T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DAĞITIK VERİ YÖNETİM VE İŞLEME MİMARİSİ KULLANILARAK MAKİNE ÖĞRENMESİ UYGULAMALARI GERÇEKLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Engin BAYSAL

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

Mayıs 2019

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DAĞITIK VERİ YÖNETİM VE İŞLEME MİMARİSİ KULLANILARAK MAKİNE ÖĞRENMESİ UYGULAMALARI GERÇEKLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Engin BAYSAL

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

Bu tez 24/05/2019 tarihinde aşağıdaki jüri tarafından oybirliği / oyçokluğu ile kabul edilmiştir.

Doç. Dr.	**Doç. Dr.**	**Doç. Dr.**
Kerem KÜÇÜK	**Nilüfer YURTAY**	**Cüneyt BAYILMIŞ**
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.

Engin BAYSAL
24.05.2019

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 Doç. Dr. Cüneyt BAYILMIŞ'a ve her aşamada beni destekleyen eşim Derya BAYSAL'a teşekkürlerimi sunarım.

Ayrıca vakit ayırıp deneyimlerini sabırla bana anlatan, veri setlerini edinme konusunda yardımlarını esirgemeyen Ahmet Tezcan TEKİN'e teşekkür ederim.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

API	: Application Programming Interface (Uygulama Programlama Arayüzü)
DAG	:Directed Acyclic Graph (Bir Yönlü Grafik)
GFS	: The Google File System (Google Dosya Sistemi)
HDFS	:The Hadoop Distributed File System (Hadoop Dağıtık Dosya Sistemi)
MAE	:Mean Absolute Error (Ortalama Mutlak Hata)
MSE	:Mean Squared Error (Ortalama Kare Hatası)
R2	:R-squared (R Kare)
RDD	:Resilient Distributed Dataset (Esnek Dağıtılmış Veri Kümesi)
RMSE	:Root Mean Squared Error (Karekök Ortalama Hata)
SDK	: Software Development Kit (Yazılım geliştirme Aracı)
SSH	:Secure Shell
YARN	:Yet Another Resource Negotiator

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Büyük Veri, Makine Öğrenmesi, Apache Spark, Pyspark, Google Cloud, Dağıtık İşlem, Sınıflandırma, Regresyon,

Her geçen gün hayatımızda daha çok yer edinen teknolojinin gelişimi ile birlikte, üretilen ve dolayısıyla depolanma ve analiz gerekliliğini beraberinden getiren verilerin bilinen yöntemlerle yönetilmesi ve işlenmesi neredeyse imkânsız hale gelmektedir. Hem veri boyutunda hem de veri çeşitliliğinde artış, bu bağlamda yeni yöntemlerin geliştirilmesini zorunlu hale getirmiştir. Bu tez çalışmasında geleneksel yöntemlerle işlenemeyecek boyut ve çeşitlilikteki veriler için geliştirilmiş olan dağıtık veri yönetim ve analiz araçları kullanılarak makine öğrenmesi uygulamaları geliştirilmektedir. Uygulamalar Google Cloud hizmeti kullanılarak oluşturulmuş Spark kümesi üzerinde pyspark kütüphaneleri kullanılarak gerçekleştirilmektedir.

Bu tez çalışmasında iki farklı veri seti kullanılarak makine öğrenmesi uygulamaları gerçekleştirilmektedir. Uygulama-1'de kablosuz sensörlerden elde edilmiş hareket verileri kullanılarak Lojistik Regresyon sınıflandırma algoritması ile makine öğrenmesi uygulaması geliştirilmektedir. Uygulamanın çalıştırılması esnasında kümedeki kaynakların kullanımları gözlenmektedir. Uygulama-2'de çevrimiçi bir turizm acentesinin kontrol panelinden elde edilmiş veriler ile Rastgele Orman ve Gradyan-artırılmış Ağaç algoritmalarının ortalama tıklama maliyeti tahmininde performansları karşılaştırılmaktadır.

IMPLEMENTING MACHINE LEARNING APPLICATIONS USING DISTRIBUTED DATA MANAGEMENT AND PROCESSING ARCHITECTURE

SUMMARY

Keywords: Big Data, Machine Learning, Apache Spark, Pyspark, Google Cloud, Distributed Transaction, Classification, Regression,

With the development of technology that takes place more and more every day in our lives, it becomes almost impossible to manage and process the data produced and thus brought about the necessity of storage and analysis. Both the data size and the increase in the variety of data have necessitated the development of new methods in this context. In this thesis, machine learning applications have been developed by using distributed data management and analysis tools which have been developed for data that cannot be processed in traditional management. Applications were implemented using pyspark libraries on the Spark cluster created using the Google Cloud service.

In this thesis, machine learning applications were carried out by using two different data sets. The application of machine learning was developed with Logistic Regression classification algorithm by using motion data obtained from wireless sensors in application-1. The use of resources in the cluster was observed during the execution of the application. In the application-2, the average clicks cost estimation performances of Random Forest and Gradient-boosted Tree algorithms were compared by using the data obtained from the control panel of an online tourism agency.

BÖLÜM 1. GİRİŞ

Büyük veri kavramsal olarak verinin hacmini ifade ediyor gibi görünsede aslında sadece hacmini değil aynı zamanda veri yapısında ki farklılıkları da tanımlayan bir kavramdır. Büyük veri yapısal ve yapısal olmayan bilinen veri tabanı yönetim teknikleriyle yönetilemeyen verilerin yönetilmesi olarak ta ifade edilmektedir.

Her geçen gün teknolojinin sağladığı imkânlar artmakta ve yaşamımızda daha çok yer edinmektedir. Özellikle sosyal medyanın gelişimi ile internet kullanıcıları resim, ses ve video gibi farklı formatlarda veri üretebilir olmuştur. Burada ifade ettiğimiz kullanıcı kavramı günümüzde sadece insanlarla sınırlı kalmamış aynı zamanda nesnelerin interneti kavramının ortaya çıkması ve sıradan cihazların internete bağlanmasıyla üretilen verilerin hem türünde hem de boyutunda büyük bir artış gerçekleşmektedir. Bu durum büyük hacimli ve karmaşık verilerin yönetilmesi için ortaya çıkmış kavram olan büyük verinin önemini arttırmaktadır.

Büyük veri (Big data) sözlük anlamı olarak "özellikle insan davranışları ve etkileşimleri ile ilişkili olarak kalıpları, eğilimleri ve ilişkileri ortaya çıkarmak için hesaplanabilir biçimde analiz edilebilen son derece geniş veri setleri" şeklinde tanımlanmaktadır. Ancak günümüzde sık kullanılan bu kavram, modern tanımının ilk olarak ne zaman ve kim tarafından kullanıldığı kesin olarak bilinmemekle birlikte çeşitli kaynaklarda farklı şekillerde tanımlanmaktadır. Gandomi ve Haider yaptıkları çalışmada, Nisan 2012'de 154 kuruluşun çevrimiçi anket sonuçlarına göre büyük veri için yapılan tanımlamaların %28'i müşterilerden gelen veriler ve tedarik zinciri de dâhil olmak üzere işlem verilerinin büyük miktarda büyümesi, %24'ü büyük verilerin hacim, çeşitlilik ve hız sorunlarını gidermek için tasarlanan yeni teknolojiler, %19'u düzen ve uyumluluk için verilerin depolanması ve arşivlenmesi gereksinimi, %18'i sosyal medya, mobil cihazlar ve eylem gerçekleştiren cihazlar gibi veri üreten

kaynakların sayısında artış şekline tanımlarken %11'ini bunların dışındaki tanımlar oluşturmaktadır [1].

Yapılan tanımlamalardan bazılarının büyük verinin ne olduğunu açıklamaya çalışırken bazılarının ne yaptığı ile ilgilendiği görülmektedir. Ancak bütün tanımlamalarda öne çıkan özellik verinin boyutudur. Diğer kavramların daha sonra eklendiğini söyleyebiliriz. Bu bağlamda Doug Laney büyük veri için zorlukları 3V (volume, velocity ve variety) hacim, hız ve çeşitlilik olarak tanımlarken IBM ve Microsoft bu kavramlara dördüncü V (veracity) olarak doğruluğu eklemiş ancak McKinsey&Co. hazırlamış olduğu Bigdata: The next frontier for innovation, competition, and productivity adlı raporunda büyük veride gizli bilgilerin değerine değinmiş ve dördüncü V (value) değer kavramını eklemiştir [2].

Hacim (volume); büyük verinin, veri boyutunu, veri kümesinin boyutsallığı ve aykırı değer tespitini ele alan temel özelliğidir. Hız (velocity); verinin üretilme hızını ve işlenmesi için çeşitli özniteliklerin hesaplanması için kullanılacak algoritmaların analizini ele almaktadır. Çeşitlilik (variety); veri kümesindeki yapısal çeşitliliği ifade etmektedir. Doğruluk (veracity); insan yargısı sonucunda üretilen veriler gibi bazı verilen kaynaklarına özgü güvenilmezliğini temsil etmektedir.

Değer (value); verilerin belirleyici özelliği olarak ifade edilen değer, Oracle'ın tanımlamasına göre orijinal haliyle ele alınan veriler hacmine kıyasla düşük değer yoğunluğuna sahiptir. Ancak büyük veriler analiz edilerek yüksek değer elde edilebilmektedir [1], [3]. Değer, büyük verinin karakteristik özelliği değil, işlenmesinin sonucu olarak tanımlanmaktadır [4].

Büyük veri yaşama, çalışma ve düşünce tarzımızı değiştirecek bir dönüşüm olarak adlandırılmaktadır. Bu dönüşümün temel amacı, bilginin keşfedilmesini ve daha iyi karar verilmesini sağlamak için büyük miktarda veri kullanmaktır. Bu amacın gerçekleşmesi için çeşitli veri analizi yaklaşımları ve teknolojilerine gereksinim duyulmaktadır. Bu tez çalışması veri analizinin temel bileşenlerinden makine öğrenmesine odaklanmaktadır. Hadoop dosya sisteminde yer alan veri seti ile Spark

uygulaması kullanılarak makine öğrenmesi algoritmalarının performans analizleri yapılmaktadır.

1.1. Literatür Özeti

Yapılan literatür taramasında çoğunlukla büyük veri ile ilgili kavramsal çalışmalar olduğu görülmektedir. Bu tez çalışmasına benzer dağıtık veri depolama ve işleme uygulamaları gerçekleştiren çalışmalarda bulunmaktadır. Keskin büyük veride makine öğrenmesi uygulaması isimli yüksek lisans tez çalışmasında R dili ile makine öğrenmesi uygulaması gerçekleştirmiştir [5]. Oğur yüksek lisans tez çalışmasında EKG verileri için gerçek zamanlı veri analitiği mimarisi geliştirmiştir [6]. Çetinkaya hadoop/mapreduce teknolojisi kullanılarak hızlı tüketim sektöründe büyük veri analizi isimli yüksek lisans tez çalışmasında büyük veri uygulamaları yapılmıştır [7]. Erdem yüksek lisans tez çalışmasında büyük verinin makine öğrenmesi yöntemleri ile Apache spark teknolojisi kullanılarak sınıflandırılması uygulaması geliştirmiştir [8].

Bu tez çalışmasında incelenen benzer çalışmalardan farklı olarak çalışma ortamı olarak google cloud hizmeti seçilmiş ve kablosuz sensörlerden toplanan hareket verileri kullanılarak gerçekleştirilen uygulamada, uygulamaların çalışması esnasında çalışma ortamı kaynaklarının kaynak yöneticisi tarafından nasıl yönetildiği gözlenmektedir. Ayrıca çevrimiçi bir turizm acentesinin kontrol panelinden edinilen verilerle ortalama tıklama maliyetlerinin tahmininde makine öğrenmesi algoritmalarının başarımları karşılaştırılmaktadır.

1.2. Tezin Amacı ve Katkıları

Tezin amacı büyük veri için depolama ve analitik teknolojilerinin incelenmesi ve makine öğrenmesi için gerekli görevlerin bir küme üzerinde dağıtılarak gerçekleşmesini sağlamaktır. Bu bağlamda büyük veri analitik çözümleri için geliştirilmiş RDD veri yapısı sayesinde işlenecek verileri esnek bir şekilde belleklere yerleştirerek hesaplama işlemlerini gerçekleştirebilen Apache Spark kullanılmaktadır. Apache Spark analitik motorunun desteklediği dillerden biri olan python programlama

dili ile Pyspark uygulama programlama arabirimi (API) sayesinde makine öğrenmesi uygulaması yapılmaktadır.

Gerçekleştirilen uygulamalardan, Uygulama-1'de hareket sensörlerinden toplanmış veri seti kullanılarak spark kümesi üzerinde hareket türünün tespit edilmesinde Lojistik Regresyon sınıflandırma algoritması ile geliştirilen modelin performansı test edilmektedir. Aynı zamanda uygulamanın spark kümesi üzerinde çalışması esnasında kaynakların kullanımları incelenmektedir.

Uygulama-2'de çevrimiçi bir turizm acentesinin kontrol panelinden edinilen veriler ile aynı spark kümesi üzerinde otellerin tıklama başına ortalama maliyetlerinin tahmininde Rastgele Orman ve Gradyan-artırılmış Ağaç regresyon algoritmalarının performansları karşılaştırılmaktadır.

1.3. Tezin Organizasyonu

Bu tez çalışması beş bölümden oluşmaktadır. Giriş bölümünde büyük veri ile ilgili genel bilgi verilmektedir. İkinci bölümde büyük veri kavramı detaylarıyla ele alınmakta ve bu kavramla ilgili geliştirilmiş teknolojiler açıklanmaktadır. Üçüncü bölümde gerçekleştirilen uygulamalar için kullanılacak ortam hakkında detaylı bilgiler verilerek bu ortamların hazırlanma aşamaları şekillerle anlatılmaktadır. Dördüncü bölümde uygulamalar için kullanılan mimari açıklanmaktadır. Makine öğrenmesi uygulamaları için kullanılan algoritmaların genel yapısı anlatılarak büyük veri teknolojileri ile uygulamalar gerçekleştirilmektedir. Aynı zamanda kaynakların kullanımları gözlenmekte ve analiz sonuçları sunulmaktadır. Beşinci bölümde ise çalışma sonucunda elde edilen bulgular yorumlanmaktadır.

BÖLÜM 2. BÜYÜK VERİ KAVRAMI VE TEKNOLOJİLERİ

Yapılandırılmış ya da yapılandırılmamış büyük hacimli verilerin istenilen sürede depolanması ve analiz edilmesi geleneksel yöntemlerle mümkün değildir. Bu nedenle büyük hacimli verilerin depolanması ve analizi için çeşitli yöntem ve teknolojiler geliştirilmiştir. Bu yöntemlerin başında dağıtık dosya sistemleri gelmektedir.

Dosya sistemleri depolama ve veri işleme uygulamalarının temelini oluşturmaktadır. Ancak, teknolojinin sağladığı olanakların artması ve ağ uygulamalarının gelişmesiyle birlikte, toplanan veri boyutu hızla artmaktadır. Bu durum geleneksel veri depolama ve işleme yöntemlerinin yetersiz kalmasına neden olmaktadır. Buna çözüm olarak ortaya çıkan dağıtık dosya sistemleri, sistem yükünün birden fazla düğüm üzerinde dağıtılması esasına dayanmaktadır. Böylece yüzlerce hatta binlerce düğüme dağıtılmış geniş bir depolama kapasitesi ve bant genişliği sunmaktadır. Bu yaklaşım kaynak yönetimi ve iş çizelgeleme ihtiyacını beraberinde getirmektedir. Çok sayıda düğüm üzerinde kaynakların yönetimi ve verinin güvenilir bir şekilde paralel olarak işlenebilmesi için Hadoop ekosistemi içerisinde geliştirilmiş YARN ve MapReduce gibi yazılımlar bulunmaktadır. Dağıtık depolanmış büyük veriye erişim ve verinin işlenmesi için de ayrıca araçlar geliştirilmiştir. Bu araçlara Pig, Spark, Hive, Hbase örnek olarak verilebilir. Bütün bu araçların birlikte çalışılabilirliğini sağlamak, veri akışlarını denetlemek ve güvenliği sağlamak üzere ZooKeeper, KNOX, Kafka, Flume gibi araçlar geliştirilmiştir.

2.1. Büyük Veri

Günümüzde, web teknolojileri, sosyal medya ve mobil ve algılayıcı cihazlardaki gelişmelerin sonucu olarak veri miktarı hızlı bir şekilde artmaktadır. Örneğin, Twitter günde 70M'in üzerinde tweet işleyerek günlük 8 TB'ın üzerinde işlem yapar [9]. ABI Research, 2020 yılına kadar 30 milyardan fazla etkileşimli cihazın olacağını tahmin

ediyor [10]. Bu büyük veriler, sağlık hizmetleri, biyoloji, ulaşım, çevrimiçi reklamcılık, enerji yönetimi ve finansal hizmetler gibi çeşitli alanlarda işletme değeri açısından büyük potansiyele sahiptir [11], [12]. Ancak, bu devasa veriyle karşılaştığınızda geleneksel yaklaşımlar yetersiz kalmaktadır.

Büyük veri kavramı, Gartner [13] tarafından bilgi keşfi, gelişmiş karar verme ve süreç optimizasyonu için yeni işlem paradigmaları gerektiren yüksek hacimli, yüksek hız ve çok çeşitlilikteki veri olarak tanımlanmaktadır. Bu açıklamaya göre büyük veri büyüklük ölçütü ile değil, geleneksel yaklaşımların, büyük verinin boyut, hız ve çeşitlilikleriyle bu verileri işlemede yetersiz kalması bakımından karakterize edilmektedir. Bu nedenle büyük verilerin işlenebilmesi geleneksel yaklaşımların iyileştirilmesi veya yenilerinin geliştirilmesine bağlıdır.

