클릭 몇 번, 간단한 고객 세그먼트 분석 서비스

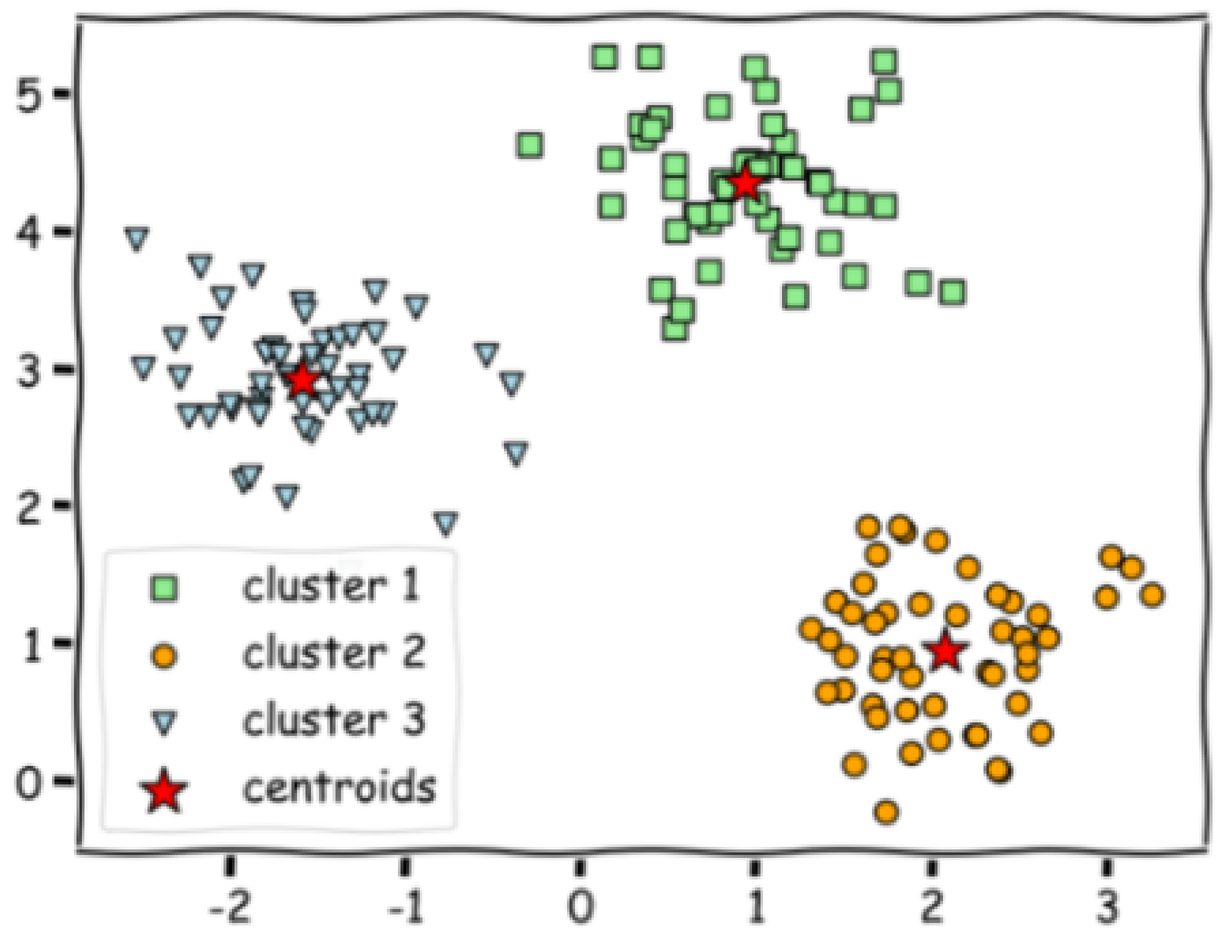
NEO k-means를 활용한 세그먼트 분석 서비스 개발

2023-2 산업경영알고리즘

2020108968 차민재 | 2021104011 임소영

CONTENTS

01 주제 선정 배경



K-means 알고리즘의 한계

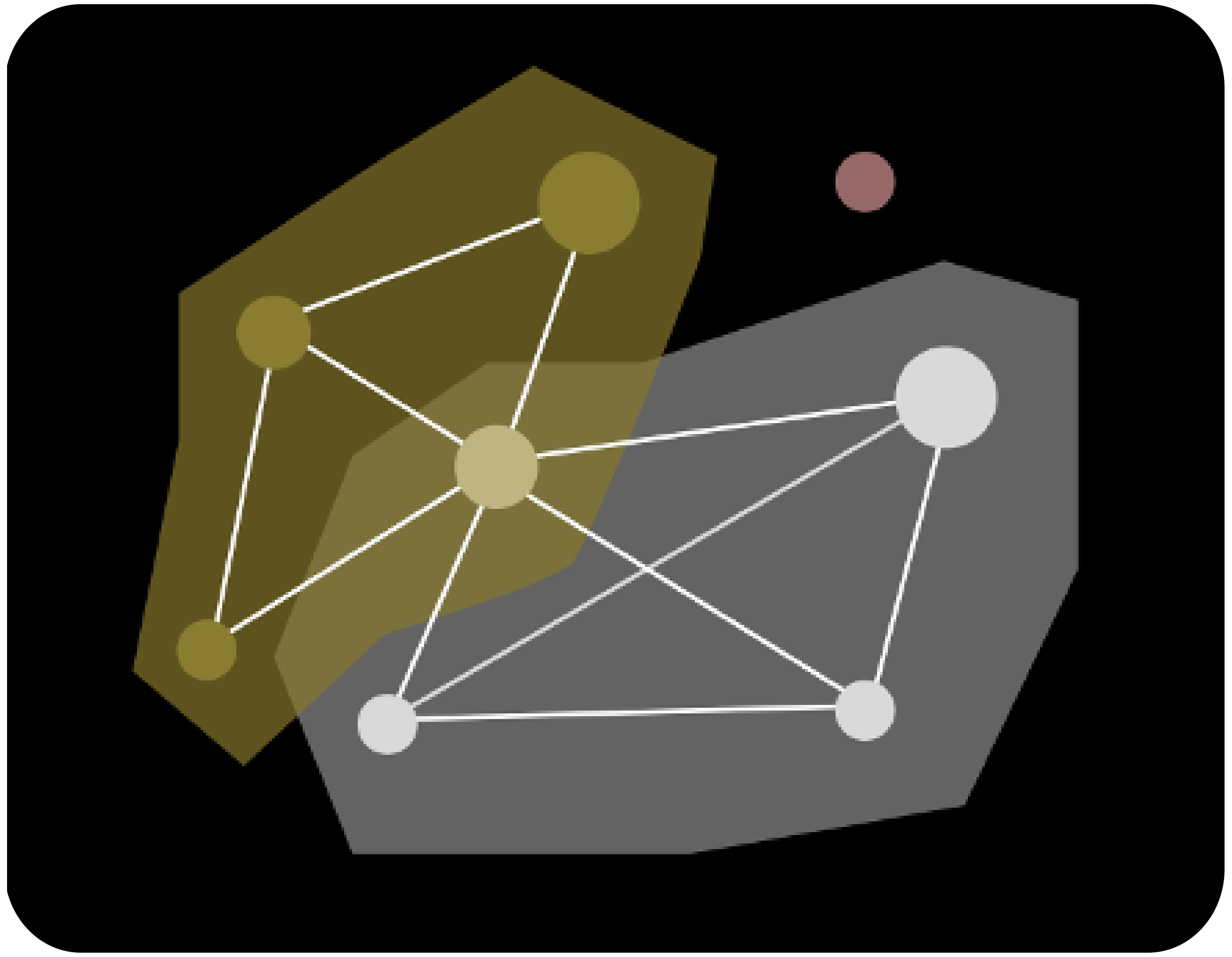
K-means는 클러스터링 알고리즘 중 잘 알려진 방법이다. 그러나 데이터가 어떤 클러스터에 꼭 속해 있어야 한다는 단점이 있다. 그래서 아래와 같은 문제가 발생한다.

