EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

ÇOK BOYUTLU VERİTABANLARINDA KÜMELEME YÖNTEMLERİ ÜZERİNE

Elvin NASİBOV

Tez Danışmanı: Doç. Dr. Burak Ordin

Matematik Anabilim Dalı

Bilim Dalı Kodu: 619.03.03
Sunuş Tarihi: 26.12.2012

Bornova-İZMİR

2012

Elvin Nasibov tarafından Yüksek Lisans tezi olarak sunulan "Çok Boyutlu Veritabanlarında Kümeleme Yöntemleri Üzerine" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 26.12.2012 tarihinde yapılan tez savunma sınavında aday oybirliği/~~oyçokluğu~~ ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	: Prof. Dr. Ugur NURİYEV	.....
Raportör Üye	: Doç. Dr. Burak ORDİN	.....
Üye	: Yrd. Doç. Dr. Mehmet KURT	.....

ÖZET

ÇOK BOYUTLU VERİTABANLARINDA KÜMELEME YÖNTEMLERİ ÜZERİNE

NASİBOV, Elvin

Yüksek Lisans Tezi, Matematik Anabilim Dalı
Tez Danışmanı: Doç. Dr. Burak ORDİN
Aralık 2012, 60 sayfa

Veri madenciliği yöntemlerinden biri olan Kümeleme Analizi, verilerin özelliklerini gözönüne alarak, birbirleri ile benzer olan verileri alt kümelere ayırmayı sağlayan çok boyutlu veri analiz yöntemidir. Kümeleme analizinin bir diğer tanımı da şu şekilde verilebilir: Özellikler arası benzerlik ya da farklılıklara dayalı olarak hesaplanan bazı ölçülerden yararlanarak verileri homojen gruplara bölmek, belirli prototipler tanımlamaktır.

Kümeleme yöntemleri hiyerarşik ve hiyerarşik olmayan yöntemler olarak iki sınıfa ayrılır. Hiyerarşik kümelemede veri noktaları belirli bölümleme düzeylerinde birleştirilir veya ayrıştırılır. Hiyerarşik olmayan kümeleme yaklaşımında ise, veri noktaları belirli bölümleme kriterlerine göre belirli sayıda kümelere ayrılır.

Bu tezde, hiyerarşik olmayan kümeleme yaklaşımına dayanan, veri setindeki veri grupları arasında kesin ayrımın söz konusu olduğu, kesin kümeleme algoritmalarından olan artımlı algoritmalar incelenmiştir. Kesin kümeleme içerisinde yer alan artımlı algoritmalar, veri setindeki veri grupları arasında kesin ayrımın söz konusu olmadığı, bir verinin belirli bir üyelik derecesiyle birden fazla kümeye ait olabildiği bulanık kümeleme yöntemleri ile beraber ele alınıp, bulanık kümeleme problemleri için yeni bir algoritma önerilmiştir. Önerilen artımlı yöntem C# dilinde MS SQL Server Veri Tabanı Yönetim Sistemi imkanları kullanılarak programlanıp, 12 gerçek veri seti üzerinde hesaplama denemeleri yapılmıştır. Önerilen algoritma Bulanık c-Ortalamalar algoritması ile kıyaslandığında yöntemin yararlılığı açıkça görülmektedir.

Anahtar kelimeler: Bulanık c-Ortalamalar, Global k-Ortalamalar, Bulanık Kümeleme, Pürüzlü Optimizasyon.

ABSTRACT

METHODS OF CLUSTERING ON MULTIDIMENSIONAL DATABASES

Nasibov, Elvin

Supervisor: Assoc. Prof. Burak ORDİN
December 2012, 60 pages

The Clustering Analysis, is one of the main techniques of data mining and it is also the method of analysis of multidimensional databases which divides the data set into clusters based on the similarity of data points. Another definition of cluster analysis can be given as follows: Divide data into homogeneous groups by using calculated measures on the basis of similarities or differences among the properties, to identify the specific prototypes.

Clustering methods are divided into two classes: Hierarchical and non-hierarchical methods. On Hierarchical clustering the data points are combined and separated with the specific levels of partitioning. On non-hierarchical approaching the data points are divided into given number of clusters, according to the given criteria.

In this thesis, the exact (hard) incremental clustering algorithms were investigated. This algoritms are based on the approach of non-hierarchical clustering. The hard incremental clustering algorithms with fuzzy clustering algorithms where there is not absolute division between the groups, and each data data may belong to more than one cluster with the fuzzy membership degree, are examined and a new algorithm is proposed for fuzzy clustering problems. The proposed incremental method programmed in C # using MS SQL Server and calculation experiments have been made on 12 real data set. The proposed method compared with the Fuzzy c-Means algorithm and usefulness of method is clearly seen.

Key words: Fuzzy c-Means, Global k-Means, Fuzzy Clustering, Nonsmooth Optimization.

TEŞEKKÜR

Yüksek lisans çalışmam boyunca, bu tezi hazırlamam için benden bilgisini ve anlayışını hiçbir zaman esirgemeyen değerli hocam Doç. Dr. Burak ORDİN'e sonsuz teşekkür ederim. Ayrıca, desteğini ve ilgisini her zaman yanımda duyduğum değerli hocam Prof. Dr. Urfat NURİYEV'e teşekkür etmeği kendime bir borc bilirim.

Hayatım boyunca bana her türlü destek olan, beni her zaman anlayışla karşılayan aileme ve tüm sevdiklerime çok teşekkür ederim.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

.

ŞEKİLLER DİZİNİ

Şekil Sayfa

ÇİZELGELER DİZİNİ

Çizelge Sayfa

1. GİRİŞ

Kümeleme (Clustering) analizi kümeler arasındaki benzerliklere dayanarak ilişkiler topluluğunun organizasyonu problemi ile ilgilidir. Kümelemede amaç, bir veri kümesini, benzer veriler aynı kümede bulunacak şekilde bölümlere ayırmaktır. Bir başka ifadeyle, veri kümelerini doğal grup yada gruplara ayırmaktır. Gözetimsiz (unsupervised) veri sınıflandırması olarak da bilinen kümeleme analizi tıp, ekonomi, bankacılık, mühendislik vb. bir çok alanda önemli uygulamalara sahiptir.

Kümeleme problemlerinin farklı türleri mevcuttur. Bu türlerden birisi olan kesin (Hard) kümeleme yönteminde her bir veri noktası tek bir kümeye aittir. Bulanık (Fuzzy-Soft) kümelemede ise veri elemanları belirli bir üyelik derecesi ile birden fazla kümeye ait olabilir. Üyelik derecesi kavramını Bulanık kümeler teorisi içerisinde yer alıp, 1965 yılında Lütfi Zadeh (A. Zadeh., 1965) tarafından önerilmiştir.

Kümeleme problemi aslında bir global optimizasyon problemidir ve böyle bir problem için çok sayıda çözümler bulunmaktadır. Bu çözümlerden global olanları en iyi çözümlerdir ve veri kümeleri için en iyi kümeleme yapısına karşı gelmektedirler. Kümeleme problemini çözmek için çeşitli optimizasyon modelleri ve bu modeller yardımıyla çözüm yöntemleri önerilmiştir. Bu teknikler dinamik programlama, dal ve sınır, düzlem kesme, iç nokta arama metodları, değişken komşuluk arama algoritması, tabu arama, genetik algoritmalar gibi kesin ve yaklaşık çözüm yöntemleridir. Ancak yukarıda belirtilen yöntemler orta büyüklükteki veri setlerinde bile kümeleme problemlerinin çözümü için etkili değillerdir. Bu sorunu çözmek için daha etkin çözüm yaklaşımlarına ihtiyaç vardır.

Kesin (Hard) ve bulanık (fuzzy) kümeleme problemlerinin çözümü için çeşitli algoritmalar önerilmiş ve bir çok alanda farklı veri kümeleri için uygulanmıştır:

D.N. Georgiou ve arkadaşları 2009 yılında, bulanık kümeleme tekniği ile 20 amino asitin sınıflandırılmasına ait bir çalışma yapmışlardır. Bu çalışmada veriler arasındaki uzaklığı hesaplamak için iki farklı uzaklık fonksiyonu kullanılmıştır:

Minkowski uzaklık fonksiyonu ve NTV uzaklık fonksiyonu. Amino asitlerin çeşitli fiziksel özellikleri veri özellikleri olarak ele alınıp, kümeleme yöntemleri ile bir analiz yapılmıştır (Georgiou et al., 2008).

2005 yılında Keh-Shih Chuang ve arkadaşları kümeleme problemi için geleneksel bulanık c-means algoritmasında kullanılan üyelik fonksiyonunu mekansal fonksiyon değeri yardımıyla tanımlayıp, yeni bir Bulanık c-Ortalamalar (Fuzzy c-Means - FCM) algoritması sundular. Buradaki mekansal fonksiyonun değeri ele alınan her piksel için onun komşuluğundaki üyelik fonksiyon değerlerinin toplamından oluşmaktadır. Yeni yaklaşımın avantajları şu şekilde verilebilir: (1) bu yöntem diğer yöntemlere göre bölgeleri daha homojen ayırır (2) sahte lekeleri azaltıyor (3) gürültülü noktalar ortadan kaldırılıyor ve (4) diğer yöntemlere göre gürültüye daha az duyarlılık gösteriyor (Chuang et al., 2006).

Michael E. Brandt ve arkadaşları 1994 yılında beyin omurilik sıvısı (BOS), beyaz cevher ve beyin transaksial manyetik rezonans görüntülerinden (MRG) gelen gri cevher hacimleri ölçmek için bir algoritma önermişlerdir. Burada kullanılan algoritma doku tanımlama için kullanılan bulanık c-ortalamalar kümeleme yönteminin bir türü olup, bu algoritmanın beyin omurilik sıvısının beklenen artış miktarını ve hidrosefalik beynin beyaz cevher karakteristiğinin beklenen azalış miktarını tespit edebileceği ifade edilmiştir (Brandt et al., 1994).

2005 yılında Antonino Staiano ve arkadaşları RBFNN (Radyal Taban Fonksiyonlu Sinir Ağları) nin gizli birimlerinin merkezlerinin aranması için denetimli bulanık kümeleme yöntemi kullanma fikrini ileri sürmüşlerdir. Bu yöntem RBFNN eğitiminde diğer kümeleme algoritmalarına göre daha iyi performans sağlamaktadır (Staianoa et al., 2005).

Fabio Dell'Acqua ve arkadaşları 2001 yılında kentsel ortamlarda havadan sentetik açıklıklı radar (SAR) görüntülerinin analizi için bulanık bir kümeleme yaklaşımı üzerinde çalışmışlardır. Bu çalışmada denetimsiz bulanık kümeleme yaklaşımını kentsel SAR görüntüleri üzerinde uygulayıp, sokak izleme problemleri üzerinde analizler yapmışlardır (Dell'Acqua and Gamba, 2001).

2001 yılında Jar-Ferr Yang ve arkadaşları piksel segmentasyonu için iki adet öz-tabanlı bulanık c-ortalamalar (FCM) kümeleme algoritması önermişlerdir.

Bu yaklaşıma göre özuzay dönüşümü ve FCM yöntemi kombine edilerek renkli görüntü segmentasyonunun daha etkin olması sağlanmıştır. Ayrık Özuzay FCM (Separate Eigenspace FCM) algoritmasında temel ve rezidüel projeksiyonlara iki ara parçalı görüntü elde etmek ve mantıksal ortak piksellerini seçerek onları birleştirmek için bağımsız şekilde FCM yöntemi uygulanmıştır. Simülasyonlar önerilen algoritmaların önemli bir hassasiyetle ve istenilen renk görüntü başarısıyla segmentasyon yapabilme yeteneğine sahip olduğunu göstermiştir (Yang et al., 2001).

Uzay Kaymak ve Magne Setnes 2000 yılında amaç fonksiyonuna dayalı bulanık kümeleme algoritması üzerinde iki yeni fikir önermişlerdir. İlki, prototiplerin (noktaların) boyutlarının kümelenecek verilerden otomatik etkilenen hiper hacimlere genişletilmesi ile ilgili, ikincisi, optimizasyon sırasında kümeler arasındaki benzerliği değerlendirerek kümelerin birleştirilmesi ile ilgili çalışmayı tanıtmışlardır. Çalışmada veri kümesindeki tahmini küme sayısı ile başlayarak, benzer veri kümeleri daha uygun bölünme elde edilmesi için kümelenme sırasında birleştirilmeye başlamıştır. Makalede bu birleştirme için bir adaptif eşik tanıtılmıştır. Yukarıda ifade edilen fikirler Gustafson-Kessel ve bulanık c-ortalamalar algoritmalarına uyarlanmış ve elde edilen genişletilmiş algoritmalar sunulmuştur (Kaymak and Setnes, 2000).

2010 yılında T. Velmurugan ve T. Santhanam yoğunluk tabanlı k-means ve temsili nesneye dayalı bulanık c-ortalamalar algoritmalarını ve bunların performansı üzerine çalışmalar yapmışlardır. Her iki algoritmanın davranışlarının veri noktalarının sayısının yanı sıra küme sayısına da bağlı olduğunu göstermişler. Burada girdi veri noktaları normal dağılım ve tekdüze dağılım (Box-Muller formülü) olmak üzere iki yolla oluşturulmuştur. Algoritmanın performansı programın veri girişi noktalarında farklı şekillerde çalıştırılmasıyla incelenmiştir. Her bir algoritma için çalışma süreleri ve çıkan sonuçlar da ayrıca araştırılmıştır (Velmurugan and Santhanam, 2010).

Kümeleme algoritmaları için başlangıç merkez noktalarının seçimi çok önemlidir. Seçilen nokaların yerel minimumlara götürmesi bir problem olup, global çözüme ulaşmak için farklı çözüm yaklaşımlarının incelenmesi önemlidir.

Kesin kümeleme içerisinde son 10-15 yıldır k-ortalamalar algoritmasındaki başlangıç çözüm noktalarının seçimindeki zorlukları aşmayı hedefleyen artımlı (inkremental) algoritmalar ileri sürülmüştür. Artımlı kümeleme algoritmaları her aşamada en iyi şekilde yeni bir küme merkezinin probleme eklenmesi mantığıyla çalışırlar. Global olmayan yöntemlerdeki gibi ilk baştan k sayıda küme merkeziyle hesaplamaya başlamak yerine bu türden algoritmalar (k - 1)-kümeleme problemi için (k – 1) sayıda merkez noktasıyla bir sonraki (k.) merkez noktasının en iyi şekilde hesaplanmasına dayanır.

Aristidis Likas ve arkadaşları 2001 yılında Global k-Ortalamalar (GKM) isimli bir algoritma önermişler. Bu algoritma artımlı bir algoritma olup, k-ortalamalar algoritması yerel en iyi çözümlere takılabiliyorken bu algoritma global minimum yada global minimuma yakın çözümler üretmektedir (Likas A. et al., 2001).

Adil M. Bagirov ve Karim Mardaneh 2006 yılında modifiye edilmiş bir Global k-Ortalamalar algoritması (Modified Global k-Means Algorithm, MGKM) önermişlerdir. Bu artımlı algoritmada küme merkezlerinin bulunması için yardımcı bir optimizasyon problemin çözümü gerekmektedir. Sonuçlar göstermiştir ki MGKM algoritması Global k-Ortalamalar algoritmasından daha iyi sonuçlar vermektedir. Çalışmada UCI Veri Ambarı üzerinden alınan veri kümeleri için k-Ortalamalar, Global k-Ortalamalar ve MGKM algoritmaları kıyaslanmıştır.

Sonraki çalışmada, Global k-Ortalamalar algoritmasının bir başka versiyonu önerilip, önerilen yöntemin avantaj ve dezavantajları 14 gerçek veri kümesi üzerinde incelenmiştir (Bagirov, 2008).

Genellikle, gerçek uygulamalarda verilerin kümeleri arasında sınırlar çok keskin şekilde olmuyor, bundan dolayı bulanık kümeleme yöntemleri bu tarzda veriler için daha uygun olmaktadır. Diğer yandan kesin kümeleme yöntemlerinden artımlı olanları optimal ve ya optimal değere yakın sonuç değerleri veriyorlar. Bu tezde bulanık yaklaşımın ve artımlı hesaplamanın avantajlarını kullanarak yeni bir artımlı bulanık kümeleme yöntemi önerilmiştir. Önerilen algoritmada amaç fonksiyonunu minimize etmek için bulanık değerlere sahip modifiye edilmiş

Global c-Ortalamalar algoritması kullanılmıştır. Yöntem literatürdeki 13 gerçek veri kümesi üzerinde uygulanıp Bulanık c-Ortalamalar algoritması ile kıyaslanıp, elde edilen sonuçlar ifade edilmiştir.

Bu tez, girişi izleyen 7 bölümden oluşmaktadır.

İkinci bölümde; kümeleme problemi tanıtılıp, problemin matematiksel modelleri incelenmiştir. Bulanık kümeler ve bulanık bölünme hakkında bilgiler verilmiştir.

Üçüncü bölümde; kümeleme problemleri için kesin ve bulanık çözüm yöntemleri incelenip, literatürde yer alan bazı önemli algoıritmalar incelenmiştir.

Dördüncü bölümde; modifiye edilmiş bir bulanık global c-means algoritması ifade edilmiştir.

Beşinci bölümde; UCI Veri Ambarından alınan gerçek veri kümeleri üzerinde önerilen yeni algoritma Bulanık c-Ortalamalar algoritması ile kıyaslanmış ve yapılan hesaplama denemelerinin sonuçları verilmiştir.

Altıncı bölümde; kümeleme yapılması için oluşturulan yazılımın özellikleri ve parametreleri hakkında bilgiler verilmiştir.

Yedinci bölüm sonuç bölümüdür.

2. KÜMELEME PROBLEMİNİN MATEMATİKSEL MODELLENMESİ

2.1. Kesin Kümeleme Problemi

Bu bölümde Kümeleme probleminin matematiksel modelleri ifade edilmiştir.

n-boyutlu m noktadan oluşan R^n uzayında sonlu sayıda elemana sahip A kümesini ele alalım:

$$A = \{a^1, \dots, a^m\}, \text{ ve } a^i \in \mathbb{R}^n, \ \ i = 1, \dots, m.$$

Buna göre kısıtsız ve tam bölümlemeli kümeleme probleminin amacı, A kümesindeki noktaları, verilen k adet ayrık A^j, $j = 1, \dots, k$ altkümeye, önceden tanımlanmış aşağıdaki kurallara göre ayırmaktır.

1) $A^j \neq \emptyset, \ \ j = 1, \dots, k;$
2) $A^j \cap A^l = \emptyset, \ \ j, l = 1, \dots, k, \ \ j \neq l;$
3) $A = \bigcup_{j=1}^{k} A^j;$
4) $A^j, \ \ j = 1, \dots, k$ kümeleri için bir kısıt bulunmamaktadır.

A^j, $j = 1, \dots, k$ altkümelerine, küme (cluster) adı verilir. Burada oluşacak her bir kümenin boş küme olmaması, hiç bir küme çiftinin ortak bir elemana sahip olmaması, kümelerin birleşiminin veri kümesine eşit olması ve hiç bir küme için bir kısıt bulunmaması kuralları esas alınmaktadır.

Aynı kümeden olan noktalar birbirlerine benzer ve farklı kümeden olan noktalar ise birbirlerinden farklıdırlar. Veri noktalarının benzerlikleri benzerlik ölçüsü (similarity measure) olarak adlandırılan bir ölçümle tanımlanır. Bu ölçüm incelenen noktanın ait olduğu kümenin merkezine uzaklığı ile tanımlanır.

$d(x, y)$: x ve y noktaları arasındaki uzaklık olsun. Bu durumda kümeleme problemi aşağıdaki optimizasyon problemine dönüşür:

$$\text{Min:}\quad \psi_k(x,w) = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} w_{ij}\, d\left(x^j, a^i\right) \tag{2.1}$$

$$\text{Burada,}\ \ x = \left(x^1, \dots, x^k\right) \in \mathbb{R}^{n\times k}, \tag{2.2}$$

$$\sum_{j=1}^{k} w_{ij} = 1\,, i = 1, \dots, m, \tag{2.3}$$

$$w_{ij} = 0 \text{ veya } 1,\ \ i = 1, \dots, m,\ \ j = 1, \dots, k \tag{2.4}$$

Burada w_{ij} : a^i örneğinin j kümesine aşağıdaki koşullara göre ait olma ağırlığıdır.

$$w_{ij} = \begin{cases} 1, \text{ eğer } a^i \text{ elemanı } j \text{ kümesine atandıysa,} \\ 0, \text{ aksi halde} \end{cases}$$

$$\text{ve} \qquad x^j = \frac{\sum_{i=1}^{m} w_{ij} a^i}{\sum_{i=1}^{m} w_{ij}}\,,\ j = 1, \dots, k. \tag{2.5}$$

burada w, mxk boyutlu bir matristir.