Geleneksel yaklaşım ve teknolojilerin üstesinden gelmesi gereken büyük verinin karakteristik özelliklerini Hacim(volume), Çeşitlilik(variety), Hız(velocity), Doğruluk(veracity) olmak üzere 4V şeklinde ifade edebiliriz.

2.1.1. Hacim (volume)

Büyük verinin en çok bahsedilen özelliği hacimdir. Hacim verinin miktarını, büyüklüğünü ve ölçüsünü ifade eder. Makine öğrenmesi bağlamında, boyut bir veri kümesindeki kayıtların veya örneklerin sayısıyla dikey olarak, ya da içerdiği özelliklerin veya niteliklerin sayısıyla yatay olarak tanımlanabilir. Ayrıca hacim verinin tipiyle de ilgilidir. Daha az sayıda çok karmaşık veri noktalarının daha büyük miktarda basit veriye eşdeğer olduğu düşünülebilir [1]. Büyük verinin tanımlanması en kolay boyutu gibi görünse de birçok zorluğun nedeni de aynı zamanda bu özelliğidir.

Bu zorlukları işlem performansı, modülerlik ve çok boyutluluk şeklinde ifade edebiliriz.

2.1.1.1. İşlem performansı

Büyük veri ile yapılan hesaplamalarda karşılaşılan temel zorluk işlem karmaşıklığıdır. Sonuç olarak ölçü büyüdükçe önemsiz işlemler bile maliyetli olmaktadır. Birçok makine öğrenmesi algoritması yüksek zaman karmaşıklığına sahiptir. Örneğin, standart destek vektörü makinesi (SVM) algoritması, eğitim süresi karmaşıklığı $O(n^3)$ ve bellek karmaşıklığı $O(n^2)$ şeklindedir [14]. Lojistik regresyon (Logistic Regression) $O(nm^2+m^3)$ şeklindedir [15]. Burada ki n eğitim örneklerinin sayısını m ise nitelik sayısını ifade etmektedir. Bu nedenle n'deki artış SVM algoritmasını eğitmek için gereken zaman ve bellek boyutunu büyük ölçüde etkileyecektir. Aynı şekilde n ve m deki artış Lojistik regresyon algoritmasının performansını önemli ölçüde etkilemektedir.

Ayrıca, veri boyutu arttıkça, algoritmaların performansı verileri depolamak ve taşımak için kullanılan mimariye daha bağımlı hale gelecektir. Paralel veri yapıları, veri bölümleme ve yerleştirme, verilerin yeniden kullanımı, veri büyüklüğündeki büyüme ile daha çok önem kazanmaktadır [16]. Esnek dağıtılmış veri kümeleri (RDD) [16], büyük kümelerdeki bellek içi hesaplamalar için yeni bir soyutlamaya örnektir. RDD'ler, Spark küme hesaplama çerçevesinde uygulanmaktadır [17].

2.1.1.2. Modülerlik

Pek çok öğrenme algoritması, işlenen verilerin tamamen bellekte veya bir diskte tek bir klasörde tutulabileceği varsayımına dayanmaktadır [18]. Birden fazla algoritma sınıfı, bu varsayımın geçerliliğine bağlı stratejiler ve yapı taşları üzerinde tasarlanır. Ancak veri boyutu bu ilkenin başarısızlığına yol açtığında, algoritmaların tümü etkilenmektedir [19]. Buna çözüm olarak öne sürülen yaklaşımlardan biri, çok sayıda düğümde paralel yürütme yoluyla büyük veri kümelerini işlemek için ölçeklenebilir bir programlama paradigması olan MapReduce'dur. Bazı makine öğrenme algoritmaları doğası gereği paraleldir ve MapReduce paradigmasına kolaylıkla adapte edilebilmektedirler. Ancak bazıları ise çok sayıda hesaplama düğümünden yararlanabilecek şekilde ayrıştırılması zordur. MapReduce paradigmasını kullanmaya

çalışırken modülerlik problemi ile karşılaşan üç ana algoritma kategorisi mevcuttur [20]. Bunlar Iterative Graph, Gradient Descent ve Expectation Maximization algoritmalarıdır [20]. Yinelemeli oluşları, hafıza içi verilere bağımlılıkları paralel ve dağıtık yapıya uygulanmalarını zorlaştırmaktadır. Dolayısıyla MapReduce veya başka bir dağıtık hesaplama mimarisine adapte olmaları zorlaşmaktadır.

2.1.1.3. Çok boyutluluk ve özellik mühendisliği

Büyük verinin hacmiyle ilgili olan diğer bir konu ise verinin çok boyutlu olmasıdır. Boyutluluk veri setinde bulunan özellik (features) ya da nitelik (attributes) sayısı olarak ifade edilebilir. İşlem performansında ifade edildiği gibi bir veri setindeki boyut sayısı arttıkça algoritmanın performansı azalmaktadır.

Çok boyutluluk ile yakından ilişkili olan özellik mühendisliği (feature engineering), makine öğreniminin daha iyi performans göstermesini sağlamak için alan bilgisini kullanan özellikler oluşturma ya da özellik azaltma süreci olarak ifade edilmektedir. En uygun özelliklerin seçilmesi makine öğreniminde en çok zaman alan işlemdir. Özellik mühendisliği makine öğrenme çıktılarını iyileştirmek için yeni özellikler eklemeyi amaçlarken, özellik seçimi (boyut indirgeme) en uygun özellikleri seçmeyi amaçlamaktadır. Özellik seçimi her ne kadar boyut sayısını azaltıp makine öğrenmesini hızlandırsa da, yüksek boyutlu verilerde açıklayıcı değişkenlerin rastlantısal hata payları ve yapay korelasyonlar bakımından makine öğrenmesini olumsuz yönde etkilemektedir [21].

Genel olarak, hem özellik seçimi hem de mühendislik büyük veri bağlamında hala çok önemlidir, ancak aynı zamanda karmaşıklaşmaktadırlar.

2.1.2. Çeşitlilik (variety)

Büyük veri için çeşitlilik, yalnızca bir veri kümesinin ve içerdiği veri türlerinin yapısal çeşitliliğini değil aynı zamanda neyi temsil ettiğini, anlamsal yorumunu [22] ve

kaynaklarını da temsil ettiği çeşitliliği tanımlamaktadır. Diğer V boyutlarında olduğu kadar olmasa da, bu boyutla ilişkili zorlukların önemli bir etkisi vardır.

2.1.2.1. Verinin konumu

Çeşitlilikle ilgili ilk zorluk veri konumudur [23]. Makine öğrenme algoritmaları veri kümesinin tümünün bellekte veya diskte tek bir dosyada bulunduğunu varsaymaktadır [19]. Ancak büyük veri için bu mümkün olmayabilir; veri belleğe sığmayabilir, aynı zamanda farklı fiziksel konumlarda bulunan çok sayıda dosyaya dağıtılmış olabilir.

Geleneksel makine öğreniminde veri işleneceği konuma aktarılmak istenir. Söz konusu büyük veri kümeleri olunca, aktarım gecikmelere ve yoğun ağ trafiğine neden olabilir. Sonuç olarak hesaplama konumuna veri getirmek yerine veri konumuna hesaplamayı götürme yaklaşımı ortaya çıkmıştır. Zaman ve bant genişliği kullanımı göz önüne alındığında hesaplamayı taşımak veriyi taşımaktan daha az maliyetli olmaktadır. MapReduce paradigması da bu yaklaşımı kullanmaktadır. Map görevleri, verilerin bulunduğu düğümlerde çalışmaktadır [24].

Küçük veri setlerinde fiziksel konum önemli değildir, ancak büyük veri için aynı şey söylenemez. Hesaplamalar ile işlenecek verinin farklı konumlarda olması uygulanacak hesaplamalarda performansın düşmesine sebep olmaktadır. Bu nedenle MapReduce temelli yaklaşımlar yüksek yinelemeli algoritmaların kullanıldığı uygulamalarda güçlüklerle karşılaşmaktadır.

2.1.2.2. Veri farklılıkları

Büyük veri analizi, çoğu zaman çeşitli kaynaklardan gelen çeşitli verilerin birleştirilmesini gerektirir. Bu veriler, tür, format, veri modeli ve anlamsal açıdan farklı olabilirler. İki ana heterojenite kategorisi tanınabilir; sözdizimsel ve anlamsal heterojenite.

Sözdizimsel heterojenite, veri tiplerindeki, dosya formatlarındaki, veri kodlamadaki, veri modelindeki vb. çeşitliliği ifade eder. Bu şekilde bir araya getirilmiş veri setleri ile analitik yapmak için bu sözdizimsel varyasyonların çözümlenmesi gerekir [22]. Bu nedenle makine öğrenmesi, verileri belirli bir modele uyacak şekilde yapılandırmak için genellikle bir veri ön işleme ve temizleme adımı gerektirmektedir.

Anlamsal heterojenite anlam ve yorumlardaki farklılıkları ifade eder. Farklı taraflarca geliştirilen bir dizi veri kümesi birleştirildiğinde oluşan büyük veriye de bu farklılıklar aktarılmış olacaktır. Bu nedenle veriler birleştirilmeden bu farklılıkların giderilmesi gerekmektedir. Anlamsal olarak farklı verileri işlemek için makine öğrenmesi yaklaşımları geliştirilmemiştir [25].

2.1.3. Hız (velocity)

Büyük verinin hız boyutu yalnızca verilerin üretildiği hızı değil, aynı zamanda analiz edilmeleri gereken oranı ifade eder. Akıllı telefonlar ve gerçek zamanlı algılayıcıların her yerde bulunması, akıllı evler gibi teknolojilerin geliştirilmesi çevremizle hızlı bir şekilde etkileşime girme ihtiyacını oluşturmaktadır. Bu nedenle büyük verinin hız boyutu göz önünde bulundurulması gereken önemli bir faktör olmaktadır.

2.1.3.1. Verinin kullanılabilirliği

Geleneksel makine öğrenmesi kavramı verinin kullanılabilirliğine bağlıydı. Yani öğrenmeye başlamadan önce tüm veri kümesinin mevcut olduğu anlamına geliyordu. Ancak veri akışı bağlamında tüm verinin hazır olması sağlanamaz.

Makine öğreniminde, bir model genellikle eğitim setinden öğrenir daha sonra öğrenilen bu görevi yeni veriler üzerinde gerçekleştirir. Ancak veri akışının olduğu bir sistemde yeni gelen verilerden öğrenemez, sadece önceden öğrendiği görevi yeni gelen veriler üzerinde gerçekleştirir. Bu durum modelin gelen yeni verilerle yeniden eğitilmesini gerektirir. Bu nedenle, yeni bilgilere adapte olmak için, algoritmalar bazen bir ardışık öğrenme olarak adlandırılan artımlı öğrenmeyi desteklemelidirler, bu da bir

algoritmanın yeni verilerin gelmesine dayanarak öğrenmeye adapte edilmesine gerek kalmadan bir algoritmanın öğrenmesini uyarlama kabiliyeti olarak tanımlanmaktadır [26].

2.1.4. Doğruluk (veracity)

Büyük verinin doğruluğu yalnızca veri setini oluşturan verilerin güvenilirliğini değil aynı zamanda IBM'in tanımladığı gibi veri kaynaklarının doğal güvenilmezliğini ifade eder [1]. Büyük verinin kaynağı ve kalitesi birlikte Doğruluk bileşenini [27] tanımlamaktadır.

2.1.4.1. Veri kaynağı tespiti

Veri kaynağı tespiti, verilerin kaynağını ve konumlar arasındaki hareketlerini izleme ve kaydetme işlemidir [28]. Kaydedilen veriler, işlem hatasının kaynağını tespit etmek için kullanılmaktadır. Hatanın kaynağını belirlemek hatalı verinin bütün geçişlerinin ve işlemlerinin belirlenmesi sağlamaktadır. Bu nedenle bu meta verileri yakalamak ve saklamak önemlidir [22]. Bu veriler makine öğrenmesine ciddi bir katkı sağlarken kendisi de aynı zaman büyük bir veri olarak karşımıza çıkmaktadır. Dahası, sadece çok büyük değil, aynı zamanda bu ek yükü taşımanın hesaplama maliyeti de ciddi boyutlara ulaşmaktadır [22].

Ayrıca veri artık hayatımızın çeşitli yönleriyle ilgili farklı şekillerde toplanmaktadır. Bununla birlikte, veri toplamak için kullanılan araçlar ve yöntemler belirsizlikler içerebilmektedir. Bu da bir veri kümesinin doğruluğunu etkileyebilmektedir. Örneğin duyarlılık verileri sosyal medya aracılığıyla toplanmaktadır [29], ancak bu veriler çok önemli olmasına rağmen öznel bilgilere dair değerli bilgiler içerdiklerinden, verilerin kendileri kesin değildir. Bu tür verilerin kesinliği de objektif değildir, çünkü yalnızca insanın yargısına dayanmaktadır [30]. Verilerde nesnellik veya mutlak doğruluk eksikliği, bir makine öğrenme algoritmasının ondan öğrenmesini zorlaştırmaktadır.

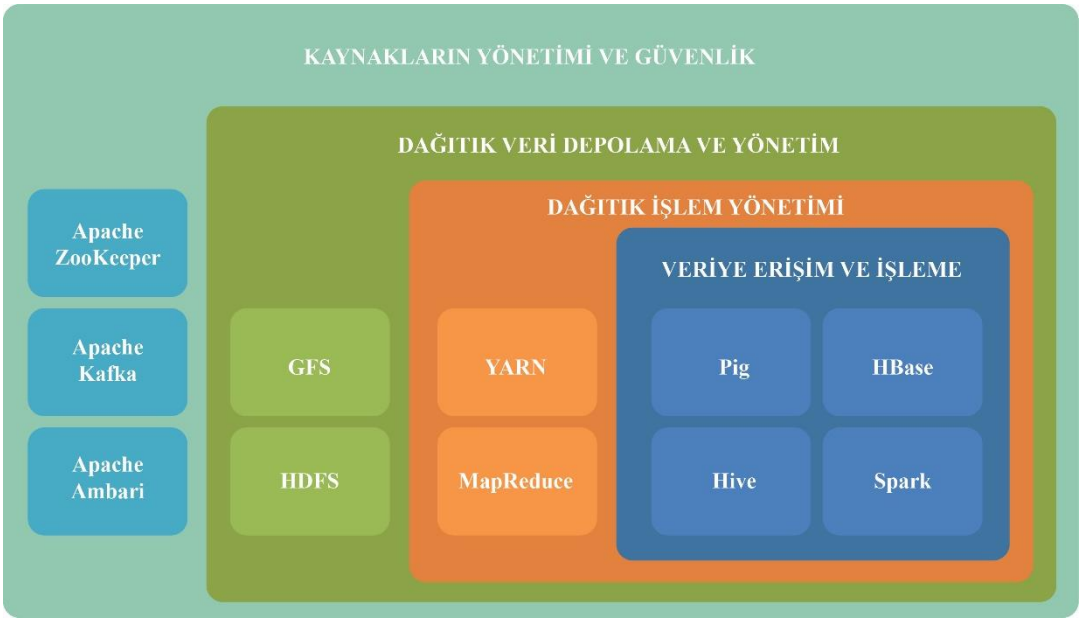
2.2. Apache Hadoop

Apache Hadoop, güvenilir, ölçeklenebilir, dağıtık veri işlem için geliştirilmiş açık kaynak kodlu bir yazılımdır. Apache Hadoop yazılım kütüphanesi, büyük veri kümelerinin basit programlama modelleri kullanarak bilgisayar kümeleri arasında dağıtık çalıştırılmalarını sağlayan bir çerçevedir. Tek makineden binlerce makineye kadar ölçeklenebilecek şekilde tasarlanmıştır. Yüksek kullanılabilirlik sağlamak için donanıma güvenmek yerine, kütüphanenin kendisi uygulama katmanındaki hataları saptamak ve ele almak için tasarlanmıştır, bu nedenle her biri arızalara açık olabilecek bir bilgisayar kümesinin üzerine yüksek düzeyde kullanılabilir bir hizmet sunmaktır.

2.2.1. Apache hadoop mimarisi

Apache Hadoop basit bir programlama modeli kullanarak, binlerce basit yapıdaki makineyi birbirine bağlayarak devasa boyuttaki veriyi depolamak ve işlemek için kullanan açık kaynak kodlu bir yazılım çerçevesidir. Apache Hadoop projesi temel olarak; dosya sistemi için HDFS, iş planlama ve küme kaynak yönetimi için YARN (Yet Another Resource Negotiator) ve büyük veri setlerinin paralel işlenmesi için veri işleme aracı olarak MapReduce'den oluşmaktadır. Bunların dışında yine açık kaynak kod olarak geliştirilmiş olan, dağıtık uygulamalar için yüksek performanslı bir koordinasyon hizmeti sunan ZooKeeper, veri özetleme ve sorgulama sağlayan bir veri ambarı altyapısı olan Hive, Hadoop verileri için hızlı ve ölçeklenebilir bir makine öğrenmesi ve veri madenciliği kütüphanesi Spark projeleri bulunmaktadır. Tüm bu projeler için destekler içeren, Apache Hadoop kümelerinin kurulumu, yönetimi ve izlenmesi için geliştirilmiş web tabanlı bir proje olan Ambari gibi projeler ayrıca Apache Hadoop tarafından desteklenmektedir.

Apache Hadoop ekosistemini, Dağıtık Veri Depolama ve Yönetim, Dağıtık İşlem Çerçevesi, Veri Erişim Motorları, Operasyon Yönetim ve Güvenlik şeklinde sıralayabilir.



