Büyük Veri Analizi Proje Raporu

Elif Özkan
Bilişim Sistemleri Mühendisliği
Kocaeli Üniversitesi
İzmit/Kocaeli
elifozkan@gmail.com

Andaç Akyüz
Bilişim Sistemleri Mühendisliği
Kocaeli Üniversitesi
İzmit/Kocaeli
akyuzandac@gmail.com

***Abstract*— Bu çalışma, büyük ölçekli veri setlerinde anomali tespitine odaklanmaktadır. Proje kapsamında makine öğrenimi ve derin öğrenme modelleri kullanılarak, Apache Spark ile entegre bir büyük veri işleme altyapısı geliştirilmiştir. Gerçek zamanlı veri akışı ve depolama işlemleri Apache Kafka ile sağlanmıştır. Karar Ağaçları ve Uzun Kısa Süreli Bellek (LSTM) ağları gibi ileri modelleme teknikleri kullanılarak, anomali tespitinde yüksek doğruluk elde edilmiştir. Gerçek zamanlı veri işleme ve dinamik veri yönetimi, sistemin ölçeklenebilir ve verimli bir şekilde çalışmasını sağlamıştır. IoT, finans, sağlık ve güvenlik gibi veri yoğun sektörlerde uygulanabilir bir çözüm sunan bu çalışma, model performansını RMSE, MAE ve R^2 gibi metriklerle değerlendirmiştir.**

Keywords*— *Büyük Veri, Anomali Tespiti, Apache Spark, Apache Kafka, Karar Ağacı, LSTM, Gerçek Zamanlı Analitik, Makine Öğrenimi.

I. PROJE AMACI

Bu proje kapsamında IntelliJ IDEA editörünü kullanarak Scala dilinde geliştirme yapıldı ve veri setimizde anomalileri tespit etmeyi amaçladık. Projenin temelinde, makine öğrenimi ve derin öğrenme algoritmalarını kullanarak bir model eğitimi gerçekleştirdik. Eğitilen model, büyük veri işleme altyapılarından biri olan Apache Spark aracılığıyla entegre edildi. Bu süreçte model, veri setindeki anomalileri algılayabilmek için eğitim aldı ve performansı optimize edildi.

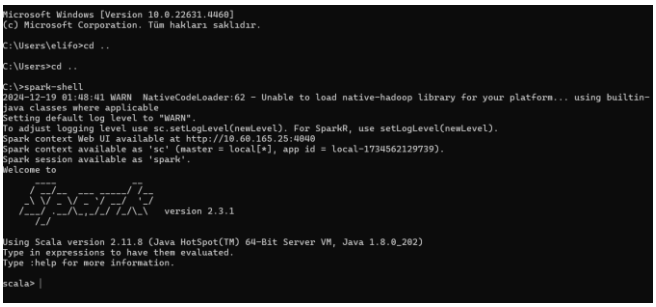
Projenin dinamik veri işleme boyutunda, Apache Kafka kullanılarak gerçek zamanlı veri üretimi gerçekleştirildi. Yeni üretilen veriler, Kafka üzerinde **topic** olarak organize edildi. Bu yapı, verilerin düzenli bir şekilde depolanmasını ve erişimini sağladı. Tespit edilen anomalili veriler ise yine Kafka topic'lerinde tutuldu, böylece bu verilere hızlı bir şekilde erişim sağlandı. Kafka'nın yüksek performanslı veri işleme yetenekleri sayesinde hem eğitim verileri hem de dinamik olarak üretilen yeni veriler üzerinde etkin bir işleme yapıldı.

Gerçek zamanlı veri işleme süreci, Spark Streaming teknolojisi kullanılarak hayata geçirildi. Kafka'dan alınan veriler, Spark Streaming tarafından işlenmek üzere eğitilmiş modele aktarıldı. Eğitilen model, gelen veri akışı üzerinde çalışarak anomalileri tespit etti. Bu işlemin ardından sonuçlar tekrar Kafka Topic yapılarına iletildi ve burada düzenli bir şekilde saklanarak analiz ve görselleştirme için hazır hale getirildi.

