

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KABA KÜMELEME TEORİSİ KULLANILARAK DİYABET VERİ SETİNİN
ANALİZ EDİLMESİ VE DİYABET HASTALIĞININ TEŞHİSİ İÇİN GERÇEK
ZAMANLI BİR VERİ ANALİTİĞİ SİSTEMİ KURULMASI

YÜKSEK LİSANS TEZİ

Mehmet YÜKSEL

Bilişim Sistemleri Mühendisliği Anabilim Dalı

KASIM 2025

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KABA KÜMELEME TEORİSİ KULLANILARAK DİYABET VERİ SETİNİN
ANALİZ EDİLMESİ VE DİYABET HASTALIĞININ TEŞHİSİ İÇİN GERÇEK
ZAMANLI BİR VERİ ANALİTİĞİ SİSTEMİ KURULMASI

YÜKSEK LİSANS TEZİ

Mehmet YÜKSEL

Bilişim Sistemleri Mühendisliği Anabilim Dalı

Tez Danışmanı: Dr. Öğr. Üyesi Hüseyin DEMİRCİ

KASIM 2025

Mehmet Yüksel tarafından hazırlanan “Kaba Kümeleme Teorisi Kullanılarak Diyabet Veri Setinin Analiz Edilmesi ve Diyabet Hastalığının Teşhisi İçin Gerçek Zamanlı Bir Veri Analitiği Sistemi Kurulması” adlı tez çalışması 21.11.2025 tarihinde aşağıdaki jüri tarafından oy birliği ile Sakarya Üniversitesi Fen Bilimleri Enstitüsü Bilişim Sistemleri Mühendisliği Anabilim Dalı Yüksek Lisans tezi olarak kabul edilmiştir.

Tez Jürisi

Jüri Başkanı :

Jüri Üyesi :

Jüri Üyesi :



ETİK İLKE VE KURALLARA UYGUNLUK BEYANNAMESİ

Sakarya Üniversitesi Fen Bilimleri Enstitüsü Lisansüstü Eğitim-Öğretim Yönetmeliğine ve Yükseköğretim Kurumları Bilimsel Araştırma ve Yayın Etiği Yönergesine uygun olarak hazırlamış olduğum “Kaba Kümeleme Teorisi Kullanılarak Diyabet Veri Setinin Analiz Edilmesi ve Diyabet Hastalığının Teşhisi İçin Gerçek Zamanlı Bir Veri Analitiği Sistemi Kurulması” başlıklı tezin bana ait, özgün bir çalışma olduğunu; çalışmamın tüm aşamalarında yukarıda belirtilen yönetmelik ve yönergeye uygun davrandığımı, tezin içerdiği yenilik ve sonuçları başka bir yerden almadığımı, tezde kullandığım eserleri usulüne göre kaynak olarak gösterdiğimi, bu tezi başka bir bilim kuruluna akademik amaç ve unvan almak amacıyla vermediğimi ve 20.04.2016 tarihli Resmi Gazete’de yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince Sakarya Üniversitesi’nin abonesi olduğu intihal yazılım programı kullanılarak Enstitü tarafından belirlenmiş ölçütlere uygun rapor alındığını, çalışmamla ilgili yaptığım bu beyana aykırı bir durumun ortaya çıkması halinde doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi beyan ederim.

(21/11/2025).

(imza)

Mehmet YÜKSEL





Beni her aşamada destekleyen, yol gösteren tüm hocalarıma



TEŞEKKÜR

Yüksek lisans öğrenimim süresince bilgi birikimi, yol göstericiliği ve desteğiyle her zaman yanımda olan, çalışmamın tüm aşamalarında değerli katkılarını sunan, yönlendirici yaklaşımı ve sabrıyla bana ilham veren danışmanım Dr. Öğr. Üyesi Hüseyin DEMİRCİ'ye en içten teşekkürlerimi sunuyorum.

Ayrıca, bu süreçte manevi desteğini her zaman hissettiğim, sabırları ve sevgileriyle bana güç veren aileme de minnettarım.

Mehmet YÜKSEL

İÇİNDEKİLER

Sayfa

ETİK İLKE VE KURALLARA UYGUNLUK BEYANNAMESİ	v
TEŞEKKÜR.....	ix
İÇİNDEKİLER	xi
KISALTMALAR.....	xiii
SİMGELER	xv
TABLO LİSTESİ	xvii
ŞEKİL LİSTESİ.....	xix
ÖZET.....	xxiii
SUMMARY	xxvii
1. GİRİŞ.....	1
1.1. Çalışmanın Amacı	2
1.2. Çalışmanın Kapsamı ve Önemi.....	3
1.3. Tezin Organizasyonu.....	6
2. LİTERATÜR TARAMASI	9
2.1. Diyabet Hastalığı ve Risk Faktörleri	9
2.2. Sağlık Bilişiminde Veri Analitiği.....	10
2.3. Kaba Kümeleme Teorisi (Rough Set Theory) Üzerine Çalışmalar	12
2.4. Gerçek Zamanlı Veri İşleme Araçları Üzerine Çalışmalar	13
2.5. Makine Öğrenmesi Yöntemleriyle Diyabet Teşhisi.....	14
3. VERİ SETİ ÖN İŞLEME SÜRECİ.....	17
3.1. Diyabet Hastalığı ve Risk Faktörleri	17
3.2. Eksik ve Aykırı Değerlerin Analizi.....	18
3.3. SMOTE ile Sınıf Dengesinin Sağlanması	24
3.4. CTGAN ile Sentetik Veri Üretimi	27
3.5. Veri Setinin Eğitim ve Test Olarak Bölünmesi.....	31
3.6. Kategorik Dönüşüm Süreci	32
3.6.1. Pregnancies sütunu için kategorikleştirme süreci	33
3.6.2. Glucose sütunu için kategorikleştirme süreci	35
4. KABA KÜMELEME TEORİSİ VE KARAR KURALLARI ÜRETİMİ.....	37
4.1. Kaba Kümeleme Teorisine Giriş.....	37
4.2. Öznitelik İndirgeme Süreci	40
4.3. Karar Kurallarının Elde Edilmesi.....	46
5. GERÇEK ZAMANLI VERİ ANALİTİĞİ MİMARİSİ	49
5.1. Gerçek Zamanlı Mimari Bileşenleri.....	50
5.1.1. Apache kafka.....	50
5.1.2. Apache zookeeper	51
5.1.3. Gerçek zamanlı veri akışı ve tahmin sonuçlarının yönetimi	51
5.2. Java ile Streaming Producer ve Consumer Yapısı	52
5.2.1. Producer bileşeni.....	54
5.2.2. Consumer bileşeni.....	56
5.2.3. Sistem entegrasyonu ve gerçek zamanlılık	60

5.2.4. Güvenilirlik ve yüksek performans	61
5.2.5. Gerçek zamanlı veri akış süreci	61
5.3. Makine Öğrenmesi Algoritmalarının Java ile Uygulanması	64
5.3.1. Naive bayes	66
5.3.2. KNN ve KNC	68
5.3.3. Logistic regression	73
5.3.4. AdaBoost.....	76
5.3.5. Ensemble algoritmalar.....	79
5.3.5.1. Ensemble ağırlıklı oylama algoritması.....	79
5.3.5.2. Ensemble çoğunluk oylaması algoritması.....	80
5.4. Tahmin Sonuçlarının Kayıt ve Görselleştirilme Süreci.....	81
5.5. MSSQL Veritabanı Entegrasyonu	83
6. SONUÇLAR VE DEĞERLENDİRME.....	87
6.1. Ön İşleme Sürecinin Etkileri	87
6.2. Kaba Kümeleme Sonuçlarının Analizi	88
6.3. Gerçek Zamanlı Tahminleme Performansı	89
7. GENEL SONUÇLAR VE ÖNERİLER.....	93
7.1. Çalışmanın Katkıları.....	93
7.2. Gelecek Çalışmalar için Öneriler	95
KAYNAKLAR.....	97
ÖZGEÇMİŞ.....	101

KISALTMALAR

AI	: Artificial Intelligence (Yapay Zekâ)
API	: Application Programming Interface
CSV	: Comma-Separated Values (Virgülle Ayrılmış Değerler)
CTGAN	: Conditional Tabular Generative Adversarial Network
FIFO	: First In First Out (İlk Giren İlk Çıkar)
KNN	: K-Nearest Neighbors (K-En Yakın Komşu)
KNC	: K-Nearest Centroid (K-En Yakın Merkez)
LR	: Logistic Regression (Lojistik Regresyon)
ML	: Machine Learning (Makine Öğrenmesi)
MSSQL	: Microsoft Structured Query Language Server
NB	: Naive Bayes
RST	: Rough Set Theory (Kaba Kümeleme Teorisi)
SMOTE	: Synthetic Minority Over-sampling Technique
SQL	: Structured Query Language
UI	: User Interface (Kullanıcı Arayüzü)
ZK	: Zookeeper (Apache Zookeeper)



SİMGELER

k	: K-En Yakın Komşu (KNN) algoritmasında komşu sayısı [tamsayı]
%	: Yüzdelik değer (doğruluk oranı, başarı ölçümü vb.)
t	: Süre/zaman (işlem süresi, saniye olarak) [s]
n	: Toplam örnek sayısı [adet]
w	: Model katsayısı/ağırlığı (Logistic Regression’da)
Q1	: Birinci çeyrek değer (veri dağılımında)
Q3	: Üçüncü çeyrek değer (veri dağılımında)
IQR	: Interquartile Range – çeyrekler arası açıklık
mean	: Ortalama değer
std	: Standart sapma