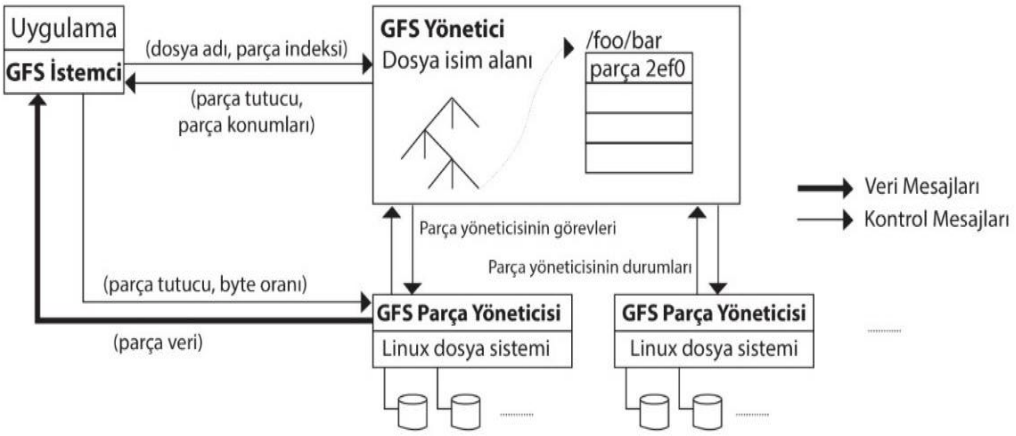
Şekil 2.1. Büyük veri depolama ve yönetim mimarisi

2.3. Dağıtık Veri Depolama ve Yönetim

2.3.1. Google dosya sistemi (gfs)

Google'ın hızla veri işleme ihtiyaçlarının karşılanması için Google Dosya Sistemi (GFS) uygulanmaktadır. GFS, çok sayıda makineden oluşan kümeler üzerinde dağıtık olarak verilerin işlenmesine olanak tanıyacak bir dosya sistemidir [31]. GFS tabanlı uygulamaların geliştirilmesini kolaylaştırmak için, dosya sistemi bu dağıtım ve yönetim yönlerinden soyutlamayı amaçlayan bir programlama arayüzü sağlamaktadır. GFS Google gereksinimlerine göre özel olarak tasarlanmış bir dosya sistemidir. Büyük veri işlemenin yanı sıra GFS, büyük hacimli veri akışlarının okunması ve optimize edilmesi için tasarlanmıştır.

Şekil 2.2.'de görüldüğü gibi bir GFS kümesi bir ana makineden ve birçok parça denetleyiciden (chunkserver) oluşur. Parça denetleyicilere birçok istemci erişebilir durumdadır. Bu istemcilerin her biri kullanıcı seviyesinde sunucu işlemlerini yürüten Linux makineleridir. Makine kaynakları uygulamaları çalıştırabildiği sürece hem parça denetleyiciyi hem de istemciyi aynı makine çalıştırmak mümkündür.



Şekil 2.2. Google dosya sistemi mimarisi [31]

Dosyalar sabit 64 MB parçalardan oluşmaktadır. Her parça oluşturulurken ana makine tarafından atanan sabit ve genel olarak benzersiz olan 64 bit parça tutucu (chunkhandler) ile tanımlanmaktadır. Parça sunucuları, yerel disklerde Linux dosyaları olarak saklanmaktadır. Verinin güvenliği için her bir yığın birden fazla sunucuda çoğaltılmaktadır. Varsayılan olarak en az üç kopya saklanmaktadır. Ancak kullanıcılar farklı kopyalama seviyeleri belirleyebilmektedir. Ana makine tüm dosya sisteminin meta verilerini barındırmaktadır. Bu meta veriler, isim alanlarını, erişim kontrol bilgilerini, dosyalardan parçalara eşlemeyi ve parçaların geçerli konumlarını içermektedir. Ana makine durum bilgilerini toplamak için periyodik olarak parça sunucularına mesajlar göndermektedir. İstemciler meta veri işlemleri için ana makine ile etkileşime girer ancak tüm veri haberleşmesi parça sunucularla gerçekleştirilmektedir. Birçok uygulama büyük dosyaların önbelleğe alınmasını gerektirir ancak istemci ve parça sunucu yeterli önbellek sağlayamayacağından dosya verileri önbelleğe alınmamaktadır. Dosyaların önbelleğe alınmadan sadece meta verilerin önbelleğe alınması sayesinde önbellek sorunları ortadan kaldırılarak sistem basitleştirilmiştir [31].

2.3.2. Hadoop dağıtık dosya sistemi (hdfs)

Hadoop Dağıtık Dosya Sistemi (HDFS), temel donanım üzerinde çalışmak üzere tasarlanmış dağıtılmış bir dosya sistemidir. HDFS'nin mevcut dağıtık dosya

sistemleriyle benzerlikleri vardır, ancak bazı yönleriyle öne çıkmaktadır. HDFS, hataya dayanıklıdır ve düşük maliyetli donanımlara dağıtılmak üzere tasarlanmıştır. Uygulama verilerine yüksek verimli erişim sağlar ve büyük veri kümelerine sahip uygulamalar için uygundur. HDFS başlangıçta Apache Nutch web arama motoru projesi için altyapı olarak geliştirildi ancak şu an bir Apache Hadoop alt projesidir.

HDFS, her biri dosya sistemi verilerinin bir bölümünü depolayan yüzlerce veya binlerce sunucu makinesinden oluşabilir. Çok sayıda bileşen olması ve her bir bileşenin başarısızlık olasılığına sahip olması, HDFS'nin bazı bileşenlerinin her zaman işlevsel olmadığı anlamına gelmektedir. Bu nedenle, arızaların tespiti ve hızlı bir şekilde giderilmesi HDFS'nin temel mimari hedeflerinden biridir.

HDFS'de çalışan bazı uygulamaların veri kümelerine akış erişimi gerektirmektedir. Bu nedenler HDFS, kullanıcılar tarafından etkileşimli kullanım yerine toplu işleme daha uygun tasarlanmıştır. Burada ki amaç veri erişiminin gecikme süresini düşürmekten ziyade yüksek veri erişimi sağlamaktadır.

HDFS'de çalışan uygulamalar büyük veri kümelerine sahiptir. Bu nedenle HDFS büyük dosyaları destekleyecek şekilde tasarlanmıştır. Bu sayede yüksek bant genişliği sağlar ve bir kümede yüzlerce düğüme ölçeklenebilmektedir.

HDFS uygulamaları, dosyalar için bir defa-oku-çok defa eriş modeline ihtiyaç duymaktadır. Yani bir kez oluşturulmuş, yazılmış ve kapatılmış bir dosyanın değiştirilmesine gerek yoktur. Bu varsayım, veri tutarlılığı sorunlarını basitleştirir ve yüksek verimli veri erişimi sağlamaktadır.

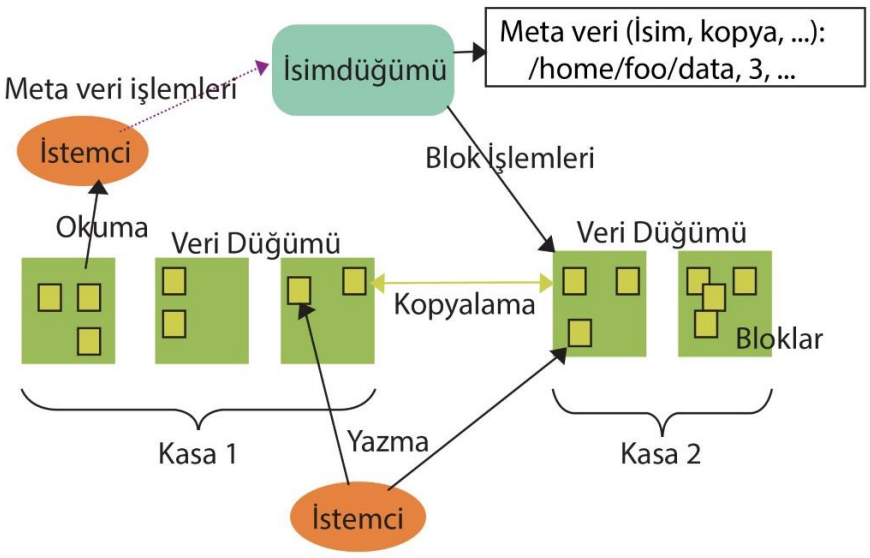
Bir uygulama tarafından talep edilen bir hesaplama, üzerinde çalıştığı verinin yakınında yürütülürse çok daha verimlidir. Bu özellikler veri çok büyük boyutlarda olduğunda daha etkilidir. Aynı zamanda ağ tıkanması en aza indirgenir ve sistemin verimliliğini arttırmaktadır. HDFS'de ki varsayım, veriyi hesaplama yapılacak yere taşımak yerini hesaplamaya verinin yakınına taşımanın daha verimliği olduğudur.

Ayrıca HDFS, bir platformdan diğerine kolayca taşınabilmektedir. Bu da, HDFS'nin geniş bir uygulama grubu için tercih edilen bir platform olarak yaygın şekilde benimsenmesini sağlamaktadır.

2.3.2.1. Hdfs mimarisi

Bir HDFS kümesi, Şekil 2.3.'te görüldüğü gibi dosya sistemi isim alanını yöneten ve istemciler tarafından dosyalara erişimi düzenleyen bir ana sunucu, bir NameNode (İsim Düğümü) ve her düğümde depolamayı yöneten bir DataNode (veri düğümü) mevcuttur. İsim Düğümü(NameNode) ve veri düğümü (DataNode), temel donanıma sahip makinelerde çalışacak şekilde tasarlanmış yazılım parçalarıdır.

Bir dosya bir veya daha fazla bloğa bölünür ve bu bloklar kümeyi oluşturan düğümlerde (DataNodes) saklanmaktadır. NameNode, dosya ve dizinlerin açılması, kapatılması ve yeniden adlandırılması gibi dosya sistemi isim alanı işlemlerini gerçekleştirmektedir. Ayrıca blokların veri düğümleri ile eşlenmesini de belirlemektedir. Veri düğümleri (DataNodes), dosya sisteminin istemcilerinden okuma ve yazma istekleri sunmaktan sorumludur. Ayrıca, NameNode'dan gelen talimat üzerine blok oluşturma, silme ve çoğaltma işlemini gerçekleştirmektedir.



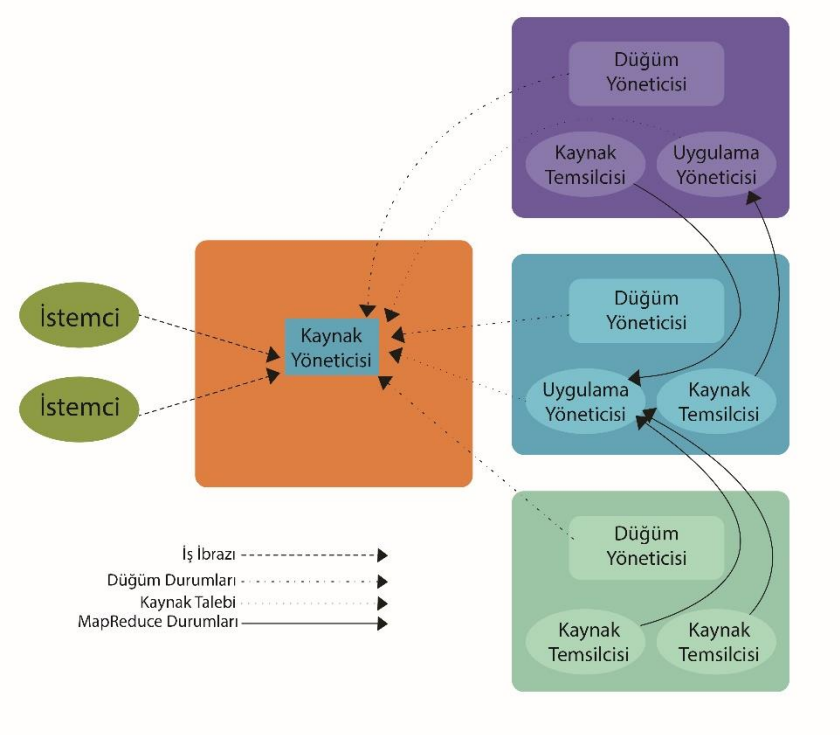
Şekil 2.3. Hadoop dosya sistemi mimarisi [32]

2.4. Dağıtık İşlem Yönetimi

2.4.1. Hadoop YARN

YARN'ın ortaya çıkmasındaki temel fikir, kaynak yönetimi ve iş çizelgeleme/izleme işlevlerini ayrı ayrı bölümlere ayırmaktır. Burada ki amaç küme başına bir kaynak yöneticisi ve her uygulama için bir uygulama yöneticisine (ApplicationMaster) sahip olmaktır. Bir uygulama tek bir iş olabileceği gibi bir yönlü düz ağaç (DAG-Directed Acyclic Graph) işler dizisi de olabilmektedir. Kaynak yöneticisi ve düğüm yöneticisi, veri hesaplama çerçevesini oluşturmaktadır. Şekil 2.4.'te gösterildiği gibi kaynak yöneticisi, sistemdeki tüm uygulamalar arasında kaynakları paylaştıran tek merkezdir. Düğüm yöneticisi düğümlerin kaynak kullanımlarını (CPU, RAM, Disk, Ağ) izleyen ve bunları kaynak yöneticisine bildiren bir araçtır. Her uygulama için oluşturulan uygulama yöneticisi (ApplicationMaster), çerçeveye özgü bir kütüphanedir. Kaynak yöneticisi tarafından kaynakların kullanımını yürütmek ve izlemek için düğüm yöneticisi (NodeManager) ile birlikte çalışmakla görevlendirilmektedir.

Kaynak yöneticisinin, planlayıcı ve uygulama yöneticisi olmak üzere iki ana bileşeni vardır. Zamanlayıcı, bilinen kapasite kısıtları ve kuyruklar gibi koşullara bağlı olarak çalışan çeşitli uygulamalara kaynak tahsis etmekten sorumludur. Uygulama yöneticisi işleri kabul etmekten, uygulama için üretilmiş uygulama yöneticisini (ApplicationMaster) ilk yükleniciye atamaktan ve eğer hata ile karşılaşırsa yeniden başlatmaktan sorumludur.



Şekil 2.4. Yarn mimarisi [33]

2.4.2. Hadoop mapreduce

Hadoop MapReduce, büyük miktarlarda veriyi temel donanımdan oluşan binlerce düğüm üzerinde, güvenilir ve hataya dayanıklı bir şekilde paralel olarak işleyen uygulamaların kolayca yazılması için oluşturulmuş yazılım çerçevesidir.

2.5. Kaynakların Yönetimi ve Güvenlik

2.5.1. Apache zookeeper

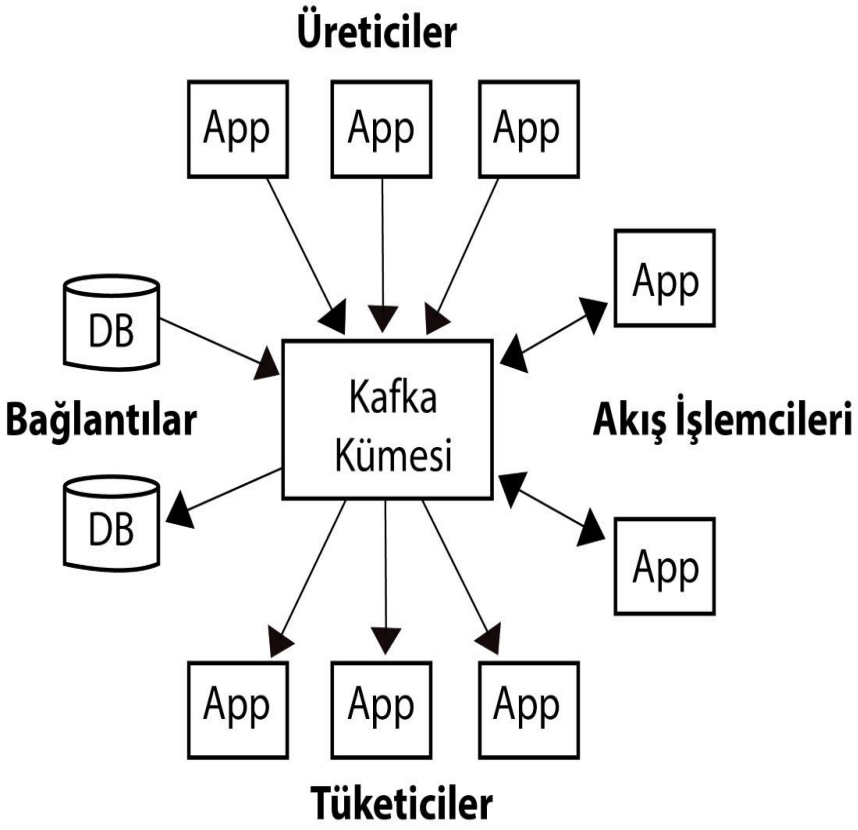
Apache ZooKeeper, dağıtık uygulamalar için yüksek performanslı bir koordinasyon servisidir. Adlandırma, yapılandırma yönetimi, eşleme ve grup servisleri gibi genel servisleri için basit bir arayüz sunmaktadır. Uygulamaya özel ihtiyaçlar için, grup yönetimi, lider seçimi ve varlık protokolleri gibi servisleri hazır bir şekilde kullanıma sunmaktadır.

Apache ZooKeeper, dağıtık işlemlerin standart bir dosya sistemi gibi bir hiyerarşik isim uzayı aracılığıyla birbirleriyle koordine olmalarını sağlamaktadır. İsim alanı, dosya ve dizinlere benzeyen znode adı verilen kaydediciler bulundurmaktadır. Tipik bir dosya sisteminden farklı olarak, ZooKeeper verileri bellekte tutulur, bu da ZooKeeper'ın yüksek verimlilik ve düşük gecikme süresi elde edebileceği anlamına gelmektedir.

2.5.2. Apache kafka

Apache Kafka dağıtık veri akış platformudur. Kafka temel olarak sistemler ya da uygulamalar arasında güvenilir şekilde veri alan gerçek zamanlı akışlı veri hatları oluşturmak ve bu verileri dönüştüren veya bunlara tepki veren gerçek zamanlı akış uygulamaları oluşturmak için kullanılmaktadır. Kafka, birden fazla veri merkezine yayılabilen bir veya daha fazla sunucuda küme olarak çalıştırılmaktadır. Kafka kümesi, konu (*topic)* adında kategorilerde kayıt akışlarını saklamaktadır. Her kayıt bir anahtar, bir değer ve bir zaman damgasından oluşmaktadır.

