. Select a core candidate from the not visited points *(i.e. nan_idx).*
2. With the core candidate, get all directly reachable points.
3. If the number of directly reachable points is less than MinPts, label the current core candidate as noise and go back to step 1. If not, label it as the current cluster.

4. If there are unvisited directly reachable points, draw a next core candidate from it. If not, go back to step 1
    a. label the next core candidate as the current cluster.
    b. With it, get all directly reachable points.
    c. If the number of directly reachable points from it is more than MinPts, union the points with the directly reachable points from step 2.
    d. Eliminate visited and outlier points from the union.
    e. Go back to step 4.

Writing the result to the disk

```
def write_result_to_disk(result_df, input_file_path, n_of_clusters):
    target_cluster_number = result_df[2].nunique() - n_of_clusters - 1

    clusters = result_df.groupby([2]).size()

    while target_cluster_number > 0:
        min_label_idx = clusters.argmin()
        min_label = clusters.index[min_label_idx]

        if min_label == -1:
            clusters.pop(min_label)
            min_label_idx = clusters.argmin()
            min_label = clusters.index[min_label_idx]

        result_df.loc[result_df[2] == min_label, 2] = -1
        target_cluster_number = result_df[2].nunique() - n_of_clusters - 1
        clusters = result_df.groupby([2]).size()

    basic_path = "./test-2/"
    for idx, num in enumerate(clusters.index):
        if num == -1:
            continue
        target = result_df[result_df[2] == num].index
        target = target + 1
        if idx == 0:  # means no outlier
            idx = idx + 1
        result_file_path = basic_path + "input" + input_file_path[-5] + f"_cluster_{idx - 1}.txt"
        with open(result_file_path, 'w') as file:
            for line in target:
                file.write(f"{line}\n")
```

If the DBSCAN algorithm finds m clusters for an input data and m is greater than n (n = the number of clusters given), remove (m-n) clusters based on the number of objects within each cluster.

In order to remove (m-n) clusters, select (m-n) clusters with small sizes in ascending order.

This program saves the outputs in the “test-2” folder.