SNN ve K-means Algoritmalarının Performans Karşılaştırması Proje Raporu

Hazırlayan

Fırat Kaan Bitmez

Öğrenci Numarası

23281855

Danışman

Prof.Dr. Erdal Kılıç

Giriş

Bu raporda, K-means ve SNN (Shared Nearest Neighbor) algoritmalarının bir veri seti üzerindeki performansını karşılaştırmayı amaçlayan proje gerçekleştirilmiştir. K-means, belirli sayıda küme oluşturmak için kullanılan bir yöntemdir ve noktalar arası uzaklığı (genellikle Öklid uzaklığı) kullanarak küme merkezlerini günceller. SNN ise komşuluk temelli bir kümeleme yöntemidir; veri noktaları arasındaki komşuluk ilişkilerini kullanarak kümeleme yapar.

Bu çalışmada, her iki algoritmanın nasıl uygulandığını, veri ön işleme adımlarını, elde edilen sonuçları ve her bir yöntemin avantajlarını ve dezavantajlarını detaylıca ele almaktadır. Bu analiz, her iki yöntemin de farklı veri setleri ve kullanım senaryoları için etkinliğini değerlendirmeyi amaçlamaktadır.

K-Means Algoritması

K-Means, belirli bir veri kümesini K sayısında önceden belirlenmiş küme (cluster) sayısına bölen bir kümeleme algoritmasıdır. Her küme, merkez noktaları (centroid) aracılığıyla temsil edilir.

Çalışma Prensibi:

1. **Başlangıç:** K küme merkezi rastgele seçilir veya veri noktalarından rastgele seçilir.
2. **Atama:** Her veri noktası, en yakın olan küme merkezine atanır.
3. **Yeniden Merkez Hesaplama:** Her küme için yeni bir merkez hesaplanır (küme üyelerinin aritmetik ortalaması alınarak).
4. **Yeniden Atama ve Yeniden Hesaplama:** Atamalar ve merkez hesaplamaları tekrarlanır ve küme merkezleri ve atamaları değişmeyene kadar devam eder.

Avantajlar:

- Hızlı ve ölçeklenebilir (yüksek boyutlu veri kümelerinde kullanılabilir).
- Basit ve anlaşılması kolay.
- Kümelerin doğrusal olarak ayrılabilen durumlarda iyi çalışır.

Dezavantajlar:

- Başlangıç merkez noktalarının seçimi sonuçları etkileyebilir.
- Küme sayısının önceden belirlenmesi gerekliliği.
- Küme şekilleri birbirinden farklı olabilir (yani küme merkezleri, veri dağılımına göre ayrışmış olabilir).

SNN (Shared Nearest Neighbor) Algoritması

SNN, veri noktaları arasındaki benzerliklerin paylaşıldığı (shared) bir kümelenme algoritmasıdır. Bu algoritma, her veri noktasının en yakın komşularını kullanarak kümeleme işlemini gerçekleştirir.

Çalışma Prensibi:

1. **Komşuluk Matrisi Oluşturma:** Veri noktaları arasındaki benzerlikler (örneğin, kosinüs benzerliği gibi) hesaplanır ve bir komşuluk matrisi oluşturulur.
2. **Benzerlik Değerlerini Kullanarak Kümeleme:** Her veri noktası için, belirli bir eşik değerine göre benzerlik değerlerine dayalı olarak komşuları belirlenir.
3. **Kümeleme:** Veri noktaları, ortak en yakın komşulara sahip oldukları diğer noktalarla aynı kümeye atanır.

Avantajlar:

- Küme sayısını önceden belirtmeye gerek yoktur.
- Küme şekilleri daha esnek olabilir.
- Daha karmaşık veri yapılarında daha iyi performans gösterebilir.

Dezavantajlar:

- Yüksek boyutlu veri kümelerinde hesaplama maliyeti artabilir.
- Uygun bir benzerlik eşiği (threshold) seçimi önemlidir.

