Project:
K-Means Algorithm

Introduction to Machine Learning

Team 2 Mark
Youngjin Seoh
Semin Na
Yebin Pyun

2024 Spring, Seoul National University

Table of Contents

Task 0: Implementing the K-Means Algorithm

- Basic libraries

```
# fundamental libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random as rd    # used to generate random initial centroids in later simulations
```

- Euclid distance function

```
def distance(P1, P2):
  distance = np.linalg.norm(np.array(P1)-np.array(P2), 2)
  return distance
```

Task 0: Implementing the K-Means Algorithm

- Introducing K_means(): the inputs

```
def K_means(dataset, initial_centroids, max_of_iterations, mode):
```

1) **dataset:** the 2-dimensional array of P points consisting the reference data (array size = Px2)
2) **initial_centroids:** the 2-dimensional array of K initial centroid points (array size = Kx2)
3) **max_of_iterations:** the maximum number of iterations
4) **mode:** setting options for the output of K_means()

    mode == 1: output = an array of final centroid points

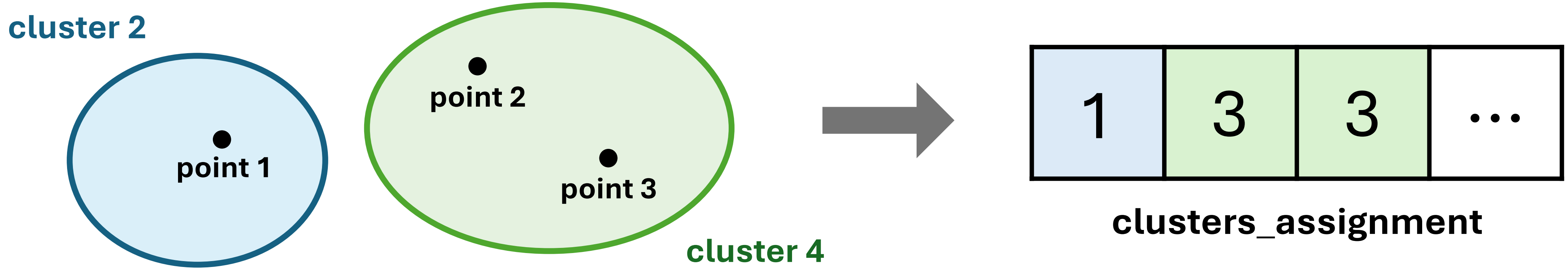
    mode == 2: output = an array of each point's cluster assignment (also used for color assignments)

    ex) If point 1 is assigned to cluster 3, then cluster_assignment[0] = 2.

Task 0: Implementing the K-Means Algorithm

- **Step 1:** Define & calculate fundamental variables

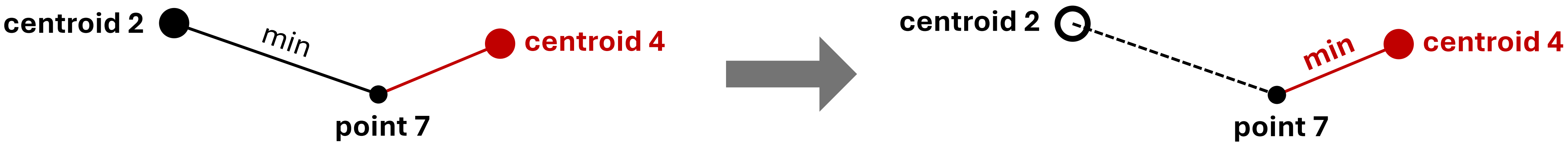
```
# Set of new centroids
new_centroids = initial_centroids      # initialize the centroid points
number_of_points = len(dataset)     # len(array): number of elements in the array
number_of_clusters = len(initial_centroids)

# Index set for clusters where the points belong
clusters_assignment = [0 for p in range(number_of_points)]
```



Task 0: Implementing the K-Means Algorithm

- **Step 2:** Iteration starts / Update cluster assignments

```
for j in range(max_of_iterations):
  # Update cluster assignments
  for p in range(number_of_points):  # iterations for finding the minimum distance for each data points
    min = distance(dataset[p], new_centroids[0])
    for k in range(number_of_clusters):  # for the pth data point, find the nearest centroid(1~k)
      a = distance(dataset[p], new_centroids[k])
      if a <= min:
        min = a                          # if the distance is smaller than the minimum so far,
        clusters_assignment[p] = k         replace it and update the cluster assignment for pth data point
```



clusters_assignment[6] = 1

clusters_assignment[6] = 3

Task 0: Implementing the K-Means Algorithm

- **Step 3:** Update centroid locations based on cluster changes

```
# Update centroid locations
for k in range(number_of_clusters):
  size = 0    # temporary variable indicating the (new) size of kth cluster
  sum = np.array([0, 0])
  centroid = np.array([0, 0])  # temporary variable indicating the (new) centroid of kth cluster
  for p in range(number_of_points):
    if clusters_assignment[p] == k:   # if the pth point belongs to kth cluster,
      size = size + 1                  increase the size of the cluster
      sum = sum + dataset[p]           and re-locate the centroid by calculating the means of data pts
  if size != 0:
    centroid = sum / size
    #delta = delta + np.linalg.norm(new_centroids[k]-centroid, 2)
    new_centroids[k] = centroid    # if the new cluster is valid, replace the centroid into new one
```

Task 0: Implementing the K-Means Algorithm

- **Step 3:** Update centroid locations based on cluster changes

for kth cluster,

size != 0	**size == 0**
the new cluster contains at least one data points	the new cluster does not contain any data points
then the centroid is re-calculated: **new_centroids[k] = centroid**	then the centroid does not change, still **new_centroids[k]**

cf) The initial values of new_centroids[k] are same as initial_centroids[k], which was set in **Step 1.**

Task 0: Implementing the K-Means Algorithm

- **Step 4:** Apply the stopping criterion / End of the Iteration

```
# variable for stopping criterion
delta = 0
```

```
# Update centroid locations
for k in range(number_of_clusters):

    (codes in Step 3)
                                        # delta = sum of each centroid's displacement in a single iteration
  delta = delta + np.linalg.norm(new_centroids[k]-centroid, 2)
```

```
print(delta)
if delta < 1e-3:
  print('the number of iterations:', j+1)    # if delta is small enough, the iteration will stop(=break)
  break                                        after printing the number of iterations progressed
```

- Stopping criterion for the iterations: **Δ = sum of centroid displacements < 10^{-3}.**

Task 0: Implementing the K-Means Algorithm

- **Step 5:** Update cluster assignments based on final centroid changes

```
# Update cluster assignments using final centroids
for p in range(number_of_points):
  min = distance(dataset[p], new_centroids[0])
  for k in range(number_of_clusters):
      a = distance(dataset[p], new_centroids[k])
      if a < min:
        min = a
        clusters_assignment[p] = k    # final cluster assignment
```

Task 0: Implementing the K-Means Algorithm

- The outputs of K_means()

```
# Mode selection for output of the algorithm
if mode == 1:
  return new_centroids    # the array of final centroid points
elif mode == 2:
  return clusters_assignment    # the array of each data point's cluster assignment
else:
  print('Mode Error')
```

Task 1: Toy problem

- **Number of iterations & Stopping criterions**

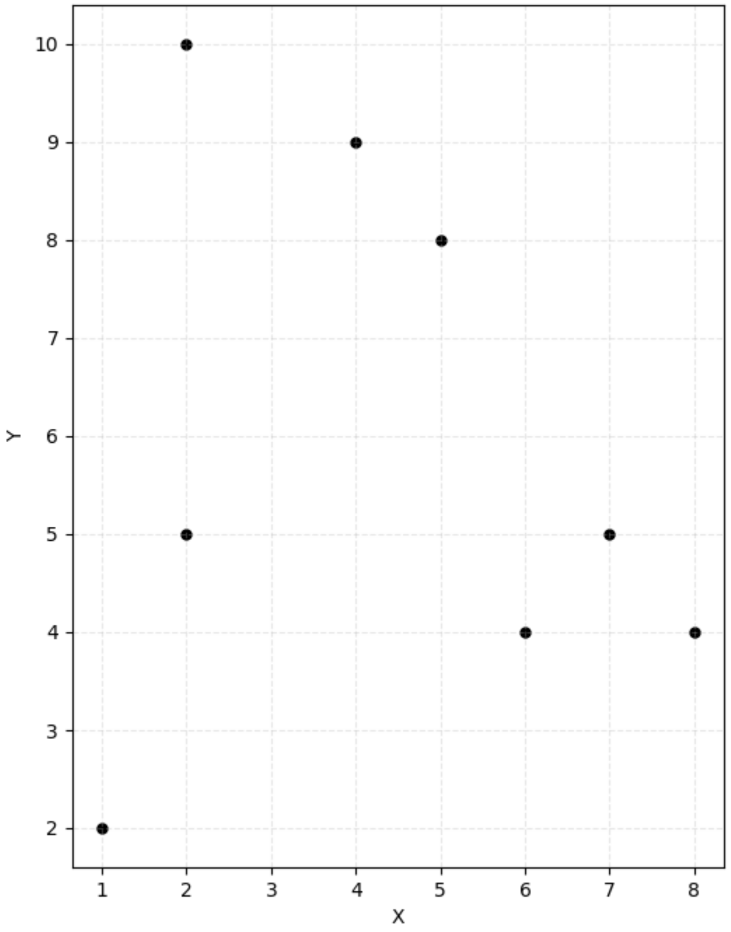
```
data1_array = [[2, 10], [2, 5], [8, 4], [5, 8], [7, 5], [6, 4], [1, 2], [4, 9]]   # given dataset for Task 1
initial_centroids_of_1 = [[2, 10], [5, 8], [1, 2]]   # given initial centroids for Task 1

new_centroids_of_1 = K_means(data1_array, initial_centroids_of_1, 100, 1)   # K-means function
```

