
GAZİ ÜNİVERSİTESİ
Gazi
1926

# K-BOYUTLU AĞAÇ, UYARLANABİLİR YARIÇAP VE ÖZNİTELİK SEÇME (KD-ARFS STREAM) TABANLI GERÇEK ZAMANLI AKAN VERİ KÜMELEME

**Ali ŞENOL**

**DOKTORA TEZİ**
**BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI**

**GAZİ ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**

**MAYIS 2019**

Ali ŞENOL tarafından hazırlanan "K-BOYUTLU AĞAÇ, UYARLANABİLİR YARIÇAP VE ÖZNİTELİK SEÇME (KD-ARFS STREAM) TABANLI GERÇEK ZAMANLI AKAN VERİ KÜMELEME" adlı tez çalışması aşağıdaki jüri tarafından OY BİRLİĞİ ile Gazi Üniversitesi BİLGİSAYAR MÜHENDİSLİĞİ Ana Bilim Dalında Doktora Tezi olarak kabul edilmiştir.

**Danışman:** Doç. Dr. Hacer KARACAN

Bilgisayar Mühendisliği Ana Bilim Dalı, Gazi Üniversitesi ……..……..

Bu tezin, kapsam ve kalite olarak Doktora Tezi olduğunu onaylıyorum.

**Başkan:** Prof. Dr. Erdoğan DOĞDU

Bilgisayar Mühendisliği Ana Bilim Dalı, Çankaya Üniversitesi ...……

Bu tezin, kapsam ve kalite olarak Doktora Tezi olduğunu onaylıyorum.

**Üye:** Prof. Dr. M. Ali AKCAYOL

Bilgisayar Mühendisliği Ana Bilim Dalı, Gazi Üniversitesi ...……

Bu tezin, kapsam ve kalite olarak Doktora Tezi olduğunu onaylıyorum.

**Üye:** Doç. Dr. Osman ABUL

Bilgisayar Mühendisliği Ana Bilim Dalı, TOBB ETÜ .....……

Bu tezin, kapsam ve kalite olarak Doktora Tezi olduğunu onaylıyorum.

**Üye:** Dr. Öğr. Üyesi İ. Alper DOĞRU

Bilgisayar Mühendisliği Ana Bilim Dalı, Gazi Üniversitesi ...……

Bu tezin, kapsam ve kalite olarak Doktora Tezi olduğunu onaylıyorum.

Tez Savunma Tarihi: 30/05/2019

Jüri tarafından kabul edilen bu çalışmanın Doktora Tezi olması için gerekli şartları yerine getirdiğini onaylıyorum

……..…….

Prof. Dr. Sena YAŞYERLİ

Fen Bilimleri Enstitüsü Müdürü

## ETİK BEYAN

Gazi Üniversitesi Fen Bilimleri Enstitüsü Tez Yazım Kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- Tez içinde sunduğum verileri, bilgileri ve dokümanları akademik ve etik kurallar çerçevesinde elde ettiğimi,
- Tüm bilgi, belge, değerlendirme ve sonuçları bilimsel etik ve ahlak kurallarına uygun olarak sunduğumu,
- Tez çalışmasında yararlandığım eserlerin tümüne uygun atıfta bulunarak kaynak gösterdiğimi,
- Kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- Bu tezde sunduğum çalışmanın özgün olduğunu,

bildirir, aksi bir durumda aleyhime doğabilecek tüm hak kayıplarını kabullendiğimi beyan ederim.

……..
Ali ŞENOL
30/05/2019

# K-BOYUTLU AĞAÇ, UYARLANABİLİR YARIÇAP VE ÖZNİTELİK SEÇME (KD-ARFS STREAM) TABANLI GERÇEK ZAMANLI AKAN VERİ KÜMELEME

(Doktora Tezi)

Ali ŞENOL

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Mayıs 2019

## ÖZET

Klasik kümeleme yaklaşımlarında veri statiktir. Veri, bir yere kaydedilerek tekrar tekrar işlenebilmektedir. Oysa günümüz teknolojisinde, verinin çok hızlı olduğu dünyada, artık veriyi akarken kümeleyecek, kullanıcıya istediği zaman sonuç verebilecek uygulamalara ihtiyaç vardır. Bu anlamda ihtiyacı karşılayan akan veri kümeleme yaklaşımlarına olan talep gün geçtikçe artmaktadır. Çünkü akan veri kümeleme yaklaşımları veriyi bir defa okumalı, hızlı ve kendisini yeni gelen veriye uyarlama özelliğine sahiptir. Yani bir yandan veri akarken, bir yandan kullanıcıya sonuç üretilebilmektedir. Bu tez çalışmasında akan veri üzerinde gerçek zamanlı kümeleme yapan KD-ARFS Stream algoritması önerilmiştir. Önerdiğimiz yaklaşım gücünü çok boyutluluğu destekleyen k-boyutlu ağaç (kd-tree), uyarlanabilir yarıçap ve standart sapma tabanlı öznitelik seçme özelliklerinden almaktadır. KD-ARFS Stream algoritmasının başarısını ölçmek için SE-Stream, CEDAS, pcStream ve DPStream algoritmaları ile toplam harcanan süre ve kümeleme başarısı açılarından karşılaştırılmıştır. Deneysel çalışmalar KD-ARFS Stream algoritmasının daha iyi kümeleme başarısını makul bir sürede verdiğini göstermiştir.

| | | |
|---|---|---|
| Bilim Kodu | : | 92429 |
| Anahtar Kelimeler | : | Akan veri kümeleme, k-boyutlu ağaç, uyarlanabilir yarıçap, tamamen çevrimiçi, evrimsel kümeleme |
| Sayfa Adedi | : | 162 |
| Danışman | : | Doç. Dr. Hacer KARACAN |

# KD-TREE, ADAPTIVE RADIUS AND FEATURE SELECTION (KD-ARFS STREAM) BASED REAL TIME DATA STREAM CLUSTERING

(Ph. D. Thesis)

Ali ŞENOL

GAZİ UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

May 2019

## ABSTRACT

In classical data clustering approaches, data is static. It is possible to store the data and process it again and again. However, in the today's technology, in which the data is very fast, it is needed to process the data while it is being streamed and results should be shown to the user whenever the user want. In this sense, demand for data stream clustering approaches, which meet the needs, is increasing day by day. Because data stream clustering approaches are fast, have once read ability and can adapt themselves to new data. In other words, while data is streaming on the other hand, results can be shown to the user on the one hand. In this thesis, KD-ARFS Stream algorithm, which clusters streaming data in real-time is proposed. The proposed approach takes its power from kd-tree, which supports multidimensionality, standard deviation based feature selection and adaptive radius. In order to present the success of KD-ARFS Stream algorithm, it is compared with SE-Stream, pcStream, CEDAS and DPStream algorithms in aspects of consumed time and clustering quality. Experimental results have shown that the KD-ARFS Stream algorithm provides better clustering quality in a reasonable time.

| | | |
|---|---|---|
| Science Code | : | 92429 |
| Key Words | : | Data stream clustering, kd-tree, adaptive radius, fully online, evolutionary clustering |
| Page Number | : | 162 |
| Supervisor | : | Assoc. Prof. Dr. Hacer KARACAN |

# TEŞEKKÜR

Bu tezin hazırlanma süreci boyunca kıymetli bilgi, birikim ve tecrübeleri ile bana yol gösterici ve destek olan değerli danışman hocam Doç. Dr. Hacer KARACAN'a; akademik katkı ve desteklerini benden esirgemeyen tez izleme komitesi üyesi kıymetli hocalarım Prof. Dr. M. Ali AKCAYOL ve Prof. Dr. Erdoğan DOĞDU'ya sonsuz teşekkür ve saygılarımı sunarım.

Doktora tez sürecinde desteğini, sevgisini ve özverisini benden hiçbir zaman esirgemeyen eşim Büşra ŞENOL'a; oğullarım Ozan Poyraz ŞENOL ve Barkın Uraz ŞENOL'a ithaf ediyorum.

# İÇİNDEKİLER

**Sayfa**

**Sayfa**

**Sayfa**

# ÇİZELGELERİN LİSTESİ

**Çizelge** **Sayfa**

# ŞEKİLLERİN LİSTESİ

**Şekil** **Sayfa**

**Şekil** **Sayfa**

**Şekil** **Sayfa**

**Şekil** **Sayfa**

**Şekil** **Sayfa**

# SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış simgeler ve kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

| Simgeler | Açıklamalar |
|---|---|
| **Data** | Veri |
| **C** | Küme tablosu |
| **bufferedData** | Tamponlanmış veri |
| **bufferedDataSize** | Tamponlanmış veri sayısı |
| **AllData$C_i$** | $C_i$ kümesinin bütün verileri |
| **deletedData** | Silinen veriler |
| **t** | Veriler için ömrünü tamamlama eşik değeri |
| **TN** | Veri miktarı çok büyük olduğunda tampona alınan veri miktarı |
| **r** | Küme yarıçapı |
| **r_threshold** | Kümenin yarıçapını arttırma eşik değeri |
| **r_max** | Kümenin ulaşabileceği maksimum değer |
| **d** | Nitelik sayısı |
| **newD** | Seçilen nitelikler |
| **N** | Yeni küme oluşturmak için gerekli minimum örnek sayısı |
| **tree** | K-boyutlu ağaç |
| **$C_{temp}$** | Aday küme |
| **dataSize** | Veri sayısı |
| **dist** | İki nokta arasındaki mesafe (Veri-küme veya küme-küme mesafesi) |
| **numOfDataInArea1** | Belirlenmiş alandaki (Area1) veri sayısı |
| **numOfDataInArea2** | Belirlenmiş alandaki (Area2) veri sayısı |
| **$C_{Radius}$** | Kümenin yarıçapı, burada kümenin kabuk yarıçapı olarak adlandırılmaktadır. |
| **$C_{Std}$** | Kümenin standart sapması, burada kümenin çekirdek yarıçapı olarak adlandırılmaktadır. |

| Simgeler | Açıklamalar |
|---|---|
| **StdOfFetures** | Niteliklerin standart sapması |
| **ClusterNo** | Küme indeksi |
| **bufferedData$_i$clusterNo** | bufferedData$_i$ verisinin ait olduğu küme |
| **coeffThreshold** | Öznitelik seçme eşik değeri |
| **CenterCoordinate** | Küme merkezi koordinatı |
| **$C_i$Active** | i kümesinin aktif olma durumu |
| **v** | Kök düğüm |
| **depth** | Derinlik |
| **$v_{left}$** | Sol alt ağaç |
| **$v_{right}$** | Sağ alt ağaç |
| **mean** | Aritmetik ortalama |
| **min** | Minimum değer |
| **max** | Maksimum değer |
| **sqrt** | Karekök |
| **σ** | Standart sapma |
| **μ** | Verilerin ağırlık merkezi (ortalaması) |

| Kısaltmalar | Açıklamalar |
|---|---|
| **K-b ağacı** | K-boyutlu ağaç |
| **HDFS** | Hadoop distributed file system |
| **EM** | Expectation maximization |
| **FCS** | Fading clustering structure |
| **MOA** | Massive çevrimiçi analysis |
| **DFT** | Discrete fourier transform |
| **FCH** | Fading Cluster Structure |
| **MC** | Micro-Clusters |

# 1. GİRİŞ

Teknolojinin hızlı bir şekilde gelişmesine paralel olarak bilgisayar, akıllı telefon ve sensör gibi veri üreten cihazların kullanımı yaygınlaşmıştır. Bu tür cihazlar kullanılarak "Büyük Veri" olarak tanımlanan inanılmaz bir veri miktarı üretilmektedir. Bu verilerin çok büyük bir kısmı hiçbir şekilde analiz edilememektedir. Çünkü internet ortamındaki verilerin çok büyük bir kısmı belli bir yapıda değildir. Bu nedenle analizi oldukça zordur. Bu açıdan bakıldığı zaman büyük veri yığınlarını analiz edecek yöntemlere olan ilgi günden güne artmaktadır (Holzinger, Stocker, Ofner, Prohaska, Brabenetz ve Hofmann-Wellenhof, 2013).

Günümüzde üretilen veriler çok hızlı bir şekilde arttığından klasik veri analiz yöntemleri pek çok açıdan yetersiz kalmaktadır. Bunların başında verileri bir yere kaydetme gelmektedir. Çünkü çok büyük veri miktarını kayıt altına almak için gereken kaynak ihtiyacı da çok büyük ve maliyetlidir (Ankleshwaria ve Dhobi, 2014). Bir diğer yetersizlik nedeni çoğu uygulama için gerçek zamanlı sonuç üretmeye olan ihtiyaçtır. Örneğin banka gibi finansal kuruluşlar için analizlerin anlık yapılabilmesine ihtiyaç vardır. Çünkü dolandırıcılık gibi durumlar için 1 dk. bile oldukça geç olabilir. Bu ve buna benzer nedenlerden dolayı gerçek zamanlı analizler üretebilecek uygulamalara olan ihtiyaç günden güne artmaktadır (Şenol ve Karacan, 2018).

Veri madenciliğinin sınıflandırma veya kümeleme yaklaşımları büyük verilerden değerli bilgileri elde etmek amacıyla çok sık bir şekilde kullanılmaktadır. Klasik anlamda bu yaklaşımlar statik veri üzerinde işlem yaparak sonuç üretmektedir. Ancak bu yaklaşımları temel alan ve gerçek zamanlı veriler üzerinde sonuç üretmeye yönelik çeşitli yaklaşımlar da üretilmiş ve üretilmeye devam etmektedir. Veri setleri üzerinden anlamlı bilgiye ulaşmak adına çok sayıda sınıflandırma ve kümeleme yaklaşımı geliştirilmiştir. Sınıflandırma algoritmaları olarak Naive Bayes (Mitchell, 1997), karar ağaçları (decision trees) (Quinlan, 1986) gibi yöntemler kullanılmaktadır. Bunların yanında yapay sinir ağları ve genetik algoritma gibi yaklaşımlar veya bunların hibrit yapıları kullanılmaktadır. Bu yöntemler ya verileri sabit alarak sonuç üretmekte ya da rastgele bir kabul yaparak kümeleme yapmaktadır. Bu açıdan bakıldığı zaman yeni veri geldikçe küme sayısında değişiklik yapan yeni yaklaşımlara ihtiyaç vardır (Şenol ve Karacan, 2018).

Herhangi bir kümeleme algoritması var olan bir veri seti üzerinde işlem yaparak bu veri setini kümelere ayırmaktadır. Kaç küme olacağını ya kullanıcıdan girdi olarak alır ya da rastgele belirleyerek işlem yapmakta ve sonuçta bu girdilere göre her zaman doğruluğu kesin olmayan bir kümeleme yapmaktadır (Kaur ve Garg, 2014). Klasik kümeleme algoritmaları yeni bir veri geldiğinde, dolayısıyla veri seti değiştiğinde kümeleme işlemini baştan yapar. Oysaki veri seti dinamik bir şekilde sürekli değişiyor da olabilir. Bu noktada, mevcut yaklaşımların yetersiz kaldığı görülmektedir. Çünkü bilgisayar ortamına taşınmış inanılmaz boyutta veri bulunmakta ve bu miktar günden güne artmaktadır. Yani herhangi bir alanda elde edilmiş olan veri veya veri seti sürekli değişmektedir. Bu nedenle bu verileri dinamik bir şekilde, yeni veri geldikçe kümelemeyi güncelleyecek yaklaşımlara ihtiyaç duyulmaktadır (Şenol ve Karacan, 2018).

K-means (Lloyd, 1982), En Yakın K Komşu (K nearest neighbour) (Cover ve Hart, 2006), DBSCAN (Ester, Kriegel, Sander ve Xu, 1996) kümeleme yaklaşımları da ya kullanıcının küme sayısını ön tanımlı olarak girmesini beklemekte ya da bunu rastgele belirleyerek veri setini kümelere ayırmaktadır. Bu ön tanımlı koşullar sonucu doğrudan etkilediğinden zaman zaman beklenen başarı elde edilememektedir. Bu tür kümeleme yaklaşımları da yeni veri geldikçe küme sayısında bir değişikliğe gitmemektedir.

Klasik veri analiz yöntemleri sabit veri üzerinde işlem yapmaktadır. Ancak artan veri miktarı nedeniyle kaynak kısıtlılığı, verinin hızlı bir şekilde değişmesi ve sabit bir yapıda olmaması nedeniyle gerçek zamanlı olarak işlenmesi gerekmektedir. Akan veri işleme (stream processing) olarak adlandırılan bu yaklaşımlar karar destek sistemleri için oldukça önemlidir. Çünkü çoğu uygulama için veri kaydetme imkânı yoktur veya maliyetlidir. Bu nedenle gerçek zamanlı çözüm üreten yaklaşımlar çok önemlidir.

Şekil 1.1'de de görüldüğü gibi sonu belli olmayan, hızlı bir şekilde akan veri objelerine akan veri denilmektedir (Aggarwal, 2007). Data stream akan veri, data stream mining akan veri madenciliği ve data stream clustering akan veriyi kümeleme anlamlarına gelmektedir (Ankleshwaria ve Dhobi, 2014). x, n tane nitelikten oluşan bir veri olmak üzere, S akan veri dizisi $S = x_1, x_2, x_3, \dots, \infty$ olarak ifade edilebilir veya kısaca $S = \langle x_i \rangle_{j=1}^{n}$ olarak da ifade edilebilir (J. A. Silva, Faria, Barros, Hruschka, Carvalho ve Gama, 2013). Akan veri kümeleme finans, network izleme, telekomünikasyon, veri yönetim, web, sensör ağ, meteorolojik, bilim ve mühendislik gibi pek çok alanda kullanılmaktadır (Ankleshwaria ve

Dhobi, 2014; Ikonomovska, Loskovska ve Gjorgjevik, 2007). Adı geçen uygulamalarda yüksek hızda anlık veri akmaktadır. Geliştirilecek uygulamaların tek yönlü tarama yapan, on-line, çok seviyeli ve çok yönlü olması gerekir (Ankleshwaria ve Dhobi, 2014).


Şekil 1.1. Akan veri örneği

Akan veride verinin sonu belli değil ve çoğunlukla da boyutu sonsuzdur. Bu nedenle veriyi bir yerde biriktirip işleme imkânı yoktur, yani sınırsız depolama ihtiyacı doğmaktadır. Ayrıca veri yüksek hızda akmaktadır. Bu veriyi kümeleyecek yaklaşımın da hızlı bir şekilde çalışması gerekmektedir. Bunun yanında akan bu verinin genellikle belli bir sınırı yoktur. Başı ve sonu önceden bilinemez. Dolayısıyla verinin geneli hakkında bilgi sahibi olma imkânı yoktur. Kısaca akan veri, sayı olarak büyük, sonsuz, devamlı ve ardışıktır (Aggarwal, 2007; Bifet ve Kirkby, 2009).

Son yıllarda akan veri üzerinde kümeleme yapmaya yönelik çeşitli çalışmalar yapılmış ve bu konu gün geçtikçe ilgi çekmektedir. Akan veri, miktar olarak çok büyük, sonsuz ve devamlıdır (C. C. Aggarwal, 2007; Bifet ve Kirkby, 2009). Akan veriyi kümeleme olarak çevirebileceğimiz “data stream clustering” dinamik bir şekilde değişen veriyi kümelemeye yönelik yapılan çalışmalardır. Bu tür çalışmalar gerçek zamanlı ve veri akarken hızlı bir şekilde sonuç üreten çalışmalardır. Günümüzün popüler konulardan birisi olan akan veri kümeleme yaklaşımları pek çok alanda kullanılmaktadır.

Bu tez çalışmasında yoğunluk tabanlı bir akan veri kümeleme yaklaşımı olan KD-ARFS Stream algoritmasını öneriyoruz. Önerdiğimiz yaklaşım çok boyutluluğu desteklemesi amacıyla k-boyutlu ağaç tabanlı bir yapı kullanmaktadır. Akan verinin değişken yapısına uyum sağlaması amacıyla kullanılan yarıçap uyarlanabilir bir yapıdadır. Oluşturulan kümeler zaman içinde aktif veya pasif olabilmektedir. Zaman tabanlı özetleme ile kayan pencere (sliding window) tabanlı özetleme yaklaşımlarını birleştiren bir hibrit veri özetleme yaklaşımı kullanmaktadır. Ayrıca çalışma zamanını iyileştirmek amacıyla standart sapma tabanlı bir öznitelik seçme yaklaşımı kullanmaktadır. Bütün bu özellikleri ışığında önerdiğimiz yaklaşımın literatüre katkılarını kısaca şu şekilde sıralayabiliriz:

- Tamamen çevrimiçi çalışır, çevrimdışı bir bileşene ihtiyaç duymaz;

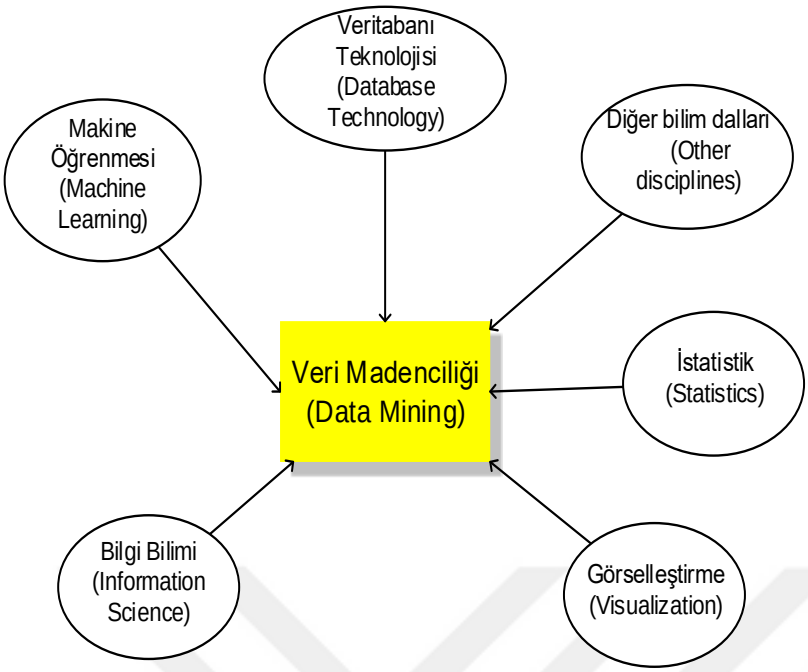
- Kümeleri aktif/pasif yaparak akan verinin değişken yapısına tam uyum sağlar;
- Zaman ve miktar tabanlı özetlemeyi birleştirerek ikisinin de avantajlarını kullanmaktadır;
- Çok boyutlu verileri öznitelik seçme yaklaşımı ile hızlı bir şekilde işleyebilmektedir;
- Yüksek kümeleme başarısına sahiptir;
- Sapan verilere karşı dirençli bir yapısı vardır;

Önerdiğimiz KD-ARFS Stream yaklaşımı SE-Stream, pcStream, DPStream ve CEDAS algoritmaları ile karşılaştırılmıştır. Deneysel çalışmalar sonucunda kümeleme başarısı açısından daha başarılı sonuçlar verdiği gözlemlenmiştir. Bununla beraber çalışma zamanı açısından da karşılaştırılan algoritmaların genelinden daha hızlı sonuç verdiği tespit edilmiştir.

Veri madenciliği ve makine öğrenmesi

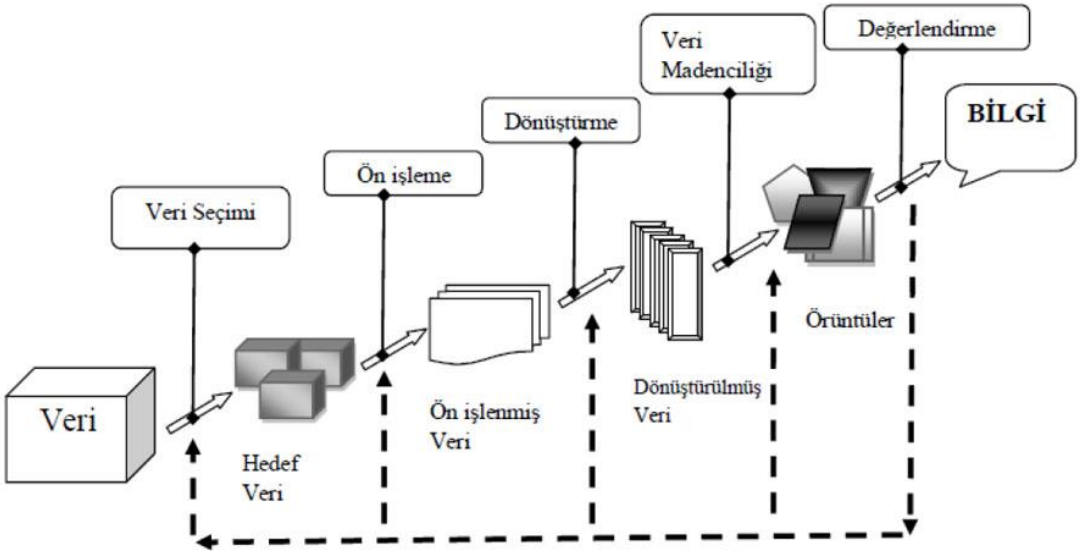
Veri madenciliği karmaşık ve büyük boyutlu verilerden anlamlı bilginin çıkarılması üzerine gelişmiş bir disiplindir. Yukarıda da üzerinde durduğumuz gibi internet ortamındaki verilerin çok büyük bir bölümü yapısız veya yarı yapılı durumdadır. Bu nedenle bu verileri kolay bir şekilde işlemek ve değerli bilgiye ulaşmak kolay bir iş değildir. Veri madenciliğinin temel amaçlarından biri bu tür problemlerin üstesinden gelerek anlamlı bilgiyi kullanıcıya sunmaktır (Hand, Smyth ve Mannila, 2001).

Veri madenciliği Şekil 1.2'de de görüldüğü gibi veritabanı teknoloji, istatistik, makine öğrenmesi, bilgi bilimi, veri görselleştirme gibi pek çok alanda kullanılmaktadır (Jiawei, Kamber ve Pei, 2011). Çünkü veriden değerli bilgiye ulaşmak her alanda önemli kazançlar sunmaktadır. Örneğin bir marketin müşterilerinin alışverişlerini veri madenciliği yöntemleri kullanarak analiz etmesi ve buna göre kampanya veya raf düzenlemesi yapması satış miktarında önemli artışlar sağlamaktadır. Bunun yanında bir banka veri madenciliği yöntemlerini kullanarak hangi müşterilerine kredi vermesinin daha doğru olacağına karar verebilmektedir. Bu tür örnekleri çoğaltmak mümkündür.




Şekil 1.2. Veri madenciliğinin diğer disiplinlerle bağlantısı (Jiawei ve diğerleri, 2011)

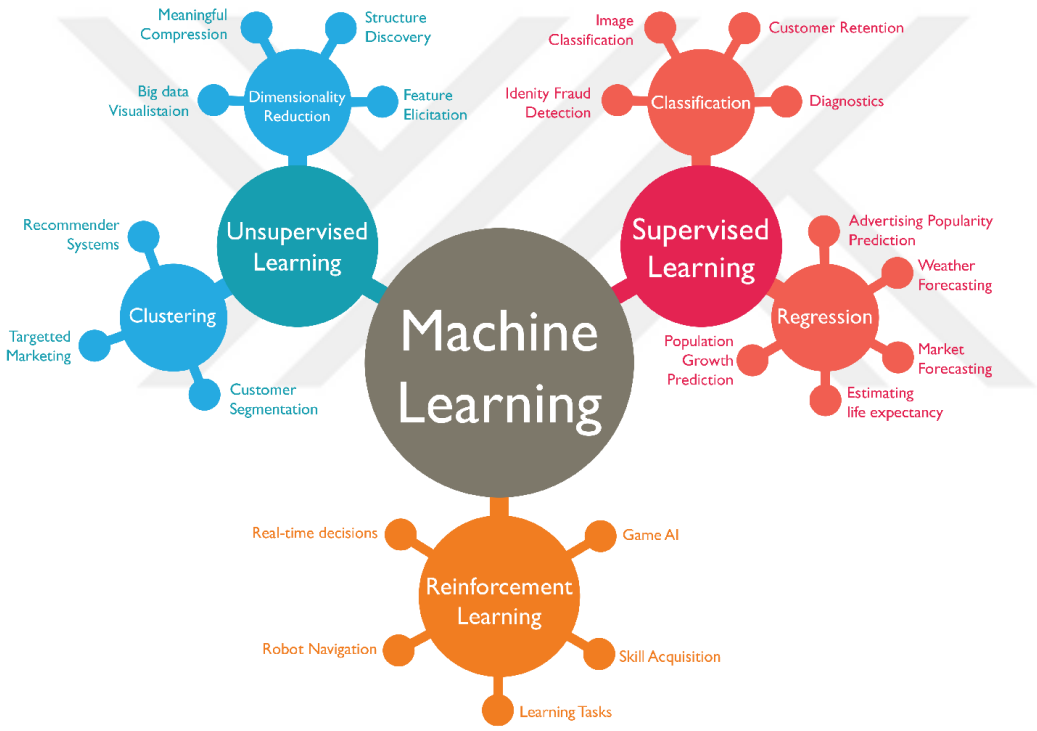
Veri madenciliği yapısal olarak belli bir yapısı olmayan veya tamamen yapısal olmayan veriler üzerinde çalışan bir disiplindir (Mehra, Kumar Gupta ve Bhatt, 2014). Bu nedenle veri üzerinde veri madenciliği yöntemlerini uygulamadan önce Şekil 1.3'te de görüldüğü gibi veri temizleme, bütünleştirme, dönüştürme gibi veri önişleme işlemleri yapılmaktadır. Bu tür işlemleri nasıl ve ne şekilde yapılacağıyla ilgili pek çok çalışma yapılmış ve yapılmaya devam etmektedir.




Şekil 1.3. Veri madenciliği süreci (Jiawei ve diğerleri, 2011)

Veri madenciliğine olan ilgi, talep ve ihtiyaç bu alanda yapılan çalışmaların farklılaşmasına neden olmuştur. Bu çalışmaların başında makine öğrenmesi (machine learning) gelmektedir. Makine öğrenmesi yöntemleri veri işleme şeklini bir adım öteye taşıyarak veriyi işlerken öğrenmeyi de işin içine katarak çözümü zor olan problemleri çözmektedir ve Şekil 1.4'te de görüldüğü gibi pek çok bileşenden oluşmaktadır. Makine öğrenmesi yöntemlerini temel anlamda üç gruba ayırabiliriz (Krzyk, 2019). Bunlar:

- Denetimli öğrenme (Supervised learning)
- Denetimsiz öğrenme (Unsupervised learning)
- Takviyeli öğrenme (Reinforcement learning)




Şekil 1.4. Makine öğrenmesi bileşenleri (Wahid, 2019)

*Denetimli öğrenme*

Bu tür algoritmalar belirli bir miktar veriyi öğrenmek için kullandıktan sonra, bu öğrendikleri ile yeni veri hakkında tahminde bulunur. Eğitim için kullanılan verilerin gerçek etiketleri bulunmaktadır. Yani aslında her verinin hangi sınıfa ait olduğu bilinmektedir. Denetimli öğrenmede en önemli bileşen eğitim verisidir. Özensiz hazırlanmış eğitim verisi başarıyı düşürür. Denetimli öğrenme yöntemleri iki gruba ayrılır (Krzyk, 2019):

- Sınıflandırma Yöntemi (Classification Method)

- Regresyon Yöntemi (Regression Method)

*Sınıflandırma*

Veri madenciliğinin en popüler dallarından birisidir. Sınıflandırmada verinin bir kısmı eğitim ve bir kısmı da test için kullanılır. Eğitim işlemi belirli kurallara göre gerçekleştirilir. Böylece yeni bir veri geldiğinde bu kurallar kullanılarak hangi sınıfa ait olduğu tespit edilir.

Veri sınıflandırma iki aşamadan oluşur. Öncelikle bir model ortaya konulur. Daha sonra da oluşturulan model test verisi ile test edilir. Karar ağaçları, Naive Bayes, yapay sinir ağları gibi algoritmalar başlıca veri sınıflandırma algoritmaları olarak sayılabilir.
Sınıflandırma yaklaşımlarını kısaca şu şekilde kategorize edebiliriz (Krzyk, 2019):

- İkili sınıflandırma (Binary classification): Bu tür sınıflandırmada iki tane sınıf etiketi söz konusudur. Örneğin cinsiyet açısından bir sınıflandırma yapılacağını varsayarsak; sınıflarımız erkek ve kadın olacaktır.
- Çoklu sınıf sınıflandırma (Multi-class classification): Bu tür sınıflandırmada ikiden fazla sınıf söz konudur veveriler aynı anda birden fazla sınıfa ait olamazlar.
- Çoklu etiket sınıflandırma (Multi-label classification): Bu tür sınıflandırmada veriler aynı anda birden fazla sınıfa ait olabilir. Örneğin bir roman hem hem bilim-kurgu hem de macera türüne ait olabilir.

Naive Bayes, karar ağaçları, destek vektör makineleri (support vector machines) ve yapay sinir ağları en önemli sınıflandırma algoritmalarına örnek olarak verilebilir.

*Regresyon yöntemi*

Regresyon yöntemleri devamlı veriler üzerinden hesaplama yaparak sonuç üretir. Örneğin bir evin fiyatını belirlemek pek çok devamlı parametreye bağlıdır. Bu parametreler üzerinden matematiksel bir fonksiyon ile fiyat hesaplamak bir regresyon analizidir. Bu işlemi gerçekleştirmek için öncelikle bir regresyon modeli ortaya koymak gerekir. Modelin doğru belirlenmesi istenen sonucun tutarlı olması açısından çok önemlidir. Doğrusal (linear) regresyon, lojistik (logistic) regresyon, çoklu doğrusal (multi-linear) regresyon ve polinomik (polinomial) regresyon başlıca regresyon algoritmalarıdır.

*Denetimsiz öğrenme*

Denetimli öğrenmenin aksine bir sınıf etiketine ihtiyaç duymaz. Denetimsiz öğrenme, veriler arasında çeşitli açılardan bağıntı veya benzerlikler bularak gruplayan yaklaşımlardır. Bu bağıntı veya benzerlik çoğu zaman uzaklıktır. Bu tür yaklaşımlarda verilerin ait olduğu sınıf/küme hakkında herhangi bir bilgi bulunmaz. Denetimsiz öğrenme çeşitlerini kısaca şu şekilde sıralayabiliriz (Krzyk, 2019):

- Kümeleme (Clustering)
- Birliktelik kuralları (Association rule mining)
- Boyut azaltma (Dimensionality reduction)

*Kümeleme*

Kümeleme, birbirine benzerlik teşkil eden verileri gruplayan veri madenciliği yaklaşımıdır. Bir başka deyişle, farklı kümeler birbirine en az benzeyen verileri içerir. Benzerlikten kastedilen bir uzaklık ölçütüdür. Öklid uzaklığı, Mahalanobis uzaklığı, Manhattan uzaklığı bunlardan bazılarıdır. *n* boyutlu nümerik bir veri uzayında, *p* ve *q* gibi iki nokta arasındaki Öklid uzaklığı olan *d(p,q)*, Eş. 1.1 ile hesaplanır.

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{1.1}$$

Mahalanobis uzaklığında Öklid uzaklığından farklı olarak dairesel değil eliptik bir uzaklık söz konusudur. *x, n* tane değişkenden oluşan bir vektör, $\mu$ bu verilerin ortalama vektörü ve *S* de kovaryans matrisi olmak üzere $d_M(x)$ Eş. 1.2. ile hesaplanır.

$$d_M(x) = \sqrt{(x-\mu)^T S^{-1}(x-\mu)} \tag{1.2}$$

Manhattan uzaklığı, köşegen uzaklık yerine koordinat bazında uzaklık hesaplar. Örneğin iki boyutlu iki nokta arasındaki mesafeyi ker iki noktanın x ve y bileşenleri arasındaki farkın toplamı şeklinde hesaplar. *x* ve *y*, *n* tane nitelikten oluşan birer veri olmak üzere iki veri arasındaki Manhattan uzaklığı, $d_{Man}(x,y)$, Eş. 1.3 ile hesaplanır.

$$d_{Man}(x, y) = \sum_{i=1}^{n} |x_i - y_i| \quad (1.3)$$

En yakın k komşu (K-Nearest Neighbours - KNN) (Cover ve Hart, 2006), k-means (Lloyd, 1982), k-median, DBSCAN (Ester ve diğerleri, 1996), OPTICS (Ankerst, Breunig, Kriegel ve Sander, 1999) gibi kümeleme yaklaşımları başlıca kümeleme yaklaşımları olarak sayılabilir. Hangisinin daha başarılı ve kullanışlı olacağı tamamen verinin yapısına ve özelliklerine göre değişir.

K-means (Lloyd, 1982) veriyi kullanıcıdan aldığı k tane kümeye ayıran bir kümeleme yaklaşımıdır. Algoritma ilk olarak veri üzerinde rastgele k tane merkez nokta belirler. Daha sonra verileri bu merkezlere olan mesafelerine göre atar. Sonra her küme için merkezi günceller. Sonra tekrar verileri mesafeye göre kümelere atar ve kararlı bir durum elde edilene kadar sistem bu şekilde devam eder. K-means algoritmasının en büyük dezavantajı küme sayısının önceden belirlenmesinin gerekliliğidir. K-median k-means'e benzemektedir. Ancak k-medians'ta ortalama yerine ortanca veri kullanılmaktadır.

DBSCAN (Ester ve diğerleri, 1996) algoritması yuvarlak kümeleme dışında farklı küme şekillerini de tespit etmeye yönelik önerilmiş bir algoritmadır. Bunu gerçekleştirmek için önce mikro-küme (micr-cluster) denilen küçük kümecikler tespit edilir. Daha sonra bu kümecikler aralarındaki mesafeye göre birleştirilerek asıl kümeler elde edilmektedir. OPTICS algoritması, DBSCAN algoritmasının gelişmiş bir versiyonudur. DBSCAN algoritmasında kullanılan MinPts ve Eps değişkenlerine bağımlılığı azaltmak için tespit edilen kümecikleri sıralı bir şekilde inceleyerek birleştiren bir yaklaşımdır.

En yakın k komşu (Cover ve Hart, 2006) algoritması verilerin çevresinde bulunan verilerin ait olduğu kümelere bakarak küme ataması yapan bir algoritmadır. Yeni bir veri geldiğinde bu veriye en yakın olan k tane komşusuna bakar. Komşularının en fazla ait olduğu küme verinin atanacağı küme olarak atanır. KNN bir anlamda denetimli bir öğrenme yaklaşımıdır. Çünkü verilerin ait olduğu küme/sınıf etiketleri üzerinden bir eğitim söz konusudur.

*Birliktelik kuralları*

Veriler arasındaki ilişkileri inceleyerek hangilerinin birbiriyle bağlantısı olduğunu ortaya koymaya yönelik çalışan veri madenciliği koludur. Özellikle pazarlama ve ticarette çok

kullanılan yöntemlerdir. Hangi ürünün hangisi ile ilişkili olduğunu bulmak pazarlama açısından kâr getirebilecek önemli bir bilgidir. Örneğin pazar sepet analizi en önemli kullanım alanıdır. Tahin alanların %80'i pekmez de alıyor bilgisine ulaşmak yapılacak kampanyalara ışık tutacak önemli bir bilgidir. Apriori (Agrawal, Imielinski ve Swami, 1993) algoritması en önemli birliktelik kuralı algoritması örneğidir. Bunun yanında Eclat (Zaki, 2000), FP-growth (Han, Pei ve Yin, 2000) algoritmaları kullanılmaktadır.

*Boyut indirgeme*

Boyut indirgeme çok boyutlu verileri işlerken ortaya çıkan performans kaybının önüne geçmek için gereksiz nitelikleri bertaraf etme yöntemleridir. Böylece hem değerli nitelikler kullanılarak başarı sağlanmakta hem de performans yükselmektedir. Bu alanda yapılan yaklaşımları üç ana gruba ayırmak mümkündür (Krzyk, 2019):

- Öznitelik çıkarma (Feature extraction): Veri ham haldeyken gereksiz niteliklerin silinmesi yaklaşımlarıdır.
- Öznitelik dönüşümü (Feature transformation): Birden fazla niteliğin birleştirilerek yeni bir niteliğe dönüştürülmesi yaklaşımlarıdır.
- Öznitelik seçimi (Feature selection): Gereksiz olan niteliklerin silinmesi yaklaşımlarıdır.

*Takviyeli öğrenme*

Bu tür yaklaşımlar canlı davranışlarını modeller. Bu yaklaşımlarda başarıyı arttırmak için ceza ve ödül yaklaşımlarından faydalanılır. İstenmeyen durumlarda ceza; istenen durumlarda ödül verilerek en iyiye ulaşma amaçlanır. Deep Learning bu yaklaşımlara örnektir (Krzyk, 2019).

Büyük veri

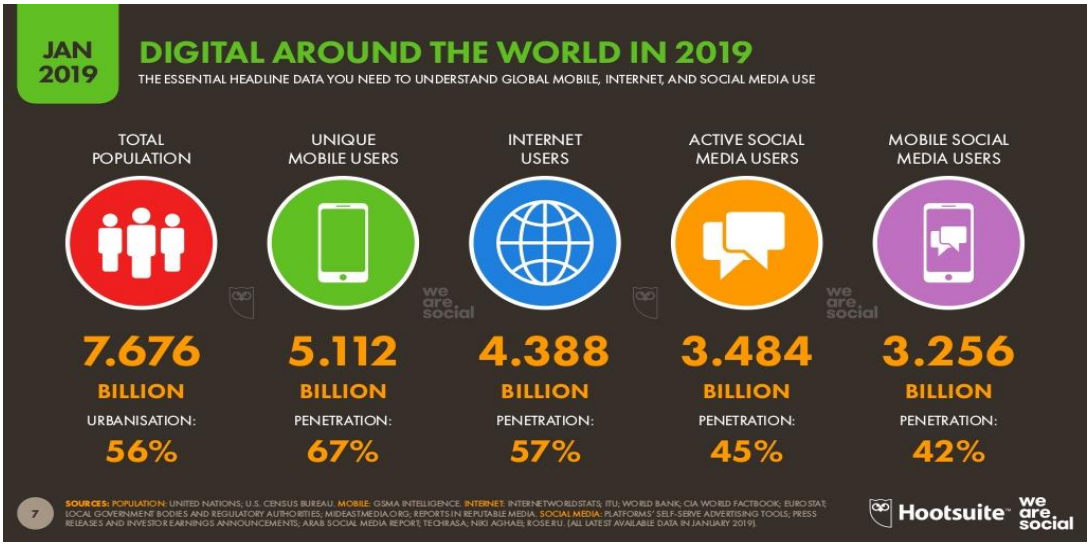
Bilgisayar teknolojisinin gelişmesine paralel olarak teknolojik cihazlara sahip olmak da ucuzlamıştır. Günümüzde insanların çoğu akıllı telefon, tablet, bilgisayar, sosyal medya hesapları gibi teknolojik araç ve ortamlardan birkaçına veya hepsine sahiptir. Her kullanıcı internet ortamını sıkça kullanmakta, paylaşımda bulunmakta ve var olan veri yığınına yenilerini eklemektedir. Bu nedenle internet ortamındaki veri günden güne artmaktadır. Artık günümüzde internet ortamındaki veri miktarı exabyte'lar ile ifade edilmektedir.

Günümüz dünyasındaki verilerin yaklaşık %90'ı son iki yılda yaratılmıştır. McKinsey Global Reportun 2011 verilerine göre 2020 yılında bu değer 44 katına çıkacaktır (Gobble, 2013).

Devletler ve işletmeler için bu büyük veriden faydalı bilgiyi çıkarmak oldukça önemlidir. İnternet ortamındaki verilerin çok azı yapısal veridir. Bu nedenle bu veriyi işleyecek yaklaşımlar önem arz etmektedir. Bu noktada "Büyük Veri" kavramı ortaya çıkmaktadır. Günümüzde büyük veri işlenmemiş petrol olarak nitelendirilmektedir. Çünkü bu devasa veriyi kullanan şirketler kârlarını arttırmakta, bu verileri analiz edebilen devletler hem gelişme ivmelerini attırmakta hem de ulusal güvenliklerini sağlamaktadır.

Günümüzde büyük veri kavramını daha da büyük bir veri haline getiren sadece insanlar değildir. Artık hayatımıza nesnelerin interneti (Internet of Things IoT) kavramı da girmiştir. Sıradan bilgisayar veya akıllı telefon kullanıcılarının yanı sıra GPS sinyalleri, sensörler gibi sürekli veri üreten kaynaklar da inanılmaz boyutlarda veri üretmektedir.

Büyük veri kavramının ne kadar büyük olduğuna bakmak için bazı istatistiksel verilere bakmak yeterli olacaktır. Şekil 1.5'te de görüldüğü gibi We Are Social'ın "Global Dijital Rapor 2019" Ocak 2019 verilerine göre dünya nüfusunun 7 milyar 676 milyon olduğu ve bunların 4 milyar 388 milyonunun internet kullandığı, 3 milyar 484 milyonunun aktif sosyal medya kullanıcısı olduğu, 5 milyar 112 milyonunun aktif cep telefonu kullanıcısı olduğu ve 3 milyar 256 milyonunun da aktif mobil sosyal medya kullanıcısı olduğu ortaya konmaktadır. Yıllık bazda bakıldığı zaman dünya nüfusu %1,1 artmasına rağmen mobil kullanıcı sayısı %2, internet kullanıcı sayısı %9,1, sosyal medya kullanıcı sayısı %9 ve mobil sosyal medya kullanıcı sayısı %10 oranında artmıştır (Kemp, Ocak 2019). Aynı rapora göre 2014 yılında ortalama bir insan günde 1 saat 38 dakika telefonla internette dolaşırken, bu rakam 2019 yılında 3 saat 14 dakikaya çıkmaktadır. Bu istatistikler de bilgisayar ortamına aktarılan veri miktarının ne kadar büyük olduğunu ortaya koymaktadır.



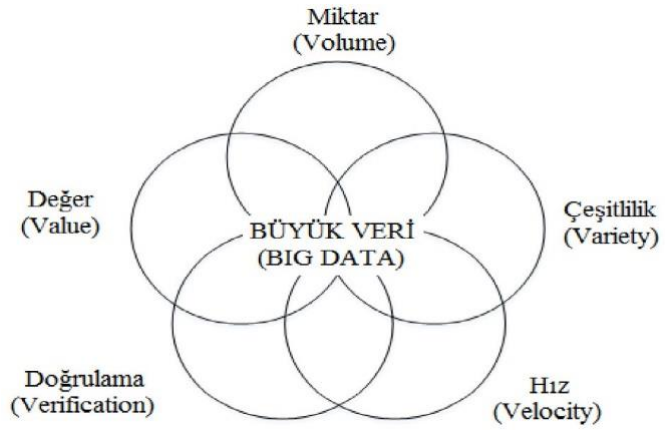

Şekil 1.5. 2018 verilerine göre teknoloji kullanımı (Kemp, Ocak 2019)

Büyük veri, geleneksel veri işleme yöntemleri ile işlenmesi mümkün olmayan verileri ifade etmektedir (Ohlhorst, 2012). Büyük veriyi işlemek, firmalar açısından tasarruf anlamına da gelmektedir. Dünyanın en büyük lojistik şirketi olan UPS 2011 yılında araçlarına GPS cihazları taktırarak minimum maliyetli yol bulma çalışması ile en elverişli yolları kullanmak suretiyle 48 300 000 km daha az km yaparak kâr etmiştir (Mayer-Schnberger, 2013). Bu çalışmanın kaynağı büyük veri işleme yaklaşımına dayanmaktadır. Mastercard şirketi 210 ülkedeki 1 milyar 500 milyon kullanıcısının yaptığı 65 milyar alışverişi analiz ederek kullanıcı trendlerini tespit etmiştir. Bu bilgileri kullanarak kullanıcılara özel kampanyalar yapmış ve önemli oranda kâr etmiştir. Amerika'da toplu taşıma yapan St. Missouri şirketi otobüslerinin motoruna kablosuz sensörler ekleyerek araç motorlarının bozulmadan önce gerekli bakımlarını yapmak suretiyle bakım maliyetini yaklaşık %10 azaltmıştır (Mayer-Schnberger, 2013). Bütün bu örnekler büyük verinin şirketler açısından ne kadar önemli olduğunu ortaya koymaktadır.

Literatürde genel anlamda verinin büyüklüğünün 3V bileşene bağlı olduğu kabul edilmektedir (Big Data Now: 2014 Edition, 2015; Davis, 2012; Dong ve Srivastava, 2013; Hitzler ve Janowicz, 2013; IBM, Zikopoulos ve Eaton, 2011; Reeve, 2013; Sagiroglu ve Sinanc, 2013). Bunlar hacim (volume), hız (velocity) ve çeşitlilik (variety)'tir. Ancak Şekil 1.6'da da görüldüğü gibi, bu bileşenlere 2V daha ekleyen çalışmalar da mevcuttur (Hitzler ve Janowicz, 2013; Manyika, Chui, Brown, Bughin, Dobbs, Roxburgh ve Byers, 2011). Bunlar da doğruluk (verification) ve değer (value)'dir. Bunlardan kısaca bahsedecek olursak:

Hacim (volume): Hacim büyük verinin büyüklüğünü ifade etmek için kullanılan bileşendir. Yukarıda da üzerinde durulduğu gibi veri miktarı yani hacmi günden güne artmaktadır.

Hız (velocity): Bu bileşen yaratılan verinin akışkanlığını yani hızını ifade etmektedir. Teknoloji geliştikçe ve bant genişliği arttıkça verinin bir noktadan bir noktaya gönderilme hızı da artmaktadır. Bu nedenle verinin akış hızı da artmaktadır. Dolayısıyla bu verileri işlemek daha zor olmaktadır.




Şekil 1.6. Büyük veri bileşenleri (Hitzler ve Janowicz, 2013)

Çeşitlilik (variety): Çeşitlilik verinin aynı yapıda olmamasını, belli bir yapısının olmamasını ifade eder. Yani veri homojen değildir. Bu nedenle geleneksel veri işleme yöntemleri ile işleme imkanı yoktur. Genel anlamda büyük veride veriler farklı kaynaklardan geldiği için farklı yapıdadır.

Doğrulama (verification): Veri akarken doğru kişiye doğru şekilde ve doğru yetkilendirme ile ulaştırılması gerekir. Yetkisi olmayan kişilerin veriye ulaşamaması gerekir.

Değer (value): Eldeki devasa verinin bir anlam, bir değer ifade etmesi gerekir. Bu verinin işlenmesi suretiyle değerli bilgiye ulaşmak ve kuruluşa bir değer katmak gerekir. Bir şirket için pazarlama veya maliyet açısından bir avantaj ortaya koyması gerekir

Tezin katkıları

Bu tez çalışması akan veri üzerinde kümeleme için yeni bir yöntem önermektedir. Klasik kümeleme yaklaşımlarından farkı kümelemeyi gerçek zamanlı, yani veri seti değiştikçe

yapmasıdır. Veri seti değiştikçe model de kendisini buna göre uyarlayarak gereken yerde var olan kümelerden bazılarını birleştirme, bir kümeyi bölme, var olan bir kümeyi aktif/pasif yapma veya var olan kümelerin dışında yeni bir küme oluşturma yoluna gitmektedir. Bu işlemler tamamen veri setinin özelliğine göre gerçekleşmektedir. Önerdiğimiz yaklaşım uzaklık ölçütü olarak Öklid uzaklığı kullanmaktadır.

Önerilen kümeleme yaklaşımının var olan akan veri kümeleme yaklaşımlarından en önemli farkı kümeleme işleminin tamamen gerçek zamanlı olarak yapmasıdır. Çoğu akan veri kümeleme yaklaşımında kümeleme işlemi çevrimiçi ve çevrimdışı olarak iki bileşenden oluşur. Çevrimiçi bölümde veri akarken belli oranda akan veriden bir özet alınmakta ve çevrimdışı bölümde de bu özet üzerinde kümeleme işlemi yapılmaktadır.

Akan veri kümeleme yaklaşımlarının desteklemesi gereken önemli özelliklerden biri kümelerin geçmişi ile ilgili bilgileri tutmasıdır. Çünkü zamana bağlı olarak kümeler aktif/pasif duruma dönüşebilmelidir. Eğer sahip olduğu veri sayısı belirli bir sayının altına düşen veya sıfırlanan bir küme silinirse aynı bölgede oluşan bir küme yeni bir küme olarak tanımlanacaktır. Oysa bu küme aslında daha önce tanımlanmış ama zamana bağlı olarak sahip olduğu verileri silinmiş bir kümedir. Literatürdeki çalışmaların neredeyse tamamı söz konusu durumda kümeleri silmektedir. Önerdiğimiz yaklaşım bu kümeler ile ilgili bilgileri tutarak kümelere zamana bağlı olarak aktif/pasif olma imkânı tanımaktadır.

Bu çalışmanın en önemli yenilikçi yaklaşımlardan biri uyarlanabilir yarıçap özelliğidir. Bir kümenin oluşabilmesi için kullanıcı tarafından belirlenmiş r yarıçap içerisinde N tane veri olması gerekir. Küme oluştuktan sonra her adımda r_treshold kadar uzaklıktaki veriler de kümeye dâhil edilir ve kümenin yeni yarıçapı r+r_treshold olarak belirlenir. Kümenin yarıçapı belirlenmiş olan r_max değerine ulaşıncaya kadar artabilmektedir. Benzer şekilde küme yarıçapının azaltılması da söz konudur. Ayrıca bir küme pasif duruma dönüştüğünde yarıçapı r/2 olarak atanacaktır. Yani kümelerin yarıçapı r/2 ve r_max değerleri arasında verinin özelliğine bağlı olarak değişebilmektedir.

Var olan akan veri kümeleme yaklaşımlarında çeşitli özetleme yaklaşımları kullanılmaktadır. Pencere tabanlı (Sliding window), rastgele örneklem (random sampling) veya fading (verilerin ağırlıklarının zamana bağlı olarak azaltılması) gibi özetleme yaklaşımları kullanılmaktadır. Önerdiğimiz yöntemde aging, yani yaşlandırma işlemi

yapılmaktadır. Yani her veri zamana bağlı bir şekilde yaşlanmakta ve belirlenmiş eşik değerini aştığında ömrünü tamamlamakta ve silinmektedir. Böylece eski değerler silinerek kümelemenin güncel veriler üzerinde yapılması sağlanmaktadır. Bir yönü ile pencere tabanlı veri özetleme yaklaşımlarına benzemektedir. Ancak burada sabit bir pencere büyüklüğü söz konusu değildir. Burada sabit bir zaman söz konusudur. Bu yaklaşım zamanın önemli olduğu uygulamalar için çok faydalı olacaktır. Ayrıca belirlenen zaman zarfında gelen veri sayısı çok büyük olduğunda, ön tanımlı bir değişken olan TN tanesi (son gelen TN tane) özet olarak alınır. Yani önerdiğimiz yaklaşım hibrit bir özetleme özelliğine sahiptir.

Önerdiğimiz yöntemin bir diğer önemli özelliği küme sayısına bir sınır getirmemesi ve kullanıcının ön tanımlı olarak bir değer girmesine gerek olmamasıdır. Çünkü önerdiğimiz yöntemde algoritma kaç tane küme olacağına bakmadan ihtiyaç oldukça yeni kümeler oluşturmaktadır.

Kümeleme yaklaşımlarında kullanıcının belirlediği parametrelerin mümkün olduğunca az olması önemlidir. Çünkü parametre sayısı başarıyı ve sonucu doğrudan etkilemektedir. Geliştirdiğimiz bu yöntemde kullanıcının mümkün olduğunca az parametre belirlemesine önem verdik. Yöntemimizde kullanıcı 7 parametreyi belirlemektedir. Bu parametrelerin açılamaları Çizelge 1.1'de verilmiştir.

Çizelge 1.1. KD-ARFS Stream algoritmasında kullanılan parametreler

| Parametre | Açıklaması |
|---|---|
| r | Küme oluşturma yarıçapı |
| r_threshold | Küme yarıçapını arttırma/azaltma eşik değeri |
| r_max | Bir kümenin ulaşabileceği maksimum yarıçap |
| N | Küme oluşturmak için gerekli minimum veri sayısı |
| t | Verinin ömrünü tamamlama eşik değeri |
| TN | Gelen veri miktarı çok fazla olduğu zaman ne kadarlık kısmının özet olarak alınacağını belirleyen değer |
| coefThreshold | Öznitelik seçme eşik değeri |

Akan veri kümeleme yaklaşımları performansı arttırmak adına çeşitli boyut indirgeme yaklaşımı kullanmaktadır. Geliştirdiğimiz yöntemde standart sapma tabanlı bir öznitelik seçme işlemi yapılmaktadır. Standart sapma değeri düşük olan niteliklerin seçici özelliğinin de azdır. Standart sapması az olan nitelikler elemine edilerek hem performans arttırılmakta, hem de yüksek kümeleme başarısı elde edilebilmektedir.

Tez organizasyonu

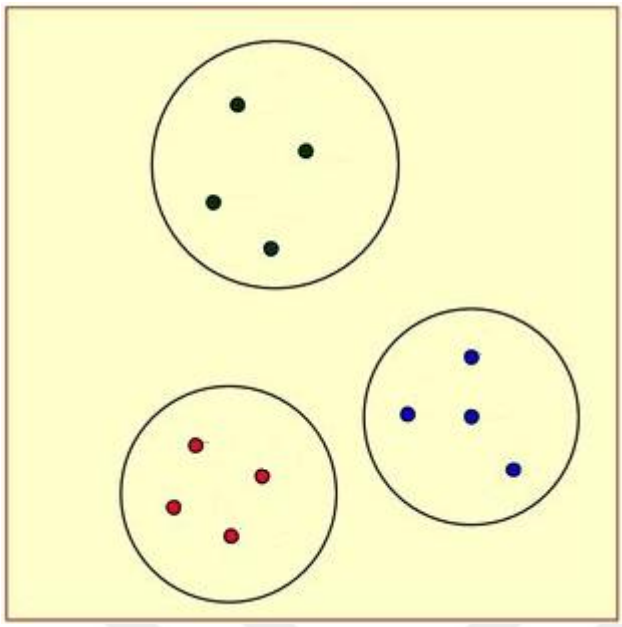
Tezin organizasyonu şu şekildedir: 2. bölümde akan veri kümeleme alanında yapılan çalışmalar derlenmekte ve bunların karşılaştırılması yapılarak bu alanda bulunduğumuz noktanın fotoğrafı çekilmektedir. 3. bölümde önerdiğimiz K-boyutlu ağaç, uyarlanabilir yarıçap ve öznitelik seçme (KD-ARFS Stream) tabanlı gerçek zamanlı akan veri kümeleme algoritması tüm detayları ile açıklanmaktadır. 4. bölümde yapılan deneysel çalışmalar ve elde edilen sonuçlar verilmekte ve değerlendirmeler yapılmaktadır. 5. ve son bölümde sonuç açıklanırken ve gelecekte yapılması planlanan çalışmalar verilmektedir.

# 2. AKAN VERİ KÜMELEME

## 2.1. Kümeleme ve Akan Veriye Giriş

Kümeleme, veriler arasındaki benzerlik veya farklılıkları baz alarak sınıflandırma yapan bir veri madenciliği yöntemidir. Kümeleme yaklaşımında iki temel özellik vardır: birincisi aynı kümede yer alan nesneler arasındaki benzerlikler maksimum olmalı ve farklı kümelerde yer alan nesneler arasındaki benzerlik minimum, bir başka deyişle farklılık maksimum olmalıdır. Kümeleme algoritmaları veri madenciliği (Judd, McKinley ve Jain, 1998), veri sıkıştırma (Abbas ve Fahmy, 1994), resim segmentasyonu (Coleman ve Andrews, 1979; Jain ve Dubes, 1988; Ray ve Turi, 2000), makine öğrenmesi (Carpineto ve Romano, 1996), istatistik gibi pek çok alanda kullanılır. Kümeleme algoritmaları hiyerarşik ve parçalı (partitional) olarak ikiye ayrılabilir (Frigui ve Krishnapuram, 1999; Leung, Zhang ve Xu, 2000). Hiyerarşik kümeleme yaklaşımında veri seti bir ağaç yapısında birbiriyle ilişkilendirilirken, partitional kümelemede veriler birbirlerine olan benzerliklerine göre gruplandırılır. Kümeleme ile ilgili yapılan çalışmaların karşılaştığı en önemli problem sınıf sayısının önceden belirlenmesinin gerekliliğidir. Çünkü çoğu problem için kullanıcının düşündüğünden farklı sayıda küme sayısı olabilir (Omran, Salman ve Engelbrecht, 2006).