(1) 이상치를 표현하지 못 한다

(2) 하나의 데이터가 여러 개의 클러스터에 속하지 않는다

현실에서는 이상치가 존재하고, 여러 개의 군집에 포함되는 경우가 많다.
이렇게 현실을 더 잘 반영하는 클러스터링은 없을까?

01 주제 선정 배경



NEO K-means

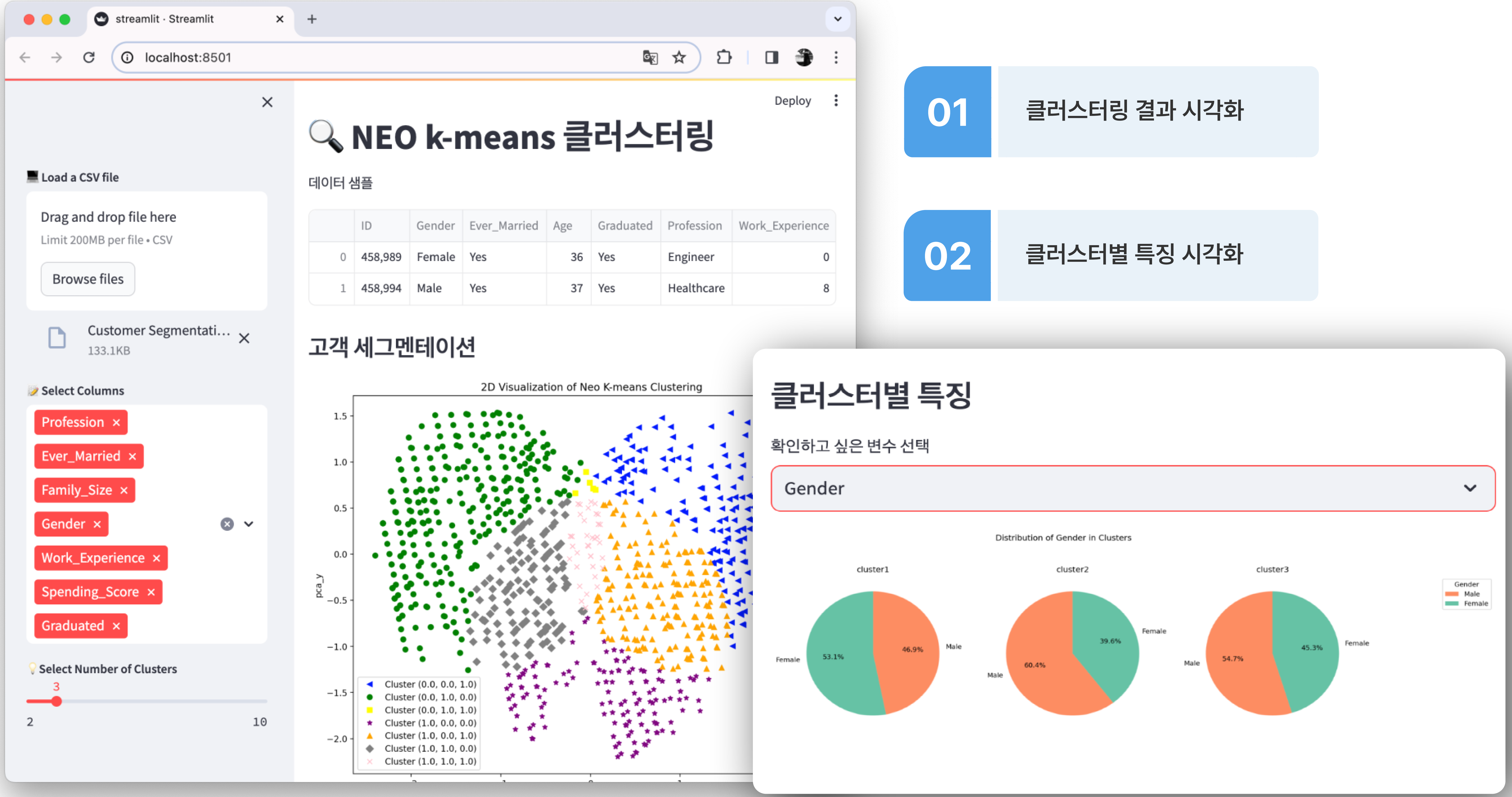
기존 K-means의 한계를 극복한 것이 NEO(Non-Exhaustive, Overlapping) K-means이다. 왼쪽 사진처럼 두 개의 클러스터에 속한 데이터가 있을 수 있고, 어떤 클러스터에도 속하지 않는 이상치가 존재하기도 한다.

다른 클러스터링 알고리즘도 많지만, NEO K-means는 비교적 최신 알고리즘이며 파이썬으로 구현된 코드를 찾아보기 어려워 이를 구현해보고자 한다.