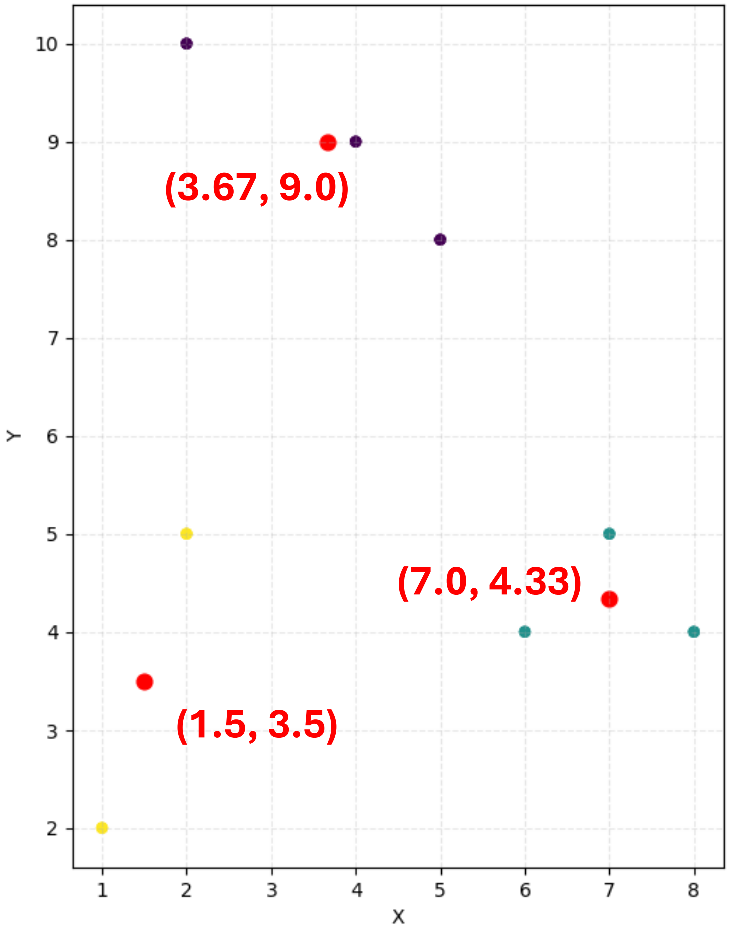
```
3.81720680758398
2.0194218076158923
1.877497007178472
0.0
the number of iterations: 4
```

The value of delta decreases to 0(=no centroid changes) after **4 iterations**, satisfying the stopping criterion($<10^{-3}$).

