**SNN ve K-means Algoritmaları Başarım Karşılaştırması Proje Raporu**

**Hazırlayan**

Fırat Kaan Bitmez

**Öğrenci Numarası**

23281855

**Dersin Hocası**

Prof.Dr. Erdal Kılıç

**Giriş**

Bu raporda, K-means ve Ortak En Yakın Komşu (Shared Nearest Neighbor, SNN) algoritmalarının iki boyutlu bir veri seti üzerinde uygulamaları ve sonuçları incelenecektir. Amacımız, bu iki algoritmanın performansını kıyaslayarak veri kümeleme yöntemlerinin etkinliğini değerlendirmektir.

**Kullanılan Algoritmalar ve Parametreler**

**K-Means Algoritması**

K-means algoritması, veri noktalarını belirli bir sayıda kümeye ayıran yaygın bir kümeleme yöntemidir. Temel adımları şunlardır:

Başlangıçta rastgele kkk merkez (merkez noktalar) seçilir.

Her veri noktası, en yakın merkeze atanır.

Yeni merkezler, bu kümelerdeki veri noktalarının ortalamaları olarak güncellenir.

Merkezler sabitlenene veya maksimum iterasyon sayısına ulaşılana kadar adımlar tekrarlanır.

**Ortak En Yakın Komşu (SNN) Algoritması**

SNN algoritması, veri noktalarını komşuluk ilişkilerine dayanarak kümeler. Temel adımları şunlardır:

Her veri noktası için komşuları belirlenir.

Komşuların ortak komşuluk sayısına göre küme genişletilir.

Belirli bir eşik değerin altında kalan noktalar gürültü olarak kabul edilir.

**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**

İki boyutlu veri seti oluşturmak için belirtilen ortalamalar ve standart sapma kullanılarak iki küme oluşturulmuştur. Her küme için 100 örnek üretilmiştir. 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()

metin, diyagram, ekran görüntüsü, öykü gelişim çizgisi; kumpas; grafiğini çıkarma içeren bir resim

Açıklama otomatik olarak oluşturuldu

**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.

**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.

**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)

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**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.

**Sonuç**

Bu çalışmada, K-means ve SNN algoritmaları kullanılarak iki boyutlu bir veri seti üzerinde kümeleme işlemleri gerçekleştirilmiş ve sonuçları karşılaştırılmıştır. Her iki algoritmanın da avantajları ve dezavantajları bulunmaktadır ve seçilecek yöntem, veri setinin yapısına ve analiz amacına bağlı olarak değişkenlik gösterebilir.

**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)