Projenin en önemli hedeflerinden biri, gerçek zamanlı çalışan bir büyük veri işleme sistemi kurmaktı. Bu doğrultuda, gerçek zamanlı olarak alınan veriler Spark tarafından işlenirken, sistemin yüksek performanslı ve ölçeklenebilir olması sağlandı. Kafka ile entegre edilen bu yapı, yeni verilerin dinamik bir şekilde işlenmesine ve anomali tespit sonuçlarının hızlı bir şekilde sunulmasına olanak tanıdı. Bu yapı, gerçek dünyadaki IoT, finans, sağlık ve güvenlik gibi veri yoğun sektörlerde uygulanabilir bir çözüm sunmaktadır.

II. GEREKLI KURULUMLAR

Bu yapıyı kurabilmek için öncelikle java 1.8 versiyonu kurulumu gerçekleştirdik ve bununla uyumlu olarak spark için 2.3.1 versiyonu scala için ise 2.11 versiyonunu kurduk. Eski versiyonları tercih etme sebebimiz ise projenin ilerleyen kısımlarında sorun yaşamamızdı. Kurulum yaparken öncelikle var olan java sürümü ile sorun yaşamamak için komple bilgisayarda bulunan java kaldırdık. Bizim için gerekli olan java sürümünü indirdikten sonra ortam değişkenlerinden java yolunu belirttikten sonra Spark kurulumuna geçtik. Bu kısımda ortam ve sistem değişkenlerini bununla birlikte SPARK_HOME yolunu belirttik ve projenin dilini yazmayı Scala ile planladığımız için Intellij IDEA indirip sonrasında Scala için eklenti kurduk Spark ve Scala yapısının düzgün çalışıp çalışmadığını kontrol etmek için terminale **spark-shell** komutunu yazdık ve aşağıdaki şekildeki gibi ekran görüntüsü aldık.



Şekil 2.1 (Spark ve Scala Kurulum)

Bu ekranın gelmesi kurulumu başarılı şekilde gerçekleştirdiğimizi gösterir. Proje geliştirirken sorun yaşamamak için gerekli kütüphane eklentilerini **built.sbt** dosyasına ilave ediyoruz. İlk başta eklediğimiz eklentiler de şu şekilde:

"org.apache.spark" %% "spark-core" % "2.3.1",

“org.apache.spark" %% "spark-sql" % "2.3.1",

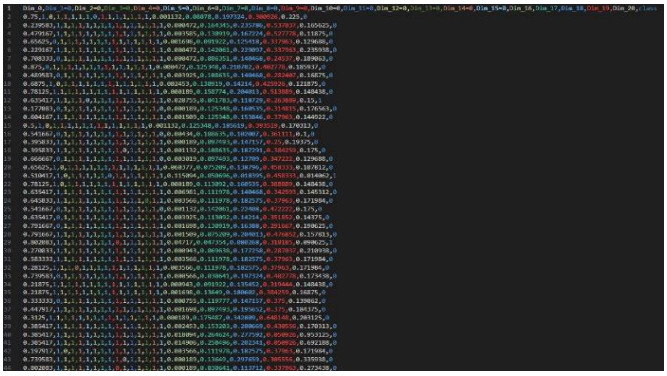
"org.apache.spark" %% "spark-mllib" % "2.3.1",

Buradaki bağımlılıklar sayesinde scala ile geliştirirken ilk kısım için gerekli olan bağımlılıkları yükleyip veri seti ile işlemlere geçtik.

