목차

1. KMeans Clustering 클래스 구현

★ Class로 구현한 KMeans

- KMeans 입력 parameter

✓ **k** : 클러스터 개수
✓ **tolerance = 1e-04** : 종료조건 1
✓ **max_iter = 300** : 최대 반복 횟수(종료조건 2)

★ KMeans model fit 과정 1

1. Dataset을 numpy array구조로 변환
2. k개의 클러스터에 데이터 랜덤 할당
3. 초기 중심점, 평가값 계산

★ KMeans model fit 과정 2

- 종료조건이 될 때까지 알고리즘 과정 반복

★ 최종 결과값 저장

1

```python
class KMeans:
    def __init__(self, k, tolerance = 1e-04, max_iter = 300):
        self.k = k
        self.tolerance = tolerance
        self.max_iter = max_iter
        self.centroids = None
        self.clusters = None
        self.cluster_labels = None
        self.evaluation = None
        self.iter_num = None
```

2

```python
    def fit(self, dataset):
        X = np.array(dataset)

        # k개의 클러스터에 데이터 랜덤 할당
          # 각 데이터에 대해 랜덤한 클러스터 선택
        cluster_labels = np.random.choice(self.k, size=len(X))
          # 각 클러스터에 속하는 데이터들을 저장할 빈 리스트 생성
        clusters = [[] for _ in range(self.k)]

          # 모든 데이터에 대해, 해당하는 클러스터 리스트에 추가
        for i in range(len(X)):
            clusters[cluster_labels[i]].append(X[i])

          # numpy array로 변환
        clusters = [np.array(cluster) for cluster in clusters]

        centroids = self.Centroids(clusters)
        distance_sums = self.Distance_sums(clusters, centroids)
        evaluation = self.Evaluation(distance_sums)
```

3

```python
        new_centroids = np.zeros(centroids.shape)
        iter_num = 0
        error = np.ones((self.k, X.shape[1]))

        while np.any(error != self.tolerance) and iter_num < self.max_iter:
            iter_num += 1

            clusters, cluster_labels = self.New_Cluster(X, centroids)
            new_centroids = self.Centroids(clusters)
            error = self.Error(centroids, new_centroids)
            centroids = deepcopy(new_centroids)

            distance_sums = self.Distance_sums(clusters, centroids)
            evaluation = self.Evaluation(distance_sums)

        self.centroids = centroids
        self.clusters = clusters
        self.cluster_labels = cluster_labels
        self.evaluation = evaluation
        self.iter_num = iter_num
```

1. KMeans Clustering 클래스 구현

★ Class 안에서 사용되는 함수

- **Centroids(self, clusters)** : 중심점 계산

★ Class 안에서 사용되는 함수

- **Distance_sums(self, clusters, centroids)** : 각 클러스터에서 데이터와 중심점과의 거리 계산
- **Evaluation(self, distance_sums)** : 거리를 모두 합하여 평가값 계산

★ Class 안에서 사용되는 함수

- **New_Cluster(self, X, centroids)** : 인접 클러스터로 데이터를 재배치하여 새로운 클러스터 해 생성
- **Error(self, centroids, new_centroids)** : 이전 클러스터 해의 중심점과 새로운 클러스터 해의 중심점의 차이 계산

4

```
def Centroids(self, clusters):
    centroids = []
    for cluster in clusters:
        if len(cluster) > 0:
            centroids.append(np.mean(cluster, axis=0))
        else:
            # 빈 클러스터일 경우 임의의 값으로 대체하거나 건너뛸 수 있습니다.
            # 여기서는 NaN으로 설정합니다.
            centroids.append(np.nan)

    centroids = np.array(centroids)
    return centroids
```

5

```
def Distance_sums(self, clusters, centroids):
    distance_sums = []

    for i in range(len(clusters)):
        cluster = clusters[i]
        centroid = centroids[i]

        if len(cluster.shape) == 1:
            distance_sum = np.sqrt(np.sum((cluster - centroid)**2))
        else:
            distance_sum = np.sum(np.sqrt(np.sum((cluster - centroid)**2, axis=1)))

        distance_sums.append(distance_sum)

    return distance_sums

def Evaluation(self, distance_sums):
    evaluation = np.sum(distance_sums)
    return evaluation
```

6

```
def New_Cluster(self, X, centroids):
    cluster_labels = np.zeros(len(X))
    new_clusters = [[] for _ in range(len(centroids))]

    for i, data in enumerate(X):
        distances = [np.sqrt(np.sum((data - centroid)**2)) for centroid in centroids]
        closest_centroid_index = np.argmin(distances)
        new_clusters[closest_centroid_index].append(data)
        cluster_labels[i] = closest_centroid_index

    new_clusters = [np.array(cluster) for cluster in new_clusters]
    return new_clusters, cluster_labels

def Error(self, centroids, new_centroids):
    k = len(centroids)
    error = np.zeros((k, centroids.shape[1]))

    for i in range(k):
        error[i] = np.sqrt(np.sum((centroids[i] - new_centroids[i])**2))

    return error
```

