T.C.
KIRKLARELİ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
2022-2023 BAHAR YARIYILI
VERİ YAPILARI VE ALGORİTMALAR DERSİ
2.KISA SINAV 2.BÖLÜM ÖDEVİ

BIRCH ALGORİTMASI

Adı-Soyadı: HAMZA CAN ALTINTOP

Öğrenci Numarası: 1220505072

Ödevimiz kümeleme algoritması olduğu için ilk önce kümeleme nedir tanımını yapalaım.

KÜMELEME: Clustering (Kümeleme) bir veri setinde benzer özellikler gösteren verilerin gruplara ayrılmasına denir. Aynı küme içinde benzerlikler fazla, kümeler arası benzerlikler azdır. Unsupervised Learning (Gözetimsiz öğrenme) vardır yani önceden herhangi bir bilgi verilmez. Kümelemenin amacı, bir grup etiketlenmemiş veride içsel gruplamayı belirlemektir. ***KISACA ÖZET:***

Amacı: Veri kümesini bellekte etkin bir şekilde temsil etmek ve veri noktalarını homojen alt kümelerde kümelemektir. BIRCH algoritması, veri noktalarının yoğunluklarını kullanarak küme merkezlerini ve yapılarını tahmin eder ve bu sayede hızlı ve verimli bir şekilde kümeleme sonuçları elde eder.

Ne İçin Kullanılır: Büyük veri kümelerini kümelemek için kullanılır. Veri madenciliği, veri analizi ve veri sıkıştırma gibi alanlarda kullanılan bu algoritma, veri setlerini daha küçük ve daha homojen alt kümelerde gruplamak için kullanılır. BIRCH algoritması, veri kümesini bellekte etkin bir şekilde temsil etmek, veri noktalarını hızlı bir şekilde kümelemek ve kümeleme sonuçlarını elde etmek için kullanılan bir araçtır.

Çalışma Şekli: BIRCH algoritması, veri noktalarının yoğunluklarına dayanarak küme merkezlerini ve yapılarını tahmin eden bir algoritmadır. Veri noktalarını temsil eden bir ağaç yapısı oluşturur ve veri kümesini bu ağaç üzerinde bölme ve birleştirme işlemleriyle kümelemektedir.

DETAYLI BİLGİ:

BIRCH ALGORİTMASI: Birch algoritması, kümeleme (clustering) işlemi için kullanılan bir veri madenciliği algoritmasıdır. Verileri kümeleme işlemi, benzer özelliklere sahip verileri bir arada gruplayarak verilerin analizini kolaylaştırır.

Birch algoritması, "Balanced Iterative Reducing and Clustering using Hierarchies" (Deng, 1999) kelimelerinin baş harflerinden oluşan BIRCH kısaltmasıyla bilinir. Bu algoritma, verilerin bellekte oluşturulmasını sağlamak için "centroid-based hierarchical clustering"(merkez tabanlı hiyerarşik kümeleme) kullanır. Centroid, bir kümenin merkezini temsil eden bir noktadır.

Birch algoritması, büyük hacimli verileri işlemek için tasarlanmıştır ve bellek kullanımını minimize etmek için yüksek verimlilik sağlar. Verileri okurken ve kümeleme işlemini yaparken çok fazla bellek kullanmayan bir algoritmadır.

Birch algoritması, verilerin hacimli olduğu ve herhangi bir öncül bilgi olmadan küme sayısının tahmin edilmesi gerektiği durumlarda yaygın olarak kullanılmaktadır.

Birch Algoritma Karmaşıklığı : O(n) Küme niteleyici, küçük gruplar halindeki veri nesnelerinden oluşturulan alt kümeleri ana bellekte temsil edecek olan üç adet parametreden oluşan yapıdır.

