Datascience Asssignment #3 - DBSCAN

2016025241 김영진

Summary of algorithm

1. 주어진 데이터 포인트들을 Geocoding을 이용해 가까운 것들끼리 클러스터링 한다.
2. 1번에서 얻은 클러스터를 토대로, 근처의 데이터포인트들 중, eps보다 가까운 것들을 모두 찾아, 서로 이웃인 것으로 정의한다.
3. 이웃을 min_pts보다 많이 가지고 있는 포인트는 코어로 정의한다.
4. 전달받은 데이터 포인트 순서대로,
    - 해당 포인트가 코어라면, 이웃인 코어들과 같은 그룹으로 묶는다.
    - 해당 포인터가 코어가 아니라면, 이웃인 코어들 중 가장 먼저 입력으로 들어온 포인트와 같은 그룹으로 묶는다.
        - 만약 이웃인 코어가 없다면 해당 포인트를 노이즈로 정의한다.
5. 데이터 포인트의 그룹들을 크기가 큰 순으로 정렬한다.

Detailed description of code

Geocoding

```
class Geocoding:
    '''
    Assume N-dim euclid space is quantumized by certain length (`box_size`),
    and each points in space are mapped into certain quantumized hyper-cube
    many-to-one. Points in same hyper-cube satisfy locality, and we can easily
    find neighbor hyper-cube.
    '''
    def __init__(self, data: np.array, box_size: float):
        '''
        Calculate mapper (`geocode unitbox index` => `item indices`)
        '''
        self.data = data
        assert(len(data.shape) == 2)
        self.n = data.shape[0]
        self.dim = data.shape[-1]
        self.box_size = box_size

        base = Geocoding._base(self.data, self.box_size)
        hs = [Geocoding._hash(x) for x in base]
        mapper = defaultdict(list)
        for index, h in enumerate(hs):
            mapper[h].append(index)
        self.mapper = mapper
        self.hs = hs
```

```
    ...
```

Geocoding 클래스는 주어진 `data`들을 `box_size`로 나눈 후에 floor연산을 취한 결과(`Geocoding._base`)가 같은 포인트들 같은 클러스터로 묶는다. 이 때 이 `_base` 값을 해당 클러스터의 고유벡터라고 한다.

이 때, `box_size`를 DBSCAN에서 `eps`와 동일하게 설정했을 때, 같거나 인접한 클러스터 안에 L2 거리가 eps보다 작은 포인트가 모두 존재함이 자명하다.

여기서 클러스터끼리 인접하다는 것은 두 클러스터들의 고유벡터 간의 차이의 각 요소의 절대값이 1을 넘지 않는 것을 의미한다.

```
def neighbors(self, point: np.array, rg: int = 0):
    '''Find geocode neighbors'''
    base_from = Geocoding._base(point, self.box_size) - rg # 인접 클러스터의 고유 벡터들중 가장 낮은 값
    diam = (2 * rg) + 1
    iterator = np.array(list(np.ndindex((diam,) * self.dim)))
    bs = iterator + base_from # 각 인접 클러스터들의 고유 벡터
    hs = [Geocoding._hash(x) for x in bs] # 고유 벡터를 해시
    ds = [self.mapper[x] for x in hs] # 인접한 클러스터들의 포인트들의 리스트
    return np.int64(np.concatenate(ds))
```

위와 같이 어떤 포인트가 주어졌을 때, 인접한 클러스터의 포인트들을 모두 찾아 반환할 수 있다.

Union Find

```
class UnionFind:
    def __init__(self, n: int):
        self.n = n
        self.par = np.arange(n)

    def root(self, x: int):
        if self.par[x] == x:
            return x
        pp = self.root(self.par[x])
        self.par[x] = pp
        return pp

    def merge(self, a: int, b: int):
        pa = self.root(a)
        pb = self.root(b)
        if pa == pb: return False
        self.par[pb] = pa
        return True
```

Union find는 n개의 요소들이 동적으로 같은 그룹으로서 병합되는 것을 구현한 자료구조이다.

각 요소는 처음 상태에서 독립적인 그룹으로 존재하지만, `merge`를 통해 두 요소를 병합하면, 두 요소의 그룹은 같은 그룹으로 합쳐진다.

각 그룹은 요소는 자신의 부모를 가지며, 따라서 각 그룹은 트리 형태를 가지고 있다. 만약 두 요소가 포함된 트리의 루트가 같다면 두 요소는 같은 그룹에 포함된 것으로 볼 수 있다.

DBScan

```
def dbscan(
    items: List[DBScanItem],
    eps: float,
    min_pts: int,
    fn_distance: Callable[[np.array, np.array], float] = l2_distance,
    **kwargs,
) -> List[List[DBScanItem]]:
    ...
```

DBScan의 인자는 위와 같다.

```
sz = len(items)
arr = np.array([x.data for x in items], dtype=np.float64)

geo = Geocoding(arr, eps)
neighbors = [
    [x for x in geo.neighbors(arr[i], rg=1) # Geocoding을 통해 구한 적당히 가까운 포인트들
        if fn_distance(arr[i] - arr[x]) <= eps # 거리가 eps 이하면 이웃
    ]
    for i in range(sz)
]
is_core = np.array([len(i) for i in neighbors]) >= min_pts # 이웃이 min_pts 보다 같거나 많으면 코어 포인트
is_noise = np.zeros(sz)
```

입력받은 포인트들을 Geocoding을 이용해 가까운 것들 끼리 클러스터링 하여 이웃 포인트들을 모두 찾아놓는다.

이웃 포인트들의 개수가 `min_pts` 이상인 것들은 이미 `core` 인 것으로 찾아놓는다.

```
uf = UnionFind(sz)
for p in range(sz):
    if is_core[p]: # 코어라면
        for q in neighbors[p]: # 이웃들 중 코어를 모두 병합
            if is_core[q]:
                uf.merge(p, q)
    else: # 코어가 아니라면
        core_neighbors = [q for q in neighbors[p] if is_core[q]]
        if len(core_neighbors) == 0: # 이웃들 중에도 코어가 없다면
            is_noise[p] = 1 # 노이즈로 표시
        else: # 이웃들 중 가장 먼저 인풋으로 받은 코어와 병합
            uf.merge(p, min(core_neighbors))
```

입력으로 들어온 순서대로 처리하여 코어는 이웃 코어들 끼리 병합하고, 코어가 아닌 포인트들은 이웃들 중 가장 먼저 들어온 코어와 병합한다. 둘 중 아무것도 수행할 수 없을 경우 노이즈.

```
clusters: DefaultDict[int, List[int]] = defaultdict(list)
for i in range(sz):
    if is_noise[i]: continue
    clusters[uf.root(i)].append(i)
clusters = [[items[i] for i in c] for c in clusters.values()]
clusters = sorted(clusters, key=lambda x: -len(x)) # 크기 순으로 정렬
```

UnionFind를 통해 자신이 속한 트리의 부모가 같은 것들끼리 (같은 그룹끼리) 묶는다.

Instructions for compiling

Python dependencies

- numpy

How to run

```
./clustering.py input_filename n eps min_pts
# example: ./clustering.py input1.txt 8 15 22
# example: ./clustering.py input2.txt 5 2 7
# example: ./clustering.py input3.txt 4 5 5
```

Usage

```
usage: clustering [-h] input n eps min_pts

positional arguments:
  input       input filename
  n           the maximum number of clusters to emit
  eps         maximum distance of neighbor
  min_pts     the minimum number of neighbors of core

optional arguments:
  -h, --help  show this help message and exit
```

Other specifications

None