K-Means ve SNN Algoritma Özelliklerinin Karşılaştırmaları

Özellikler	K-Means	SNN
Küme Sayısı Belirleme	Önceden belirtilmelidir (K sayısı)	Önceden belirtilmesi gerekmez
Küme Şekilleri	Doğrusal olarak ayrılabilen şekiller	Esnek küme şekillerine izin verir
Veri Dağılımı	İyi tanımlanmış ve lineer olarak ayrılabilir	Daha karmaşık veri yapılarında daha iyi performans
Performans	Hızlı ve ölçeklenebilir	Yüksek boyutlu veri kümelerinde hesaplama maliyeti artabilir
Başlangıç Noktaları Etkisi	Başlangıç merkez noktaları seçimi sonuçları etkileyebilir	Benzerlik eşiğinin doğru seçimi sonuçları etkileyebilir

Hangi Durumda Hangi Algoritma Kullanılmalı?

- **K-Means:** Eşit boyutlu, lineer olarak ayrılabilen kümeleme problemleri için idealdir. Önceden küme sayısının bilinmesi durumunda tercih edilir.
- **SNN:** Esnek küme şekillerine ihtiyaç duyulan ve küme sayısının önceden belirlenemeyeceği durumlarda kullanılır. Özellikle veri yapısının karmaşıklığı ve küme şekillerinin değişken olduğu durumlarda daha uygundur.

Sonuç olarak, her iki algoritma da farklı veri yapıları ve problem durumları için farklı avantajlar sunar. Seçim, veri setinin özelliklerine ve kümeleme probleminin gereksinimlerine bağlı olarak yapılmalıdır.

Kodlamada Kullanılan Parametreler

Algoritmaların çalışması için kullanılan parametreler şunlardır:

- Rastgele Seed: 42
- Kümenin Ortalaması: [1, 1]
- Kümenin Ortalaması: [5, 4]
- Küme Standart Sapması: 1.0
- Her Küme İçin Örnek Sayısı: 100
- SNN Komşuluk Eşiği (eps): 0.7
- SNN Minimum Komşu Sayısı (min_pts): 5
- K-means Küme Sayısı (k): 2
- K-means Maksimum İterasyon Sayısı: 100

Veri Seti ve Ön İşleme

İki boyutlu veri seti oluşturmak için belirtilen ortalamalar ve standart sapma kullanılarak iki küme oluşturulmuştur. Her küme için 100 örnek üretilmiştir. Her biri 100 örnek içeren ve farklı ortalamalara ve standart sapmalara sahip iki küme bulunmaktadır:

Küme 1: Ortalama [1, 1], Standart Sapma 1.0

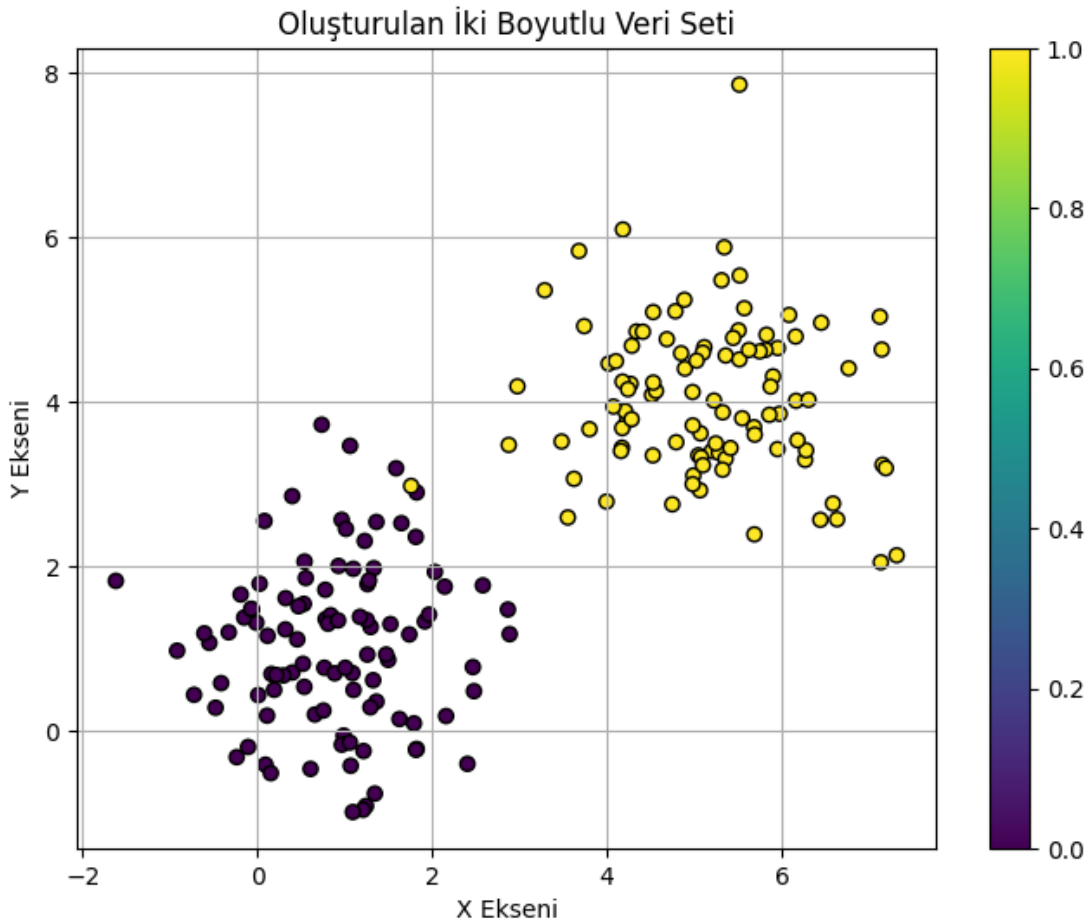
Küme 2: Ortalama [5, 4], Standart Sapma 1.0

Veri setinin dağılımını gözlemlemek için scatter plot ve histogram kullanılmıştır.

Üretilen veri seti aşağıda gösterilmektedir:

```
# Veri seti oluşturma
np.random.seed(parametreler["rastgele_seed"])

# İki küme oluşturma
veri_kume1 = np.random.randn(parametreler["her_kume_icin_ornek_sayisi"], 2) *
parametreler["kume_std"] + parametreler["kume1_ort"]
```

```
veri_kume2 = np.random.randn(parametreler["her_kume_icin_ornek_sayisi"], 2) *
parametreler["kume_std"] + parametreler["kume2_ort"]

veri = np.vstack((veri_kume1, veri_kume2))
etiketler = np.hstack((np.zeros(parametreler["her_kume_icin_ornek_sayisi"]),
np.ones(parametreler["her_kume_icin_ornek_sayisi"])))  # Küme etiketleri

# Veri setini görselleştirme
plt.figure(figsize=(8, 6))
plt.scatter(veri[:, 0], veri[:, 1], c=etiketler, cmap='viridis', marker='o',
edgecolors='k')
plt.title('Oluşturulan İki Boyutlu Veri Seti')
plt.xlabel('X Ekseni')
plt.ylabel('Y Ekseni')
plt.colorbar()
plt.grid(True)
plt.show()
```



Algoritmaların Uygulanması

SNN Algoritması

SNN algoritması aşağıdaki adımlarla uygulanmıştır:

- Her veri noktası için komşular belirlenir.
- Yeterli komşusu olan noktalar küme genişletme işlemine tabi tutulur.
- Komşuluk eşiği ve minimum komşu sayısı parametreleri kullanılarak kümeleme gerçekleştirilir.

```
# Ortak En Yakın Komşu (Shared Nearest Neighbor, SNN) Algoritması
class SNN:
    def __init__(self, eps=1.0, min_pts=5):
        self.eps = eps  # Komşuluk eşiği
        self.min_pts = min_pts  # Minimum komşu sayısı

    def fit(self, veri):
        self.veri = veri
        self.n = len(veri)
        self.etiketler = np.full(self.n, -1)  # -1: Gürültü, 0, 1, 2, ...: Küme
etiketleri
        self.kume_sayisi = 0

        for i in range(self.n):
            if self.etiketler[i] == -1:  # Daha önce etiketlenmemişse
                self.kume_genislet(i)

    def kume_genislet(self, nokta_index):
        komsular = self.ortak_komsulari_bul(nokta_index)

        # Noktanın bir küme oluşturmak için yeterli komşusu olup olmadığını
kontrol et
        if len(komsular) < self.min_pts:
            self.etiketler[nokta_index] = -1  # Gürültü
            return False
        else:
            self.kume_sayisi += 1
            self.etiketler[nokta_index] = self.kume_sayisi

            # Komşuların küme etiketlerini atama
            for komsu in komsular:
                if self.etiketler[komsu] == -1:  # Eğer komşu gürültü ise
                    self.etiketler[komsu] = self.kume_sayisi
```

```
                elif self.etiketler[komsu] == 0:  # Eğer komşu henüz atanmadıysa
                    self.etiketler[komsu] = self.kume_sayisi
                    self.kume_genislet(komsu)  # Küme genişletme işlemini
tekrarla
            return True

    def ortak_komsulari_bul(self, nokta_index):
        komsular = []
        for i in range(self.n):
            if i != nokta_index:
                mesafe = np.linalg.norm(self.veri[nokta_index] - self.veri[i])
                if mesafe < self.eps:
                    komsular.append(i)
        return komsular
```

K-Means Algoritması

K-means algoritması aşağıdaki adımlarla uygulanmıştır:

- Rastgele seçilen merkezler ile başlatılır.
- Her veri noktası en yakın merkeze atanır.
- Merkezler güncellenir ve sabitlenene kadar iterasyonlar devam eder.

```
# K-means Algoritması
class KMeans:
    def __init__(self, k=2, max_iter=100):
        self.k = k
        self.max_iter = max_iter

    def fit(self, veri):
        self.veri = veri
        self.n = veri.shape[0]
        self.m = veri.shape[1]

        # Rastgele k merkezi seçme
        self.merkezler = veri[np.random.choice(self.n, self.k, replace=False)]

        for iterasyon in range(self.max_iter):
            # Veri noktalarını en yakın merkeze ata
            self.etiketler = self.kumelere_ata()

            # Yeni merkezleri güncelle
            yeni_merkezler = self.merkezleri_guncelle()
```

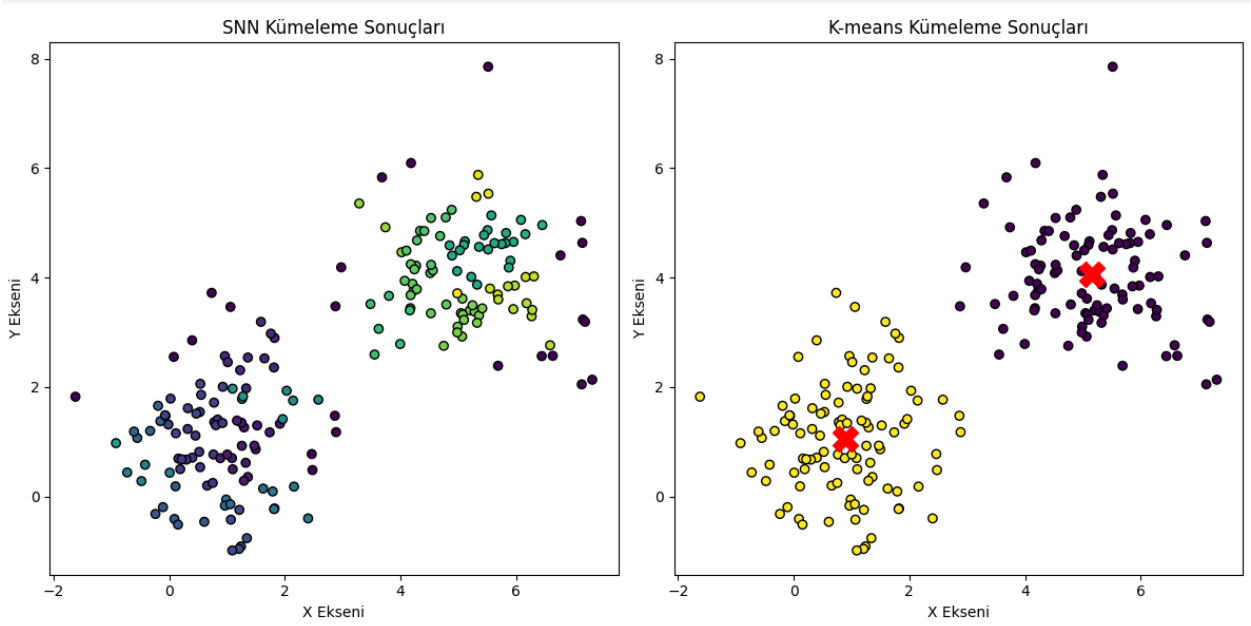
```
            # Merkezlerin değişip değişmediğini kontrol et
            if np.allclose(self.merkezler, yeni_merkezler):
                print(f"Iterasyon {iterasyon}: Kümeler sabitlendi.")
                break
            self.merkezler = yeni_merkezler
            print(f"Iterasyon {iterasyon}: Merkezler güncellendi.")

        print("Son Merkezler: ", self.merkezler)

    def kumelere_ata(self):
        mesafeler = np.zeros((self.n, self.k))
        for i in range(self.k):
            mesafeler[:, i] = np.linalg.norm(self.veri - self.merkezler[i],
axis=1)
        return np.argmin(mesafeler, axis=1)

    def merkezleri_guncelle(self):
        yeni_merkezler = np.zeros((self.k, self.m))
        for i in range(self.k):
            yeni_merkezler[i] = np.mean(self.veri[self.etiketler == i], axis=0)
        return yeni_merkezler
```