Apache Kafka platformunun dört temel uygulama programlama arayüzü (API) vardır. Şekil 2.5.'te yer alan Producer (Üretici), bir uygulamanın bir veya daha fazla Kafka konusuna bir kayıt akışı yayınlamasına izin vermektedir. Consumer (Tüketici) bir uygulamanın bir veya daha fazla konuya abone olmasını ve kendilerine üretilen kayıt akışını işlemesini sağlamaktadır. Streams (Akış), bir uygulamanın bir akış işlemcisi olarak hareket etmesine, bir veya daha fazla konudan bir girdi akışını tüketmesine ve bir veya daha fazla çıkış konularına bir çıkış akışı oluşturmasına izin vererek giriş akışlarını çıkış akışlarına etkili bir şekilde dönüştürmektedir. Connector (Bağlayıcı), Kafka konularını mevcut uygulamalara veya veri sistemlerine bağlayan yeniden kullanılabilir üreticiler veya tüketiciler oluşturmaya ve çalıştırmaya izin vermektedir.



Şekil 2.5. Apache kafka mimarisi [34]

2.5.3. Apache ambari

Apache Ambari, Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig ve Sqoop gibi Hadoop araçları için destek içeren Apache Hadoop kümelerini hazırlama, yönetme ve izleme için geliştirilmiş web tabanlı bir araçtır. Ambari, Hadoop kümelerinin hazırlanmasını, Hadoop hizmetlerinin yüklenmesini, Hadoop küme servislerinin yapılandırılmasını, başlatılmasını ve durdurulmasını sağlamaktadır. Ayrıca Ambari kontrol paneli sayesinde Hadoop kümesinin durumu, Ambari Metrics System ile sistem kaynaklarının kullanımını ve Ambari Alert Framework ile bakım ihtiyaçlarının giderilmesi için uyarı hizmeti sunmaktadır.

2.6. Dağıtık Veriye Erişim ve İşleme

2.6.1. Apache pig

Apache Pig, büyük veri setlerini analiz etmek hazırlanmış yüksek seviyeli bir dille yazılmış analiz programlarını alt yapı ile eşleştirerek değerlendiren bir platformdur. Pig programlarının paralel çalışmaya müsait olması büyük veri kümelerini yönetmeyi kolaylaştırmaktadır. Pig, çok sayıda birbiriyle ilişkili veri dönüşümünden oluşan karmaşık görevler, açıkça veri akış dizileri olarak kodlanır, böylece yazılması, anlaşılması ve sürdürülmesi kolaylaşmaktadır. Görevlerin kodlanma şekli, sistemin yürütmeyi otomatik olarak optimize etmesine izin verir ve kullanıcının verimlilik yerine anlambilimine odaklanmasını sağlamaktadır. Ayrıca kullanıcılar, kendi fonksiyonlarını oluşturabilir.

2.6.2. Apache hive

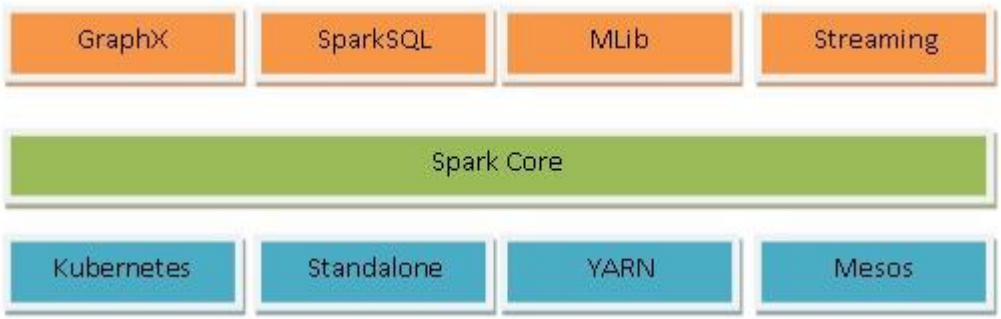
Hive SQL kullanarak dağıtılmış depolamada bulunan büyük veri setlerinin okunmasını, yazılmasını ve yönetilmesini kolaylaştıran bir Apache projesidir. SQL üzerinden verilere kolay erişim sağlayan bir araç olan Hive, çıkarma / dönüştürme / yükleme (ETL- extract/transform/load), raporlama ve veri analizi gibi veri depolama görevlerini mümkün kılmaktadır. Çeşitli veri yapılarını istenen formata dönüştürmeyi kolaylaştırmaktadır. Apache HDFS veya Apache HBase gibi diğer veri depolama sistemlerinde saklanan dosyalara erişimi sağlamaktadır.

2.6.3. Apache hbase

Apache Hbase, büyük veri tabloları için yapılandırılmış veri depolamayı destekleyen ölçeklenebilir, dağıtılmış bir veri tabanıdır. Apache HBase, açık kaynaklı, dağıtılmış, ilişkisel olmayan bir veri tabanıdır.

2.6.4. Apache spark

Apache Spark, genel amaçlı bir veri işleme motorudur. Büyük veri kümelerindeki etkileşimli sorgular, sensörlerden veya finansal sistemlerden gelen veri akışlarının işlenmesi ve makine öğrenmesi işlemleri günümüzde çoğunlukla Spark ile ilişkilendirilmektedir. Ayrıca Spark'ın kapsamlı geliştirici kütüphanelerinden ve API'lerinden yararlanılarak, Java, Python, R ve Scala gibi diller için sağladığı desteği kullanılarak diğer veri işleme görevleri için de kullanabilmektedir. Spark, Hadoop'un veri depolama birimi HDFS ile birlikte kullanılır, ancak HBase, Cassandra, MongoDB ve Amazon S3 gibi diğer popüler veri depolama sistemleriyle de aynı derecede entegre edilebilmektedir. Şekil 2.6.'da Apache Spark mimarisi yer almaktadır. SQL için Spark SQL, makine öğrenmesi için Spark MLib, grafik tabanlı hesaplamalar için GraphX ve Spark Streaming ile yapılandırılmış veri işleme dâhil olmak üzere zengin bir yüksek seviyeli araç setini desteklemektedir.



Şekil 2.6. Apache spark mimarisi

Spark uygulamaları yerel olarak veya bir kümeye, etkileşimli bir Shell kullanarak veya bir uygulama göndererek dağıtılabilmektedir. Bir kümeye dağıtılmış uygulamaları çalıştırmak için, Spark bir küme yöneticisine ihtiyaç duymaktadır. Küme yöneticisi olarak, Hadoop MapReduce ve servis uygulamalarını da çalıştırabilen Mesos, Hadoop 2 sürümü ile birlikte geliştirilen YARN ve uygulama paketlerinin ölçeklendirilmesini ve yönetimini otomatikleştirmek için açık kaynaklı bir sistem olan Kubernetes kullanılabilmektedir.

2.7. Apache Spark: Temel Kavramlar Ve Mimarisi

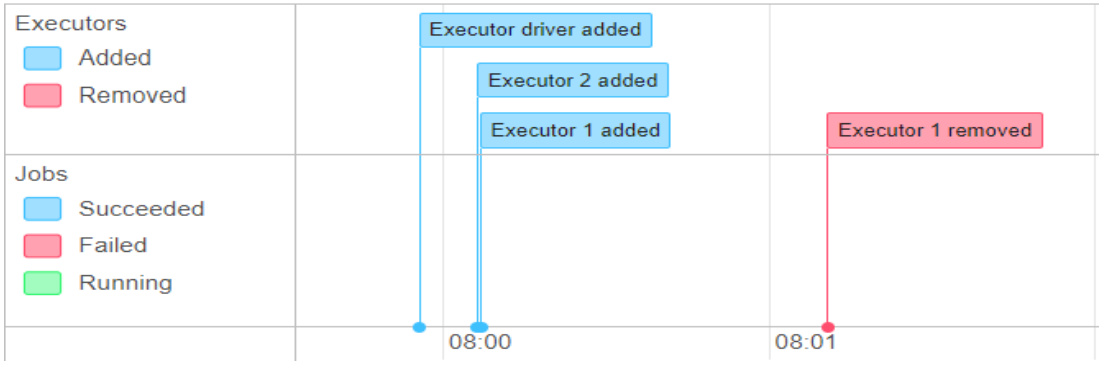
Apache Spark, Apache Hadoop kümelerinde genel amaçlı veri işleme için MapReduce'un alternatifi olarak kabul edilmektedir. MapReduce uygulamaları gibi, her Spark uygulaması da bir sonucu hesaplamak için kullanıcı tarafından sağlanan kodu sunucu bilgisayar tarafında çalıştıran yazılımdır. MapReduce'ta olduğu gibi, Spark uygulamaları da birden fazla ana bilgisayarın kaynaklarını kullanabilmektedir. Ancak, Spark'ın MapReduce'a göre birçok avantajı vardır.

MapReduce'ta en üst hesaplama birimi bir iş (job)'tir. Bir iş, veri okuma, map işlevi, reduce işlevi ya da verinin diske yazılması olabilmektedir. Spark'ta ise üst düzey hesaplama birimi bir uygulamadır. Bir Spark uygulaması tek bir yığın iş için kullanılabildiği gibi çok işlevli etkileşimli oturumlarda ve uzun süreli hizmet taleplerinde de kullanılabilmektedir. MapReduce her görev için bir işlem başlatır. Buna karşılık, bir Spark uygulamasının, bir görev yapmasa bile adına çalışan işlemleri olabilmektedir. Ayrıca, aynı yürütücü (Executor) içinde birden fazla görev yürütebilmektedir. Spark, bellek içi depolamayı mümkün kıldığından her görev için MapReduce'a göre çok daha hızlı görev başlangıç zamanı sağlamaktadır.

2.7.1. Spark yürütme modeli

Bir Spark uygulaması yürütme aşamasında, sürücü (Driver), yürütücü (Executor), görev (Task), iş (Job), DAG (Directed Acyclic Graph) ve aşama (Stage) olmak üzere altı temel kavramdan bahsedilebilir. Çalışma zamanında, bir Spark uygulaması Şekil 2.7.'de gösterildiği gibi bir sürücü (driver) işlemine ve bir kümedeki ana bilgisayarlara dağıtılmış bir dizi yürütücü (Executor), işlemine eşlenir. Sürücü (driver), iş (job) akışını düzenler ve her iş akışı için DAG aşamaları hesaplanır. Şekil 2.8.'de yürütme esnasında oluşan aşamalar yer almaktadır. Bu aşamalar görev (Task) zamanlayıcısına gönderilir ve işleri çalıştırmak için en kısa süre hesaplanır. Sürücü (driver) uygulamanın çalıştığı süre boyunca aktiftir. Genellikle sürücü (driver), bir işi başlatmak için kullanılan istemci ile aynıdır. Şekil 2.9.'da ihtiyaç duyulması haline eklenen yürütücüler ve atanan işler yer almaktadır. YARN kullanıldığında sürücü

küme içerisinde çalışabilir. Ancak etkileşimli modda sürücü Shell'in kendisidir. Yürütücüler işleri görevler formunda çalıştırmaktan ve ön belleğe alınmış verileri depolamaktan sorumludur. Yürütücünün ömrü, dinamik ayırmanın etkin olup olmamasına bağlıdır. YARN dinamik ayırma özelliğine sahiptir. Küme yöneticisi olarak YARN kullanıldığında sürücü başlangıçta birden fazla yürütücü çalıştırır ancak kullanılmadığında yürütücü durdurulur.



Şekil 2.7. Etkileşimli arayüzde başlatılan pyspark uygulaması sonucu driver ve executor durumları

Bir Spark uygulaması, bir işin başlatılması için bir eylem gerçekleştirir. Bu eylemin ilişkili olduğu veri kümesi incelenir ve buna göre bir yürütme planı oluşturulur. Veri kümesine göre yürütme eylemi aşamalara ayrılır. Aşama, her biri farklı bir veri kümesinde aynı kodu çalıştıran görevler topluluğudur.

Stages for All Jobs

Active Stages: 1
Pending Stages: 1
Completed Stages: 3

Fair Scheduler Pools (1)

Pool Name	Minimum Share	Pool Weight	Active Stages	Running Tasks
default	0	1	1	2

Active Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input
3	default	count at NativeMethodAccessorImpl.java:0 +details (kill)	2019/04/16 08:15:23	2 s	0/2 (2 running)	12.4 MB

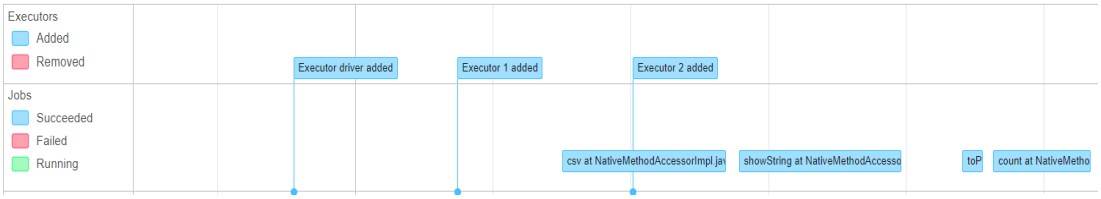
Pending Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Outp
4	default	count at NativeMethodAccessorImpl.java:0 +details	Unknown	Unknown	0/1		

Completed Stages (3)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input
2	default	toPandas at <ipython-input-6-69f767f9c5fc>:1 +details	2019/04/16 08:15:22	0.7 s	1/1	25.5 MB
1	default	showString at NativeMethodAccessorImpl.java:0 +details	2019/04/16 08:15:13	6 s	1/1	25.5 MB
0	default	csv at NativeMethodAccessorImpl.java:0 +details	2019/04/16 08:15:07	6 s	1/1	25.5 MB

Şekil 2.8. Yürütme esnasına oluşan aşamalar



Şekil 2.9. Yürütme esnasında yürütücü (executor) eklenmesi ve iş durumları

2.7.2. Rdd

RDD (Esnek Dağıtılmış Veri Kümesi), kümenin farklı düğümünde hesaplanan değişmez bir nesne koleksiyonu olan Apache Spark'ın temel veri yapısıdır. Spark RDD'deki her veri kümesi, birçok sunucuda mantıklı bir şekilde bölümlenir, böylece kümenin farklı düğümlerinde hesaplanabilirler.

RDD soy grafiği olan DAG yardımı ile hatalara karşı dayanıklıdırlar bu sayede düğüm arızalarından dolayı eksik ve hasarlı bölümler yeniden hesaplanabilir. RDD'ler verilerin birden fazla düğüm üzerinde dağıtılması sağlanmaktadır. Birçok formattaki veri setleri herhangi bir araca ihtiyaç duymadan kolayca harici depolama birimlerinde okunur ve RDD veri yapısına dönüştürülür. Bir RDD oluşturulduktan sonra değiştirilemez, ancak başka bir RDD'ye dönüştürülebilir. Bu dönüşümler Spark kullanılan operasyonlardan biri olan transformasyonlar (transformation) sayesinde gerçekleştirilir. Diğer Spark fonksiyonu olan aksiyonlar (Action) ise RDD'ler üzerinde gerçekleştirilen hesapların sürücüye (driver) gönderilmesini sağlar.

BÖLÜM 3. MAKİNE ÖĞRENMESİ İÇİN KULLANILACAK UYGULAMA ORTAMI VE HAZIRLANMASI

Makine öğrenmesi, yapay zekâ, istatistik, matematik gibi birçok bilim dalını birleştiren disiplinler arası bir araştırma alanıdır. Makine öğrenmesinin odağında hızlı ve etkili öğrenme algoritmalarının geliştirilmesi bulunmaktadır. Günümüzde geliştirilmiş olan makine öğrenmesi algoritmaları ile mevcut veri analiz teknolojileri kullanılarak büyük veri olarak tanımlanan yapılandırılmış ya da yapılandırılmamış verilerde makine öğrenmesinin istenilen sürede gerçekleştirilmesi konusunda yetersiz kalmaktadırlar. Bu duruma çözüm olarak geliştirilen en önemli yaklaşımlardan biri olan dağıtık veri işleme sayesinde görevler bir kümede bulunan sistemler üzerinden dağıtılarak gerçekleştirilir. Kapsamlı kütüphaneleri ve Java, Python, Scala ve R dilleri ile uygulama geliştirmeyi sağlayan Apache Spark MLib ile büyük veride makine öğrenmesi uygulamaları gerçekleştirilebilmektedir.

Bu çalışmada bir Google Cloud Platform hizmeti olan Dataproc ile oluşturulmuş Spark kümesi üzerinde, Jupyter Notebook arayüzü kullanılarak Pyspark ile makine öğrenmesi uygulamaları gerçekleştirilmiştir.

