BÜYÜK VERİ TEKNOLOJİLERİ İLE HİBRİT TAVSİYE SİSTEMİ GELİŞTİRİLMESİ

Muhammet Hüseyin OLGUN
Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Doç. Dr. Galip AYDIN

TEMMUZ-2018

T.C.

FIRAT ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BÜYÜK VERİ TEKNOLOJİLERİ İLE HİBRİT TAVSİYE SİSTEMİ GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Muhammet Hüseyin Olgun

141129113

Anabilim Dalı: Bilgisayar Mühendisliği

Programı: Kuramsal Temeller

Tezin Enstitüye Verildiği Tarih : 27 Temmuz 2018

Tez Danışmanı: **Doç. Dr. Galip AYDIN**

Üye: **Doç. Dr. Taner TUNCER**

Üye: **Dr. Öğr. Üyesi Muhammed TALO**

ÖNSÖZ

Klasik bilgi getirimi araçları, Google, Yandex, DuckDuckGo gibi, internet üzerindeki bilginin artmasıyla birlikte kullanıcılara ilgili bilgilerin getirilmesinde önemli bir rol oynamaktadır. Ancak kullanıcıların daha fazla kişiselleştirilmiş filtrelemeye ihtiyaç duyması tavsiye sistemlerinin son yıllarda internet üzerinde önemli bir yere sahip olmasında rol oynamıştır. İnternet üzerindeki kullanıcı verilerinin üstel olarak artışı ile birlikte internet üzerinde kişiselleştirilmiş bilgilerin kullanılarak kullanıcılara kişisel tavsiyeler üretilmesine olanak sağlamaktadır. Ancak verilerin büyüklüğü büyük veri teknolojilerinin kullanılarak tavsiye sistemi oluşturulması ihtiyacını ortaya çıkartmıştır. İnternet üzerinde hizmet veren Netflix, Amazon ve Spotify gibi önemli internet şirketleri tavsiye sistemleri üzerinde önemli çalışmalar ve yatırımlar yapmış, tavsiye sistemlerinin geliştirilmesinde önemli bir rol oynamışlardır.

Tavsiye sistemleri kullanıcıların istedikleri ürünlere daha kolay ulaşmasını sağladığından kullanıcılara büyük bir kolaylık sağlamaktadır. Ürünlere hızlıca ulaşan kullanıcılar tabi ki servis sağlayıcılara satış olarak geri dönmektedir. Aynı şekilde tavsiye sistemleri e-ticarette olduğu gibi sağlık, finans ve doküman sistemleri gibi alanlarda da kullanılarak büyük faydalar üretebilir.

Bu çalışmada dijital yayıncılık uygulaması olan GalePress için büyük veri teknolojileri kullanılarak tavsiye sistemi geliştirilmiştir. Tez, TÜBİTAK tarafından desteklenen 7150247 numaralı "Mobil Aygıtlar ile Kullanılabilecek Dijital Yayıncılık Platformu için Büyük Veri Altyapısının Kurulması ve Bulut Tabanlı Tavsiye Sistemi Geliştirilmesi" TEYDEB projesinin önemli bir kısmını içermektedir.

Muhammet Hüseyin OLGUN
ELAZIĞ - 2018

İÇİNDEKİLER

Sayfa No

ÖZET

Tez çalışmasında dijital yayıncılık platformu olan GalePress üzerinden alınan verilerle üzerinden tavsiye sistemi geliştirilmesi konusu işlenmiştir. Tez çalışması kapsamında tavsiye sistemleri ile ilişkili olan çalışma alanları ve tarihçeleri incelenmiştir. Ayrıca büyük veri teknolojileri kullanılarak oluşturulan mimariler incelenmiş, ilgili teknolojilerin literatür taraması yapılmış, teknolojiler karşılaştırılmış ve yine literatürde kullanılan tavsiye sistemi yöntemleri araştırılmıştır.

Yapılan genel literatür taraması ve araştırılmalarının ardından GalePress'in tavsiye sistemi için kullanacağı genel mimari ve kullanılacak teknolojiler ortaya konulmuştur. Kullanılacak teknolojiler ve mimari belirlendikten sonra, veriler oluşturulan platforma aktarılmış, ön analizi yapılmış daha sonra ön işlemlerden geçirilmiştir. Elde edilen veriler ile model eğitilmiş ve test edilmiştir. Son olarak hibritleştirme uygulaması ile sistem tamamlanmıştır.

Anahtar Kelimeler: Tavsiye sistemleri, büyük veri, bilgi getirimi

SUMMARY

Building a Hybrid Recommendation Engine on Big Data Technologies

In this thesis, development of recommendation system for GalePress, which is a digital publishing platform, has been discussed. Within the scope of the thesis study, the study areas and histories related to the recommendation systems were examined. In addition, architectures that built on big data technologies were examined, related technologies and type of recommendation systems in the literature were reviewed.

After the literature review, the architecture and the technologies to be used for GalePress's recommendation system had been revealed. Subsequently the technologies and architecture to be used had been determined, the data had been transferred to the platform, pre-analyzed and then pre-processed. The model had been trained and tested with the obtained data. Finally, the system has been completed with the hybridization application.

Keywords: Recommendation systems, big data, information retrieval

ŞEKİLLER LİSTESİ

Sayfa No

TABLOLAR LİSTESİ

Sayfa No

KISALTMALAR LİSTESİ

HDFS **:** Hadoop Distributed File System

SVD **:** Singular Value Decomposition

RDD **:** Resilient Distributed Dataset

TTY **:** Teletype Terminals

CSV **:** Comma-separated Values

TF-IDF **:** Term Frequency – Inverse Document Frequency

1. GİRİŞ

İnternet kullanımı ve internetin ticari kullanımı her geçen gün artmaktadır. [1] Artan kullanım ile birlikte tavsiye sistemleri özellikle ticari kullanımda önem kazanmaktadır. Tavsiye sistemleri, internet üzerinde artan kullanıcı sayısı ve veri boyutu ile beraber araştırmacılar arasında popüler bir hale gelmiştir. Çünkü artan veri boyutu ile birlikte tavsiye sistemlerinin başarımı da artmıştır. Bu noktada tavsiye sistemleri bilgi getirimi, dağıtık sistemler ve makine öğrenmesi gibi konuları da içererek gelişimini sürdürmektedir. Bu bölümde tavsiye sistemlerinde popüler olan bu alanların literatür incelemesi gerçekleştirilmiştir.

1.1. Bilgi getirimi

Bilgisayar bilimcileri, doğada, matematikte veya sosyal hayatta bulunan durumları gündelik problemleri çözmek amacıyla bilgisayar üzerinde benzetimini gerçekleştirir. Bilginin derlenmesi, depolanması, getirimi veya sınıflandırılması gibi problemler sadece bilgisayar bilimlerine özgü bir problem değil, yüz yıllar ve hatta binlerce yıldan beri insanların üzerinde çalıştığı bir konudur. Bu çalışma alanlarını, tabii olarak, ele alan ilk kurumlar kütüphanelerdir. Kütüphaneler binlerce yıldan beri bilginin nesilden nesile aktarılmasını sağlayan kurumlardır. Bilginin kütüphanelerde çoğalmasıyla birlikte, kolay ulaşılabilir olması için bilgi getirimi ve bir bilgi getirimi yöntemi sayılabilecek olan bilgi sınıflandırma yöntemleri geliştirilmiştir. Kütüphanelerde bulunan bilginin ilk sınıflandırma örneği M.Ö. 2500 ve M.Ö. 2250 tarihlerinde oluşturulduğu düşünülen Ebla tabletlerinde bulunmuştur. Bu tabletlerde bilginin konusuna göre sınıflandırıldığı tespit edilmiştir. [2]

Bilginin getiriminde önemli yeri olan bir yöntem kuskusuz kütüphanelerde de kullanılan kaynakçalardır. Bilinen ilk kaynakça İskenderiye Kütüphanesi'nde Callimachus tarafından oluşturulmuş daha sonra Abbasiler devrinde yasayan İbnü'n Nedîm tarafından yazılan Fihrist kitabıyla günümüze kadar medeniyetler arasında taşınmıştır. [3, 4] Günümüz kitaplarında da yine bir bilgi getirimi yöntemi olarak bulunan kaynakçalar, konuya göre ve terimlere göre fihristler bulunmaktadır.

1700’lerin sonlarından günümüze kadar modern bilim adamlarının araştırmaları sonucunda bilgi üretiminde büyük bir artış sağlanmıştır. Ayrı bir kitap olarak hazırlanan kaynakçaların ve katalogların yeni bilgiler eklendiğinde güncellenmesi ve bilgi artımıyla bu kadar bilginin bir kitaba sığdırılması zor hale gelmiştir. Bilgi getirimi amacıyla küçük boyutlarda dizin kartları oluşturulmuştur. Bu kartları geliştiren kişinin Carl Linnaeus olduğu kabul edilir. Linnaeus aslen bir biyolog olup bitkiler üzerinde araştırmalar ve sınıflandırmalar yapmaktadır. Bu anlamda modern taksonominin de kurucusu kabul edilir. [5]

1890 yılında bilgi getirimi ticari olarak Paul Otlet ve Henri La Fountain tarafından ele alınmış ve bilgi getiriminin dijital olmayan bir sürümünü hayata geçirmişlerdir. Kurdukları Mundaneum isimli şirketle milyonlarca bilginin dizin kartlarını tutmuşlar ve insanlık tarihinde üretilen bilgilerin merkezi olma hedefiyle yola çıkmışlardır. Mundaneum posta veya telgraf ile gelen isteklerle alakalı dizin kartlarının kopyalarını istek yapan kişiye iletilmek suretiyle bilgi getiriminin insan gücüne dayalı bir modelini ortaya koymuştur. [6]

Bilgisayar bilimlerinde bilgi getirimi yöntemleri binlerce yıldan beri süre gelen bu yöntemlerin benzetimi olarak ortaya çıkmaktadır.

1.1.1. Bilgisayar Bilimlerinde bilgi getirimi

Bilgisayar bilimlerinde, bilgi getiriminin tanıma yakın olarak yer alması Vannevar Bush’un 1945 yılında yayınladığı, *As We May Think*, isimli makalesinde *memex* ismini verdiği bir cihaz fikriyle ortaya çıkmıştır. Bush’un düşüncesindeki cihaz kişisel bir bilgi yönetimi aracı işlevi görmekte ve bilgi getirimi özelliği de bulunmaktaydı. [7]

Bilgisayar bilimlerinde bilgi getirimi teriminin teorik temellerini ise Calvin N. Mooers atmış ve bilgi getirimi terimini literatüre kazandırmıştır. Moorse, *Sayısal olmayan bilginin dijital olarak idare edimi ve makine ekonomisine etkilerinin teorisi*, isimli yayınında ilk olarak bilgi getirimi terimini kullanmış ancak tam olarak tanımını yapmamıştır. Bunu, *Sayısal olmayan bilginin dijital olarak idare edimi*, altında incelemiştir. [8] Bu makalesinde Moorse bir bilgi getirimi sisteminin en az iki kritere göre değerlendirmesi gerektiğini söylemiştir. Bunlar,

1. Sistem verilen işi yapıyor mu?
2. Sistemin depolama kapasitesi veya zamana göre maliyeti nedir?

Mooers'in koyduğu bu iki kriter sistemin uygulanmasında bilgisayarın donanımsal olarak kapasitesiyle birlikte ayrıca bilgi getirimi işleminde uygulanan yöntemin verimliliğini de kapsamaktadır.

Daha sonra yaptığı farklı bir yayında Mooers, *Making information retriaval pay*, başlığını kullanmış ve bilgi getiriminin tanımını, yaklaşık konu bilindiğinde depolanmış olan bilginin getirilme işlemi olarak tanımlamıştır. [9] Bu yayında ayrıca düşündüğü kavramın tam olarak anlaşılabilmesi için bilgi ambarından farkını, bilgi getiriminin, bilginin depolanması ve listelenmesiyle ilgili olmadığını bunun bilgi ambarının konusu olduğunu açıklayarak yapmıştır.

Mooers'in temelini attığı bilgi getiriminde üzerinde durulması gereken nokta aslında bilgi getiriminin klasik bir arama yöntemi değil daha çok keşif yöntemi olmasıdır. Mooers'a göre kullanıcı zaten aradığı bilginin ne olduğunu biliyorsa bilgi getirimi çok kolaydır. [9] Asıl konu kullanıcı ne aradığını aslında bilmiyorsa aradığı bilginin getirilmesidir. Bu nedenle Moorse'un temelini attığı bilgi getirimi konusu aslında soyut ve sezgisel bir konudur. Böyle olmasından dolayı kullanıcının aslında tam olarak ihtiyacı olan bilgiyi bulduğunu ispat etmek zordur. Kullanıcı bilgi getirimi sistemini istediğine yakın bir bilgi elde ettiğinde bırakabilir ancak bu kullanıcının tam olarak istediği bilgiyi ona ulaştırıldığı anlamına gelmez. Bu nedenle bilgi getiriminin başarımları genel olarak getirilen sonuçların, yapılan istek ile alakasına bakarak hesaplanır.

Sistemin başarımının zamana göre maliyetinin azaltılmasının tahmin edilebileceği gibi iki yolu vardır. Kullanılan donanımın teknolojik olarak gelişmiş olması veya algoritmik olarak kullanılan yöntemin verimli olması. Bu nedenle bilgi getiriminde indeksleme yöntemlerinin önemli bir yeri vardır.

1.1.2. Bilgi getiriminde indeksleme yöntemleri

Yapılan çalışmada literatür üzerinde 3 çeşit ana indeksleme yöntemine rastlanmıştır.

1.1.2.1. İkili getirim

İkili getirim yönteminde koleksiyonun içinde bulunan dokümanların içindeki terimlerin seti, satırları, dokümanlar ise sütunları oluşturur. Elde edilen terim-doküman

isabet matrisinde *(t, d)* değeri, *t* teriminin *d* dokümanında olup olmadığını ifade eder. Eğer terim dokümanda bulunuyorsa 1 değerini, eğer bulunmuyorsa 0 değerini alır.