III. VERİ SETİ İŞLEMLERİ

Projenin devamında anomaly tespiti yapma kapsamında seçtiğimiz veri seti “annthyroid_21feat_normalised.csv” veri setidir. Bu veri setini öncelikle model eğitimine dahil etmeden önce ön işleme aşamalarından geçiriyoruz. Bunların detaylarına inmek gerekirse ilk olarak veri setindeki tüm özellikler ve hedef değişkenleri yükledik daha sonra tahmin edilecek hedef değişken belirledik ve veriyi modelin anlayabileceği vektör haline dönüştürüp sonrasında

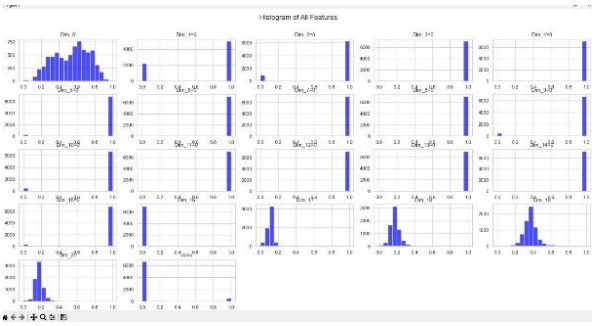
özellikleri ölçeklendirdik bu işlemler farklı özelliklerin aynı uzunlukta olmasını ve işlem yapılmasını kolaylaştırdı. Modelin işleme sorunsuz bir şekilde verileri alabilmesi için değişkenler sayısal değerlere dönüştürüldü. Belirlenen kategorik özellikler indekslenmesi sonucu ön işleme işlemini tamamlanmış oldu.



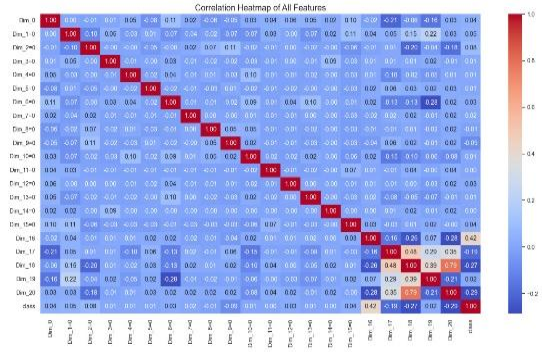
Şekil 3.1 (Veri Seti Görüntüsü)

IV. VERİ GÖRSELLEŞTİRME

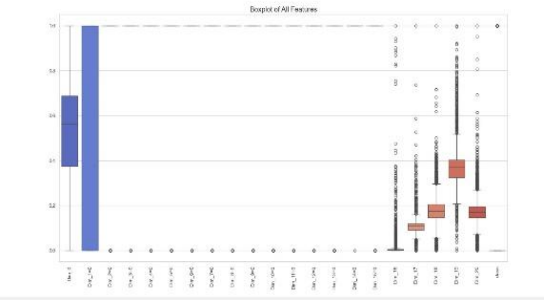
Projenin bu aşamasında ise elimizdeki veri setini Python kullanarak görselleştirdik. İlk aşamada Scala kullanarak yapmaya çalıştık ancak sonrasında Python kütüphaneleri ile daha verimli sonuçlar elde ettik.



Şekil 4.1 (Veri sütunlarının ağırlıklı histogramı)



Şekil 4.2 (Veri sütunlarının ısı haritası)



Şekil 4.3 (Veri sütunlarının boxplot grafiği)

V. MODELLEME SÜRECİ (KARAR AĞACI)

Ön işleme aşamasında modelin belli bir standarda getirdiğimiz verileri regresyon için hazırladık. Bu süreçte tahmin edilecek hedef değişeni seçtik. Modelin veriyi işleyebilmesi ve optimize şekilde çalışabilmesi için kategorik verilerin sayısal veri haline getirilmesi önemli bir adımdı. İlk aşamada verileri işleyip anomali tespiti yapabilmek için Karar Ağacı Modeli kullandık. Bu modeli kullanma sebebimiz ise elimizdeki veriye daha iyi uyum sağlayıp optimize bir sonuç vermesiydi. Karar ağacı regresyon algoritmasına önceden belirlediğimiz hedef değişkeni vererek ve modelin gerektirdiği tahmin yapılacak özellikleri de verdikten sonra veri dönüştürme ve modelleme aşamalarını Pipeline ile birleştirdik. Bu işlem modelleme Sürecini daha kolay ele almamızı sağladı. Bu işlem sonrasında ise modelin hiper parametrelerini grid yapısı ile denedik ve sonrasında değerlendirme sürecini başlattık. Modelin doğruluğunu arttırmak için Cross Validation uyguladık. Bu işlem modelin daha iyi performans göstermesini sağladı. Eğitim işlemi sonrası modelin performansını değerlendirmek için performans metriklerini hesapladık. Eğittiğimiz model önceden ayırdığımız test ve train verileri üzerinde tahmin yaptı. Modeli değerlendirmek için ise şu metrikleri hesapladık:

RMSE (Root Mean Squared Error):

- Tahmin hatalarının karesinin ortalamasının karekökü alınarak hesaplanır.
- Daha düşük bir RMSE, modelin tahminlerinin gerçek değerlere daha yakın olduğunu gösterir.

MSE (Mean Squared Error):

- Tahmin hatalarının karesinin ortalamasını hesaplar.
- RMSE'nin kare alınmamış versiyonudur.

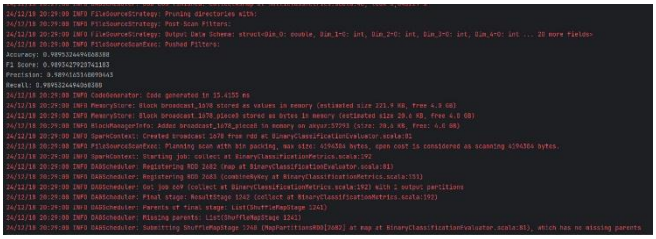
MAE (Mean Absolute Error):

- Tahmin hatalarının mutlak değerlerinin ortalamasını hesaplar.
- RMSE veya MSE'den daha az etkilenir aşırı uç hatalardan.

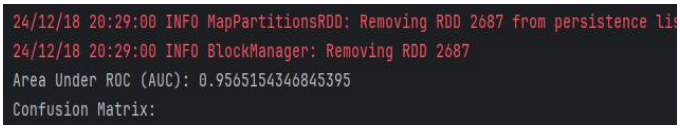
R^2 (R-Squared):

- Modelin açıklayabildiği toplam değişkenliğin yüzdesini ifade eder.

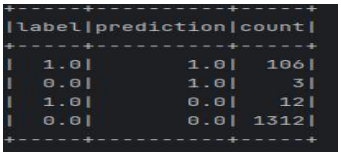
- 1.0'a ne kadar yakınsa, model o kadar iyi tahmin yapar.



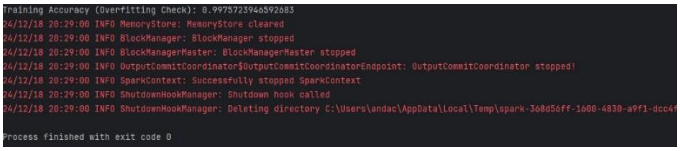
Şekil 5.1 (Model Doğruluk Parametreleri)



Şekil 5.2 (Model ROC Parametresi)



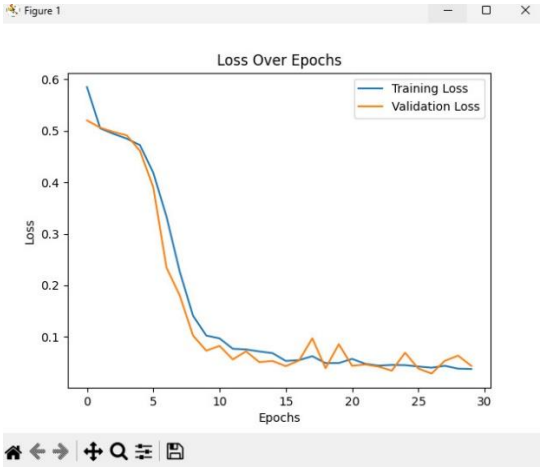
Şekil 5.3 (Confusion Matrix Sapma Sonuçları)



