Ahmet GÜR
02200201053
Örgün Öğretim

K-means (Kümeleme Algoritması)

Bu algoritma denetimsiz öğrenme için kullanılan bir makine öğrenimi algoritmasıdır. Veri noktalarının benzerliğine dayalı olarak verileri önceden tanımlanmış sayıda küme (k) içinde gruplama yöntemidir.

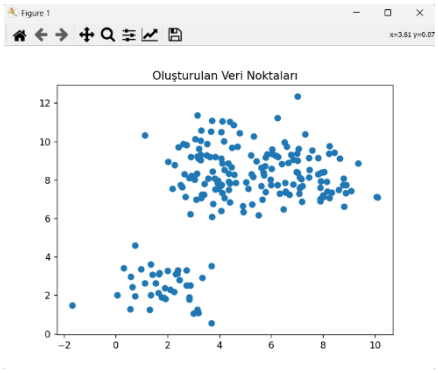
K-means algoritması için aşağıdaki adımları gerçekleştiriz;

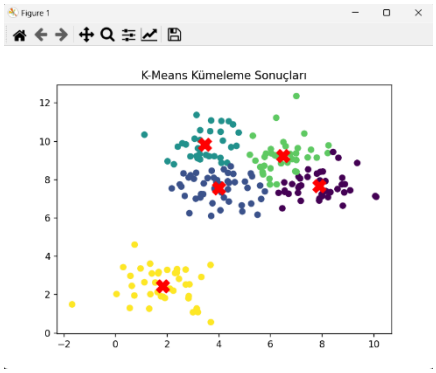
1. Başlangıçta, k küme merkezi rastgele seçilir.
2. Her veri noktası, ona en yakın küme merkezine atanır.
3. Her küme için yeni bir merkez hesaplanır (küme elemanlarının ortalaması alınır).
4. Küme merkezleri artık değişmiyorsa veya belirli bir iterasyon sayısına ulaşıldıysa algoritma sona erer; aksi halde, adımlar 2-3 tekrarlanır.
5. Sonuç olarak, veri noktaları k kümesine atanmış olur.

Kod üzerinde örneği:

```
import numpy as np
import matplotlib.pyplot as plt

def generate_data(n_samples=200, n_features=2, n_centers=5,
random_state=None):
    np.random.seed(random_state)
    data = []

    for _ in range(n_centers):
        center = np.random.rand(n_features) * 10  # Rastgele bir merkez
oluştur
        cluster = center + np.random.randn(n_samples // n_centers,
n_features)  # Merkez etrafında rastgele veri noktaları oluştur
        data.append(cluster)

    return np.vstack(data)

def visualize_data(data):
    plt.scatter(data[:, 0], data[:, 1])
    plt.title('Oluşturulan Veri Noktaları')
    plt.show()

def k_means(data, k=5, max_iters=100, tolerance=1e-4):
    centroids = data[np.random.choice(len(data), k, replace=False)]

    iteration = 0
    while iteration < max_iters:
        distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)
```

```
        new_centroids = np.array([data[labels == i].mean(axis=0) for i in
range(k)])

        if np.linalg.norm(new_centroids - centroids) < tolerance:
            break
        centroids = new_centroids
        iteration += 1

    return labels, centroids

def visualize_results(data, labels, centroids):
    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', s=200,
color='red')
    plt.title('K-Means Kümeleme Sonuçları')
    plt.show()

# Veri oluşturma
data = generate_data(n_samples=200, n_features=2, n_centers=5,
random_state=42)

# Veriyi görselleştirme
visualize_data(data)

# K-Means modelini kullanma
labels, centroids = k_means(data, k=5)

# Sonuçları görselleştirme
visualize_results(data, labels, centroids)
```





K-means dışında birçok farklı kümeleme algoritması bulunmaktadır. Bunlardan bazıları şunlardır;

1. **Hierarchical Clustering (Hiyerarşik Kümeleme)**

   Amacı veri noktalarını hiyerarşik bir yapıda düzenlemektir.

Agglomerative (birleştirici) ve divisive (ayırıcı) olmak üzere iki temel yaklaşım vardır. Agglomerative yöntemde, her veri noktası bir küme olarak başlar ve en yakın küme birleştirilir. Divisive yöntemde, tüm veri noktaları bir küme olarak başlar ve her adımda bir küme iki alt küme olarak ayrılır.

Kod üzerinde örneği:

```
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

data = [[1, 2], [5, 8], [1.5, 1.8], [8, 8], [1, 0.6], [9, 11]]

# Hiyerarşik kümeleme
linkage_matrix = linkage(data, "single")  # "single" linkage kullanılabilir
dendrogram(linkage_matrix)
plt.show()
```

2. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Bu algoritmanın amacı Yoğunluğa dayalı olarak veri noktalarını kümeleme işlemidir. Belirli bir yoğunluk eşiği ve bir mesafe eşiği kullanılır. Yoğun bölgeler bir araya getirilirken, düşük yoğunluktaki bölgeler gürültü olarak ele alınır ve küme sayısı önceden belirtilmez.

Kod üzerinde örneği:

```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Örnek veri
data, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# DBSCAN kümeleme
dbscan = DBSCAN(eps=0.3, min_samples=5)
labels = dbscan.fit_predict(data)

# Sonuçları görselleştirme
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')
plt.show()
```

2. Mean Shift

Bu algoritma veri noktalarını yoğunluk zirvelerine doğru kaydırır.

Her veri noktası, çevresindeki veri noktalarının yoğunluğunu değerlendirir ve yoğunluk zirvesine doğru kayar. Küme sayısı önceden belirtilmez.

Kod üzerinde örneği:

```
from sklearn.cluster import MeanShift
```

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Örnek veri
data, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Mean Shift kümeleme
mean_shift = MeanShift()
labels = mean_shift.fit_predict(data)

# Sonuçları görselleştirme
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')
plt.show()
```