Bu açıdan bakıldığında en temel kümele algoritması olarak bahsedilmesi gereken algoritma K-means algoritmasıdır (Hartigan ve Wong, 1979). Bu algoritmada veri seti K tane kümeye ayrıştırılır ve iteratif bir şekilde her adımda merkez, seçilen bir uzaklık ölçütüne(Öklit, Minkowski veya Manhattan gibi) göre yeniden hesaplanarak işlem sürdürülür (Lee ve Antonsson, 2000). Bu algoritmada ilk merkezler rastgele seçildiğinden seçilen ilk noktaların yeri sonucu etkileyebilen önemli noktalardan biridir. Bu nedenle küme sayısı hakkında fikir sahibi olmak için veri seti üzerinde bir ön çalışma yapmaya ve veri hakkında yeterince bilgi sahibi olmaya ihtiyaç vardır (Omran ve diğerleri, 2006). Bu amaçla yapılmış çeşitli çalışmalar vardır (Halkidi, Batistakis ve Vazirgiannis, 2001; Theodoridis ve Koutroumbas, 2008). Optimum küme sayısını bulmaya yönelik çalışmalar yine de en iyi sonucu yani en doğru küme sayısını vermeyi garanti etmez (Rosenberger ve Chehdi, 2000). Şekil 2.1'de kümelemeye örnek verilmektedir.



Şekil 2.1. Veri kümeleme örneği (Mokhtari, Melkemi, Michelucci ve Foufou, 2014)

Kümeleme alanındaki son zamanların gözde konularından biri de akan veri kümeleme konusudur. Akan veriyi kümelemek klasik veri kümelemeye göre çok daha zor bir kümeleme işlemidir. Çünkü akan veri kümelemede geleneksel veri kümeleme yaklaşımlarında olduğu gibi veriyi bir yere kaydederek tekrar tekrar işleme imkanı yoktur. Bunun yanında veri sürekli değişmekte, çoğu zaman veri seti çok hızlı bir şekilde değişmektedir. Dolayısıyla veri miktarı da sürekli artmaktadır. O yüzden akan veri kümeleme yaklaşımlarının çok hızlı olması ve veriyi bir defa işlemesi gerekir.

## 2.2. Akan Veri Kümelemede Karşılaşılan Problemler

Pek çok açıdan bakıldığı zaman akan veri kümeleme klasik veri kümeleme yaklaşımlarından farklıdır. Geleneksel kümeleme yaklaşımları ve akan veri kümeleme yaklaşımları arasındaki farkları şu şekilde sıralayabiliriz (Yogita ve Toshniwal, 2013):

• Geleneksel yöntemde veriler statiktir; ancak akan veride veri dinamiktir, sürekli değişmektedir.

• Akan veride geleneksel yöntem gibi veriyi bir yere kaydedip tekrar tekrar işleme imkânı yoktur.

• Geleneksel yöntemde sonuçlar sabittir; ancak akan veride sonuç zamana bağlı olarak değişir.

Akan veri pek çok açıdan kısıtlayıcıdır. Geliştirilecek yöntemin kaynakları etkin kullanması ve tatmin edici sonuçları makul bir zamanda sunması ihtiyacı bunlardan bazılarıdır. Çünkü sınırlı kaynaklara ve zamana sahibiz. Bu nedenle geliştirilecek yöntemlerin bu problemleri göz önünde bulundurması gerekir. Akan veri kümelemede en sık karşılaşılan problemleri şu şekilde sıralayabiliriz (Yogita ve Toshniwal, 2013):

• Sonsuz boyut ve yüksek hız: Akan veride verinin sonu genelde yoktur veya sınırları bilinmez.

• Dinamizm: Veri dinamiktir, her an değişir.

• Veriye genel bakış: Geleneksel kümeleme yaklaşımlarında veriye genel bir bakış attıktan sonra daha iyi sonuçlar elde edilebilmektedir; ancak akan veride verinin sonu belli olmadığından veri hakkında kestirimde bulunmak mümkün değildir.

• Sapan veri: Geleneksel kümeleme yaklaşımlarında veri ön işleme ile sapan veri olup olmadığını, var ise bunları bertaraf etme imkânı vardır; ancak akan veride bunu tespit edip düzeltmek oldukça zordur.

• Çok boyutluluk: Akan veride çok boyutlu veriyi işlemek geleneksel yönteme göre oldukça zordur.

• Parametreleri belirleme: Akan veride parametreleri belirlemek tüm veriye hakim olunmadığından oldukça zordur.

Bütün bu problemlerin yanında akan veri için geliştirilmiş kümeleme yaklaşımlarının sağlaması gereken minimum kriterler de mevcuttur. Bunları şu şekilde sıralayabiliriz (Yogita ve Toshniwal, 2013):

• Uyarlanabilirlik: Akan veri kümeleme yaklaşımlarının her yeni gelen veriye göre baştan kümeleme yapmak yerine yeni gelen veriye göre kendisini uyarlayabilmesi gerekir.

• Veriyi bir defa işleme: Akan veri kümeleme yaklaşımlarının veriyi tekrar tekrar işlemek gibi bir imkânı olmadığından bir defa işleyerek sonuç üretebilmesi gerekir.

• Zaman kısıtı ve alan kriteri: Veri sürekli aktığından bu veriyi hafızaya kaydedip işleme imkanı yoktur. Bu nedenle veriyi çok kısa bir zamanda ve çok az kaynak harcayarak işlemek gerekir.

• Kümeleme adaptasyonu (Concept Evolution): Veri sürekli değiştiğinden daha önce oluşturulmuş kümelerin gelen veriye göre adapte olabilmesi gerekir. Gerektiğinde var olan kümeler dışında yeni bir küme oluşturma, var olan bir kümeyi yok etme, var olan iki kümeyi birleştirme veya var olan bir kümeyi farklı sayıda kümelere ayırmak gerekebilir.

• Devamlı kümeleme modeli: Akan verinin sonuna kadar bekleyip kümeleme yapma imkânı

yoktur. Bu nedenle veri akarken kümeleme modelinin de bir taraftan kümeleme işlemini gerçekleştirmesi gerekir. Kullanıcının istediği her an küme sonuçlarını sunabilmesi gerekir.
• Sapan veri tespiti: Geliştirilen kümeleme yaklaşımının sapan verileri tespit edebilmesi gerekir. Çünkü sapan veriler kümeleme başarısını doğrudan etkilemektedir. Özellikle çok sayıda sapan veri içeren veri setlerinde sapan verileri gerçek verilerden ayırt etmek çok önemlidir.

## 2.3. Akan Veri Kümeleme Yaklaşımlarının Uygulama Alanları

Akan veri kümeleme yaklaşımları gerçek zamanlı yapılan pek çok uygulamada kullanılmaktadır. Kullanıcıya ait tıklama verisi analizi ile kullanıcı eğilimi tespiti (Antonellis, Makris ve Tsirakis, 2009), saldırı tespit sistemleri (Li, 2014; Yin, Xia ve Wang, 2017, 2018), sosyal medya (Barddal, Gomes ve Enembreck, 2015; Hawwash, 2013; Weiler, Grossniklaus ve Scholl, 2016), finansal uygulamalar (Hendricks, 2017), bilimsel araştırmalar (Aggarwal, 2010), salgıntespit sağlık araştırmaları (Gravina, Alinia, Ghasemzadeh ve Fortino, 2017; King, Villeneuve, White, Sherratt, Holderbaum ve Harwin, 2017; Manzi, Dario ve Cavallo, 2017), mobil uygulamalar (Tasnim, Caldas, Pissinou, Iyengar ve Ding, 2018), nesnelerin interneti (IoT) (Diaz-Rozo, Bielza ve Larrañaga, 2018) ve sensor ağ (Sabit, Al-Anbuky ve Gholam-Hosseini, 2009; A. d. Silva, Chiky ve Hebrail, 2012) gibi pek çok alanda kullanılmaktadır. Salgın erken tespit sistemleri, bankacılıkta dolandırıcılık tespiti ve hükümetlerin güvenlik uygulamalarında zamanla çok daha fazla kullanılması beklenmektedir.

## 2.4. Akan Veri Kümeleme Alanında Kullanılan Test Veri Setleri

Akan veri kümeleme alanında yapılan çalışmaların büyük bir çoğunluğunda sentetik veriler kullanılmaktadır. Çünkü çalışmaların odaklandığı sapan veri tespiti veya çok boyutluluk gibi problemleri aynı anda sağlayan gerçek verileri bulmak çok zordur. Bu nedenle bu alanda çalışma yapan araştırmacılar genelde kendi verilerini üretmektedir. Bunun yanında performans veya başarı karşılaştırmaları için KDD-CUP '99, KDD-CUP '09 veya UCI'nin orman örtüsü verisi de sıkça kullanılmaktadır. Kullanıma açık akan veri kümeleme verilerini sunan kaynakları şu şekilde sıralayabiliriz (J. A. Silva ve diğerleri, 2013):
• UCI Knowledge Discovery in Databases Archive - http://kdd.ics.uci.edu: Makine öğrenmesi ile ilgili veri setlerinin paylaşıldığı web sayfası.

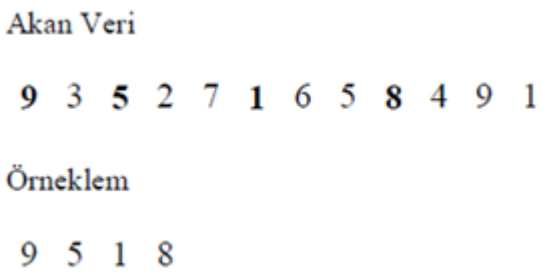
• KDD Cup Center - http://www.kdd.org/: Bilgi keşfi ve veri madenciliği ile ilgili verilerin paylaşıldığı dizin

• UCR Time-Series Datasets - http://www.cs.ucr.edu/~eamonn/time_series_data: Zaman damgalı verilerin paylaşıldığı web dizini

## 2.5. Akan Veride Temel Veri Özetleme Metodolojileri

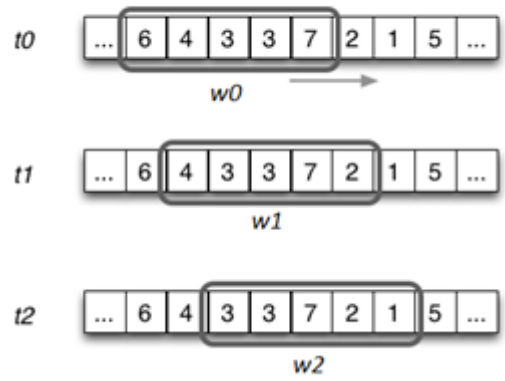
Sonsuz depolama imkânına sahip olunmadığından kümeleme doğruluğu ve depolama alanı arasında makul bir oranın tutturulması gerekir. Bütün veriyi depolamak ve kümeleme işlemine tabi tutmak kümeleme başarısı açısından daha iyi olabilir; ancak bu durum, büyük depolama ihtiyacı doğurmaktadır. Bu nedenle verinin tamamı yerine özetini almak hem kümeleme başarısı açısından makul bir sonuç almayı mümkün kılmakta, hem de alan açısından tasarruf sağlamaktadır. Bu amaçla çeşitli özetleme algoritmaları geliştirilmiştir:

Rastgele örneklem alma (Random sampling): En basit özetleme yaklaşımıdır. Şekil 2.2'de de görüldüğü gibi bu özetleme yaklaşımında belli aralıklarla örnek alınır. Bu özetleme yaklaşımının en büyük özelliği çoğu uygulama açısından kullanımı oldukça kolaydır.



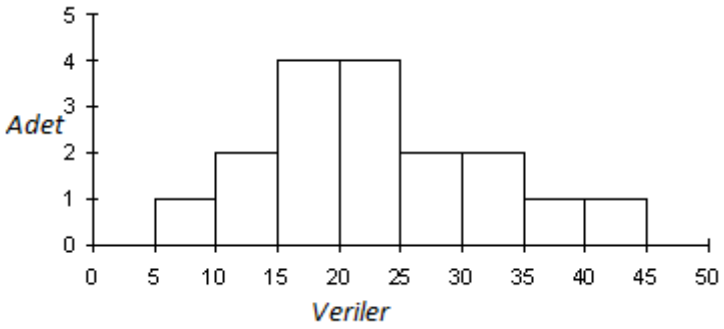

Şekil 2.2. Rastgele örneklem

Kayan pencere (Sliding window): Bu yaklaşımda örnekleme yapmak yerine sadece en son gelen belli sayıda veri alınır (Datar, Gionis, Indyk ve Motwani, 2002). Şekil 2.3'te de görüldüğü gibi, herhangi bir t anında en son gelen w kadar veri örnek olarak alınır. Stok veya sensör network uygulamaları için oldukça uygun bir yaklaşımdır. En son gelen verilerin daha önemli olduğu uygulamalar için kullanımı oldukça uygundur. Benzer şekilde son 10 dk. içerisinde gelen veriler gibi sabit veri penceresi de kullanılmaktadır. Hafıza açısından oldukça tasarrufludur, sadece w kadar hafızaya gereksinim duyar.




Şekil 2.3. Sliding window örnekleme

Histogramlar: Veriye ait değerler için frekans değerlerine göre özet çıkaran bir yaklaşımdır. Bu yaklaşımda veriler kova (bucket) denilen parçalara bölünür. Şekil 2.4'te de görüldüğü gibi, her kova içerisinde bulunan veri sayısı o kova için frekans değeri olarak alınır. Her kovanın boyutu gerekli hafıza miktarını belirler. Dezavantajı geriye dönük işlem yapamamasıdır. Veri için histogram bulunduktan sonra asıl veriler göz ardı edilmektedir.




Şekil 2.4. Histogram örneği

Multiresolution metodlar: Büyük veri miktarı ile baş etmek için geliştirilmiş bir yöntemdir. Bu yaklaşımda veri miktarını azaltmak için parçala ve fethet yaklaşımı izlenir. Bu yaklaşımın en önemli avantajı veriyi yönetme anlamında doğruluk ve kaynak açısından optimum sonuç vermesidir. Ayrıca veriyi farklı açılardan anlamaya yardımcı olur. Örnek olarak dengeli ikili arama ağaçlarını ele alırsak kökten yapraklara doğru indikçe her adımda sonuç detaylandırılmış olur. Bu yaklaşım da buna benzemektedir. İki çeşit multiresolution metodu vardır:

- Micro-clusters (Aggarwal, Han, Wang ve Yu, 2003): En çok kullanılan özetleme yaklaşımlarından biridir. En büyük avantajı çok boyutlu veriler için oldukça kullanışlı

olmasıdır.

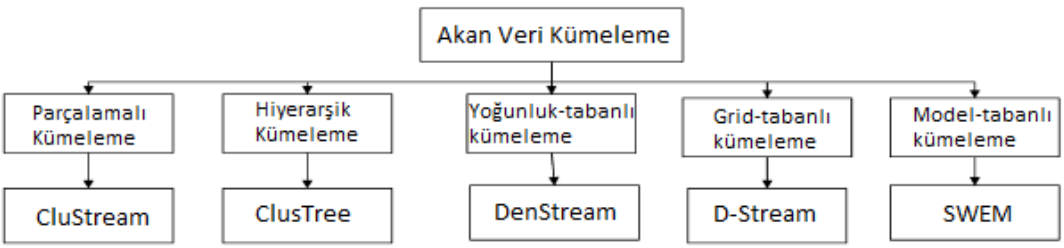
- Wavelets (Keim ve Heczko, 2001): Wavelets veritabanlarında hiyerarşik verilerin ayrıştırılmasında ve özetlenmesinde sıkça kullanılan bir yöntemdir. Temelde hiyerarşik verileri ayrıştırmak için kullanılan bir yaklaşımdır.

Sketches: En doğruya yakın bir cevabı elde etmeyi garanti eden bir yaklaşımdır. Veriyi bir defa işleme mantığına göre çalışır. Veriye ait sıklık momenti (frequency moment) sketch denilen özetler ile elde edilir.

Randomized Algorithms: Çok boyutlu ve büyük verileri işlemek için geliştirilmiş bir yaklaşımdır. Random sampling ve sketching yaklaşımlarının birleşiminden oluşur.

## 2.6. Akan Veri Kümelemede Temel Yaklaşımlar

Akan veriyi kümelemek için geliştirilmiş yaklaşımları iki ana başlığa ayırabiliriz: Örnekleme tabanlı akan veri kümeleme ve değişken tabanlı veri kümeleme (Yogita ve Toshniwal, 2013). Bunun yanında bu alanda yapılmış çalışmaları parçalamalı (partitional) veya hiyerarşik olmak üzere iki gruba ayıran (Kavitha ve Punithavalli, 2010) veya Şekil 2.5'te görüldüğü gibi parçalamalı, hiyerarşik, yoğunluk-tabanlı, grid-tabanlı ve model tabanlı olarak beş ana gruba ayıran çalışmalar da mevcuttur (Amini, Wah ve Saboohi, 2014). Bu yöntemlerin avantaj ve dezavantajları Çizelge 2.1'de verilmiştir.




Şekil 2.5. Akan veri kümeleme yaklaşımları (Amini, Wah ve Saboohi, 2014)

Parçalama temelli akan veri kümeleme yaklaşımında veri k-means gibi uzaklık tabanlı bir algoritma ile çeşitli sayıda kümelere ayrılır. Bu tür yaklaşımlarda küme şekli yuvarlaktır. Oysaki küme şekli verinin özelliğine bağlı olarak değişebilir. Bu yaklaşımlarda genelde sonuç gürültü ve sapan veriden etkilenir. STREAM (O'Callaghan, Mishra, Meyerson, Guha

ve Motwani, 2002) ve CluStream (Aggarwal, Han, Wang, ve Yu, 2003). bu yaklaşımlara örnek olarak verilebilir.

Hiyerarşik kümeleme, veriyi kümelerden oluşan bir ağaç yapısında gruplar. Veri örnekleme (summarization) ve veri görselleştirme açısından faydalı bir yaklaşımdır. Küme birleştirme (merge) ve küme bölme (split) işlemi sürekli yapılır. BIRCH (Zhang, Ramakrishnan ve Livny, 1996) ve Chameleon (Karypis, Han ve Kumar, 1999) bu yaklaşımlara örnek verilebilir. Hiyerarşik kümelemenin başarısını arttırmak için diğer yöntemlerle birleştirilmiş yaklaşımlar da mevcuttur. Bu yaklaşımlara örnek olarak ClusTree (Kranen, Assent, Baldauf ve Seidl, 2011) verilebilir.

Grid tabanlı kümeleme, verinin dağılımından bağımsızdır. Veriyi grid denilen parçalara ayırır. Zaman açısından hızlı bir kümeleme yaklaşımıdır. STING (Wang, Yang ve Muntz, 1997), WaveCluster (Sheikholeslami, Chatterjee ve Zhang, 2000) ve CLIQUE (Agrawal, Gehrke, Gunopulos ve Raghavan, 1998) bu yaklaşımlara örnek verilebilir. Grid tabanlı yaklaşımların yoğunluk tabanlı yaklaşımlarla birleştirildiği çok sayıda çalışma da mevcuttur. D-Stream (Tu ve Chen, 2009) ve MR-Stream (Wan, Ng, Dang, Yu ve Zhang, 2009) bunlara örnek verilebilir.

Model tabanlı kümeleme yaklaşımları, EM (Expectation Maximization) (Dempster, Laird ve Rubin, 1976) gibi matematiksel modellerle veri arasında bir optimizasyon bulmaya çalışan kümeleme yaklaşımlarıdır. EM algoritması k-means algoritmasının değişik bir versiyonu olarak düşünülebilir. SWEM (Dang, Lee, Ng, Ciptadi ve Ong, 2009) algoritması bu yaklaşımlara örnek verilebilir.

Yoğunluk tabanlı kümeleme, yoğunluğu baz alarak kümeleme yapan akan veri kümeleme yaklaşımlarıdır. Bu yaklaşımlarda kümeler verinin yoğunlaştığı alanlarda yoğunlaşır. Temel yaklaşımı kümeyi verinin yoğunlaştığı yerlere doğru genişletmeye dayanır. Bu işlemi gerçekleştirirken belirli bir eşik değer kullanır. Böylece sapan verileri bertaraf etmek ve veriye göre şekiller elde etmek mümkün olabilmektedir. DBSCAN (Ester ve diğerleri, 1996), OPTICS (Ankerst ve diğerleri, 1999) ve DENCLUE (Hinneburg ve Keim, 1998) yoğunluk tabanlı kümeleme algoritmalardır. DenStream (Cao, Estert, Qian ve Zhou, 2006) akan veriyi kümelemeye yönelik geliştirilmiş yoğunluk tabanlı kümeleme algoritması olarak verilebilir.

Çizelge 2.1. Akan veri kümeleme yaklaşımlarının karşılaştırması (Mousavi, Bakar ve Vakilian, 2015)

| Metot | Avantaj | Dezavantaj | Örnekler |
|---|---|---|---|
| Parçalamalı | • Uygulaması kolaydır<br>• İteratif bit şekilde kümeleri oluşturur | • Küme sayısı kullanıcı tarafından tanımlanmalıdır<br>• Sadece dairesel şekiller elde edilebilir | CluStream (Aggarwal, ve diğerleri, 2003), StreamKM++ (Ackermann, Martens, Raupach, Swierkot, Lammersen ve Sohler, 2012) |
| Hiyerarşik | • Mesafe veya benzerliği ele alış şekli işi kolaylaştırır | • İşlemi tamamlama kriterlerinde problem olabiliyor<br>• Çalışma zamanı yüksektir | BIRCH (Zhang, Ramakrishnan ve Livny, 1996), E-Stream (Udommanetanakit, Rakthanmanon ve Waiyamai, 2007) |
| Model-tabanlı | • Standart istatistiki bilgilere dayanarak küme sayısını otomatik bir şekilde tanımlayabilir<br>• Sapan verileri tespit edebilir | • Model veya tanımlanmış yapıya bağlıdır | SWEM (Dang ve diğerleri, 2009) |
| Grid-tabanlı | • İşlemleri hızlı bir şekilde yapar<br>• Sapan verileri tespit edebilir | • Çok boyutlu veriler kullanılamaz<br>• Grid boyutu tanımlanmak zorundadır | D-Stream (Tu ve Chen, 2009), CEDAS (Richard Hyde, Angelov ve MacKenzie, 2017) |
| Yoğunluk-tabanlı | • Farklı şekle sahip kümeleri tespit edebilir.<br>• Sapan verileri tespit edebilir | • Çok sayıda parametre tanımlanmalı<br>• Çoklu yoğunluk durumlarında çalışmaz | DenStream (Cao ve diğerleri, 2006), DENCLUE (Hinneburg ve Keim, 1998) |

## 2.7. Başlıca Akan Veri Kümeleme Algoritmaları

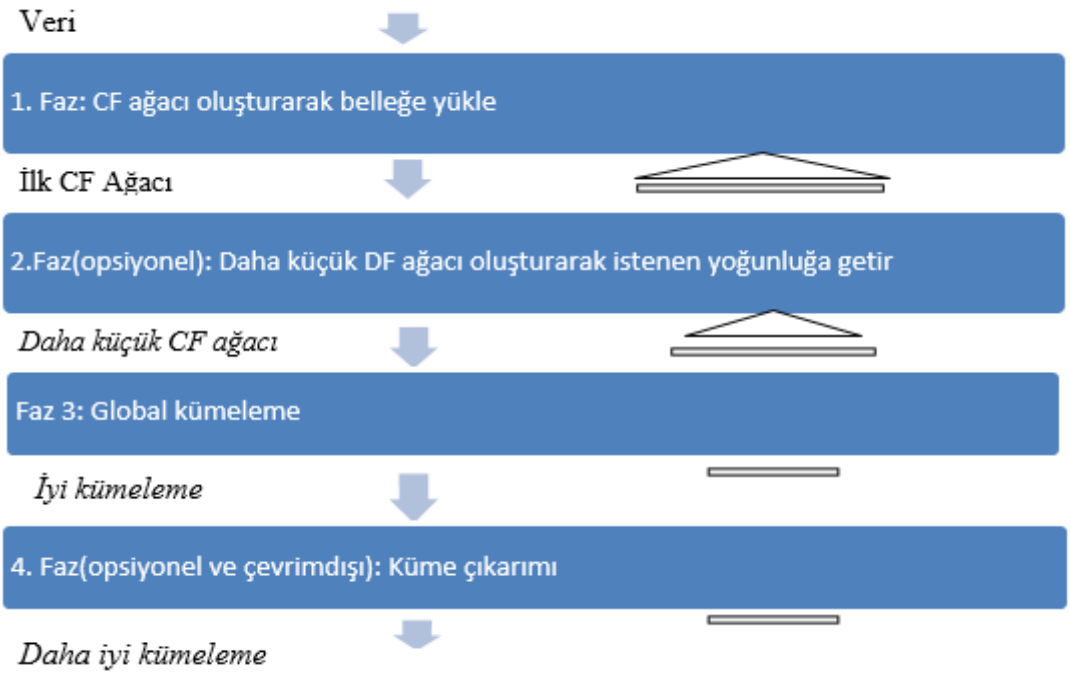
### 2.7.1. Örnekleme tabanlı akan veri kümeleme algoritmaları

Örnekleme tabanlı akan veri kümeleme yaklaşımlarında verinin farklı kaynaklardan geldiği düşünülür. Bir banka örneğini ele alırsak birbirinden faklı işlemlere ait veriler akmaktadır.

Kredi bilgisi, kredi kartı bilgisi, farklı müşterilere ait bilgiler vs. Bu alanda yapılmış başlıca çalışmaları şu şekilde sıralayabiliriz:

STREAM (O'Callaghan ve diğerleri, 2002), k-means tabanlı iki aşamadan oluşan bir yaklaşımdır. Divide and conquer yaklaşımına dayanır. İlk aşamada akan veri kova (bucket) denilen parçalara bölünür ve her kova için k-median uygulanarak k tane küme bulunur. Küme merkezleri kaydedilerek küme merkezleri sahip oldukları veri miktarlarına göre ağırlıklandırılır. Bu noktada asıl veriler göz ardı edilerek ağırlıklandırılmış veri merkezleri veri olarak kabul edilir. İkinci aşamada ağırlıklandırılmış veri merkezleri kümelenerek daha az sayıda küme elde edilir.

CluStream (Aggarwal ve diğerleri, 2003), bir çevrimiçi ve bir çevrimdışı bileşenden oluşur. Çevrimiçi bileşen veriyi micro-cluster yapısında özetler. Burada micro-clusterlar BIRCH (Zhang ve diğerleri, 1996)'in kümeleme özelliğinin geçici genişletilmiş bir versiyonudur. Veriye ait özetleme istatistikleri belleğe kaydedilir. Bu durum, kullanıcıya zaman açısından esneklik sağlar. Çevrimdışı komponent ise k-means kümeleme uygulayarak micro-clusterları daha büyük kümelere dönüştürür. Bu algoritmada dairesel olmayan şekilleri bulmakta çok verimli değildir. Ayrıca sapan veriden çok etkilenir. Bunun yanında yapısı gereği büyük veri setleri için çok başarılı değildir. CluStream algoritmasının gelişmiş bir versiyonu olan StreamSamp (Csernel, Clerot ve Hébrail) algoritması geliştirilmiştir. Bu yaklaşım temel anlamda verimli bellek kullanımı ile ön plana çıkmaktadır. StreamSamp'te sabit boyutlu bellek kullanımı benimsenmiştir. İki çeşit hafızalama yoluna gidilmektedir. Kısa boyutlu ve daha uzun boyutlu hafızalama. Kısa boyutlu hafızalama küçük boyutlu periyodik hafızalama yaparken, uzun boyutlu hafızalama büyük boyutlu periyodik hafızalama yapmaktadır. StreamSamp akan veriyi hızlı bir şekilde özetleme yeteneğine sahip olmasına rağmen kümeleme başarısı zamana bağlı olarak azalmaktadır.




Şekil 2.6. BIRCH genel yapısı (Zhang ve diğerleri, 1996)

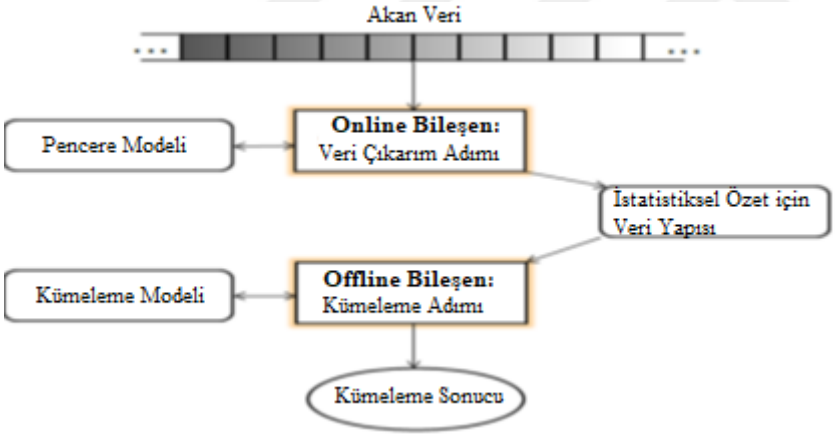
ClusTree (Kranen ve diğerleri, 2011) kümeleme işlemini yine çevrimiçi ve çevrimdışı olmak üzere iki aşamada gerçekleştirir. Çevrimiçi kısım micro-clusterları bulmak için kullanılır. Micro-clusterlar hiyerarşik yapıda birleştirilir. Böylece herhangi bir adımda çevrimdışı komponent uygulanabilir. Sistem kendinden uyarlamalıdır. Bu nedenle herhangi bir anda sistem sonuç üretebilmektedir.

HPStream (Charu, Jiawei, Jianyong ve Philip, 2004), Fading Cluster Structure (FCS) kullanarak zamana bağlı olarak eski verilerin ağırlığını azaltır. CluStream algoritmasının gelişmiş bir versiyonudur. ClueStream algoritmasına göre avantajı çok boyutluluğu desteklemesidir. Çok boyutlu verilerde boyutların bir alt kümesini alır. Nitelik sayısı her küme için farklı olabilir. Nitelik sayısı güncellenebilmektedir. Yeni gelen verilere daha fazla önem verilmesi gereken uygulamalar için uygundur. Bu yaklaşım sensör network uygulamaları için oldukça faydalı bir yaklaşımdır. Çok boyutluluğu desteklemesine rağmen çok boyutluluk konusunda optimum bir sonuç verememektedir. Küme şekillerini belirlemekte problem çıkmaktadır. Ancak küme şekillerini belirlemekte problem çıkmaktadır. İyi performans için verinin tamamı hakkında yeterli bilgiye sahip olmak gerekir. Bu da akan veri için her zaman mümkün değildir.

DUCstream (Gao, Li, Zhang ve Tan, 2005) veriyi birbiri ile örtüşmeyen gridlere ayırır ve

veriyi chunk denilen parçalara ayırır. Her bir chunk M tane veriden oluşur. Gridler noktalar halinde haritalanır ve depth first search algoritması uygulanarak kümeler oluşturulur. Kümeleri bir grafın parçaları gibi birbirine bağlar. Kümeleme açısından oldukça elverişli bir algoritmadır. Ancak veriyi gridlere ayırmak için veri hakkında yeterli bilgiye sahip olmak gerekir.

DenStream (Cao ve diğerleri), yoğunluk tabanlı bir kümeleme yaklaşımıdır. Bu yaklaşımda DBSCAN algoritmasının geliştirilmiş bir versiyonu ve micro-clusterlar kullanılır. Şekil 2.7'de de görüldüğü gibi çevrimiçi ve çevrimdışı olmak üzere iki bölümden oluşur. Çevrimiçi bölümde veriler micro-clusterlar şeklinde temsil edilir. Çevrimdışı bölümde micro-cluster yapısındaki veri DBSCAN algoritması ile kümelenir. Bu algoritmanın en büyük avantajı sapan verileri tespit edebilmesidir. rDenStream (Liu, Huang, Guo ve Chen, 2009) algoritması da DenStream algoritması gibidir. rDenStream algoritmasının en önemli farklı drop edilen verilere ikinci bir şans vermesidir.



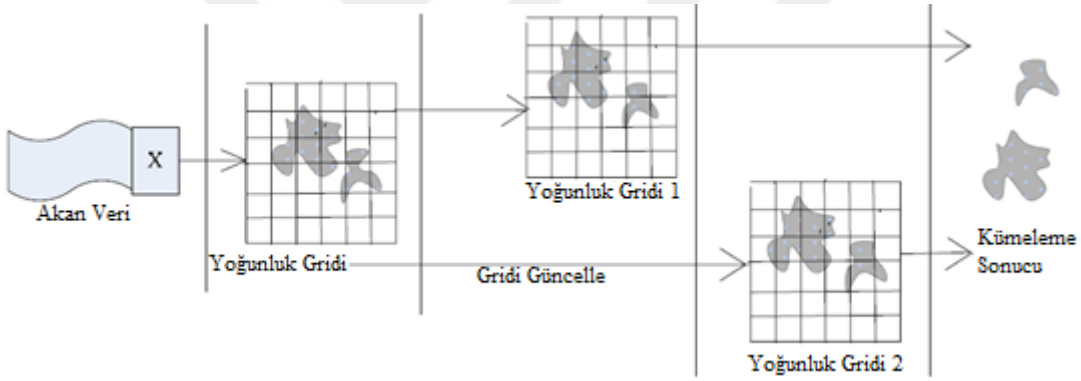

Şekil 2.7. DenStream algoritmasına ait framework (Cao ve diğerleri)

E-Stream, beş tip işlem gerçekleştirmektedir (Udommanetanakit, Rakthanmanon ve Waiyamai, 2007): Yeni bir kümenin ortaya çıkması, var olan bir kümenin yok edilmesi, büyük bir kümenin bölünmesi, iki benzer kümenin birleştirilmesi, kümenin karakteristiğinin değiştirilmesi. Fading cluster yapısını ve histogram kullanır. Bu nedenle performansı HPStream algoritmasından daha iyidir. Ancak kullanıcının pek çok parametreyi tanımlamasına ihtiyaç duyar.

SE-Stream (Chairukwattana, Kangkachit, Rakthanmanon ve Waiyamai, 2013), E-Stream algoritmasının gelişmiş bir versiyonudur. Boyut problemini de ele alan bir algoritmadır.

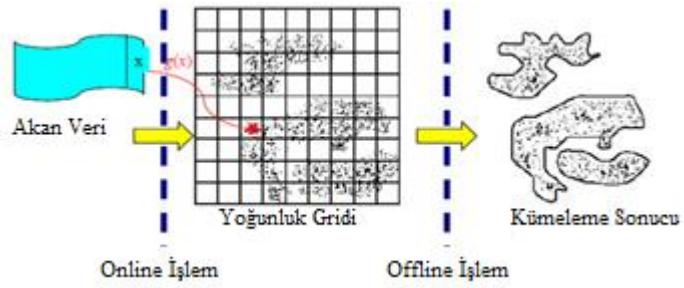
Amaç çok boyutlu veriler için performansı arttırmak. Boyut indirgeme işlemi yapılır. Bunu yaparken de sonuçla en fazla alakası olan nitelikler seçilir. E-Stream algoritmasında MergeOvelapClustering ve LimitMaximumCluster işlemleri çok zaman almaktadır. SE-Stream algoritmasında bunlar optimize edilir.

DD-Stream (Jia, Tan ve Yong, 2008), yoğunluk ve grid tabanlı yaklaşımları birleştiren bir yaklaşımdır. n-boyutlu veriyi gridlere ayırır. Şekil 2.8'de de görüldüğü gibi gelen stream datayı gridlere ayırır ve eigenvector (özvevektör) ile update eder. Eigenvector grid merkezinin koordinatlarını, gridin en son update edildiği zamanı, gridin grid merkezinden silindiği zamanı ve en son grid yoğunluğu gibi veri gruplarını içerir. Eigenvector ile gridin yoğun mu yoksa seyrek bir grid mi olduğuna karar verilir. Yoğun gridlere ait kümeleri bulmak için yoğunluk tabanlı kümeleme yaklaşımı kullanılır.




Şekil 2.8. DD-Stream algoritması (Jia ve diğerleri, 2008)

D-stream (Tu ve Chen, 2009), yoğunluk tabanlı grid kümeleme algoritmasıdır. Şekil 2.9'da da görüldüğü gibi veriyi gridlere böler. İki aşamadan oluşur. Çevrimiçi bölümde gelen veri gridlere göre haritalanır. Çevrimdışı aşamada her grid için yoğunluk hesaplayarak veriyi atar. Son olarak grid yoğunluğuna göre bir kümeleme yapar. Eski gridlerin ağırlığını azaltmak için zamana bağlı olarak fading fonksiyonu uygular. Eğer belirlenen bir gridin yoğunluk değeri eşik değerinin altına düşerse ve yeni veriler eklenmez ise drop edilir. Boyut (dimension) arttıkça grid sayısı exponential artar, bu nedenle çok boyutlu problemler için uygun değildir.




Şekil 2.9. D-Stream kümeleme (Tu ve Chen, 2009)

HUE-Stream (Meesuksabai, Kangkachit ve Waiyamai, 2011), E-Stream algoritmasının geliştirilmiş bir versiyonudur. Kategorik niteliklerde belirsizliği ortadan kaldırmak için iki objenin uzaklık fonksiyonunu olasılık dağılımı ile kullanır. Küme yapısında değişiklik yapılıp yapılmayacağına adı geçen uzaklık fonksiyonu ile karar verir. Bu fonksiyon ile bir kümenin ikiye ayrılmasına veya iki kümenin birleştirilmesine karar verir.

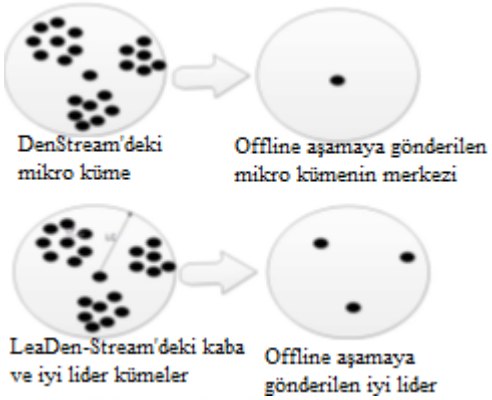
Rough set (D. T. Yogita, 2012), bu yaklaşımda belirsizlik Rough set teorisine göre ele alınır. Kümeleri iki grup olarak ele alır: Lower approximation ve Upper approximation. Lower approximation'a ait veriler kesin bir yargı ile bir kümeye atanırken Upper approximation'a ait objeler kesin bir yargıyla atanmaz. Sapan verilere karşı dirençli bir yapıya sahiptir.

STREAMKM++ (Ackermann ve diğerleri 2012), k-means ve Öklid uzaklık tabanlı bir akan veri kümeleme algoritmasıdır. Eğer küme merkezlerinin sayısı fazla ise BIRCH algoritmasından daha iyi sonuç vermektedir. Ancak performans açısından bakıldığı zaman BIRCH algoritmasına göre daha yavaş bir algoritmadır.

HDDStream (Ntoutsi, Zimek, Palpanas, Kröger ve Kriegel, 2012), HPStream benzeri bir algoritmadır. HDDStream üç aşamadan oluşur. İlk aşamada micro-clusterlar oluşturulur. Sonra çevrimiçi bölümde sapan veriler tespit edilir ve geri kalan veri kümelenir. Çevrimdışı bölümde ise elde edilmiş kümeler daha büyük kümelere dönüştürülür. HPStream ile HDDStream arasındaki fark küme sayısının sabit olmasıdır. HDDStream algoritmasında küme sayısı zamana bağlı olarak değişir.

LeaDen-Stream (Leader Density-based clustering algorithm over evolving data Stream) (Amini ve Wah, 2013), yoğunluk tabanlı bir akan veri kümeleme yaklaşımıdır. Şekil 2.10'da da görüldüğü gibi bu yaklaşımın yoğunluk tabanlı diğer çalışmalardan farkı oluşturulmuş olan micro-clusterların içindeki verilerin dağılımını da göz önünde bulundurmasıdır. Burada

micro-cluterların içinde verinin yoğunlaştığı yerde bulunan veri lider olarak seçilir. Çevrimdışı bölümde lider noktalar kullanılarak sonuç kümeleri oluşturulur.




Şekil 2.10. LeaDen-Stream algoritmasına ait mini micro ve micro lider kümeler (Amini ve Wah, 2013)

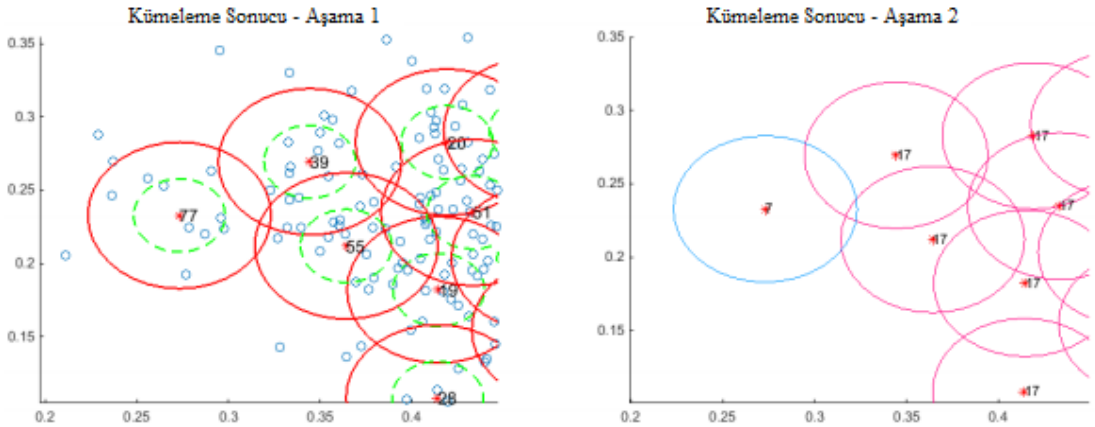
CL-Ant (N. Masmoudi, Azzag, Lebbah ve Bertelle, 2014) ve CL-AntInc (Nesrine Masmoudi, Azzag, Lebbah, Bertelle ve Jemaa, 2016) algoritmaları karınca kolonisini örnekleyen akan veri kümeleme yaklaşımlarıdır. Her iki algoritma da karınca kolonisi algoritmasını modellemektedir. Karıncalar salgıladıkları feromen maddesine göre yiyecek ve yuvaları arasında yol bulurlar. Feromen miktarının en fazla olduğu yol en çok kullanılan yoldur, bu aslında en kısa yol problemi için üretilmiş bir çözümdür. CL-Ant ve CL-AntInc algoritmalarında veriler karınca olarak temsil edilmektedir. Karınca kolonilerinin her biri de küme olarak kabul edilmektedir. Çok sayıda yuva (küme) ve çok sayıda karınca olduğunu varsaydığımızda salgılanmış feromen miktarı birbirine karışacak ve karınca kendi yuvasını bulmakta zorlanacaktır. Bu durumda bir karınca bütün yuvalara teker teker bakarak hangi yuvanın kendi yuvası olduğuna karar verecektir. Algoritma başlangıçta k-means algoritması uygulayarak kümeleme yapmaktadır. Daha sonra dinamik bir graf yapısı oluşturularak dinamik bir kümeleme yapılmaktadır.

Akan veri kümeleme için geliştirilmiş bir başka çalışma, geliştirilmiş yoğunluk tabanlı akan veri kümeleme algoritmasıdır (Mousavi ve Abu Bakar, 2015). Yoğunluk tabanlı kümeleme algoritmaları özellikle veriye has şekilleri bulmakta ve gürültüleri tespit etmekte oldukça başarılı algoritmalardır. Ancak yoğunluk tabanlı algoritmaların en önemli eksikliği lokal maksimuma takılmalarıdır. Yapılan çalışma bu probleme çözüm üretmek amacıyla geliştirilmiştir. MinPts adı verilen parametreyi kullanarak çevrimdışı aşamayı olumlu yönde geliştirmektedir. Bu algoritmada parçala ve fethetin tersi gibi bir yaklaşım söz konusudur.

Önce veriyi tek bir küme olarak ele alır ve her seferinde en uzak elemanı silerek işleme devam eder.

HSDStream (High Speed and Dimensions data stream clustering) (I. Ahmed, Ahmed ve Shahzad, 2015), çok boyutlu verileri kümelemek amacıyla gerçekleştirilmiş bir algoritmadır. Akan veri çok boyutlu ise bu veriyi kümelemek normalden daha fazla zaman alacaktır. Veri sürekli aktığında zamana karşı bir yarış söz konusudur. HSDStream üç aşamadan oluşmaktadır. Birinci aşamada veri sabit boyutlu micro-clusterlara(core-mc) dönüştürülür. İkinci aşama olan çevrimiçi aşamada sapan veriler tespit edilir. Çevrimdışı bölümde ise sonuç kümeleri oluşturulur. Bu algoritmanın HDDStream algoritmasından farkı çok boyutlu veriyi işleyebilmesidir. Yapılan deneysel çalışmalar HSDStream algoritmasının HDDStream algoritmasından daha iyi sonuçlar verdiğini ortaya koymuştur.

CODAS (Clustering Online Data-streams into Arbitrary Shapes) (R. Hyde ve Angelov, 2015) Şekil 2.11'de de görüldüğü gibi yoğunluk tabanlı bir akan veri kümeleme yaklaşımıdır. CODAS'ın getirdiği en önemli yenilik tamamen çevrimiçi çalışmasıdır. Veriyi çevrimiçi bir şekilde işler, çevrimdışı bileşeni bulunmamaktadır. Çevrimiçi olarak veriyi micro-clusterlara böler, sonra bu mikro-clusterlar birleştirilerek asıl kümeler elde edilir. Çevrimdışı bileşeni olmadığından hafızaya gerek duymaz. Sistem çevrimiçi çalıştığından kümeleme işlemini hızlı bir şekilde yapması gerekir. Bunu gerçekleştirmek için kümelemede hyper-spherical micro-cluster denilen yapıyı kullanır. Sapan verileri tespit etmek için bir eşik değer kullanılır. Bu kümeleme yaklaşımının en büyük eksiği çevrimiçi çalışmasından dolayı çok hızlı akan veriyi işlemede oluşabilecek problemler ve son gelen verileri daha önemli kılacak bir yapıya sahip olmamasıdır.




a. Çekirdek mikro-kümeler b. Birleştirilmiş mikro-kümeler

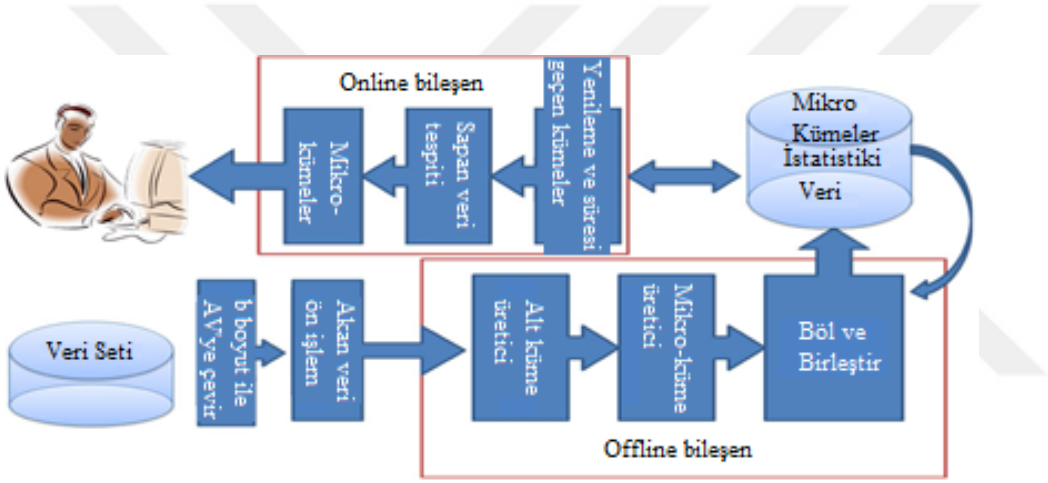
Şekil 2.11. CODAS algoritmasına ait mikro ve birleştirilmiş kümeler (R. Hyde ve Angelov, 2015)

SOC (Skeleton-based Çevrimiçi Clustering) (Choromanski, Kumar ve Liu, 2015), temelde hızlı bir şekilde ve çevrimiçi kümeleme yapmayı amaçlayan bir yaklaşımdır. k-means ve k-medoid gibi yaklaşımların aksine herhangi bir varsayımda bulunmaz. Kümeler skeleton denilen parçalar ile temsil edilir. Skeleton denilen parçalar verinin yoğunluğa göre ağırlıklandırılmış şeklidir. Hızlı bir şekilde kümeleme yapmak için her yeni veri gelişinde kümeler update edilir. Önerilen yaklaşım kümeleri otomatik bir şekilde tespit ederken sapan verileri de tespit etme yeteneğine sahiptir. Mevcut yaklaşımlarla karşılaştırıldığında daha iyi sonuçlar verdiği tespit edilmiştir.

Sosyal medyada akan veriden başlıkları tespit edip çevrimiçi bir şekilde bunları kümeleyen bir yaklaşım da mevcuttur (Popovici, Weiler ve Grossniklaus, 2014). Bu yaklaşım temel anlamda DenStream algoritmasını temel almaktadır. Çünkü DenStream algoritması kategorik verilere kolaylıkla uygulanabilmektedir. DenStream algoritmasına birkaç özellik eklenmiştir. Burada kümeleme işleminde DBSCAN algoritmasından da faydalanılmaktadır.

STREAMLEADER (Merino, 2015), CODAS algoritması gibi çevrimdışı bölüme gerek duymadan veriyi çevrimiçi olarak kümeleyen bir kümeleme yaklaşımıdır. Bu yaklaşımda kullanıcıdan sadece bir parametreyi belirlemesi beklenmektedir. Sistem MOA (Massive Çevrimiçi Analysis) (Yeni Zelanda'nın Waikato Üniversitesi'nin akan veri kümeleme için geliştirdiği açık kaynak kodlu bir frameworktur) platformuna adapte edilmiştir. Geliştirilen yöntem CluStream, DenStream ve ClusTree algoritmaları ile karşılaştırıldığında daha iyi sonuçlar verdiği tespit edilmiştir.

DCSTREAM (Khalilian, Mustapha ve Sulaiman, 2016), STREAM gibi divide and conquer ve k-means tabanlı bir kümeleme algoritmasıdır. Büyük verileri kümelemek için geliştirilmiştir. Veriler w boyutunda parçalara ayrılır. $w_i$ penceresine alınmış veri $w_i$+1 süresi içerisinde yok edilir. Yani geriye dönüş w süresince olabilir. Şekil 2.12'de de görüldüğü gibi biri çevrimiçi ve biri çevrimdışı olmak üzere iki bileşenden oluşur. Veri çevrimiçi bölümden önce veri temizleme, nitelik seçme, nitelik azaltma ve veri dönüşümü gibi işlemler gerçekleştirilir. Çevrimiçi bölümde veriler micro-clusterlara bölünür. Burada merge, split(küme birleştirme ve ayırma) k-means ile yapılır. Çevrimdışı bölümde $w_i$+1 süresi sonunda $w_i$ yok olacağından son gelen verilere önem verilmiş olarak kümeleme yapılmış olur.




Şekil 2.12. DCSTREAM algoritmasına ait framework (Khalilian ve diğerleri, 2016)

DBSTREAM (Hahsler ve Bolaños, 2016) mikro cluster yapısını kullanırken micro clusterlar arsındaki verileri de işleme alır. Çoğu akan veri kümeleme yaklaşımı çevrimiçi ve çevrimdışı iki aşamadan oluşur. Çevrimiçi kısımda veri özetlenirken özeti alınmış veriler yoğunluğa göre ağırlıklandırılmaktadır. DBSTREAM veriyi micro cluster'lara bölerken sadece micro cluster (MC)'lar arsı mesafeyi almaz aynı zamanda orijinal veriler arasındaki yoğunluk bağlantılarını da alır.

FEAC-Stream (J. d. A. Silva, Hruschka ve Gama, 2017) küme sayısında sınırlama getirmeyen ve veriyi özetlerken verinin değişim noktalarını tespit etmeye çalışan böylece performansı arttıran bir akan veri kümeleme yaklaşımıdır. Yani oluşturulmuş kümeleri değiştirmeyi belirli şartlara bağlar aksi durumda kurulmuş olan yapı devam ettirilmektedir.

DPStream (Xu, Wang, Li, Deng ve Gou, 2017) algoritması DPClust yapısını kullanan bir hiyerarşik veri kümeleme yaklaşımıdır. Verileri hiyerarşik bir yapıda birleştiren DPStream bunu yaparken verileri yoğunluğa bağlı olarak en yakın yüksek yoğunluklu olarak birbirine bağlar. DPStream algoritması ile ilgili detaylı açıklama aşağıda yapılmıştır.

CEDAS (Richard Hyde, Angelov ve MacKenzie, 2017) graf yapısını kullanan bir akan veri kümeleme yaklaşımıdır. Verileri ilk olarak mikro küme yapısında birleştiren daha sonra bu mikro kümelerden yakınlık derecesine bakarak birleştiren ve mikro kümeleri oluşturan çevrimiçi çalışan ve farklı şekildeki kümeleri tespit etme yeteneğine sahip bir yaklaşımdır. Her veri gelişinde graf yapısı kontrol edilmekte ve eğer gerek varsa bu yapı güncellenmektedir. CEDAS ile ilgili detaylı açıklama aşağıda yapılmıştır.

LLDStream (Laohakiat, Phimoltares ve Lursinsap, 2017), DenStream ve DBSCAN algoritmalarını temel alan yoğunluk tabanlı bir algoritmadır. LLDStream çevrimiçi ve çevrimdışı olarak iki aşamadan oluşur. Çevrimiçi aşamada gelen verinin atanacağı mikro-kümeyi belirler. Çevrimdışı aşamada ise kullanıcıya mikro-kümelerden oluşan makro kümeler sunulur. LLDStream boyut indirgemeyi LDA (Linear Discriminant Analysis)'dan faydalanarak yapar. Çok boyutluluğu ve farklı şekildeki kümeleri tespit edebilmesi avantajı olarak öne çıkmaktadır.

Bir diğer önerilen akan veri kümeleme yaklaşımı Xinxin ve ark. (X. Shao, Zhang ve Meng, 2018) sapan verileri tespit edebilen, en yakın komşu yaklaşımından güç alan yoğunluk ve grid tabanlı algoritmadır. Bu yaklaşım kayan pencere veri özetleme yaklaşımını kullanır. Özeti alınmış veri bir graf yapısında yapılandırılır. En yakın komşu yaklaşımına göre bir birine yakın olan veriler aynı kümenin elemanları olarak tanımlanır. Farklı şekildeki kümeleri ve sapan verileri tespit edebilmesi en güçlü yönleri olarak öne çıkarken, parametrelerin optimum belirlenmesi ihtiyacı zayıf tarafı olarak öne çıkmaktadır.

SyncTree (J. Shao, Tan, Gao, Yang, Plant ve Assent, 2018) algoritması, hiyerarşi ve yoğunluk tabanlı ve evrimsel değişimi destekleyen bir akan veri kümeleme algoritmasıdır. SyncTree, mikro-küme yapısını kullanan ve etkin bir veri özetleme yaklaşımına sahip bir algoritmadır. Performans açısından tatmin edici olmasına rağmen çok sayıda parametrenin tanımlanmasına ihtiyaç duymaktadır.

evoStream (Carnein ve Trautmann, 2018) algoritması yoğunluk tabanlı bir akan veri kümeleme yaklaşımıdır. Akan verinin evrimsel değişimini desteklemektedir. Evrimsel optimizasyon yöntemini sistemin boş olduğu zamanlarda yaparak daha etkin bir kaynak kullanımı sunmaktadır. evoStream algoritmasının en büyük dezavantajı çok sayıda parametre kullanıldığından en iyi parametreleri tespit etmek güçtür.

StreamSW (Reddy ve Bindu, 2018) algoritması DBSCAN tabanlı çalışan yoğunluk tabanlı bir akan veri kümeleme algoritmasıdır. DBSCAN algoritmasını farklı şekildeki kümeleri tespit edebilmek için kullanmaktadır. Yoğunluk tabanlı bir algoritma olduğundan sapan verileri tespit etmekte ve farklı şekildeki kümeleri belirleme konusunda oldukça başarılı bir algoritmadır. Ancak çok boyutlu verileri işlemede problem yaşamaktadır.

BOCEDS (M. Ahmed, 2019), yoğunluk tabanlı algoritmaların karşılaştığı bir problem olan sabit mikro küme yarıçapını ihtiyaca göre güncelleme yeteneğine sahip bir akan veri kümeleme algoritmasıdır. Başarılı bir kümeleme yeteneğine sahip olması, akan verinin değişken yapısına uyum sağlayabilmesi ve farklı şekildeki kümeleri tespit edebilme yeteneğine sahip olması güçlü yanları iken fazla kaynak kullanması ise zayıf yönü olarak göze çarpmaktadır.

ChronoClust (Putri, Read, Koprinska, Singh, Röhm, Ashhurst ve King, 2019) yine yoğunluk tabanlı ve farklı şekilli kümeleri tespit edebilen bir akan veri kümeleme algoritmasıdır. ChronoClust algoritması çok boyutlu ve zaman damgalı verileri işlemeye yönelik önerilmiş bir algoritmadır. Ayrıca akan verinin değişken yapısına uyum sağlayabilmesi de yine önemli bir özelliğidir. Ancak başarılı bir kümeleme yapabilmesi için kullanılan çok sayıdaki parametrenin doğru belirlenmesine ihtiyaç duymaktadır.

### 2.7.2. Değişken tabanlı akan veri kümeleme algoritmaları

Değişken tabanlı akan veri kümeleme yaklaşımlarında aynı özelliklere sahip verilerin aynı kaynaktan aktığı düşünülür. Örneğin hasta takip sistemlerinde hastaya ait veriler aynı özelliklere sahiptir, kalp atım sayısı, kan basıncı, vücut ısısı vs. Bu alanda yapılmış başlıca çalışmaları şu şekilde sıralayabiliriz:

SPE-Cluster (Keogh, Chu, Hart ve Pazzani, 2001), Oto-regresyon modelleme tekniğini

kullanarak akan veriler arasındaki korelasyonu hesaplar. Bunun için akan veriden birbiri ile alakalı nitelikleri bulmak için frekans spektrumunu bulur. Çevrimiçi ve çevrimdışı iki bölümden oluşur. Çevrimiçi bölüm sliding window ile akan verilerin spektral komponenetlerini hesaplar. Çevrimdışı bölüm ise Dinamik-kmenas kullanarak akan veriyi kümeler. SPE-Cluster dinamik yapısı ile concept evolution (kümeleme adaptasyonu) ve küme sayısı problemlerine çözüm üretmektedir.

Jurgen ve ark. Paralel akan veriyi kümelemek için değişken tabanlı bir yaklaşım önermişlerdir (Beringer ve Hüllermeier, 2006). Veriyi özetlemek için veriler arasındaki uzaklığı ve Discrete Fourier Transform (DFT) katsayısını kaydeder. Akan veriyi işlemek için sliding window yapısını kullanır. Kümeleme işlemini gerçekleştirmek için k-means kümelemeyi kullanır. Concept evolution gerçekleştirmek için küme sayısını birer birer artırır ve birer birer azaltır. Küme sayısını artırırken merkeze uzak veriyi yeni küme olarak tanımlar. Küme sayısını azaltırken var olan bir kümenin elemanlarını diğer kümelere atar.

ODAC (The Çevrimiçi Divisive-Agglomerative Clustering) (Rodrigues, Gama ve Pedroso, 2008), concept evolution'ı desteklemek için ayırıcı ve kümeleyici bir yapıdadır. Top-down bir yapıyı destekler, hiyerarşik bir ağaç yapısındadır. Veriler arasındaki uzaklığı bulmak için korelasyona benzeyen bir hesaplama yaklaşımı vardır. Hiyerarşik ağaçta her bir node'u (burada küme) uzaklığa bakarak ayırır. Aynı şekilde uzaklık korelasyonuna göre iki node'u birleştirir. Var olan kümelerin çapına bakarak kümelerin birleştirilmesine veya ikiye ayrılmasına karar verir. Bu işlem zamana bağlı olarak periyodik bir şekilde yapar.

POD-Clus (Probability and Distribution-based Clustering) (Chaovalit ve Gangopadhyay, 2009), model tabanlı bir akan veri kümeleme yaklaşımıdır. Bu yaklaşım hem örnek hem de değişken tabanlı yaklaşımları destekler. Örnekleme için küme özetini kullanır. Bunun için mean, standart sapma ve her küme için nokta sayısına bakar. Kümelerin ayrıştırılması, iki kümenin birleştirilmesi, yeni bir küme oluşturulması veya var olan bir kümenin yok edilmesini destekler. Kümeleme başarısı tatmin edici olmasına rağmen işlem performansı kötüdür.

COMET-CORE (Clustering Over Multiple Evolving sTreams by CORrelation and Events) (Yeh, Dai ve Chen, 2007), çoklu akan veriyi çevrimiçi manada kümelemek için geliştirilmiş bir yaklaşımdır. Benzerlik ölçütü olarak ağırlıklandırılmış korelasyon kullanır. ODAC'ın

aksine COMET-CORE veriyi periyodik olarak kümelemez. Belirli bir durum oluştuğunda kümeleri böler veya birleştirir. Akan veriler arasındaki ağırlıklandırılmış korelasyonu sürekli günceller.