$d(x,y)$ uzaklığı farklı normlarla tanımlanabilir. Bu çalışmada aşağıdaki şekilde tanımlanan Öklid normu kullanılmıştır:

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^2\right)^{1/2} \tag{2.6}$$

(2.1)-(2.4) problemi kümeleme probleminin karma tamsayılı doğrusal olmayan programlama modelidir. (2.1)-(2.4) deki kümeleme probleminin Pürüzlü Dışbükey olmayan (Nonsmooth Nonconvex) formülasyonu ise aşağıdaki şekilde verilebilir: (Bagirov vd., 2002; Bagirov vd.,2003; Bock, 1998))

$$\text{Min: } f_k(x^1, \dots, x^k) = \frac{1}{m}\sum_{i=1}^{m} min_{j=1,\dots,k}\, d\left(x^j, a^i\right). \tag{2.7}$$

$$\left(x^1, \dots, x^k\right) \in \mathbb{R}^{n\times k}, \tag{2.8}$$

Hem ψ_k hem de f_k fonksiyonları küme (cluster) fonksiyonları olarak adlandırılır. Yukarıda ifade edilen model ile (2.1)-(2.4) denklemlerinde ifade edilen model kıyaslandığında:

1. ψ_k amaç fonksiyonu, w_{ij} , $i = 1, \dots, m$, $j = 1, \dots, k$ (0 veya 1 olan katsayılar) ve $x^1, x^2, \dots, x^k$, $x^j \in \mathbb{R}^n$, $j = 1, \dots, k$ (sürekli değişkenler olan küme merkezleri) değişkenlerine bağlıdır. Diğer yandan f_k fonksiyonu yalnızca $x^1, x^2, \dots, x^k$ değişkenlerine bağlıdır. Buna dayanarak her iki fonksiyonun dışbükey olmadığını söyleyebiliriz.

2. (2.1)-(2.4) problemindeki değişkenlerin sayısı $(m + n) * k$ iken (2.7)-(2.8) probleminde bu sayı sadece $n * k$ dır ve dolayısıyla değişken sayısı veri sayısına bağlı değildir. Dikkat edilmelidir ki, gerçek hayattaki veri kümelerinde verilerin sayısı olan m, özelliklerin sayısı olan n değerinden çok daha büyüktür.

3. (2.1)-(2.4) ve (2.7)-(2.8) problemleri global enküçükleyicilerinin aynı olması anlamında birbirlerine denktirler. (2.1)-(2.4) modeli üzerinde kullanılan k-ortalamalara karşın (2.7)-(2.8) modeli üzerinde kullanılan global k-ortalamalar algoritması global optimum yada global optimuma yakın sonuçlar vermektedir.

2.2. Bulanık Kümeler ve Bulanık Bölümleme

Bulanık kümeler kendi başlangıcını Azerbaycan kökenli Amerika bilim adamı olan L.Zadeh tarafından 1965 yılında yayınlanmış bir çalışmadan almaktadır (A. Zadeh., 1965). Her hangi bir küme değildiğinde, genellikle, belirli bir özelliği sağlayan elemanlar kümesi anlaşılmaktadır. Belli ki, hayatta elemanlar bu özelliği çoğu zaman tam olarak değil, belirli bir düzeyde sağlaya biliyorlar. Bu halde eleman kümeye tam olarak değil, özelliği sağladığı kadar ait olacaktır. Klasik matematikte bu gibi durumlar modellenemiyor, yani elemanlar bir kümeye

ya ait olur, ya da ait olmazlar. Fakat bulanık mantık ve bulanık kümeler teorisi bu gibi durumları modellemek için geniş imkanlar sağlamaktadır.

Matematiksel olarak bulanık $A \subset X$ kümesi (bazen altküme de denilir) aşağıdaki gibi tanımlanır:

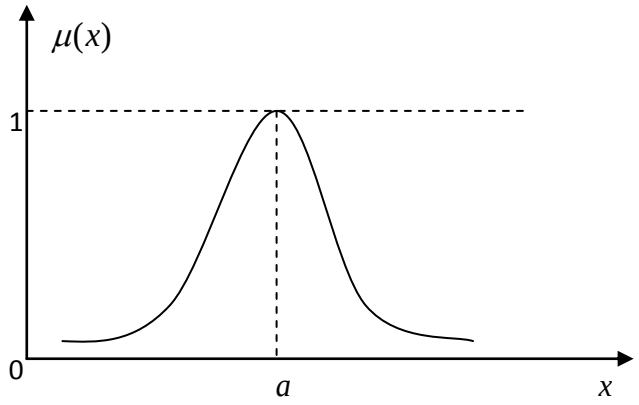
$$A = \{(x, \mu_A(x)) \mid x \in X\}. \qquad (2.9)$$

Burada X - evrensel küme, $\mu_A(x)$ ise $x \in X$ elemanının bulanık A kümesine üyelik derecesidir. Her bir bulanık A kümesine sadece bir $\mu_A : X \to [0,1]$ üyelik fonksiyonu uygun olur. Bununla, her hangi bir kümenin tanımlanması o kümenin üyelik fonksiyonunun da tanımlanması anlamına geliyor. Örneğin, eger bulanık A kümesi {a,b,c,d,e} elemanlarından oluşmuşsa ve bu elemanların kümeye üyelik dereceleri uygun olarak {0.2,0.7,1.0,0.6,0.9} ise, bu zanam bulanık küme aşağıdaki yazılışların biriyle tanımlana bilinir:

$$A = \{(a \mid 0.2), (b \mid 0.7), (c \mid 1), (d \mid 0.6), (e \mid 0.9)\}$$

veya

$$A = 0.2 \mid a + 0.7 \mid b + 1 \mid c + 0.6 \mid d + 0.9 \mid e.$$



Şekil 2.1. Üyelik fonksiyonu (2.10) formülü olan bulanık küme.

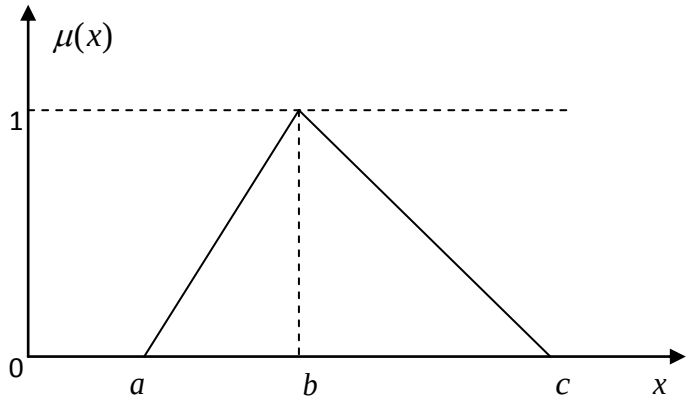
Bulanık kümelerin üyelik fonksiyonlarının formül şeklinde tanımlanmasına örnek olarak aşağıdakiler verilebilir (Şekil 2.1 ve 2.2):

$$\mu_A(x) = e^{-(x-a)^2}, \tag{2.10}$$

veya

$$\mu_A(x) = \begin{cases} \dfrac{x-a}{b-a}, & x \in [a,b] \ \ oldukta, \\ \dfrac{c-x}{c-b}, & x \in [b,c] \ \ oldukta, \\ 0 \ , & x \notin [a,c] \ \ oldukta. \end{cases} \tag{2.11}$$

Göründüğü gibi, üyelik fonksiyonu verilmiş kümede herhangi bir elemanın üyelik derecesini, o elemanı üyelik fonksiyonu içerisinde ilgili yerine tanımlamakla hesaplamak mümkündür.



Şekil 2.2. Üyelik fonksiyonu (2.11) formülü olan bulanık küme.

Bulanık kümelerle ilgili bazı kavramlar aşağıdaki şekildedir:

Her hangi $\alpha \in [0,1]$ için bulanık A kümesinin α-seviye kümesi:

$$A_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha\} \tag{2.12}$$

Hatırlatalım ki, α-seviye kümesi artık bulanık değil, klasik (kesin) bir kümedir.

A kümesinin güçlü α-seviye kümesi:

$$A_\alpha^s = \{x \in X \mid \mu_A(x) > \alpha\} \qquad (2.13)$$

Çözünülme (resolution) prensipine göre her bir bulanık küme kendi α-seviye kümeleri aracılığıyla sadece ve sadece bir şekilde aşağıdaki gibi verilebilir:

$$A = \bigcup_{\alpha \in [0,1]} (\alpha, A_\alpha). \qquad (2.14)$$

Bulanık A kümesinin geçiş noktaları dendiğinde geçit $\mu_A(x) = 0.5$ koşulunu sağlayan noktalar anlaşılmaktadır.

Bulanık A kümesine en yakın klasik A^* kümesi aşağıdaki gibi tanımlanır:

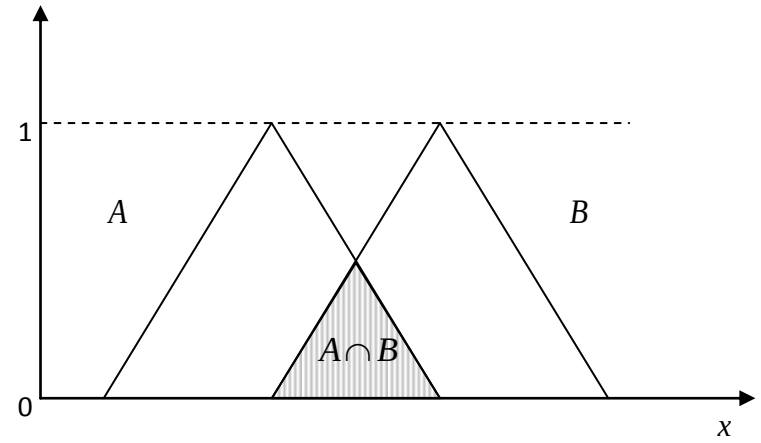
$$\mu_{A^*}(x) = \begin{cases} 0, & \mu_A(x) < 0.5 \ \textit{oldukta,} \\ 1, & \mu_A(x) > 0.5 \ \textit{oldukta,} \\ 0 \ \textit{veya} \ 1, & \textit{diger durumlarda} \end{cases}$$

Klasik küme işlemlerinin bulanık kümeler haline aşağıdaki genişlemeleri mevcuttur:

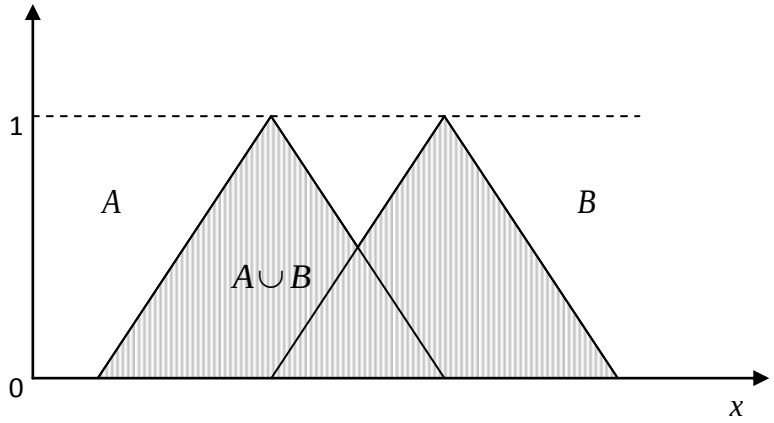
$C = A \cap B \Rightarrow \mu_C(x) = \min(\mu_A(x), \mu_B(x)), \ \forall x \in X$;

$C = A \cup B \Rightarrow \mu_C(x) = \max(\mu_A(x), \mu_B(x)), \ \forall x \in X$;

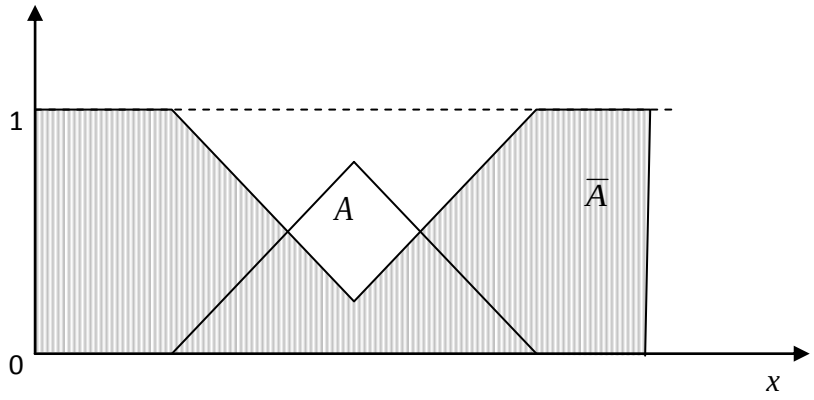
$C = \overline{A} \Rightarrow \mu_C(x) = 1 - \mu_A(x), \ \forall x \in X$.



Şekil 2.3. Bulanık kümelerin kesişmesi.



Şekil 2.4. Bulanık kümelerin birleşmesi.



Şekil 2.5. Bulanık kümenin değil'i.

Bulanık boş küme ve evrensel küme kavramları uygun olarak aşağıdaki gibi tanımlanır:

$\mu_{\emptyset}(x) = 0, \ \forall x \in X$,

$\mu_{X}(x) = 1, \ \forall x \in X$

Bulanık A ve B kümeleri arasında $A \subseteq B$ aitlik ilişkisi aşağıdaki gibi tanımlanır:

$A \subseteq B \Rightarrow \mu_A(x) \leq \mu_B(x), \ \forall x \in X$

Bulanık kümeler için De Morqan kuralları sağlanır, yani herhangi bulanık A ve B kümeleri için:

$\overline{A \cap B} = \overline{A} \cup \overline{B}$,

$\overline{A \cup B} = \overline{A} \cap \overline{B}$.

Fakat bulanık kümelerde klasik kümelerden farklı olarak

$A \cup \overline{A} \neq X$,

$A \cap \overline{A} \neq \emptyset$,

olması mümkündür.

Bulanık Bölümleme

n-boyutlu m noktadan oluşan R^n uzayında sonlu sayıda elemana sahip A kümesini ele alalım:

$$A = \{a^1, \dots, a^m\}, \text{ ve } a^i \in \mathbb{R}^n, \ \ i = 1, \dots, m.$$

A kümesinin bulanık bölünmesi dendiğinde, onun noktalarını aşağıdaki koşulları sağlayan k adet $A^j,\ j = 1, \dots, k$ altkümeye ayrılması anlaşılır:

1) $max_i\, u_{ij} > 0,\ \ j = 1, \dots, k;$

2) $\sum_{j=1}^{k} u_{ij} = 1\,,\ i = 1, \dots, m;$

burada $u_{ij};\ a^i$ elemanının A^j kümesine aitlik derecesini gösterir. İkinci kısıta normallik kısıtı denir. Göründüğü gibi, bulanık bölümlemede $A^j,\ j = 1, \dots, k$ kümelerinin ayrık olması yerine normallik kısıtını sağlaması gerekmektedir.

3. KÜMELEME PROBLEMİ İÇİN KESİN VE BULANIK ÇÖZÜM ALGORİTMALARI

Bölümlemeli kümeleme yöntemleri genellikle sık kullanılan ve geniş uygulama alanına sahip yöntemlerdir. n nesneden oluşan bir veri kümesi verildiğinde, bölümlemeli kümeleme algoritması verinin k-bölünmesini oluşturur. Burada her bir kümenin, belirli bir kümeleme kriterini sağlaması istenmektedir. Küme içerisindeki noktaların, orta noktadan uzaklıklarının kareler toplamının minimizasyonu, kümeleme kriterine örnek olarak verilebilir.

Kümeleme literatürde birçok araştırmacı tarafından incelenmiştir (Höppner et al., 1999). Bu bölümde kesin kümeleme yöntemleri içerisinde yer alan yöntemlerden k-ortalamalar (James MacQueen, 1967), global k-ortalamalar (Aristidis Likas ve arkadaşları, 2001) ve modifiye edilmiş global k-ortalamalar (Bagirov vd.,2003; Bock, 1998) yöntemleri ile bulanık kümeleme içerisinde yer alan bulanık c-ortalamalar algoritması (Dunn, 1973; Bezdek, 1981) incelenmiştir.

3.1. k-Ortalamalar algoritması

Kesin (hard) kümeleme için en çok kullanılan algoritmalardan biri k-Ortalamalar algoritmasıdır. Bu algoritma aşağıdaki şekilde ifade edilebilir:

Algoritma 1 (k-Ortalamalar algoritması):

Adım 1: k adet merkez içeren (merkezlerin A kümesine ait olması gerekmez) bir başlangıç çözümü seçilir.

Adım 2: A kümesindeki verilerin herbiri, seçilen k adet merkez gözönüne alınarak en yakın merkezli kümeye atanır.

Adım 3: Adım 2 deki kümeleme gözönüne alınarak, her bir kümenin yeni merkezleri hesaplanır ve hiçbir veri, kümesini değiştirmeyinceye kadar adım 2 ye dönülür.

Bu algoritma, bahsedilen başlangıç noktalarının seçimine oldukça duyarlıdır ve büyük veri kümeleri için, bu başlangıç noktalarına bağlı olarak, global çözümden oldukça farklı yerel çözümler bulabilmektedir.

3.2. Global k-Ortalamalar algoritması

2001 yılında Likas ve arkadaşları tarafından sunulan Global k-Ortalamalar algoritması ise artımlı bir kümeleme algoritmasıdır ve $k \leq m$ için aşağıdaki şekilde ifade edilir:

Algoritma 2 (Global k-Ortalamalar Algoritması):

Adım 1 (Başlangıç): A kümesinin x^1 merkezi aşağıdaki şekilde hesaplanır:

$$x^1 = \frac{1}{m} \sum_{i=1}^{m} a^i, \quad a^i \in A, \quad i = 1, \dots, m$$

ve $q = 1$ alınır.

Adım 2 (Durma Kriteri): $q = q + 1$. Eğer $q > k$ ise durulur.

Adım 3: $x^1, x^2, \dots, x^{q-1}$ merkezleri önceki iterasyondan alınır ve A kümesindeki herbir a elemanı q. küme merkezi için bir başlangıç noktası olarak ele alınır. Buna göre $(x^1, \dots, x^{q-1}, a)$ şeklinde q boyutlu, m adet başlangıç çözümleri oluşturularak bunların her biri için k-ortalamalar algoritması uygulanır ve elde edilen en iyi kümeleme ile bu kümelemeye karşılık gelen merkezler saklanır $(y^1, y^2, \dots, y^q)$.

Adım 4: $x^i = y^i, \quad i = 1, \dots, q$ ataması yapılır ve adım 2 ye gidilir.

Global k-Ortalamalar algoritması, global en iyiyi yada globale yakın çözümleri bulabilmesi sebebiyle, k-Ortalamalar algoritmasından, pozitif anlamda farklılaşmaktadır. Gerçek hayat uygulamalarında, verideki doğal gruplamayı

ortaya çıkarabilecek olan en uygun k değerinin bilinmesi, genellikle mümkün olamasa da, çalışılan alanda uzmanlaşmış kişilerin tecrübeleri bu değerin belirlenmesinde kullanılmaktadır. Fakat Global k- Ortalamalar yönteminin doğası gereği, k değerinin baştan belirlenmesi gerekli değildir; çünkü artımlı bir algoritma söz konusudur. Bu algoritmada amaç, kümeleme problemlerine, her aşamada en iyi olan, yeni bir küme merkezini eklemektir. Veri kümesinin k-bölünmesini hesaplamak için, $(k-1)$-kümeleme problemindeki, $k-1$ merkezli başlangıç durumundan yola çıkılarak, k. merkez için en uygun noktayı seçmek amaç edinilir. (Bagirov ve Yearwood, 2006; Hansen vd., 2002).

3.3. Modifiye Edilmiş Global k-Ortalamalar Algoritması

Yukarıda adı geçen her iki yöntem de küçük ve orta boyutlu veri kümeleri için uygulanabilmektedir (Vanisri and Loganathan, 2010). Global k-Ortalamalar algoritması için, başlangıç noktalarının seçimi, global çözüme yakınsayabilme anlamında çok önemlidir. Böyle bir yaklaşım en azından yakın bir global enküçükleyiciye ulaşır. Bu yöntem k-ortalamalara göre eniyiye ulaşmada daha iyi olmakla birlikte, hesaplama zamanı açısından daha fazla hesaplama zamanı gerektimektedir (k-Ortalamalar algoritması m defa uygulandığından). Bu algoritmanın bir başka versiyonu daha da iyi çözümler bulabilmek için, Bagirov tarafından geliştirilmiştir. Önerilen yöntemde, çözüm için iyi başlangıç noktalarının seçilmesi fikri esas alınmaktadır (Bagirov, 2008).

Algoritma 3 (Modifiye Edilmiş Global k-Ortalamalar Algoritması):

Adım 1: (Başlangıç) $\varepsilon > 0$ olacak şekilde bir hata toleransı seçilir. A kümesinin merkezi $x^1 \epsilon \mathbb{R}^n$ hesaplanır. f^1 e, (2.7)-(2.8) deki amaç fonksiyonunun karşılık gelen değeri atanır ve $k = 1$ atanır.

Adım 2: (Bir sonraki küme merkezinin hesaplanması). $k = k + 1$ ataması yapılır. $(k-1)$-bölünmeli problem için başlangıç küme merkezleri $x^1, \dots, x^{k-1}$

olmak üzere. k. kümenin merkezi için bir başlangıç noktası ($\bar{y}\epsilon\mathbb{R}^n$) bulmak adına $Algoritma$ 4 uygulanır.

Adım 3: (Tüm küme merkezlerinin düzeltilmesi) $(x^1, \dots, x^{k-1}, \bar{y})$ yeni bir başlangıç noktası olarak seçilir, k-kümeleme problemi için k-Ortalamalar algoritması uygulanır. $y^1, \dots, y^k$ bu problemin bir çözümü ve bu çözüme karşılık gelen (7)-(8) deki amaç fonsiyonunun değeri f^k ya atanır.

Adım 4: (Durma kriteri) $\frac{f^{k-1}-f^k}{f^1} < \varepsilon$ ise durulur, değilse $x^i = y^i$, $i = 1, \dots, k$ ataması yapılır ve Adım 2 ye dönülür.

Açıktır ki burada her $k > 1$ için $f^k \geq 0$ dır ve $\{f^k\}$ dizisi azalandır, yani her $k > 1$ değeri için, $f^{k+1} \leq f^k$ dır.

Bu demektir ki Adım 4 teki durdurma kriteri, sonlu iterasyonlar sonrasında sağlanacaktır. Dolayısıyla Algoritma 3, $\varepsilon > 0$ toleransına bağlı olarak, A kümesi ne kadar küme içeriyorsa hepsini hesaplayacaktır.