TABLO LİSTESİ

Sayfa

Tablo 3.1. Pima Indians Veri Setindeki İlk Beş Kayıt.	17
Tablo 3.2. Pima Indians Veri Seti Değişken Analizi.....	18
Tablo 3.3. Eksik ve Aykırı Değer Analizi Öncesi Veri Setinin İstatistiksel Analizi.	19
Tablo 3.4. Eksik ve Aykırı Değer Analizi Sonucu Oluşan Veri setinin İstatistiksel Analizi.	22
Tablo 5.1. Hastalık Sonuçlarının Tutulduğu Tablo.	85
Tablo 5.2. Hasta Bilgileri Tablosu.	85
Tablo 6.1. Gerçek Zamanlı Tahminleme Mimarisinde Kullanılan Algoritmaların Performans Sonuçları.	91



ŞEKİL LİSTESİ

Sayfa

Şekil 3.1. Veri ön işleme öncesi sıfır değerlerinin dağılımı.	19
Şekil 3.2. Eksik ve Aykırı Değer Analizi öncesi veri setinin histogram grafiği.....	20
Şekil 3.3. Eksik ve Aykırı Değer Analizi öncesi veri setinin korelasyon matrisi.....	20
Şekil 3.4. IQR Yaklaşımı; Alt sınır ve Üst sınır formülü.	21
Şekil 3.5. Veri ön işleme sonucu sıfır değerlerinin dağılımı.	21
Şekil 3.6. Uç değer temizliği python kodu.	22
Şekil 3.7. Eksik ve Aykırı Değer Analizi sonrası oluşan veri setinin histogram grafiği.	23
Şekil 3.8. Eksik ve Aykırı Değer Analizi sonrası oluşan veri setinin korelasyon matrisi.	23
Şekil 3.9. SMOTE öncesi Diyabet durum dağılımı.	25
Şekil 3.10. SMOTE algoritması python kodu.....	26
Şekil 3.11. SMOTE sonrası Diyabet durum dağılımı.	26
Şekil 3.12. CTGAN modeli uygulama python kodu.....	28
Şekil 3.13. CTGAN sonucu oluşan veri setinin analizini yapan python kodu.	29
Şekil 3.14. CTGAN sonrası oluşan veri setinin histogram grafiği.	29
Şekil 3.15. CTGAN sonrası oluşan veri setinin korelasyon matrisi.	30
Şekil 3.16. CTGAN sonrası oluşan veri setinin diyabet dağılım grafiği python kodu.	30
Şekil 3.17. CTGAN sonrası oluşan veri setinin diyabet dağılımı.....	31
Şekil 3.18. Pregnancies sütununa ait temel istatistiksel bilgiler ve python kodu.	33
Şekil 3.19. Pregnancies sütunu histogram grafiği ve python kodu.....	33
Şekil 3.20. Pregnancies sütunu kategorik yapı ve python kodu.	34
Şekil 3.21. Pregnancies sütunu kategorik dağılım ve python kodu.	34
Şekil 3.22. Glucose sütunu histogram grafiği ve python kodu.	35
Şekil 3.23. Glucose sütunu kategorik yapı ve python kodu.....	35
Şekil 3.24. Glucose sütunu kategorik dağılım ve python kodu.	36
Şekil 3.25. Kategorik veri setine ait ilk beş kayıt.	36
Şekil 4.1. Alt küme yaklaşımı, Üst küme yaklaşımı ve Sınır Bölgesi ifade biçimi... 37	
Şekil 4.2. Kaba kümeleme yaklaşımları [12].....	38
Şekil 4.3. Rosetta yazılımına ait bir ekran görüntüsü.	38
Şekil 4.4. Veri setine ait alt küme yaklaşımı, üst küme yaklaşımı, küme türü ve doğruluk değeri analizi.....	39
Şekil 4.5. Veri setinin hiçbir öznitelik çıkartılmadığında elde edilen ayrılmazlık ilişisine ait ekran görüntüsü.	40
Şekil 4.6. Veri setinden Pregnancies özniteliği çıkartıldığında elde edilen ayrılmazlık ilişisine ait ekran görüntüsü.	41
Şekil 4.7. Veri setinden Glucose özniteliği çıkartıldığında elde edilen ayrılmazlık ilişisine ait ekran görüntüsü.	41
Şekil 4.8. Veri setinden BloodPressure özniteliği çıkartıldığında elde edilen ayrılmazlık ilişisine ait ekran görüntüsü.	42

Şekil 4.9. Veri setinden SkinThickness özniteliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.	42
Şekil 4.10. Veri setinden Insulin özniteliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.....	43
Şekil 4.11. Veri setinden BMI özniteliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.....	43
Şekil 4.12. Veri setinden DiabetesPedigreeFunction özniteliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.....	44
Şekil 4.13. Veri setinden Age özniteliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.....	44
Şekil 4.14. Veri setinden çıkartılabilecek özniteliklere ait ekran görüntüsü.....	45
Şekil 4.15. Geliştirilen yazılım ile elde edilen karar kuralları örnek ekran görüntüsü.	47
Şekil 4.16. Rosetta yazılımı ile elde edilen karar kuralları örnek ekran görüntüsü. ..	48
Şekil 5.1. Apache Kafka yapısı [34].	50
Şekil 5.2. Apache Zookeeper ayağa kaldırma komutu.	51
Şekil 5.3. Apache Kafka ayağa kaldırma komutu.....	53
Şekil 5.4. Yeni bir topic oluşturma komutu.	53
Şekil 5.5. Mevcut topicleri listeleme komutu.	54
Şekil 5.6. Producer tarafı kod yapısı.	55
Şekil 5.7. Producer tarafı kod yapısı (Devamı-1).	55
Şekil 5.8. Producer tarafı kod yapısı (Devamı-2).	56
Şekil 5.9. Consumer tarafı kullanılan kütüphaneler.....	57
Şekil 5.10. Consumer tarafı kod yapısı.	58
Şekil 5.11. Consumer tarafı kod yapısı (Devamı-1).	58
Şekil 5.12. Consumer tarafı kod yapısı (Devamı-2).	59
Şekil 5.13. Consumer tarafı kod yapısı (Devamı-3, Tüm kodlar paylaşılmamıştır.).	59
Şekil 5.14. Kafka Consumer başlatıldı.....	61
Şekil 5.15. Kafka Producer başlatıldı.....	61
Şekil 5.16. Hastane Otomasyon Sistemi arayüzü.....	62
Şekil 5.17. Hastane Otomasyonuna örnek hasta tahlil verisi girişi.....	62
Şekil 5.18. Kafka Producer, girilen veriyi Consumer'a iletti.....	62
Şekil 5.19. Consumer veriyi analiz etti ve çıktıyı kaydetti.	63
Şekil 5.20. Tahmin sonucu arayüzde görüntülendi.....	63
Şekil 5.21. Tahmin sonucu veritabanına kaydedildi.	64
Şekil 5.22. Naive Bayes Algoritması uygulama kodu.	67
Şekil 5.23. Naive Bayes Algoritması uygulama kodu (Devamı-1).....	67
Şekil 5.24. Naive Bayes Algoritması uygulama kodu (Devamı-2).....	68
Şekil 5.25. Naive Bayes Algoritması uygulama kodu (Devamı-3).....	68
Şekil 5.26. KNN Algoritması uygulama kodu.	69
Şekil 5.27. KNN Algoritması uygulama kodu (Devamı-1).	70
Şekil 5.28. KNN Algoritması uygulama kodu (Devamı-2).	70
Şekil 5.29. KNC Algoritması uygulama kodu.	71
Şekil 5.30. KNC Algoritması uygulama kodu (Devamı-1).	72
Şekil 5.31. KNC Algoritması uygulama kodu (Devamı-2).	72
Şekil 5.32. Logistic Regression Algoritması uygulama kodu.....	74
Şekil 5.33. Logistic Regression Algoritması uygulama kodu (Devamı-1).....	75
Şekil 5.34. Logistic Regression Algoritması uygulama kodu (Devamı-2).....	75
Şekil 5.35. Logistic Regression Algoritması uygulama kodu (Devamı-3).....	76
Şekil 5.36. Logistic Regression Algoritması uygulama kodu (Devamı-4).....	76

Şekil 5.37. AdaBoost Algoritması uygulama kodu.	77
Şekil 5.38. AdaBoost Algoritması uygulama kodu (Devamı-1).....	78
Şekil 5.39. AdaBoost Algoritması uygulama kodu (Devamı-2).....	78
Şekil 5.40. Ensemble Ağırlık Algoritması uygulama kodu.	79
Şekil 5.41. Ensemble Çoğunluk Algoritması uygulama kodu.	80
Şekil 5.42. Algoritmaların doğruluk çıktıları.....	82
Şekil 5.43. Algoritmaların performans karşılaştırması.	82
Şekil 5.44. Algoritmaların test süreleri.	83





KABA KÜMELEME TEORİSİ KULLANILARAK DİYABET VERİ SETİNİN ANALİZ EDİLMESİ VE DİYABET HASTALIĞININ TEŞHİSİ İÇİN GERÇEK ZAMANLI BİR VERİ ANALİTİĞİ SİSTEMİ KURULMASI

ÖZET

Bu tez çalışması, diyabet hastalığının erken teşhisine yönelik olarak çağdaş veri bilimi yöntemleri, istatistiksel ön işleme teknikleri, makine öğrenmesi algoritmaları, sentetik veri üretim sistemleri, belirsizlik tabanlı matematiksel analiz yaklaşımları ve gerçek zamanlı veri akış teknolojilerinin bir araya getirilmesiyle oluşturulan bütüncül bir karar destek sistemi geliştirilmesini amaçlamaktadır. Diyabet, küresel ölçekte hem yaygınlığı hem de yol açtığı uzun vadeli sağlık komplikasyonları nedeniyle günümüz tıbbının en kritik kronik hastalıklarından biridir. Hastalığın erken dönemde yüksek doğrulukla tespit edilmesi, ilerleyen dönemlerde ortaya çıkabilecek kardiyovasküler, nörolojik ve metabolik komplikasyonların önlenmesi açısından büyük önem taşımaktadır. Bu nedenle modern tıpta veri temelli karar destek mekanizmaları gün geçtikçe daha önemli hâle gelmektedir. Bu tez, yalnızca bir makine öğrenmesi modeli oluşturmakla kalmayıp, verinin toplanma anından analiz edilmesine ve sonuçların kullanıcıya gerçek zamanlı olarak sunulmasına kadar uzanan tüm süreci kapsayan çok katmanlı bir mimari ortaya koymaktadır. Böylece klasik sınıflandırma yaklaşımlarının ötesinde, gerçek zamanlı ve klinik olarak uygulanabilir bir çözüm hedeflenmiştir.