Çizelge 2.2'de akan veri kümeleme alanında yapılan başlıca çalışmaların karşılaştırılması görülmektedir (Şenol ve Karacan, 2018). Karşılaştırma yapılan algoritmalar avantaj ve dezavantajları, kullandıkları parametre sayısı ve evrimsel değişimi destekleyip desteklemedikleri açısından karşılaştırılmıştır.

Çizelge 2.2. Akan veri kümeleme alanındaki algoritmaların bazılarının karşılaştırması

| Algoritma | Yıl | Parametreler | Avantajlar | Dezavantajlar | Evrimsel değişim desteği var mı? | Kümeleme türü |
|---|---|---|---|---|---|---|
| SPE-Cluster (Keogh, Chu, Hart ve Pazzani, 2001) | 2001 | Sliding ve temel pencerelerin tanımlanması gerekir | Küme sayısı belirlemede ve zamana bağlı küme adaptasyonunda avantajlıdır | Dairesel olmayan kümeleri bulmakta problemli | Evet | Parçalamalı |
| STREAM (O'Callaghan, Mishra, Meyerson, Guha ve Motwani, 2002) | 2002 | Küme sayısı | Hızlı çalışan bir algoritma | Küme sayısını sınırlaması | Hayır | Parçalamalı |
| CluStream (Aggarwal, Han, Wang ve Yu, 2003) | 2003 | Küme sayısı ve zaman penceresi | Özetleme sayesinde kullanıcıya esneklik sağlar | Dairesel olmayan kümeleri tespit etmekte problemli<br>Sapan veriden etkilenir<br>Büyük verilerde problemli | Evet | Parçalamalı & Hiyerarşik |
| HPStream (Charu, Jiawei, Jianyong ve Philip, 2004) | 2004 | Max küme sayısı, ortalama boyut değeri | Çok boyutlu verileri destekler<br>Yeni gelen verilerin ağırlığı daha fazla | Dairesel olmayan kümeleri bulmakta problemli | Evet | Parçalamalı & Hiyerarşik |
| DUCstream (Gao, Li, Zhang ve Tan, 2005) | 2005 | Grid hücrelerinin yoğunluk eşik değeri | Dairesel olmayan şekilleri bulabiliyor | Tüm verilerin ağırlığı aynı | Evet | Yoğunluk & Grid tabanlı |
| DenStream (Cao, Estert, Qian ve Zhou, 2006) | 2006 | Küme yarıçapı eşik değeri<br>Data fading eşik değeri<br>Sapan veri eşik değeri<br>Küme ağırlığı<br>Decay factor | Farklı şekillerdeki kümeleri bulabilir<br>Sapan verileri tespit edebilir | Sapan verileri bulma işlemi performansı düşürüyor<br>Split ve merge işlemlerini yapmıyor | Evet | Yoğunluk tabanlı |

Çizelge 2.2. (Devam) Akan veri kümeleme alanındaki algoritmaların bazılarının karşılaştırması

| Algoritma | Yıl | Parametreler | Avantajlar | Dezavantajlar | Evrimsel değişim desteği var mı? | Kümeleme türü |
|---|---|---|---|---|---|---|
| E-Stream (Udommanetanakit, Rakthanmanon ve Waiyamai, 2007) | 2007 | Maksimum küme sayısı, radius factor, stream hızı, silme eşik değeri, aktif olma eşik değeri, birleştirme eşik değeri | Kümelerin zamana bağlı dönüşümünü destekler<br>Zamanla split ve merge işlemleri gerçekleşir | Dairesel olmayan kümeleri bulmakta problemli, çok sayıda parametrenin tanımlanması gerekir | Evet | Hiyerarşik |
| D-Stream (Tu ve Chen, 2009) | 2007 | Decay factor<br>Grid yoğunluk eşik değeri<br>Data fading oranı | Yoğunluğa bakarak kümeleri gerçek zamanlı olarak tespit edebiliyor<br>Zamana bağlı olarak kümelerin dönüşümünü destekliyor | Algoritmanın zaman karmaşıklığı pek çok değişkenden etkilenir | Evet | Yoğunluk & Grid tabanlı |
| DD-Stream (Jia, Tan ve Yong, 2008) | 2008 | Grid hücrelerinin yoğunluk eşik değeri, yoğunluk fading oranı gibi parametreler | Farklı şekillerdeki kümeleri bulabilir<br>Kümeleme başarısı tatmin edici | Çok boyutluluğu destekleme | Evet | Yoğunluk & Grid tabanlı |
| ODAC (Rodrigues, Gama ve Pedroso, 2008) | 2008 | Kümeleri bölme eşik değeri | Kümeleri bölme ve birleştirme işleminde başarılı | Dairesel olmayan kümeleri bulmakta problemli | Evet | Hiyerarşik |
| POD-Clus (Chaovalit ve Gangopadhyay, 2009) | 2009 | Küme bölme ve birleştirme eşik değerleri, data decay oranı | Dairesel olmayan kümeleri tespit edebilir | İşlem performansı kötü | Evet | Model tabanlı |
| COMET-CORE (Yeh, Dai ve Chen, 2007) | 2009 | Korelasyon eşik değeri, küme bölme ve birleştirme eşik değerleri | Çok boyutluluğu destekler | Dairesel olmayan kümeleri bulmakta problemli | Evet | Hiyerarşik |

Çizelge 2.2. (Devam) Akan veri kümeleme alanındaki algoritmaların bazılarının karşılaştırması

| Algoritma | Yıl | Parametreler | Avantajlar | Dezavantajlar | Evrimsel değişim desteği var mı? | Kümeleme türü |
|---|---|---|---|---|---|---|
| HUE-Stream (Meesuksabai, Kangkachit ve Waiyamai, 2011) | 2011 | Veri decay oranı ve E-Stream'in diğer parametreleri | Kümelerin zamana bağlı dönüşümlerini destekler, performansı E-Stream'e göre daya iyi | Dairesel olmayan kümeleri bulmakta problemli, çok sayıda parametrenin tanımlanması gerekir | Evet | Hiyerarşik |
| Rough set tabanlı (Yogita, 2012) | 2012 | Sapan veri eşik değeri, eski verileri silme eşik değeri | Sapan verilere karşı dirençli | Dairesel olmayan kümeleri bulmakta problemli | Evet | Rough set tabanlı parçalamalı |
| HDDStream (Ntoutsi, Zimek, Palpanas, Kröger ve Kriegel, 2012) | 2012 | Küme yarıçapı, küme ağırlığı, sapan veri eşik değeri, decay factor | Farklı şekildeki kümeleri tespit edebilir, çok boyutlu verileri destekler, sapan verileri tespit edebilir, | Algoritmanın performansı düşük | Evet | Yoğunluk |
| LeaDen-Stream (Amini ve Wah, 2013) | 2013 | MMLC ve MMLCweight, MMLC ve MMLCCenter, MMLC ve MMLCradius | Kümeleme başarısı makul<br>Sapan verilere karşı dirençli, performansı tatmin edici | Çok boyutlu veriyi desteklemiyor | Evet | Yoğunluk |
| SE-Stream (Chairukwattana, Kangkachit, Rakthanmanon ve Waiyamai, 2013) | 2013 | Silme eşik değeri, radius factor, veri akış hızı gibi çok sayıda parametre tanımlanmalıdır | E-Stream algoritmasına göre performans daha iyi, algoritma optimize edilmiştir. | Çok sayıda parametrenin tanımlanması gerekir | Evet | Hiyerarşik |
| HSDStream (I. Ahmed, Ahmed ve Shahzad, 2015) | 2015 | Pencere genişliği, yarıçap eşik değeri, sapan veri eşik değeri, veri sayısı eşik değeri, varyans eşik değeri, boyut indirgeme eşik değeri | Sapan verilere karşı dirençli, çok boyutlu verileri destekler, performans tatmin edicidir | Çok sayıda verinin tanımlanması gerekir | Evet | Yoğunluk |

Çizelge 2.2. (Devam) Akan veri kümeleme alanındaki algoritmaların bazılarının karşılaştırması

| Algoritma | Yıl | Parametreler | Avantajlar | Dezavantajlar | Evrimsel değişim desteği var mı? | Kümeleme türü |
|---|---|---|---|---|---|---|
| SOC (Choromanski, Kumar ve Liu, 2015) | 2015 | Küme bölme ve birleştirme eşik değerleri | Sapan verileri tespit edebilir, kümeleme başarısı tatmin edici | Küme başarısı için belli sayıda verinin ulaşmış olması gerekir | Evet | Model tabanlı |
| pcStream (Mirsky, Shapira, Rokach ve Elovici, 2015) | 2015 | Drift eşik değeri, maksimum drift boyutu, varyans yüzdesi ve modeldeki veri sayısı | Kümelerin çakışmasını tespit edebilmektedir. | Sapan verilerden etkilenmekte ve başarı oranı düşmektedir. | Evet | Model tabanlı |
| DCSTREAM (Khalilian, Mustapha ve Sulaiman, 2016) | 2016 | Küme bölme ve birleştirme eşik değerleri, sapan veri eşik değeri | STREAM algoritmasına göre çok daha başarılı, Çok boyutlu veriyi destekler | Dairesel olmayan kümeleri tespit etmek problemli | Evet | Parçalamalı |
| DBSTREAM (Hahsler ve Bolaños, 2016) | 2016 | Fading factor, yarıçap, α vs. | Micro-clusterlar arasında kalan verileri de kullanır buna göre gerekirse kümeler güncellenmektedir. | MC'lar arasındaki verileri de işlediği için zaman karmaşıklığı bir miktar artmaktadır. | Evet | Yoğunluk |
| FEAC-Stream (Silva, Hruschka ve Gama, 2017) | 2017 | InitSize, 10 k-means iterasyonu vs. | Küme sayısının ön tanımlı olmasına gerek yok | Eski oluşturulmuş kümeler silinir | Evet | Yoğunluk |
| DPStream (Xu, Wang, Li, Deng ve Gou, 2017) | 2017 | Lokal yarıçap, global yarıçap, işleme alınan veri boyutu vs. | Tamamıyla çevrimiçi ve farklı şekilleri destekler. Ayrıca çok hızlı çalışan bir algoritmadır. | Çok sayıda parametre var ve parametrelerin çok doğru seçilmesi gerekir. | Evet | Hiyerarşik ve Yoğunluk |

Çizelge 2.2. (Devam) Akan veri kümeleme alanındaki algoritmaların bazılarının karşılaştırması

| Algoritma | Yıl | Parametreler | Avantajlar | Dezavantajlar | Evrimsel değişim desteği var mı? | Kümeleme türü |
|---|---|---|---|---|---|---|
| CEDAS (Hyde, Angelov ve MacKenzie, 2017) | 2017 | Yarıçap, küme yaşlanma oranı ve minimum eşik değeri | Tamamıyla çevrimiçi çalışan ve farklı şekildeki kümeleri tespit edebilir. | Mikro kümelerin birleşimi ile oluşan makro kümelerin doğru tanımlanabilmesi optimum yarıçapın tanımlanması gerekir | Evet | Hiyerarşik |
| LLDStream (Laohakiat, Phimoltares ve Lursinsap, 2017) | 2017 | Yarıçap ve yoğunluk parametresi | Boyut indirgemeyi destekler, farklı şekilli kümeleri tespit edebilir, performansı iyi | Tamamen çevrimiçi değil ve seçilen parametreler sonucu etkiler | Evet | Yoğunluk |
| Xinxin ve ark. önerdiği algoritma (X. Shao, Zhang ve Meng, 2018) | 2018 | Yoğunluk eşik değeri, yoğunluk katsayısı, en yakın komşu değeri paylaşım değeri | Farklı şekildeki kümeleri bulabilir, sapan verileri tespit edebilir, küme sayısında sınırlama yok | Optimum parametreleri belirlemek zor | Hayır | Yoğunluk & Grid tabanlı |
| SyncTree (J. Shao, Tan, Gao, Yang, Plant ve Assent, 2018) | 2018 | Küme sayısı, kova boyutu, kesişim aralığı, çevrim dışı kümeleme değişkeni vs. | Veri özetleme şekli kendine özgü, performansı iyi | Çevrimiçi çevrimdışı evrelerden oluşması, çok sayıda parametre belirlenmeli | Evet | Hiyerarşi & Yoğunluk |
| evoStream (Carnein ve Trautmann, 2018) | 2018 | Küme sayısı, popülasyon boyutu, bozunum faktörü, yarıçap vs. | Kaynakları etkin kullanması, kümeleme başarısının iyi olması | En iyi parametreleri belirlemek güç, tamamen çevrimiçi değil | Evet | Yoğunluk |
| StreamSW (Reddy ve Bindu, 2018) | 2018 | Sapan veri eşik değeri, yarıçap eşik değeri ve ağırlık eşik değeri | Yüksek performanslı ve başarılı kümeleme yapması | Çevrimiçi ve çevrimdışı evrelerden oluşması ve çok boyutlu verileri işlemede problem yaşamaktadır. | Evet | Yoğunluk |

Çizelge 2.2. (Devam) Akan veri kümeleme alanındaki algoritmaların bazılarının karşılaştırması

| Algoritma | Yıl | Parametreler | Avantajlar | Dezavantajlar | Evrimsel değişim desteği var mı? | Kümeleme türü |
|---|---|---|---|---|---|---|
| BOCEDS (M. Ahmed, 2019) | 2019 | İşlenecek veri miktarı, minimum ve maksimum yarıçap ve minimum eşik değeri | Çevrimiçi çalışan, yerel en iyi yarıçapı belirleme, farklı küme şekillerini destekleme | Kullanılan hafıza biraz yüksek | Evet | Yoğunluk |
| ChronoClust (Putri, Read, Koprinska, Singh, Röhm, Ashhurst ve King, 2019) | 2019 | Minimum ağırlık eşik değeri, maksimum yarıçap eşik değeri, maksimum mikro-küme sayısı vs. | Zaman damgalı verileri işlemesi, | Çevrimiçi-çevrimdışı evrelerden oluşması, çok sayıda parametre kullanması | Evet | Yoğunluk |

# 3. K-BOYUTLU AĞAÇ, UYARLANABİLİR YARIÇAP VE ÖZNİTELİK SEÇME (KD-ARFS STREAM) TABANLI GERÇEK ZAMANLI AKAN VERİ KÜMELEME

Önerdiğimiz yaklaşımın temelini çok boyutluluğu destekleyen K-boyutlu ağaç (K-b ağacı), uyarlanabilir yarıçap, zaman tabanlı özetleme ve standart sapma tabanlı öznitelik seçme oluşturmaktadır. Önerdiğimiz yaklaşımın literatüre olan temel katkılarını şu şekilde sıralayabiliriz:

• Tamamen çevrimiçi: Önerdiğimiz yaklaşım bütün işlemleri çevrimiçi gerçekleştirmektedir. Oysa literatürdeki çalışmaların büyük çoğunluğu çevrimiçi ve çevrimdışı olarak iki aşamadan oluşmaktadır;

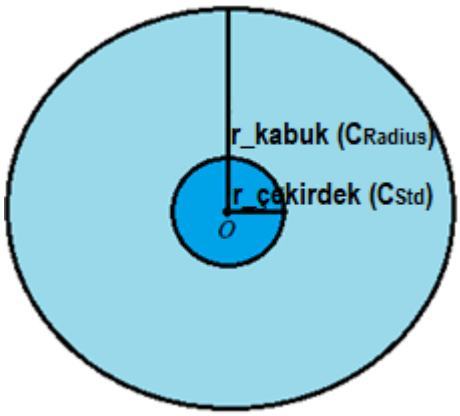
• Evrimsel yapı: Önerdiğimiz yaklaşım küme oluşturma, kümeleri aktif/pasif yapma, kümeleri birleştirme veya bölme ve kümelerin yarıçap ile merkez koordinatlarını güncelleme gibi bütün verileri, verinin yapısına uygun olarak gerçekleştirmektedir. Bu evrimsel işlemleri kısaca şu şekilde sıralayabiliriz:

- Yeni bir kümenin oluşması (Appearence): Belirli bir yarıçap içinde ve diğer kümelerden yeterince uzakta belirli sayıdaki veri yeni bir küme olarak tanımlanır.
- Aktif bir kümenin pasif olması/Pasif bir kümenin tekrar aktif olması (Activation-Inactivation): Aktif bir kümenin sahip olduğu veri sayısı ömrünü tamamlayan verilerin silinmesi nedeniyle belirlenmiş eşik değerinin altına düşerse bu küme pasif yapılır. Pasif durumdaki bu kümenin sahip olduğu veri sayısı yeni veri almak suretiyle yeniden eşik değerinin üzerine çıkarsa bu küme yeniden aktive edilir.
- Küme evrimi (Self-evolution): Küme yarıçapı, kümenin sahip olduğu veri sayısı ve küme merkezinin bulunduğu koordinatlar gibi değerler küme yeni veriler aldıkça veya yaşam süresi biten veriler silindikçe değişir. Bu parametreler gerçek zamanlı olarak güncellenmektedir.
- Küme birleştirme (Merge): Eğer aktif iki kümeden birinin kabuk yarıçapı ile diğerinin çekirdek yarıçapının toplamı merkezleri arasındaki mesafeden fazla ise bu iki küme birleştirilir.
- Küme bölme (Split): Aktif bir kümede bulunan verilerden belirli sayıdaki bir kısmının oluşturduğu aday kümenin kabuk yarıçapı ile kümenin geri kalanının oluşturduğu aday kümenin kabuk yarıçapının toplamı iki aday kümenin merkezleri arasındaki mesafeden az ise bu iki küme bölünür.

• Uyarlanabilir yarıçap: Kümlerin sahip olduğu verilerin değişmesi nedeniyle kümlerin yarıçapları artıp azalabilmektedir. KD-ARFS Stream bu durumu desteklemektedir;
• Zaman ve kayan pencereler tabanlı hibrit özetleme: Önerdiğimiz yaklaşım son 1 sn. son 1 dk. veya son 1 saat gibi zaman tabanlı bir özetlemeyi kayan pencere tabanlı özetleme ile birleştiren hibrit bir yapı kullanmaktadır;
• Kümeleme hafızası: KD-ARFS Stream algoritması kümelerin geçmişi ile ilgili bilgileri tutmaktadır. Belirli bir alanda tanımlanmış olan bir kümenin sahip olduğu veri sayısının belirlenen eşik değerinin altına düşmesi durumunda bu küme silinmez, pasif olarak işaretlenir. Bir süre sonra aynı alana veri gelmesi durumunda bu küme yeniden aktif olarak işaretlenir. Oysa literatürdeki çalışmaların büyük çoğunluğunda böyle bir durumda küme silinmekte ve aynı alana yeterli verinin gelmesi durumunda yeni küme olarak tanımlanmaktadır;
• Öznitelik seçme: Önerdiğimiz yaklaşım standart sapma tabanlı bir öznitelik seçme işlemi yapmaktadır. Böylece gereksiz nitelikler elemine edilerek performans arttırılmaktadır;

### 3.1. Yarıçap Türleri

Şekil 3.1.'de de görüldüğü gibi KD-ARFS Stream algoritmasında iki çeşit yarıçap kullanılmaktadır. Bunlardan ilki niteliklerin standart sapmalarının ortalaması olan çekirdek (core) yarıçaptır. Çekirdek yarıçap küme birleştirme işleminde kullanılmaktadır. Kümenin verilerinin yoğunlaştığı alanı ifade etmektedir. Önerdiğimiz yaklaşım her küme için çekirdek yarıçapını otomatik olarak sahip olduğu veriler üzerinden hesaplamaktadır. Dolayısıyla her kümenin çekirdek yarıçapı farklı olmaktadır.




Şekil 3.1. KD-ARFS Stream algoritmasında kullanılan yarıçap türleri

σ, yani kümenin çekirdek yarıçapı Eş. 3.1 ile hesaplanmaktadır. Burada *d* nitelik sayısını, *N* kümenin sahip olduğu veri sayısını, $X_i$ üzerinde işlem yapılan veriyi ve μ kümenin *j*'inci niteliğine ait ağırlık merkezini (ortalamasını) göstermektedir.

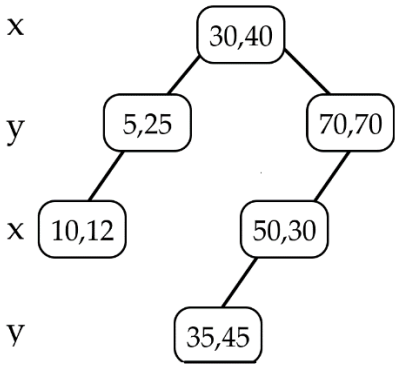
$$\sigma = \frac{1}{d}\sum_{j}^{d}\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(X_i^j - \mu_i^j\right)^2} \tag{3.1}$$

Diğer yarıçap türü kabuk (shell) yarıçaptır. Kabuk yarıçap kümenin sınırlarını belirler. Küme bölme işleminde kullanılmaktadır. Ayrıca yeni bir veri geldiğinde bu verinin ilgili kümeye aktarılıp aktarılmayacağına karar verirken de kabuk yarıçaptan faydalanılmaktadır. Kabuk yarıçap değeri belirlenmiş maksimum değere kadar adım adım artmaktadır.

### 3.2. K-b Ağacı ve Alan Arama (rangesearch)

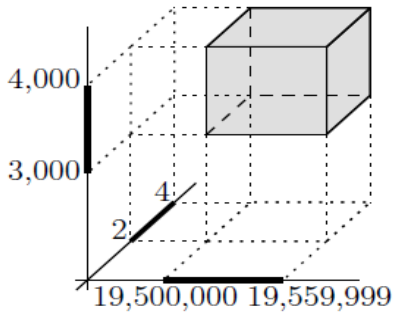
K-b ağaçları dengeli ikili arama ağaçlarıdır. K-b ağaçlarında veriler yapraklarda bulunur ve düğümler veriyi daha küçük alt ağaçlara böler. düğümler solundaki ağaç kendisinden küçük verileri tutarken, sağındaki alt ağaç kendisinden büyük verileri tutar. Kısaca her node veriyi ikiye böler (Bentley, 1975).

Genel anlamda ağaçlar tek boyutlu verileri tutmaktadır. Ancak K-b ağaçları çok boyutlu verileri ağaç yapısında yapılandırmaya imkan vermektedir. Örneğin bir sınıfın öğrencilerini hem yaşlarına hem de boylarına göre aynı anda bir ağaç yapısında yapılandırmak mümkündür. Ancak bu işlem gerçekleştirilirken ağacın her seviyesinde veri farklı boyuta göre yerleştirilir. Aynı şekilde arama işlemi de ağacın her seviyesinde farklı niteliğe göre yapılır. Şekil 3.2'de de görüldüğü gibi iki boyutlu bir veri setinde ilk seviye (kök düğüm) x niteliğine göre yerleştirilmektedir. İkinci seviyede y niteliğine yerleştirilirken üçüncü seviyede yine x niteliğine göre yerleştirilir. Bu işlem bu şekilde devam eder.



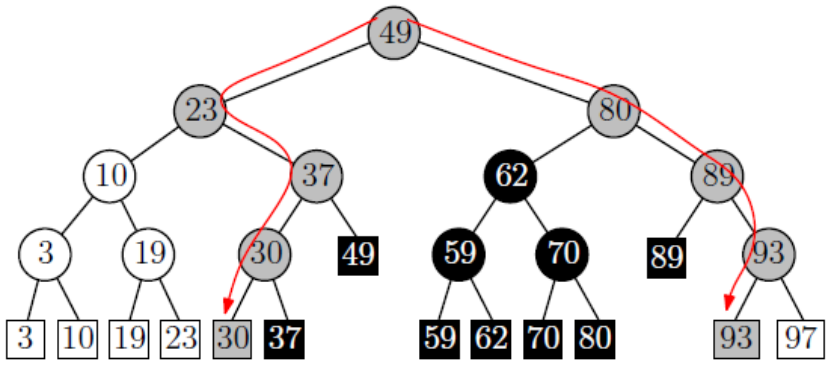

Şekil 3.2. K-Boyutlu ağaç örneği

Çok boyutlu verilerde belirli sınırlar arasındaki verilere ulaşma çabası bir rangesearch problemdir. Aşağıdaki şekilde de görüldüğü gibi bir problemi ele alalım. Bir şirkette çalışan kişilerden 2-4 çocuk sahibi, maaşı 3000-4000 arasında olan ve 1950-1956 yılları arasında doğan kişileri bulmak istediğimizde elde edeceğimiz alan Şekil 3.3'teki koyu alan olacaktır.




Şekil 3.3. Rangrsearch örneği (Kreveld ve Toll, 2018)

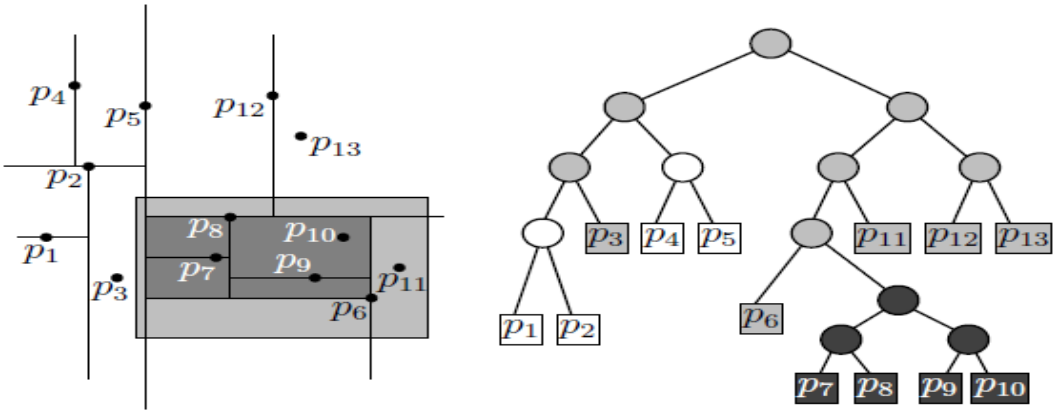
Tek boyutlu bir veriyi ağaç yapısında yapılandırdığımızda ve belli bir aralıktaki verileri almak istediğimizde elde edeceğimiz Şekil 3.4'teki gibi olacaktır.




Şekil 3.4. [25, 90] Aralığındaki verileri elde etme (Kreveld ve Toll, 2018)

Şekil 4.4'teki ağaçta da görüldüğü gibi node'lar farklı renkte işaretlenmiştir. Beyaz renkli node'lar hiç ziyaret edilmemiş anlamına gelirken, gri node'lar ziyaret edildi ancak sonuca eklenmedi ve siyah olanlar ise hem ziyaret edildi hem de sonuca eklendi anlamına gelir. Yani siyah node'lar aranan aralıktaki node'lardır.

K-b ağaçları veriyi bölerken x ekseninde ve y ekseninde olarak sırayla böler. Bölme işlemini tüm verileri bölene kadar devam eder. Her bölme işleminde veriyi ikiye böler. Aşağıdaki şekilde de göründüğü gibi eğer [p7, p10] aralığındaki verileri elde etmek istediğimizde bölme işlemini Şekil 3.5'te görüldüğü gibi yapacaktır.




Şekil 3.5. K-b ağaçlarında arama (Kreveld ve Toll, 2018)

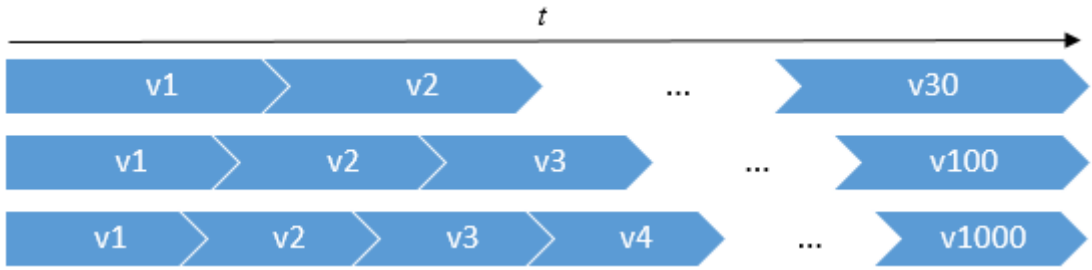
K-b ağaçlarının karmaşıklığına bakacak olursak pek çok açıdan avantajlı veri yapılarıdır. K-b ağacını oluşturma işleminin karmaşıklığı O(nlogn)'dir. Çünkü özyinelemeli (recursive) bir şekilde çalışmaktadır. k arama yapılan alanda bulunan nokta düğüm olmak üzere k-boyutlu ağaçlarda arama işleminin karmaşıklığı O($\sqrt{n} + k$)'dır.

### 3.3. Zaman Tabanlı Özetleme

KD-ARFS Stream algoritması zaman tabanlı bir özetleme yapmaktadır. Bankacılık, sağlık veya güvenlik amaçlı uygulamalarda son 1 dk. son 1 saat veya son 1 günlük veriler üzerinde işlem yapmaya ihtiyaç duyulabilir. Bu durumlarda zaman tabanlı bir akan veri özetlemeye ihtiyaç duyulmaktadır. Önerdiğimiz yaklaşım bu talebe cevap vermektedir.

Şekil 3.6'da da görüldüğü gibi zaman tabanlı özetleme yapıldığı zaman belirtilen zaman zarfında gelen veri miktarı değişmektedir. Bazen gelen veri miktarı işlenemeyecek kadar

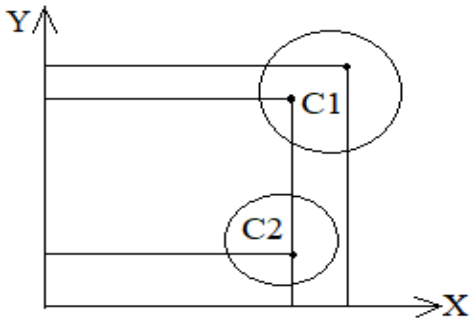
fazla da olabilir. Bu problemi aşmak için hibrid bir yapı kullanılmaktadır. Eğer belirlenen zaman zarfında kullanıcının belirlediği değerden (TN) daha fazla veri gelirse, son gelen TN kadarı özet olarak alınmaktadır.




Şekil 3.6. Veri yaşlandırma ve veri miktarı ilişkisi

### 3.4. Standart Sapma Tabanlı Öznitelik Seçme

Akan veri kümeleme alanında karşılaşılan problemlerden biri de çok boyutlu verileri işlerken performansın düşmesidir. Bu problemi aşmak adına öz nitelik seçme veya boyut indirgeme gibi işlemler çokça kullanılmaktadır. Bir niteliğin standart sapmasının çok düşük olması ayırt edici bir özelliğinin olmadığını göstermektedir. Yani standart sapması fazla olan niteliklerin seçici özelliği vardır. Bu nedenle standart sapması fazla olan verileri seçmek performansı arttıracaktır. Şekil 3.7'de de görüldüğü gibi X niteliğinin seçici bir özelliği yoktur. Bu yüzden Y niteliğini seçmek mantıklı olandır.




Şekil 3.7. İki nitelikli bir veri setinde standart sapma örneği (Yousefpour, Ibrahim, Abdull Hamed ve Hajmohammadi, 2014)

KD-ARFS Stream algoritmasında standart sapma tabanlı bir öznitelik seçme işlemi yapılmaktadır. Standart sapma verilerin ortalamaya ne kadar yakın olduğunu gösteren bir istatistiki veridir. Ancak öznitelik seçme işlemini gerçekleştirirken her nitelik için öncelikle *z-skor* normalizasyonu yapılmaktadır. Çünkü her nitelik için standart sapmanın boyutu kendi

içerisinde değerlendirilmelidir. *x* girdi değeri, $\mu$ ortalama ve $\sigma$ standart sapma olmak üzere *z-skor* Eş. 3.2 ile hesaplanır.

$$Z - skor = \frac{x - \mu}{\sigma} \tag{3.2}$$

KD-ARFS Stream algoritmasında öznitelik seçme işlemi aşağıdaki durumlarda yapılmaktadır:

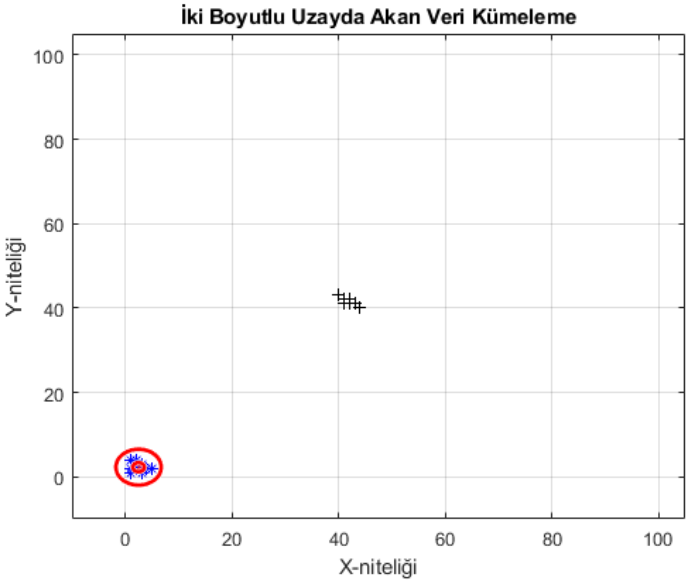
• Yeni bir kümenin tanımlanması
• İki kümenin birleştirilmesi
• Bir kümenin ikiye bölünmesi
• Kümelerden birinin aktif/pasif durumunun değişmesi

Öznitelik seçme işleminin sıklığı kullanılan veri setine bağlı olarak değişir. Çünkü her veri seti için yukarıda üzerinde durduğumuz öznitelik seçme durumlarının ortaya çıkması farklıdır. Öznitelik seçme işlemi buffer'a alınmış veri ve pasif tüm kümeler için silinmiş olan verilerden N'er tane ile oluşan veri grubu üzerinde yapılmaktadır. Amaç belirleyici nitelikleri seçmektir.

## 3.5. Temel Operasyonlar

### 3.5.1. Küme oluşturma

Şekil 3.8'de de görülebileceği gibi r kadar yarıçapta hiçbir kümeye ait olmayan N tane veri oluştuğunda bu veriler birleştirilerek yeni bir küme olarak tanımlanmakta ve bu verilerin ağırlıkları hesaplanarak kümenin merkezi olarak atanmaktadır. Burada dış çember kümenin kabuk yarıçapını gösterirken iç çember çekirdek yarıçapını göstermektedir.




Şekil 3.8. Küme oluşturma örneği

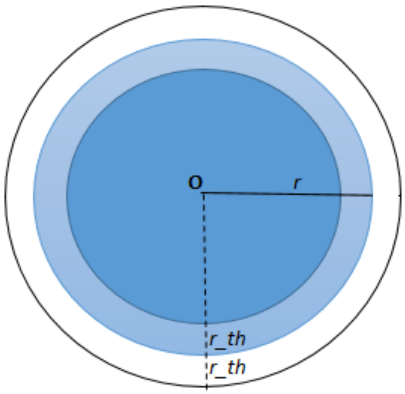
$X_i^j$, $i$'nci verinin $j$'inci niteliğini, $j$ bir niteliği ve $N$ kümenin sahip olduğu eleman sayısını göstermek üzere bu kümenin ilgili niteliği için $\mu^j$ ağırlık merkezi Eş. 3.3 ile hesaplanmaktadır.

$$\mu^j = \frac{1}{N}\sum_{i=1}^{N} X_i^j \tag{3.3}$$

### 3.5.2. Küme yarıçapının arttırılması ve azaltılması

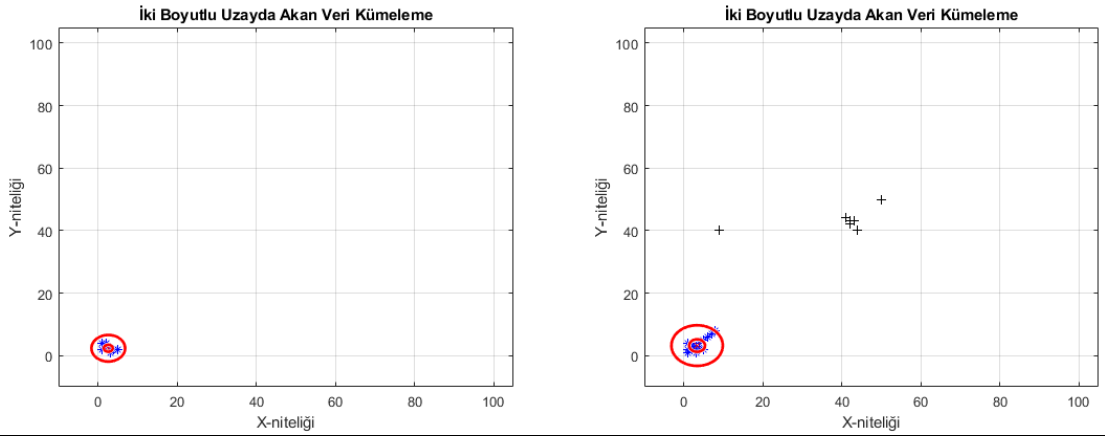
Önerdiğimiz yaklaşımda çekirdek ve kabuk olmak üzere iki çeşit yarıçap bulunmaktadır ve akan verinin yapısı gereği bu değerler verinin özelliğine göre sürekli değişmektedir. Bu iki yarıçap ayrı ayrı ve birbirinden bağımsız olarak hesaplanmaktadır. Çekirdek yarıçap değeri kümelerin sahip olduğu verilerin niteliklerine ait standart sapmalarının ortalaması üzerinden hesaplanırken, kabuk yarıçap kümenin uzak verileri üzerinden hesaplanır.

Kümelerin kabuk yarıçaplarının arttırılıp azaltılması belirli şartlara bağlanmıştır. Kabuk yarıçapın arttırılması Şekil 3.9'da da görüldüğü gibi beyaz alana sadece bir tane yeni verinin gelmesi yeterli değildir (r mevcut yarıçap değeridir). Çünkü bu veri tek başına bir sapan veri de olabilir. Bu nedenle önerdiğimiz algoritmada kabuk yarıçapın arttırılabilmesi için söz konusu alana N/10 tane verinin gelmesi gerekmektedir.




Şekil 3.9. Kümenin mevcut yarıçapı ve yarıçap arttırma eşik değeri

Bir kümenin kabuk yarıçapının artması gibi azalması da mümkündür. Şekil 3.9'da görünen açık mavi alanda bulunan veri sayısı N/10'un altına düştüğünde kümenin mevcut kabuk yarıçapı bir kademe (r_th - r_threshold) azaltılır. Bu iki yaklaşım modele daha kararlı bir uyarlanabilir yarıçap özelliğine sahip olma imkânı tanımaktadır. Özellikle birbirine çok yakın kümelerin yanlışlıkla birleştirilmesinin de önüne geçmektedir. Pasif durumdaki kümelerde kümenin yarıçapı r/2 olarak atanmaktadır. Bunun nedeni kümenin sapan verilere karşı direncini arttırmaktır. Şekil 3.10'da kümenin kabuk yarıçapının artışına örnek verilmiştir.

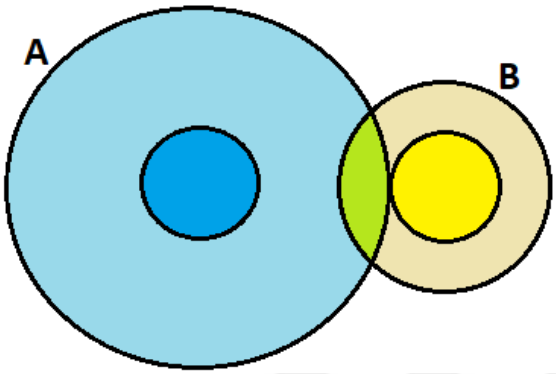



Şekil 3.10. Küme yarıçapının arttırılması örneği

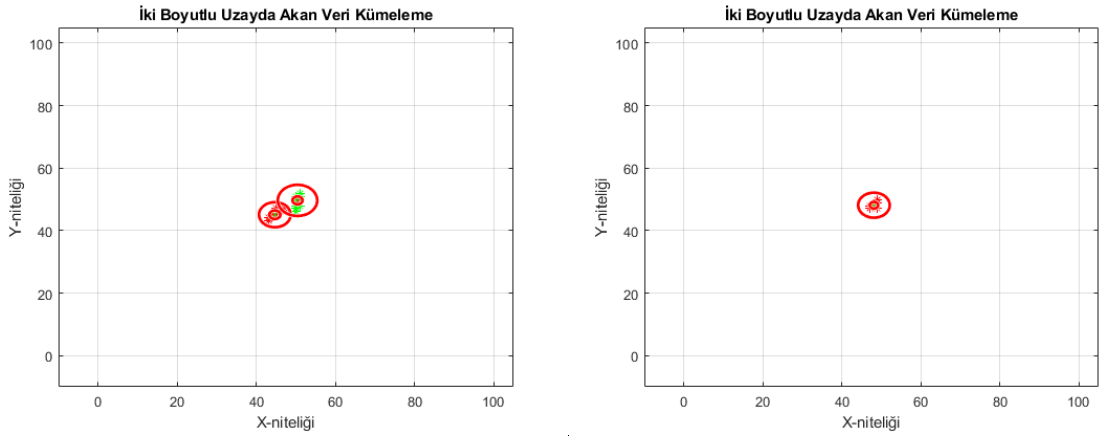
### 3.5.3. İki kümenin birleştirilmesi

Akan verinin yapısı nedeniyle kümelerin özellikleri sürekli değişmektedir. Kümelerin koordinat merkezleri de bu değişimden etkilenir. Bu değişim bazen ayrı iki küme olarak

tanımlanmış kümeleri birbirine çok fazla yaklaştırır. Böyle bir durumda bu kümelerin birleştirip birleştirilmeyeceği değerlendirilir. Karar vermek için de kümelerin merkezleri arasındaki mesafe birinin kabuk yarıçapı ile diğerinin çekirdek yarıçapının toplamı ile kıyaslanır. Şekil 3.11'de de görüldüğü gibi eğer merkez koordinatları arasındaki mesafe birinin çekirdek yarıçapı ile diğerinin kabuk yarıçapı toplamından az ise bu durumda iki küme arasında bir çakışma söz konusudur ve bu kümeler birleştirilir. Şekil 3.12'de iki kümenin birleştirilmesine örnek verilmiştir.




Şekil 3.11. İki kümenin birleştirilmesi örneği

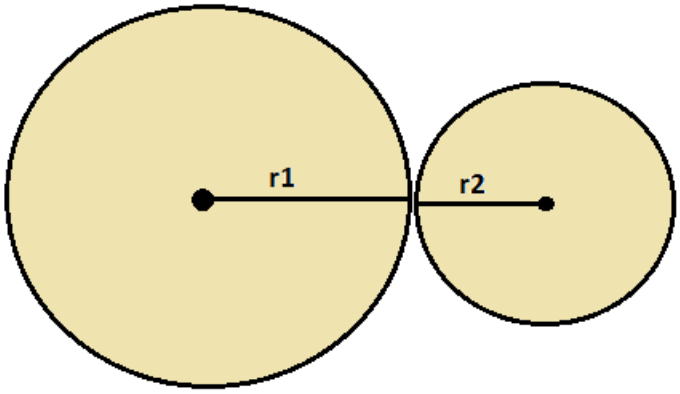



Şekil 3.12. İki kümenin birleştirilmesi örneği

### 3.5.4. Bir kümenin ikiye bölünmesi

Zamana bağlı olarak kümelerin sahip olduğu veriler değişmektedir. Bu değişim kümede yapısal değişimlere neden olabilmektedir. Kümenin yarıçapının, sahip olduğu veri sayınının veya çekirdek yarıçapının değişmesinin yanında çok daha radikal değişimler de olabilmektedir. Bunlardan biri de küme içerisindeki verilerin kendi aralarında gruplaşmalar oluşturmasıdır. Bazen öyle bir durum olur ki küme içindeki bu grupların bir birleri ile pek

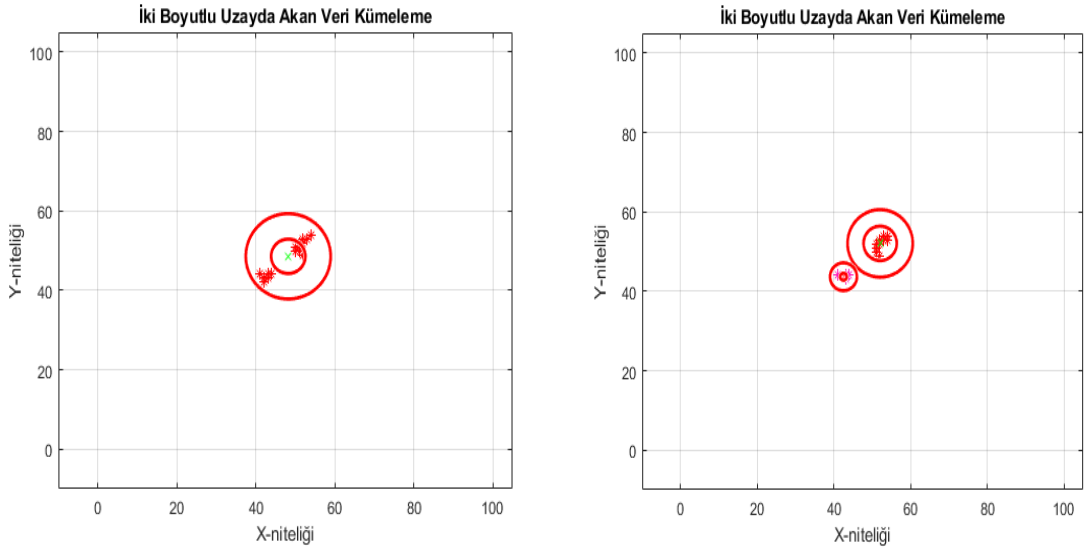
bir bağı kalmayabilir. Yani bu gruplar arasındaki mesafe maksimum değere ulaşabilmektedir. Bu durumda artık bu verilerin ayrı birer küme olarak bölünmesi daha doğru olacaktır. Şekil 3.13'te de görüldüğü gibi bir kümede oluşan aday kümelerin merkezleri arasındaki mesafe kabuk yarıçaplarının toplamından daha fazla olur. Bu durumda bu küme ikiye bölünür.




Şekil 3.13. Bir kümenin ikiye bölünmesinde kabuk yarıçapların rolü

KD-ARFS Stream algoritmasında küme oluşturma işleminde olduğu gibi split işleminde de K-Boyutlu ağaç yapısından faydalanılmaktadır. Bir kümenin bölünüp bölünmeyeceğine karar verme noktasında en önemli parametre kümenin çekirdek yarıçapıdır. Bir kümede veriler arasında bir gruplaşmanın olması demek o kümenin aynı zamanda çekirdek yarıçapının da artması anlamına gelmektedir. Bu nedenle performansı arttırmak adına bir küme üzerinde split araştırması yapılıp yapılmayacağına karar verirken o kümenin çekirdek yarıçapına bakılır.

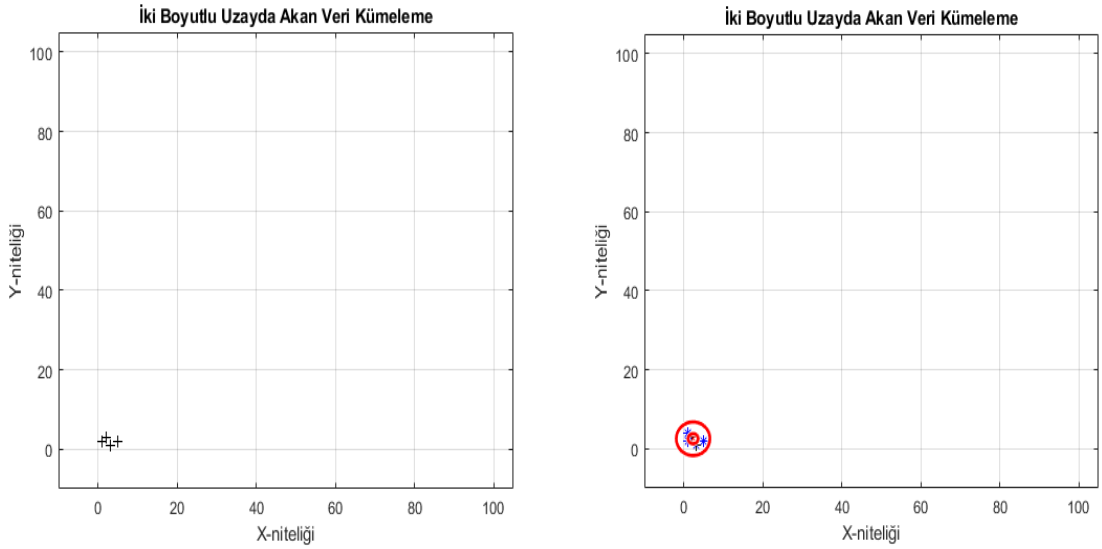
Bir küme üzerinde split araştırması yapılacağı zaman kümenin verileri K-b ağacına yerleştirilir ve bu ağaç üzerinde N ve r değerlerine bağlı olarak rangesearch araştırması yapılır. Eğer r yarıçapta N tane veri oluşmuşsa ve kümenin çekirdek yarıçapı r'den büyükse bu verilerin oluşturduğu aday küme merkezinin, kümenin geri kalan verilerinin oluşturduğu kümenin merkezine olan uzaklığı her iki kümenin oluşturduğu kabuk yarıçaplarla karşılaştırılır. Eğer iki küme arasındaki mesafe ikisinin kabuk yarıçaplarının toplamından fazla ise, bir başka deyişle iki küme kesişmiyorsa bu küme iki yeni küme olarak bölünür. Şekil 3.14'te iki kümenin bölünmesine örnek verilmiştir.




Şekil 3.14. Bir kümenin ikiye bölünmesi örneği

### 3.5.5. Kümelerin aktif/pasif durumlarının değişmesi

Aktif bir kümenin elemanları zamana bağlı olarak silinebilir ve sahip olduğu veri miktarı küme oluşturmak için gerekli minimum veri miktarı sayısının (N) altına düşebilir. Bu durumda bu küme pasif olarak işaretlenir. Şekil 3.15'te de görüldüğü gibi pasif durumdaki bu kümeye yeni veriler eklendikçe sahip olduğu veri sayısı yeniden eşik değerini aşabilir. Bu durumda da pasif durumdaki bu küme yeniden aktive edilir. Yani kümeler zaman içinde aktif veya pasif olabilmektedir.




Şekil 3.15. Pasif durumdaki bir kümenin yeniden aktive edilmesi

## 3.6. Başlıca Hesaplamalar

### 3.6.1. Veri-Küme mesafesi

Bir verinin bir kümenin merkezine olan Öklid uzaklığını ifade eder. $C^j$ bir kümeyi, *X* bir veriyi, *d* verilerin sahip olduğu nitelik sayısını ve *j* o anki niteliği göstermek üzere *C* kümesi ve *X* verisi arasındaki uzaklık *dist* Eş. 3.4 ile hesaplanır.

$$dist(C, X) = \sqrt{\sum_{j=1}^{d} (C^j - X^j)^2} \tag{3.4}$$

### 3.6.2. Küme-Küme mesafesi

İki kümenin merkezleri arasındaki Öklid uzaklığını ifade eder. $C_a$ ve $C_b$birer kümeyi, *d* verilerin sahip olduğu nitelik sayısını ve *j* o anki niteliği ifade etmek üzere $C_a$ ve $C_b$ kümeleri arasındaki mesafe *dist,* Eş. 3.5 ile hesaplanır.

$$dist(C_a, C_b) = \sqrt{\sum_{j=1}^{d} (C_a^j - C_b^j)^2} \tag{3.5}$$

### 3.6.3. Kümeden veri silinmesi sonucu küme merkezinin güncellenmesi

Bir kümeden bir verinin silinmesi durumunda kümenin ağırlık merkezi değişecektir. Bu durumda kümenin ağırlık merkezinin güncellenmesi gerekecektir. $C_{old}^j$ kümenin *j* niteliğine ait eski ortalamayı, *N* kümenin sahip olduğu eleman sayısını ve $X^j$ silinen verinin *j* niteliğine ait değeri göstermek üzere kümenin *j* niteliğine ait yeni ortalaması Eş. 3.6 ile hesaplanır.

$$C_{new}^j = \frac{C_{old}^j \,.\, N - X^j}{N - 1} \tag{3.6}$$

### 3.6.4. Kümeye veri eklenmesi sonucu küme merkezinin güncellenmesi

Bir kümeye yeni bir verinin eklenmesi durumunda kümenin ağırlık merkezi yine değişecektir. Bu durumda kümenin ağırlık merkezinin güncellenmesi gerekecektir. $C_{old}^{j}$ kümenin *j* niteliğine ait eski ortalamayı, *N* kümenin sahip olduğu eleman sayısını ve $X^{j}$ silinen verinin *j* niteliğine ait değeri göstermek üzere kümenin *j* niteliğine ait yeni ortalaması Eş. 3.7 ile hesaplanır.

$$C_{new}^{j} = \frac{C_{old}^{j} \,.\, N + X^{j}}{N + 1} \tag{3.7}$$

## 3.7. Algoritma

### 3.7.1. Tanımlar

Algoritmada kullanılan notasyonlar ve bunların açıklamaları Çizelge 3.1'de görüldüğü gibidir.

Çizelge 3.1. KD-ARFS Stream algoritmasında kullanılan notasyonlar ve anlamları

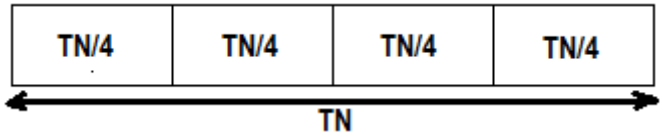
| Notasyon | Açıklaması |
|---|---|
| Data | Veri |
| C | Küme tablosu |
| bufferedData | Üzerinde işlem yapılan veriler |
| bufferedDataSize | Üzerinde işlem yapılan veri sayısı |
| AllData$C_i$ | Ci kümesine ait veriler |
| deletedData | Silinen veriler |
| t | Veri ömrü eşik değeri |
| TN | Özet olarak alınmış olan veri miktarı çok ise ne kadarının alınacağını belirten eşik değer |
| r | Kümenin kabuk yarıçapı |

Çizelge 3.1. (Devam) KD-ARFS Stream algoritmasında kullanılan notasyonlar ve anlamları

| | |
|---|---|
| r_threshold | Yarıçap arttırma eşik değeri |
| r_max | Yarıçap için maksimum değer |
| d | Verilerin sahip olduğu nitelik sayısı |
| newD | Seçilen öznitelik sayısı |
| StdOfFeatures | Niteliklerin standart sapması |
| coeffThreshold | Öznitelik seçme eşik değeri |
| N | Küme oluşturmak için gerekli minimum veri sayısı |
| tree | K-boyutlu ağaç |
| $C_{temp}$ | Aday küme |
| dataSize | Veri sayısı |
| numOfDataInTheArea1 | Kabuk yarıçap arttırmak için eşik değer bölgesinde bulunan verilerin sayısı |
| numOfDataInTheArea2 | Kabuk yarıçap azaltmak için eşik değer bölgesinde bulunan verilerin sayısı |
| dist | İki nokta arasındaki Öklid uzaklığı (Küme-veri veya küme-küme mesafesi) |
| $C_{Radius}$ | Kümenin kabuk yarıçapı |
| $C_{Std}$ | Kümenin çekirdek yarıçapı<br>(Niteliklerin standart sapmalarının ortalaması) |
| clusterNO | Kümenin indeksi |
| bufferedData$_i$clusterNo | Tamponlanmış i verisinin ait olduğu küme |
| centerCoordinate | Kümenin ağırlık merkezi, diğer bir deyişle kümenin merkez koordinatları |
| $C_i$Active | $C_i$ kümesinin aktiflik durumu |
| v | Kök düğüm |
| $v_{left}$ | Sol alt ağaç |
| $v_{right}$ | Sağ alt ağaç |

### 3.7.2. KD-ARFS Stream algoritması

Algoritmanın genel çalışma adımları Şekil 3.17'de göründüğü gibidir. Algoritma Şekil 3.16'da da görüldüğü gibi her TN/4 veri biriktikçe çalışmaktadır. Yani kullanıcının belirlediği TN değişkeni boyunca 4 defa çalışmaktadır. Amaç performansı arttırmaktır.




Şekil 3.16. TN değişkeninin algoritmanın çalışmasını kontrolü

Kısaca algoritma şu şekilde çalışmaktadır: AgingAll fonksiyonunda bütün veriler yaşlandırılmakta ve ömrünü tamamlayan veriler silinmektedir. NewClusterAppear fonksiyonunda yeni bir kümenin oluşup oluşmadığı kontrol edilmektedir. Eğer yeni bir kümenin oluşması gerekiyorsa bu kümeyi tanımlamaktadır. CheckOverlapClusters'da aktif kümeler içerisinde birbiri ile kesişen küme varsa bu kümeler birleştirilir. CheckSplit'te aktif kümelerden sahip olduğu veriler içerisinde gruplaşan ve bu gruplar arasındaki mesafe belirli bir oranda ise bu küme bölünmektedir.

FindClosestCluster'da yeni gelen veri ve herhangi bir kümeye ait olmayan veriler içerisinde herhangi bir kümeye olan uzaklığı kümenin kabuk yarıçapı ve r_threshold toplamından az olan varsa, bu veriler ilgili kümeye atanmaktadır. UpdateCenters'da kümelerin ağırlık merkezleri güncellenmektedir. Eleman sayısı N'den büyük olan kümelerin ağırlık merkezleri yani kümelerin merkez koordinatları kümelere ait işlenen verilerden hesaplanırken, eleman sayısı N'den az olan kümelerin merkezleri kümelere ait silinmiş ve işlenen (toplamı N olacak şekilde) verilerden hesaplanmaktadır. Bunun amacı kümelerin sapan verilere karşı direncini arttırmaktır.