Tablo 1.1. Örnek ikili getirim tablosu

	A	B	C	D	E	F	…..
Osman	1	0	0	0	1	1	
Ali	1	1	0	1	0	0	
Mehmet	0	0	1	1	0	0	
Aydın	1	1	1	1	1	0	
Cevdet	0	0	0	0	0	1	
……							

Tablo 1.1'de örnek olarak, A dokümanına bakıldığında Osman, Ali ve Aydın terimleri geçerken Mehmet ve Cevdet terimleri geçmediği görülmektedir.

Kullanıcı, Osman *AND* Ali *OR* Cevdet şeklinde bir sorgu yaptığında;

100011 *AND* 110100 *OR* 000001 = 100001 sonucu elde edilir. Yani sonuç A ve F dokümanları olacaktır.

1.1.2.2. Vektör Uzayı ve Terim Ağırlığı

Vektör uzayı ile bilgi getiriminde terim doküman isabet matrisine benzer bir matris oluşturulur. Bu yöntemin farkı dokümanları tam olarak eşleştirmek yerine skorlanmasıdır.

$$skor(\vec{k}, \vec{q}) = \sum_{i=1}^{m} k_i \cdot q_i \qquad (1.1)$$

Bir önceki bölümdeki örnek olarak alınırsa elimizde olan 100001 sorgusu, seriler üzerinden skorlandığında aşağıdaki sonuçlar Tablo 1.2'deki gibi olacaktır.

Tablo 1.2. Örnek skor tablosu

Doküman	Skor
A	1
B	0
C	0
D	0
E	1
F	2

Sonuçlar skorlara göre sıralandığında daha, F dokümanın yüksek skorla daha uygun bir eşleştirme olduğu görülür.

Salton ve Yang [10], vektör uzayı yöntemi yönteminden sonra sorgu vektörünün koleksiyonda ne kadar eşleştiği kadar sorgu elemanın koleksiyondaki ağırlığının da etki ettiğini görmüşlerdir ve buna ters doküman frekansı demişlerdir. Böylelikle terim frekansı ile ters doküman frekansını birleştirip aşağıdaki skorlama yöntemini ortaya koymuşlardır.

Terim frekansı f_i^k ile ifade edilecek olursa bu i teriminin k dokümanındaki bulunma sayısıdır. i teriminin N sayıdaki dokümanın bulunduğu koleksiyonda toplam bulunma frekansı, TF_i ise denklem 1.2'deki gibi ifade edilebilir,

$$TF_i = \sum_{i=1}^{N} f_i^k \qquad (1.2)$$

i teriminin koleksiyonda geçtiği doküman frekansını d_i ile, N ile de koleksiyondaki toplam doküman sayısını ifade edersek, IDF_i değeri denklem 1.3'deki gibi ifade edilebilir

$$IDF_i = 1 + \left(\log \frac{N}{d_i}\right) \qquad (1.3)$$

k dokümanın, j adet terimi bulunan q sorgusuna göre tf-idf skoru denklem 1.4'deki gibi her bir q_i değerinin TF_{q_i} ve IDF_{q_i} çarpımının toplamı olarak elde edilir,

$$skor(\vec{k}, \vec{q}) = \sum_{i=1}^{j} TF_{q_i} \cdot IDF_{q_i} \qquad (1.4)$$

1.1.2.3. Olasılıksal Skorlama

Olasılıksal skorlama bilgi getiriminde popüler ve modern yöntemlerden biridir. Bu yöntem verilen sorgunun dokümanlarla olan ilgisini, olasılıksal olarak hesaplayarak verir. Bu yöntem ilk olarak van Rijsbergen tarafıdan ortaya atılmış ve olasılıksal skorlama prensibi olarak adlandırılmıştır. [11] Böylelikle, sorgu q, doküman d, ikili arasındaki benzerlik skoru R, benzememe skoru NR olarak ifade edilecek olursa.

$$p(R \mid d,q) = \frac{p(d|R,q) \cdot p(R|q)}{p(d|q)} \quad (1.5)$$

$$p(NR \mid d,q) = \frac{p(d|NR,q) \cdot p(NR|q)}{p(d|q)} \quad (1.6)$$

$$p(R \mid d,q) + p(NR \mid d,q) = 1 \quad (1.7)$$

$$O(R \mid d,q) = \frac{p(R \mid d,q)}{p(NR \mid d,q)} \quad (1.8)$$

elde edilebilir.

Bu yöntemin devamı olarak ikili bağımsızlık modeli denilen olasılıksal bilgi getirim modeli, klasik Bayes kuralı sayılabilecek bu yöntemi dokümanların vektörel gösterimiyle birleştirir.

$$O(R \mid \vec{d},\vec{q}) = \frac{\frac{p(\vec{d}|R,\vec{q}) \cdot p(R|\vec{q})}{p(\vec{d}|\vec{q})}}{\frac{p(\vec{d}|NR,\vec{q}) \cdot p(NR|\vec{q})}{p(\vec{d}|\vec{q})}} = \frac{p(\vec{d}|R,\vec{q}) \cdot p(R|\vec{q})}{p(\vec{d}|NR,\vec{q}) \cdot p(NR|\vec{q})} \quad (1.9)$$

$$O(R \mid \vec{d},\vec{q}) = O(R \mid \vec{q}) \cdot \frac{\prod_i p(d_i|R,q_i)}{\prod_i p(d_i|NR,q_i)} \quad (1.10)$$

d_i burada 0 veya 1 değeri alacağından model aşağıdaki gibi düzenlenebilir.

$$O(R \mid \vec{d},\vec{q}) = O(R \mid \vec{q}) \cdot \frac{\prod_i p(d_i = 1|R = 1,q_i)}{\prod_i p(d_i = 1|R = 0,q_i)} \cdot \frac{\prod_i p(d_i = 0|R = 1,q_i)}{\prod_i p(d_i = 0\,|R = 0,q_i)} \quad (1.11)$$

Eğer, $p_i = p(d_i = 1|R = 1, q_i)$ ve $u_i = p(d_i = 1|R = 0, q_i)$ dersek ve değerleri logaritmik olarak sarmalarsak aşağıdaki ifade elde edilebilir.

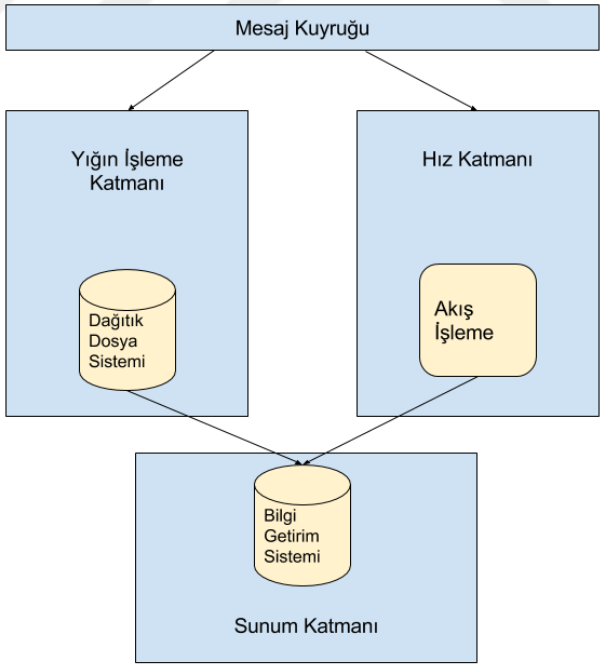
$$skor(d, q) = \log \prod_i \frac{p_i(1 - u_i)}{u_i(1 - p_i)} = \sum_i \log \frac{p_i(1 - u_i)}{u_i(1 - p_i)} \quad (1.12)$$

1.2. Dağıtık Veri İşleme

1.2.1. Lambda Mimarisi

Lambda mimarisi hem kullanıcıların geçmişteki hareketlerinde hem de yakın zamanda yapılan hareketlerine göre tavsiye verilmesini sağlayan bir veri işleme mimarisidir. Mimari hız katmanı, yığın işleme ve sunum katmanlarından oluşmaktadır.

Hız katmanı, kullanıcıların etkinliklerini akış işleme yöntemi ile işleyerek sonuç üretir. Bu katmanda Apache Kafka, RabbitMQ, Apache RocketMQ gibi mesaj kuyruğu teknolojileri kullanılmaktadır.



Şekil 1.1. Lambda mimarisi

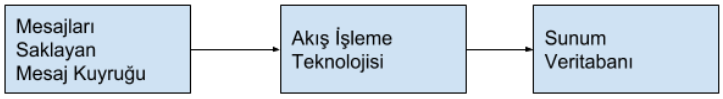
Yığın işleme, katmanı büyük hacimdeki verileri işleyerek sonuç üretir. Sonuç üretmesi verinin büyüklüğüne göre değişmektedir. Bazı uygulamalarda sonuçların üretilmesi haftaları bulmaktadır. Yığın işlemede veriler HDFS, GlusterFS gibi dağıtık dosya sistemlerinde tutulmaktadır. Bulunan veriler Apache Hadoop YARN, Apache Spark gibi yığın işleme teknolojileri ile işlenmektedir.

Sunum katmanı, sonuçların kullanıcılara sunulduğu katmandır. Bu katmanda Apache Solr, ElasticSearch gibi teknolojiler kullanılmakta ve sonuçların milisaniyeler mertebesinde ulaştırılması sağlanmaktadır.

1.2.2. Kappa Mimarisi

Şekil 1.2'de gösterilen Kappa mimarisi, Jay Kreps tarafından Lambda mimarisinden esinlenilerek ortaya çıkarılmış ve Lambda mimarisinin daha basitleştirilmiş halidir. [12] Kappa mimarisi Lambda mimarisinden farklı olarak yığın işleme katmanını kullanmamaktadır. Onun yerine bütün kullanıcı etkinlikleri kendi içerisinde kayıt altına alan bir mesaj kuyruğu kullanmak ve yığın işleme gerektiği zaman bu etkinlikleri tekrar üreten bir akış sağlamaktır. Böylelikle zaten kendi içerisinde yönetimi zorlayıcı olan dağıtık sistemlerden birinin yönetimi yapmak durumunda kalınmaz.

Kappa mimarisinin kullanımı için Apache Kafka, Apache RocketMQ gibi paylaşılan etkinlikleri kaydeden ve tekrar üretebilen bir mesaj kuyruğu kullanmak zorunludur.



Şekil 1.2. Kappa mimarisi

1.2.3. Liquid Mimarisi

Liquid mimarisi Kappa mimarisinin geliştirilmesi üzerine kurulmuştur. Kappa mimarisi yığın işleme işlemini bütün etkinliklerin tekrar üretilmesiyle yapmaktadır. Bu yüksek saklama ve işlem hacmi gerektirmektedir. Liquid mimarisinin de elde edilen etkinlikler sonuçları ile işleme zamanı ve yazılım versiyonu gibi açıklama verileri ile

saklanarak durum tutan iş parçacıkları oluşturur. Böylelikle daha önce hesaplanmış olan sonuçlar tekrar tekrar hesaplanmaz ve gerçek zamana yakın sonuçlar elde edilebilir. [13]

1.3. Tavsiye sistemleri

Tavsiye sistemleri günümüzde bilgisayar bilimlerinde makine öğrenmesi, bilgi getirimi ve dağıtık sistemler gibi birçok kuramın kesişim noktasında bulunmaktadır. Tavsiye sistemlerinin başarılı çalışması için yüksek başarımlı algoritmalar, yüksek hacimde veri ve yüksek hacimde hesaplama kaynağı gerekmektedir. Tavsiye sistemleri internet kullanımının artmasıyla beraber uygulamaların kullanıcılarına daha faydalı bilgiye ya da ürünlere ulaştırmak amacıyla kullanılmaktadır. Genelde kullanıcılar tavsiyeleri tarihsel olarak popüler olan hareketlere göre alırlar. Ancak genellikle gerçek dünyada insanlar tarihsel olarak popüler olan hareketlerin yanı sıra son zamanlarda popüler olan hareketlere de ilgi gösterir. Örneğin, medikal tavsiye sistemi üzerinde düşünüldüğünde, kışın soğuk algınlığı hastalığı popüler olabilir ve hastanın şikayetlerine göre bu tavsiyenin verilmesi gerekebilir. Diğer taraftan şeker hastalığı gibi mevsime bağlı olmayan hastalıklarda geleneksel olarak kullanıcıların şikayetleri benzerlik gösterdiğinden geçmişteki veriler kullanılabilir. Aynı şekilde bir müzik tavsiye sistemi, elektronik ticaret tavsiye sistemi gibi farklı alanlarda çalışan tavsiye sistemlerinde de kullanıcıda aynı davranışlar gözlenebilir.

Tavsiye sistemlerinde genel olarak müşterek filtreleme, içerik bazlı filtreleme ve hibrit filtreleme olarak üç genel yaklaşım bulunmaktadır. [14]

1.3.1. Müşterek Filtreleme

Müşterek filtreleme, sistemdeki bütün kullanıcıların hareket geçmişine bakıp kullanıcı nesne ilişkilerini belirleyerek, hedef kullanıcıya tavsiye vermeye yönelik bir tavsiye yöntemidir. [15]

Müşterek filtreleme, çalıştığı bilgi alanından bağımsız çalışabilir çünkü kullanıcılar ve nesneler arasında ilişkiler kurulmasını sağlar. Diğer taraftan verimli çalışabilmesi için yüksek hacimde kullanıcı hareketine ihtiyaç duymaktadır. Bu sebepten dolayı yeni

kullanıcılar soğuk başlangıç problemi ile karşılaşmaktadır. Soğuk başlangıç problemi veri seyrekliği nedeniyle kullanıcıya tavsiye verilememesi olarak tanımlanabilir. [16]

Müşterek filtrelemede iki yöntem bulunmaktadır. Bunlar komşuluk yöntemi ve saklı faktör yöntemidir. [17]

Komşuluk yöntemi, kullanıcıların tercihlerine göre birbirlerine yakınlıklarını ölçerek komşu olan nesnenin önerilmesi yöntemidir. Kısacası kendisine benzer tercihleri olan kullanıcıların diğer tercihlerine göre hedef kullanıcıya öneride bulunulur.