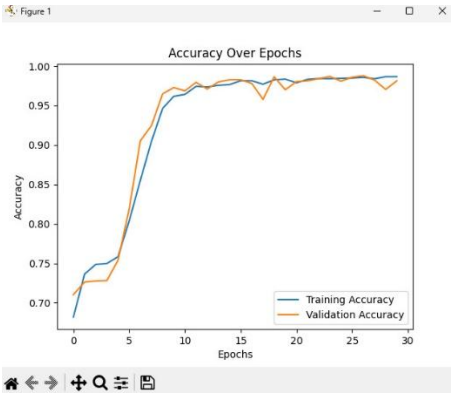
Şekil 5.4 (Model Training Acc Skoru)

LSTM MODELİ İLE EĞITIM

Projede hedeflediğimiz anomali tespitini daha performası yüksek sonuçlar elde edebilmek için LSTM modeli ile de eğitim yaptık. Burada model eğitimi sürecine Python ile başladık. Daha sonrasında ise modeli eğitirken dengesiz sınıf sorununu çözmek için SMOTE tekniği kullandık. Veri setini ayırırken özellikler ve hedef değişkenleri ayrıştırdık. Sonrasında dengesiz şekilde bulunan veri kümemizdeki sınıf sorununu çözmek için SMOTE tekniğini kullandık bu aşamadan sonra veri setini %80'e %20 şeklinde ayırdık. Verileri StandardScaler kullanılarak sıfır ortalama ve birim varyans olacak şekilde standardize ettik. LSTM modeli genel olarak üç boyutlu girdi formatı içerir (örnek sayısı, zaman adımı ve özellik sayısı) modelin eğitimini gerçekleştirdikten sonra eğitim kaybı ve doğruluğunda iyileşme olduğunu gördük. Eğitim ve doğrulama grafiklerini epoch bazında inceledik.



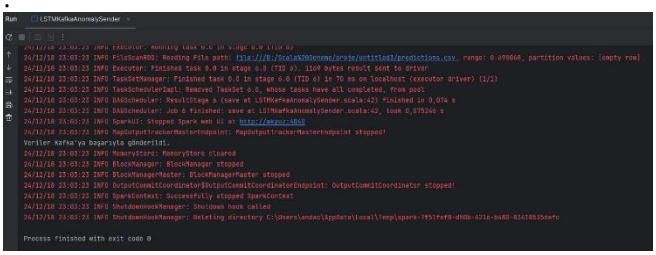
Şekil 5.1 LSTM Loss-Epoch Grafiği



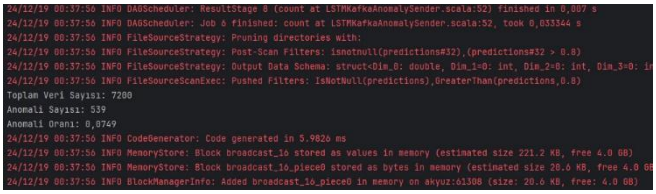
Şekil 5.2 LSTM Accuacy Grafiği

Sonraki aşamada ise eğittiğimiz modele tahmin yaptırıp sonuçları csv formatında kaydettik Verileri csv formatında istediğimiz şekilde scalada kullanamayacağımız için csv dosyalarını işledik. Scala tarafına tekrar döndüğümüzde ise python betiğini çalıştırabilmek için **scala.sys.process._** import ettik. Kullandığımız model ile scala ortamında modelin tahmin yapmasını sağladık Anomalili veriler ile normal verileri ayırıp anomali oranını hesapladıktan sonra bunları json formatına çevirip Kafka'ya gönderdik. Sürecin detaylarına inecek olursak

.