UpdateRadius'ta kümelerin kabuk yarıçapı ve çekirdek yarıçapı değerleri güncellenmektedir. Eğer küme aktif değilse kümenin kabuk yarıçapı r/2 olarak atanmaktadır. FlagActiveClusters'da aktif olma şartını taşıyan kümeler aktif olarak tanımlanırken diğerleri pasif küme olarak tanımlanmaktadır. Son olarak SelectFeatures fonksiyonunda öznitelik seçme işlemi gerekiyorsa öznitelik seçme işlemi yapılmaktadır. En son olarak algoritma başa dönmekte ve yeni veri gelişini beklemektedir.

```
Algorithm 1: KD-ARFS Stream
Input: DataStream={x1,x2,x3,...,∞}; //Akan veri
t; //Her verinin ömrü için belirlenen eşik değer
TN; //Belirlenen zaman zarfında gelen veri miktarı fazla ise özet
    //olarak alınacak verinin miktarı
N; //Küme oluşturmak için gerekli mininmum veri sayısı
r; //Küme oluşturmak için taranacak alanın yarıçapı
r_threshold; //Küme yarıçapı arttırma/azaltma eşik değeri
r_max; //Küme yarıçapının ulaşabileceği maksimum değer
coeffThreshols; //Niteliklerin standart sapmalarının (ağırlık)
                //toplamının ulaşması gereken eşik değer
Output: Clusters; //Algoritmanın ürettiği kümeler
while New data available in DataStream do
    if DataStreamSize % (TN/4) = 0 //TN/4 veri birikti mi? then
        AgeingAll; //Verileri yaşlandır, ömrünü tamamlayanları sil
        NewClusterAppear; //Gerekiyorsa yeni küme tanımla
        CheckOverlapClusters; //Çakışan kümeleri birleştir
        CheckSplit; //Bölünmesi gereken kümeleri böl
        FindClosestCluster; //Yeni gelen veri bir kümeye yeterince
                            //yakınsa ilgili kümeye at
        UpdateCenters; //Kümelerin merkez koordinatlarını güncelle
        UpdateRadius; //Küme yarıçaplarını (kabuk, çekirdek)
         güncelle
        FlagActiveClusters; //Şartları sağlayan kümeleri aktifleştir
        if A new cluster is formed or any cluster is split or any two
         clusters are merged or Active/Passive status of any cluster is
         changed //Öznitelik seçme şartlarından biri oluştuysa then
            SelectFeatures; //Öznitelik seçme işlemini gerçekleştir
        end
    end
end
```

Şekil 3.17. KD-ARFS Stream algoritmasının temel fonksiyonları

Şimdi sözünü ettiğimiz fonksiyonları kısaca açıklayalım:

AgingAll fonksiyonunda işlenen tüm veriler yaşlandırılmakta ve ömrünü tamamlayan, yani yaşı belirlenen eşik değerinin üzerinde olan veriler silinerek değişkenine eklenmektedir. Eğer yaşını doldurmayan veri sayısı belirlenen TN veriden fazla ise en son gelen TN tane veri işlenecek veri olarak alınır. AgingAll fonksiyonunda buffer'a alınmış veriler her seferinde işlendiğinden, n üzerinde işlem yapılan veri sayısı olmak üzere karmaşıklığı O(n) olmaktadır. Aşağıda AgingAll fonksiyonunun sözde kodu Şekil 3.18'de görüldüğü gibidir.

```
Algorithm 2: AgingAll
Input: C;//Küme tablosu
bufferedData;//Tamponlamış veriler
deletedData; //Silinen veriler
t; //Her verinin ömrü için belirlenen eşik değer
TN; //Belirlenen zaman zarfında gelen veri miktarı fazla ise özet
    //olarak alınacak verinin miktarı
Output: C;//Küme tablosu
bufferedData;//Tamponlanmış veriler
deletedData;//Silinen veriler
if bufferedDataSize > 1 //Tamponlanmış veri 1'den büyükse then
    foreach Data_i ∈ bufferedData //Tamponlanmış her veri için do
        if elapsed time for Data_i t_i > t //Data_i ömrü bittiyse then
            if Data_i belongs to any Cluster //Data_i bir kümeye aitse
             then
                deletedData_i ← Data_i; //Data_i'yi deletedData'ya ekle
            end
            Delete Data_i; //Data_i'yi sil
        end
    end
    if bufferedData > TN //BufferedData TN'den büyükse then
        bufferedData ← last arrived TN of bufferedData;//En son gelen
                       //TN tane veriyi tamponlanmış veri olarak al
    end
end
```

Şekil 3.18. AgingAll fonksiyonunun sözde kodu

NewClusterAppear fonksiyonu hiçbir kümeye ait olmayan veriler arasında belirlenen r yarıçapta belirlenen eşik değeri kadar veri gruplanmış mı kontrol etmektedir. Eğer bu durum söz konusu ise bu verileri yeni bir küme olarak belirlemektedir. Bu işlemi gerçekleştirmek için söz konusu verileri K-Boyutlu ağaca yerleştirmekte ve r ile N değerlerine bağlı olarak rangesearch işlemi yapmaktadır. Eğer r yarıçap içerisinde N tane veri varsa bu verilerin oluşturduğu aday kümenin diğer kümelere uzaklığı kontrol edilmektedir. Eğer bütün kümelerden yeterince uzakta ise bu aday kümeyi yeni bir küme olarak küme tablosuna eklemektedir. K-b ağacını oluşturma işleminin karmaşıklığı d her bir verinin sahip olduğu nitelik sayısı ve n işlenen veri sayısı olmak üzere O(dnlogn)'dir. Oluşturulan ağaç üzerinde rangesearch işleminin karmaşıklığı k aranan alandaki veri sayısı olmak üzere O(k+$\sqrt{n}$)'dir. NewClusterAppear algoritmasında range search işlemi her veri için yapıldığından rangesearch işleminin karmaşıklığı O(nk+n$\sqrt{n}$ olur. Dolayısıyla NewClusterAppear algoritmasının karmaşıklığı O(dnlogn+nk+n$\sqrt{n}$) olmaktadır. NewClusterAppear fonksiyonunun sözde kodu Şekil 3.19'da görüldüğü gibidir.

```
Algorithm 3: NewClusterAppear
Input: C; //Küme tablosu
bufferedData; //Tamponlamış veriler
r;/ /Yarıçap eşik değeri
N; //Küme oluşturmak için gerekli mininmum veri sayısı
Output: C; //Küme tablosu
bufferedData; //Tamponlamış veriler
Get all Data that does not belong to any cluster;//Hiçbir kümeye ait
                                                 //olmayan verileri Data'ya at
tree ← BuildKDTree(Data);//KB ağacına yerleştir
if dataSize(Data) > N //Minimum N tane veri varsa then
    foreach Data_i ∈ Data //Data'nın her verisi için do
        C_temp ← rangeSearch(tree, r);//Alan araştırması yap
        if dataSize(C_temp) > N //Aday küme veri sayısı > N then
            Assign C_temp as new cluster;//Aday kümeyi yeni küme
                                         //olarak tanımla
        end
    end
end
```

Şekil 3.19. NewClusterAppear fonksiyonunun sözde kodu

CheckOverlapClusters fonksiyonu aktif durumdaki kümeler arasında, aralarındaki mesafe belirlenen eşik değerinin altına düşen var ise bu kümeleri birleştirmektedir. Kümelerin birleştirilip birleştirilmeyeceğine karar verilirken kümelerin hem çekirdek hem de kabuk yarıçap değerlerinden faydalanılmaktadır. Eğer iki kümenin merkezleri arasındaki mesafe birinin kabuk yarıçapı ile diğerinin çekirdek yarıçapının toplamından daha az ise bu iki küme birleştirilir. CheckOverlapClusters algoritması ikili olarak kümelerin çakışıp çakışmadığını kontrol etmektedir. Bu nedenle m küme sayısı olmak üzere CheckOverlapClusters fonksiyonunun karmaşıklığı $O(m^2)$ olmaktadır. CheckOverlapClusters fonksiyonunun sözde kodu Şekil 3.20'de görüldüğü gibidir.

```
Algorithm 4: CheckOverlapClusters
Input: C; //Küme tablosu
bufferedData; //Tamponlamış veriler
deletedData; //Silinen veriler
d; //Nitelik sayısı
Output: C; //Küme tablosu
bufferedData; //Tamponlanmış veriler
foreach cluster C_i ∈ C //Her küme için do
    foreach cluster C_j ∈ C //Her küme için do
        if C_i and C_j active clusters and i ≠ j //Her iki küme aktif ise
                                                 //ve aynı küme değilse then
            dist ← findDistance(C_i,C_j,d);//Aralarındaki mesafeyi bul
            if dist<(C_i(Radius) + C_j(Std)) or dist<(C_i(Std) + C_j(Radius))
             //Birinin kabuk yarıçapı diğerinin çekirdek yarıçapı ile
             kesişiyorsa then
                merge(C_i, C_j); //Bu kümeleri birleştir
            end
        end
    end
end
```

Şekil 3.20. CheckOverlapClusters fonksiyonunun sözde kodu

CheckSplit fonksiyonu aktif kümeler arasında sahip olduğu veriler arasında belirli bir oranda belirli bir mesafede gruplaşmalar var ise bu kümeleri böler. Bunu gerçekleştirmek için kümenin elemanlarını K-Boyutlu ağaca yerleştirir. Daha sonra bu ağaç üzerinde kabuk yarıçap (r) ve minimum veri sayısı (N) kullanarak rangesearch işlemi yapar. r yarıçapta N tane veri varsa bu verilerin oluşturduğu aday kümenin, kümenin geri kalanının oluşturduğu kümeye olan uzaklığına bakılır. Eğer mesafe iki kümenin kabuk yarıçaplarının toplamından daha fazla ise bu küme iki kümeye bölünür. CheckSplit fonksiyonunun karmaşıklığı NewCluaterAppear fonksiyonuna benzerdir. CheckSplit, NewClusterAppear fonksiyonuna benzer şekilde verileri önce ağaca yerleştirmekte ve bu ağaç üzerinde rangesearch işlemi gerçekleştirmektedir. Ama NewClusterAppear fonksiyonundan farklı olarak bu işlemi her küme için ayrı ayrı yapmaktadır. m küme sayısı olmak üzere CheckSplit fonsiyonunun karmaşıklığı O(dmnlogn+mnk+mn√n)‘dir. CheckSplit fonksiyonunun karmaşıklığı pratikte düşünüldüğü gibi yüksek değildir. Çünkü yaklaşımımız, performansı arttırmak için her küme için CheckSplit araştırması yapmaz. Sadece en az 2N tane elemanı olan ve çekirdek yarıçapı r’den büyük olan kümeler üzerinde CheckSplit araştırması yapar. Sözde kodu Şekil 3.21’de görüldüğü gibidir.

```
Algorithm 5: CheckSplit
Input: C; //Küme tablosu
bufferedData; //Tamponlamış veriler
r; //Yarıçap eşik değeri
N; //Küme oluşturmak için gerekli mininmum veri sayısı
d; //Nitelik sayısı
Output: C; //Küme tablosu
bufferedData; //Tamponlanmış veriler
foreach active clusters C_i ∈ C //Her aktif küme için do
    if dataSize(C_i) >2*N and C_i(Radius)>=2*r and C_i(Std)>=r
     //Kümenin veri sayısı 2N, kabuk yarıçapı 2r ve çekirdek
     yarıçapı r kadar ise then
        tree ← BuildKDTree(AllData_Ci);//Verilerini ağaca yerleştir
        foreach data of C_i //Her verisi için do
            C_temp ← rangeSearch(tree, r);//Alan araştırması yap
            if dataSize(C_temp) >= N //C_temp verisayısı>=N ise then
                dist ← findDistance(C_temp, C_i-C_temp, d);//Aday küme
                                  //ile ana küme arasındaki mesafeyi bul
                r_temp ← findRadius(C_temp); //Aday küme yarıçapı bul
                r_Ci ← findRadius(C_i-C_temp);//Ana küme yarıçapı bul
                if dist > r_temp+r_Ci //Aralarındaki mesafe yarıçapları
                                      //toplamından fazla ise then
                    assign C_temp as a new cluster;//C_temp yeni bir küme
                    C_i ← (C_i-C_temp); //Kümenin kalanı C_i kümesi
                    break;
                end
            end
        end
    end
end
```

Şekil 3.21. CheckSplit fonksiyonunun sözde kodu

FindClosestCluster fonksiyonu yeni gelen ve hiçbir kümeye ait olmayan veriler arasında kümelerin durumunun değişmesi nedeniyle ilgili kümelere dahil edilmek için yeterli yakınlığa ulaşan var ise bu verileri ilgili kümelere eklemektedir. Bu işlemi yaparken öncelikle verinin en yakın olduğu kümeyi bulur. Eğer en yakın olduğu küme ile arasındaki mesafe kümenin r kabuk yarıçapı ve r_threshold toplamını aşmıyorsa bu veri ilgili kümeye eklenir. FindClosestCluster fonksiyonunun karmaşıklığı, veri sayısına (n), her verinin sahip olduğu nitelik sayısına (d) ve küme sayısına (m) bağlıdır. Yani karmaşıklığı O(dmn)'dir ve sözde kodu Şekil 3.22'de görüldüğü gibidir.

```
Algorithm 6: FindClosestCluster
  Input: C; //Küme tablosu
  bufferedData; //Tamponlamış veriler
  d; //Nitelik sayısı
  Output: C; //Küme tablosu
  bufferedData; //Tamponlanmış veriler
  foreach bufferedData_i that does not belong to any cluster//Herhangi
   bir kümeye ait olmayan tüm veriler için do
    distance ← max;//distance değişkenine maksimum değer ata
    foreach active clusters C_j ∈ C // Tüm aktif kümeler için do
      dist ← findDistance(C_j, bufferedData_i);//i verisi ve j kümesi
                          //arasındaki mesafeyi dist değişkenine ata
      if dist < distance and dist<C_j(Radius) +r_threshold and
       dist<r_max //Eğer dist distance'tan küçük, j kümesinin
       mevcut kabuk yarıçapı ve yarıçap arttırma eşik değerinin
       toplamından küçün ve r_max'tan küçük ise bu veriyi j
       kümesine ata then
        distance ← dist;//distance değişkenine dist'i ata
        clusterNo ← j;//Atanan küme olarak j'yi seç
      end
    end
    bufferedData_iclusterNo ← clusterNo;//i verisinin kümesi j
  end
```

Şekil 3.22. FindClosestCluster fonksiyonunun sözde kodu

UpdateCenters kümelerin merkez koordinatlarını günceller. Ancak bu güncelleme işlemini kümenin aktiflik durumuna bağlı olarak iki şekilde yapmaktadır. Eğer küme aktif bir küme ise bu kümenin merkez koordinatlarını kümenin aktif verileri üzerinden yapar. Ama eğer küme pasif bir küme ise bu durumda kümenin merkez koordinatlarını varsa kümenin aktif verileri ve kümenin silinmiş verileri üzerinden hesaplar. Örneğin N=20 ise ve kümenin eleman sayısı 15 ise bu durumda kümenin eleman sayısını 20'ye tamamlamak için bu kümeden en son silinen 5 veri ekleyerek küme merkezini hesaplayacaktır. Bunu yapmasının nedeni kümenin daha tutarlı bir ağırlık merkezine sahip olmasını sağlamaktır. Örneğin bütün verileri silinmiş bir kümenin merkez koordinatları kümeye ait en son verinin koordinatları olacaktır. Bu durumda küme tekrar aktive olmayı beklerken en son verisinin koordinatlarını merkez alarak aktive olmayı bekleyecektir ve sapan bir veriyi kümeye ait bir veri olarak alabilecektir. UpdateCenters fonksiyonunun karmaşıklığı, küme sayısına (m), her kümenin sahip olduğu veri sayısına (n) ve her verinin sahip olduğu nitelik sayısına (d) bağlıdır. Bu nedenle UpdateCenters fonksiyonun karmaşıklığı O(dmn)'dir ve sözde kodu Şekil 3.23'te görüldüğü gibi gerçekleştirmektedir.

```
Algorithm 7: UpdateCenters
  Input: C; //Küme tablosu
  bufferedData; //Tamponlamış veriler
  deletedData; //Silinen veriler
  r;/ /Yarıçap eşik değeri
  N; //Küme oluşturmak için gerekli mininmum veri sayısı
  d; //Nitelik sayısı
  Output: C; //Küme tablosu
  bufferedData; //Tamponlanmış veriler
  foreach cluster C_i ∈ C // Tüm kümeler için do
    dataSize(C_i) ← size(bufferedData_i);//i kümesine ait
                                          //tamponlanmış verileri al
    if C_iActive = true //Eğer i kümesine aktif bir küme ise then
      foreach dimension d_j ∈ d //Tüm nitelikler için do
        C_icenterCoordinate^j ← mean(buffredData_i^j);//Ortalama
                                                      //hesapla
      end
    end
    else
      //Eğer küme aktif bir küme değilse
      Complete missing data of cluster C_i with last deleted ones;
      //Kümenin eksik verilerini silinen son verilerden tamamla
       foreach dimension d_j ∈ d do
        C_icenterCoordinate^j ← mean(bufferedData_i^j);//Orta.
                                                      //hesapla
      end
    end
  end
```

Şekil 3.23. UpdateCenters fonksiyonunun sözde kodu

UpdateRadius fonksiyonu hem kabuk yarıçapın arttırılıp arttırılmayacağına hem de gerekirse azaltılıp azaltılmayacağına karar verir. Kabuk yarıçapın arttırılıp arttırılmayacağına karar verirken yarıçap arttırma eşik değeri kadar yarıçapı aşan bölgede en az N/10 veri var mı diye kontrol eder. Eğer öyleyse kümenin kabuk yarıçapı bir üst seviyeye çıkartılır. Benzer şekilde kümenin kabuk yarıçapının azaltılması da gerekebilir. Bunun için kabuk yarıçap ile kabuk yarıçap ve yarıçap arttırma eşik değerinin farkıyla oluşan bölgede bulunan veri sayısının N/10'un altına düşüp düşmediği kontrol edilir. Eğer söz konusu alandaki veri sayısı N/10'un altında ise kümenin kabuk yarıçapı bir kademe (yarıçap arttırma eşik değeri kadar) azaltılır. UpdateRadius fonksiyonunun karmaşıklığı, küme sayısına (m), her kümenin sahip olduğu veri sayısına (n) ve her verinin sahip olduğu nitelik sayısına (d) bağlıdır. Bu nedenle UpdateRadius fonksiyonun karmaşıklığı O(dmn)'dir ve sözde kodu Şekil 3.24'te görüldüğü gibidir.

**Algorithm 8:** UpdateRadius

```
Input: C; //Küme tablosu
bufferedData; //Tamponlamış veriler
r_max; //Küme yarıçapının ulaşabileceği maksimum değer
r; //Yarıçap eşik değeri
d; //Nitelik sayısı
r_threshold; //Küme yarıçapı arttırma/azaltma eşik değeri
Output: C; //Küme tablosu
bufferedData; //Tamponlanmış veriler
foreach cluster C_i ∈ C // Tüm kümeler için do
    foreach Data_i ∈ bufferedData belongs to C_i //C_i verileri için do
        if C_i is active cluster //Eğer C_i kümesi aktif ise then
            if numOfDataInTheArea1 >= N/4 //Veri yeterli? then
                if C_i(Radius) +r_threshold<= r_max //r_max'ı
                  aşmıyorsa then
                    C_i(Radius) ← C_i(Radius) +r_threshold;//Kabuk arttır
                    C_i(Std) ← mean(std(Data belongs to C_i));
                     //Çekirdek yarıçap hesapla
                end
                else
                    C_i(Radius) ← r_max;//r_max'ı yarıçap olarak belirle
                     C_i(Std) ← mean(std(Data belongs to C_i));
                     //Çekirdek yarıçapı hesapla
                end
            end
            else if numOfDataInTheArea1 +numOfDataInTheArea2
              < N/4 //alandaki veri sayısı N/4'ten azsa then
                C_i(Radius) ← r − r_threshold;//Yarıçapı azalt
                C_i(Std) ← mean(std(Data belongs to C_i); //Çekirdek
                  yarıçapı hesapla
            end
            else
                C_i(Std) ← mean(std(Data belongs to C_i));//Çekirdek
                  yarıçapı hesapla
            end
        end
        else
            if C_i(Radius) > r/2 //Kabuk yarıçap r/2'den azsa then
                C_i(Radius) ← r/2; //Kabuk yarıçapı r/2 seç
                C_i(Std) ← mean(std(Data belongs to C_i));//Çekirdek
                  yarıçapı hesapla
            end
        end
    end
end
```

Şekil 3.24. UpdateRadius fonksiyonunun sözde kodu

FlagActiveClusters Her kümenin sahip olduğu veri sayısına bakarak kümelerin aktif mi yoksa pasif mi olacağına karar vermektedir. Eğer kümelerin sahip olduğu veri sayısı kullanıcının belirlediği ön tanımlı bir değişken olan N değerine eşit veya daha büyükse bu kümeler aktif kümeler olarak tanımlanır. Aksi durumdaki kümeler ise pasif kümeler olarak atanır. FlagActiveClusters fonksiyonunun karmaşıklığı küme sayısına (m) bağlıdır. Bu nedenle Karmaşıklığı O(m)'dir. Kümelerin aktif mi yoksa pasif mi olacağına karar veren FlagActiveClusters fonksiyonunun sözde kodu Şekil 3.25'te görüldüğü gibidir.

**Algorithm 9:** FlagActiveClusters

```
Input: C; //Küme tablosu
bufferedData; //Tamponlamış veriler
N; //Küme oluşturmak için gerekli mininmum veri sayısı
d; //Nitelik sayısı
Output: C; //Küme tablosu
bufferedData; //Tamponlanmış veriler
foreach cluster C_i ∈ C //Tüm kümeler için do
    if dataSize(C_i) >= N//i kümesinin verisi N'den büyük eşitse
     then
        C_iActive ← true; //i kümesini aktif olarak işaretle
    end
    else
        C_iActive ← false;//i kümesini pasif olarak işaretle
    end
end
```

Şekil 3.25. FlagActiveClusters fonksiyonunun sözde kodu

SelectFeatures algoritması standart sapma tabanlı bir öznitelik seçme işlemi gerçekleştirir. Ancak özniteliklerin standart sapmasını almadan önce her niteliği z-skor normalizasyona göre normalize eder. SelectFeatures fonksiyonu her nitelik için verileri zscore ile normalize ettiğinden karmaşıklığı veri sayısına (n) ve nitelik sayısına (d) bağlıdır. Dolayısıyla karmaşıklığı O(dn)'dir. SelectFeratures algoritmasının sözde kodu Şekil 3.26'da görüldüğü gibidir.

```
Algorithm 10: SelectFeatures
Input: C; //Küme tablosu
bufferedData; //Tamponlamış veriler
deletedData; //Silinen veriler
coeffThreshold; //Öznitelik seçme eşik değeri
Output: newBufferedData; //Yeni niteliklele tamponlanmış veri
newDeletedData; //Yeni niteliklele silinmiş veriler
newD //Seçilen öz nitelikler
Data ← bufferedData + deletedData;//Silinmiş ve tamponlanmış
                                  // verileri Data değişkenine ata
reset all bits of SD to 0; //Tüm nitelikleri seçilmemiş işaretle
foreach feature f ∈ Data //Her nitelik için do
    NormalizedData ← zscore(f);//z-skor ile normalize et
    StdOfFeatures ← std(NormalizedData);//Standart sapma bul
end
sort StdOfFeatures in Desc order;//StdOfFeatures azalan sırada
 sırala
summation ← 0; //Toplamı sıfırla
foreach stdDev ∈ StdOfFeatures //Her nitelik için do
    if summation <= coeffThreshold //Toplam coeffThreshold'a
                                   //ulaşmadıysa then
        summation ← summation + stdDev;//Niteliğin std'sini ekle
        set stdDev^th bit of SD to 1;//İlgili niteliği seçildi işaretle
    end
end
newD ← sum(SD);//Seçilen nitelikleri yeni nitelikler olarak ata
newBufferedData←select features, which set to 1 in SD, of
 buffredData;//Tamponlanmış verileri yeni niteliklere göre ayarla
newDeletedData←select features, which set to 1 in SD, of
 deletedData;//Silinmiş verileri yeni niteliklere göre ayarla
```

Şekil 3.26. SelectFeatures fonksiyonunun sözde kodu

KD-ARFS Stream algoritması yukarıda açıklanan temel fonksiyonların dışında yardımcı fonksiyonlardan da faydalanır. Bunlar bir veri ile bir küme merkezi veya iki küme merkezi arasındaki uzaklığı hesaplayan findDistance, belirli verileri bir K-Boyutlu ağaca yerleştiren BuildKDTree ve K-Boyutlu ağaçta r ve N değerlerine göre alan araştırması işlemi yapan rangeSearch fonksiyonlarıdır.

BuildKDTree fonksiyonu aldığı verileri bir K-Boyutlu ağaca yerleştirir. BuildKDTree recursive bir yapıda çalışarak verileri sağ ve sol alt ağaçlara yerleştirir. Bu şekilde çalışarak oluşturduğu ağacı döndürür. BuildKDTree fonksiyonunun karmaşıklığı veri sayısına (n) ve her verinin sahip olduğu nitelik sayısına (d) bağlıdır. Recursive çalıştığından her adımda

veriyi ikiye bölerek çalışmaktadır. Dolayısıyla karmaşıklığı O(dnlogn)'dir BuildKDTree fonksiyonunun sözde kodu Şekil 3.27'de görüldüğü gibidir.

**Algorithm 11:** BuilKdTree

```
Input: Data; //Ağaca yerleştirilecek veri
Output: tree; //KB ağaç
if numOfData = 1 //Veri sayısı 1 ise then
  | return Data; //Veriyi döndür
end
else if depth is even //Derinlik çift ise then
  | split Data into two subsets through median of X vertically as
  |   Data1 and Data2; //Veriyi düşey olarak olarak böl
end
else
  | split Data into two subsets through median of Y horizontally as
  |   Data1 and Data2;//Veriyi yatay olarak olarak böl
end
create node v;//Yeni v düğümünü oluştur
v_left ← BuildKdTree(Data1,depth+1);//Data1'i gönder
v_right ← BuildKdTree(Data1,depth+1);//Data2'yi gönder
Assign v_left as left child of v and v_right as right child;//v_left'i v'nin
                                  //sol çocuğu v_right'ı sağ çocuk olarak ata
return v; //v'yi döndür
```

Şekil 3.27. BuildKDTree fonksiyonunun sözde kodu

rangeSearch fonksiyonu K-Boyutlu ağaçta alan araştırması yapar. Ağaçta kullanıcının belirlediği r yarıçapta minimum N tane veri var mı diye araştırır. rangeSearch fonksiyonu da BuildKDTree fonksiyonu gibi recursive olarak çalışır. rangeSearch fonksiyonunun taranan bölgedeki veri sayısı (k) ve veri sayısına bağlıdır. Dolayısıyla karmaşıklığı O(k+$\sqrt{n}$)'dir. rangeSearch fonksiyonunun sözde kodu Şekil 3.28'de görüldüğü gibidir.

```
Algorithm 12: rangeSearch
Input: v; //KB ağacı
r; //Yarıçap eşik değeri
Output: Index of data in the range; //Verilerin indeksleri
if v is leaf //Eğer v bir yaprak ise then
    Report the stored at v if it lies in r;//r içerisinde olarak bildir
end
else
    Consider the left subtree;//Sol alt ağacı incele
    if left subtree fully contained in the r//Eğer sol alt ağaç
     tamamıyla r içerisindeye then
        Report the left subtree;//Sol alt ağaçta olarak bildir
    end
    else if left subtree intersects r //Sol ağaç r ile kesişiyorsa then
        rangeSearch(leftsubtree, r);//Sol ağacı araştır
    end
    Consider the right subtree;//Sağ alt ağacı incele
    if right subtree fully contained in the r //Eğer sağ alt ağaç
     tamamiyle r içerisindeyse then
        Report the right subtree;//Sağ alt ağaçta olarak bildir
    end
    else if right subtree intersects r //Sağ ağaç r ile kesişiyorsa then
        rangeSearch(rightsubtree, r);//Sol ağacı araştır
    end
end
```

Şekil 3.28. rangeSearch fonksiyonunun sözde kodu

findDistance fonksiyonu bir veri ve bir küme merkezi veya iki küme merkezi arasındaki uzaklığı hesaplar. Uzaklık ölçütü olarak Öklid uzaklığını kullanır. findDistance fonksiyonunun karmaşıklığı nitelik sayısına (d) bağlıdır. Bu nedenle karmaşıklığı O(d)'dir. findDistance fonksiyonunun sözde kodu Şekil 3.29'da görüldüğü gibidir.

**Algorithm 13:** findDistance

**Input:** $Data_A$; //*A noktasının koordinatları*
$Data_B$; //*B noktasının koordinatları*
d; //*Nitelik sayısı*
**Output:** dist; //*Mesafe*
sum ← 0; //*Toplamı sıfırla*
**foreach** $i \in d$ //Her nitelik için **do**
  sum ← sum + ($Data_{iA}$ - $Data_{iB}$) * ($Data_{iA}$ - $Data_{iB}$);//*İki nokta arasındaki mesafenin karesini toplama ekle*
**end**
dist ← sqrt(sum);//*Karekökünü bul*
return dist; //*Sonucu döndür*

Şekil 3.29. findDistance fonksiyonunun sözde kodu

### 3.7.3. Algoritma karmaşıklığı

Algoritmanın mümkün olduğunca basit çalışarak performansı düşürmemesine dikkat ettik. Çünkü geliştirdiğimiz yöntem akan veri kümeleme işlemi gerçekleştirecek. Zaman kısıtı olan bir durum söz konusudur. Bu nedenle algoritma makul sürede cevap vermelidir.

*n*, her adımda üzerinde işlem yapılan veri, *d* her verinin sahip olduğu nitelik sayısı, *m* oluşturulmuş küme sayısı ve k k-boyutlu ağaçta taranan alandaki veri sayısı olmak üzere fonksiyonların karmaşıklığı Çizelge 3.2'de görüldüğü gibidir.

Çizelge 3.2. Fonksiyonların karmaşıklığı

| Fonksiyon Adı | Karmaşıklığı |
|---|---|
| AgingAll | $O(n)$ |
| NewClusterAppear | $O(dn\log n + nk + n\sqrt{n})$ |
| CheckOverlapClusters | $O(m^2)$ |
| CheckSplit | $O(dmn\log n + mnk + mn\sqrt{n})$ |
| FindClosestCluster | $O(dmn)$ |
| UpdateCenters | $O(dmn)$ |
| UpdateRadius | $O(dmn)$ |

Çizelge 3.2. (Devam) Fonksiyonların karmaşıklığı

| Fonksiyon Adı | Karmaşıklığı |
|---|---|
| FlagActiveClusters | $O(m)$ |
| SelectFeatures | $O(dn)$ |
| BuildKDTree | $O(dn\log n)$ |
| rangeSearch | $O(k+\sqrt{n})$ |
| findDistance | $O(d)$ |

# 4. DENEYSEL ÇALIŞMA

## 4.1. Algoritma Başarı Ölçüm Metrikleri

Kümeleme modelinin başarısını doğru bir şekilde ölçmek kümeleme yaklaşımlarının önemli bileşenlerinden biridir. Kümeleme değerlendirme yöntemleri bir anlamda kümeleme modelinin tutarlılığını ortaya koymaya çalışır. Kümeleme değerlendirme yöntemleri iki gruba ayrılır:

- Dâhili yöntemler
- Harici yöntemler

### 4.1.1. Dâhili yöntemler

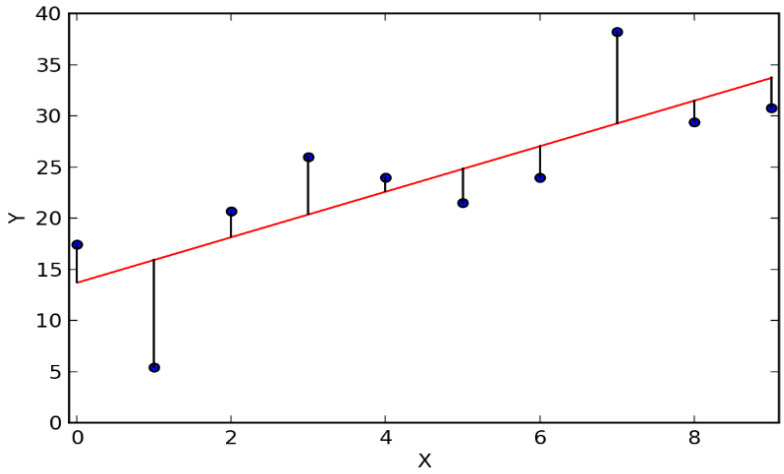
Bu tür yöntemler yapılan kümelemeyi değerlendirirken verinin kendisini kullanır. Yani modelin ortaya koyduğu kümeleri çeşitli hesaplamalarla değerlendirir. Özünde aynı kümede bulunan verilerin birbirlerine benzerliklerini ve diğer kümelerle farklılıklarını hesaplamaya çalışır. Bu yöntemlerin bazılarını şu şekilde sıralayabiliriz:

SSE (sum of squared errors)

Kendi içerisinde verilerin hata payını hesaplar. Verilerin hata paylarının karelerinin toplamını bulur. $n$ veri sayısını, $y_i$, $i$. verinin değerini ve $\overline{y_i}$ tahminlerin ortalaması olmak üzere SSE Eş. 4.1 ile hesaplanır.

$$SSE = \sum_{i=1}^{n} (y_i - \bar{y}_i)^2 \tag{4.1}$$

Şekil 4.1'de de görüldüğü gibi her verinin ortalama doğrusuna olan uzaklığı hata payını göstermektedir. Yani veri doğruya ne kadar yakın olursa hata paya o kadar azalır.




Şekil 4.1. SSE örneği

Silhouette indeks

Şekil 4.2'de de görüldüğü gibi, verinin bulunduğu kümedeki her veriye uzaklığının ortalamasının en yakın olduğu kümenin verilerine uzaklığını baz alarak kümelemenin ne kadar başarılı olduğunu ortaya koymaya çalışır ve Eş. 4.2 ile hesaplanır.

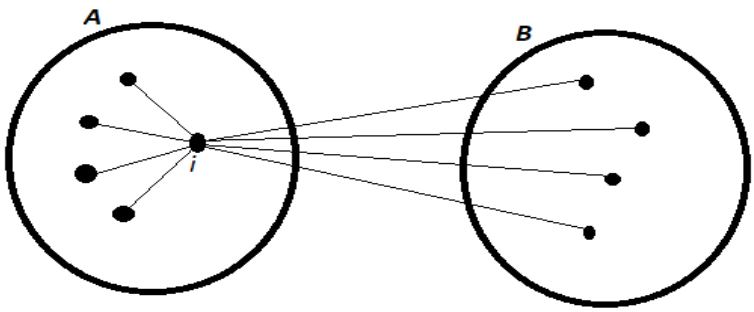
$$S_i = \frac{b - a}{\max(a, b)} \tag{4.2}$$

Burada;

$S_i$: $i$ verisinin Silhouette indeks değerini göstermektedir.

$a$: $i$ verisinin bulunduğu kümedeki diğer verilere olan uzaklıklarının ortalaması;

$b$: $i$ verisinin en yakın kümenin elemanlarına olan uzaklıklarının ortalaması;




Şekil 4.2. Silhouette indeks örneği

Rand indeks (RI) ve adjusted rand indeks (ARI)

Rand İndeks (RI) (Rand, 1971), bir kümedeki verinin değerini hesaplarken o veriyi aynı kümede olan diğer verilerle ikili olarak karşılaştırır. Bu işlemi aynı şekilde gerçek sınıf etiketleri için de yapar. Sonrasında bu yapılan işlemi ikili olarak karşılaştırır. Aynı mı yoksa farklı mı olduğunu bir tabloda tutar. Bu tablo üzerinden RI değerini hesaplar. $X_{Gerçek}$, $X$ parçasına ait gerçek sınıf etiketleri ve $X_{Tahmin}$ $X$ parçasına ait tahmin, $Uyumlu_{Aynı}$ ikili karşılaştırmanın hem $X_{Gerçek}$'de hem $X_{Tahmin}$'de aynı olmasını, $Uyumlu_{Farklı}$ ikili karşılaştırmanın birinde farklı olmasını, $Uyumsuz_{Aynı}$ ikili karşılaştırmanın hem $X_{Gerçek}$'de hem $X_{Tahmin}$'de aynı olmasını, $Uyumsuz_{Farklı}$ ikili karşılaştırmanın birinde farklı olmasını ifade etmek üzere RI, Eş. 4.3 ile hesaplanır.

$$RI = \frac{Uyumlu_{Aynı} + Uyumlu_{Farklı}}{Uyumlu_{Aynı} + Uyumlu_{Farklı} + Uyumsuz_{Aynı} + Uyumsuz_{Farklı}} \tag{4.3}$$

Adjusted Rand Index (ARI) (Hubert ve Arabie, 1985), RI üzerinden hesaplanan bir kümeleme başarısı değerlendirme yöntemidir ve Eş. 4.4 ile hesaplanır.

$$ARI = \frac{RI - Beklenen\ RI}{Max\ RI - Beklenen\ RI} \tag{4.4}$$

Jaccard indeks

Jaccard index (Jaccard, 1901), modelin etiketlediği verileri gerçek etiketler ile karşılaştırır. Modelin verileri atadığı kümenin etiketi *A*, söz konusu verilerin gerçek küme etiketi *B* olmak üzere Jaccard index (*Js*) iki kümenin kesişimlerinin birleşimlerine oranıdır ve Eş. 4.5 ile hesaplanır.

$$Js = \frac{A \cap B}{A \cup B} \tag{4.5}$$

### 4.1.2. Harici yöntemler

Bu tür yöntemleri kullanabilmek için çeşitli yollarla elde edilmiş sınıf etiketinin olması gerekir. Yani aslında bir verinin hangi kümeye ait olduğunu gösteren bir etiketi mevcuttur.

Bu nedenle bu tür yöntemler kümeleme modelinin ürettiği küme etiketinin gerçek sınıf etiketi ile karşılaştırır. Başlıca yöntemler şunlardır:

Purity

Purity önerilen modelin yaptığı kümeleme yaklaşımının saflık derecesini hesaplar. Her küme için içerisinde barındırdığı verilerden sayısı en fazla olan verilerin toplam veri sayısına oranıdır. Bu işlemi tüm kümeler için yapar ve elde edilen sonuçların tamamını toplar. Bu toplamın toplam veri sayısına oranı Purity'dir. *K* küme sayısını, $C_i^d$, $C_i$ kümesindeki baskın sınıfa ait veri sayısını ifade etmek üzere Purity, Eş. 4.6 ile hesaplanır.

$$Purity = \frac{\sum_i^K \frac{\left|C_i^d\right|}{|C_i|}}{K} x100 \tag{4.6}$$

Entropy ve bilgi kazanımı

Entropy veya Shanon entropy (Shannon, 2001) belirsizliğin oranını hesaplamayı amaçlayan bir yaklaşımdır. Örneğin bir torbadaki bilyelerin rengi farklı ise çekilen bilyenin hangi renk olacağını kesin olarak bilebilmek mümkün değildir. Ancak bütün bilyelerin rengi aynı ise çekilecek bilyenin rengini bilmek mümkündür. Dolayısıyla Entropy bir olayın olma olasılığını hesaplamayı amaçlar. *b* logaritmanın tabanını, *n* toplam olasılık sayısını göstermek üzere *H(x)* bilgi kazancı Eş. 4.7 ile hesaplanır.

$$H(x) = -\sum_{i=1}^{n} P(x_i) log_b P(x_i) \tag{4.7}$$

Accuracy

Accuracy kümelenen verilerin gerçekte ne kadarının doğru sınıfa atandığını bulmaya yarayan bir kümeleme başarısı değerlendirme yaklaşımıdır. $a_i$, $i$ kümesine atanmış ve gerçekte de $i$ kümesine ait olan verilerin sayısı ve $n$ de veri setindeki toplam veri sayısı olmak üzere Accuracy Eş. 4.8 ile hesaplanır.

$$Accuracy = \frac{\sum_{i=1}^{k} a_i}{n} \tag{4.8}$$

Precision ve recall

Precision ve Recall en temel kümeleme başarısını değerlendirme yöntemleridir. Bir dokümandaki ilgili olan dokümanların sayısı $|R|$, R olduğu tahmin edilenlerin sayısı $|A|$ ve $R$'ye ait olduğu tahmin edilenlerden gerçekte R'ye ait olanlar $|R_a|$ olmak üzere Precision ve Recall değerleri Eş. 4.9 ve 4.10 numaralı eşitlikler ile hesaplanır.

$$Precision = \frac{|R_a|}{A} \tag{4.9}$$

$$Recall = \frac{|R_a|}{|R|} \tag{4.10}$$

F-Score (F-measure)

Kümeleme başarısını değerlendirme yöntemlerinden bir diğeri de F-Score'dur (Brun, Sima, Hua, Lowey, Carroll, Suh ve Dougherty, 2007). F-Score Precision ve Recall değerlerinin harmonik ortalamasıdır ve Eş. 4.11 ile hesaplanır.

$$F - Score = \frac{2.Precision.Recall}{Precision + Recall} \tag{4.11}$$

Yukarıda saydığımız küme başarısı değerlendirme yöntemlerinin dışında Calinski-Harabasz indeks (Caliński ve Harabasz, 1974), I indeks (Maulik ve Bandyopadhyay, 2002), Dunn indeks (Dunn†, 1974), Davies-Bouldin indeks (Davies ve Bouldin, 1979), Fowlkes & Mallows (Wallace, 1983), BIC (Bayesian Information Cirtaria) indeks (Raftery, 1986), NMI (Normalized Mutual Information) (Strehl ve Ghosh, 2003) ve Entropy (Shannon, 2001) gibi küme başarısı değerlendirme yöntemleri de vardır.

Dahili yöntemlerde bir küme etiketine ihtiyaç olmadığından modelin kümeleme yaparken verdiği etiketin bir önemi yoktur. Bu nedenle Gerçekte 1 numaralı küme olarak etiketlenmiş verilerin model tarafından 2 veya 3 olarak etiketlenmesinin bir farkı yoktur ve kümeleme başarısını etkilemez. Oysaki harici yöntemlerde verilen etiketin gerçek küme etiketi ile uyuşmaması, sonucu doğrudan etkiler. Çalışma zamanı açısından bakıldığı zaman ARI, Jaccard indeks, Fowlkes & Mallows ve Silhouette indeks gibi kümeleme başarısı değerlendirme yöntemlerinin çalışma zamanı diğer yöntemlerden daha yüksektir.

## 4.2. Akan Veride Parametre Optimizasyonu

Akan veri gibi çok büyük verilerin söz konusu olduğu durumlarda en iyi parametrelerin hangisi olduğunu tespit etmek çok zordur. Bu zorluğun temel nedeni hangi parametrelerin en iyi sonucu verdiğini kestirmenin zor olmasıdır. Bu probleme çözüm olarak çeşitli yöntemler önerilmiştir. Bunların başında Grid Search ve Random Search gelmektedir.

Grid Search olası bütün parametreler için modeli test ederek en iyi sonucu hangi parametrelerin verdiğini test eder. Bu yaklaşım için en önemli problem veri miktarı büyük olduğu zaman sonuç elde etmenin çok uzun zaman almasıdır. Çünkü olası tüm parametreler test edilmektedir. Bir diğer problem sonucu değerlendirmek için bir amaç fonksiyonunun olması gerekir. Bu amaç fonksiyonu bazen Purity testi, bazen Accuracy, bazen ARI veya herhangi başka bir test parametresi olabilir.

Random Search olası parametre matrisinden rastgele seçimler yaparak daha çabuk sonuç üretmeyi amaçlamaktadır. Grid Search'e göre daha çabuk sonuç üretmesine rağmen en iyi parametreleri bulmayı garanti etmez.

Bayesian Optimizasyonu yönteminde önceki iki yöntemden farklı olarak parametreleri de zamana bağlı olarak optimize etmeye çalışır. Bu yöntemde çok iyi kurgulanmış bir modelin belirlenmesi çok önemlidir. Amaç, parametreler ile modelin ürettiği sonuçlar arasındaki ilişkiyi tespit ederek bir sonraki adımda kullanılacak parametreleri belirlemektir. Performans açısından diğer iki yönteme göre daha iyi olmasına rağmen kullanımı zordur.

## 4.3. Deneysel Çalışma

Geliştirdiğimiz yöntem, yöntemimizle çok sayıda ortak özelliğe sahip olmaları ve başarılı sonuçlar vermeleri nedeniyle SE-Stream (Chairukwattana ve diğerleri, 2013), pcStream (Mirsky ve diğerleri, 2015), CEDAS (Richard Hyde ve diğerleri, 2017) ve DPStream (Xu ve diğerleri, 2017) algoritmaları ile karşılaştırılmıştır. Karşılaştırma işlemi hem kümeleme başarısı, hem de çalışma zamanı açısından gerçekleştirilmiştir. Tüm karşılaştırma işlemlerinde Intel® Core™ i5-4460S CPU 2.90 GHz işlemcili ve 8 GB RAM kapasiteli ve Windows 10 yüklü bir bilgisayarda, Matlab 2017b ile gerçekleştirilmiştir.

### 4.3.1. Kullanılan veri setleri

Deneysel çalışmada UCI'nin KDD CUP '99 ile Fisheriris, Breast Cancer, Occupancy; DPStream algoritmasında kullanılan ExclaStar, MrData veri setleri ve KD-ARFS Stream algoritmasında kendi ürettiğimiz IdealData veri setleri kullanılmıştır. Bu veri setlerinden üçü gerçek veri seti iken, üçü sentetik veri setleridir. Bu veri setlerinin özellikleri Çizelge 4.1'de görüldüğü gibidir.

Çizelge 4.1. Test işleminde kullanılan veri setlerinin özellikleri

| Veri Seti | Türü | Veri Miktarı | Nitelik Sayısı | Sınıf Sayısı | Özellik |
|---|---|---|---|---|---|
| KDD | Gerçek | 494 021 | 38 | 23 | Akan veri kümelemede sıkça kullanılır |
| Fisher Iris | Gerçek | 150 | 4 | 3 | Sıkça kullanılır |
| Breast Cancer | Gerçek | 699 | 9 | 2 | Veri Kümelemede sıkça kullanılır |
| MrData | Sentetik | 42 470 | 2 | 4 | Sapan veri içeriyor |
| ExclaStar | Sentetik | 755 | 2 | 3 | DPStream de kullanılan sentetik |
| IdealData | Sentetik | 211 | 2 | 5 | Kendi oluşturduğumuz algoritmamıza uygun bir veri seti |
| Occupancy | Gerçek | 8143 | 6 | 2 | Zaman damgalı veri seti |

KDD CUP'99 veri seti

KDD'99 veri seti, 9 temel ve 32 türetilmiş olmak üzere toplam 41 nitelikten oluşmaktadır. Toplamda 494 000 civarı veriden oluşan bir veri setidir ve ağ saldırıları ile ilgili verileri içeren bir veri setidir. Veri madenciliği alanında çok sık kullanılmaktadır.Temel anlamda bu nitelikler üç ana gruba ayrılabilir. Bunlar:

• İçerik ile ilgili nitelikler (contecnt features) : duration, src_bytes, dst_bytes, wrong_fragment, urgent vs.

• Network trafiği ile ilgili nitelikler (host-based traffic features): hot, num_failed_logins, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shell vs.
• Zamana bağlı trafik ile ilgili nitelikler (time-based traffic features): serror_rate, rerror_rate, same_srv_rate, Diff_srv_rate, Srv_error_rate vs.

Veri setinde 23 çeşit etiket bulunmaktadır. Bunlardan bazıları back, buffer_overflow, ftp_write, guess_passwd, imap, ipsweep, land, loadmodule, multihop ve normal gibi etiketlerdir. Bu etiketlerin her biri bir saldırı çeşidini temsil etmektedir.

Fisher Iris

Şekil 4.3'te görüldüğü gibi her veri 4 nitelik ve bir etiketten oluşur. Her veri setosa, versicolor ve virginica kümelerinden birine aittir. Bu veri setinde iris çiçeğinin yaprak ölçümlerine ait veriler bulunmaktadır. Bu ölçümler sepal-length (alt yaprak uzunluğu cm), sepal-with (alt yaprak genişliği cm), pedal-length (üst yaprak genişliği cm), pedal-width (üst yaprak uzunluğu cm) alanlarından oluşmaktadır. Bu veri setinde temel amaç bu ölçümleri kullanarak ölçüm yapılan çiçeğin hangi türe ait olduğunu tespit etmektir.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 5.1000 | 3.5000 | 1.4000 | 0.2000 | 1 |
| 2 | 4.9000 | 3 | 1.4000 | 0.2000 | 1 |
| 3 | 4.7000 | 3.2000 | 1.3000 | 0.2000 | 1 |
| 4 | 4.6000 | 3.1000 | 1.5000 | 0.2000 | 1 |
| 5 | 5 | 3.6000 | 1.4000 | 0.2000 | 1 |
| 6 | 5.4000 | 3.9000 | 1.7000 | 0.4000 | 1 |
| 7 | 4.6000 | 3.4000 | 1.4000 | 0.3000 | 1 |
| 8 | 5 | 3.4000 | 1.5000 | 0.2000 | 1 |
| 9 | 4.4000 | 2.9000 | 1.4000 | 0.2000 | 1 |
| 10 | 4.9000 | 3.1000 | 1.5000 | 0.1000 | 1 |

Şekil 4.3. Örnek Fisher Iris verisi

Breast Cancer veri seti

Breast Cancer veri seti göğüs kanseri ile ilgili verileri içeren bir veri setidir. 9 nitelik ve 1 sınıf etiketinden oluşmaktadır. Sınıf etiketi tümörün iyi huylu ya da kötü huylu mu olduğunu

gösterirken nitelikler clump kalınlığı, hücre boyutu düzenliliği, hücre şekli düzenliliği, marjinal yapışma, tek epital hücre boyutu, bare nuclei, bland kromatin, normal nükleoli ve mitoz alanlarından oluşmaktadır. oluşmaktadır. Verilerin 458'i iyi huylu sınıfa, 241'i kötü huylu sınıfa ait olmak üzere toplam 699 örnek bulunmaktadır.

MrData

MrData veri seti DPStream algoritmasında kullanılan ve algoritmanın sapan verilere karşı direncini ölçmek için üretilmiş bir veri setidir. Veri setinin %10'u sapan verilerden oluşmaktadır. Veri setinde 2 nitelik ve bir sınıf etiketi olarak toplam 42 470 adet örnek bulunmaktadır.

ExclaStar

ExclaStar veri seti de yine MrData veri seti gibi DPStream algoritmasında kullanılan bir veri setidir. Bu veri seti daha çok evrimsel bir veri seti söz konusu ise algoritmasnın nasıl tepki verdiğini ölçmek için üretilmiştir. Veri seti 2 adet nitelikten ve 1 sınıf etiketinden oluşmaktadır. Sınıf etiketi ise üç çeşit etiket almaktadır. Veri seti toplam 755 adet örnekten oluşmaktadır. Bu veri setinde iki kümenin birleştirilmesi söz konusudur. 708. örnekten sonra 2. ve 3. kümelerin birleştirilmesi gerekmektedir.

IdealData

IdealData veri seti bu çalışmada üretilmiş bir veri setidir. Bu veri setinin amacı önerdiğimiz yöntemin yaptığı işlemleri göstermektir. Bu nedenle bu veri seti kümelerin merge ve split işlemlerinin nasıl gerçekleştirildiğini göstermek adına önemli bir veri setidir. IdealData veri seti 211 örnekten, her örnek 2 nitelik ve 1 sınıf etiketinden oluşmaktadır. Her veri 5 farklı sınıftan herhangi birine ait olabilmektedir. Veri akarken 80. örnekten sonra 2 ve 3 numaralı kümeler çakıştığı için birleştirilmektedir. 180. örnekten sonra ise 2 numaralı küme bölünmektedir. Bunun yanında zamanla bazı kümeler deaktive edilirken yeni veriler alarak sonradan tekrar aktive edilebilmektedir.

Occupancy

Occupancy veri seti, UCI'nin sıcaklık, nem, ışık ve karbondioksit gibi parametreleri kullanarak odaların dolu olup olmadığını belirten bir veri setidir. Occupancy veri seti 7 nitelik ve 1 sınıf etiketinden oluşur. Toplam 20560 veriden oluşmaktadır. Occupancy veri setinin en önemli özelliği bir zaman damgasına sahip olmasıdır. Bu çalışmada kullanılmasının nedeni de budur. Amaç gelen veri miktarının çok fazla olması durumunda önerdiğimiz yaklaşımın vereceği tepkiyi ölçmektir. Ancak gerçek zaman damgaları sabit 1 dk.'dır. Yöntemimizin tepkisini ölçmek adına bu kısım rastgele olacak şekilde değiştirilmiştir. Buna göre veriler arasındaki zaman birkaç milisaniye ile değişmektedir.

### 4.3.2. Karşılaştırma yapılan akan veri kümeleme algoritmaları

Önerdiğimiz yöntemin başarısını ölçmek için SE-Stream (Chairukwattana ve diğerleri, 2013), pcStream (Mirsky ve diğerleri, 2015), CEDAS (Richard Hyde ve diğerleri, 2017) ve DPStream (Xu ve diğerleri, 2017) algoritmaları ile karşılaştırılmıştır. Çizelge 4.2'de adı geçen algoritmaların özellikleri kısaca karşılaştırılmaktadır.

Çizelge 4.2. Karşılaştırma yapılan algoritmaların özellikleri

| Algoritma | Yıl | Avantajlar | Dezavantajlar | Evrimsel değişimi destekliyor | Kümeleme türü | Önemi |
|---|---|---|---|---|---|---|
| SE-Stream (Chairukwattana, Kangkachit, Rakthanma | 2013 | E-Stream algoritmasına göre performans daha iyi, algoritma optimize edilmiştir. Boyut indirgemeyi | Çok sayıda parametrenin tanımlanması gerekir. Devamlı kümelerde performans düşmektedir. | Evet | Hiyerarşik | Beş evrimsel yapıyı desteklemesi |
| pcStream (Mirsky ve diğerleri, 2015) | 2015 | Kümelerin çakışmasını tespit edebilmektedir. | Sapan verilerden etkilenmekte ve başarı oranı düşmektedir. | Evet-Kısmi | Model tabanlı | Model tabanlı bir yapı kullanması |
| DPStream (Xu ve diğerleri, 2017) | 2017 | Tamamıyla çevrimiçi ve farklı şekilleri destekler. Ayrıca çok hızlı çalışan bir algoritmadır. | Çok sayıda parametre var ve parametrelerin çok doğru seçilmesi gerekir. | Evet-Kısmi | Hiyerarşik ve Yoğunluk | Çok hızlı çalışması |
| CEDAS (Richard Hyde ve diğerleri, | 2017 | Tamamıyla çevrimiçi çalışan ve farklı şekildeki kümeleri tespit edebilir. | Mikro kümelerin birleşimi ile oluşan makro kümelerin doğru tanımlanabilmesi optimum yarıçapın | Evet-Kısmi | Hiyerarşik | Mikro-küme tabanlı bir algoritma |

SE-Stream algoritması

SE-Stream (Chairukwattana ve diğerleri, 2013) algoritması E-Stream (Udommanetanakit ve diğerleri, 2007) algoritmasının boyut indirgeme işlemini gerçekleştiren gerçek zamanlı olarak küme oluşturma, silme, iki kümeyi birleştirme, bir kümeyi ikiye bölme ve kümelerin yapısında olan değişimleri göz önünde bulunduran bir versiyonudur. Amaç E-Stream algoritmasını hem performans açısından hem de kümeleme başarısı açısından geliştirmektir.

Temel anlamda küme yapısını FCH olarak tarif edilen Fading Cluster Structure yapısında tutmaktadır. Bu yapı sayesinde kümeleri ağırlıklandırarak zamana bağlı olarak kümelerin pasif yapılıp yapılmayacağına karar vermektedir. Küme ağırlığını hesaplamak için kullanılan fading işlemi Eş. 4.12 üzerinden hesaplanmaktadır.

$$f(t) = 2^{-2\lambda t} \tag{4.12}$$

Yukarıdaki eşitlikten faydalanarak küme yapısı ise Eş. 4.13 ile hesaplanmaktadır. Burada *j* bir niteliği ifade ederken $T_i$, *i* verisinin geldiği zamanı ifade etmektedir. *x* ise o verinin değerini ifade etmektedir.

$$FC1 = \sum_{i=1}^{N} f(t - T_i).(x_i^j) \tag{4.13}$$

*FC2* de *t* anında her niteliğin ağırlıklandırılmış kareleri toplamını hesaplar. *j*'inci niteliğin *FC2* değeri Eş. 4.14 ile hesaplanır.

$$FC2 = \sum_{i=1}^{N} f(t - T_i).(x_i^j)^2 \tag{4.14}$$

Bir kümenin ağırlığını bulmak için W(t) eşitliği kullanılmaktadır. W(t) Eş. 4.15 ile hesaplanmaktadır.

$$W(t) = \sum_{i=1}^{N} f(t - T_i) \tag{4.15}$$

*BS(t), t* anındaki niteliklerin indirgenmiş bir alt kümesi ve *j* bir niteliği belirtmek üzere *j*'inci niteliğin bit vektörü Eş. 4.16 ile hesaplanır.

$$BS^{j} = \begin{cases} 1 & Şayet \quad j \quad igili \quad kümedeyse \\ 0 & Kümede \quad yoksa \end{cases} \tag{4.16}$$

Kümelerin bölünüp bölünmeyeceğine karar vermek için α-bin histogram yapısı kullanılmaktadır. Ona da aşağıdaki eşitlik ile karar verilmektedir. *H(t)* verilerin α-bin histogramını, *j* niteliğinin *t* anındaki histogramı aşağıdaki gibi hesaplanmaktadır. *H(t)* eşit α genişliği içerisinde bulunan verilerin α-bin histogramı olmak üzere *j*'inci niteliğin *l*'inci binin histogramı Eş. 4.17 ile hesaplanır.

$$H_l^j(t) = \sum_{i=1}^{N} f(t - T_i).(x_i^j).(y_{i,l}^j) \tag{4.17}$$

Burada;

$$y_{i,l}^j = \begin{cases} 1 & Şayet \quad l.r + \max(x^j) \le x_i^j \le (l+1).r + \min(x^j); r = \dfrac{\max(x^j + \min(x^j)}{\alpha} \\ 0 & Aksi halde \end{cases} \tag{4.18}$$

Bir verinin bir kümeye olan mesafesini bulmak için aşağıdaki eşitlik kullanılmaktadır. *C* bir kümeyi, *X* bir veriyi, *n* nitelik sayısını ve $radius_c$ kümenin standart sapmasın göstermek üzere, verinin kümeye olan uzaklığı *dist(C,X)* Eş. 4.20 ile hesaplanır.

$$dist(C, X_i) = \frac{1}{n}. \sum_{j \in BS(t)} \left| \frac{center_C^j - x_i^j}{radius_C^j} \right| \tag{4.20}$$

Yine benzer şekilde iki küme arasındaki mesafe ise Eş. 4.21 ile hesaplanmaktadır. Burada *n* nitelik sayısını $C_a, A$ kümesinin merkezini, $C_b$, *B* kümesinin merkezini ifade ederken, $dist(C_a, C_b)$ A ve B kümeleri arasındaki mesafeyi ifade etmektedir.

$$dist(C_a, C_b) = \frac{1}{n}. \sum_{j \in bs(t)} \left| center_{Ca}^j - center_{Cb}^j \right| \tag{4.21}$$

Bu eşitlikler ışığında SE-Stream algoritmasının ana algoritması ve alt fonksiyonları Şekil 4.4'te görüldüğü gibi şekillenmektedir.

| **Algorithm** *SE-Stream* | SE-Stream Runtime-complexity | E-Stream Runtime-Complexity |
|---|---|---|
| 1: Retrieve new data $X_i$ | | |
| 2: FadingAll | O(K) | O(K) |
| 3: CheckSplit | O(Kd) | O(Kd) |
| 4: MergeOverlapCluster | $O(K^2l)$ | $O(K^2d)$ |
| 5: LimitMaximumCluster | $O(K^2d)$ | $O(K^2d)$ |
| 6: FlagActiveCluster | O(K) | O(K) |
| 7: If (Found new active cluster(s) or active cluster(s) turn into inactive cluster(s)) | | |
| 8: ProjectDimension | O(Kd) | n/a |
| 9: (minDistance, index) ← FindClosestCluster | O(Kl) | O(Kd) |
| 10: If minDistance < radius_factor | | |
| 11: Add $X_i$ to $FCHactive_{index}$ | | |
| 12: Else | | |
| 13: Create new FCH from $X_i$ | | |
| 14: Waiting for new data | | |

**Algorithm** *CCDistance($FCH_i$, $FCH_j$)*
1: Create $BS_{temp}$ and Set all bit in to 1
2: if ($FCH_i$ is Active Cluster and $FCH_j$ is Active Cluster)
3: $BS_{temp}$ = BS(FCHi) | BS(FCHj)
4: Else if (FCHi is Active Cluster Or FCHj is Active Cluster)
5: $BS_{temp}$ = BS(FCH is Active)
6: for k ← 1 to d
7: if ($BS_{temp}$ == 1)
8: distance = abs(Center($FCH_i$)$^k$ – Center($FCH_j$)$^k$)
9: d' = the number of bits in $BS_{temp}$ with value of 1
10: distance = distance / d'
11: return distance

**Algorithm** *FadingAll*
1: for i ← 1 to |FCH|
2: fading $FCH_i$
3: if ($FCH_i$.W < ε)
4: delete $FCH_i$

**Algorithm** *CheckSplit*
1: for i ← 1 to |FCHactive|
2: for j ← 1 to d
3: if (BS($FCHactive_i$)$^j$ == 1)
4: if ($FCHactive_i$ have split point)
5: split $FCHactive_i$
6: S ← S U {(i, |FCH|)}

**Algorithm** *CPDistance(FCHi, $X_i$)*
1: for k ← 1 to d
2: if (BS($FCH_i$)$^k$ == 1)
3: distance = abs(Center($FCH_i$)$^k$ – $X_i^k$ / Radius($FCH_i$)$^k$)
4: d' = the number of bits in BS($FCH_i$) with value of 1
5: distance = distance / d'
6: return distance

**Algorithm** *LimitMaximumCluster*
1: while |FCH| > K
2: for i ← 1 to |FCH|
3: for j ← i+1 to |FCH|
4: dist[i, j] ← CCDistance($FCH_i$, $FCH_j$)
5: (first, second) ← $argmin_{(i,j)}$(dist[i, j])
6: merge ($FCH_{first}$, $FCH_{second}$)

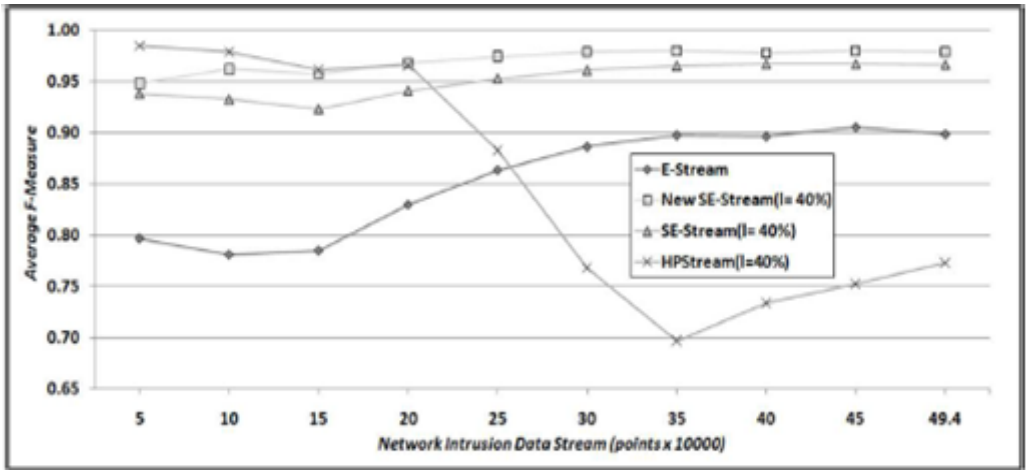
**Algorithm** *MergeOverlapCluster*
1: for i ← 1 to |FCHactive|
2: for j ← i + 1 to |FCHactive|
3: overlap[i,j] ← CCDistance ($FCHactive_i$, $FCHactive_j$)
4: m ← merge_threshold
5: if (overlap[i, j] > m * ($FCHactive_i$.sd + $FCHactive_j$.sd)
6: if (i, j) not in S
7: merge ($FCHactive_i$, $FCHactive_j$)

**Algorithm** *FindClosestCluster*
1: for i ← 1 to |FCHactive|
2: dist[i] ← CPDistance($FCHactive_i$, $X_i$)
3: (minDistance, i) ← min(dist[i])
4: return (minDistance, i)

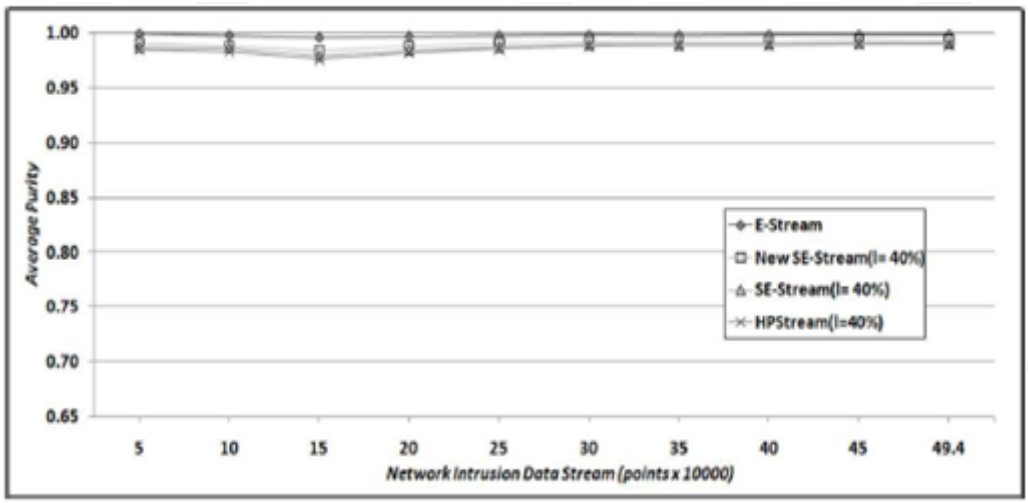
**Algorithm** *ProjectDimension*
1: Compute the |FCHactive| * d radius along d dimensions
2: Pick the |FCHactive| * l dimensions with the least radius
3: Set bit vector for each cluster reflecting its projected dimesion

Şekil 4.4 SE-Stream algoritması (Chairukwattana ve diğerleri, 2013)

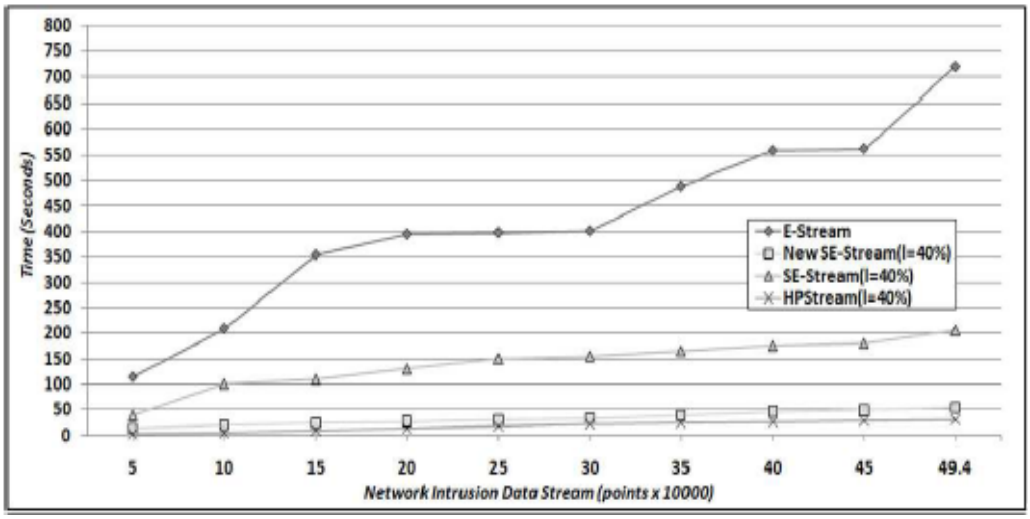
SE-Stream algoritmasının başarısını ölçmek için KDD CUP '99 veri seti kullanılmış ve sonuçlar E-Stream ve HPStream (Charu ve diğerleri, 2004) algoritmaları ile karşılaştırılmıştır. Şekil 4.5, 4.6 ve 4.7'de de görüldüğü gibi elde edilen sonuçlar SE-Stream algoritmasının hem kümeleme başarısı hem de çalışma zamanı açısından daha başarılı olduğunu göstermiştir.