02

서비스 소개

- 분석 원하는 데이터 입력
- 원하는 변수만 선택 가능
- 클러스터 개수 지정



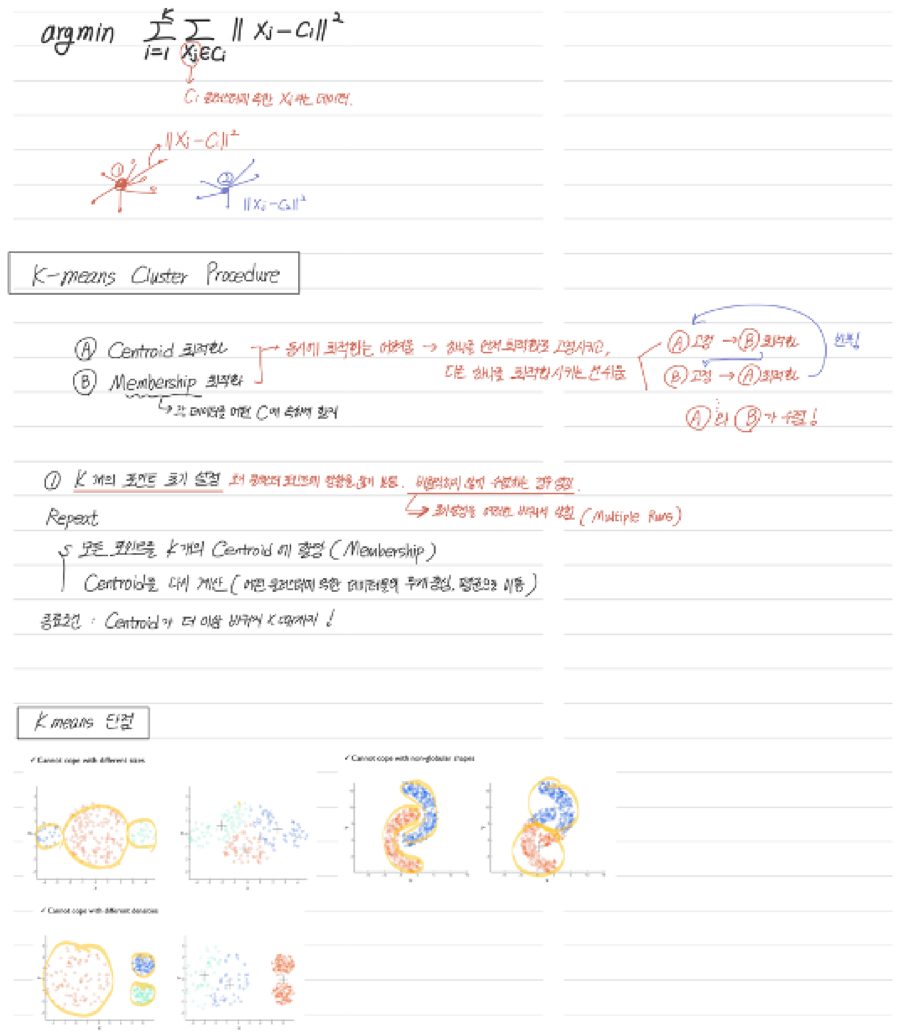
01 클러스터링 결과 시각화

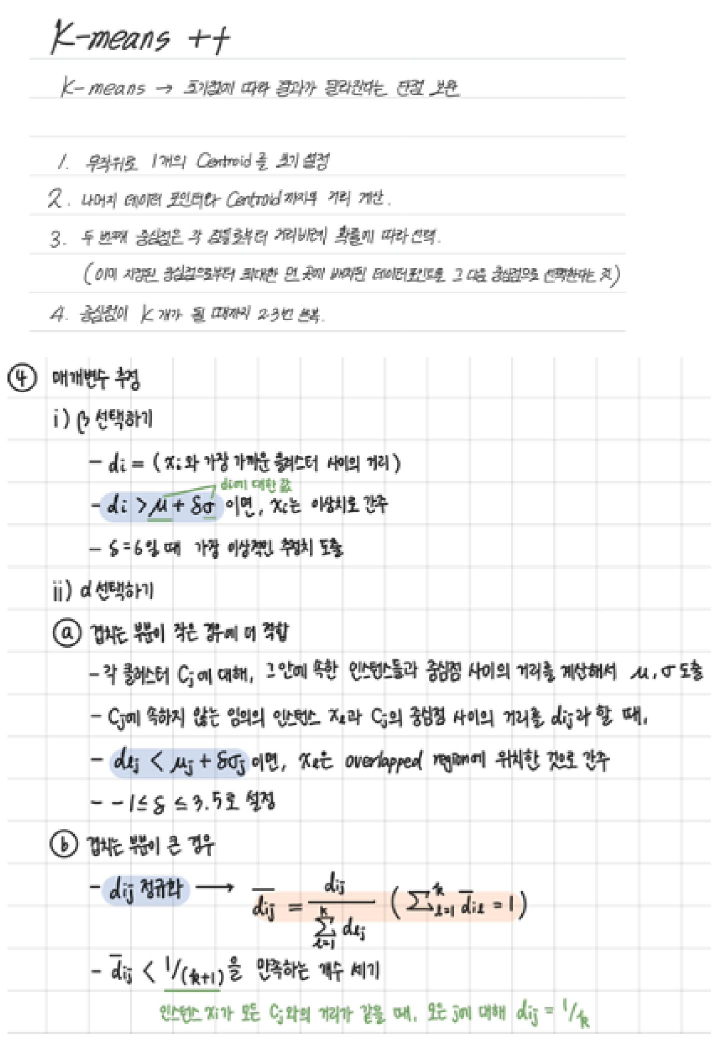
02 클러스터별 특징 시각화

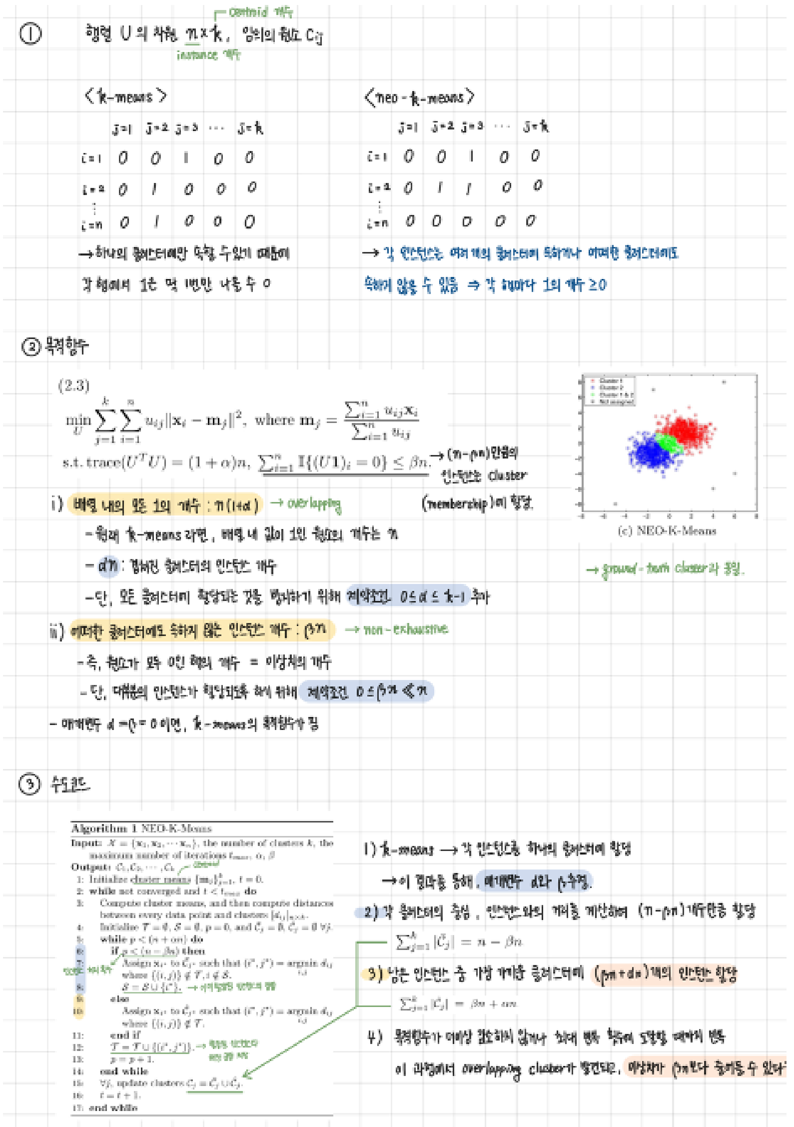
03 NEO Kmeans 공부

NEO K-means는 2015년 논문으로 발표되었다. 논문을 보며 개념 공부를 했고, 필요한 사전지식은 구글링을 통해 습득했다.

J. J. Whang, Y. Hou, D. F. Gleich and I. S. Dhillon, "Non-Exhaustive, Overlapping Clustering," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 41, no. 11, pp. 2644-2659, 1 Nov. 2019, doi: 10.1109/TPAMI.2018.2863278.







03 NEO Kmeans

알고리즘

목적함수와 제약식

$$\min_{U} \sum_{j=1}^{k} \sum_{i=1}^{n} u_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|^2, \text{ where } \mathbf{m}_j = \frac{\sum_{i=1}^{n} u_{ij}\mathbf{x}_i}{\sum_{i=1}^{n} u_{ij}}$$

$$\text{s.t. } \operatorname{trace}(U^T U) = (1+\alpha)n, \ \sum_{i=1}^{n} \mathbb{I}\{(U\mathbf{1})_i = 0\} \le \beta n.$$

Overlapping

배열 내의 모든 원소의 개수를
(1+α)n 개가 되도록 함.
αn : 겹쳐진 데이터의 개수
단, 모든 클러스터에 할당되는 것을 방지하기 위해 제약조건 0 ≤ α ≤ k-1 을 추가

Non-exhaustive

어떠한 클러스터에도 속하지 않는
데이터의 개수를 βn 으로 설정함.
단, 대부분의 데이터가 클러스터에 할당되도록 하기 위해 제약조건 0 ≤ βx ≤ n 을 추가

Algorithm 1 NEO-K-Means

Input: $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n\}$, the number of clusters k, the maximum number of iterations t_{max}, α, β

Output: $\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k$

Initialize cluster means $\{\mathbf{m}_j\}_{j=1}^{k}$, $t = 0$.
while not converged and $t < t_{max}$ **do**
