T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİR ABONENİN ELEKTRİK TÜKETİM VERİLERİNİN VERİ MADENCİLİĞİ YÖNTEMLERİ İLE ANALİZ EDİLMESİ VE DOĞRU TARİFENİN BELİRLENMESİ

YÜKSEK LİSANS TEZİ

Seda BALTA

Enstitü Anabilim Dalı	**:**	**BİLGİSAYAR MÜHENDİSLİĞİ**
Tez Danışmanı	**:**	**Doç. Dr. Cüneyt BAYILMIŞ**

Temmuz 2020

BEYAN

Yüksek Lisans tezi olarak sunduğum bu çalışmanın içindeki tüm verilerin akademik kurallar çerçevesinde hazırlandığını, tezde yer alan gerek görsel gerekse yazılı tüm bilgilerin tarafımca elde edildiğini, başkalarının çalışmalarından yararlanılması durumunda ise bütün bilgi ve yorumları bilimsel etik kurallar çerçevesinde kaynak gösterdiğimi beyan ederim.

Seda BALTA

7/9/2020

TEŞEKKÜR

Bu çalışmanın yürütülmesi sırasında her konuda hiç bir zaman desteğini üzerimden eksik etmeyen, bilgi ve tecrübelerini paylaşmaktan kaçınmayan, her zaman yol gösterici ve teşvik edici rolde olan, değerli danışman hocam Doç Dr. Cüneyt BAYILMIŞ'a teşekkürlerimi sunarım.

Yoğun çalışmalarım sırasında motive edici konuşmaları ve gösterdikleri sabır için aileme, IoT ile ilgili çalışmalarımda yardımcı olan Khaled Wagdy'e, yine bilgi ve deneyimlerinden yararlandığım sayın hocam Dr. Öğr. Üyesi Süleyman EKEN'e teşekkür ederim.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

IOT	: Internet of Things
MLLIB	: Machine Learning Library
HDFS	: Hadoop distributed file system
EKG	: Elektrokardiyogram
WBAN	: Wireless Body Area Networks
Wi-Fi	: Wireless Fidelity
NFC	: Near Field Communication
UCODE	: Ubiquitous code
UUID	: Universal Unique Identifier
RFID	: Radio Frequency Identification
MQTT	: Message Queuing Telemetry Transport
COAP	: Constrained Application Protocol
AMQP	: Advance Message Queue Protocol
DDS	: Data Distribution Service
XMPP	: Extensible Messaging and Presence Protocol
SOAP	: Simple Object Access Protocol
TCP	: Transmission Control Protocol
UDP	: User Datagram Protocol
TLS	: Transport Layer Security
SSL	: Güvenli Soket Katmanı
DTLS	: Datagram Transport Layer Security
ETL	: Extract Transform Load
IDE	: Integrated Development Environment
RDD	: Resilient distributed dataset
API	: Application Programming Interface
SQL	: Structured Query Language

JSON	: JavaScript Object Notation
ROC	: Receiver Operating Characteristic
SVM	: Support Vector Machine

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Nesnelerin interneti, akıllı priz, makine öğrenmesi, elektrik tüketimi, gerçek zamanlı analiz

Her geçen gün hayatımızda daha çok yer edinen teknolojinin gelişmesi ile birlikte büyük veri hacmi, depolanması ve analizi gibi sorunlar ortaya çıkmaktadır. Teknolojinin gelişimi aynı zamanda elektriğe olan bağımlılığı da arttırmaktadır. Elektrik tüketiminin artması maliyet sorununu ortaya çıkarmaktadır. Bu sebeplerden dolayı bireyler optimum fayda-maliyet ilişkisini aramaktadır. Optimum fayda-maliyet ilişkisini bulabilmek için elde edilen büyük veriyi anlamlı hale getirmek gerekmektedir. Bu nedenle büyük veri analitiği için Apache Spark özel olarak geliştirilmiş platformdur. Apache Spark büyük veriler üzerinde çeşitli makine öğrenmesi algoritmaları kullanarak veri analizi yapılmasına olanak sağlayan bütünleşik bir hesaplama motorudur. Diğer bir veri analitiği sistemi olan Apache Kafka ise sistemler arasındaki entegrasyon problemlerine bir çözüm olarak geliştirilen, üreticiler ve tüketiciler arasında gerçek zamanlı iletişime olanak sağlayan bir mesajlaşma protokolüdür.

Bu tezde, öncelikle kullanıcıların elektrik tüketim bilgilerini incelemek ve kullanıcılardan elde edilen tüketim verilerinin analizi için yazılım kontrollü akıllı priz geliştirilmiştir. Geliştirilen bu akıllı priz aracılığıyla elde edildiği varsayılan 5000 adet kullanıcının elektrik tüketim verisi Türkiye'de varolan elektrik faturalandırma sistemine göre incelenerek, kullanıcıya en uygun tüketici tipi seçimi yapılmıştır. Tüketici tipi seçimi yapılırken, aynı veri seti ile Lojistik Regresyon kullanılarak iki farklı makine öğrenmesi uygulaması geliştirilmiştir. Uygulama-1'de Spark Structured Streaming ile gerçek zamanlı veri analizi yapılmaktadır. Uygulama-2'de ise Apache Kafka ile gerçek zamanlı veri analizi yapılmaktadır. Bu iki uygulama arasında yaklaşım ve kullanılan teknolojiler açısından farklılıklar bulunmaktadır. Uygulamalara göre elde edilen sonuçlar, önerilen mimarinin gerçek zamanlı veri analizi için kullanılabilir olduğunu göstermektedir.

ANALYSIS OF ELECTRIC CONSUMPTION DATA BY DATA MINING METHODS AND DETERMINING THE RIGHT TARIFF

SUMMARY

Keywords: IoT, smart plug, machine learning, electric consumption, real-time analysis

With the development of technology, which takes more place in our lives day by day, problems such as big data volume, storage and analysis arise. The development of technology also increases dependence on electricity. The increase in electricity consumption raises the cost problem. For these reasons, individuals seek the optimum cost-benefit relationship. To find the optimum cost-benefit relationship, it is necessary to make the big data obtained meaningful. Therefore, Apache Spark is a specially developed platform for big data analytics. Apache Spark is an integrated calculation engine that enables data analysis on large data using various machine learning algorithms. Apache Kafka, another data analytics system, is a messaging protocol developed as a solution to integration problems between systems, enabling real-time communication between producers and consumers.

In this thesis, firstly, software controlled smart plug was developed to examine the electricity consumption information of the users and to analyze the consumption data obtained from the users. Developed this smart plug is obtained through the default data 5000's by examining the power consumption by the electricity billing system existing in Turkey are made optimum consumer type selection to the user. While selecting the consumer type, two different machine learning applications were developed by using Logistic Regression with the same data set. Real-time data analysis is performed with Spark Structured Streaming in Application-1. In application-2, real-time data analysis is performed with Apache Kafka. There are differences between these two applications in terms of approaches and technologies. Results obtained according to applications show that the proposed architecture is available for real-time data analysis.

BÖLÜM 1. GİRİŞ

Günümüzde mobil cihazlar, Nesnelerin İnternet'i (Internet of Things, IoT) gibi uygulamaların yaygınlaşması, internet üzerindeki veri kaynaklarında çeşitliliğe ve verinin artış hızına sebep olmuştur. Verinin sürekli artması büyük veri kavramının ortaya çıkmasına neden olmuştur. Büyük veri kişilerin yaşama, çalışma ve düşünce tarzlarını değiştirerek hayatlarını kolaylaştıracak bir dönüşüm olarak adlandırılmaktadır. Giderek artan bu büyük verinin üretilmesi, depolanması ve analiz edilmesi büyük bir öneme sahiptir. Büyük veri, makul sürelerde işlenebilir ve anlamlaştırılabilir olmalıdır. Bu amaçlar, büyük verinin özellik ve sınırlamalarını dikkate alan dağıtık dosya sistemi, mesaj dağıtıcı, gerçek zamanlı veri işleme araçları, ilişkisel olmayan veritabanları gibi Büyük Veri Teknolojileri geliştirilmiştir. Bu teknolojilerin, sağlıktan finans ve pazar analizine, afet durumlarından uzaktan izleme ve yönetim sistemlerine kadar çok çeşitli alanlarda kullanımı gün geçtikçe artmaktadır.

1.1. Literatür Özeti

Büyük verilerin üretilmesi, depolanması ve analiz edilmesi büyük bir öneme sahiptir. Büyük veri üzerinde analiz işlemleri için yapılan sınıflandırma ve kümeleme işlemleri zaman alıcı olabilmektedir. Erdem ve ark. yaptığı çalışmada büyük veri analiz işlemleri için Apache Spark'ın MLlib kütüphanesinden yararlanarak çeşitli algoritmaları kullanmış ve algoritmaların çalışma sürelerini karşılaştırmıştır [1].

Hayatımızın her alanına giren internet ile birlikte hızlıca ürettiğimiz verilerin anlamlı bir bütün haline gelmesi konusunda çeşitli sektörlerde değişik analizler yapılmaktadır. Çetinkaya, Hortonworks'un Hadoop platformunda MapReduce ve HDFS ile Pazar sepeti analizi üzerinde çalışmıştır. Yapılan bu çalışmalar neticesinde

müşterilerin alışveriş alışkanlıklarına göre yeni müşteri kanalları oluşturulmuştur. Bu müşteri kanalları oluşturulurken müşterilere ürün önerme, raf dizilimi, kampanya modellerinin belirlenmesi ve müşteri memnuniyeti ölçümü gibi maddeler amaçlanmıştır [2]. Yapılan çalışmada paralel olarak yapılan bu analizlerin belirli bir düğüm sayısına kadar kazanım sağladığı sonucuna ulaşmıştır.

Sağlık personeline yardımda bulunmak amacıyla biyomedikal görüntüler, klinik testler, sinyaller gibi verilerin analiz edilmesi çalışmaları gün geçtikçe önem kazanmaktadır. Hastaların ilgili verileri Apache Kafka, Hadoop ve NoSQL Cassandra gibi teknolojilerle gerçek zamanlı olarak işlenip karar verilmesi sağlanmaktadır [3]. Oğur [4], Sağlık alanında büyük veri analizleri ile ilgili olarak Apache Spark, Kafka, MongoDB teknolojilerini kullanarak gelen EKG verilerini gerçek zamanlı olarak analiz etmiş ve bu analiz sonucunda ilgili hastalara hasta veya hasta değil şeklinde teşhis koymuştur.

İnternet üzerinden çevrimiçi bisiklet, motosiklet parçaları satışı yapan Bikeberry.com firması müşterilerinden biri ile kar marjını arttırmak amacıyla anlaşma yapmıştır. Öncelikle web tarayıcılar, geçmiş alışverişler ve daha başka şekillerde büyük bir müşteri verisi elde edilmiştir. [5] numaralı çalışmada, elde edilen bu verilere göre toplamda 5 ayrı teklif oluşturulmuştur. Müşterilerin alışkanlıklarına göre bu tekliflerden en uygun olanı müşteriye sunulmuş ve sonuçta satışlar %133 oranında arttırılmıştır.

Büyük veriyi oluşturan en kapsamlı kaynaklardan biri sosyal medyadır. İnsanlar duygularını çoğunlukla sosyal medya uygulamaları üzerinden ifade etme isteği duymaktadırlar. Bu istekler doğrultusunda milyonlarca sosyal medya kullanıcıları olduğu göz önünde bulundurulduğunda devasa bir büyük veri akışından söz edilebilir. Bu sosyal medya uygulamalarından birisi de Twitter'dır. [6] numaralı çalışmada, Suudi kullanıcıların farklı mesele ve konulardaki eğilimlerini bilmek için uygulama üzerinde yaptıkları yorumlar gerçek zamanlı olarak analiz edilmiştir. Suudi lehçesi göz önünde bulundurularak yapılan bu çalışmada Spark ve Flume kullanılmıştır. Bu çalışmada gerçek zamanlı sorulara hemen yanıt vermek

amaçlanmıştır. Bu amaç doğrultusunda Suudi lehçesi duygu analizi için çözüm önerilmiştir.

Yapılan bir başka çalışmada [7], yine Twitter uygulamasından elde edilen veriler Spark veri işleme motoru kullanılarak gerçek zamanlı olarak analiz edilmiştir. İşlenen veriler sorgulanmıştır. Örneğin son 10 dakika içerisinde en çok konuşulan diller, aranılacak kelimeye uygun olarak o kelime ile ilişkili toplam twit sayısı gibi bilgiler elde edilmiştir.

Teknoloji ile etkileşimi olan insanların çoğu, şüphesiz internet üzerinden de alışveriş yapmaktadır. Ancak kredi kartı dolandırıcıları sadece rakamları değiştirerek kredi kartlarını kullanmaya çalışmaktadır. Kredi kartı numarasını kabul eden her sistemin doğru kart numarasını tanımlayacak bir mekanizması olmalıdır. Bu konuyla ilgili olarak kredi kartı dolandırıcılıklarını önlemek için çalışma gerçekleştirmiştir [8]. Bu çalışmada K-means algoritması geliştirilerek sahte kredi kartı numaralarının tespiti için bir sistem önerilmiştir.

Nesnelerin İnternet'i teknolojisi için geniş ve çeşitli uygulama ortamları sağlanmaktadır. Otomatik izleme, kontrol, yönetim ve bakım işlemleri gibi çeşitli amaçlara yönelik geliştirilen bu cihazlardan değişik yapılarda büyük veriler elde edilmektedir. 2014 yapılan bir çalışmada IoT nesneleri aracılığıyla kayıt altına alınan verilerin analitiği sonucunda elde edilen bilgilerin gelecekteki çalışmalara ışık tutacağı öngörülmüştür [9].

Çeşitli sensörler aracılığıyla geliştirilen IoT birimlerinden elde edilen verilerin analizi üzerine literatürde çok sayıda çalışma bulunmaktadır. Bu analizler afet yönetimi, sağlık takibi, enerji tasarrufu gibi konuları gündeme getirmiştir. Kablosuz vücut alan ağı (WBAN), çeşitli sensörler ile uzaktaki hastaların hayati bilgilerini izler ve baz istasyonuna gönderir. Baz istasyonunda büyük veriler toplanır. WBAN aracılığı ile toplanan bu büyük verileri Apache Spark veri işleme moturu ile gerçek zamanlı olarak analiz ederek sağlık bakım servislerini iyileştirmeyi amaçlamıştır [10].

Yine geliştirilen bir IoT birimi aracılığıyla elde edilen verilerin analiz edilmesi afet yönetimi alanına da katkıda bulunmuştur. Örneğin, [11] numaralı çalışmada afet durumunda gerçek zamanlı bilgi izleme için önce IoT birimi geliştirmiş daha sonra geliştirilen bu birim sayesinde Wi-Fi ve hücresel ağlar aracılığıyla afet anında veriler gerçek zamanlı olarak toplanmıştır. Toplanan bu veriler gerçek zamanlı olarak Kafka, Spark, Esri, Hadoop gibi teknolojiler kullanılarak analiz edilmiştir. Bu analizler sonucunda insan yoğunluğunun fazla olduğu yerler belirlenmiş ve hasar durumu izlenmiştir. Belirlenen yerlere kurtarma ekiplerinin gönderilmesi sağlanmıştır. Bu çalışmada herhangi bir afet anında kurtarma ekiplerinin yönetimi ve kaynak takibi yapılması amaçlanmıştır.

1.2. Tezin Amacı ve Katkıları

Bu tez çalışmasında, kullanıcıların elektrik tüketim bilgileri IoT ve büyük veri teknolojileri ile analiz edilerek, kullanıcı profiline uygun tarife öneri sistemi geliştirilmesi amaçlanmaktadır. Bu amaç doğrultusunda tez çalışmasının katkıları şunlardır:

- Kullanıcının başta elektrik tüketimi olmak üzere yaşam bilgileri/davranışlarını (evde bulunma, elektrikli ürün kullanma özellikleri vb.) elde etmek için yazılım kontrollü akıllı priz geliştirilmesi,
- Geliştirilen priz ile kullanıcının cihazlarına açma-kapama erişiminin yanısıra tüketim bilgilerini izleme, elektrik kesintilerinden haberdar olma gibi yaşam kalitesini arttırıcı unsurlar sunulması,
- Kullanıcı davranışlarını dikkate alan ve kullanıcılardan çevrimiçi toplanan elektrik tüketim verilerinin analizi için büyük veri teknolojilerine dayalı tarife öneri sistemi geliştirilmesi,

Bu çalışmada öncelikle tasarlanan taşınabilir akıllı priz aracılığıyla kullanıcının mobil uygulama üzerinden prizin açma ve kapatma işlemlerini kolayca yapması amaçlanmaktadır. Aynı zamanda kullanıcının belirleyeceği akım limiti ile priz üzerinden geçen akım aşıldığında prizin otomatik kapatılması amaçlanmaktadır. Böylelikle priz, aşırı akım koruma özelliğine sahip olacaktır. Kullanıcı, mobil

uygulamasındaki canlı veri izleme sayesinde grafikler aracılığı ile prizin üzerinden geçen akımın miktarını Amper türünde izleyebilmektedir. İzlenen bu akım değerlerine göre bir ev modellenmektedir. Modellenen evde, tasarlanan akıllı prizden gelen verilerle ortalama bir veri seti oluşturulmaktadır. Bu verisetinde kullanıcının elektrik tüketim verileri çeşitli teknolojiler kullanılarak gerçek zamanlı olarak işlenmektedir. Sonuç olarak kullanıcının elektrik tüketim verilerine göre uygun tarifenin seçilerek elektrik tasarrufu yapılmasına katkı sunulması amaçlanmıştır.

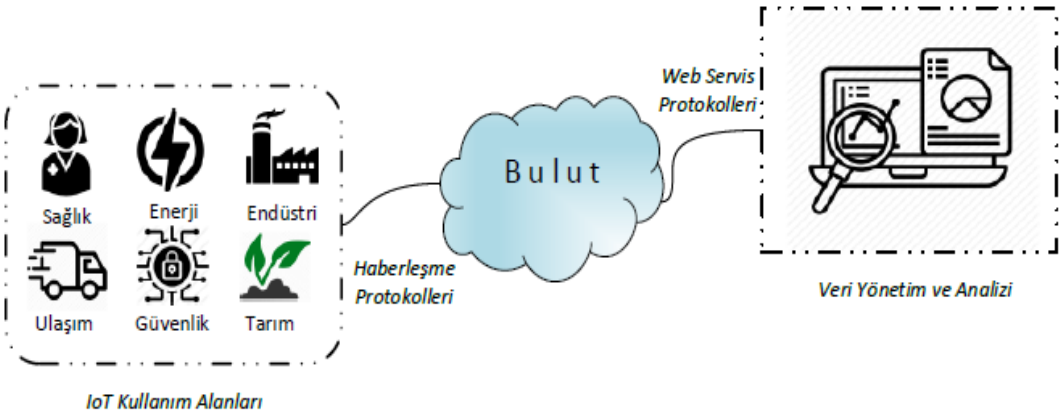
1.3. Tezin Organizasyonu

Bu tez çalışması beş bölümden oluşmaktadır. Giriş bölümünde tez konusu ile ilgili genel motivasyon, literatürdeki benzer çalışmalar, tezin amacı ve katkıları sunulmaktadır. İkinci bölümde çalışma boyunca kullanılan teknoloji ve yöntemler detaylarıyla ele alınmaktadır. Üçüncü bölümde önerilen çözüme ait mimari ve yaklaşım verilmektedir. Dördüncü bölümde ise önerilen çözümün gerçekleştirilebilmesi için gerekli olan uygulama ortamının hazırlanma aşaması anlatılmaktadır. Geliştirilen uygulama aşama aşama beşinci bölümde sunulmaktadır. Uygulama kapsamında elde edilen analiz sonuçları da bu bölümde ele alınmaktadır. Altıncı bölümde ise çalışma sonucunda elde edilen bulgular yorumlanıp, gelecek çalışmalar için önerilerde bulunulmaktadır.