$\varepsilon > 0$ toleransının seçimi, Algoritma 3 için önem taşımaktadır. ε un büyük değerleri, büyük kümeler oluşturarak sonuç verecek; ε un küçük değerleri ise, yapay (artificial) kümeler üretecektir. ε için tavsiye edilen değer aralığı $\varepsilon \in [0.01, 0.1]$ dir (Bagirov, 2008).

3.4. Başlangıç Küme Merkezleri Bulmak Algoritması

Algoritma 3 için iyi başlangıç küme merkezlerinin seçimini sağlayan $Algoritma$ 4 bu algoritmanın en önemli parçasıdır.

Varsayalım ki, $k > 1$ ve $(k-1)$ bölünme problemi için $(x^1, \dots, x^{k-1})$ küme merkezleri biliniyor. k-bölünme problemi için aşağıdaki fonksiyon tanımlanabilir

$$\bar{f}_k(y) = \frac{1}{m} \sum_{i=1}^{m} min\left\{d_{k-1}^i, \left\|y - a^i\right\|^2\right\} \quad (3.1)$$

Burada $y \in R^n$, k. kümenin merkezi ve d_{k-1}^i de (k-1)-bölünme problemindeki i. elemanın, kendisine en yakın merkeze olan uzaklığı olarak tanımlansın. $\bar{f}_k$ fonksiyonu, yardımcı kümeleme fonksiyonu olarak isimlendirilmektedir. Bu fonksiyon sadece y değişkene bağlı olup, aşağıdaki şekilde ifade edilebilir.

$$\bar{f}_k(y) = f_k(x^1, \dots, x^{k-1}, y) \quad (3.2)$$

Her bir $a \in A^i$ veri noktası için, aşağıdaki φ fonksiyonunu ele alalım

$$\varphi_{ik}(y) = min\left\{d_{k-1}^i, \left\|y - a^i\right\|^2\right\} \quad (3.3)$$

Bu fonksiyon sabitlerin minimumu olarak tanımlanır ve quadratik bir fonksiyondur. Eğer a^i veri noktası bir küme merkezi ise o zaman $d_{k-1}^i = 0$ ve $\varphi_{ik}(y) = 0$ olacaktır. Aksi durumda ise

$$\varphi_{ik}(y) = \begin{cases} \left\|y - a^i\right\|^2 & , \left\|y - a^i\right\|^2 < d_{k-1}^i \\ d_{k-1}^i & , \left\|y - a^i\right\|^2 \geq d_{k-1}^i \end{cases} \quad (3.4)$$

$k < m$ için açıktır ki, $\varphi_{ik}(y) > 0$ olur (bazı $a^i \in A$ ve $y \in R^n$ için) ve her $y \in R^n$ için $\bar{f}_k(y) > 0$ dır. Bir minimum fonksiyonu olarak, φ_{ik} pürüzlü (nonsmooth) ve dışbükey olmayan (nonconvex) bir fonksiyondur. Bu fonksiyon $\left\|y - a^i\right\|^2 = d_{k-1}^i$ eşitliğini sağlayan $y \in R^n$ noktalarında diferansiyallenebilir değildir. Dolayısıyla $\bar{f}_k$ fonksiyonu da pürüzlüdür ve dışbükey değildir. Bu fonksiyonun diferansiyallenemeyen olduğu noktaları içeren küme, φ_{ik} fonksiyonunun diferansiyallenemeyen olduğu kümelerin birleşimi olarak ifade edilebilir.

$\bar{f}_{k-1}$ fonksiyonunun minimum değeri olan f_{k-1}^* aşağıdaki şekilde ifade edilir:

$$f_{k-1}^{*} = \frac{1}{m} \sum_{i=1}^{m} d_{k-1}^{m} \quad (3.5)$$

Eğer $a^j \epsilon A$ veri noktası bir küme merkezi değilse, o zaman

$$f_k(\mathrm{x}^1, \dots, \mathrm{x}^{\mathrm{k}-1}, \mathrm{a}^{\mathrm{j}}) = \frac{1}{\mathrm{m}} \sum_{\mathrm{i}=1}^{\mathrm{m}} \min\left\{\mathrm{d}_{\mathrm{k}-1}^{\mathrm{i}}, \left\|\mathrm{a}^{\mathrm{j}} - \mathrm{a}^{\mathrm{i}}\right\|^2\right\} \quad (3.6)$$

Bu noktada, $a^j \epsilon A$ verildiğinde aşağıda belirtilen iki indis kümesini ele alalım:

$$I_1 = \left\{i \epsilon \{1, \dots, m\}: \left\|\mathrm{a}^{\mathrm{j}} - \mathrm{a}^{\mathrm{i}}\right\|^2 \geq \mathrm{d}_{\mathrm{k}-1}^{\mathrm{i}}\right\}, \quad (3.7)$$

$$I_2 = \left\{i \epsilon \{1, \dots, m\}: \left\|\mathrm{a}^{\mathrm{j}} - \mathrm{a}^{\mathrm{i}}\right\|^2 < \mathrm{d}_{\mathrm{k}-1}^{\mathrm{i}}\right\} \quad (3.8)$$

Bu notasyon kullanılarak b_j aşağıdaki şekilde yeniden yazılabilir

$$b_j = \sum_{i \epsilon I_2} \left(\mathrm{d}_{\mathrm{k}-1}^{\mathrm{i}} - \left\|\mathrm{a}^{\mathrm{j}} - \mathrm{a}^{\mathrm{i}}\right\|^2\right) \quad (3.9)$$

O halde,

$$\overline{f_k}(a^j) = f_k(\mathrm{x}^1, \dots, \mathrm{x}^{\mathrm{k}-1}, \mathrm{a}^{\mathrm{j}}) = \frac{1}{\mathrm{m}}\left(\sum_{i \epsilon I_1} \mathrm{d}_{\mathrm{k}-1}^{\mathrm{i}} + \sum_{i \epsilon I_2} \left\|\mathrm{a}^{\mathrm{j}} - \mathrm{a}^{\mathrm{i}}\right\|^2\right) \quad (3.10)$$

ve dolayısıyla

$$f_{k-1}(\mathrm{x}^1, \dots, \mathrm{x}^{\mathrm{k}-1}) - f_k(\mathrm{x}^1, \dots, \mathrm{x}^{\mathrm{k}-1}, \mathrm{a}^{\mathrm{j}}) = \sum_{i \epsilon I_2} \left(\mathrm{d}_{\mathrm{k}-1}^{\mathrm{i}} - \left\|\mathrm{a}^{\mathrm{j}} - \mathrm{a}^{\mathrm{i}}\right\|^2\right) = b_j \quad (3.11)$$

Bu demektir ki, eğer k. kümenin merkezine bir başlangıç noktası olarak a^j seçilirse, o zaman f_{k-1} fonksiyonunun optimal değeri $b_j \geq 0$ kadar azaltılabilir. Dolayısıyla k. kümenin merkezi için bir başlangıç noktası olarak, en büyük b_j değerine sahip olan veri noktasını seçmemiz mantıklı olacaktır.

Şimdi aşağıdaki kümeyi ele alalım

$$\overline{D} = \left\{ y \in R^n : \left\| y - a^i \right\|^2 \geq d_{k-1}^i \right\}. \quad (3.12)$$

$\overline{D}$ kümesi, kendi elemanı olan herhangi bir y noktası ile herhangi bir $a^i \in A$ veri noktası arasındaki uzaklığın; a^i veri noktası ile bu veri noktasının kendi küme merkezine olan uzaklığından daha küçük olmadığı bir kümedir. Ayrıca aşağıdaki kümeyi de ele alalım:

$$D_0 = R^n \setminus \overline{D} \equiv \left\{ y \in R^n : \exists I \subset \{1, \dots, m\}, I \neq \emptyset : \left\| y - a^i \right\| < d_{k-1}^i, \forall i \in I \right\} \quad (3.13)$$

Aşağıdaki denklem göz önüne alınacak olursa

$$\bar{f}_k(y) = d_0 \equiv \frac{1}{m} \sum_{i=1}^{m} d_{k-1}^i, \quad \forall y \in \overline{D} \quad (3.14)$$

Açıktır ki, burada her $j = 1, \dots, k-1$ değeri için $x^j \in \overline{D}$ ve her $a^i \in A$ elemanı için $a^i \in D_0$, $a^i \neq x^j$, $j = 1, \dots, k-1$ dir. Ayrıca açıktır ki her $y \in D_0$ için $\bar{f}_k(y) < d_0$ dır.

k. kümenin merkezi için bir aday başlangıç noktası olarak herhangi bir $y \in D_0$ seçilebilir. $\bar{f}_k$ fonksiyonu pek çok lokal minimuma sahip olan dışbükey olmayan bir fonksiyondur ve bu fonksiyonun global enküçükleyicisi, k. kümenin merkezi için bir başlangıç noktası olmaya en iyi adaydır. Fakat $\bar{f}_k$ nın global enküçükleyicisini, polinom zamanda bulabilmek, her zaman mümkün değildir. Bu nedenle, $\bar{f}_k$ fonksiyonunun bir lokal enküçükleyicisini bulmak için bir algoritma önerilmiştir.

Her $y \in D_0$ için aşağıdaki kümeler ele alınsın

$$S_1(y) = \left\{ a^i \in A : \left\| y - a^i \right\| = d_{k-1}^i \right\}. \quad (3.15)$$

$$S_2(y) = \left\{ a^i \in A : \left\| y - a^i \right\| < d_{k-1}^i \right\}. \quad (3.16)$$

$$S_3(y) = \left\{ a^i \in A : \left\| y - a^i \right\| > d_{k-1}^i \right\}. \quad (3.17)$$

Her $y \in D_0$ için $S_2(y) \neq \emptyset$.

Aşağıdaki algoritma k. kümenin merkezi için bir başlangıç noktası bulmak için önerilmiştir.

Algoritma 4, (Başlangıç noktası bulmak için bir algoritma):

Adım 1: Her bir $a^i \in D_0 \cap A$ için $S_2(a^i)$ kümesi; bu kümenin merkezi olan c^i; ve $\bar{f}_k$ fonksiyonunun c^i deki değeri olan $\bar{f}_{k,a^i} = \bar{f}_k(c^i)$ hesaplanır.

Adım 2:
$$\bar{f}_{k,min} = min_{a^i \in D_0 \cap A} \bar{f}_{k,a^i}\,, \tag{3.18}$$

$$a^j = \arg min_{a^i \in D_0 \cap A} \bar{f}_{k,a^i}, \tag{3.19}$$

ve karşılık gelen c^j merkezi ile $S_2(c^j)$ kümesi bulunur.

Adım 3: $S_2(c^j)$ kümesi ve bu kümenin merkezi, bu kümeden hiçbir veri noktası ayrılmayıncaya veya bu kümeye hiçbir veri noktası geri dönmeyinceye kadar Adım 2 ve 3 yeniden hesaplanır.

3.5. Bulanık c-Ortalamalar algoritması

Kesin-kısıtsız tam bölümlü kümeleme problemi gibi bulanık-kısıtsız tam bölümlü kümeleme probleminde de A kümesindeki noktaların k sayıda S_l ($l = 1, \ldots, k$) kümesine ayrılması amaçlanmıştır. Buradaki fark veri noktalarının her küme için üyelik derecelerinne sahip olmalarıdır. Bu bir noktanın birden fazla kümeye farklı üyelik dereceleriyle ait olması demektir.

Bulanık kümeleme mantığı kümeler arasında kesin sınırlar olmaması sebebiyle gerçek hayat uygulamalarına daha uygundur. Bulanık kümelemede sıfırla bir arasında olan üyelik derecesi gerçek uygulamalarda noktanın bir

kümeye net atanması yerine kullanılıyor. Bu yöntem 1973 yılında Dunn tarafından oluşturulmuş, 1981 yılında Bezdek tarafından daha da geliştirilmiştir.

Bulanık kümelemede amaç fonksiyonu aşağıdaki şekilde verilebilir:

$$J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^{m} \left\| x_{i-} c_j \right\|^2, \ 1 \leq m < \infty \tag{3.20}$$

Burada u_{ij}; x_i noktasının j kümesine üyelik değeridir, x_i; d - boyutlu veri kümesinin i-nci elemanıdır. c_j; j kümesinin d - boyuta sahip merkez noktası ve $|| * ||$ ise herhangi veri noktasıyla merkez noktası arasında benzerliğin normal ifadesidir. Bulanık kümeleme aşağıda belirtilen amaç fonksiyonunun u_{ij} üyelik değerlerinin ve c_j merkezlerinin güncellenmesiyle iterasyonal optimizasyon yapılarak gerçekleştirilir:

$$U_{ij} = \frac{1}{\sum_{k-1}^{C} \left(\frac{\left\| x_i - c_j \right\|}{\left\| x_i - c_k \right\|} \right)^{\frac{2}{m-1}}} \tag{3.21}$$

$$C_j = \frac{\sum_{i=1}^{N} u_{ij}^{m} . x_i}{\sum_{i=1}^{N} u_{ij}^{m}} \tag{3.22}$$

Bu iterasyon $\max_{ij} \left\{ \left| u_{ij}^{(k-1)} - u_{ij}^{(k)} \right| \right\} < \xi$ eşitsizliği sağlanınca sonlandırılır. Burada ξ; 0 ile 1 arasında durma kriteri, k ise iterasyon sayısıdır. Bu prosedür yerel minimuma ve ya J_m noktasına yakınsar.

Algoritma adımları aşağıdaki gibidir:

Algoritma 5 (Bulanık c-Ortalamalar Algoritması):

Adım 1. $U^{(0)} = [u_{ij}]$ matrisi tanımlanır, $k = 0$ atanır;

Adım 2. k.cı iterasyonda $C^{(k)} = [c_{ij}]$ merkez vektörları $U^{(k)}$ yardımıyla hesaplanır

$$C_j^{(k)} = \frac{\sum_{i=1}^{N} u_{ij}^{m} . x_i}{\sum_{i=1}^{N} u_{ij}^{m}}$$

Adım 3. $U^{(k)}$ güncellenir:

$$U_{ij}^{(k)} = \frac{1}{\sum_{k-1}^{c} \left(\frac{\left\| x_i - c_j \right\|}{\left\| x_i - c_k \right\|} \right)^{\frac{2}{m-1}}}$$

Adım 4. Eğer $\left\| \mathrm{U}^{(\mathrm{k}+1)} - \mathrm{U}^{(\mathrm{k})} \right\| < \xi$ ise durulur, aksi halde $k = k + 1$ atanır ve Adım 2'ye gidilir.

Bulanık üyelik değeri 1 (doğru, nokta kümeye aittir) ye va 0 (yanlış, nokta kümeye ait değil) değerleri yerine, (3.21) eşitliğinde u_{ij} ağırlık değeri kümeye kısmi üyeliği temsil eder. Bu kısmi ağırlık bulanık ağırlık olarak adlandırılır. Bulanık ağırlıkları üretmek için farklı yollar vardır. Bu yollardan bir tanesi aşağıdaki şekilde hesaplanır (A. Zadeh., 1965):

$$\mathrm{u}_{ij} = \frac{1}{\mathrm{D}_{\mathrm{ij}}} \ , (\mathrm{u}_{ij} = 1 \text{ eğer } \mathrm{D}_{\mathrm{ij}} = 0) \qquad (3.23)$$

Öznitelik vektörüyle veri örneğinin arasındaki mesafe büyükse, ağırlık değeri küçükdür. Diğer taraftan mesafe küçüldükce ağırlık değeri de artmış olur.

Yukarıda ifade edilen kesin artımlı ve bulanık modeller gözönüne alınarak bu çalışmada iki farklı kümeleme türünün olumlu yönleri yeni bir algoritmada birleştirilmiştir.

4. MODİFİYE EDİLMİŞ BULANIK GLOBAL KÜMELEME ALGORİTMASI

4.1. Modifiye Edilmiş Bulanık Global c-Ortalamalar Algoritması

Bu bölümde kümeleme probleminin çözümü için artımlı bulanık bir global kümeleme algoritması sunulmuştur. Bulanık değişkenlere sahip artımlı kümeleme algoritması kümeleri dinamik şekilde her seferinde yeni bir küme merkezi ekleyerek oluşturmaktadır. A veri kümesinin k-bölümünün bulunması için artımlı kümeleme algoritması aşağıdaki şekilde verilebilir:

Algoritma 6 (Modifiye Edilmiş Bulanık Global c-Ortalamalar Algoritması - FGMCM):

Adım 1: (Başlangıç) $\varepsilon > 0$ olacak şekilde bir hata toleransı seçilir. A kümesinin merkezi $x^1 \epsilon \mathbb{R}^n$ hesaplanır. U_{i1} = 1, i = 1,…,m. f^1 e, (3.20) deki amaç fonksiyonunun karşılık gelen değeri ve $k = 1$ atanır.

Adım 2: (Bir sonraki küme merkezinin hesaplanması). $k = k + 1$ ataması yapılır. $(k-1)$-bölünmeli problem için küme merkezleri $x^1, \dots, x^{k-1}$ olmak üzere. k. kümenin merkezi için bir başlangıç noktası $(\bar{y} \epsilon \mathbb{R}^n)$ bulmak adına $Algoritma\ 7$ uygulanır.

Adım 3: (Tüm küme merkezlerinin düzeltilmesi) $(x^1, \dots, x^{k-1}, \bar{y})$ yeni bir başlangıç noktası olarak seçilir, k-kümeleme problemi için Bulanık c-Ortalamalar algoritması uygulanır. $y^1, \dots, y^k$ bu problemin bir çözümü ve bu çözüme karşılık gelen (2.7)-(2.8) deki amaç fonsiyonunun değeri f^k ya atanır.

Adım 4: (Durma kriteri) $\frac{f^{k-1}-f^k}{f^1} < \varepsilon$ ise durulur, değilse $x^i = y^i$, $i = 1, \dots, k$ ataması yapılır ve Adım 2 ye dönülür.

Açıktır ki, burada her $k > 1$ için $f^k \geq 0$ dır ve $\{f^k\}$ dizisi azalandır, yani her $k > 1$ değeri için, $f^{k+1} \leq f^k$ dır.

Bu demektir ki, Adım 4 teki durdurma kriteri, sonlu iterasyonlar sonrasında sağlanacaktır. Dolayısıyla Algoritma 6, $\varepsilon > 0$ toleransına bağlı olarak, A kümesi ne kadar küme içeriyorsa hepsini hesaplayacaktır.

$\varepsilon > 0$ toleransının seçimi, Algoritma 6 için önem taşımaktadır. ε un büyük değerleri, büyük kümeler halinde görünüm ile sonuç verecek; ε un küçük değerleri ise, yapay (artificial) kümeler üretecektir. ε için tavsiye edilen değer aralığı $\varepsilon \in [0.01, 0.1]$ dir (Bagirov, 2008).

4.2. Bulanık Kümelemede Başlangıç Küme Merkezlerinin Bulunması

Algoritma 6 için doğru başlangıç noktalarının seçimini sağlayan Algoritma 7 önem taşımaktadır.

Algoritma 7 (Bulanık kümeleme problemi için başlangıç noktası bulunması):

Adım 1: Her bir $a^i \in D_0 \cap A$ için $S_2(a^i)$ kümesi; bu kümenin merkezi olan c^i; ve $\tilde{f}_k$ fonksiyonunun c^i deki değeri olan $\tilde{f}_{k,a^i} = \tilde{f}_k(c^i)$ (3.20) e göre hesaplanıyor.

Adım 2:
$$\tilde{f}_{k,min} = min_{a^i \in D_0 \cap A} \tilde{f}_{k,a^i}, \tag{4.1}$$

$$a^j = \arg min_{a^i \in D_0 \cap A} \tilde{f}_{k,a^i}, \tag{4.2}$$

ve son olarak (3.22) formülüne uygun olarak c^j merkezi bulunur.

5. HESAPLAMA DENEMELERİ

Sunulan algoritmanın (Algoritma 6) verimliliğini göstermek için PC Intel(R) Core(TM) i3 CPU 2.27 GHz ve RAM 4 GB parametrelerine sahip bilgisayarda gerçek veriler üzerinde sayısal hesaplamalar gerçekleştirildi. Aynı zamanda sunulan algoritma lieteratürde yer alan Bulanık k-Ortalamalar algoritmasıyla kıyaslandı.

Hesaplama denemelerinde UC Irvine Machine Learning Repository (http://archive.ics.uci.edu/ml/) veri ambarından alınan 12 adet veri seti kullanıldı (UCI Machine Learning Repository – Yapay öğrenme algoritmalarının analizi için Yapay öğrenme topluluğu tarafından kullanılan veri tabanları koleksiyonlarıdır). Kullanılan verilerle ilgili bilgiler Çizelge 5.1'de verilmiştir.

Bavaria Postal 1 ve Bavaria Postal 2 veri setleri için sonuçlar sırasıyla Çizelge 5.2 ve Çizelge 5.3'de, Iris veri seti için sonuçlar Çizelge 5.4'de, WINE veri seti için sonuçlar Çizelge 5.5'de, CLEVELAND veri seti için sonuçlar Çizelge 5.6'da, LIVER veri seti için sonuçlar Çizelge 5.7'de, IONOSPHERE veri seti için sonuçlar Çizelge 5.8'de, BREAST veri seti için sonuçlar Çizelge 5.9'da, DIABETS veri seti için sonuçlar Çizelge 5.10'da, TSP veri seti için sonuçlar Çizelge 5.11'de, PAGE BLOCK veri seti için sonuçlar Çizelge 5.12 de, PENDIGIT veri seti için sonuçlar Çizelge 5.13'de gösterilmiştir.