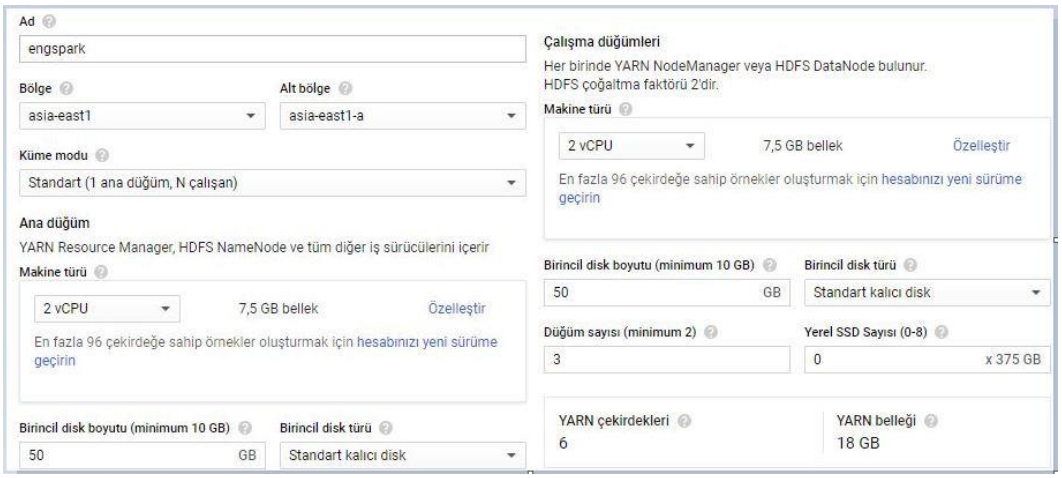
3.1. Çalışma Ortamının Hazırlanması

Google Cloud Platform (GCP) hizmetlerinden yararlanabilmek için ilk olarak bir google hesabının oluşturulması gerekmektedir. Ücretli bir platform olan Google Cloud Platform her hesap için bir defaya mahsus olmak üzere bir yıl içinde süresi dolacak şekilde 300 Dolarlık bir kullanımı ücretsiz olarak sunmaktadır. Bunun için Google hesabı ile Google Cloud Platforma giriş yapılarak ödeme seçeneğinin tanımlanması gerekmektedir. Ücretsiz denemenin başlayabilmesi için tanımlanan ödeme seçeneğinden 1 dolar ödeme yapılması gerekmektedir. Google bu ücreti kısa bir süre

sonra iade etmektedir. Amazon, IBM, Microsoft gibi kuruluşlar da benzer şekilde bulut hizmetleri sunmaktadırlar.

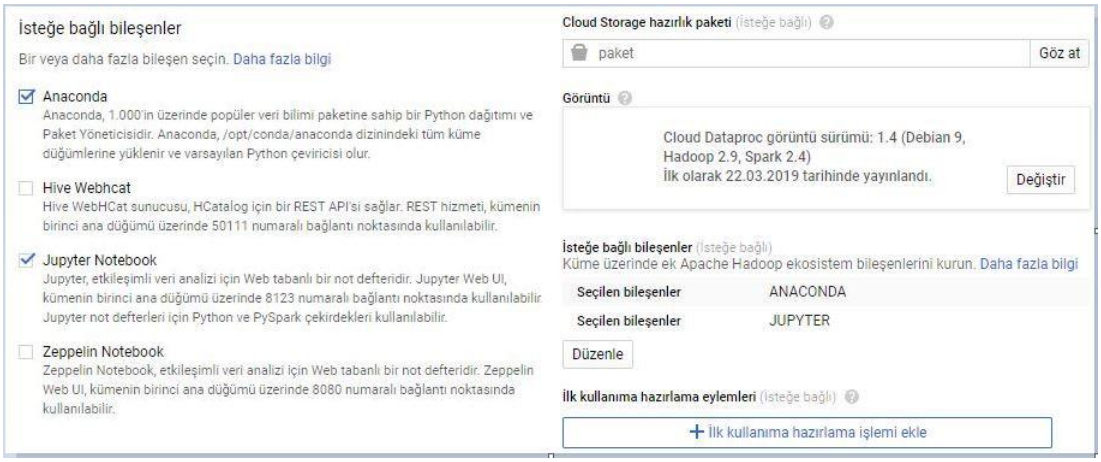
3.1.1. Spark kümesinin oluşturulması

GCP'ye giriş yaptıktan sonra konsol ekranında Dataproc->Kümeler seçilir. Bir defaya mahsus olmak üzere ilk ekranda API ardından Faturalandırmanın etkinleştirilmesi istenmektedir. Bu adımlardan sonra "Küme Oluştur" seçeneği ile Şekil 3.1.'de gösterildiği gibi oluşturulacak kümenin özelliklerinin belirleneceği ekrana geçiş yapılmaktadır. Bu ekranda küme ismi, kümenin barındırılacağı bölge, ana makine özellikleri, düğümlerin özellikleri ve düğüm sayısı belirlenmektedir.



Şekil 3.1. Ana makine ve düğümlerin özelliklerinin belirlenmesi

Uygulamada kullanılacak Jupyter Notebook için "Gelişmiş" seçeneğine tıklandıktan sonra "İsteğe Bağlı Bileşenler" tıklanmaktadır. Şekil 3.2.'de verilen pencerede Anaconda ile birlikte Jupyter Notebook seçilmektedir. Oluştur seçeneği ile Şekil 3.3'de görüldüğü gibi Spark kümesi oluşturulmaktadır. Şekil 3.4. ve Şekil 3.5.'te oluşturulmuş kümenin sırasıyla kaynakları ve düğümleri görüntülenmektedir.



Şekil 3.2. İsteğe bağlı bileşenler

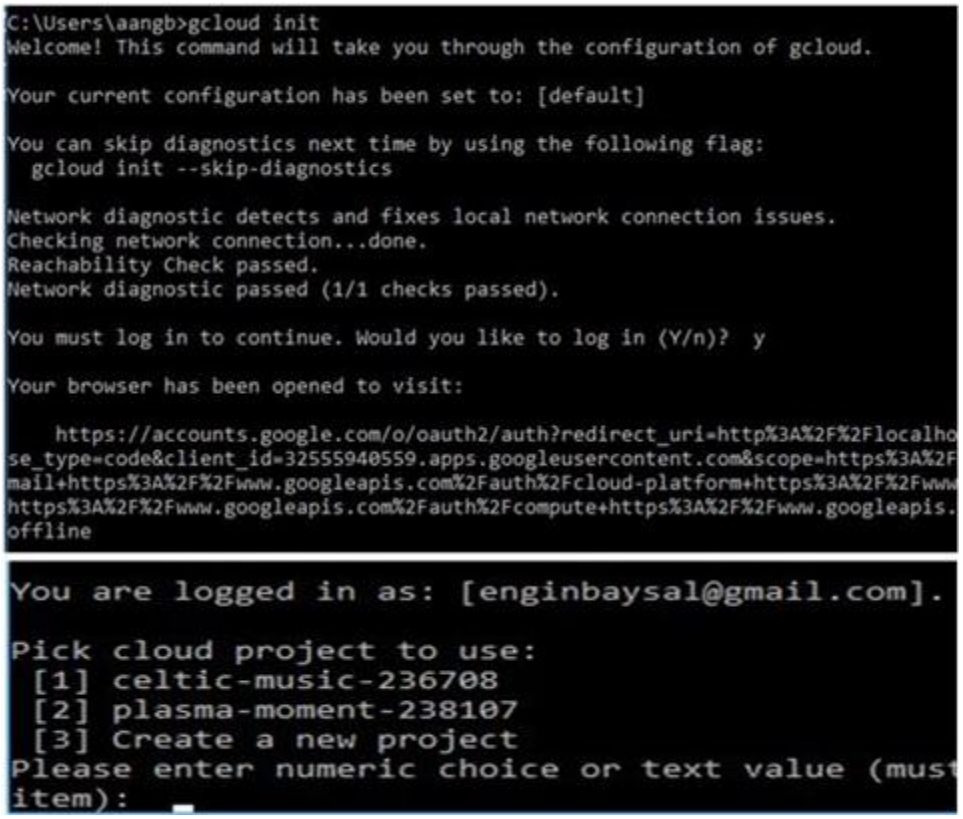
Şekil 3.3. Oluşturulan spark kümesi

Şekil 3.4. Spark küme kaynaklarının izlenmesi

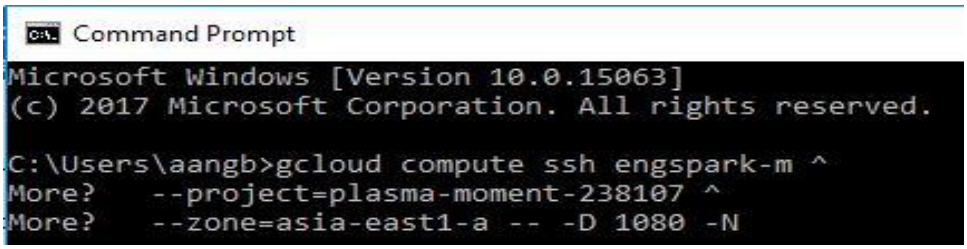
Şekil 3.5. Spark kümesinde çalışan makineler

3.1.2. Spark bileşenlerinin web arayüzlerine erişim

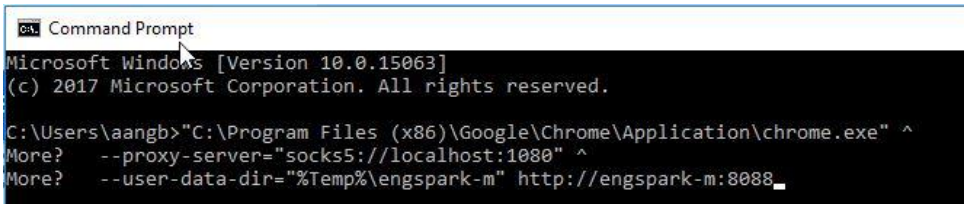
Yerel makineden Google Cloud Platformunda oluşturulmuş Spark kümesi bileşenlerinin web arayüzlerine erişebilmek için Google Software Development Kit (SDK) ve PuTTY uygulamalarının indirilip kurulması gerekmektedir. Bu sayede Google SDK uygulamaları kullanılabilecek ve PuTTY sayesinde web arayüzleri için bir SSH tünel oluşturulabilecektir. Şekil 3.6.'da verilen Windows komut ekranında "C:\gcloud init" uygulaması ile web tarayıcısında GCP ile ilişkili google hesabı ile oturum açılması istenmektedir. Kullanıcı girişi yapıldıktan sonra komut ekranına proje seçimi gelmektedir. Böylece SDK uygulamalarının hangi kullanıcı ve proje için çalıştırılacağı belirlenmiş olmaktadır. Kullanıcı ve proje değişikliklerinde "C:\gcloud auth <parametreler>" yazılarak istenilen kullanıcı ve proje varsayılan olarak tanımlanabilmektedir. Şekil 3.7.'deki komutlarla projenin barındırıldığı sunucuya bir SSH tüneli oluşturulmaktadır. Şekil 3.8.'de gösterilen komut ekranı ile 8088 portunda yer alan Şekil 3.9.'da görüntülenen YARN web arayüzüne erişim sağlanmaktadır. Adres satırında port numarasında değişiklikler yapılarak diğer bileşenlerin arayüzlerine erişim sağlanabilmektedir. Şekil 3.10.'da görüldüğü gibi uygulamada kullanılacak Jupyter Notebook için erişim portu 8123 şeklindedir.



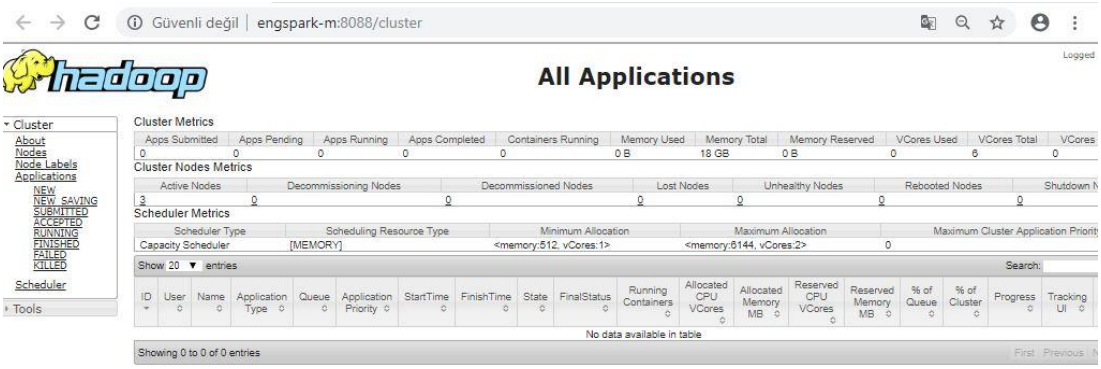
Şekil 3.6. gcloud init ile oturum açma



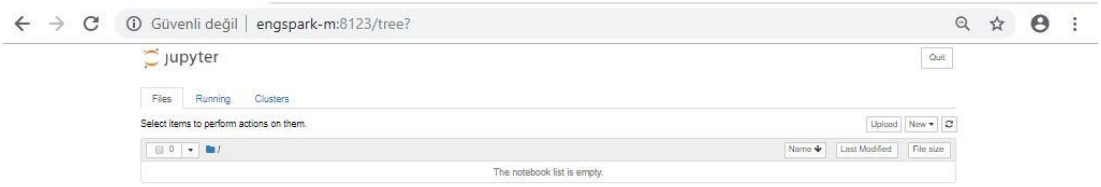
Şekil 3.7. Ssh tünel oluşturulması



Şekil 3.8. Spark kümesi web arayüzlerine erişim



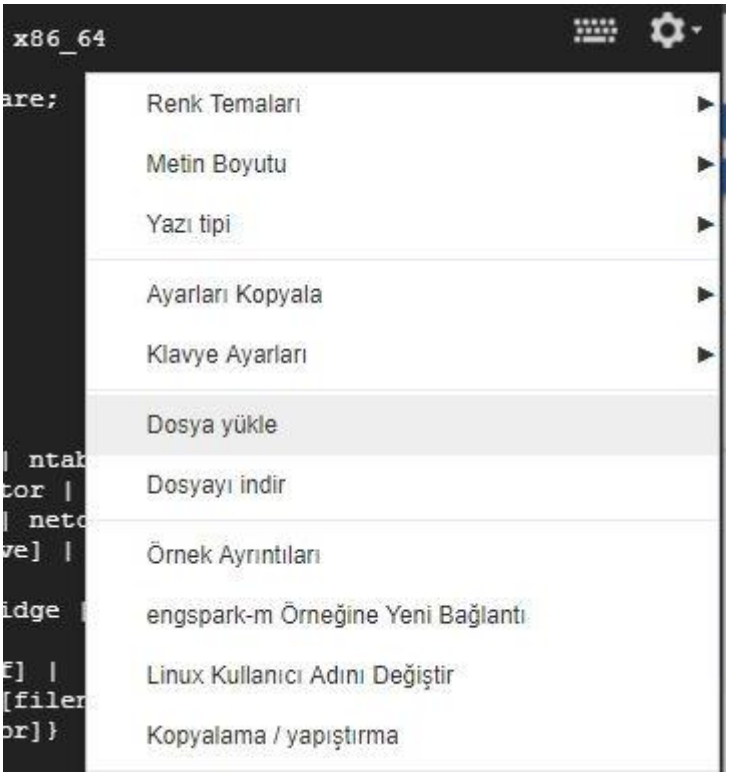
Şekil 3.9. YARN web arayüzü



Şekil 3.10. Jupyter notebook web arayüzü

3.1.3. Veri setlerinin hadoop kümesine aktarılması

Veri setlerinin Hadoop kümesine yüklenmesi için ilk olarak ana düğüme aktarılması gerekmektedir. Bu aktarım ana makinenin SSH arayüzü ile kolaylıkla yapılabilmektedir. Spark Kümesinde çalışan makinelere bakıldığında ana makine için SSH seçeneği mevcut bulunmaktadır. Bu seçenek ile Debian Linux çalıştıran makinenin komut ekranına erişilebilmektedir. Şekil 3.11.'de görüldüğü gibi seçenekler menüsünden dosya aktarımı seçilerek yerel makinede bulunan veri setleri Spark kümesinin ana makinesine aktarılmaktadır.



Şekil 3.11. Ana makine ssh arayüzü

Şekil 3.12.'de Hadoop komutları kullanılarak ana makinede buluna dosyaların Hadoop kümesine aktarılması ve listelenmesi gösterilmektedir.

```
enginbaysal@engspark-m:~$ hadoop fs -copyFromLocal /home/enginbaysal/clickdata_norm-1_1.csv /user/root/
enginbaysal@engspark-m:~$ hadoop fs -ls /user/root
Found 2 items
drwxr-xr-x   - root        hadoop          0 2019-04-19 08:52 /user/root/.sparkStaging
-rw-r--r--   2 enginbaysal hadoop  106528768 2019-04-19 13:58 /user/root/clickdata_norm-1_1.csv
```

Şekil 3.12. Veri setlerinin hadoop kümesine aktarılması

3.2. Düzenleyici Olarak Jupyter Notebook

Jupyter Notebook, konsol temelli yaklaşımı etkileşimli hesaplamalara yeni bir yön vererek kod yazmak ve yürütmek, sonuçları bildirmek gibi tüm hesaplama sürecinin yönetilmesini sağlayan web tabanlı bir uygulamadır. Jupyter Notebook'un özelliklerini şu şekilde sıralanabilir:

- Kodlamanın tarayıcıda düzenlenmesi, otomatik sözdizimi vurgulama, girinti ve sekme ile tamamlama ve içe aktarılması.

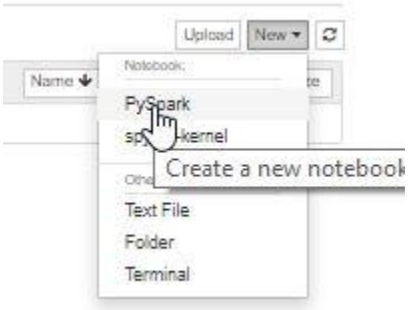
- Tarayıcıdan kod yürütme yeteneği ve hesaplamaların sonuçlarıyla döndürülmesi.

- HTML, LaTeX, PNG, SVG, vb. zengin medya sunumlarını kullanarak hesaplama sonucunun gösterilmesi.

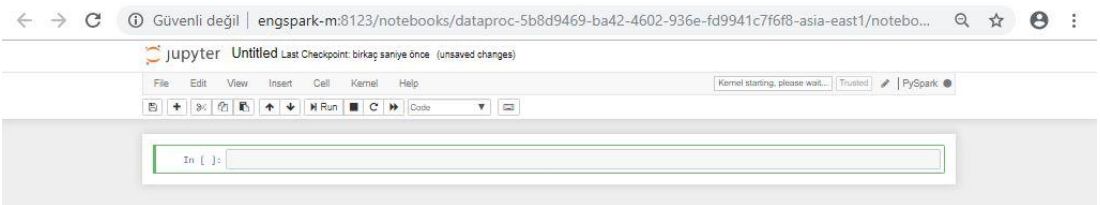
- Kod için yorum sağlayabilen Markdown işaretleme dilini kullanarak zengin metinler için tarayıcıda düzenleme.