1. KMeans Clustering 클래스 구현

★ Class 안에서 사용되는 함수

- **get_evaluation(self)** : **최종 평가값** 반환
- **get_clusters(self)** : **최종 클러스터 해** 반환
- **get_cluster_labels(self)** : **최종 데이터들의 클러스터 label** 반환
- **get_centroids(self)** : **최종 중심점** 반환
- **get_iteration_count(self)** : **최종 반복횟수** 반환

7

```
def get_evaluation(self):
    return self.evaluation

def get_clusters(self):
    return self.clusters

def get_cluster_labels(self):
    return self.cluster_labels

def get_centroids(self):
    return self.centroids

def get_iteration_count(self):
    return self.iter_num
```

2. KSA Clustering 구현

★ 평가값 계산에 필요한 함수 정의(KMeans와 동일)

- **Centroids(clusters)** : 중심점 계산
- **Distance_sums(clusters, centroids)** : 각 클러스터에서 데이터와 중심점과의 거리 계산
- **Evaluation(distance_sums)** : 거리를 모두 합하여 평가값 계산

★ 평가값 계산 함수

- **evaluate_solution(clusters)** : **clusters 변수**를 넣어 **Centroids**, **Distance_sums**, **Evaluation**함수를 한 번에 계산하여 평가값 반환

1

```python
def Centroids(clusters):
    centroids = []
    for cluster in clusters:
        if len(cluster) > 0:
            centroids.append(np.mean(cluster, axis=0))
        else:
            # 빈 클러스터일 경우 임의의 값으로 대체하거나 건너뛸 수 있습니다.
            # 여기서는 NaN으로 설정합니다.
            centroids.append(np.nan)

    centroids = np.array(centroids)
    return centroids

def Distance_sums(clusters, centroids):
    distance_sums = []

    for i in range(len(clusters)):
        cluster = clusters[i]
        centroid = centroids[i]

        if len(cluster.shape) == 1:
            distance_sum = np.sqrt(np.sum((cluster - centroid)**2))
        else:
            distance_sum = np.sum(np.sqrt(np.sum((cluster - centroid)**2, axis=1)))

        distance_sums.append(distance_sum)

    return distance_sums

def Evaluation(distance_sums):
    evaluation = np.sum(distance_sums)
    return evaluation
```

2

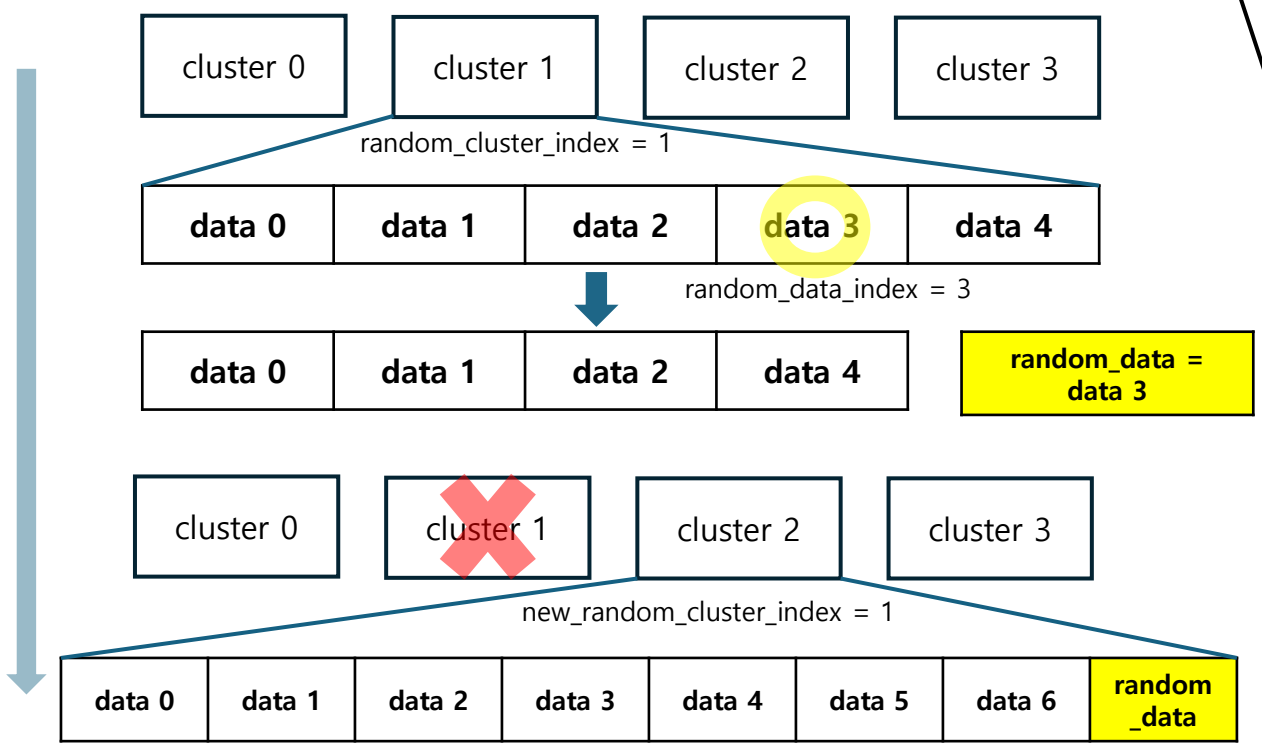
```python
def evaluate_solution(clusters):
    return Evaluation(Distance_sums(clusters, Centroids(clusters)))
```

