T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

EKG VERİLERİ İÇİN GERÇEK ZAMANLI VERİ ANALİTİĞİ MİMARİSİ

YÜKSEK LİSANS TEZİ

Nur Banu OĞUR

Enstitü Anabilim Dalı	**:**	**BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ**
Tez Danışmanı	**:**	**Prof. Dr. Celal ÇEKEN**

Haziran 2018

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

EKG VERİLERİ İÇİN GERÇEK ZAMANLI VERİ ANALİTİĞİ MİMARİSİ

YÜKSEK LİSANS TEZİ

Nur Banu OĞUR

Enstitü Anabilim Dalı : **BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ**

Bu tez 28.06.2018 tarihinde aşağıdaki jüri tarafından oybirliği / oyçokluğu ile kabul edilmiştir.

Prof.Dr.	**Doç. Dr.**	**Dr. Ögr. Üyesi**
Celal Çeken	**Ali Çalhan**	**Veysel Harun Şahin**
Jüri Başkanı	**Üye**	**Üye**

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim.

Nur Banu OĞUR

28.05.2018

TEŞEKKÜR

Yüksek lisans eğitimim boyunca değerli bilgi ve deneyimlerinden yararlandığım, her konuda bilgi ve desteğini almaktan çekinmediğim, araştırmanın planlanmasından yazılmasına kadar tüm aşamalarında yardımlarını esirgemeyen, teşvik eden, aynı titizlikte beni yönlendiren değerli danışman hocam Prof. Dr. Celal ÇEKEN'e ve bana sabırla katlanan canım aileme teşekkürü borç bilirim.

İÇİNDEKİLER

ŞEKİLLER LİSTESİ

ÖZET

Anahtar kelimeler: EKG veri seti, Apache Spark, Apache Kafka, veri analitiği, Apache Spark MLlib, Lojistik regresyon, makine öğrenmesi

İnternetteki veri hacimlerinin genişlemesiyle ortaya çıkan büyük veri kavramı, hayatın birçok alanında olduğu gibi tıp dünyasında da adından bahsettirmeye başlamıştır. İçerisinde makine öğrenmesi yöntemlerinin kullanımını da gerektiren büyük veri analitiği, geniş ve karmaşık veri setleri üzerinden faydalı bilginin çıkartılarak karar süreçlerinde kullanımını sağlar. Büyük veri kapsamındaki veri setleri üzerinde makine öğrenme stratejileri uygulamak işlemci ve hafıza alanı gibi kaynakların yoğun olarak kullanımını gerektirdiği için pahalı bir süreçtir. Bu nedenle, büyük veri analitiği için özel olarak geliştirilmiş platformlar tasarlanmıştır. Büyük veri analitiği sistemlerinden biri olan Apache Spark regresyon, sınıflandırma ve kümeleme yapabilen çeşitli makine öğrenmesi algoritmalarını bünyesinde bulundurmaktadır. Diğer bir veri analitiği sistemi olan Apache Kafka ise temelde sıralı diziler halinde kayıtları tutan ve diğer sistemlere mesajlaşma kuyruğu şeklinde sunan bir yapıdır. Muadil sistemlere göre performansı ve kolay entegre edilip çalışılabilmesi sayesinde yüksek önceliğe sahiptir.

In this thesis, a real-time data analytic system was developed to diagnose disease from biomedical ECG data using Logistic Regression algorithm. Apache Kafka, MongoDB, Apache Spark Streaming and MLlib technologies are used in the proposed architecture. The results show that the developed architecture can be used for real-time data analytics.

REAL TIME DATA ANALYTICS ARCHITECTURE FOR ECG

SUMMARY

Keywords: ECG Dataset, Apache Spark, Apache Kafka, data analytics, Apache Spark MLlib, Logistic Regression, machine learning

The concept of big data emerging from the expansion of data volumes on the Internet has begun to talk about its name in medicine as well as in many fields of life. Big data analytics, which also require the use of machine learning methods, enable the use of decision-making processes by extracting useful information from large and complex data sets. Implementing machine learning strategies on data sets within big data is an expensive process because it requires extensive use of resources such as CPU and memory. For this reason, platforms specially developed for big data analysis are designed. One of these systems, Apache Spark, has built-in machine learning algorithms ranging from regression to classification and clustering, which is easy to integrate. Another data analysis system, Apache Kafka, is basically a structure that holds records in sequential order and presents it as a messaging queue to other systems. It has high priority due to its performance and easy integration and operation according to equivalent systems.

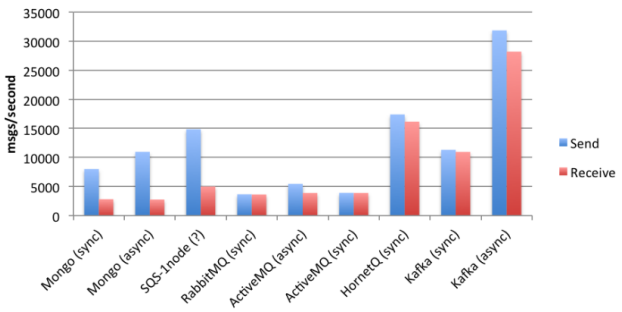
In this study, biomedical ECG data were taken and interacted with the Spark MLlib Logistic Regression algorithm, and a system was developed to determine whether persons were "patients". When the system was developed, Apache Kafka, Mongo DB, Apache Spark Streaming + MLlib technologies were used and real-time large data sets were reached the conclusion successful.

BÖLÜM 1. GİRİŞ

Medikal alanda dakikalar değil saniyeler bile çok önemli olabilmektedir. Günümüzde çoğu hastalığın tedavi edilememesinin altında yatan neden, vaktinde müdahale yapılamamasıdır. Bu sebeple, hastalıkların çözüme ulaşmasında "erken teşhis", büyük öneme sahiptir. Hastalığın teşhisinin mümkün olduğunca çabuk yapılması ve müdahaleye en kısa sürede başlanılması gerekmektedir. Fakat yoğun hasta sayısından, mevcut donanımsal cihazların yetersizliğinden ve doktor sayısının azlığından gecikmeler ortaya çıkabilmektedir. Erken teşhisin ve kaynakların yetersizliğinin yanı sıra özellikle hastanelerde yapılan tahlil sonuçlarını elde etmek, gerekli prosedürsel işlemlerden dolayı hastalar için çoğu zaman yıldırıcı ve vakit alıcı süreçlerdir. Bilişim ve biyomedikal cihaz teknolojilerinin sağlık alanında kullanımı bu tür gecikmelerin önlenmesi adına önemli faydalar sağlamaktadır.

Teşhisin elde edilmesi için ise hasta olunup olunmadığına karar veren bir mekanizmaya ihtiyaç duyulmaktadır. Bu mekanizma, gelen verileri anlık olarak işleyebilen ve karar verme yöntemlerini kendi bünyesinde bulunduran, gerçek zamanlı veri analitiği sistemleri kullanılarak geliştirilebilmektedir. Burada, en güncel teknolojileri ve büyük veri analiz ortamlarını kendi bünyesinde bulunduran Apache Spark devreye girmektedir. Apache Spark Streaming yapısı sayesinde gerçek zamanlı veri işlenmesine ve SparkSQL sayesinde özel veri birimi olan dataframe'ler yardımıyla sorgu yazılmasına olanak sağlamaktadır. Ayrıca Spark MLlib kütüphaneleriyle makine öğrenme algoritmalarını zahmetsiz entegre edilebilir hale getirmektedir. Apache Spark MLlib, dağıtık mimariden yararlanan, büyük veri makine öğrenmesi için açık kaynak kütüphaneleri ve platform bağımsız olması özellikleriyle en çok talep edilen platformlardan biridir.

Günümüz teknolojisinde büyük veri, hızlanarak artmaktadır ve zahmetsiz yollarla elde edilebilmektedir. [9]'da yapılan istatistiklere göre; geçtiğimiz iki yıl içinde insan ırkının bu zamana kadar olan tüm geçmişinden daha fazla veri oluşturulmuştur. Sadece Google üzerinde anlık 40.000 civarında arama sonuçları oluşmaktadır. Bu da günlük olarak yaklaşık 3.4 civarına ve yıllık olarak ise 1.2 trilyon araştırma verisi yapmaktadır. Video ve fotoğraf verilerinde, her dakika 300 saate kadar videonun yalnızca YouTube' a yüklendiği büyük bir büyüme görülmektedir ayrıca 5 yıl içinde dünya genelinde hepsi veri toplamak, analiz etmek ve veri paylaşmak için geliştirilen 50 milyar akıllı cihaz olacağı tahmin edilmektedir. Büyük veriyi hızlı bir şekilde çalıştırabilmek için ölçeklenebilir, anlık işlemeye izin veren, veriler transfer edilirken herhangi bir kullanıcı pasif olduğunda mesajları bünyesinde saklayabilen (replication factor), mesajları belli süre bekletebilen (persistence), kurulumu ve kullanımı kolay bir mesajlaşma sistemi olan Apache Kafka son derece uygundur. Şekil 1.1.' de Apache Kafka'nın diğer mesajlaşma platformlarıyla karşılaştırılmasına yer verilmiştir [1].



Şekil 1.1. Apache Kafka'nın diğer sistemlerle karşılaştırılması [1].

Son zamanlarda yapılan bu konularla ilgili literatür taramasına bakıldığında, güncel sistemlerin ihtiyaç çerçevesinde özelleştirildiği görülmektedir. Bunlardan birkaçına aşağıda yer verilmiştir:

Akış hesaplama (stream-computing) gerçek zamanlı büyük veriyi geliştirmeyi mümkün kılan hız ile ilgili konularda oldukça faydalı olan bir yeni hesaplama paradigmasıdır. [2] numaralı makalede Apache Spark akış yapısı ile makine öğrenmesi paralel ilerletilerek anlık olarak TV kanallarının tahmin edilmesi gerçeklenmiştir. [3] çalışması içinde, Apache Spark'ın temel teknolojileri ve üyeleri açıklanmış, temel üyelerinden biri olan MLlib kütüphanesinin içinde bulanan Lojistik Regresyon ile bir

uygulama çalıştırılmıştır. Mevcut sistem olan Apache Hadoop ile, Apache Spark'ın performans karşılaştırması yapılmıştır. [4] numaralı çalışma ise zaman ve band genişliğinin fazla kullanılmasının önüne geçmek için çoklu yapılarda (node) sistemin dağıtık çalışmasının sonucunu ortaya çıkan performans testini içermektedir. Apache Spark çekirdek yapısının üstünde 4 temel yapı ile oluşturulmuştur. Spark Streaming bunlardan biridir. [5] numaralı çalışmada ise Spark Streaming kullanıp hastanın verilerini "decision tree" makine öğrenmesi algoritmasına sokarak gerekli sonuçlar elde edilmiştir.

Java Sanal Makinesi (JVM), geliştirme için hangi framework'un kullanıldığına bakılmaksızın yürütme platformu olarak kullanılır. [6] numaralı makalede JVM davranışlarını gözlemleyebilmek için bir framework önerilmektedir. Önerilen çatı, JVM kullanarak çalışan yazılım dillerini karakterize etmek için başarıyla kullanılmıştır.

Çalışmamızda ise, kalbe giden damarlarda tıkanıklık, daralma veya küçülme olup olmadığı test edilmiş ve bu bağlamda en iyi sonuç ortaya koyan sistemler kullanılmıştır. Hastanın EKG verileri mesaj dağıtıcısı olarak görev yapan Apache Kafka sayesinde aynı anda hem NoSql veri tabanı MongoDB'de tutulmuş hem de Apache Spark'a gönderilmiştir. Apache Spark'a gelen veriler, Apache Spark'ın bünyesinde bulunan MLlib paketinin sunduğu "lojistik regresyon" algoritmasından geçirilmiş ve teşhisi gerçek zamanlı olarak gösterilmiştir. Geliştirilen projede bir tane broker iki tane tüketici kullanılmıştır. Özetlemek gerekirse:

Apache Kafka'nın üretici kısmı, verileri alıp tüketici olan MongoDB ve Apache Spark birimlerine eş zamanlı olarak yollamaktadır. MongoDB'ye gelen kişilere ait gerçek EKG verileri herhangi bir ön işlemden geçmeksizin veritabanına kaydedilmektedir. Apache Spark'a akış yapısı kullanılarak gerçek zamanlı olarak gelen kişilere ait gerçek EKG verileri ise Spark'ın MLlib kütüphanesinde bulunan lojistik regresyon algoritmasından geçmektedir. Algoritmada oluşturulan model ile gelen EKG verileri etkileşime sokulup, hastalığa dair teşhis ortaya çıkarılmaktadır. Sisteme veri geldiğinde akış yapısı kullanıldığından tekrar çalışıp hastalık teşhisini göstermektedir.

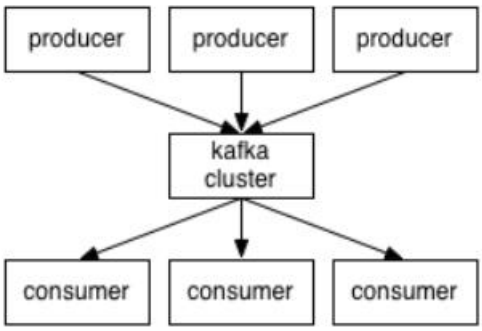
Bu tez çalışması beş bölümden oluşmaktadır. 2. Bölümde bulunan “Materyal ve Yöntem” kısmında, kullanılan teknolojiler olan Apache Kafka, Apache Spark, MongoDB ve makine öğrenmesi tekniği olarak kullanılan Lojistik Regresyon hakkında detaylı bilgiler sunulmaktadır. Ayrıca konularla ilgili bilgiler, şekiller ve tablolar ile desteklenmiştir. “Gerçeklenen Uygulama” kısmında, önerilen mimarinin tanıtımı ve çalıştırılması gösterilmiştir. “Araştırma Bulguları” kısmında, mimari üzerinde yapılan performans test ve sonuçları, “Tartışma ve Sonuç” kısmında ise geliştirilen mimarinin genel bir özeti ve çıkarılan sonuçların ne olduğu anlatılmıştır.

BÖLÜM 2. KULLANILAN TEKNOLOJİ VE YÖNTEMLER

2.1. Apache Kafka

İlk başta LinkedIn bünyesinde geliştirilip, sonrasında ise açık kaynak bir hale getirilen Apache Kafka [7], sürekli büyüyen büyük veri akışının düşük bir gecikme ile işlenmesine yardımcı olmayı hedefleyen ve bunun için, gerçek zamanlı veri boru hattı (data pipeline) ve akış (streaming) uygulamaları geliştiren ölçeklenebilir ve dağıtılmış bir yayın-abone (publish-subscribe) mesajlaşma platformudur.

Apache Kafka'nın çalışma prensibini ve bileşenleri için aşağıdaki diyagram son derece açıklayıcıdır.



Şekil 2.1. Apache Kafka çalışma prensibi diyagramı [2].

Diyagramda görüldüğü üzere üreticiler (producer) verilerini öncelikle Kafka kümeye sunmaktadır. Burada gözüken Kafka küme sadece tek bir cihazdan değil broker olarak adlandırılan sunuculardan veya bir broker grubundan oluşabilir. Sonrasında tüketiciler (consumer) belirlenen topic (konu) isimleri vasıtasıyla verilerini Kafka kümeden almaktadırlar. Bir konu, yayınlanan mesajların bir kategorisidir ve birden çok sunucu arasında bölümlenebilir, çoğaltılabilir. Konular sürekli eklenen sıralı ve değişmez

mesaj dizisi olan bölümlere (partition) ayrılır. Bölümler ise offset diye adlandırılan ve her bölümdeki iletileri benzersiz olarak tanımlayan parçalara ayrılır.

Apache Kafka terminolojisini kısaca anlatmak gerekirse;

- KONULAR (TOPICS): Yollanacak mesaj, konular yardımıyla sunucularda saklanmaktadır.

- ÜRETİCİ (PRODUCER): Konulara mesaj gönderen kısımdır. Hangi mesajların hangi bölüme gideceğine karar verebilirler. Bu karar verme işlemini ya bir scheduling algoritması olan round-robin şeklinde gerçekleştirirler ya da semantic partitioning ile gerçekleştirir. Örneğin mesaj "message-key" ile özelleştirebilir, istenilen sırada çalışması sağlanabilir. Eğer özelleştirilme yapılmaz ise round-robin algoritması ile mesaj iletimi gerçekleştirilir.

- TÜKETİCİ (CONSUMER): Konuları dinleyenlerdir. Bir veya daha fazla konuya abone olup(subscribe), konuların mesajlarını tüketebilirler.

- BÖLÜM (PARTITION): Konular bölümlerden oluşabilir. Bu nedenle istenilen miktarda veri işleyebilirler.

- OFFSET: Bölümler benzersiz sıra ID'lerine sahip offset'lere bölünmektedir.

- BÖLÜM KOPYALARI (REPLICAS OF PARTITION): Veri kaybını önlemek amacıyla alınan bölümlerin kopyalarıdır.

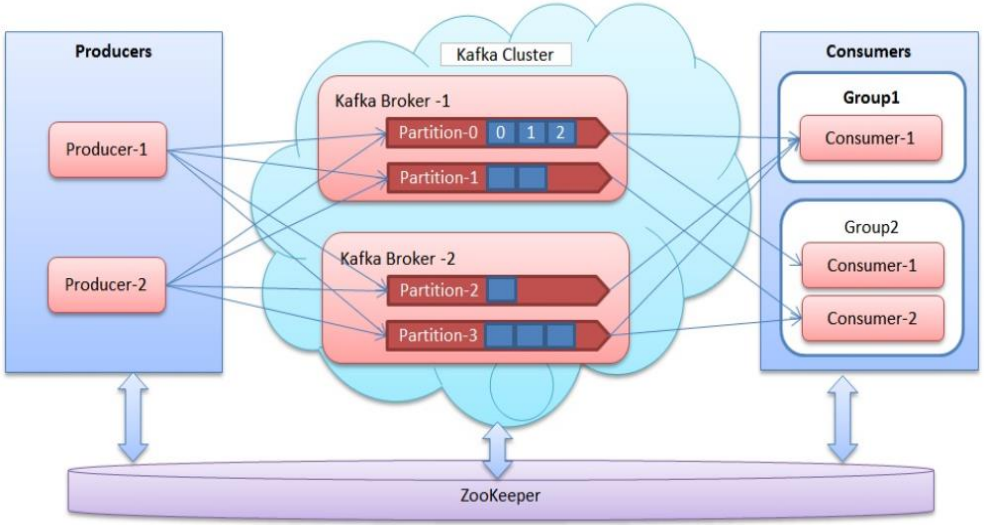
- BROKERS: Apache Kafka'nın sunucularıdır. Toplu halde kümeyi oluştururlar.