- LaTeX kullanarak Markdown hücrelerin içine matematiksel notasyonu kolayca dâhil edebilme ve yerel olarak MathJax tarafından oluşturulabilme.

Jupyter Notebook'u oluşturduğumuz Spark kümesinde ki ana makinenin komut ekranına "Jupyter notebook" yazarak çalıştırabildiğimiz gibi tarayıcıdan ana makine ismiyle birlikte 8123 portu kullanılarak da erişilmektedir. Şekil 3.13.'te görülen ekranda "New" seçeneği tıklandıktan sonra kodlama için kullanılacak "Pyspark" seçeneği seçilmektedir. Şekil 3.14.'te kodlama ekranı görülmektedir.



Şekil 3.13. Pyspark düzenleyicisinin başlatılması



Şekil 3.14. Jupyter notebook pyspark kodlama ekranı

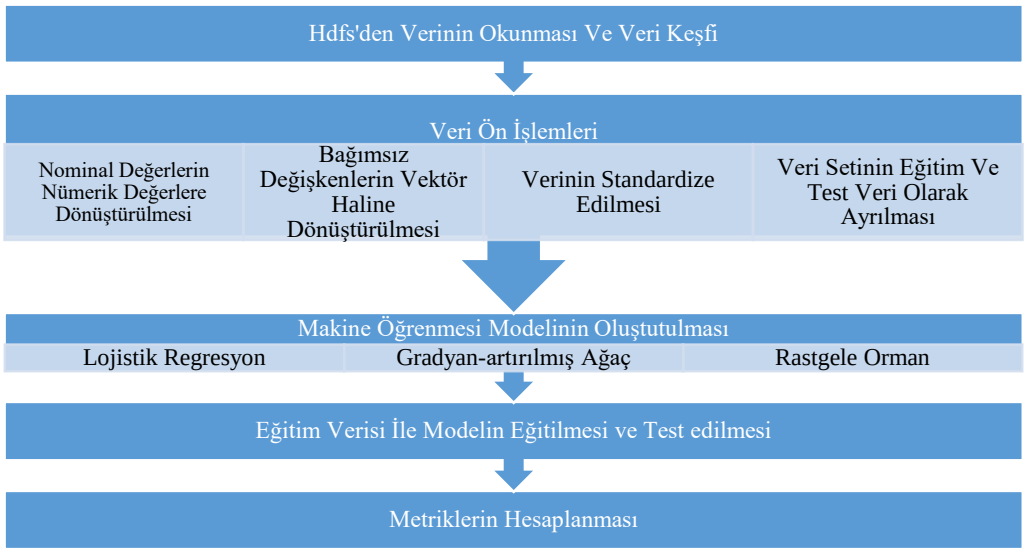
BÖLÜM 4. UYGULAMALAR

Bu bölümde büyük veri için geliştirilmiş dağıtık veri erişim ve işleme teknolojilerinden Spark kümesi üzerinde makine öğrenmesi uygulamaları geliştirilmektedir.

Çalışma ortamı olarak Google Cloud ortamında çalışmak üzere 5 düğümlü bir Spark kümesi oluşturulmuştur. Her bir düğüm için belirlenen özellikler aşağıda sıralanmıştır.

- 3.75 GB RAM,
- 1 vCPU
- 32 GB

Her iki uygulamada kullanılacak model Şekil 4.1.'de yer almaktadır.



Şekil 4.1. Uygulama modeli

Bu tez çalışmasında MLib kütüphanesinin pyspark API ile geliştirilmiş sınıflandırma ve regresyon makine öğrenmesi algoritmaları kullanılmaktadır. MLib kütüphanesinin desteklediği sınıflandırma ve regresyon makine öğrenmesi algoritmaları aşağıdaki gibidir [35];

- Sınıflandırma Algoritmaları
  a. Logistic Regression
  b. Decision Tree Classifier
  c. Gradient-boosted Tree Classifier
  d. Multilayer Perceptron Classifier
  e. Lineer Support Vector Machine
  f. One-vs-Rest Classifier
  g. Naive Bayes
- Regression
  a. Linear Regression
  b. Generalized Linear Regression
  c. Decision Tree Regression
  d. Random Forest Regression
  e. Gradient-boosted Tree Regression
  f. Survival Regression
  g. Isotonic Regression

4.1. Deneysel Çalışmalarda Kullanılan Makine Öğrenmesi Algoritmaları

MLib kütüphanesinin pyspark API ile desteklediği algoritmalardan uygulama-1'de Lojistik Regresyon (Logistic Regression), uygulama-2'de Rastgele Orman (Random Forest) ve Gradyan-artırılmış Ağaç (Gradient-boosted Tree) algoritmaları ile makine öğrenmesi uygulaması geliştirilmektedir.

4.1.1. Lojistik regresyon

Lojistik regresyon isminin aksine regresyondan çok sınıflandırma için kullanılan doğrusal bir modeldir. Bağımlı değişkenin kategori sayısına göre ikili olabileceği gibi çok terimli sınıflandırmalarda da kullanılabilmektedir. Çok terimli sınıflandırmada bağımlı değişken kategorilerinin tamamen ayrı olduğunu varsaymaktadır. Bir bağımlı değişken temel kategori olarak görev yapmaktadır. J bağımlı değişkenli bir sistem için J-1 lojistik model oluşturmaktadır. Lojistik model oluşturulurken bağımsız değişkenlerden bağımlı değişkenin elde edilme olasılığı hesaplanmaktadır. Bunun için her zaman ikili veriler kullanılmaktadır. Çok terimli modelde bir bağımlı değişken temel kategori ile eşleştirilerek elde edilme olasılıkları hesaplanmaktadır. J'nin temel kategori olduğu varsayıldığında logit fonksiyonu (Denklem 4.1);

$$log\frac{n_j}{n_j} = a_j + \beta_{1jk1j} + \cdots + \beta_{ijkij} \tag{4.1}$$

Burada j=1,..,J-1 iken a=kesişimi, β=bağımsız değişkeni ve k=regresyon katsayısını ifade etmektedir [36].

Sınıflandırma çalışması için kolay uygulanabilirliği ve yaygın kullanımı sebebiyle Lojistik regresyon algoritması tercih edilmektedir.

4.1.2. Rastgele Orman

Rastgele Orman uyum riskini azaltmak için birçok karar ağacının bir araya gelmesiyle oluşan bir algoritmadır. Her karar ağacının örneklemi değiştirilerek alt veri setleri ile oluşturulan karar ağaçlarının ortalamasını alarak sonuca ulaşmaktadır [37]. Uygulama-2 için rastgele orman seçilmesinin temel nedeni en başarılı genel amaçlı algoritmalardan biri olmasıdır. Ayrıca Gradyan-artırılmış Ağaç gibi karar ağaçları temelli bir algoritma olması tercih edilmesinin nedenlerinden biridir.

4.1.3. Gradyan-artırılmış ağaç

Gradyan-artırılmış Ağaç hataları en aza indirmek için karar ağaçlarını yinelemeli bir şekilde eğitmektedir. Güçlü bir öğrenen model elde etmek için zayıf öğrenmiş modellerden faydalanmaktadır. Bu sayede sonraki tahminler önceki tahminlerden daha az hata payı ile oluşmaktadır. Algoritma, kullanıcı tarafından sağlanan maksimum yineleme sayısına ulaşılana kadar yinelemeye devam etmektedir. Bu süreç aşama olarak adlandırılır, yani her yeni adımda önceki adımlarda modele eklenen karar ağaçları değiştirilmemektedir. Karar ağaçları artıklara yerleştirilerek model iyi performans göstermediği bölgelerde geliştirilmektedir [38]. Gradyan-artırılmış Ağaç algoritmasının tercih edilmesinin temel nedeni, yinelemeli ve çoklu karar ağaçları yöntemlerinin performanslarının karşılaştırılmasıdır.

4.2. Uygulama-1: Kablosuz Sensörlerden Toplanan Verilerde Makine Öğrenmesi

Bu uygulamada WISDM (Kablosuz Sensör Veri Madenciliği) olarak bilinen açık bir veri seti kullanılmaktadır. Bu veri setine http://www.cis.fordham.edu/wisdm/dataset.php bağlantısından erişilebilir. Veri setinde otuz altı kullanıcının aktivite bilgisi mevcut olup txt formatında ve başlıksız şeklinde kaydedilmektedir.

Veri setini incelemek için ilk olarak Şekil 4.2.'de görüldüğü gibi "Spark" nesnesiyle HDFS'de yer alan veri seti okunmaktadır. Spark nesnesi SparkContext olarak bilinen küme üzerinde nasıl erişim sağlanacağını bildiren bir sınıftan varsayılan olarak türetilmektedir. Şekil 4.3.'te okunan veri seti görüntülenmektedir.

```
df=spark.read \
.option("header","False") \
.option("infeschema","True") \
.option("sep",",") \
.csv("wisdm.txt")
```

Şekil 4.2. Veri setinin okunması

```
df.limit(5).toPandas().head()
```

	_c0	_c1	_c2	_c3	_c4	_c5	_c6	_c7	_c8	_c9	...	_c36	_c37	_c38	_c39	_c40	_c41	_c42	_c43	_c44	_c45
0	1	33	0.04	0.09	0.14	0.12	0.11	0.1	0.08	0.13	...	293.94	1550	3.29	7.21	4	4.05	8.17	4.05	11.96	Jogging
1	1	15	0.42	0.1	0.11	0.13	0.13	0.11	0.01	0.01	...	428.57	2783.33	7.88	7.76	4.69	9.93	8.9	9.93	14.89	Jogging
2	1	32	0.21	0.17	0.11	0.1	0.12	0.11	0.08	0.07	...	845.45	1816.67	3.58	4.21	3.05	4.23	5.2	4.23	12.15	Walking
3	1	5	0.34	0.12	0.12	0.1	0.09	0.07	0.07	0.04	...	961.11	4500	2.58	4.41	1.9	3.25	5.29	3.25	10.56	Walking
4	1	29	0.01	0.02	0.03	0	0.06	0	0	0.07	...	350	213.64	2.02	0.45	1.26	7.5	1.66	7.5	2.73	Sitting

5 rows × 46 columns

Şekil 4.3. Veri seti

Veri setindeki toplam kayıt sayısı Şekil 4.4.'teki kodla elde edilmektedir.

```
print("toplam Kayıt Sayısı :",df.count())
```

```
toplam Kayıt Sayısı : 5418
```

Şekil 4.4. Kayıt sayısı

Şekil 4.5.'te görüldüğü gibi "printShema" metoduyla veri setinde ki verilerin türleri listelenmektedir. Kullanılan veri setinde sınıflandırma için kullanacağımız son sütun (_c45) dışında bütün sütunlarda ki verilerin double olduğu gözlenmektedir.

```
df0.printSchema()
```

```
root
 |-- _c0: double (nullable = true)
 |-- _c1: double (nullable = true)
 |-- _c2: double (nullable = true)
 |-- _c3: double (nullable = true)
 |-- _c4: double (nullable = true)
 |-- _c5: double (nullable = true)
 |-- _c6: double (nullable = true)
 |-- _c7: double (nullable = true)
 |-- _c8: double (nullable = true)
 |-- _c9: double (nullable = true)
```

Şekil 4.5. Veri setinin özellikleri

Şekil 4.6.'da veri setinde aykırı değerler olup olmadığının belirlenmesi için sayısal değerlerin istatistikleri incelenmektedir.

```
num_cols=["_c0","_c1","_c2","_c3","_c4","_c5","_c6","_c7","_c8","_c9","_c10",\
          "_c11","_c12","_c13","_c1","_c15","_c16","_c17","_c18","_c19",\
         "_c20","_c21","_c22","_c23","_c24","_c25","_c26","_c27","_c28","_c29",\
         "_c30","_c31","_c32","_c33","_c34","_c35","_c36","_c37","_c38","_c39",\
         "_c40","_c41","_c42","_c43","_c44"]
```

```
df0.describe(num_cols).toPandas().head()
```

	summary	_c0	_c1	_c2	_c3	_c4	_c5	
0	count	5418	5418	5418	5418	5418	5418	
1	mean	284.39553340716134	18.868955334071615	0.09414359542266498	0.09894610557401239	0.09836840162421588	0.09693244739756395	0.09841085271
2	stddev	177.20011360223202	10.29350480380091	0.07254578011713958	0.05598610395976485	0.057048468183225044	0.051550479741276474	0.05166673256
3	min	1.0	1.0	0.0	0.0	0.0	0.0	
4	max	728.0	36.0	1.0	0.81	0.95	1.0	

5 rows × 46 columns

Şekil 4.6. İstatistikler

Şekil 4.7.'de sınıflandırma için kullanılacak hedef sütun istatistikleri yer almaktadır. Burada Spark'ın SQL sorgu kütüphanesinden faydalanılmaktadır.

```
from pyspark.sql import functions as F
```

```
df0.groupBy(F.col("_c45")) \
.agg({"*":"count"}) \
.toPandas().head(10)
```

	_c45	count(1)
0	Sitting	306
1	Walking	2081
2	Downstairs	528
3	Standing	246
4	Upstairs	632
5	Jogging	1625

Şekil 4.7. Hedef sütun dağılımı

Aynı şekilde Spark'ın SQL fonksiyonlarından yararlanılarak hedef sütunda boş değer olup olmadığı tespit edilmektedir. Şekil 4.8.'de yer alan "trim" komutu ile boş değerlerin çıkarılması sonucu elde edilen "dfna" ile "df0" arasında fark olmadığından hedef sütunda boş kayıt olmadığı görülmektedir.

```
dfna=df0.withColumn("_c45", trim(col("_c45")))
```

```
df0.count()
```

```
5418
```

```
dfna.count()
```

```
5418
```

Şekil 4.8. Hedef sütun boşluk tespiti

Şekil 4.9.'da yer alan kod ile pyspark'ın "ml" kütüphanesinde yer alan "StringIndexer" ile hedef sütunu makine öğrenmesi algoritmalarının kullanabileceği şekilde nümerik olmayan ifadeler nümerik hale dönüştürülmektedir.

```
from pyspark.ml.feature import StringIndexer
```

```
indexer=StringIndexer() \
.setInputCol("_c45") \
.setOutputCol("label")
```

```
indexer_model=indexer.fit(df0)
indexed_df=indexer_model.transform(df0)
```

Şekil 4.9. Nominal değerlerin nümerik hale dönüştürülmesi

Dönüştürme işlemini en çok tekrarlanan nominal değerden başlayarak numaralandırma şeklinde gerçekleştirmektedir.

Şekil 4.10.'da görüldüğü üzere en çok tekrar eden "Walking" değeri, "0.0" şeklinde numaralandırılmaktadır.

```
indexed_df.groupBy(F.col("label")) \
.agg({"*":"count"}) \
.toPandas().head(10)
```

	label	count(1)
0	0.0	2081
1	1.0	1625
2	4.0	306
3	3.0	528
4	2.0	632
5	5.0	246

Şekil 4.10. Nümerik haldeki nominal değerler

Şekil 4.11.'de veri setinde gerçekleştirilen boşluk kontrolü görülmektedir. Bunun sonucunda veri setimizde "_c35", "_c36" ve "_c37" sütunlarında "NULL" değer içeren kayıtlar tespit edilmektedir.

```
isNull = 1
for s in indexed_df.columns:
    if(indexed_df.filter(col(s).isNull()).count() > 0):
        print(isNull, ". ", s, " null değer var.")
    else:
        print(isNull, ". ", s)
    isNull+=1
```

Şekil 4.11. Veri setinde boşluk kontrolü

Boş kayıtlar ile makine öğrenmesi gerçekleşemeyeceğinden bu kayıtları boş kayıt dışındaki kayıtların ortalaması ile doldurma işlemi Şekil 4.12.'deki gibi gerçekleştirilmektedir. "df1", "indexed_df" içerisinden ortalaması alınamayacağı için nominal değer içeren "_c45" sütunun çıkarılmış haline eşitlenmektedir.

```
for x in df1.na.drop().columns:
    meanValue=df1.agg(avg(x)).first()[0]
    df1=df1.na.fill(meanValue, [x])
```

Şekil 4.12. Boş kayıtların doldurulması

Makine öğrenmesi algoritmaları bir vektör sütun ve bir hedef sütun üzerinde işlem gerçekleştirdiğinden bağımsız değişkenlerimizi vektör sütun haline getirilmesi gerekmektedir. Şekil 4.13. ve Şekil 4.14'te sırasıyla bağımsız değişkenler vektör haline getirilmesi işlemi ve vektör haline dönüştürülen sütun görüntülenmektedir.

```
from pyspark.ml.feature import VectorAssembler
```

```
in_cols=["_c1","_c2","_c3","_c4","_c5","_c6","_c7","_c8","_c9","_c10",\
              "_c11","_c12","_c13","_c1","_c15","_c16","_c17","_c18","_c19",\
             "_c20","_c21","_c22","_c23","_c24","_c25","_c26","_c27","_c28","_c29",\
             "_c30","_c31","_c32","_c33","_c34","_c35","_c36","_c37","_c38","_c39",\
             "_c40","_c41","_c42","_c43","_c44"]
```

```
assembler=VectorAssembler() \
.setInputCols(in_cols) \
.setOutputCol("vec_features")
```

```
as_df=assembler.transform(df1)
```