2. KSA Clustering 구현

★ 이웃해를 구하는 함수

- **neighborhood_cluster(clusters)**

과정



3

```python
# 이웃해를 구하는 함수
def neighborhood_cluster(clusters):
    non_empty_clusters = []
    empty_clusters = []

    # 빈 클러스터에 대한 처리
    for i in range(len(clusters)):
        if len(clusters[i])==0:
            empty_clusters.append(clusters[i])
        else:
            non_empty_clusters.append(clusters[i])

    # 랜덤으로 클러스터 1개 선택
    random_cluster_index = random.randint(0, len(non_empty_clusters)-1)

    # 선택된 클러스터에서 랜덤으로 1개의 데이터 선택
    random_data_index = random.randint(0, len(non_empty_clusters[random_cluster_index]) - 1)
    random_data = non_empty_clusters[random_cluster_index][random_data_index]

    # 원래 클러스터에서 데이터 삭제
    non_empty_clusters[random_cluster_index]  = np.delete(non_empty_clusters[random_cluster_index], random_data_index, 0)

    # 이전에 선택한 클러스터를 제외한 다른 클러스터를 선택하기 위한 처리
    available_clusters = []
    if len(empty_clusters) > 0:  # 빈 클러스터가 있던 경우
        available_clusters.append(empty_clusters)
        for cluster in non_empty_clusters:
            if cluster is not non_empty_clusters[random_cluster_index]:  # 이전에 선택한 클러스터는 제외
                available_clusters.append(cluster)
    else:
        for cluster in non_empty_clusters:
            if cluster is not non_empty_clusters[random_cluster_index]:  # 이전에 선택한 클러스터는 제외
                available_clusters.append(cluster)

    # 새로 넣을 클러스터를 랜덤으로 선택
    new_random_cluster_index = random.randint(0, len(available_clusters)-1)

    # 새로운 클러스터에 데이터 추가
    available_clusters[new_random_cluster_index] = np.append(available_clusters[new_random_cluster_index], [random_data], axis = 0)

    # 이전 클러스터를 클러스터 리스트에 추가
    available_clusters.append(non_empty_clusters[random_cluster_index])

    # 클러스터 갱신
    clusters = available_clusters

    return clusters
```

2. KSA Clustering 구현

★ KSA Clustering 함수

- KSA_Clustering 입력 parameter

✓ **k** : 클러스터 개수
✓ **dataset** : 사용할 데이터셋
✓ **max_iter = 300** : 최대 반복 횟수(종료조건 1)
✓ **initial_temperature = 100** : 초기 온도(T)
✓ **delta_t = 0.01** : 온도 감소량(ΔT)
✓ **final_temparature = 1e-04** : 최소 온도(종료조건 2)
✓ **t = 20000** : 이웃해 탐색 횟수

★ KMeans 결과값으로 초기 Setting

- 초기 **best_solution** = kmeans의 최종 **클러스터 해**
- 초기 **best_solution_evaluation** = kmeans의 최종 **평가값**

1

```python
def KSA_Clustering(k, dataset, max_iter = 300, initial_temperature = 100, delta_t = 0.01, final_temperature = 1e-04, t = 2000):
    while True:
        try:
            T = initial_temperature
            kmeans = KMeans(k)
            kmeans.fit(dataset)
            best_solution = kmeans.get_clusters()  # 초기 clusters는 kmeans의 결과 clusters
            iterations = 0
            best_solution_evaluation = kmeans.get_evaluation()  # 초기 best_solution은 처음 kmeans를 돌린 평가값
            evaluations = []

            while T >= final_temperature and iterations <= max_iter:
                # 이웃해 생성
                for i in range(t):
                    neighbor_solution = neighborhood_cluster(best_solution)

                    # 이웃해 평가
                    neighbor_solution_evaluation = evaluate_solution(neighbor_solution)

                    # 좋은 이웃해는 항상 받아들임
                    if neighbor_solution_evaluation < best_solution_evaluation:
                        best_solution = neighbor_solution
                        best_solution_evaluation = neighbor_solution_evaluation

                    # 나쁜 해는 확률적으로 받아들임
                    elif random.uniform(0, 1) < np.exp((best_solution_evaluation - neighbor_solution_evaluation) / T):
                        best_solution = neighbor_solution
                        best_solution_evaluation = neighbor_solution_evaluation

                # 온도 감소
                T -= delta_t
                iterations += 1
                evaluations.append(best_solution_evaluation)

            # 시각화
            plt.plot(range(0, iterations), evaluations, marker='o', linestyle='-')
            plt.title('Evaluation over Iterations')
            plt.xlabel('Iteration')
            plt.ylabel('Evaluation')
            plt.grid(True)
            plt.show()

            # 중심점
            best_centorids = Centroids(best_solution)

            return best_solution, best_solution_evaluation, best_centorids, iterations

        except ValueError as e:
            print("Error occurred:", e)
            print("Retrying the function...")
            continue
```