- LİDER (LEADER): İlgili konuda gerekli yazma ve okuma işlemlerinden sorumlu olan kısımdır. Her bölüm bir tane lidere sahiptir.

- TAKİPÇİ (FOLLOWER): Liderin talimatlarını takip eder. Lider bölümde herhangi bir kayıp yaşandığı zaman takipçilerden biri lider konumuna geçer.

- ZOOKEEPER: Sunucularda herhangi bir problem ile karşılaşılmasından ötürü sunucunun işlevini yerine getiremez olduğu durumlarda, başka bir sunucuyu devreye sokarak akışın bozulmadan devam etmesini sağlar. Aynı durum yeni bir sunucu sisteme entegre edildiğinde de gözlemlenir. Ayrıca hangi mesajın hangi bölümde ve hangi ofsette olduğu bilgisini de tutmakla görevlidir.

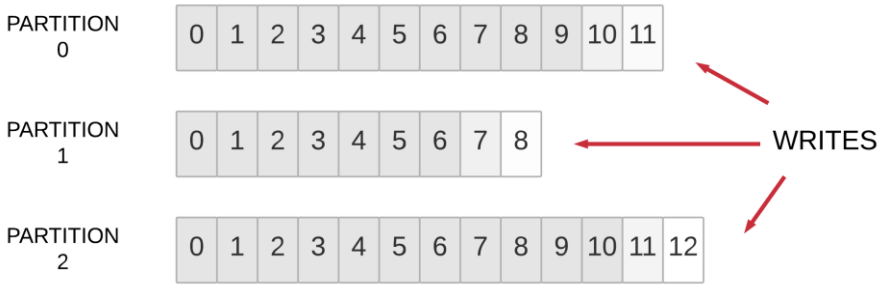
Şekil 2.2.'de Kafka kümede iki bölüme sahip iki broker gösterilmiştir. İki üretici mesajlarını dört bölüme sahip bir konu üzerinden yayınlamaktadır. Grup bir tüketici, grup iki ise iki tüketiciye sahip olacak şekilde konfigüre edilmiştir. Mesajların en az bir kere iletileceğinin garanti edilmesi ise tüketici grup oluşturulmasıyla sağlanmıştır.



Şekil 2.2. Apache Kafka'nın genel çalışma mekanizması [18].

Apache Kafka küme, tüm yayınlanmış mesajları özelleştirilebilir bir zaman aralığı boyunca korur. Bir üretici tarafından gönderilen mesajlar, gönderildikleri sıraya göre bir konu bölümüne eklenir. Şekil 2.3.'de üreticinin verilerini, offset'lere vasıtasıyla üç bölüme ayrılmış konuya yerleştirilmesi resmedilmiştir. Tüketici (consumer), iletileri kaydedildikleri sırada görür yani gerçek zamanlı büyük veri işlemek için elverişli ortam sunar. Bunların yanı sıra ölçeklenebilme (scalable), tekrarlama (replication),

verileri saklayıp bekletme (persistence) gibi özellikleri sayesinde büyük veri kullanıcıları için ideal bir teknolojidir [8].



Şekil 2.3. Apache Kafka'nın bölümlere verileri dağıtması

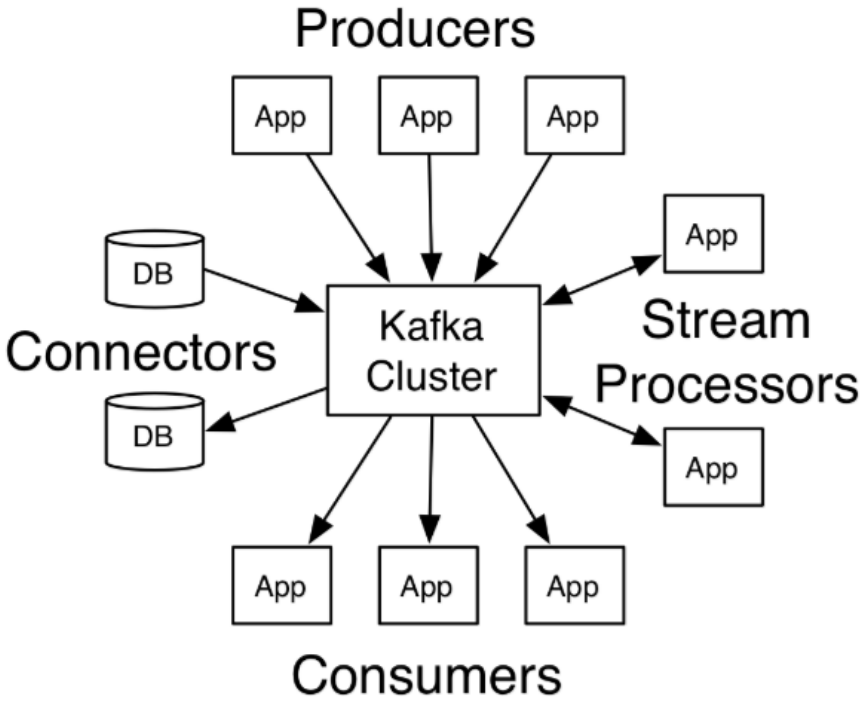
Kafka'nın dört temel API'si (Application Programming Interface-Uygulama Programlama Arayüzü) bulunmaktadır:

- Üretici API, bir uygulamanın bir veya daha fazla Kafka konusuna bir kayıt akışı yayınlamasına izin verir.

- Tüketici API, bir uygulamanın bir veya daha fazla konuya abone olmasını ve bu konulara, üretilen kayıtların akışının işlenmesini sağlar.

- Akışlar API, bir uygulamanın bir veya daha fazla konudan bir girdi akışını tüketmesini ve bir veya daha fazla çıkış konusuna bir çıkış akışı üretmesini ve giriş akışlarını çıkış akışlarına etkili bir şekilde dönüştürmesini sağlayan bir akış işlemcisi olarak görev yapmaktadır.

- Konektör API, Kafka konularını mevcut uygulamalara bağlayan üreticilerin veya tüketicilerin oluşturulmasına ve çalıştırılmasına imkân verir. Örneğin, ilişkisel bir veri tabanına bağlanan bir bağlantı, bir tablodaki her değişikliği yakalayabilir.

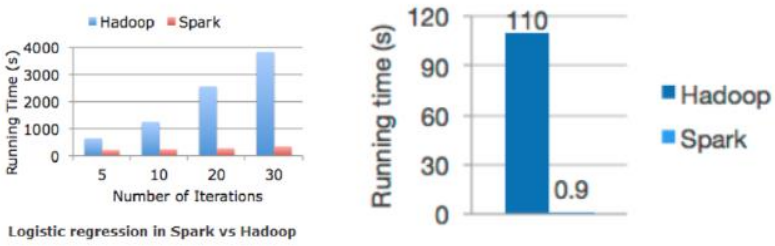


Şekil 2.4. Apache Kafka Küme yapısı [3].

Biz çalışmamızda, Apache Kafka'yı yüksek performanslı mesaj dağıtıcısı olması ve büyük veri için en ideal teknoloji olması nedeniyle tercih ettik. Çalışmamızda önemli yere sahip Apache Kafka'yı tüketici olarak belirlediğimiz Apache Spark ve MongoDB' ye verileri anlık olarak göndermek için kullandık.

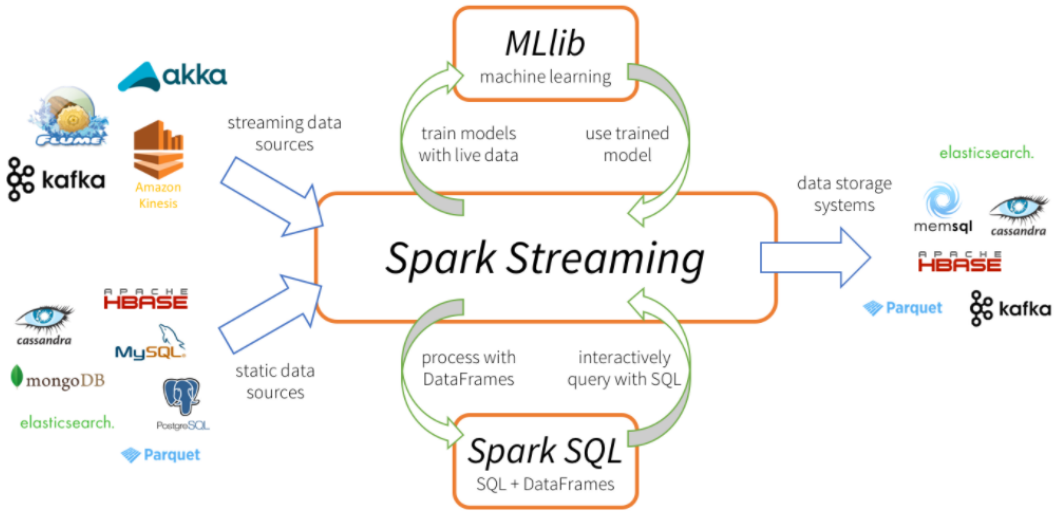
2.2. Apache Spark

Apache Hadoop, sıradan sunuculardan (commodity hardware) oluşan küme üzerinde büyük verileri işlemek için uygulamaları çalıştıran ve Hadoop Distributed File System (HDFS) olarak adlandırılan bir dağıtık dosya sistemi ile Hadoop MapReduce özelliklerini bir araya getiren, Java ile geliştirilmiş platformdur. Apache Spark ise, büyük veri kümeleri üzerinde paralel olarak işlem yapılmasını sağlayan Scala ile geliştirilmiş açık kaynak kodlu olan bir cluster computing (küme hesaplama) platformudur. Apache Spark Apache Hadoop'un karşısında yer almak yerine Hadoop ailesinin bir üyesi olup Hadoop'un zayıf kaldığı bazı konulardaki eksiklikleri gidermekle görevlidir denilebilir.



Şekil 2.5. Apache Spark ve Hadoop karşılaştırması [16].

En önemli farklılıkları arasında Hadoop, Hadoop Distributed File System (HDFS)'den gelen verileri diske yazıp diskten okuma işlemi gerçekleştirdiğinden zaman kaybına sebep olurken, bu işlem Spark'da hafızaya yazıp hafızadan okuma (in-memory) ile gerçekleştiği için hız açısından oldukça performans sağlamaktadır. Bir diğer fark ise Apache Spark'ın bünyesinde bulundurduğu SQL, Streaming, MLlib ve GrapX teknolojileri ile kolay entegre olup çalışabilmesidir. Bu teknolojiler vasıtasıyla kullanım alanı açısından geniş kitlelere hitap eden Apache Spark, MLlib ten dolayı veri bilimcilerin, Spark SQL den dolayı veri analistlerin ilgisini çekmiştir. Şekil 2.6.'da Spark Streaming yapısının hangi başka sistemlerle entegre çalıştığı resmedilmiştir.



Şekil 2.6. Apache Spark Streaming [16].

2.2.1. Neden Apache Spark?

Apache Spark'ı seçmek için birden çok sebep vardır, bunlardan en önemlileri aşağıda sıralanmıştır.

Basitlik: Spark'ın özelliklerine, tümü kapsamlı ve hızlı ve kolay bir şekilde, etkileşimde bulunacak biçimde tasarlanmış bir dizi zengin API aracılığıyla erişilebilir. Bu API'ler iyi bir şekilde dokümante edilmiş ve veri bilimcilerin Spark'ı geliştirebilmesi için anlaşılabilirlik yüksek tutulmuştur.

Hız: Spark Hadoop kümesinde uygulamayı çalıştırmaya yardımcı olur. Bunu hafızada 100 kata kadar daha hızlı, disk üzerinde çalışırken ise 10 kata kadar daha hızlı gerçekleştirir. Hızlı olmasını, diskte okuma yazma işlemlerini azaltmasıyla ve ara işlem verilerini hafızaya kaydetmesiyle mümkün kılar.

Çoklu Dil Desteği**:** Birden çok dili destekler. Spark Java veya Python gibi yazılım dillerinde yerleşik API'ler sağlar. Bu nedenle farklı dillerde uygulama yazılmasına elverişli ortam sunar.

Gelişmiş Analitik: Spark 'map' ve 'reduce' özelliklerini desteklemesinin yanı sıra ayrıca SQL sorguları, akış ile alınan veri (streaming data), makine öğrenimi (machine learning) ve grafik algoritmaları gibi kolay entegre edilebilir sistemleri de bünyesinde kolaylıkla çalıştırabilir.

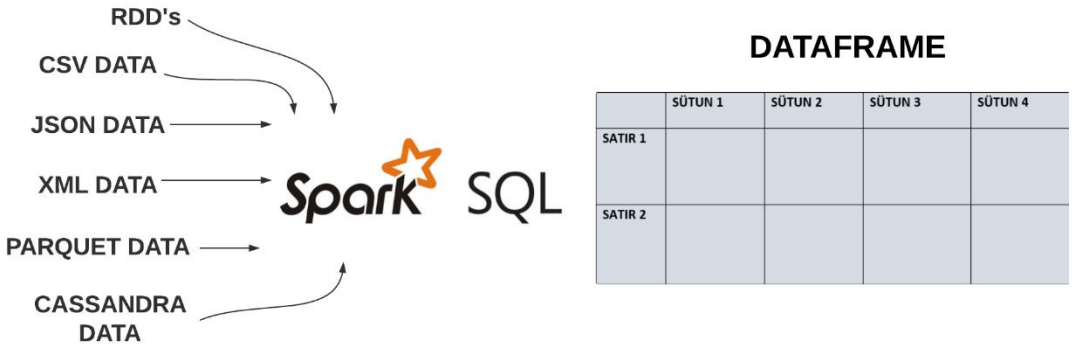
2.2.2. Apache Spark kullanım yolları

Apache Spark'a erişmek kanallar vasıtasıyla oluşmaktadır. Bunlardan biri "SparkContext" oluşturarak gerçekleştirilir. İhtiyaca göre, farklı API'lerin giriş noktaları tanımlanır. Her farklı fonksiyonu (Streaming, Hıve, Sql) kullanmak için ayrı context'ler oluşturulmak zorundadır. Diğeri ise "Spark Session" kanal yapısıdır. Spark'ın fonksiyonları ile bağlantı kurmak için yalnız bir giriş noktası sağlar. Tüm fonksiyonları kullanmak için ayrı Session oluşturmaya gerek yoktur. Kanallarda kullanılacak verisetleri farklılık göstermektedir. Spark Core'da oluşturulan tüm

fonksiyonlar RDD (Resilient Distributed Dataset)'ler üzerinde gerçeklenmekte iken SparkSQL'de DataFrame, Spark Streaming de ise DStream üzerinde gerçeklenmektedir.

2.2.3. Apache Spark veri kümeleri

DataFrame: Bir RDD gibi, bir DataFrame, değişmez bir dağıtılmış veri topluluğudur. Bir RDD'den farklı olarak, veriler ilişkisel veritabanındaki bir tablo gibi adlandırılmış satır ve sütunlara düzenlenir.



Şekil 2.7. Apache Spark Dataframe [16].

RDD: Esnek Dağıtılmış Veri Kümeleri (RDD) Apache Spark'in temel bir veri yapısı, nesnelerin değişmez bir dağıtılmış topluluğudur. Üç şekilde oluşturulur:

- Yerel bilgisayardan dosya ile,
- HDFS yardımıyla,
- Paralelize metodu kullanılarak.

Oluşturulan RDD'ler genel olarak transformasyon ve aksiyon olmak üzere 2'ye ayrılır.

2.2.3.1. Transformasyon

Mevcut RDD üzerinden yeni bir RDD oluşturma yöntemidir. Spark'daki tüm transformasyonlar tembeldir, çünkü sonuçları hemen hesaplamamaktadır. Transformasyonlar, yalnızca bir 'Action' metot çalıştırıldığı zaman hesaplanır. Bu tasarım Spark'ın daha verimli çalışmasını sağlar. Örneğin, 'map' her veri kümesi öğesini bir işlevden geçiren ve sonuçları temsil eden yeni bir RDD döndüren bir transformasyondur.

2.2.3.2. Aksiyon

Aksiyon işleminde RDD üzerinden hesaplama, dış sistemlere verileri kaydetme işlemleri yapılır. Örneğin 'count', RDD sayısını hesaplamaya yarayan, 'first' ise ilk RDD' yi döndüren aksiyonlardır. Şekil 2.7.'de bazı özel operasyonlardan gösterilmiştir.

```
// Break every message into words and return list of words
JavaDStream<String> words = lines.flatMap(new FlatMapFunction<String, String>() {
    @Override
    public Iterator<String> call(String line) throws Exception {
        return Arrays.asList(line.split(" ")).iterator();
    }
});

// Take every word and return Tuple with (word,1)
JavaPairDStream<String,Integer> wordMap = words.mapToPair(new PairFunction<String, String, Integer>() {
    @Override
    public Tuple2<String, Integer> call(String word) throws Exception {
        return new Tuple2<>(word,1);
    }
});

// Count occurance of each word
JavaPairDStream<String,Integer> wordCount = wordMap.reduceByKey(new Function2<Integer, Integer, Integer>() {
     @Override
     public Integer call(Integer first, Integer second) throws Exception {
         return first+second;
     }
 });
```

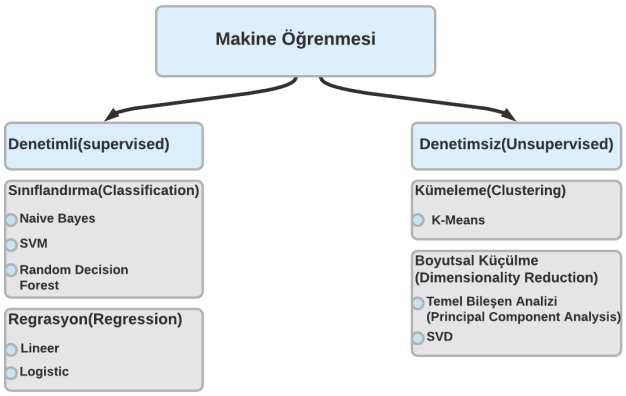
Şekil 2.8. Apache Spark özel operasyonları

2.2.4. Spark SQL

Spark SQL, Spark çekirdeğinin en üst kısmında, DataFrame gibi yapılar kullanarak, yapısal ve yarı yapılandırılmış büyük veriler üzerinde, SQL tabanlı analizler yapıp gelen verinin kullanımının kolaylaştırılmasına olanak tanımaktadır.