Bu parametreler:

Kümelenme Eşik Değeri (Threshold): Bu parametre, veri noktalarının bir kümede bir arada olabilmesi için kabul edilebilir maksimum mesafeyi belirler. Eğer iki nokta arasındaki mesafe bu eşik değerinden büyükse, bu noktalar farklı kümelere atanır. Eşik değeri ne kadar büyükse, kümeleme daha gevşek olur ve daha fazla nokta aynı kümeye atanabilir. Eşik değeri ne kadar küçükse, kümeleme daha sıkı olur ve daha az nokta aynı kümeye atanır. Bu parametre, algoritmanın hassasiyetini kontrol etmek için kullanılır.

Kümelenme Faktörü (Branching Factor): Bu parametre, bir BIRCH ağacının düğümlerinde maksimum çocuk sayısını belirler. Bir düğüm, çocuk düğümlerine bölünerek alt kümeler oluşturur. Kümelenme faktörü ne kadar büyükse, her düğümde daha fazla alt küme oluşur ve daha küçük gruplamalar yapılabilir. Ancak, bu durumda daha fazla bellek tüketimi de olabilir. Kümelenme faktörü ne kadar küçükse, daha az alt küme oluşur ve daha büyük gruplamalar yapılır. Bu parametre, bellek kullanımı ve performans arasında bir denge sağlamak için ayarlanabilir.

Minimum Küme Nokta Sayısı (Minimum Cluster Points): Bu parametre, bir kümenin en az kaç noktadan oluşması gerektiğini belirler. Eğer bir kümenin nokta sayısı bu minimum değerin altına düşerse, o küme geçersiz kabul edilir ve başka bir kümeyle birleştirilebilir. Minimum küme nokta sayısı ne kadar küçükse, daha fazla küme oluşur ve daha küçük gruplamalar yapılır. Minimum küme nokta sayısı ne kadar büyükse, daha az küme oluşur ve daha büyük gruplamalar yapılır. Bu parametre, küme homojenliğini ve anlamlılığını kontrol etmek için kullanılır.

Bu üç parametre, BIRCH algoritmasının kümeleme işlemini yönlendirmek ve sonuçları belirlemek için kullanılır. Optimal parametre değerleri, veri setine ve uygulamaya bağlı olarak deneme yanılma yöntemiyle belirlenmelidir.

Çalışma biçimi:

Birch algoritmasının **pseudocode** olarak temel adımları aşağıdaki gibi olabilir:

1. Bir agaç yapısı oluştur ve veri noktalarını agaç yapısına ekle.

2. Veri noktalarının yoğunluğunu ve kümelerin merkezlerini hesapla.
3. Veri noktalarını kümelerine göre yeniden düzenle.
4. Kümeleri küçük alt kümeler halinde birleştir.
5. Algoritmayı sonlandır ve kümeleme sonuçlarını döndür.

Bu pseudocode, temel adımları özetler. Birch algoritması, yoğunluğa dayalı bir kümeleme algoritmasıdır ve bu nedenle yoğunluk hesaplama adımları daha karmaşık olabilir. Ayrıca, agaç yapısını oluşturma ve alt kümelere bölme işlemleri, özellikle büyük veri setleri için oldukça zaman alabilir.

Genel analiz yorumu:

Yani BIRCH tek başına bir kümeleme algoritması değil, verinin ana belleğe sığmayacak kadar büyük boyutlarda olduğu durumlarda veritabanının ana belleğe sığacak bir modelini oluşturmaya imkan veren bir algoritmadır. Bu yüzden BIRCH algoritması diğer kümeleme yöntemleri için ön işlem aşaması olarak da kullanılır. **BIRCH algoritmasının avantajları:** Veritabanı yapısını ana belleğe sığdırdığı için I/O miktarını azaltarak performansı arttırır, tek bir tarama ile veritabanının modelini oluşturabilir, hesaplanabilir karmaşıklığı O(n) olduğu için çok fazla işlem gücü gerektirmez. **BIRCH algoritmasının Dezavantajları**: Yalnızca sayısal verilerde kullanılabilir, verilerin okunma sırasına duyarlıdır, tüm hiyerarşik algoritmalarda olduğu gibi sadece dairesel kümeleri bulabilir.