Şekil 4.13. Bağımsız değişkenlerin vektör haline getirilmesi

```
as_df.select("vec_features").limit(5).toPandas().head()
```

	vec_features
0	[33.0, 0.04, 0.09, 0.14, 0.12, 0.11, 0.1, 0.08, 0.13, 0.13, 0.08, 0.09, 0.1, 33.0, 0.11, 0.08, 0.04, 0.16, 0.13, 0.1, 0.03, 0.12, 0.08, 0.09, 0.12, 0.1, 0.1, 0.08, 0.11, 0.12, 0.1, 0.0, 8.4, 1.76, 2075.0, 293.94, 1550.0, 3.29, 7.21, 4.0, 4.05, 8.17, 4.05, 11.96]
1	[15.0, 0.42, 0.1, 0.11, 0.13, 0.13, 0.11, 0.01, 0.01, 0.0, 0.0, 0.1, 0.06, 15.0, 0.09, 0.09, 0.11, 0.15, 0.14, 0.14, 0.02, 0.11, 0.13, 0.07, 0.11, 0.08, 0.09, 0.06, 0.15, 0.12, 0.09, 0.0, 8.08, -0.33, 2675.0, 428.57, 2783.33, 7.88, 7.76, 4.69, 9.93, 8.9, 9.93, 14.89]
2	[32.0, 0.21, 0.17, 0.11, 0.1, 0.12, 0.11, 0.08, 0.07, 0.03, 0.02, 0.11, 0.12, 32.0, 0.08, 0.05, 0.12, 0.07, 0.17, 0.12, 0.1, 0.13, 0.15, 0.09, 0.1, 0.04, 0.09, 0.15, 0.08, 0.07, 0.13, 0.0, 10.11, -1.65, 1075.0, 845.45, 1816.67, 3.58, 4.21, 3.05, 4.23, 5.2, 4.23, 12.15]
3	[5.0, 0.34, 0.12, 0.12, 0.1, 0.09, 0.07, 0.07, 0.04, 0.04, 0.04, 0.13, 0.11, 5.0, 0.08, 0.11, 0.05, 0.09, 0.1, 0.1, 0.18, 0.07, 0.09, 0.09, 0.09, 0.11, 0.14, 0.1, 0.12, 0.16, 0.04, 0.0, 9.6, 1.33, 3175.0, 961.11, 4500.0, 2.58, 4.41, 1.9, 3.25, 5.29, 3.25, 10.56]
4	(29.0, 0.01, 0.02, 0.03, 0.0, 0.06, 0.0, 0.0, 0.07, 0.04, 0.0, 0.02, 0.03, 29.0, 0.0, 0.0, 0.09, 0.0, 0.0, 0.1, 0.0, 0.03, 0.0, 0.04, 0.0, 0.0, 0.08, 0.0, 0.0, 0.08, 0.0, 0.0, 0.62, 1.73, 240.0, 350.0, 213.64, 2.02, 0.45, 1.26, 7.5, 1.66, 7.5, 2.73)

Şekil 4.14. Vektör haline getirilmiş sütun

Birçok makine öğrenmesi algoritması mesafe hesaplaması yapmaktadır. Bu mesafe hesaplamalarında Öklid mesafesi yaygın olarak kullanılmaktadır. Bu nedenle Şekil 4.15.'te görüldüğü gibi veri setinde normal bir dağılım sağlamak için standardizasyon işlemi gerçekleştirilmektedir. Şekil 4.16'da standardize edilmiş veri görüntülenmektedir.

```
from pyspark.ml.feature import StandardScaler
```

```
scaler=StandardScaler() \
.setInputCol("vec_features") \
.setOutputCol("features")
```

```
model=scaler.fit(as_df)
```

```
scaled_df =  model.transform(as_df)
```

Şekil 4.15. Veri setinin standardizasyonu

	features
0	[3.2059051439714334, 0.5513759716335271, 1.6075417583027292, 2.4540536224453198, 2.327815387989809, 2.1290295426455663, 2.0418458659167675, 1.5853086301308046, 2.492455822996172, 2.1140372963126977, 0.9925987751580455, 1.648561904801064, 1.9504933181313628, 3.2059051439714334, 2.2829125425994423, 1.5923050465206896, 0.7638776507634212, 2.885919365758033, 2.26568267420464, 1.7433483347382368, 0.38104924259869866, 2.0799407861435144, 1.5996163088837299, 1.9759815974983101, 2.260444386825452, 2.2081810848190924, 2.077824085996005, 1.5694475296087178, 2.245218765589301, 2.443281040658958, 1.4181328468461016, 0.0, 2.238365852963895, 0.7999564625076131, 2.044362965479102, 0.37587132952427316, 1.5009076903028442, 1.1833104254258315, 3.1410113466539333, 2.5441260592493395, 1.2663846228364244, 3.1222227643876908, 1.2663846228364244, 4.703127653407584]
1	[1.457229610896106, 5.789447702152035, 1.7861575092252548, 1.9281849890641796, 2.5218000036556263, 2.5161258231265786, 2.246030452508444, 0.19816357876635057, 0.19172737099970555, 0.0, 0.0, 1.8317354497789602, 1.1702959908788177, 1.457229610896106, 1.8678375348540892, 1.7913431773357757, 2.100663539599408, 2.705549405398156, 2.439965956835766, 2.4406876686335317, 0.25403282839913244, 1.9066123872982217, 2.599376501936061, 1.5368745758320193, 2.072074021256664, 1.7665448678552738, 1.8700416773964044, 1.1770856472065383, 3.061661953076319, 2.443281040658958, 1.2763195621614913, 0.0, 2.153094772850985, -0.14999183672017746, 2.6355040639308891, 0.5480274059135121, 2.695175097839107, 2.8341903198649097, 3.3806169278827354, 2.982987804469851, 3.1049874826581965, 3.4011973810343266, 3.1049874826581965, 5.85531528087282]
2	[3.108756503245026, 2.8947238510760176, 3.036467765682933, 1.9281849890641796, 1.9398461566581742, 2.3225776828860725, 2.246030452508444, 1.5853086301308046, 1.342091596997939, 0.4878547606875456 3, 0.2481496937895 1138, 2.014908994756856, 2.3405919817576355, 3.108756503245026, 1.6603000309814127, 0.9951906540754311, 2.2916329522902634, 1.2625897225191396, 2.9628158047291446, 2.092018001685884, 1.2701641419956624, 2.253269184988808, 2.999280579156993, 1.9759815974983101, 1.8837036556878766, 0.8832724339276369, 1.8700416773964044, 2.9427141180163456, 1.632886374974037, 1.4252472737177255, 1.8435727008999319, 0.0, 2.694033187317259, -0.7499591836008872, 1.0591278013927878, 1.0811064011236877, 1.7591315959628828, 1.2876143838980174, 1.834071812678649, 1.939896120177621 3, 1.3226683838513769, 1.9872164473458989, 1.3226683838513769, 4.777842892048675]

Şekil 4.16. Standardizasyon sonucu veri seti

Şekil 4.17'de kullanılacak gözetimli makine öğrenmesi algoritması için veri seti %80'i eğitim ve %20'si test verisi olacak şekilde rastgele veri setlerine ayırma işlemi yapılmaktadır.

```
train_df, test_df=scaled_df.randomSplit([0.8,0.2],seed=0)
```

Şekil 4.17. Veri setinin eğitim ve test veri setlerine ayrılması

Veri hazırlığı aşamaları tamamlandıktan sonra veri setinde uygulanacak makine öğrenmesi algoritmaları çalışma ortamına tanımlanarak kullanabilmektedir.

Şekil 4.18.'de görüldüğü gibi "ml" kütüphanesinden sınıflandırma için kullanılacak Lojistik Regresyon algoritması çalışma ortamına tanımlanmaktadır. Bu sınıftan bir "lg_obj" nesnesi oluşturduktan sonra test verisi ile bir model oluşturulmaktadır.

Oluşturulan bu modeli test veri seti kullanılarak sınanmaktadır. Elde edilen sonuçlar "sonuc_df" adında oluşturulan veri setine aktarılmaktadır. Şekil 4.19.'da test verisinde ki asıl değerler ile modelin tahmin ettiği değerler görüntülenmektedir.

```
from pyspark.ml.classification import LogisticRegression
```

```
lg_obj = LogisticRegression() \
.setFeaturesCol("features") \
.setLabelCol("label") \
.setPredictionCol("prediction")
```

```
lg_model = lg_obj.fit(train_df)
```

```
sonuc_df=lg_model.transform(test_df)
```

Şekil 4.18. Lojistik regresyon algoritmasının eğitilmesi ve test edilmesi

```
sonuc_df.select("label", "prediction").toPandas().head(10)
```

	label	prediction
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	1.0	1.0
4	0.0	0.0
5	0.0	0.0
6	4.0	4.0
7	1.0	1.0
8	0.0	0.0
9	0.0	0.0

Şekil 4.19. Test verisi asıl değerler ve tahmin edilen değerler

Bu uygulama veri setinin %80'i eğitim verisi olarak seçilmiş geri kalan %20'si test verisi olarak kullanılmıştır. Gerçeklenen makine öğrenmesi uygulamasında elde edilen sonuçlar Tablo 4.1.'de yer almaktadır. Hareket verisi ile eğitilen Lojistik Regresyon

algoritması test verisi sınanması sonucunda hareket durumlarından “Walking”, “Jogging”, “Sitting” ve “Standing” pozisyonlarının yüksek oranda doğru sınıflandırıldığı ancak “Upstairs” ve “Downstairs” sınıflarının tespitinin çok düşük oranda doğru olduğu görülmektedir.

Tablo 4.1. Her sınıf için doğruluk matrisi

	0.0 (Walking)	1.0 (Jogging)	2.0 (Upstairs)	3.0 (Downstairs)	4.0 (Sitting)	5.0 (Standing)
Precision	0,96	0,96	0,27	0,06	0,94	0,83
Recall	0,74	0,95	0,39	0,35	0,92	0,92
F1 Measure	0,84	0,96	0,32	0,1	0,92	0,87

4.2.1. Uygulama 1’in Çalışması ve Spark Uygulama Modeli

5 düğümlü bir Spark kümesi üzerinde çalıştırılan toplam 224 iş (job) ve 328 aşamadan oluşan uygulama yaklaşık olarak 1 dakika 27 saniye zaman almaktadır. Şekil 4.20.’de görüldüğü gibi toplamda 727 görev tamamlanmış durumdadır. Başarısız görev bulunmamaktadır.

	RDD Blocks	Storage Memory	Complete Tasks	Total Tasks
Active(2)	0	0.0 B / 2.3 GB	394	394
Dead(3)	0	0.0 B / 4.3 GB	333	333
Total(5)	0	0.0 B / 6.7 GB	727	727

Şekil 4.20. Tamamlanan görev sayısı

Şekil 4.21.’de görüldüğü gibi bu uygulamada ncluster-m isimli düğüm sürücü görevine atanarak ve yürütme süresince aktif kalmaktadır. Diğer düğümler yürütme esnasında ihtiyaç duyuldukça aktifleştirilmektedir.

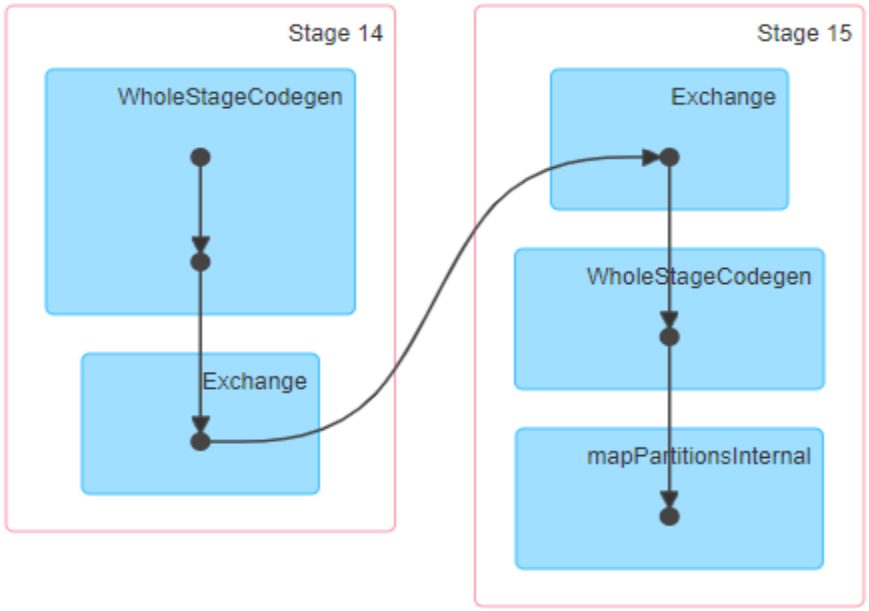
Executor ID	Address	Status	RDD Blocks	Storage Memory
driver	ncluster-m.europe-west2-c.c.aangproject.internal:36147	Active	0	0.0 B / 885.2 MB
1	ncluster-w-1.europe-west2-c.c.aangproject.internal:33563	Dead	0	0.0 B / 1.4 GB
2	ncluster-w-2.europe-west2-c.c.aangproject.internal:36177	Active	0	0.0 B / 1.4 GB
3	ncluster-w-0.europe-west2-c.c.aangproject.internal:39573	Dead	0	0.0 B / 1.4 GB
4	ncluster-w-4.europe-west2-c.c.aangproject.internal:42571	Dead	0	0.0 B / 1.4 GB

Şekil 4.21. Yürütme esnasında kullanılan düğümler

Birçok hesaplama ve RDD dönüşüm işlemleri bir aşama (Stage) ve birkaç görevden oluşurken Şekil 4.22 ve Şekil 4.23'de görüldüğü gibi sırasıyla veriyi görüntülemek için kullanılan "toPandas" fonksiyonu 200'ün üzerinde görevden ve en az iki aşama (Stage) ve oluşmaktadır.

Description		Tasks: Succeeded/Total
toPandas at <ipython-input-16-bdda68c3ca89>:2	+details	200/200
toPandas at <ipython-input-16-bdda68c3ca89>:2	+details	1/1

Şekil 4.22. toPandas fonksiyonu görev sayısı



Şekil 4.23. toPandas fonksiyonu DAG grafiği

Şekil 4.24.'te metriklerin hesaplanması sırasında veri ve çalıştırılan kodun konum farklılıkları gösterilmektedir. NODE_LOCAL, ncluster-w-0 düğümünde çalıştırılan kodun hesaplama için kullandığı veri ile aynı düğümde ancak farklı yürütücülerde yer aldığını göstermektedir. RACK_LOCAL, ncluster-w-4 düğümünde çalıştırılan kod ile hesaplama için kullanılan verinin aynı kümede ancak farklı düğümlerde olduğunu belirtmektedir.

Locality Level	Executor ID	Host
NODE_LOCAL	11	ncluster-w-0.europe-west2-c.c.aangproject.internal
RACK_LOCAL	12	ncluster-w-4.europe-west2-c.c.aangproject.internal

Şekil 4.24. Düğüm ve konum düzeyi

4.3. Uygulama-2: Rastgele Orman ve Gradyan-artırılmış Ağaç Makine Öğrenmesi Algoritmaları Kullanılarak Tıklanma Maliyetlerinin Tahmini

Bu uygulamada kullanılan veri seti bir çevrimiçi turizm acentesinin kontrol panelinden alınan rapor verisidir. Veri seti, hem sayısal hem de kategorik özellikler içermektedir. Veri seti "csv" formatında ve nitelik başlıkları mevcuttur.

Bu çalışmada veri setinde bulunan "avg_cpc" olarak adlandırılan tıklama başına düşen ortalama maliyetin tahmininde Rastgele Orman ve Gradyan-artırılmış Ağaç algoritmalarının performansları karşılaştırılmaktadır.

Veri setinin bazı önemli nitelikleri şu şekildedir.

- Hotel_impr: bir otelin gösterim sayısı
- Click_hotel_impr: tıklanma sayısının gösterim sayısına oranı
- Avg_cpc: bir dönemde otel tarafından ödenen ortalama tıklanma maliyeti
- Clicks: gösterilen teklifin tıklanma sayısı
- Booking_value_index: otelin tıklanma başına satış potansiyeli
- Stars: otelin yıldız sayısı

- Rating: otelin 100 üzerinden hesaplanmış değerlendirme puanı
- Weekday: kayıttaki verilerin haftanın hangi gününe denk geldiği
- Hotel_types: otelin kıyı ya da şehir oteli olduğu
- My_min_position: acente teklifinin diğer tekliflere göre sırası
- My_price: acentenin otel için belirlediği fiyat
- Total_min_price: o otel için verilmiş tüm teklifleri arasındaki en iyi teklif
- Hotels_price: hotels.com'un o otel için listelenen fiyatı
- Odamax_price: odamax.com'un o otel için listelenen fiyatı
- Top_pos_share: teklifin ilk sırada görüntülenmeme oranı

Şekil 4.25.'te görüldüğü gibi okunduktan sonra incelenen veri setinin 84 sütun ve 477519 kayıttan oluştuğu ve ilk beş satırı listelenen veride "NULL" değerler olduğu görülmektedir.

	ota_id	click_hotel_impr	hotel_impr	avg_cpc	clicks	booking_value_index	stars	rating	weekday	hotel_types	...
0	5963554	0.0203	2512	0.05	51	5	5	84.71	Saturday	City	...
1	5139884	0.0000	161	0.02	0	1	0	79.00	Monday	City	...
2	1335949	0.0198	2219	0.09	44	2	3	73.71	Saturday	City	...
3	5191864	0.0000	211	0.04	0	2	4	79.00	Monday	City	...
4	2830710	0.0000	1278	NaN	0	1	2	79.88	Thursday	City	...