Şekil 4.5. SE-Stream algoritmasının F-Skor sonuçları (Chairukwattana ve diğerleri, 2013)




Şekil 4.6. SE-Stream algoritmasının Purity testi sonuçları (Chairukwattana ve diğerleri, 2013)

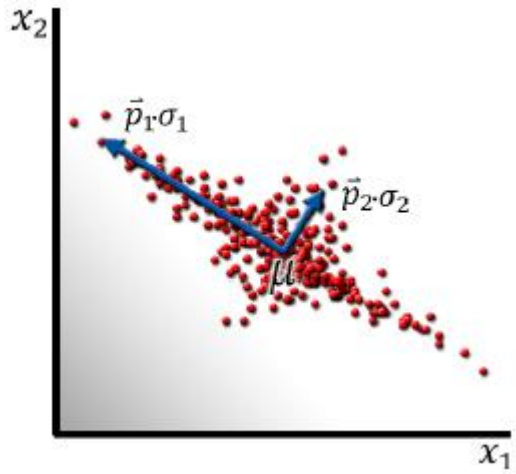



Şekil 4.7. SE-Stream algoritmasının çalışma zamanı (Chairukwattana ve diğerleri, 2013)

pcStream algoritması

pcStream (Mirsky ve diğerleri, 2015) akan veride kavram ve konseptleri zamansal ve uzaysal boyutta bulmayı amaçlayan bir akan veri kümeleme yaklaşımıdır. pcStream gerçek zamanlı olarak değişim ve evrimsel değişiklikleri tespit edebilmektedir. Temel anlamda sensörler üzerinden işlem yapan bir yaklaşımdır. Ayrıca, pcStream anomalileri tespit edebilmekte ve çok boyutlu verileri işleme yeteneğine sahiptir.

pcStream Principal Component Analysis (PCA) (Jolliffe, 2005) tabanlı bir kümeleme yapmaktadır. PCA nitelikler arasındaki ilişkiyi tespit etmek için kullanılmaktadır. Şekil 4.8'de kullanılan PCA yaklaşımının bir örneği görülmektedir.




Şekil 4.8. pcStream'de kullanılan örnek bir PCA (Mirsky ve diğerleri, 2015)

pcStream algoritması belirlenmiş parametrelere pek çok model önerebilmekte ve çok sayıda model olması da performansı düşürmektedir. Bu nedenle CluStream (Aggarwal ve diğerleri, 2003) algoritmasında kullanılan merge işlemi ile en yakın modeller birleştirilmektedir.

pcStream algoritmasının en kritik noktalarından biri yeni bir içeriğin ortaya çıkışını tespit etmektir. pcStream içeriklerin sabit bir dağılıma sahip olduğunu varsaymaktadır. Bu nedenle dağılıma uymayan bir içeriğin tespit edilmesi durumunda yeni içerik olarak atanmaktadır. Şekil 4.9'da pcStream algoritmasının sözde kodu görülmektedir.

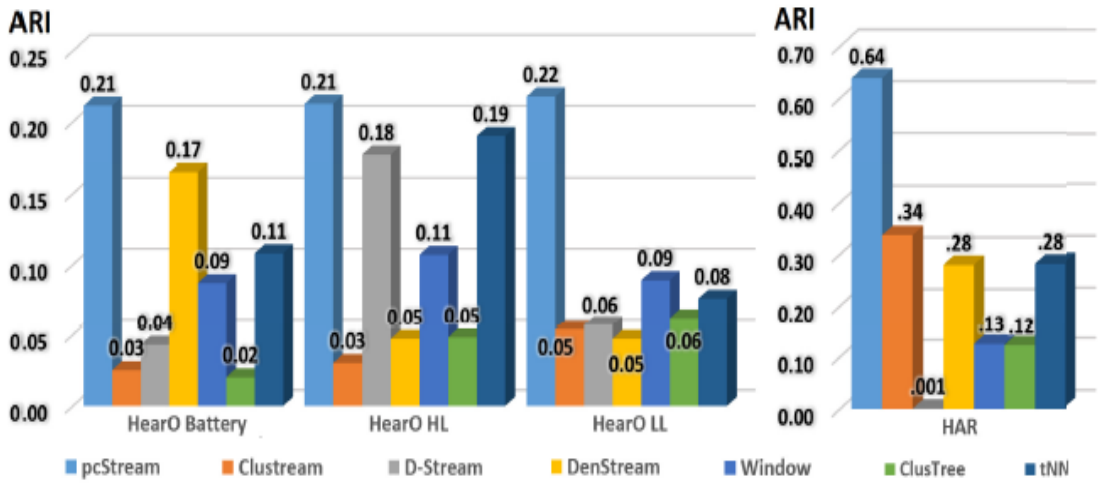
**Online Algorithm 1: pcStream $\{S\}$**

Input Parameters $\{\varphi, t_{min}, m, \rho\}$
Anytime Outputs: $\{c_t, d_{\boldsymbol{C}}(\vec{x}_t)\}$

1. $\boldsymbol{C} \leftarrow init(S, t_{min}, m, \rho)$
2. $c_t \leftarrow c_1$
3. $D \leftarrow \emptyset$
4. *loop*
   - 4.1. $\vec{x}_c \leftarrow next(S)$
   - 4.2. $scores \leftarrow d_{\boldsymbol{C}}(\vec{x}_c)$
   - 4.3. $i \leftarrow IndexOfMin(scores)$
   - 4.4. *if* $scores(i) < \varphi$
     - 4.4.1. $UpdateModel(c_i,\ Dump(D))$
     - 4.4.2. $UpdateModel(c_i, \vec{x}_c)$
     - 4.4.3. $c_t \leftarrow c_i$
   - 4.5. *else*
     - 4.5.1. $Insert(\vec{x}_c, D)$
     - 4.5.2. *if* $length(D) == t_{min}$
       - 4.5.2.1. $c \leftarrow CreateModel(Dump(D), m, \rho)$
       - 4.5.2.2. $AddModel(c, \boldsymbol{C})$
       - 4.5.2.3. $c_t \leftarrow c$
5. *end loop*

Şekil 4.9. pcStream algoritmasının sözde kodu (Mirsky ve diğerleri, 2015)

pcStream algoritmasının başarısını ölçmek için CluStream (Aggarwal ve diğerleri, 2003), D-Stream (Tu ve Chen, 2009), DenStream (Cao ve diğerleri) ve ClusTree (Kranen ve diğerleri, 2011) algoritmalarıyla KDD CUP'99 veri seti üzerinde ARI (Adjusted Rand Index) metriği açısından karşılaştırılmıştır. Şekil 4.10'da görülen sonuçlar elde edilmiştir. Şekilde de görüldüğü gibi diğer algoritmalara oranla yüksek kümeleme başarısı elde edilmektedir.




Şekil 4.10. pcStream algoritmasının başarısı (Mirsky ve diğerleri, 2015)

CEDAS algoritması

CEDAS (Richard Hyde ve diğerleri, 2017) algoritması micro-cluster yapısını kullanan graf tabanlı bir akan veri kümeleme algoritmasıdır. CEDAS algoritmasının temel geliştirilme hedefi dairesel de olmayan kümeleri tespit etmektir. Literatürde bulunan çoğu algoritma uzaklık ölçütü olarak Öklid veya Mahalonobis uzaklıklarını kullanmaktadır. Bu nedenle elde edilen küme şekilleri küresel veya eliptik yapıdadır. Ancak bazen kümelerin şekli farklı şekillerde de olabilmektedir. CEDAS bu probleme çözüm olarak önerilmiştir.

CEDAS algoritması iki aşamadan oluşur. Birinci aşamada mikro kümeler oluşturulur, ikinci aşamada ise bu mikro kümelerden kesişenler graf yapısında birleştirilerek makro kümeler elde edilir. Kısaca CEDAS algoritmasının yaptığı işlemi şu şekilde sıralayabiliriz:
0. Parametre seçimi
1. İlklendirme
2. Mikro kümeleri güncelleme
3. Silinmesi gereken kümeleri silme
4. Küme çizge güncelleme

İlklendirme aşamasında gelen veri için mikro küme oluşturulur. Her mikro küme için merkez koordinatları, mikro kümeye atanan veri sayısı, mikro kümenin ait olduğu makro küme referansı, mikronun enerji bilgisi ve kesişen mikro kümelerin listesi tutulur.

Mikro kümeleri güncelleme aşamasında yeni bir veri gelişinde bu verinin ait olduğu bir küme olup olmadığı kontrol edilir. En yakın olduğu kümeyi bulmak için tüm kümlerin merkezine olan Öklid uzaklığına bakılır. Eğer en yakın olduğu mikro kümeye uzaklığı ön tanımlı olan yarıçap değerinden az ise bu veri bu makro kümeye atanır. Mikro küme yeni bir veri aldığı için kümenin enerjisi 1'e çıkarılır. Eğer mesafe yarıçaptan daha fazla ise bu durumda gelen veri yeni bir mikro küme olarak atanır.

Silinmesi gereken kümelerin silinmesi aşamasında zamana bağlı olarak mikro kümelerin enerjisi azaltıldığından enerjisi 0'a düşen varsa bu mikro kümeler silinir. Mikro kümelerin enerjilerinin azaltılması işlemi kullanıcının belirlediği parametre üzerinden hesaplanır.

Küme graflarını güncelleme aşamasında makro kümelerin oluşturduğu graf yapısı güncellenmektedir. Graf yapısını güncelleme işlemi üç şarttan herhangi birinin oluşması durumunda gerçekleşir. Bunlar:

• Yeni bir küme oluşmuş ve atanan veri sayısı belirlenmiş eşik değerden fazla ise,

• Küme merkezi güncellenmişse

• Bir küme silinmiş ve bu nedenle graf yapısından çıkartılması gerekiyorsa,

Mikro kümelerin birleştirildiği graf yapısı güncellenmektedir.

CEDAS algoritmasının yukarıda sözünü ettiğimiz aşamalarına ait algoritmalar Şekil 4.11, 4.12, 4.13 ve 4.14'te görülmektedir.

**Algorithm 1:** CEDAS: initialization.

**Input:** $x, r_0$
Create micro-cluster structure containing:
  $C_1(Centre) = x$
  $C_1(Count) = 1$
  $C_1(Macro) = 1$
  $C_1(Energy) = 1$
  $C_1(Edge) = 1$
Set number of micro-clusters to 1
Set modified micro-cluster number, for use updating the graph structure

Şekil 4.11. CEDAS algoritmasının ilklendirme bölümüne ait sözde kodu (Richard Hyde ve diğerleri, 2017)

**Algorithm 2:** CEDAS: update micro-cluster.

**Input:** $x, C, r_0$
find distance to nearest micro-cluster centre, $d_{\min}$
**if** $d_{\min} < r_0$ **then**
  reset micro-cluster *Energy* to 1
  increment number of samples contained in micro-cluster
  **if** *data is within micro-cluster shell* **then**
    recursively update micro-cluster centre
  **end**
**else**
  Create new micro-cluster
**end**

Şekil 4.12. CEDAS'ta makro kümeleri güncelleme bölümüne ait sözde kodu (Richard Hyde ve diğerleri, 2017)

**Algorithm 3:** CEDAS: kill micro-cluster.

**Input:** *C*, *Decay*
Reduce all *C*(*Energy*) by *Decay*
**if** *Any C*(*Energy*) < 0 **then**
  Remove micro-cluster
  Remove all edges containing the micro-cluster
  Decrement the number of micro-clusters
**end**

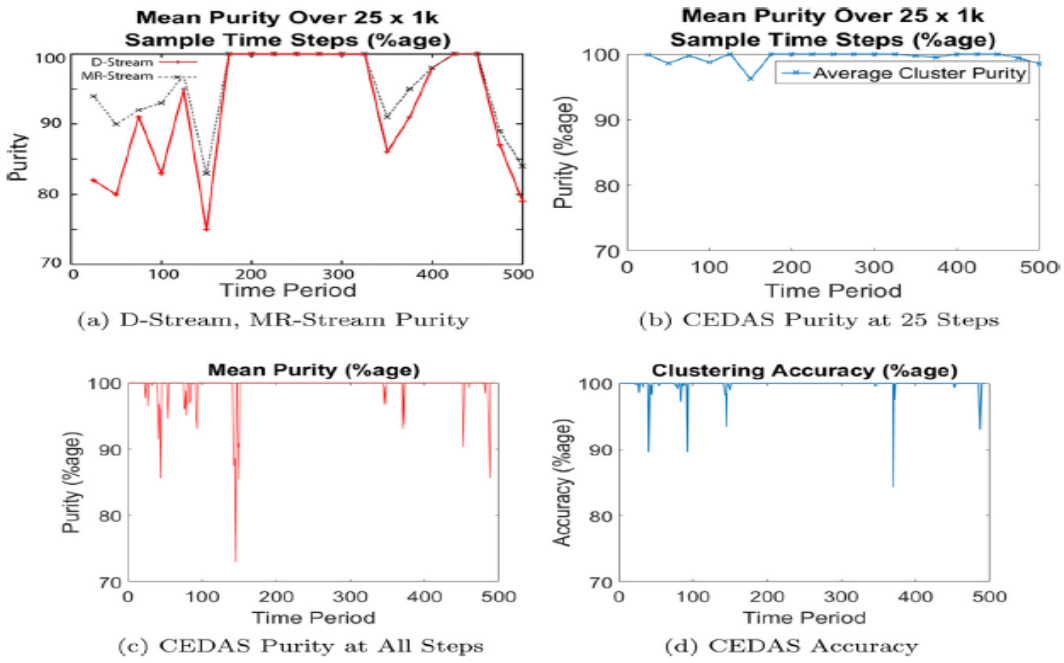
Şekil 4.13. CEDAS algoritmasının küme silme bölümüne ait sözde kodu (Richard Hyde ve diğerleri, 2017)

**Algorithm 4:** CEDAS: update graph.

**if** *A micro-cluster has been modified* **then**
  **if** *the micro-cluster edge list has changed* **then**
    Set a new macro-cluster number throughout the graph
  **end**
**end**
**if** *Any micro-clusters have died* **then**
  Set new macro-numbers for the sub-graphs of its previous edges
**end**

Şekil 4.14. CEDAS algoritmasının graf yapısının güncelleme sözde kodu (Richard Hyde ve diğerleri, 2017)

CEDAS algoritmasının başarısını ölçmek için D-Stream ve MR-Stream algoritmaları ile karşılaştırılmıştır. Şekil 4.15'te de görüldüğü gibi deneysel sonuçlara göre CEDAS algoritması hem Purity hem de Accuracy parametreleri açısından daha iyi sonuç vermiştir




Şekil 4.15. CEDAS algoritmasına ait deneysel sonuçlar (Richard Hyde ve diğerleri, 2017)

DPStream algoritması

DPStream (Xu ve diğerleri, 2017) algoritması tamamen çevrimiçi çalışan ve farklı şekildeki kümeleri tespit etme yeteneğine sahip Leading Tree (LT) tabanlı bir akan veri kümeleme algoritmasıdır. DPStream hiyerarşi ve yoğunluk tabanlı bir akan veri kümeleme algoritmasıdır. Temel anlamda verileri yoğunluğa bağlı olarak birbirleri ile bir ağaç yapısında birleştirir. Aslında DPStream DPClust algoritmasının bir uzantısı şeklindedir.

DPClust algoritması küme merkezlerini kullanıcının interaktif bir şekilde belirlemesine imkan vermektedir. Bu statik bir veri seti için kullanışlı ve başarılı bir yaklaşımdır. Ancak söz konusu olan akan veri ise iteratif bir veri işleme yöntemi ve hızlı akış söz konusudur. Bu nedenle DPClust algoritmasının akan veri için kullanılması durumunda küme merkezlerinin otomatik bir şekilde tespit edilmesine ihtiyaç vardır. Çizelge 4.3'te DPStream algoritmasında kullanılan semboller verilmiştir.

Çizelge 4.3. DPClust ve DPStream algoritmalarında kullanılan sembollerin anlamları (Xu ve diğerleri, 2017)

| Sembol | Anlamı |
|---|---|
| X | Veri seti |
| $d_{i,j}$ | i ve j arasındaki mesafe |
| I | Verilerin indisleri |
| C | Kümelerin merkezleri |
| ρ | Verilerin yerel yoğunluğu |
| δ | Yerel yoğunluğu en yüksek komşu notalara olan uzaklık |
| Q | ρ'nin azalan sırada sıralanmış hali |
| $N_n$ | En yakın komşuların indisleri |
| γ | ρ ve δ sonucu oluşan değer |
| W | FNLT'de oluşan ağırlık |
| Cl | FNLT'nin küme etiketi |

DPStream çalışma mantığını anlamak için DPClust algoritmasına yakından bakmak gerekir. DPClust algoritması aşağıdaki işlem basamaklarını takip etmektedir (Xu ve diğerleri, 2017):

1. $\rho$ değerlerini aşağıdaki eşitlik ile hesapla;

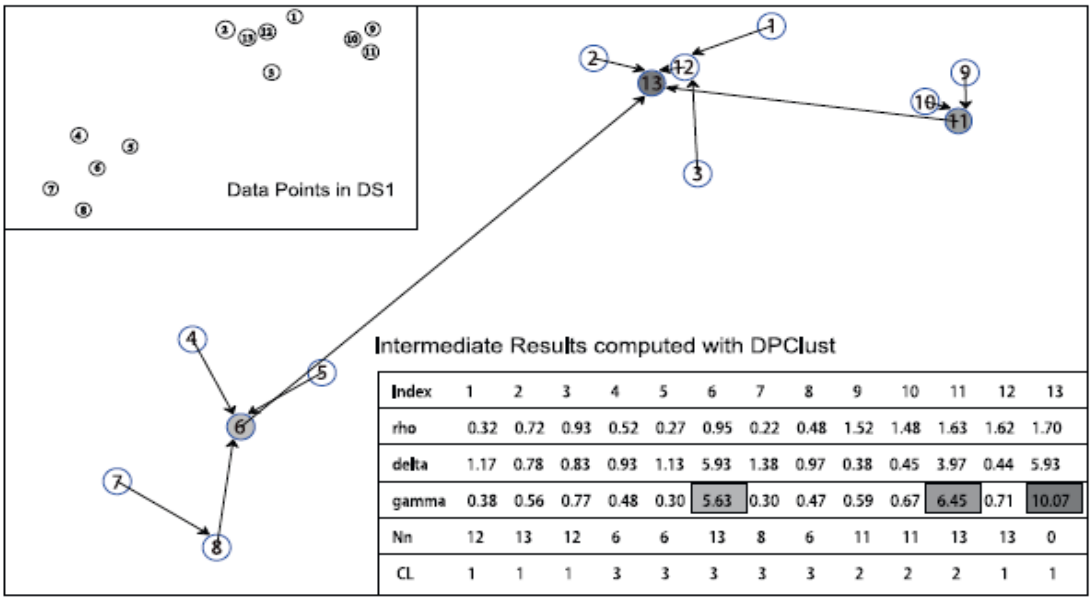
$$\rho_i = \sum_{j \in I/\{i\}} e^{-(\frac{d_{i,j}}{d_c})^2} \tag{4.22}$$

2. $\rho$ değerlerini azalan sırada sırala;
3. $\delta$ aşağıdaki eşitliğe göre hesapla

$$\delta_{q_i} = \begin{cases} \min\{d_{q_i}, q_j\}, x \geq 2 & j \lhd i \\ \max\{d_{q_i}, q_j\}, i = 1 & j \geq 2 \end{cases} \tag{4.23}$$

ve $\rho$ değeri yüksek olan en yakın komşuların indekslerini *Nn*'ye ekle;

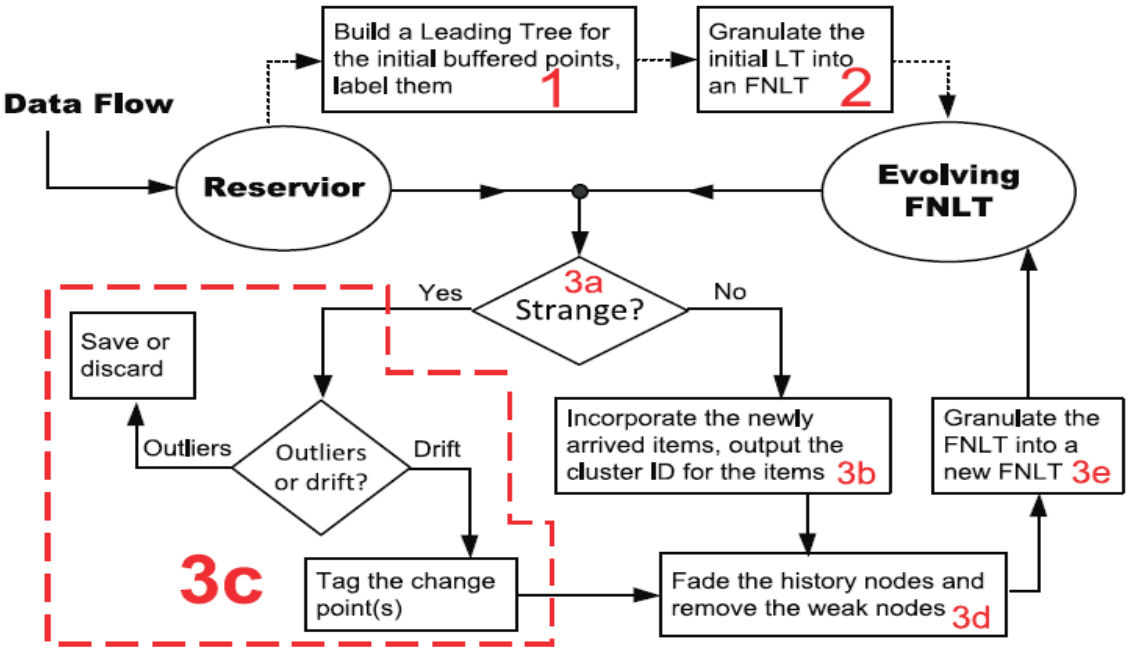
4. İnteraktif olarak $\rho$ ve $\delta$ değerlerini seç;
5. $\rho$ değeri yüksek olan en yakın komşuları aynı kümeye ata.




| Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| rho | 0.32 | 0.72 | 0.93 | 0.52 | 0.27 | 0.95 | 0.22 | 0.48 | 1.52 | 1.48 | 1.63 | 1.62 | 1.70 |
| delta | 1.17 | 0.78 | 0.83 | 0.93 | 1.13 | 5.93 | 1.38 | 0.97 | 0.38 | 0.45 | 3.97 | 0.44 | 5.93 |
| gamma | 0.38 | 0.56 | 0.77 | 0.48 | 0.30 | 5.63 | 0.30 | 0.47 | 0.59 | 0.67 | 6.45 | 0.71 | 10.07 |
| Nn | 12 | 13 | 12 | 6 | 6 | 13 | 8 | 6 | 11 | 11 | 13 | 13 | 0 |
| CL | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 |

Şekil 4.16. DPClust algoritmasının çalışma mantığı (Xu ve diğerleri, 2017)

DPStream algoritması Şekil 4.17'te görüldüğü gibi çalışmaktadır.




Şekil 4.17. DPStream algoritmasının akış diyagramı (Xu ve diğerleri, 2017)

**Algorithm 1:** DP-Stream algorithm.

**Input**: Data stream $X$
**Output**: $\boldsymbol{Cl}$, outliers, and change points
Construct the initial LT;
Granulate the LT (Algorithm 2);
**while** *Data* $X_{new}$ *streaming in* **do**
  **if** $X_{new}$ *is not strange* **then**
    Merge $X_{new}$ into FNLT, output $\boldsymbol{Cl}_{new}$ (Algorithm 3);
  **end**
  Buffer $BufferSize$ new data points ;
  **for** *each data point* $X_{new}$ *in the Buffer* **do**
    **if** $X_{new}$ *is noise* **then**
      Discard $X_{new}$ or store it on hard disk;
    **end**
    **else**
      Merge $X_{new}$ into FNLT, output $\boldsymbol{Cl}_{new}$ ;
    **end**
  **end**
  Detect drift ;
  Fade out, remove weak nodes ;
  Granulate and update FNLT ;
**end**

Şekil 4.18. DPStream algoritmasının sözde kodu (Xu ve diğerleri, 2017)

DPStream algoritması ExclaStar, MrData, ChameleonDS3, RBF10a, RBFDrift, CoverType ve KDD'99 veri setleri üzerinde test edilmiştir. DPStream DenStream, CluStream ve STRAP algoritmaları ile karşılaştırılmıştır. DPStream algoritmasının sözde kodları Şekil

4.18, 4.19 ve 4.20'de görüldüğü gibidir. Şekil 4.21'de de görüldüğü gibi DPStream algoritması daha başarılı sonuçlar vermiştir.

**Algorithm 2:** granulating the FNLT.

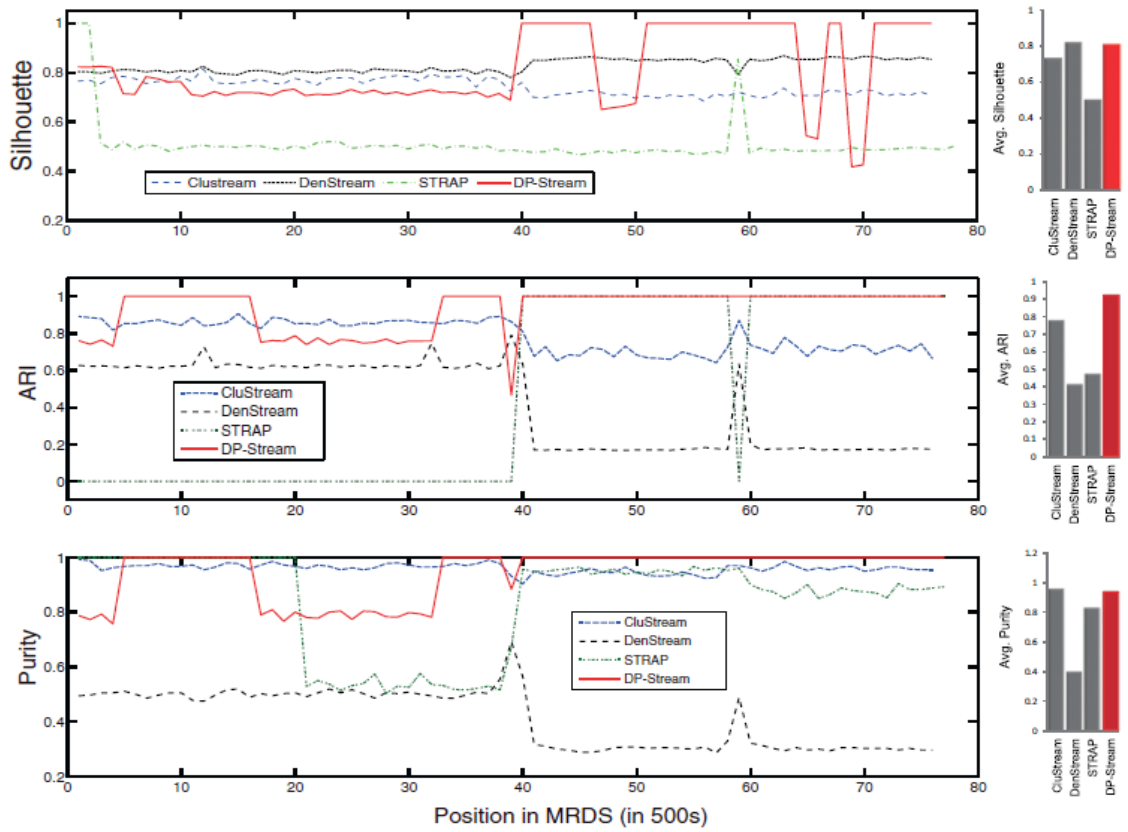
**Input**: the FNLT $T$, Sketch Index $(SI)$
**Output**: a newly granulated FNLT $T'$
**Procedure**:
$N_{merge} \leftarrow \lceil N(1-SI) \rceil$ ;
$\boldsymbol{newNn} \leftarrow T.\boldsymbol{Nn}$;
$[\delta^s, \boldsymbol{SInd}_\delta] \leftarrow SortAscend(T.\delta)$;
$RemainInds \leftarrow \emptyset$;
$i \leftarrow 0$;
**for** *each* $j$ *in the first* $N_{merge}$ *elements of* $\boldsymbol{SInd}_\delta$ **do**
  $i \leftarrow i+1$;
  $RemainInds_i \leftarrow newNn_j$;
  **if** $newNn_m == j$ *for any* $m \in [1, N]$ **then**
    $newNn_m \leftarrow newNn_j$;
  **end**
  **if** $RemainInds_m == j$ *for any* $m \in [1, i-1]$ **then**
    $RemainInds_m \leftarrow newNn_j$;
  **end**
**end**
Append $N - N_{merge}$ elements in $\boldsymbol{SInd}_\delta$ to $RemainInds$;
$U \leftarrow$ Unique$(RemainInds)$;
$T'.X \leftarrow T.X_U$;
$T'.\boldsymbol{w}_i \leftarrow \sum T.w_{M_i}$, where $M_i$ is the nodes set in $T$ merged into $T'.X_i$;
$T'.\rho_i \leftarrow \sum T.\rho_{M_i}$, ;
Extract new $T'.D$ from $T.D$ via $T'.X$;
Update $T'.\delta$ and $T'.\boldsymbol{Nn}$ with $T'.D$ and $T'.\rho$ ;

Şekil 4.19. DPStream algoritmasının granulate the FNLT fonksiyonunun sözde kodu (Xu ve diğerleri, 2017)

**Algorithm 3:** Incrementally updating the FNLT.

**Input:** the FNLT $T$, a new item $x$
**Output:** an updated FNLT
**Procedure:**
**Step1:** //Update $T.\rho$ and compute $\rho_x$ for $x$
**for** *each point* $x_i$ *in* $T.X$ **do**
  $d_{i,new} \leftarrow$ computeDistance$(x_i, x)$;
  IncreValue $\leftarrow \exp(-(d_{i,new}/dc)^2)$;
  $T.\rho_i \leftarrow T.\rho_i$ + IncreValue;
  $\rho_x \leftarrow \rho_x$ + IncreValue* $T.W_i$;
**end**
Append $\rho_x$ to $T.\rho$;
**Step2:** // Expand $T.\boldsymbol{D}_{N\times N}$ to $T.\boldsymbol{D}_{(N+1)\times(N+1)}$
The bottom row of $T.\boldsymbol{D}_{(N+1)\times(N+1)} \leftarrow [\boldsymbol{d_{new}}, 0]$;
The last column of $T.\boldsymbol{D}_{(N+1)\times(N+1)} \leftarrow [\boldsymbol{d_{new}}, 0]^T$;
**Step3:**//Compute $\delta_x$ and $Nn_x$ for $x$
**if** $\rho_x$ *is not the biggest* **then**
  $Nn_x \leftarrow \underset{i}{\text{argmin}} \left\{T.D_{i,N+1} | i \in [1, N], T.\rho_i > \rho_x\right\}$;
  $\delta_x \leftarrow \min\{\text{T.D}_{\text{i,N+1}} | \text{i} \in [1, \text{N}], \text{T.}\rho_\text{i} > \rho_\text{x}\}$;
**end**
**else**
  $Nn_x \leftarrow 0$;
  $\delta_x \leftarrow \max\{\text{T.D}_{\text{i,N+1}} | 1 \leq \text{i} \leq \text{N}\}$;
**end**
**Step4:** //Output the clustering result for $x$
**if** $\rho_x$ *is not the biggest* **then**
  $Cl_x \leftarrow Cl_{Nn_x}$;
**end**
**else**
  $Cl_x \leftarrow Cl_s$, where $s = \arg\min_i\{D_{i,N+1}\}$;
**end**
**Step5:** //update $T.\delta$ and $T.\boldsymbol{Nn}$
**if** $x$ *does not change the order of* $T.\rho$ **then**
  $\boldsymbol{SI} \leftarrow \{i | \rho_i < \rho_x, 1 \leq i \leq N\}$ ;
  **for** *each si in* $\boldsymbol{SI}$ **do**
    **if** $T.D_{si,N+1} < T.\delta_{si}$ **then**
      $T.\delta_{si} \leftarrow T.D_{si,N+1}$;
      $T.Nn_{si} \leftarrow N + 1$;
    **end**
  **end**
**end**
**else**
  Recompute $T.\delta$ and $T.\boldsymbol{Nn}$ according to the definitions;
**end**
Append $\delta_x$, $Nn_x$, $x$, 1 to $T.\delta$, $T.\boldsymbol{Nn}$, $X$, $\boldsymbol{W}$, respectively;

Şekil 4.20. DPStream algoritmasının FNLT'yi güncelleme fonksiyonuna ait sözde kod (Xu ve diğerleri, 2017)




Şekil 4.21. DPStream algoritmasının MrData veri setinde başarısı (Xu ve diğerleri, 2017)

### 4.3.3. Kümeleme başarısı

Algoritmaların başarısını karşılaştırmak için kümeleme kalitesini ölçme yöntemlerinden Purity, Accuracy, Precision, Recall, F-Score ve Silhouette İndeks yöntemleri kullanılmıştır. Her algoritma için en iyi sonuç üreten parametreler seçilmiştir. Eğer kullanılan veri seti algoritmaların herhangi birisinde kullanılmış ise o çalışmada kullanılan parametreler seçilmiştir. Aksi durumda ise en iyi sonucu üreten parametreler kullanılmıştır. Örneğin KDD veri seti bütün algoritmalarda kullanılmıştır. Bu nedenle makalelerde belirtilen parametreler kullanılmıştır.

KD-ARFS Stream algoritması için en iyi sonucun hangi parametre olduğunu tespit etmek için Grid Search ve Random Search kullanılmıştır. Veri sayısı az olan veri setlerinde Grid Search kullanılırken, KDD gibi veri sayısı çok olan veri setlerinde Random Search kullanılmıştır. Amaç fonksiyonu olarak da Accuracy Testi kullanılmıştır. Çizelge 4.4, 4.5, 4.6, 4.7 ve 4.8'de algoritmaların test edilmesinde kullanılan parametreler görülmektedir.

Çizelge 4.4. KD-ARFS Stream algoritmasının test parametreleri

| | Veri Setleri | | | | | | |
|---|---|---|---|---|---|---|---|
| Parametreler | KDD | Fisher Iris | Breast Cancer | MrData | ExclaStar | IdealData | Occupancy |
| N | 85 | 6 | 3 | 25 | 14 | 5 | 40 |
| t (sn) | 0,5 | 0,1 | 3 | 0,1 | 0,1 | 0,1 | 0,58 |
| TN | 130 | 65 | 200 | 200 | 40 | 25 | 290 |
| r | 1,2 | 0,85 | 7,5 | 15 | 5 | 3,5 | 380 |
| r_threshold | 2,45 | 0,75 | 2,5 | 2,5 | 2,75 | 2,25 | 80 |
| r_max | 6,2 | 2,2 | 10,35 | 26 | 9,75 | 12 | 455 |
| coeffThreshold | 90 | 97,5 | 95 | 100 | 100 | 100 | 100 |

Çizelge 4.5. SE-Stream algoritmasının test parametreleri

| | Veri Setleri | | | | | | |
|---|---|---|---|---|---|---|---|
| Parametreler | KDD | Fisher Iris | Breast Cancer | MrData | ExclaStar | IdealData | Occupancy |
| α | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| max_cluster | 10 | 3 | 4 | 4 | 3 | 5 | 4 |
| stream speed | 100 | 100 | 1000 | 1000 | 100 | 100 | 1000 |
| decay_rate | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,9 | 0,1 |
| radius_factor | 3 | 1 | 5 | 7 | 2 | 8 | 5 |
| remove_threshold | 0,1 | 0,1 | 0,01 | 0,01 | 0,1 | 0,9 | 0,01 |
| merge_threshold | 1,25 | 0,5 | 2 | 1,5 | 1,5 | 3 | 2 |
| active_threshold | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| number of projected dimension (l) | 10 | no projection | 3 | no projection | no projection | no projection | no projection |

Çizelge 4.6. pcStream algoritmasının test parametreleri

| | Veri Setleri | | | | | |
|---|---|---|---|---|---|---|
| Parameters | KDD | Breast Cancer | MrData | ExclaStar | IdealData | Occupancy |
| driftThreshold | 13 | 20 | 177 | 6 | 20 | 2 |
| maxDriftSize | 15 | 50 | 450 | 15 | 15 | 10 |
| percentVarience | 0,26 | 0,69 | 0,80 | 0,97 | 0,72 | 0,25 |
| modelMemSize | 400 | 200 | 3040 | 95 | 70 | 105 |

Çizelge 4.7. CEDAS algoritmasının test parametreleri

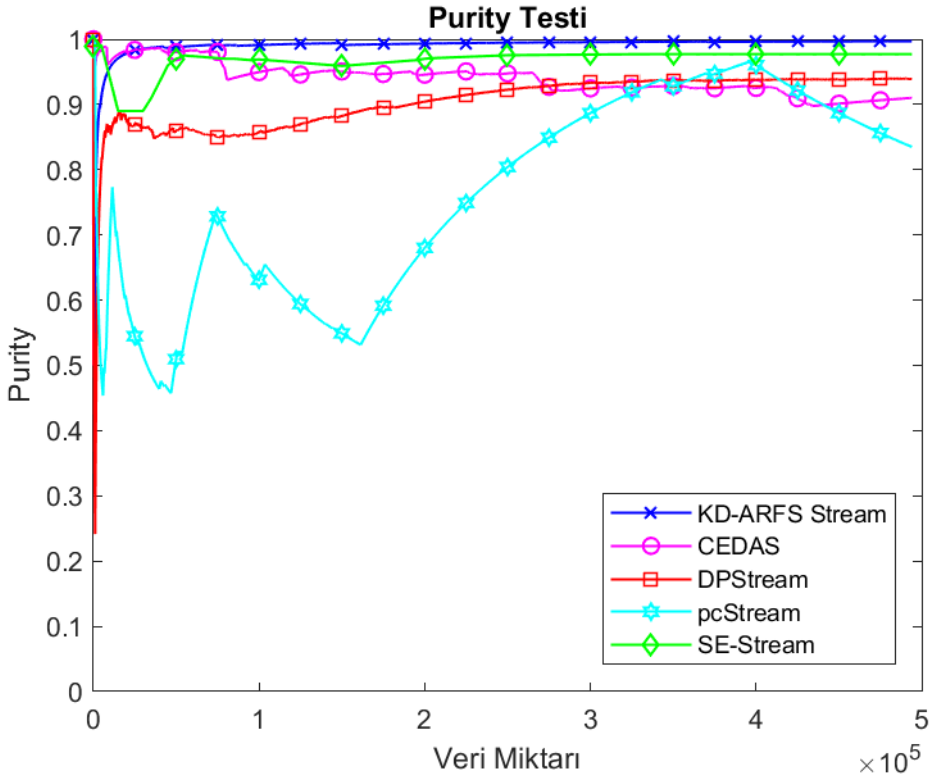
| | Veri Setleri | | | | | | |
|---|---|---|---|---|---|---|---|
| Parametreler | KDD | Fisher Iris | Breast Cancer | MrData | ExclaStar | IdealData | Occupancy |
| radius | 0,05 | 0,05 | 0,4 | 0,5 | 0,05 | 0,2 | 0,05 |
| Decay | 100 | 200 | 200 | 200 | 200 | 200 | 100 |
| MinThreshold | 4 | 3 | 4 | 5 | 3 | 3 | 4 |

Çizelge 4.8. DPStream algoritmasının test parametreleri

| | Veri Setleri | | | | | | |
|---|---|---|---|---|---|---|---|
| Parametreler | KDD | Fisher Iris | Breast Cancer | MrData | ExclaStar | IdealData | Occupancy |
| percent | 2 | 5 | 2 | 2 | 5 | 5 | 5 |
| LocalR | 1,6 | 1,2 | 10 | 4,8 | 4,8 | 5 | 7,5 |
| GlobalR | 0,05 | 0,6 | 0,2 | 0,3 | 0,1 | 0,5 | 0,5 |
| bufferSize | 50 | 25 | 30 | 50 | 20 | 40 | 50 |
| SketchIndex | 0,90 | 0,70 | 0,75 | 0,90 | 0,75 | 0,8 | 0,75 |
| startBufferSize | 1000 | 25 | 100 | 1000 | 180 | 40 | 100 |
| HalfLife | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| RemovePop | 0,5 | 0,5 | 0,5 | 0,5 | 0,5 | 0,5 | 0,5 |

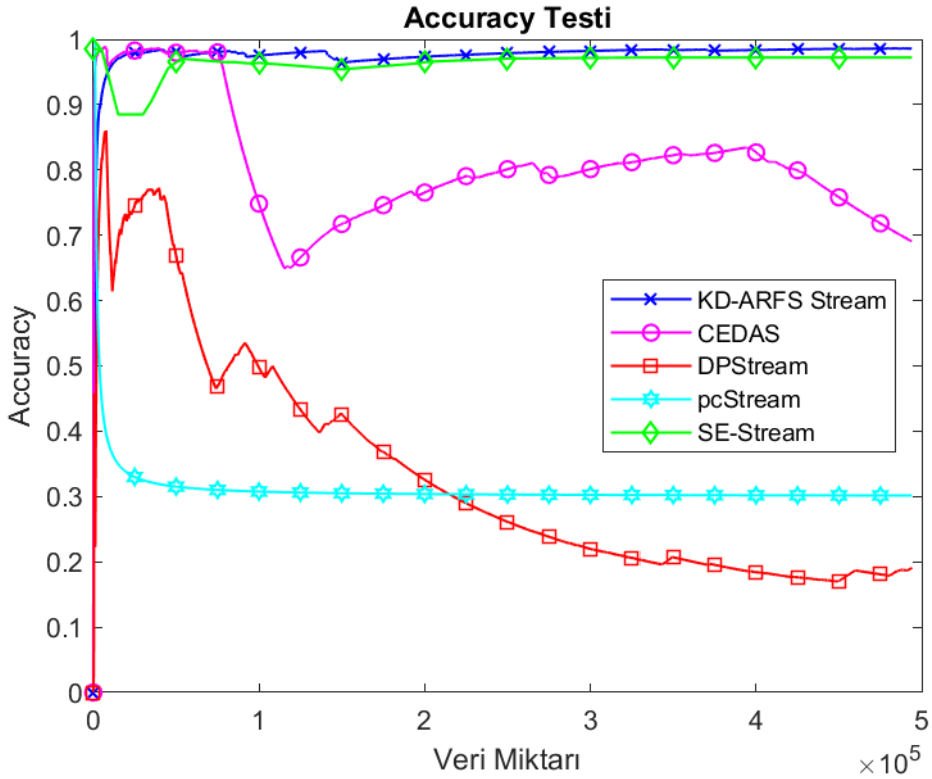
KDD veri seti ile algoritmaların başarılarının karşılaştırması

Beş algoritmanın KDD veri seti üzerinde test edilmesinde Çizelge 4.4, 4.5, 4.6, 4.7 ve 4.8'de belirtilen parametreler kullanılmıştır. KD-ARFS Stream, SE-Stream, CEDAS ve DPStream algoritmaları için makalelerde belirtilen parametreler kullanılmıştır. pcStream algoritmasında ise KDD veri seti için en iyi sonuç üreten parametreler grid search ile tespit edilmiştir.




Şekil 4.22. KDD veri setinde algoritmaların Purity değerleri üzerinden karşılaştırması

Şekil 4.22'de de görüldüğü gibi pcStream algoritması dışındaki diğer algoritmalar yüksek başarı elde etmektedir. pcStream algoritması konsept değişimlerini tespit etmeye yönelik önerilmiş bir algoritmadır. Her yeni konsept tespitinde yeni bir küme tanımlamaktadır. Dolayısıyla başarısının düşük olması beklenen bir durumdur. DPStream algoritması da pcStream algoritması gibi verinin değiştiği noktaları tespit eder. Verinin ilk bölümlerinde küme atamalarını doğru yapmadığından başarısı nispeten düşüktür. CEDAS ve SE-Stream algoritmaları evrimsel değişimi desteklediğinden başarısı diğer algoritmalardan daha yüksektir. Yine de en yüksek Purity sonucunu üreten algoritma önerdiğimiz algoritmadır.

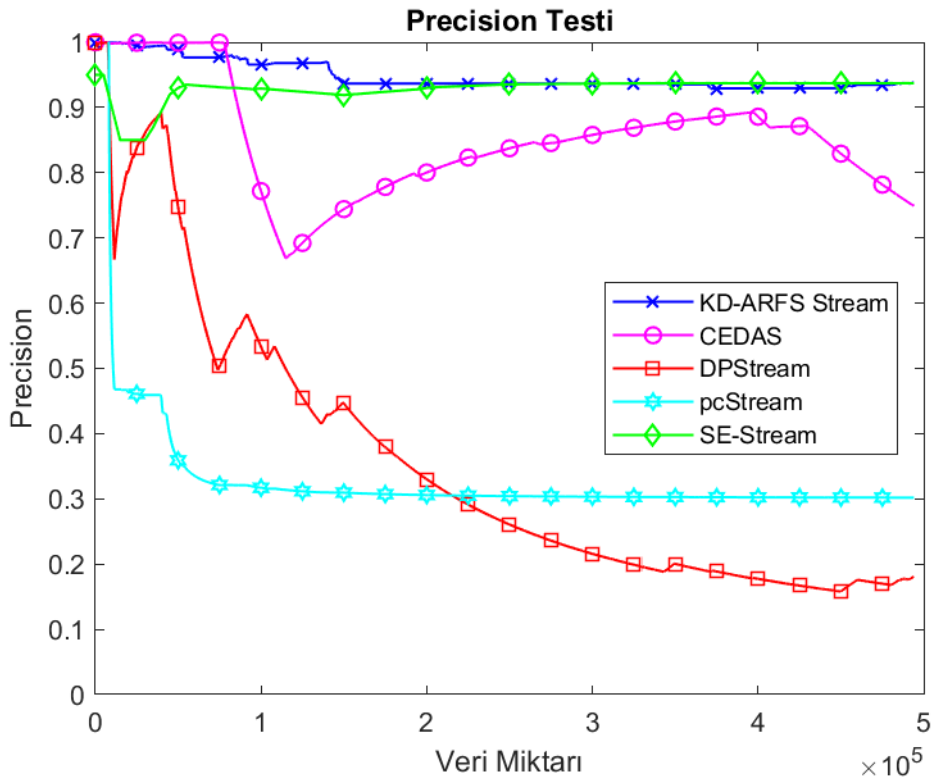



Şekil 4.23. KDD veri setinde algoritmaların Accuracy üzerinden karşılaştırması

Şekil 4.23'te de görüldüğü gibi Accuracy testinde önerdiğimiz yöntem en iyi sonucu üretmektedir. KDD veri seti toplam 23 sınıftan oluşmaktadır. Bu kümeler bazen yok olup daha sonra tekrar ortaya çıkmaktadır. Bu durumu en iyi önerdiğimiz yaklaşım desteklemektedir. SE-Stream algoritması da tam evrimsel bir algoritma olduğu için yüksek başarı elde etmesine rağmen bu başarı önerdiğimiz yaklaşımdan daha düşüktür. Diğer yöntemler veri setinde bulunan kümeleri zaman içinde sildiğinden başarı oranları düşüktür. Verilerin atandığı kümelerin gerçek küme etiketinden farklı olması Accuracy oranını doğrudan etkilemektedir. Özellikle pcStream konsept değişminde yeni bir küme tanımladığından Accuracy değeri çok düşük çıkmaktadır. Benzer şekilde, CEDAS algoritmasının 91000-140000 arasında veriyi atadığı küme etiketi gerçek küme etiketinden farklı olduğu için başarısında düşüş yaşamaktadır. Benzer durumlar diğer algoritmalar için de geçerlidir. KD-ARFS Stream'e baktığımız zaman 140000-145000 aralığında sınırlı bir düşüş söz konusudur. Ancak bu düşüş başarıyı çok fazla etkilememektedir.

Şekil 4.24'te görüldüğü gibi en yüksek Precision değerini elde eden yaklaşım önerdiğimiz KD-ARFS Stream algoritmasıdır. Çünkü önerdiğimiz yaklaşım, evrimsel bir yapıya sahip olduğundan gerçek küme etiketlerini daha kararlı bir şekilde elde etmektedir. Yanlış küme

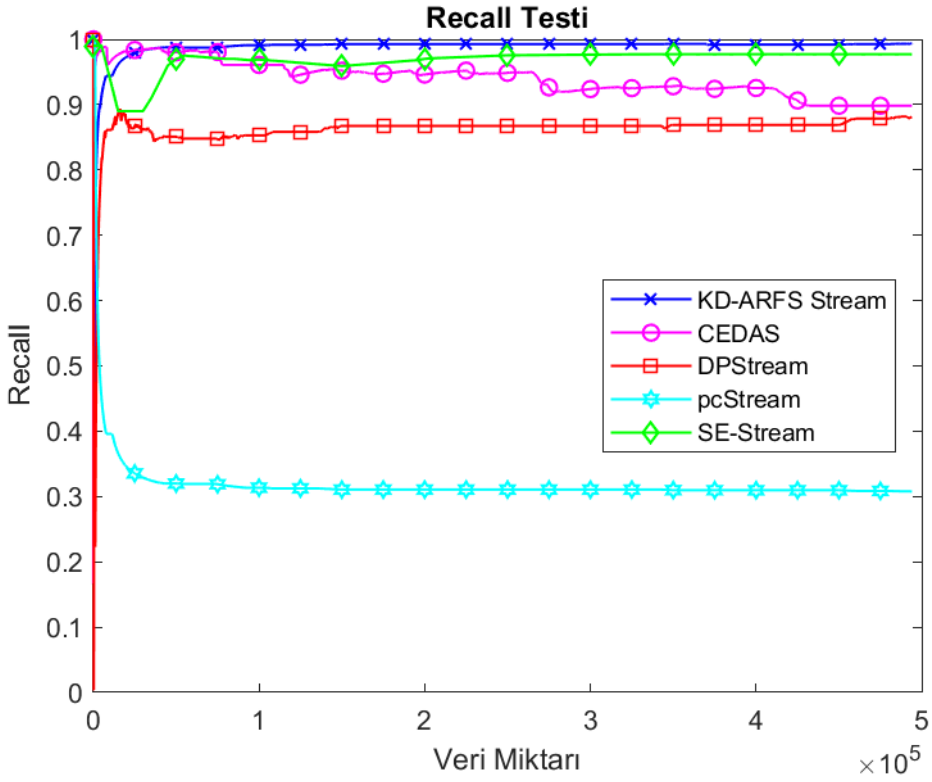
atamaları çok sınırlıdır. Bu da Precision değerinin yüksek çıkmasını sağlamaktadır. Accuracy grafiğine benzer şekilde 140000-145000 aralığında sınırlı bir düşüş söz konuşur. Bunun da nedeni bazı verileri yanlış kümelere atamasıdır. SE-Stream algoritması da önemli bir başarı elde etmesine rağmen veri setinin özellikle ilk 50000'lik döneminde başarısında düşüş yaşamaktadır. KDD veri setinin bu bölümünde çok sayıda küme bulunmakta ve bu kümelerin bazıları da birbirine çok yakındır. Bunu tespit etmek zor olmasına rağmen önerdiğimiz yaklaşım önemli bir başarı elde etmektedir. Çünkü KD-ARFS Stream'de FP sayısı oldukça azdır. Diğer algoritmalarda oluşan düşüşler bu bölümlerde gerçekte söz konusu etikete ait olduğu düşünülenlerden önemli bir kısmının aslında bu etikete ait olmadığını göstermektedir. Özellikle pcStream algoritmasının küme tanımlama işlemini yanlış yapmasından dolayı başarı oranı çok düşüktür.




Şekil 4.24. KDD veri setinde algoritmaların Precision üzerinden karşılaştırması

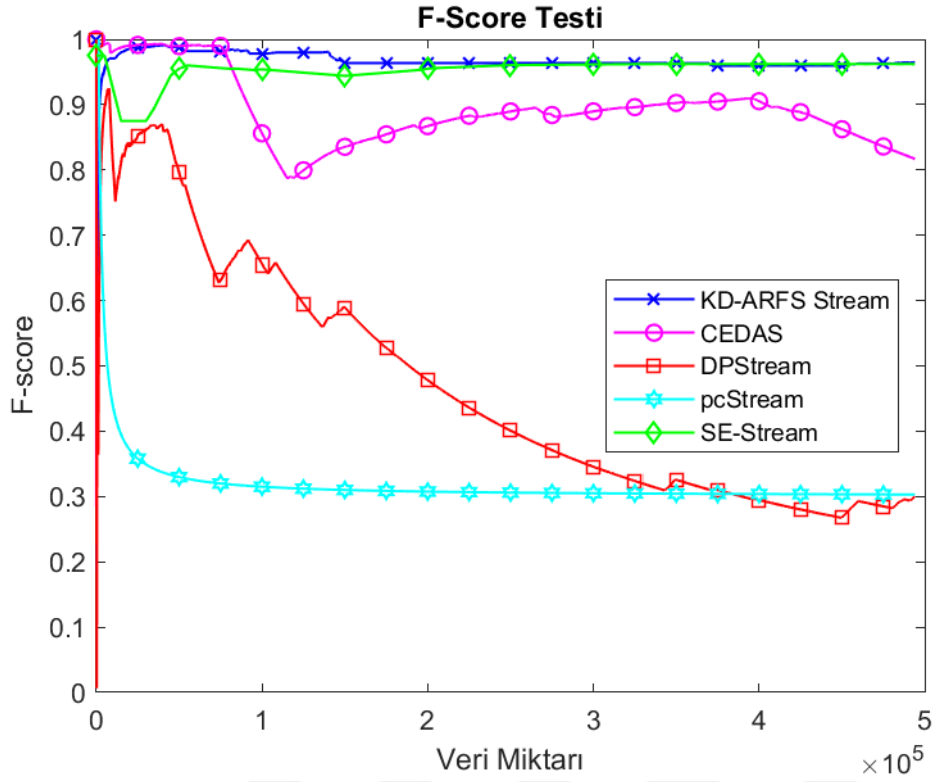
Recall bir kümeye ait olduğu düşünülen verilerden gerçekte bu kümeye ait olanlara oranıdır. Şekil 4.25'te de görüldüğü gibi KD-ARFS Stream algoritması dışındaki algoritmalar, düşüşün olduğu bölümlerde aslında o kümeye ait olan verilerin önemli bir kısmını tespit edememektedir. DPStream ve CEDAS algoritmalarında Accuracy ve Precision değerlerinin düşük çıkmasına rağmen Recall değerlerinin daha yüksek çıkmasının nedeni kümeleri

tanımladıktan sonra gerçekten o kümeye ait olan verileri o kümeye atamasına rağmen söz konusu kümeye ait olmayan verileri atamış olmasıdır. Yani FN sayısı azdır. Önerdiğimiz yaklaşımın Recall değerlerinin Accuracy ve Precision grafiklerine paralel bir şekilde yüksek çıkması yaptığı kümeleme işleminin ne kadar tutarlı ve başarılı olduğunu göstermektedir.