  Compute cluster means, and then compute distances between every data point and clusters $[d_{ij}]_{n \times k}$.
  Initialize $\mathcal{T} = \emptyset$, $\mathcal{S} = \emptyset$, $p = 0$, and $\bar{\mathcal{C}}_j = \emptyset$, $\hat{\mathcal{C}}_j = \emptyset \ \forall j$.
  while $p < (n + \alpha n)$ **do**
    if $p < (n - \beta n)$ **then**
      Assign $\mathbf{x}_{i^*}$ to $\bar{\mathcal{C}}_{j^*}$ such that $(i^*, j^*) = \operatorname{argmin}_{i,j} d_{ij}$ where $\{(i,j)\} \notin \mathcal{T}, i \notin \mathcal{S}$.
      $\mathcal{S} = \mathcal{S} \cup \{i^*\}$.
    else
      Assign $\mathbf{x}_{i^*}$ to $\hat{\mathcal{C}}_{j^*}$ such that $(i^*, j^*) = \operatorname{argmin}_{i,j} d_{ij}$ where $\{(i,j)\} \notin \mathcal{T}$.
    end if
    $\mathcal{T} = \mathcal{T} \cup \{(i^*, j^*)\}$.
    $p = p + 1$.
  end while
  $\forall j$, update clusters $\mathcal{C}_j = \bar{\mathcal{C}}_j \cup \hat{\mathcal{C}}_j$.
  $t = t + 1$.
end while

03 서비스 구현과정

알고리즘 구현은 K-means++및 beta, alpha 추정과 NEO Kmeans까지 하였으며,
그 이후로는 라이브러리를 활용하였음.

K-means ++ 알고리즘 구현

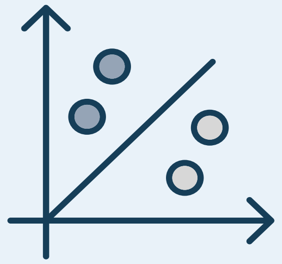
각 데이터가 어떤 군집에 속했는지를 알아내기 위함 → **initU** 설정

- 몇 개의 데이터가 이상치가 되는지 → **BetaN** 설정
- 몇 개의 데이터가 여러 군집에 포함될 수 있는지 → **AlphaN** 설정

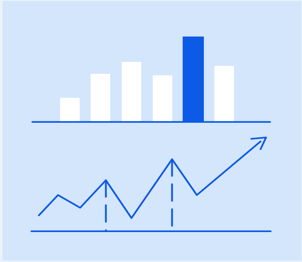
NEO K-means

시각화

클러스터링 결과 시각화



클러스터별 특징 시각화



UI 서비스 구현

Streamlit

Streamlit를
활용한 대시보드 구현

*모든 코드를 보여주기에 공간적, 시간적 한계로 수도코드만 첨부했습니다.
전체 코드는 마지막에 첨부한 깃허브 링크에서 확인할 수 있습니다.