Kmeans 적용

2. KSA Clustering 구현

★ SA 과정

- 종료조건(둘 중 하나만 만족해도 종료)

1. **T(초기 온도) < final_temperature(최소 온도)**
2. **iterations(반복횟수) > max_iter(최대 반복횟수)**

---

t번 반복

1. 이웃해 생성
2. 이웃해 평가
3. [Minimize] 이웃해와 현재해를 비교하여 현재해 갱신

- 이웃해 < 현재해 -> 항상 받아들임
- rand(0, 1) < exp{-(이웃해–현재해) / T} -> 확률적으로 받아들임

- 온도 감소
- 반복 횟수 + 1
- iteration 1번마다 평가값 저장

★ 함수 최종 반환값

- **best_solution** : 최종 클러스터 해
- **best_solution_evaluation** : 최종 평가값
- **best_centroids** : 최종 클러스터의 중심점
- **iterations** : 최종 반복횟수

2

```
def KSA_Clustering(k, dataset, max_iter = 300, initial_temperature = 100, delta_t = 0.01, final_temperature = 1e-04, t = 2000):
  while True:
    try:
      T = initial_temperature
      kmeans = KMeans(k)
      kmeans.fit(dataset)
      best_solution = kmeans.get_clusters()  # 초기 clusters는 kmeans의 결과 clusters
      iterations = 0
      best_solution_evaluation = kmeans.get_evaluation()  # 초기 best_solution은 처음 kmeans를 돌린 평가값
      evaluations = []

      while T >= final_temperature and iterations <= max_iter:
          # 이웃해 생성
          for i in range(t):
              neighbor_solution = neighborhood_cluster(best_solution)

              # 이웃해 평가
              neighbor_solution_evaluation = evaluate_solution(neighbor_solution)

              # 좋은 이웃해는 항상 받아들임
              if neighbor_solution_evaluation < best_solution_evaluation:
                  best_solution = neighbor_solution
                  best_solution_evaluation = neighbor_solution_evaluation

              # 나쁜 해는 확률적으로 받아들임
              elif random.uniform(0, 1) < np.exp((best_solution_evaluation - neighbor_solution_evaluation) / T):
                  best_solution = neighbor_solution
                  best_solution_evaluation = neighbor_solution_evaluation

          # 온도 감소
          T -= delta_t
          iterations += 1
          evaluations.append(best_solution_evaluation)
      # 시각화
      plt.plot(range(0, iterations), evaluations, marker='o', linestyle='-')
      plt.title('Evaluation over Iterations')
      plt.xlabel('Iteration')
      plt.ylabel('Evaluation')
      plt.grid(True)
      plt.show()

      # 중심점
      best_centorids = Centroids(best_solution)

      return best_solution, best_solution_evaluation, best_centorids, iterations

    except ValueError as e:
      print("Error occurred:", e)
      print("Retrying the function...")
      continue
```