2.2.5. Spark MLlib

Apache Spark'ın bünyesinde barındırdığı zengin MLlib kütüphaneleri (korelasyon, sınıflandırma ve regresyon, işbirlikçi filtreleme, kümeleme ve boyut küçültme) sayesinde makine öğrenmesi uygulamaları kolaylıkla projeye entegre edilebilmektedir. Şekil 2.8.'de Apache Spark MLlib üzerinde çalışan makine öğrenmesi algoritmaları gösterilmiştir.



Şekil 2.9. Apache Spark makine öğrenmesi algoritmaları

Apache Spark makine öğrenmeleri algoritmalarını iki paket halinde sunulmaktadır. Birinin kullanılabilmesi için "org.apache.spark.mllib" paketi, diğerinin kullanılabilmesi için "org.apache.spark.ml" kütüphanesi projeye dahil edilmelidir.

- spark.mllib: RDD yapılarının üzerine kurulu orijinal API içerir.

- spark.ml: ML boruhattını (pipeline) oluşturmak için DataFrame'ler üzerine kurulu yüksek düzeyli API sağlar.

İki farklı paketin kendi içinde olumlu olumsuz değerlendirecek yönleri bulunmaktadır. DataFrame ile API, daha yeni, çok yönlü ve esnek olduğundan, pratik makine öğrenme öğrenme boru hattı kolay bir şekilde kurulabilmektedir bu sebeple spark.ml'nin kullanılması önerilmektedir. Bununla birlikte, spark.mllib daha eski olması ve uzun süredir geliştirilmiş olması dolayısıyla daha fazla özelliğe sahiptir.

Biz uygulamamızı geliştirirken mevcut olan iki makine öğrenmesi paketini de kullandık ve performans açısından değerlendirdik.

2.2.5.1. MLLIB-RDD tabanlı API

MLlib, tek bir makinede depolanan yerel vektörleri, matrisleri ve bir veya daha fazla RDD tarafından desteklenen dağıtılmış matrisleri (distributed matris) destekler. Yerel vektörler (local vektör), yerel matrisler (local matris) ve aşağıda sıralanmış bütün ortak arabirimler aracılığıyla kullanılan basit veri modelleridir.

1. Local vector
2. Labeled point
3. Local matrix
4. Distributed matrix
    a. RowMatrix
    b. IndexedRowMatrix
    c. CoordinateMatrix
    d. BlockMatrix

2.2.5.2. ML-DataFrame tabanlı yüksek seviyeli API

Apache Spark makine öğrenmesi paketlerinden bir diğeri olan spark.ml paketi, kullanıcıların pratik makine öğrenimi boru hatlarını oluşturmasına ve ayarlamasına yardımcı olan DataFrame üzerine kurulu tek tip yüksek düzeyli API'ler sağlamayı amaçlamaktadır [16].

Aşağıda spark ML API'sı tarafından tanıtılan temel kavramlara yer verilmişti [16].

- DataFrame: Spark ML, Spark SQL'den oluşan DataFrame'i çeşitli veri türlerini barındırabilen bir, ML veri kümesi olarak kullanır.

- Dönüştürücü (Transformer): Dönüştürücü, bir DataFrame'i başka bir DataFrame'e dönüştürebilen bir algoritmadır. Örneğin, bir ML modeli,

DataFrame'i özellikleri olan bir DataFrame'e tahminlerle dönüştüren bir ara elemandır.

- Tahminci (Estimator): Bir Tahminci, bir dönüştürücü üretmek için DataFrame'e uyan bir algoritmadır. Örneğin, bir öğrenme algoritması, bir DataFrame üzerinde çalışan ve bir model üreten bir tahmin edicidir.

- Boru Hattı (Pipeline): Boru hattı, bir ML iş akışı belirtmek için birden fazla dönüştürücüyü ve tahminciyi zincirleyen yapıya denilmektedir.

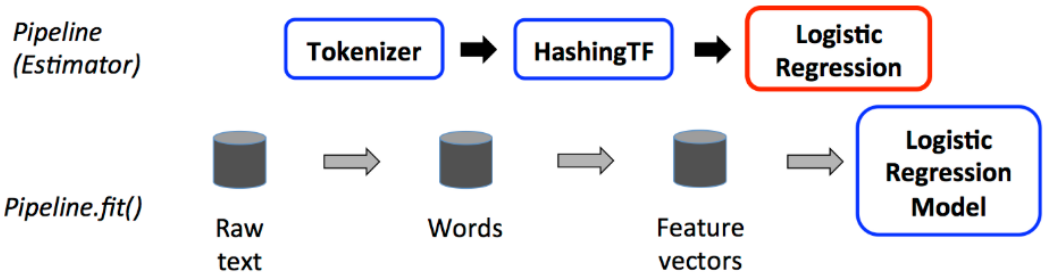
Makine öğrenmesinde, veri işlemek ve öğrenmek için bir dizi algoritmanın çalışması, sıklıkla gerçekleştirilen uygulamalar arasındadır. Örneğin, basit bir metin belgesi işleme için oluşturulan iş akışı birkaç aşama içerebilir:

- Her bir dokümanın metninin kelimelere ayrılması

- Her bir dokümanın sözlerinin sayısal bir özellik vektörüne dönüştürülmesi.

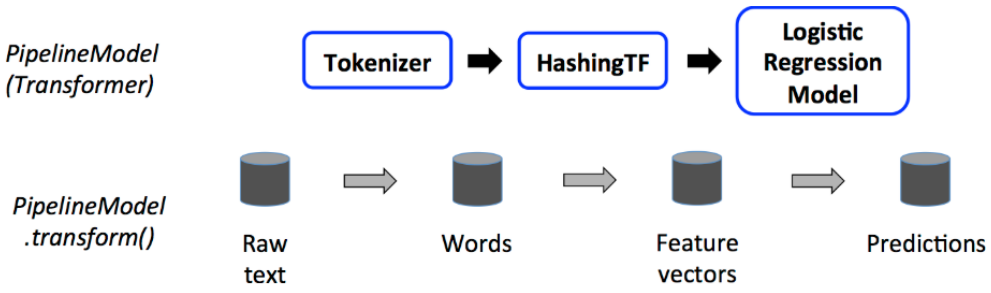
- Özellik vektörlerini ve etiketleri kullanarak bir tahmin modelinin oluşturulması.

Apache Spark ML, boru hattı aşamaları (PipelineStages-Transformers and Estimators) dizisinin belirli bir sırayla çalıştırılacağı bu gibi bir boru hattı, iş akışını temsil etmektedir ve her bir aşama ya bir düzenleyici veya tahmin edicidir. Bu aşamalar sırayla çalıştırılır ve DataFrame girişi her bir aşamadan geçtiği zaman dönüşümü gerçekleştrilir. Dönüşüm aşamaları için DataFrame'de transform() yöntemi çağrılır. Tahminci aşamalarında, bir dönüştürücü üretmek için fit() yöntemi çağrılır.



Şekil 2.10. Apache Spark boru hattı modellemesi-eğitim seti [16]

Text formatındaki verilerin modele dönüşüm aşaması gösterilen Şekil 2.9.'da üst sıra üç aşamalı bir boru hattını temsil etmektedir. İlk ikisi (Tokenizer ve HashingTF) dönüştürücü (mavi) üçüncüsü ise (LogisticRegression) bir tahminciyi göstermektedir (kırmızı). Tokenizer.transform() yöntemi, ham metin belgelerini kelimelere böler ve DataFrame'e kelimelerle yeni bir sütun ekler. HashingTF.transform () yöntemi, kelime sütununu özellik vektörlerine dönüştürür ve bu vektörlerle DataFrame'e yeni bir sütun ekler. Lojistik regresyon bir tahminci olduğundan, boru hattı öncelikle bir lojistik regresyon model üretmek için LogisticRegression.fit() çağırır. Boru hattı daha fazla aşamaya sahipse, DataFrame'i bir sonraki aşamaya geçirmeden önce DataFrame'de LogisticRegressionModel'in transform() yöntemini çağırır. Böylece, bir Pipeline'ın fit() yöntemi çalıştıktan sonra, bir dönüştürücü olan bir boru hattı modeli (PipelineModel) üretir. Bu PipelineModel test zamanında kullanılır.



Şekil 2.11. Apache Spark boru hattı modellemesi-test seti [16]

Boru hattı modelin transform() yöntemi bir test veri kümesinde çağrıldığında, veriler sıralı olarak boru hattından geçirilir. Her aşamanın transform() yöntemi veri kümesini günceller ve bir sonraki aşamaya geçirir.

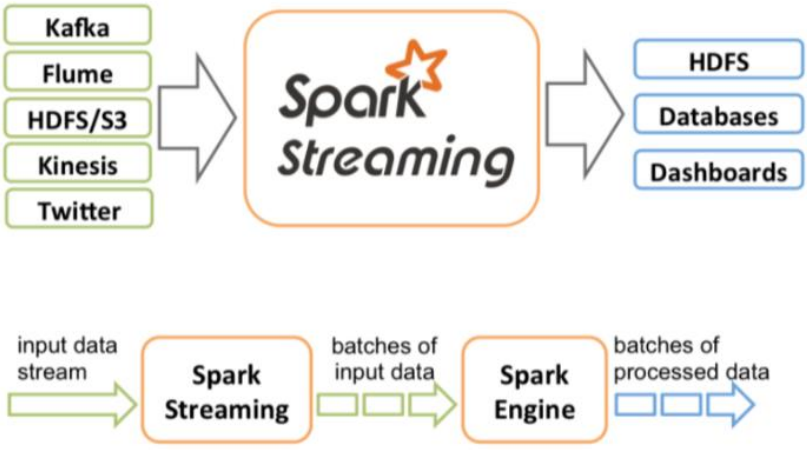
Boru hatları ve boru hattı hodelleri, eğitim ve test verilerinin aynı özellik işleme adımlarından geçtiğinden emin olunması sağlar.

2.2.6. Spark GraphX

GraphX, Apache Spark'ta dağıtılmış bir grafik işleme çerçevesidir. Kullanıcı tanımlı grafikleri modelleyebilen paralel işlemler yapılmasına olanak sağlayan grafik hesaplamayı ifade etmek için bir API sağlar.

2.2.7. Spark Streaming

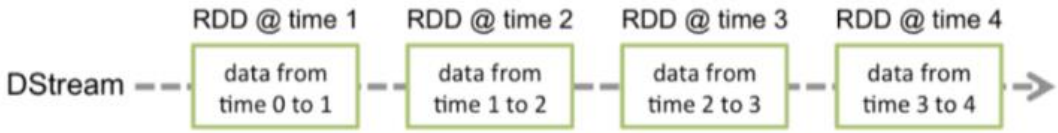
Sürekli akıp giden verilerin gerçek zamanlı yani anlık olarak işlenmesi noktasında Spark'ın büyük veri kümeleri üzerindeki çok hızlı veri işleme kapasitesi, bu verileri anlık olarak analiz etmeye, maksimum verimlilikte olanak tanımaktadır. Şekil 2.9.'da Apache Spark Streaming yapısının genel özelliklerine yer verilmiştir.



Şekil 2.12. Apache Spark Streaming genel mekanizması [16].

Veri Kafka, Flume, Kinesis veya TCP soketleri gibi birçok kaynaktan alınabilir ve planlama (map), azaltma (reduce), birleştirme (join) ve pencereleme (window) gibi üst düzey işlevlerle ifade edilen karmaşık algoritmalar kullanılarak işlenebilir. Son olarak, işlenen veriler dosya sistemlerine, veri tabanlarına ve anlık veri işleyen sistemlere aktarılabilir.

Genel çalışma mekanizması olarak Spark Streaming, canlı giriş veri akışlarını alır ve verileri toplu işlere böler. Sonrasında alınan veriler, harman halindeki sonuçların son akışını oluşturmak için Spark motoru tarafından işlenir.



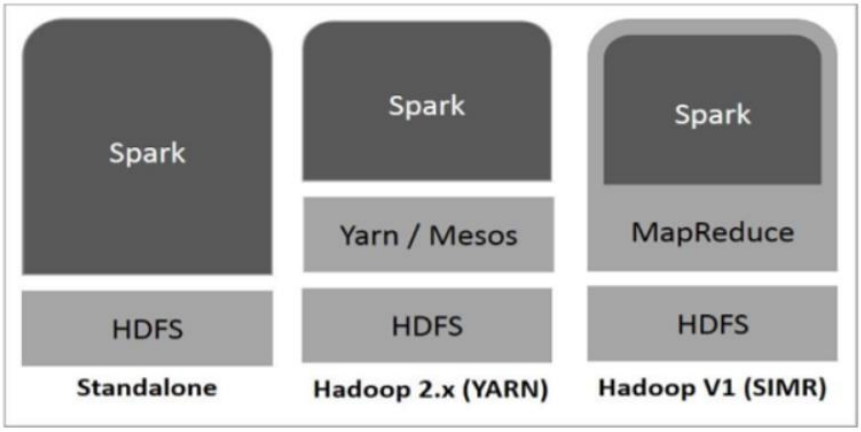
Şekil 2.13. Apache Spark Streaming DStream yapısı [16].

Spark Streaming sürekli bir veri akışını temsil eden DStream adı verilen yüksek düzeyli bir soyutlama mekanizması sağlar. DStreams, Kafka, Flume ve Kinesis gibi kaynaklardan gelen girdi veri akışlarından veya diğer DStreams üzerinde yüksek düzeyli işlemler uygulayarak oluşturulabilir. DStream Şekil 2.10.'da görüldüğü gibi farklı zaman dilimlerine bölünen RDD dizilerinden oluşmaktadır.

2.2.8. HDFS sistemi üzerinde Apache Spark

Apache Hadoop teknolojisinin bir parçası olan HDFS (Hadoop Distributed File System-Hadoop Dağıtık Dosya Sistemi) sistemini Apache Spark da kullanmaktadır. Apache Spark'ın HDFS'yi etkin hale getirebilmesi için üç yol mevcuttur. Şekil 2.11.'de Apache Spark için HDFS yapısı gösterilmiştir.

Standalone: Spark standalone dağıtımı, Spark'ın HDFS (Hadoop Dağıtılmış Dosya Sistemi) üzerindeki yerini kapladığı ve alanın HDFS için açıkça ayrıldığı anlamına gelir. Burada Spark ve MapReduce, kümedeki tüm Spark işlerini yapacak şekilde birlikte çalışır.



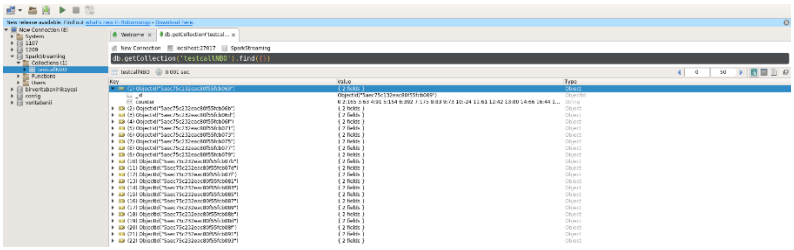
Şekil 2.14. Apache Spark HDFS yapısı [19].

Hadoop YARN: Apache Spark ön kurulum ya da kök erişimi gerekmeksizin basit ve kolay bir şekilde çalışır. Spark'ı Hadoop ekosistemine veya Hadoop yığınına entegre etmeye yardımcı olur. Diğer bileşenlerin yığının üstünde kolaylıkla çalışmasına izin verir.

Spark Üzerinde MapReduce (SIMR): Standalone dağıtımın yanı sıra Spark ile ilgili işlemleri başlatmak için kullanılır. SIMR ile kullanıcı Spark'ı başlatabilir ve herhangi bir yönetimsel erişim olmaksızın kabuğunu kullanabilir.

2.3. MongoDB

MongoDB, geliştirme ve ölçekleme kolaylığı için tasarlanmış açık kaynak, belge yönelimli (document-oriented) NoSQL veri tabanıdır. MongoDB'de her kayıt, aslında bir veri bütünüdür. Dokümanlar MongoDB'de JSON benzeri Binary JSON (BSN) formatında saklanır. BSON belgeleri, sakladıkları elemanların sıralı bir listesini içerir nesnelerdir. Her bir eleman, bir alan adı ve belirli tipte bir değerden oluşur.



Şekil 2.15. Robomongo ara yüzü

2.4. Lojistik Regresyon

Lojistik regresyon, bağımlı değişkenin tahmini değerlerini olasılık olarak hesaplayarak, olasılık kurallarına uygun sınıflama yapma imkânı sunan ham veri setlerini analiz edip yorumlayabilen bir istatistiksel yöntemdir [9]. Sürekli ya da ayrık olabilecek açıklayıcı değişkenlerin ağır sınırlandırmalarına yer vermeyen Lojistik regresyon [10], tıp, jeoloji, biyoloji, sağlık ve finans gibi çoklu kullanım alanlarına sahiptir.