BÖLÜM 2. KULLANILAN TEKNOLOJİ VE YÖNTEMLER

2.1. Nesnelerin İnterneti (IoT)

Nesnelerin İnternet'i, Bluetooth, Wi-Fi, Zigbee, Z-Wave, NFC gibi çeşitli iletişim protokollerini kullanarak birbirleriyle haberleşebilen, algılama ve veri işleme yeteneğine sahip cihazlardan oluşan bir ağdır [12-13]. Nesnelerin İnterneti her zaman, her yerden ve herkes ile bağlantı hedefler. Elde ettiği bu bağlantılar ile yaşam kalitesini arttırmayı ve hayatımızı kolaylaştırmayı amaçlamaktadır. Amaçları doğrultusunda hayatımızda uzaktan izleme, sağlık, tarım, endüstri, afet tespiti, enerji gibi birçok alanda kullanılmaktadır. Bu küresel ağın genel sistem mimarisi Şekil 2.1.'de görülmektedir. Şekilde görüldüğü üzere, IoT teknolojilerini kullanan bir uygulama temel olarak 3 ana bileşenden oluşmaktadır.



Şekil 2.1. IoT genel sistem mimarisi.

2.1.1. IoT nesnesi

Bir IoT nesnesinden genel olarak beklenen bulunduğu ortamdan verileri toplamak, cihazlar ile etkileşim halinde bulunmak ve İnternet üzerinden erişilebilmektir [14]. Bir IoT nesnesinin bu özellikleri karşılayabilmesi için sahip olması gereken bileşenler mevcuttur.

Adres ve Tanımlanma: IoT nesnelerinin çevrimiçi olarak haberleşebilmeleri aynı zamanda kendi aralarında haberleşebilmeleri için bir adrese sahip olmaları gerekmektedir. Her bir IoT nesnesi kendisini tanımlayan kimlik numarasına sahiptir. Bu kimlik numaraları için Yaygın kod(ubiquitous Code, uCode), Global Tekil Kimlik(Universal Unique Identifier, UUID) gibi tanımlamalar kullanılmaktadır.

Algılayıcılar ve Eyleyiciler: Bir IoT nesnesinin bulunduğu ortamdan verileri toplayabilmesi ve diğer IoT nesneleri ile haberleşebilmesi için algılayıcı(sensör) ve eyleyicilere sahip olmalıdır. Algılayıcılar, sistem fonksiyonlarının doğru bir şekilde yerine getirilmesini sağlayan sistem elemanlarıdır. Algılayıcıları birbirinden farklı birçok sınıfa ayırmak mümkündür. Algılayıcı olarak sıcaklık, nem, basınç, ışık sensörleri düşünülebilir. IoT nesnesinde bulunan algılayıcılar aracılığı ile elde edilen bilgiler sonrasında bir üst katman olan ağ katmanına aktarılmaktadır [15].

Gömülü Sistem: IoT nesnelerinin belirli görevleri yerine getirebilmeleri için gerekli olan mikroişlemcili sistemlere gömülü sistem adı verilmektedir. Kullanıcılara daha fazla işlem kapasitesi sağlamak için gömülü sistemler üzerinde çeşitli işletim sistemleri de kullanılmaktadır.

Haberleşme Birimleri: IoT nesneleri hem bulutla hem de diğer IoT nesneleri ile haberleşmelerini sağlamak için çeşitli haberleşme teknolojilerine sahip olmalıdır. RFID, NFC, Bluetooth, Z-Wave, Wi-Fi, hücresel ağlar(3G, 4G, LTE vd.) bu haberleşme teknolojileri arasında yer almaktadır.

2.1.2. IoT mesajlaşma protokolleri

Nesnelerin İnternet'i alanında ağ üzerinden nesnelerin birbirleriyle haberleşmelerini sağlayan web servis yazılımları mesajlaşma ya da haberleşme protokolleri olarak adlandırılmaktadır. Nesnelerin İnternet'i alanında varolan cihazların kısıtlı donanımsal kaynaklara sahip olmasından dolayı mesajlaşma protokolü seçimi önem kazanmaktadır. Bu ihtiyaçları karşılamaya yönelik son yıllarda farklı kuruluşlar tarafından çeşitli mesajlaşma protokolleri geliştirilmiş ve kullanılmaktadır.

Geliştirilen bu mesajlaşma protokollerine Resfful, MQTT, CoAP, AMQP, DDS, XMPP, SoAP örnek olarak verilebilir. Bununla birlikte hiç bir mesajlaşma protokolü bir IoT sistemindeki tüm ihtiyaçları tam olarak karşılayamamaktadır [16]. Bu haberleşme protokolleri Tablo 2.1.'de görüldüğü üzere kullandıkları ulaşım katmanı, mimari yapı, güvenlik, servis kalitesi desteği gibi çeşitli unsurlara göre farklılık göstermektedir. 2014'te yapılan çalışmaya göre, haberleşme protokollerini destekledikleri mimari yapıları ve taşıdıkları mesaj açısından iki farklı unsura göre sınıflandırmak mümkündür [17].

Tablo 2.1. Bazı protokoller arasındaki farklılıklar [18].

Protokol	Ulaşım Katmanı	Mimari Yapı	Güvenlik
CoAP	Udp	İstemci / Sunucu	DTLS
MQTT	Tcp	Yayımcı / Abone	TLS / SSL
XMPP	Tcp	İstemci / Sunucu Yayımcı / Abone	TLS / SSL
AMQP	Tcp	Yayımcı / Abone	TLS / SSL

2.1.2.1. Destekledikleri mimari yapılarına göre

Mesajlaşma protokolleri destekledikleri mimari yapılarına göre sunucu temelli ve veri yolu temelli olarak iki ayrı başlık olarak sınıflandırılmaktadır.

Sunucu temelli mimaride Yayımcı (publisher) bilgiyi üretir, Sunucu (broker) bilginin dağıtımını kontrol eder ve bilgiyi filtreler, depolar, iletir. Abone (subscriber) ise bilgiyi alandır. Bu mimariyi kullanan protokollere AMQP, CoAP, MQTT örnek olarak verilebilir. Veri yolu temelli mimaride ise herhangi merkezi bir sunucu bulunmaksızın yayımcı tarafından üretilen bilgi doğrudan abonelere iletilmektedir. DDS, REST, XMPP bu mimariyi kullanan protokollere örnek olarak verilebilir.

2.1.2.2. Taşıdıkları mesaj açısından

Mesajlaşma protokolleri taşıdıkları mesaj açısından mesaj merkezli ve veri merkezli olmak üzere iki ayrı başlık olarak sınıflandırılmaktadır. Mesaj merkezli IoT

uygulama katmanı mesajlaşma protokolleri mesaj içeriğine bakmaksızın ilgili mesajın alıcılara ulaştırılmasına odaklanır. Mesaj merkezli IoT uygulama katmanı mesajlaşma protokollerine örnek olarak AMQP, MQTT, REST verilebilir. Veri merkezli IoT uygulama katmanı mesajlaşma protokolleri ise mesajın alıcı tarafından anlaşıldığını varsayarak verinin ulaştırılmasını sağlar. Veri merkezli IoT uygulama katmanı mesajlaşma protokollerine örnek olarak CoAP, XMPP verilebilir.

2.1.3. Bulut bilişim ve bulut platformlar

IoT uygulamaları nesneler tarafından üretilen verilerin web ortamında depolanacağı, görselleştirileceği ve analiz edileceği ortamlara ihtiyaç duymaktadır. Bu ihtiyaçlar doğrultusunda IoT uygulamalarını popüler kılan en önemli bileşenlerden biri bulut platformlarıdır. Bulut platformlara internete bağlı herhangi bir cihazdan erişilebilmektedir. Bu platformların bir kısmı ücretsiz olarak herkesin kullanımına açıktır, bir kısmı ise belirli bir süre ücretsiz daha sonra ücretlendirilecek şekilde sunulmaktadır. Piyasada bulunan ve sıklıkla kullanılan IoT bulut platformlarına ThingSpeak, Temboo, Carriots, ThingWors, Xivelys, IBM Watson örnek olarak verilebilir. Bulut platformları mimari yapı, desteklenen uygulama yazılım araçları, uygulama protokolleri, veri görselleştirme, uzaktan kontrol desteği, gerçek zamanlılık ve bildirim sistemi gibi çeşitli unsurlar altında farklılıklar göstermektedir. Bu farklılıklar Tablo 2.2.'de kabaca gösterilmektedir.

Tablo 2.2. Bulut platformların karşılaştırılması.

Platform	Mimari Yapı	Desteklenen Uygulama Yazılım Araçları			Uygulama Protokolleri			
		Android	Ios	Web	REST	MQTT	CoAP	XMPP
Temboo	Bulut tabanlı	+	+	+	+	+	+	-
Xivelys	Bulut tabanlı	+	+	+	+	+	+	-
Carriots	Bulut tabanlı	+	+	+	+	+	-	-
ThingWorx	Bulut tabanlı	+	+	+	+	+	-	-

Bulut platformlar, IoT nesnelerinden elde edilen büyük verinin işlenmesi ve analiz edilerek anlamlı sonuçlar elde edilmesini sağlayan araçlardır. Bu platfomların

senkronizasyon, standardizasyon, güvenilirlik, enerji yönetimi, hata toleransı gibi bir takım çözülmesi gereken problemleri bulunmaktadır.

Bölüm 2.1.'de IoT teknolojilerini kullanan bir uygulamanın 3 ana bileşeni açıklanmaya çalışılmıştır. Nesnelerin İnternet'i mobil teknolojilerin hızla yaygınlaşması ile çeşitli kullanım alanlarına sahip olmaktadır. Sosyal hayatımızın bir çok alanında kullandığımız IoT teknolojilerinin verimlilik, zaman kazancı, uzaktan erişim gibi olumlu yanları olduğu gibi gizlilik, güvenlik, standardizasyon, birlikte çalışılabilirlik gibi olumsuz yanlarından da söz edilebilmektedir [19].

2.2. Büyük Veri Teknolojileri

Büyük veri, kelime anlamı olarak verinin hacminin büyüklüğünü ifade etmektedir. Başka bir deyişle, mevcut olan sistem ve araçlarla işlenmesi, analiz edilmesi ve yönetilmesi oldukça zor olan büyüklükteki veri, Büyük Veri olarak adlandırılmaktadır. Büyük veri daha fazla işlem ve sonuç için depolama, analiz etme ve görselleştirme güçlükleri olan karmaşık yapıya sahip büyük veri kümeleri için kullanılan bir terimdir. Büyük veri analizi modern bilim ve ticaretin merkezindedir. Bu veriler çevrimiçi işlemler, e-postalar, videolar, sesler, resimler, sağlık kayıtları, sosyal ağ etkileşimleri, sensörler ve cep telefonları gibi farklı alanlardan üretilebilir. Üretilen bu veriler gün geçtikçe daha da büyürler ve aynı zamanda veritabanlarında depolanmaktadırlar. Bu verilerin büyümesiyle birlikte depolanması, yönetilmesi, paylaşılması, analiz edilmesi ve görselleştirilmesi zorlaşmaktadır [20]. Büyük verinin artma nedenlerinin başında Nesnelerin İnternet'i cihazları ve uygulamalarının, iş, sağlık ve sosyal çevre başta olmak üzere hayatımızın çeşitli alanlarında hızla yer alması gelmektedir. IoT nesnelerinin ürettiği verilerin anlamlandırılması ihtiyacı bulunmaktadır [21]. Elde edilen bu verilerin anlamlandırılması kabul edilebilir süreler içinde gerçekleştirilmelidir.

2.2.1. Büyük veri bileşenleri

Büyük veri, 5V olarak adlandırılan hacim(volume), hız(velocity), çeşitlilik(variety), doğruluk(veracity) ve değer(value) bileşenlerinden oluşmaktadır. 5V olarak adlandırılan bu bileşenler bazı kaynaklarda 3V (volume, velocity, variety) olarak da verilmektedir [22]. Varolan bir veri kümesinin, büyük veri olarak tanımlanabilmesi için bu bileşenlere sahip olması gerekmektedir.

2.2.1.1. Hacim

Büyük verinin boyutunu ifade etmektedir. Büyük verinin boyutu geleneksel yöntemlerle depolanamayacak kadar büyüktür. Bu verilerin depolanması için dağıtık dosya sistemleri kullanılmaktadır.

2.2.1.2. Hız

Verinin üretilme hızını ifade etmektedir. İnternet uygulamalarının gelişmesiyle birlikte her gün çeşitli alanlardan milyarlarca veri elde edilmektedir. Boyutu hızla artan bu verilerin hızlı bir şekilde anlamlandırılması gerekmektedir. Hızlı veri işleme için paralel programlama teknikleri geliştirilmiştir. Bu teknikler sayesinde akan veri gerçek zamanlı olarak analiz edilmektedir.

2.2.1.3. Çeşitlilik

Üretilen veri kümesindeki yapısal çeşitliliği ifade etmektedir. Büyük veri kaynaklarından elde edilen veriler farklı türlerde olabilir. Veri kaynaklarının farklılığı aynı zamanda veri türlerinin de farklılığını ifade etmektedir. Veriler yapısal, yarı-yapısal veya yapısal olmayan formlarda olabilirler. Bu verilerin bütünleşik ve birbirlerine dönüştürülebilir olması gerekmektedir [23].

2.2.1.4. Doğruluk

Büyük veri içerisinde yer alan her bir verinin doğruluğunu ifade etmektedir. Analiz edilecek verilerin gerçek, güvenilir ve hatasız olması ile ilgilidir.

2.2.1.5. Değer

Yukarıda ifade edilen büyük veri bileşenlerinin sonucunda elde edilecek olan değerdir. Veri üretim, işletme ve analiz evrelerinin yanı sıra elde edilen verilerin bir değere ifade ediyor olması gerekmektedir.

2.2.2. Apache Spark

Apache Spark, Büyük Veri ile uğraşan veri bilimci, veri mühendisi, makine öğrenmesi mühendisi, uygulama geliştiriciler vb. kişiler için birçok araç, API ve programlama dili sunan bütünleşik bir hesaplama motorudur. Bu bütünleşik hesaplama motoru, büyük veri uygulamaları için her türlü ihtiyacı karşılayacak şekilde geliştirilmiştir. Veri okuma/yazma, programlama, Sql sorgusunu makine üzerinde çalıştırma, Makine öğrenmesi uygulamaları, streaming, graf analizi, python, scala, java, R vb. hepsiyle aynı motor ve birbiriyle uyumlu API setlerini içerir.

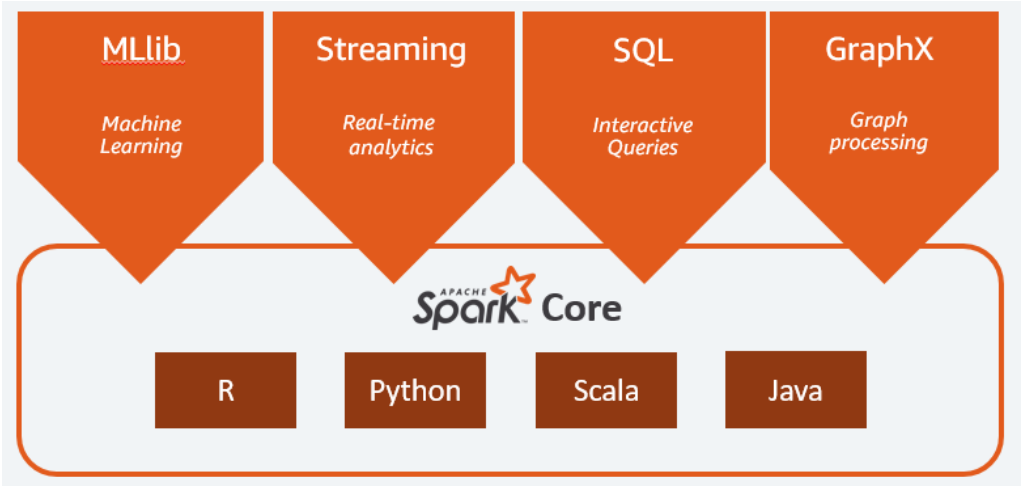
Apache Spark'ın amacı gerçek dünyanın büyük veri analitiği problemlerine çözüm olmaktır. Bunun için interaktif veri analizi yapabilmektedir. IDE ile canlı ortamda uzun uygulama geliştirme, ETL, veri analizi uygulamaları geliştirme gibi amaçlar için kullanılmaktadır. Spark birçok aracı içinde barındıran bir hesaplama motorudur. Veriyi depolamak yerine olduğu yerde analiz eder. Apache Spark, bilgisayar kümelerinde paralel veri işleme için Spark SQL, Mllib, GraphX, Streaming, Spark Core gibi çeşitli bir dizi kütüphaneye sahiptir.

Apache Spark, bir bilgisayardan binlerce sunucuya kadar veriyi geniş ölçekli olarak hızlı bir şekilde işleyebilir. Aynı zamanda dağıtık veri işleme özelliği sayesinde binlerce sunucu üzerinde Petabayt ölçeğinde veri işleme yapabilir.

Apache Spark, dağıtık veri işleme sistemi olan Hadoop ile çok uyumludur ancak çalışması için Hadoop şart değildir. Apache Spark, MapReduce ile rakip olarak görünmektedir. Bu iki sistemin farklılıklarını şu şekilde listeleyebiliriz:

- İkisi de Hadoop YARN üzerinde çalışabilir,
- Spark, bellekten okuma yaptığı için 100 kata kadar daha hızlı çalışabilir,
- Spark, Hadoop'a bağımlı değildir,
- Apache Spark'ın erişilebilirliği daha yüksektir. Scala, Java, Python ve R dillerini destekler.

Yukarıdaki maddelerde gözüktüğü gibi Apache Spark diğer teknolojilere göre daha avantajlı durumdadır. R, Python, Scala ve Java dil desteğine sahip açık kaynak kodlu bir platform olan Apache Spark, Şekil 2.2.'de gösterildiği gibi gelişmiş analitik bileşenlerden oluşmaktadır. Makine öğrenmesi için MLlib, gerçek zamanlı veri analizi için Spark Streaming, veri analizi ile alakalı sorgular için SQL, grafik algoritmaları için GraphX gibi teknolojiler ile entegre bir biçimde çalışmaktadır.