Tez kapsamında kullanılan Pima Indians Diabetes veri seti, tıbbi veri setlerinde yaygın olarak görülen eksik veri, dengesiz sınıf yapısı ve ölçüm hataları gibi temel problemleri içerdiği için çalışmanın başlangıç noktası olmuştur. Veri setinde bazı fizyolojik ölçümlerin sıfır değer içerdiği tespit edilmiş; glikoz, insülin veya cilt kalınlığı gibi değerlerin sıfır olmasının biyolojik olarak mümkün olmaması nedeniyle bu veriler eksik veya hatalı veri olarak değerlendirilmiştir. Bu nedenle sistematik bir veri ön işleme süreci yürütülmüştür, aykırı değerlerin belirlenmesi için IQR (Interquartile Range) yöntemi kullanılmış ve dağılımı bozan uç değerler gerektiği şekilde düzeltilmiştir. Bu işlemler sonucunda veri setinin istatistiksel bütünlüğü artırılmış ve makine öğrenmesi modellerinin daha doğru bir temelde eğitilmesi mümkün hâle gelmiştir.

Veri setinin önemli problemlerinden biri olan sınıf dengesizliği ise SMOTE (Synthetic Minority Over-sampling Technique) algoritması ile giderilmiştir. Pozitif sınıfın düşük temsili, modellerin negatif sınıfa eğilim göstermesine neden olduğundan SMOTE aracılığıyla azınlık sınıfı veri uzayı içinde artırılmış ve dengeli bir eğitim kümesi oluşturulmuştur. Ayrıca veri setinin genişletilmesi amacıyla CTGAN (Conditional Tabular GAN) yöntemi uygulanmış ve veri seti 2000 gözleme çıkarılmıştır. CTGAN'ın karmaşık dağılımları öğrenebilme kapasitesi sayesinde sentetik verilerin özgün veri setiyle tutarlı bir yapıda üretildiği görülmüş, bu durum modelin genelleme becerisini artıran önemli bir faktör olmuştur. Bu aşamalar, çalışmanın sonraki bölümleri için güçlü bir veri tabanı oluşturmuştur.

Veri ön işleme sürecinin başarıyla tamamlanmasının ardından, tez çalışmasının ikinci önemli bileşenini oluşturan belirsizlik analizi aşamasına geçilmiştir. Bu kapsamda,

Kaba Kümeleme Teorisi (Rough Set Theory – RST) kullanılarak veri kümesi üzerinde açıklanabilirlik temelli bir bilgi çıkarımı yapılmıştır. RST, özellikle kategorik veriler üzerinde çalışmaya uygun olan ve belirsizlik içeren veri kümelerinde hem indirgeme işlemleri hem de karar kuralları üretme kapasitesi ile bilinen güçlü bir matematiksel yöntemdir. Bu nedenle veri setindeki sürekli değişkenler klinik açıdan anlamlı aralıklara bölünerek kategorik forma dönüştürülmüştür. Kategorik dönüşümün ardından C# dili ile özel olarak geliştirilen RST yazılımı kullanılmış; reduct, core, pozitif bölge ve karar kuralları hesaplanmıştır. Bu analizler sonucunda hiçbir özneliliğin tamamen gereksiz olmadığı, tüm değişkenlerin diyabet teşhisine belirli düzeyde katkı sağladığı görülmüştür. Reduct sonuçlarının bu biçimde çıkması, diyabetin çok boyutlu yapısını ve tek bir parametre üzerinden açıklanamayacak kadar karmaşık bir hastalık olduğunu ortaya koymaktadır. Ayrıca RST'nin oluşturduğu “Eğer ... ise ... olur” yapısındaki karar kuralları, makine öğrenmesi modellerinin genellikle eksik bıraktığı açıklanabilirlik ihtiyacına yanıt veren önemli bir çıktı üretmiştir. Bu kurallar, özellikle klinik çalışanların sınıflandırma sonuçlarını daha rahat değerlendirebilmesi açısından önemli bir avantaj sağlamaktadır.

Belirsizlik analizi bileşeninden sonra tesis edilen sistemin omurgasını oluşturan çok katmanlı yazılım mimarisine geçilmiştir. Bu mimari, temel olarak üç ana bileşenden oluşmaktadır: (1) C# tabanlı kullanıcı arayüzü, (2) Apache Kafka tabanlı gerçek zamanlı veri iletim sistemi ve (3) Java tabanlı analiz katmanı. Bu mimari yapı, verinin sisteme girişinden analiz sonuçlarının veritabanına kaydedilmesine kadar uzanan süreci tamamen otomatikleştiren bir yaklaşım sunmaktadır. C# tarafında geliştirilen kullanıcı arayüzü, laborantın hastaya ait ölçümleri sisteme girmesi için kullanıcı dostu bir platform sağlamaktadır. Ancak bu arayüz klasik CRUD işlemleriyle veriyi doğrudan veritabanına yazmak yerine, Kafka Producer bileşeni aracılığıyla veriyi Kafka'ya iletmektedir. Kafka, büyük ölçekli veri akışları için tasarlanmış yüksek performanslı bir mesajlaşma platformu olduğundan, sistemin gerçek zamanlı çalışması için kritik bir rol üstlenmektedir.

Kafka Producer bileşeni, her yeni hasta verisini JSON formatında Kafka topic'ine iletmekte ve veri akışını başlatmaktadır. Topic yapısı, gelen verilerin sıralı ve tutarlı biçimde işlenmesini sağlar. Java tarafında çalışan Kafka Consumer bileşeni ise bu mesajları anında algılamakta ve analiz sürecini başlatmaktadır. Bu yapı, sistemin olay tetiklemeli (event-driven) olarak çalışmasını mümkün kılmaktadır. Consumer bileşeninin veriyi algılamasıyla birlikte Java tabanlı analiz katmanı devreye girer ve makine öğrenmesi algoritmaları otomatik olarak çalıştırılır. Bu süreç, klasik veri tabanı sorgulamalarından çok daha hızlı ve daha ölçeklenebilir bir mekanizma sunmaktadır.

Java tabanlı analiz katmanında yer alan makine öğrenmesi algoritmaları (Naive Bayes, KNN, KNC, Logistic Regression, AdaBoost ve Ensemble), manuel olarak kodlanmış olup gerçek zamanlı veri akışına uyumlu çalışacak şekilde optimize edilmiştir. Bu algoritmaların manuel olarak geliştirilmesi, hem performans üzerinde daha fazla kontrol sağlamış hem de hazır kütüphanelerin getirdiği kısıtlamalardan kaçınılmasına olanak tanımıştır. Bu analiz aşamasında özellikle KNN ve Ensemble modellerinin yüksek doğruluk performansı sergilediği gözlemlenmiştir. Ensemble modeli, farklı algoritmaların güçlü yönlerini bir araya getirerek daha kararlı ve yüksek doğruluklu sonuçlar üretmesi bakımından sistemin en başarılı sınıflandırıcılarından biri olarak öne çıkmıştır. Her yeni veri Kafka üzerinden geldiğinde bu algoritmalar otomatik olarak çalıştırılmış ve tahmin sonuçları elde edilmiştir.

Analiz süreci tamamlandığında sonuç MSSQL veritabanına kaydedilmiş ve kullanıcı arayüzü üzerinden anında görüntülenebilir hâle getirilmiştir. Böylece verinin girişinden sonuçların elde edilmesine kadar geçen süreç tamamen otomatik, senkronize ve gerçek zamanlı bir yapıya kavuşmuştur. Bu mimarinin özellikle klinik ortamlarda hızlı karar verme gerektiren senaryolarda önemli avantajlar sunduğu düşünülmektedir.

Bu tezde önerilen çok katmanlı ve yüksek performanslı mimari yalnızca tek bir sınıflandırma problemini çözmek için değil, aynı zamanda modern sağlık bilişimi sistemlerinin nasıl kurgulanması gerektiğine yönelik daha geniş bir bakış açısı sunmak amacıyla tasarlanmıştır. Bu mimari, verinin üretildiği noktadan analiz sonuçlarının kullanıcıya iletilmesine kadar geçen sürecin her aşamasını optimize eden bir yapı ortaya koymaktadır. C# tabanlı kullanıcı arayüzünden başlayan veri akışı, gerçek zamanlı veri iletimini mümkün kılan Kafka sistemi üzerinden Java tabanlı analiz katmanına ulaşmakta; analiz tamamlandıktan sonra sonuçlar MSSQL veritabanına kaydedilmekte ve kullanıcıya geri dönmektedir. Bu akış içinde veri, hiçbir aşamada gereksiz şekilde bekletilmemekte veya yavaşlatılmamaktadır. Bu nedenle sistem hem hız hem de güvenilirlik açısından klinik uygulamalara uygun bir yapı sergilemektedir.