03 K-means ++ 알고리즘

K-means++ 알고리즘은 K-means와 달리 초기 중심점을 서로 떨어져 있게 배치함으로써 K-means가 초기값 설정에 따라 결과가 달라진다는 단점을 보완한다.

random_init
주어진 데이터 포인트 중 무작위로 한 개를 선택해 첫 번째 중심으로 지정
이미 지정된 중심점으로부터 최대한 먼 곳에 배치된 데이터 포인트를 다음 중심점으로 지정

```
random_init(array)
{
  M ← []
  indices ← []

  i ← randint(0,len(array)
  M ← array[i]
  indices ← i

  while len(M) < k

    max_dist ← -∞
    max_index ← -1

    for i ← 1 o len(array)
      avg_dist ← 0
      if (i ∈ indices) then continue
      for j ← 1 o len(M)
        dist ← euclidean(array[i], array[j])
        avg_dist += dist
      avg_dist /= len(M)

      if (max_dist < avg_dist)  then max_dist ← avg_dist; max_index ← i

    M ← array[max_index]
    indices ← max_index

  return M
}
```

fit
데이터 포인트들을 클러스터에 할당

```
fit(X)
{
  centroids ← random_init(X)
  for iter ← 1 to max_iter
    assign_cluster(X)
    update_centroids(X)

  if (calc_diff(prev_centroids, centroids) < tol) then break

  return assignments, centroids
}
```

assign_cluster
클러스터 지정

```
assign_cluster(X)
{
  assignments ← []
  for d in X :
    min_dist ← ∞
    min_index ← -1
    for i, centroid in enumerate(centroids)
      dist = euclidean(d, centroid)
      if (dist < min_dist)  then min_dist ← dist; min_index ← i
    assignnments ← min_index
}
```

update_centroids
클러스터 중심섬 수정

```
update_centroids(X)
{
  prev_centroids ← centroids
  for i ← 1 to k
    data_indices ← []
    for k ← 1 to len(assignemtns)
      if (assignments[k] == i) data_indices ← k

  if (data_indices = ∅)
    then r ← randint(0, len(X); centroids[i] ← X[r]
    continue

    cluster_data ← []
  for index ∈ data_indices
    cluster_data ← X[index]
    centroids[i] ← calc_vector_mean(cluster_data)
}
```

03 Alpha, Beta 추정 알고리즘

④ 매개변수 추정

i) β 선택하기

- d_i = (x_i와 가장 가까운 클러스터 사이의 거리)
- $d_i > \mu + \delta\sigma$ 이면, x_i는 이상치로 간주 (di에 대한 값)
- $\delta = 6$일 때 가장 이상적인 추정치 도출

ii) α 선택하기

경우 a 를 사용

ⓐ 겹치는 부분이 작은 경우에 더 적합

- 각 클러스터 C_j에 대해, 그 안에 속한 인스턴스들과 중심점 사이의 거리를 계산해서 μ, σ 도출
- C_j에 속하지 않는 임의의 인스턴스 x_ℓ과 C_j의 중심점 사이의 거리를 $d_{\ell j}$라 할 때,
- $d_{\ell j} < \mu_j + \delta\sigma_j$ 이면, x_ℓ은 overlapped region에 위치한 것으로 간주
- $-1 \le \delta \le 3.5$로 설정

ⓑ 겹치는 부분이 큰 경우

- d_{ij} 정규화 $\longrightarrow$ $\bar{d}_{ij} = \dfrac{d_{ij}}{\sum_{\ell=1}^{k} d_{\ell j}}$ $\left(\sum_{\ell=1}^{k} \bar{d}_{i\ell} = 1\right)$
- $\bar{d}_{ij} < 1/(k+1)$을 만족하는 개수 세기

인스턴스 x_i가 모든 C_j와의 거리가 같을 때, 모든 j에 대해 $d_{ij} = 1/k$

```
estimate_alpha_beta(X, C, alpha_delta, beta_delta)
{
  n ← no. of data points
  k ← no. of centroids
  D ← [0]_n*k

  for i ← 1 to k
    diff ← X - tile(C[j, :], (n, 1))
    D[:, j] ← sqrt(sum(diff**2))

  dist  ← min d_ij
  ind  ←  argmin d_ij

  threshold ← (mean(dist) + beta_delta * std(dist)
  betaN ← count(dist > threshold == True)
  beta ← betaN / n

  overlap ← 0
  for j ← 1 to k
    if count((ind == j) == True)> 0
      then cdist ← dist[ind == j]
      threshold ← (mean(cdist) + beta_delta * std(cdist)
      v ← D[ind ∉ j  , j] < threshold
      overlap += count(v == True)

  alpha ← overlap / n
  return alpha, beta
}
```

03 NEO K-means 알고리즘

Algorithm 1 NEO-K-Means

Input: $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n\}$, the number of clusters k, the maximum number of iterations t_{max}, α, β

Output: $\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k$

Initialize cluster means $\{\mathbf{m}_j\}_{j=1}^{k}$, $t = 0$.

while not converged and $t < t_{max}$ **do**

  Compute cluster means, and then compute distances between every data point and clusters $[d_{ij}]_{n \times k}$.

  Initialize $\mathcal{T} = \emptyset$, $\mathcal{S} = \emptyset$, $p = 0$, and $\bar{\mathcal{C}}_j = \emptyset$, $\hat{\mathcal{C}}_j = \emptyset \; \forall j$.

  while $p < (n + \alpha n)$ **do**

    if $p < (n - \beta n)$ **then**

      Assign $\mathbf{x}_{i^*}$ to $\bar{\mathcal{C}}_{j^*}$ such that $(i^*, j^*) = \underset{i,j}{\operatorname{argmin}}\, d_{ij}$ where $\{(i,j)\} \notin \mathcal{T}, i \notin \mathcal{S}$.

      $\mathcal{S} = \mathcal{S} \cup \{i^*\}$.

    else

      Assign $\mathbf{x}_{i^*}$ to $\hat{\mathcal{C}}_{j^*}$ such that $(i^*, j^*) = \underset{i,j}{\operatorname{argmin}}\, d_{ij}$ where $\{(i,j)\} \notin \mathcal{T}$.

    end if

    $\mathcal{T} = \mathcal{T} \cup \{(i^*, j^*)\}$.

    $p = p + 1$.

  end while

  $\forall j$, update clusters $\mathcal{C}_j = \bar{\mathcal{C}}_j \cup \hat{\mathcal{C}}_j$.