Lojistik regresyon algoritmasında tahmin edilen değer, negatif sonsuz ile pozitif sonsuz arasında herhangi bir yerde olabilir. Sınıf değişkenine, yani 0-hayır, 1-evet olmak için algoritmanın çıktısına ihtiyaç duyulmaktadır. Bu nedenle, lineer denklemin çıktısını bir [0,1] aralığına sıkıştırılmaktadır [20]. Lojistik regresyonun ikili kategorik bağımlı değişkene sahip olmasından dolayı olasılıklar üzerine kurulu olduğunu söylenebilmektedir.

$$\text{Logit}(p) = \log[p/(1-p)] = \ln[p/(1-p)]$$

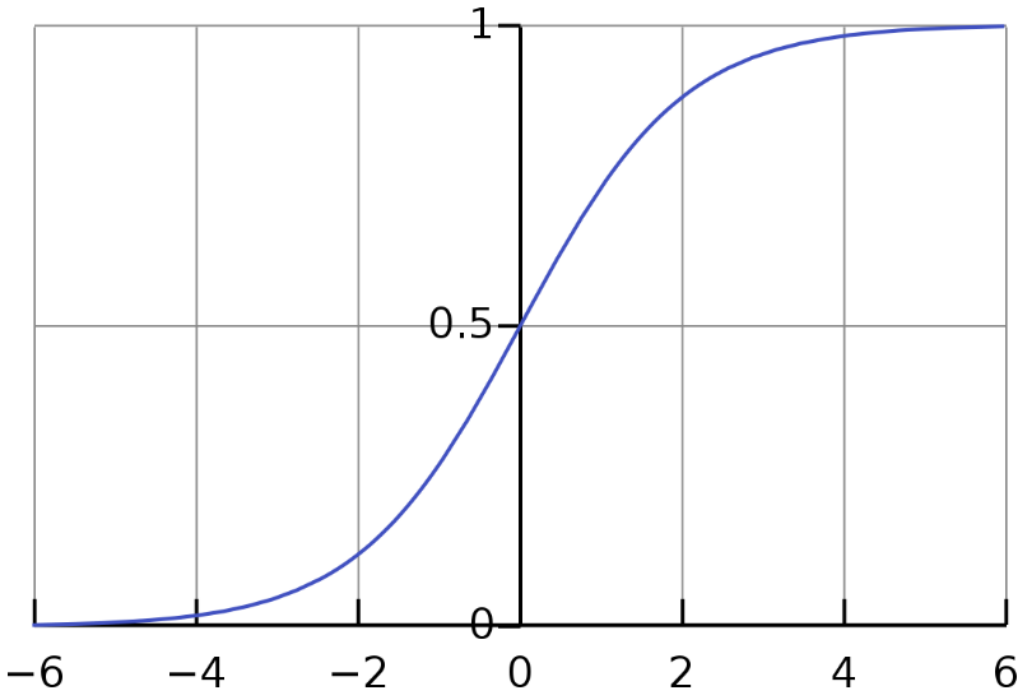
$$\text{Logit}(p) = \log[p/(1-p)] = \ln[p/(1-p)]$$

$$\text{Logit}[p(x)] = \log[p(x)/1-p(x)] = a+b_1x_1+b_2x_2+.....+b_ix_i$$

$$P = \exp^{(a+b1x1+b2x2+.....+bixi)} / 1+\exp^{(a+b1x1+b2x2+.....+bixi)}$$

Şekil 2.16. Lojistik regresyon fonksiyonu [22].

Şekil 2.16.'da görüldüğü gibi model, bir olayın gerçekleşme olasılığı ve gerçekleşmeme olasılığının birbirine bölünmesinin doğal logaritmasının alınması ile kurulmaktadır. Logaritmik dağılımın kullanılmasının nedeni dağılımı normalleştirebilmektir. Formüller neticesinde kategorik değişken Şekil 2.14.'de gözüktüğü üzere 1 ve 0 iken + sonsuz ile –sonsuz arasında değer almaktadır.

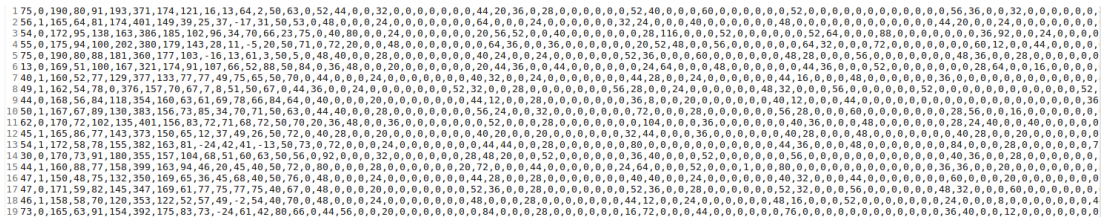


Şekil 2.17. Lojistik regresyon fonksiyonu grafiği [20].

Çalışmamızda Lojistik Regresyonun kullanım alanı açısından benzerlik göstermesinin yanında tercih edilmesinin bir diğer sebebi ise performans açısından diğer algoritmaları geride bırakmasıdır. Bazı çalışmalar, Lojistik regresyon modelinin, frekans oranı, iki değişkenli istatistikler, yapay sinir ağları, destek vektör makinaları (SVM) ve bazı durumlarda sınıflandırma ağaçları gibi diğer çok değişkenli istatistiksel yöntemlerden daha doğru ve verimli olduğunu ileri sürmüşlerdir [10]. Ekvador Andes den bir vaka çalışması, lojistik regresyonun en düşük hata oranları ve en iyi genelleme yeteneklerini gösterdiğinin sonucuna varmıştır [11].

BÖLÜM 3. GERÇEKLENEN TEST DÜZENEĞİ

Çalışmamızda, EKG verilerini lojistik regresyon makine öğrenmesi algoritması ile etkileşime sokup, hastalığa dair sonuç çıkarılması sağlanıldı. Sonuç çıkarılması aşamasında Apache Spark bünyesinde mevcut olan spark.ml ve spark.mllib paketlerine ait logistik regresyon algoritmaları ayrı ayrı test edilip performansları ölçümlendi. Bunu yaparken UCI veri setlerinden olan 279 sınıflandırmaya sahip Cardiac Arrhythmia veri setini kullandık [15]. Amacımız kalbe giden damarlarda herhangi bir iletim bozukluğu ya da tıkanıklık olup olmadığını ölçümlemeyebilmektir.



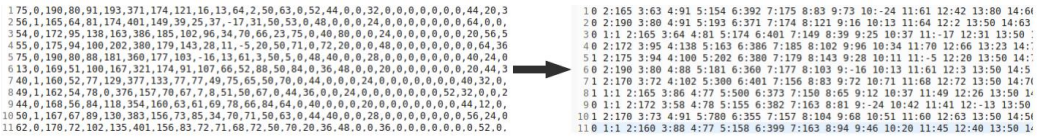
Şekil 3.1. EKG veri seti [15].

Kullanılan veri setinde 452 kişinin EKG verisi vardır. Bir kişinin EKG verisi 279 ayrı kategoriden oluşmaktadır. Veri setinin ilk değer, kişinin yaşı, sonrasında cinsiyeti, boyu, kilosu, QRS duraklaması, P-R aralığı, Q-T aralığı diyerek devam etmektedir.

EKG verilerinde, kişinin kalbe giden damarında problem olup olmadığını analiz etmek için birden çok veri üzerinde sınırlandırma yapılabilir ama problemin en açık şekilde gözler önüne serilmesi için P-R aralığı değerlendirilmelidir. P-R aralığı eğer 200 ms' den büyükse birinci derece kalp bloğu varlığı söz konusudur, eğer 90 ms' den düşükse pre-eksitasyonu (atriyum ve ventriküller arası aksesuar yolak varlığı) veya AV nodal (junctional) ritmini akla getirir [13].

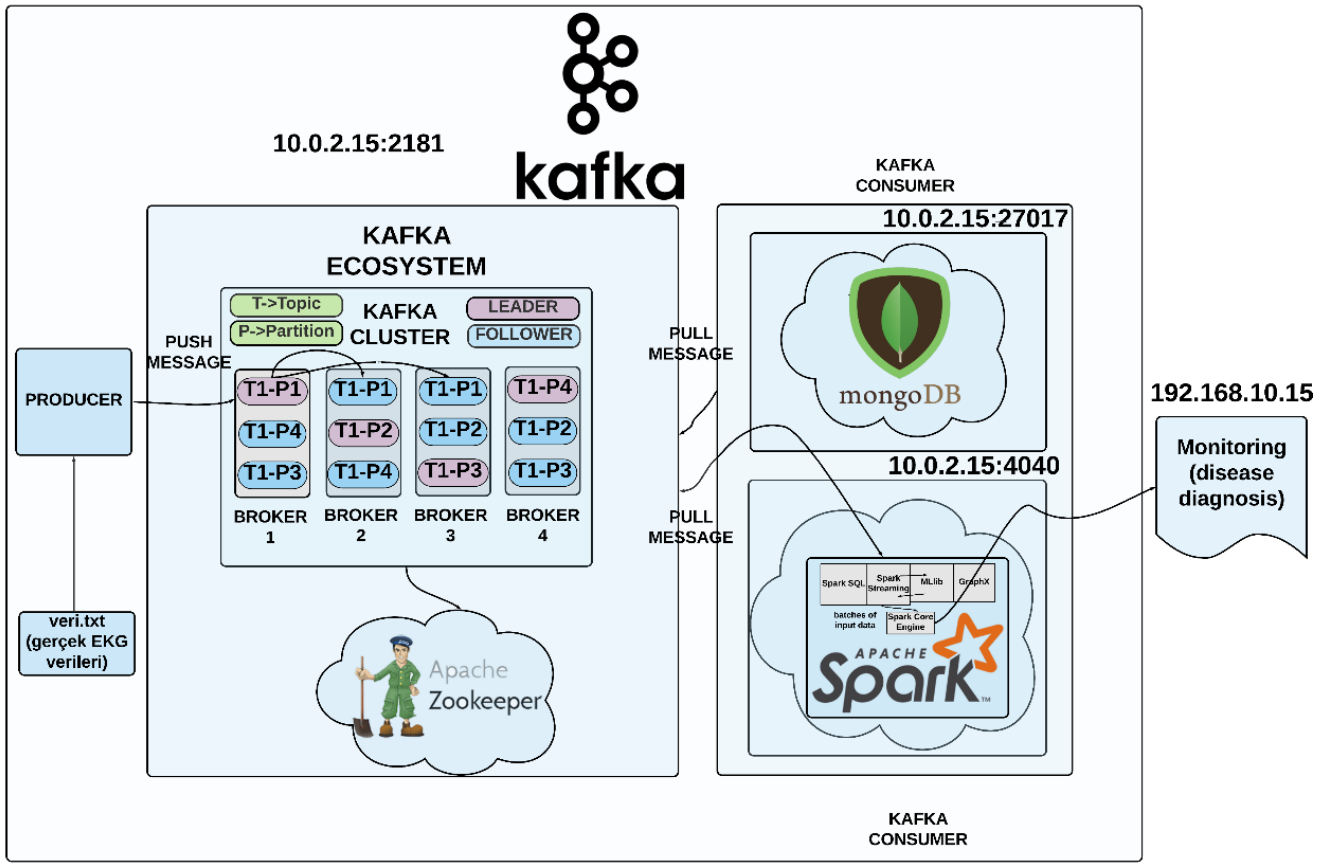
3.1. Apache Spark-MLlib Paketi

Geliştirdiğimiz sistemde, P-R değer aralığı, girilen bütün verilerde gözetilmiştir. Verilerin tüm etiketli çıkışları bu değer aralıklarına göre belirlenmiştir. Modelin doğruluk oranı da yine bu değer aralıklarına bakılarak oluşturulan etiketler sayesinde sağlanmıştır.



Şekil 3.2. EKG verilerinin LIBSVM formatına dönüşümü

Şekil 3.2.'de solda görünen gerçek EKG verilerini Spark MLlib'in yorumlayabileceği format olan LIBSVM formatına dönüşümünü gerçekleştirdik.



Şekil 3.3. Gerçeklenen sistemin büyük resmi

Geliştirilen modelin büyük resmi Şekil 3.3.'te, zamanlama şeması ise Şekil 3.4.'te gösterilmiştir. EKG verilerinden hastalık teşhisi için kullanılan mimarinin genel işlem adımları Şekil 3.5.'te yer alan zamanlama şemasına göre aşağıda özetlenmiştir.

- Gerçek EKG verileri veri.txt dosyasından okunur ve LIBSVM formatına Apache Kafka üreticiye gönderilir.

- Üretici, verileri "broker" a gönderir, veriler "broker" 'larda bulunan bölümler (partition) içinde tutulur.

- İlgili konu ile ilişiği olan tüketiciler yollanan verileri eşzamanlı olarak alır. Oluşturulan mimaride MongoDB ve Apache Spark olmak üzere iki tane tüketici tanımlanmıştır

- Tüketici olan MongoDB, verileri ilgili konu üzerinden okur, veri tabanına yazar.

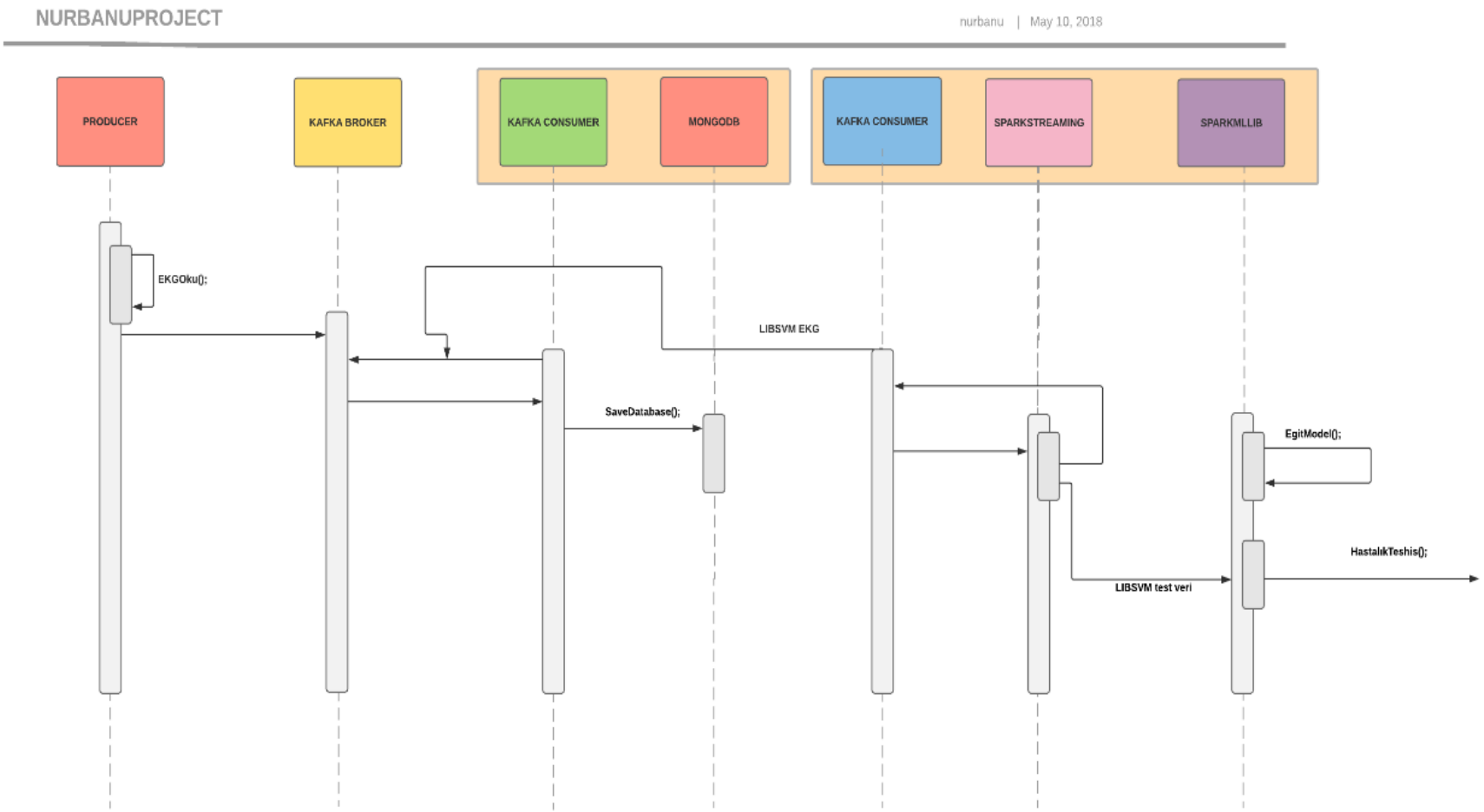
- Diğer tüketici olan Apache Spark, akış şeklinde gelen verileri, kullanıcı tarafından belirlenen (1 saniye) çerçeveler halinde okur.

- Apache Spark içinde eğitim seti kullanılarak oluşturulan model, üreticiden yollanan veriyi test seti olarak kullanır.

- Mevcut olan model ile gelen veriler etkileşime sokulur ve model gelen verilere dair hastalık teşhisinde bulunur.

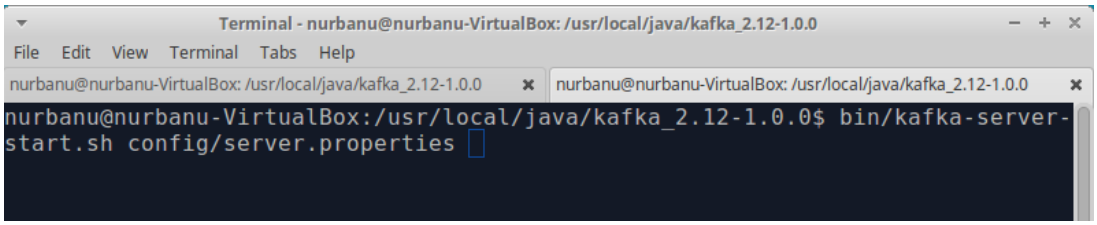
- Modelin bulunan teşhislere göre doğruluk oranı gösterilir.