3. 데이터셋 Test 결과

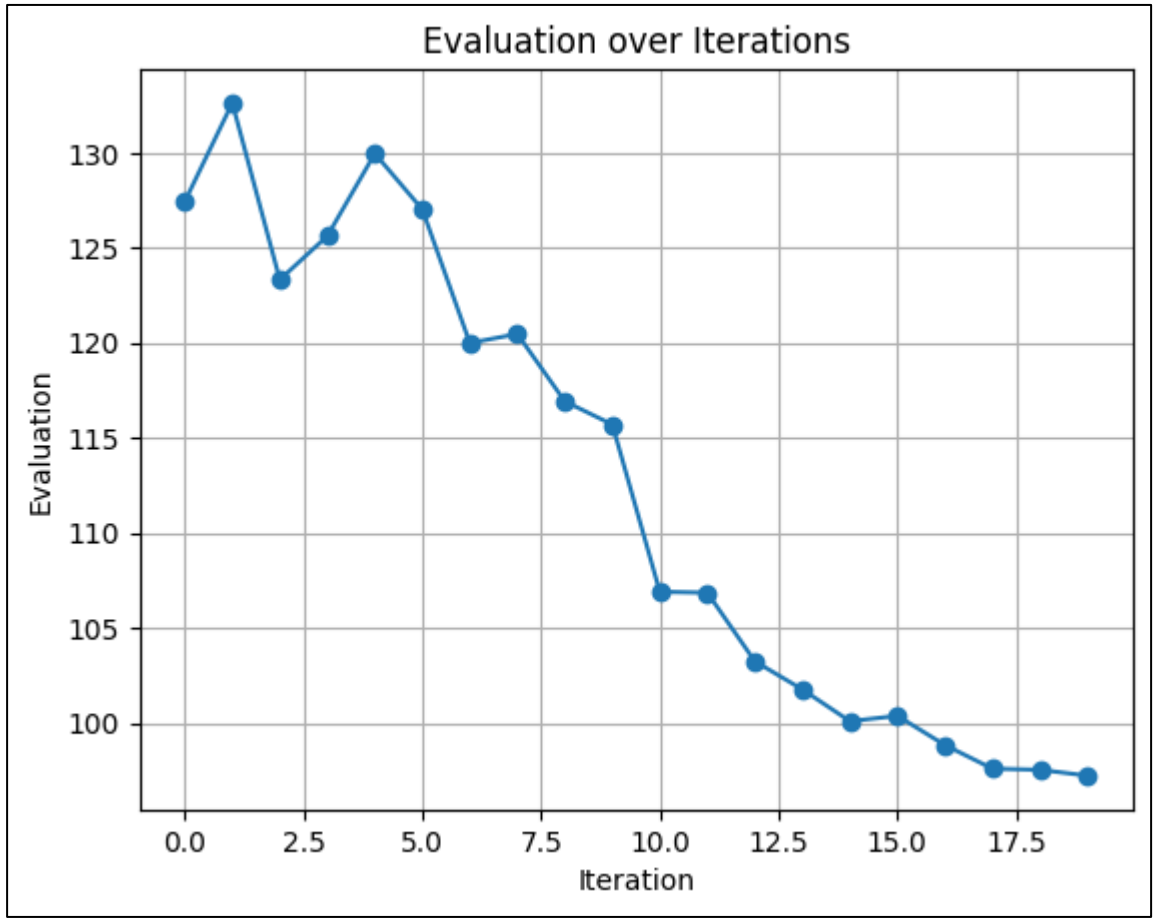
3.1. Iris

★ iris 데이터셋 적용 parameter

- ✓ **k = 3**
- ✓ **dataset = iris**
- ✓ **max_iter = 300**
- ✓ **initial_temperature = 1**
- ✓ **delta_t = 0.05**
- ✓ **final_temparature = 1e-04**
- ✓ **t = 1500**

★ 결과

- Final Evaluation = **97.22212765100771**
- Total Iterations: 20



3. 데이터셋 Test 결과

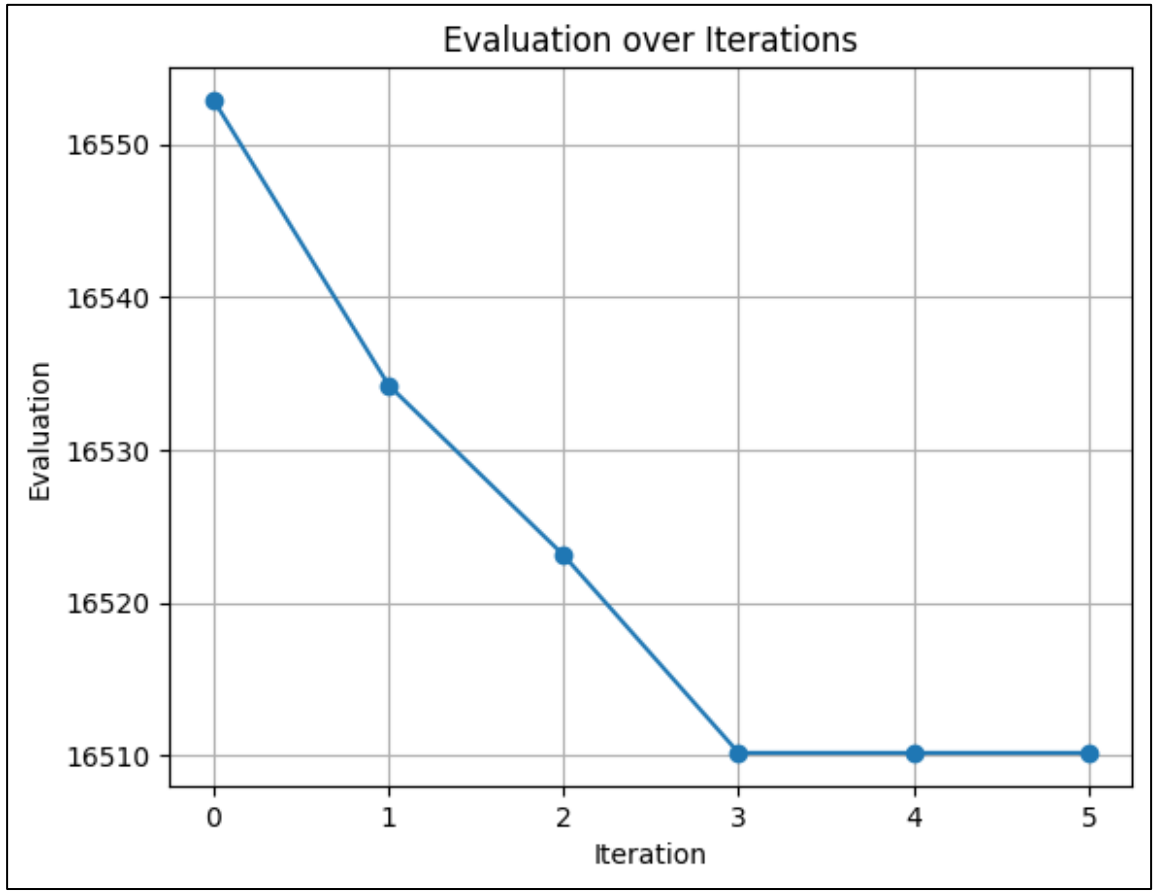
3.2. Wine

★ wine 데이터셋 적용 parameter

- ✓ **k = 3**
- ✓ **dataset = wine**
- ✓ **max_iter = 300**
- ✓ **initial_temperature = 6**
- ✓ **delta_t = 1**
- ✓ **final_temparature = 1e-04**
- ✓ **t = 1780**