Task 1: Toy problem

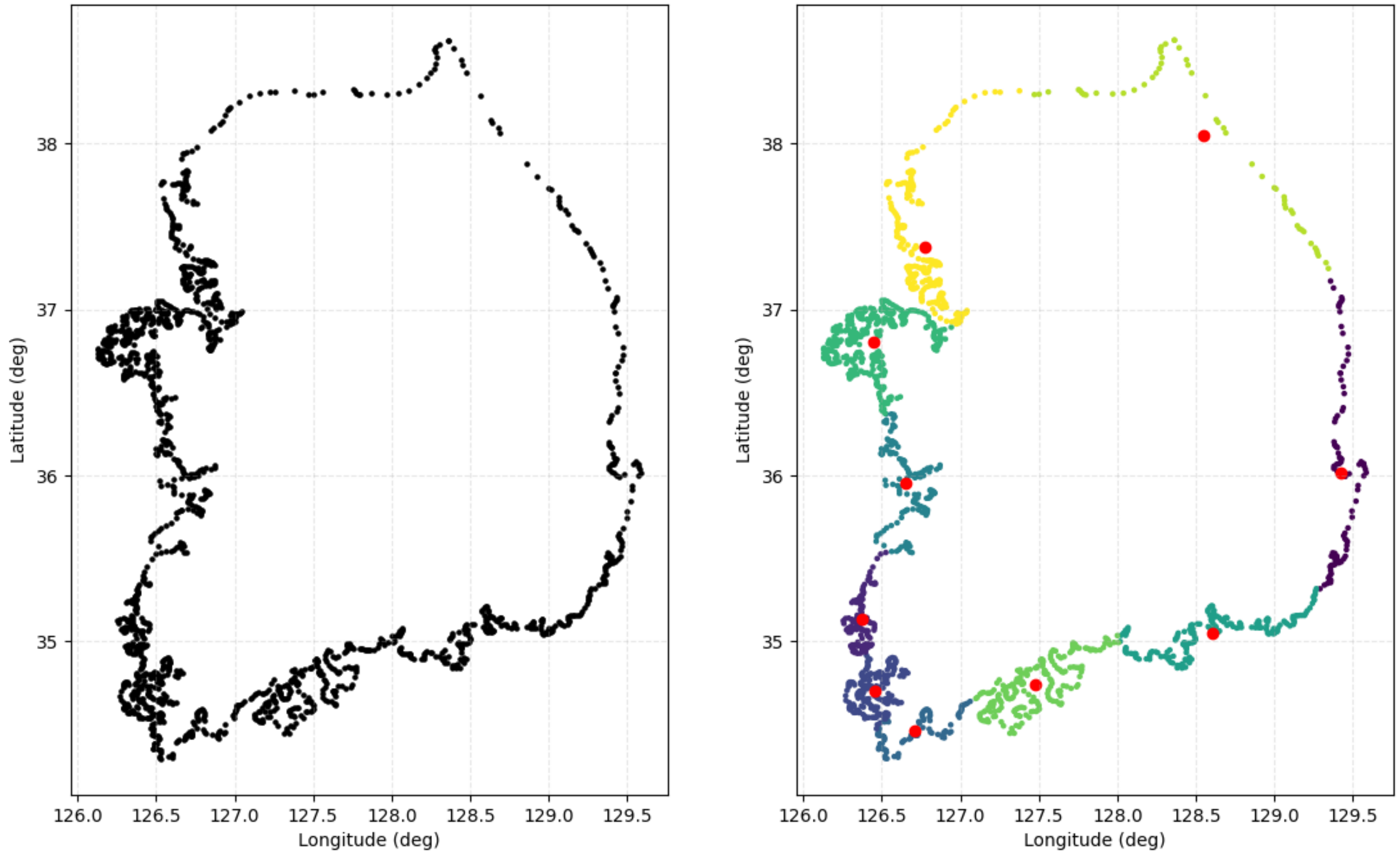
- **Clustering Results**





Task 2: Open-ended problem

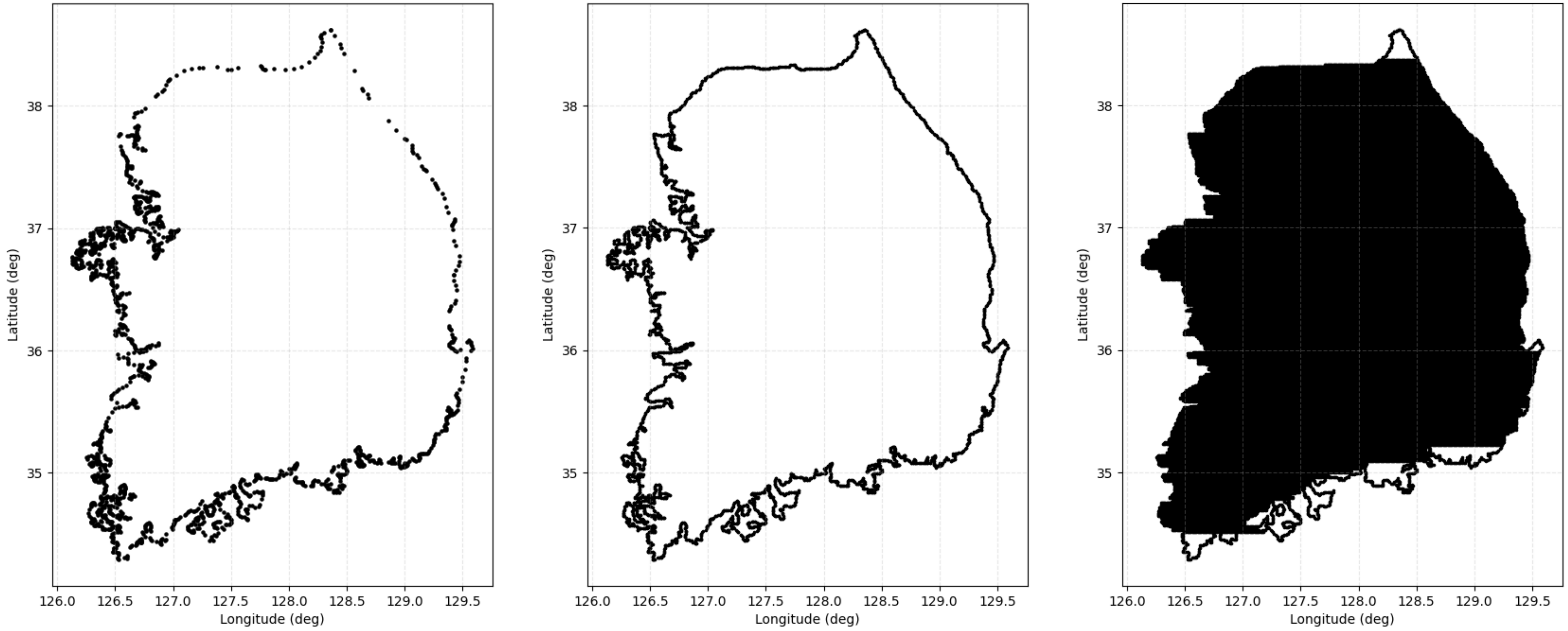
- **Problem approach:** necessity for data processing



- Clustering the boundary data points right away will result in **undesired centroid points.**

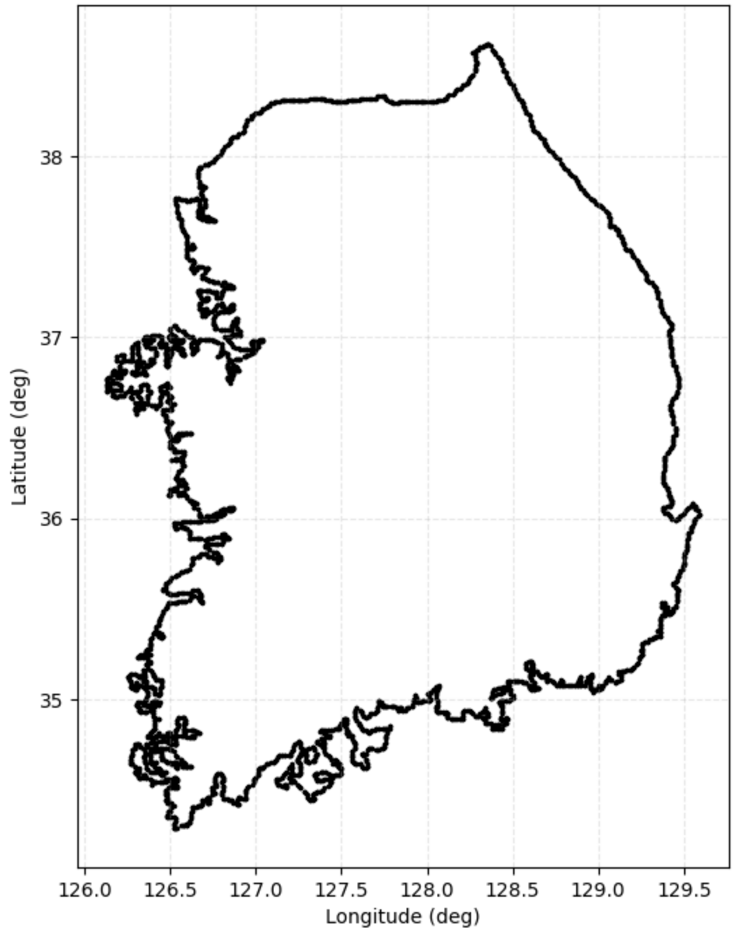
Task 2: Open-ended problem

- **Key idea:** generate new data points composing the internal area



Task 2: Open-ended problem

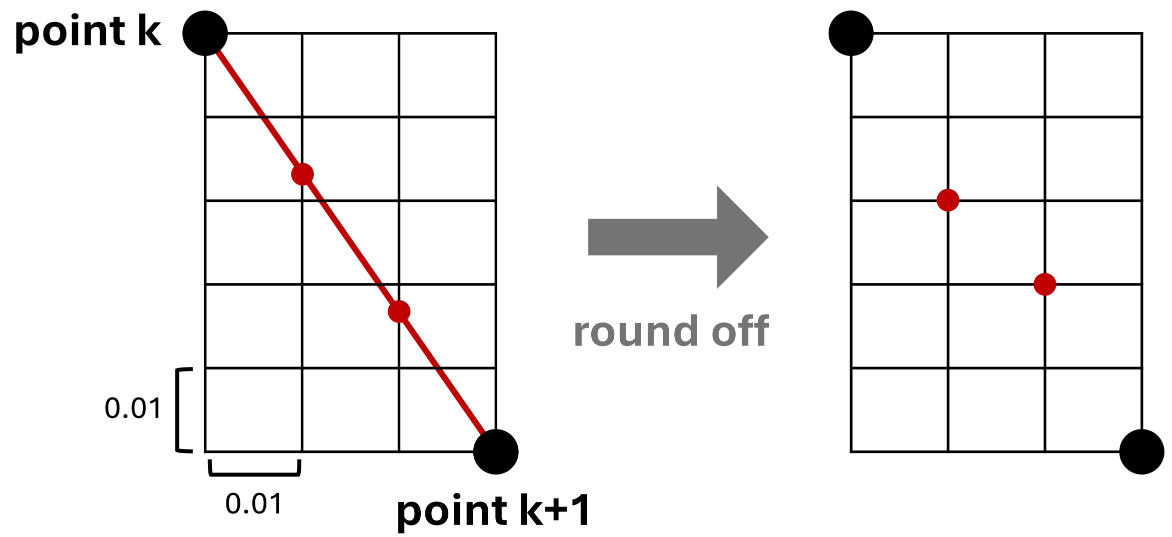
- **Process 1:** smooth the boundary data points



1) fit every data points to a **grid** with unit length = 0.01 (deg)

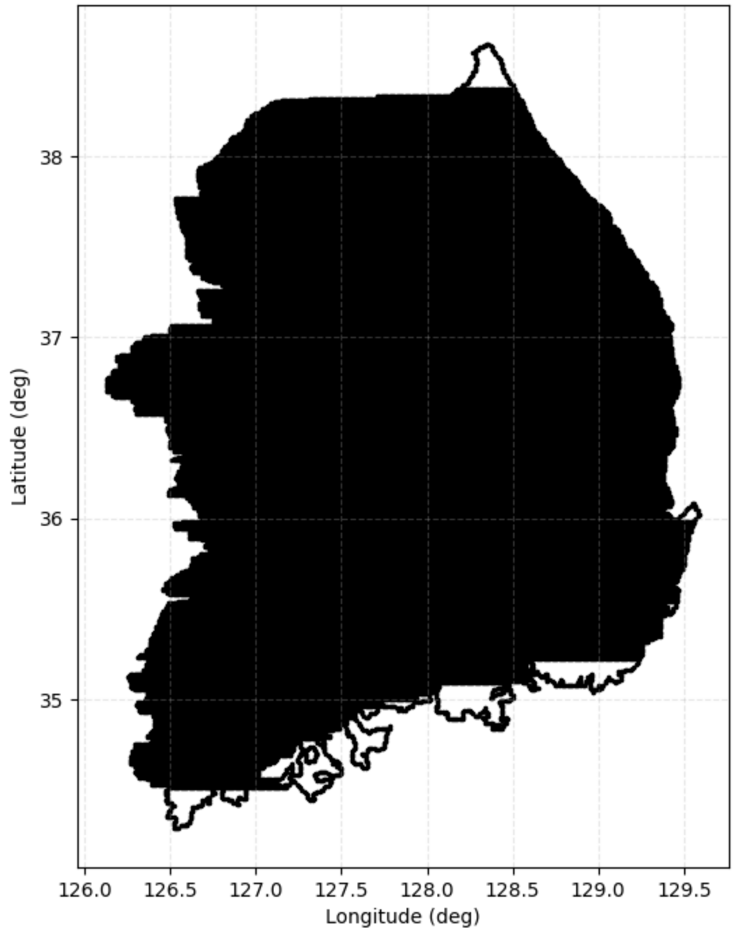
   ex) If point 0 = [128.364919, 38.624335],
   revise it into [128.36, 38.62] (round off to the 100ths)