5 rows × 84 columns

```
print("data satır sayısı: ", df.count())
```

```
data satır sayısı:  477519
```

Şekil 4.25. Veri seti

Şekil 4.26.'da görüldüğü gibi veri setinin özellikleri incelendiğinde biri "ota_id" olmak üzere 81 sayısal değer ve 3 nominal değer yer almaktadır.

```
root
 |-- ota_id: integer (nullable = true)
 |-- click_hotel_impr: double (nullable = true)
 |-- hotel_impr: integer (nullable = true)
 |-- avg_cpc: double (nullable = true)
 |-- clicks: integer (nullable = true)
 |-- booking_value_index: integer (nullable = true)
 |-- stars: integer (nullable = true)
 |-- rating: double (nullable = true)
 |-- weekday: string (nullable = true)
 |-- hotel_types: string (nullable = true)
 |-- my_min_position: integer (nullable = true)
 |-- my_price: integer (nullable = true)
 |-- top4_min_price: integer (nullable = true)
 |-- total_min_price: integer (nullable = true)
 |-- hotels_price: integer (nullable = true)
 |-- hotels_min_position: integer (nullable = true)
 |-- odamax_price: integer (nullable = true)
 |-- odamax_min_position: integer (nullable = true)
```

Şekil 4.26. Veri setinin özellikleri

Nominal değerlerin istatistikleri Şekil 4.27.'deki gibi olup, boş kayıt mevcut değildir.

	weekday	count(1)
0	Wednesday	67607
1	Tuesday	69792
2	Friday	66772
3	Thursday	70273
4	Saturday	56825
5	Monday	73999
6	Sunday	72251

	hotel_types	count(1)
0	Summer	173411
1	City	304108

	bolge	count(1)
0	Ege	101661
1	Akdeniz	73486
2	Marmara	189252
3	Karadeniz	40264
4	IcAnadolu	54576
5	GuneyDoguAnadolu	10536
6	DoguAnadolu	7744

Şekil 4.27. Nominal değerler

Şekil 4.28.'de görüldüğü gibi StringIndexer sınıfı kullanılarak nominal değerler nümerik değerlere dönüştürülerek ve "df1" olarak adlandırılan nominal değer içermeyen yeni veri seti oluşturulmaktadır.

```
from pyspark.ml.feature import StringIndexer
```

```
weekday_indexer=StringIndexer() \
.setInputCol("weekday") \
.setOutputCol("weekday_index")
```

```
weekday_indexer_model=weekday_indexer.fit(df)
weekday_indexer_df=weekday_indexer_model.transform(df)
```

```
hotel_types_indexer=StringIndexer() \
.setInputCol("hotel_types") \
.setOutputCol("hotel_types_index")
```

```
hotel_types_indexer_model=hotel_types_indexer.fit(weekday_indexer_df)
hotel_types_indexer_df=hotel_types_indexer_model.transform(weekday_indexer_df)
```

```
bolge_indexer=StringIndexer() \
.setInputCol("bolge") \
.setOutputCol("bolge_index")
```

```
bolge_indexer_model=bolge_indexer.fit(hotel_types_indexer_df)
bolge_indexer_df=bolge_indexer_model.transform(hotel_types_indexer_df)
```

```
df1=bolge_indexer_df.drop("ota_id","weekday","hotel_types","bolge")
```

Şekil 4.28. Nominal değerlerin nümerik değerlere dönüştürülmesi

Bütün sütunlarda yapılan “NULL” değer kontrolünde birçok sütunun boş kayıt içerdiği tespit edilmektedir. Boş kayıtlar silinmeden dolu kayıtların ortalama değerleri ile doldurulmaktadır.

Hedef değişken olan “avg_cpc” sütununun ismi Rastgele Orman ile makine öğrenmesi gerçekleştirebilmek için “label” olarak değiştirilmektedir. Şekil 4.29.’da regresyon analizi için kullanılacak kütüphaneler çalışma ortamına dâhil edilmektedir. Şekil 4.30.’da görüldüğü gibi makine öğrenmesi için model oluşturularak test edilmektedir.

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator
```

Şekil 4.29. Regresyon analizi için kullanılacak kütüphaneler

```
featureIndexer =\
    VectorIndexer(inputCol="vec_features", outputCol="indexedFeatures", maxCategories=4).fit(label_indexer_df)

(trainingData, testData) = label_indexer_df.randomSplit([0.7, 0.3])

rf = RandomForestRegressor(featuresCol="indexedFeatures")

pipeline = Pipeline(stages=[featureIndexer, rf])

model = pipeline.fit(trainingData)

predictions = model.transform(testData)
```

Şekil 4.30. Modelin oluşturulması ve test edilmesi

Regresyon analizi için oluşturulan modelin test verisi ile test edilmesi sonucu Şekil 4.31.'de ilk on satırı görülen test verisindeki hedef değişkenlere karşılık gelen algoritmanın tahminleri görüntülenmektedir. Şekil 4.32.'de modelin testi sonucunda elde edilen metriklerin hesaplanması görüntülenmektedir.

```
predictions.select("label", "prediction").show(10)
```

```
+-----+--------------------+
|label|          prediction|
+-----+--------------------+
|  0.0| 0.03547831313365091|
|  0.0| 0.03041669733744986|
| 0.01|0.012099106287749618|
| 0.01|0.012242912530510809|
| 0.01|0.021524613556499837|
| 0.01|0.013001086938867121|
| 0.01| 0.01630341397375161|
| 0.01|0.021524613556499837|
| 0.01|0.014150650587192504|
| 0.01|0.012793219337554822|
+-----+--------------------+
only showing top 10 rows
```

Şekil 4.31. Hedef değişken ve tahmin edilen

```
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)

print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
Root Mean Squared Error (RMSE) on test data = 0.0108844
```

Şekil 4.32. Metriklerin hesaplanması

Şekil 4.33.'te Gradyan-artırılmış Ağaç modelinin oluşturulması, test edilmesi ve metriklerin elde edilmesi görüntülenmektedir.

```
from pyspark.ml.regression import GBTRegressor

gbt = GBTRegressor(featuresCol="indexedFeatures", maxIter=10)

pipeline = Pipeline(stages=[featureIndexer, gbt])

model = pipeline.fit(trainingData)

predictions = model.transform(testData)

evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
gbtrmse = evaluator.evaluate(predictions)

print("Root Mean Squared Error (RMSE) on test data = %g" % gbtrmse)
```

```
Root Mean Squared Error (RMSE) on test data = 0.00995183
```

Şekil 4.33. Gradyan-artırılmış ağaç modeli ve metriklerin elde edilmesi

Tablo 4.2. Ölçülen metrikler

Ölçülen Metrikler	Rastgele Orman	Gradyan-artırılmış Ağaç
RMSE	0,0108844	0,00995183
R2	0,912755	0,927808
MAE	0,00490055	0,00444994
MSE	0,00011847	0,000099

Veri seti ile gerçekleştirilen makine öğrenmesi modellerinin başarımları Tablo 4.2.'de yer almaktadır. Yinelemeli bir algoritma olan Gradyan-artırılmış Ağaç algoritmasının çok örneklemli karar ağaçlarından oluşan Rastgele Orman algoritmasına göre ortalama hataların kareköklerinin daha düşük olduğu görülmektedir. Aynı zamanda açıklayıcı değişkenlerin hedef değişkeni Rastgele Orman algoritmasında %91, Gradyan-artırılmış Ağaç algoritmasında %92 oranında açıkladığı görülmektedir. Bu uygulamada kullanılan veri seti ile ortalama tıklama maliyetlerinin hesaplanmasında yinelemeli karar ağaçlarının çok örneklemli karar ağaçlarına göre daha iyi performansa sahip olduğu gözlenmektedir.

BÖLÜM 5. SONUÇ

Veri boyutu ve çeşitliliğinde ki artış veri depolanması ve analiz edilmesi konusunda yeni yaklaşımlar gerektirmektedir. Bu tez çalışmasında büyük veri için geliştirilmiş dağıtık veri depolama ve işleme araçları kullanılarak iki farklı veri seti ile makine öğrenmesi uygulamaları geliştirilmiştir. Depolama için HDFS, verinin analitiği için ise Spark üzerinde pyspark kullanılmıştır. Geliştirilen sınıflandırma ve regresyon uygulamalarının Spark uygulama modeline göre nasıl çalıştığı gözlenmiştir.

Beş düğümden oluşan Spark kümesinde uygulamaların çalıştırılması esnasında YARN sadece ihtiyaç duyulduğu kadar kaynağı kullanmıştır. YARN'ın dinamik yerleştirme özelliği sayesinde executor'ların ihtiyaç duyulmadığında sonlandırıldığı gözlenmiştir. Bu sayede executorlar gerektiğinde başka uygulamalar için kullanılabilirler.

Transformation ve Action olarak adlandırılan Spark operasyonlarının çalıştırılması esnasında oluşan görev (Task) sayısı bakımından ayrıştıkları gözlenmiştir. Transformation operasyonu sonucu Spark sürücüye herhangi bir verinin dönmemesi daha az sayıda görev (Task) oluşmasına neden olmaktadır. Ayrıca action operasyonları küme içerisinde ki ağ trafiğini de arttırmaktadırlar. Verinin konumu uygulama kodunun çalıştığı yere yakın bir yerde bulunuyorsa verinin ağda taşınmasına gerek yoktur. Ancak veri ve kodun farklı düğümlerde bulunması ve bu kodun birçok defa çalıştırılması ağ trafiğini arttıracaktır.

Uygulama 2'de çevrimiçi bir turizm acentesinin web sitesinin kontrol panelinden toplanmış veri seti 5 düğümlü bir Spark kümesinde pyspark API kullanılarak veri keşfi ve önişlemleri gerçekleştirilmiştir. Hazırlanan veri setinde Rastgele Orman ve Gradyan-artırılmış Ağaç makine öğrenmesi algoritmaları ile ortalama tıklama maliyetinin tahmini çalışması gerçekleştirilmiştir. Oluşturulan model test edildiğinde, Rastgele Orman ile açıklayıcı değişkenlerden hedef değişken %91 oranında

açıklayabildiği, Gradyan-artırılmış Ağaç ile %92 oranında açıklanabildiği görülmektedir.

İlerleyen çalışmalarda küme düğüm sayısın dağıtık veri analitiğine etkileri detaylı bir şekilde incelenebilir, Spark MLib için yeni algoritmalar tasarlanabilir kaynak yöneticilerinin (YARN, Mesos, Kubernetes) performansları karşılaştırılabilir.

KAYNAKLAR

[1] A. Gandomi, and M. Haider, “Beyond the hype: Big data concepts, methods, and analytics,” International Journal of Information Management, vol. 35, pp. 137–144, 2015.

[2] I.Yaqoob, I. A. T. Hashema, A. Gani, S. Mokhtar, E. Ahmed, N. B. Anuar, A. V. Vasilakos, “Big data: From beginning to future,” International Journal of Information Management, vol. 36, pp. 1231–1247, 2016.

[3] T. Sajana, C. M. S. Rani, and K. V. Narayana, “A Survey on Clustering Techniques for Big Data Mining,” Indian Journal of Science and Technology, vol. 9(3), January, 2016.

[4] M. A. Khan, M. F. Uddin, and N. Gupta,“Seven V’s of big data understanding big data to extract value,” 2014 Zone 1 Conference of the American Society for Engineering Education, pp. 1-5, April, 2014.

[5] M. V. Keskin, “Büyük Veride Makine Öğrenmesi Uygulamaları,” Yıldız Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Uygulamalı İstatistik ABD, İstatistik Programı, Yüksek Lisans Tezi, 2018.

[6] N. B. Oğur, “EKG Verileri İçin Gerçek Zamanlı Veri Analitiği Mimarisi,” Sakarya Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar ve Bilişim Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi, 2018.

[7] S. Çetinkaya, “Hadoop/Mapreduce Teknolojisi Kullanılarak Hızlı Tüketim Sektöründe Büyük Veri Analizi,” İstanbul Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği ABD, Bilgisayar Mühendisliği P., Yüksek Lisans Tezi, 2016.

[8] Y. Erdem, “Büyük Verinin Makine Öğrenmesi Yöntemleri İle Apache Spark Teknolojisi Kullanılarak Sınıflandırılması,” Karabük Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği ABD., Yüksek Lisans Tezi, 2017.

[9] A. L’heureux, K. Grolinger, and M. A. M. Carpetz,”Machine Learning With Big Data: Challenges and Approaches,” IEEE Access, vol. 5, 2017.

[10] https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne/, Erişim Tarihi:24.03.2019.

[11] W. Raghupati, and V. Rawhupati, "Big data analytics in healthcare: Promise and potantial," Health Inf. Sci. Syst., vol. 2, pp.1-10, 2014.

[12] O. Y. Al-Jarrah, P. D. Yoo, S. Muhaidat, G. K. Karagiannidis, and K. Taha, "Efficient machine learning for big data: A review," Big Data Res., vol. 2, pp. 87-93, Sep. 2015.

[13] M. A. Beyer, and D. Laney, "The Importance of 'Big Data': A Deffinition," Gartner Research, 2012.

[14] I.W. Tsang, J. T. Kwok, and P.-M.Cheung, "Core Vector Machines: Fast SVM Training on Very Large Data Sets," Journal of Machine Learning Research, vol. 6, pp. 363-392, 2005.

[15] C.-T Chu, S.K. Kim, Y.-A. Lin, Y. Y.Yu, G.Bradski, A. Y. Ng, and K. Olukotun, "Map-Reduce for Machine Learning on Multicore,"20th Conf. Adv. Neural Inf. Process. Syst. (NIPS), pp. 281-288, 2006.

[16] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," 9th USENIX Conf. Netw. Syst. Design Implement. (NSDI), 2012.

[17] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing With working sets," 2nd USENIX Conf. Hot Topics Cloud Comput., 2010.

[18] K. A. Kumar, J. Gluck, A. Deshpande, and J. Lin, "Hone: 'Scaling down' Hadoop on shared-memory systems," VLDB Endowment, vol. 6, no. 12, pp. 1354_1357, 2013.

[19] C. Parker, "Unexpected challenges in large scale machine learning," 1st Int. Workshop Big Data, Streams Heterogeneous Source Mining Algorithms, Syst. Programm. Models Appl. (BigMine), pp. 1-6, 2012.

[20] K. Grolinger, M. Hayes, W. A. Higashino, A. L'Heureux, D. S. Allison, and M. A. M. Capretz, "Challenges for MapReduce in big data," IEEE World Congr. Services (SERVICES), Jun., pp. 182_189, 2014.

[21] J. Fan, F. Han, and H. Liu, "Challenges of big data analysis," Nat. Sci. Rev., vol. 1, no. 2, pp. 293-314, 2014.

[22] H.V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, And C. Shahabi, “Big data and its technical challenges,” Commun. ACM, vol. 57, no. 7, pp. 86-94, 2014.

[23] J. Leskovec, A. Rajaraman, and J. D. Ullman, “Mining of Massive Datasets,” vol. 13. Cambridge, U.K.: Cambridge Univ. Press, 2014.

[24] K. Grolinger, W. A. Higashino, A. Tiwari, and M. A. Capretz, “Data management in cloud environments: NoSQL and NewSQL data stores,” J. Cloud Comput., Adv., Syst. Appl., vol. 2, no. 1, p. 22, 2013.

[25] Y. Zheng, “Methodologies for cross-domain data fusion: An overview,” IEEE Trans. Big Data, vol. 1, no. 1, pp. 16-34, Mar. 2015.

[26] X. Geng and K. S-Miles, “`Incremental learning,” in Encyclopedia Biometrics. New York, NY, USA: Springer, pp. 731-735, 2009.

[27] J. Wang, D. Crawl, S. Purawat, M. Nguyen, and I. Altintas, “Big data provenance: Challenges, state of the art and opportunities,” IEEE Int. Conf. Big Data (Big Data), Oct. 2015, pp. 2509_2516.

[28] P. Buneman, S. Khanna, and W.-C. Tan, “Data provenance: Some basic issues,” Foundations of Software Technology and Theoretical Computer Science. Berlin, Germany, pp. 87-93, 2000.

[29] N. Cao, L. Lu, Y.-R. Lin, F. Wang, and Z. Wen, “SocialHelix: Visual analysis of sentiment divergence in social media,” J. Vis., vol. 18, no. 2, pp. 221-235, 2015.

[30] D. Singh and C. K. Reddy, “A survey on platforms for big data analytics,” J. Big Data, vol. 2, no. 1, pp. 1-20, 2015.

[31] R.Vijayakumari, R.Kirankumar, and K.G. Rao, “Comparative analysis of Google File System and Hadoop Distributed File System,” International Journal of Advanced Trends in Computer Science and Engineering, vol. 3, no. 1, pp. 553-558, 2014.

[32] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, Erişim Tarihi: 28.04.2019.

[33] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html, Erişim Tarihi: 28.04.2019.

[34] https://kafka.apache.org/intro, Erişim Tarihi: 28.04.2019.

[35] https://spark.apache.org/docs/latest/ml-classification-regression.html, Erişim Tarihi: 30.04.2019.

[36] H. C. Chan, C. C. Chang, P. A. Chen and, J. T. Lee, “Using Multinomial Logistic Regression For Prediction of Soil Depth In an Area of Complex Topography in Taiwan,” Catena, vol. 176, pp. 419-429, 2019.

[37] Y. Zhou and, G. Qui, “Random Forest for Label Ranking,” Expert System With Application, vol. 112, pp. 99-109, 2018.

[38] S. Touzani, J. Granderson, and S. Fernandes, “Gradient Boosting Machine for Modeling the Energy Consumption of Commercial Buildings,” Energy and Buildings, vol. 158, pp. 1533-1543, 2018.

ÖZGEÇMİŞ

Engin BAYSAL, 10.12.1980 Şanlıurfa Birecik'te doğdu. İlk ve orta eğitimini Birecik'te, lise eğitimini ise Gaziantep'te tamamladı. 2008 yılında Marmara Üniversitesi Bilgisayar ve Kontrol Öğretmenliği (ing) bölümünü bitirdi. 2016 yılında Sakarya Üniversitesi Bilgisayar Mühendisliği Lisans programını tamamlayarak Mühendis unvanı aldı. Sakarya Üniversitesi Bilgisayar ve Bilişim Mühendisliği ana bilim dalında yüksek lisans eğitimine başladı. Halen İstanbul Gedik Üniversitesinde Öğretim Görevlisi olarak görev yapmaktadır.