Şekil 2.2. Apache Spark gelişmiş analitik bileşenleri [24].

2.2.2.1. Apache Spark MLlib

Spark MLlib, Apache Spark makine öğrenmesi kütüphanesidir. MLlib(**M**achine **L**earning **Li**brary) sınıflandırma, kümeleme, regresyon, birliktelik analizi gibi büyük

veriler üzerinde anlamlı sonuçlar elde edilebilecek algoritmaların dağıtık bir biçimde çalışmasına imkan tanımaktadır. Ayrıca çeşitli temel istatistikler, doğrusal cebir ve optimizasyon yapmaktadır. MLlib kütüphanesi Scala, Python, R, Java dillerini desteklemektedir [25].

Apache Spark makine öğrenme algoritmalarını iki paket olarak sunmaktadır. RDD yapıları üzerinde işlem yapmak için "org.apache.spark.mllib" kütüphanesi kullanılmalıdır. Veri çerçeveleri (dataFrame) üzerinde yüksek düzeyli API kullanımı için ML boru hattı oluşturularak "org.apache.spark.ml" kütüphanesi kullanılmalıdır [14]. RDD yapıları üzerinde kurulu olan spark.mllib daha eski olmasından dolayı daha fazla özelliğe sahiptir buna karşı dataFrame ile API ise daha yeni, esnek ve kullanımı daha pratiktir.

2.2.2.1.1. Spark düşük seviye API (MLlib)

Apache Spark, iki seviyeden oluşmaktadır. Spark 1.0 ile kullanılmaya başlanan, Spark Düşük Seviye Api RDD'lerden oluşmaktadır. RDD, dağıtık veriyi işlemek için kullanılan bir çok sabit /değişmez nesne yığınından oluşan veri setleridir. RDD olarak adlandırılan bu nesne yığınları, kümeler üzerinde çeşitli parçalar halinde dağıtılır. Temel RDD operasyonları dönüşüm(transformation) ve aksiyon(action) olarak ikiye ayrılmaktadır. Dönüşümler, rdd üzerinde dönüşümler yaparak başka bir RDD oluşturmaya yarayan metotlardır. Map, filter, flatMap, repartition, join gibi çeşitli fonksiyonlara sahiptir. Aksiyonlar ise RDD'lerden bir sonuç üreten metotlardır. Reduce, collect, count, take gibi çeşitli fonksiyonlara sahiptir. RDD'ler, üzerinde transformasyon ve aksiyon işlemleri yapılsa dahi değişmeyen yapılardır.

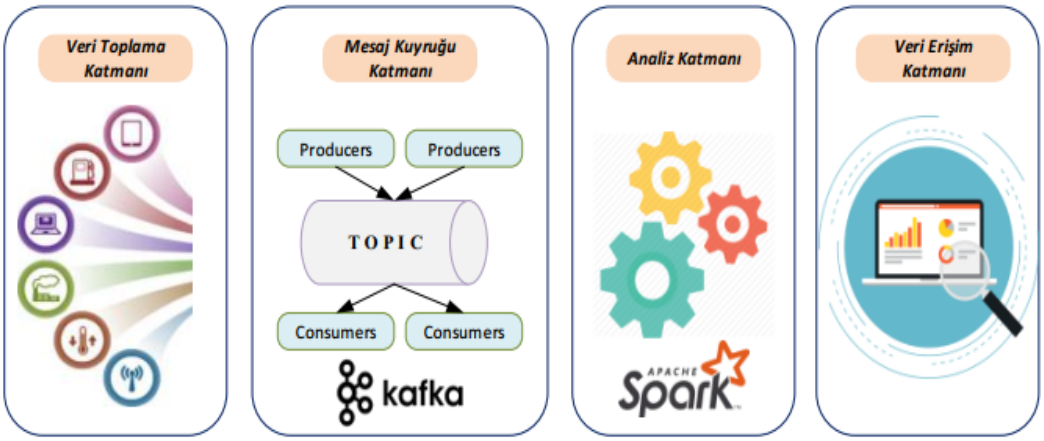
Spark 2.0'ın(Yüksek Seviye API) geliştirilmesiyle birlikte RDD'lerin kullanımı azalmaya başlamıştır.

2.2.2.1.2. Spark yüksek seviye API (ML)

Spark 2.0'ın geliştirilmesiyle birlikte veri çerçevesi (dataFrame) kavramı ortaya çıkmıştır. Yüksek seviye API'nin en önemli farkı verilerin yapısıdır. Veriler yapısal, yarı-yapısal ve yapısal olmayan veriler olarak 3 farklı bölümde incelenmektedir. Yapısal veriler sütun isimlerinin ve saklanan veri tiplerinin belli olduğu veri türleridir. Yüksek seviyeli API, yapısal API olarakta adlandırılmaktadır. En yaygın olarak kullanılan yapısal API veri çerçevesi olarak adlandırılmaktadır. Veri çerçevelerinin RDD'lerden en büyük farkı bir şemaya sahip olmasıdır. Şemalarda sütun isimleri ve veri türleri bir listede saklanmaktadır. Ayrıca R ve Python gibi dillerle oluşturulan veri çerçevelerinden kolaylıkla Spark veri çerçevesi oluşturulmaktadır.

2.2.2.2. Apache Spark Streaming

Apache Spark Streaming, çeşitli sensörler ile alınan verilerin hızlı bir şekilde gerçek zamanlı olarak işlenmesini sağlayan bir teknolojidir. Parça Veri İşleme'de (Batch Processing), veriler önce kaynaklardan toplanır ve depolanır daha sonra veri parçalar halinde işlenir ve sonuca ulaşılır. Gerçek zamanlı sistemlerde ise Şekil 2.3.'te gösterildiği gibi veri kaynaklarından alınan veriler doğrudan akan veri işleme motorlarına gelir ve sonuç üretilir.



Şekil 2.3. Gerçek zamanlı veri işleme mimarisi.

Gerçek zamanlı akan veri işlemede uzun süreli depolama olmaz, işlem süreleri saniyelerin altındadır. Apache Spark Streaming, gerçek zamanlı olarak aldığı verileri toplu işlere ayırır ardından bu toplu işler Spark moturu tarafından işlenir ve işlenen veriler çıktıya yönlendirilir.

2.2.2.3. Spark SQL

Spark SQL, Apache Spark çerçevesinin ana bileşenlerinden biridir. Temelde yapılandırılmış veri işleme için kullanılmaktadır. Python, Java, Scala ve R dillerinde çeşitli uygulama programlama arayüzleri(API) sağlamaktadır. Spark SQL, ilişkisel veri işlemeyi Spark'ın fonksiyonel programlama uygulamasıyla entegre eder. DataFrame adı verilen bir soyutlama sağlayarak, dağıtık bir sorgu motoru olarak davranır. HiveQL ve SQL ile büyük boyuttaki verileri sorgulamayı desteklemektedir. Spark SQL mimarisi 3 katmandan oluşmaktadır.

- Dil Desteği: Bu katman Python, Java, Scala, R dilleri tarafından desteklenen API'lerden oluşur.
- SchemaRDD: Spark SQL şemalar, tablolar ve kayıtlar üzerinde sorgular yaparken, DataFrame olarak da biline SchemaRDD'yi geçici tablo olarak kullanabilmektedir.
- Veri Kaynakları: Spark SQL, JSON dosyaları, Hive tabloları, Parquet dosyaları, Cassandra veritabanı gibi çeşitli kaynaklardan gelen verileri işleyebilir [26].

2.2.2.4. Spark GraphX

GraphX, ETL(çıkar, dönüştür, yükle) süreci, keşif analizi ve iteratif grafik hesaplamayı tek bir sistemde birleştirilen bir Apache Spark bileşenidir. Basit bir işlem olarak görünse de afet yönetimi, bankacılık, borsa, coğrafi sistemler gibi çeşitli alanlarda kullanılmaktadır [27].

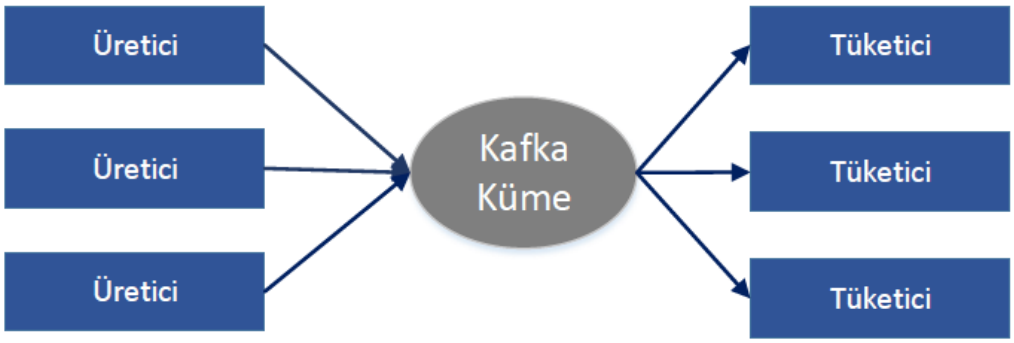
2.2.3. Apache ZooKeeper

ZooKeeper, büyük bilgisayar setlerini yönetmek için kullanılan açık kaynaklı bir Apache projesidir. Dağıtık bir ortamda bir hizmeti yönetmek ve koordine etmek oldukça karmaşık bir süreçtir. ZooKeeper basit mimarisi ile bu süreci daha kolay bir hale getirmektedir [28].

2.2.4. Apache Kafka

Günümüzde gerçek zamanlı bilgiler çeşitli uygulamalar aracılığıyla sürekli olarak oluşturulmaktadır. Nesnelerin İnternet'i, coğrafi olarak dağıtılmış konumlara yerleştirilen sensörlerin hızlı genişlemesi yoluyla akış bilgi işleminin gelişimini daha da hızlandırmaktadır [29-31]. Kafka, sistemler arasındaki entegrasyon problemlerine bir çözüm olarak geliştirilmiştir. Kafka, üreticiler ve tüketiciler arasındaki iletişimi sağlayarak bilgilerin gerçek zamanlı olarak tüketilmesine izin veren ölçeklenebilir ve dağıtık sistemlere uygun bir mesajlaşma protokolüdür [32]. Apache Hadoop üzerinde kurulan olan Kafka, gerçek zamanlı analiz ve akış verilerinin oluşturulması için Apache Storm, Apache Hbase ve Apache Spark ile birlikte çalışır [33]. LinkedIn, Spotify, NetFlix, PayPal, Uber, Cisco, Datasift, Twitter, Foursquare gibi bir çok uygulama ve kuruluş Kafka'yı kullanmaktadır.

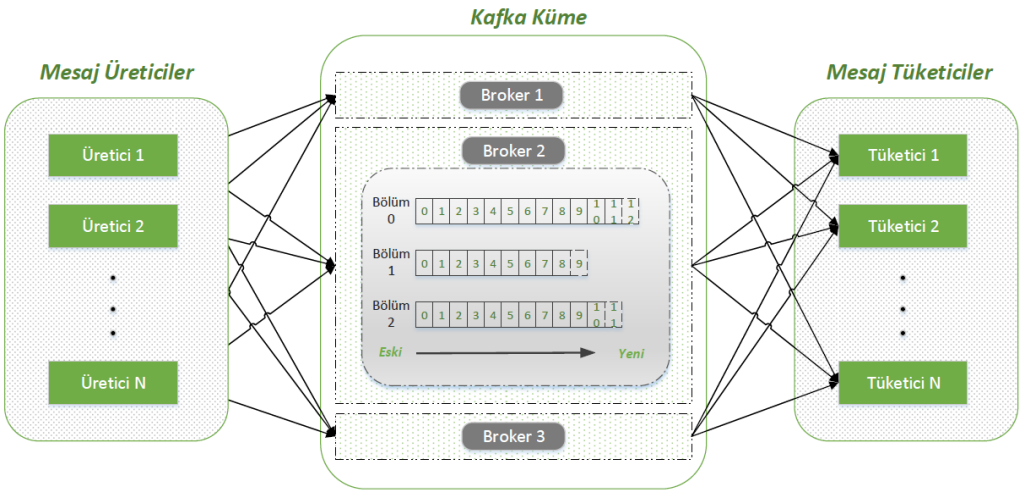
Kafka, üreticiler (producer) ile tüketiciler (consumer) arasında iletişim sağlayan dağıtık yayın-abone (publish-subscribe) mesajlaşma sistemini kullanan bir mesajlaşma platformudur. Kafka mimarisinde üreticiler yayımcı (publishers) ve tüketiciler ise aboneler (subscriber) olarak adlandırılmaktadır. Şekil 2.4.'de gösterildiği gibi temel olarak mesaj üreticileri ürettikleri mesajları Kafka kümeye (cluster) yollar, tüketiciler ise bu verileri Kafka kümeden alır ve tüketir. Kafka'da kayıt edilen veriler mesaj olarak adlandırılmaktadır. Üreticiler ve tüketiciler arasındaki iletişim konu (topic) kavramı ile sağlanmaktadır. Konular, üretici ve tüketiciler arasındaki iletişimi kolaylaştırmaktadır.



Şekil 2.4. Temel Kafka mimarisi.

2.2.4.1. Apache Kafka temel yapısı ve bileşenleri

Apache Kafka mesajlaşma sisteminin temel amacı, mesajların güvenli bir şekilde iletimini ve dağıtımını sağlamaktadır. Apache Kafka'nın genel çalışma mekanizması Şekil 2.5.'de görülmektedir.Kafka mesaj dağıtıcısı sisteminde üreticiler ve tüketiciler arasındaki mesajların dağılımı konu kavramına göre gerçekleştirilmektedir. Konular mesajlaşmayı kolaylaştırmaktadır.



Şekil 2.5. Apache Kafka temel çalışma yapısı.

- Konular: İletilmek istenen mesajların kategorize edilmiş halidir. Bir konu bir veya daha fazla üretici tarafından yazılabileceği gibi aynı zamanda bir veya daha fazla tüketici tarafından da okunabilir. İletilmek istenen bütün mesajlar konular aracılığıyla sunucularda depolanır.

- Bölümler(partition): Her bir konu partition adı verilen bölümlere ayrılmaktadır. Her bir bölüm Offset adı verilen benzersiz bir sıra numarasına sahiptir. Veri kaybının önüne geçmek için her bir bölümün verileri yedeklenmektedir.
- Üreticiler: Yayımcı olarak da adlandırılan üreticiler, verilerin kaynağıdır. Her bir üretici, Kafka kümede bulunan sunuculara(broker) mesaj gönderir. Broker mesajı bölüme ekler.
- Tüketiciler: Abone olarak adlandırılan tüketiciler, Kafka kümedeki konuları dinler. Bir tüketici bir veya birden fazla konuya abone olabilir.
- Sunucu: Kafka kümeyi oluşturan sunuculara broker adı verilir. Sunucular, üreticiden mesajları alır, işler ve yayım işlemini gerçekleştirir.
- Küme: Bir veya daha fazla sunucudan oluşan kümeler, mesajları yönetmek için kullanılmaktadır.
- Lider(Leader): Sunuculardaki her bir bölüm için okuma yazma işlemlerinden sorumlu olan sunucular dır.
- Takipçi(Follower): Bir liderin komutlarını takip eden yerine getiren düğümlerdir.

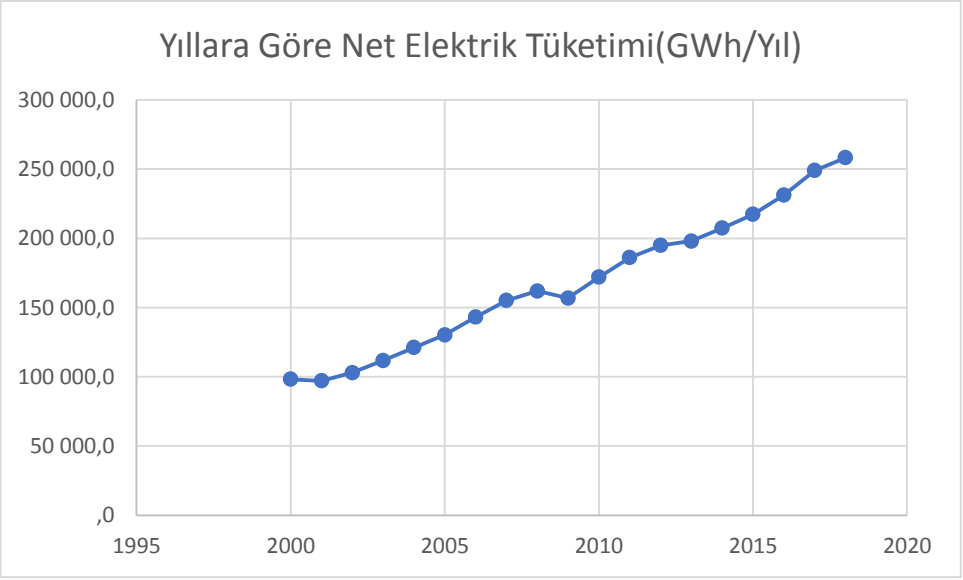
2.2.5. RapidMiner

Yüksek hacimli verilerin analizi ve veri madenciliği için bir çok program mevcuttur. Mevcut olan programlardan biri de RapidMiner yazılımıdır. RapidMiner, Ralf Klinkenberg, Ingo Mi- erswa ve Simon Fischer tarafından Dortmund Teknoloji Üniversitesi Yapay Zeka Biriminde geliştirilmiş bir yazılımdır [34]. Açık kaynaklı bir yazılım olan RapidMiner hem yeni başlayan hem de uzman seviyedeki kullanıcıların ihtiyaçlarını karşıladığı için çoğu ücretli yazılım ile rekabet halindedir. RapidMiner 400'den fazla algoritmaya sahiptir. RapidMiner arff, clm, aml, bib, att, cms, cri, csv, ioc, dat, log, mat, mod, wls, xrff gibi çeşitli dosya uzantılarını destekler. Ayrıca Oracle, MySql, PostgreSql gibi veritabanları programa kolayca entegre edilebilir. Bunların yanı sıra, elde edilen sonuçları görselleştirme ve grafik arayüzlerinde gösterme yeteneği RapidMiner programının tercih edilme sebeplerinden bir diğeridir.

BÖLÜM 3. ÖNERİLEN SİSTEM

Teknolojinin hayatımızda olumlu etkilerinden söz edilebildiği gibi olumsuz etkilerinden de söz edilebilmektedir. Teknoloji, hayatımızı kolaylaştırmakla birlikte insanlığı bazı kavramlara bağımlı kılmaktadır. Bu kavramların başında elektrik tüketimine bağımlılık gelmektedir [35].