2) for empty data point, **interpolate** with nearest points



Task 2: Open-ended problem

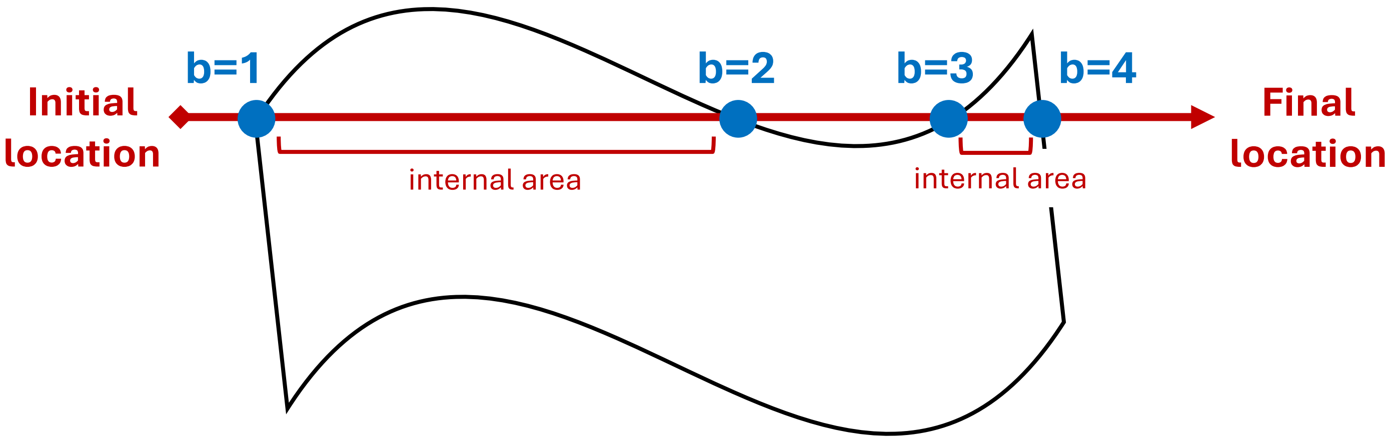
- **Process 2:** append data points if considered as a point of internal area



- a topological approach for 'internal area'

<u>Situation: searching with increasing x, constant y</u>

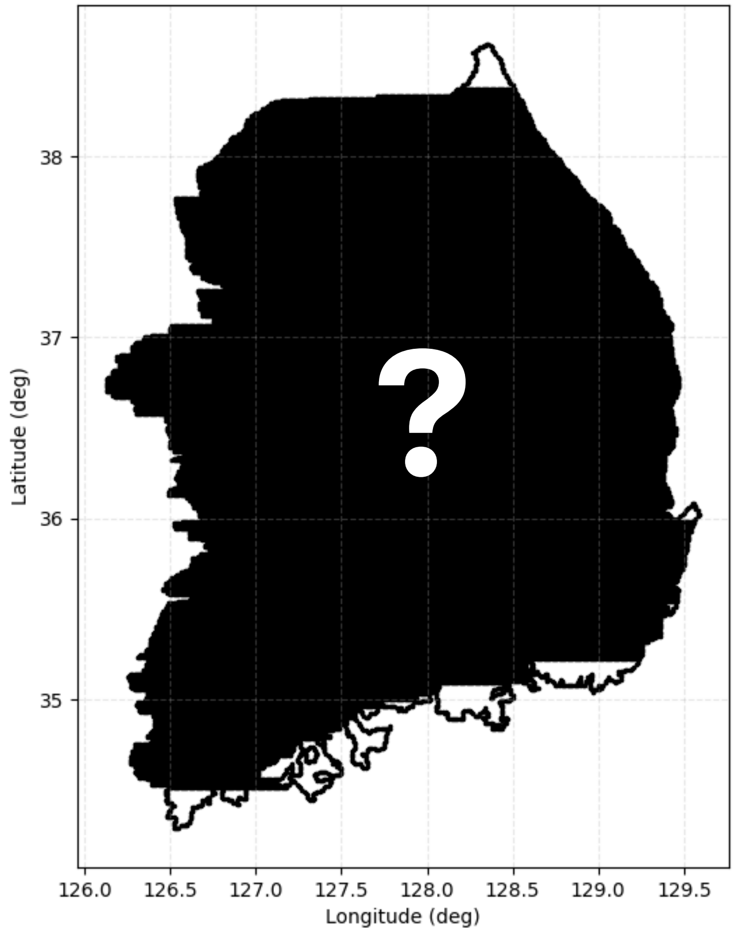
b = number of touches between search point & boundary points



→ Observation: the point is considered as an internal point if the point touches the boundary **odd times** on a single path.

Task 2: Open-ended problem

- **Process 3:** apply K_means()



K_means*(dataset, initial centroids, max. of iterations)*

- dataset: data points including the internal area (Process 2)

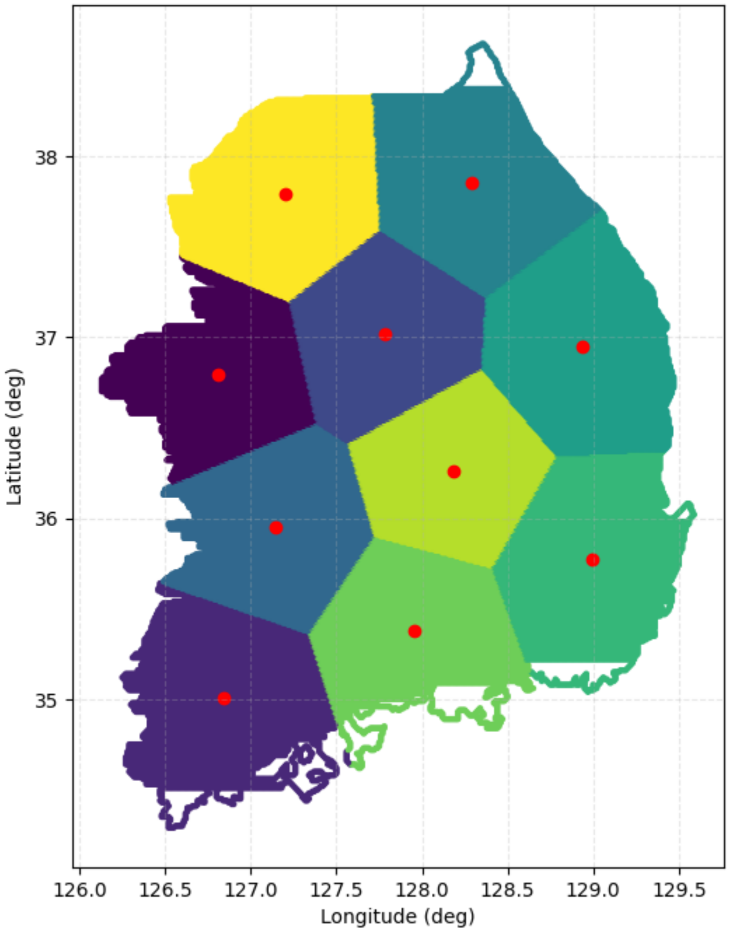
- initial centroids: choose randomly from the dataset