Analiz katmanında kullanılan makine öğrenmesi algoritmalarının her biri farklı veri davranışlarını modelleme kapasitesine sahiptir. KNN algoritmasının örüntü temelli yaklaşımı, sağlık verilerinin çeşitli alt gruplarını başarılı şekilde ayırt edebilirken; Logistic Regression daha lineer ilişkilerin hâkim olduğu alanlarda dengeli sonuçlar üretebilmektedir. AdaBoost algoritmasının zayıf sınıflandırıcıları birleştirerek daha güçlü bir model oluşturması ise veri setindeki karmaşık bölgelerin daha etkili şekilde çözümlenmesini sağlamaktadır. Ensemble modelinin diğerlerine kıyasla daha başarılı performans sergilemesinin temel nedeni, birden fazla algoritmanın güçlü yönlerini harmanlayarak daha kararlı bir tahmin gücü sunmasıdır. Bu durum, özellikle sağlık verilerinin heterojen yapısıyla uyumludur; çünkü sağlık verileri çoğu zaman tek bir modelin açıklamakta zorlanacağı karmaşık ilişkiler içerir.

RST'nin sunduğu karar kuralları, makine öğrenmesi algoritmalarının genellikle eksik kaldığı açıklanabilirlik konusuna önemli bir katkı sağlamıştır. RST ile üretilen kurallar, veri setindeki hangi kategori kombinasyonlarının diyabet riskini artırdığını açık biçimde göstermektedir. Örneğin belirli BMI değerleri, belirli glikoz aralıkları veya belirli yaş sınıfları birlikte değerlendirildiğinde yüksek riskin ortaya çıktığı görülebilmektedir. Bu tür kurallar, modelin neden belirli bir sonucu ürettiğini açıklamak açısından değerlidir ve klinik karar desteği süreçlerinde hem şeffaflığı hem de kullanıcı güvenini artıran bir unsur olarak öne çıkmaktadır.

Gerçek zamanlı uçtan uca bu sistemin güçlü yönlerinden biri, laborant tarafından girilen tüm verilerin analiz sürecinde herhangi bir manuel müdahale gerektirmeden işlenmesidir. Kafka tabanlı olay tetiklemeli yapı sayesinde sistem, veri girişi yapıldığı anda analiz sürecini başlatmakta, kullanıcıyı bekletmeden sonuç üretmekte ve bu sonuçları veritabanına kaydederek kullanıcıya sunmaktadır. Bu yapı, özellikle yoğun klinik ortamlarda zamandan tasarruf sağlamak ve insan hatasını en aza indirmektedir. Ayrıca sistemin modüler yapısı, yeni algoritmaların kolayca eklenmesine, iş akışının genişletilmesine ve veritabanı şemasının ihtiyaçlara göre güncellenmesine olanak tanımaktadır. Bu nedenle geliştirilen sistem yalnızca diyabet teşhisine yönelik bir araç değil, aynı zamanda farklı hastalık türleri için uyarlanabilir bir platform niteliği taşımaktadır.

Sonu olarak bu tez alıřması, diyabet teřhisinin doęruluk, hız, açıklanabilirlik ve gerek zamanlı veri iřleme kapasitesi aısından modern bir yaklaşım sunmaktadır. Geliřtirilen mimari, veri öniřleme süreçlerinden sentetik veri üretimine, belirsizlik analizi yöntemlerinden gerek zamanlı makine öğrenmesine kadar pek ok teknolojiyi tek bir çatı altında birleřtirmekte ve uçtan uca bir özüm üretmektedir. Bu yapı, saęlık biliřimi alanında gelecekte geliřtirilecek sistemler iin hem güçlü bir referans hem de potansiyeli yüksek bir bařlangı noktası oluřturmaktadır. Sistem, özellikle saęlık kuruluşlarında hızlı risk deęerlendirmesi gerektiren durumlarda kullanılabilircek pratik bir karar destek mekanizması sunmakta ve klinik personelin daha doęru ve zamanında kararlar almasına yardımcı olmaktadır.



ANALYZING DIABETES DATA SET USING ROUGH CLUSTERING THEORY AND ESTABLISHING A REAL-TIME DATA ANALYTICS SYSTEM FOR DIABETES DISEASE

SUMMARY

This thesis aims to develop a comprehensive decision support system by integrating contemporary data science methods, statistical pre-processing techniques, machine learning algorithms, synthetic data generation systems, uncertainty-based mathematical analysis approaches, and real-time data streaming technologies for the early diagnosis of diabetes. Diabetes is one of the most critical chronic diseases in modern medicine, both due to its prevalence on a global scale and the long-term health complications it causes. Detecting the disease with high accuracy in its early stages is of great importance in preventing cardiovascular, neurological, and metabolic complications that may arise in later stages. Therefore, data-driven decision support mechanisms are becoming increasingly important in modern medicine. This thesis presents a multi-layered architecture that encompasses the entire process, from the collection of data to its analysis and the real-time presentation of results to the user, rather than merely creating a machine learning model. Thus, beyond classical classification approaches, a real-time and clinically applicable solution is targeted.

The Pima Indians Diabetes data set used in this thesis served as the starting point for the study, as it contains fundamental problems commonly seen in medical data sets, such as missing data, imbalanced class structure, and measurement errors. It was determined that some physiological measurements in the data set contained zero values; since it is biologically impossible for values such as glucose, insulin, or skin thickness to be zero, these data were considered missing or erroneous. Therefore, a systematic data pre-processing process was carried out, the IQR (Interquartile Range) method was used to identify outliers, and extreme values that distorted the distribution were corrected as necessary. As a result of these operations, the statistical integrity of the data set was improved, enabling machine learning models to be trained on a more accurate basis.

One of the significant problems with the data set, class imbalance, was addressed using the SMOTE (Synthetic Minority Over-sampling Technique) algorithm. As the low representation of the positive class caused the models to favour the negative class, the minority class was increased within the data space using SMOTE, creating a balanced training set. Furthermore, the CTGAN (Conditional Tabular GAN) method was applied to expand the data set, increasing it to 2000 observations. Thanks to CTGAN's ability to learn complex distributions, the synthetic data was generated in a structure consistent with the original data set, which was an important factor in increasing the model's generalisation ability. These steps created a robust database for the subsequent sections of the study.

Following the successful completion of the data pre-processing stage, the study proceeded to the uncertainty analysis phase, which constitutes the second key component of the thesis. Within this scope, an explainability-based knowledge

extraction was performed on the data set using Rough Set Theory (RST). RST is a powerful mathematical method, particularly suitable for working with categorical data, known for its capacity to perform reduction operations and generate decision rules in data sets containing uncertainty. Therefore, the continuous variables in the data set were divided into clinically meaningful intervals and converted into categorical form. Following the categorical conversion, RST software developed specifically in C# was used to calculate the reduct, core, positive region, and decision rules. These analyses revealed that no attribute was entirely redundant and that all variables contributed to a certain degree to the diagnosis of diabetes. The fact that the reduct results emerged in this manner demonstrates the multidimensional nature of diabetes and that it is a complex disease that cannot be explained by a single parameter. Furthermore, the decision rules generated by RST, structured as 'If ... then ...', produced an important output that addresses the need for explainability, which machine learning models often lack. These rules provide a significant advantage, particularly in enabling clinical staff to evaluate classification results more easily.

Following the uncertainty analysis component, the system established subsequently transitions to a multi-layered software architecture that forms its backbone. This architecture essentially comprises three main components: (1) a C#-based user interface, (2) a real-time data transmission system based on Apache Kafka, and (3) a Java-based analysis layer. This architectural structure offers an approach that fully automates the process from data entry into the system to the recording of analysis results in the database. The user interface developed on the C# side provides a user-friendly platform for the laboratory technician to enter the patient's measurements into the system. However, instead of writing the data directly to the database using classic CRUD operations, this interface transmits the data to Kafka via the Kafka Producer component. Kafka, being a high-performance messaging platform designed for large-scale data streams, plays a critical role in the real-time operation of the system.

The Kafka Producer component transmits each new patient record in JSON format to the Kafka topic and initiates the data stream. The topic structure ensures that incoming data is processed in a sequential and consistent manner. The Kafka Consumer component, running on the Java side, instantly detects these messages and initiates the analysis process. This structure enables the system to operate in an event-driven manner. Once the Consumer component detects the data, the Java-based analysis layer is activated and machine learning algorithms are automatically run. This process offers a mechanism that is much faster and more scalable than traditional database queries.

The machine learning algorithms (Naive Bayes, KNN, KNC, Logistic Regression, AdaBoost, and Ensemble) within the Java-based analysis layer are manually coded and optimised to operate in real-time with data streams. The manual development of these algorithms has provided greater control over performance and allowed the avoidance of the limitations imposed by ready-made libraries. During this analysis phase, the KNN and Ensemble models in particular demonstrated high accuracy performance. The Ensemble model stood out as one of the system's most successful classifiers, producing more stable and accurate results by combining the strengths of different algorithms. Whenever new data arrived via Kafka, these algorithms were automatically run and prediction results were obtained.

Once the analysis process is complete, the results are saved to the MSSQL database and made instantly viewable via the user interface. Thus, the entire process from data entry to obtaining results has become fully automated, synchronised, and real-time.