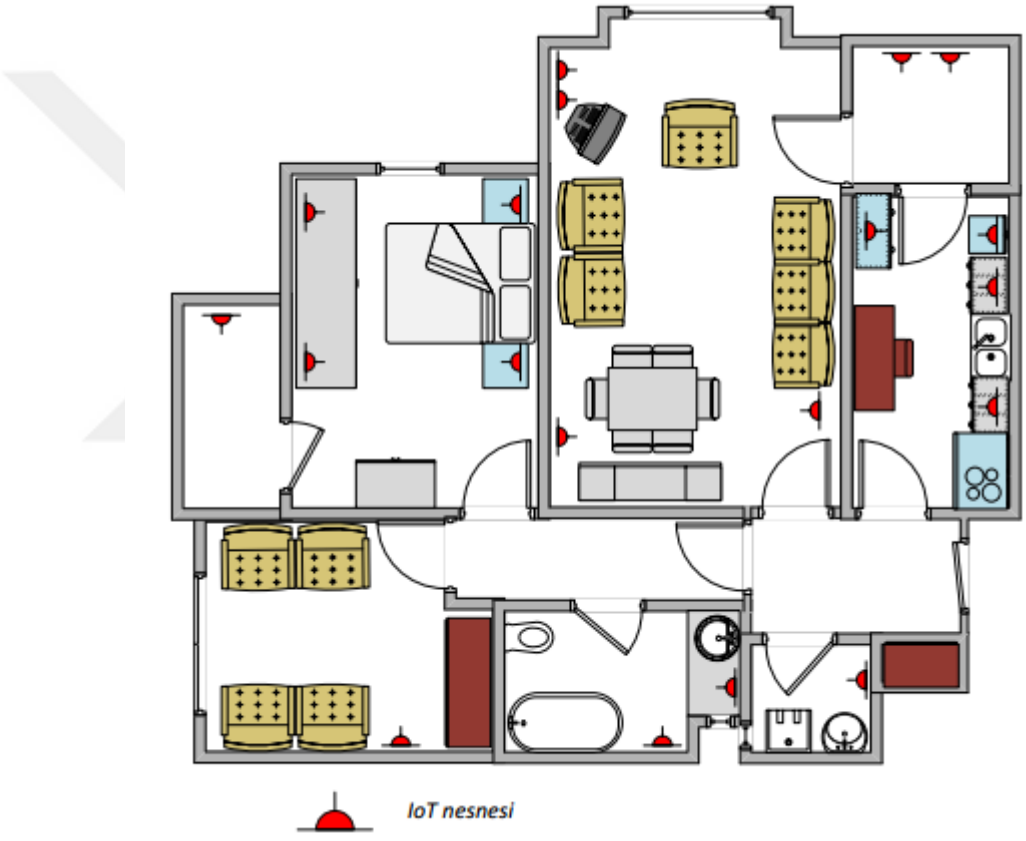
Gündelik hayatımızda bir çok yerde geniş kullanım alanına sahip olan elektrik tüketimi, teknolojinin de ilerlemesi ile birlikte gün geçtikçe hızla artmaktadır. Şekil 3.1.'de gösterildiği gibi Türkiye İstatistik Kurumu'ndan alınan veriler elektrik tüketiminin yıllar içindeki artışını gözler önüne sermektedir [36-38].



Şekil 3.1. 2000-2020 yılları arasındaki net elektrik tüketimi.

Teknolojinin gelişmesi elektriğe olan bağımlılığı arttırdıkça insanlar tüketimi icra ederken maliyeti en aza indirmeyi ve faydayı en yukarı çıkarmayı hedeflemektedir. Bu mantıkla bireyler, optimum maliyet-fayda ilişkisini aramaktadırlar [39].

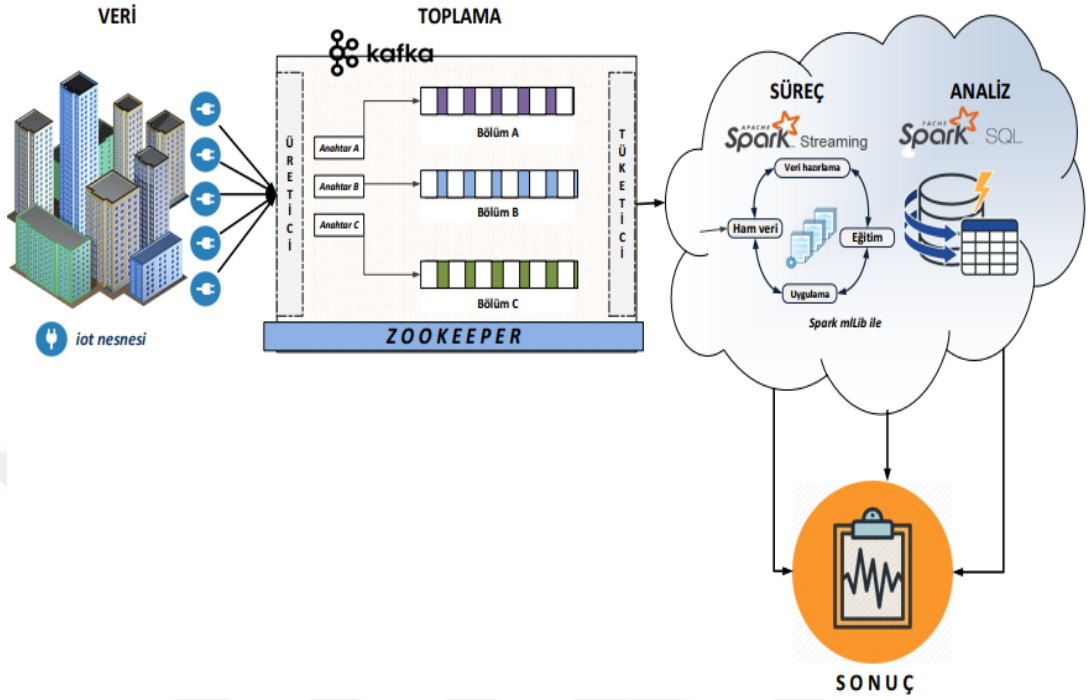
Bu tez çalışmasında, Türkiye'deki elektrik faturalandırma sistemi incelenerek kullanıcıların kendi tüketimlerine uygun tarife seçiminin yapılması bu şekilde maliyetin minimum noktaya getirilmesi amaçlanmaktadır. Bu amaçlar doğrultusunda Şekil 3.2.'de ortalama bir evde elektrik tüketim noktaları, prizler ile gösterilmektedir. Bu çalışmada elektrik tüketimini en aza indirmeyi hedefleyecek şekilde bir priz tasarımı ön görülmüştür. Priz aracılığıyla abonelerden elektrik tüketim verileri toplamak için kullanılmaktadır. Bu elektrik tüketim verileri maliyeti en aza indirmeyi amaçlayacak biçimde analiz edilmektedir.



Şekil 3.2. Ortalama bir ev tasarımı.

Şekil 3.3.'te tez çalışması kapsamında önerilen sistem mimarisi gösterilmektedir. Sistem mimarisine göre kullanımı öngörülen prizler aracılığıyla veriler Apache Kafka ile gerçek zamanlı olarak toplanmaktadır. Gerçek zamanlı olarak belirli periyotlarla toplanan bu veriler Apache Spark Streaming sayesinde alınmakta ve Spark MlLib kütüphanesi kullanılarak veriler analiz edilmektedir. Bu analizlerde Spark SQL kullanılarak anlamlı sonuçlar elde edilmektedir. Elde edilen sonuçlara

göre her bir abonenin aylık elektrik tüketimine göre hangi tarifeye uygun olduğu tespit edilmektedir.



Şekil 3.3. Önerilen sistem mimarisi.

Önerilen sistem mimarisine göre elde edilen verilerden doğru sınıflandırma elde etmek için genel işlem adımları Şekil 3.4.'te yer alan zamanlama şemasına göre aşağıda özetlenmektedir.

Tasarlanması öngörülen IoT nesneleri aracılığı ile veriler okunur ve csv formatında Apache Kafka üreticiye gönderilir.

Üretici verileri sunuculara gönderir. Veriler sunucularda bölümler içinde tutulur.

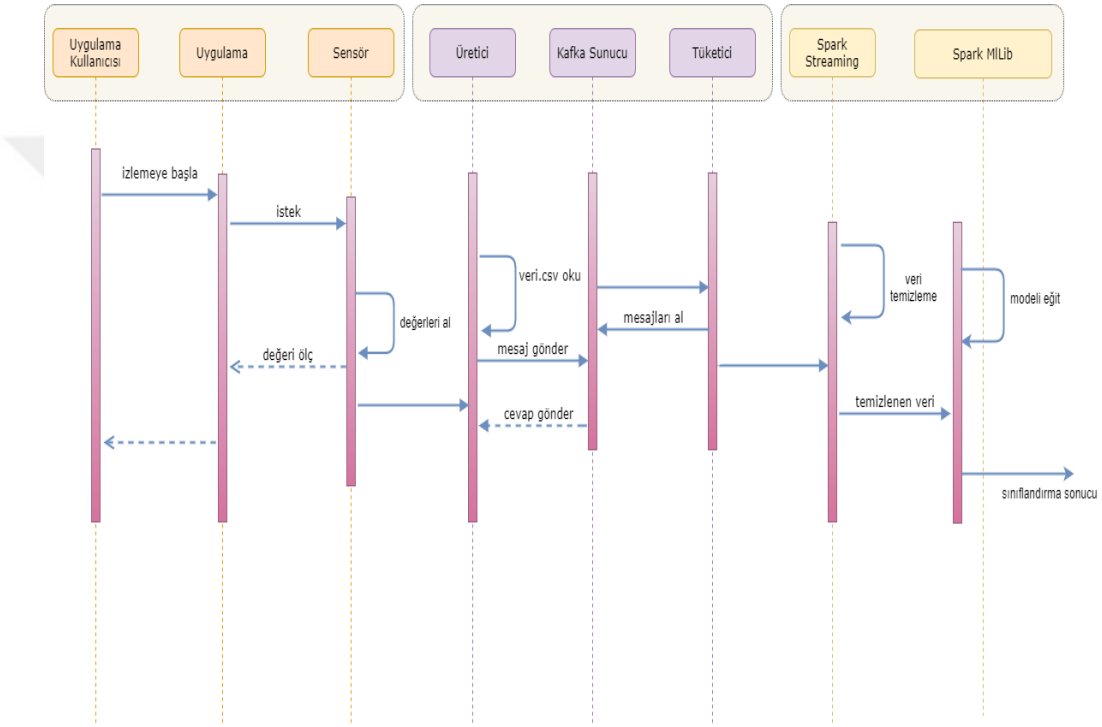
Oluşturulan "topic" ile ilişkisi olan tüketici sunucularda tutulan verileri kullanıcı tarafından belirlenen çerçeveler halinde alır. Oluşturulan mimaride tüketici Apache Spark olarak tanımlanmıştır.

Tüketici olan Apache Spark, akış halinde gelen verileri eğitim ve test verisi olmak üzere iki gruba ayırır.

Seçilen algoritma ile eğitim verisi eğitilir, daha sonra eğitilen model test verileri ile etkileşime sokulur ve tarife seçimi yapılması sağlanır.

Modelin yaptığı sınıflandırma sonuçlarına göre doğruluk oranı gösterilir.

Gerçek zamanlı olarak geliştirilen sistemde, başka veriler geldiğinde maddeler halinde listelenenler döngü şeklinde devam eder.



Şekil 3.4. Gerçeklenen sistemin zamanlama şeması.

3.1. IoT Nesnesi (Yazılım Kontrollü Akıllı Priz)

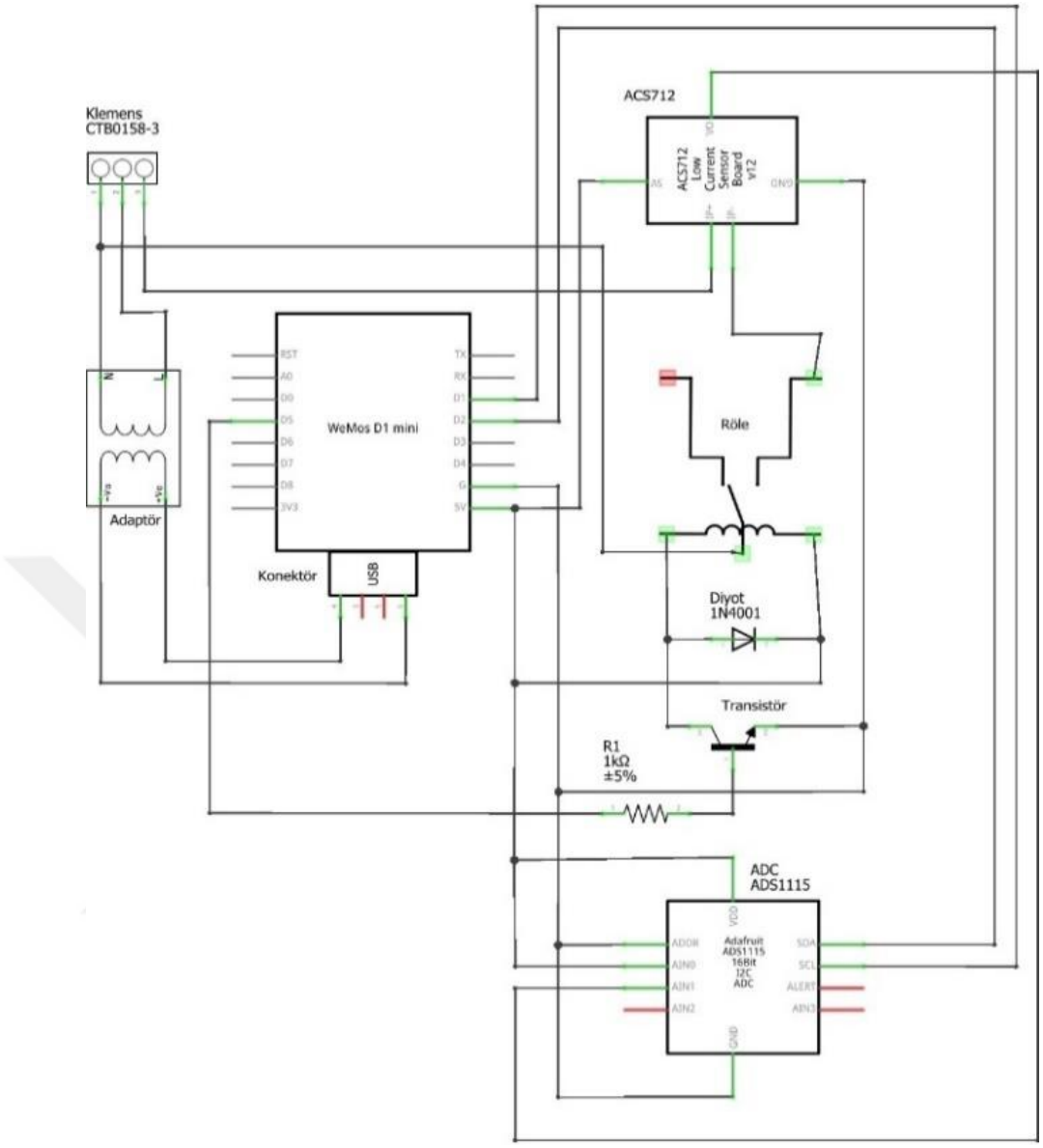
Önerilen sistemde, kullanıcıların elektrik tüketimlerini takip etmek, tüketimlerine ait verileri toplamak ve bulut platforma aktarmak amacıyla akıllı priz olarakta adlandırılabilecek bir IoT nesnesi geliştirilmiştir. Bu alt bölümde, akıllı prizin tasarım ve kullanımı kısaca anlatılmaktadır. Literatürde internet üzerinden cihaz kontrolü ile ilgili yapılmış bir çok çalışma bulunmaktadır. Sunulan IoT nesnesinin literatürdeki çalışmalardan farkı aşağıdaki gibi özetlenebilmektedir.

- Priz açma-kapama ve enerji tüketiminin izlenmesinin yanısıra ek olarak geçmiş enerji tüketimlerini de raporlayabilmektedir. Bu özellik ile kullanıcının gelecek enerji tüketim profili de tahmin edilebilir.
- Prizin takılacağı ürün mobil uygulama üzerinden tanıtılabilir.Bu sayede ürüne özel aşırı akım koruma sınırı koyularak priz üzerinden geçen akım aşıldığında prizin otomatik olarak kapanması sağlanabilmektedir.
- Gerçekleştirilen akıllı priz, ortamdaki Wi-Fi bağlantılarını tarayarak, kullanıcının istediği ağa otomatik olarak bağlanabilmektedir.

3.1.1. Donanım mimarisi

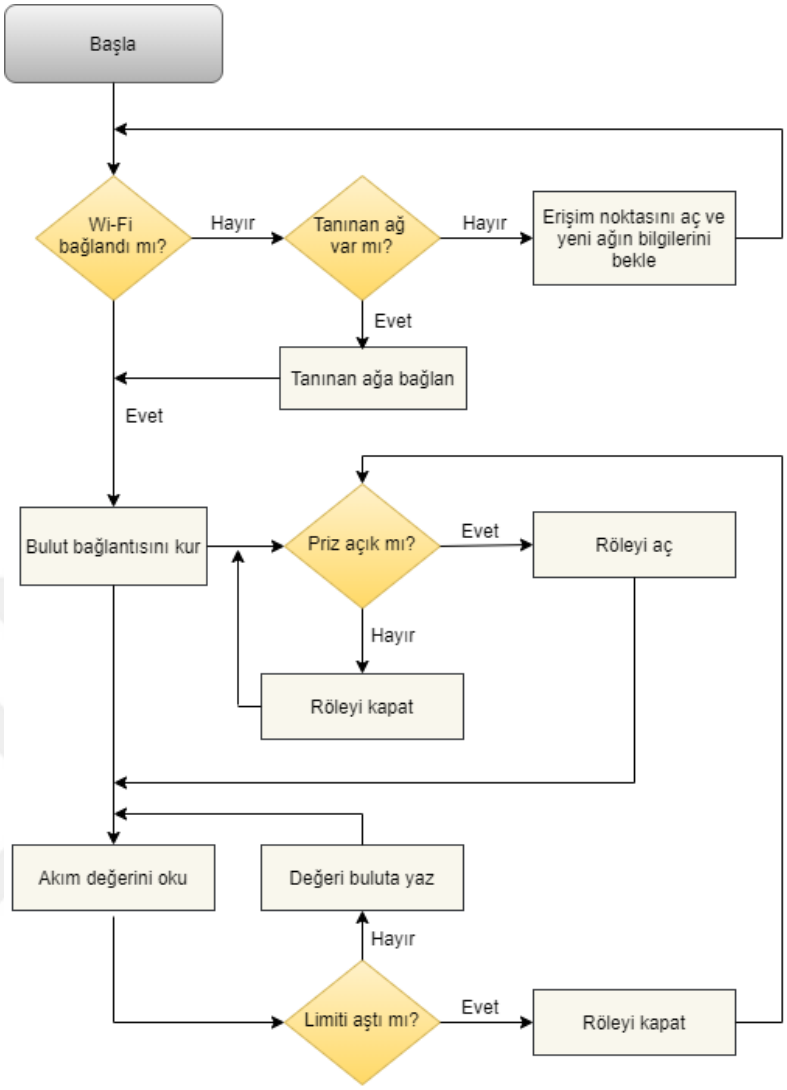
Akıllı Priz, ESP12-E Wi-Fi modülüne sahip Wemos D1 Mini, aç-kapat kontrolü için elektromanyetik röle, enerji tüketiminin hesabı için ACS712 akım sensörü, ADS1115 dönüştürücü ve prizdeki elektronik devrelerin beslemesi için 5v çıkışa sahip adaptörden oluşmaktadır. Wemos D1 Mini, IoT nesnesinin internet(bulut) üzerinden kontrolünü sağlamaktadır.Gerçekleştirilen IoT nesnesi Şekil 3.5.'te görülmektedir. Gerçekleştirilen nesnenin şematik çizimi ise Şekil 3.6.'da gösterilmektedir.