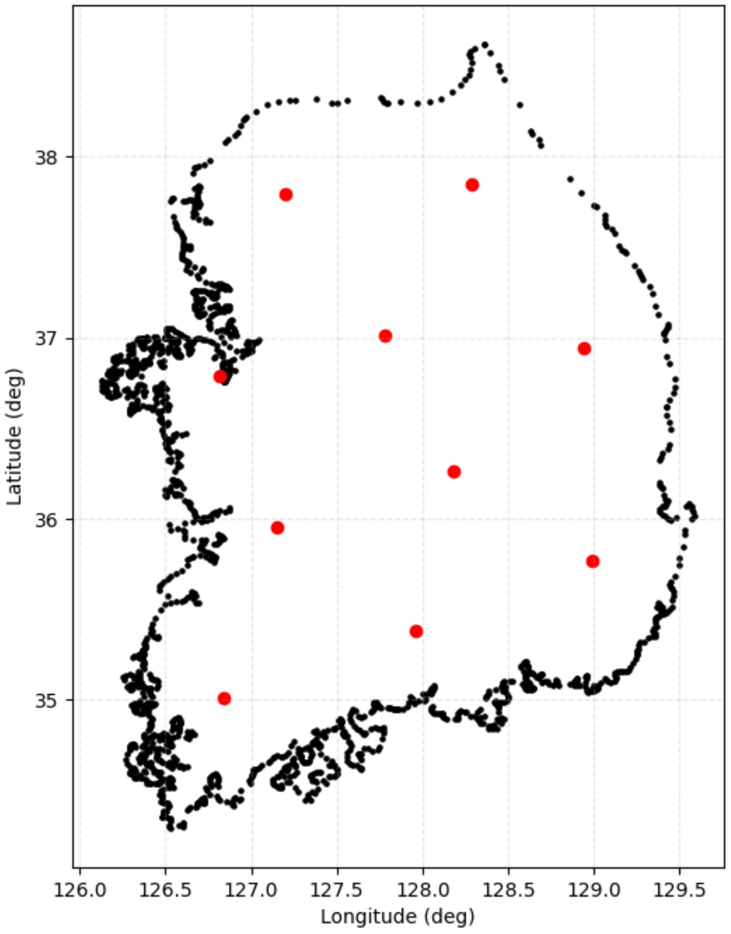
- maximum number of iterations: 100

(the stopping criterions are same as in Task 0)

Task 2: Open-ended problem

- **Clustering Results** (K = 10)



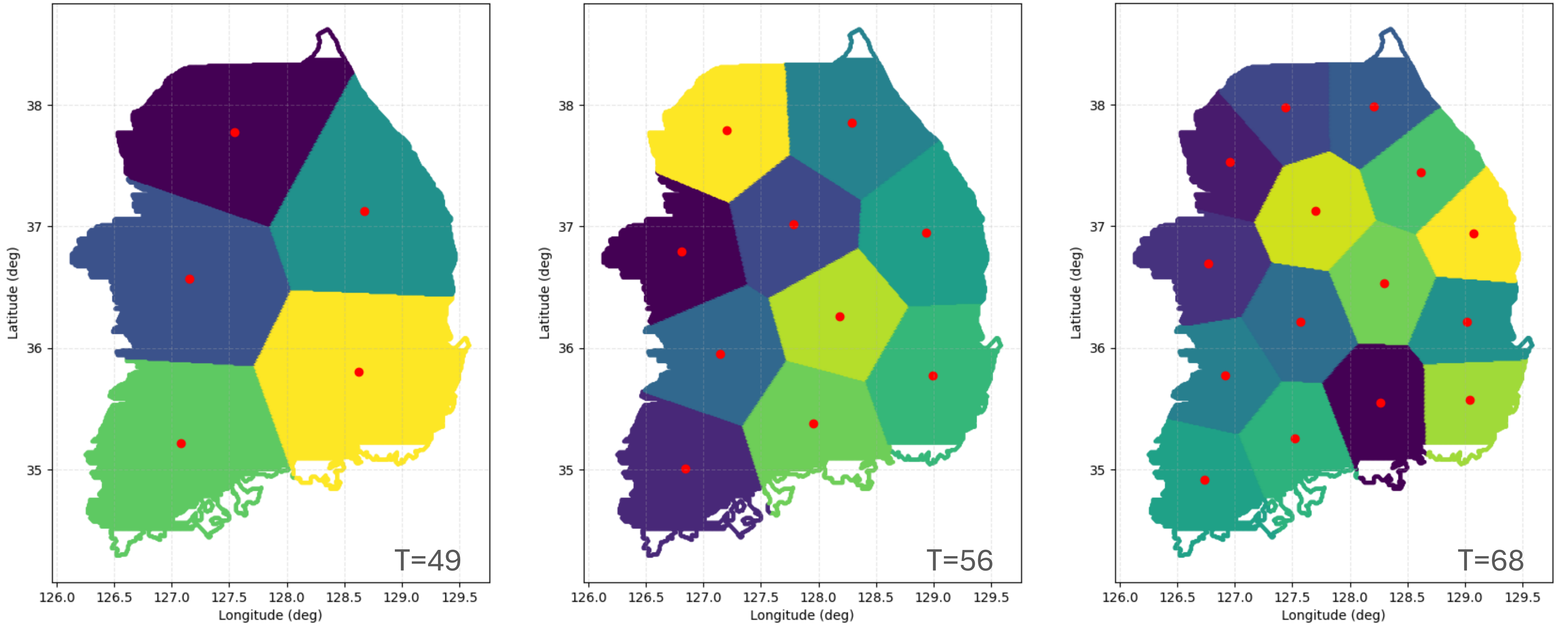


- The result would be more accurate if data points were filled properly in **complicated regions**(especially near the coastlines of south & west).

Task 2: Open-ended problem

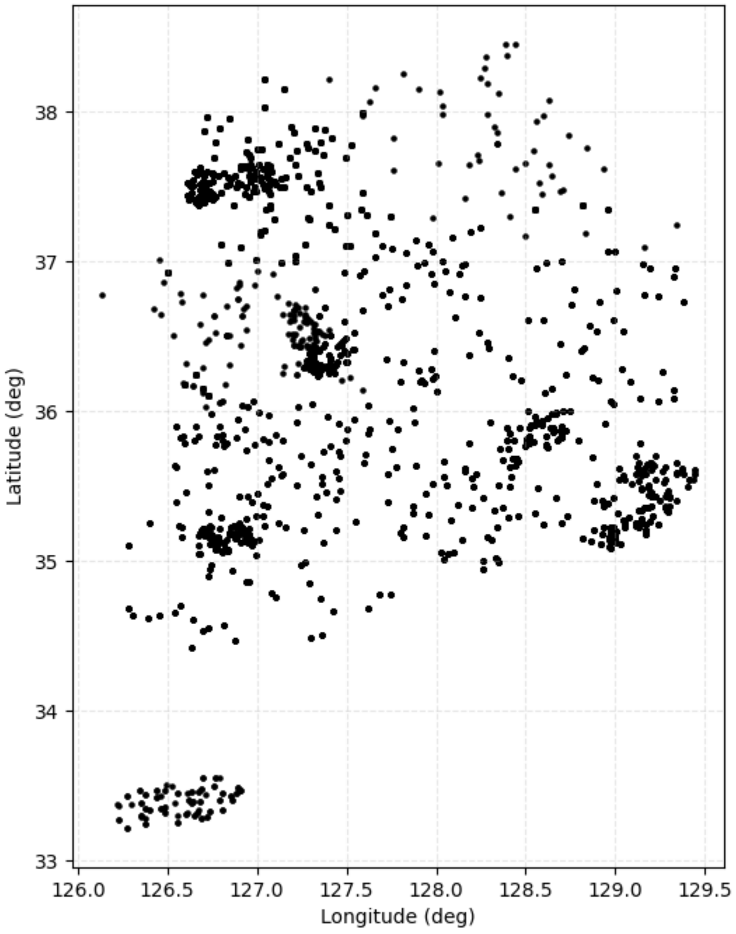
- **Clustering Results** (K = 5, 10, 15)

[T: *number of iterations*]



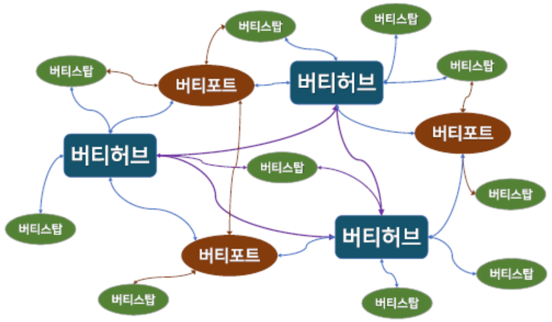
Task 3: Real-world problem: Vertiport Placement (1)

- **Research:** the specific roles of vertiports

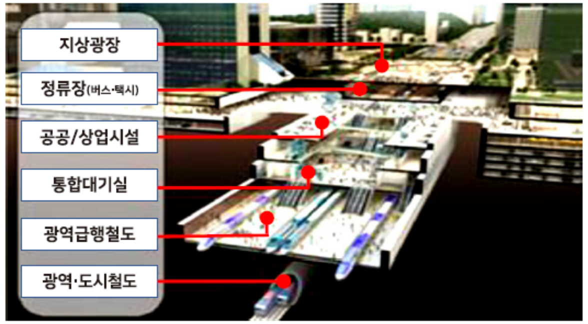


구분	버티허브	버티포트	버티스탑
개념	허브공항 개념	지역 터미널 개념	버스정류장 개념
규모	다수의 이착륙장	두 개 이상의 이착륙장	한 개의 이착륙장
시설	정비·충전·소방·의료 등 필요한 모든 서비스 시설	정비·충전시설 등	최소 시설만 보유
위치	대규모 공터가 있는 도시 외곽/ 경계 지역, 주요 공항	도심 및 주변부, 중소 도시(RAM)	건물 옥상, 도심 외곽 등
연계교통	다양한 연계교통	다양한 연계교통	연계교통 제한
비정상 상황	대체 버티포트로 활용 가능	대체 버티포트로 활용 가능	비상 시 임시 착륙장