Çizelge 5.1: Veri setlerinin açık tanımı

Veri seti	Veri sayısı	Özellik sayısı
BAVARIA1	89	3
BAVARIA2	89	4
IRIS	150	4
WINE	178	14
CLEVELAND	297	13
LIVER	345	6
IONOSPHERE	351	34
BREAST	683	9
DIABETS	768	8
TSP	1060	2
PAGE BLOCK	5473	10
PENDIGIT	10992	16

- k küme sayısıdır;
- f; FCM algoritmasıyla 10 defa hesaplama sonuçlarının ortalama değeri (Bu algoritma başlangıc noktalarına duyarlı olduğu için) ve FGMCM

algoritması için (bir defa) hesaplanan değerdir (bu algoritma hesaplama sayısından bağımsız olup her çalıştırmada aynı sonucu üreteceğinden bir defa çalıştırılmıştır).

- E; hatanin % değeridir (İki algoritmadan daha iyi sonuç (f_{opt}) olanın hatası sıfır olarak varsayılıp
  $E = \frac{(f - f_{opt})}{f_{opt}} . 100$
  formülüne göre hesaplanır.
- t; işlemci çalışma süresidir;
- k_{count} FCM algorirması için 10 defa çalıştırıldığında uzaklık hesaplamalarının ortalama sayısı ve FGMCM algoritması için bir sefer çalıştırıldığında uzaklık hesaplamaları sayıdır.

Burada f, FCM algoritmasıyla 10 defa hesaplama sonuçlarının ortalama değeri ve FGMCM algoritması için tek hesaplanan değeri ve f_{opt} k-bölünmede bu iki algoritmayla hesaplanan sonuçlardan daha iyi olanıdır.

Çizelge 5.2: Bavaria1 veri seti için sonuçlar

k	FCM (10 çalışmanın ortalaması)				FGMCM			
	f	E	t	D_{count}	f	E	t	D_{count}
2	9,93E+11	71,63	00:00:00.5500000	445	5,79E+11	0,00	00:00:00.5530413	356
3	3,90E+11	71,06	00:00:00.5600000	3123,9	2,28E+11	0,00	00:00:00.9480302	7031
4	1,50E+11	25,79	00:00:00.5700000	7796,4	1,19E+11	0,00	00:00:00.8381701	13439
5	1,28E+11	174,01	00:00:00.5270000	7431,5	4,68E+10	0,00	00:00:01.1524557	21894
6	1,15E+11	269,90	00:00:00.6100000	12762,6	3,10E+10	0,00	00:00:01.4022011	32040
7	1,14E+11	379,07	00:00:00.6400000	18752,3	2,38E+10	0,00	00:00:01.9470050	40762
8	6,84E+10	227,18	00:00:00.6270000	22570,4	2,09E+10	0,00	00:00:02.1952856	45746
9	4,86E+10	161,19	00:00:00.6500000	25712,1	1,86E+10	0,00	00:00:02.4449404	52154
10	1,01E+11	511,86	00:00:00.5800000	14952	1,65E+10	0,00	00:00:02.6445966	67284
11	7,65E+10	451,52	00:00:00.6570000	27901,5	1,39E+10	0,00	00:00:03.1785964	73158
12	8,23E+10	550,79	00:00:00.6800000	40584	1,26E+10	0,00	00:00:03.5773653	91314
13	7,35E+10	516,36	00:00:00.6500000	32743,1	1,19E+10	0,00	00:00:03.9459486	100570
14	8,36E+10	679,65	00:00:00.7530000	30651,6	1,07E+10	0,00	00:00:04.3622616	122998
15	3,56E+10	274,26	00:00:00.7370000	82770	9,51E+09	0,00	00:00:04.6280423	209773
20	2,27E+10	310,65	00:00:00.7870000	75116	5,52E+09	0,00	00:00:08.8072048	511750
25	2,18E+10	408,66	00:00:00.9370000	138840	4,28E+09	0,00	00:00:11.2148460	800021
30	1,52E+10	7503,97	00:00:00.9600000	134301	2,00E+08	0,00	00:00:19.7241270	1624517
35	4,75E+09	3767,35	00:00:01.2130000	223034	1,23E+08	0,00	00:00:25.4283391	2744671
40	9,08E+09	8417,33	00:00:01.2770000	262728	1,07E+08	0,00	00:00:30.0118952	3716462
45	5,20E+09	7081,97	00:00:01.4170000	289161	7,24E+07	0,00	00:00:41.7922911	4356906
50	5,79E+09	9446,19	00:00:01.5030000	344875	6,07E+07	0,00	00:00:38.7687368	5464689

Çizelge 5.3: Bavaria2 veri seti için sonuçlar

k	FCM (10 çalışmanın ortalaması)				FGMCM			
	F	E	t	D_{count}	f	E	t	D_{count}
2	82363418373	99,72	00:00:00.5600000	623	41240189657	0,00	00:00:00.6280775	534
3	20355298282	9,92	00:00:00.6130000	5713,8	18518388731	0,00	00:00:00.3438855	3471
4	19256065011	214,95	00:00:00.5930000	5233,2	6114075061	0,00	00:00:00.5036629	20915
5	9168143496	137,06	00:00:00.6130000	10724,5	3867391712	0,00	00:00:00.7406861	36045
6	8093248411	190,63	00:00:00.6630000	8917,8	2784707581	0,00	00:00:00.9049078	45123
7	9287097548	321,79	00:00:00.6600000	12709,2	2201839908	0,00	00:00:01.2423689	86241
8	1787847735	0,00	00:00:00.6630000	30117,6	1967525837	10,05	00:00:01.5564978	88377
9	1713840461	1,34	00:00:00.6500000	23949,9	1691103234	0,00	00:00:01.7100315	99591
10	7033077497	388,00	00:00:00.6600000	21271	1441208251	0,00	00:00:02.2136175	147651
11	4692741809	247,89	00:00:00.9100000	28978,4	1348915101	0,00	00:00:02.4434161	151567
12	7747210229	564,92	00:00:00.7170000	20185,2	1165127683	0,00	00:00:02.9141849	188947
13	3379211056	238,55	00:00:00.6800000	33784,4	998150525	0,00	00:00:03.6026968	245640
14	3357919991	256,84	00:00:00.6370000	31773	941027843,2	0,00	00:00:03.7007947	270560
15	3247665633	270,96	00:00:00.6830000	46324,5	875485513,6	0,00	00:00:04.1765584	297260
20	3036435294	350,57	00:00:00.7200000	55002	673911876	0,00	00:00:07.0666346	520650
25	806169037	47,23	00:00:00.9970000	150187,5	547574949,2	0,00	00:00:11.6787706	1288097
30	1075444060	125,60	00:00:01.0700000	180225	476713099,6	0,00	00:00:16.1523527	1706664
35	546946887	31,55	00:00:01.0900000	185031	415781525,2	0,00	00:00:21.3133292	2628882
40	849139440	126,34	00:00:01.4000000	196156	375160443,5	0,00	00:00:28.3237560	3768883
45	417538722	28,66	00:00:01.3300000	251514	324519713,3	0,00	00:00:35.6682668	4565967
50	133093271	0,00	00:00:01.4030000	266555	288094273,6	116,46	00:00:42.4022704	5305913

Çizelge 5.4: IRIS veri seti için sonuçlar

k	FCM (10 çalışmanın ortalaması)				FGMCM			
	f	E	t	D_{count}	f	E	t	D_{count}
2	340,159714680	163,90	00:00:00.2270000	780	128,89531411	0,00	00:00:00.9375454	1500
3	151,817429309	150,91	00:00:00.2200000	3600	60,50571063	0,00	00:00:01.5426924	35250
4	92,309831866	121,56	00:00:00.2900000	15900	45,81512388	0,00	00:00:02.0846915	37050
5	44,201111751	30,86	00:00:00.3100000	29850	38,71644709	0,00	00:00:02.0913297	38550
6	43,878545902	77,45	00:00:00.2800000	33480	33,21883266	0,00	00:00:02.6636480	40350
7	37,311542254	83,01	00:00:00.2770000	24150	20,59406096	0,00	00:00:03.1572483	71850
8	19,843070491	13,02	00:00:00.3270000	33120	17,66723459	0,00	00:00:03.7352598	82650
9	22,749204832	47,68	00:00:00.3400000	45360	15,39515618	0,00	00:00:04.3483871	165000
10	19,099375950	36,84	00:00:00.3900000	79500	13,67260549	0,00	00:00:04.8610650	223500
11	24,674673359	104,71	00:00:00.3400000	41415	12,05363990	0,00	00:00:06.0355337	358800
12	16,535218858	48,41	00:00:00.4070000	84420	11,73356941	0,00	00:00:06.0276412	364200
13	18,999855153	83,64	00:00:00.4600000	99840	10,06710274	0,00	00:00:07.5465784	473400
14	9,435079323	2,07	00:00:00.6200000	183120	9,22500376	0,00	00:00:07.9796838	588900
15	15,518082961	77,23	00:00:00.5000000	106875	8,69183553	0,00	00:00:08.7852087	703650
20	6,282515872	0,00	00:00:00.6200000	170100	6,57414583	4,76	00:00:16.1532486	1217550
25	5,389403857	1,70	00:00:00.8400000	294000	5,35673476	0,00	00:00:21.1573325	1631550
30	4,407470752	2,91	00:00:00.8070000	243450	4,28303495	0,00	00:00:28.9922859	3335700
35	3,243244788	0,00	00:00:01.2930000	474075	3,49567080	10,43	00:00:37.5927287	4514550
40	4,127971044	33,49	00:00:01.0370000	324600	3,07156528	0,00	00:00:45.6548817	5996400
45	2,421431186	0,00	00:00:01.6930000	675675	2,79167298	12,33	00:00:55.5718831	8160300
50	2,057044877	0,00	00:00:01.6630000	655500	2,51314329	19,26	00:01:12.1013739	10182000

Çizelge 5.5: WINE veri seti için sonuçlar

k	FCM (10 çalışmanın ortalaması)				FGMCM			
	f	E	t	D_{count}	f	E	t	D_{count}
2	5587448	50,60	00:00:00.9070000	783,2	3710066,915	0,00	00:00:01.6639851	1780
3	1776407	0,31	00:00:00.9430000	17355	1770908,144	0,00	00:00:01.7457751	42898
4	1076785	14,35	00:00:01.0570000	38519,2	941655,4366	0,00	00:00:02.7418371	55714
5	652597	0,35	00:00:01.2030000	66216	650327,1178	0,00	00:00:04.0991923	114454
6	458738	0,85	00:00:01.0930000	50196	454888,1759	0,00	00:00:05.6628581	139018
7	415892	4,49	00:00:01.2170000	67159,4	398005,6152	0,00	00:00:07.3178803	151478
8	331821	0,00	00:00:01.2670000	72481,6	339228,7419	2,23	00:00:09.4445816	195622
9	293507	0,00	00:00:01.3830000	97401,6	307772,5895	4,86	00:00:12.3898409	397474
10	205795	44,39	00:00:01.4770000	126380	142522,5848	0,00	00:00:15.2490709	598614
11	167895	31,20	00:00:01.4230000	117088,4	127972,0731	0,00	00:00:18.8798232	831616
12	178845	50,11	00:00:01.8670000	233678,4	119144,6586	0,00	00:00:21.7609895	902104
13	95663	10,19	00:00:01.8970000	231168,6	86815,19428	0,00	00:00:26.0773774	1258460
14	131495	56,76	00:00:01.4130000	100427,6	83882,14004	0,00	00:00:29.9118022	1340696
15	80035	8,38	00:00:01.9730000	247776	73849,4733	0,00	00:00:33.3743522	1434146
20	50971	0,00	00:00:02.2270000	302244	51842,68016	1,71	00:00:57.4538352	2306168
25	38855	0,00	00:00:02.0100000	223390	48208,53159	24,07	00:01:30.7781916	4970650
30	22980	0,00	00:00:03.5400000	525456	46051,5459	100,40	00:02:06.0206362	6861544
35	18831	0,00	00:00:04.0470000	765667	40895,91949	117,17	00:02:47.5433922	8542576
40	15196	0,00	00:00:04.6930000	947672	33657,61977	121,49	00:03:47.4724369	14463568
45	12772	0,00	00:00:05.5900000	1198296	30044,45408	135,23	00:04:49.4770356	17905554
50	10665	0,00	00:00:05.5200000	1172130	25091,27463	135,26	00:05:46.8838754	20414820

Çizelge 5.6: CLEVELAND veri seti için sonuçlar

k	FCM (10 çalışmanın ortalaması)				FGMCM			
	f	E	t	D_{count}	f	E	t	D_{count}
2	319,34	2,23E+02	00:00:00.8430000	1485	98,77186337	0,00	00:00:03.3924531	1188
3	118,93	3,22E+04	00:00:01.1970000	3118,5	3,69E-01	0,00	00:00:03.3290343	2970
4	34,29	1,38E+29	00:00:01.7030000	7484,4	2,48E-26	0,00	00:00:05.4786372	6534
5	4,86	1,96E+28	00:00:01.4570000	12028,5	2,48E-26	0,00	00:00:08.0853441	9504
6	21,74	1,34E+29	00:00:01.6100000	15147	1,63E-26	0,00	00:00:11.5452714	13068
7	1,81	1,21E+28	00:00:01.6000000	16839,9	1,50E-26	0,00	00:00:14.7463253	17226
8	14,73	1,48E+29	00:00:01.6530000	14256	9,94E-27	0,00	00:00:20.2609879	24354
9	1,26	1,53E+28	00:00:01.5870000	19780,2	8,25E-27	0,00	00:00:23.7485486	29700
10	1,47	1,93E+28	00:00:01.6530000	21087	7,63E-27	0,00	00:00:30.4869915	44550
11	1,64	2,15E+28	00:00:01.6630000	30709,8	7,63E-27	0,00	00:00:35.1628026	51084
12	1,43	4,03E+28	00:00:01.6630000	32432,4	3,55E-27	0,00	00:00:42.1137532	76032
13	1,04	2,91E+28	00:00:01.7430000	33976,8	3,55E-27	0,00	00:00:49.6678915	83754
14	0,99	1,92E+28	00:00:01.9170000	35758,8	5,12E-27	0,00	00:00:56.1926109	117018
15	0,29	5,71E+27	00:00:01.6730000	28066,5	5,12E-27	0,00	00:01:02.1736826	125928
20	0,23	1,01E+28	00:00:01.9230000	37422	2,29E-27	0,00	00:01:47.2848947	225720
25	0,30	9,63E+27	00:00:02.1030000	53460	3,09E-27	0,00	00:02:58.2028488	3590136
30	0,09	4,16E+27	00:00:02.2570000	56133	2,09E-27	0,00	00:04:11.9694065	7576470
35	0,00	5,62E+01	00:00:02.0730000	60291	1,80E-27	0,00	00:05:42.8086813	12821787
40	0,26	1,38E+28	00:00:02.2270000	91476	1,87E-27	0,00	00:07:28.2022964	18659322
45	0,00	2,14E+01	00:00:02.2770000	77517	2,20E-27	0,00	00:09:10.5472465	19073637
50	0,00	0,00E+00	00:00:02.3600000	86130	1,65E-27	8,14	00:11:01.4780363	19442511

Çizelge 5.7: LIVER veri seti için sonuçlar

k	FCM (10 çalışmanın ortalaması)				FGMCM			
	f	E	t	D_{count}	f	E	t	D_{count}
2	361081,4698	2,51	00:00:02.0700000	1932	352228,5524	0,00	00:00:02.3444286	2760
3	208748,7075	0,40	00:00:01.9700000	45643,5	207911,5909	0,00	00:00:03.3543936	74175
4	151557,8038	2,77	00:00:02.3230000	49818	147469,946	0,00	00:00:03.6011962	82455
5	114737,605	1,66	00:00:02.0530000	61237,5	112863,8385	0,00	00:00:06.3070420	229080
6	92288,26695	0,00	00:00:02.1600000	123372	92308,31475	0,02	00:00:06.7393495	311880
7	79628,36074	1,37	00:00:02.1900000	80419,5	78552,28314	0,00	00:00:10.2360135	343275
8	68278,19681	0,50	00:00:02.4100000	160632	67936,73969	0,00	00:00:18.2921962	456435
9	60356,37821	0,75	00:00:02.3270000	156802,5	59905,28053	0,00	00:00:14.3302942	807300
10	54941,91151	1,91	00:00:02.5470000	211140	53910,20023	0,00	00:00:16.3938192	1052250
11	48674,39112	0,00	00:00:02.5430000	229977	48930,33375	0,53	00:00:21.6681558	1192665
12	44890,84695	2,61	00:00:02.9200000	421038	43748,38644	0,00	00:00:24.6618096	1975125
13	41342,8072	2,76	00:00:02.6830000	275827,5	40233,38129	0,00	00:00:27.6341652	2441565
14	38367,78315	2,58	00:00:02.9530000	391230	37401,57463	0,00	00:00:30.1996937	2552655
15	37224,62377	6,67	00:00:02.5300000	186817,5	34896,85398	0,00	00:00:34.0220685	2925255
20	26895,96196	4,57	00:00:04.2100000	917010	25719,75919	0,00	00:01:00.6380743	5621085
25	20992,33719	2,74	00:00:04.7100000	1113487,5	20432,55093	0,00	00:01:32.7546106	10701210
30	17515,14729	3,24	00:00:06.1630000	1687050	16966,05013	0,00	00:02:03.4281892	14406510
35	15069,7572	3,36	00:00:07.2170000	2169877,5	14579,43693	0,00	00:02:42.0318315	20134200
40	13189,13933	2,08	00:00:06.9530000	2042400	12920,64225	0,00	00:03:27.3590131	24714420
45	11474,34715	1,14	00:00:07.7770000	2376877,5	11345,30516	0,00	00:04:41.2715947	39758145
50	10418,80884	2,90	00:00:07.9070000	2178675	10124,76613	0,00	00:06:21.4615916	63304050

Çizelge 5.8: IONOSPHERE veri seti için sonuçlar

k	FCM (10 çalışmanın ortalaması)				FGMCM			
	f	E	t	D_{count}	f	E	t	D_{count}
2	31,75387852	14,72	00:00:01.7370000	1404	27,67961051	0,00	00:00:04.5004389	1404
3	16,9500439	107,82	00:00:01.8870000	2106	8,155952887	0,00	00:00:09.7954798	35100
4	1,07E+01	117,54	00:00:02.1770000	2808	4,93566319	0,00	00:00:17.8372711	183924
5	9,05E+00	249,33	00:00:02.0400000	3510	2,589313397	0,00	00:00:27.7795093	385749
6	5,42E+00	196,28	00:00:01.8170000	4212	1,830838645	0,00	00:00:38.8232925	453141
7	3,91E+00	220,60	00:00:01.8300000	4914	1,220517966	0,00	00:00:51.5455842	541593
8	3,96E+00	344,87	00:00:01.9570000	5616	0,89114514	0,00	00:01:07.1292686	701649
9	2,88E+00	332,88	00:00:01.9370000	6318	0,664739995	0,00	00:01:26.8850087	1371357
10	2,17E+00	316,95	00:00:01.9930000	7020	0,519378636	0,00	00:01:44.8668367	1522287
11	2,24E+00	480,95	00:00:02.0530000	7722	0,385601845	0,00	00:02:08.2191856	2004912
12	1,70E+00	403,31	00:00:02.0700000	8424	0,337453314	0,00	00:02:35.1415130	2902068
13	1,87E+00	587,76	00:00:01.9970000	9126	0,272398349	0,00	00:03:03.7613350	3969810
14	8,75E-01	273,56	00:00:02.0070000	9828	0,234158889	0,00	00:03:35.9019093	5060718
15	1,47E+00	580,17	00:00:02.5100000	10530	0,215834142	0,00	00:04:02.2875618	5108103
20	7,32E-01	546,47	00:00:02.2870000	14040	0,113237392	0,00	00:07:05.4621985	9009468
25	4,83E-01	446,55	00:00:02.4070000	17550	0,088296992	0,00	00:11:33.8665315	16988049
30	3,23E-01	377,05	00:00:02.4630000	21060	0,06763112	0,00	00:16:33.5639800	27455571
35	2,27E-01	251,52	00:00:02.6800000	24570	0,064467225	0,00	00:23:29.2769294	45186687
40	1,91E-01	202,08	00:00:02.7630000	28080	0,063269142	0,00	00:30:40.8971535	60633846
45	1,10E-01	78,37	00:00:02.8000000	31590	0,061800776	0,00	00:39:59.4908467	86223852
50	1,56E-01	155,87	00:00:02.9300000	35100	0,060810998	0,00	00:48:49.4123436	106847559