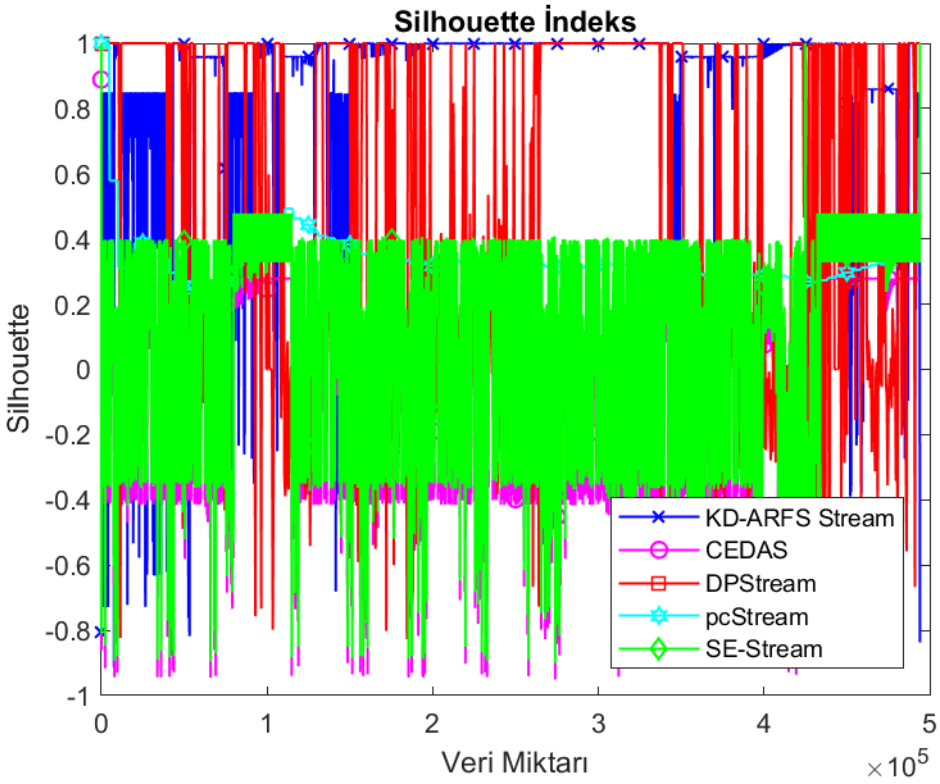



Şekil 4.25. KDD veri setinde algoritmaların Recall üzerinden karşılaştırması

F-Score testi Accuracy testine benzerlik göstermektedir. KD-ARFS Stream 140000-145000 aralığında bulunan kümeyi tespit edemediğinden söz konusu aralıkta bir düşüş söz konusudur. Diğer yandan SE-Stream algoritması dışındaki diğer algoritmalar veri setinin önemli bir bölümünde söz konusu kümeleri tespit edemediğinden belirgin düşüşler söz konusudur. Şekil 4.26'da da görüldüğü gibi KD-ARFS Stream algoritması en yüksek F-Score sonucunu üretmektedir.




Şekil 4.26. KDD veri setinde algoritmaların F-Score üzerinden karşılaştırması

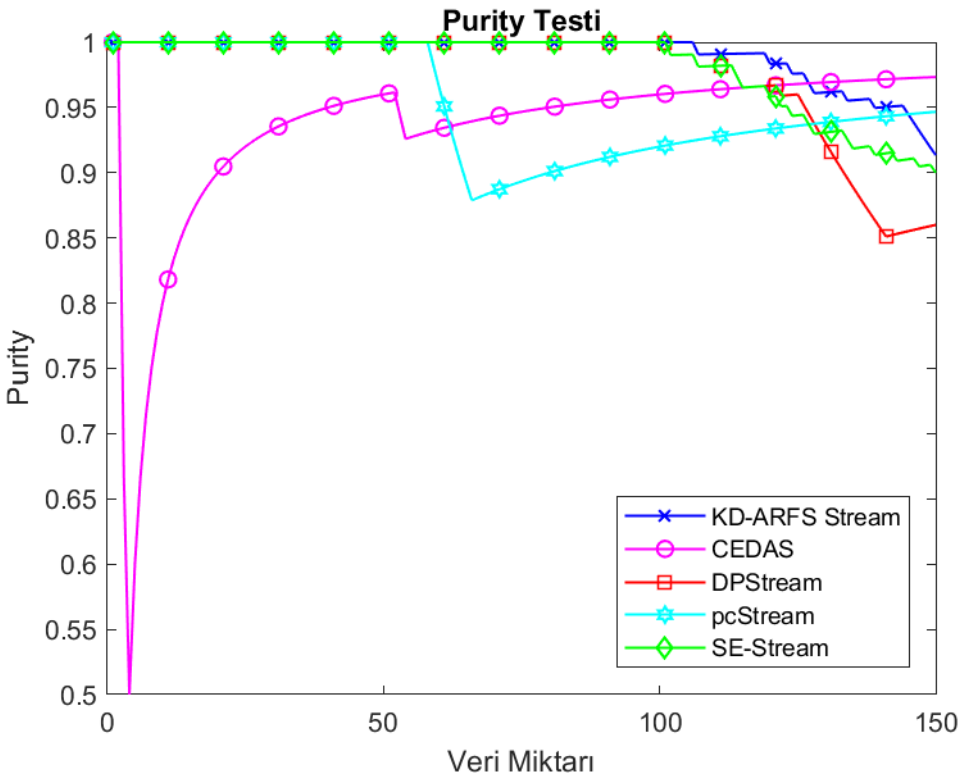



Şekil 4.27. KDD veri setinde algoritmaların Silhouette indeks üzerinden karşılaştırması

Silhouette indeks bir kümenin içerisindeki verilerin birbirine ne kadar yakın olduğunu ve aynı zamanda da diğer kümelerden ne kadar uzak olduğunu göz önünde bulunduran bir test yöntemidir. Şekil 4.27'de de görüldüğü gibi KD-ARFS Stream çok iyi sonuç üretmektedir. Çünkü KD-ARFS Stream'in oluşturduğu kümelerde bulunan veriler birbirine diğer algoritmaların oluşturduğu kümelerdeki verilerden daha çok benzemektedir.

Fisher Iris veri seti ile algoritmaların başarılarının karşılaştırması

Beş algoritmanın Fisher Iris veri seti üzerinde test edilmesinde Çizelge 4.4, 4.5, 4.6, 4.7 ve 4.8'de belirtilen parametreler kullanılmıştır. Her beş algoritma için de en iyi sonuç üreten parametreler tespit edilerek test edilmiştir.

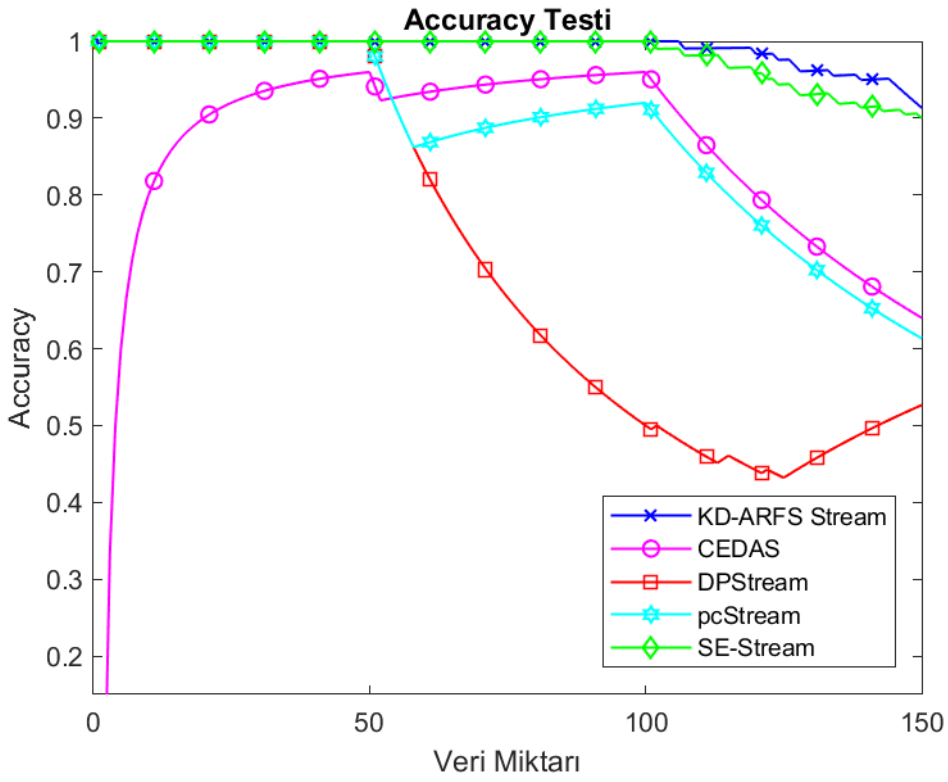



Şekil 4.28. Fisher Iris veri setinde algoritmaların Purity üzerinden karşılaştırması

Fisher Iris veri seti 50'şer veri içeren 3 kümeden oluşmaktadır. Purity testi tespit edilen kümedeki verilerin ait olduğu kümeyi karşılaştırmaktadır. Yani küme etiketlerinin birebir aynı olması gerekmez. Bu nedenle Purity testine göre DPStream ilk 2 kümede %100 başarı verirken Accuracy testinde sadece ilk kümede %100 başarı göstermektedir. Şekil 4.28'de de görüldüğü gibi önerdiğimiz yöntem en yüksek başarıya ulaşmaktadır. İlk iki küme için

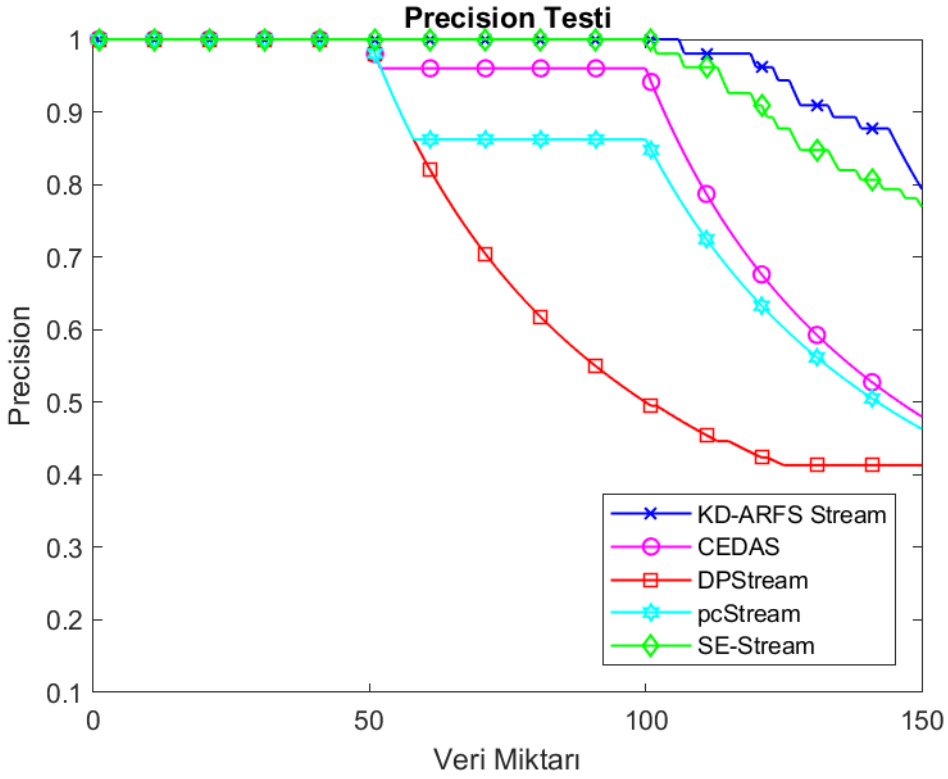
verileri çok doğru bir şekilde tespit etmektedir. Sadece son kümede bazı verileri yanlış kümelere atamaktadır. Bunun da nedeni üçüncü kümenin ikinci kümeye oldukça yakın olmasıdır.

Accuracy testinde Purity testine ek olarak küme etiketleri de göz önünde bulundurulur. Bu nedenle kümelerin tespit ettiği kümelere verdiği etiketlerin gerçek etiketlerle uyuşması gerekir. KD-ARFS Stream açısından bakıldığında ilk 2 küme %100 başarı ile tespit edilirken, üçüncü küme ikinci kümeye çok yakın olduğundan bu bölümde bir miktar düşüş gözlenmektedir. Ancak, Şekil 4.29'da da görüldüğü önerdiğimiz yaklaşım en yüksek başarıya ulaşmaktadır.

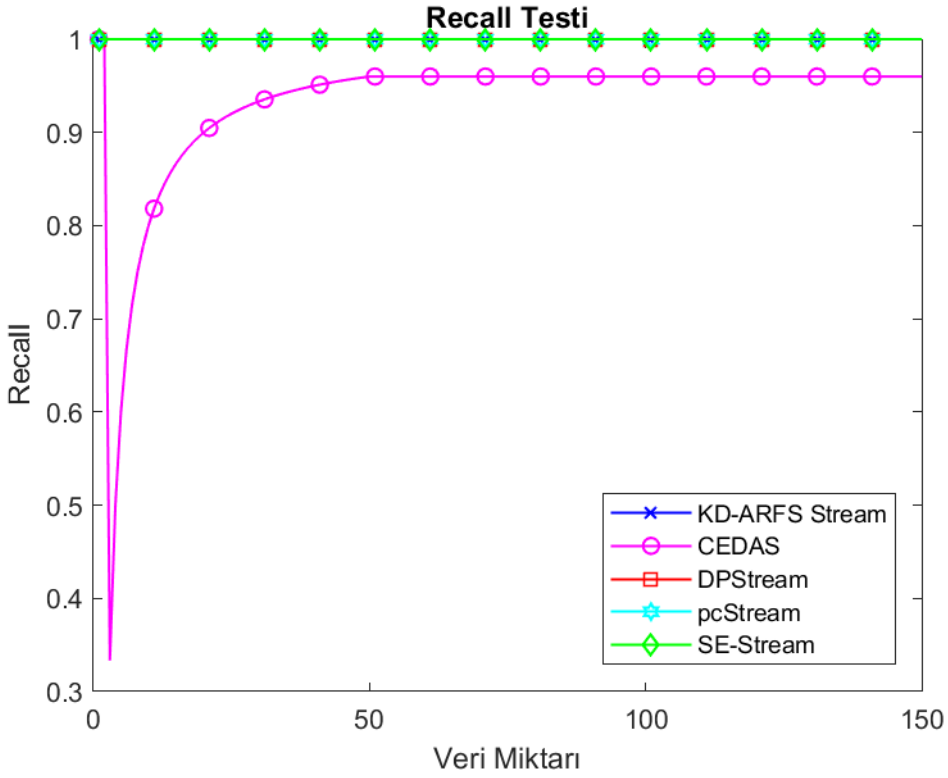



Şekil 4.29. Fisher Iris veri setinde algoritmaların Accuracy üzerinden karşılaştırması

Precision testine göre ilk kümenin tespiti ve etiketlenmesinde tüm algoritmalar %100'lük başarı elde etmektedir. Ancak sonrasında tüm algoritmaların başarısı düşmektedir. Özellikle DPStream ve CEDAS algoritmalarının başarısında belirgin bir düşüş söz konusudur. Çünkü bu algoritmalarda FP sayısı oldukça fazladır. Bu da 2. ve 3. kümelere atanan verilerin önemli bir kısmının aslında o kümelere ait olmadığını göstermektedir. Şekil 4.30'da da görüldüğü gibi KD-ARFS Stream en yüksek Precision sonucu üretmektedir.



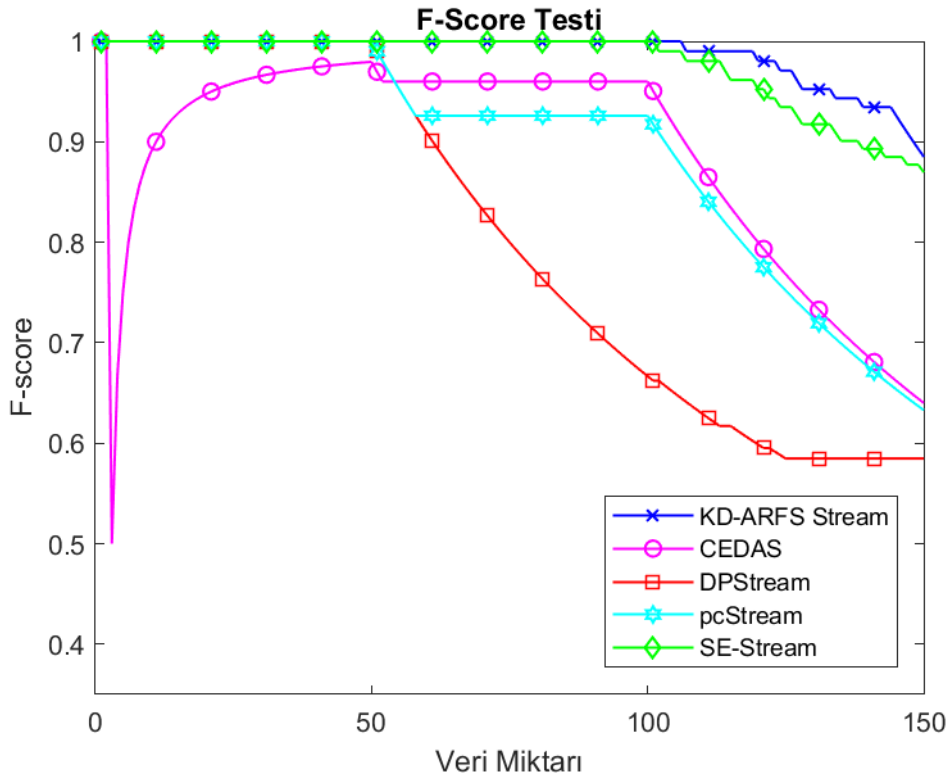

Şekil 4.30. Fisher Iris veri setinde algoritmaların Precision üzerinden karşılaştırması




Şekil 4.31. Fisher Iris veri setinde algoritmaların Recall üzerinden karşılaştırması

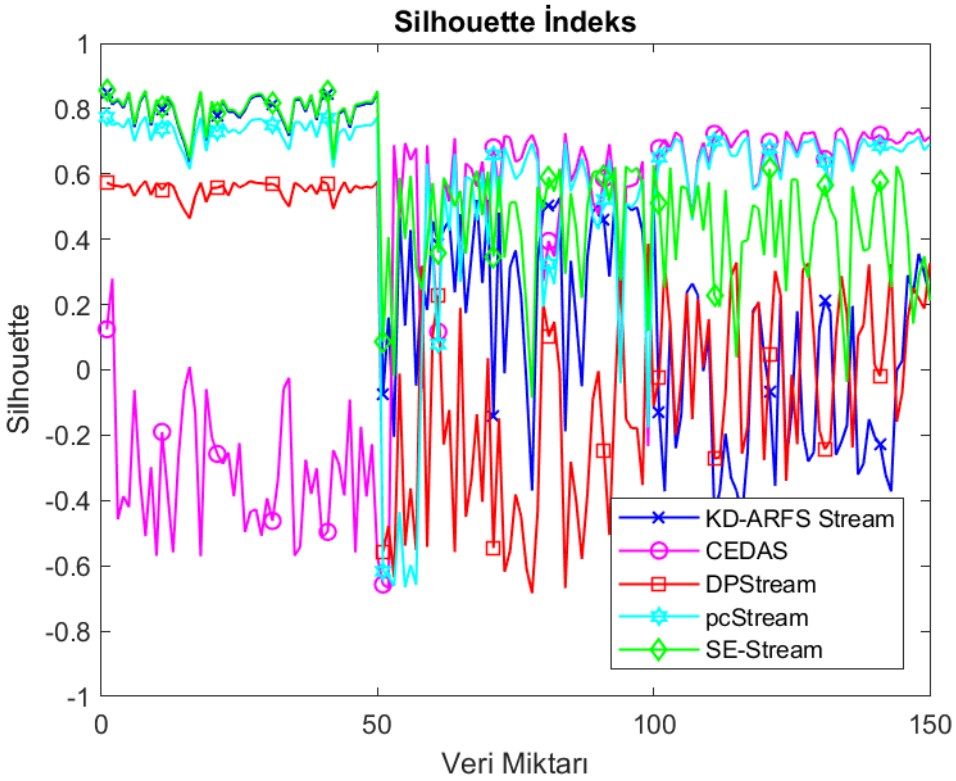
Recall testi göz önüne alındığı zaman CEDAS algoritmasının başarısında bir düşüş söz konusudur. Diğer algoritmaların başarısı %100 olarak görünmektedir. Bunun nedeni CEDAS dışında kalan algoritmalar FN sayısını 0 olarak tespit etmesidir. Şekil 4.31 incelendiğinde KD-ARFS Stream'in maksimum başarıya ulaştığı görülmektedir.

KD-ARFS Stream algoritması diğer test parametrelerinde olduğu gibi yüksek başarı elde etmektedir. Bir başka deyişle Precision ve Recall değerlerinin önerdiğimiz yaklaşım açısından çok dengeli olduğunu söylemek mümkündür. Şekil 4.32'de de görüldüğü gibi KD-ARFS Stream en yüksek F-Score değerini elde etmektedir.




Şekil 4.32. Fisher Iris veri setinde algoritmaların F-Score üzerinden karşılaştırması

Fisheriris veri setinde ikinci ve üçüncü kümelerin birbirine çok yakın olması Silhouette indeks değerinin KD-ARFS Stream açısından nispeten daha düşük çıkmasına neden olmaktadır. Çünkü Accuracy testine göre söz konusu bu verilerin farklı iki kümeye atanması başarıyı arttırırken, Silhouette indeks değerinin düşük çıkmasına sebep olmaktadır. Buna rağmen önerdiğimiz yaklaşımın yüksek bir Silhouette indeks başarısı yakaladığını söylemek mümkündür. Şekil 4.33 algoritmaların Silhouette indeks değerlerini göstermektedir.

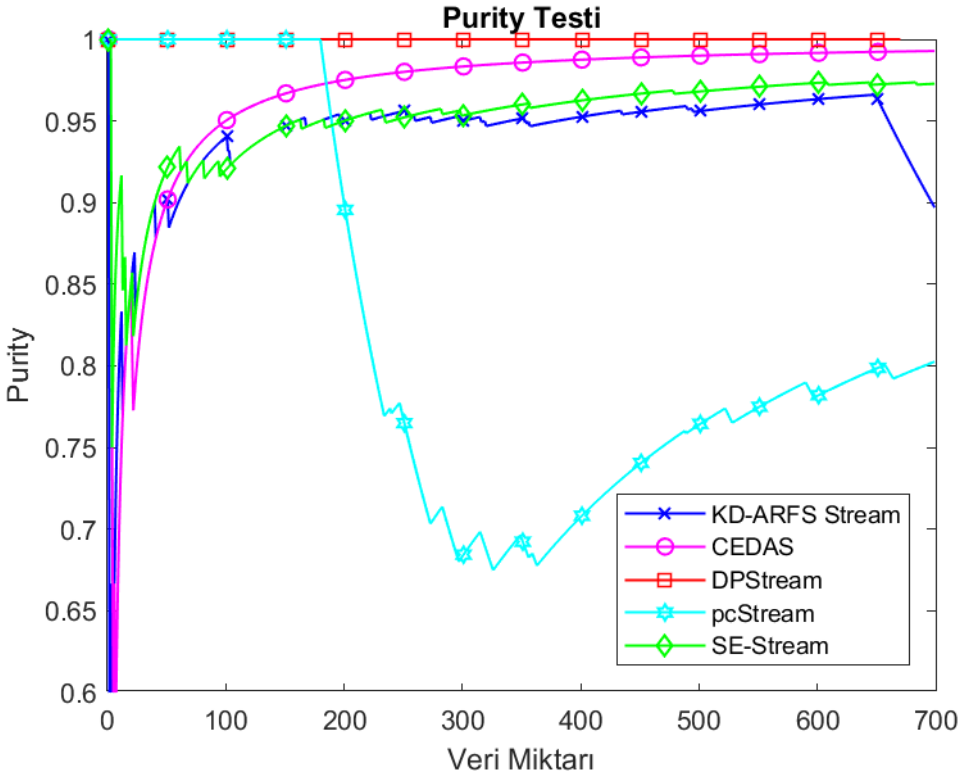



Şekil 4.33. Fisher Iris veri setinde algoritmaların Silhouette üzerinden karşılaştırması

Breast Cancer veri seti ile algoritmaların performanslarının karşılaştırması

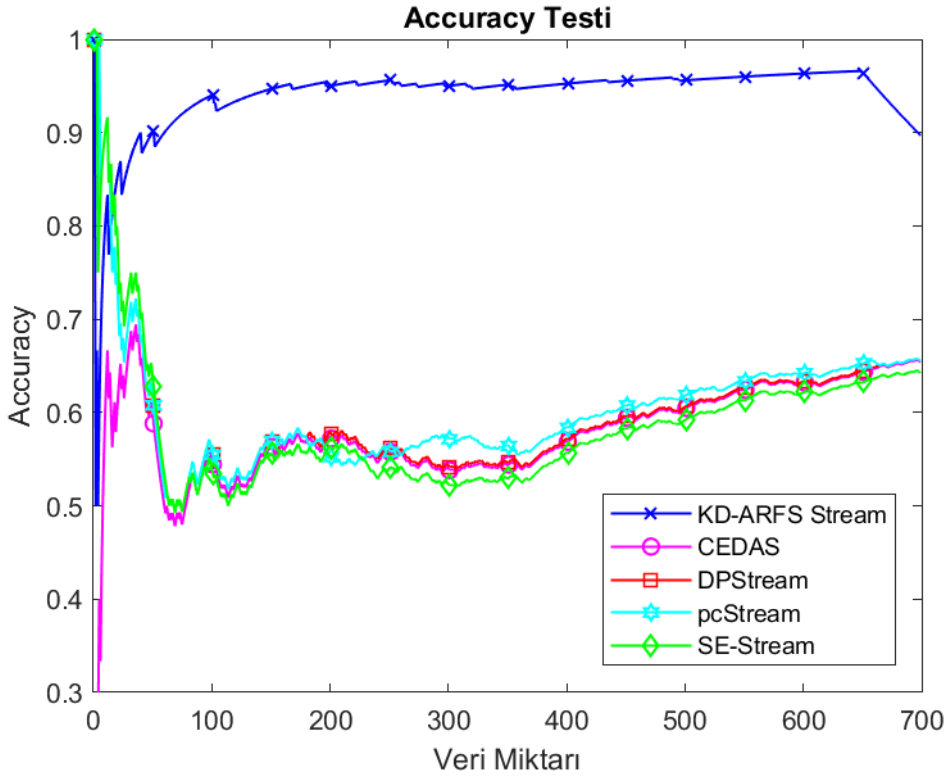
Beş algoritmanın Breast Cancer veri seti üzerinde test edilmesinde Çizelge 4.4, 4.5, 4.6, 4.7 ve 4.8'de belirtilen parametreler kullanılmıştır. Her beş algoritma için de en yüksek Accuracy sonucunu üreten parametreler tespit edilerek test edilmiştir.

Purity testine göre en yüksek başarı oranını DPStream göstermektedir. Ancak bir veri setindeki tüm verileri tek bir etikete atamak %100'lük bir Purity değerine ulaşmayı sağlar. Oysa bu veri setinde 2 tane sınıf etiketi bulunmaktadır. Bu nedenle Purity testi tek başına değerlendirme yapmak için yeterli değildir. Ancak genel anlamda tüm algoritmaların %90 ve üstü Purity değerini yakaladığını söylemek gerekir. Önerdiğimiz yaklaşımın da %95'lere yakın bir Purity değeri elde etmesi oldukça başarılı bir orandır. Özellikle 650'den sonra KD-ARFS Stream algoritmasının başarısındaki düşüşün nedeni söz konusu bölümde Kanser ve Kanser Değil kümelerinin birbirine oldukça yakın olmasıdır. Şekil 4.34 algoritmaların Purity değerlerini karşılaştırmaktadır.



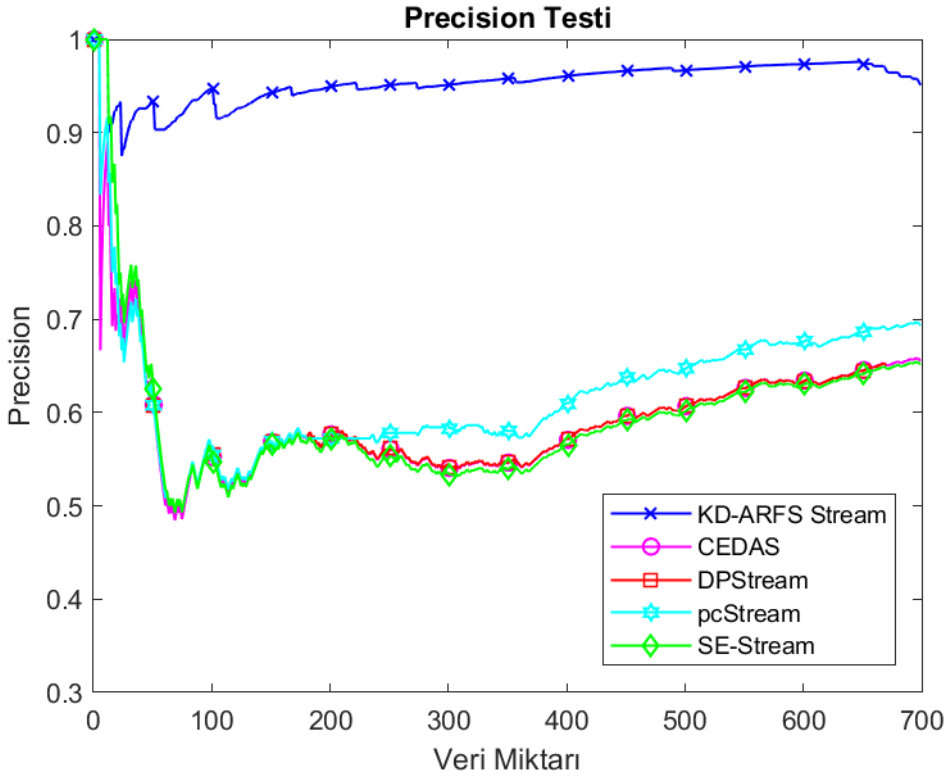

Şekil 4.34. Breast Cancer veri setinde algoritmaların Purity üzerinden karşılaştırması

Şekil 4.35'te de görüldüğü gibi Accuracy testine göre KD-ARFS Stream en yüksek başarıya sahip olan algoritmadır. KD-ARFS Stream'in özellikle evrimsel yapısı sayesinde kümelerin zaman içinde aktif pasif olması başarısındaki en önemli etkendir. Diğer algoritmalardaki düşüşün nedeni yanlış kümelere aktarılan verilerin sayısının KD-ARFS Stream'den daha fazla olmasıdır. Diğer algoritmaların grafiklerinin birbirlerine paralellik göstermesi dikkat çekicidir. Söz konusu algoritmaların aynı noktalarda benzer hatalar yaptıklarını söylemek mümkündür.

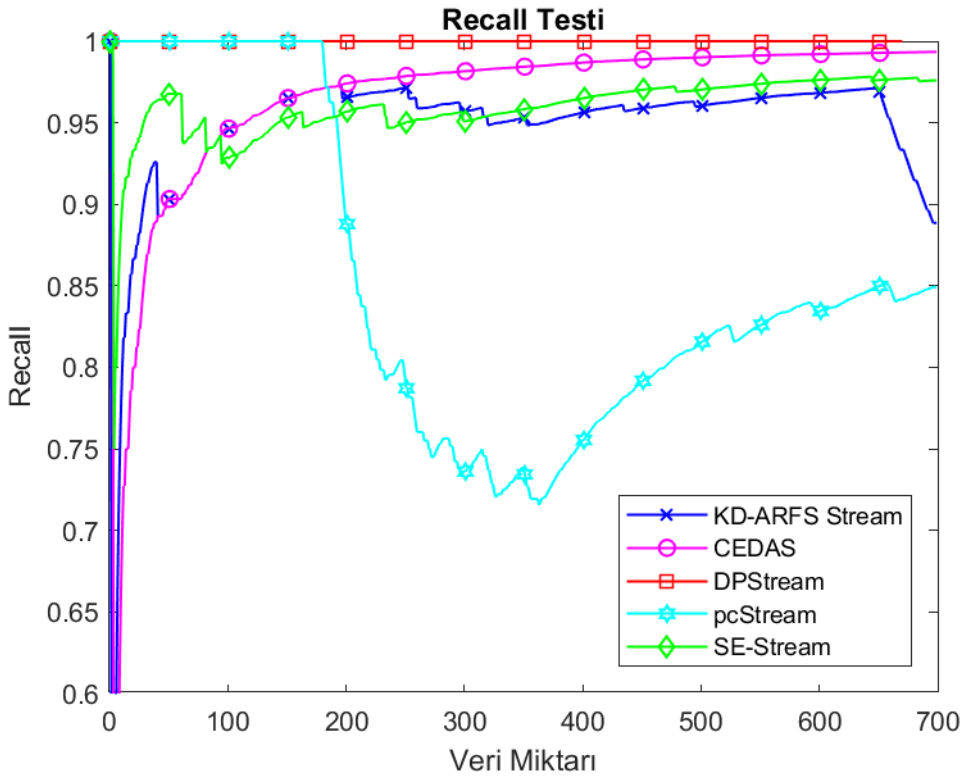



Şekil 4.35. Breast Cancer veri setinde algoritmaların Accuracy karşılaştırmaları

Şekil 4.36'da da görüldüğü gibi, Precision testi Accuracy testine benzer bir yapıdadır. KD-ARFS Stream algoritmasının diğer algoritmalara göre etiketlediği verilerin etiketleme sonucunun diğerlerinden daha iyi olduğunu söylemek mümkündür. Yani bir kümeye ait olduğunu belirttiği verilerin büyük çoğunluğu gerçekten o kümeye aittir. Başka bir deyişle FP sayısı oldukça azdır. Bunu başarmasını sağlayan en önemli etken yine evrimsel yapısıdır. Breast Cancer veri setinde farklı zamanlarda Kanser veya Kanser Değil Kümeleri oluştuğundan bu yapıyı desteklemek Precision değerinin doğrudan etkilemektedir.



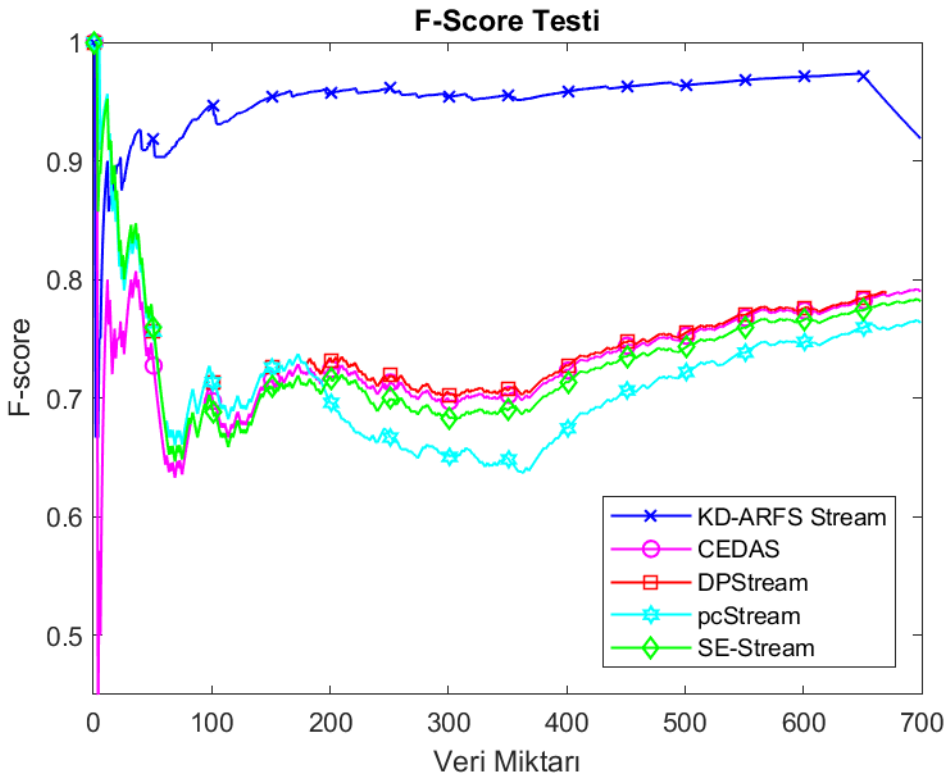

Şekil 4.36. Breast Cancer veri setinde algoritmaların Precision karşılaştırmaları




Şekil 4.37. Breast Cancer veri setinde algoritmaların Recall üzerinden karşılaştırması

Şekil 4.37'de de görüldüğü gibi Recall testine göre pcStream dışında bütün algoritmalar yüksek başarı oranlarını yakalamaktadır. FN sayısı Recall değerini doğrudan etkilemektedir. KD-ARFS Stream açısından baktığımız zaman %95 civarı bir Recall değerine sahiptir. Bu da söz konusu veri seti için iyi bir başarı oranı olduğu söylenebilir.

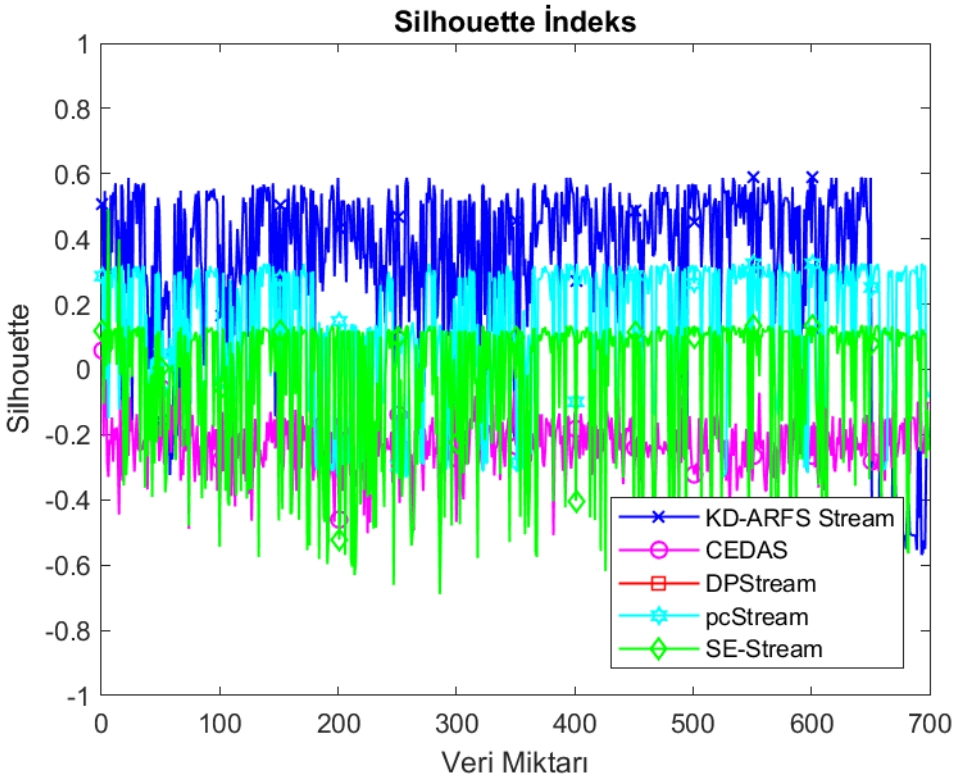
Şekil 4.38'de de görüldüğü gibi F-Score grafiğine göre en yüksek başarıyı KD-ARFS Stream göstermektedir. Dolayısıyla Precision-Recall oranının daha dengeli olduğunu söylemek de mümkündür. Yani diğer dört algoritmanın hataya düştüğü noktalarda KD-ARFS Stream'in daha az hata yaptığını söylemek mümkündür. pcStream algoritması dışındaki diğer üç algoritma yüksek Recall değerine sahip olmalarına rağmen düşük Precision değeri F-Score değerlerinin düşük olmasına sebep olmaktadır.




Şekil 4.38. Breast Cancer veri setinde algoritmaların F-Score üzerinden karşılaştırması

Şekil 4.39'da görülen Silhouette indeks grafiğine bakıldığı zaman en iyi sonucu KD-ARFS Stream'in ürettiği görülmektedir. Diğer algoritmanın eksi değerlerde sonuç ürettiği bir veri setinde KD-ARFS Stream +0.32 civarı bir ortalama tutturmaktadır. Sonuç olarak KD-ARFS Stream'in oluşturduğu kümelerin daha derli toplu; yani daha net sınırlarla belirlendiğini söylemek mümkündür. Burada DPStream açısından bir Silhouette indeks değeri

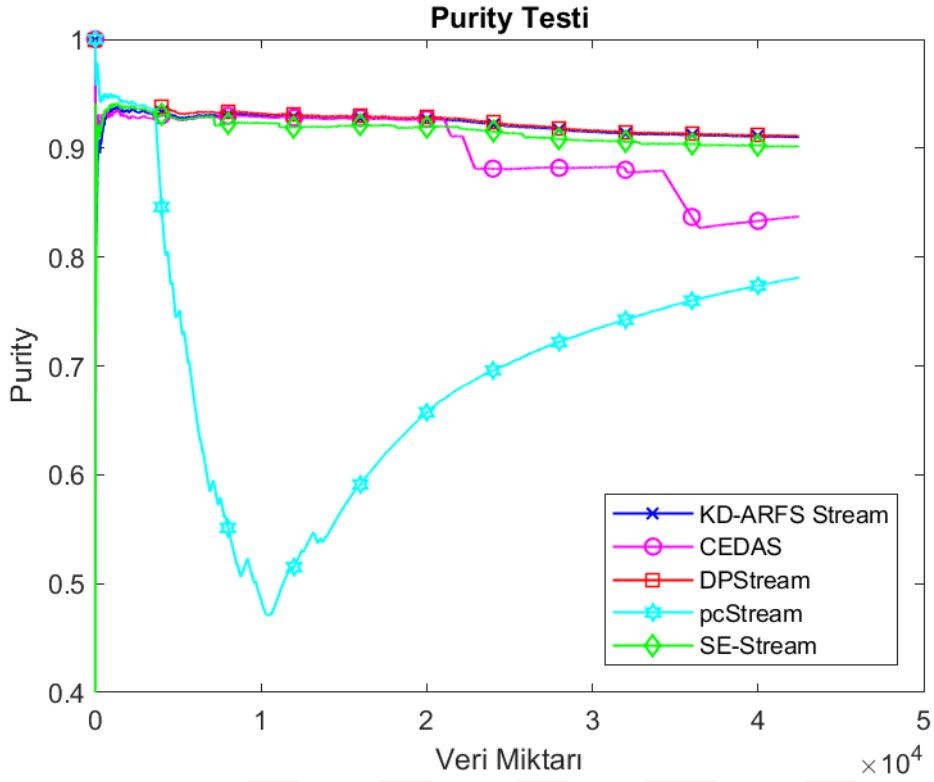
üretilmemektedir. Bunun nedeni DPStream, bütün verileri tek bir kümeye atadığından uzaklığın hesaplanabileceği bir kümenin bulunmamasıdır.




Şekil 4.39. Breast Cancer veri setinde algoritmaların Silhouette indeks karşılaştırmaları

MrData veri seti ile algoritmaların başarılarının karşılaştırması

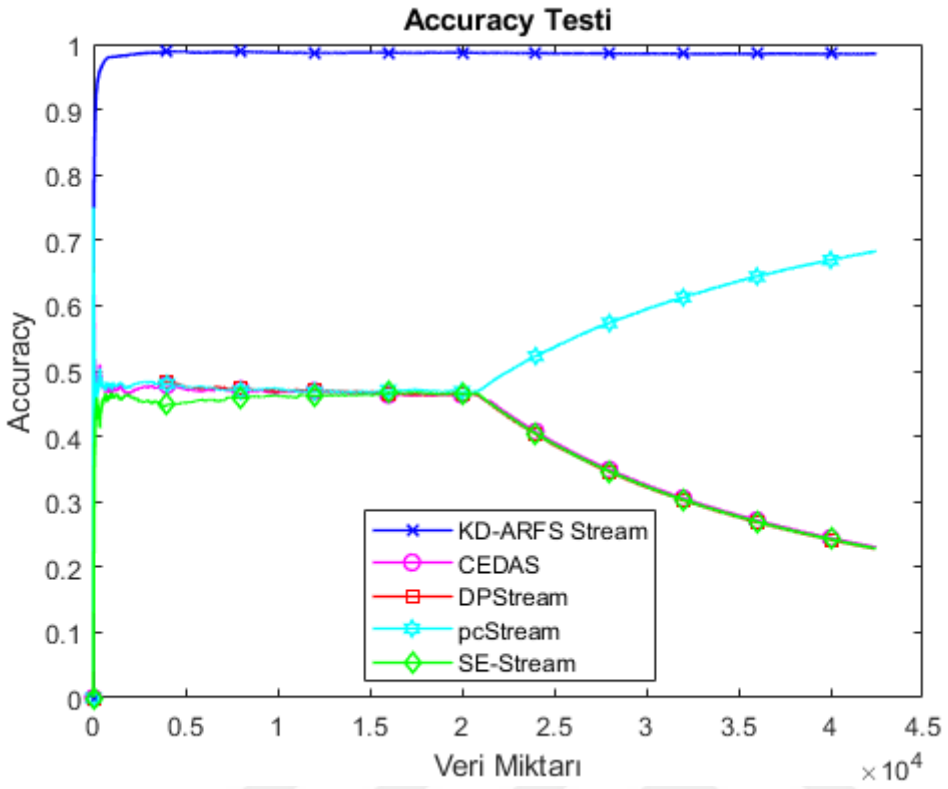
Beş algoritmanın MrData veri seti üzerinde test edilmesinde Çizelge 4.4, 4.5, 4.6, 4.7 ve 4.8'de belirtilen parametreler kullanılmıştır. MrData veri seti DPStream algoritmasında kullanılan bir veri setidir. Bu nedenle DPStream algoritması için makalede belirtilen parametreler kullanılmıştır. Diğer dört algoritma için de en yüksek Accuracy değerini üreten parametreler tespit edilerek test edilmiştir. MrData veri seti %10'luk kısmı sapan verilerden oluşan bir veri setidir. Bu nedenle algoritmaların sapan verilere karşı başarısını ölçmek adına önemli bir veri setidir.



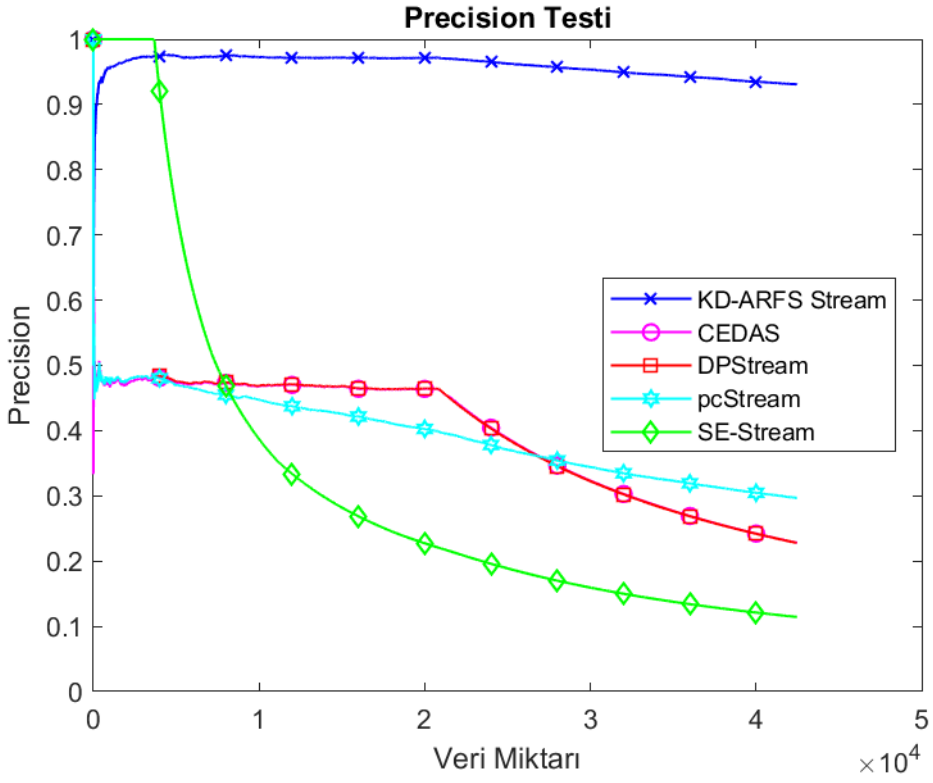

Şekil 4.40. MrData veri setinde algoritmaların Purity üzerinden karşılaştırması

Şekil 4.40'ta görülen sonuçlara göre en yüksek Purity değerlerine KD-ARFS Stream ve DPStream algoritmaları ulaşmaktadır. Bunun yanında pcStream algoritmasının ise başarısı oldukça düşüktür. Özellikle veri setinin ilk bölümünde tanımlanması gereken bir kümeyi tanımlamadığından çok büyük bir düşüş söz konusudur. İlk 21000'lik bölümde iki tane küme söz konuşur. Ancak uzun bir süre bir tanesini tespit edemediği görülmektedir. KD-ARFS Stream ise tanımlanması gereken kümeleri yerinde ve zamanında tanımlamaktadır.

Şekil 4.41'de görülen Accuracy testine göre en iyi sonucu KD-ARFS Stream üretmektedir. Yine diğer algoritmalar benzer sonuçlar üretirken önerdiğimiz yöntemin söz konusu noktalarda aynı hataları yapmaması önem arz etmektedir. %10'luk kısmı sapan veri olan bir veri setinde bu başarıyı yakalamak daha da önemlidir. Bunun yanında veri seti incelendiğinde küme şekillerinin dairesel olmadığı göz önüne alındığında bu çok iyi denilebilecek bir başarı oranıdır.

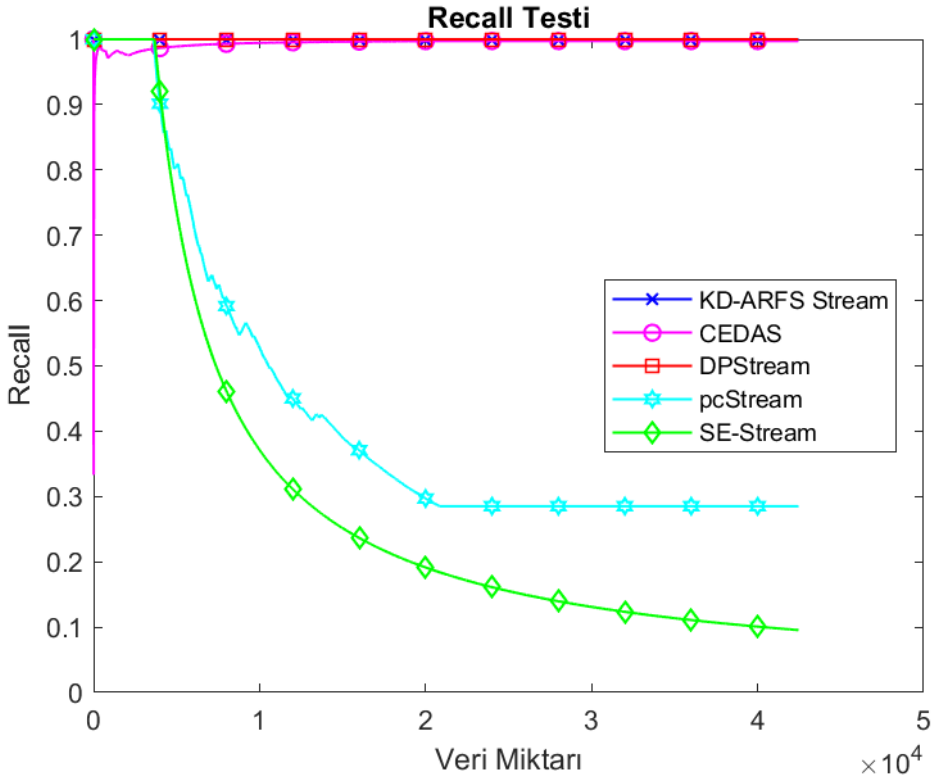



Şekil 4.41. MrData veri setinde algoritmaların Accuracy üzerinden karşılaştırması



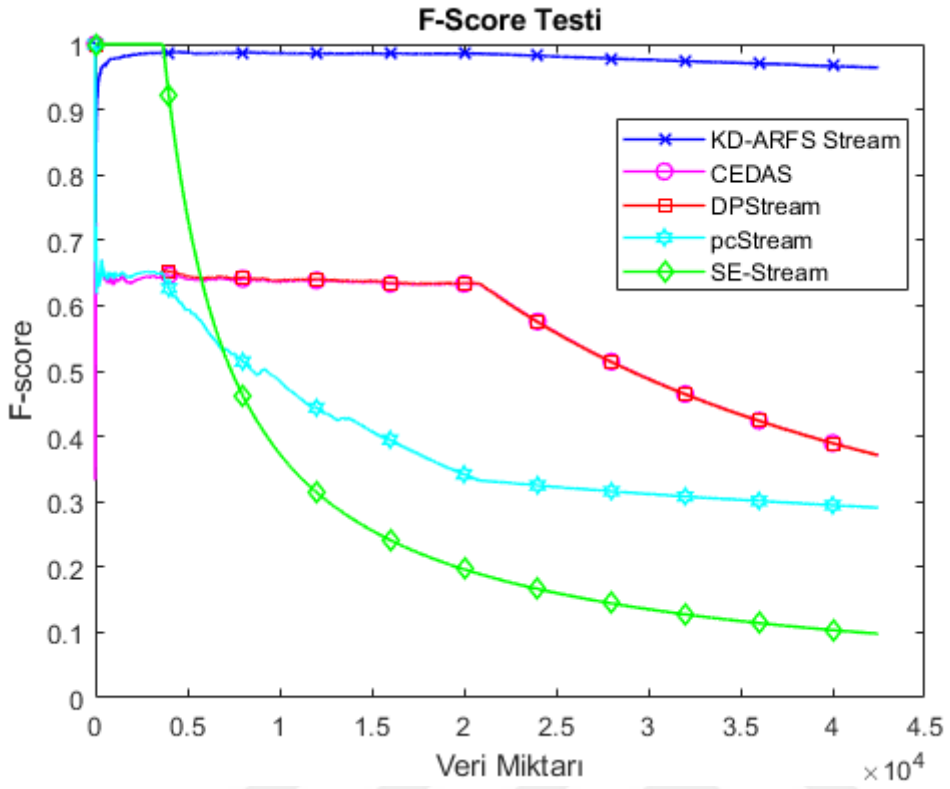

Şekil 4.42. MrData veri setinde algoritmaların Precision üzerinden karşılaştırması

Şekil 4.42'de de görülebileceği gibi Precision testinde en iyi sonucu KD-ARFS Stream'in ürettiği görülmektedir. Çünkü önerdiğimiz yaklaşımın FP değerlerinin oldukça düşük olduğunu söylemek mümkündür. Diğer algoritmalarda ise çok yüksek oranda FP değerlerinin olduğunu söylemek mümkündür.




Şekil 4.43. MrData veri setinde algoritmaların Recall üzerinden karşılaştırması

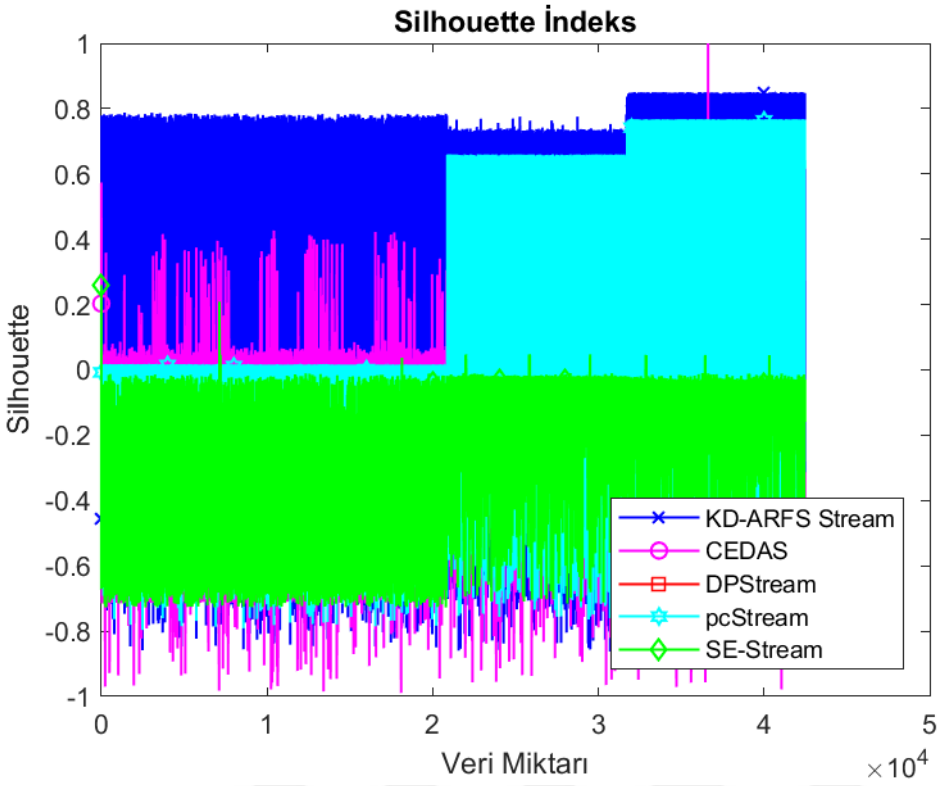
Şekil 4.43'te görüldüğü gibi Recall testine göre DPStream ve KD-ARFS Stream %100'lük başarı göstermektedir. SE-Stream ve pcStream algoritmalarında ise büyük düşüş söz konusudur. SE-Stream ve pcStream'in çok büyük oranlarda FN ürettiğini söylemek mümkündür. Burada etiketlediği verilerin çok büyük bir kısmının aslında o kümeye ait olmadığı görülmektedir.




Şekil 4.44. MrData veri setinde algoritmaların F-Score üzerinden karşılaştırması

Şekil 4.44'te görüldüğü gibi F-Score testine göre KD-ARFS Stream en iyi sonucu üretmektedir. Yine Precision ve Recall testlerinde olduğu gibi SE-Stream ve pcStream 'de çok büyük düşüş söz konusudur. F-Score'un Precision ve Recall'un harmonik ortalaması olduğu göz önüne alındığında bu beklenen bir sonuçtur.

Silhouette indeks testine göre en yüksek sonucu KD-ARFS Stream üretmektedir. +0.55 civarı bir Silhouette indeks değeri diğer algoritmalara kıyasla çok iyi denilebilecek bir başarı oranıdır. Diğer algoritmaların başarısı KD-ARFS Stream algoritmasına göre çok daha düşük kalmaktadır. Sonuç olarak KD-ARFS Stream'in oluşturduğu kümelerin kendi içinde çok daha tutarlı olduğunu söylemek gerekir.



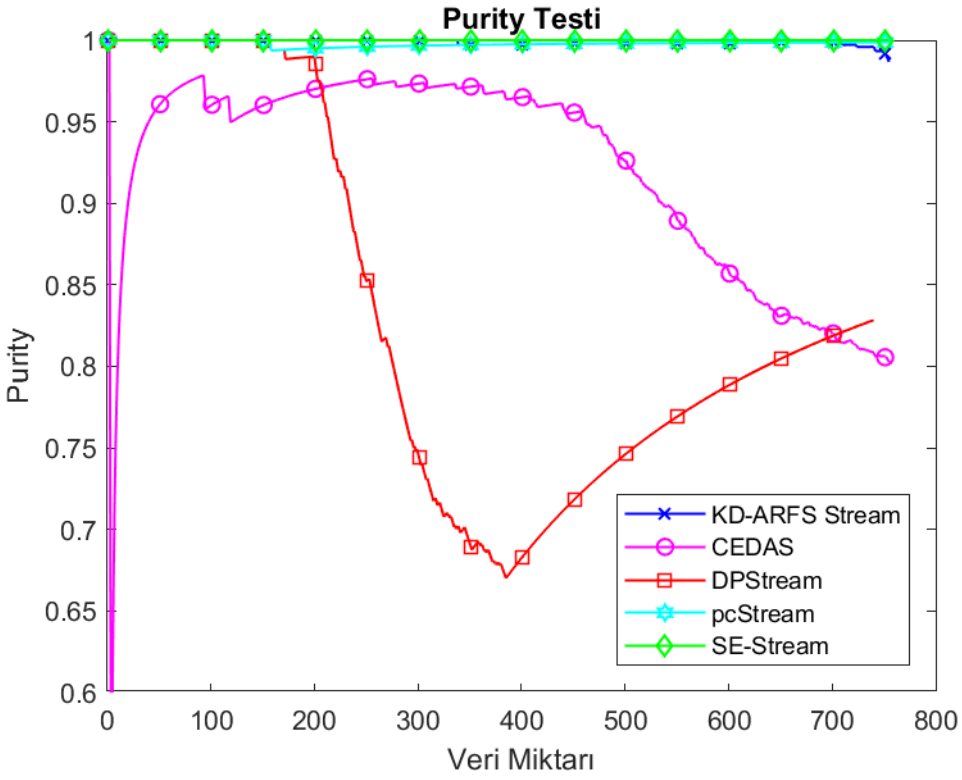

Şekil 4.45. MrData veri setinde algoritmaların Silhouette üzerinden karşılaştırması

ExclaStar veri seti ile algoritmaların performanslarının karşılaştırması

Beş algoritmanın ExclaStar veri seti üzerinde test edilmesinde Çizelge 4.4, 4.5, 4.6, 4.7 ve 4.8'de belirtilen parametreler kullanılmıştır. ExclaStar veri seti DPStream algoritmasında kullanılan bir veri setidir. Bu nedenle DPStream algoritması için makalede belirtilen parametreler kullanılmıştır. Diğer üç algoritma için de en yüksek Accuracy değerinin üreten parametreler tespit edilerek test edilmiştir. ExclaStar veri seti 755 veri ve 3 sınıftan oluşmaktadır. Algoritmaların evrimsel yapısını ortaya koymak adına önemli bir veri setidir. Çünkü oluşturulan kümelerin zaman içinde koordinatları değişmekte ve bir adet merge işlemi söz konusudur.