[버티포트 간 항로구축 개념도]



[연계교통을 위한 환승센터 개념도]



[국토교통부 토지이용계획열람, 2024-04-28]

Task 3: Real-world problem: Vertiport Placement (1)

- **Research:** examples of suitable places to construct veriports

Task 3: Real-world problem: Vertiport Placement (1)

- **Problem approach:** prioritized factors and assumptions

Prioritized Factors

- population density
- feasibility
- connection of suburbs

Assumptions

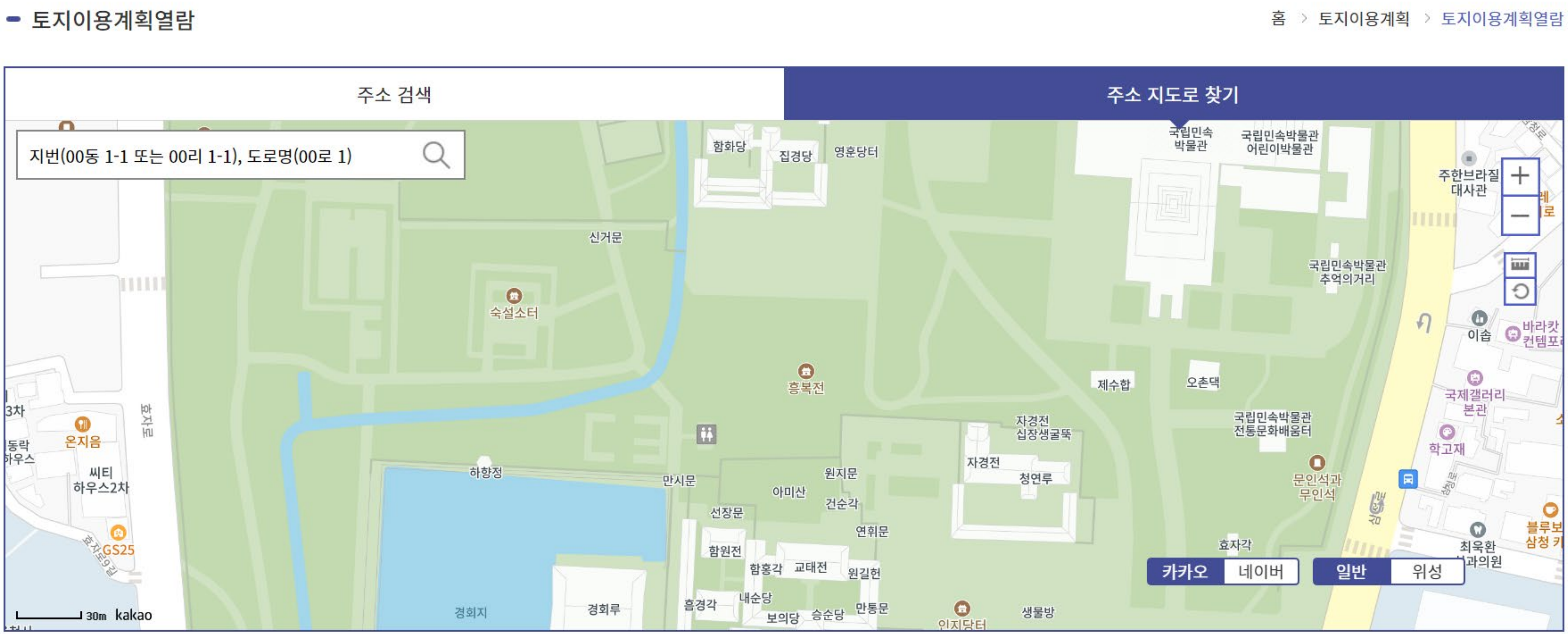
- cost of construction
- within the radius of 1.5km
- noise problems

1. land space
2. avoid **green belt areas**
3. avoid **flight prohibition areas** and other development restricted areas
4. **highways, riversides, creek-sides**
5. **urban** and **suburban** areas

Conditions for feasibility

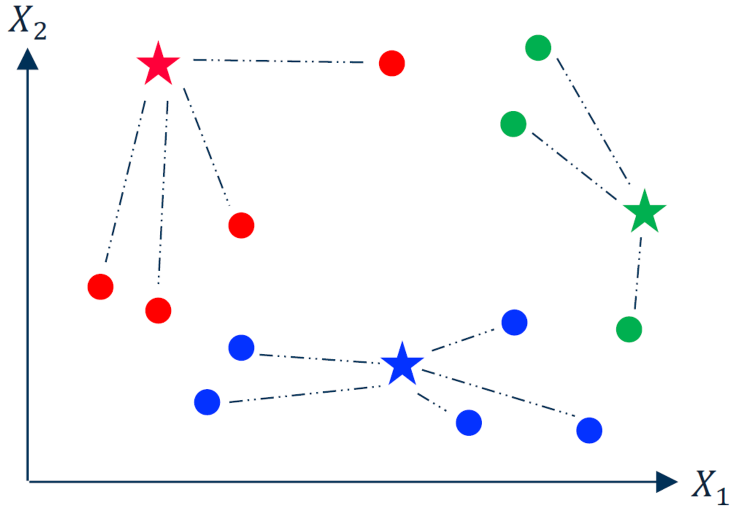
Task 3: Real-world problem: Vertiport Placement (1)

- **Problem approach:** prohibitions of the placement



Task 3: Real-world problem: Vertiport Placement (1)

- **Key idea:** examine the feasibility of clustered centroids



Apply the **K-Means algorithm**

Image from the lecture slides(Module 5-2-1)

GeoPy

centroid results

토지e음

Check the feasibility of each centroids
(Naver Map, 토지이용규제정보서비스, etc)

record feasibility

Conclude the final centroid coordinates
(if all centroids are considered feasible)

Task 3: Real-world problem: Vertiport Placement (1)

- **Process 1:** modify the K-means algorithm

1) Using existing data points

2) Retaining feasible locations

3) Ignoring infeasible locations

1) Since the original K-means algorithm computes each centroid point by the mean of data points, a centroid could result in a new data point.

→ A new data point for a centroid is same as building a new veriport(=unwanted).

→ Therefore, the algorithm was modified to pick out existing data points as centroids.

```
tolerance = 1e-14
for k in range(num_flex_clusters): #for each new centroid in flexible array,
  min = np.inf
  min_point = dataset[0]
  for p in range(number_of_points): #calculate its distance with every point
      b = distance(flexible_centroids[k], dataset[p])
      if b < min: #if that point has closer distance, replace it
        min_point = dataset[p]
        min = b
  flexible_centroids[k] = min_point
```

Task 3: Real-world problem: Vertiport Placement (1)

- **Process 1:** modify the K-means algorithm

1) Using existing data points

2) Retaining feasible locations

3) Ignoring infeasible locations

2) In order to remember feasible locations, the algorithm was modified to keep points that are already clarified as feasible points.

```
# Split into Fixed and Flexible Centroids
fixed_centroids = initial_centroids[:retain_count]
flexible_centroids = initial_centroids[retain_count:]
num_flex_clusters = len(flexible_centroids)
…
# 3. Calculate new centroid coordinates only for clusters in flexible array
for k in range(num_flex_clusters): #for every cluster k in flexible array,
    …
# 6. Replace with existing data points only for centroids in flexible array
for k in range(num_flex_clusters):
    …
# 7. Combine fixed list with flexible list
combined_array = np.concatenate((fixed_centroids, flexible_centroids), axis=0)
```

Task 3: Real-world problem: Vertiport Placement (1)

- **Process 1:** modify the K-means algorithm

1) Using existing data points

2) Retaining feasible locations

3) Ignoring infeasible locations

3) In order to exclude unfeasible locations, the algorithm was modified to ignore points that are already clarified as unfeasible points.

```
if b < min: #if that point has closer distance to centroid, replace
  #result: whether point is in skip_array
  result = any(np.allclose(dataset[p], sublist, atol=tolerance) for sublist in
  skip_array)
  if result: #if the point is in skip_array, skip
    continue
  min_point = dataset[p]
  min = b
flexible_centroids[k] = min_point
```