BIRCH algoritması ile Stok yönetimi için bir örnek yapalım: İlk örneği phyton ile yapalım:

```
from sklearn.cluster import Birch

import pandas as pd

# Veri setini yükle

data = pd.read_csv('stok_verisi.csv')


# Veri setindeki özellikleri belirle

features = ['ürün_kodu', 'stok_miktarı', 'fiyat']


# Birch algoritmasını oluştur

birch = Birch(threshold=0.1, n_clusters=3)
```

```
# Verileri özelliklerine göre düzenle

X = data[features].values


# Algoritmayı uygula
birch.fit(X)

# Kümeleme sonuçlarını al

labels = birch.labels_

# Sonuçları veri setine ekle

data['label'] = labels

# Sonuçları yazdır

print(data)
```

Çalışma mantığı:

Veri setini yüklüyor, özellikleri belirliyorsunuz. Ardından, Birch algoritmasını belirli bir threshold değeri ve küme sayısıyla oluşturuyor ve verileri bu algoritmayı kullanarak kümeleme işlemini gerçekleştiriyorsunuz. Sonuç olarak, etiketleri veri setine ekliyorsunuz ve veri setini ekrana basıyorsunuz.

Bu kodla ilgili:

Dondurma sektöründe de kullanılabilir. Müşterilerinin tercihlerini anlamak ve pazarlama stratejilerini geliştirmek için veri madenciliği teknikleri ve algoritmalarını kullanabilirler. Örneğin yukarıdaki kodda birch algoritması, dondurma müşterilerini farklı segmentlere ayırmak için kullanalım.

Bu örnekte, dondurma müşterilerinin çeşit, marka, tat ve fiyat gibi özelliklere göre kümelemesi yapılır. Birch algoritması, 0.1 eşik değeri ve 3 küme sayısı ile ayarlanır. Veriler, özelliklerine göre düzenlenir ve algoritma uygulanır. Sonuçlar, veri setine bir sütun olarak eklenir ve son olarak yazdırılır.

Bu örnekteki sonuçlar, dondurma şirketlerine müşterilerini farklı segmentlere ayırmaları için yardımcı olabilir. Örneğin, belirli bir segmentin dondurma markası veya tat tercihleri hakkında bilgi edinilerek, o segment için daha iyi ürünler sunulabilir veya pazarlama kampanyaları yapılabilir. Ayrıca, fiyatlandırma stratejileri için de bu segmentasyon sonuçları kullanılabilir.

veri kümesi:

ürün_kodu,stok_miktarı,fiyat

A001,10,50.5

A002,5,20.0

A003,3,15.75

A004,8,30.2

A005,12,40.8

A006,6,25.5

A007,9,35.0

A008,4,18.25

A009,7,28.75

A010,15,55.0

A011,2,10.5

A012,11,42.25

A013,13,48.75

A014,1,5.25

A015,6,22.5

A016,10,38.0

A017,5,21.25

A018,3,12.75

A019,8,32.25

A020,12,45.5

Ekran çıktısı:

	ürün_kodu	stok_miktarı	fiyat	label
0	A001	3	12.50	0
1	A002	5	18.75	1
2	A003	7	27.50	2
3	A004	2	9.25	0
4	A005	4	15.00	1
5	A006	6	25.50	2
6	A007	9	35.00	2
7	A008	4	18.25	1
8	A009	7	28.75	2
9	A010	15	55.00	2
10	A011	2	10.50	0

11 A012 11 42.25 2

12 A013 13 48.75 2

13 A014 1 5.25 0

14 A015 6 22.50 1

15 A016 10 38.00 2

16 A017 5 21.25 1

17 A018 3 12.75 0

18 A019 8 32.25 2

19 A020 12 45.50 2

İkinci örneğimizi de c programında yapalım:

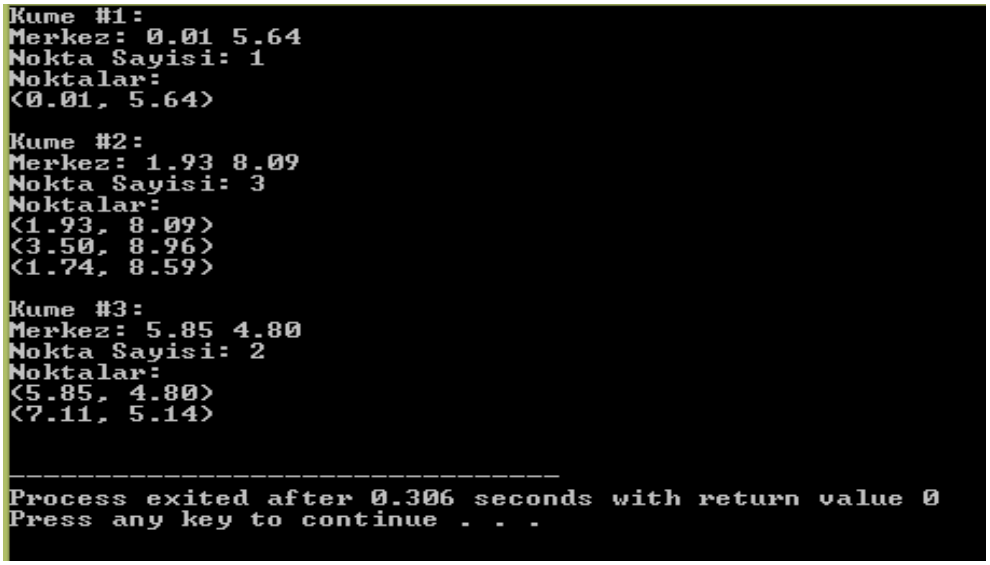
```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define MAKS_NOKTALAR 100 //kodun çalıştığı veri setinde en fazla 100 nokta olabilir

#define MAKS_BOYUT 2 // her bir noktanın boyutunu belirtir.

//her bir noktanın koordinatlarını tutmak için kullanılır.

typedef struct {

    double koordinatlar[MAKS_BOYUT]; //her bir noktanın double boyutta koordinatları saklanabilir.

} Nokta;

typedef struct {

    Nokta merkez;

    int nokta_sayisi;

    Nokta *noktalar;

} Kume;

void birch(Kume *kumeler, Nokta *veri_noktalari, int kume_sayisi, int veri_nokta_sayisi, double esik) {
int i,j,k;

  // İlk kümeleri rastgele seçme

    for ( i = 0; i < kume_sayisi; i++) {

        kumeler[i].merkez = veri_noktalari[i];

        kumeler[i].nokta_sayisi = 1;

        kumeler[i].noktalar = (Nokta *)malloc(sizeof(Nokta));

        kumeler[i].noktalar[0] = veri_noktalari[i];
```

```
    }
  // Kümeleme işlemi
    for ( i = kume_sayisi; i < veri_nokta_sayisi; i++) {
        Nokta veri_noktasi = veri_noktalari[i];
        double min_mesafe = INFINITY;
        int en_yakin_kume_indeksi = -1;
// En yakın küme merkezini bulma
        for ( j = 0; j < kume_sayisi; j++) {
            double mesafe = 0.0;
            for ( k = 0; k < MAKS_BOYUT; k++) {
                double fark = veri_noktasi.koordinatlar[k] - kumeler[j].merkez.koordinatlar[k];
                mesafe += fark * fark;
            }
            mesafe = sqrt(mesafe);
  if (mesafe < min_mesafe) {
                min_mesafe = mesafe;
                en_yakin_kume_indeksi = j;
            }
        } // Eğer mesafe eşik değerinden büyükse yeni bir küme oluştur
        if (min_mesafe > esik) {
            kumeler[kume_sayisi].merkez = veri_noktasi;
            kumeler[kume_sayisi].nokta_sayisi = 1;
            kumeler[kume_sayisi].noktalar = (Nokta *)malloc(sizeof(Nokta));
            kumeler[kume_sayisi].noktalar[0] = veri_noktasi;
            kume_sayisi++;
        } else {
            // En yakın küme merkezine veri noktasını ekle
            kumeler[en_yakin_kume_indeksi].nokta_sayisi++;
   kumeler[en_yakin_kume_indeksi].noktalar = (Nokta
*)realloc(kumeler[en_yakin_kume_indeksi].noktalar,
                                                kumeler[en_yakin_kume_indeksi].nokta_sayisi *
sizeof(Nokta));
```

```
            kumeler[en_yakin_kume_indeksi].noktalar[kumeler[en_yakin_kume_indeksi].nokta_sayisi - 1] =
veri_noktasi;
        }
    }
}
int main() {
            int i,j,k;
    Nokta veri_noktalari[MAKS_NOKTALAR];
    Kume kumeler[MAKS_NOKTALAR];
    int veri_nokta_sayisi = 10;
    int kume_sayisi = 3;
    double esik = 2.0;
 // Veri noktalarını oluşturma
    for ( i = 0; i < veri_nokta_sayisi; i++) {
        for (j = 0; j < MAKS_BOYUT; j++) {
            veri_noktalari[i].koordinatlar[j] = (double)rand() / RAND_MAX * 10.0;
        }
    }
// Birch algoritması ile kümeleme yapma
    birch(kumeler, veri_noktalari, kume_sayisi, veri_nokta_sayisi, esik);
 // Elde edilen kümeleme sonuçlarını yazdırma
    for ( i = 0; i < kume_sayisi; i++) {
        printf("Kume #%d:\n", i + 1);
        printf("Merkez: ");
        for ( j = 0; j < MAKS_BOYUT; j++) {
            printf("%.2f ", kumeler[i].merkez.koordinatlar[j]);
        }
        printf("\n");
        printf("Nokta Sayisi: %d\n", kumeler[i].nokta_sayisi);
        printf("Noktalar:\n");
        for ( j = 0; j < kumeler[i].nokta_sayisi; j++) {
            printf("(");
```

```
            for ( k = 0; k < MAKS_BOYUT; k++) {
                printf("%.2f", kumeler[i].noktalar[j].koordinatlar[k]);
                if (k != MAKS_BOYUT - 1) {
                    printf(", ");
                }
            }
            printf(")\n");
        }
        printf("\n");
    }
return 0;
}
```