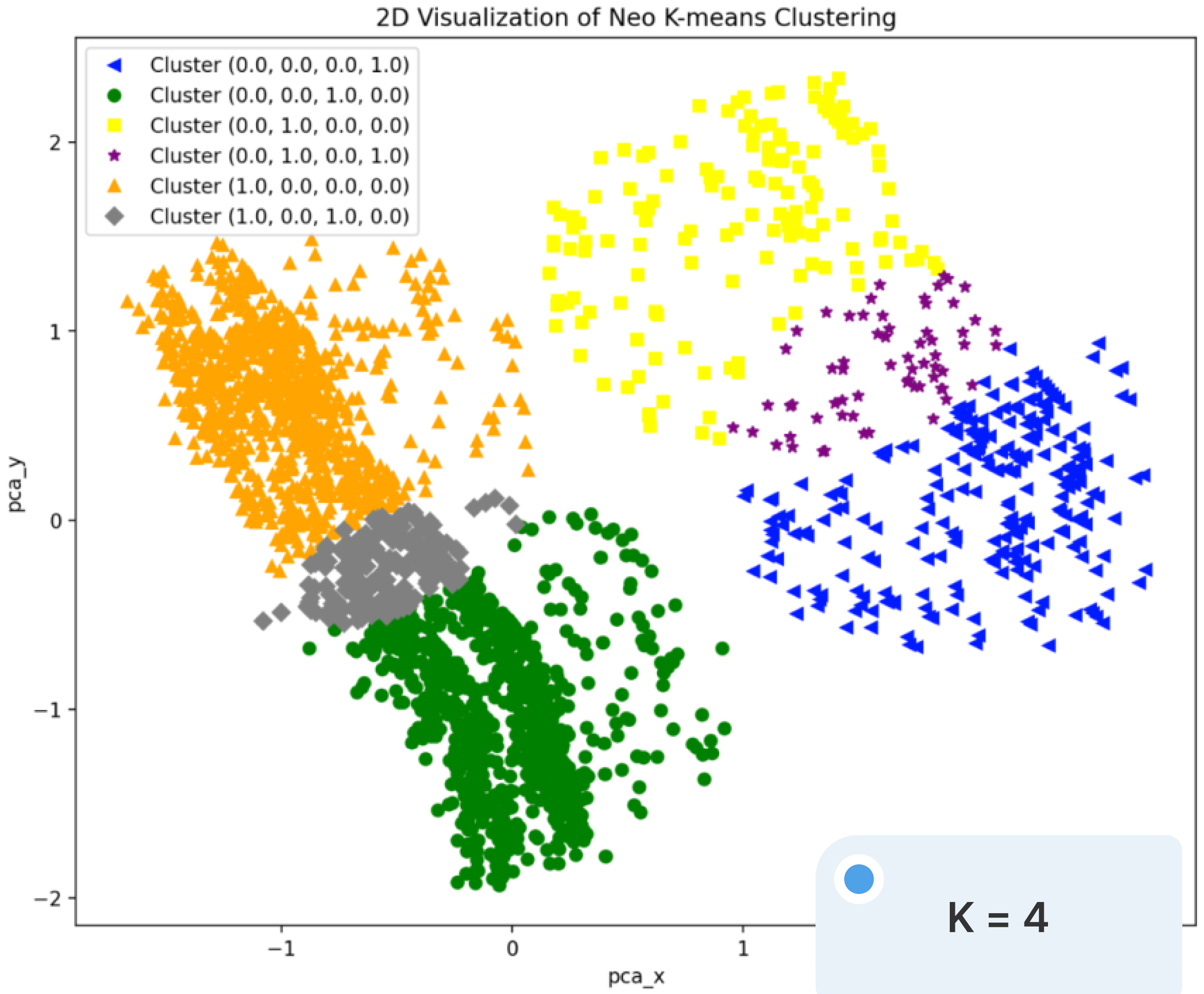
  $t = t + 1$.

end while

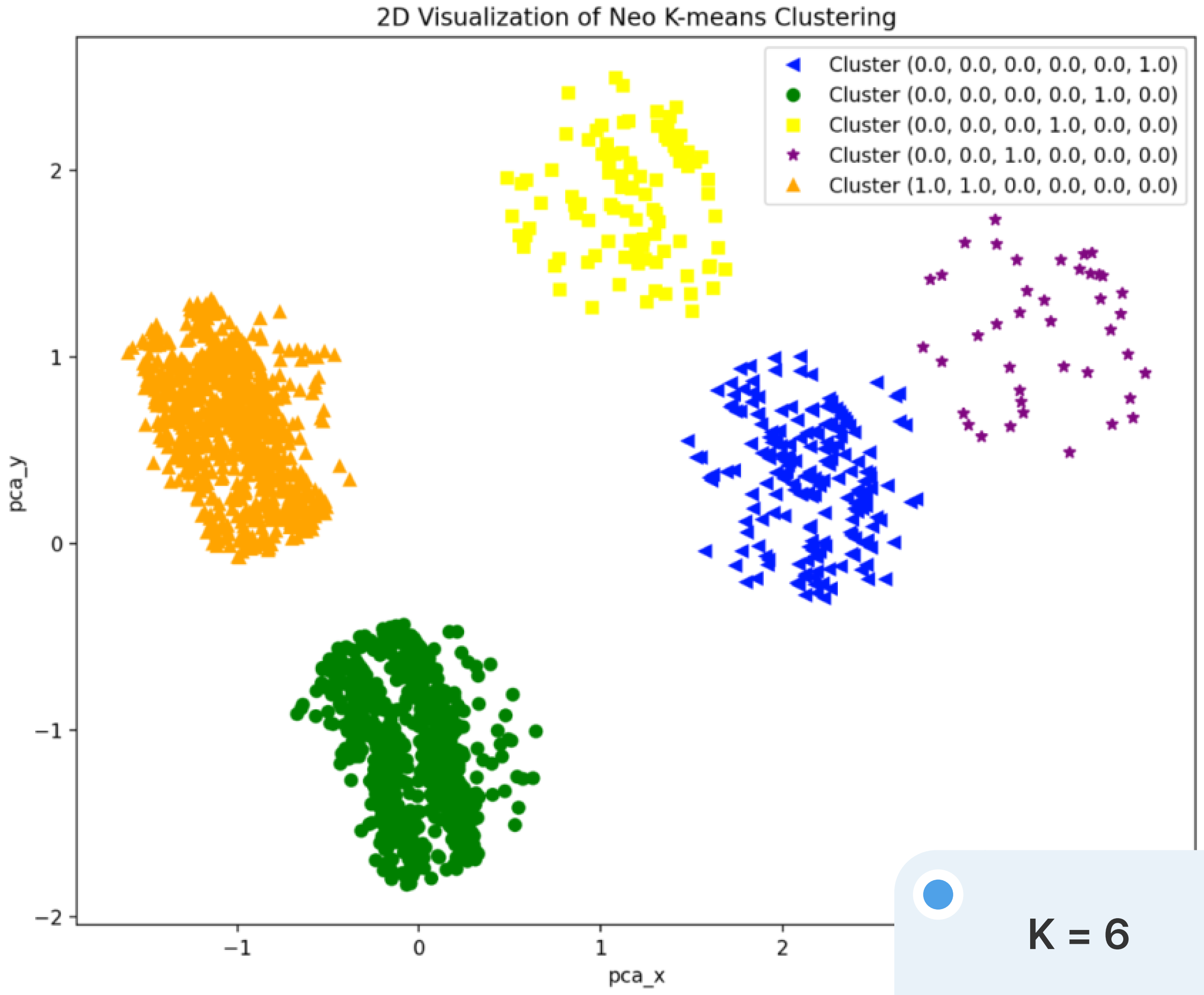
```
neo_kmeans(X, k, alpha, beta, initU)
{
  N ← no. of data points
  dim ← dim of data points
  U ← initU

  t ← 0
  t_max ← 100
  alphaN ← alpha * N
  betaN ← beta * N

  J ← ∞
  oldJ ← 0
  epsilon ← 0.5

  D ← [0]_n*k
  M ← [0]_dim*k

  while not converged and t < t_max
    odlJ ←  J
    J ← 0

    for j ← 1 to k
      M[:, j] ← mean(X[ind ∈ cluster j, :], axis=0) # j에 속하는 index에 대해

    for j ← 1 to k
      diff ← X - tile(M[:, j].reshape(1, -1), (N, 1))
      D[:, j] ← sum(diff**2)

    ind ← argpartition(D, N - betaN)[:N - betaN]
    sorted_n, sorted_k ← unravel_index(ind, D.shape)
    sorted_d ← D[sorted_n, sorted_k]

    M ← [0]_n*k

    U[sorted_n[:N-betaN], sorted_k[:N-betaN]] ← 1
    D[sorted_n[:N-betaN], sorted_k[:N-betaN]] ← ∞
    J += sum(sorted_d[:num_assign])

    n ← 0
    while n < alphaN + betaN
      min_d ← min d_ij
      J += min_d
      i_star, j_star ← argmin d_ij
      U[i_star[0], j_star[0]] ← 1
      D[i_star[0], j_star[0]] ← ∞

    t += 1
}
```