This architecture is considered to offer significant advantages, particularly in clinical settings where rapid decision-making is required.

The multi-layered, high-performance architecture proposed in this thesis is designed not only to solve a single classification problem, but also to offer a broader perspective on how modern health informatics systems should be structured. This architecture presents a structure that optimises every stage of the process, from the point at which data is produced to the delivery of analysis results to the user. The data flow, starting from the C#-based user interface, reaches the Java-based analysis layer via the Kafka system, which enables real-time data transmission. Once the analysis is complete, the results are stored in the MSSQL database and returned to the user. Within this flow, data is not unnecessarily delayed or slowed down at any stage. Therefore, the system exhibits a structure suitable for clinical applications in terms of both speed and reliability.

Each machine learning algorithm used in the analysis layer has the capacity to model different data behaviours. While the pattern-based approach of the KNN algorithm can successfully distinguish between various subgroups of health data, Logistic Regression can produce balanced results in areas where linear relationships prevail. The AdaBoost algorithm's ability to combine weak classifiers to create a more powerful model ensures that complex areas in the data set are resolved more effectively. The main reason ensemble models perform better than others is that they combine the strengths of multiple algorithms to provide more stable prediction power. This is particularly compatible with the heterogeneous nature of health data, as health data often contains complex relationships that a single model would struggle to explain.

The decision rules provided by RST have made a significant contribution to the issue of interpretability, an area where machine learning algorithms often fall short. The rules generated by RST clearly show which category combinations in the data set increase the risk of diabetes. For example, it can be seen that high risk emerges when specific BMI values, specific glucose ranges, or specific age groups are evaluated together. Such rules are valuable in explaining why the model produced a specific result and stand out as an element that increases both transparency and user confidence in clinical decision support processes.

One of the key strengths of this real-time end-to-end system is that all data entered by the laboratory technician is processed during the analysis process without requiring any manual intervention. Thanks to its Kafka-based event-triggered architecture, the system initiates the analysis process as soon as data is entered, produces results without delaying the user, and records these results in the database for presentation to the user. This architecture saves time, particularly in busy clinical environments, and minimises human error. Furthermore, the system's modular structure allows for the easy addition of new algorithms, the expansion of the workflow, and the updating of the database schema according to needs. Therefore, the developed system is not only a tool for diabetes diagnosis but also serves as an adaptable platform for different disease types.

In conclusion, this thesis presents a modern approach to diabetes diagnosis in terms of accuracy, speed, explainability, and real-time data processing capabilities. The developed architecture integrates numerous technologies under a single umbrella, from data pre-processing to synthetic data generation, uncertainty analysis methods to real-time machine learning, thereby producing an end-to-end solution. This structure serves as both a robust reference and a high-potential starting point for systems to be

developed in the field of health informatics in the future. The system offers a practical decision support mechanism that can be used in situations requiring rapid risk assessment, particularly in healthcare institutions, and assists clinical staff in making more accurate and timely decisions.



1. GİRİŞ

Dijitalleşmenin hızla yayıldığı günümüzde, sağlık sektöründe veri analitiği ve karar destek sistemleri giderek daha kritik bir rol oynamaktadır. Hastalıkların erken teşhisi ve tedavi süreçlerinin optimize edilmesi, yalnızca bireylerin yaşam kalitesini artırmakla kalmamakta, aynı zamanda sağlık sistemlerine olan yükü de azaltmaktadır. Bu kapsamda, makine öğrenmesi ve veri madenciliği teknikleri sağlık bilişimi alanında büyük bir potansiyel barındırmaktadır. Ancak, bu potansiyelin etkin şekilde hayata geçirilebilmesi için, sadece yüksek doğruluklu modellerin değil, aynı zamanda gerçek zamanlı çalışan, esnek ve sürdürülebilir altyapıların da geliştirilmesi gerekmektedir.

Bu çalışmada, veri madenciliği ve büyük veri teknolojileri bir araya getirilerek uçtan uca bir teşhis simülasyonu tasarlanmıştır. Özellikle diyabet hastalığının erken teşhisine odaklanan bu sistem, kapsamlı bir veri ön işleme süreciyle başlamış, Kaba Kümeleme Teorisi ile desteklenen karar kurallarının çıkarımı ile devam etmiş ve son olarak büyük veri teknolojileri aracılığıyla gerçek zamanlı teşhis süreci ile sonlanmıştır. Sistem; literatürde sıkça kullanılan Pima Indians Diabetes veri seti üzerinde geliştirilmiş, C# tabanlı masaüstü uygulamalar, Apache Kafka mimarisi ve Java ile geliştirilen makine öğrenmesi modelleri kullanılarak tamamlanmıştır.

Bu tez çalışmasında, diyabet hastalığının erken teşhisine yönelik olarak Kaba Kümeleme Teorisi (Rough Set Theory) temelli bir veri analizi süreci gerçekleştirilmiş ve ardından büyük veri teknolojileri ile desteklenen gerçek zamanlı bir karar destek sistemi geliştirilmiştir. Literatürde sıklıkla kullanılan Pima Indians Diabetes veri seti temel alınmış ve bu veri seti üzerinden eksiksiz bir ön işleme süreci yürütülerek sistemin kararlılığı ve doğruluğu maksimize edilmiştir. Tez çalışması üç temel yapıdan oluşmaktadır: veri ön işleme ve kaba kümeleme analizi, hastane otomasyon sistemi simülasyonu ve büyük veri temelli gerçek zamanlı teşhis altyapısı.

1.1. Çalışmanın Amacı

Diyabet, dünya genelinde giderek artan bir sağlık sorunu haline gelmiş, milyonlarca bireyin yaşam kalitesini ve süresini olumsuz etkileyen kronik bir metabolik bozukluktur. Dünya Sağlık Örgütü'ne göre, 2021 yılı itibarıyla dünya genelinde yaklaşık 537 milyon yetişkinin diyabet hastalığına sahip olduğu bildirilmiştir ve bu sayının 2030 yılında 643 milyona ulaşması beklenmektedir [1]. Diyabetin bu denli yaygınlaşması, erken teşhis ve etkili tedavi yöntemlerinin geliştirilmesini zorunlu hale getirmiştir. Ancak mevcut sistemler çoğu zaman statik ve reaktif yapıdadır; hastalığın ilerleyişi ancak belirtiler şiddetlendiğinde tespit edilebilmektedir. Bu da, hastalığın seyrini olumsuz yönde etkileyerek hem bireysel sağlık hem de toplumsal sağlık yükü açısından ciddi sonuçlar doğurmaktadır.

Bu tez çalışmasının temel amacı, diyabet teşhisine yönelik hem analitik hem de teknolojik olarak entegre bir yaklaşım sunmaktır. Bu doğrultuda ilk olarak, literatürde sıkça kullanılan Pima Indians Diabetes veri seti temel alınarak; eksik, anlamsız ve uç değerler içeren kayıtların temizlendiği, sınıf dengesizliğinin SMOTE yöntemiyle giderildiği ve ardından sentetik veri üretimiyle genişletilmiş bir veri seti oluşturulmuştur. Veri madenciliği ve makine öğrenmesi süreçlerinde veri kalitesi, model başarımını doğrudan etkileyen temel bir faktördür [2]. Bu nedenle verinin özenle hazırlanması, sistemin doğruluk ve güvenilirlik düzeyini doğrudan etkilemiştir.

Çalışmanın ikinci ayağında ise, karar destek sistemlerinde alternatif bir yaklaşım olan Kaba Kümeleme Teorisi (Rough Set Theory) kullanılmıştır. Kaba kümeleme, özellikle belirsiz, eksik ya da kısmi bilginin hâkim olduğu veri yapılarında anlamlı sonuçlar üretebilme potansiyeliyle öne çıkmaktadır. Pawlak tarafından geliştirilen bu teori, klasik kümeleme yöntemlerinden farklı olarak, alt ve üst yaklaşım kümeleri, pozitif bölge ve gereklilik analizleri gibi özgün kavramlarla verinin içsel yapısını analiz etmeye olanak tanımaktadır [3]. Bu özellikleri sayesinde, sağlık gibi kompleks ve çok değişkenli alanlarda önemli avantajlar sunmaktadır.

Tezin üçüncü hedefi, geliştirilen analitik yapının gerçek zamanlı sistemlerle entegrasyonunu sağlamaktır. Bu amaçla Apache Kafka ve Zookeeper gibi büyük veri teknolojileri kullanılarak uçtan uca bir teşhis simülasyonu gerçekleştirilmiştir. Gerçek zamanlı hasta verilerinin anlık olarak analiz edilip karar destek sistemine entegre edilmesi, sadece teşhis hızını artırmakla kalmamış; aynı zamanda sistemin dinamik, ölçeklenebilir ve sürdürülebilir olmasını da sağlamıştır.

Sonuç olarak bu çalışmanın amacı, diyabet teşhisinde hem klasik hem de modern analitik yaklaşımları bir araya getiren, veri kalitesine odaklanan, belirsizlikleri değerlendirebilen ve büyük veri teknolojileriyle desteklenen bütünsel bir karar destek sistemi sunmaktır. Böylece hem akademik alanda yenilikçi bir yöntem önerisi getirilmiş hem de sağlık sektörüne uygulanabilir bir çözüm önerisi sunulmuştur.