- Akış yapısı kullanarak geliştirilen sistemde, başka veriler yollandığında yukarda anlatılanlar döngü halinde devam etmektedir. Herhangi bir zaman diliminde yollanan EKG verisinin gerçek zamanlı olarak teşhisi koyulup, gösterilmektedir.



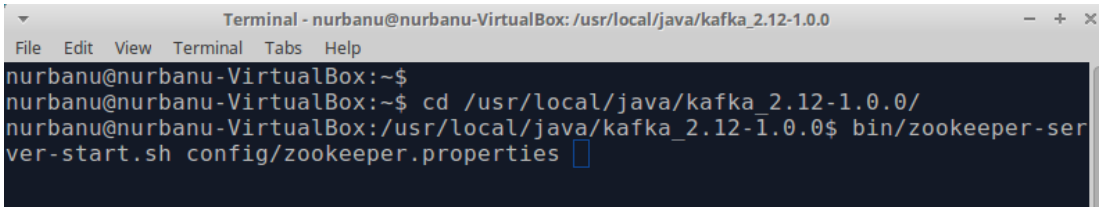
Şekil.3 4. Gerçeklenen sistemin zamanlama şeması

Projemizde bulunan Apache Kafka'nın aktif edilebilmesi için öncelikle Kafka Server başlatılmalıdır. Aşağıdaki kodu Kafka Server'ın başlatılması uygulamalı olarak göstermektedir.



Şekil 3.5. Apache Kafka Server'ın başlatılması

Apache Kafka'da gerekli broker atama ve lider bölüm belirlemekle görevli olan haliyle Apache Kafka için bir hayli önem teşkil eden Zookeeper Server' in başlatılması için aşağıdaki kod çalıştırılmalıdır.



Şekil 3.6. Zookeeper'ın başlatılması

Apache Kafka'da haberleşmenin sadece üzerinden gerçekleşebileceği konu tanımlamak gerekmektedir. Console'dan Apache Kafka dosyasına girerek yapılan bu işlemde, konuyu özelleştirmek mümkündür. Aşağıda "ba" topic ismine sahip, tekrarlama faktörü bir ve bölümü üç olan konu tanımlanmıştır.

Şekil 3.7. Konu tanımlanması

Çalışma kapsamında oluşturulan ilgili konuda, bir bölüm ve bir tekrarlama faktörü oluşturulmuş, Apache Kafka, bir üretici ve iki tüketiciden oluşturulmuştur. Kafka kümede ise bir tane broker üzerinden alışveriş sağlanmıştır. Apache Kafka üreticinin çalıştırılmasıyla veriler, ilgili konuya gönderilmiştir. Kafka'nın parametreleri ve bağlantı kurulacak olan konu ismi tüm tüketicilerde tanımlanmıştır. (Şekil 3.8.)

```
public static void main(String[] args) {
  Properties props = new Properties();
  props.put("bootstrap.servers", "10.0.2.15:9092");
  props.put("zk.connect", "localhost:2181");
  props.put("group.id", "groupsS");
  props.put("enable.auto.commit", "true");
  props.put("auto.commit.interval.ms", "1000");
  props.put("auto.offset.reset", "earliest");
  props.put("session.timeout.ms", "30000");
  props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
  props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");


  KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(props);
  kafkaConsumer.subscribe(Arrays.asList("ba"));
```

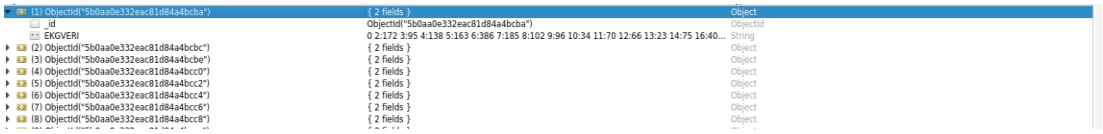
Şekil 3.8. Tüketici MongoDB'nin Apache Kafka ile konfigürasyonu

Böylelikle tüketicilerin çalıştırılmasıyla aynı tanımlı konu ismindeki verileri alma işlemi, eş zamanlı olarak gerçeklenebilecektir. Tüketici olarak tanımlananlardan biri NoSQL veri tabanı olan MongoDB'ye verileri kaydederken, diğer tüketici ise Apache Spark'a verilerini yollamaktadır.

Gelen verilerin aşağıdaki kodlama vasıtasıyla, Şekil 3.9.'da gösterildiği gibi "EKGVERI" Collection'unun "EKGVERI" adlı satırında tutulması sağlanmıştır. Şekil 3.9.'da tüketici MongoDB'nin kod kısmı, Şekil 3.10.'da ise MongoDB ara yüzü Robomongo'ya kaydedilen EKG verileri gösterilmiştir.

```
KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(props);
kafkaConsumer.subscribe(Arrays.asList("bir"));
while (true) {
      ConsumerRecords<String, String> records = kafkaConsumer.poll(100);

          for (ConsumerRecord<String, String> record : records) {
              Mongo mongo = new Mongo("localhost", 27017);
              DB  db = mongo.getDB("SparkStreaming");
              DBCollection table = null;
              table = db.getCollection("EKGVERI");
              System.out.println("Collection başarılı bir şekilde seçildi.");
              BasicDBObject document = new BasicDBObject();
              document.put("EKGVERI", record.value());
              table.insert(document);
              System.out.println("Ekleme başarıyla sağlandı.");
          }
  for (ConsumerRecord<String, String> record : records) {
    System.out.println("Partition: " + record.partition() + " Offset: " + record.offset()
        + " Value: " + record.value() );
  }
}
```

Şekil 3.9. Tüketici MongoDB'nin kod kısmı



Şekil 3.10. Ara yüz Robomongo'daki kayıtlı veriler

Diğer tüketici olan Apache Spark'a gelen veriler ise akış özelliği sayesinde gerçek zamanlı olarak alınmaktadır. Bu sistemin kullanılmasıyla, herhangi bir zaman diliminde yollanan EKG verilerinin anlık olarak işlenmesi hedeflenmiştir. Şekil 3.11.'de Kafka üreticinin verileri konuya ataması gösterilmiştir.

```java
import org.apache.kafka.clients.producer.Callback;

public class ProducerTest extends Thread {

    private static final String topicName
            = "bir";
    public static final String fileName = "veri.txt";

    private final KafkaProducer<String, String> producer;
    private final Boolean isAsync;

    public  ProducerTest(String topic, Boolean isAsync) {
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092,localhost:9093");
        props.put("client.id", "DemoProducer");
        props.put("key.serializer",
                "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
                "org.apache.kafka.common.serialization.StringSerializer");
        producer = new KafkaProducer<String, String>(props);
        this.isAsync = isAsync;
    }

    public void sendMessage(String key, String value) {
        long startTime = System.currentTimeMillis();
        if (isAsync) { // Send asynchronously
            producer.send(
                    new ProducerRecord<String, String>(topicName, key),
                    (Callback) new DemoCallBack(startTime, key, value));
         }
        else
        { // Send synchronously
            try {

                  for (int i = 0; i < 1; i++) {
                    producer.send(
                            new ProducerRecord<String, String>(topicName, key, value))
                            .get();
                    System.out.println("Sent message: (" + key + ", " + value + ")");


                  }

            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (ExecutionException e) {
                e.printStackTrace();
            }
        }
    }
```

Şekil 3.11. Üretici konuya veri yollama

Apache Spark'ın paralel olarak çalışması .setMaster özelleştirerek sağlanır. Eğer local [*] ise sistemin uygulandığı cihazda mevcut olan çekirdeklerin hepsini kullanarak çalışmasını, maksimum performans elde edilmesini sağlar. Şekil 3.12.'de görüldüğü gibi gerçeklenen çalışmada local [*] setMaster'ı kullanılmış, yüksek performans elde edilmiştir.

```
//Configure Spark to listen messages in topic test
Collection<String> topics = Arrays.asList("bir");

// Create a local StreamingContext with two working thread and batch interval of 5 second
SparkConf conf = new SparkConf().setAppName("LogisticRegressionClassifier")
        .setMaster("local[*]").set("spark.executor.memory","2g");


SparkContext sp=new SparkContext(conf);

StreamingContext ssc=new StreamingContext(sp,Durations.seconds(1));
//Read messages in batch of 5 seconds
JavaStreamingContext jssc = new JavaStreamingContext(ssc);
```

Şekil 3.12. Apache Spark konfigürasyonu

```
String path1 = "LibsvmFormat.txt";

JavaRDD data = MLUtils.loadLibSVMFile(sp, path1).toJavaRDD();//mutils

// Split initial RDD into two... [80% training data, 20% testing data].
JavaRDD<LabeledPoint>[] splits = data.randomSplit(new double[] {0.8, 0.2}, 11L);
JavaRDD<LabeledPoint> training = splits[0].cache();
JavaRDD test = splits[1];

System.out.println(training.count());
System.out.println(test.count());

// System.out.println(TestVeri.count());
// Run training algorithm to build the model.
LogisticRegressionModel model = new LogisticRegressionWithLBFGS()
        .setNumClasses(10)
        .run(training.rdd());
```

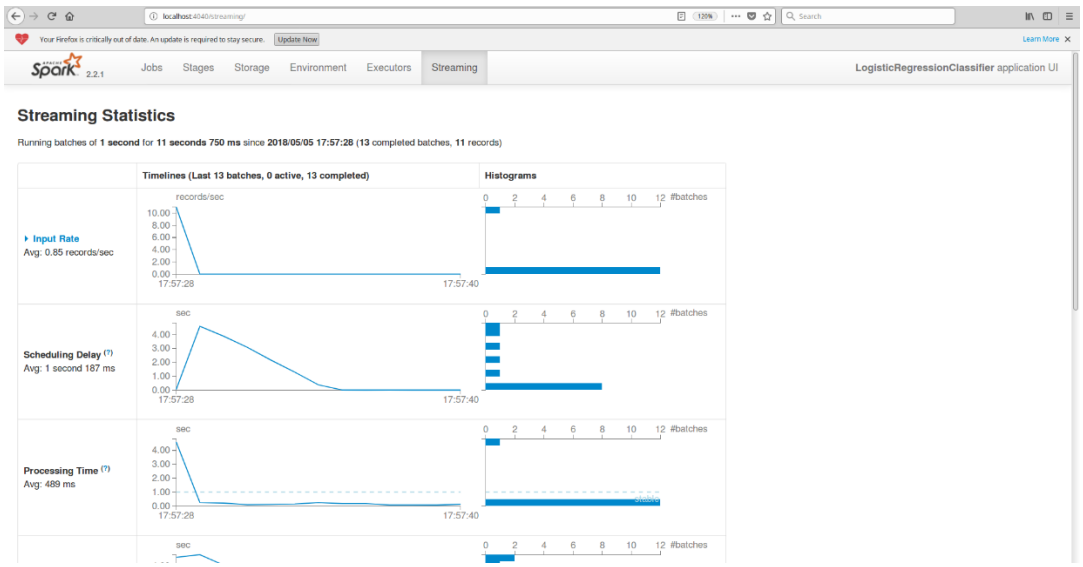
Şekil 3.13. Modeli oluşturma

```
String path = "dosya.txt";
JavaRDD TestVeri = MLUtils.loadLibSVMFile(sp, path).toJavaRDD();//mutils
System.out.println(TestVeri.count());

JavaRDD<Integer> Tahmin = TestVeri.map(s -> model.predict(((LabeledPoint) s).features()));

Tahmin.saveAsTextFile("Tahmin");


JavaPairRDD<Object, Object> predictionAndLabels = TestVeri.mapToPair(p ->
new Tuple2<>(model.predict(((LabeledPoint) p).features()), ((LabeledPoint) p).label()));


MulticlassMetrics metrics = new MulticlassMetrics(predictionAndLabels.rdd());



double accuracy = metrics.accuracy();
System.out.println("Accuracy = " + accuracy);
```

