Summary algorithm

DBSCAN algorthim 을 class로 만들어 eps, minpts 등의 정보를 저장할 수 있도록 하였습니다.
Class 안에 label list를 만들어 cluster_id를 저장할 수 있도록 하였습니다.
첫번째 점부터 마지막 점까지 돌면서 이웃을 찾고 outlier 라면 label에 -1을 저장 아니라면 cluster_id를 저장하였습니다.
현재 점의 이웃들을 돌면서 이웃의 이웃을 찾아 cluster를 확장해가는 식으로 구현하였습니다.
UNCLASSFIED =999, OUTLIER = -1 로 설정하였습니다.

Detail Each function

프로그램 실행 시 시작되는 메인 함수

```
def main():
    argv = sys.argv

    if len(argv) == 5:
        input_file = argv[1]
        n = int(argv[2])
        eps = int(argv[3])
        minpts = int(argv[4])
    else:
        print("usage : python3", argv[0].split('/')[-1], "<cluster_num> <eps> <min_pts>")
        exit(0)

    file_number = input_file.split('.')[0][5:]
    output_file_format = 'input' + file_number + '_cluster_{}.txt'

    input_data = pd.read_csv(input_file, sep='\t', header=None).values
    model = DBSCAN(input_data, eps, minpts)
    model.fit()

    clusters = sorted(model.cluster.keys(), key=lambda x: len(model.cluster[x]), reverse=True)[:n]
    for num, c in enumerate(clusters):
        output_file_name = output_file_format.format(num)
        with open(output_file_name, 'w') as f:
            f.write('\n'.join(map(str, model.cluster[c])))
```

DBSCAN 으로 data를 clustering 한 다음 cluster들을 크기순으로 정렬하여 입력받은 N개 만큼 파일의 저장합니다.

DBSCAN class 를 초기화하는 method

```
def __init__(self, data, eps, minpts):
    self.data = data
    self.label = [UNCLASSFIED] * len(data)

    self.eps = eps
    self.minpts = minpts

    self.cluster_num = 0
    self.size = len(data)

    self.cluster = defaultdict(list)
```

입력받은 변수들을 저장하고 label 변수를 만들어 index가 어떤 cluster로 분류되었는지 저장합니다.
Cluster 변수는 해당 cluster의 포함된 index들을 저장하여 파일 저장에 용이하도록 합니다.

Data를 학습시키는 method

```
def fit(self):
    for idx in range(self.size):
        if self.label[idx] == UNCLASSFIED:
            if self.expand_cluster(idx):
                self.cluster_num += 1

    for idx, value in enumerate(self.label):
        self.cluster[value].append(idx)

    self.cluster.pop(OUTLIER)
```

Data내에 모든 점들을 돌면서 아직 분류되지 않은 점이라면 새로운 cluster를 만들어 확장시킵니다.
분류를 마친후 cluster 변수의 label들을 저장하여 주고 outlier를 제거해줍니다.
만약 클러스터가 확장되었다면 클러스터 번호를 하나 증가시켜줍니다.

현재 점의 이웃을 구하는 method

```
def get_neighbor(self, idx):
    neighbors = set()
    for datum in self.data:
        if np.linalg.norm(self.data[idx][1:] - datum[1:]) ⩽ self.eps:
            neighbors.add(datum[0])

    return neighbors
```

자기 자신을 포함하는 모든 점을 돌면서 거리가 eps보다 작은 점들을 neighbor의 추가해 반환해줍니다.

클러스터를 확장시키는 method

```
def expand_cluster(self, idx):
    neighbors = self.get_neighbor(idx)
    if len(neighbors) < self.minpts:
        self.label[idx] = OUTLIER
        return False
    else:
        for neighbor in map(int, neighbors):
            self.label[neighbor] = self.cluster_num
        while len(neighbors) > 0:
            neighbor = int(neighbors.pop())
            sub_neighbors = self.get_neighbor(neighbor)
            if len(sub_neighbors) ⩾ self.minpts:
                for sub_neighbor in map(int, sub_neighbors):
                    if self.label[sub_neighbor] == UNCLASSFIED or self.label[sub_neighbor] == OUTLIER:
                        if self.label[sub_neighbor] == UNCLASSFIED:
                            neighbors.add(sub_neighbor)
                        self.label[sub_neighbor] = self.cluster_num
        return True
```

현재 점의 이웃을 구하고 이웃의 수가 minpts보다 적으면 False를 반환합니다.
이웃들을 같은 클러스터로 표시해주고
이웃의 이웃을 돌면서 클러스터를 확장해갑니다.

Instruction for compile

제출한 source code는 .py 형태이므로 python3 명령어를 이용해 실행하면 됩니다.
Ex) python3 clustering.py input1.txt 8 15 22