04 클러스터링 결과 시각화

PCA → 2차원으로 축소



K = 4



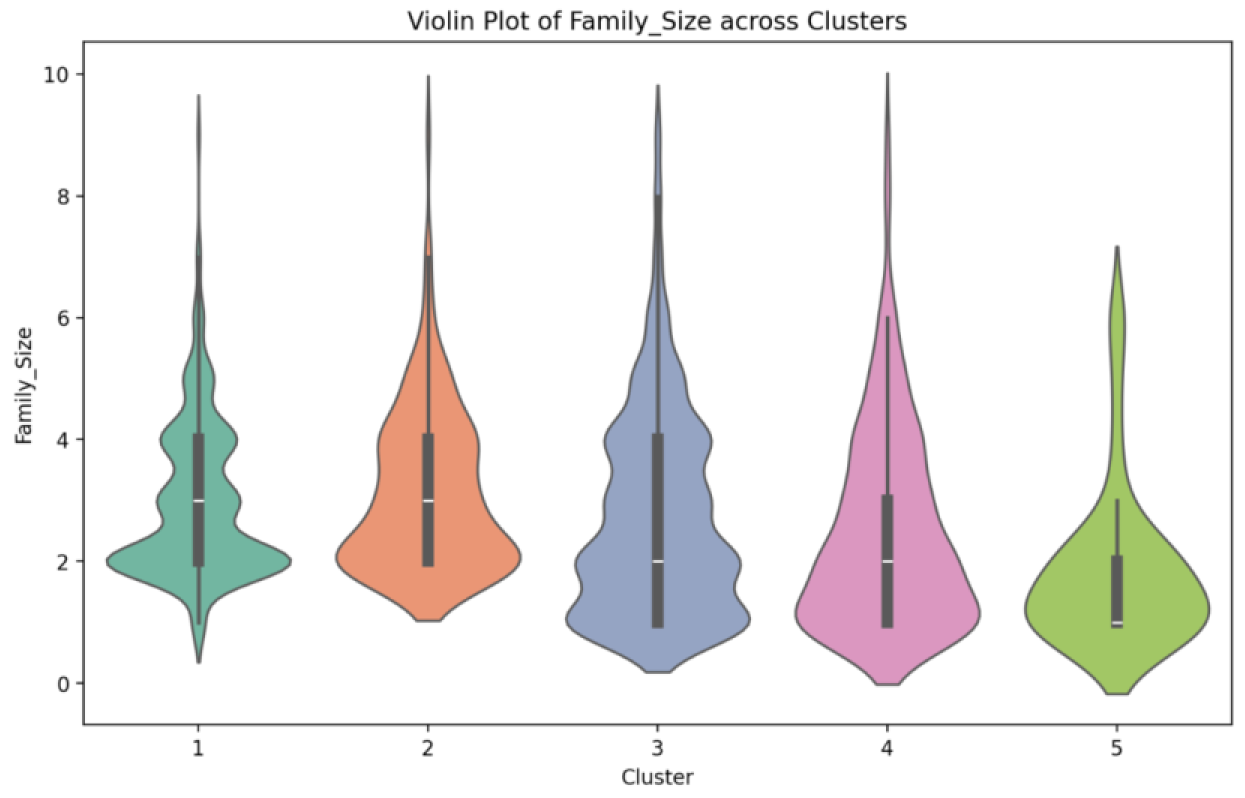
K = 6

04 클러스터별 특징 시각화

01 연속형 변수 클러스터별 특징 시각화 Violin Plot

확인하고 싶은 변수 선택

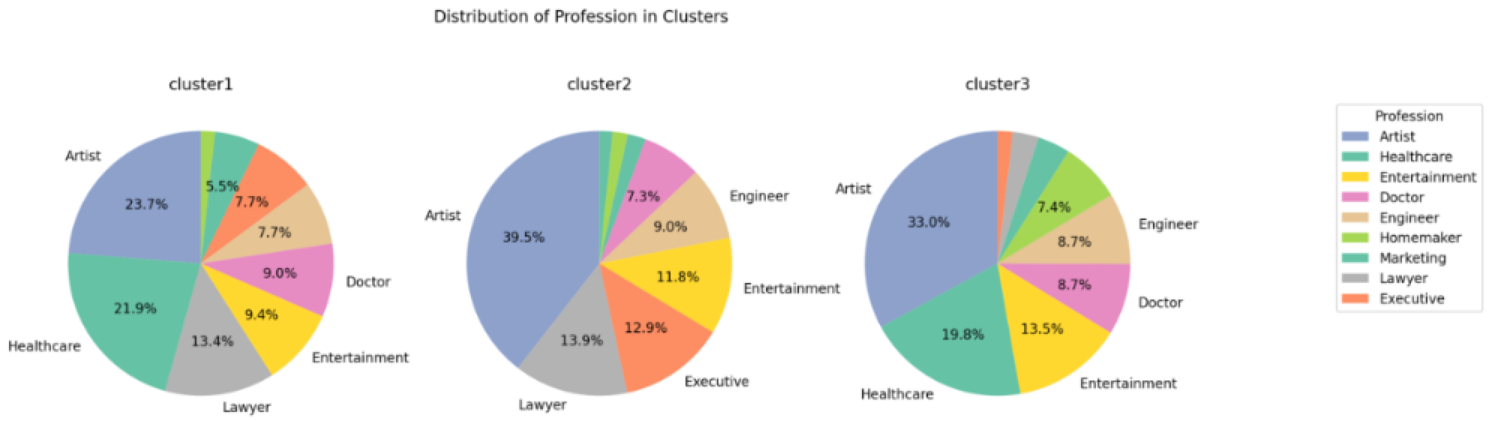
Family_Size



02 범주형 변수 클러스터별 특징 시각화 Pie Plot

확인하고 싶은 변수 선택

Profession



04 데이터 전처리

Step.1

결측치 처리

결측치가 있는 데이터도 한 고객을 의미하므로 이 고객이 어느 클러스터에 속하는지 알 필요 존재
→ 결측행 제거가 아닌 결측값 대체 방법을 사용

범주형은 Mode Imputation,
연속형은 Median Imputation

Step.2

레이블 인코딩

범주형 데이터를 레이블 인코딩을 하면 순서와는 관계 없는 변수에 순서가 생긴다는 단점 존재

그러나 사용자는 이를 인지하고 원하는 변수만 선택할 수 있으므로 레이블 인코딩을 진행

Step.3

Robust Scaling

클러스터링 알고리즘 특성 상 거리에 민감하고, 변수마다 단위가 다르므로 스케일링을 진행

이상치 제거를 하지 않기 때문에 이상치에서도 다른 scaler에 비해 안정적인 robust scaler를 사용