Task 3: Real-world problem: Vertiport Placement (1)

- **Process 2:** repeat the modified algorithm until all K-feasible centroids are obtained

Results 1-2

율리영해1길, 울주군, 울산광역시, 44602, 대한민국 (35.53608654312682, 129.2211948530974) ❌ mountain

태봉리.목다라맨골, 해운로, 태봉리, 서산시, 충청남도, 31946, 대한민국 (36.78277753915864, 126.56941052239416) ❌ too rural

덕평리, 여주시, 12667, 대한민국 (37.18172679839141, 127.65915182871387) ❌ too rural

26100, 북평면, 정선군, 강원특별자치도, 대한민국 (37.45006324125764, 128.5934748370431) ❌ too rural

공항로, 대저1동, 강서구, 부산광역시, 경상남도, 46703, 대한민국 (35.22326457307819, 128.99272367315277) ✅ bus stops nearby, land space, suburban

기와집길, 상면, 가평군, 12444, 대한민국 (37.79203100352012, 127.351541512092) ❌ too rural

세천동, 대청동, 동구, 대전, 34501, 대한민국 (36.32441773035698, 127.49985568869496) ❌ too rural

후정동로, 삼산동, 부평구, 인천광역시, 21318, 대한민국 (37.5192288643404, 126.73648511066368) ❓ suburban

송내리, 금강로, 마서면, 서천군, 충청남도, 33657, 대한민국 (36.02857668679363, 126.7104527157177) ❓ ✅ suburban but a little far from public transportation

작원길, 관문동, 북구, 대구광역시, 41489, 대한민국 (35.90289386606794, 128.54962196444197) ✅ suburban, 매천시장역 근처

금호로13길, 금호동2가, 금호2·3가동, 성동구, 서울특별시, 04723, 대한민국 (37.55533815289316, 127.01924322691028) ✅ suburban, many apartments - 옥상, 신금호역 바로 옆

Task 3: Real-world problem: Vertiport Placement (1)

- **Clustering Results** (K = 17)

📌 부산광역시 강서구 대저1동

📌 충청남도 서천군 마서면 송내리

📌 대구광역시 북구 매천동

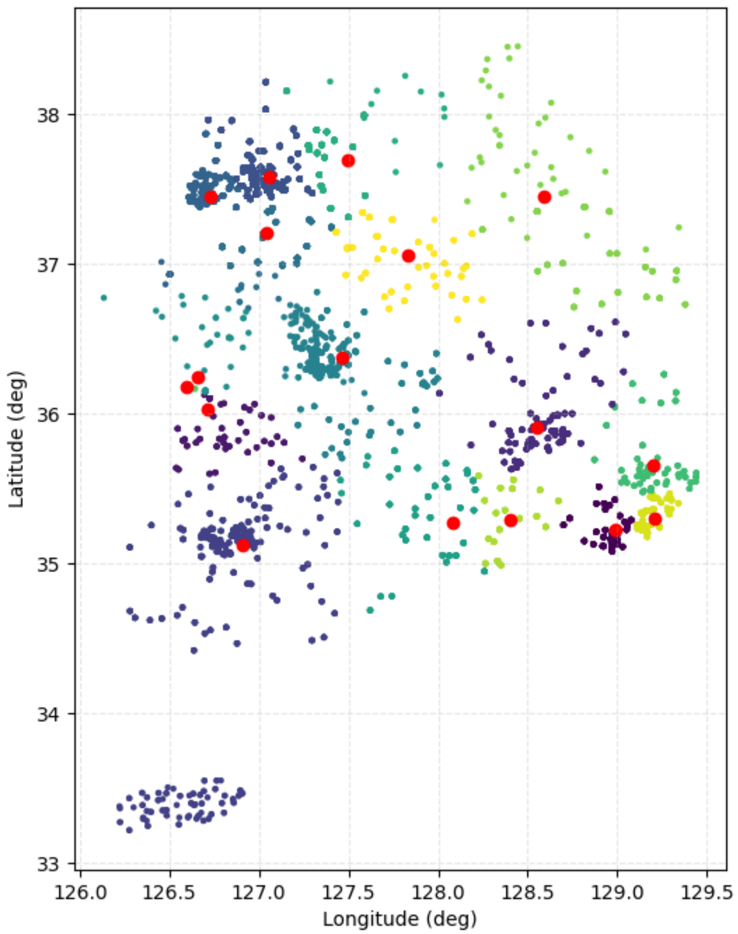
📌 광주광역시 남구 봉선동

📌 인천광역시 남동구 수산동

📌 서울특별시 동대문구 사가정로

📌 경기도 화성시 병점1로

📌 대전광역시 동구 추동



📌 충청남도 보령시 미산면

📌 경상남도 진주시 집현면 대암리

📌 경기도 가평군 설악면 사룡리

📌 울산광역시 울주군 두동면 구미리

📌 충청남도 보령시 주산면 유곡리

📌 강원특별자치도 정선군 북평면 중봉길

📌 경상남도 함안군 가야읍 도항리

📌 부산광역시 기장군 일광읍 용천리

📌 충청북도 충주시 중앙탑면 봉황리

Task 3: Real-world problem: Vertiport Placement (1)

- Evaluation of the method

1) Subjective standards

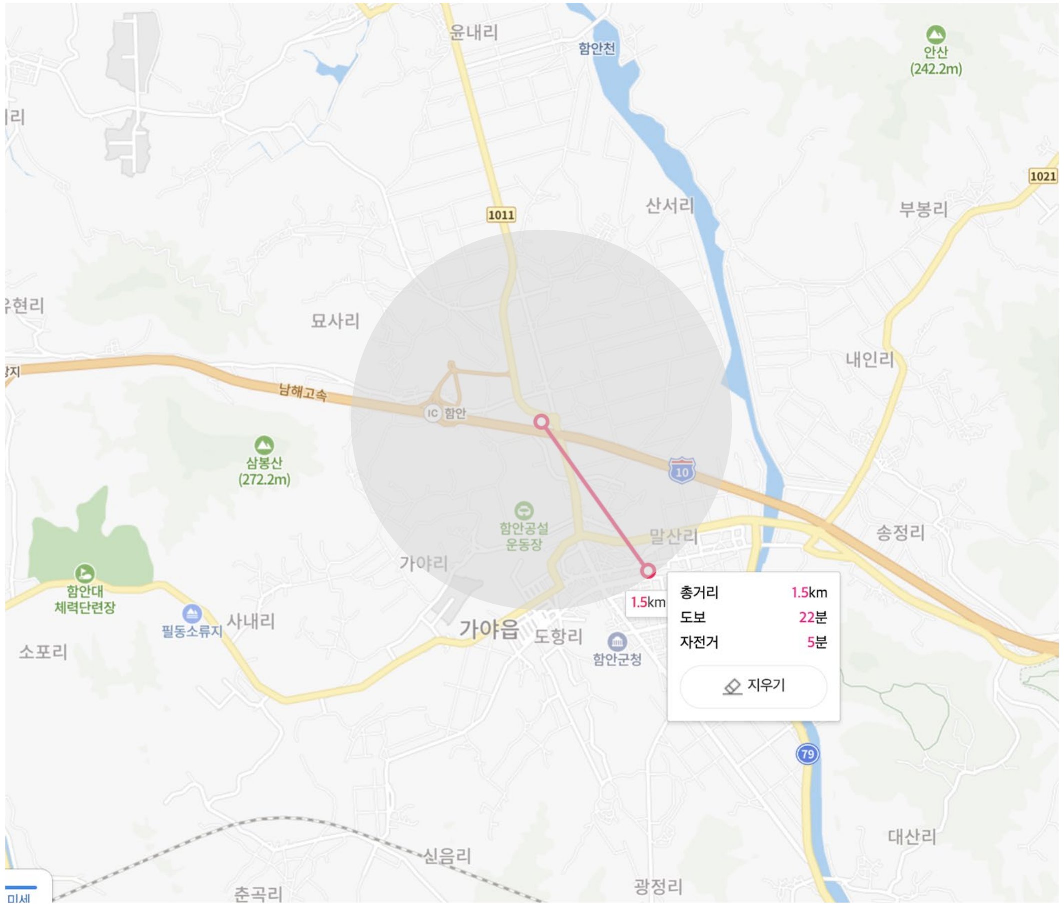
2) Within the radius of 1.5km

3) Unequal distribution of density

4) Regions separated by mountains

5) Analyzing the ***best*** feasible locations

6) Assumptions



Task 3: Real-world problem: Vertiport Placement (1)