Şekil 3.5. Akıllı Priz.



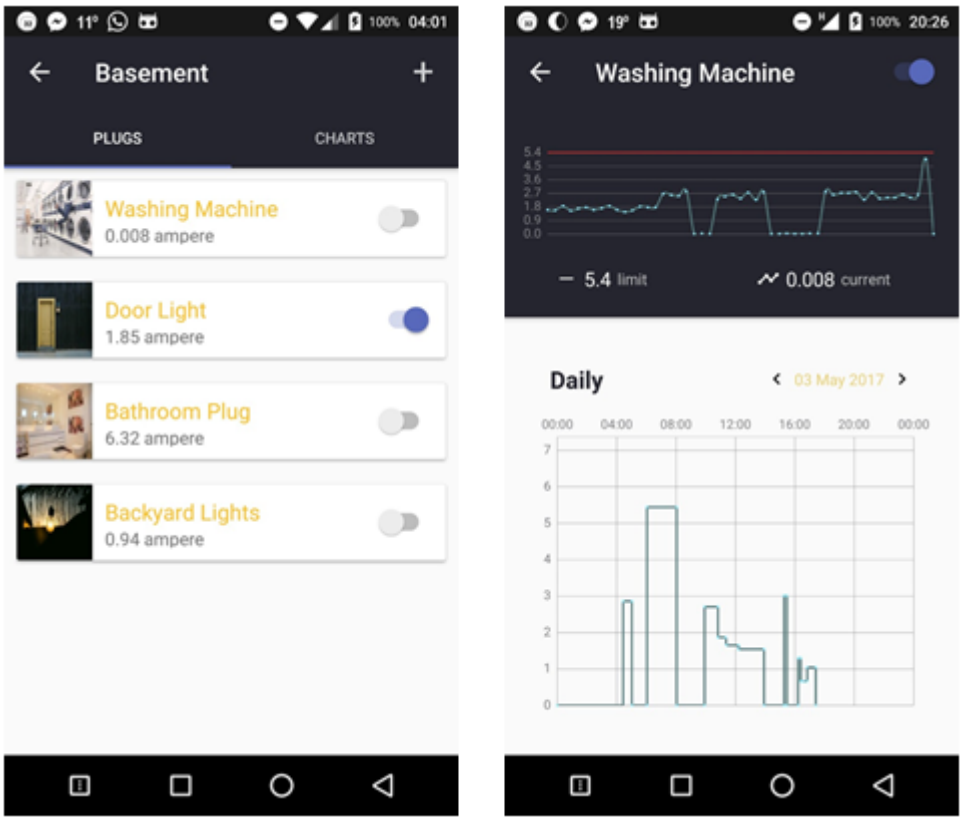
Şekil 3.6. Şematik Çizim.

Geliştirilen sistemin çalışması Şekil 3.7.'de verilen akış diyagramında özetlenmektedir. Yazılım kontrollü akıllı priz, ilk olarak ortamdaki Wi-Fi ağları taramakta ve tanınan bir ağ üzerinden internet bağlantısını sağlamaktadır. Ardından prizin açık-kapalı olma durumu kontrol edilmektedir. Priz açık olduğu sürece üzerinden geçen akım değerini okunmakta aynı zamanda yazılımsal olarak tanımlanan akım sınırı kontrol edilmektedir. Limit aşıldığında prizi otomatik olarak kapatmaktadır.



Şekil 3.7. IoT nesnesi çalışma prensibi.

Kullanıcıların Akıllı Priz'i esnek ve kolay kontrol edebilmeleri amacıyla geliştirilen mobil yazılım Şekil 3.8.'de görülmektedir. Bu yazılım aracılığıyla kullanıcı evin içerisnde Akıllı Priz'i hangi cihaza (çamaşır makinesi, bulaşık makinesi, tv vb.) taktığını tanımlayabilir, ilgili prizlerin anlık, günlük veya aylık gibi enerji tüketimlerini izleyebilir. Ayrıca bu cihazları uzaktan açıp, kapatabilir.



Şekil 3.8. Akıllı priz mobil yazılım.

3.2. IoT Enerji Analiz Platformu

Sunulan akıllı priz ile elde edilen veriler analiz edilirken kullanıcıya özgü tarife önerisi yapılması hedeflenmiştir. Elektrik tüketiminde faturalandırma yapılırken dikkat edilen bazı önemli hususlar bulunmaktadır. Bu hususların en önemlisi zaman dilimlerine göre ücretlendirmenin esas alınmasıdır. Faturalandırma işlemi yapılırken bir gün(24 saat), Tablo 3.1.'de gösterildiği gibi 3 farklı zaman dilimine bölünmektedir. Bu zaman dilimleri Gündüz(T1), Puant(T2) ve Gece(T3) olarak adlandırılmaktadır.

Tablo 3.1. Bir günün zaman dilimlerine ayrılması.

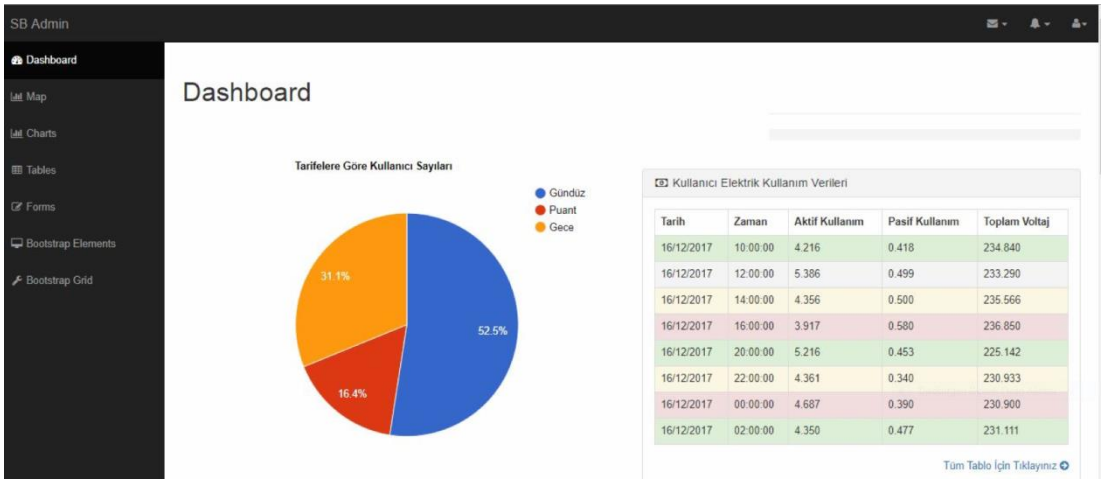
Zaman Dilimleri	Adlandırma	Saat Dilimleri
Gündüz	T1	06:00-17:00
Puant	T2	17:00-22:00
Gece	T3	22:00-06:00

Faturalandırma yapılırken iki farklı tarife seçimi(tek,çok) yer almaktadır. Bu tarifelerin her bir zaman dilimindeki birim fiyatları Tablo 3.2.'de görüldüğü üzere farklılık göstermektedir.

Tablo 3.2. Tarife türü ve zaman dilimine göre birim fiyatlar.

Tarife Tipi	Zaman Dilimi	Birim Fiyat (TL)
Tek Zamanlı Tarife	Gündüz-Puant-Gece	0,2791
Çok Zamanlı Tarife	Gündüz	0,2846
Çok Zamanlı Tarife	Puant	0,4848
Çok Zamanlı Tarife	Gece	0,1245

Enerji tüketim IoT analiz platformu, abonelerin enerji tüketimlerinin takip edilmesini, kullanıcı profillerinin analizine dayalı olarak kullanıcılara maliyeti en aza indirgeyecek tarife türü önermeyi amaçlamaktadır. Akıllı priz prototipinden elde edilen akım verileri web arayüze çekilerek grafikler oluşturulmuştur. Web arayüzü aracılığıyla kullanıcıların tarife bilgileri, aktif kullanım, pasif kullanım, toplam voltaj değerleri gibi özellikleri Şekil 3.9. ve Şekil 3.10.'da gösterildiği gibi görmesine olanak sağlanmıştır.

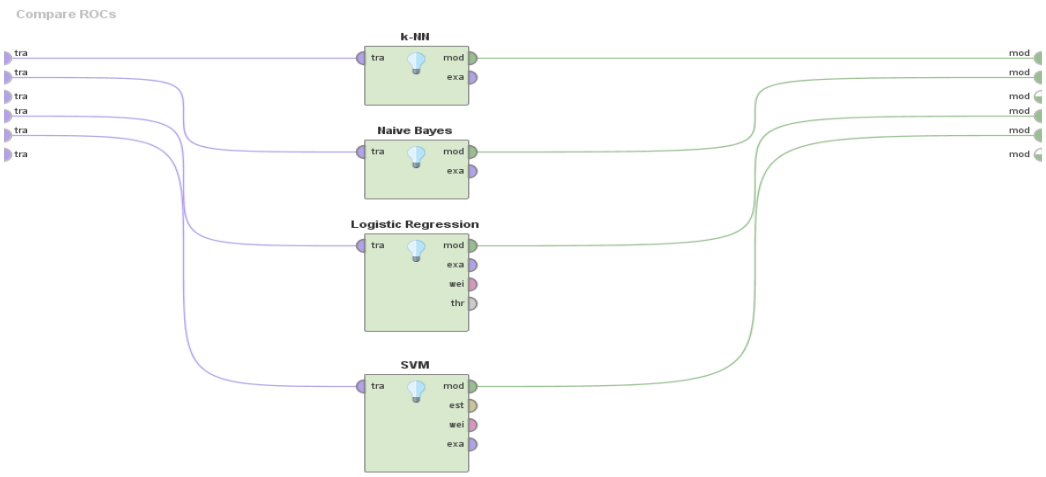


Şekil 3.9. Kontrol paneli ve kullanıcı verileri.

Şekil 3.10. Kontrol paneli grafikleri.

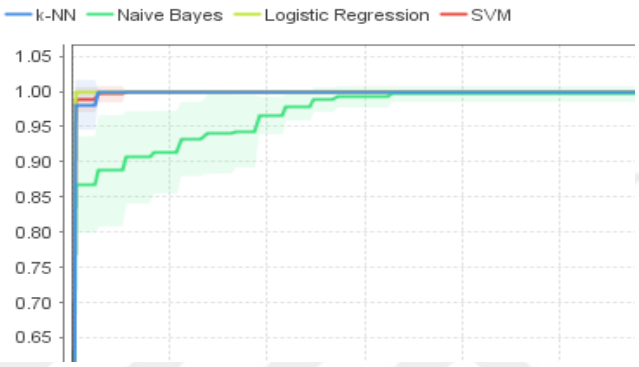
3.3. Veri Analiz Tekniği

Abonelerin enerji tüketiminin takip edilmesi, analiz edilmesi ve sınıflandırılması aşamasında en iyi sonuçları elde edebilmek için doğru algoritma seçimi büyük önem taşımaktadır. Veri madenciliği alanı çeşitli sınıflandırma algoritmalarını barındırmaktadır. Doğru sınıflandırma algoritmasının seçimi ile ilgili çeşitli çalışmalar yapılmıştır [40-43]. Bu çalışmada en uygun sınıflandırma algoritması seçilirken popüler olan algoritmalar arasında RapidMiner yazılımı kullanılarak Şekil 3.11.'deki gibi ROC analizi yapılmıştır.



Şekil 3.11. ROC analizi.

Şekil 3.12.'de verilen ROC analizi grafiği yorumlanırken, ilk kez 1.00 tam sayısına en yakın sonucu veren algoritma en başarılı olarak seçilmektedir. Naive Bayes, k-NN, Lojistik Regresyon, SVM algoritmalarının kullanıldığı ROC analizinde en yüksek başarım oranına sahip algoritmanın Lojistik Regresyon olduğu görülmektedir.



Şekil 3.12. ROC analiz sonuç grafiği.

3.3.1. Lojistik regresyon

Lojistik regresyon, kategorik olarak belirlenen bir bağımlı değişkenin bir veya birden fazla bağımsız değişken ile arasındaki ilişkiyi araştıran istatistiksel bir analizdir [44]. Lojistik regresyonun en çok uygulandığı durum, bağımlı değişkenin iki kategorili olduğu durumdur. Lojistik regresyon yönteminde bağımlı değişken için alınan yanıt dönüşüme tabi tutulmaktadır.

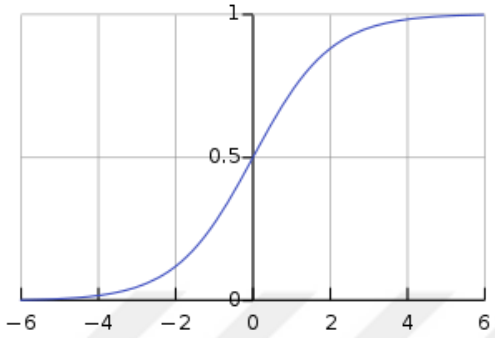
Lojistik regresyon fonksiyonu, başarı olasılığının "p" logit dönüşümüdür [45].

$$p' = \ln\frac{p}{1-p} = \beta 0 + \beta 1\, x1 + \cdots + \beta k\, xk \qquad (3.1)$$

Denklemde:

P	: olayın başarı olasılığı
p-(1-p)	: odds ratio
ln (p-(1-p))	: odds ratio'nun doğal logaritması

Denklem 3.1.'de verilen model doğrultusunda kategorik değişken 1 ve 0 iken fonksiyon grafiği +∞ ve -∞ arasında değerler almaktadır. Şekil 3.13.'te gösterilen Lojistik regresyon fonksiyon grafiği yorumlanırken sonuç 0 ile 0,5 arasında ise 0 kabul edilir ve değişkenler arasında ilişki yok kabul edilir. Fakat sonuç 0,5 ile 1 arasındaysa 1 kabul edilerek değişkenler arasında bir ilişkiden söz edilebilir.



Şekil 3.13. Lojistik regresyon grafiği [46].

BÖLÜM 4. UYGULAMA ORTAMININ HAZIRLANMASI

Makine öğrenmesi, matematik, olasılık gibi bilim dallarını birleştiren aynı zamanda yapay zekanın bir alt kümesi olarak görülen disiplinler arası bir alandır. Makine öğrenmesi sınıflandırma, kümelendirme, regresyon gibi özelliklere sahip algoritmaları kullanarak hızlı ve etkili sonuçlar almayı hedeflemektedir. Günümüzde geliştirilmiş olan makine öğrenmesi algoritmaları elde edilen yapılandırılmış veya yapılandırılmamış büyük verileri istenilen sürede işlemekte yetersiz kalmaktadır. Bu yetersizliğe çözüm olarak dağıtık veri işleme motorları oluşturulmuştur. Bu motorlar, birden çok bilgisayarın hard disk, ram, işlemci gücü gibi özelliklerini birleştirerek tek bir bilgisayarmış gibi davranmasına olanak sağlamaktadır. Bu sayede veriyi daha kısa sürede analiz edebilmektedir. Bu veri işleme motorlarından biri de Apache Spark'tır. Java, Python, Scala, R gibi dilleri destekleyerek uygulama geliştirmeyi sağlayan açık kaynak kodlu Apache Spark, paralel veri işleme için de bir çok kütüphaneye sahiptir. Büyük veri üzerinde makine öğrenmesi uygulamaları için Apache Spark MLlib kütüphanesi kullanılmaktadır.

Bu çalışmada Spark kümesi üzerinde, Windows 8 IntelIJ Idea arayüzü kullanılarak, Scala dilinde Apache Kafka ile makine öğrenmesi uygulamaları gerçekleştirilmiştir.

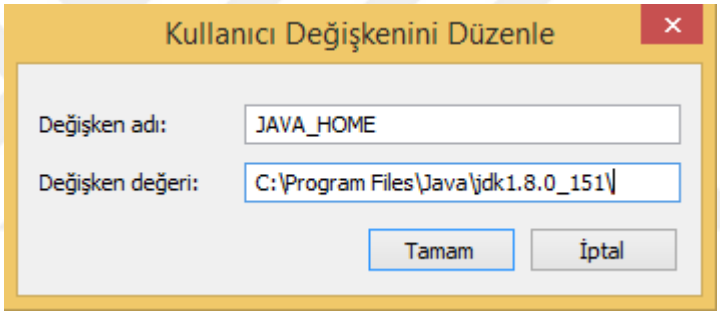
4.1. Çalışma Ortamının Hazırlanması

Apache Spark kurulumu için öncelikle bilgisayarda Java kurulu olmalıdır. Eğer javanın herhangi bir sürümü yüklü değilse işletim sistemine uygun olan jdk dosyası indirilir. İndirilen jdk dosyasının kurulum aşamasında kaydedildiği dizin önemlidir. Java kurulumu tamamlandıktan sonra Apache Spark indirilmelidir. İndirilen .tgz uzantılı dosya klasör haline gelene kadar açılmalıdır. Bu klasör içindeki dosyaların tümü seçilerek C dizini altında yeni oluşturulan spark klasörüne aktarılmalıdır.

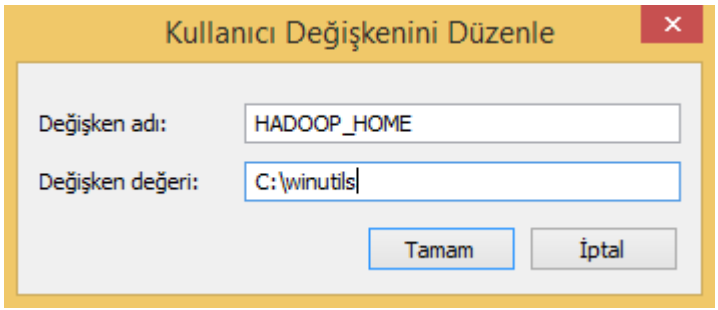
Hadoop için ise ilgili kaynaklardan winutils.exe dosyası indirilmelidir. Winutils.exe dosyası c dizininde oluşturulan winutils adlı klasörün bin adlı alt dizinine eklenmelidir.