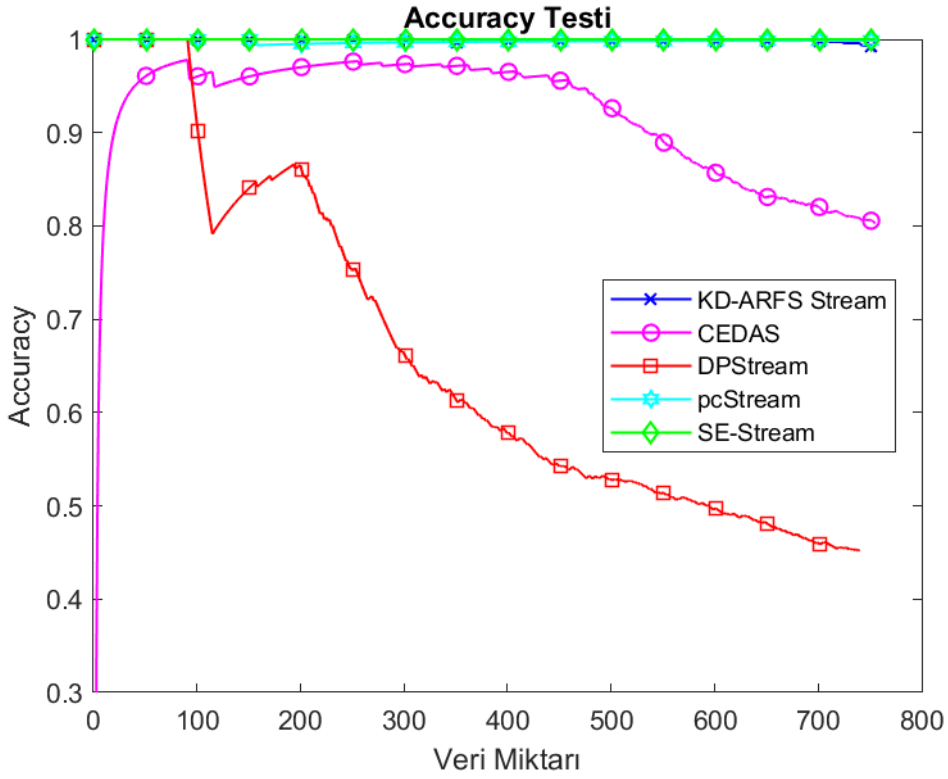
Şekil 4.46'da görülen Purity grafiğine bakıldığı zaman SE-Stream'in %100'lük bir Purity değerine sahip olduğu görülmektedir. KD-ARFS Stream ve pcStream ise %100'e çok yakın bir sonuç üretmektedir. CEDAS ve DPStream algoritmalarında ise dalgalanmalar söz konusudur. CEDAS ve DPStream algoritmaları için kümeleme işleminin diğer algoritmalar kadar başarılı olmadığını söylemek mümkün. Önerdiğimiz algoritma sön bölümde oluşan merge işlemini yapmadığından bir miktar düşüş söz konusudur. Bunun da nedeni

önerdiğimiz yaklaşımın merge işlemini aktif kümeler üzerinde yapmasıdır. Bu noktada ise kümelerden biri pasif duruma geçtiğinden yöntemimiz merge işlemini yapmamaktadır. Ancak buna rağmen çok yüksek Purity değerine ulaşmaktadır.

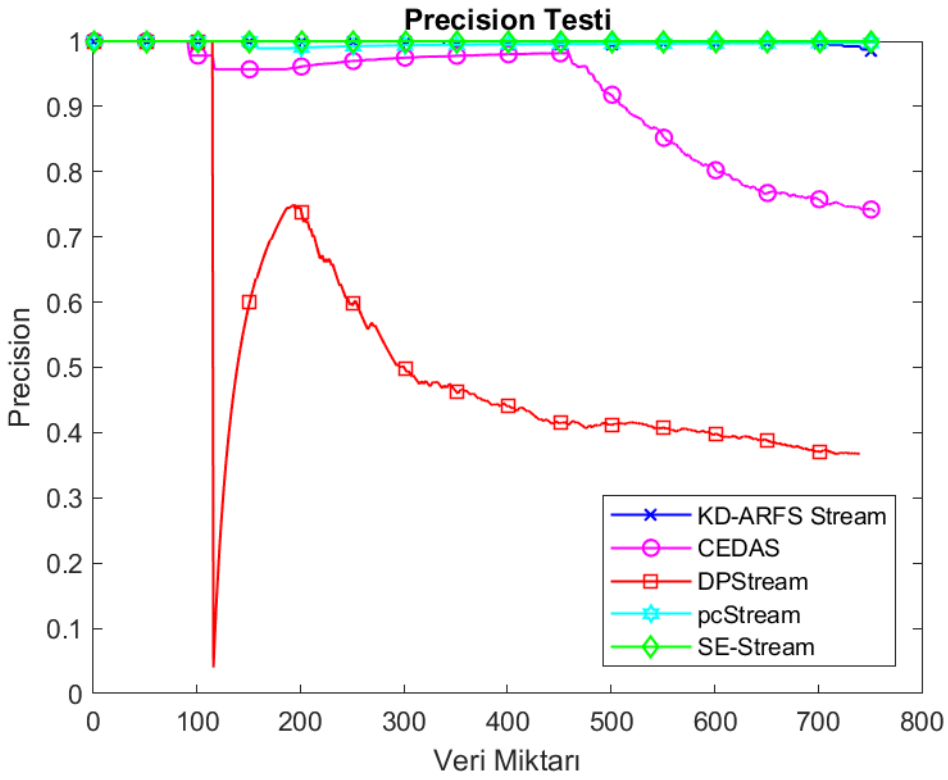



Şekil 4.46. ExclaStar veri setinde algoritmaların Purity üzerinden karşılaştırması

Şekil 4.47'de Purity testinde olduğu gibi Accuracy testinde de KD-ARFS Stream'in SE-Stream ve pcStream ile birlikte çok iyi sonuç ürettiğini söylemek mümkün. Önerdiğimiz yaklaşım kümelerinin zaman içinde evirilmesini (Küme koordinatları ve aktif-pasif dönüşümü) tam desteklediğinden yüksek Accuracy değerlerine ulaşmaktadır.



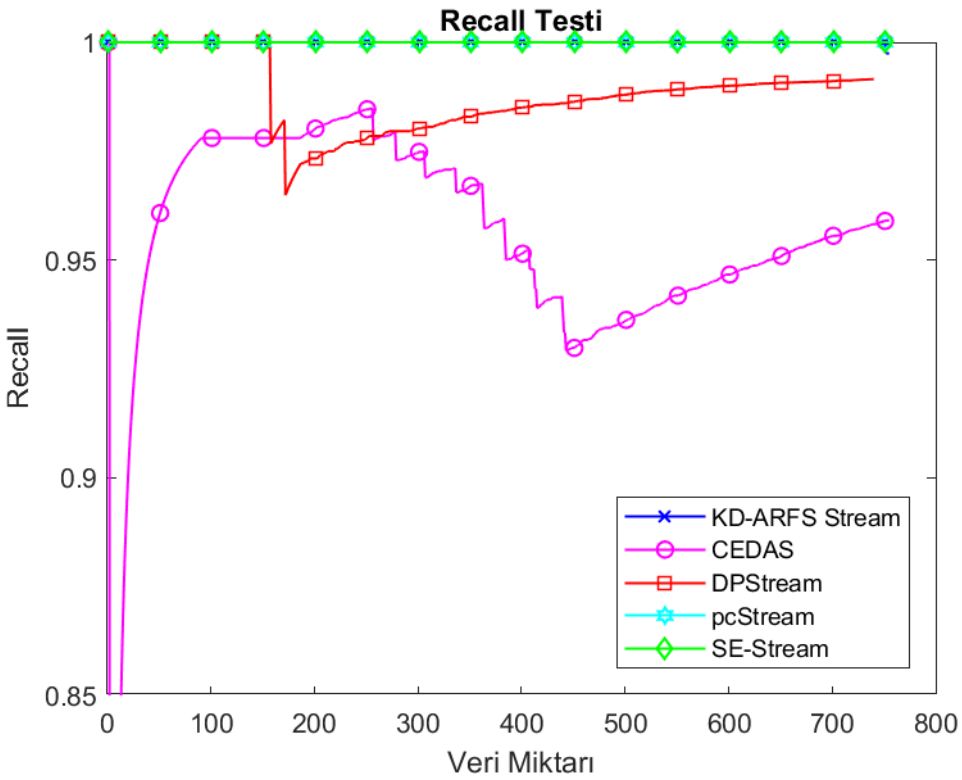

Şekil 4.47. ExclaStar veri setinde algoritmaların Accuracy üzerinden karşılaştırması




Şekil 4.48. ExclaStar veri setinde algoritmaların Precision üzerinden karşılaştırması

Precision testine göre CEDAS ve DPStream algoritmalarının bir kümeye ait olarak etiketlediği verilerin gerçekte önemli bir kısmının o kümeye ait olmadığı söylenebilir. Yani FP sayısının yüksek olduğunu söylemek mümkündür. Oysa KD-ARFS Stream algoritmasında FP oranı oldukça düşüktür. Şekil 4.48 algoritmaların Precision sonuçlarını göstermektedir.

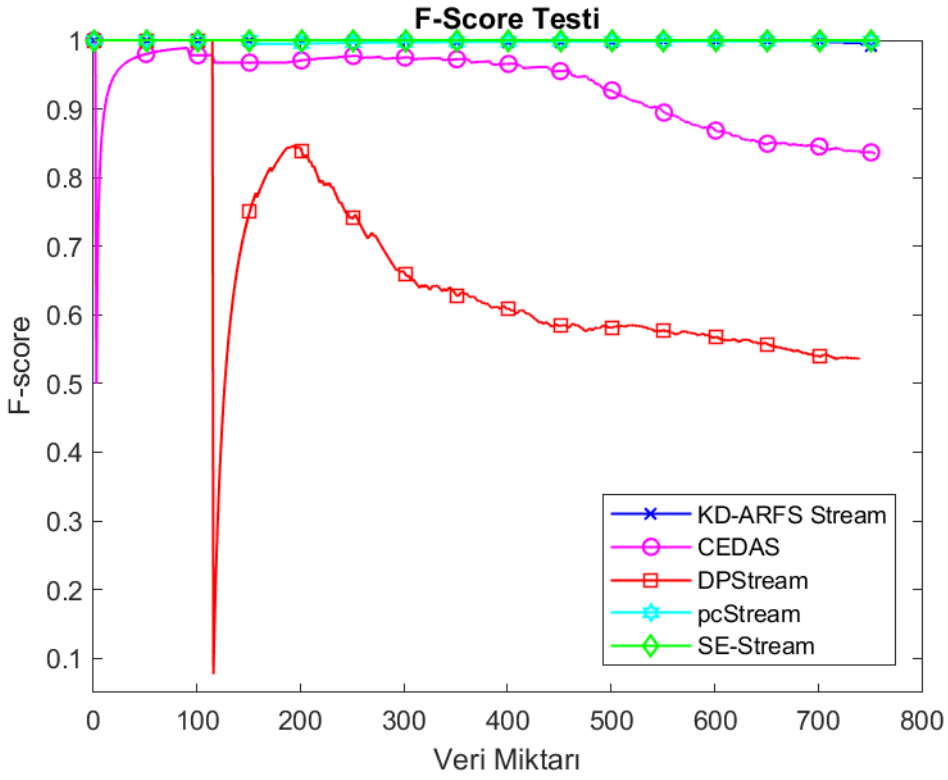
Şekil 4.49'da görülen Recall testi incelendiği zaman ExclaStar veri setine ait önceki grafiklerde olduğu gibi yine DPStream ve CEDAS algoritmalarının başarı oranlarının daha düşük olduğu görünmektedir. Ancak buradaki düşüş önceki grafiklere oranla daha azdır. Bunun nedeni FN sayısının az olmasından kaynaklanmaktadır. Yani gerçekte bir kümeye ait verilerden başka bir kümeye atanan veri sayısı azdır. KD-ARFS Stream ise yine çok yüksek Recall değerlerine ulaşmaktadır. Çünkü KD-ARFS Strean algoritması, ExclaStar veri setinin evrimsel yapısına tam uyum sağlamaktadır.




Şekil 4.49. ExclaStar veri setinde algoritmaların Recall üzerinden karşılaştırması

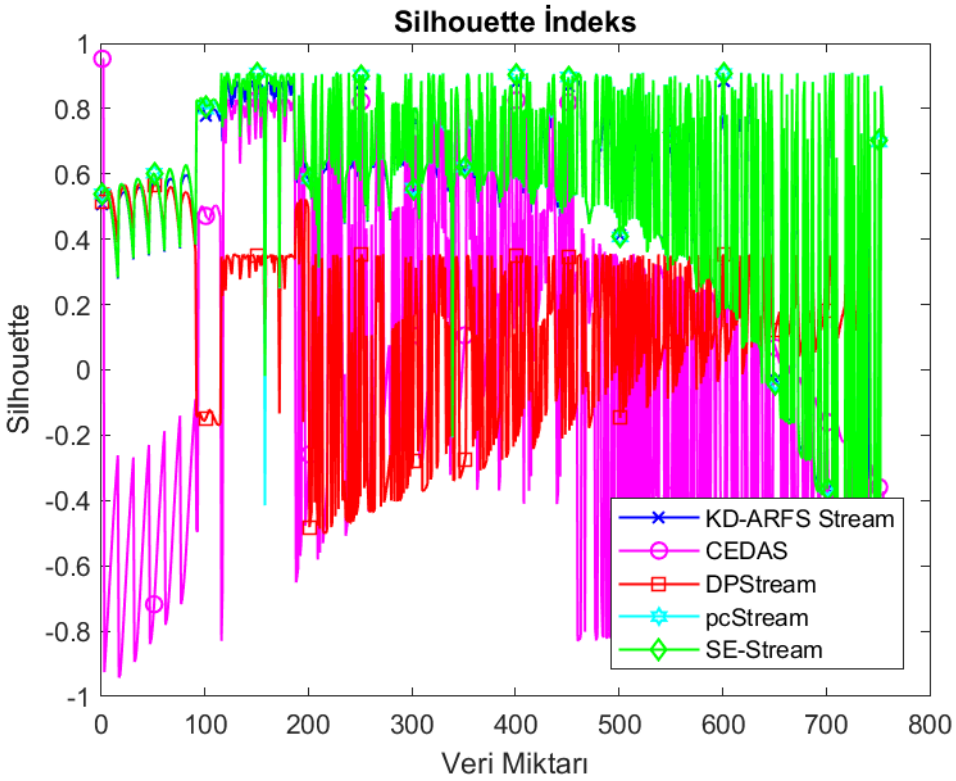
Şekil 4.50'de görülen F-Score grafiğine göre yine CEDAS ve DPStream algoritmaları daha düşük başarı göstermektedir. Bu düşüşe neden olan etken Precision değerlerinin düşük olmasıdır. Yani bir kümeye ait olmadığı halde o kümeye aitmiş gibi etiketlenen veri sayısının

fazla olması bu iki algoritmanın F-Score değerlerini düşürmektedir. Bunun yanında KD-ARFS Stream ise çok yüksek başarı oranları yakalamaktadır. Çünkü çok yüksek Precision ve Recall değerlerine ulaştığından F-Score değeri de yüksek çıkmaktadır.




Şekil 4.50. ExclaStar veri setinde algoritmaların F-Score üzerinden karşılaştırması

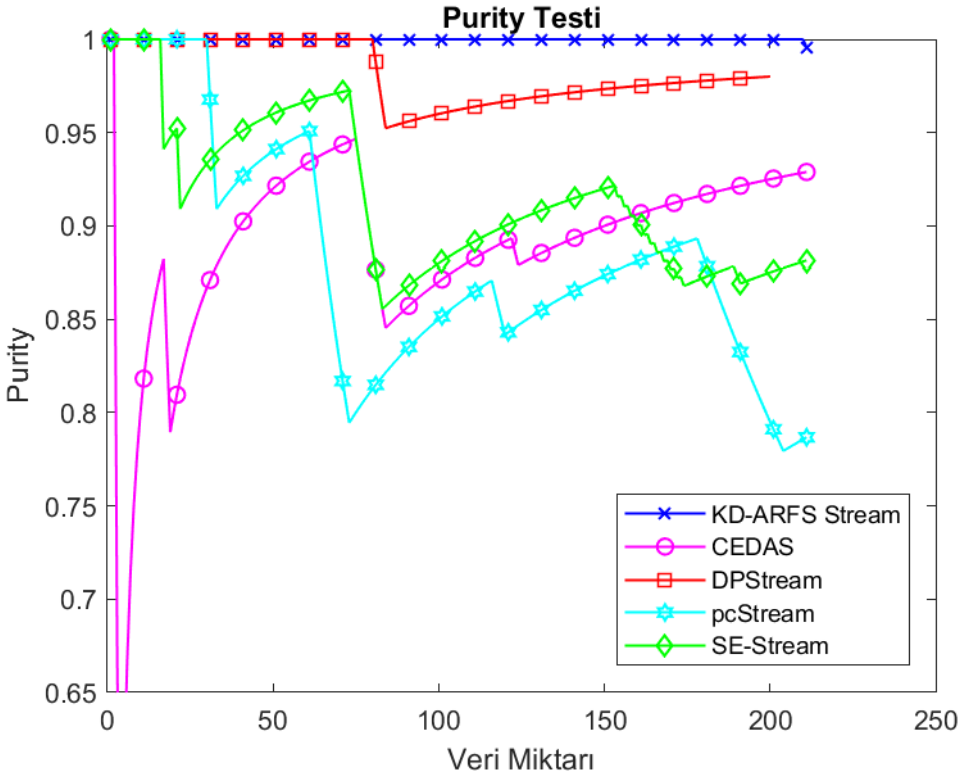
Şekil 4.51'de görüldüğü gibi, önceki grafiklere benzer şekilde yine en yüksek sonuçları Silhouette indekste de KD-ARFS Stream ve SE-Stream algoritmaları üretmektedir. Yani kümeleme işlemi KD-ARFS Stream algoritmasında oldukça düzgündür. Aynı kümedeki verilerin birbirlerine olan uzaklıklarının ortalamasının diğer kümelere olan uzaklık ortalamalarına göre bu iki algoritmada daha düşük olduğu sonucuna varılabilir.




Şekil 4.51. ExclaStar veri setinde algoritmaların Silhouette üzerinden karşılaştırması

IdealData veri seti ile algoritmaların performanslarının karşılaştırması

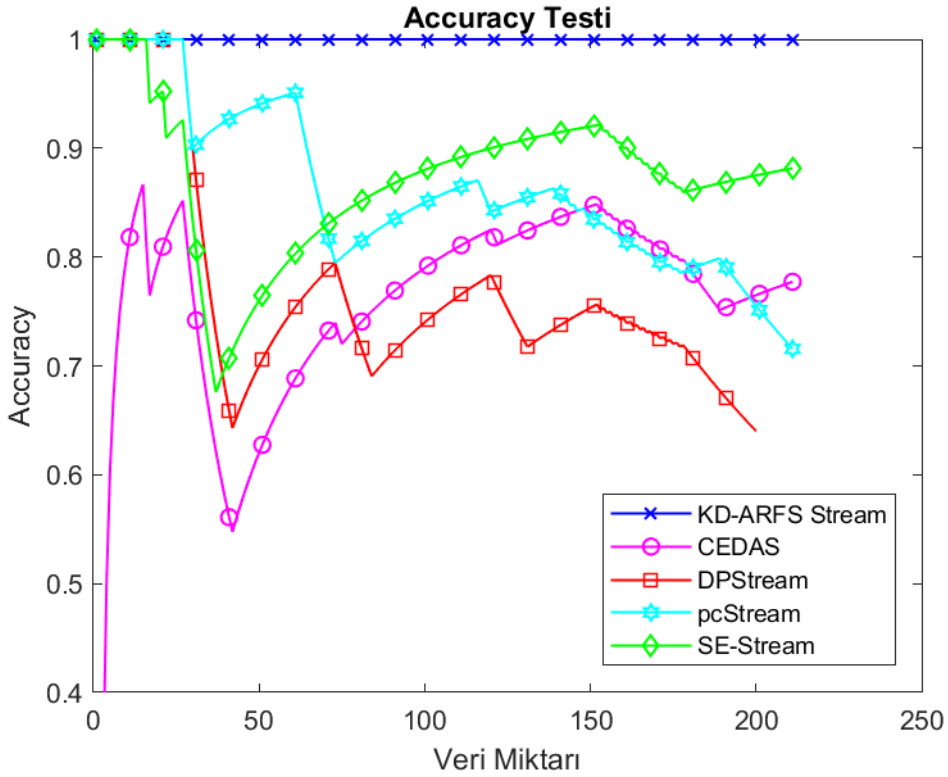
Beş algoritmanın IdealData veri seti üzerinde test edilmesinde Çizelge 4.4, 4.5, 4.6, 4.7 ve 4.8'de belirtilen parametreler kullanılmıştır. IdealData veri seti bu çalışmada önerdiğimiz yöntemin split ve merge işlemlerinin nasıl gerçekleştiğini görmek için oluşturduğumuz sentetik bir veri setidir. Test işleminde her beş algoritma için de en yüksek Accuracy değerini üreten parametreler tespit edilerek test edilmiştir.




Şekil 4.52. IdealData veri setinde algoritmaların Purity üzerinden karşılaştırması

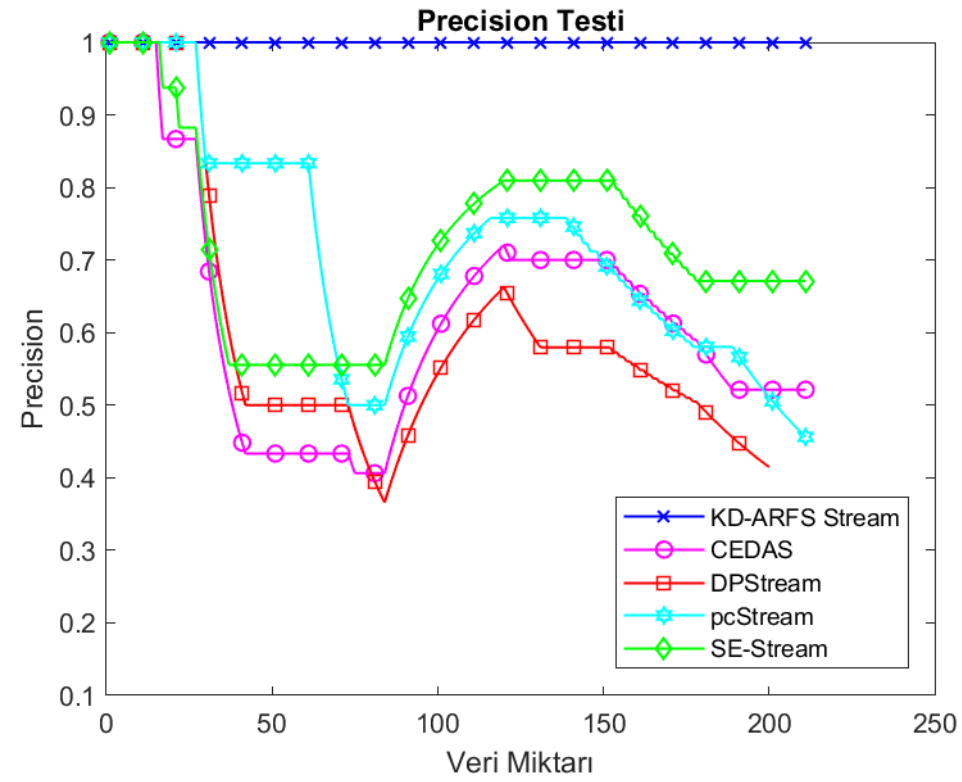
Şekil 4.52'de de görüldüğü gibi Purity testine göre KD-ARFS Stream %100'lük bir başarı oranı yakalamaktadır. Diğer algoritmalarda ise dalgalanmalar söz konusudur. Bu dalgalanmaların nedeni merge-split işlemlerinin KD-ARFS Stream kadar başarılı bir şekilde yapılamamasıdır. Yani KD-ARFS Stream tam evrimsel bir algoritma olduğundan kümelerin yapısal değişimlerini tam desteklemektedir. Bu da başarısının çok yüksek çıkmasını sağlamaktadır. Purity testi de bu gerçeği destklemektedir.

Şekil 4.53'te de görüldüğü gibi, Accuracy testinde de önerdiğimiz yöntemin %100'lük bir başarı yakaladığını görmekteyiz. Diğer algoritmalarda ise dalgalanma söz konusudur. Özellikle 40-50 aralığında KD-ARFS Stream dışındaki bütün algoritmalarda çok keskin bir düşüş söz konusudur. Bunun nedeni söz konusu aralıkta yeni bir küme tanımlanmalıdır. Ancak söz konusu algoritmalar bunu tespit edememektedir.

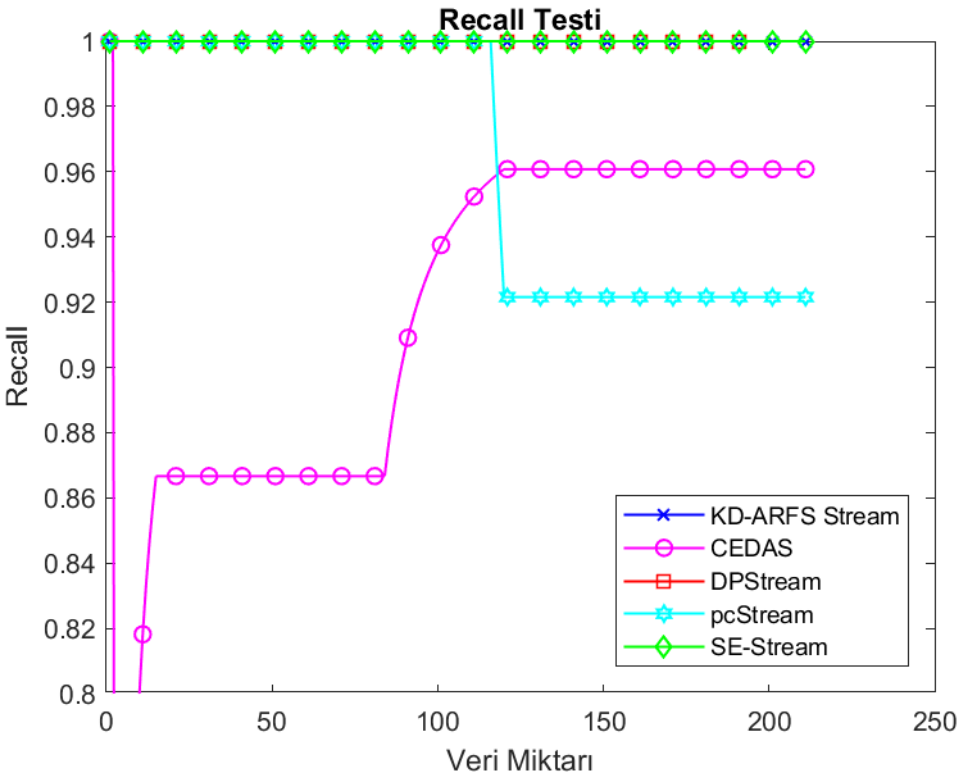



Şekil 4.53. IdealData veri setinde algoritmaların Accuracy üzerinden karşılaştırması

Şekil 4.54'te de görülebileceği gibi Accuracy testine benzer bir durum Precision testinde de söz konusudur. Ancak önerdiğimiz algoritma dışındaki algoritmalarda düşüş Accuracy testine göre çok daha fazladır. Accuracy testinde de üzerinde durduğumuz gibi tanımlanması gereken kümeler tanımlanmamaktadır. Bu nedenle FP sayısı artmakta ve başarı oranı düşmektedir. Yani tanımlanması gereken kümeler tanımlanmadığından söz konusu kümeye ait veriler daha önce tanımlanmış kümelere atandığından başarı oranı düşmektedir.

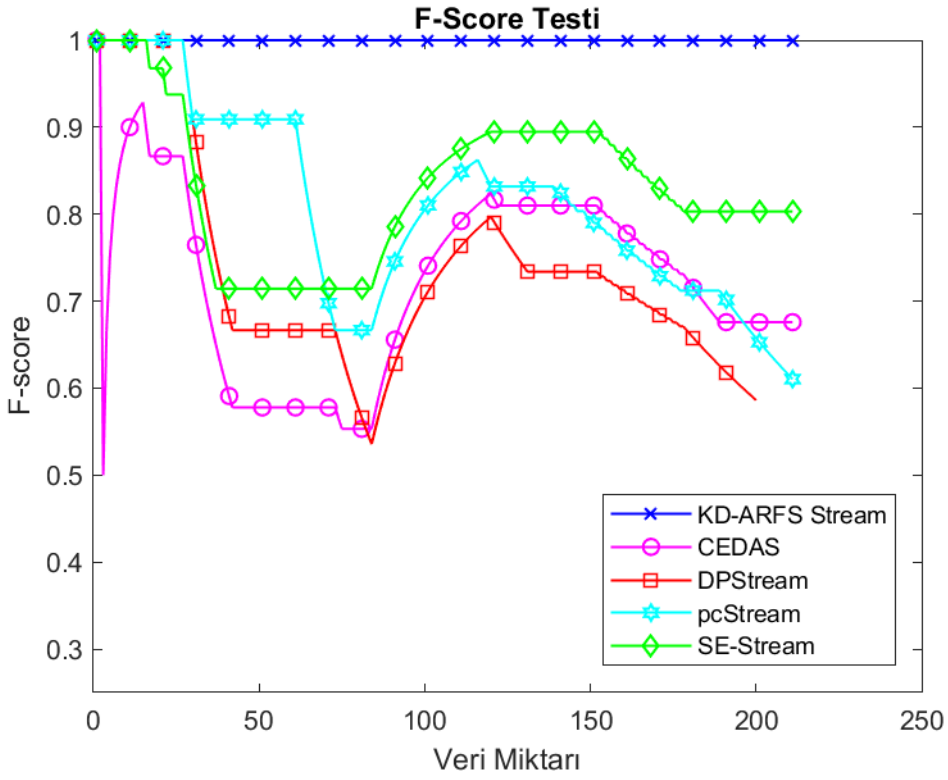



Şekil 4.54. IdealData veri setinde algoritmaların Precision üzerinden karşılaştırması




Şekil 4.55. IdealData veri setinde algoritmaların Recall üzerinden karşılaştırması

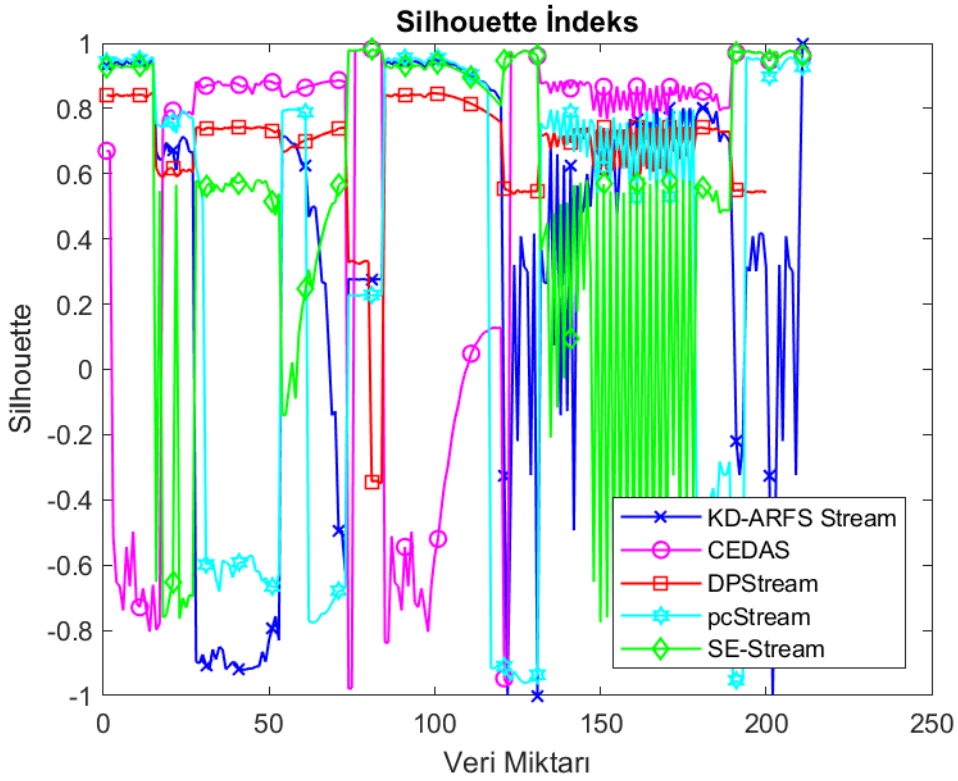
Recall testinde önceki testlerden farklı olarak CEDAS ve pcStream dışındaki bütün algoritmalar %100'lük başarıyı yakalamaktadır. Bunun nedeni tanımlanmış kümelerde olması gereken verilerin tamamı söz konusu kümelerde bulunmaktadır. Dolayısıyla FN sayısı 0'dır. pcStream ve CEDAS algoritmalarında bu durum geçerli değildir. Olması gereken kümede olmayan veri sayısı oldukça fazladır. Şekil 4.55 algoritmaların Recall sonuçlarını karşılaştırmaktadır.




Şekil 4.56. IdealData veri setinde algoritmaların F-Score üzerinden karşılaştırması

Şekil 4.56'da görülen F-Score grafiğine göre yine önerdiğimiz algoritma dışındaki diğer algoritmalarda dalgalanma söz konusudur. Bu dalgalanmaya neden olan etken Precision değerlerinin düşük olmasıdır. Yani bir kümeye ait olmadığı halde o kümeye aitmiş gibi etiketlenen veri sayısının fazla olması diğer algoritmaların F-Score değerlerini düşürmektedir. Çünkü IdealData veri setinde zaman içinde kümeler tanımlanmakta, birleşmekte ve bölünmektedir. Bu işlemi tam olarak destekleyen algoritma önerdiğimiz KD-ARFS Stream algoritmasıdır.

Şekil 4.57'de görülen Silhouette indeks grafiği incelendiğinde KD-ARFS Stream'in +0.50 civarı bir başarı oranı ile en iyi olmasa da yine çok iyi denilebilecek bir başarı oranı yakaladığı görülmektedir.



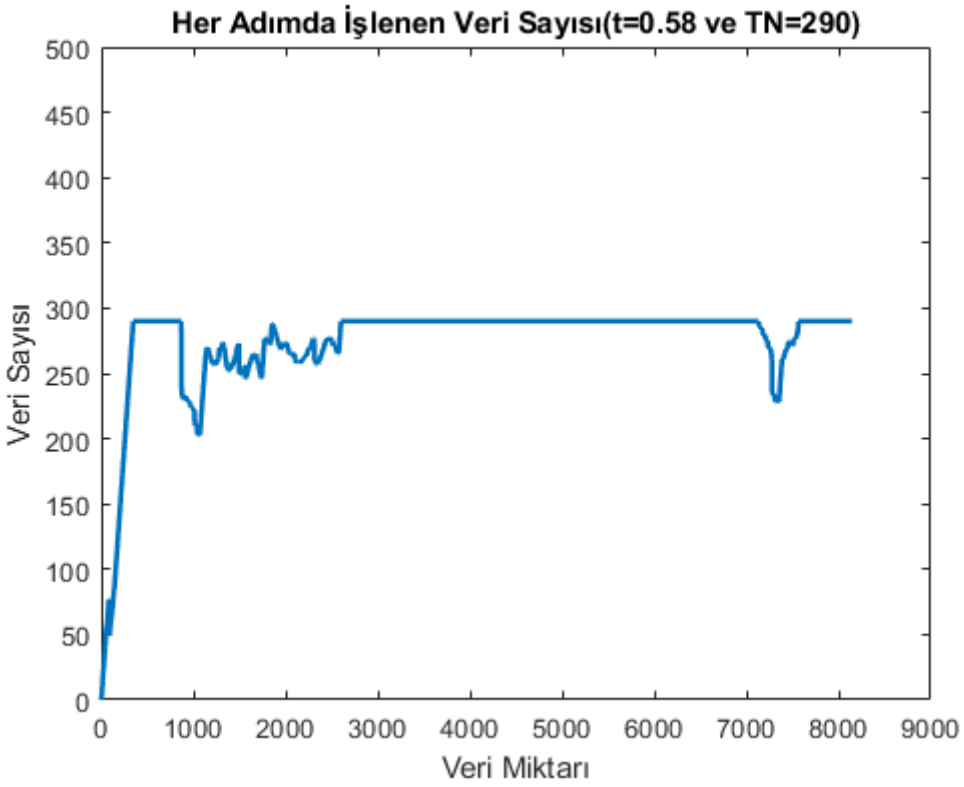

Şekil 4.57. IdealData veri setinde algoritmaların Silhouette üzerinden karşılaştırması

Zaman damgalı gerçek bir veri seti olarak Occupancy veri seti

Occupancy veri seti UCI veri havuzunda bulunan sıcaklık, nem, zaman damgası, ışık ve $CO_2$ gibi 7 adet nitelikten oluşan odaların tutulup tutulmadığı ile ilgili bilgileri içeren bir veri setidir. Bu veri setini zaman damgasına göre verilerin gelişi söz konusu olduğu için kullanmayı uygun bulduk. Ancak burada her iki veri arasındaki zaman farkı 1 sn olduğu için biz bu kısmı ile biraz oynayarak değiştirdik. Her iki veri arasındaki zaman farkı daha az ve rast gele olacak şekilde düzenledik. Bu durumda bazen sn'de 2-3 tane veri gelirken bazen de 15-20 verinin gelişi söz konusu olabilmektedir.

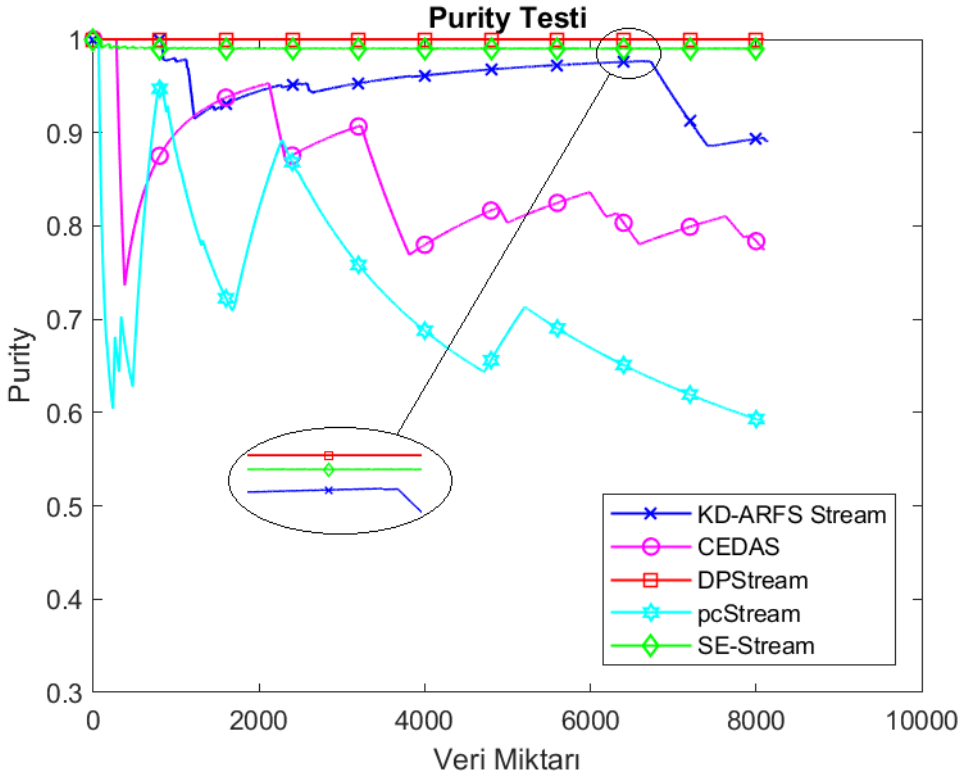
Bu veri setini seçmemizin nedeni zaman damgası olan gerçek bir veri seti söz konusu olduğunda önerdiğimiz yöntemin vereceği tepkiyi ölçmektir. Bu açıdan baktığımız zaman verinin geliş hızıyla orantılı olarak veri miktarı çoksa; yani işlenebilecek veri sayısı beklenenden çoksa; diğer bir deyişle worst-case söz konusu ise bu durumda en son gelen TN

(kullanıcının belirlediği değişken) tane veri işlenmektedir. Aşağıdaki şekilde de göründüğü ilk TN değeri 290 seçildiğinde ve t de 0.58 sn. seçildiğinde zaman zaman t zaman özetleme devreye girerken veri miktarı TN'den fazla olduğunda son gelen TN tanesi seçilmektedir. Şekil 4.58'de de görüldüğü gibi 800-2800 ve 7000-7500 aralıklarında gelen veri sayısı belirlenmiş eşik değerin altında olduğundan tüm veriler işlenmiştir. Ancak diğer zamanlarda beklenenden daha fazla veri geldiğinden veri sayısı sınırlandırılmıştır.

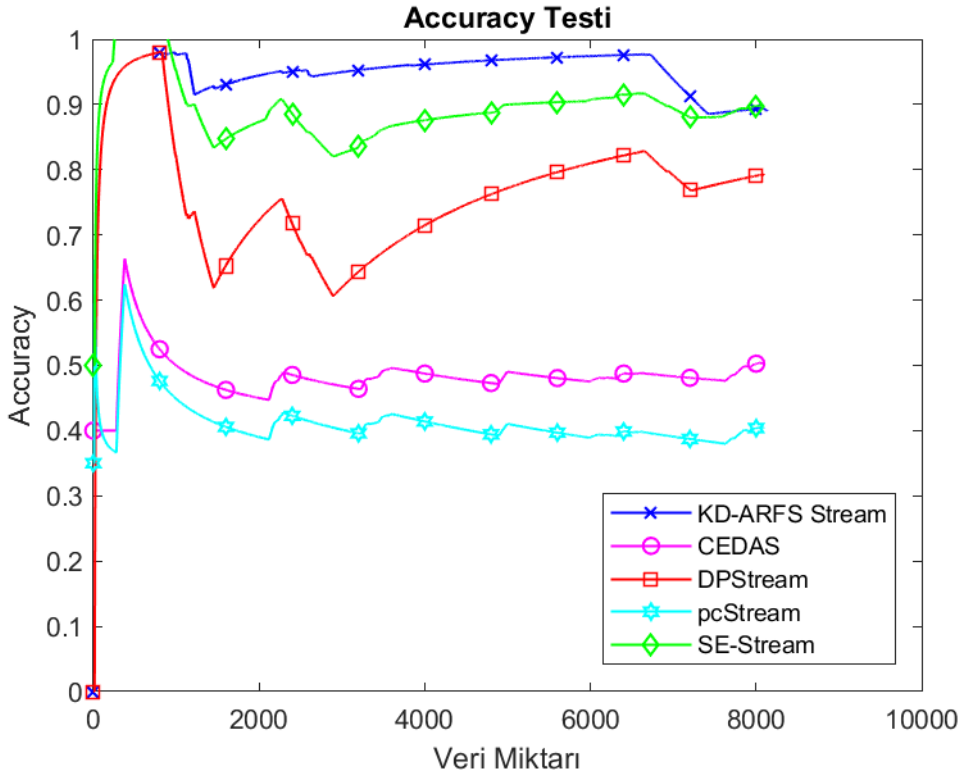



Şekil 4.58. t zamanda gelen veri çok fazlaysa KD-ARFS Stream'in veriyi sınırlaması

Tüm verileri bir kümeye atmak Purity değerinin yüksek çıkmasına neden olmaktadır. Bu nedenle DPStream algoritmasının ürettiği sonuç maksimum değerde olmaktadır. Ancak tek başına Purity testi kesin bir yargıya varmak için yeterli değildir. Şekil 4.59'da da görüldüğü gibi algoritmalarda oluşan düşüşlerin nedeni bu aralıklarda yanlış küme atamalarının yapılmış olmasıdır. Occupancy veri setinde bulunan iki küme özellikle 7000'den sonra koordinat sisteminde birbirine çok yaklaşmaktadır. Bu nedenle önerdiğimiz yaklaşım bu kümeleri birleştirmektedir. Buna rağmen önerdiğimiz KD-ARFS Stream algoritması oldukça yüksek Purity değerine sahiptir.



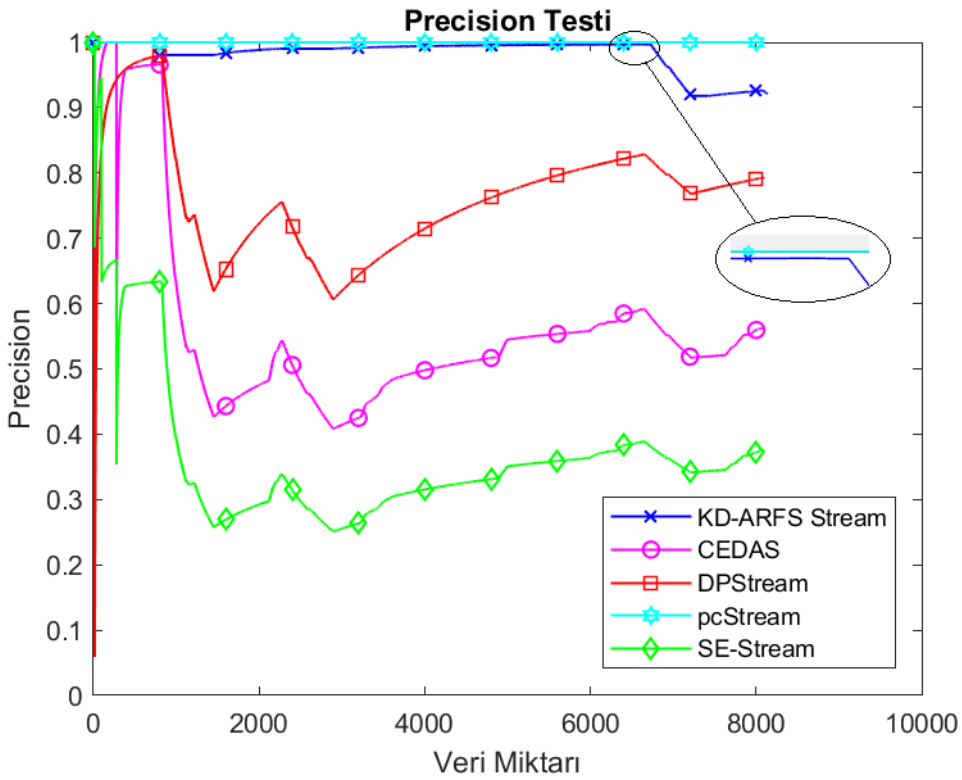

Şekil 4.59. Occupancy veri setinde algoritmaların Purity üzerinden karşılaştırması




Şekil 4.60. Occupancy veri setinde algoritmaların Accuracy üzerinden karşılaştırması

Şekil 4.60'ta da görüldüğü gibi KD-ARFS Stream algoritması en yüksek Accuracy sonucuna sahiptir. 1700 ve 7000 civarında Occupancy veri setindeki Dolu/Boş sınıflarına ait veriler birbirlerine oldukça yaklaştığından önerdiğimiz yaklaşım bu kümeleri birleştirmektedir. Bu da önerdiğimiz yaklaşımın Accuracy değerinin bir miktar düşmesine neden olmaktadır. Ancak buna rağmen önerdiğimiz yaklaşım en yüksek Accuracy başarısını elde etmektedir. Çünkü önerdiğimiz yaklaşım Occupancy veri setinde kümelerin zaman içinde aktif-pasif olma özelliğini tam desteklemektedir.

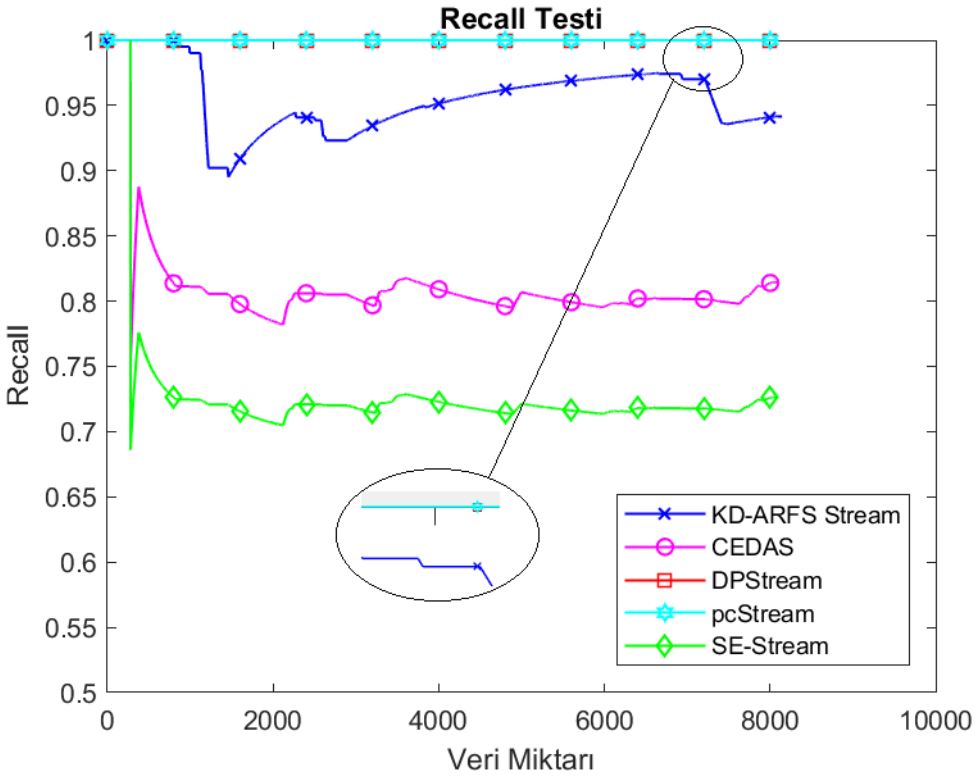
Precision testinde FP sayısı başarıyı doğrudan etkilemektedir. Şekil 4.61'de de görüldüğü gibi KD-ARFS Stream bu hedefe büyük oranda ulaşmaktadır. Ancak 7000 civarında yanlış bir küme birleştirme işlemi yaptığından FP sayısında bir miktar artış olmuştur. Bu da Precision değerinin bir miktar düşmesine neden olmaktadır. Yine de önerdiğimiz yaklaşım oldukça yüksek bir Precision başarısına ulaşmaktadır.




Şekil 4.61. Occupancy veri setinde algoritmaların Precision üzerinden karşılaştırması

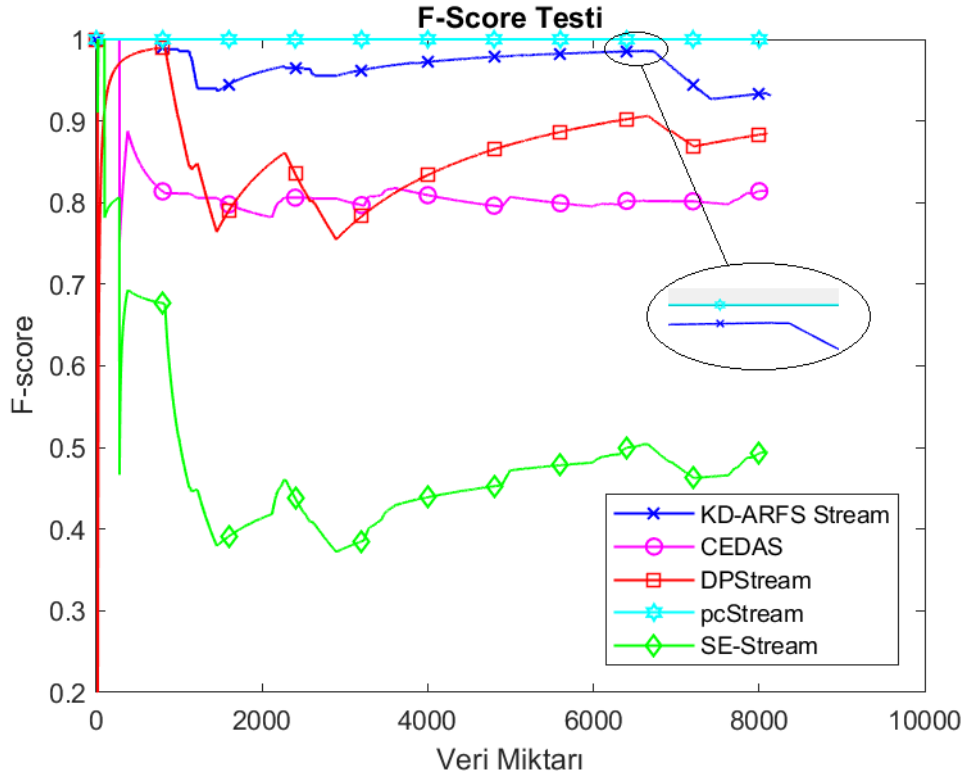
Recall testinde FN sayısı başarıyı etkilemektedir. Precision testine benzer şekilde önerdiğimiz yaklaşım 1700 ve 7000 civarında yanlış küme birleştirme yaptığından bir kümeye ait olan verileri yanlış kümeye atamaktadır. Çünkü bu aralıkta Occupancy veri

setine ika farklı kümeye ait olan veriler birbirlerine çok yaklaşmaktadır. Buna rağmen Şekil 4.62'de de görüldüğü gibi KD-ARFS Stream algoritması yüksek Recall değerine ulaşmaktadır.

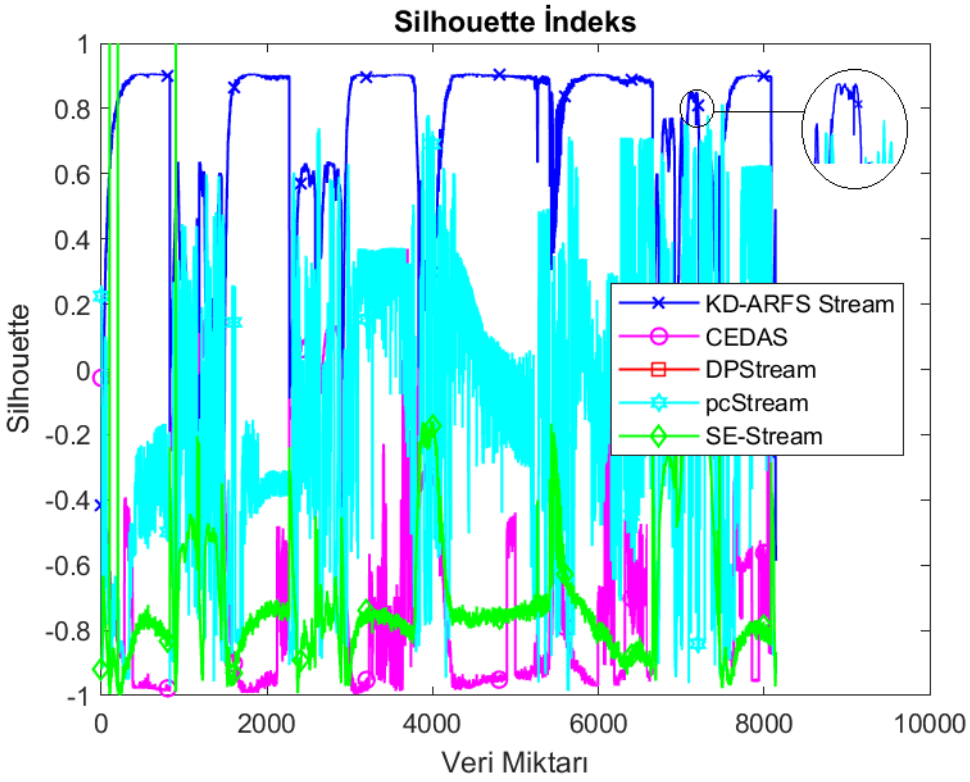



Şekil 4.62. Occupancy veri setinde algoritmaların Recall üzerinden karşılaştırması

F-Score, Precision ve Recall parametrelerinin harmonik ortalamasıdır ve Precision ve Recall değerlerinin ne kadar dengeli olduğunu gösterir. Şekil 4.63'e baktığımız zaman önerdiğimiz yaklaşımın dengeli olduğunu söylemek mümkündür. Düşüşün olduğu noktalar yanlış küme birleştirme işleminin yapıldığı noktalardır ve Precision ve Recall testlerine paralellik göstermektedir. Bununla beraber önerdiğimiz yaklaşım çok yüksek bir F-Score değerine ulaşmaktadır.




Şekil 4.63. Occupancy veri setinde algoritmaların F-Score üzerinden karşılaştırması

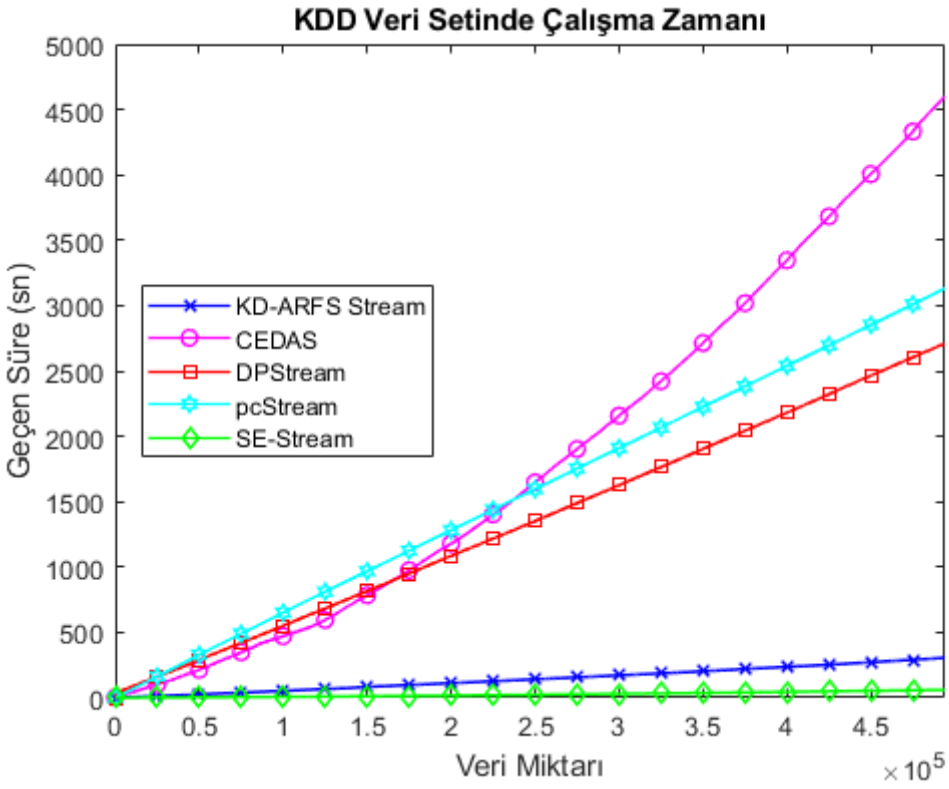



Şekil 4.64. Occupancy veri setinde algoritmaların Silhouette üzerinden karşılaştırması

Silhouette indeks kümelerin ne kadar başarılı bir şekilde ayrıştırıldığını tespit eden bir test yöntemidir. Şekil 4.64'e baktığımız zaman önerdiğimiz yaklaşımın en yüksek Silhouette indeks değerine sahip olduğunu görmek mümkündür. Çünkü önerdiğimiz yaklaşım kümeleri daha kompakt bir şekilde ayrıştırmaktadır.