Çizelge 5.9: BREAST veri seti için sonuçlar

k	FCM (10 çalışmanın ortalaması)				FGMCM			
	f	E	t	D_{count}	f	E	t	D_{count}
2	940,6184349	1,71E+02	00:00:02.2830000	3824,8	346,8923941	0,00	00:00:06.4915657	2732
3	472,2605213	3,82E+02	00:00:03.9100000	5942,1	98,05803835	0,00	00:00:09.3775000	12977
4	3,47E+02	3,85E+02	00:00:02.3030000	13660	7,15E+01	0,00	00:00:17.2634890	18441
5	2,27E+02	5,73E+02	00:00:03.1430000	22880,5	3,38E+01	0,00	00:00:16.7294729	28686
6	1,67E+02	1,58E+03	00:00:03.1670000	27046,8	9,93E+00	0,00	00:00:22.3379983	53274
7	9,70E+01	1,18E+03	00:00:03.7930000	54981,5	7,60E+00	0,00	00:00:28.6304916	62836
8	9,67E+01	1,85E+03	00:00:04.4570000	44258,4	4,95E+00	0,00	00:00:36.0961571	79228
9	8,18E+01	5,59E+03	00:00:04.3970000	135848,7	1,44E+00	0,00	00:00:42.5925525	91522
10	7,40E+01	1,30E+15	00:00:04.2770000	118842	5,70E-12	0,00	00:00:51.0623051	105182
11	4,15E+01	6,96E+02	00:00:04.5970000	176555,5	5,21E+00	0,00	00:00:59.1359739	120208
12	5,84E+01	7,25E+14	00:00:04.3370000	160641,6	8,05E-12	0,00	00:01:08.0449127	136600
13	7,35E+01	1,43E+03	00:00:04.2830000	205104,9	4,82E+00	0,00	00:01:18.8184168	154358
14	3,17E+01	2,08E+24	00:00:05.4300000	151079,6	1,52E-21	0,00	00:01:29.1551789	211730
15	3,81E+01	1,64E+10	00:00:05.1670000	243831	2,33E-07	0,00	00:01:42.3472052	252710
20	1,95E+01	2,17E+03	00:00:05.0100000	357892	8,59E-01	0,00	00:03:01.8417591	483564
25	9,81E+00	1,72E+02	00:00:05.8430000	486637,5	3,60E+00	0,00	00:04:44.1612973	960981
30	9,79E+00	7,56E+02	00:00:06.4800000	592161	1,14E+00	0,00	00:06:35.8563273	1361902
35	9,02E+00	1,67E+03	00:00:06.4370000	645435	5,09E-01	0,00	00:08:43.4235393	1743016
40	9,08E+00	2,93E+04	00:00:05.6670000	483564	3,09E-02	0,00	00:35:55.6384977	2908214
45	6,20E+00	1,00E+04	00:00:07.0330000	968152,5	6,11E-02	0,00	00:14:15.3606881	3410219
50	4,80E+00	2,58E+03	00:00:08.0530000	1280625	1,79E-01	0,00	00:17:28.5998876	4981119

Çizelge 5.10: DIABETS veri seti için sonuçlar

k	FCM (10 çalışmanın ortalaması)				FGMCM			
	f	E	t	D_{count}	f	E	t	D_{count}
2	27659,72	17,51	00:00:04.1670000	3072	23538,95964	0,00	00:00:07.2175795	3072
3	9763,50	0,37	00:00:04.6100000	149068,8	9727,035467	0,00	00:00:16.5557688	79104
4	5487,53	0,00	00:00:05.3130000	538521,6	5487,534984	0,00	00:00:26.1325551	678144
5	3455,16	0,18	00:00:05.8700000	619776	3448,991188	0,00	00:00:39.6741339	1000704
6	2359,75	0,00	00:00:05.9230000	814694,4	2359,703507	0,00	00:00:58.4543885	2341632
7	1992,64	13,02	00:00:08.7770000	1237555,2	1763,067551	0,00	00:01:19.9486974	3782400
8	1318,21	0,00	00:00:08.7370000	1731993,6	1318,212627	0,00	00:01:46.6716950	6381312
9	1048,69	2,40	00:00:08.4670000	1774310,4	1024,107255	0,00	00:02:15.4989234	8648448
10	866,82	10,02	00:00:09.6730000	1499136	787,8450969	0,00	00:02:44.2900472	10184448
11	710,54	4,16	00:00:10.7700000	2257305,6	682,1922277	0,00	00:03:22.6204079	14408448
12	588,96	2,34	00:00:10.3570000	2547302,4	575,4993158	0,00	00:03:54.6121772	14804736
13	531,67	4,30	00:00:11.2700000	2432102,4	509,770523	0,00	00:04:33.5251038	18099456
14	464,47	1,24	00:00:13.8970000	3114854,4	458,7970226	0,00	00:05:10.9153024	18572544
15	391,31	2,15	00:00:10.4730000	2287872	383,0645111	0,00	00:05:55.2400818	21314304
20	215,37	0,00	00:00:21.6330000	6248448	229,1530317	6,40	00:10:30.6439724	42306048
25	133,48	0,00	00:00:26.3530000	7981440	148,9155976	11,56	00:16:43.5791164	77933568
30	83,12	0,00	00:00:22.0000000	6465024	113,656639	36,73	00:23:36.1694973	105924096
35	51,83	0,00	00:00:20.8870000	5693184	89,00901311	71,74	00:31:41.5688671	136035072
40	30,39	0,00	00:00:26.2900000	7928832	57,61058954	89,57	00:40:03.2513109	149618688
45	10,58	59,94	00:00:17.3500000	4064256	6,613947929	0,00	00:49:30.0585452	157671168
50	1,83	95,69	00:00:13.2730000	1693440	0,935796807	0,00	00:59:55.9337937	168603648

Çizelge 5.11: TSP veri seti için sonuçlar

k	FCM (10 çalışmanın ortalaması)				FGMCM			
	f	E	t	D_{count}	f	E	t	D_{count}
2	9719128577109350	19,56	00:00:06.1130000	5300	8129108495305950	0,00	00:00:04.0158374	8480
3	4665127657947810	0,48	00:00:05.9170000	198750	4642745259160100	0,00	00:00:16.3566228	145220
4	3159697751165390	1,90	00:00:06.0170000	256520	3100888935996900	0,00	00:00:22.8667654	509860
5	2300623027779090	0,20	00:00:10.3300000	601020	2296035022204430	0,00	00:00:24.6412358	780160
6	1821437985299850	0,68	00:00:07.7300000	587028	1809104108312710	0,00	00:00:34.2224419	1079080
7	1480846726821120	0,00	00:00:06.4770000	454104	1585254399143180	7,05	00:00:43.1700302	1093920
8	1215726163183300	1,05	00:00:10.5770000	919232	1203133596411060	0,00	00:00:49.4258591	1687520
9	1069711233374000	3,60	00:00:07.1370000	776556	1032519982872180	0,00	00:01:06.7601022	2393480
10	941778471367549	5,94	00:00:08.1330000	1555020	888987852202098	0,00	00:01:18.3914718	3061280
11	792679429413066	1,20	00:00:08.2370000	1449338	783245503482232	0,00	00:02:22.6365732	4472140
12	709653914233167	1,13	00:00:07.7800000	1235112	701749788992608	0,00	00:02:21.8094964	5985820
13	634689619571697	0,00	00:00:08.3100000	1588834	647523726170927	2,02	00:02:48.5704486	6123620
14	567569983196393	0,00	00:00:10.8930000	2916060	569019653459218	0,26	00:03:07.8699913	8572220
15	523169414306253	0,00	00:00:09.0870000	2012940	523293796046833	0,02	00:02:46.2050137	10639220
20	365448632911904	2,65	00:00:10.8270000	3014640	356003644140291	0,00	00:04:48.1725781	26112040
25	268649582971782	0,00	00:00:15.4200000	5936000	270921111891177	0,85	00:06:53.4707062	57902500
30	213715693050397	0,00	00:00:14.9930000	5440980	219575878203061	2,74	00:09:23.3998918	74704560
35	169934325218088	0,00	00:00:19.8470000	8614620	176736446719630	4,00	00:12:33.1384653	121143160
40	146582090968849	0,00	00:00:18.6770000	7844000	148402537620216	1,24	00:15:55.8313359	137049520
45	125351974908950	0,00	00:00:16.4770000	6057900	130645067240350	4,22	00:20:46.8459217	194088120
50	107969120161041	0,00	00:00:21.9500000	9736100	119926939218492	11,08	00:23:52.5072863	232558700

Çizelge 5.12: PAGE BLOCK veri seti için sonuçlar

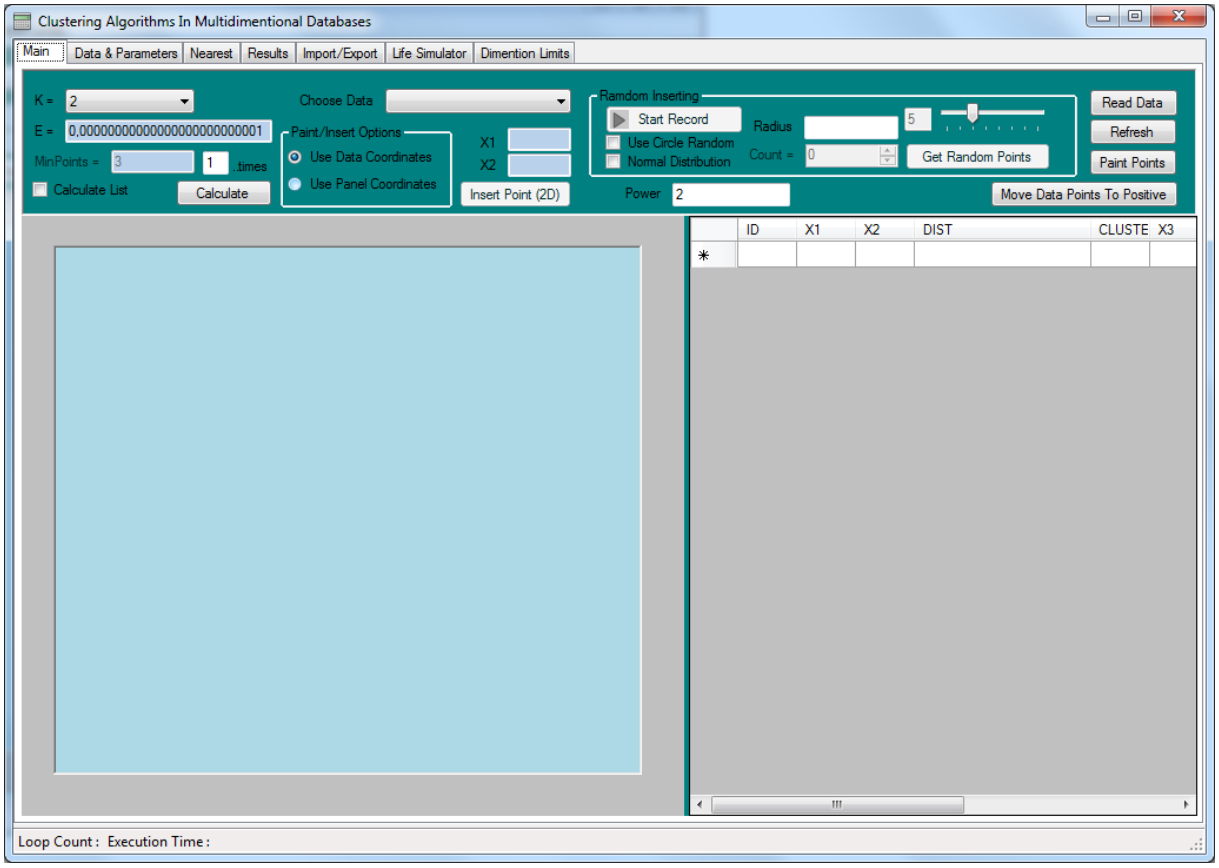
k	FCM (10 çalışmanın ortalaması)				FGMCM			
	f	E	t	D_{count}	f	E	t	D_{count}
2	71340283,55	58,51	00:00:05.6570000	21892	45006430,35	0,00	00:05:38.5083680	21892
3	26203146,3	5,58	00:00:07.7530000	666611,4	24817651,3	0,00	00:14:22.9942258	4356508
4	1,47E+07	0,00	00:00:08.4230000	849409,6	1,57E+07	6,95	00:24:54.6211853	11537084
5	1,02E+07	16,61	00:00:09.9600000	1526967	8738043,192	0,00	00:38:29.0036795	18405699
6	7,57E+06	20,22	00:00:11.4900000	2026104,6	6294490,316	0,00	00:53:55.8814841	25236003
7	6,35E+06	24,17	00:00:13.3400000	2654952,3	5115395,261	0,00	01:13:28.3586868	44391503
8	4,34E+06	28,57	00:00:11.4700000	1983415,2	3376092,89	0,00	01:35:12.7572140	66283503
9	3,66E+06	36,64	00:00:14.5870000	3088413,9	2678738,189	0,00	01:58:17.4563351	90912003
10	2,69E+06	24,12	00:00:18.9670000	4641104	2166409,273	0,00	02:23:59.2562800	106510053
11	2,03E+06	12,32	00:00:25.8730000	7200278,8	1807288,427	0,00	02:52:30.4134416	117948623
12	2,03E+06	34,02	00:00:25.0570000	6928818	1516504,801	0,00	03:23:12.9475463	147896879

Çizelge 5.13: PENDIGIT veri seti için sonuçlar

k	FCM (10 çalışmanın ortalaması)				FGMCM			
	f	E	t	D_{count}	f	E	t	D_{count}
2	2281122,524	26,02	00:00:11.7670000	43968	1810136,892	0,00	00:09:23.5020858	43968
3	567209,4033	0,00	00:00:15.3000000	890352	570543,3882	0,59	00:23:57.9966188	241824
4	3,14E+05	1,53	00:00:16.9430000	1336627,2	309233,7121	0,00	00:42:33.2228959	8551776
5	1,78E+05	2,75	00:00:27.7370000	4105512	173077,8539	0,00	01:04:21.6022562	13278336
6	1,28E+05	0,00	00:00:32.9870000	5427849,6	130775,6089	1,81	01:31:24.1976474	13542144
7	8,27E+04	0,95	00:00:25.8970000	3777950,4	81965,07607	0,00	02:03:20.1091332	20082384
8	6,39E+04	2,98	00:00:33.7400000	5751014,4	62097,19949	0,00	02:39:49.8736324	21489360

6. KÜMELEME YÖNTEMLERİNİN GERÇEKLEŞTİRİLMESİ İÇİN GELİŞTİRİLEN YAZILIM

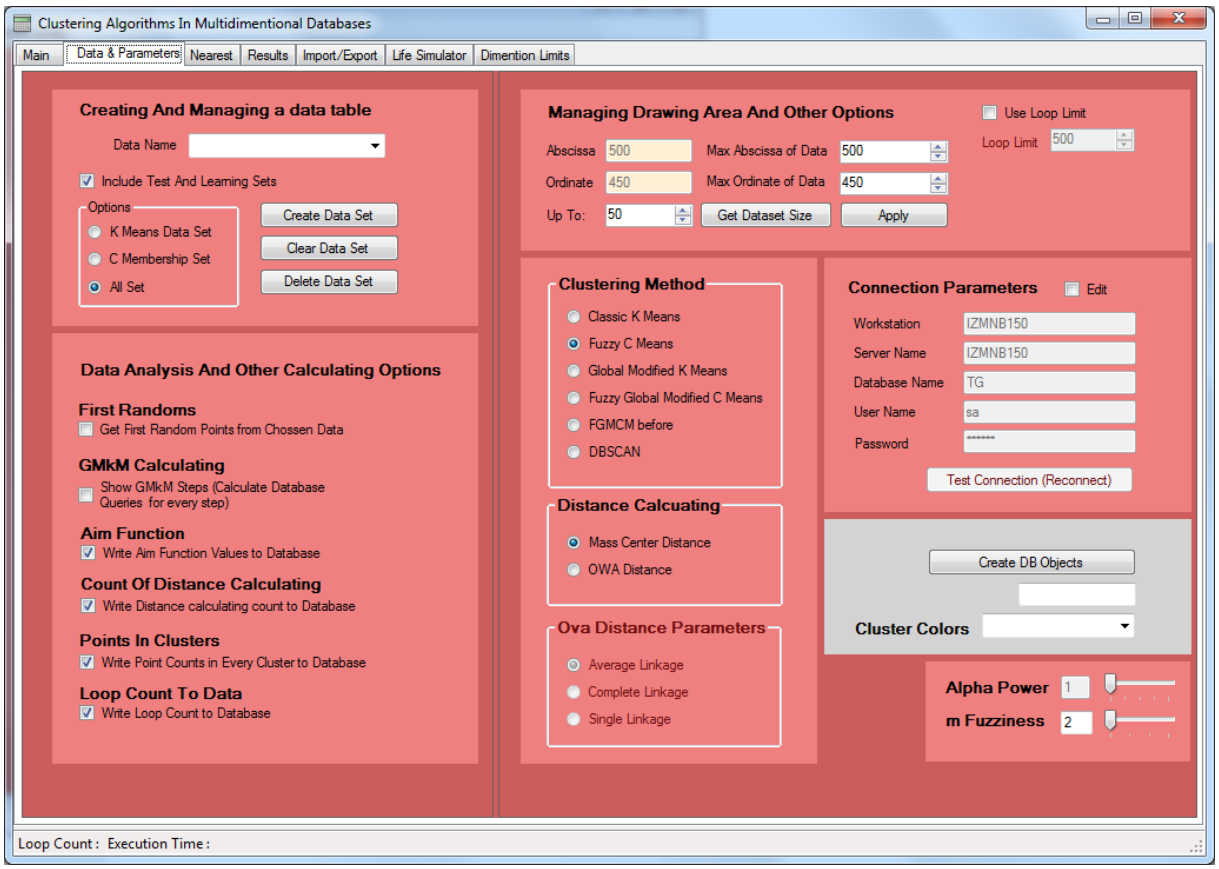
Kümeleme yöntemlerinin farklı parametrelerle farklı veriler üzerinde uyarlanması, sonuçların saklanması, sunulması ve gerekli diğer ek hesaplamalar için C# programlama dilinde bir yazılım geliştirilmiştir. Ek'te yazılımın kodu verilmektedir. Program Microsoft SQL Server imkanlarını kullanmaktadır. Farklı kümeleme algoritmalarını çeşitli parametre değerleriyle ele alarak, isteğe bağlı olarak seçilen veri seti üzerinde kümeleme algoritmaları uygulanabilmektedir (Şekil 6.1).



Şekil 6.1: Ana sekmenin görüntüsü.

Belirtildiği gibi, yazılımın veritabanı altyapısında Microsoft SQL Server kullanılmıştır. Program veritabanı katmanında çok sayıda tablo, prosedür ve fonksiyon kullanmaktadır. Diğer taraftan, yeni bir veritabanına kolaylıkla geçiş sağlanabilmektedir. Veritabanı nesnelerinin oluşturulması için *Data&Parameters*

sekmesindeki *Connection Parameters* bölümünden yeni varitabanına bağlantı bilgileriyle bağlantı yapılabilmektedir (Şekil 6.2). Bağlantı zamanı otomatik olarak program desteği için gereken veritabanı nesneleri oluşturuluyor ve program kullanıma hazır olmuş oluyor. (Kurulum tarzında bir işleme gerek kalmıyor.)

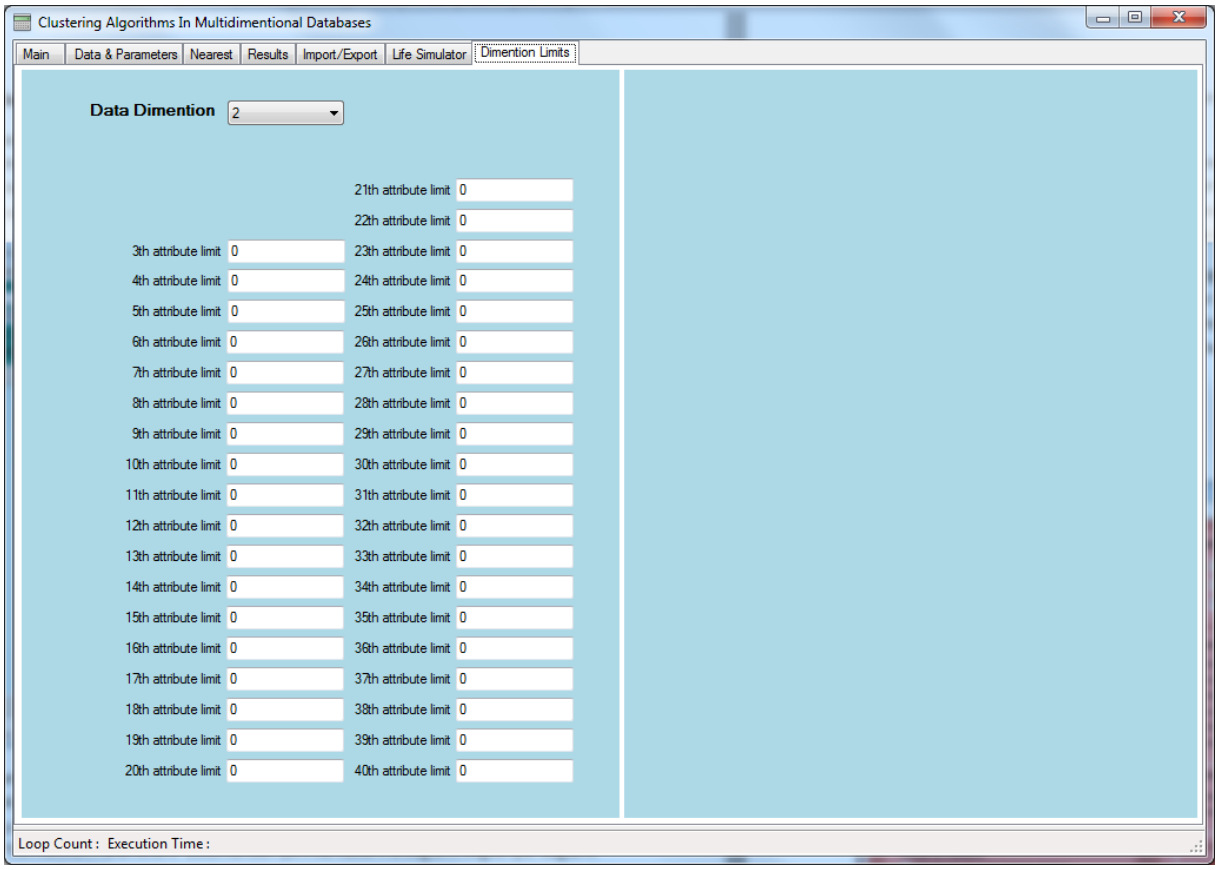


Şekil 6.2: *Data&Parameters* sekmesi.