1.2. Çalışmanın Kapsamı ve Önemi

Bu tez çalışması, diyabet teşhisinde veri analitiği, belirsizlik temelli karar sistemleri ve büyük veri teknolojilerinin bütüncül bir biçimde kullanıldığı çok katmanlı bir mimari sunmaktadır. Çalışma; veri ön işleme, kaba kümeleme analizi, gerçek zamanlı veri akışı ve makine öğrenmesi modellerinin entegrasyonunu içermekte olup, sağlık bilişimi alanında uygulanabilir, yenilikçi ve yüksek doğruluk sunan bir teşhis simülasyonu hedeflemektedir.

İlk olarak, çalışmanın kapsamı Pima Indians Diabetes veri seti ile sınırlandırılmıştır. Bu veri seti, 768 gözlem ve 8 giriş nitelik + 1 hedef değişken içeren, diyabet teşhisinde dünya genelinde sıklıkla kullanılan bir benchmark veri kümesidir [3]. Veri seti üzerinden gerçekleştirilen ön işleme aşamasında eksik değer kontrolü, anlamsız sıfırların temizlenmesi, uç değerlerin IQR yöntemi ile yeniden üretilmesi, SMOTE algoritmasıyla sınıf dengesinin sağlanması ve ardından CTGAN ile sentetik veri üretimi gerçekleştirilmiştir. Böylece sadece verinin hacmi değil, içeriğinin istatistiksel anlamlılığı ve sınıf yapısındaki dengesizlikler de giderilerek, algoritmalar için uygun bir temel oluşturulmuştur.

Çalışmanın ikinci katmanı, belirsiz ve eksik bilgiye dayanıklı karar destek yaklaşımlarından biri olan Kaba Kümeleme Teorisi'nin (Rough Set Theory) uygulanmasıdır. Bu teori, klasik makine öğrenmesi algoritmalarından farklı olarak, verideki her bir özniteliğin karar sürecine olan etkisini ölçebilmekte ve bu sayede gereksiz öznitelikleri elimine edebilmektedir [2, 4]. Ancak bu çalışmada yapılan analiz sonucunda veri setinde yer alan tüm özniteliklerin gerekli olduğu tespit edilmiş ve bu durum sistemin her bileşeninin karar sürecine katkı sunduğunu göstermiştir.

Çalışmanın en önemli katkılarından biri ise, geliştirilen sistemin gerçek zamanlı veri akışıyla çalışabilen, çok katmanlı ve dağıtık bir mimariye sahip olmasıdır. Bu kapsamda, C# diliyle geliştirilen kullanıcı arayüzü üzerinden laborant tarafından hastaya ait ölçümler sisteme girilmekte ve bu veriler, geleneksel veritabanı sorguları

yerine, C# uygulamasına entegre edilmiş Kafka Producer bileşeni aracılığıyla doğrudan Apache Kafka altyapısına aktarılmaktadır. Producer bileşeni, önceden tanımlanmış topic ve broker ayarları üzerinden Java tarafında sürekli dinlemede olan Kafka Consumer bileşeniyle iletişim kurmaktadır.

Veri işleme süreci, bileşenler arasında görev ayrımı yapılmış modüler bir yapıya sahiptir. Makine öğrenmesi algoritmalarının ana işlemleri, Java ortamında konumlandırılan Kafka Consumer bileşeni aracılığıyla gerçekleştirilmektedir. Naive Bayes, KNN, KNC, Logistic Regression ve AdaBoost gibi algoritmalar Java diliyle manuel olarak geliştirilmiş ve Kafka mimarisiyle bütünleşik bir şekilde yapılandırılmıştır. Her yeni veri Kafka üzerinden ulaştığında, bu algoritmalar aktif hâle gelerek tahmin sürecini başlatmaktadır. Öte yandan, C# tarafında yürütülen kullanıcı arayüzü ve veri iletim süreçleri, sistemin yalnızca tek bir dile bağlı kalmaksızın çoklu platformlar üzerinden işlem yapmasına olanak tanımaktadır. Bu yapı sayesinde veri işleme yalnızca Java tabanlı bir platformla sınırlı kalmamakta, aynı zamanda C# tarafındaki çıktı analizi, yorumlama ve kullanıcıya geri bildirim gibi süreçlerle desteklenmektedir.

Veri işleme süreci, olay tetiklemeli (event-driven) bir mimari yapıya kavuşturularak tüm işlem adımları Java ortamında merkezi olarak gerçekleştirilmiştir. Geliştirilen sistemde, Kafka Consumer bileşeni Apache Kafka üzerinden gelen her yeni veriyi anlık olarak algılamakta; bu tetiklenme sonucunda, makine öğrenmesi algoritmaları doğrudan ve eş zamanlı olarak çalıştırılmaktadır. Naive Bayes, KNN, KNC, Logistic Regression, AdaBoost ve çeşitli Ensemble modelleri de dahil olmak üzere tüm tahminleme işlemleri Java ortamında kodlanmış ve Kafka altyapısı ile sıkı bir şekilde bütünleştirilmiştir. Bu yapı sayesinde, veri girişinden sonuç elde edilmesine kadar geçen tüm süreç uçtan uca otomatikleştirilmiş, manuel müdahale ihtiyacı tamamen ortadan kaldırılmıştır.

Makine öğrenmesi modüllerinin tamamının Java platformuna taşınması, sistemin yapısal bütünlüğünü ve işlem güvenliğini önemli ölçüde artırmıştır. Tüm analiz süreci, Apache Kafka'nın yüksek hacimli ve sıralı (FIFO) veri iletim yetenekleriyle desteklenerek, dağıtık bir sistem mimarisi içinde güvenilir biçimde yürütülmektedir. Her yeni hasta verisi Kafka üzerindeki tanımlı topic'e ulaştığında, Kafka Consumer bileşeni bu veriyi sırasıyla algılamakta ve olay tetiklemeli (event-driven) yapı sayesinde tahminleme sürecini başlatmaktadır. Veriler, işlenmek üzere sırayla

kuyruktan çekilmekte; her bir örnek için makine öğrenmesi algoritmaları ilgili tahmin işlemini tamamladıktan sonra bir sonraki veriye geçilmektedir. Bu mimari, hem işlem sürekliliğini hem de veri akış kontrolünü sağlayarak, gerçek zamanlı analiz kavramını teknik olarak doğru ve kararlı bir biçimde hayata geçirmektedir.

Gerçek zamanlı analiz sürecinin tamamlanmasının ardından, elde edilen tahmin sonuçları herhangi bir geçici tampon dosyaya ya da manuel aktarım sürecine ihtiyaç duyulmaksızın doğrudan MSSQL veritabanına kaydedilmektedir. Bu yaklaşım, sistemin hem hızını hem de hata toleransını artırmakta; veri akışını tek bir iş hatasıyla kesintiye uğramayacak şekilde düzenlemektedir. Kafka Consumer bileşeni, yalnızca analiz görevinden sorumlu olacak şekilde yapılandırılmış; veritabanı işlemleri ise analiz tamamlandıktan sonra sistem tarafından tetiklenen ikinci bir modül aracılığıyla güvenli biçimde gerçekleştirilmiştir. Böylece sistem, görev ayrımı ilkesine sadık kalarak hem analiz sürecinde yüksek işlem performansı sağlamak hem de veri bütünlüğünü korumaktadır.

Bu uçtan uca olay tetiklemeli yapı, yalnızca yüksek doğruluk oranları elde etmekle kalmayıp, aynı zamanda düşük gecikme süresi, tutarlılık (consistency) ve sürdürülebilirlik gibi modern yazılım mimarisi kriterlerini de karşılamaktadır. Her veri girişi, arka planda manuel müdahaleye gerek olmadan otomatik olarak analiz edilmekte; sonuçlar gecikme olmaksızın veritabanına iletilmekte ve kullanıcı arayüzünden anlık olarak erişilebilir hâle gelmektedir. Bu yapı, sağlık sektöründe sıkça ihtiyaç duyulan anlık geri bildirim gereksinimini karşılayarak, hem klinik kararlarda hızlı destek sunmakta hem de dijitalleşmiş hastane altyapıları için sürdürülebilir ve genişletilebilir bir çözüm sunmaktadır.

Sonuç olarak, bu tez çalışması veri bilimi, belirsizlik teorileri ve büyük veri teknolojilerini disiplinlerarası bir perspektifle harmanlayarak, sağlık alanında uygulanabilir, sürdürülebilir ve gerçek zamanlı karar destek sistemleri için örnek teşkil eden çok katmanlı bir mimari ortaya koymuştur. Geliştirilen sistem yalnızca yüksek algoritmik doğruluk sağlamakla kalmayıp, veri ön işleme kalitesi, sınıf dengesi, platformlar arası görev dağılımı ve yapısal tutarlılık gibi kritik boyutlarda da üstün performans göstermiştir. Bu yönüyle çalışma, akademik literatüre metodolojik katkılar sunmanın ötesinde, sağlık kurumlarının dijital dönüşüm süreçlerine entegre edilebilecek, esnek, güvenilir ve pratik bir çözüm altyapısı niteliği taşımaktadır.

1.3. Tezin Organizasyonu

Bu tez, diyabet hastalığının teşhisine yönelik olarak geliştirilen veri analitiği mimarisini sistematik bir şekilde açıklamak amacıyla yedi ana bölümden oluşmaktadır. Her bölüm, çalışmanın farklı bir yönünü detaylı şekilde ele almakta ve birbirini tamamlayacak biçimde kurgulanmıştır.

Birinci bölüm olan **Giriş** kısmında, çalışmanın arka planı, amacı, önemi ve kapsamı sunulmuş; tezin genel çerçevesi çizilmiştir. Bu bölüm, okuyucunun tez boyunca ele alınacak konulara dair bütüncül bir perspektif edinmesini sağlamak amacıyla hazırlanmıştır.