4.1.1. Ortam değişkenlerinin ayarlanması

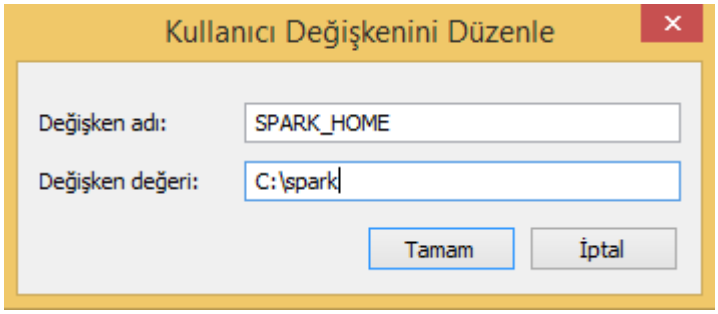
Bu aşamada C dizininde oluşturulan klasörlerin ortam değişkenlerine eklenmesi gerekmektedir. Gelişmiş sistem ayarlarından ortam değişkenlerine gelinir. C dizininde oluşturulan spark klasörü için SPARK_HOME, winutils klasörü için HADOOP_HOME, java için JAVA_HOME adında kullanıcı değişkenleri Şekil 4.1., 4.2. ve 4.3.'te görüldüğü üzere ilgili yollar tanıtılarak ortam değişkenlerine eklenmelidir.



Şekil 4.1. Javanın ortam değişkenlerine eklenmesi.



Şekil 4.2.Winutils klasörünün ortam değişkenlerine eklenmesi.



Şekil 4.3. Spark klasörünün ortam değişkenlerine eklenmesi.

Oluşturulan home dizinleri ayrı ayrı ortam değişkenlerine eklendikten sonra Path değişkenine de eklenmelidir. Düzenlemeler tamamlandıktan sonra işlemlerin gerçekleştirilebilmesi için bilgisayar yeniden başlatılmalıdır. Yeniden başlattıktan sonra cmd açılır. Şekil 4.4.'te gösterildiği üzere Java kontrol edilir.

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\sedab_000>java --version
java 9.0.1
Java(TM) SE Runtime Environment (build 9.0.1+11)
Java HotSpot(TM) 64-Bit Server VM (build 9.0.1+11, mixed mode)
```

Şekil 4.4. Java versiyon kontrolü.

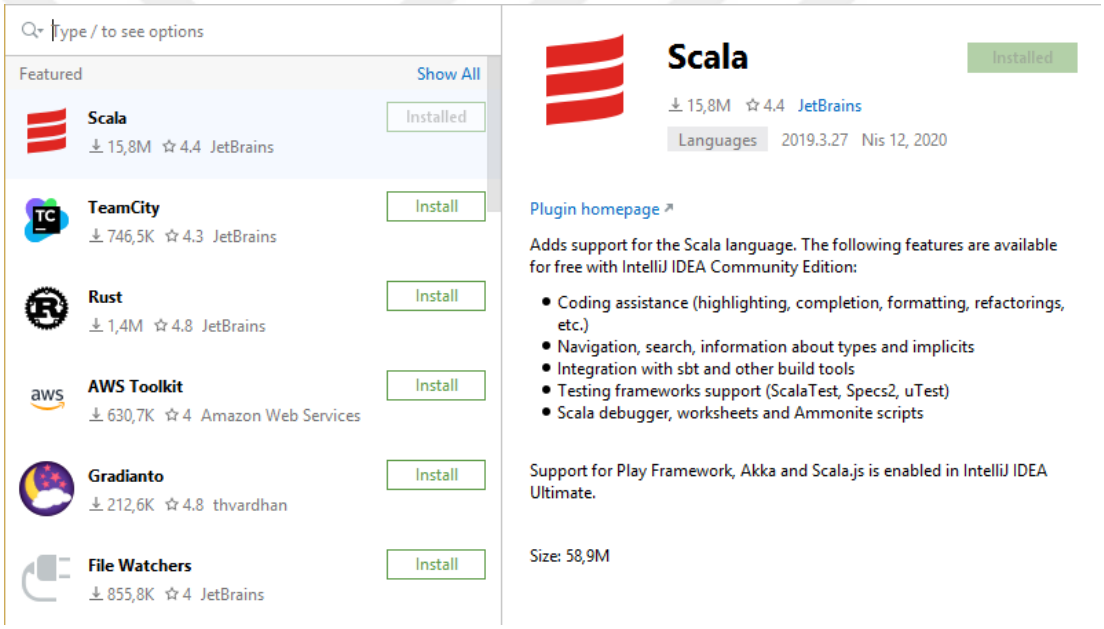
Java kurulumu doğru bir şekilde tamamlanmış ise cmd üzerinde Spark'ın kurulu olduğu dizine gelinerek Şekil 4.5.'teki gibi Spark başlatılır.

```
C:\Users\sedab_000>cd c:\spark

c:\spark>spark-shell
2020-04-13 00:48:00 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform...
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://SedaBalta:4040
Spark context available as 'sc' (master = local[*], app id = local-1586728115354).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Şekil 4.5. Apache Spark'ın başlatılması.

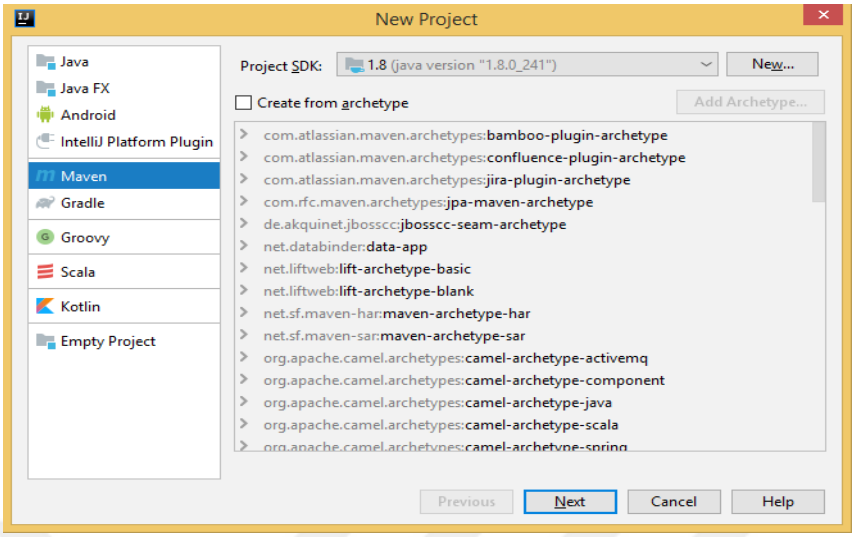
4.2. Kod Geliştirme Ortamının Hazırlanması

Bu çalışma kapsamında Eclipse, Netbeans, IntelliJ gibi çeşitli kod geliştirme ortamlarından IntelliJ IDEA kod geliştirme ortamı kullanılmaktadır. IntelliJ IDEA ilgili kaynaklardan indirilikten sonra kurulumu tamamlanmaktadır. Apache Spark üzerinde geliştirilecek olan uygulamalar için yüksek performans, fonksiyonel programlama, basit sözdizimi gibi çeşitli sebeplerden dolayı Scala dili tercih edilmektedir. IntelliJ IDEA java dilinde kurulu gelmektedir. Scala programlama dilinde kullanabilmek için Şekil 4.6.'da gösterildiği gibi ilgili eklentiler ayarlanmalıdır.



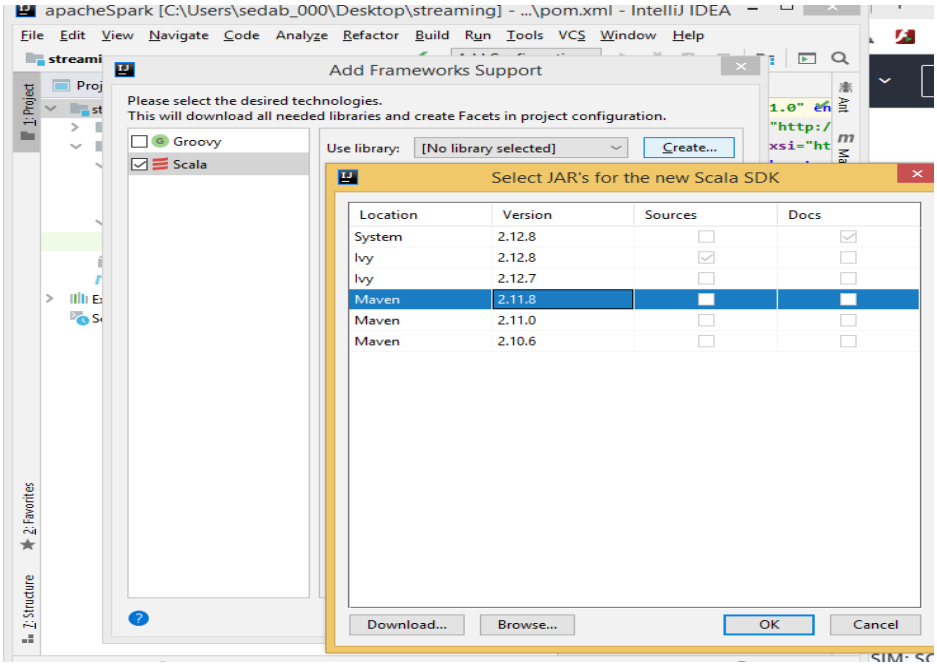
Şekil 4.6. Scala eklentilerinin konfigure edilmesi.

Scala dili uzantısı kod geliştirme ortamına eklendikten sonra artık yeni proje oluşturulabilir. Yeni proje oluşturuluken Şekil 4.7.'de görünen arayüzle karşılaşılır. Bu arayüzde öncelikle çalışmamız için kullanılacak olan jdk dosyası eklenir. Bu çalışmada bağımlılıkların ve kütüphanelerin kolayca yönetilmesi, tüm arayüzlerde esnek kullanım sağlaması, sürüm yönetimini kolaylaştırması gibi sebeplerden dolayı Maven projesi kullanılmaktadır. Maven projesi, bağımlılıklarını POM adı verilen dosya üzerinden yönetir. POM, Maven'deki temel çalışma birimidir [47].



Şekil 4.7. Yeni proje oluşturma.

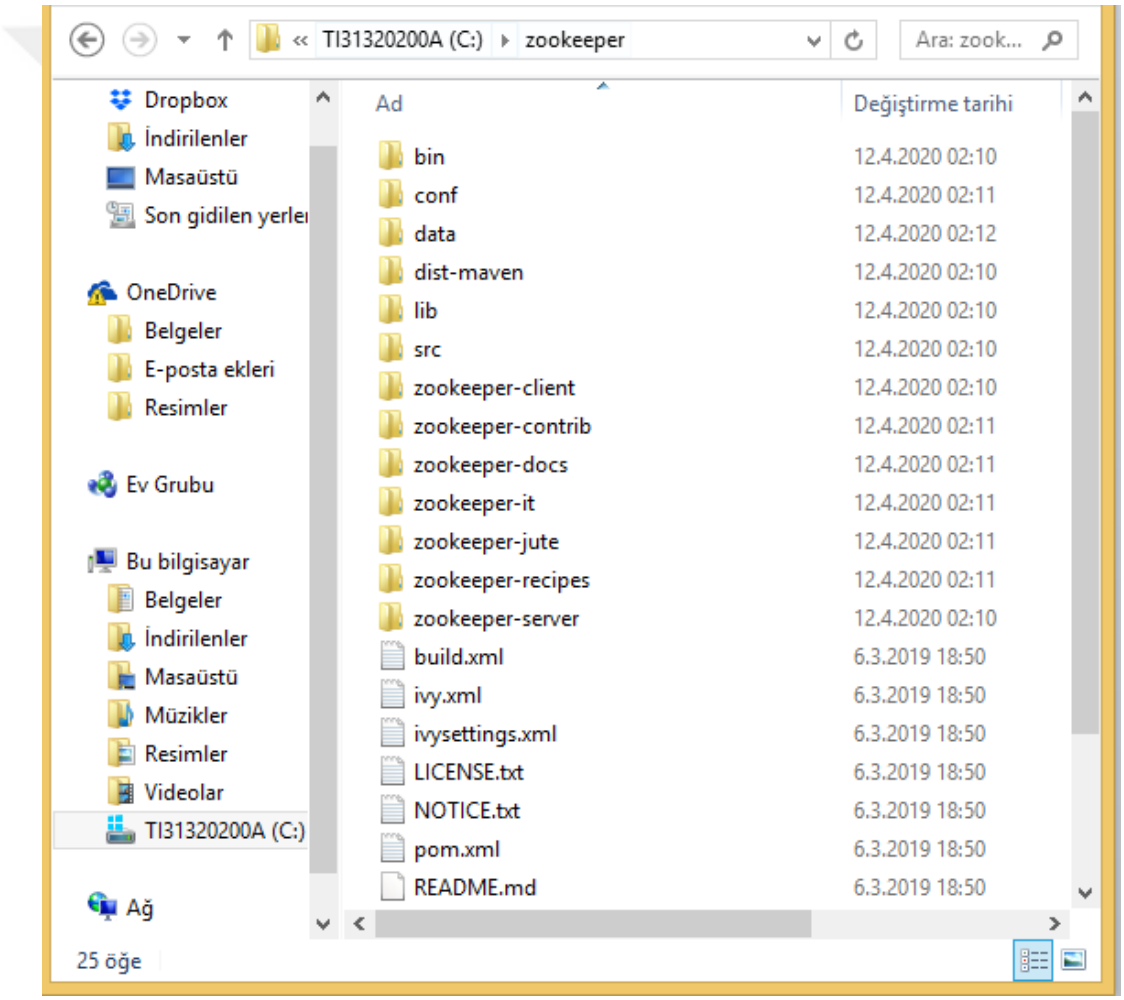
Yeni proje oluşturulduktan sonra oluşturulan projede varsayılan olarak java dili gelmektedir. Bu çalışmada Scala dili kullanılacağından dolayı main ve test klasörlerinde ki dosyaların isimleri scala olarak güncellenmektedir. Daha sonra Şekil 4.8.'de gösterildiği gibi Scala projeye eklenmektedir. Bu işlemlerden sonra IntelliJ IDEA, Scala dilinde kod geliştirme açık bir ortam haline gelmektedir.



Şekil 4.8. Çerçeve desteği ekleme.

4.3. ZooKeeper ve Apache Kafka Kurulumu

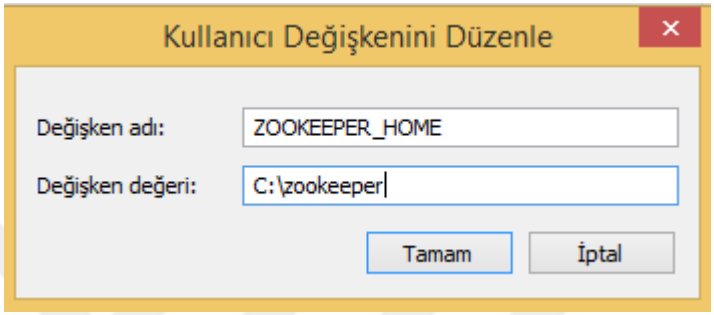
Çalışma ortamı hazırlanıp uygun dil için ayarlar yapıldıktan sonra gerçek zamanlı veri analizine olanak sağlayan Apache Kafka kurulmaktadır. Apache Kafka, bir çok Apache projesinde olduğu gibi çalışmak için ZooKeeper uygulamasına ihtiyaç duyar. Apache ZooKeeper, dağıtık yapıda varolan servislerin koordine edilmesi için geliştirilmiş bir uygulamadır. İlgili ZooKeeper dosyaları indirildikten sonra C dizininde Şekil 4.9.'daki gibi zookeeper isimli klasör oluşturulur ve dosyalar içerisine aktarılır.



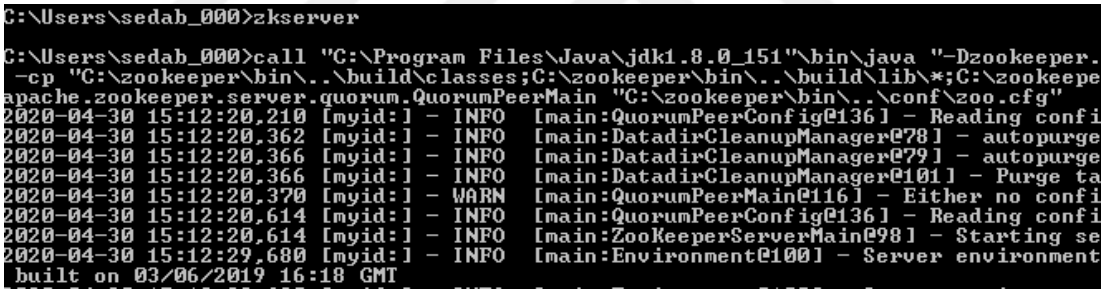
Şekil 4.9. Zookeeper klasör dizini.

Zookeeper klasörü dizini altında "data" isimli bir klasör oluşturularak klasör yolu kopyalanır. Daha sonra conf dizini içerisindeki zoo_sample.cfg dosyasının adı

zoo.cfg olarak güncellenmelidir. Güncelleme işlemi tamamlandıktan sonra zoo.cfg dosyası içerisindeki "dataDir=/tmp/zookeeper" değeri kopyalanan değerle "dataDir=C:\zookeeper\data"değiştirilmelidir. C dizini altındaki ilgili işlemler tamamlandıktan sonra Zookeeper Şekil 4.10.'da gösterildiği gibi ortam değişkenlerine eklenmelidir. Ortam değişkenlerine eklendikten sonra cmd ekranında Şekil 4.11.'de gösterildiği gibi ilgili komut yazılarak Zookeeper başlatılabilir.



Şekil 4.10. ZooKeeper uygulamasının ortam değişkenlerine eklenmesi.



Şekil 4.11. ZooKeeper başlatılması.

Apache Kafka için Zookeeper kurulumu tamamlandıktan sonra, Kafka uygun sürümü indirilir [48]. C dizininde ilgili klasör oluşturularak ortam değişkenlerine eklenir. Kafka klasörü dizini altında "logs" isimli bir klasör oluşturularak klasör yolu kopyalanır. Daha sonra config dizini içerisindeki server.properties dosyası içerisinde yer alan "log.dirs=/tmp/kafka-logs" değeri kopyalanan değerle "log.dirs=C:\kafka\logs" değiştirilmelidir.

Kurulumlar tamamlandıktan sonra bir cmd ekranında Zookeeper çalışırken farklı bir cmd ekranı açılır ve Apache Kafka çalıştırılır.