Şekil 5.3 LSTM Model Veriler Kafka'ya gönderildi

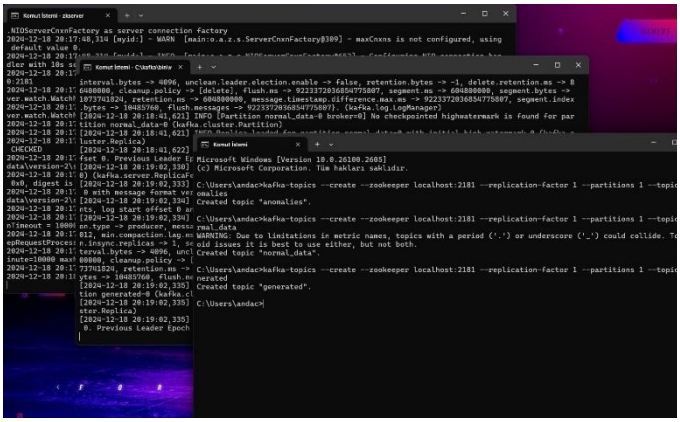


Şekil 5.4 Anomali sayısı ve oranı

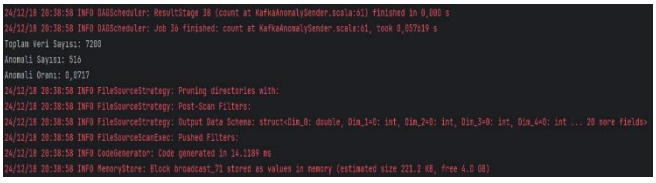
Verileri Kafka'ya başarılı bir şekilde gönderebilmek için Kafka ayarlarını sağladık. Kafka üzerinde topic oluşturduktan sonra

VI. KAFKA İLE ANOMALİ TESPİTİ

Kafka ile anomali tespiti yapabilmek için verileri uygun hale getirip özellikleri standart hale getirip Kafka için anomali ve normal olan verilere topic sütunu ilave ettik. Daha önceden eğittiğimiz modelin tahmin yaptı ve verileri gönderilmek üzere ayarlanmış oldu. Kafka yapısının dinleyeceği portu ayarladık. Topic oluşturma işlemini terminalden gerçekleştirdik. Anormali oranını hesaplayıp verileri Kafka'ya gönderdik. Bunu yapmadan önce de zookeeper yapısını başlattık.



Şekil-6.1 (Zookeeper ve Kafka çalışırken kullanacağımız Kafka Topic'lerinin oluşturulma aşaması)

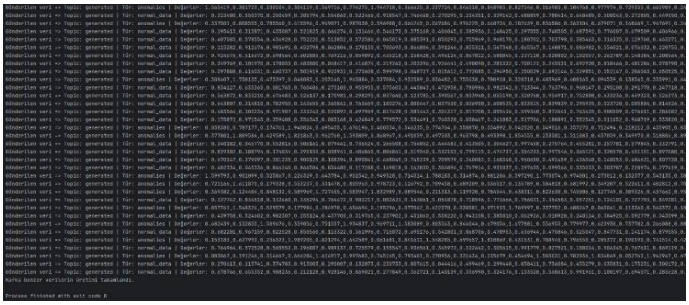


Şekil 6.2 (Verilerin Kafka'ya gönderilmeden önce anomali ve normal veri olarak oranlanmasının sonuçları)

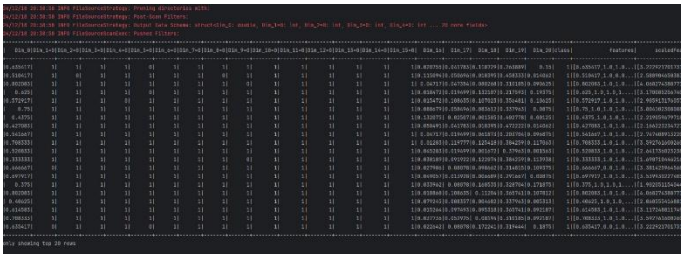
Şekil 6.3 (Verilerin Kafka'ya gönderilme aşaması)