Şekil 3.14. Modelin tahmin işlemi kodu

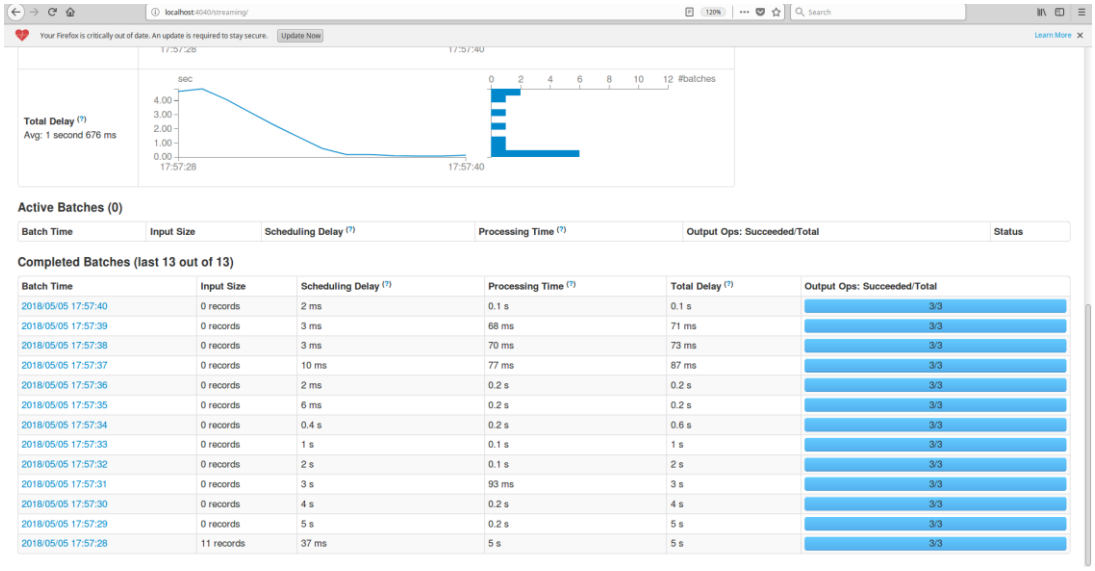
```
Tahmin.saveAsTextFile("Tahmin");

File file = new File("Tahmin/part-00000");
try {
    List<String> satir = FileUtils.readLines(file);

    for (int i=0;i<satir.size();i++) {

        String p1 = satir.get(i).toString();
        System.out.print(p1);

        if(p1.equals("1.0"))
        {
            System.out.println("<<HASTA>>");
        }
        else
        {
            System.out.println("<<HASTA DEĞİL>>");
        }
    }
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Şekil 3.15. Hastalık teşhisi kodu



Şekil 3.16. Apache Spark web ara yüzü histogramları

Akış uygulaması çalıştırıldıktan sonra Apache Spark'ın kullanıcılarını sunduğu web ara yüzünü kullanarak (localhost:4040) beklenen gecikme, işlem süresi, toplam gecikme, verilerin giriş oranı, yollanan veri miktarı, yollanan veri boyutu gibi bilgilere ve çeşitli histogram çizimlerine anlaşılabilir teması ile erişim sağlanmış, çıktıları Şekil 3.16. ve Şekil 3.17.'de gösterilmiştir.



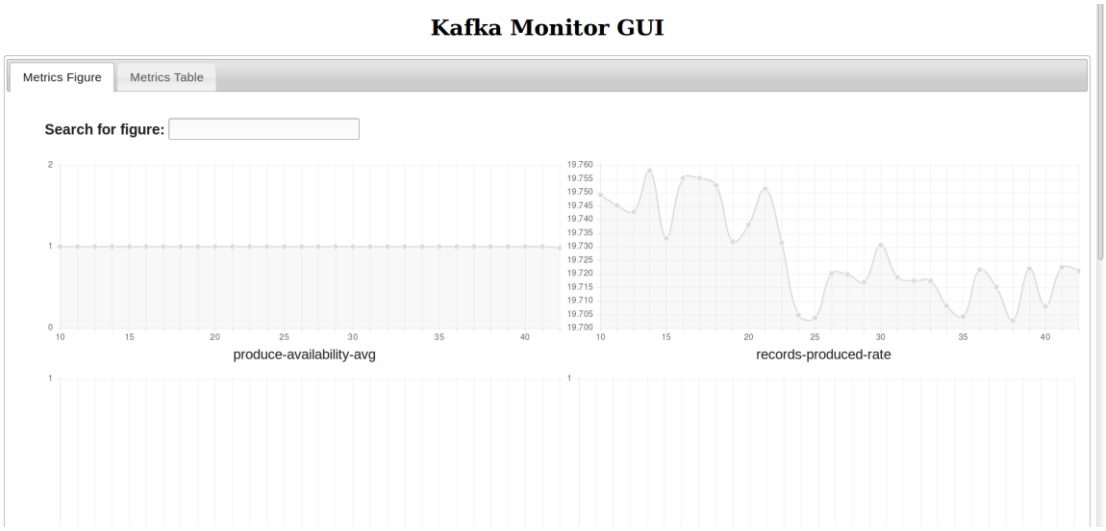
Şekil 3.17. Apache Spark Web Ara yüzü

Tüketici Apache Spark çalıştırıldığında yerelde yüklü etiketli veriler, lojistik regresyon algoritması ile etkileşime girerek modeli oluşturur. Apache Spark'ın akış teknolojisi sayesinde gerçek zamanlı olarak alınan veriler test verisi olarak kullanılır. Model, test verisinin sonucunu tahmin eder. Bu sonuçlara göre kişinin hasta olup olmadığı bilgisi ekranda gösterilir. Aynı zamanda modelin bulduğu sonuçlar ile gelen etiketli verilerin karşılaştırması yapılarak eğitim seti ile oluşturulan modelin doğruluğu test edilir, ekranda gösterilir. Sağlık alanında yapılan çalışmalar için modelin doğruluk oranı daha fazla önem arz etmektedir. Bu hassasiyeti göz önünde bulundurarak geliştirdiğimiz çalışmamızda, model Şekil 3.18.'de gözüktüğü gibi %100 doğrulukla çalışmaktadır.

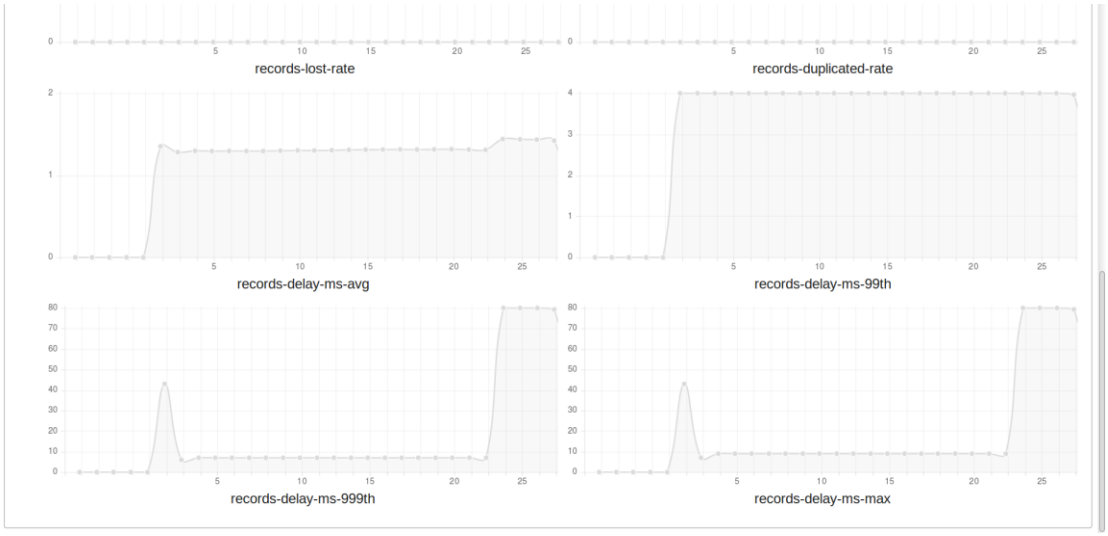
```
5
0.0<<HASTA DEĞİL>>
1.0<<HASTA>>
0.0<<HASTA DEĞİL>>
0.0<<HASTA DEĞİL>>
0.0<<HASTA DEĞİL>>
Accuracy = 1.0
---New RDD with3partitions and5records
0 2:165 3:63 4:91 5:154 6:392 7:175 8:83 9:73 10:-24 11:61 12:42 13:80 14:66 16
1 2:175 3:94 4:100 5:202 6:380 7:179 8:143 9:28 10:11 11:-5 12:20 13:50 14:71 1
0 2:172 3:95 4:138 5:163 6:386 7:185 8:102 9:96 10:34 11:70 12:66 13:23 14:75 1
0 2:190 3:80 4:91 5:193 6:371 7:174 8:121 9:16 10:13 11:64 12:2 13:50 14:63 16:
0 1:1 2:165 3:64 4:81 5:174 6:401 7:149 8:39 9:25 10:37 11:-17 12:31 13:50 14:5
-------------------------------------------
Time: 1525442153000 ms
-------------------------------------------
(262:4,1)
(29:52.1)
```

Şekil 3.18. Hastalığın teşhisi

Apache Kafka'da olmayan, bağımsız kurulumu gerçekleştirilen monitör tool'lar, işleyişin nasıl olduğunu görmek ve performans testlerini daha kolay gözlemleyebilmek için büyük kolaylıklar sağlar. Ayrıca uçtan uca gecikme, hizmet kullanılabilirliği ve mesaj kaybı oranı gibi bir dizi üretilmiş hayati istatistiksel verileri elde etmek için uçtan uca boru hatlarını kullanarak Kafka Cluster'ı izlemeye olanak tanır. Biz uygulamamızda Linkedin/kafka-monitor uygulamasını kullandık, ekran görüntüleri ise Şekil 3.19. ve Şekil 3.20.'deki gibidir.



Şekil 3.19. Kafka Monitor Tool-1



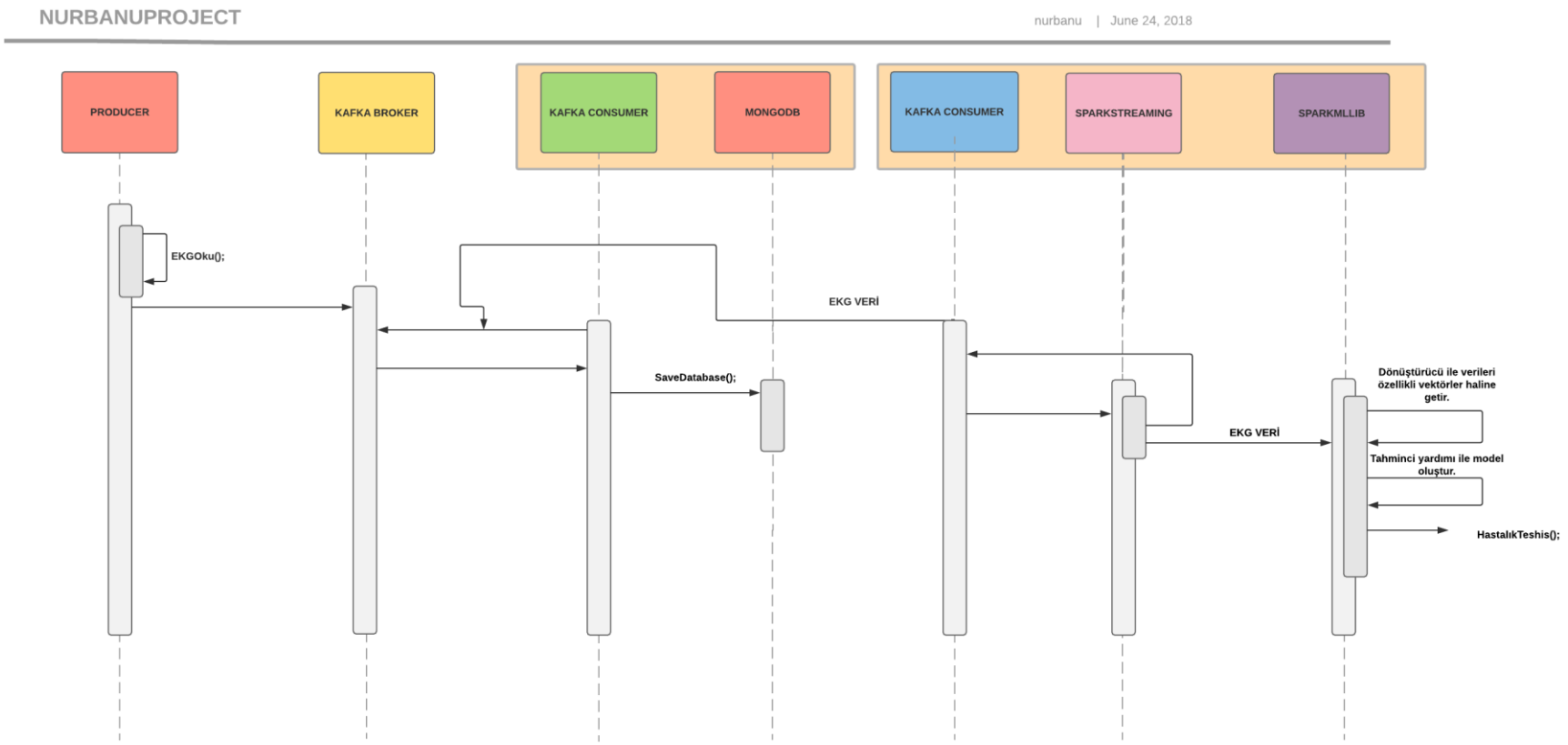
Şekil 3.20. Kafka Monitor Tool-2

3.2. Apache Spark-ML Paketi

Apache Spark MLlib paketinin kullanıldığı uygulamadan farklı olarak bu uygulamada spark ML paketi kullanılmıştır. Apache Spark'taki makine öğrenmesine gelene kadar gerçekleştirilen bütün aşamalar spark ML ile de gerçeklenmiştir. Aynı şekilde, P-R değer aralığı, girilen bütün verilerde gözetilmiştir. Girilen bütün verilerin etiketli değerleri bu değere göre değerlendirilmiştir. Modelin doğruluk oranı da yine bu değer aralıklarına bakılarak oluşturulan etiketler sayesinde sağlanmıştır. Şekil 3.21.'de gerçeklenen sisteme gönderilen EKG verileri gösterilmiştir.

```
1,75,0,190,80,91,250
0,56,1,165,64,81,174
1,54,0,172,95,138,630
1,55,0,175,94,100,202
0,75,0,190,80,88,181
0,13,0,169,51,100,167
0,40,1,160,52,77,129
1,49,1,162,54,78,550
0,44,0,168,56,84,118
0,50,1,167,67,89,130
0,62,0,170,72,102,135
```

Şekil 3.21. Gerçek EKG verileri



Şekil 3.22. Gerçeklenen sistemin zamanlama şeması

EKG verilerinden hastalık teşhisi için kullanılan mimarinin genel işlem adımları Şekil 3.22.'de yer alan zamanlama şemasına göre aşağıda özetlenmiştir.

- Gerçek EKG verileri veri.txt dosyasından okunur ve Apache Kafka üreticiye gönderilir.

- Üretici, verileri "broker" a gönderir, veriler "broker" 'larda bulunan bölümler (partition) içinde tutulur.

- İlgili konu ile ilişiği olan tüketiciler yollanan verileri eşzamanlı olarak alır. Oluşturulan mimaride MongoDB ve Apache Spark olmak üzere iki tane tüketici tanımlanmıştır.

- Tüketici olan MongoDB, verileri ilgili konu üzerinden okur, veri tabanına yazar.

- Diğer tüketici olan Apache Spark, akış şeklinde gelen verileri, kullanıcı tarafından belirlenen (1 saniye) çerçeveler halinde okur.

- Apache Spark içinde gelen verilerin %70'i eğitim seti olarak %30'u test seti olarak kullanılır.

- Model ile test verileri etkileşime sokulur ve model test verilerine dair hastalık teşhisinde bulunur.

- Modelin bulunan teşhislere göre doğruluk oranı gösterilir.

- Akış yapısı kullanarak geliştirilen sistemde, başka veriler yollandığında yukarda anlatılanlar döngü halinde devam etmektedir. Herhangi bir zaman diliminde yollanan EKG verisinin gerçek zamanlı olarak teşhisi koyulup, gösterilmektedir.

Gelen EKG verilerini istenilen formata dönüştürme işlemini gerçekleştirdikten sonra (Şekil 3.23.) şema oluşturup Spark Context'ten türeyen SqlContext'i kullanarak DataFrame oluşturulmuştur (Şekil 3.24.).

```
if(!rdd.isEmpty()){

      JavaRDD<Row> rowRDD = rdd.map((Function<String, Row>) record -> {
          String[] attributes = record.split(",");
          return RowFactory.create(Integer.valueOf(attributes[0]), Integer.valueOf(attributes[1]), Integer.valueOf(attributes[2]),Integer.
      });
```

Şekil 3.23. Gelen verilerin dönüşümü

```
StructType schema = createStructType(new StructField[]{

      createStructField("label", IntegerType, false),
      createStructField("Yas", IntegerType, false),
      createStructField("Cinsiyet", IntegerType, false),
      createStructField("Boy", IntegerType, false),
      createStructField("Kilo", IntegerType, false),
      createStructField("QRS", IntegerType, false),
      createStructField("P-R", IntegerType, false)


      });

// SparkSession spark = JavaSparkSessionSingleton.getInstance(rdd.context().getConf());
 Dataset<Row> msgDataFrame = sqlContext.createDataFrame(rowRDD, schema).toDF();
 msgDataFrame.show();
```

Şekil 3.24. Şema oluşturulması

Gelen veriler şemaya atandıktan sonra makine öğrenmesi olan lojistik regresyon algoritmasından atanmak üzere özellik çıkarmak maksadıyla VectorAssembler() fonksiyonundan geçirilmiştir. Böylelikle makine öğrenmesinin hastalık tahmini yaparken sadece iki tane sütunu işleme tabii tutması sağlanmış olunur (Şekil 3.25.).

```
VectorAssembler assembler = new VectorAssembler()
        .setInputCols(new String[] {"label","P-R"})
        .setOutputCol("features");
        Dataset<Row> finalDS = assembler.transform(msgDataFrame);
        Dataset<Row>[] splits = finalDS.randomSplit(new double[] { 0.7, 0.3 });
        Dataset<Row> trainingData = splits[0];
        Dataset<Row> testData = splits[1];
        trainingData.show();
        testData.show();
    LogisticRegression lr = new LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8).setLabelCol("label");
    System.out.println("LogisticRegression parameters:\n" + lr.explainParams() + "\n");

    LogisticRegressionModel lrModel = lr.fit(trainingData);
    System.out.println("Model 1 was fit using parameters: " + lrModel.parent().extractParamMap());
    // Transform the model, and predict class for test dataset
    Dataset<Row> output = lrModel.transform(testData);
    output.show();
```

Şekil 3.25. Özellik çıkartma

Özellik çıkarıldıktan sonra gelen EKG verileri eğitim seti ve test seti olarak ayrılmıştır. Eğitim seti ile lojistik regresyon modeli oluşturulmuş, test seti ile de modelin hastalığa dair ürettiği sonuçlara ulaşılmıştır. Test sonuçlarıyla modelin bulduğu sonuçlar karşılaştırılmış ve modelin doğruluk oranı hesaplanmıştır.

```
// obtain evaluator.
MulticlassClassificationEvaluator evaluator = new MulticlassClassificationEvaluator()
        .setMetricName("accuracy");

// compute the classification error on test data.
double accuracy = evaluator.evaluate(output);
System.out.println("Test Error = " + (1 - accuracy));
System.out.println("Modelin Dogrulugu = " +  accuracy);
```

Şekil 3.26. Doğruluk oranı

Uygulama çalıştırıldıktan sonraki ekran çıktıları eklenmiştir. Şekil 3.21.'de oluşturulan şemaya atılmış EKG verileri gösterilmiştir. EKG verilerinin test ve eğitim verilerine ayrıldıktan sonraki haline Şekil 3.28.'de yer verilmiş, modelin test sonuçları için tahminlerine ise Şekil 3.29.'da yer verilmiştir.

label	Yas	Cinsiyet	Boy	Kilo	QRS	P-R
1	75	0	190	80	91	250
0	56	1	165	64	81	174
1	54	0	172	95	138	630
1	55	0	175	94	100	202
0	75	0	190	80	88	181
0	13	0	169	51	100	167
0	40	1	160	52	77	129
1	49	1	162	54	78	550
0	44	0	168	56	84	118
0	50	1	167	67	89	130
0	62	0	170	72	102	135
0	45	1	165	86	77	143
0	54	1	172	58	78	155
0	30	0	170	73	91	180
0	44	1	160	88	77	158
0	47	1	150	48	75	132
0	47	0	171	59	82	145
1	46	1	158	58	70	220
0	73	0	165	63	91	154

Şekil 3.27. Verilerin DataFrame'i

```
+-----+---+--------+---+----+---+---+-----------+
|label|Yas|Cinsiyet|Boy|Kilo|QRS|P-R|   features|
+-----+---+--------+---+----+---+---+-----------+
|    0| 13|       0|169|  51|100|167|[0.0,167.0]|
|    0| 30|       0|170|  73| 91|180|[0.0,180.0]|
|    0| 40|       1|160|  52| 77|129|[0.0,129.0]|
|    0| 45|       1|165|  86| 77|143|[0.0,143.0]|
|    0| 47|       0|171|  59| 82|145|[0.0,145.0]|
|    0| 47|       1|150|  48| 75|132|[0.0,132.0]|
|    0| 56|       1|165|  64| 81|174|[0.0,174.0]|
|    0| 62|       0|170|  72|102|135|[0.0,135.0]|
|    1| 46|       1|158|  58| 70|220|[1.0,220.0]|
|    1| 49|       1|162|  54| 78|550|[1.0,550.0]|
|    1| 54|       0|172|  95|138|630|[1.0,630.0]|
|    1| 55|       0|175|  94|100|202|[1.0,202.0]|
|    1| 75|       0|190|  80| 91|250|[1.0,250.0]|
+-----+---+--------+---+----+---+---+-----------+

+-----+---+--------+---+----+---+---+-----------+
|label|Yas|Cinsiyet|Boy|Kilo|QRS|P-R|   features|
+-----+---+--------+---+----+---+---+-----------+
|    0| 44|       0|168|  56| 84|118|[0.0,118.0]|
|    0| 44|       1|160|  88| 77|158|[0.0,158.0]|
|    0| 50|       1|167|  67| 89|130|[0.0,130.0]|
|    0| 54|       1|172|  58| 78|155|[0.0,155.0]|
|    0| 73|       0|165|  63| 91|154|[0.0,154.0]|
|    0| 75|       0|190|  80| 88|181|[0.0,181.0]|
+-----+---+--------+---+----+---+---+-----------+
```

Şekil 3.28. Eğitim ve test veri setleri

```
+-----+---+--------+---+----+---+---+-----------+--------------------+--------------------+----------+
|label|Yas|Cinsiyet|Boy|Kilo|QRS|P-R|   features|       rawPrediction|         probability|prediction|
+-----+---+--------+---+----+---+---+-----------+--------------------+--------------------+----------+
|    0| 44|       0|168|  56| 84|118|[0.0,118.0]|[1.16007887089532...|[0.76134704580028...|       0.0|
|    0| 44|       1|160|  88| 77|158|[0.0,158.0]|[1.16007887089532...|[0.76134704580028...|       0.0|
|    0| 50|       1|167|  67| 89|130|[0.0,130.0]|[1.16007887089532...|[0.76134704580028...|       0.0|
|    0| 54|       1|172|  58| 78|155|[0.0,155.0]|[1.16007887089532...|[0.76134704580028...|       0.0|
|    0| 73|       0|165|  63| 91|154|[0.0,154.0]|[1.16007887089532...|[0.76134704580028...|       0.0|
|    0| 75|       0|190|  80| 88|181|[0.0,181.0]|[1.16007887089532...|[0.76134704580028...|       0.0|
+-----+---+--------+---+----+---+---+-----------+--------------------+--------------------+----------+

Test Error = 0.0
Modelin Dogrulugu = 1.0
```

Şekil 3.29. Hastalığa dair tahminler ve doğruluk oranı

BÖLÜM 4. DENEYSEL ÇALIŞMALAR

Gerçeklenen çalışma üzerinde çeşitli deneysel çalışmalar gerçeklenmiştir. Bunlardan biri, bölüm sayısının hıza veya performansa katkısının olup olmadığıdır. Test için yapılan uygulamaların ilk üç tanesi Apache.spark.mllib kitaplığı kullanılarak yapılmıştır. Gerçeklenen son örnek ise Apache.spark.ml kitaplığı kullanılarak yapılmıştır.

4.1. Uygulama 1

Performansı ölçmek için aynı sistem üzerinde farklı bölüm sayısına sahip iki konu belirlendi. Birinci konuya bir bölüm, ikinci konuya ise üç bölüm ataması yapıldı. Sonuçların daha iyi gözlemlenebilmesi için akış hızı ise bir saniye olarak belirlendi ve veri sayısı olarak 11 kişinin EKG verisi ilgili konuya yollandı.

Üretici çalıştırıldıktan sonra verilerin alıp işlenmesi bölümü üç olarak tanımlanan konuda (Şekil 4.2.) tek adımda sağlanırken, bölümü bir olan konuda (Şekil 4.1.) daha uzun sürede sağlanmıştır.