İkinci bölüm olan **Literatür Taraması** kısmında, diyabet hastalığının tıbbi arka planı ve teşhis sürecinde öne çıkan risk faktörleri tanıtılmış; sağlık bilişiminde veri analitiği yaklaşımlarına yer verilmiştir. Bu kapsamda, özellikle Kaba Kümeleme (Rough Set) Teorisi'ne ilişkin yerli ve yabancı literatürde yapılmış olan çalışmalar incelenmiş; ayrıca gerçek zamanlı veri işleme araçları ve makine öğrenmesi tekniklerinin diyabet teşhisi bağlamındaki kullanımları değerlendirilmiştir.

Üçüncü bölüm olan **Veri Seti ve Ön İşleme Süreci**, çalışmanın teknik temelini oluşturmaktadır. Bu bölümde Pima Indians Diabetes veri seti tanıtılmış; eksik ve aykırı değerlerin analizi, SMOTE algoritmasıyla sınıf dengesinin sağlanması, CTGAN ve sklearn tabanlı sentetik veri üretimi, ardından tüm değişkenlerin kategorik hale getirilme süreci detaylı biçimde açıklanmıştır. Ayrıca, veri setinin %70 eğitim ve %30 test şeklinde ayrıldığı ve gerçek zamanlı sistemin yalnızca test verileri üzerinde performans değerlendirmesine tabi tutulduğu belirtilmiştir.

Dördüncü bölüm olan **Kaba Kümeleme Teorisi ve Karar Kuralları Üretimi** başlığı altında, Rough Set Teorisi'nin temel kavramları, öznitelik indirgeme, pozitif/negatif bölgeler ve karar kurallarının çıkarımı açıklanmış; geliştirilen C# yazılımının mimarisi ve karar kurallarının görselleştirilmesi anlatılmıştır.

Beşinci bölüm olan **Gerçek Zamanlı Veri Analitiği Mimarisi**, Apache Kafka altyapısı ile desteklenen ve C# ile Java platformlarının mikroservis benzeri entegrasyonunu içeren çok katmanlı gerçek zamanlı teşhis sistemini kapsamlı şekilde tanıtmaktadır. Bu bölümde, laborantın C# ile geliştirilen kullanıcı arayüzü üzerinden hasta ölçümlerini girerek Kafka Producer bileşeni aracılığıyla verilerin doğrudan Apache Kafka'ya iletilmesi süreci anlatılmaktadır. Veri işleme görevleri tamamen Java

tabanlı Kafka Consumer bileşeni tarafından yürütülmekte; Naive Bayes, KNN, KNC, Logistic Regression ve AdaBoost gibi makine öğrenmesi algoritmaları burada aktif hale gelmektedir. Sistem, gelen verilerin FIFO mantığıyla sıralandığı Kafka kuyruğu sayesinde birden fazla veri aynı anda geldiğinde bile işlemleri sırasıyla ve kesintisiz yürütmektedir. Tahmin sonuçları Kafka Consumer tarafından doğrudan MSSQL veritabanına kaydedilmekte olup, ara dosya veya tamponlama mekanizması kullanılmamaktadır. Bu yapı, gerçek zamanlı veri işleme ve hızlı tepki süresi gerektiren sağlık uygulamaları için yüksek performans ve güvenilirlik sunmaktadır. Bu bölümde ayrıca sistem entegrasyonu, veri iletim mantığı ve gerçek zamanlı veri görselleştirme süreçleri detaylı biçimde açıklanmaktadır.

Altıncı bölüm olan **Sonuçlar ve Değerlendirme**, önceki bölümlerde geliştirilen tüm bileşenlerin çıktılarının değerlendirilmesini kapsamaktadır. Ön işleme sürecinin etkileri, Kaba Kümeleme sonuçları ve karar kurallarının başarımı, gerçek zamanlı mimari ile entegre edilen makine öğrenmesi modellerinin doğruluk oranları, test süreleri ve tahmin kabiliyetleri ayrıntılı biçimde sunulmuştur. Bu kapsamda iki ayrı grafikte algoritmaların doğruluk oranları ve işlem süreleri karşılaştırmalı olarak görselleştirilmiştir.

Yedinci ve son bölüm olan **Genel Sonuçlar ve Öneriler** kısmında ise, çalışmanın literatüre olan katkıları vurgulanmış, geliştirilen sistemin pratikteki uygulanabilirliği üzerine çıkarımlar yapılmış ve gelecekte yapılabilecek çalışmalar için öneriler sunulmuştur. Özellikle karar şeffaflığı, gerçek zamanlı alarm sistemleri ve mobil uygulama önerileri gibi açılımlar detaylandırılmıştır.

Tüm bu bölümler birlikte değerlendirildiğinde, bu tez çalışmasının hem teorik katkı hem de uygulamalı fayda açısından bütüncül bir yapıya sahip olduğu görülmektedir.



2. LİTERATÜR TARAMASI

2.1. Diyabet Hastalığı ve Risk Faktörleri

Diyabet, modern çağın en yaygın kronik hastalıklarından biri olup, bireylerin yaşam kalitesini ciddi şekilde etkileyen ve dünya genelinde sağlık sistemlerine büyük bir yük getiren metabolik bir bozukluktur. Diyabetin temel özelliği, kan şekeri düzeyinin normalin üzerine çıkması (hiperglisemi) ve bu durumun uzun süreli olmasıdır. Dünya Sağlık Örgütü'nün (WHO) 2023 yılı verilerine göre, dünya genelinde diyabet hastası birey sayısı 500 milyonu aşmış olup, bu sayının önümüzdeki yıllarda artması beklenmektedir [1]. Özellikle düşük ve orta gelirli ülkelerde diyabet kaynaklı ölüm oranlarının yüksekliği dikkat çekmektedir.

Tıbbi literatürde diyabet başlıca iki tipe ayrılmaktadır: Tip 1 Diyabet ve Tip 2 Diyabet. Tip 1 Diyabet genellikle çocukluk veya genç erişkinlik döneminde başlayan, insülin üretiminin durduğu otoimmün bir hastalık iken, Tip 2 Diyabet daha çok erişkin yaşlarda ortaya çıkan, insülin direnci ve pankreasın yeterli insülin üretememesi ile karakterize edilen bir durumdur. Çalışmalarda, Tip 2 diyabetin, küresel diyabet vakalarının yaklaşık %90'ını oluşturduğu bildirilmektedir [5].

Diyabetin erken teşhisi, komplikasyonların önlenmesi ve yaşam kalitesinin artırılması açısından büyük önem taşımaktadır. Ancak bu süreç, çoğu zaman bireylerin semptom göstermemesi veya hastalığın sinsi ilerlemesi nedeniyle zorlaşmaktadır. Bu nedenle diyabetin teşhisinde sadece semptomlar değil, bireylerin taşıdığı risk faktörleri de dikkate alınmalıdır.

Diyabetin gelişiminde etkili olan başlıca risk faktörleri arasında şunlar yer almaktadır:

- **Genetik yatkınlık:** Ailede diyabet öyküsünün bulunması, bireyin hastalığa yakalanma riskini artırmaktadır.
- **İleri yaş:** Yaş arttıkça insülin duyarlılığı azalmakta, diyabet riski artmaktadır.
- **Obezite:** Vücut kitle indeksinin (BMI) 30'un üzerinde olması, özellikle Tip 2 Diyabet için güçlü bir risk faktörüdür.

- **Fiziksel aktivite eksikliği:** Hareketsiz yaşam tarzı, glikoz metabolizmasını olumsuz etkileyerek insülin direncini tetiklemektedir.
- **Yüksek tansiyon ve dislipidemi:** Kan basıncındaki bozukluklar ve kan yağlarının dengesizliği, diyabetle birlikte sıklıkla görülmekte ve hastalığın komplikasyonlarını artırmaktadır.
- **Gebelik diyabeti öyküsü:** Gebelik sırasında diyabet tanısı almış bireylerde, ilerleyen dönemlerde Tip 2 Diyabet gelişme olasılığı daha yüksektir [6].

Bu risk faktörlerinin erken dönemde tespit edilmesi ve bireylerin sağlık takibine alınması, yalnızca bireysel sağlık açısından değil, aynı zamanda toplum sağlığının korunması ve sağlık harcamalarının azaltılması bakımından da kritik öneme sahiptir. Bu noktada, sağlık verilerinin dijital ortamda işlenmesi, büyük veri analitiği ve makine öğrenmesi teknikleri ile bu risk faktörlerinin daha etkin şekilde değerlendirilebilmesi mümkün hale gelmiştir.

Bu tez çalışmasında kullanılan Pima Indians Diabetes veri seti, yukarıda belirtilen risk faktörlerinden birçoğunu içermektedir. Gebelik sayısı, glikoz seviyesi, kan basıncı, cilt kalınlığı, insülin düzeyi, vücut kitle indeksi, genetik yatkınlık ve yaş gibi değişkenleri içeren bu veri seti, özellikle Tip 2 diyabetin teşhisine yönelik olarak dünya genelinde sayısız çalışmada referans veri kümesi olarak kullanılmıştır [3]. Dolayısıyla, bu veri setinin kullanılması, yalnızca akademik literatürle uyum sağlamakla kalmayıp, aynı zamanda gerçek dünyadaki klinik verilere yakın bir modelleme imkânı sunmaktadır.