Program aynı zamanda yeni veri setleri oluşturup, farklı yöntemlerle setlere yeni veri eklemek imkanına sahiptir. Veri setindeki seçilen özelliğe göre setin içeriğini görsel sunma yeteneğine sahiptir ve hesaplama aşamasında ise analiz için çeşitli verilerin veritabanına depolanması özelliklerini de desteklemektedir.

Kümelenmek istenen verinin oluşturulmasından sonra (bu veriler hazır veri setleri olmakla beraber program tarafından manuel de oluşturulabilmektedir) *Data&Parameters* sekmesinde *Clustering Method* bölümünden seçilen veri setini kümelemek için kullanılacak algoritma seçilebilir. *Managing Drawing Area And Other Options* bölümünden verinin ilk iki boyutu için sınırlarının gerekirse manuel veya otomatik olrak tanımlaması yapılabilmektedir. Diğer 38 boyut için

sınırlar *Dimention Limits* sekmesinden yapılabilir(Şekil 6.3). Program 40 boyuta kadar verilerle çalışmayı desteklemektedir.



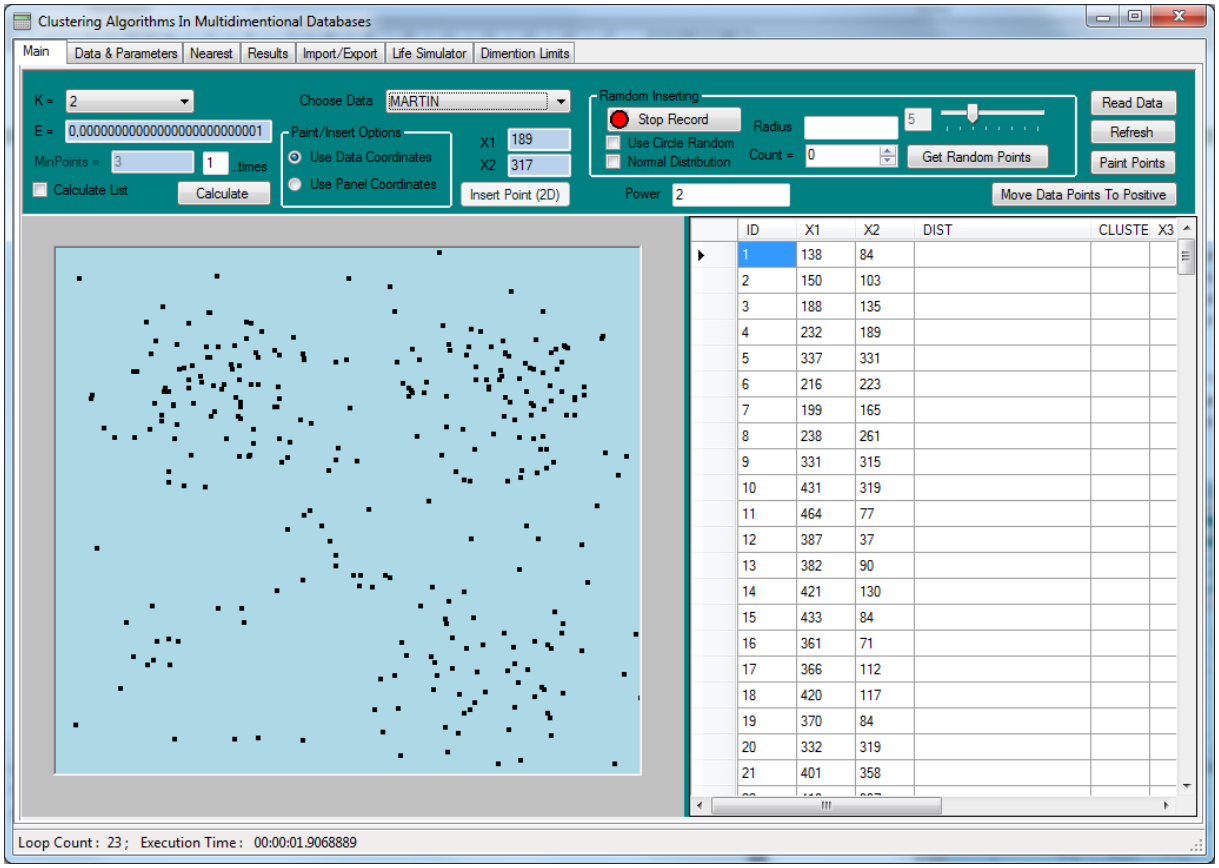
Şekil 6.3: *Dimention Limits* sekmesi.

Hesaplama yapılmadan önce *Data Analysis And Other Calculating Options* bölümünden; hesaplama aşamasında amaç fonksiyon değeri, her kümeye kaç nokta atandığı, kaç uzaklık fonksiyonu hesaplandığı, iterasyonel fark olan epsilon değerinin altına inene kadar kaç döngü olduğu gibi bilgileri gelecek analizler için veritabanına depolamak mümkündür.

Veri girişi

Varolan veya yeni bir veri setine veri girişi yapılması için veri setini seçtikten sonra *Main* sekmesindeki sağ üst köşede bulunan *Random Inserting* sekmesinden ayarlanarak yapıla bilinmektedir. Bu bölümde *Start Record*

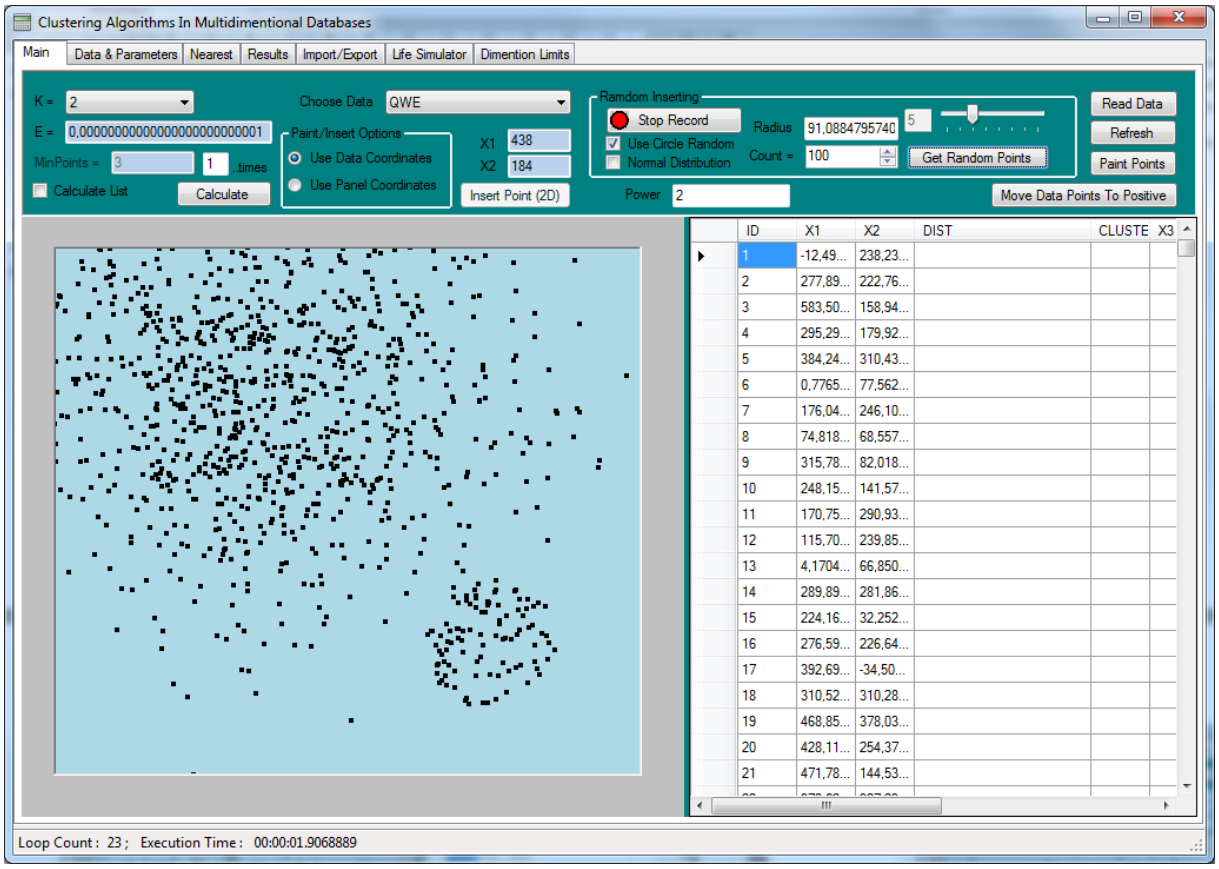
butonunu basarak veri ekleme moduna geçilebilir. Verinin görsel sunulduğu alanda mouse ile tıklayarak noktalar eklenebilir(Şekil 6.4).



Şekil 6.4: Veri Girişi.

Belli sayıda noktayı rastgele oluşturmak için "*Count* =" alanına nokta sayısı yazıp *Get Random Points* butonu tıklanır. Diğer yandan, belli bir dairenin içinde verilen sayıda rastgele nokta oluşturulması için *Use Circle Random* kutucuğunu işaretleyerek verinin sunulduğu alanda merkez noktası ve yarıçap (*Radius*) değerlerini vererek yapılır (*Radius* değeri manuel de verilebilir).

Bir başka veri ekleme yöntemide Müller-Boks dönüştürme (Box–Muller transform) (George Edward Pelham Box ve Mervin Edgar Muller; 1958) yöntemi kullanılarak yapıla bilmektedir. Bu işlem *Normal Distribution* kutucuğunu işaretleyerek yapılabilir (burada eğer *radius* değeri verilirse, veri oluşturulması *radius* değeriyle sınırlandırılacaktır, verilmezse sınırsız dağılımlı şekilde olacaktır (Şekil 6.5).

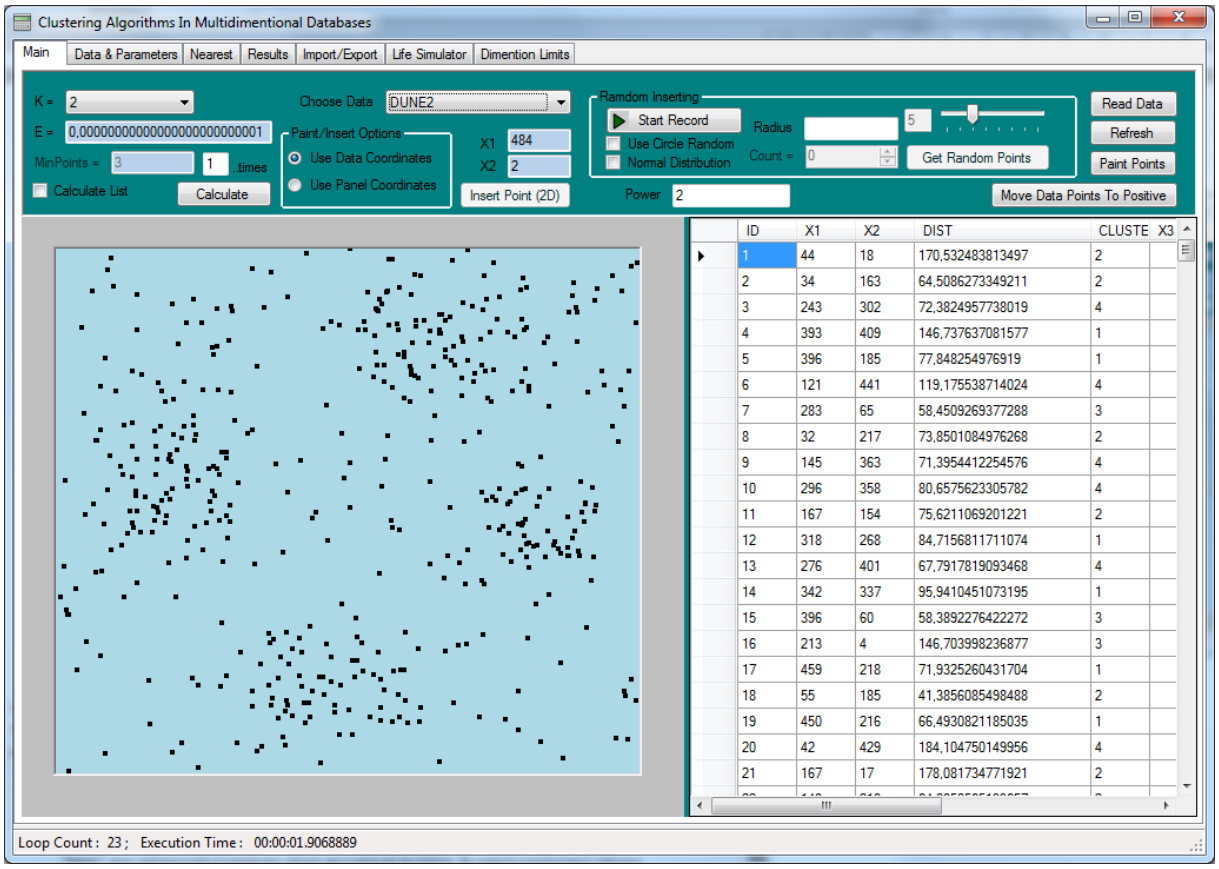


Şekil 6.5: Müller – Box dönüştürme yöntemiyle sınırlı ve sınırsız dağılım.

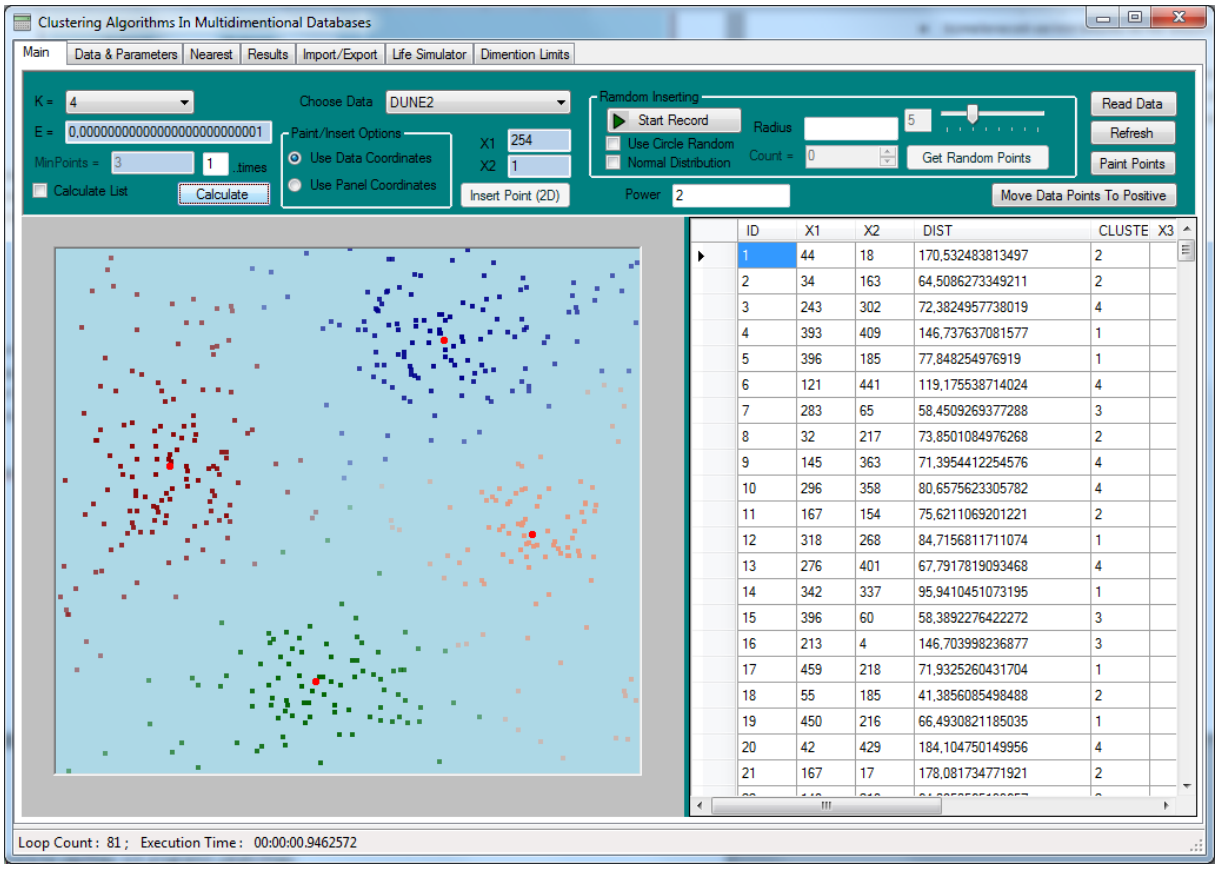
Kümeleme işlemi

Main ana sekmesinde kümeleme işlemi gerçekleştirilebilir. Burada kümelenmesi istenen veri seçildikten sonra k-Ortalamalar tabanlı algoritmalar için k; verilen küme sayısı, iterasyondan çıkış koşulu seçilerek, kümeleme işlemi gerçekleştirilir. Ayrıca kümelemenin kaç defa art-arda yapılması ve her bir k; küme sayısı için çalışması istendiğinde bunun otomatik yapılması için seçenekler bulunmaktadır.

Kümeleme işlemi için veri seti ve parametreler ayarlandıktan sonra *Calculate* butonuyla işlem gerçekleştirilir (Şekil 6.6). Kümeleme sonucunda oluşan her bir kümenin elemanları farklı renklerle birbirlerinden ayrılmaktadırlar. Oluşan kümelerin merkezleri de kırmızı noktalarla gösterilir.



Şekil 6.6: Veri setinin seçilmesi



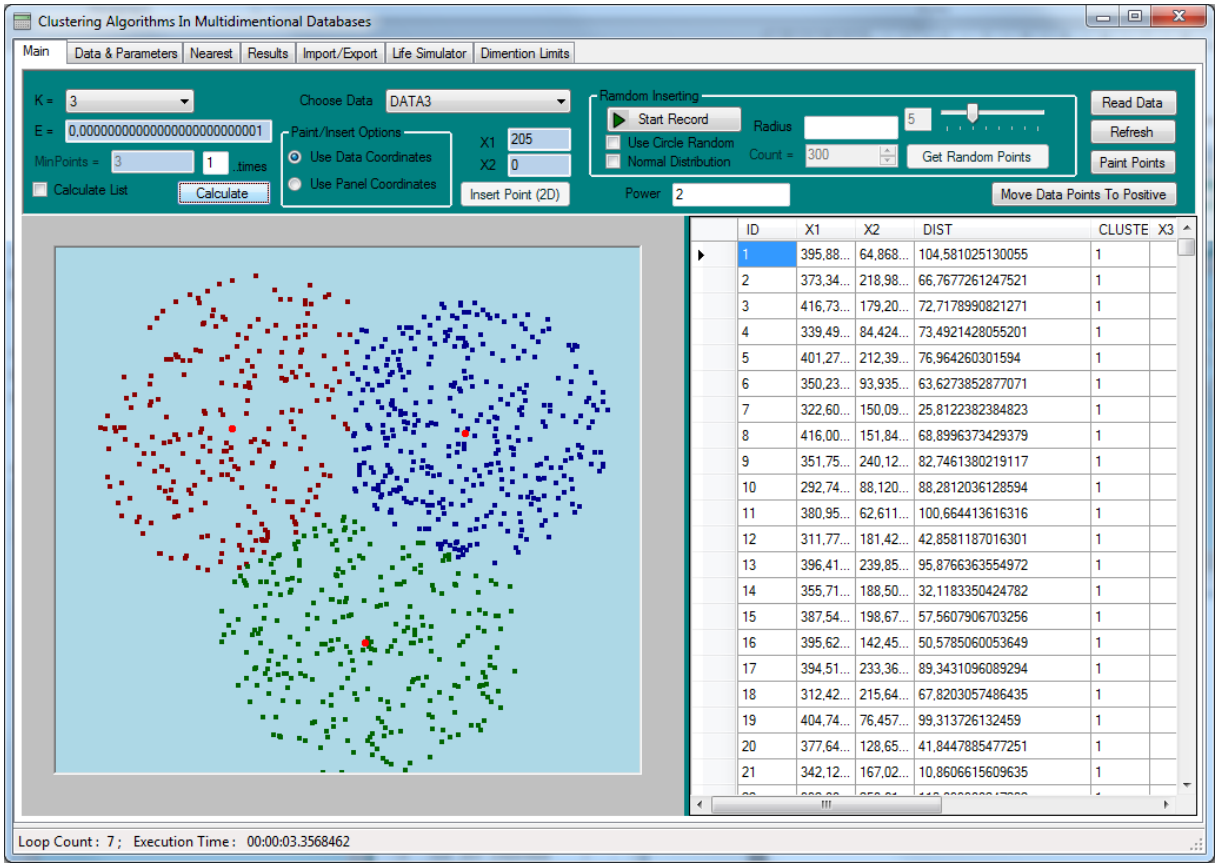
Şekil 6.7: Kümeleme işlemi sonucu (2 boyutlu veri setinin bulanık 4 kümeye ayrılması)

Kümeleme bittiğinde iterasyon sayısı ve çalışma süresi son hesaplama için ekranın alt kısmında görüntülenmektedir. Kümeleme öncesi seçilen parametrelere göre (Şekil 6.7). veritabanında değerler oluşmaktadır. Bu toplanan değerlere göre yöntemlerin detaylı analizi ve kıyaslanması yapılabilir.