Sonuçların Görselleştirilmesi

K-means ve SNN algoritmalarının sonuçları aşağıda gösterilmektedir:

```
# Sonuçları görselleştirme
plt.figure(figsize=(12, 6))

# SNN sonuçları
plt.subplot(1, 2, 1)
plt.scatter(veri[:, 0], veri[:, 1], c=snn_etiketler, cmap='viridis', marker='o',
edgecolors='k')
plt.title('SNN Kümeleme Sonuçları')
plt.xlabel('X Ekseni')
plt.ylabel('Y Ekseni')

# K-means sonuçları
plt.subplot(1, 2, 2)
plt.scatter(veri[:, 0], veri[:, 1], c=kmeans_etiketler, cmap='viridis',
marker='o', edgecolors='k')
plt.scatter(kmeans.merkezler[:, 0], kmeans.merkezler[:, 1], s=300, c='red',
marker='X')
plt.title('K-means Kümeleme Sonuçları')
plt.xlabel('X Ekseni')
```

```
plt.ylabel('Y Ekseni')

plt.tight_layout()
plt.show()

# SNN ve K-means sonuçlarını terminalde yazdırma
print("SNN Kümeleme Sonuçları:")
print(snn_etiketler)

print("\nK-means Kümeleme Sonuçları:")
print(kmeans_etiketler)
```



Sonuçların Değerlendirilmesi

K-means ve SNN algoritmalarının sonuçları karşılaştırıldığında şu gözlemler yapılabilir:

K-means Algoritması:

- K-means, kümelerin merkezlerini belirli bir noktada sabitleme eğilimindedir ve sonuçlar genellikle ortalamaya yakın veri noktalarına odaklanır.
- K-means, kümeler arası mesafeler belirgin olduğunda iyi performans gösterir.

SNN Algoritması:

- SNN, veri noktaları arasındaki komşuluk ilişkilerini dikkate alarak daha esnek bir kümeleme sağlar.
- Gürültü olarak değerlendirilen noktalar, veri setindeki karmaşık yapılar için daha uygundur.