★ 결과

- Final Evaluation = **16510.196677408763**
- Total Iterations: 6



3. 데이터셋 Test 결과

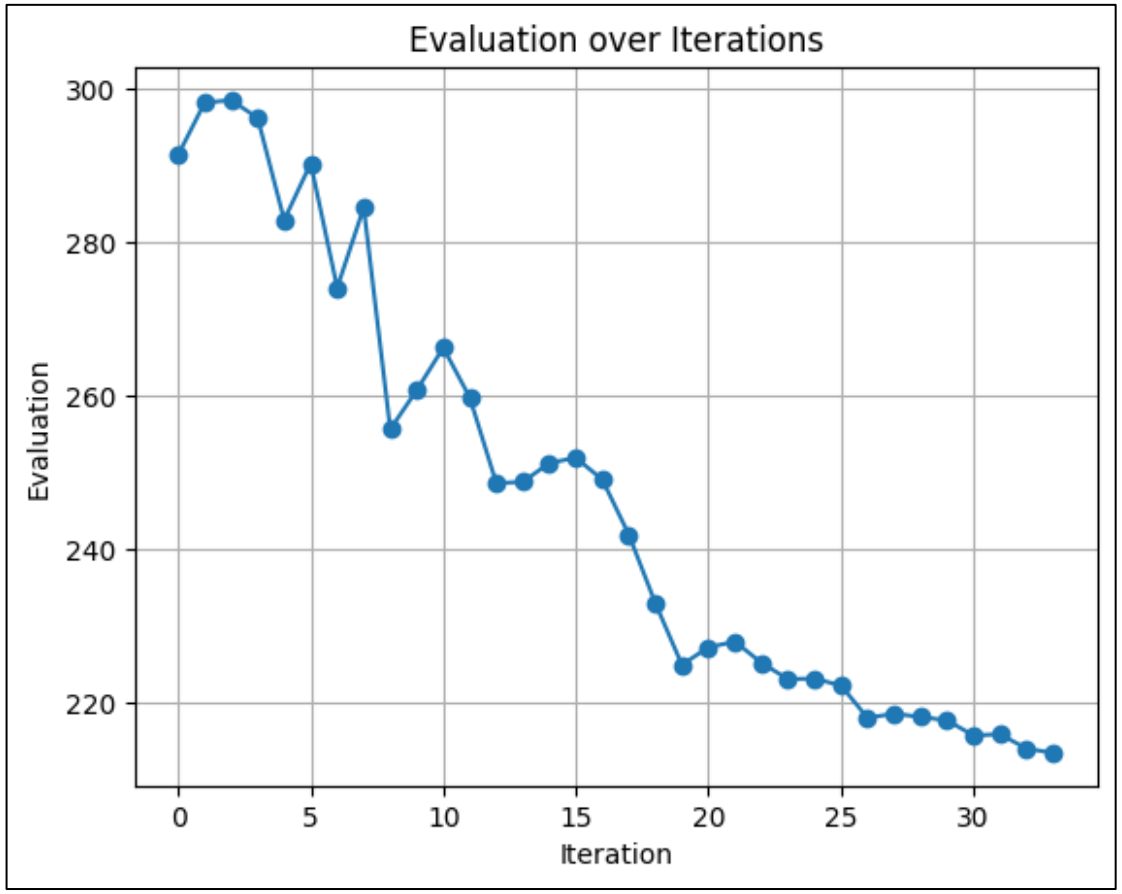
3.3. Glass

★ glass 데이터셋 적용 parameter

- ✓ **k = 6**
- ✓ **dataset = glass**
- ✓ **max_iter = 300**
- ✓ **initial_temperature = 1**
- ✓ **delta_t = 0.03**
- ✓ **final_temparature = 1e-04**
- ✓ **t = 2140**