Saklı faktör yöntemi kullanıcılar tarafından derecelendirilmiş nesneleri, kullanıcıları birçok faktörü ele alarak karşılaştırır. Matris faktörlerine ayırma yöntemi saklı faktör yöntemlerinden en popüleri olarak karşımıza çıkmaktadır. Bu yöntemin avantajı yüksek kestirim oranına sahip olmasının yanı sıra paralel ve dağıtık işlemeye elverişli bir yöntem olmasıdır. [18]

Matris faktörlerine ayırma modelinde kullanıcılar ve nesneler f boyutunda faktörlerine ayırdığımızı varsayalım. Bu aşamada kullanıcıları ve nesneleri ifade eden iki matris elde ederiz. Bu matrislerden nesne matrisini q ile ifade edersek her i nesnesi q_i olarak ifade edilebilir. Kullanıcı matrisini p ile ifade edersek her u kullanıcısı p_u ile ifade edilebilir. q_i^T ile p_u'nun iç çarpımı bize yaklaşık bir tahmin verir.

$$r_{ui} = q_i^T \, p_u \qquad (1.13)$$

Bu noktada oluşturmak istediğimiz model tekil değer ayrışımı diğer ismi ile, *Singular Value Decomposition* ya da *SVD,* yöntemine benzetilmiş olur. SVD'de faktörlerine ayırma yöntemi A matrisini, V ile sağ tekil vektörlerini, U ile sol tekil vektörlerini ve D ile de tekil değerleri ifade edecek şekilde ayrıştırması probleminin çözülmesine dayanır.

Bulunan vektörlerin daha başarılı olması için ortaya çıkan tahminin kullanıcının gerçek tercihine yakın olması beklenir. Tahminin edilen değer ile gerçek değer birbirlerine ne kadar yakınsa, tahmin o kadar başarılı demektir. Bunun değerlendirmek için en küçük kare hatası yöntemi kullanılabilir. En küçük kare hatası yöntemini SVD ile bütünleştirildiğinde aşağıdaki gibi ifade edilebilir.

$$\min \sum_{(u,i)\in k} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \quad (1.14)$$

Bu aşamadan sonra bu sorun bir optimizasyon sorunu olarak görülebilir. Nasıl bir Q ve Pt matrisi gerçek matrisi en iyi şekilde temsil edebilir? Bu noktada iki yöntem kullanılmaktadır. Birincisi skokastik gradyan düşümü yöntemi diğeri ise değişen en az kareler yöntemi. [17]

Skokastik gradyan düşümü yönteminde her eğitim aşamasında sistem r_{ui}'yi tahminler ve tahmin hatasını ortaya çıkarır.

$$e_{ui} = r_{ui} - q_i^T p_u \quad (1.15)$$

Daha sonra her bir adımda kat sayılarını güncelleyerek gradyana ters uzantılı maksimum katsayıyı elde eder. [19]

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i) \quad (1.16)$$

$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u) \quad (1.17)$$

Burada elde edilen q_i ve p_u her ikiside bilinmezdir. Eğer bunlardan örneğin p_u değeri bilinirse q_i üzerinden en az kare problemi çözülerek hedefteki değer bulunabilir. Bu tam tersi için de mümkündür. Değişen kareler yöntemi daha masraflı olmasına karşın paralel olarak hesaplamaya daha uygundur. [20]

1.3.2. İçerik temelli filtreleme

İçerik temelli filtreleme, müşterek filtrelemeden farklı olarak, kullanıcılar ile diğer kullanıcılar arasında değil, nesneler ile kullanıcının profili üzerindeki korelasyonu üzerinden filtreleme yapar. [21] Örneğin, kullanıcının belirli bir dokümanı görüntülediğini varsayalım. Kullanıcının dokümanı okuması tam olarak o dokümanın istediği bilginin

kaynağı olduğu anlamına gelmeyebilir. Bu noktada kullanıcıya ona benzer dokümanlar önermek kullanıcıya yardımcı olacaktır. Bunun için ilk olarak kullanıcının okuduğu dokümanın terimleri sayılır ve ilk n terim alınarak her bir terimin koleksiyondaki ters doküman frekansı ile çarpılarak skorlanır. Skorlar üzerinden ağırlandırılarak sisteme kullanıcının ilgi alanlarına göre oluşturulan vektör ile sorgu gönderilerek içerik bazlı filtreleme yapılır. [22] Apache Solr'da kullanılan buna benzer nesneler özelliği bunun bir uygulamasıdır ve ileri ki bölümlerde uygulama ayrıntıları verilmiştir.

1.3.3. Hibrit filtreleme

Hibrit yöntemler müşterek filtreleme ve içerik temelli filtreleme yöntemlerinin birbirlerine olan dezavantajlarını ortadan kaldırmak amacıyla ortaya çıkmıştır. Hibrit yöntemlerde aşağıdaki yöntemler uygulanarak hibritleştirme yapılabilir, [23]

1. İçerik temelli ve müşterek filtreleme yöntemleri ayrıca uygulanarak sonuçlarını birleştirmek
2. Müşterek filtrelemeyi içerik temelli filtreleme ile birleştirmek
3. İçerik temelli filtrelemeyi müşterek filtreleme ile birleştirmek
4. Her iki filtrelemenin karakteristiğine sahip yeni bir model oluşturmak

Bu çalışmada birinci yöntem kullanılarak hibritleştirme yapılmış ve uygulama detayları ileri ki bölümlerde verilmiştir.

1.4. İlgili teknolojiler

Büyük veri uygulamalarının gelişmesi ile birlikte tavsiye sistemler üzerinde yapılan çalışmalar da artmaktadır. Bu bölümde sektörde ve literatürde bulunan açık kaynak tavsiye sistemi araçları incelenmiş ve bölümün sonunda ise karşılaştırmaları yapılmıştır.

1.4.1. Apache Mahout

Apache Mahout, dağıtık veri işleme teknolojileri üzerinde çalışabilen, makine öğrenmesi algoritmaları ve veri madenciliği uygulamalarının hızlı bir şekilde geliştirilmesini sağlayan bir kütüphanedir. Apache Mahout ilk sürümü HDFS üzerinde

MapReduce algoritmaları ile makine algoritmaları çalıştırsa da ilerleyen sürümlerinde değişen ekosisteme uyum sağlamış, Apache Spark, Apache Flink ve H2O gibi teknolojileri de kullanarak değişime uğramıştır.

1.4.2. Oryx 2

Oryx 2, Lambda mimarisinin bir uygulamasıdır. Oryx 2, mimari gereği, dört katmandan oluşmaktadır.

1.4.2.1. Yığın işleme katmanı

Yığın işleme katmanı sistemin başlangıç zamanından beri toplanan kullanıcı verilerini kullanarak model oluşturur. Bu işlem için Apache Spark'ı kullanmaktadır.

1.4.2.2. Hız katmanı

Hız katmanı kullanıcının son zamanlarda popülerleşen tercihlerini göz önünde bulundurarak modeli güncelleme görevi görmektedir. Oryx 2 bu işlem için Apache Spark Streaming kullanmaktadır.

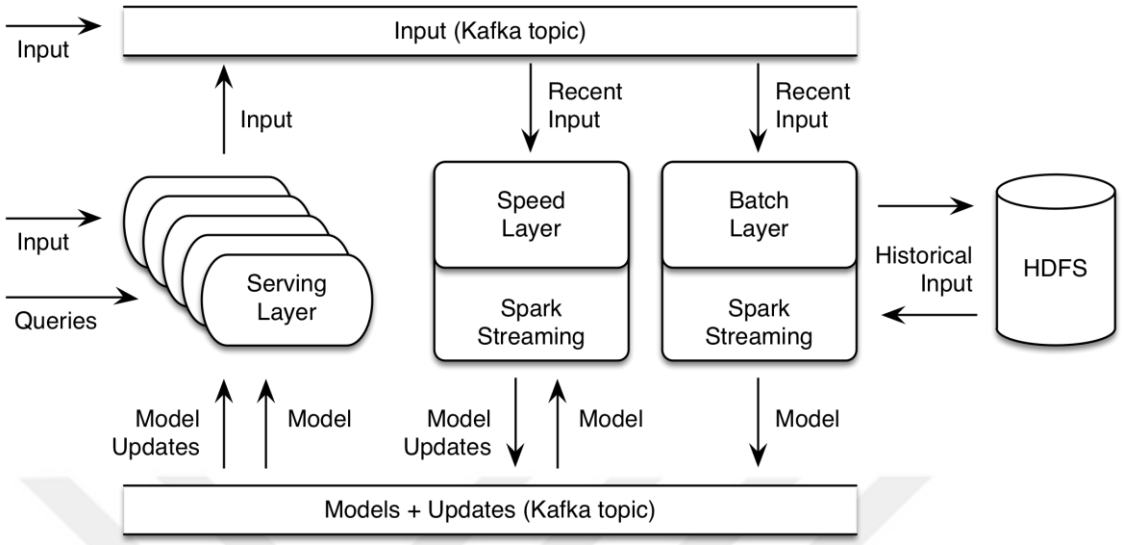
1.4.2.3. Sunum katmanı

Sunum katmanı, oluşturan modelleri ve güncellemeleri alarak istemciye hizmet sunan katmandır.

1.4.2.4. Veri gönderim katmanı

Bu katman katmanlar arasındaki iletişimi sağlar. Şekil 1.3'te verilen Oryx 2 bu katmanda Apache Kafka mesaj kuyruğunu kullanmıştır.

Sistemin mimari tasarımı aşağıdaki görselde gösterilmiştir. [24]



Şekil 1.3. Oryx mimarisi [24]

1.4.3. Seldon

Seldon, uygulama geliştiriciler için paket olarak makine öğrenmesi platformu sunan diğer bir teknolojidir. Ayrıca bulut bilişim için hazır paketler sunarak sistemin otomatik ölçeklenebilmesini sağlamaktadır. Seldon, veri ambarı olarak GlusterFS, konteynır yönetim teknoloji olarak Kubernetes, veri görselleştirme için Grafana gibi teknolojileri kullanarak bir paket halinde kullanılmaktadır. [25]

1.4.4. Apache PredictionIO

PredictionIO, popüler büyük veri teknolojileri kullanan diğer bir tavsiye sistemi uygulamasıdır. Veri ambarı olarak HBase kullanmasının yanı sıra PostgreSQL gibi geleneksel veri tabanlarını da desteklemektedir. PredictionIO'nun en önemli özelliği açık kaynak olarak platforma eklentiler yapılabilmesidir. [26]

1.4.5. Tavsiye sistemi araçlarının karşılaştırılması

İncelenen tavsiye sistemleri Tablo 1.3'te bulunan kriterlere göre karşılaştırılmıştır.

Tablo 1.3. Karşılaştırma tablosu

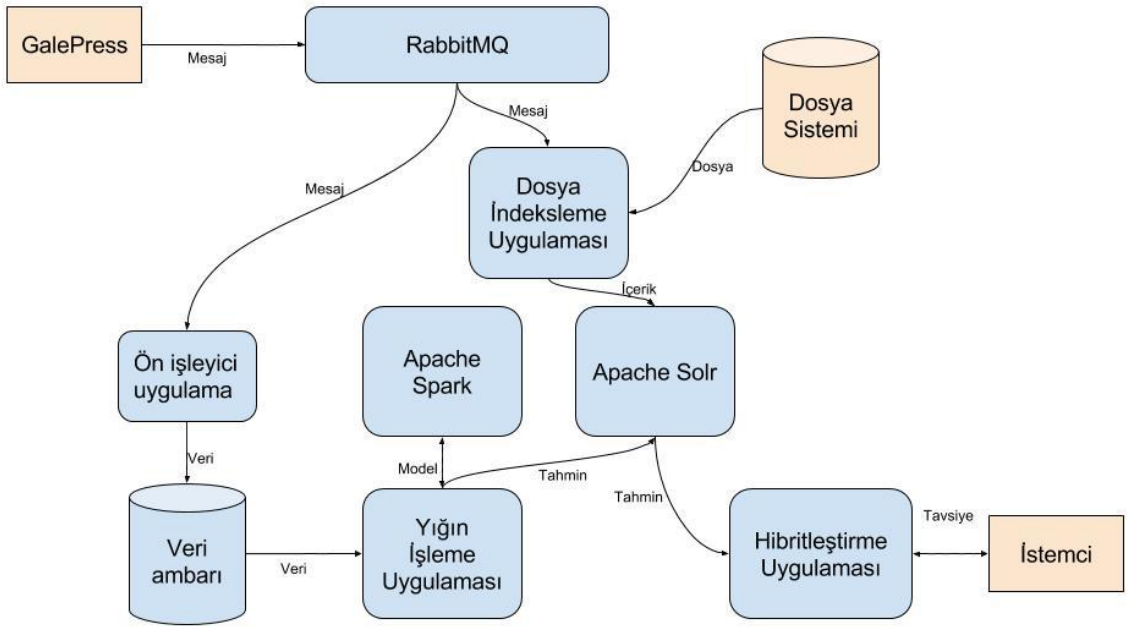
Teknoloji Adı	**Veri ambarı**	**Bulut desteği**	**Ölçeklenme**	**Otomatik ölçeklenme**	**Ticari destek**
Apache Mahout	Hayır	Evet	Evet	Hayır	Hayır
Oryx 2	Evet	Hayır	Evet	Hayır	Evet
Seldon	Evet	Evet	Evet	Evet	Evet
Apache PredictionIO	Evet	Hayır	Evet	Hayır	Evet

2. MATERYAL VE METOT İLE BULGULAR

2.1. Genel Mimari

Şekil 2.1'de verilen sistemin genel mimarisi, GalePress uygulamasının kapasitesi, ihtiyaçları, kaynakları ve bütçesi düşünülerek geliştirilmiştir. Bu nedenle Tavsiye Sistemleri bölümünde bahsedilen mimariler birebir uygulanmamıştır.