05 Streamlit 구현 및 배포

Streamlit

https://streamlit.io/

서비스 UI 구현

파이썬으로 간단히 구현 가능한 Streamlit 프레임워크를 사용하여 UI 를 구현함.
앞서 구현한 NEO K-means가 작동하도록 만듦.

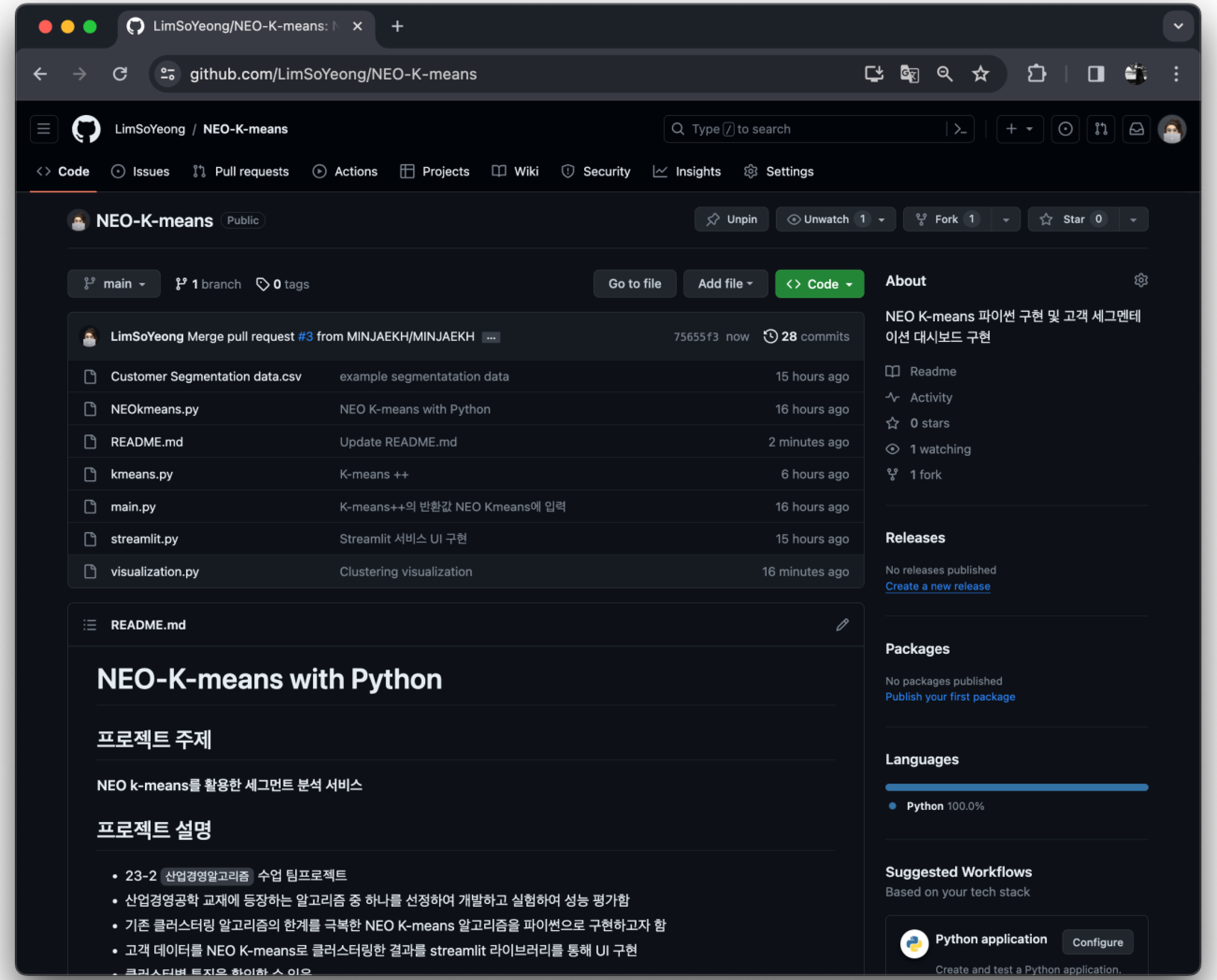
06 Github

자세한 구현 코드는 아래 깃허브 페이지에서 확인 가능합니다.

https://github.com/LimSoYeong/NEO-K-means

파일 설명

- **NEOK.py**
  kmeas.py에서 initU, Beta, Alpha 값 추정한 것을 입력값으로 사용하여 NEO K-means 알고리즘 구현
- **kmeans.py**
  Kmeans ++ 알고리즘 구현
- **main.py**
  streamlit에서 NEOK와 kmeans를 사용할 수 있도록 연결
- **streamlit.py**
  streamlit 구현 → UI 서비스 화면
- **visualization.py**
  데이터 전처리 과정, 클러스터링 결과 시각화, 클러스터별 특징 시각화



NEO k-means를 활용한 세그먼트 분석 서비스 개발

Thank You

2021104011 임소영
2020108968 차민재