★ 결과

- Final Evaluation = **213.41597074463465**
- Total Iterations: 34



3. 데이터셋 Test 결과

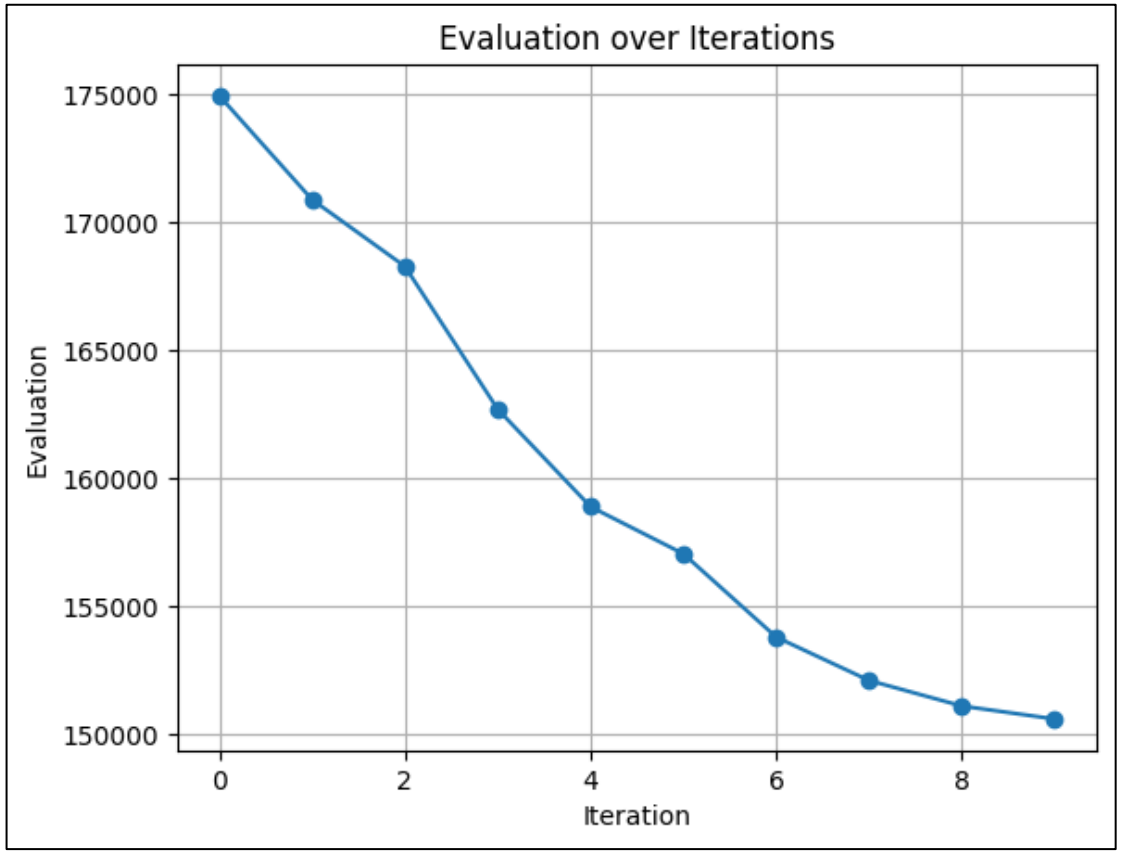
3.4. Vowel

★ vowel 데이터셋 적용 parameter

- ✓ **k = 6**
- ✓ **dataset = vowel**
- ✓ **max_iter = 300**
- ✓ **initial_temperature = 100**
- ✓ **delta_t = 10**
- ✓ **final_temparature = 1e-04**
- ✓ **t = 8710**