Mimari GalePress uygulamasında en az değişiklik yapılarak geliştirilmeye çalışılmıştır. GalePress uygulaması bu noktada sadece kullanıcıların etkinliklerini mesaj olarak sisteme göndermektedir. Mesajları RabbitMQ yönetmektedir. Gelen etkinlik mesajları ön işleyici uygulama ve dosya indeksleme uygulaması tarafından dinlenmektedir. Elde edilen etkinlikler ön işleyici uygulama tarafından sayfa izleme etkinliğine dönüştürülüp veri ambarına kaydedilmektedir. Bu noktada GalePress'in etkinlik verilerinin yakın zamanda üstel bir şekilde artış beklenmediği için ayrı bir MySQL veritabanı veri ambarı olarak kullanılmıştır.



Şekil 2.1. Genel mimari

Diğer taraftan dosya indeksleme uygulaması yeni bir içerik etkinliği olması durumunda ilgili dokümanı dosya sisteminden çekip Apache Solr'a sayfa bazlı olarak indekslemekten sorumludur.

Yığın işleme uygulaması veri ambarından dosyaları Apache Spark'a aktararak belirlenmiş işlerin çalıştırılarak model oluşumunu yapmakla sorumludur. İçerik temelli filtreleme için kullanıcı profillerini oluşturmanın yanı sıra yığın işleme uygulaması kullanıcıların tercihlerini model üzerinden geçirerek, tahminleri Apache Solr'a indeksler.

Hibritleştirme uygulaması da içeriğe ve kullanıcı hareketlerine göre kullanıcıya tavsiye vermekle sorumludur. Bu uygulamanın tasarımının ayrıntıları ileriki bölümlerde anlatılmıştır.

2.2. Kullanılan Teknolojiler

2.2.1. Scala

Scala, Martin Odersky öncülüğünde geliştirilen, nesne yönelimli programlama ve fonksiyonel programlama paradigmalarını birleştiren bir Java sanal makine dilidir. [27] Scala fonksiyonel programlama dillerinin paradigmalarını da aldığından, veri ve mesaj yönelimli programlama yöntemleriyle yazılan kod ile çok daha okunaklı bir dil olma özelliğini taşımaktadır.

2.2.1.1. Desen Eşleştirme

Desen eşleştirme, orijinal adıyla *Pattern Matching*, fonksiyonel programlama dillerinin vazgeçilmez bir fonksiyonudur. Bulunan herhangi bir veri yapısının veya değişkenin hangi türden olduğu desen eşleştirme söz dizimleri ile bulunmaktadır. [28]

Örneğin, yukarıda verilen kod parçacığında parametre olarak alınan doğal sayı *match* söz dizimi kullanılarak desen eşleştirme fonksiyonuna sokuluyor. Verilen değer *x* değişkenine atanıyor ve 9 ile 12 arasında ise 1, 13 ile 24 arasında ise 2, 24'ten büyük ise 3, diğer durumlarda ise 0 döndürüyor.

```
def decideRateFromDuration(duration: Int): Double = {
  duration match {
    case x if 9 until 12 contains x => 1
    case x if 13 until 24 contains x => 2
    case x if x > 24 => 3
    case _ => 0
  }
}
```

2.2.1.2. Kuyruklu Özyineleme

Saf fonksiyonel programlama dillerin olmazsa olmazı özyinelemeli fonksiyonlardır. Nesne yönelimli programlama dillerinde de özyinelemeli metotlar yazılsa da birbirini uzun süre çağıran özyinelemeli metotlar adres alanlarını şişireceğinden yığın taşması (*stack overflow*) durumlarıyla karşılaşmaktadır. Ancak fonksiyonel dillerin çoğunda bulunan kuyruklu öz yineleme (*tail recursion*) yöntemleri bu sorunu ortadan kaldırmaktadır. Scala'da örnek bir faktoriyel alan kuyruklu özyineleme fonksiyonu gösterilmiştir. [28]

```
def factorial(n: Int): Int = {
    @tailrec
    def factorialAccumulator(acc: Int, n: Int): Int = {
      if (n == 0) acc
      else factorialAccumulator(n * acc, n - 1)
    }
    factorialAccumulator(1, n)
  }
```

2.2.2. Docker

Docker, 2013 yılında geliştirilen ve Linux üzerinde kolay bir şekilde soyutlama sağlayarak çalışacak olan uygulamalar için bir sanallaştırma teknolojisidir. Klasik soyutlaştırma teknolojilerinin aksine Docker var olan Linux çekirdeği üzerinde çalıştığı için bir işletim sistemi için gerekli olan azami miktardaki kaynağı bile üzerinde çalıştığı

Linux çekirdeği kullanılarak uygulama sanallaştırma alanında büyük bir yenilik sağlamıştır. [29]

Docker üzerinde uygulama geliştiriciler kendi uygulamalarını basit bir söz dizimi ile paylaşabilmekte saniyeler içerisinde kendi imajlarını oluşturup, ortadan kaldırabilmektedir. Sunucu bilgisayar üzerinden sanal konteynır üzerinde portları birbirini bağlayabilmekte, aynı zamanda çeşitli dizinleri ortak kullanılması sağlanabilmektedir.

Docker, dağıtık sistemlerin yönetilmesini kolaylaştırmaktadır. Oluşturulan uygulamalar hızlı bir şekilde ayağa kalkmakta, *Kubernetes* ve *Docker Swarm* konteynır yönetim araçları ile hızlı bir şekilde ölçeklenmesi sağlanabilmektedir.

2.2.3. Apache Spark

Berkeley Üniversitesi'de bir grup araştırmacı, 2009 yılında dağıtık küme işleme teknolojisi olan Apache Spark'ı geliştirdiler. Apache Spark, Apache Hadoop gibi disk üzerinde çalışmak yerine hafıza üzerinde çalışıyor ve bu nedenle klasik *MapReduce*'a göre 100 kata kadar daha hızlı işlem yapıyor. [30]

Apache Spark çekirdeğinin dışında Spark SQL, Spark ML ve MLlib, Spark Streaming, Spark GraphX gibi dağıtık veri üstünde SQL komutları çalıştırılabilen, makine öğrenmesi algoritmaları koşabilen, akan verileri işleyebilen bir ekosisteme sahiptir.

2.2.3.1. Apache Spark Çekirdek Bileşeni

Apache Spark'ın çekirdek elementini RDD denilen elastik dağıtık veri seti nesneleri oluşturmaktadır. RDD nesneleri dağıtık ortamda herhangi bir düğüm sunucusunun düşmesi durumda verileri koruyabilen bir yapıya sahiptir. Bilinenin aksine Apache Spark veri kaynağı olarak Apache Hadoop'un dosya sistemi olan HDFS üzerinde çalışmak zorunda değildir. Bunun dışında GlusterFS, Amazon S3 gibi dağıtık dosya sistemleri üzerinde de çalışabilir.

RDD'ler temel olarak üç özelliğe sahiptir;

- RDD koleksiyonları değişmezdir ve sadece okunabilir.
- Düğüm sunucu kayıplarına karşı dirençlidir.
- Dağıtıktır.

2.2.4. Apache Zeppelin

Apache Zeppelin, Apache Spark üzerinde bulunan verinin hızlı bir şekilde işlem görerek incelenmesi ve incelemelerin görsel olarak izlenmesi için geliştirilmiş web ara yüzü bulunan bir araçtır.

2.2.5. Apache Solr

Apache Solr, açık kaynak büyük veri topluluğu tarafından bilinen metin indeksleme yöntemlerini kullanarak ölçeklenebilir olarak dokümanları indeksleyen bir arama motorudur. Apache Solr, metin analiz aracı olan Apache Lucene üzerine kurulmuş ve hızlı metin analizi için gerekli birçok özelliği karşılamaktadır.

2.3. Veri Tabanının İncelenmesi

2.3.1. İstatistik Tablosu

Galepress ürünün veritabanı elli tablodan oluşmaktadır. Veri tabanında bulunan bütün etkinlikler *Statistic* isimli tabloda tutulmaktadır. *Statistic* tablosunda Tablo 2.1'de verilen etkinlik türleri tutulmaktadır.

Tablo 2.1. Etkinlik türleri tablosu

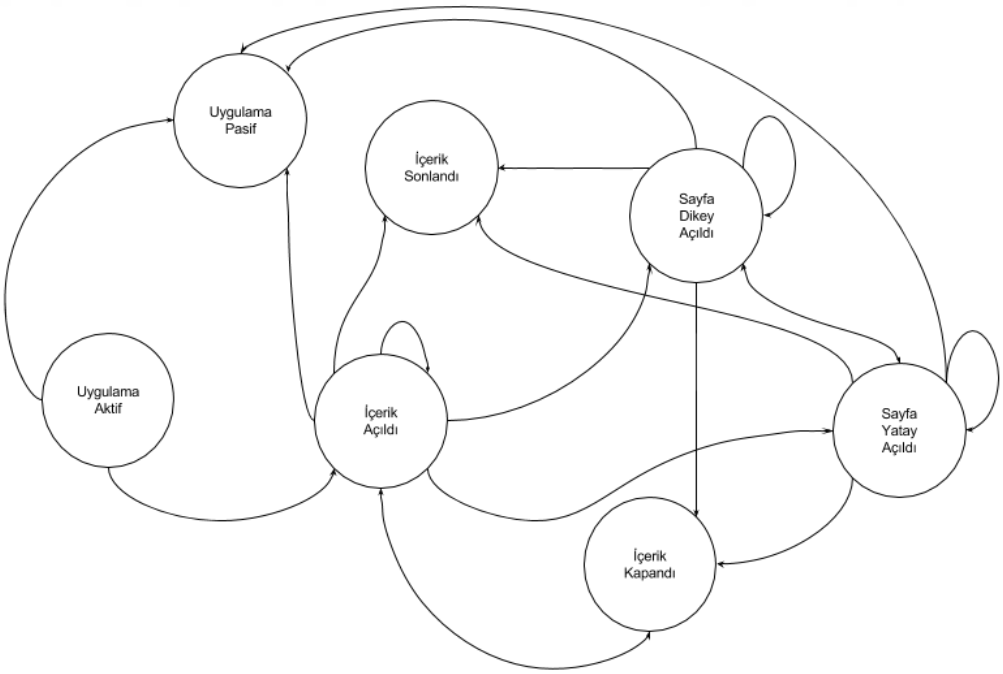
Uygulama Aktif	1
Uygulama Pasif	2
Uygulama Sonlandı	3
İçerik İndirildi	10
İçerik Güncellendi	11
İçerik Açıldı	12
İçerik Kapatıldı	13
Sayfa Dikey Olarak Açıldı	21
Sayfa Yatay Olarak Açıldı	22

Bunun yanında etkinliğin hangi yayıncının, hangi yayının, hangi sayfasında işlem yaptığı gibi bilgilerde tutulmaktadır.

2.3.2. Durum Geçişleri

Kullanıcının uygulama üzerinde yaptığı etkinlikler belirli ve sıralı bir şekilde hareket etmesi beklenmektedir. Ancak mobil uygulama üzerinden kullanıcı hareketleri izlenirken hemen hemen her durumdan neredeyse her durumu geçiş yapılabildiği tespit edilmiştir. Ayrıca etkinlik kaydı tutulması olgunlaşmadığından uygulamanın internet ile bağlantısının kesildiği durumlarda uygulama mobil cihazda yapılan etkinlikleri depolamadığı için tutarlı olmayan durum geçişleri yaşanması da çok yüksek bir olasılık olarak karşımıza çıkmaktadır.

Aşağıdaki çizgede kullanıcı etkinliklerinin hangi durumlardan hangi durumlara geçebileceği gösterilmiştir.



Şekil 2.2. Durum geçiş diagramı

2.4. Veri Tabanının Hazırlanması

Veri tabanı üzerinde veriyi analiz edebilmek için veri tabanı elimizde bulunan veri setinin her platformda kolaylıkla okunabilecek bir format olan CSV formatına çevirmek gerekir. Bunun için ilk olarak Galepress'in her ay alınan veri tabanı yedeğini lokal veri tabanı üzerinde aktarılması işlemi yapılmıştır.

İlk olarak yerel bilgisayar üzerinde *MySQL* veritabanı kurulmuştur. *MySQL* veritabanı kurulumu *Docker* yardımıyla aşağıda verilen komut ile kurulmuştur.

$> docker run --name galepress -p 3306:3306
-v /Users/molgun/database/docker:/var/lib/mysql
-v /Users/molgun/database/import:/tmp/import
-e MYSQL_ROOT_PASSWORD=galepress
-e MYSQL_USER=galepress
-e MYSQL_PASSWORD=galepress
-e MYSQL_DATABASE=galepress
-d mysql/mysql-server:8.0

docker run komutu Docker'ın var olan bir imaj üzerinden direkt olarak konteynırımızı başlatması için verilen komuttur.

Kullanılan parametreler aşağıda açıklanmıştır;

- name: Konteynırımıza var olan diğer konteynırlardan ayırmak için verdiğimiz isim için gerekli olan parametredir. Lokal makinede konteynırın adı 'galepress' olarak belirlendi.
- port: Konteynırın bir portunu yerel ağımızda bulunan porta yönlendirmek için gerekli olan parametredir. Port numarası belirlenirken <konteynır_portu> : <yerel_port> olarak bir söz dizimi kullanılır. MySQL'in var sayılan portu 3306 olduğundan konteynırın 3306 numaralı portu yerel ağın 3306 numaralı portuna bağlandı. Böylelikle yerel ağ üzerinden veri tabanına erişebilmesinin önü açılmış oluyor.
- volume: Docker yerel bilgisayarda bulunan bir klasörü konteynır ve yerel bilgisayarın birlikte kullanabileceği şekilde kullanabilir. Burada MySQL'in dosyalarını sağladığı dizini yerel sunucumuzun dizinine bağlayarak konteynırı herhangi bir sebeple kaybetmemiz durumunda yerel sunucudan erişebilmemizin önü açılıyor. Burada iki dizin

yerel sunucunun ve konteynırın ortak kullanımına açıldı bunlardan biri veri tabanı dosyaları ve veri tabanında içeri aktaracak olduğumuz SQL dosyasının dizini.