```
---New RDD with1partitions and0records
-------------------------------------------
Time: 1525968342000 ms
-------------------------------------------

0 2:165 3:63 4:91 5:154 6:392 7:175 8:83 9:73 10:-24 11:61 12:42 13:8
0 2:190 3:80 4:91 5:193 6:371 7:174 8:121 9:16 10:13 11:64 12:2 13:50
0 1:1 2:165 3:64 4:81 5:174 6:401 7:149 8:39 9:25 10:37 11:-17 12:31
0 2:172 3:95 4:138 5:163 6:386 7:185 8:102 9:96 10:34 11:70 12:66 13:
1 2:175 3:94 4:100 5:202 6:380 7:179 8:143 9:28 10:11 11:-5 12:20 13:
0 2:190 3:80 4:88 5:181 6:360 7:177 8:103 9:-16 10:13 11:61 12:3 13:5
6
0.0<<HASTA DEĞİL>>
0.0<<HASTA DEĞİL>>
0.0<<HASTA DEĞİL>>
0.0<<HASTA DEĞİL>>
1.0<<HASTA>>
0.0<<HASTA DEĞİL>>
Accuracy = 1.0
---New RDD with1partitions and6records
0 2:165 3:63 4:91 5:154 6:392 7:175 8:83 9:73 10:-24 11:61 12:42 13:8
0 2:190 3:80 4:91 5:193 6:371 7:174 8:121 9:16 10:13 11:64 12:2 13:50
0 1:1 2:165 3:64 4:81 5:174 6:401 7:149 8:39 9:25 10:37 11:-17 12:31
0 2:172 3:95 4:138 5:163 6:386 7:185 8:102 9:96 10:34 11:70 12:66 13:
1 2:175 3:94 4:100 5:202 6:380 7:179 8:143 9:28 10:11 11:-5 12:20 13:
0 2:190 3:80 4:88 5:181 6:360 7:177 8:103 9:-16 10:13 11:61 12:3 13:5
-------------------------------------------
```

Şekil 4.1. Bir bölüme sahip konu üzerinden dağıtım