★ 결과

- Final Evaluation = **150590.9570138252**
- Total Iterations: 10



3. 데이터셋 Test 결과

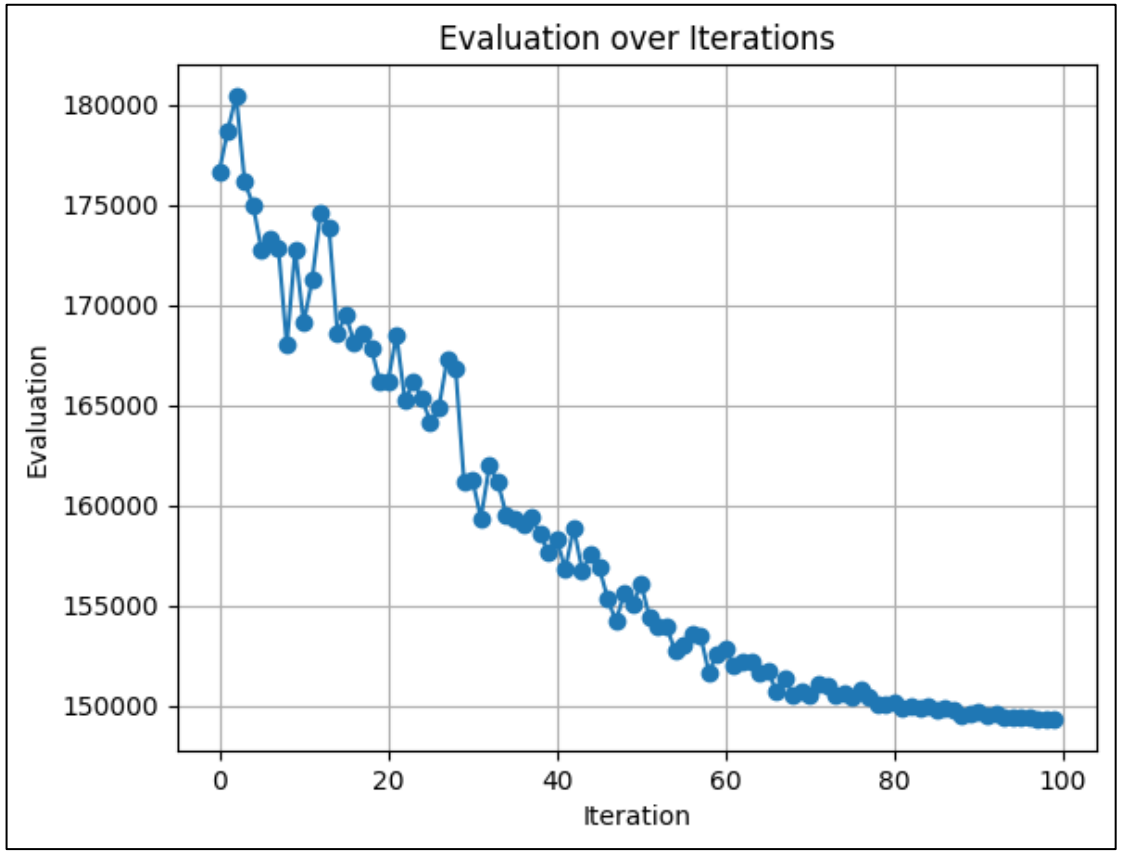
3.4. Vowel

★ vowel 데이터셋 적용 parameter

- ✓ **k = 6**
- ✓ **dataset = vowel**
- ✓ **max_iter = 300**
- ✓ **initial_temperature = 100**
- ✓ **delta_t = 1**
- ✓ **final_temparature = 1e-04**
- ✓ **t = 8710**

★ 결과

- Final Evaluation = **149332.31430808967**
- Total Iterations: 100



3. 데이터셋 Test 결과

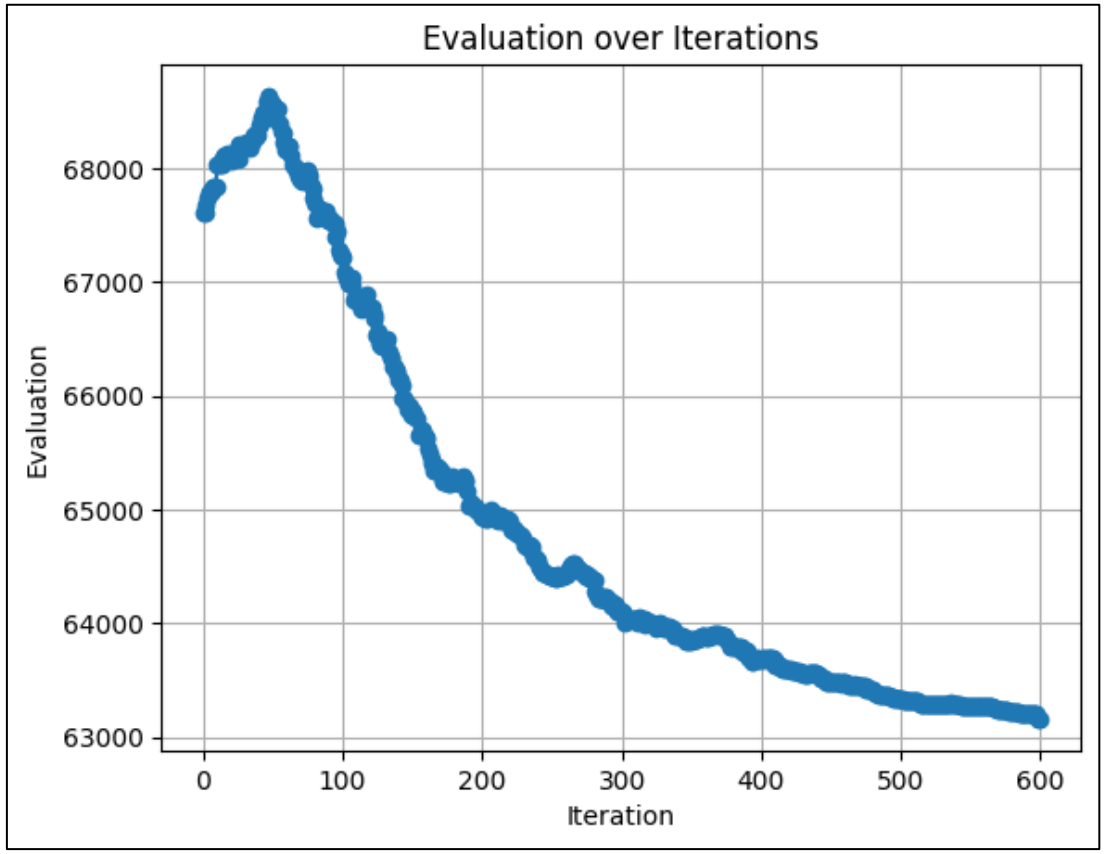
3.5. Cloud

★ cloud 데이터셋 적용 parameter

- ✓ **k = 10**
- ✓ **dataset = cloud**
- ✓ **max_iter = 800**
- ✓ **initial_temperature = 15**
- ✓ **delta_t = 0.025**
- ✓ **final_temparature = 1e-04**
- ✓ **t = 250**

★ 결과

- Final Evaluation = **63157.336553888475**
- Total Iterations: 600



4. 논문 결과와 비교

★ best 기준

	논문 K-means best	논문 SA best	논문 KSA best	Custom KSA
IRIS	97.3259	97.2221	97.2221	97.2221
WINE	16555.7	16530.5	16530.5	16510.2
GLASS	215.678	221.69	214.727	213.416
VOWEL	149384	149407	149405	150591 149332(delta_t 감소)
CLOUD	66194.641	62889.885	62937.95	63157.34

5. Q/A

Q / A