- env: Konteynır başlamadan önce çevresel değişkenlerini kaydetmek için kullanılan parametredir. Liste olarak değer aldığından birden fazla kez kullanılabilir. Bu parametre bu komutta MySQL veri tabanın yönetici şifresi, kullanıcı adı, kullanıcı şifresi ve veri tabanı ismini kaydetmek için kullanıldı.
- detach: Bu parametre konteynırın arka planda çalışmasını sağlamak için verilmektedir. Verilmediği taktirde komut satırında anlık olarak çıktıları devamlı olarak paylaşır.

Son olarak Docker imaj sunucunda var olan MySQL veri tabanı imajının sekiz numaralı versiyonunu kullanacağımızı belirtiyoruz.

2.5. Veri Setinin Veri Tabanına Aktarılması

Veri tabanı kurulduktan sonra SQL dosyası formatında olan veri seti veritabanına aktarılır. Bunun için çalışmakta olan konteynırın komut satırını aşağıdaki komut ile bağlanılır.

$> docker exec -it galepress bash

docker exec komutu Docker'ın dışarıdan konteynır içinde komut çalıştırma komutudur. Yerel sunucu üzerinden konteynıra istenilen komutu gönderilebilir. Bu çalıştırmada konteynırın sözde komut satırına interaktif olarak erişmek için farklı parametreler kullanılmıştır.

Kullanılan parametreler aşağıda açıklanmıştır;

- interactive: Yerel sunucunun giriş akışını açık tutar.
- tty: Konteynır içerisinde komutların çalıştırılması için sözde bir TTY konumlandırır.

Komut çalıştırıldıktan sonra konteynır içerisindeki sözde komut satırı üzerinden komutlar çalıştırılabilir.

Daha sonra konteynır içerisinde aşağıdaki komut çalıştırılarak yerel sunucudan konteynırın ortak kullanımına sunulan SQL dosyasının veri tabanının içe aktarma işlemi yapılır.

$> cd /tmp/import && mysql -u galepress -p galepress < admin_galepress_2016_10_08.sql

mysql komutu veri tabanı üzerinde temel işlemleri yapmak için kullanılır. Kullanılan küçüktür işareti giriş olarak dosya verildiğini işaret eder.

Kullanılan parametreler aşağıda açıklanmıştır;

- u: Bağlanılacak olan veri tabanının kullanıcı adı. Konteynır oluştururken vermiş olunan *galepress* girilir.
- p: Bağlanılacak olan veri tabanı kullanıcısının şifresi. Konteynır oluştururken vermiş olunan *galepress* girilir.

Komut çalıştırıldıktan uzun bir süre sonra veri seti veri tabanına aktarıldı.

2.6. Veri Setinin CSV Formatında Dışa Aktarılması

Veri tabanı konteynırının sözde komut satırı üzerinden veri tabanının komut satırına aşağıdaki komut ile bağlanılır.

$> mysql -uroot -p

MySQL'in SQL komutları çalıştırma alanında aşağıdaki SQL komutu istatistik tablosunu verilen dizine aktarır.

use galepress;

SELECT StatisticID, Type, Time, DeviceID, CustomerID, ApplicationID, ContentID, Page INTO OUTFILE '/var/lib/mysql-files/statistics.csv' FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' LINES TERMINATED BY '\n' FROM Statistic;

USE komutu, üzerinde çalışacak olan veri tabanını komut satırında belirlemek için kullanılır. Var olan veri tabanının adı galepress olduğundan değere galepress yazılmıştır.

SELECT FROM söz dizimi, ilişkisel veri tabanlarında sıklıkla kullanılan seçme komutudur. İstatistik anahtarı, istatistik tipi, istatistik zamanı, cihaz anahtarı, kullanıcı anahtarı, uygulama anahtarı, içerik anahtarı ve sayfa numarası alanlarını istatistik tablosundan çekmek için kullanıldı.

INTO OUTFILE söz dizimi, seçilmiş olan verinin bir dosyaya yazılması gerektiğini işaret eden parametredir. Verilen dosya dizini MySQL dosyalarının bulunduğu dizini işaret etmektedir. Diğer dosya dizinleri kullanıcı izinleri nedeniyle yetkilendirme hatası almaktadır.

FIELDS TERMINATED BY söz dizimi, seçilmiş olan satırlardaki kolonların hangi karakter ile ayrılması gerektiğini işaret eden parametredir. Bu parametre CSV dosyalarında virgül olduğundan virgül değeri verilmiştir.

OPTIONALLY ENCLOSED BY söz dizimi, seçili olan hücrelerin belirli bir karakterle sarmalanması için verilen parametredir. Bu parametre CSV formatında çift tırnak olduğundan değer olarak bu karakter verilmiştir.

LINES TERMINATED BY söz dizimi, her bir satırın hangi karakter ile bittiğini işaret eden parametredir. Parametre CSV formatında alt satır olduğundan alt satır karakteri verilmiştir.

İşlem tamamlandıktan sonra yerel sunucu ve konteynırın ortak kullandığı dosya dizinine CSV dosyası kopyalanır.

$> cp /var/lib/mysql-files/statistics.csv /tmp/import/

cp, en bilinen UNIX komutlarından biridir. Komut, belirli bir dosyayı belirli bir dosya dizinine kopyalar. İlk kaynak dosya ikinci değer ise hedef dosya dizinidir.

2.7. Apache Zeppelin Kurulumu

docker run -d -v /Users/molgun/database/import:/tmp/data -p 8080:8080 -p 7077:7077 -p 4040:4040 epahomov/docker-zeppelin

Apache Zeppelin, Docker üzerinde kurmak için aşağıda bulunan komutu çalıştırmak yeterlidir. Açık kaynak olarak geliştirilen Apache Zeppelin Docker imajı, içerisinde Apache Hadoop ve Apache Spark ve gerekli olan yan araçları da içermektedir.

Komut çalıştırıldıktan sonra Apache Zeppelin web arayüzüne yerel sunucunun 8080 numaralı portundan erişilebilir.

2.8. Veri Ön Analizi

Veri setinin ne türlü bir ön işlemden geçirilmesi gerektiğini anlamak için ilk olarak çeşitli hipotezlerin cevaplarını görsel olarak Apache Zeppelin üzerinden görülmesi gerekmektedir. Bunun için ilk olarak CSV formatındaki veri seti Apache Zeppelin üzerinden Apache Spark'ın hafızasına yazılır.

val statistics = sc.textFile("/tmp/data/statistics.csv")

Çalıştırılan kod ile CSV dosyası RDD dosyası olarak Apache Spark üzerinde depolanmış olacaktır. Daha sonra veri setini oluşturmak için her bir etkinliği tutmak için Scala'da bulunan durum sınıfı oluşturulur.

case class Event(statisticID: Int, sType: Int, date: String, deviceId: String, customerID: String, applicationID: String, contentID: String, page: String)

Tanımlamış olunan etkinlik durum sınıfı, istatistik tablosundaki her bir satırı ifade etmektedir. Daha sonra RDD üzerinde bulunan CSV dosyamızı durum sınıfları tarafından tanımlanmış olan nesnelere dönüştürüyoruz.

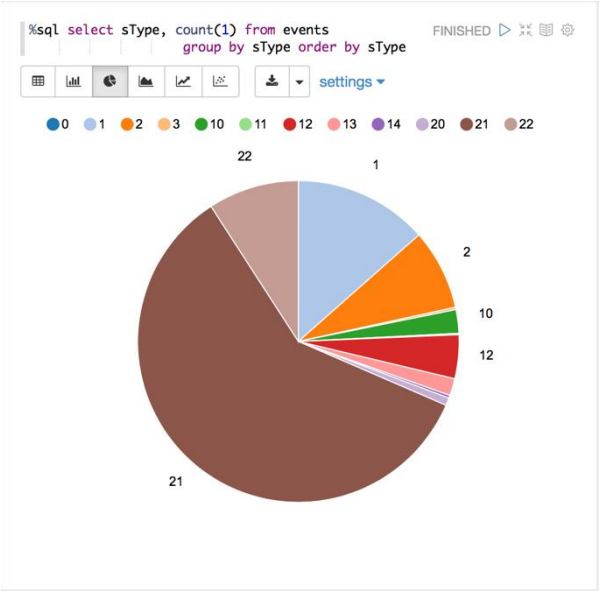
```
val events = statistics.map(s=>s.split(","))
  .map(
     s=>Event(s(0).toInt,
           s(1).toInt,
           s(2).replaceAll("\"", ""),
           s(3).replaceAll("\"", ""),
           s(4).replaceAll("\"", ""),
           s(5).replaceAll("\"", ""),
           s(6).replaceAll("\"", ""),
           s(7).replaceAll("\"", "")
     )
)
```

Çalıştırılan 'map' metodu RDD üzerinde bulunan her bir kayıt için işlemi tekrarlayan bir döngü niteliğindedir. Daha çok veri dönüşümleri için kullanılır. 'Map' metodu ile ilk olarak CSV standardı gereği virgül ile ayrılmış kayıtları birbirinden 'split' metodu ile ayrılarak bir dizi haline getiriliyor. Ayrılan her bir eleman üzerindeki veriler üzerinden durum nesnesi oluşturuluyor.

events.toDF().registerTempTable("events")

Daha sonra elde edilen etkinlik RDD'sini veri çerçevesine çevrilip 'events' isimli bir tabloya kaydedilir. Elde edilen tablo ile Apache Spark SQL üzerinden hızlı sorgular çalıştırıp Apache Zeppelin ile görsellerini elde edebilir hale getirilebilir.

Veri seti sorgulanmaya hazır hale geldikten sonra ilk olarak kullanılabilecek olan etkinlik sayısını incelendi.

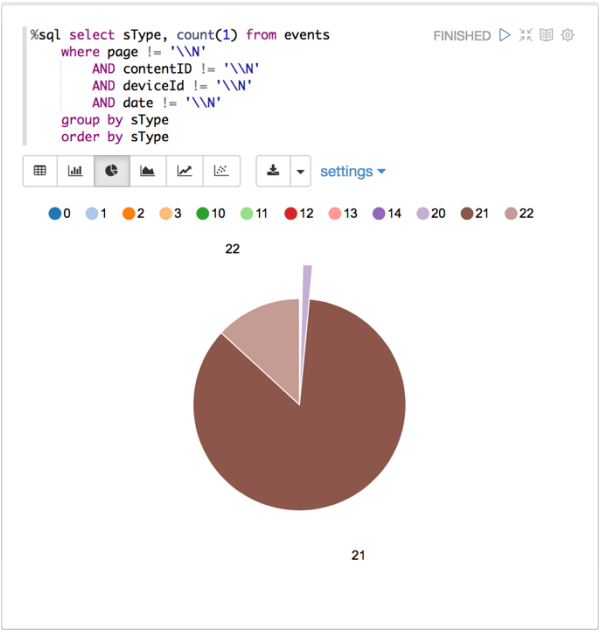


%sql select sType, count(1) from events
group by sType order by sType
FINISHED

sType	count(1)
0	574
1	781,309
2	469,931
3	13,844
10	137,250
11	8,054
12	254,800
13	99,894
14	15,910
20	44,745
21	3,443,016
22	530,124

Şekil 2.3. Etkinlik sayılarının türlere göre dağılımı

Verilen sorguyla en fazla etkinlik 21 ve 22 numaralı etkinlikler olduğu görülmektedir. Bu etkinlikler bir derginin sayfasının açıldığı zamanı tutmaktadır.



%sql select sType, count(1) from events
where page != '\\N'
AND contentID != '\\N'
AND deviceId != '\\N'
AND date != '\\N'
group by sType
order by sType
FINISHED

sType	count(1)
1	4,714
2	2,254
3	29
10	1,952
11	357
12	3,554
13	2,319
14	160
20	44,745
21	3,443,014
22	530,120

Şekil 2.4. Filtrelenmiş etkinlik sayılarının dağılımı

Sayfa açılma etkinlik verisinin kullanılabilmesi için hangi derginin hangi sayfasının açıldığının da bilinmesi gerekmektedir. Bunun için bu bilgilerin bulunmadığı verileri

filtrelenmelidir. Şekilde görüldüğü gibi bu veriler filtrelendiğinde yaklaşık altı adet kaydın tutarsız olduğu görülmektedir.

2.8.1. Kayıt Tarihinde Veri Tutarsızlığı

Dikkat edildiği üzere tarih kolonu veri tabanında karakter dizisi olarak tutulmaktadır. Kayıt tarihleri istemci mobil cihaz tarafından sunucuya gönderilmekte ve sunucu tarafında tarihin geçerli olup olmadığı kontrol edilmediğinden, çeşitli tutarsızlıklar oluşmaktadır.

