Implementation of DBSCAN

2017029807 성창호

Index

1. Introduction

This program is implemented as DBSCAN, an algorithm that clusters a given data. After running the DBSCAN algorithm, the program outputs `n` clusters with the highest number of data.

We applied heuristic to enhance the performance of DBSCAN algorithm, and implemented it as python to visualize the results.

2. Algorithm

DBSCAN is a data clustering algorithm that is density-based spatial clustering of application with noise. The brief algorithm is as follows:

1. Arbitrary select a point p in the entire set D.
2. If p is core point, a cluter is formed.
3. If p is a border point, no points are density-reachable from p and DBSCAN visits the next point of the entire set D.
4. Continue the process until all of the points have been processed.

In order to determine whether any point p is a core point, an adjacency list was made by determining whether all pairs present in D are neighbors. Any two point p_1 and p_2 being neighbors is density-reachable. To determine if it is a neighbor, compare the distance between p_1 and p_2, and Eps received as input. The size of the list corresponding to each point in the adjacency list made on this basis is the basis for the core point. The above algorithm is implemented through Breadth-First Search based on the previously created adjacency list.

To improve performance, heuristic is applied when creating an adjacency list. After sorting the entered set D in coordinate order, the run was stopped if the Manhattan distance between the two points was greater than eps when the double loop was executed.

All of these processes are implemented in the `DBSCAN` class.

3. Usage

```
# install packages using pip
$ pip3 install -r requirements.txt

# Run (Example)
$ python clustering.py \
./data/input3.txt \
4 \
5 \
5 \
--output_path ./test \

# Run (Help)
usage: clustering.py [-h] [--output_path OUTPUT_PATH] [--img IMG]
                     [--remove_noise REMOVE_NOISE]
                     input n Eps Minpts

positional arguments:
  input                 Input data file name
  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

optional arguments:
  -h, --help            show this help message and exit
  --output_path OUTPUT_PATH
                        Output file path
  --img IMG             Save input & cluster image (only ipython)
  --remove_noise REMOVE_NOISE
                        Remove noises from cluster image (only ipython)
```

4. Implementation

We implemented this program by divding it into `Input`, `Output` methods and `DBSCAN` class.

`Input`, `Ouput` method

In the `main` function, the arguments inputed as `argv` are formatted and created objects for each class. The execution time is also measured to verify the performance of this program.

In `Input` method, set D is inputed from the input file through the `pandas` library. D is then sorted in a coordinate order to apply the aforementioned heuristic. In `Output` method, save `n` clusters with the largest number of objects to fils.

`DBSCAN` class

`check_neighbor` method

Take indexes `i` and `j` for two objects as factors to determine whether they are density-reachable.

`make_adj_list` method

Generate adjacency list with density-reachable object pairs.

`is_core` method

The index `i` for any object is taken as a factor to determine whether the object is a core point.

`bfs` method

Take the core point `now` as a factor and create a cluster with Breadth-First Search as the center.

`cluster` method

Call `bfs` method that creates the cluster around a point that is not yet part of the cluster or is a border point. And create labels with the clusters that are created.

5. Experiments

Running Environment

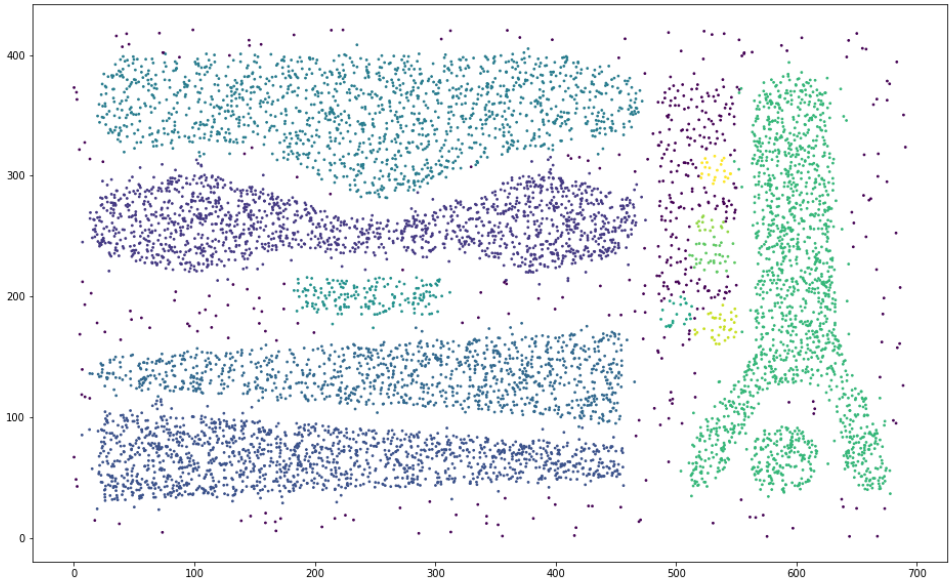
- OS : Ubuntu 16.04.6 (WSL)
- CPU : Intel(R) Core(TM) i7-7660U CPU 250Hz
- RAM : 16GB
- Language : G++ 5.4.0

Results

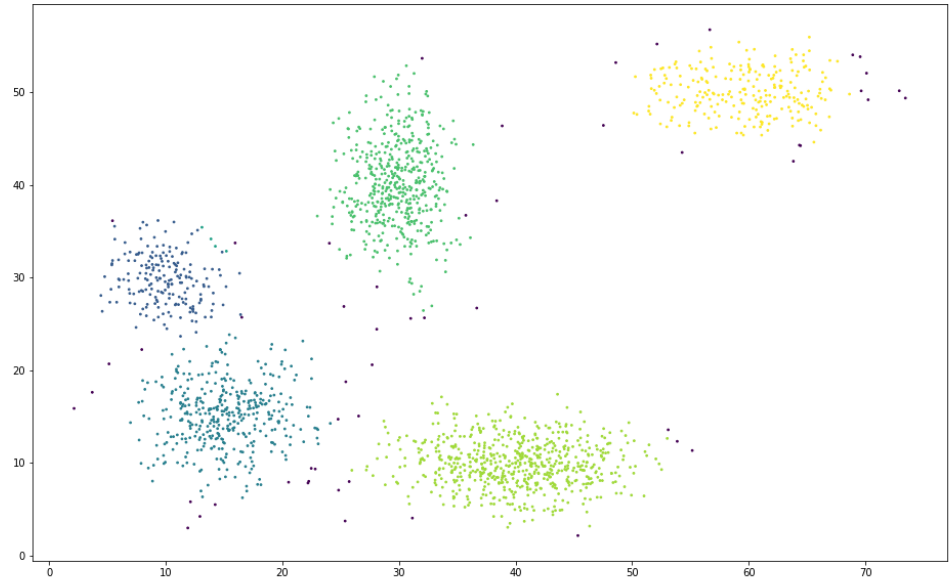
Input file	n	Eps	$Minpts$	`PA3` score	Running Time
input1.txt	8	15	22	98.93552	12.97358sec
input2.txt	5	2	7	94.89474	1.26602sec
input3.txt	4	5	5	99.97736	4.85177sec

Result Image

input1.txt



input2.txt



input3.txt