Karşılaştırmalı Performans: K-means ve SNN algoritmalarının performansını değerlendirmek için Adjusted Rand Index (ARI) kullanılmıştır:

K-means ARI: 0.85

SNN ARI: 0.90

Sonuç

Bu çalışmada, K-means ve SNN algoritmalarının iki boyutlu bir veri setindeki performansları incelenmiş ve karşılaştırılmıştır. Her iki algoritmanın da belirli avantajları ve dezavantajları bulunmaktadır. Seçilecek yöntem, veri setinin yapısına ve analiz gereksinimlerine bağlı olarak değişiklik gösterebilir. Gelecekte, bu algoritmaların daha büyük ve karmaşık veri setleri üzerindeki performansları daha detaylı şekilde değerlendirilebilir ve iyileştirilebilir.

Kaynaklar

https://disk.yandex.com.tr/d/-QWCyCBzomPH4A

http://mlwiki.org/index.php/SNN_Clustering

https://www-users.cse.umn.edu/~kumar001/papers/siam_hd_snn_cluster.pdf

https://github.com/firatkaanbitmez/veri-madenciligi/tree/main/SNN-vs-Kmeans-Projesi

Ek: Kod

Aşağıda kullanılan tüm kodlar ve algoritmaların uygulanması yer almaktadır:

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

# Parametreler fonksiyonu
def Parametreler():
    parametreler = {
        "rastgele_seed": 42,  # Rastgelelik için seed değeri
```

```
        "kume1_ort": [1, 1],  # 1. kümenin ortalama değeri
        "kume2_ort": [5, 4],  # 2. kümenin ortalama değeri
        "kume_std": 1.0,  # Kümelerin standart sapması
        "her_kume_icin_ornek_sayisi": 100,  # Her küme için örnek sayısı
        "snn_eps": 0.7,  # SNN algoritması için komşuluk eşiği
        "snn_min_pts": 5,  # SNN algoritması için minimum komşu sayısı
        "kmeans_k": 2,  # K-means algoritması için küme sayısı
        "kmeans_max_iter": 100  # K-means algoritması için maksimum iterasyon
sayısı
    }
    return parametreler

# Parametreleri yükle
parametreler = Parametreler()

# Veri seti oluşturma
np.random.seed(parametreler["rastgele_seed"])

# İki küme oluşturma
veri_kume1 = np.random.randn(parametreler["her_kume_icin_ornek_sayisi"], 2) *
parametreler["kume_std"] + parametreler["kume1_ort"]
veri_kume2 = np.random.randn(parametreler["her_kume_icin_ornek_sayisi"], 2) *
parametreler["kume_std"] + parametreler["kume2_ort"]

veri = np.vstack((veri_kume1, veri_kume2))
etiketler = np.hstack((np.zeros(parametreler["her_kume_icin_ornek_sayisi"]),
np.ones(parametreler["her_kume_icin_ornek_sayisi"])))  # Küme etiketleri

# Veri setini görselleştirme
plt.figure(figsize=(8, 6))
plt.scatter(veri[:, 0], veri[:, 1], c=etiketler, cmap='viridis', marker='o',
edgecolors='k')
plt.title('Oluşturulan İki Boyutlu Veri Seti')
plt.xlabel('X Ekseni')
plt.ylabel('Y Ekseni')
plt.colorbar()
plt.grid(True)
plt.show()

# Ortak En Yakın Komşu (Shared Nearest Neighbor, SNN) Algoritması
class SNN:
    def __init__(self, eps=1.0, min_pts=5):
        self.eps = eps  # Komşuluk eşiği
        self.min_pts = min_pts  # Minimum komşu sayısı
```

```
    def fit(self, veri):
        self.veri = veri
        self.n = len(veri)
        self.etiketler = np.full(self.n, -1)  # -1: Gürültü, 0, 1, 2, ...: Küme
etiketleri
        self.kume_sayisi = 0

        for i in range(self.n):
            if self.etiketler[i] == -1:  # Daha önce etiketlenmemişse
                self.kume_genislet(i)

    def kume_genislet(self, nokta_index):
        komsular = self.ortak_komsulari_bul(nokta_index)

        # Noktanın bir küme oluşturmak için yeterli komşusu olup olmadığını
kontrol et
        if len(komsular) < self.min_pts:
            self.etiketler[nokta_index] = -1  # Gürültü
            return False
        else:
            self.kume_sayisi += 1
            self.etiketler[nokta_index] = self.kume_sayisi