Bunu anlamak için veri seti üzerinden bir örnek parça alındı.

events.takeSample(false, 1)

Sonuç olarak aşağıdaki dizi elde edildi,

Array[Event] = Array(Event(1184450,21,2015-03-11 01:30:37??????+2,432D0FBE-3E10-4992-ABFC-A01E86DAC5BC,90,95,1745,14))

Görüldüğü gibi etkinlik nesnesinin kayıt tarihi değerinde çeşitli tanımlanamayan karakterler elde edilmiştir. Bu eksiklik verinin dönüştürülmesini zorlaştıracaktır. Tarih verisinin kaç farklı türde olduğu tespit edildiğinde veri dönüşümü için çözüm üretmek kolaylaşacaktır. Bunun için her bir tarih karakter dizisinin kaç parçadan oluştuğunu kayıt altına alıp yeni bir sorgulama tablosu oluşturulup sorgulanmış ve aşağıdaki sonuçlar elde edilmiştir.

```
val preProcessed = events.filter(e => e.date != "\\N")
                .map(e => e.copy(date= e.date.split(" ").length.toString))
preProcessed.toDF().registerTempTable("preProcessed")
```

```
%sql select date, count(1) from preProcessed
                group by date
```

FINISHED

date	count(1)
3	3,487,801
5	157
1	3,628
4	20,066
2	2,287,799

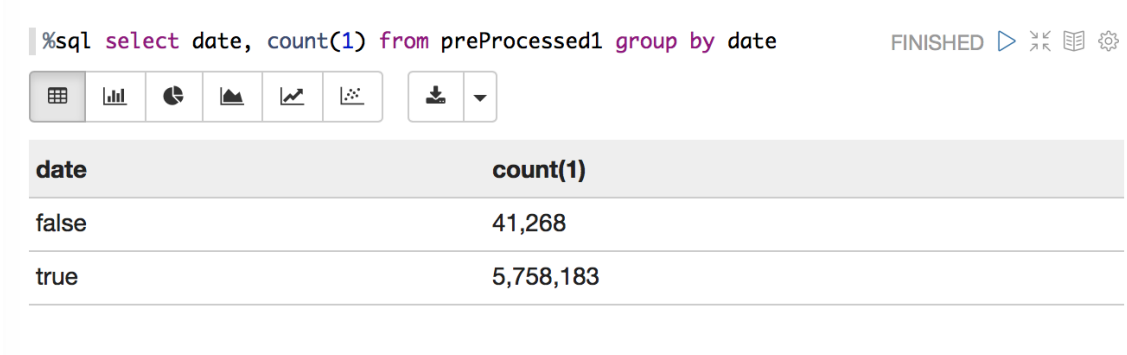
Şekill 2.5. Tarih desenin kaç parçadan oluştuğunu gösteren verilerin dağılımı

Görüldüğü üzere tarih karakter dizileri en fazla iki ve üç parçadan oluşmaktadır. İki parçadan oluşan tarihler tarih ve zaman, üç parçadan oluşan tarihler ise tarih, zaman ve yerel tarih bilgisini içermektedir.

Mobil cihazlar çok yüksek oranda aynı yerel lokasyonda bulunduğundan verimizin kalitesine etki etmeyecektir. Yerel tarih bilgisi yok sayılarak her bir tarih, tarih ve saat formatına çevrilip işlem yapılmıştır. Bu veri dönüştürmesine uyan veri sayısı aşağıdaki şekilde sorgulanmıştır.

```
val preProcessed1 = events.filter(e => e.date != "\\N")
                .map(e => {
                  val dateRX = """(\d\d\d\d)-(\d\d)-(\d\d) (\d\d):(\d\d):(\d\d)""".r
                  val value = dateRX findFirstIn e.date match {
                    case Some(dateRX(y, mo, d, h, min, s)) => "true"
                    case _ => "false"
                  }
                  e.copy(date= value)
                })
preProcessed1.toDF().registerTempTable("preProcessed1")
```

Verilen kod parçacığında ilk olarak veri setinden tarih verisi eksik olan nesneler filtreleniyor. Tarih ve saati ifade eden kurallı ifade *dateRX* değişkenine atanmıştır. Scala dilinde bulunan desen eşleştirme ifadesi kullanılarak verilen kurallı ifade ile eşleşen değerler için *true* değeri, eşleşmeyenler içinse *false* değeri atanmıştır. Her bir değerin sayıları aşağıdaki gibi bulunmuştur.



date	count(1)
false	41,268
true	5,758,183

Şekil 2.6. Geçerli desene sahip verilerin dağılımı

Sonuç olarak kayıt tarihi verisinin büyük bir çoğunluğunun istenilen desene uyduğu görülmüş ve dönüşümü yapılmıştır.

2.9. Veri Ön İşleme

Daha önceki bölümlerde de belirtildiği gibi kullanıcının her bir sayfa üzerinde geçirdiği vakit sayfayla ilgilenme oranı olarak kabul edilecektir. Bunun için istatistik verisinde bulunan tarihlerin karakter dizisi tipinden tarih formatına çevrilip iki etkinlik arasındaki farktan sayfa üzerinde geçirilen zaman bulunacaktır. Yapılan ön analiz sonrasında bu bölümde veri ön işleme ve dönüşümlerinin nasıl yapıldığı anlatılmıştır.

2.9.1. Tarihin Karakter Dizisi Tipinden Tarih Tipine Çevrilmesi

Spark üzerinde aşağıdaki metot kullanılarak karakter dizisi tarih dizisine çevrilmiştir.

```
def convertDate(s: String):Option[java.sql.Timestamp] = {
  val dateRX = """(\d\d\d\d)-(\d\d)-(\d\d) (\d\d):(\d\d):(\d\d)""".r
  dateRX findFirstIn s match {
   case Some(dateRX(y, mo, d, h, min, s)) => {
    val df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
    Option.apply(
      new java.sql.Timestamp(
        df.parse(y + "-" + mo + "-" + d + " " + h + ":" + min + ":" + s).getTime
      )
    )
   }
   case _ => Option.empty
  }
}
```

Veri ön analizinde kullanılan dönüştürme metotları bu bölümde *refaktör* edilerek ve tarih dönüşümü yapılarak aşağıdaki biçime getirildi.

```
val events = statistics.map(s=>s.split(","))
  .filter(s =>  s(2) != "\\N")
```

```
.filter(s => convertDate(s(2).replaceAll("\"", "")) isDefined)
.map(
  s=>Event(s(0).toInt,
        s(1).toInt,
        convertDate(s(2).replaceAll("\"", "")) get,
        s(3).replaceAll("\"", ""),
        s(4).replaceAll("\"", ""),
        s(5).replaceAll("\"", ""),
        s(6).replaceAll("\"", ""),
        s(7).replaceAll("\"", "")
  )
```

Kullanıcıların sayfa başında harcadığı süreleri bulmak için cihaz anahtarı üzerinden her bir kullanıcıların hareketleri aşağıdaki komut ile birleştirildi.

```
val x = events
  .map((e) => (e.deviceId, e))
  .aggregateByKey(List[Event]())(
    (acc, e) => acc ::: List(e),
    (acc1 , acc2) => acc1 ::: acc2
  )
```

Bu dönüşümden sonra her bir sayfada kalınan süreyi temsil eden sınıf aşağıdaki alanlar ile oluşturuldu.

case class PageViewed(id: Int, duration: Long, deviceId: String, customerID: String, applicationID: String, contentID: String, page: String)

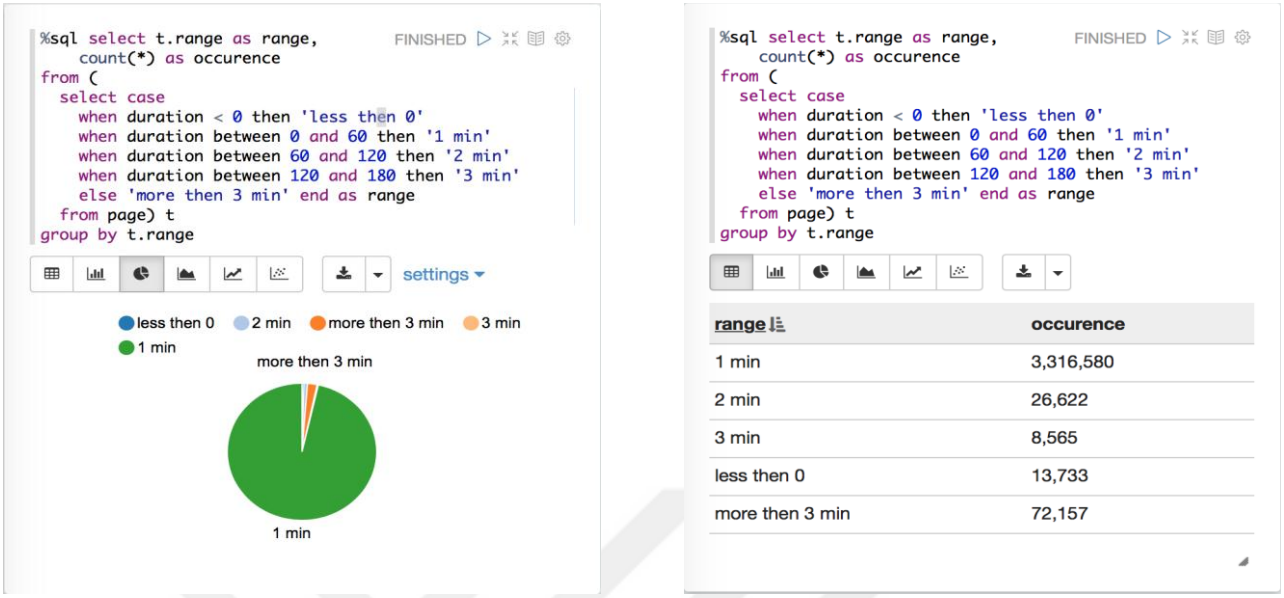
Daha sonra cihaz anahtarı üzerinde birleştirilen hareketlerin Scala ile uygulaması aşağıdaki gibidir.

```
def transform(list: List[Event]) : List[PageViewed] = {
  @scala.annotation.tailrec
  def viewAccumulator(list: List[Event], acc: List[PageViewed] =
List.empty[PageViewed]) : List[PageViewed] = {
    list match {
      case Nil => acc
```

```
        case e :: tail if (e.sType == 20 || e.sType == 21) && tail.nonEmpty =>
          viewAccumulator(
            tail,
            PageViewed(e.statisticID,
              (tail.head.date.getTime - e.date.getTime) / 1000,
              e.deviceId,
              e.customerID,
              e.applicationID,
              e.contentID,
              e.page
            ) :: acc
          )
        case e :: tail => viewAccumulator(tail, acc)
      }
    }
    viewAccumulator(list)
  }
```

2.10. Sayfa Görüntülenme Sürelerinin İncelenmesi

Aşağıda gösterilen şekilde kullanıcıların sayfaları sıfır saniyeden az, 0 ile 60 saniye, 60 ile 120 saniye, 120 ile 180 saniye aralıklarında ve 180 saniyeden daha fazla olacak şekilde hareketleri gösterilmiştir. Kullanıcılar en fazla hareketi 0 ile 60 saniye arasında yaptığı görülmektedir. 0 saniyeden küçük olan hareketler daha önce verilerin toplanmasından kaynaklanan sorunlardan kaynaklandığından bu veri çalışmada filtrelenmiştir.



Şekil 2.7. Geçerli desene sahip verilerin dağılımı

2.11. Ortalama Sayfa Görüntülenme Sürelerinin İncelenmesi

1. Şekil 2.8'de görüldüğü üzere, her aralık için ortalama izlenme sürelerine bakıldığında,

2. Bir dakika aralığında ortalama 3.8 saniye
3. İki dakika aralığında ortalama 83 saniye
4. Üç dakika aralığında ise 146 saniye izlenme süresi olduğu görülmüştür.

En fazla sayfa görüntülenme süresi sıfır ile 60 sayfa arasında olduğundan bu aralık bir kere daha incelenmiştir.

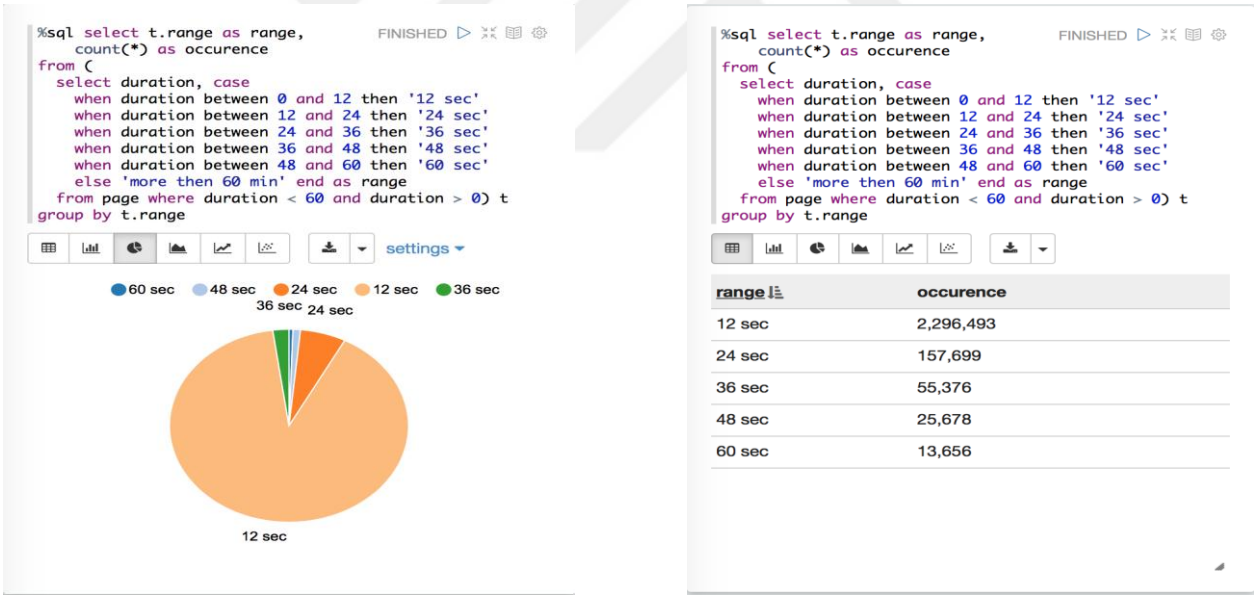
```
%sql select t.range as range,
avg(duration) as occurence
from (
  select duration, case
    when duration < 0 then 'less then 0'
    when duration between 0 and 60 then '1 min'
    when duration between 60 and 120 then '2 min'
    when duration between 120 and 180 then '3 min'
    else 'more then 3 min' end as range
  from page) t
group by t.range
```