BÖLÜM 5. ÖNERİLEN YAKLAŞIMDA VERİ ANALİZİ

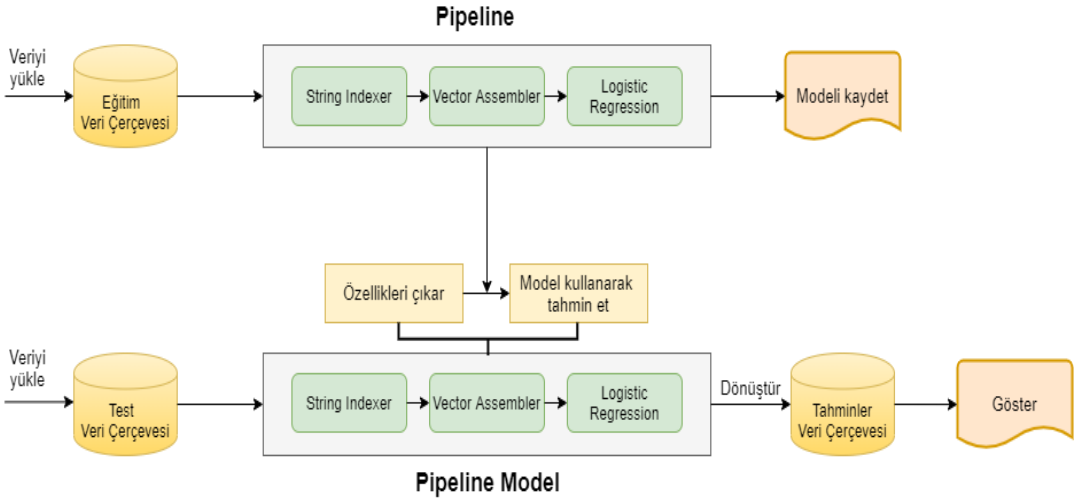
Çalışmamızda, geliştirilen akıllı priz aracılığıyla elde edildiği varsayılan elektrik tüketim verilerini makine öğrenmesi algoritması olan lojistik regresyon ile etkileşime sokup, tüketicilerin tarife önerisine dair sonuç çıkarılması sağlandı. Sonuç çıkarılma aşamasında iki farklı yaklaşım kullanılmıştır. Bu yaklaşımlar arasında dosyadan veri okuma- mesaj dağıtıcısı olan Kafka'dan veri okuma, DataFrame kullanarak analiz-RDD kullanarak analiz, modelin eğitim aşaması gibi farklılıklar bulunmaktadır. İlgili analizler yapılırken Şekil 5.1.'de verilen 5000 adet tüketicinin belirli periyotlardaki elektrik tüketim miktarlarını gösteren veri seti kullanılmıştır. Bu çalışmada amacımız, tüketicilere en uygun tarife tipi önerisini yapabilmektir.

gunduz_ilkendeks	puant_ilkendeks	gece_ilkendeks	gunduz_sonendeks	puant_sonendeks	gece_sonendeks	gunduz	puant	gece	tarife
3447.0	3441.0	3449.0	3728.0	3741.0	3747.0	281.0	300.0	298.0	tek
3432.0	3468.0	3421.0	3756.0	3663.0	3672.0	324.0	195.0	251.0	tek
3567.0	3430.0	3529.0	3739.0	3688.0	3700.0	172.0	258.0	171.0	tek
3572.0	3461.0	3573.0	3784.0	3783.0	3780.0	212.0	322.0	207.0	tek
3425.0	3598.0	3444.0	3704.0	3693.0	3781.0	279.0	95.0	337.0	cok
3427.0	3454.0	3402.0	3715.0	3766.0	3650.0	288.0	312.0	248.0	tek
3450.0	3549.0	3488.0	3676.0	3691.0	3796.0	226.0	142.0	308.0	cok
3575.0	3550.0	3496.0	3736.0	3699.0	3719.0	161.0	149.0	223.0	cok
3531.0	3525.0	3432.0	3736.0	3715.0	3773.0	205.0	190.0	341.0	cok

Şekil 5.1. Elektrik tüketim veri seti.

5.1. Uygulama-1: Structured Streaming ile Gerçek Zamanlı Veri Tahmini

Gerçekleştirilen uygulamada kullanılan model Şekil 5.2.'de gösterilmektedir. Uygulamanın ilk aşamasında veri seti eğitilerek model oluşturulacak, ikinci aşamada ise oluşturulan model, yeni test verilerini tahmin etmek için Spark Structured Streaming gerçek zamanlı uygulamasında kullanılacaktır.



Şekil 5.2. Uygulama-1 model.

5.1.1. Eğitim

Spark oturum ayarları yapıldıktan sonra eğitim veri setinin okunacağı adres ve oluşturulan modelin kaydedileceği adres Şekil 5.3.'te görüntülenmektedir.

```
val filePath= "C:\\Users\\sedab_000\\Desktop\\data1.csv"
val modelPath = "C:\\Users\\sedab_000\\Desktop\\model_batch"
```

Şekil 5.3. Dosya ve model yollarının tanımlanması.

Dosya dizinleri belirtildikten sonra okunacak olan eğitim verinin şeması Şekil 5.4.'te gösterildiği gibi oluşturulmaktadır. Şemanın elle oluşturulmasının sebebi daha az kaynak tüketimi ve alanlar üzerinde tam kontrole sahip olunmasıdır.

```
val schema = StructType(
  Array(StructField("gunduz_ilkendeks", DoubleType),
    StructField("puant_ilkendeks", DoubleType),
    StructField("gece_ilkendeks", DoubleType),
    StructField("gunduz_sonendeks", DoubleType),
    StructField("puant_sonendeks", DoubleType),
    StructField("gece_sonendeks", DoubleType),
    StructField("gunduz", DoubleType),
    StructField("puant", DoubleType),
    StructField("gece", DoubleType),
    StructField("tarife", StringType)
  ))
```

Şekil 5.4. Şemanın oluşturulması.

Şema oluşturulduktan sonra ilgili dizin ve şema gösterilerek eğitim verisi Şekil 5.5.'te gösterildiği gibi okunmaktadır.

```
val df_raw = spark
  .read
  .option("header", "true")
  .schema(schema)
  .csv(filePath)
```

Şekil 5.5. Eğitim verisinin okunması.

Eğitim verisi okunduktan sonra Şekil 5.6.'da gösterildiği gibi veri ön hazırlığı işlemleri yapılmaktadır. StringIndexer kategorik verileri nümerik hale getirmektedir. VectorAssembler, analize girecek olan bütün girdi değişkenleri tek bir sütun içerisinde vektör halinde toplamaktadır.

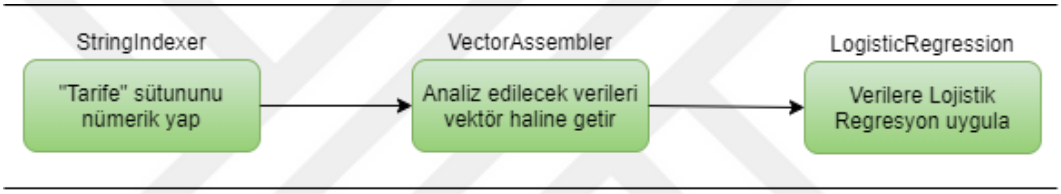
```
val stringIndexer = new StringIndexer()
  .setInputCol("tarife")
  .setOutputCol("label")
  .setHandleInvalid("skip")

val assembler = new VectorAssembler()
  .setInputCols(Array("gunduz","puant","gece"))
  .setOutputCol("features")
```

Şekil 5.6. Eğitim verisi ön hazırlık işlemleri.

Şekil 5.7. ve 5.8.'de gösterildiği üzere eğitim verisi ön hazırlık işlemleri tamamlandıktan sonra, Pipeline aracılığı ile verilere Lojistik regresyon uygulanmaktadır.

```
val logregObj = new LogisticRegression()
  .setFeaturesCol("features")
  .setLabelCol("label")

val pipeline = new Pipeline()
  .setStages(Array(stringIndexer, assembler, logregObj))

val cvModel = pipeline.fit(df)
```

Şekil 5.7. Eğitim verisi boru hattı (pipeline).



Şekil 5.8. Boru hattı oluşturma aşamaları.

Son aşama olarak Şekil 5.9.'daki gibi eğitilen veri seti modelinin "modelPath" olarak belirtilen dizine kaydedilmesi sağlanmaktadır.

```
cvModel.write.overwrite.save(modelPath)
println("Kaydedildi")
```

Şekil 5.9. Oluşturulan modelin ilgili dizine kaydedilmesi.

5.1.2. Tahmin

Bu uygulamada SparkSql'in üzerine kurulmuş olan, Spark Structured Streaming veri akışı sağlamak için kullanılmaktadır. Gerçek dünya uygulamalarında genellikle veriler ApacheKafka gibi özel dağıtılmış kuyruklardan okunmaktadır. Eğitim veri seti için yaptığımız SparkSession oluşturma, şema oluşturma, dizin oluşturma ve kaynağı okuma aşamaları test dosyası içinde Şekil 5.10.'da gösterildiği gibi gerçekleştirilmektedir. "fileDir" olarak belirtilen dizin içerisinde csv formatında bir veya birden fazla dosya bulunabilmektedir.

```
val spark: SparkSession = SparkSession.builder()
  .appName( name = "RunAPP_realtime")
  .master( master = "local[*]")
  .config("spark.driver.memory", "2g")
  .config("spark.executor.memory", "4g")
  .getOrCreate()

val schema = StructType(
  Array(StructField("gunduz", DoubleType),
    StructField("puant", DoubleType),
    StructField("gece", DoubleType),
    StructField("tarife", DoubleType)
  ))

val df = spark
  .readStream
  .option("header","true")
  .schema(schema)
  .csv(fileDir)
```

Şekil 5.10. Test verisi işlemleri.

Şekil 5.11.'de test verisi kaynaktan okunduktan sonra eğitilen model alınarak üzerinden test verileri geçirilmektedir.

```
val df2 = assembler.transform(df)

val pipeModel = PipelineModel.load( path = "C:\\Users\\sedab_000\\Desktop\\model_batch")

val streamModel = pipeModel.stages.last.asInstanceOf[LogisticRegressionModel]

val transformedDF = streamModel.transform(df2)
```

Şekil 5.11. Test verisi tahmini.

Son adım olarak verilerin akış içerisinde tahminini gösteren Şekil 5.12.'deki gibi bir sorgu yazılmaktadır.

```
val query  = transformedDF.writeStream
  .outputMode( outputMode = "append")
  .format( source = "console")
  .option("numRows",5L)
  .start()

query.awaitTermination()
```

Şekil 5.12. Test verisi sorgu tanımlama.

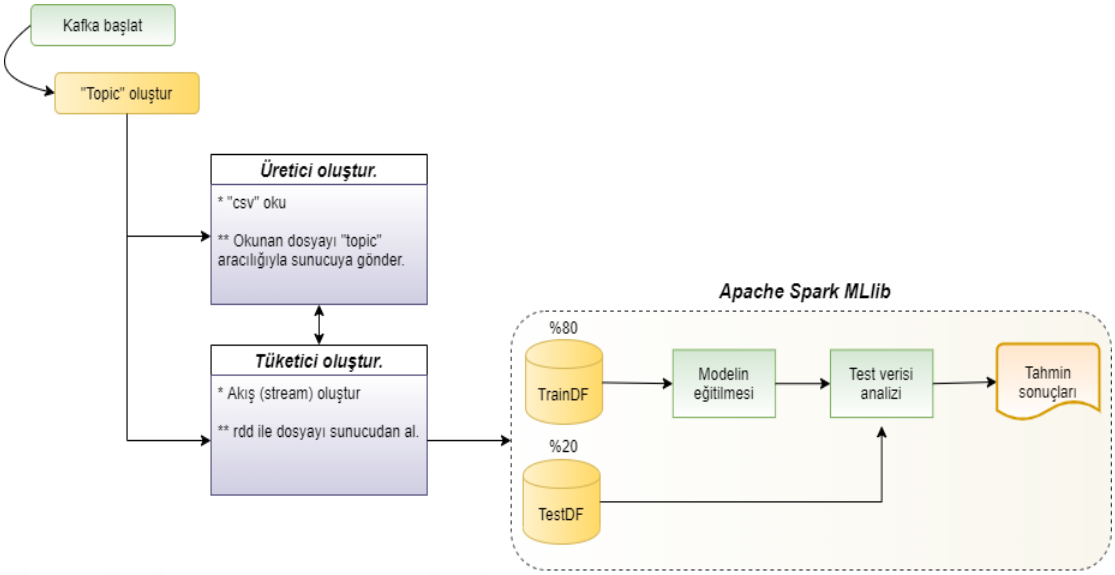
İlgili adımlar tamamlandıktan sonra uygulama çalıştırılır ve Şekil 5.13.'te gösterildiği gibi ilgili dizine test verisi yollandıkça analiz yapılarak tahmin işlemi gerçekleştirilmektedir.

```
-------------------------------------------
Batch: 2
-------------------------------------------
+------+-----+-----+------+------------------+--------------------+-----------+----------+
|gunduz|puant| gece|tarife|          features|       rawPrediction|probability|prediction|
+------+-----+-----+------+------------------+--------------------+-----------+----------+
| 158.0|126.0|118.0|   0.0|[158.0,126.0,118.0]|[5010.54575663755...|  [1.0,0.0]|       0.0|
| 108.0|136.0|194.0|   1.0|[108.0,136.0,194.0]|[-824.76168538316...|  [0.0,1.0]|       1.0|
| 159.0|111.0|122.0|   0.0|[159.0,111.0,122.0]|[2864.79804829907...|  [1.0,0.0]|       0.0|
| 156.0|136.0|177.0|   0.0|[156.0,136.0,177.0]|[874.566121561273...|  [1.0,0.0]|       0.0|
| 166.0|184.0|194.0|   0.0|[166.0,184.0,194.0]|[5098.89052488265...|  [1.0,0.0]|       0.0|
+------+-----+-----+------+------------------+--------------------+-----------+----------+
only showing top 5 rows

-------------------------------------------
Batch: 3
-------------------------------------------
+------+-----+-----+------+------------------+--------------------+--------------------+----------+
|gunduz|puant| gece|tarife|          features|       rawPrediction|         probability|prediction|
+------+-----+-----+------+------------------+--------------------+--------------------+----------+
| 190.0|108.0|129.0|   0.0|[190.0,108.0,129.0]|[1983.00866419296...|           [1.0,0.0]|       0.0|
| 200.0|102.0|193.0|   1.0|[200.0,102.0,193.0]|[-4470.0855798960...|           [0.0,1.0]|       1.0|
| 117.0|107.0|150.0|   1.0|[117.0,107.0,150.0]|[-282.80587977553...|[1.50996528837808...|       1.0|
| 141.0|179.0|171.0|   0.0|[141.0,179.0,171.0]|[6490.68223000880...|           [1.0,0.0]|       0.0|
| 180.0|197.0|112.0|   0.0|[180.0,197.0,112.0]|[14093.0807904983...|           [1.0,0.0]|       0.0|
+------+-----+-----+------+------------------+--------------------+--------------------+----------+
only showing top 5 rows
```

Şekil 5.13. Gerçek zamanlı veri tahmini.

5.2. Uygulama-2: Apache Kafka ile Gerçek Zamanlı Veri Tahmini

Gerçekleştirilen ikinci uygulamada kullanılan model Şekil 5.14.'te gösterilmektedir. Uygulama 1'den farklı olarak Uygulama 2'de veriler mesajlaşma platformu olan Apache Kafka tarafından okunmaktadır. Aynı zamanda Uygulama 1'de anlatılan veri çerçeveleri yerine esnek dağıtık veri kümeleri kullanılmaktadır. Uygulama 2, iki farklı başlık altında incelenebilir. İlk aşama Apache Kafka işlemleri, ikinci aşama ise Spark ML işlemlerinden oluşmaktadır.



Şekil 5.14. Uygulama-2 model.

5.2.1. Apache Kafka

Apache Kafka ile ilgili kütüphaneler projeye dahil edildikten sonra Şekil 5.15.'te gösterildiği gibi Kafka başlatılır ve konu oluşturulur.

```
val kafkaServer = new EmbeddedKafkaServer()
kafkaServer.start()
kafkaServer.createTopic(topic,  partitions = 4)
```

Şekil 5.15. Kafka başlatma ve konu(topic) oluşturma.

Kafka başlatıldıktan sonra Bölüm 2.2.4'de anlatıldığı gibi üretici(*producer*) ve tüketici(*consumer*) tanımlanmalıdır. Üreticiler ve tüketiciler arasında iletişim oluşturulan konu(*topic*) ile sağlanmaktadır. Kafka başlatıldıktan sonra Şekil 5.16.'da gösterildiği gibi RDD oluşturma işlemleri yapılmaktadır. SparkContext RDD oluşturmak için kullanılırken, SparkConf uygulamanın çalışacağı ortam ile ilgili konfigrasyon bilgilerini tutan nesne olarak tanımlanmaktadır.

```
val conf = new SparkConf().setAppName("SimpleStreaming").setMaster("local[4]")
val sc = new SparkContext(conf)
val ssc = new StreamingContext(sc, Seconds(10))
```

Şekil 5.16. RDD başlatma.

5.2.1.1. Üretici

Yayımcı olarak da adlandırılan üreticiler, Şekil 5.17.'de verildiği gibi verileri konular üzerinden okuyarak Kafka kümede bulunan sunuculara göndermektedir.

```
val producerThread = new Thread( name = "Producer") {
  override def run() {

    val client = new SimpleKafkaClient(kafkaServer)
    val producer = new KafkaProducer[String, String](client.basicStringStringProducer)
    var n = 0

    val src = Source.fromFile(csv_file).getLines
    println("csv okundu")
    for(l <- src) {
      producer.send(new ProducerRecord(topic,  key = "" + n, l))
      n = n + 1
    }
    Thread.sleep( millis = 5000)
  }
}
```

Şekil 5.17. Üretici (producer) işlemleri.

Sunuculara gönderilen veriler, üreticiden mesajları alarak yayım işlemini gerçekleştirmektedir.

5.2.1.2. Tüketici

Abone olarak da adlandırılan tüketiciler, Kafka kümede yer alan konuları sürekli olarak dinlemektedir. Şekil 5.18.'de gösterildiği gibi önce tüketici tanımlanmaktadır daha sonra sürekli olarak dinlemeyi sağlayabilmek için bir akış(stream) oluşturulmaktadır.

```
val consumerThread = new Thread( name = "Consumer") {
  override def run() {
    val props: Properties = SimpleKafkaClient.getBasicStringStringConsumer(kafkaServer)

    val kafkaStream =
      KafkaUtils.createDirectStream(
        ssc,
        LocationStrategies.PreferConsistent,
        ConsumerStrategies.Subscribe[String, String](
          Arrays.asList(topic),
          props.asInstanceOf[java.util.Map[String, Object]]
        )
      )
```

Şekil 5.18. Tüketici (consumer) akış oluşturma.

Üreticiler tarafından sunuculara gönderilen veriler tüketiciler tarafından dinlenirken aynı zamanda Kafka akışı veri ürettiğinde ortaya çıkan RDD'ler Şekil 5.19.'da gösterildiği gibi yazdırılmaktadır. Daha sonra yazdırılan bu RDD'ler işlenmeye hazır hale gelerek Spark makine öğrenmesi işlemlerine gönderilmektedir.

```
kafkaStream.foreachRDD(r => {
  println("ConsumerThread")
  val c = r.count()
  println("*** got an RDD, size = " + c)

  mlRDD = r.map(s => s.value().split( regex = ",").map(_.toInt))
  trainFlag = true
})
```

Şekil 5.19. RDD yazdırma.