VII. KAFKA ILE PRODUCER YAPISI

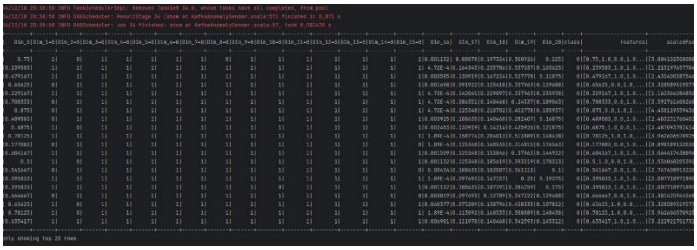
Projenin bundan sonraki aşamasında Kafka içine gönderdiğimiz verilerin ardından Kafka içinde rastgele veri üretmek ve mesaj göndermek amacıyla localhost ayarladık bununla birlikte mesaj gönderebilmek producer nesnesi de oluşturduk. Üreticinin mesaj anahtarı ve değeri String türündedir. Anomalies ve normal_data topicleri okunur ve bu topiclerdeki verilere benzer veriler %20 İhtimalle anomali %80 ihtimalle normal veri olmak üzere üretilir ve generated topic'ine gönderilir. Rastgele veri üretebilmek için Scala'nın Random sınıfı kullanarak Rastgele bir Boolean değer (true veya false) üretilir. Bu değere bağlı olarak mesajın gönderileceği Kafka başlığı belirlenir (anomalies veya normal_data).



Şekil 7.1 (Kafka Producer tarafından üretilen ve generated sınıfına gönderilen tüm verileri)



Şekil 7.2 (Kafka Producer tarafından üretilen anomalies sınıfı verileri)



Şekil 7.3 (Kafka Producer tarafından üretilen normal_data sınıfı verileri)

VIII. KAFKA ÜZERINDEN GELEN VERILERI OKUMA

Kafka üzerinden gelen veriler Spark kullanılarak işlenmiş ve analiz edilmiştir. Kafka ile Spark Streaming entegrasyonu sağlanarak, Kafka'daki generated topic'i KafkaUtils.createDirectStream metodu ile dinlenmiş ve gelen mesajlar Spark Streaming tarafından RDD formatında alınmıştır. Alınan veriler, analiz ve işlem kolaylığı için DataFrame yapısına dönüştürülmüştür.

Dönüştürülen veriler geçici olarak temp_output klasörüne .csv formatında kaydedilmiş, ardından bu dosyalar kafka_output/generated_data.csv adlı kalıcı dosyaya taşınmıştır. Geçici dosyalar, Scala'nın FileSystem API'si ile yönetilmiş ve gereksiz dosyalar temizlenmiştir.

Verilerin işlenmesi için Python betiği kullanılmış ve Scala üzerinden csv_processing.py adlı betik çağrılarak veriler düzenlenmiştir. Scala'nın scala.sys.process._ kütüphanesi ile Python betiği çalıştırılmış, sütunlar yeniden düzenlenerek analiz edilebilir hale getirilmiştir.

LSTM modeli ile tahmin işlemleri gerçekleştirilmiştir. Bu model, Python üzerinde çalıştırılarak tahmin edilen sonuçlar predictions.csv dosyasına kaydedilmiştir. Veriler Spark

DataFrame'e yüklenmiş ve tahmin değerlerine göre predictions > 0.8 olanlar anomali, predictions <= 0.2 olanlar normal veri olarak sınıflandırılmıştır. Bu sınıflandırma sonrasında anomali oranı hesaplanmıştır.

Sınıflandırılan veriler, JSON formatına dönüştürülerek Kafka'ya geri gönderilmiştir. Spark'ın Kafka yazma özelliği

kullanılarak, veriler anomalies ve normal_data topic'lerine aktarılmıştır.

REFERENCES

[1] P. Viola ve M. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, cilt 57, no. 2, s. 137–154, 2004.

[2] T. Baltrusaitis, A. Zadeh, Y

[3] https://www.veribilimiokulu.com/windows-10-uzerine-kafka-kurmak/

[4] https://tr.d2l.ai/chapter_recurrent-modern/lstm.html

[5] https://spark.apache.org/docs/3.5.1/structured-streaming-kafka-integration.html

[6] https://github.com/veribilimiokulu

[7] https://erdincuzun.com/makine_ogrenmesi/decision-tree- karar-agaci-id3-algoritmasi-classification-siniflama/