FINISHED

range	occurence
1 min	3.84526
2 min	83.43873
3 min	146.34349
less then 0	-2,082,331.92261
more then 3 min	979,506.742

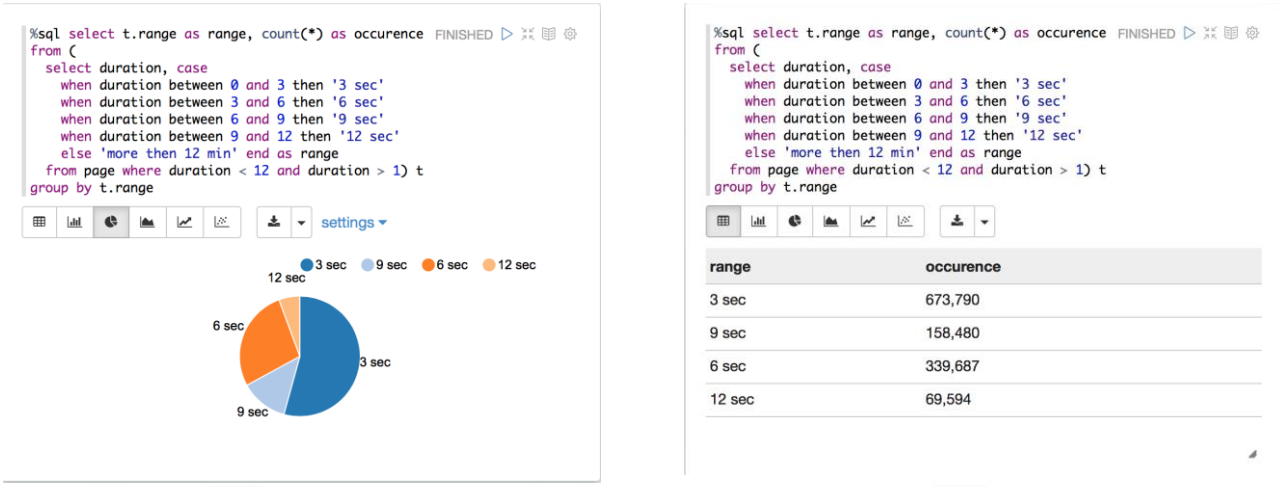
Şekil 2.8. Ortalama izlenme sürelerine göre verilerin dağılımı

Şekil 2.9'da 60 saniye aralığı 5 eşit parçaya bölünerek hangi aralıklarda daha fazla görüntülenme süresi olduğu inceleme sonuçları görülmektedir.



Şekil 2.9. Bir dakika aralığı için izlenme sürelerine göre verilerin dağılımı

Görüntülenme süresi verilerinin 0 ile 12 saniye arasında hala bir çoğunluk oluşturulduğu görülmektedir. Bu nedenle 0 ile 12 saniye aralığı da 4 parçaya bölünerek dağılımına bakılmıştır.



Şekil 2.10. Daraltılmış aralıkta verilerin dağılımı

Şekil 2.10'da görüleceği üzere dağılım bu noktada düzenli bir şekil almıştır.

2.12. Ön işlemden geçirilen verilerin dışa aktarılması

Veri seti ön işlemden geçtikten sonra Apache Zeppelin gibi not defteri amacıyla oluşturulmuş bir araçtan çıkıp, elde edilen çıktıların gerçek dünya uygulamasında çalıştırılması için dışa aktarılması gerekmektedir. Ön işlemden geçirilen sayfa görüntülenme süreleri aşağıdaki komut ile dışa aktarılmıştır.

df.write.csv("/tmp/data/galepress")

Bu komut ile oluşturulan Spark DataFrame'i belirtilen dizine veri buketi olarak kaydedilmiştir. Bu çalışma da ortalama 15 megabyte büyüklüğünde 22 dosya CSV formatında dışarı aktarılmıştır.

Veri ön işlemi sonrası dışa aktarılan veri seti daha sonra Apache Spark'ın MLlib kütüphanesinde bulunan müşterek filtreleme algoritması kullanılarak eğitilmiştir.

2.13. Verinin eğitilmesi ve test edilmesi

Veri ön işleme bölümünden elde edilen veriler ışığında kullanıcıların izleme sürelerine göre sayfaları aşağıdaki şekilde puanlandırıldığı hipotezi savunulmaktadır.

- 8 ile 13 saniye aralığında görüntülenme süresi için 1 puan
- 13 ile 25 saniye aralığında görüntülenme süresi için 2 puan

- 24 saniyeden büyük görüntülenme süresi için 3 puan.

Bu hipotezi ifade eden kod aşağıdaki gibi uygulanmıştır.

```
def decideRateFromDuration(duration: Int): Double = {
  duration match {
    case x if 9 until 12 contains x => 1
    case x if 13 until 24 contains x => 2
    case x if x > 24 => 3
    case _ => 0
  }
}
```

Veri setinin eğitilmesi için dışa aktarılan CSV dosyalarının RDD nesnelerine dönüştürülmesi gerekmektedir. Aşağıdaki metot kullanılarak dosyalar, RDD nesnesine dönüştürülmüştür.

```
def readRatingsFromFile(path: String): RDD[Rating] = {
  val data = sc.textFile(path)
  data.map(_.split(',') match {
    case Array(id, duration, deviceId, customerId, applicationId, contentId, page) =>
     Rating(deviceId.hashCode, (applicationId + "/" + contentId + "/" + page).hashCode, decideRateFromDuration(duration.toInt))
  }).filter( r => r.rating > 0.0)
}
```

Bu metotta her bir dosya satır satır okunmuş ve virgüllere göre ayrılmış, elde edilen dizi *Rating* nesnesine dönüştürülmüştür. *Rating* nesnesi MLlib kütüphanesinde bulunan müşterek filtreleme algoritması için gerekli olan kullanıcının belirli bir nesneyi derecelendirmesini ifade eder. Daha sonra kullanıcının sadece sıfırdan büyük olan derecelendirmeleri eğitim veri setine alınmıştır.

İlk olarak eğitim veri seti ve test veri seti aşağıdaki komut ile birbirlerinden rastgele olarak ayrılmıştır.

val Array(training, test) = ratings.randomSplit(Array(0.8, 0.2))

Eğitim veri seti aşağıdaki şekilde eğitilmiştir.

val model = ALS.train(ratings, 10 ,10, 0.01)

Elde edilen model test aşağıdaki şekilde test edilmiştir. İlk olarak test veri setinden derecelenme değeri kaldırılmıştır.

```
val usersPages = test.map { case Rating(user, page, rate) =>(user, page)}
```

Daha sonra elde edilen veri seti modelden geçirilip sonuçlarıyla kaydedilmiştir.

```
val predictions =
 model.predict(usersPages).map { case Rating(user, page, rate) =>
  ((user, page), rate)
 }
```

Bir sonraki adımda tahmin değeri ile gerçek değer *Mean Square Error* yöntemi kullanarak değerlendirilmiştir.

```
val rate2Preds = test.map { case Rating(user, page, rate) =>
 ((user, page), rate)
}.join(predictions)
val MSE = rate2Preds.map { case ((user, page), (p1, p2)) =>
 val error = (p1 - p2)
 error * error
}.mean()
println("Mean Squared Error = " + MSE)
```

Hata oranı test verisi her defasında rastgele seçildiğinden 0.7 ile 0.8 arasında değişmektedir.

2.14. İçeriğe göre filtreleme

Müşterek filtreleme tavsiye sistemleri arasında başarımı yüksek bir yöntem olsa da, kullanıcı sayısının az olduğu durumlarda kullanıcılara tavsiye verme konusunda yetersiz kalmaktadır. Diğer kullanıcıların tercihleri henüz bilinmediğinden tavsiyelerin geçerli olma olasılığı düşmektedir. Çalışmada hibrit tavsiye sistemi kullanma sebebi budur.

İçeriğe göre filtrelemede kullanıcı vektörleri en uzun görüntülediği 5 doküman olarak belirlenmiştir. Nesne vektörleri ise Apache Solr'da bulunan TF-IDF vektörleridir.

2.14.1. Apache Solr kurulumu

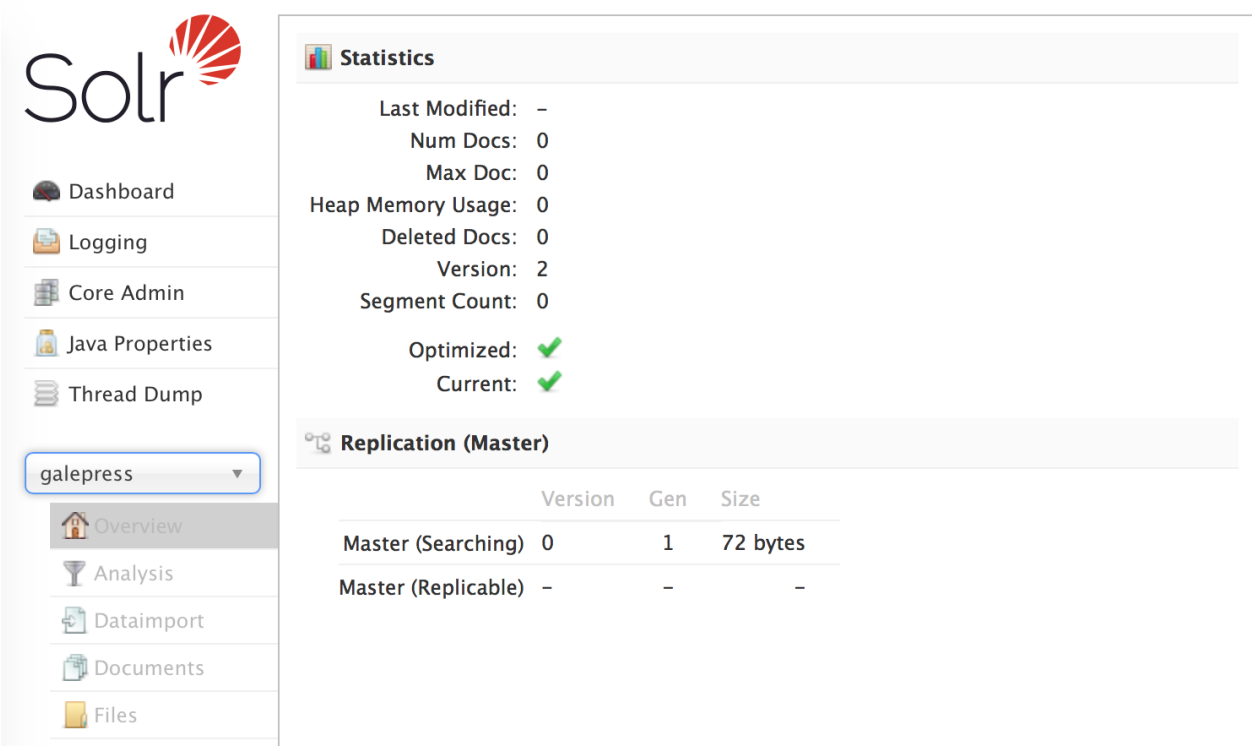
docker run --name galepress_solr -d -p 8983:8983 -t solr

Apache Solr da kullanıcıları ilgili sayfalara yönlendirebilmek amacıyla her bir dokümanın, her bir sayfası yeni bir doküman gibi indekslendi. Bunun yapılmasının nedeni kullanıcının belirli sayfadaki ürüne bakarken ilgisini çekebilecek diğer ürünlere bakmasını sağlamak. Ayrıca kullanıcılar bu indeksleme sayesinde doküman içerisinde istediği içeriğe hızlı bir şekilde ulaşabilmektedir.

Apache Solr şeması, konteynırın içinde bulunan kabuk üzerinden aşağıdaki komut ile oluşturuldu

docker exec galepress_solr bash bin/solr create -c galepress

Oluşturulan çekirdek artık Apache Solr web arayüzü üzerinden görülebilir

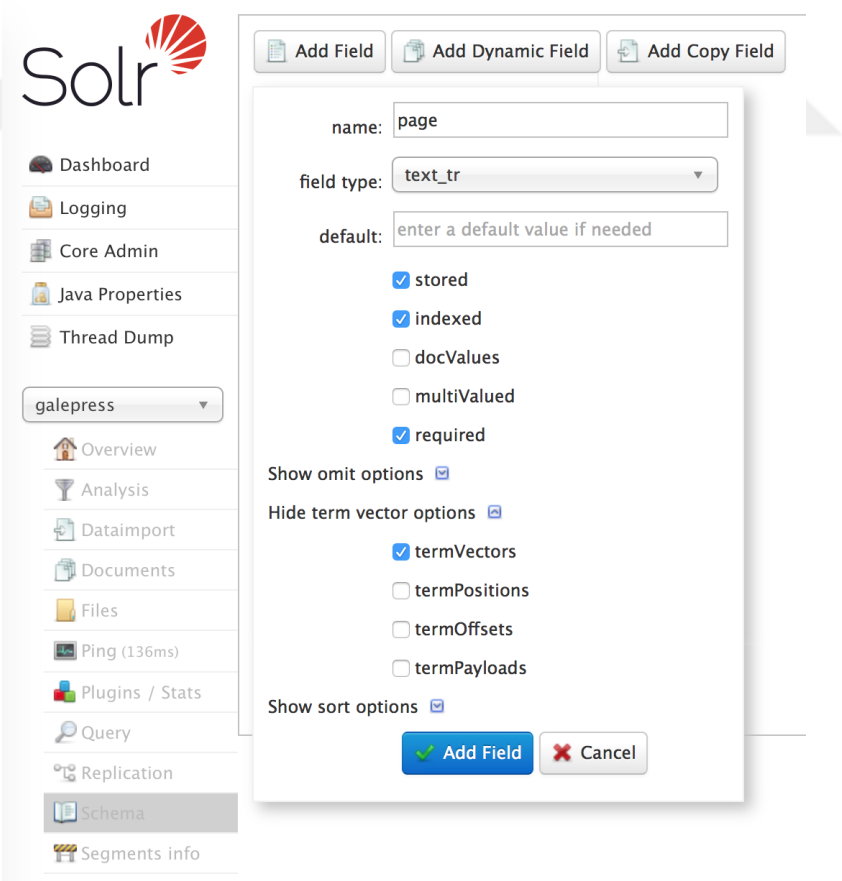


Şekil 2.11. Apache Solr web arayüzü

Apache Solr'da metin bazlı işlemler yapıldığında şemayı indekslemeden önce oluşturmak gerekmektedir. Şemayı sonradan oluşturmak bazı durumlarda indeksleme işleminin tekrarlanmasını gerektirebilir. Oluşturacağımız şemanın iki alanı bulunmaktadır.