Tüketici konuları sürekli olarak dinlerken, her veri okuduğunda mlRDD değişmektedir. Aynı zamanda tüketicinin konuyu dinleyip dinlemediğini kontrol etmek için trainFlag adında değişken oluşturulmuştur.

5.2.2. Apache Spark ML

Kafka tüketici tarafından gönderilen verilerin gösterilmesi için ilk olarak veri seti şeması oluşturulmaktadır. Uygulama 1'den farklı olarak şema oluşturulurken Şekil 5.20.'de ki gibi sınıf tanımlanmaktadır.

```
case class dataSchema(Gunduz: Int, Puant: Int, Gece: Int, Tarife: Int)
```

Şekil 5.20. Şema sınıfı oluşturma.

Sınıf tanımlandıktan sonra tüketici tarafından oluşturulan akış içerisinde herhangi bir veri okunduysa Şekil 5.21.'de gösterildiği gibi oluşturulan şema sınıfı kullanılarak veri alınmaktadır.

```
if(trainFlag && mlRDD.count() > 0)
 {
  println("mlThread")
  val inputDF = mlRDD.map(attributes => dataSchema
  (attributes(0), attributes(1), attributes(2), attributes(3))).toDF()
  inputDF.show()
```

Şekil 5.21. Tüketici tarafından gönderilen verinin listelenmesi.

Şekil 5.22.'de gerçeklenen sisteme gönderilen elektrik tüketim verilerinden bir kısmı gösterilmektedir.

```
+------+-----+----+------+
|Gunduz|Puant|Gece|Tarife|
+------+-----+----+------+
|   164|  125| 114|     1|
|   159|  106| 195|     0|
|   146|  138| 143|     1|
|   110|  108| 188|     0|
|   179|  102| 123|     1|
|   100|  107| 168|     0|
|   177|  181| 179|     1|
```

Şekil 5.22. Elektrik tüketim verileri.

Elektrik tüketim verileri alındıktan sonra analiz edilebilmesi için gerekli olan Uygulama 1'de de anlatılan veri ön işleme işlemleri Şekil 5.23.'te gösterildiği gibi yapılmaktadır.

```
val assembler = new VectorAssembler()
  .setInputCols(Array("Gunduz","Puant","Gece"))
  .setOutputCol("Features")

val vectorDF = assembler.transform(inputDF)
vectorDF.printSchema()
```

Şekil 5.23. Veri ön işleme.

Veri ön işleme hazırlıkları yapıldıktan sonra artık veriler analize hazır hale gelmektedir. Uygulama 1'den farklı olarak veri seti rastgele %80 eğitim ve %20 test veri seti olarak iki parçaya ayrılmaktadır. Eğitim veri seti Şekil 5.24.'te gösterildiği gibi lojistik regresyon kullanılarak eğitilmektedir.

```
val Array(trainDF, testDF) = vectorDF.randomSplit(Array(0.80, 0.20), seed = 142L)

val logregObj = new LogisticRegression()
  .setFeaturesCol("Features")
  .setLabelCol("Tarife")

println("*** Model training")
val logregModel = logregObj.fit(trainDF)
```

Şekil 5.24. Modelin eğitilmesi.

Model oluşturulduktan sonra, test verisi oluşturulan model ile işleme sokularak Şekil 5.25.'te gösterildiği gibi test edilmektedir.

```
println("*** Model testing")
val transformedDF = logregModel.transform(testDF)
transformedDF.show()
```

Şekil 5.25. Modelin test edilmesi.

Oluşturulan modelin test edilmesi sonucu elde edilen çıktı Şekil 5.26.'da gösterilmektedir. Bu çıktı yorumlanırken Tarife sütunu ile prediction sütunu karşılaştırılmalıdır. Tarife sütunu olması gereken değeri gösterirken, prediction sütunu lojistik regresyon kullanılarak elde edilen modelin test verisi üzerindeki tahminini göstermektedir.

```
*** Model testing
+------+-----+----+------+------------------+--------------------+--------------------+----------+
|Gunduz|Puant|Gece|Tarife|          Features|       rawPrediction|         probability|prediction|
+------+-----+----+------+------------------+--------------------+--------------------+----------+
|   100|  136| 105|     1|[100.0,136.0,105.0]|[-5524.3072584596...|           [0.0,1.0]|       1.0|
|   100|  143| 148|     1|[100.0,143.0,148.0]|[-3184.1272113755...|           [0.0,1.0]|       1.0|
|   100|  156| 124|     1|[100.0,156.0,124.0]|[-6049.6208007135...|           [0.0,1.0]|       1.0|
|   100|  163| 169|     1|[100.0,163.0,169.0]|[-3570.5664875015...|           [0.0,1.0]|       1.0|
|   101|  141| 169|     1|[101.0,141.0,169.0]|[-1544.1426212253...|           [0.0,1.0]|       1.0|
|   102|  108| 184|     0|[102.0,108.0,184.0]|[2538.37872976679...|           [1.0,0.0]|       0.0|
|   102|  151| 171|     1|[102.0,151.0,171.0]|[-2330.2350015702...|           [0.0,1.0]|       1.0|
|   102|  164| 166|     1|[102.0,164.0,166.0]|[-3876.4230626939...|           [0.0,1.0]|       1.0|
|   102|  182| 144|     1|[102.0,182.0,144.0]|[-7064.1971535211...|           [0.0,1.0]|       1.0|
```

Şekil 5.26. Model testi tahmini.

Farklı yaklaşımlar kullanılarak elektrik tüketim verileri ile gerçekleştirilen uygulamaların başarım oranları doğruluk, FPR, TPR, F-skor, kesinlik, duyarlılık gibi çeşitli metrikler ile Şekil 5.27.'de gösterildiği gibi analiz edilmiştir.

```
val accuracy = trainingSummary.accuracy
val falsePositiveRate = trainingSummary.weightedFalsePositiveRate
val truePositiveRate = trainingSummary.weightedTruePositiveRate
val fMeasure = trainingSummary.weightedFMeasure
val precision = trainingSummary.weightedPrecision
val recall = trainingSummary.weightedRecall
println(s"Doğruluk: $accuracy\nFPR: $falsePositiveRate\nTPR: $truePositiveRate\n"
  + s"F-score: $fMeasure\nKesinlik: $precision\nDuyarlılık: $recall")
```

Şekil 5.27. Ölçülen metrikler.

Analiz sonuçlarına göre oluşturulan modelin başarım oranı Şekil 5.28.'de verilen çıktıya göre %100 olarak görülmektedir.

```
Doğruluk: 1.0
FPR: 0.0
TPR: 1.0
F-score: 1.0
Kesinlik: 1.0
Duyarlılık: 1.0
```

Şekil 5.28. Ölçüm sonuçları.

BÖLÜM 6. SONUÇ VE ÖNERİLER

Çağdaşlaşmanın önemli bir parçası olan aynı zamanda ulaşım, tıp, tarım, iletişim, sanayi ve daha birçok alanda kullandığımız elektrik, hayatımızın vazgeçilmez bir parçası haline gelmiştir. Teknolojinin de gelişmesiyle birlikte elektriğe olan bağımlılık giderek artmaktadır. Bu çalışmada ilk olarak son kullanıcıya hitap eden, kullanıcının elektrik tüketim verilerini uzaktan izlemesine ve müdahale etmesine olanak sağlayan akım korumalı akıllı priz geliştirilmiştir. Daha sonra geliştirilen bu prizden elde edildiği varsayılan elektrik tüketim verileri incelenerek tüketicilerin maliyet açısından maksimum fayda sağlayabileceği tarife tipi önerisi yapılması amaçlanmıştır. Bu amaçlar doğrultusunda aynı sonuca yönelen iki farklı yaklaşım önerilmiştir. Bu yaklaşımları uygularken dağıtık veri işleme ve büyük veriyi gerçek zamanlı olarak işleme konusunda performans olarak en gözde platformlardan biri olan Apache Spark kullanılmıştır. İlk önerilen mimaride gelen elektrik tüketim verileri, gerçek zamanlı olarak dosyadan okunarak verilerin analiz edilmesi için model oluşturulmuş ve daha sonra gelen veriler oluşturulan model üzerinden test edilmiştir. İkinci önerilen mimaride ise veriler mesaj dağıtıcısı olarak görev yapan Apache Kafka sayesinde okunarak, gelen veri eğitim ve test seti olarak ikiye ayrılarak test edilmiştir. Oluşturulan modeller test edilirken Apache Spark Mllib kütüphanesinin sunduğu Lojistik Regresyon kütüphanesinden yararlanılmıştır. Her iki yaklaşım için yapılan analizler sonucunda %100 doğruluk oranı elde edilmiştir. Bu sonuç nitelik sayısının az olması açısından beklenen bir sonuçtur.

Türkiye'de elektrik tüketicileri varsayılan tüketici tipi olarak tek zamanlı tarifeye üyedir. Oluşturulan sistem ve kullanılan veriseti aracılığıyla elde edilen sonuçlara göre yaklaşık olarak her 4 tüketiciden 1'i tarife değişikliğine gittiğinde tasarruf sağlayabilecektir. Bu sebeple önerilen sistemin gerçekleştirilebilir olduğu sonucuna varılabilmektedir.

İlerleyen aşamalarda çalışmanın gerçek dünya problemleriyle daha fazla örtüşmesi maksadıyla, geliştirilen akıllı prizin donanımsal özelliklerinin ve işlevselliğinin arttırılması, her akıllı priz kullanıcısının kendi elektrik tüketim verilerini uzaktan izleyerek tüketim tipine gerçek zamanlı olarak mobil uygulama aracılığıyla karar vermesi sağlanabilir.

KAYNAKLAR

[1] Erdem, Y., Büyük verinin makine öğrenmesi yöntemleri ile Apache Spark teknolojisi kullanılarak sınıflandırılması. Karabük Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Bölümü, Yüksek Lisans Tezi, 2017.

[2] Çetinkaya, S., Hadoop / Mapreduce teknolojisi kullanılarak hızlı tüketim sektöründe büyük veri analizi. İstanbul Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Bölümü, Yüksek Lisans Tezi, 2016.

[3] Ta, V. D., Liu, C.M., Nkabinde, G. W., Big data stream computing in healtcare real-time analytics. IEEE International Conference on Cloud Computing and Big Data Analysis., 37-42, 2016.

[4] Oğur, N. B., Ekg verileri için gerçek zamanlı veri analitiği mimarisi. Sakarya Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Bölümü, Yüksek Lisans Tezi, 2018.

[5] Dülger, Ü., Stratejik büyük veri yönetiminin yatırımlar üzerindeki etkileri. İstanbul Üniversitesi, Fen Bilimleri Enstitüsü, Mühendislik Bilimleri Bölümü, Yüksek Lisans Tezi, 2015.

[6] Assiri, A., Emam, A., Al-Dossori, H., Real-time sentiment analysis of Saudi dialect tweets using SPARK. IEEE International Conference on Big Data., 3947-3950, 2016.

[7] Shoro, A. G., Soomro, T. R., Big data analysis: Ap Spark perspective. Global Journal of Computer Science and Technology., 15(1), 2015.

[8] Singh, M., Asshima, Raheja, S., Credit card fraud recognition by modifying K-Means. IJARCSSE., 4(5), 1291-1296, 2014.

[9] He, W., Li, S., Internet of Things in industries: a survey. IEEE Transactions on Industrial Informatics., 10(4), 2232-2243, 2014.

[10] Rakhee., Ohri, K., R, Saraswathi., Vinita, L. J., Performance analysis of wireless body area sensor analytics using clustering technique. International Conference on Communication and Signal Processing., India, 0278-0281, 2019.

[11] Kucuk, K., Bayilmis, C., Sonmez, A. F., Kacar, S., Crowd sensing aware disaster framework design with IoT technologies. Journal of Ambient Intelligence and Humanized Computing., Germany, 1709-1725, 2019.

[12] Al-Fuquha, A., Guizani, M., Mohammadi, M., Aledhari, M., Internet of Things: A survey on Enabling Technologies, Protocols, and Applications., 17(4), 2347-2376, 2015.

[13] Atzori, L., Lera, A., Morabito, G., The Internet of Things: A survey. Computer Networks., 54(15), 2787-2805, 2010.

[14] Kucuk, K., Bayilmis, C., Nesnelerin Interneti: Teori ve Uygulamaları, 1.Cilt. Papatya Bilim Yayınevi, 13, 2019.

[15] Arslan, K., Kırbaş, İ., Nesnelerin interneti uygulamaları için Algılayıcı/Eyleyici kablosuz düğüm ilkörneği geliştirme. Mehmet Akif Ersoy Üniversitesi Fen Bilimleri Enstitüsü Dergisi., 7, 35-43, 2016.

[16] Naik, N., Choise of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. IEEE International Systems Engineering Symposium., 2017.

[17] Vermesan, O., Friess, P., Internet of things-from research and innovation to market deployment. River publishers., 29, 2014.

[18] Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., A survey on application layer protocols fort he Internet of Things. Transaction on IoT and Cloud computing., 3(1), 11-17, 2015.

[19] Torğul, B., Şahbanşua, L., Balo, F. B., Internet of things: a survey. International Journal of Applied Mathematics Electronics and Computers., 104-110, 2016.

[20] Sagiroglu, S., Sinanc, D., Big data: A review. International conference on collaboration Technologies and Systems., 42-47, 2013.

[21] Ahmed, E., Yaqoob, I., Hashem, I. A. T., Khan, I., Ahmed, A. I. A., Imran, M., Vasilakos, A. V., The role of big data analytics in Internet of Things. Computer Networks, 129, 459-471, 2017.

[22] Chen, H., Chiang, R. H., Storey, V. C., Business intelligence and analytics: From big data to big impact. MIS Quarterly, 36(4), 1165-1188, 2012.

[23] www.netvent.com/big-data-nedir?., Erişim Tarihi: 28.02.2020.

[24] https://aws.amazon.com/tr/big-data/what-is-spark., Erişim Tarihi: 28.02.2020.

[25] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Mllib: Machine learning in apache spark. The journal of Machine Learning Research., 17(1), 1235-1241, 2016.

[26] https://jaceklaskowski.gitbooks.io/mastering-spark-sql/spark-sql.html., Erişim Tarihi: 9.03.2020.

[27] https://www.edureka.co/blog/spark-graphx., Erişim Tarihi: 09.03.2020.

[28] https://www.ibm.com/analytics/hadoop/zookeeper., Erişim Tarihi: 9.03.2020.

[29] Cermak, M., Tovarnak, D., Lastovicka, M., Celeda, P., A performance benchmark for NetFlow data analysis on distributed stream processing systems. IEEE/IFIP Network Operations and Management Symposium, 919-924, 2016.

[30] Shukla, A., Simmhan, Y., Benchmarking distributed stream processing platforms for iot applications. In Technology Conference on Performance Evaluation and Benchmarking. 90-106, 2016.

[31] Buddhika, T., Pallickara, S., Neptune: Real time stream processing for internet of things and sensing environments. In 2016 IEEE International Parallel and Distributed Processing Symposium. 1143-1152, 2016.

[32] Kreps, J., Narkhede, N., & Rao, J., Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB., 11, 1-7, 2011.

[33] Thein, K. M. M., Apache kafka: Next generation distributed messaging system. International Journal of Scientific Engineering and Technology Research., 3(47), 9478-9483, 2014.

[34] https://rapidminer.com/., Erişim Tarihi: 11.03.2020

[35] Akan, Y., Tak, S., Türkiye elektrik enerjisi ekonometrik talep analizi. Atatürk Üniversitesi İktisadi ve İdari Bilimler Dergisi., Erzurum, 1-2, 2003.

[36] Haliloğlu, E. Y., Tutu, B, E., Short-term electicity power demand forecasting for Turkey. Journal of Yasar University., 13(51), 243-255, 2018.

[37] Nişancı, M., Türkiye'de elektrik enerjisi talebi ve elektrik tüketimi ile ekonomik büyüme arasındaki ilişki. Sosyal Ekonomik Araştırmalar Dergisi., 5(9), 107-121, 2005.

[38] http://www.tuik.gov.tr/., Erişim Tarihi: 15.03.2020.

[39] Güloğlu, B., Akın, E., Türkiye'de hane halkları elektrik talebinin belirleyicileri: sıralı logit yaklaşımı. Siyaset, Ekonomi ve Yönetim Araştırmaları Dergisi., 2(3), 1-20, 2014.

[40] Moreno, M., Ubeda, B., Skarmeta, A., Zamora, M., How can we tackle energy efficiency in iot based smart buildings?. Sensors., 14(6), 9582-9614, 2014.

[41] Chen, W., Zhou, K., Yang, S., Wu, C., Data quality of electricity consumption data in a smart grid environment. Renewable and Sustainable Energy Reviews., 75, 98-105, 2017.

[42] Kavaklioglu, K., Ceylan, H., Ozturk, H. K., Canyurt, O. E., Modeling and prediction of Turkey's electricity consumption using artificial neural networks. Energy Conversion and Management., 50(11), 2719-2727, 2009.

[43] https://www.quora.com/What-exactly-is-a-logistic-regression-algorithm-in-machine-learning-What-are-its-applications., Erişim Tarihi: 17.06.2020.

[44] Ürük, E., İstatistiksel uygulamalarda lojistik regresyon analizi. Marmara Üniversitesi, Sosyal Bilimler Enstitüsü, Yüksek Lisans Tezi, 2007.

[45] Sümbüloğlu, K., Akdağ, B., Regresyon Yöntemleri ve Korelasyon Analizi, 1.Cilt. Hatipoğlu Yayınevi, 37, 2007.

[46] https://medium.com/deeplearning-turkiye/sinir-ağları-ve-derin-öğrenme-iii-lojistik-regresyon-cc9686981c6b., Erişim Tarihi: 17.06.2020.

[47] https://maven.apache.org/guides/introduction/introduction-to-the-pom.html., Erişim Tarihi: 15.04.2020.

[48] https://kafka.apache.org/downloads., Erişim Tarihi: 01.05.2020.

ÖZGEÇMİŞ

Seda Balta, 29.01.1994'te Sakarya'da doğdu. İlk, orta ve lise eğitimini Sakarya'da tamamladı. 2012 yılında Akyazı Anadolu Öğretmen Lisesi'nden mezun oldu. 2013 yılında başladığı Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nü 2017 yılında bitirdi. 2017 yılında Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nde yüksek lisans eğitimine başladı. 2019 yılında Kocaeli Üniversitesi'nde Araştırma Görevlisi olarak çalışmaya başladı akabinde yüksek lisans eğitimine Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nde devam etti. Hala Kocaeli Ünivesitesi Bilişim Sistemleri Mühendisliği Bölümü'nde Araştırma Görevlisi olarak görev yapmaktadır.