- Potential extensions and Approaches

  1) Adding more factors to consider:

     e.g. Cost, weather conditions, regions separated by mountains, public opinion

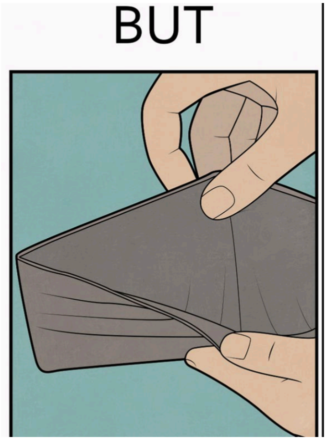
  2) Structuring each point as nodes with weight that carry information about feasibility, cost, etc

     → convert factors into numerical data

     → calculate the distribution that maximizes the total sum of the weights

Task 3: Real-world problem: Vertiport Placement (2)

- Finding the optimal number of veriports



✓ AS MANY VERTIPORTS AS WE WANT

✓ NO FINANCIAL CONSTRAINT

✓ WE DO NOT WANT TO SPEND UNNECESSARY

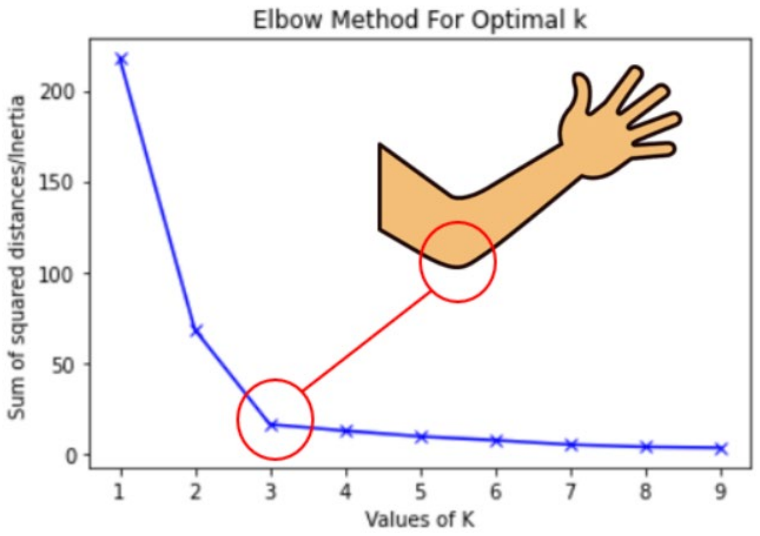
✓ WANT TO INVEST MONEY IN AN EFFICIENT MANNER

Task 3: Real-world problem: Vertiport Placement (2)

- Finding the optimal number of veriports

ELBOW METHOD

✓ Determining the the optimal number of clusters in a dataset



Line plot between K and inertia

SILHOUETTE SCORE

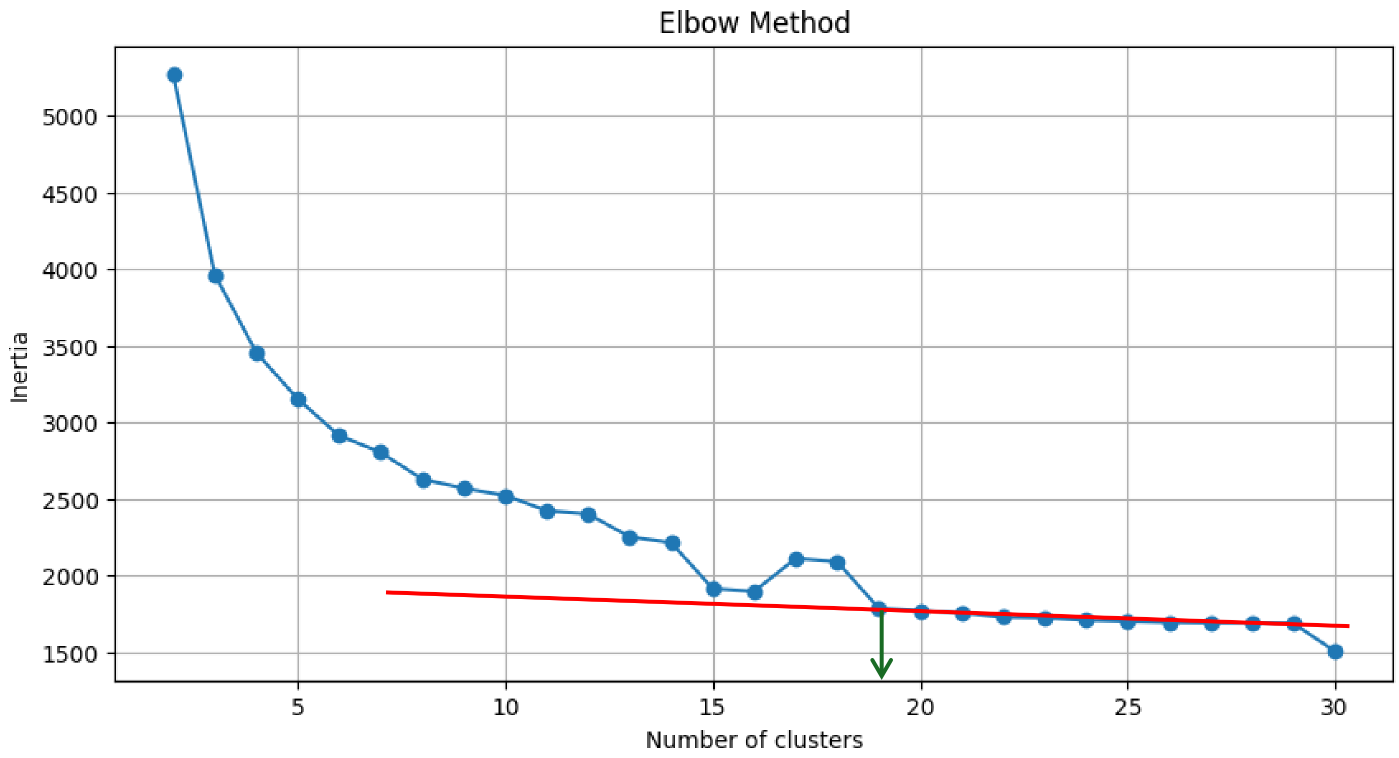
✓ to evaluate the quality of clusters in a clustering

✓ provides a measure of how similar a data point is to its own cluster

Task 3: Real-world problem: Vertiport Placement (2)

- Finding the optimal number of veriports: **Elbow Method**

```
#inertia for the elbow method
centroids = K_means(data3_array, initial_centroids, 100, 1)
inertia.append(sum(np.min(np.array([np.linalg.norm(data3_array - centroid, axis=1) for centroid in centroids]), axis=0)))
```




```
# Plotting elbow method
plt.figure(figsize=(10, 5))
plt.plot(range(2, max_clusters+1), inertia, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.grid(True)
plt.show()
```

Roles of the project

- Roles of the project

Team 2	Task 0	Task 1	Task 2	Task 3	Presentation
Mark	brainstorming & coding	coding & discussion	review	coding	Task 3-2
Youngjin Seoh			research	review	Task 0, 1
Semin Na			coding	review	Task 2
Yebin Pyun			review	coding	Task 3-1

Reference

- Reference

[1] Junghyun Kim. (2024, March 29). *Module 5-2-1 K-Means algorithm* [Lecture Slides]. Seoul National University. https://myetl.snu.ac.kr/courses/262625

[2] 한국전자기술연구원(KETI). (2022). *KETI Issue Report: 국내 UAM 산업육성을 위한 정책 제언*. 한국전자기술연구원 기술정책실. https://www.keti.re.kr/_upload//issue/2023/01/13/application_9dd195116671a8f33990d7d8563bb85b.pdf

[3] 토지이용계획열람[국토교통부 토지e음]. (2024.04.28). URL: https://www.eum.go.kr/web/ar/lu/luLandDet.jsp

[4] KDI 경제정보센터 자료연구팀. (2023). 새로운 모빌리티의 등장, 도심항공교통(UAM). *해외동향*. https://eiec.kdi.re.kr/publish/reviewView.do?ridx=16&idx=170&fcode=000020003600003

[5] 현대트랜시스. (2022, Aug 5). 하늘길 여는 미래 항공 모빌리티, UAM과 RAM 뭐가 다를까?. https://blog.hyundai-transys.com/302

[6] 유태영. (2022, Dec 22). GS·현대·롯데 건설, '버티포트' 시장 선점 위해 속도 경쟁. *Opinion News*. http://www.opinionnews.co.kr/news/articleView.html?idxno=78130

[7] Junghyun Kim. (2024, April 9). *Module 5-4-1 Clustering model evaluation* [Lecture Slides]. Seoul National University. https://myetl.snu.ac.kr/courses/262625