Bulanık kümeleme

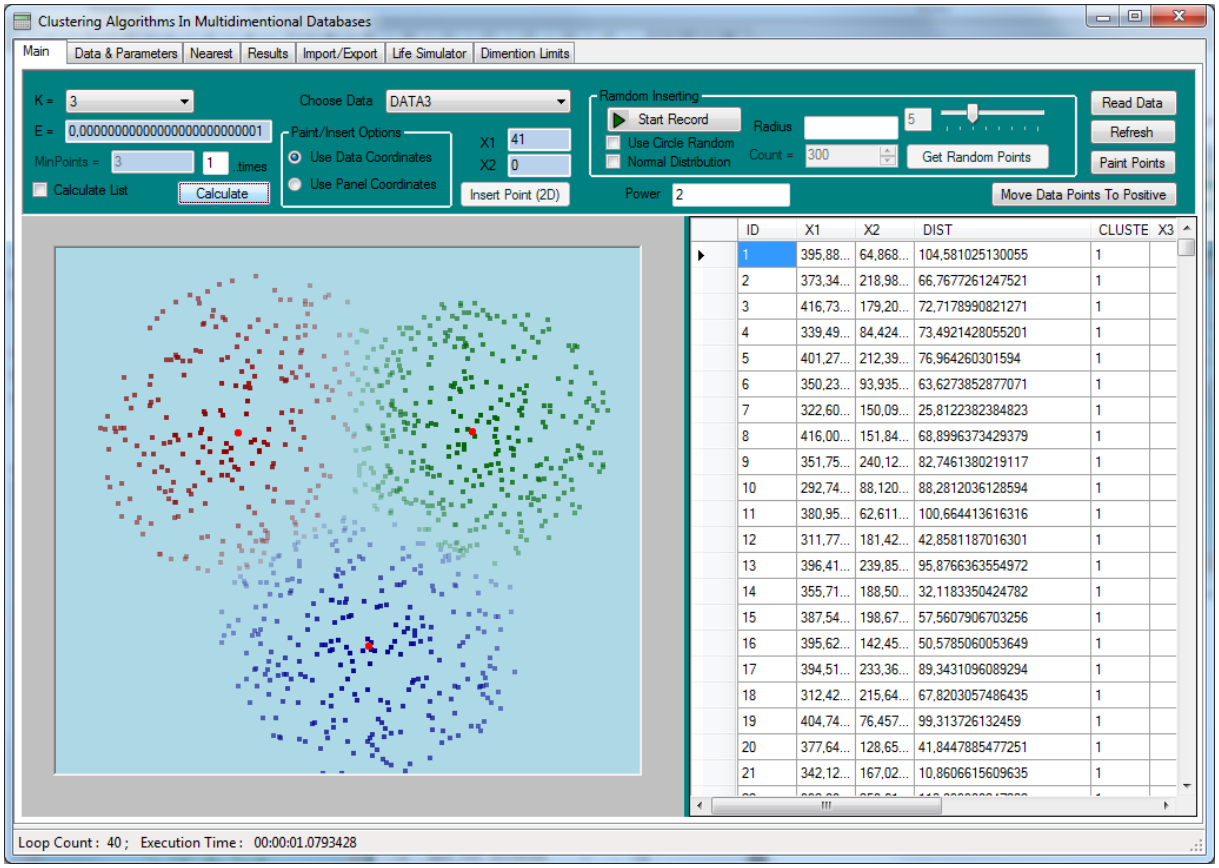
Kesin (Hard) ve bulanık (fuzzy) kümeleme yöntemleri arasında başlıca fark kesin kümelemede her bir veri elemanı sadece tek bir kümeye ait olduğu halde bulanık kümelemede her bir eleman birden fazla kümelere 0 ile 1 arasında değişen üyelik derecelerine göre ait olmaktadır. Bir noktanın bütün üyelik dereceleri toplamı 1' eşit olmalıdır. Buna normallık koşulu denir.

Bu programda kesin ve bulanık yöntemlere göre hesaplanan sonuçların sunumunda noktaların bulanıklılığı da gösterilmektedir. Aynı veri için kesin kümeleme yöntemi olan k-Ortalamalar ve bulanık kümeleme yöntemi olan Bulanık c-Ortalamalar algoritmalarıyla kümeleyerek oluşan sonuçları görsel şekilde görmek mümkündür (Şekil 6.8).



Şekil 6.8: Kesin kümeleme sonucu (küme sınırları kesin ayrılmakta).

Bulanık kümeleme sonucunda ise iki küme sınırlarındaki noktaların her iki kümeye yakın değerlerle ait olmalarını renklerindeki saydamlık düzeylerine göre görmek mümkündür (Şekil 6.9).



Şekil 6.9: Bulanık kümeleme sonucu (küme sınırlarının bulanık şekli).

7. SONUÇ

Bu tezde, bulanık kümeleme probleminin çözümü için yeni bir Bulanık Global Modifiye Edilmiş k-Ortalamalar (FGMCM) algoritması geliştirilmiştir.

Kesin kümeleme probleminin pürüzlü-dışbükey olmayan (nonsmooth nonconvex) formülasyonu yardımıyla tanımlanmış olan Global k-Ortalamalar ve Modifiye Edilmiş Global k-Ortalamalar algoritmaları global çözümü bulmada önemli anavtajlara sahiptir. Kesin kümelemede geliştirilen bu algoritmalar bulanık kümeleme problemlerine uygulanıp, artımlı FGMCM algoritması sunulmuştur.

Sunulan algoritmanın yazılımı Microsoft SQL Server kullanılarak ve C# programlama dilinde arayüz oluşturularak gerçekleştirilmiştir.

Oluşturulan yazılım kullanılarak UC Irvine Machine Learning Repository (http://archive.ics.uci.edu/ml/) veri ambarından seçilen 12 veri seti için hesaplama denemeleri yapılmış, Bulanık k-Ortalamalar algoritması ile karşılaştırmalı analizi verilmiştir.

Yapılan denemeler FGMCM algoritmasının etkinliğini açıkça ortaya koymaktadır.

Gelecek zamandaki çalışmalar içerisinde önerilen algoritmanın hesaplama zamanının iyileştirilmesi planlanmaktadır.

Tez konusu ile dolayılı ilişkisi olan, komşuluğa dayalı bulanık kümeleme algoritması FN-DBSCAN (Fuzzy Neighborhood Density-Based Spatial Clustering of Applications with Noise) konusunda da araştırmalar yapılmış ve bir yayın yayınlanmıştır (Diker and Nasibov, 2012).

KAYNAKLAR DİZİNİ

A. Zadeh L., 1965, *Information and Control* 8, pp. 338-353, Fuzzy Sets

Bagirov A.M., Modified global k-means algorithm for minimum sum-of-squares clustering problems, 2008, *Pattern Recognition* 41, pp. 3192- 3199.

Brandt M.E., Bohant T.P., Kramer L.A. and Fletcher J.M., 1994, Estimation of CSF, white and gray matter volumes in hydrocephalic children using fuzzy clustering of Mr Images, *Computerized Medical Imaging and Graphics*, January–February 1994, pp. 25–34

Chuang K., Tzeng H., Chen S., Wu J. and Chen T., 2006, Fuzzy c-means clustering with spatial information for image segmentation, *Computerized Medical Imaging and Graphics* 30, pp. 9–15 .

Dell'Acqua F. and Gamba P., 2001, Detection of Urban Structures in SAR Images by Robust Fuzzy Clustering Algorithms: The Example of Street Tracking, IEEE *Transactıons On Geoscıence And Remote Sensıng*, No.10, October 2001

Diker A.C. and Nasibov E.E., Estimation of Traffic Congestion Level via FN-DBSCAN Algorithm by Using GPS Data, *Proc. IV International Conference "Problems of Cybernetics and Informatics" (PCI'2012),* September 12-14, 2012, Baku, pp. 65–68..

Georgiou D.N., Karakasidis T.E., Nieto J.J. and Torres A., 2008, Use of fuzzy clustering technique and matrices to classify amino acids and its impact to Chou's pseudo amino acid composition, *Journal of Theoretical Biology* 257, pp. 17–26.

Höppner F., Klawonn F., Krusea R. and Runkler T., 1999, Fuzzy Cluster Analysis, Wiley, Chichester.

Kaymak U. and Setnes M., 2000, Extended Fuzzy Clustering Algorithms, ERIM Report Series *Research In Management*, November 2000, http://www.erim.eur.nl. (Erişim tarihi Haziran 2012)

KAYNAKLAR DİZİNİ (devam)

Likas A., Vlassis N. and Verbeek J., 2001, The global k-means clustering algorithm, *Pattern Recognition* 36, pp. 451 – 461.

Staianoa A., Tagliaferria R. and Pedrycz W., 2005, Improving RBF networks performance in regression tasks by means of a supervised fuzzy clustering, *Neurocomputing* 69, pp. 1570-1581.

UC Irvine Machine Learning Repository (http://archive.ics.uci.edu/ml/) (Erişim tarihi Eylül 2012)

Vanisri D. and Loganathan C., 2010, An Efficient Fuzzy Clustering Algorithm Based on Modified K-Means, *International Journal of Engineering Science and Technology*, pp. 5949-5958.

Velmurugan T. and Santhanam T., 2010, Performance Evaluation of K-Means and Fuzzy C-Means Clustering Algorithms for Statistical Distributions of Input Data Points, *European Journal of Scientific Research*, ISSN 1450-216X No.3, pp.320-330.

Yang J., Hao S. and Chung P., 2001, Color image segmentation using fuzzy C-means and eigenspace projections, *Signal Processing* 82, pp. 461 – 472.

EKLER

EK 1. PROGRAM KODU VE PROSEDÜRLER

EK 1. PROGRAM KODU VE PROSEDÜRLER

a) Artımlı Bulanık Global c-Ortalamalar Algoritması:

```
private void CalculateFuzzyGlobalModifiedCMeans(int k_count, string datasetName,
double epsilon, int dimentionCount,
                                                double m_fuzzifier /*The fuzzifier m
determines the level of cluster fuzziness.*/)
        {
            try
            {
                System.Diagnostics.Stopwatch execTotal = new
System.Diagnostics.Stopwatch();
                execTotal.Start();

                int k = k_count - 1; // 1 less loop should be
                int distCount = 0;
                int fgmcmDistCount = 0;
                // get the dataset
                da = new SqlDataAdapter("SELECT " + columns + " FROM K_MEANS_" +
datasetName + " ", MSSQLConnString);
                DataTable dt = new DataTable();
                da.Fill(dt);

                if (getAimToData)
                {
                    GlobalPoint.AimTableClear(datasetName, "FGMCM", k_count);
                }
                // get the massives
                double[] sumCenters = new double[6];
                // get data to massive
                int dataCount = 0;
                PointDouble[] DataPoints = new PointDouble[0];
                DataPoints = GlobalPoint.GetDataPoints("K_MEANS_" + datasetName,
out dataCount);

                PointDouble[] TempCenterPoints = new PointDouble[k + 1];
                for (int i = 0; i < TempCenterPoints.Length; i++)
                {
                    TempCenterPoints[i] = new PointDouble(0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
                                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                }
                double[,] uMembershipTemp = new double[DataPoints.Length,k + 1];

                double generalAimFunc = 0;
                double generalAimFuncTemp = 0;


                // main loop!!!
                for (int step = 0; step < k; step++)
                {
                    // initilize massive of center points (before and after
chosing the next candidate)
                    PointDouble[] beforeCenterPoints = new PointDouble[step + 1];
                    for (int i = 0; i < beforeCenterPoints.Length; i++)
                    {
                        beforeCenterPoints[i] = new PointDouble(0, 0, 0, 0, 0, 0,
0, 0, 0, 0,
                                                0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
                                                0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
                                                0, 0, 0, 0, 0, 0, 0, 0, 0,
0);
```

```
                        beforeCenterPoints[i].CopyPointFrom(TempCenterPoints[i]);
                    }

                    PointDouble[] centerPoints = new PointDouble[step + 2]; //
size = step+1+1 because need to keep the First Center Point of Data
                    for (int i = 0; i < centerPoints.Length; i++)
                    {
                        centerPoints[i] = new PointDouble(0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
                                                          0, 0, 0, 0, 0, 0, 0, 0,
0,
                                                          0, 0, 0, 0, 0, 0, 0, 0,
0,
                                                          0, 0, 0, 0, 0, 0, 0, 0,
0);
                        centerPoints[i].CopyPointFrom(TempCenterPoints[i]);
                    }

                    double[,] uMembership = new double[DataPoints.Length, step +
2];
                    for (int i = 0; i < step + 2; i++)
                    {
                        for (int j = 0; j < step + 2; j++)
                        {
                            uMembership[i, j] = uMembershipTemp[i, j];
                        }
                    }

                    // massive for getting the new centers after c means
calculating
                    PointDouble[] clCent = new PointDouble[step + 2];
                    for (int i = 0; i < clCent.Length; i++)
                    {
                        clCent[i] = new PointDouble(0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                                          0, 0, 0, 0, 0, 0, 0, 0,
0,
                                                          0, 0, 0, 0, 0, 0, 0, 0,
0,
                                                          0, 0, 0, 0, 0, 0, 0, 0,
0);
                    }

                    // calculatin the mass center on first step
                    if (step == 0)
                    {
                        // write first center point to centersMassive
                        beforeCenterPoints[0] = GlobalPoint.Center(DataPoints);
                        centerPoints[0].CopyPointFrom(beforeCenterPoints[0]);
                        generalAimFunc = GlobalPoint.AimFunction(DataPoints,
beforeCenterPoints, dimentionCount);

                        // first step (for only one center point) the U[i,j] will
be {1,1,...,1}(where j = 1)
                        double[,] uMembershipfirst = new double[DataPoints.Length,
1];
                        for (int i = 0; i < DataPoints.Length; i++)
                        {
                            for (int j = 0; j < step + 1; j++)
                            {
                                uMembershipfirst[i, j] = 1;
                            }
                        }
                        generalAimFunc = GlobalPoint.AimFunctionFuzzy(DataPoints,
beforeCenterPoints, dimentionCount, uMembershipfirst, m_fuzzifier);
                        if (getAimToData)
                        {
                            GlobalPoint.AimValueToData(datasetName, "FGMCM", true,
step + 1, k_count, generalAimFunc);
                        }
                    }

                    // calculating the general aim function (for current center
points)(not in the first step)
                    if (step != 0)
                    {
                        generalAimFunc = GlobalPoint.AimFunctionFuzzy(DataPoints,
beforeCenterPoints, dimentionCount, uMembership, m_fuzzifier);
```

```
                        if (getAimToData)
                        {
                            GlobalPoint.AimValueToData(datasetName, "FGMCM", true,
step + 1, k_count, generalAimFunc);
                        }
                    }

                    // calculating the aim function for every data point and
compare each with general aim function value
                    //(to get the point which minimize aim function)
                    int index = 0;
                    PointDouble candidatePoint = new PointDouble(0, 0, 0, 0, 0, 0,
0, 0, 0, 0,
                                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                    PointDouble selectedNextPoint = new PointDouble(0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
                                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                    double aimFunc = 0;

                    PointDouble[] secondaryTempCenters = new
PointDouble[centerPoints.Length];
                    PointDouble[] oldCentersWithCandidate = new
PointDouble[centerPoints.Length];

                    for (int h = 0; h < centerPoints.Length; h++)
                    {
                        oldCentersWithCandidate[h] = new PointDouble(0, 0, 0, 0,
0, 0, 0, 0, 0, 0,
                                                                     0,
0, 0, 0, 0, 0, 0, 0, 0,
                                                                     0,
0, 0, 0, 0, 0, 0, 0, 0,
                                                                     0,
0, 0, 0, 0, 0, 0, 0, 0, 0);
                        oldCentersWithCandidate[h].CopyPointFrom(centerPoints[h]);
                        // no candidate yet
                    }

                    double[,] secondaryTemp_uMembership = new
double[DataPoints.Length, centerPoints.Length];
                    // compare each data result
                    for (int i = 0; i < DataPoints.Length; i++)
                    {
                        //use current data point as a center and calculate aim
function copy DataPoint to Candidate
                        candidatePoint.CopyPointFrom(DataPoints[i]);
                        oldCentersWithCandidate[step +
1].CopyPointFrom(candidatePoint);

                        // candidate noktayla beraber diger merkezlerle olusan
kumelerin merkezleri hesaplaniyor
                        secondaryTempCenters =
GlobalPoint.GetNewCentersFromSetWithOldCenters(DataPoints,
oldCentersWithCandidate, dimentionCount);

                        // calculate the secondary temp U[i,j] membership massive
                        secondaryTemp_uMembership =
GlobalPoint.GetMembershipValues(DataPoints, secondaryTempCenters, dimentionCount,
m_fuzzifier);

                        // calculatin the aim function value for last center
points (secondaryTempCenters)
                        aimFunc = GlobalPoint.AimFunctionFuzzy(DataPoints,
secondaryTempCenters, dimentionCount, secondaryTemp_uMembership, m_fuzzifier);


                        if (aimFunc < generalAimFunc)
                        {
                            generalAimFunc = aimFunc;
                            if (getAimToData)
                            {
```

```
                                GlobalPoint.AimValueToData(datasetName, "FGMCM",
false, index + 1, k_count, aimFunc);
                            }
                            // make Data point and new center points from old
points as a new Center points
                            for (int h = 0; h < centerPoints.Length; h++)
                            {

centerPoints[h].CopyPointFrom(secondaryTempCenters[h]);
                                uMembership = secondaryTemp_uMembership;
                            }

                            //selectedNextPoint.CopyPointFrom(candidatePoint);
                        }
                        index++;

                    }

                    //centerPoints[step + 1].CopyPointFrom(selectedNextPoint);

                    // massive for getting the new centers after fuzzy c means
calculating
                    //PointDouble[] clCent = new PointDouble[step + 2];
                    for (int i = 0; i < clCent.Length; i++)
                    {
                        clCent[i] = new PointDouble(0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                    }
                    // calculating the Fuzzy c-means for centers and new best
point

                    // if show selected(true) then doDatabaseWorks = true -> do
every database step visually

                    bool doDatabaseWorks = false;
                    // if show selected(true) then doDatabaseWorks = true -> do
every database step visually
                    if (showGMkM || step == k - 1) // if last step
                    {
                        doDatabaseWorks = true;
                    }

                    //CalculateClassicKMeans(step + 2, datasetName, epsilon,
dimentionCount,centerPoints, true, doDatabaseWorks, ref clCent);
                    CalculateFuzzyCMeans(step + 2, datasetName, epsilon,
dimentionCount,
                                         m_fuzzifier,
                                         centerPoints, true, doDatabaseWorks,
true,
                                         ref distCount, ref clCent, ref
uMembership);

                    // get the fcmeans cluster centers to massive
                    for (int s = 0; s < centerPoints.Length; s++)
                    {
                        TempCenterPoints[s].CopyPointFrom(clCent[s]);
                    }
                    for (int i = 0; i < DataPoints.Length; i++)
                    {
                        for (int j = 0; j < step + 2; j++)
                        {
                            uMembershipTemp[i, j] = uMembership[i, j];
                        }
                    }
                    fgmcmDistCount = fgmcmDistCount + distCount;

                    generalAimFuncTemp = GlobalPoint.AimFunctionFuzzy(DataPoints,
clCent, dimentionCount, uMembership, m_fuzzifier);
                    // if General Aim Function Value not changed write to database
last General Aim Value and break the loop
                    if (generalAimFuncTemp == generalAimFunc)
                    {
                        GlobalPoint.AimValueToData(datasetName, "FGMCM", true,
step + 2, k_count, generalAimFunc);
                        break;
```

```
                    }
                    // calculate last the best Aim Value and write to database
                    if (step == k-1) // if last step
                    {
                        if (getAimToData)
                        {
                            generalAimFunc = generalAimFuncTemp;
                            GlobalPoint.AimValueToData(datasetName, "FGMCM", true,
step + 2, k_count, generalAimFunc);
                        }
                    }
                }

                GlobalPoint.ClearCentersFromDatabase(datasetName, "FGMCM");
                GlobalPoint.WriteCentersToDatabase(datasetName, TempCenterPoints,
"FGMCM");

                // write distanceCalcCount to database
                if (distCalcCount)
                {
                    GlobalPoint.ClearDistCountFromDatabase(datasetName, "FGMCM",
k_count);
                    GlobalPoint.WriteDistCountToDatabase(datasetName, "FGMCM",
k_count, dimentionCount, fgmcmDistCount);
                }

                // write the count of points for every cluster to database
                if (writePointPerCluster)
                {
                    ClearPointsPerClusterTable(datasetName, "FGMCM", k_count,
true);
                    AddPointsPerClusterTable(datasetName, "FGMCM", k_count, true);
                }
                execTotal.Stop();

                string queryExecuteTotal = "INSERT INTO CLUSTERING_EXECUTION_TIME
VALUES ( 'K_MEANS_" + datasetName + "', '" + (k+1).ToString() + "', " +
                                    " 'FGMCM', GETDATE(), '" + execTotal.Elapsed +
"', 'TOTAL' )";
                SqlCommand cmdExecuteTotal = new SqlCommand(queryExecuteTotal,
sconn);
                cmdExecuteTotal.ExecuteNonQuery();

                execTime.Text = execTotal.Elapsed.ToString();
            }
            catch (Exception ex) { MessageBox.Show("Error on Global K Means
clustering : " + ex.Message); }

            //-------------------------------------------------------------------
----------------------


            }
```