### 4.3.4. Çalışma zamanı

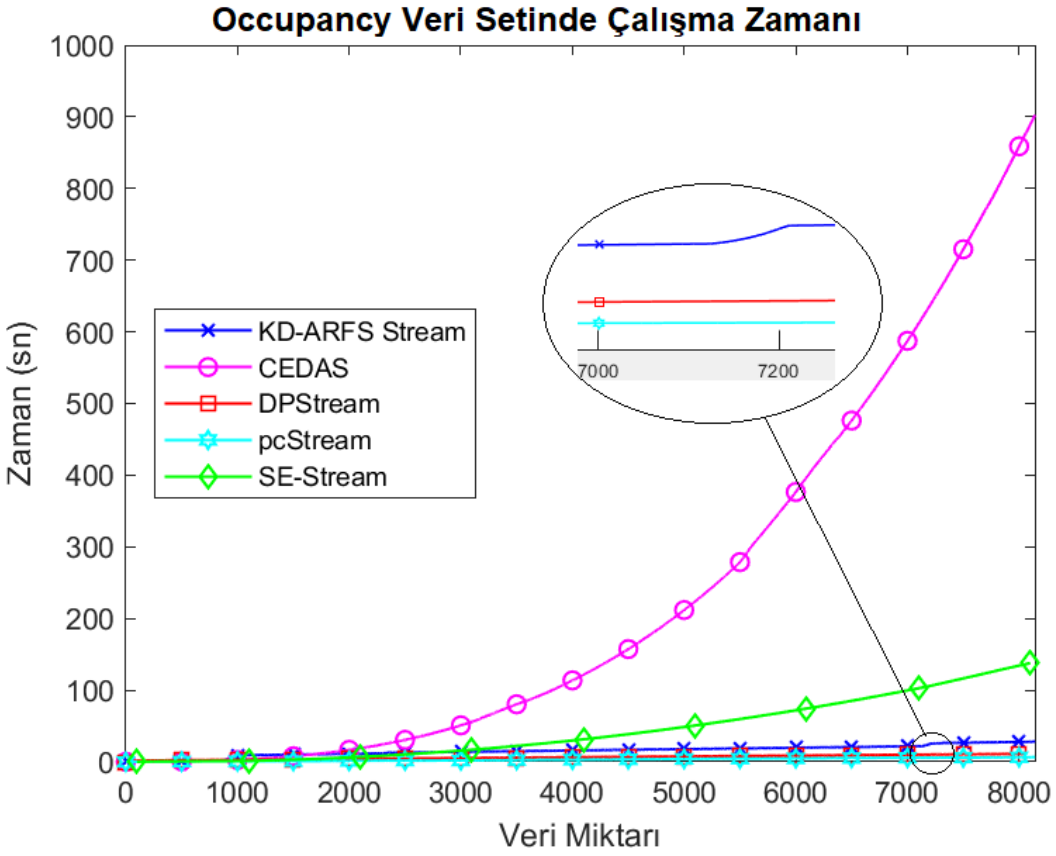
Şekil 4.65'te, KDD veri setini işlerken algoritmaların harcadığı zaman görülmektedir. KD-ARFS Stream, SE-Stream ve CEDAS algoritmalarından daha iyi performans göstermektedir. DPStream algoritmasının ise veriyi işlemek yerine sadece değişimleri tespit etmeye yönelik geliştirilmiş bir algoritma olduğu için daha yüksek performans göstermesi beklenen bir durumdur. Özellikle önerdiğimiz yaklaşımın çalışma zamanının doğrusal artması en önemli artısıdır.




Şekil 4.65. KDD veri setini işlerken algoritmaların harcadığı toplam süreler

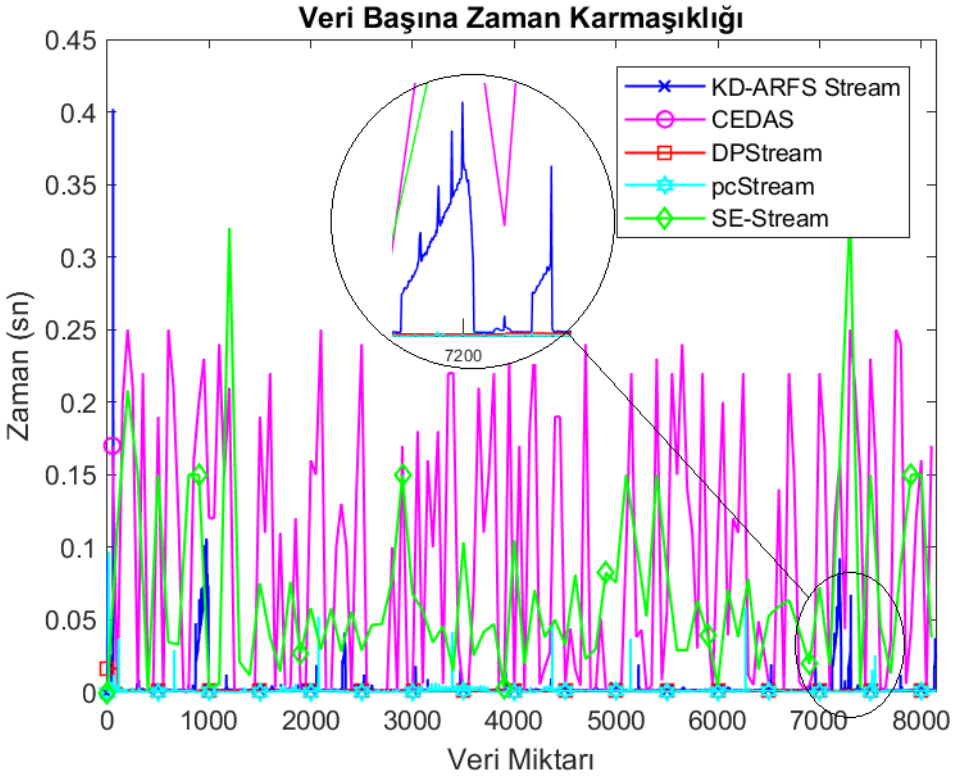
Şekil 4.66'da toplam harcanan zaman, zaman damgalı bir veri seti olan Occupancy veri seti üzerinde yapılmıştır. Verilerin gelişi arasında sabit olmayan bir zaman söz konusudur. Veriler arasındaki zaman farkı bazen 2-3 milisaniye iken, bazen 15-20 mili saniyedir. Elde

dilen sonuçlara bakıldığı zaman KD-ARFS Stream'in verileri çok hızlı bir şekilde işlediği görülmektedir.




Şekil 4.66. Occupancy veri setini işlerken algoritmaların harcadığı toplam süreler

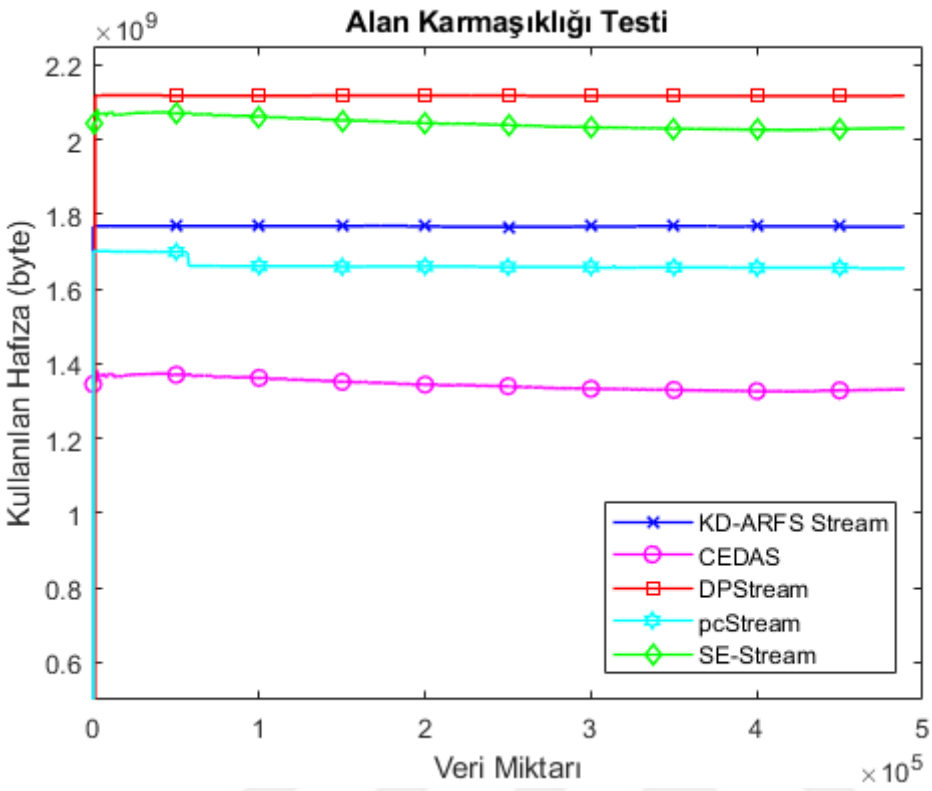
Şekil 4.67 her bir algoritmanın veri setini işlerken veri başına harcadığı zamanı göstermektedir. Çalışma zamanı grafiğine paralel bir sonuç görünmektedir. Önerdiğimiz yaklaşımın bazı aralıklarda performansında oluşan hafif dalgalanmaların nedeni bu aralıklarda hiçbir kümeye ait olmayan N tane veri olduğu için bunların bir küme oluşturup oluşturmadığını tespit etmek için NewClusterAppear fonksiyonu çalışmaktadır. Bu nedenle söz konusu aralıkta veri başına düşen zaman miktarı yüksek çıkmaktadır. Ancak söz konusu aralıklarda küme oluşturulduktan sonra veri başına harcanan süre normal değerlere dönmektedir.




Şekil 4.67. Occupancy veri setini işlerken algoritmaların veri başına harcadığı süreler

### 4.3.5. Alan karmaşıklığı

Algoritmaların alan karmaşıklıklarını karşılaştırmak için algoritmaların kullandıkları hafıza miktarları baz alınmıştır. Bu amaçla algoritmaların MATLAB çalışırken kullandıkları hafıza miktarları MATLAB'ın memory fonksiyonu kullanılarak tespit edilmiştir. MATLAB boşta iken yaklaşık 1.2 GB hafıza kullanmaktadır. Toplam kullanılan hafıza açısından karşılaştırma yapıldığında Şekil 4.68'de görülen sonuçlar elde edilmiştir. Şekilde de görüldüğü gibi önerdiğimiz yaklaşım yaklaşık olarak ortalama 1.75 GB hafıza kullanmaktadır. Bu değer diğer algoritmalar için şu şekildedir: CEDAS 1.375 GB, pcStream 1.7 GB, SE-Stream 2.05 GB ve DPStream 2.1 GB'tır. Şekilde de görüldüğü gibi önerdiğimiz yaklaşım makul bir hafıza miktarı kullanmaktadır.

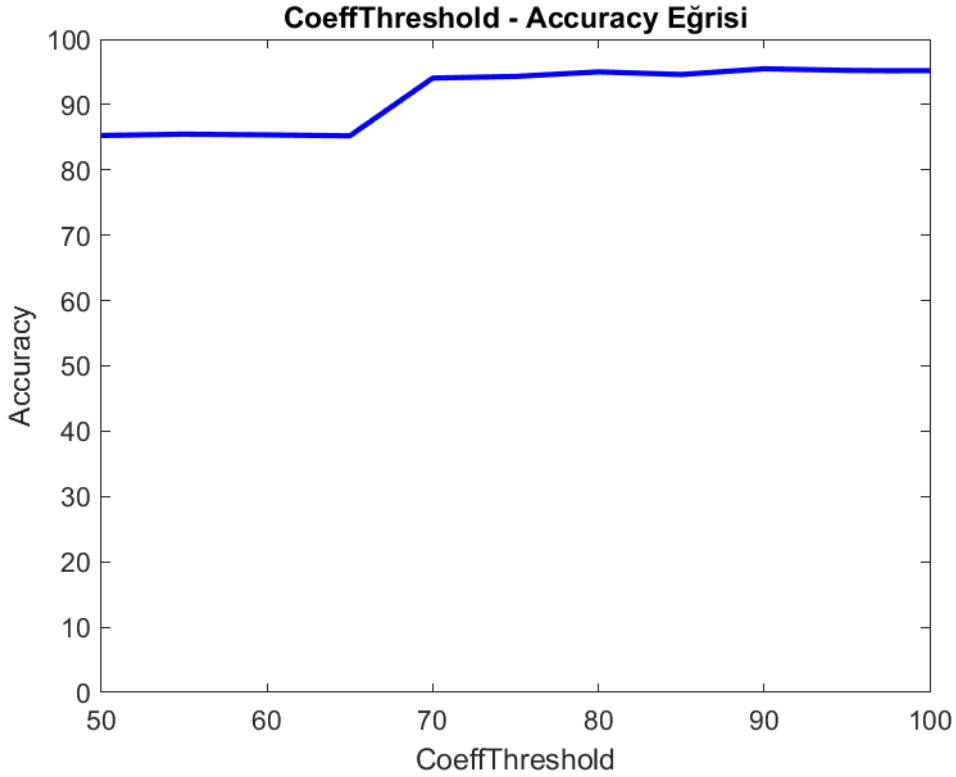



Şekil 4.68. KDD veri setini işlerken algoritmaların kullandıkları hafıza miktarları

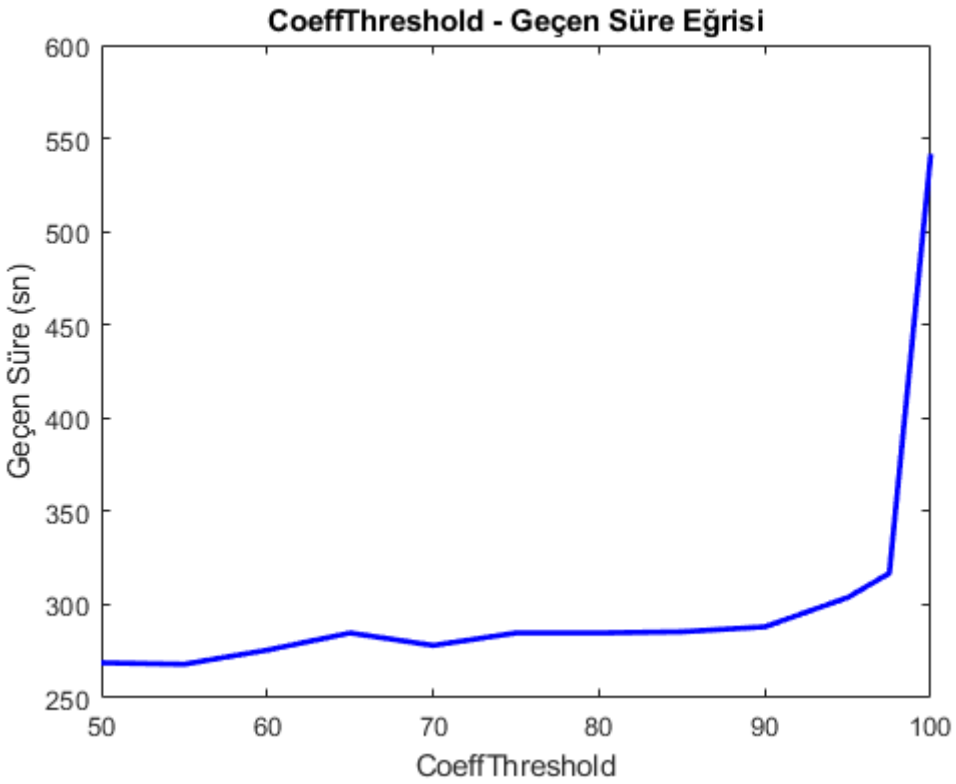
### 4.3.6. Standart sapma tabanlı öznitelik seçimi

KD-ARFS Stream algoritması öznitelik seçerken niteliklerin standart sapmasını göz önünde tutmaktadır. Amaç belirleyici nitelikleri kullanmak ve gereksiz nitelikleri eleyerek performansı arttırmaktır. Şekil 4.69'da da görüldüğü gibi KDD veri seti için CoeffThreshold değerinin 70'in üzerinde seçilmesi yüksek kümeleme başarısının elde edilmesi için yeterlidir.

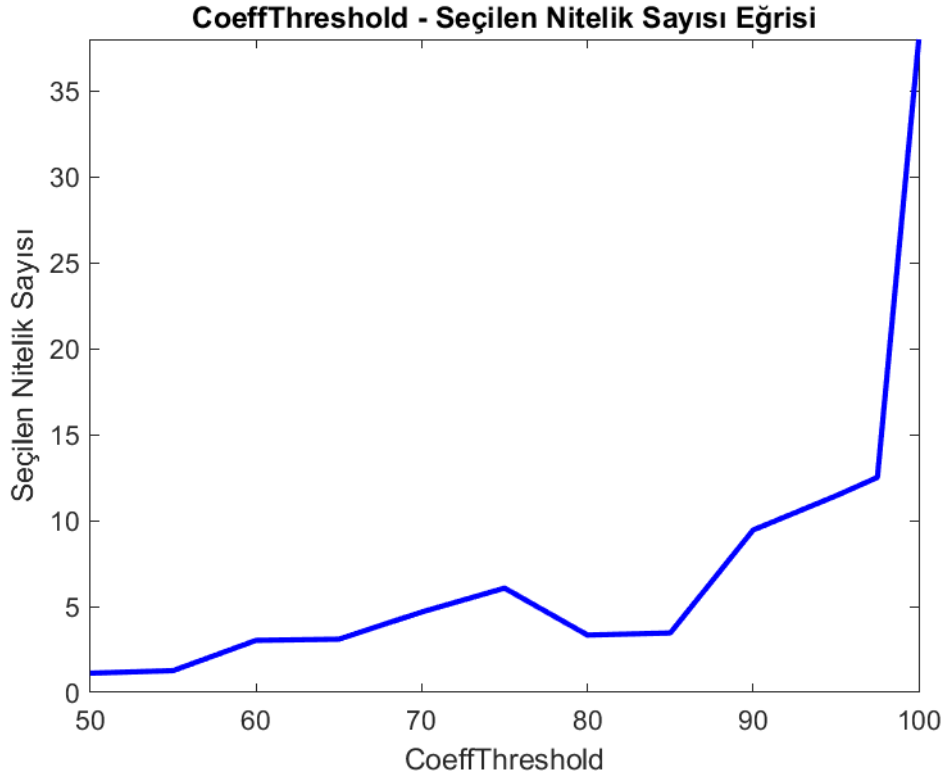
Nitelik sayısının fazla olması durumunda performans düşmektedir. Ancak gereksiz niteliklerin elenmesi durumunda performans düşüşü engellenmektedir. Şekil 4.70'te de görüldüğü gibi CoeffThreshold değerinin 98 ve altı seçilmesi durumunda performansta %40'ları aşan oranlarda artış olmaktadır. Çünkü veri setindeki tüm nitelikler ayırt edici özelliğe sahip değildir. Şekil 4.71'de de görüldüğü gibi belirleyici nitelik sayısı toplam 38 niteliğin 10-15 tanesidir.




Şekil 4.69. CoeffThreshold - Accuracy eğrisi




Şekil 4.70. CoeffThreshold - geçen zaman eğrisi




Şekil 4.71. CoeffThreshold - nitelik sayısı eğrisi

Çizelge 4.9. Algoritma başarılarının tek tabloda karşılaştırması

| | | Purity (%) | Accuracy (%) | Precision (%) | Recall (%) | F-Score (%) | Silhouette İndeks |
|---|---|---|---|---|---|---|---|
| KD-ARFS Stream | KDD | **99,11** | **97,68** | **94,79** | **99,28** | **98,83** | **0,92** |
| | Fisher Iris | **99,03** | **99,03** | **97,66** | **100** | **98,76** | 0,34 |
| | Breast Cancer | 94,23 | **94,23** | **95,75** | 94,78 | **95,20** | **0,32** |
| | MrData | 92,25 | **98,45** | **95,97** | **100** | **97,93** | **0,55** |
| | ExclaStar | 99,87 | 99,87 | 99,75 | **100** | 99,87 | 0,54 |
| | IdealData | **100** | **100** | **100** | **100** | **100** | 0,50 |
| | Occupancy | 95,46 | **94,50** | 97,48 | 95,57 | 96,42 | **0,71** |

Çizelge 4.9. (Devam) Algoritma başarılarının tek tabloda karşılaştırması

| | | Purity (%) | Accuracy (%) | Precision (%) | Recall (%) | F-Score (%) | Silhouette İndeks |
|---|---|---|---|---|---|---|---|
| **SE-Stream** | KDD | 97,21 | 97,25 | 93,95 | 97,99 | 97,15 | -0,15 |
| | Fisher Iris | 98,21 | 98,21 | 96,00 | **100** | 97,82 | **0,55** |
| | Breast Cancer | 95,19 | 58,39 | 59,47 | 95,97 | 73,14 | -0,03 |
| | MrData | 92,22 | 43,87 | 32,67 | 30,46 | 30,78 | -0,23 |
| | ExclaStar | **100** | **100** | **100** | **100** | **100** | **0,57** |
| | IdealData | 91,52 | 87,22 | 72,05 | **100** | 83,10 | 0,53 |
| | Occupancy | 99,04 | 89,77 | 36,48 | 73,45 | 48,20 | -0,69 |
| **pcStream** | KDD | 76,75 | 33,35 | 33,62 | 34,64 | 33,75 | 0,27 |
| | Fisher Iris | 95,27 | 88,12 | 82,96 | **100** | 89,63 | **0,57** |
| | Breast Cancer | 82,16 | 60,17 | 62,08 | 85,31 | 71,43 | 0,12 |
| | MrData | 69,82 | 53,71 | 39,16 | 43,19 | 39,80 | 0,26 |
| | ExclaStar | 99,79 | 99,79 | 99,60 | **100** | 99,80 | **0,57** |
| | IdealData | 88,37 | 86,27 | 71,30 | 96,52 | 81,20 | 0,12 |
| | Occupancy | 71,97 | 41,26 | **100** | **100** | **100** | -0,08 |
| **CEDAS** | KDD | 93,92 | 80,67 | 85,00 | 93,93 | 89,05 | -0,04 |
| | Fisher Iris | 93,38 | 85,36 | 87,37 | 93,33 | 88,86 | 0,28 |
| | Breast Cancer | 96,85 | 57,86 | 59,41 | 96,55 | 73,15 | -0,23 |
| | MrData | 89,68 | 39,37 | 39,33 | 99,45 | 55,73 | -0,25 |
| | ExclaStar | 92,13 | 91,81 | 91,55 | 95,42 | 93,17 | 0,01 |
| | IdealData | 88,93 | 75,64 | 61,86 | 91,09 | 72,22 | 0,53 |
| | Occupancy | 84,63 | 48,36 | 56,28 | 82,04 | 72,04 | -0,55 |

Çizelge 4.9. (Devam) Algoritma başarılarının tek tabloda karşılaştırması

| | | Purity (%) | Accuracy (%) | Precision (%) | Recall (%) | F-Score (%) | Silhouette İndeks |
|---|---|---|---|---|---|---|---|
| **DPStream** | KDD | **90,46** | 33,76 | 35,60 | 86,29 | 46,92 | 0,50 |
| | Fisher Iris | 97,67 | 72,04 | 70,69 | **100** | 80,37 | 0,12 |
| | Breast Cancer | **100** | 59,13 | 59,31 | **100** | 74,12 | - |
| | MrData | **92,49** | 39,37 | 39,37 | **100** | 55,87 | - |
| | ExclaStar | 83,39 | 66,62 | 54,96 | 98,81 | 68,37 | 0,14 |
| | IdealData | 98,21 | 77,05 | 59,98 | **100** | 73,63 | **0,68** |
| | Occupancy | **100** | 76,12 | 76,32 | **100** | 86,28 | - |

### 4.3.7. Sonuçların değerlendirilmesi ve tartışma

Önerdiğimiz yöntemin başarısını değerlendirmek adına Purity, Accuracy, Precision, Recall, F-Score ve Silhouette İndeks testleri gerçekleştirilmiş ve Çizelge 4.9'da verilen sonuçlar elde edilmiştir. Genel olarak bakıldığı zaman KD-ARFS Stream'in diğer algoritmalardan daha iyi sonuç verdiği görülmektedir.

7 veri seti üzerinde yapılan toplam 42 adet testin 28'inde KD-ARFS Stream daha başarılı sonuç vermiştir. Özellikle önerdiğimiz yöntemin kümelere aktif veya pasif olma imkânı vermesi nedeniyle Accuracy, Precision, Recall ve F-Score testlerinde belirgin bir üstünlük kurmaktadır. Çünkü aktif-pasif olma imkânı ile bir küme belirli bir süre pasif olabilmekte ve yeterli miktar veri geldiğinde tekrar aktif olabilmektedir. Özetle önerdiğimiz yöntemin bir hafızası bulunmaktadır.

Çizelge 3.2'de de görüldüğü gibi CheckSplit ve NewClusterAppear fonksiyonlarının karmaşıklığı yüksektir. Çünkü bu iki fonksiyonda veriler bir ağaca yerleştirilmekte ve üzerinde bir rangesearch işlemi yapılmaktadır. Bu nedenle çalışma zamanları yüksektir. Ancak performansı artırmak adına bu iki fonksiyonun çalışması Şekil 3.19 ve 3.21'da görüldüğü gibi belirli şartlara bağlıdır. Dolayısıyla bu iki fonksiyon her adımda değil

belirtilen şartların oluşması durumunda çalışmaktadır. Şekil 4.65 ve 4.66'da da görüldüğü gibi KD-ARFS Stream çok hızlı sonuç vermektedir.

Kullanılan hafıza bakımından algoritmalar karşılaştırıldığında da Şekil 4.68'de görülen sonuçlar elde edilmektedir. Önerdiğimiz yaklaşım diğer algoritmalarla kıyaslandığında ortalama civarında hafıza kullanmaktadır. Bu da kullanılan kaynak açısından da önerdiğimiz yaklaşımın makul olduğunu göstermektedir.

KD-ARFS Stream algoritması zaman tabanlı bir özetleme yaklaşımına sahiptir. Son 1 sn, 1 saat veya son 1 günlük veriler ihtiyaca göre verinin özeti olarak alınabilmektedir. Bu zaman periyodunu kullanıcı belirlemektedir. Ancak Şekil 4.58'te de görüldüğü gibi, eğer belirlenen zaman periyodunda gelen veri miktarı algoritmanın işleyebileceğinden fazla ise bu durumda algoritmanın işleyebileceği en son TN (kullanıcının tanımladığı miktar) tanesi özet olarak alınmaktadır. Bu yaklaşım algoritmanın ön görülemeyen durumlara hazırlıklı olmasını sağlamaktadır.

Önerdiğimiz yaklaşımın sapan verilere karşı direncini ölçmek adına DPStream algoritmasında kullanılan MrData veri seti kullanılmıştır. MrData veri setindeki 42470 adet verinin %10'u sapan verilerden oluşmaktadır. Çizelge 4.9'da da görüldüğü gibi KD-ARFS Stream %98.74'lük bir doğruluk (Accuracy) ile sonuç üretmektedir. Dolayısıyla önerdiğimiz yöntemin yaklaşık %98'lik başarı oranıyla sapan verileri tespit etmektedir diyebiliriz. Sonuç olarak KD-ARFS Stream sapan verilere karşı dirençli bir algoritmadır.

KD-ARFS Stream algoritmasında Çizelge 4.4'te de görüldüğü gibi 7 parametrenin tanımlanması gerekmektedir. Çizelge 4.5, 4.6, 4.7 ve 4.8'de de görüldüğü gibi CEDAS'ta 3, pcStream'de 5, DPStream ve SE-Stream algoritmalarında 8 parametrenin tanımlanmasına ihtiyaç duyulduğu göz önüne alındığında önerdiğimiz yöntemdeki parametre sayısı makul olarak kabul edilebilir.

KD-ARFS Stream algoritmasının en önemli üstünlüklerinden bir diğeri ise çok boyutlu verileri performansı düşürmeden işleyebilmesidir. Şekil 4.69, 4.70 ve 4.71'de de görüldüğü gibi öznitelik seçimi ile belirleyici nitelikler seçilmekte ve kümeleme başarısı düşürülmeden performans arttırılmaktadır.

Önerdiğimiz yaklaşımın avantajlarını şu şekilde sıralayabiliriz:

• Tamamen çevrimiçi çalışan bir algoritma olması

• Akan verinin zamanla yapısal değişime uğramasına tam uyum sağlayabilmesi

• Uyarlanabilir bir küme yarıçapına sahip olması

• Hibrit bir veri özetleme yaklaşımına sahip olması

• Bir kümeleme hafızasına sahip olması

• Çok boyutlu verilerde bile yüksek performansa sahip olması

• Yüksek kümeleme başarısına sahip olması

• Sapan verilere karşı dirençli bir yapıya sahip olması

• Makul sayıda değişken kullanması

Önerdiğimiz yaklaşımın dezavantajlarını ise kısaca şu şekilde sıralayabiliriz:

• Bu alandaki tüm çalışmalar için geçerli bir problem olan en doğru parametreleri belirleyememe

• Bu alandaki çoğu çalışmada da karşılaşılan dairesel olmayan kümeleri tespit edememe

Sonuç olarak önerdiğimiz KD-ARFS Stream algoritması pek çok açıdan literatüre katkı yapan yenilikler getiren çok başarılı ve oldukça hızlı çalışan bir algoritmadır.

# 5. SONUÇ

Akan veri kümeleme son zamanların popüler konularından biridir. Çünkü veri miktarının artması ve sürekli bir veri akışının söz konusu olduğu günümüzde gerçek zamanlı uygulamalara olan talep artmıştır. Ayrıca klasik veri madenciliği yöntemlerinin yetersiz kalmasından dolayı artık gerçek zamanlı ve makine öğrenmesi temelli çalışmalar önem kazanmaktadır. Bu nedenle bu alanda yapılan çalışmaların sayısı da gün geçtikçe artmaktadır.

Bu tez çalışmasında K-Boyutlu ağaç, uyarlanabilir yarıçap ve öznitelik seçme tabanlı bir akan veri kümeleme yöntemi (KD-ARFS Stream) önerilmiştir. Tamamen çevrimiçi çalışması, zaman tabanlı özetleme yapması, evrimsel bir algoritma olması, yarıçap parametresinin uyarlanabilir olması ve öznitelik seçerek çok boyutlu verileri hızlı bir şekilde işleyebilmesi KD-ARFS Stream algoritmasının öne çıkan özellikleri olarak göze çarpmaktadır. Ayrıca zamanla sahip olduğu verileri kaybeden kümeleri silmek yerine pasif duruma getirmesi ve daha sonra bu tür kümelerin yeni veri almak suretiyle tekrar yeterli veriye sahip olması durumunda yeniden aktive etmesi KD-ARFS Stream'in akan veri kümeleme yaklaşımının değişken yapısını destekleyen önemli bir diğer özelliği olarak ön plana çıkmaktadır.

Önerdiğimiz yaklaşım bu alanda önerilmiş olan algoritmalardan dördü olan SE-Stream, pcStream, DPStream ve CEDAS algoritmaları ile hem kümeleme başarısı, hem de çalışma zamanı açısından karşılaştırılmıştır. Veri seti olarak 4'ü gerçek veri seti olmak üzere 7 adet veri seti üzerinde test edilmiştir. 7 veri seti üzerinde yapılan toplam 42 testin 28'inde önerdiğimiz KD-ARFS Stream algoritması en yüksek kümeleme başarısı elde etmiştir. En yüksek kümeleme başarısının elde edilmediği testlerde de önerdiğimiz yaklaşım en yüksek kümeleme başarısı elde eden algoritmalardan biridir. Yani önerdiğimiz yaklaşım farklı farklı veri setleri üzerinde farklı testlerde istikrarlı ve tutarlı bir kümeleme başarısı yakalamaktadır. Oysa diğer algoritmalar bir veri setinde çok yüksek bir kümeleme başarısı elde ederken bir başkasında bu başarı çok fazla düşmektedir.

Akan veri kümeleme uygulamalarında çok önemli noktalardan biri de çalışma zamanıdır. Çünkü veri setinin zamana bağlı olarak sürekli değişmesi nedeniyle zamana karşı bir yarış

söz konusudur. Veri akarken bir taraftan da sonuç üretmek gerektiğinden, veriyi hızlı bir şekilde işlemek gerekir. Yapılan deneysel çalışma sonuçlarına göre önerdiğimiz yaklaşım çalışma zamanı açısından da hızlı bir şekilde sonuç üretebilmektedir. 494 020 veriden oluşan KDD veri setini karşılaştırma yapılan diğer algoritmalar binlerle ifade edilen sürelerde işlerken, önerdiğimiz KD-ARFS Stream algoritması 300 saniyenin altında işleyebilmektedir. Önerdiğimiz yaklaşımın bu kadar hızlı bir şekilde verileri işleyebilmesinin en önemli nedeni kullandığı öznitelik seçme yaklaşımıdır. Standart sapma tabanlı öznitelik seçme yaklaşımı ile KD-ARFS Stream, verileri hızlı bir şekilde işlerken yüksek kümeleme başarısı da elde edebilmektedir.

Kullanılan hafıza açısından bakıldığı zaman da önerdiğimiz yaklaşım diğer algoritmalara kıyasla oldukça az hafıza kullanmaktadır. Önerdiğimiz yaklaşım yaklaşın 550 MB hafıza kullanırken, sırasıyla CEDAS 175 MB, pcStream 500 MB, SE-Stream 850 MB ve DPStream 900 MB hafıza kullanmaktadır. Sonuç olarak önerdiğimiz yaklaşım kullanılan kaynak açısından da oldukça makul bir algoritmadır.

Önerdiğimiz yaklaşımın bir diğer önemli özelliği de yoğunluk tabanlı bir yaklaşım kullanmasından dolayı sapan verilere karşı dirençli bir yapısının olmasıdır. Çünkü akan veri uygulamalarında sapan verileri tespit etmek zordur. Ayrıca sapan veriler kümeleme başarısını da oldukça düşürmektedir. Dolayısıyla sapan verilere karşı dirençli olan yaklaşımlar çok değerlidir. Deneysel çalışmalar sonucunda önerdiğimiz yaklaşımın çok sayıda sapan veri içeren veri setinde yüksek kümeleme başarısı elde etmesi oldukça önemlidir.

Gelecekte önerilen yaklaşımın daha az parametre kullanarak sonuç üretebilmesine ve yuvarlak olmayan kümeleri tespit edebilmesine yönelik çalışmaların yapılması planlanmaktadır.

# KAYNAKLAR


Abbas, H. M. ve Fahmy, M. M. (1994). Neural networks for maximum likelihood clustering. *Signal Process., 36*(1), 111-126.

Ackermann, M. R., Martens, M., Raupach, C., Swierkot, K., Lammersen, C. ve Sohler, C. (2012). StreamKM++: A clustering algorithm for data streams. *J. Exp. Algorithmics, 17*, 2.1-2.30.

Aggarwal, C. C. (2007). *Data Streams: Models and Algorithms* (1 ed.): Springer US.

Aggarwal, C. C. (2010). Data Streams: An Overview and Scientific Applications. In M. M. Gaber (Ed.), *Scientific Data Mining and Knowledge Discovery: Principles and Foundations* (377-397). Berlin, Heidelberg: Springer Berlin Heidelberg.

Aggarwal, C. C., Han, J., Wang, J. ve Yu, P. S. (2003). *A framework for clustering evolving data streams*. Proceedings of the 29th international conference on Very large data bases - 29, Berlin, Germany.

Agrawal, R., Gehrke, J., Gunopulos, D. ve Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Rec., 27*(2), 94-105.

Agrawal, R., Imielinski, T. ve Swami, A. (1993). Mining association rules between sets of items in large databases. *SIGMOD Rec.,* 22(2), 207-216. doi:10.1145/170036.170072

Ahmed, I., Ahmed, I. ve Shahzad, W. (2015). Scaling up for high dimensional and high speed data streams: HSDStream. *CoRR, abs/1510.03375*.

Ahmed, M. (2019). Buffer-based Online Clustering for Evolving Data Stream. *Information Sciences, 489,* 113-135.

Amini, A. ve Wah, T. Y. (2013). LeaDen-Stream: A Leader Density-Based Clustering Algorithm over Evolving Data Stream. *Journal of Computer and Communications, 1*, 26-31.

Amini, A., Wah, T. Y. ve Saboohi, H. (2014). On Density-Based Data Streams Clustering Algorithms: A Survey. *Journal of Computer Science and Technology, 29*(1), 116-141.

Ankerst, M., Breunig, M. M., Kriegel, H.-P. ve Sander, J. (1999). OPTICS: ordering points to identify the clustering structure. *SIGMOD Rec., 28*(2), 49-60.

Ankleshwaria, T. B. ve Dhobi, J. S. (2014). Mining Data Streams: A Survey. *International Journal of Advance Research in Computer Science and Management Studies, 2*(2), 379-386.

Antonellis, P., Makris, C. ve Tsirakis, N. (2009). Algorithms for clustering clickstream data. *Information Processing Letters, 109*(8), 381-385.

Barddal, J. P., Gomes, H. M. ve Enembreck, F. (2015). *SNCStream: a social network-based data stream clustering algorithm*. Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM, 18*(9), 509-517.

Beringer, J. ve Hüllermeier, E. (2006). Online clustering of parallel data streams. *Data & Knowledge Engineering, 58*(2), 180-204.

Bifet, A. ve Kirkby, R. (2009). Data stream mining a practical approach. *Big Data Now: 2014 Edition*. (2015). (2014 ed.): O'Reilly Media.

Brun, M., Sima, C., Hua, J., Lowey, J., Carroll, B., Suh, E. ve Dougherty, E. R. (2007). Model-based evaluation of clustering validation measures. *Pattern Recognition, 40*(3), 807-824.

Caliński, T. ve Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics, 3*(1), 1-27.

Cao, F., Estert, M., Qian, W. ve Zhou, A., (2006).Density-Based Clustering over an Evolving Data Stream with Noise. *Proceedings of the 2006 SIAM International Conference on Data Mining* (328-339).

Carnein, M. ve Trautmann, H. (2018). evoStream – Evolutionary Stream Clustering Utilizing Idle Times. *Big Data Research, 14*, 101-111.

Carpineto, C. ve Romano, G. (1996). A Lattice Conceptual Clustering System and Its Application to Browsing Retrieval. *Machine Learning, 24*(2), 95-122.

Chairukwattana, R., Kangkachit, T., Rakthanmanon, T. ve Waiyamai, K. (2013, 4-6 Sept. 2013). *Efficient evolution-based clustering of high dimensional data streams with dimension projection.* 2013 International Computer Science and Engineering Conference (ICSEC).

Chaovalit, P. ve Gangopadhyay, A. (2009). *A method for clustering transient data streams*. Proceedings of the 2009 ACM symposium on Applied Computing, Honolulu, Hawaii.

Charu, C. A., Jiawei, H., Jianyong, W. ve Philip, S. Y. (2004). A framework for projected clustering of high dimensional data streams *Proceedings of the Thirtieth international conference on Very large data bases - 30* (852-863). Toronto, Canada: VLDB Endowment.

Choromanski, K., Kumar, S. ve Liu, X. (2015). Fast Online Clustering with Randomized Skeleton Sets. *CoRR, abs/1506.03425*.

Coleman, G. B. ve Andrews, H. C. (1979). Image segmentation by clustering. *Proceedings of the IEEE, 67*(5), 773-785.

Cover, T. ve Hart, P. (2006). *Nearest neighbor pattern classification*. IEEE Trans. Inf. Theor., 13(1), 21-27.

Csernel, B., Clerot, F. ve Hébrail, G. (2006). *StreamSamp: DataStream Clustering Over Tilted Windows Through Sampling.* ECML PKDD 2006 Workshop on Knowledge Discovery from Data Streams.

Dang, X. H., Lee, V., Ng, W. K., Ciptadi, A. ve Ong, K. L. (2009). *An EM-Based Algorithm for Clustering Data Streams in Sliding Windows*, Berlin, Heidelberg.

Datar, M., Gionis, A., Indyk, P. ve Motwani, R. (2002). *Maintaining stream statistics over sliding windows: (extended abstract)*. Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, San Francisco, California.

Davies, D. L. ve Bouldin, D. W. (1979). A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-1*(2), 224-227.

Davis, K. (2012). *Ethics of Big Data: Balancing Risk and Innovation*: O'Reilly Media, Inc.

Dempster, A., Laird, N. M. ve Rubin, D. B. (1976). *Maximum Likelihood from Incomplete Data via the EM Algorithm*. Royal Statistical Society at a meeting organized by the Research Section.

Diaz-Rozo, J., Bielza, C. ve Larrañaga, P. (2018). Clustering of Data Streams with Dynamic Gaussian Mixture Models. An IoT Application in Industrial Processes. *IEEE Internet of Things Journal*, 1-1.

Dong, X. L. ve Srivastava, D. (2013, 8-12 April 2013). *Big data integration.* 2013 IEEE 29th International Conference on Data Engineering (ICDE).

Dunn, J. C. (1974). Well-Separated Clusters and Optimal Fuzzy Partitions. *Journal of Cybernetics, 4*(1), 95-104.

Ester, M., Kriegel, H.-P., Sander, J. ve Xu, X. (1996). *A density-based algorithm for discovering clusters in large spatial databases with noise*. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, Portland, Oregon.

Frigui, H. ve Krishnapuram, R. (1999). A Robust Competitive Clustering Algorithm With Applications in Computer Vision. *IEEE Trans. Pattern Anal. Mach. Intell., 21*(5), 450-465.

Gao, J., Li, J., Zhang, Z. ve Tan, P.-N. (2005). *An Incremental Data Stream Clustering Algorithm Based on Dense Units Detection*, Berlin, Heidelberg.

Gobble, M. M. (2013). Big Data: The Next Big Thing in Innovation. *Research-Technology Management, 56*(1), 64-67.

Gravina, R., Alinia, P., Ghasemzadeh, H. ve Fortino, G. (2017). Multi-sensor fusion in body sensor networks: State-of-the-art and research challenges. *Information Fusion, 35*, 68-80.

Hahsler, M. ve Bolaños, M. (2016). Clustering Data Streams Based on Shared Density between Micro-Clusters. *IEEE Transactions on Knowledge and Data Engineering, 28*(6), 1449-1461.

Halkidi, M., Batistakis, Y. ve Vazirgiannis, M. (2001). On Clustering Validation Techniques. *Journal of Intelligent Information Systems, 17*(2), 107-145.

Han, J., Pei, J. ve Yin, Y. (2000). Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2), 1-12.

Hartigan, J. A. ve Wong, M. A. (1979). Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics), 28*(1), 100-108.

Hawwash, B. (2013). *Stream-dashboard : a big data stream clustering framework with applications to social media streams.* (PhD), University of Louisville.

Hendricks, D. (2017). Using real-time cluster configurations of streaming asynchronous features as online state descriptors in financial markets. *Pattern Recognition Letters, 97*, 21-28.

Hinneburg, A. ve Keim, D. A. (1998). *An efficient approach to clustering in large multimedia databases with noise*. Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, New York, NY.

Hitzler, P. ve Janowicz, K. (2013). Linked Data, Big Data, and the 4th Paradigm. *Semant. web, 4*(3), 233-235.

Holzinger, A., Stocker, C., Ofner, B., Prohaska, G., Brabenetz, A. ve Hofmann-Wellenhof, R. (2013). *Combining HCI, Natural Language Processing, and Knowledge Discovery - Potential of IBM Content Analytics as an Assistive Technology in the Biomedical Field*, Berlin, Heidelberg.

Hubert, L. ve Arabie, P. (1985). Comparing partitions. *Journal of Classification, 2*(1), 193-218.

Hyde, R. ve Angelov, P. (2015, 24-26 June 2015). *A new online clustering approach for data in arbitrary shaped clusters.* 2015 IEEE 2nd International Conference on Cybernetics (CYBCONF).

Hyde, R., Angelov, P. ve MacKenzie, A. R. (2017). Fully online clustering of evolving data streams into arbitrarily shaped clusters. *Information Sciences, 382-383*, 96-114.

IBM, Zikopoulos, P. ve Eaton, C. (2011). *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*: McGraw-Hill Osborne Media.

Ikonomovska, E., Loskovska, S. ve Gjorgjevik, D. (2007). *A survey of stream data mining*. Eighth International Conference with International Participation – ETAI 2007, Ohrid, Republic ofMacedonia.

İnternet: Kemp, S. *Digital 2019.* URL: http://www.webcitation.org/query?url=https%3A%2F%2Fwearesocial.com%2Fglobal-digital-report-2019&date=2019-05-13, Son Erişim Tarihi: 14.05.2019.

İnternet: Kreveld, M. v. ve Toll, W. v. *Computational Geometry - Lecture 7: Range searching and kd-trees*. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fwww.cs.uu.nl%2Fdocs%2Fvakken%2Fga%2Fslides%2Fslides5a.pdf&date=2019-05-13, Son Erişim Tarihi: 14.05.2019.

İnernet: Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C. ve Byers, A. H. *Big Data: The Next Frontier for Innovation*, Competition, and Productivity. URL:http://www.webcitation.org/query?url=https%3A%2F%2Fwww.mckinsey.com%2Fbusiness-functions%2Fdigital-mckinsey%2Four-insights%2Fbig-data-the-next-frontier-for-innovation&date=2019-05-13, Son Erişim Tarihi:14.05.2019.

İnternet: Merino, J. A. *Streaming data clustering in MOA using the leader algorithm*. (Master), Universitat Polit`ecnica de Catalunya. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fhdl.handle.net%2F2117%2F79235+&date=2019-05-13, Son Erişim Tarihi: 14.05.2019.

İnternet: Wahid, A., *Big Data and Machine Learning for Bussinesses*, URL: http://www.webcitation.org/query?url=https://www.slideshare.net/awahid/big-data-and-machine-learning-for-businesses&date=2019-06-13.

İnternet: Krzyk, K., Coding Deep Learning For Beginners-Types of Machine Learning, URL:http://www.webcitation.org/query?url=https://towardsdatascience.com/coding-deep-learning-for-beginners-types-of-machine-learning-b9e651e1ed9d&date=2019-06-13.

Jaccard, P. (1901). Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles, 37*, 241-272.

Jain, A. K. ve Dubes, R. C. (1988). *Algorithms for clustering data*: Prentice-Hall, Inc.

Jia, C., Tan, C. ve Yong, A. (2008, 25-26 Sept. 2008). *A Grid and Density-Based Clustering Algorithm for Processing Data Stream.* 2008 Second International Conference on Genetic and Evolutionary Computing.

Jiawei H., M. Kamber, J. Pei, (2011). *Data Mining: Concepts and Techniques: Concepts and Techniques*: ISBN 978-0-12-381479-1, Elsevier.

Jolliffe, I. (2005). Principal Component Analysis. *Encyclopedia of Statistics in Behavioral Science*.

Judd, D., McKinley, P. K. ve Jain, A. K. (1998). Large-Scale Parallel Data Clustering. *IEEE Trans. Pattern Anal. Mach. Intell., 20*(8), 871-876.

Hand, D. J., Smyth, P. ve Mannila, H. (2001). *Principles of data mining*: MIT Press.

Kaur, M. ve Garg, S. (2014). *Survey on Clustering Techniques in Data Mining for Software Engineering* (3).

Karypis, G., Han, E.-H. ve Kumar, V. (1999). Chameleon: Hierarchical Clustering Using Dynamic Modeling. *Computer, 32*(8), 68-75.

Kavitha, V. ve Punithavalli, M. (2010). Clustering Time Series Data Stream - A Literature Survey. *CoRR, abs/1005.4270*.

Keim, D. A. ve Heczko, M. (2001). *Wavelets and their Applications in Databases*. 17th International Conference on Data Engineering (ICDE'01), Heidelberg, Germany, 2001.

Keogh, E., Chu, S., Hart, D. ve Pazzani, M. (2001, 29 Nov.-2 DEC. 2001). *An online algorithm for segmenting time series.* Proceedings 2001 IEEE International Conference on Data Mining San Jose, CA, USA, USA.

Khalilian, M., Mustapha, N. ve Sulaiman, N. (2016). Data stream clustering by divide and conquer approach based on vector model. *Journal of Big Data, 3*(1), 1.

King, R. C., Villeneuve, E., White, R. J., Sherratt, R. S., Holderbaum, W. ve Harwin, W. S. (2017). Application of data fusion techniques and technologies for wearable health monitoring. *Medical Engineering & Physics, 42*, 1-12.

Kranen, P., Assent, I., Baldauf, C. ve Seidl, T. (2011). The ClusTree: indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems, 29*(2), 249-272.

Laohakiat, S., Phimoltares, S. ve Lursinsap, C. (2017). A clustering algorithm for stream data with LDA-based unsupervised localized dimension reduction. *Information Sciences, 381*, 104-123.

Lee, C.-Y. ve Antonsson, E. K. (2000). *Dynamic Partitional Clustering Using Evolution Strategies*. Proceedings of the Third Asia Pacific Conference on Simulated Evolution and Learning.

Leung, Y., Zhang, J.-S. ve Xu, Z.-B. (2000). Clustering by Scale-Space Filtering. *IEEE Trans. Pattern Anal. Mach. Intell., 22*(12), 1396-1410.

Li, Z. Q. (2014). A New Data Stream Clustering Approach about Intrusion Detection. *Advanced Materials Research, 926-930*, 2898-2901.

Liu, L. x., Huang, H., Guo, Y. f. ve Chen, F. c. (2009, 19-20 Dec. 2009). *rDenStream, A Clustering Algorithm over an Evolving Data Stream.* 2009 International Conference on Information Engineering and Computer Science.

Lloyd, S. (1982). *Least squares quantization in PCM*. IEEE Transactions on Information Theory, 28(2), 129-137.

Manzi, A., Dario, P. ve Cavallo, F. (2017). A Human Activity Recognition System Based on Dynamic Clustering of Skeleton Data. *Sensors (Basel, Switzerland), 17*(5), 1100.

Masmoudi, N., Azzag, H., Lebbah, M. ve Bertelle, C. (2014, July 30 2014-Aug. 1 2014). *Incremental clustering of data stream using real ants behavior*. 2014 Sixth World Congress on Nature and Biologically Inspired Computing (NaBIC 2014).

Masmoudi, N., Azzag, H., Lebbah, M., Bertelle, C. ve Jemaa, M. B. (2016). CL-AntInc Algorithm for Clustering Binary Data Streams Using the Ants Behavior. *Procedia Comput. Sci., 96*(C), 187-196.

Maulik, U. ve Bandyopadhyay, S. (2002). Performance evaluation of some clustering algorithms and validity indices. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*(12), 1650-1654.

Mayer-Schnberger, V. (2013). *Big Data: A Revolution That Will Transform How We Live, Work and Think. Viktor Mayer-Schnberger and Kenneth Cukier*: John Murray Publishers.

Meesuksabai, W., Kangkachit, T. ve Waiyamai, K. (2011). *HUE-Stream: Evolution-Based Clustering Technique for Heterogeneous Data Streams with Uncertainty*, Berlin, Heidelberg.

Mehra, L., Kumar Gupta, M. ve Bhatt, M. (2014). *An Effectual and Secure Approach for the Detection and Efficient Searching of Network Intrusion Detection System,* (NIDS) (108).

Mirsky, Y., Shapira, B., Rokach, L. ve Elovici, Y. (2015, 2015). *pcStream: A Stream Clustering Algorithm for Dynamically Detecting and Managing Temporal Contexts.* Advances in Knowledge Discovery and Data Mining, Cham.

Mitchell, T. M. (1997). *Machine Learning*: McGraw-Hill, Inc.

Mokhtari, B., Melkemi, K., Michelucci, D. ve Foufou, S. (2014). *Dynamic Clustering-Based Method for Shape Recognition and Retrieval*.

Mousavi, M. ve Abu Bakar, A. (2015). Improved density based algorithm for data stream clustering. *Jurnal Teknologi, 77*(18), 73-77.

Mousavi, M., Bakar, A. A. ve Vakilian, M. (2015). Data stream clustering algorithms: A review. *International Journal of Advances in Soft Computing and its Applications, 7*(Specialissue3), 1-15.

Ntoutsi, I., Zimek, A., Palpanas, T., Kröger, P. ve Kriegel, H.-P. (2012). *Density-based Projected Clustering over High Dimensional Data Streams.* SIAM International Conference on Data Mining.

O'Callaghan, L., Mishra, N., Meyerson, A., Guha, S. ve Motwani, R. (2002, 26 Fe.-1 March 2002). *Streaming-data algorithms for high-quality clustering.* Proceedings 1st International Conference on Data Engineering, San Jose, CA, USA, USA.

Ohlhorst, F. J. (2012). *Big Data Analytics: Turning Big Data into Big Money*: Wiley Publishing.

Omran, M. G., Salman, A. ve Engelbrecht, A. P. (2006). Dynamic clustering using particle swarm optimization with application in image segmentation. *Pattern Anal. Appl.*,

Popovici, R., Weiler, A. ve Grossniklaus, M. (2014). *On-line clustering for real-time topic detection in social media streaming data.* Second Workshop on Social News on the Web @ the 23rd International World Wide Web Conference, Seoul, Korea.

Putri, G. H., Read, M. N., Koprinska, I., Singh, D., Röhm, U., Ashhurst, T. M. ve King, N. J. C. (2019). ChronoClust: Density-based clustering and cluster tracking in high-dimensional time-series data. *Knowledge-Based Systems*.

Quinlan, J. R. (1986). *Induction of Decision Trees*. Mach. Learn., 1(1), 81-106.

Raftery, A. E. (1986). A Note on Bayesian Factors for Log-Linear Contingency Table Models with Vague Prior Information. *Journal of the Royal Statistical Society, Series B, 48*(B), 249-250.

Rand, W. M. (1971). Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association, 66*(336), 846-850.

Ray, S. ve Turi, R. H. (2000). *Determination of Number of Clusters in K-Means Clustering and Application in Colour Image Segmentation.*

Reddy, K. S. S. ve Bindu, C. S. (2018). StreamSW: A Density-based Approach for Clustering Data Streams over Sliding Windows. *Measurement*.

Reeve, A. (2013). Managing Data in Motion. Boston: Morgan Kaufmann.

Rodrigues, P. P., Gama, J. ve Pedroso, J. (2008). Hierarchical Clustering of Time-Series Data Streams. *IEEE Transactions on Knowledge and Data Engineering, 20*(5), 615-627.

Rosenberger, C. ve Chehdi, K. (2000, 2000). *Unsupervised clustering method with optimal estimation of the number of clusters: application to image segmentation.* Proceedings 15th International Conference on Pattern Recognition. ICPR-2000.

Sabit, H., Al-Anbuky, A. ve Gholam-Hosseini, H. (2009, 7-9 July 2009). *Distributed WSN Data Stream Mining Based on Fuzzy Clustering.* 2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing.

Sagiroglu, S. ve Sinanc, D. (2013, 20-24 May 2013). *Big data: A review*. 2013 International Conference on Collaboration Technologies and Systems (CTS).

Shannon, C. E. (2001). A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev., 5*(1), 3-55.

Shao, J., Tan, Y., Gao, L., Yang, Q., Plant, C. ve Assent, I. (2018). Synchronization-based clustering on evolving data stream. *Information Sciences*.

Shao, X., Zhang, M. ve Meng, J. (2018, 25-26 Jan. 2018). *Data Stream Clustering and Outlier Detection Algorithm Based on Shared Nearest Neighbor Density.* 2018 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS).

Sheikholeslami, G., Chatterjee, S. ve Zhang, A. (2000). WaveCluster: a wavelet-based clustering approach for spatial data in very large databases. *The VLDB Journal, 8*(3), 289-304.

Silva, A. d., Chiky, R. ve Hebrail, G. (2012). A clustering approach for sampling data streams in sensor networks. *Knowl. Inf. Syst., 32*(1), 1-23.

Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., Carvalho, A. C. P. L. F. d. ve Gama, J. (2013). Data stream clustering: A survey. *ACM Comput. Surv., 46*(1), 1-31.

Silva, J. d. A., Hruschka, E. R. ve Gama, J. (2017). An evolutionary algorithm for clustering data streams with a variable number of clusters. *Expert Syst. Appl., 67*(C), 228-238.

Strehl, A. ve Ghosh, J. (2003). Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res., 3*, 583-617.

Şenol, A. ve Karacan, H. (2018). A Survey on Data Stream Clustering Techniques. *European Journal of Science and Technology*(13), 17-30.

Tasnim, S., Caldas, J., Pissinou, N., Iyengar, S. S. ve Ding, Z. (2018, 5-8 March 2018). *Semantic-Aware Clustering-based Approach of Trajectory Data Stream Mining.* 2018 International Conference on Computing, Networking and Communications (ICNC).

Theodoridis, S. ve Koutroumbas, K. (2008). *Pattern Recognition, Fourth Edition*: Academic Press.

Tu, L. ve Chen, Y. (2009). Stream data clustering based on grid density and attraction. *ACM Trans. Knowl. Discov. Data, 3*(3), 1-27.

Udommanetanakit, K., Rakthanmanon, T. ve Waiyamai, K. (2007). *E-Stream: Evolution-Based Technique for Stream Clustering*, Berlin, Heidelberg.

Wallace, D. L. (1983). A Method for Comparing Two Hierarchical Clusterings: Comment. *Journal of the American Statistical Association, 78*(383), 569-576.

Wan, L., Ng, W. K., Dang, X. H., Yu, P. S. ve Zhang, K. (2009). Density-based clustering of data streams at multiple resolutions. *ACM Trans. Knowl. Discov. Data, 3*(3), 1-28.

Wang, W., Yang, J. ve Muntz, R. R. (1997). *STING: A Statistical Information Grid Approach to Spatial Data Mining*. Proceedings of the 23rd International Conference on Very Large Data Bases.

Weiler, A., Grossniklaus, M. ve Scholl, M. H. (2016). Situation monitoring of urban areas using social media data streams. *Information Systems, 57*, 129-141.

Xu, J., Wang, G., Li, T., Deng, W. ve Gou, G. (2017). Fat node leading tree for data stream clustering with density peaks. *Knowledge-Based Systems, 120*, 99-117.

Yeh, M. Y., Dai, B. R. ve Chen, M. S. (2007). Clustering over Multiple Evolving Streams by Events and Correlations. *IEEE Transactions on Knowledge and Data Engineering, 19*(10), 1349-1362.

Yin, C., Xia, L. ve Wang, J. (2017, 2017). *Application of an Improved Data Stream Clustering Algorithm in Intrusion Detection System.* Advanced Multimedia and Ubiquitous Engineering, Singapore.

Yin, C., Xia, L. ve Wang, J. (2018, 2018). *Data Stream Clustering Algorithm Based on Bucket Density for Intrusion Detection.* Advances in Computer Science and Ubiquitous Computing, Singapore.

Yogita ve Toshniwal, D. (2013, 22-23 Feb. 2013). *Clustering techniques for streaming data-a survey.* 2013 3rd IEEE International Advance Computing Conference (IACC).

Yogita, D. T. (2012). *A Novel Rough Set Based Clustering Approach for Streaming Data.* Second International Conference on Soft Computing for Problem Solving (Scrods 2012), New Delhi.

Yousefpour, A., Ibrahim, R., Abdull Hamed, H. N. ve Hajmohammadi, M. S. (2014, 2014). *Feature Reduction Using Standard Deviation with Different Subsets Selection in Sentiment Analysis.* Intelligent Information and Database Systems, Cham.

Zaki, M. J. (2000). Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3), 372-390.

Zhang, T, Ramakrishnan, R. ve Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. *SIGMOD Rec., 25*(2), 103-114.

# ÖZGEÇMİŞ

## Kişisel Bilgiler


Soyadı, adı : ŞENOL, Ali
Uyruğu : T.C.
Doğum tarihi ve yeri : 01.01.1985, Çınar
Medeni hali : Evli
Telefon : 0 (478) 211 75 31
Faks : 0 (478) 211 75 32
e-mail : alisenol@gazi.edu.tr


## Eğitim

| Derece | Eğitim Birimi | Mezuniyet Tarihi |
|---|---|---|
| Doktora | Gazi Üniversitesi / Bilgisayar Mühendisliği | Devam ediyor |
| Yüksek lisans | Gazi Üniversitesi / Bilgisayar Mühendisliği | 2013 |
| Lisans | Selçuk Üniversitesi / Bilgisayar Mühendisliği | 2009 |
| Lise | Mersin Tevfik Sırrı Gür Lisesi | 2003 |

## İş Deneyimi

| Yıl | Yer | Görev |
|---|---|---|
| 2015-Halen | Ardahan Üniversitesi | Araştırma Görevlisi |
| 2011-2015 | Gazi Üniversitesi | Araştırma Görevlisi |
| 2019-2011 | Ardahan Üniversitesi | Araştırma Görevlisi |

## Yabancı Dil

İngilizce

## Yayınlar


Şenol, A., Karacan, H. (2019). K-boyutlu ağaç ve uyarlanabilir yarıçap (KD-AR Stream) tabanlı gerçek zamanlı akan veri kümeleme. *Gazi Üniversitesi Mühendislik-Mimarlık Fakültesi Dergisi*, (Basımda).

Şenol, A., Karacan, H. (2018). Akan veri kümeleme teknikleri üzerine bir derleme. Avrupa Bilim ve Teknoloji Dergisi, (13), 17-30.

Şenol, A., Karacan, H., Akcayol, M. A. (2018). Genetic algorithm and fuzzy logic based flexible querying in databases. *Journal of Computers,* 13(6), 678-691.

Şenol, A., Karacan, H. (2012). *Sazan avlama (phishing): Kullanılan teknikler ve bunlardan korunma yöntemleri*. V. International Information Security and Cryptology Conference - ISCTURKEY 2012.

**Hobiler**

Futbol, Sinema ve Kitap Okumak

*GAZİ GELECEKTİR..*