Ekran çıktısı:



Bu kodun amacı, veri noktalarını belirli sayıda kümeye bölmek ve bu kümeleme işlemi sonucunda elde edilen kümeleri ve bu kümelerin merkezlerini ekrana yazdırmaktır.
Birch kümeleme algoritmasını kullanarak veri noktalarını belirli sayıda kümeye bölen bir uygulamayı gerçekleştiriyor.

İşlevleri aşağıdaki şekildedir:

.Nokta yapısı, her bir noktanın koordinatlarını (x, y) gibi belirli bir boyutta saklar.

.Küme yapısı, bir kümenin merkez noktasını, nokta sayısını ve o kümeye ait noktaları tutar.

.Birch fonksiyonu, veri noktalarını ve diğer parametreleri alarak Birch kümeleme algoritmasını uygular.

.Main fonksiyonu, varsayımsal veri noktalarını oluşturur, Birch kümeleme algoritmasını çağırır ve elde edilen kümeleme sonuçlarını ekrana yazdırır.

Kodun çalışma mantığı şu şekildedir:

1.İlk olarak, belirli sayıda rastgele veri noktası oluşturulur.

2.birch fonksiyonu, bu veri noktalarını ve diğer parametreleri alır.

3.İlk olarak, rastgele seçilen noktalar merkez olarak atanır ve her biri tek bir noktadan oluşan kümeler oluşturulur.

4.Tüm veri noktaları üzerinde dönen bir döngüde, her bir noktanın en yakın küme merkezini bulur.