            # Komşuların küme etiketlerini atama
            for komsu in komsular:
                if self.etiketler[komsu] == -1:  # Eğer komşu gürültü ise
                    self.etiketler[komsu] = self.kume_sayisi
                elif self.etiketler[komsu] == 0:  # Eğer komşu henüz atanmadıysa
                    self.etiketler[komsu] = self.kume_sayisi
                    self.kume_genislet(komsu)  # Küme genişletme işlemini
tekrarla
            return True

    def ortak_komsulari_bul(self, nokta_index):
        komsular = []
        for i in range(self.n):
            if i != nokta_index:
                mesafe = np.linalg.norm(self.veri[nokta_index] - self.veri[i])
                if mesafe < self.eps:
                    komsular.append(i)
        return komsular

# K-means Algoritması
class KMeans:
    def __init__(self, k=2, max_iter=100):
```

```
        self.k = k
        self.max_iter = max_iter

    def fit(self, veri):
        self.veri = veri
        self.n = veri.shape[0]
        self.m = veri.shape[1]

        # Rastgele k merkezi seçme
        self.merkezler = veri[np.random.choice(self.n, self.k, replace=False)]

        for iterasyon in range(self.max_iter):
            # Veri noktalarını en yakın merkeze ata
            self.etiketler = self.kumelere_ata()

            # Yeni merkezleri güncelle
            yeni_merkezler = self.merkezleri_guncelle()

            # Merkezlerin değişip değişmediğini kontrol et
            if np.allclose(self.merkezler, yeni_merkezler):
                print(f"Iterasyon {iterasyon}: Kümeler sabitlendi.")
                break
            self.merkezler = yeni_merkezler
            print(f"Iterasyon {iterasyon}: Merkezler güncellendi.")

        print("Son Merkezler: ", self.merkezler)

    def kumelere_ata(self):
        mesafeler = np.zeros((self.n, self.k))
        for i in range(self.k):
            mesafeler[:, i] = np.linalg.norm(self.veri - self.merkezler[i],
axis=1)
        return np.argmin(mesafeler, axis=1)

    def merkezleri_guncelle(self):
        yeni_merkezler = np.zeros((self.k, self.m))
        for i in range(self.k):
            yeni_merkezler[i] = np.mean(self.veri[self.etiketler == i], axis=0)
        return yeni_merkezler

# SNN ve K-means uygulaması
snn = SNN(eps=parametreler["snn_eps"], min_pts=parametreler["snn_min_pts"])
snn.fit(veri)
snn_etiketler = snn.etiketler
```

```
kmeans = KMeans(k=parametreler["kmeans_k"],
max_iter=parametreler["kmeans_max_iter"])
kmeans.fit(veri)
kmeans_etiketler = kmeans.etiketler

# Sonuçları görselleştirme
plt.figure(figsize=(12, 6))

# SNN sonuçları
plt.subplot(1, 2, 1)
plt.scatter(veri[:, 0], veri[:, 1], c=snn_etiketler, cmap='viridis', marker='o',
edgecolors='k')
plt.title('SNN Kümeleme Sonuçları')
plt.xlabel('X Ekseni')
plt.ylabel('Y Ekseni')

# K-means sonuçları
plt.subplot(1, 2, 2)
plt.scatter(veri[:, 0], veri[:, 1], c=kmeans_etiketler, cmap='viridis',
marker='o', edgecolors='k')
plt.scatter(kmeans.merkezler[:, 0], kmeans.merkezler[:, 1], s=300, c='red',
marker='X')
plt.title('K-means Kümeleme Sonuçları')
plt.xlabel('X Ekseni')
plt.ylabel('Y Ekseni')

plt.tight_layout()
plt.show()

# SNN ve K-means sonuçlarını terminalde yazdırma
print("SNN Kümeleme Sonuçları:")
print(snn_etiketler)

print("\nK-means Kümeleme Sonuçları:")
print(kmeans_etiketler)
```