```
1 2:170 3:72 4:102 5:300 6:401 7:156 8:83 9:72 10:71 11:68 12:72
1 1:1 2:165 3:86 4:77 5:500 6:373 7:150 8:65 9:12 10:37 11:49 12:
0 1:1 2:172 3:58 4:78 5:155 6:382 7:163 8:81 9:-24 10:42 11:41 12
1 2:170 3:73 4:91 5:780 6:355 7:157 8:104 9:68 10:51 11:60 12:63
0 1:1 2:160 3:88 4:77 5:158 6:399 7:163 8:94 9:46 10:20 11:45 12:
11
0.0<<HASTA DEĞİL>>
0.0<<HASTA DEĞİL>>
0.0<<HASTA DEĞİL>>
0.0<<HASTA DEĞİL>>
1.0<<HASTA>>
0.0<<HASTA DEĞİL>>
0.0<<HASTA DEĞİL>>
1.0<<HASTA>>
0.0<<HASTA DEĞİL>>
1.0<<HASTA>>
0.0<<HASTA DEĞİL>>
Accuracy = 0.9090909090909091
---New RDD with1partitions and5records
1 2:170 3:72 4:102 5:300 6:401 7:156 8:83 9:72 10:71 11:68 12:72
1 1:1 2:165 3:86 4:77 5:500 6:373 7:150 8:65 9:12 10:37 11:49 12:
0 1:1 2:172 3:58 4:78 5:155 6:382 7:163 8:81 9:-24 10:42 11:41 12
1 2:170 3:73 4:91 5:780 6:355 7:157 8:104 9:68 10:51 11:60 12:63
0 1:1 2:160 3:88 4:77 5:158 6:399 7:163 8:94 9:46 10:20 11:45 12:
-------------------------------------------
```

Şekil 4.2. Üç bölüme sahip konu üzerinden dağıtım

Yine aynı uygulamada, Apache Spark'ın web ara yüzünden gerekli analizlere bakılmış görüntüleri aşağı eklenmiştir. Şekil 4.3.'de bölüm sayısı bir olan konu yer alırken, Şekil 4.4.'de ise bölüm sayısı üç olan konunun sonuçları yer almaktadır. Şekillerde görüldüğü gibi bölümü üç olan konu ilk saniyede (23:39:40) çok daha fazla veri işleyerek hız anlamında öne geçmiştir.

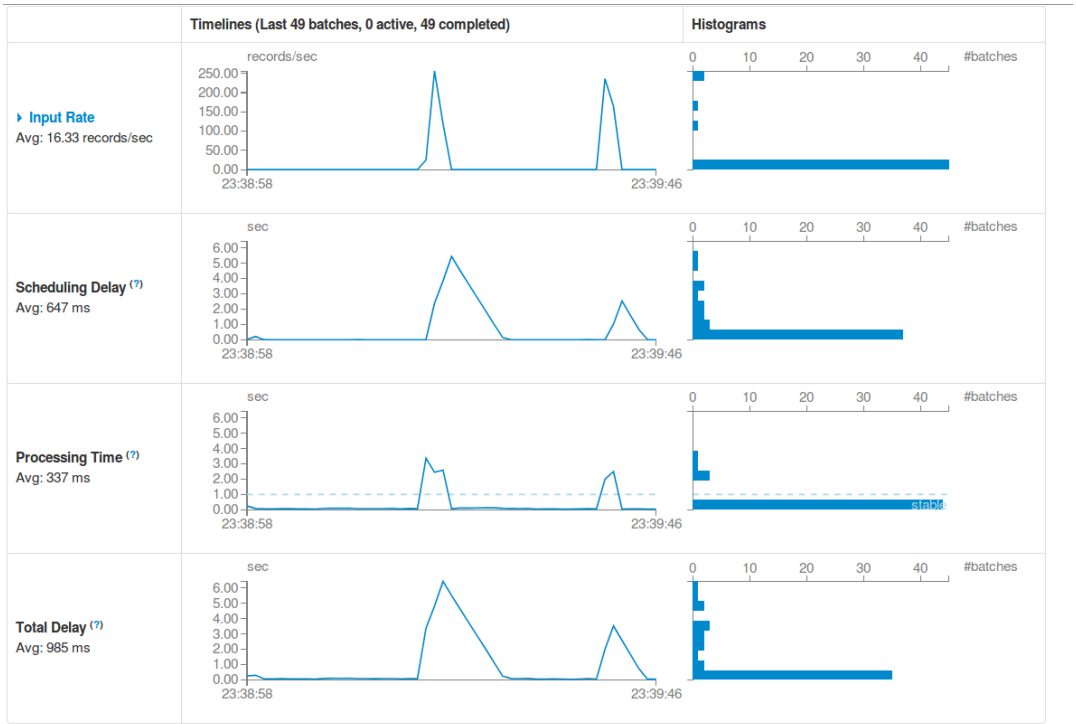
2018/05/10 23:39:41	164 records	1 s	3 s
2018/05/10 23:39:40	236 records	2 ms	2 s
2018/05/10 23:39:39	0 records	4 ms	55 ms

Şekil 4.3. Bir bölüme sahip konu üzerinden dağıtım

2018/05/10 23:42:25	19 records	1 s	2 s
2018/05/10 23:42:24	381 records	14 ms	2 s
2018/05/10 23:42:23	0 records	9 ms	74 ms

Şekil 4.4. Üç bölüme sahip konu üzerinden dağıtım

Şekil 4.5.'de tek bölüme sahip konunun çeşitli histogramları ve veri akışlarının ara yüzü, Şekil 4.6.'da ise üç bölüme sahip konunun çeşitli histogramları ve veri akışlarının ara yüzü gösterilmiştir.



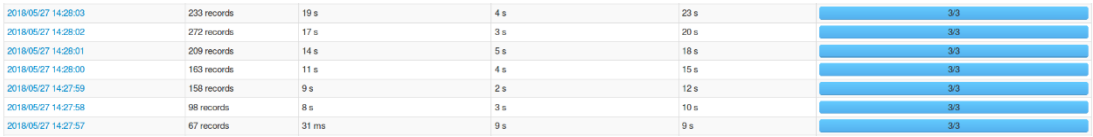
Şekil 4.5. Bir bölüme sahip konu üzerinden dağıtım

Şekil 4.6. Üç bölüme sahip konu üzerinden dağıtım

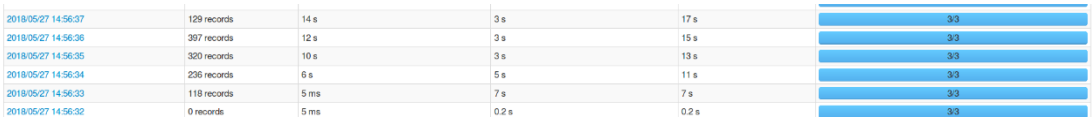
Yapılan test düzeneğinde, bir konuya sahip senaryo 1'de, 400 kişinin EKG verisi gönderilmiş ve ilk saniyede 236 verinin işlendiği gözlemlenmiştir. Üç bölüme sahip senaryo 2'de ise yollanan 400 EKG verisinin ilk saniyede 381 tanesinin işlendiği gözlemlenmiştir. Aynı test düzeneğinde senaryo 1'de beklenen toplam gecikme 985 ms iken, senaryo 2'de 803 ms olduğu sayısal veriler ve histogramlar halinde gösterilmiş, beklenen neticeye ulaşılmıştır.

4.2. Uygulama 2

Geliştirilen sistem üzerinde yürütülen başka bir deneysel çalışmada ise iki tane broker oluşturulmuştur. Sistem üzerinde biri iki bölüm diğeri altı bölüm olmak üzere iki adet konu tanımlanmıştır. Gönderilen veri miktarı ise 1200 olarak belirlenmiş ve performans sıralamaları gözlemlenmiştir.

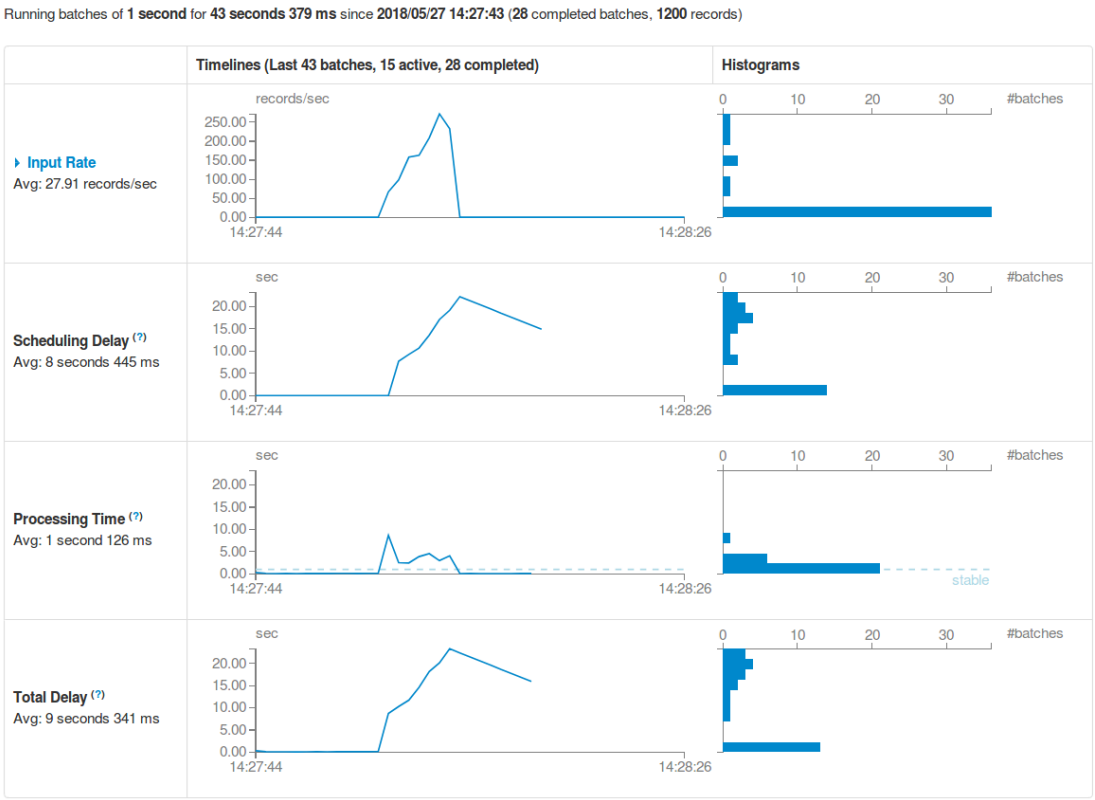


Şekil 4.7. İki bölüme sahip konu üzerinden dağıtım

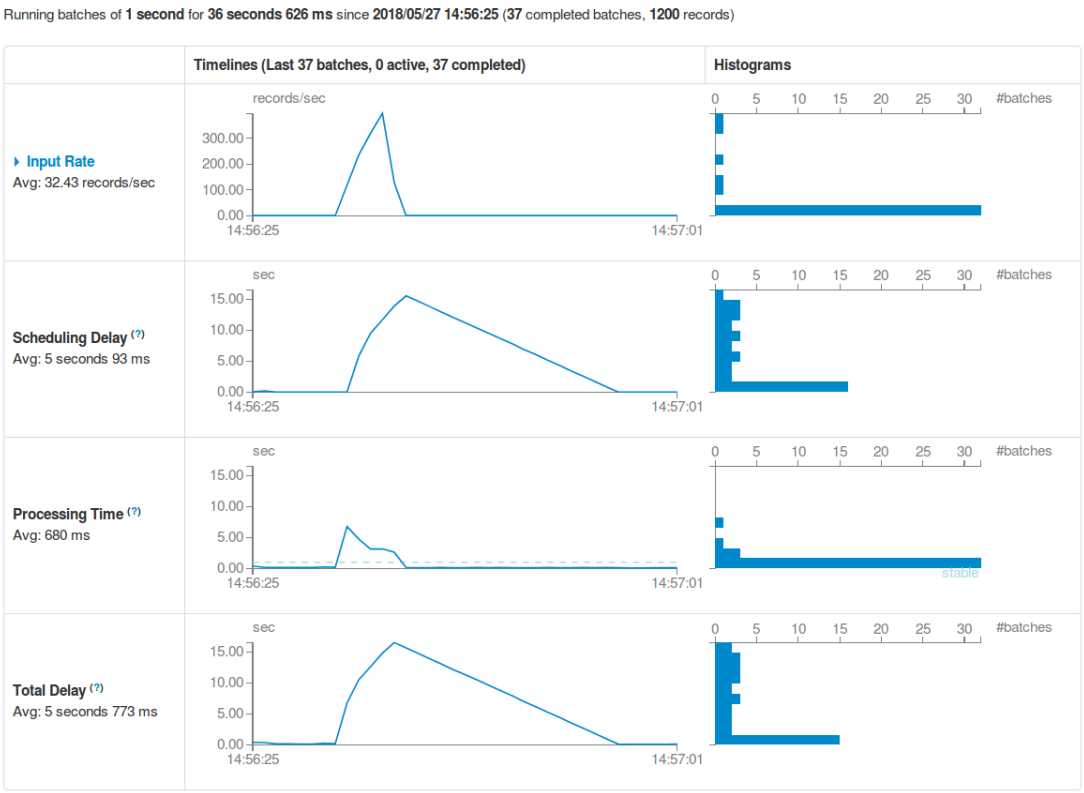


2018/05/27 14:56:37	129 records	14 s	3 s	17 s	3/3
2018/05/27 14:56:36	397 records	12 s	3 s	15 s	3/3
2018/05/27 14:56:35	320 records	10 s	3 s	13 s	3/3
2018/05/27 14:56:34	236 records	6 s	5 s	11 s	3/3
2018/05/27 14:56:33	118 records	5 ms	7 s	7 s	3/3
2018/05/27 14:56:32	0 records	5 ms	0.2 s	0.2 s	3/3

Şekil 4.8. Altı bölüme sahip konu üzerinden dağıtım



Şekil 4.9. İki bölüme sahip konu üzerinden dağıtım

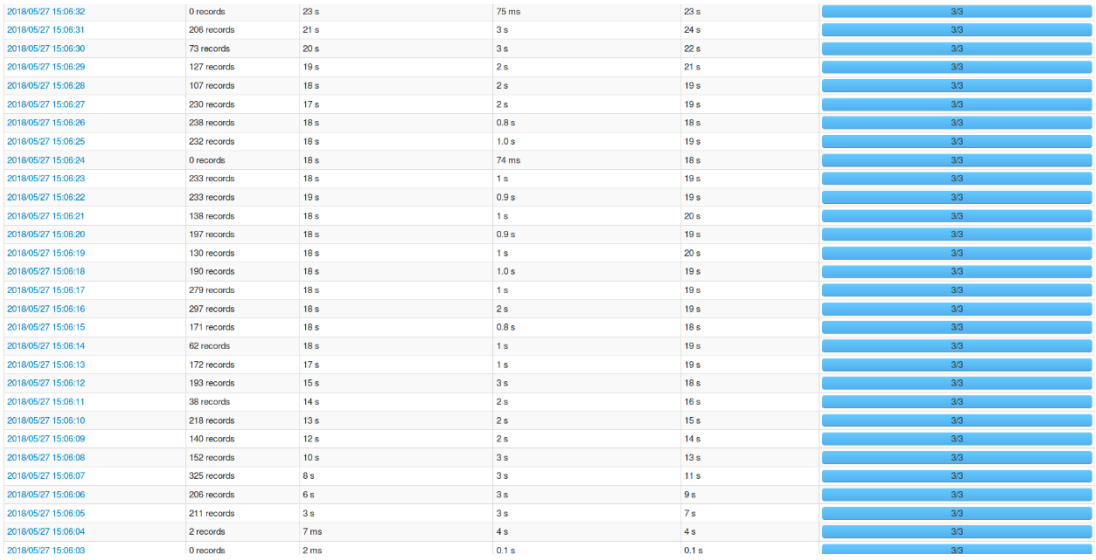


Şekil 4.10. Altı bölüme sahip konu üzerinden dağıtım

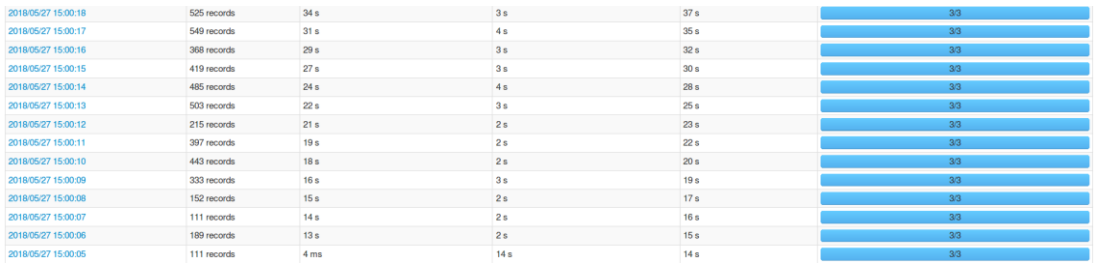
Yukarıdaki sayısal çıktıları ve histogramları değerlendirecek olursak eğer altı bölüm sayısına sahip konunun iletim hızının daha yüksek olduğunu gecikmenin daha az olduğunu gözlemleyebiliriz. İki bölüme sahip konunun giriş oranı (input rate) 27.91 kayıt/saniye, toplam gecikmesi 9 saniye 341 ms iken altı bölüme sahip konunun giriş oranı 32.43 kayıt/saniye, toplam gecikmesi 5 saniye 773 ms'dir. Bu veriler de bölüm sayısının artışının hız artışını nasıl olumlu yönde etkilediğini açık bir şekilde ifade etmektedir.

4.3. Uygulama 3

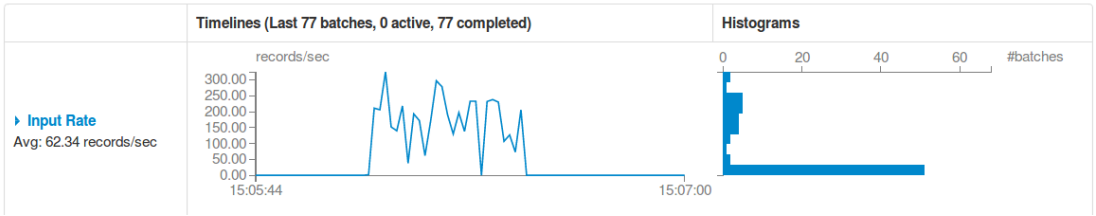
Geliştirilen sistem üzerinde yürütülen diğer bir çalışmada ise iki tane broker oluşturulmuştur. Sistem üzerinde biri bir bölüm diğeri altı bölüm olmak üzere iki adet konu tanımlanmıştır. Gönderilen veri miktarı ise 4800 olarak belirlenmiş ve performans sıralamaları gözlemlenmiştir.



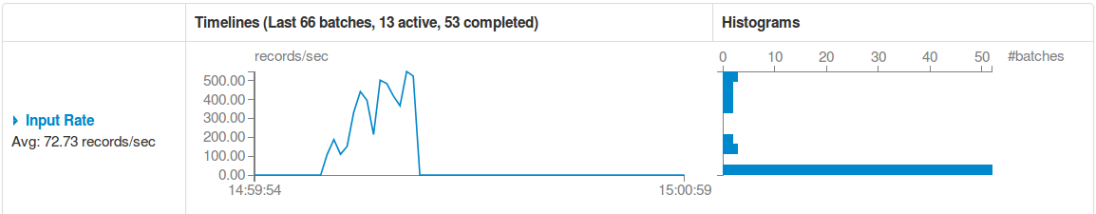
Şekil 4.11. Bir bölüme sahip konu üzerinden dağıtım



Şekil 4.12. Altı bölüme sahip konu üzerinden dağıtım



Şekil 4.13. Bir bölüme sahip konu üzerinden dağıtım



Şekil 4.14. Altı bölüme sahip konu üzerinden dağıtım

Bir bölüme sahip konunun giriş oranı (input rate) 62.34 kayıt/saniye ve 4800 veriyi işleme süresi 28 saniye iken 6 bölüme sahip konunun giriş oranı (input rate) 72.73 kayıt/saniye ve 4800 veriyi işleme süresi 14 saniyedir. Bu örnekte bölüm sayısının artışının hıza oranı incelemesi yapılmış ve beklenen sonuca uygulamalı olarak varılmıştır.

4.4. Uygulama 4

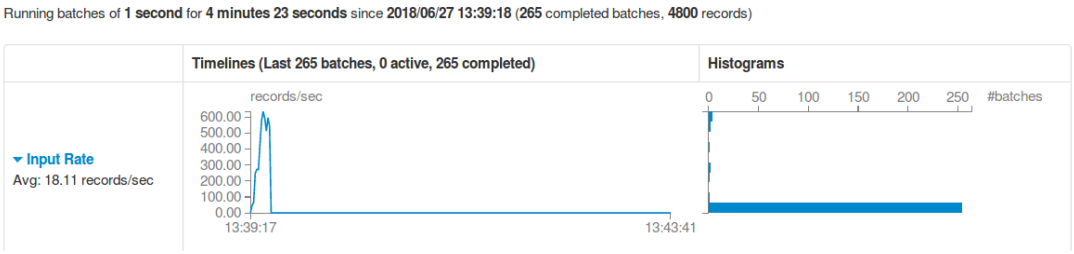
Geliştirilen sistem üzerinde yürütülen diğer uygulamada ise iki tane broker oluşturulmuştur. Sistem üzerinde biri iki bölüm diğeri altı bölüm olmak üzere iki adet konu tanımlanmıştır. Gönderilen veri miktarı ise 4800 olarak belirlenmiş ve performans sıralamaları gözlemlenmiştir

2018/06/27 13:33:46	156 records	46 s	6 s	52 s
2018/06/27 13:33:45	589 records	43 s	4 s	47 s
2018/06/27 13:33:44	560 records	40 s	4 s	44 s
2018/06/27 13:33:43	602 records	37 s	4 s	41 s
2018/06/27 13:33:42	553 records	33 s	5 s	38 s
2018/06/27 13:33:41	487 records	30 s	4 s	34 s
2018/06/27 13:33:40	616 records	28 s	3 s	31 s
2018/06/27 13:33:39	506 records	25 s	3 s	29 s
2018/06/27 13:33:38	204 records	23 s	3 s	26 s
2018/06/27 13:33:37	328 records	21 s	3 s	24 s
2018/06/27 13:33:36	198 records	17 s	5 s	22 s
2018/06/27 13:33:35	1 records	3 ms	18 s	18 s

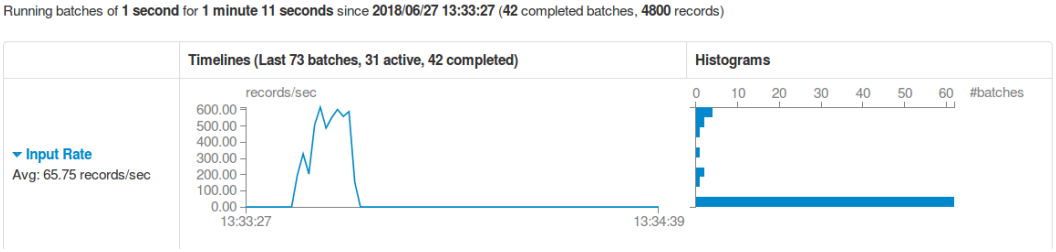
Şekil 4.15. İki bölüme sahip konu üzerinden dağıtım

2018/06/27 13:31:51	340 records	21 s	5 s	26 s
2018/06/27 13:31:50	615 records	18 s	4 s	22 s
2018/06/27 13:31:49	510 records	15 s	4 s	19 s
2018/06/27 13:31:48	630 records	11 s	5 s	16 s
2018/06/27 13:31:47	563 records	9 s	3 s	12 s
2018/06/27 13:31:46	506 records	9 s	1 s	10 s
2018/06/27 13:31:45	414 records	9 s	1 s	10 s
2018/06/27 13:31:44	389 records	9 s	1 s	10 s
2018/06/27 13:31:43	256 records	8 s	2 s	10 s
2018/06/27 13:31:42	328 records	4 s	4 s	9 s
2018/06/27 13:31:41	249 records	3 ms	5 s	5 s

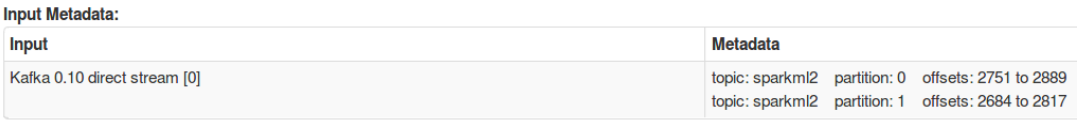
Şekil 4.16. Altı bölüme sahip konu üzerinden dağıtım



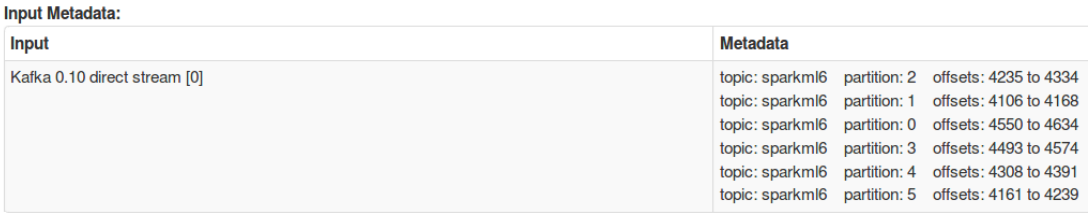
Şekil 4.17. İki bölüme sahip konu üzerinden giriş oranı



Şekil 4.18. Altı bölüme sahip konu üzerinden giriş oranı



Şekil 4.19. İki bölüme sahip konunun veri dağılımı



Şekil 4.20. Altı bölüme sahip konunun veri dağılımı

İki bölüme sahip konunun giriş oranı (input rate) 18.11 kayıt/saniye ve 4800 veriyi işleme süresi 12 saniye iken 6 bölüme sahip konunun giriş oranı (input rate) 65.75 kayıt/saniye ve 4800 veriyi işleme süresi 11 saniyedir. Bu örnekte bölüm sayısının artışının hıza oranı incelemesi yapılmış ve beklenen sonuca uygulamalı olarak varılmıştır.

BÖLÜM 5. TARTIŞMA VE SONUÇ

Hastalık teşhisinde hızın önemi yadsınamaz bir gerçektir. Mevcut sistemlerin yetersizliği ve teknoloji alanının bu konulardaki ilerlemesi, biyomedikal alanının geniş çapta kullanılmasının önünü açmıştır. Birden çok kişinin hastalık teşhisinin gerçeklenmesi için çoklu karar mekanizması içeren sistemler kullanılmalıdır. Bu sistemlerin performans açısından en gözde olanları, dağıtık mesajlaşma konusunda Apache Kafka, büyük veriyi gerçek zamanlı işleme konusunda Apache Spark'tır. Önerdiğimiz mimaride, hastalık teşhisi için gelen bireylerin EKG verileri, mesaj dağıtıcısı olarak görev yapan Apache Kafka sayesinde aynı anda hem NoSQL veri tabanı MongoDB'de tutulmuş hem de Apache Spark'a gönderilmiştir. Apache Spark'a gelen EKG verileri Apache Spark MLlib'in sunduğu lojistik regresyon algoritmasından geçirilmiş ve sonuçlar, sağlık alanında söz konusu olan gecikmelerin önüne geçebilmek adına gerçek zamanlı sistem kullanılarak, anlık olarak başarıyla gösterilmiştir. Oluşturulan modelin ise çıktılar sonucunda %100 doğruluk oranı ile çalıştığı gözlenmiştir.

Mevcut sistem üzerinde çeşitli testler yapılmış, belli parametreler dahilinde ön işlemden geçirilmiştir. Bu testlerden biri bölüm sayısının farklı belirlenmesidir. Apache Kafka'da tanımlanan bölüm (partition) kısmı değiştirilip farklı veri akışlarına sebep olacak konular (topic) tanımlanmış ve beklendiği gibi hız alanında faklı sonuçlar elde edilmiştir.

İlerleyen kademelerde projenin gerçek dünya problemleriyle daha fazla örtüşmesi maksadıyla, donanımsal cihazlardan verilerin anlık alınması ve soket yardımıyla donanımsal sistemle mevcut sistemin haberleştirilmesi planlanmaktadır.

Gerçekleştirilen çalışma kapsamında, gerçek zamanlı büyük veriyi işlemede Apache Spark ve Apache Kafka teknolojilerinin etkin bir şekilde kullanılabilir olduğu sonucuna varılabilir.

KAYNAKLAR

[1] http://mungeol-heo.blogspot.com.tr/2015/01/kafka-vs-rabbitmq-vs-activemq.html, Erişim Tarihi: 05.04.2018.

[2] http://stackoverflow.com/questions/29774223/can-kafka-producer-read-log-files, Erişim Tarihi: 03.04.2018.

[3] http://kafka.apache.org/documentation/, Erişim Tarihi: 06.05.2018.

[4] B. K. Sunny, P. S. Janardhanan, A. B. Francis, and R. Murali, "Implezmentation of a Self-Adaptive Real Time Recommendation System using Spark Machine Learning Libraries," pp. 1–7, 2017.

[5] K. Wang, J. Fu, and K. Wang, "SPARK – A Big Data Processing Platform for Machine Learning," 2016 Int. Conf. Ind. Informatics - Comput. Technol. Intell. Technol. Ind. Inf. Integr., pp. 48–51, 2016.

[6] D. Harnie et al., "Scaling Machine Learning for Target Prediction in Drug Discovery using Apache Spark," 2015 15th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput., pp. 871–879, 2015.

[7] L. R. Nair, S. D. Shetty, and S. D. Shetty, "Applying spark based machine learning model on streaming big data for health status prediction," Comput. Electr. Eng., vol. 0, pp. 1–7, 2017.

[8] S. Chidambaram, S. Saraswati, R. Ramachandra, J. B. Huttanagoudar, and N. Hema, "JVM Characterization Framework for Workload Generated as per Machine Learning Benckmark and Spark Framework," pp. 1598–1602, 2016.

[9] Apache Kafka - A distributed streaming platform. Available at: http://kafka.apache.org, Erişim Tarihi: 07.03.2018.

[10] https://streaml.io/blog/pulsar-segment-based-architecture/, Erişim Tarihi: 08.04.2018.

[11] https://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#3fd7eca17b1e, Erişim Tarihi: 30.05.2018.

[12] https://ahmetsedef.wordpress.com/tag/lojistik-regresyon-analizi-nedir/, Erişim Tarihi: 30.05.2018.

[13] Z. Zhu, C. Lin, X. Zhang, K. Wang, J. Xie, and S. Wei, "Evaluation of geological risk and hydrocarbon favorability using logistic regression model with case study," Mar. Pet. Geol., vol. 92, no. February, pp. 65–77, 2018.

[14] A. Brenning, "Spatial prediction models for landslide hazards : review , comparison and evaluation," pp. 853–862, 2005.

[15] L. Ganz, "Electrocardiography," Goldman's Cecil Med. Twenty Fourth Ed., vol. 1, pp. 272–278, 2011.

[16] https://spark.apache.org/, Erişim Tarihi: 30.05.2018.

[17] https://archive.ics.uci.edu/ml/datasets/arrhythmia, Erişim Tarihi: 15.03.2018.

[18] http://nverma-tech-blog.blogspot.com/2015/12/apache-kafka-multibroker-partitioning.html, Erişim Tarihi: 27.06.2018

[19] https://www.tutorialspoint.com/, Erişim Tarihi: 27.06.2018

[20] https://hackernoon.com/introduction-to-machine-learning-algorithms-logistic-regression-cbdd82d81a36, Erişim Tarihi: 27.06.2018

[21] http://dergipark.gov.tr/download/article-file/252030, Erişim Tarihi: 27.06.2018

[22] https://www.quora.com/What-exactly-is-a-logistic-regression-algorithm-in-machine-learning-What-are-its-applications, Erişim Tarihi: 27.06.2018

ÖZGEÇMİŞ

Nurbanu Oğur, 09.01.1993'de Sakarya'da doğdu. İlk, orta, lise ve üniversite eğitimini Sakarya'da tamamladı. 2011 yılında Ali Dilmen Anadolu Lisesi'nden mezun oldu. 2011 yılında başladığı Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nü 2015 yılında bitirdi. 2015 yılında Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nde yüksek lisans eğitimine başladı. 2017 yılında Sakarya Üniversitesi'nde Araştırma Görevlisi olarak çalışmaya başladı, akabinde yüksek lisans eğitimine Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nde devam etti. Halen Sakarya Ünivesitesi Bilgisayar Mühendisliği Bölümü'nde Araştırma Görevlisi olarak görev yapmaktadır.