b) Bulanık c-Ortalamalar algoritması:

```
private void CalculateFuzzyCMeans(int k_count, string datasetName, double epsilon,
int dimentionCount,
                                          double m_fuzzifier /*The fuzzifier m
determines the level of cluster fuzziness.*/,
                                          PointDouble[] firstPoints, bool
useFirstPoints,
                                          bool doDatabaseWorks, bool fromFGMCM,
                                          ref int distCount, ref PointDouble[]
clusteringCentersReturned, ref double[,] uMembership)
        {
            try
            {
                if (epsilon < 0) { throw new Exception("Invalid epsilon value"); }
```

```
                System.Diagnostics.Stopwatch execTotal = new
System.Diagnostics.Stopwatch();
                execTotal.Start();

                System.Diagnostics.Stopwatch execWatch = new
System.Diagnostics.Stopwatch();
                execWatch.Start();

                int k = k_count;
                int w = panel3.Width - 5;  // horizontal
                int h = panel3.Height - 5; // vertical
                double diff = 0;
                double diffBefore = 0;
                int loop = 0;
                distCount = 0;

                // initialize the Data points from dataset to massive
                int dataCount = 0;
                PointDouble[] DataPoints = new PointDouble[0];
                DataPoints = GlobalPoint.GetDataPoints("K_MEANS_" + datasetName,
out dataCount);

                PointDouble[] CenterPoints = new PointDouble[k];
                PointDouble[] NewCenterPoints = new PointDouble[k];
                for (int i = 0; i < CenterPoints.Length; i++)
                {
                    CenterPoints[i] = new PointDouble(0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                    NewCenterPoints[i] = new PointDouble(0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                }
                double[,] distances = new double[DataPoints.Length, k];
                double[,] uMembershipMassive = new double[DataPoints.Length, k];
                double[,] uMembershipMassiveBefore = new double[DataPoints.Length,
k];
                double[] totalDist = new double[DataPoints.Length];
                double[] sumSquareFuzzifier = new double[k];

                // use given first points or not?
                if (useFirstPoints == true)
                {
                    // get first points from parameter
                    for (int f = 0; f < k; f++)
                    {
                        CenterPoints[f].CopyPointFrom(firstPoints[f]);
                    }
                }
                else
                {
                    // get first points randomly
                    RandomPoints points = new RandomPoints();
                    CenterPoints = points.GetPoints(k, w, h,
Convert.ToInt32(att3Edit.Text), Convert.ToInt32(att4Edit.Text), ...
Convert.ToInt32(att40Edit.Text),
                    fromData, datasetName); // gives a massive of random points
                }

                // make center massives equal before loop begin
                for (int i = 0; i < k; i++)
                {
                    NewCenterPoints[i].CopyPointFrom(CenterPoints[i]);
                }

                // begin of algoritm do while ||diffBefore->[U(k)] - diff-
>[U(k+1)]|| < epsilon
                do
                {
                    // get last calculated New points to current center point
massive
                    for (int i = 0; i < k; i++)
                    {
```

```
                        CenterPoints[i].CopyPointFrom(NewCenterPoints[i]);
                    }
                    for (int i = 0; i < k; i++)
                    {
                        NewCenterPoints[i] = new PointDouble(0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
                                                             0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
                                                             0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
                                                             0, 0, 0, 0, 0, 0, 0, 0,
0, 0);
                    }
                    // get the massive of distances
                    for (int i = 0; i < DataPoints.Length; i++)
                    {
                        for (int j = 0; j < k; j++)
                        {
                            distances[i, j] =
GlobalPoint.EuclideanDistance(DataPoints[i], CenterPoints[j], dimentionCount);
                            distCount++;
                        }
                    }

                    // get the massive of total distances
                    for (int i = 0; i < DataPoints.Length; i++)
                    {
                        for (int j = 0; j < k; j++)
                        {
                            //totalDist[i] = totalDist[i] + 1 /
Math.Pow(distances[i, j], (2 / (m_fuzzifier - 1)));
                            if (double.IsInfinity(1 / Math.Pow(distances[i, j], (2
/ (m_fuzzifier - 1)))))
                            {
                                totalDist[i] = totalDist[i] + 0;
                            }
                            else
                            {
                                totalDist[i] = totalDist[i] + 1 /
Math.Pow(distances[i, j], (2 / (m_fuzzifier - 1)));
                            }

                        }
                    }

                    // calculate the U[i,j] memberships massive of data points and
write to database
                    for (int i = 0; i < DataPoints.Length; i++)
                    {
                        for (int j = 0; j < k; j++)
                        {
                            if (double.IsInfinity(1 / Math.Pow(distances[i, j], (2
/ (m_fuzzifier - 1))) / totalDist[i]))
                            {
                                uMembershipMassive[i, j] = 0;
                            }
                            else
                            {
                                uMembershipMassive[i, j] = 1 /
Math.Pow(distances[i, j], (2 / (m_fuzzifier - 1))) / totalDist[i];
                            }
                        }
                    }

                    //calculate the sum of U[i,j] with m fuzzifier
                    for (int j = 0; j < sumSquareFuzzifier.Length; j++)
                    {
                        for (int i = 0; i < DataPoints.Length; i++)
                        {
                            sumSquareFuzzifier[j] = sumSquareFuzzifier[j] +
Math.Pow(uMembershipMassive[i, j], (2 / (m_fuzzifier - 1)));
                        }
                    }

                    // calculate new centers to massive NewCenterPoints
                    for (int j = 0; j < k; j++)
                    {
```

```
                        for (int i = 0; i < DataPoints.Length; i++)
                        {
                            double a = Math.Pow(uMembershipMassive[i, j], (2 /
(m_fuzzifier - 1))) / sumSquareFuzzifier[j];
                            NewCenterPoints[j].X = NewCenterPoints[j].X +
DataPoints[i].X * a;
                            NewCenterPoints[j].Y = NewCenterPoints[j].Y +
DataPoints[i].Y * a;
                            if (dimentionCount > 2) { NewCenterPoints[j].Z =
NewCenterPoints[j].Z + DataPoints[i].Z * a; }
                             ...
                            if (dimentionCount > 39) { NewCenterPoints[j].X40 =
NewCenterPoints[j].X40 + DataPoints[i].X40 * a; }


                        }
                    }

                    // check if ||U(k+1) - U(k)|| < epsilon and initilize
uMembershipMassive
                    diffBefore = diff;
                    for (int i = 0; i < DataPoints.Length; i++)
                    {
                        for (int j = 0; j < k; j++)
                        {
                            diff = diff + uMembershipMassive[i, j] -
uMembershipMassiveBefore[i, j];
                            uMembershipMassiveBefore[i, j] = uMembershipMassive[i,
j];
                        }
                    }
                    // empty the temp massives
                    for (int i = 0; i < DataPoints.Length; i++) { totalDist[i] =
0; }
                    for (int i = 0; i < k; i++) { sumSquareFuzzifier[i] = 0; }


                    loop++;
                    if (useLoopLimitCheck.Checked)
                    {
                        loopLimitNotReached = loop < loopLimitBox.Value;
                    }
                    else {
                        loopLimitNotReached = true;
                    }
                } while (Math.Abs(diffBefore - diff) > epsilon &&
loopLimitNotReached);

                LoopCount.Text = loop.ToString() + " ; ";

                // get the ref parameters (center points)
                for (int i = 0; i < k; i++)
                {
                    clusteringCentersReturned[i] = new PointDouble(0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
                                                                   0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
                                                                   0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
                                                                   0, 0, 0, 0, 0, 0, 0,
0, 0, 0);
                }
                for (int i = 0; i < k; i++)
                {

clusteringCentersReturned[i].CopyPointFrom(NewCenterPoints[i]); // out parameter
                }

                //get the ref parameters (centar points) if uMembership is not
just [0, 0]
                if (uMembership.Length != 0)
                {
                    for (int i = 0; i < DataPoints.Length; i++)
                    {
                        for (int j = 0; j < k; j++)
                        {
                            uMembership[i, j] = uMembershipMassive[i, j];
```

```
                    }
                }
            }

            execWatch.Stop();

            // database queries !!!
            string columnForClusters = "";
            string zeroValues = " ";
            for (int i = 0; i < k; i++)
            {
                columnForClusters = columnForClusters + " J" + (i +
1).ToString() + " AS J" + (i + 1).ToString() + ",";
                //columnForClusters = columnForClusters + " J" + (i +
1).ToString() + ",";
            }
            columnForClusters = columnForClusters.Substring(0,
columnForClusters.Length - 1);
            for (int i = 0; i < 50 - k; i++)
            {
                zeroValues = zeroValues + "0,";
            }
            zeroValues = zeroValues.Substring(0, zeroValues.Length - 1);

            // update U Membership Masiive Values to database

            string queryTrunc = "TRUNCATE TABLE C_MEANS_MEMBERSHIP_" +
datasetName + " ";
            SqlCommand cmdTrunc = new SqlCommand(queryTrunc, sconn);
            cmdTrunc.ExecuteNonQuery();

            // creating a table of values from uMembershipMassive
            da_Bulk = new SqlDataAdapter("SELECT * FROM C_MEANS_MEMBERSHIP_" +
datasetName + " WHERE 1 = 2", MSSQLConnString);
            DataTable dt_Bulk = new DataTable();
            da_Bulk.Fill(dt_Bulk);

            for (int i = 0; i < DataPoints.Length; i++)
            {
                DataRow dr_Bulk = dt_Bulk.NewRow();
                dr_Bulk["ID"] = i + 1;

                for (int j = 0; j < k; j++)
                {
                     dr_Bulk["J" + (j + 1)] = uMembershipMassive[i, j];
                }
                dt_Bulk.Rows.Add(dr_Bulk);
            }

            // Open bulkcopy connection.
            SqlBulkCopy bulkcopy = new SqlBulkCopy(sconn);

            //Set destination table name to table previously created.
            bulkcopy.DestinationTableName = "C_MEANS_MEMBERSHIP_" +
datasetName;

            try
            {
                bulkcopy.WriteToServer(dt_Bulk); // SourceTable would come
from your DataSet
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }

            // update centerPoints massive to database
            GlobalPoint.ClearCentersFromDatabase(datasetName, "FCM");
            GlobalPoint.WriteCentersToDatabase(datasetName, NewCenterPoints,
"FCM");

            string queryMem = "EXEC C_MEANS_MEMBERSHIP_CLUSTERING_PROC
'C_MEANS_MEMBERSHIP_" + datasetName + "'";
            SqlCommand cmdMem = new SqlCommand(queryMem, sconn);
            cmdMem.ExecuteNonQuery();

            // write execution times to database
```

```
                string queryExecuteTime = "INSERT INTO CLUSTERING_EXECUTION_TIME
VALUES ( 'K_MEANS_" + datasetName + "', '" + k.ToString() + "', " +
                                          " 'FCM', GETDATE(), '" + execWatch.Elapsed +
"', 'SYSTEM' )";
                SqlCommand cmdExecuteTime = new SqlCommand(queryExecuteTime,
sconn);
                cmdExecuteTime.ExecuteNonQuery();

                if (doDatabaseWorks)
                {
                    System.Diagnostics.Stopwatch databaseQueriesTotalWatch = new
System.Diagnostics.Stopwatch();
                    databaseQueriesTotalWatch.Start();

                    colorThePointsFuzzy(datasetName, false, NewCenterPoints,
Convert.ToInt32(alphaPowerText.Text));

                    // writing aimvalues to database
                    if (getAimToData)
                    {
                        //double generalAimFunc = 0;
                        double generalFuzzyAimFunc = 0;
                        //generalAimFunc = GlobalPoint.AimFunction(DataPoints,
NewCenterPoints, dimentionCount);
                        generalFuzzyAimFunc =
GlobalPoint.AimFunctionFuzzy(DataPoints, NewCenterPoints, dimentionCount,
uMembershipMassive, m_fuzzifier);
                        // GlobalPoint.AimTableClear(datasetName, "FCM", k_count);
                        //GlobalPoint.AimValueToData(datasetName, "FCM", true, 1,
generalAimFunc);
                        GlobalPoint.AimValueToData(datasetName, "FCM", true, 1,
k_count, generalFuzzyAimFunc);
                    }

                    if (distCalcCount)
                    {
                        // GlobalPoint.ClearDistCountFromDatabase(datasetName,
"FCM");
                        GlobalPoint.WriteDistCountToDatabase(datasetName, "FCM",
k_count, dimentionCount, distCount);
                    }

                    // write the count of points for every cluster to database
                    if (writePointPerCluster)
                    {
                        // ClearPointsPerClusterTable(datasetName, "FCM", k_count,
true);
                        AddPointsPerClusterTable(datasetName, "FCM", k_count,
true);
                    }

                    if (writeLoopCountToData)
                    {
                        // GlobalPoint.LoopCountClear(datasetName, "FCM",
k_count);
                        GlobalPoint.LoopCountToData(datasetName, "FCM",
!fromFGMCM, loop, diffBefore - diff, k_count);
                    }

                    databaseQueriesTotalWatch.Stop();

                    string queryExecuteTimeDB = "INSERT INTO
CLUSTERING_EXECUTION_TIME  VALUES ( 'K_MEANS_" + datasetName + "', '" +
k.ToString() + "', " +
                                              " 'FCM', GETDATE(), '" +
databaseQueriesTotalWatch.Elapsed + "', 'DATABASE' )";
                    SqlCommand cmdExecuteTimeDB = new
SqlCommand(queryExecuteTimeDB, sconn);
                    cmdExecuteTimeDB.ExecuteNonQuery();
                }


                execTotal.Stop();

                string queryExecuteTotal = "INSERT INTO CLUSTERING_EXECUTION_TIME
VALUES ( 'K_MEANS_" + datasetName + "', '" + k.ToString() + "', " +
```

```
                                    " 'FCM', GETDATE(), '" + execTotal.Elapsed +
"', 'TOTAL' )";
                SqlCommand cmdExecuteTotal = new SqlCommand(queryExecuteTotal,
sconn);
                cmdExecuteTotal.ExecuteNonQuery();

                execTime.Text = execTotal.Elapsed.ToString();

            }
            catch (Exception ex) { MessageBox.Show("Error on Fuzzy C Means
clustering : " + ex.Message); }

        }
```

c) *PointDouble* sınıfı ve yöntemleri:

```
        class PointDouble
        {
            private double PointDoubleX;
            private double PointDoubleY;
            private double PointDoubleZ;
               ...
            private double PointDoubleX40;

            public const int NOISE = -1;
            public const int UNCLASSIFIED = 0;
            public int ClusterId;

            public PointDouble()
            {
                this.X = 0.0;
                this.Y = 0.0;
                this.Z = 0.0;
                      ...
                this.X40 = 0.0;
            }

            public PointDouble(double X, double Y, double Z,
                               ... double X40)
            {
                this.X = X;
                this.Y = Y;
                this.Z = Z;
                ...
                this.X40 = X40;
            }

            public double X
            {
                get { return PointDoubleX; }
                set { PointDoubleX = value; }
            }

            ...

            public double X40
            {
                get { return PointDoubleX40; }
                set { PointDoubleX40 = value; }
            }

            public void CopyPointFrom(PointDouble p)
            {
                this.X = p.X;
                this.Y = p.Y;
                if (dimentionCount > 2) { this.Z = p.Z; }
                   ...
                if (dimentionCount > 39) { this.X40 = p.X40; }
            }
```

```
        }
```

d) *GlobalPoint* sınıfı ve yöntemleri:

```
static class GlobalPoint
{


static public PointDouble Center(PointDouble[] pM)
            {
                // calculate the center of given points
                int k = pM.Length;
                if (k == 0)
                {
                    return new PointDouble(0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                           0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                           0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                           0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                }
                double centerX = 0; double centerY = 0; double centerZ = 0; ...
double centerX40 = 0;
                for (int i = 0; i < k; i++)
                {
                    centerX = centerX + pM[i].X;
                    centerY = centerY + pM[i].Y;
                    centerZ = centerZ + pM[i].Z;
                      ...
                    centerX40 = centerX40 + pM[i].X40;
                }
                centerX = centerX / k;
                centerY = centerY / k;
                centerZ = centerZ / k;
                      ...
                centerX40 = centerX40 / k;
                return new PointDouble(centerX, centerY, centerZ, ... centerX40);
            }



static public double AimFunction(PointDouble[] DataPoints, PointDouble[]
CenterPoints, int dimentionCount)
            {
                // calculate the aim function for given data and center points
                int DaCount = DataPoints.Length;
                int CeCount = CenterPoints.Length;
                double[] distances = new double[DaCount];
                double currentDistance;
                double funcValue = 0;

                for (int i = 0; i < DaCount; i++)
                {
                    currentDistance = 0;
                    for (int j = 0; j < CeCount; j++)
                    {
                        currentDistance =
GlobalPoint.EuclideanDistance(DataPoints[i], CenterPoints[j], dimentionCount);
                        if (j == 0 || currentDistance < distances[i]) {
                            distances[i] = currentDistance;
                        }
                    }
                    //funcValue = funcValue + Math.Pow(distances[i],2);
                    funcValue = funcValue + distances[i];
                }
                return funcValue;
              }
```

```
static public double AimFunctionFuzzy(PointDouble[] DataPoints, PointDouble[]
CenterPoints, int dimentionCount, double[,] uMembershipMassive, double
m_fuzzifier)
            {
                // calculate the fuzzy aim function for given data and center
points
                int DaCount = DataPoints.Length;
                int CeCount = CenterPoints.Length;

                double currentDistance;
                double funcValue = 0;

                for (int i = 0; i < DaCount; i++)
                {
                    currentDistance = 0;
                    for (int j = 0; j < CeCount; j++)
                    {
                        currentDistance =
GlobalPoint.EuclideanDistance(DataPoints[i], CenterPoints[j], dimentionCount);
                        funcValue = funcValue + Math.Pow(uMembershipMassive[i,j],
m_fuzzifier) * Math.Pow(currentDistance, 2);
                    }

                }
                return funcValue;
            }
```

```
static public double EuclideanDistance(PointDouble p1, PointDouble p2, int
dimcount)
            {
                double dist = 0;
                dist = dist + Math.Pow(p1.X - p2.X, 2);
                dist = dist + Math.Pow(p1.Y - p2.Y, 2);
                if (dimcount > 2) { dist = dist + Math.Pow(p1.Z - p2.Z, 2); }
                if (dimcount > 3) { dist = dist + Math.Pow(p1.A - p2.A, 2); }
                    ...
                if (dimcount > 39) { dist = dist = Math.Pow(p1.X40 - p2.X40, 2); }

                return Math.Sqrt(dist);
            }
```

```
static public PointDouble[] GetNewCentersFromSetWithOldCenters(PointDouble[]
DataPoints, PointDouble[] OldCenters, int dimentionCount)
            {
                // calculating new center points from given data set and given old
center points

                PointDouble[] NewCenters = new PointDouble[OldCenters.Length];
                int[] WhichCluster = new int[DataPoints.Length];
                double dist, minDist  = 0;
                int index = 0;

                // calculating toe which cluster belong the data and write all to
WhichCluster massive
                for (int i = 0; i < DataPoints.Length; i++)
                {
                    for (int j = 0; j < OldCenters.Length; j++)
                    {
                        dist = GlobalPoint.EuclideanDistance(DataPoints[i],
OldCenters[j], dimentionCount);
                        if (dist < minDist || j == 0)
                        {
                            minDist = dist;
                            WhichCluster[i] = j + 1;
                        }
```

```
                    }
                    dist = 0;
                    minDist = 0;
                }

                // for every old center point loop:
                for (int j = 0; j < OldCenters.Length; j++)
                {
                    // calculate how many points in current cluster to create
TempCluster massive
                    for (int i = 0; i < DataPoints.Length; i++)
                    {
                        if (WhichCluster[i] == j + 1)
                        {
                            index++;
                        }
                    }

                    PointDouble[] TempCluster = new PointDouble[index];
                    index = 0;

                    // insert into TempCluster the points of current cluster
                    int nextNo = 0;
                    for (int i = 0; i < DataPoints.Length; i++)
                    {
                        if (WhichCluster[i] == j + 1)
                        {
                            TempCluster[nextNo] = DataPoints[i];
                            nextNo++;
                        }
                    }
                    NewCenters[j] = GlobalPoint.Center(TempCluster);
                }


                return NewCenters;
            }




static public double[,] GetMembershipValues(PointDouble[] DataPoints,
PointDouble[] centerPoints, int dimentionCount, double m_fuzzifier)
            {

                double[,] distances = new double[DataPoints.Length,
centerPoints.Length];
                double[,] uMembershipMassive = new double[DataPoints.Length,
centerPoints.Length];
                double[] totalDist = new double[DataPoints.Length];
                //double[] sumSquareFuzzifier = new double[centerPoints.Length];

                // get the massive of distances
                for (int i = 0; i < DataPoints.Length; i++)
                {
                    for (int j = 0; j < centerPoints.Length; j++)
                    {
                        distances[i, j] =
GlobalPoint.EuclideanDistance(DataPoints[i], centerPoints[j], dimentionCount);
                    }
                }

                // get the massive of total distances
                for (int i = 0; i < DataPoints.Length; i++)
                {
                    for (int j = 0; j < centerPoints.Length; j++)
                    {
                        //totalDist[i] = totalDist[i] + 1 / Math.Pow(distances[i,
j], (2 / (m_fuzzifier - 1)));
                        if (double.IsInfinity(1 / Math.Pow(distances[i, j], (2 /
(m_fuzzifier - 1)))))
                        {
                            totalDist[i] = totalDist[i] + 0;
                        }
                        else
```

```
                        {
                            totalDist[i] = totalDist[i] + 1 /
Math.Pow(distances[i, j], (2 / (m_fuzzifier - 1)));
                        }

                    }
                }

                // calculate the U[i,j] memberships massive of data points
                for (int i = 0; i < DataPoints.Length; i++)
                {
                    for (int j = 0; j < centerPoints.Length; j++)
                    {
                        if (double.IsInfinity(1 / Math.Pow(distances[i, j], (2 /
(m_fuzzifier - 1))) / totalDist[i]))
                        {
                            uMembershipMassive[i, j] = 0;
                        }
                        else
                        {
                            uMembershipMassive[i, j] = 1 / Math.Pow(distances[i,
j], (2 / (m_fuzzifier - 1))) / totalDist[i];
                        }
                    }
                }

                return uMembershipMassive;

            }

}
```