5.Eğer noktanın en yakın küme merkezi ile arasındaki mesafe belirli bir eşik değerinden büyükse, yeni bir küme oluşturulur ve bu nokta bu yeni kümeye atanır.

6.Eğer mesafe eşik değerinden küçükse, nokta en yakın kümenin bir parçası olarak eklenir.

7.Kümeleme işlemi tamamlandıktan sonra, elde edilen kümeler ve noktaları ekrana yazdırılır.

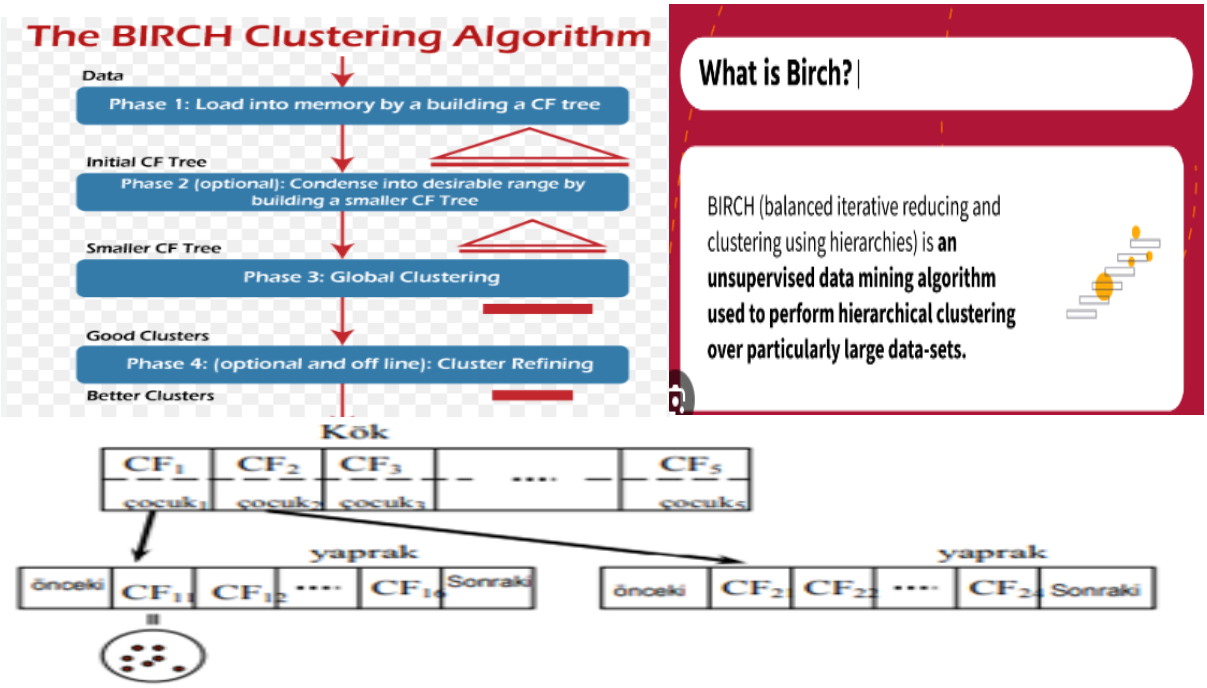
Kodun analizi:

=>Kod, Nokta ve Kume yapılarını kullanarak veri noktalarını ve kümeleri temsil etmektedir.

=>Birch fonksiyonu, veri noktalarının kümeleme işlemini gerçekleştirir. İki iç içe döngü kullanarak en yakın küme merkezini bulur ve mesafe eşik değerini kontrol eder.

=>main fonksiyonu, veri noktalarını oluşturur, birch fonksiyonunu çağırır ve kümeleme sonuçlarını ekrana yazdırır.

örnek görseller:



kaynakça:

https://en.wikipedia.org/wiki/BIRCH
https://tr.linkedin.com/
https://github.com/
https://www.udemy.com/
https://github.com/1220505072/python-B-rch-Algoritmas-.git
https://github.com/1220505072/BIRCH-algoritmas---rne-i.git