1. Sayfayı temsil eden anahtar değeri. Bu değer sayfanın <uygulama_numarası>/<doküman_numarası>/<sayfa_numarası> şeklinde oluşturulmuştur.
2. Sayfa içeriği

Apache Solr'da oluşturduğumuz çekirdek ile birlikte şemanın anahtar değeri de otomatik olarak tanımlanmaktadır. Sayfa içeriği tanımlaması ise arayüz üzerinden yapılabilmektedir. Bunun için ilk olarak sol altta bulunan çoktan seçmeli butondan *galepress* çekirdeği seçildikten sonra aşağıda beliren yan menüden Schema seçeneği seçilir. Daha sonra *Add Field* butonuna basarak ilgili alanlar şekilde gösterildiği gibi doldurulur.



Şekil 2.12. Apache Solr alan oluşturma ekranı

Oluşturulan alanın özellikleri aşağıda açıklanmıştır.

- *Name* değeri ilgili alanın ismine işaret etmektedir.
- *Field type* değeri ilgili alanın hangi tipe sahip olduğunu belirler. Seçilen *text_tr* tipi sayfanın içeriğini Türkçe'ye uygun olarak ön işlemden geçirip analiz ettikten sonra indekslenmesini sağlar.
- İlgili alan, *Stored* olarak işaretliğinde Apache Solr dokümanın orjinal verini saklar böylelikle sonuçlar arasında dokümanın orijinal hali de çekilebilir.
- İlgili alan üzerinde arama ve analiz edilmesi isteniyorsa *Indexed* olarak işaretlenmelidir. İndeks olmadan arama yapılamaz.
- İlgili alan, *Required* olarak işaretlendiğinde Apache Solr doküman kaydedilirken ilgili alanın var olması gerektiğini anlar. Gerekli alanda değer olmazsa ise Apache Solr dokümanı kabul etmeyecektir. Bizim dokümanımızda zaten tek alan olduğundan bu değer olmadan bir doküman kaydedilmesinin bir ifadesi olmayacaktır. Bu nedenle bu alan gerekli olarak işaretlenmiştir.
- İlgili alanda *Term Vectors* alanı işaretlendiğinde Apache Solr dokümanı oluştururken terimlerinin bir vektörünü oluşturur. Böylelikle gerekli algoritmalar kullanılarak veri üzerinde analiz yapmak mümkün olur.

2.14.2. Buna Benzer Dokümanlar Özelliği

Tablo 2.2. More Like This parametreleri

Parametre	**Açıklama**	**Değer**
mlt.fl	Sorgunun çalıştırılacağı alan	page
mlt	*More Like This* özelliğini açmak için kullanılır	true
fl	Geri döndürülecek alan	id
q	Sorgu	Değer olarak en çok görüntülenen 5 dokümanı alır.

Buna benzer dokümanlar orijinal adıyla *More Like This* metodu *galepress* çekirdeğinin konfigürasyon dizininde bulunan solrconfig.xml dosyasına aşağıdaki etiket eklenerek aktifleştirilir.

<requestHandler name="/mlt" class="solr.MoreLikeThisHandler">

</requestHandler>

More Like This özelliğini kullanarak kullanıcının en çok görüntülediği 5 doküman üzerinden tavsiye verilmiştir. Tablo 2.2'de sorgu ayrıntıları verilmiştir.

2.15. Hibritleştirme Uygulaması

Hibritleştirme uygulaması Apache Spark ve Apache Solr'dan aldığı sonuçları birleştirip istemciye sunmak ile görevlidir.

İstemci, hibritleştirme uygulamasına kullanıcı anahtarı ile içerik anahtarı parametreleri ile sorgu yapar. Hibritleştirme uygulaması ilk olarak kullanıcının istatistiklerine bakarak kullanıcıyı skorlar. Bu skorlama istemci kullanıcı hareketlerinin, bütün kullanıcı hareketlerinin ortalamasına göre oranına bakarak gerçekleşir. Kullanıcı skoru maksimum 0.75 kabul edilir. Daha sonra hibritleştirme uygulaması müşterek filtrelemeye göre beş öneri ve içeriğe göre filtrelemeye göre beş öneriyi çeker. Hibritleştirme uygulaması verilen tavsiyelerin sırasını da dikkate almaktadır. Her bir sıradaki tavsiye için kullanıcı skoruna 1/tavsiye sırası x 0.25 kadar skor eklenir. Bu adımdan sonra sıfır ile bir arasında bir rastgele sayı oluşturulur. Eğer sayı sıfır ile verilen oran arasında ise kullanıcıya müşterek filtreleme sonuçları gönderilir. Eğer bu oranın dışında ise içeriğe göre filtreleme sonuçları gönderilir.

Yığın işlem uygulaması haftada bir kez çalışarak oluşan veri üzerinden Apache Spark modelini eğitir ve her bir kullanıcı için tavsiyeleri Apache Solr'a indeksler. Böylelikle belirli bir kullanıcı için olan müşterek filtreleme sonuçları ihtiyaç halinde hızlı bir şekilde ulaşılabilir.

SONUÇ

Tez çalışması kapsamında tavsiye sistemleri ve tavsiye sistemleri ile ilgili olan disiplinler ve tarihçeleri araştırılmış, en son çalışma yöntemleri incelenmiştir. Şu an kullanılmakta olan en popüler büyük veri mimarileri incelenmiş, büyük veri ve tavsiye sistemlerinde kullanılan teknolojiler araştırılmış, artı ve eksileri ortaya konmuştur.

Elde edilen veriler ışığında GalePress'in tavsiye sistemi mimarisi oluşturulmuştur. Ardından kullanılacak teknolojiler incelenmiş kurulumları gösterilmiştir. Daha sonra GalePress veri tabanı incelenmiş, veriler oluşturulan platforma aktarılmıştır. Veri ön analizden geçirildikten sonra ön işlemeden geçirilmiş ve eğitime hazır hale getirilmiştir. İçeriğe hazır hale getirilen verilen verilerin modeli Apache Spark MLlib ile eğitilerek test edilmiştir. İçeriğe göre filtreleme için Apache Solr kurulumu ve gerekli olan konfigürasyonların yapılması işlenmiştir. Daha sonra hibritleştirme uygulamasının ayrıntıları verilmiştir.

Yapılan çalışmalar doğrultusunda gerçek hayatta verilerin her zaman istenilen biçimde olmadığı ve verileri eğitimden geçirmeden önce ön analiz ve ön işlemeden geçirmenin makine öğrenmesi üretim hatlarının önemli bir parçası olduğu anlaşılmıştır. Yine aynı şekilde kullanıcılardan belirgin olmayarak (implicit) alınan verinin, belirgin olarak (explicit) toplanan veriye oranla işlemenin eğitmenin çok daha zor olduğu görülmüştür.

KAYNAKLAR

[1] "World Internet Users Statistics and 2018 World Population Stats." [Online]. Available: https://www.internetworldstats.com/stats.htm. [Accessed: 22-Feb-2018].

[2] **H. H. Wellisch,** "Ebla: The World's Oldest Library," *The Journal of Library History (1974-1987)*, vol. 16. University of Texas Press, pp. 488–500.

[3] **F. J. Witty,** "The Beginnings of Indexing and Abstracting: Some Notes towards a History of Indexing and Abstracting in Antiquity and the Middle Ages."

[4] **D. Stewart,** "THE STRUCTURE OF THE FIHRIST: IBN AL-NADIM AS HISTORIAN OF ISLAMIC LEGAL AND THEOLOGICAL SCHOOLS," *Int. J. Middle East Stud.*, vol. 39, no. 03, p. 369, 2007.

[5] **H. C. J. Godfray,** "Linnaeus in the information age," *Nature*, vol. 446, no. 7133, pp. 259–260, 2007.

[6] **W. B. Rayward,** "The origins of information science and the International Institute of Bibliography/International Federation for Information and Documentation (FID)," *J. Am. Soc. Inf. Sci.*, vol. 48, no. 4, pp. 557–573, 1997.

[7] **V. Bush,** "As We May Think," *Atl. Mon.*, no. July, pp. 112–124, 1945.

[8] **C. N. Mooers,** *The theory of digital handling of non-numerical information and its implications to machine economics*, no. 48. Zator Co., 1950.

[9] **C. N. Mooers,** *Making information retrieval pay*, no. 55. Zator Company, 1951.

[10] **G. SALTON ve C. S. YANG,** "ON THE SPECIFICATION OF TERM VALUES IN AUTOMATIC INDEXING," *J. Doc.*, vol. 29, no. 4, pp. 351–372, Apr. 1973.

[11] **G. Amati ve C. J. Van Rijsbergen,** "Probabilistic models of information retrieval based on measuring the divergence from randomness," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 357–389, 2002.

[12] **J. Kreps,** "Questioning the Lambda Architecture - O'Reilly Media," 2014. [Online]. Available: https://www.oreilly.com/ideas/questioning-the-lambda-architecture. [Accessed: 21-Oct-2017].

[13] **R. C. Fernandez** *et al.*, "Liquid: Unifying Nearline and Offline Big Data Integration," *Cidr*, 2015.

[14] **G. Adomavicius ve Y. Kwon,** "New Recommendation Techniques for Multicriteria

Rating Systems," *IEEE Intell. Syst.*, vol. 22, no. 3, pp. 48–55, 2007.

[15] **L. Terveen, L. Terveen, ve W. Hill,** "Beyond Recommender Systems: Helping People Help Each Other," *HCI NEW Millenn.*, pp. 487--509, 2001.

[16] **B. Sarwar, G. Karypis, J. Konstan, ve J. Reidl,** "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the tenth international conference on World Wide Web - WWW '01*, 2001, pp. 285–295.

[17] **Y. Koren, R. Bell, ve C. Volinsky,** "Matrix Factorization Techniques for Recommender Systems," *Computer (Long. Beach. Calif).*, vol. 42, no. 8, pp. 30–37, Aug. 2009.

[18] **R. M. Bell ve Y. Koren,** "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, 2007, pp. 43–52.

[19] **Y. Koren ve Yehuda,** "Factorization meets the neighborhood," in *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, 2008, p. 426.

[20] **Y. Hu, Y. Koren, ve C. Volinsky,** "Collaborative Filtering for Implicit Feedback Datasets," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 263–272.

[21] **M. J. Pazzani ve D. Billsus,** "Content-Based Recommendation Systems," in *The Adaptive Web*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 325–341.

[22] **A. Johnson,** "How MoreLikeThis Works in Lucene," *Blog*, 2008. [Online]. Available: http://cephas.net/blog/2008/03/30/how-morelikethis-works-in-lucene/. [Accessed: 22-Oct-2017].

[23] **G. Adomavicius ve A. Tuzhilin,** "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 734–749, 2005.

[24] "Oryx – Overview." [Online]. Available: http://oryx.io/. [Accessed: 08-Oct-2017].

[25] "Introduction to Seldon | Seldon Documentation." [Online]. Available: http://docs.seldon.io/. [Accessed: 08-Oct-2017].

[26] **I. Lunden,** "Salesforce Acquires PredictionIO To Build Up Its Machine Learning Muscle," 2016. [Online]. Available: https://techcrunch.com/2016/02/19/salesforce-acquires-predictionio-to-build-up-its-machine-learning-muscle/. [Accessed: 08-Oct-2017].

[27] **M. Odersky** *et al.*, "An Overview of the Scala Programming Language," *System*, no. Section 2, pp. 1–130, 2004.

[28] **M. Odersky, L. Spoon, ve B. Venners,** *Programming in Scala*, vol. 410, no. 2–3. 2008.

[29] **D. Merkel,** "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014.

[30] **M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, ve I. Stoica,** "Spark : Cluster Computing with Working Sets," *HotCloud'10 Proc. 2nd USENIX Conf. Hot Top. cloud Comput.*, p. 10, 2010.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Muhammet Hüseyin OLGUN

mh.olgun@gmail.com

İŞ TECRÜBESİ

hepsiburada.com 2018 – Devam

Yazılım Geliştirme Uzmanı

Bilgi getirimi ve navigasyon projesi

Kullanılan Teknolojiler: Elasticsearch, Apache Kafka, Python, .NET Core, Docker

ThoughtWorks 2017 – 2018

Yazılım Geliştirme Danışmanı

Turkcell Teknoloji 2016 – 2017

Yazılım Geliştirme Uzmanı

Turkcell TV+ projesi

Kullanılan teknolojiler: Java, Apache Traffic Control

Big Data Labs 2015 – 2016

Yazılım Geliştirme Uzmanı

e-nabız projesi, Tıbbi tavsiye sistemi

Kullanılan Teknolojiler: Scala, node.js, Couchbase

Detaysoft 2012 – 2015

Yazılım Geliştirme Uzmanı

Doküman Yönetim Sistemi, Tavsiye Sistemi

Kullanılan Teknolojiler: Java, Apache ManifoldCF, Hadoop, Spark, Mahout, Solr

EĞİTİM BİLGİLERİ

Lisans Fırat Üniversitesi Bilgisayar Mühendisliği