2.2. Sağlık Bilişiminde Veri Analitiği

Günümüz sağlık sistemleri, artan nüfus, kronik hastalıkların yaygınlığı ve kaynak yetersizlikleri nedeniyle daha etkin ve verimli çözümlere ihtiyaç duymaktadır. Bu bağlamda, sağlık bilişimi (health informatics), tıbbi verilerin dijital ortamda toplanması, işlenmesi ve analiz edilmesi yoluyla, sağlık hizmetlerinin kalitesini artırmayı hedefleyen disiplinler arası bir çalışma alanıdır [7]. Sağlık bilişimi, sadece hasta kayıtlarının dijitalleştirilmesi değil; aynı zamanda bu veriler üzerinden karar destek sistemlerinin, tahmin modellerinin ve erken uyarı sistemlerinin kurulmasına da olanak tanımaktadır.

Veri analitiği, sağlık bilişiminin en temel bileşenlerinden biridir. Sağlık alanında toplanan büyük ve çeşitli verilerin (hastalık geçmişi, laboratuvar sonuçları, genetik bilgiler, yaşam tarzı verileri vb.) doğru analiz edilmesi, daha hızlı teşhis konulmasını,

hastalıkların seyrinin izlenmesini ve önleyici tedbirlerin alınmasını mümkün kılmaktadır. Özellikle kronik hastalıkların (örneğin diyabet, hipertansiyon, kalp hastalıkları) yönetiminde veri analitiği kritik bir rol oynamaktadır [8].

Veri analitiği süreçleri genellikle şu adımlardan oluşur:

- **Veri Toplama:** Hasta bilgilerinin dijital sistemler aracılığıyla kaydedilmesi (HIS, EHR vb.).
- **Veri Temizleme ve Önişleme:** Eksik, tutarsız veya anlamsız verilerin tespiti ve düzeltilmesi.
- **Veri Dönüştürme:** Gerekli formatlara ve yapılara dönüştürülerek analiz için uygun hale getirme.
- **Modelleme ve Tahminleme:** Makine öğrenmesi veya istatistiksel yöntemlerle desenlerin ortaya çıkarılması.
- **Karar Destek:** Analiz sonuçlarının klinik karar süreçlerine entegre edilmesi.

Son yıllarda gelişen **gerçek zamanlı veri işleme** yaklaşımları, sağlık analitiğinde önemli bir paradigma değişimi yaratmıştır. Gerçek zamanlı veriler sayesinde hasta bilgileri anlık olarak izlenebilmekte, riskli durumlara anında müdahale edilebilmektedir. Örneğin; IoT tabanlı sensörler ile kalp ritmi, glikoz düzeyi gibi ölçümler sürekli izlenebilir hale gelmiş, bu veriler **Apache Kafka gibi veri akış platformları** üzerinden analiz edilerek karar destek sistemlerine entegre edilebilmektedir [9]. Bu tür mimariler, hem bireysel düzeyde erken teşhisi kolaylaştırmakta hem de kurumsal düzeyde sağlık yönetim sistemlerinin proaktif hale gelmesini sağlamaktadır.

Bu tez çalışmasında geliştirilen sistem de benzer bir yaklaşımla tasarlanmıştır. Önişlemeden geçirilmiş ve yapay olarak genişletilmiş hasta verileri, **Kaba Kümeleme Teorisi** ile analiz edilerek anlamlı kurallar ve gereklilikler ortaya konmuş; ardından **Apache Kafka, Zookeeper** ve **Java tabanlı** algoritmalar kullanılarak veriler gerçek zamanlı işlenmiş ve teşhis çıktıları elde edilmiştir. Böylelikle, sağlık bilişimi alanında veri analitiği ve gerçek zamanlı karar destek sistemlerinin entegrasyonu başarıyla sağlanmıştır.

2.3. Kaba Kümeleme Teorisi (Rough Set Theory) Üzerine Çalışmalar

Kaba Kümeleme Teorisi (Rough Set Theory, RST), 1982 yılında Zdzisław Pawlak tarafından önerilen ve özellikle belirsizlik, eksik bilgi ve yaklaşık tanımlamalar içeren sistemlerin analizinde kullanılan matematiksel bir yöntemdir [10]. Bu teori, bir evrendeki öğeleri, onların niteliklerine (attributes) göre sınıflandırmakta ve kesin olmayan verilerle çalışabilme kabiliyeti sayesinde özellikle yapay zeka ve veri madenciliği gibi alanlarda önemli bir yer edinmiştir.

Kaba Kümeleme Teorisi'nin temel felsefesi, tam olarak tanımlanamayan kavramları yaklaşık olarak tanımlamak ve bilgi sistemlerinde bu belirsizliklerle başa çıkabilmektir. Bir örnek kümesinin tam olarak hangi sınıfa ait olduğunu belirlemek mümkün değilse, bu küme için **alt yaklaşım (lower approximation)** ve **üst yaklaşım (upper approximation)** kümeleri tanımlanır. Alt yaklaşım, yalnızca o sınıfa ait olduğu kesin olarak bilinen öğeleri içerirken; üst yaklaşım, o sınıfa ait olabilecek tüm öğeleri kapsar. Bu yapı sayesinde, **pozitif bölge**, **negatif bölge** ve **sınır bölgesi** gibi kavramlar yardımıyla bilgi sistemleri daha ayrıntılı analiz edilebilir [11].

Literatürde, Kaba Kümeleme Teorisi'nin farklı alanlardaki kullanımlarına dair pek çok çalışma mevcuttur. Özellikle sağlık bilişimi alanında, hasta verilerinin sınıflandırılması ve teşhis kurallarının elde edilmesi için RST sıkça tercih edilmiştir. Örneğin Taşcı ve Yıldırım [12], uçuş verilerinin RST ile analizini gerçekleştirerek verinin sadeleştirilmesi ve karar kurallarının çıkarılmasına yönelik başarılı sonuçlar elde etmişlerdir. Telçeken ve Doğan [2] ise büyük veri kümeleri üzerinde RST yardımıyla gruplama ve öznitelik indirgeme çalışmaları yürütmüş, özellikle anlamlı özelliklerin belirlenmesinde Kaba Kümeleme'nin etkinliğini vurgulamışlardır.

Ancak RST'nin uygulama alanlarında bazı zorluklar da mevcuttur. Özellikle sürekli (sayısal) değişkenler içeren veri setlerinde RST doğrudan çalışmadığından, verilerin önce kategorik hale dönüştürülmesi gerekmektedir. Bu dönüşüm sürecinde doğru aralıkların ve sınıfların seçilmesi, sonuçların kalitesini doğrudan etkilemektedir. Bu nedenle çalışmalarda, genellikle **discretization (sınıflama)** yöntemleriyle veriler önce uygun kategorilere ayrılmakta, ardından RST uygulanmaktadır.

Bu tez çalışmasında da, önışleme süreci tamamlanan ve sentetik olarak genişletilen veri seti önce kategorik hale dönüştürülmüş, ardından C# programlama dili ile geliştirilen özel bir arayüz yazılımı aracılığıyla Kaba Kümeleme Teorisi

uygulanmıştır. Bu süreç sonucunda, tüm özneliklerin diyabet teşhisi açısından anlamlı olduğu belirlenmiş, karar kuralları başarıyla çıkarılmış ve sistemin şeffaflığı artırılmıştır.

2.4. Gerçek Zamanlı Veri İşleme Araçları Üzerine Çalışmalar

Veri hacmindeki hızlı artış ve kararların anlık olarak alınma gereksinimi, geleneksel toplu veri işleme yaklaşımlarının yerini gerçek zamanlı veri işleme sistemlerine bırakmasına neden olmuştur. Gerçek zamanlı veri işleme, özellikle sağlık, finans, e-ticaret ve nesnelerin interneti (IoT) gibi alanlarda yaşamsal kararların hızlı bir şekilde alınabilmesi için kritik öneme sahiptir. Bu doğrultuda, akan verilerin düşük gecikmeyle, güvenilir biçimde ve ölçeklenebilir sistemlerle işlenmesi gerekmektedir [13].

Bu alanda öne çıkan teknolojilerden biri olan Apache Kafka, yüksek performanslı ve dağıtık bir mesaj kuyruğu sistemidir. Kafka, üretici (producer) ve tüketici (consumer) kavramlarıyla veri akışını yöneten, düşük gecikme süresiyle yüksek hacimli verileri işleyebilen bir yapıya sahiptir. Verileri sıralı ve güvenli şekilde taşıyarak, sistem bileşenleri arasında tutarlı iletişim kurulmasına olanak sağlar [14]. Kafka'nın yüksek hata toleransı ve ölçeklenebilir yapısı, gerçek zamanlı veri akışı için birçok endüstriyel sistemde tercih edilmesine neden olmaktadır.

Bu alanda öne çıkan teknolojilerden biri olan **Apache Kafka**, yüksek performanslı ve dağıtık bir mesaj kuyruğu sistemidir. Kafka, üretici (producer) ve tüketici (consumer) kavramlarıyla veri akışını yöneten, düşük gecikme süresiyle yüksek hacimli verileri işleyebilen bir yapıya sahiptir. Verileri sıralı ve güvenli şekilde taşıyarak, sistem bileşenleri arasında tutarlı iletişim kurulmasına olanak sağlar [14].

Kafka mimarisinin sürdürülebilir çalışmasını sağlamak adına, sistem bileşenleri arasındaki koordinasyonu yöneten **Apache Zookeeper** da önemli bir rol üstlenmektedir. Zookeeper, Kafka brokerlarının yönetimi, veri replikasyonu ve topic-partition eşleştirmeleri gibi görevleri üstlenerek altyapının kararlı biçimde çalışmasını sağlar.

Gerçek zamanlı veri işleme sistemlerinin sağlık alanındaki katkıları, özellikle hasta güvenliği, kişiselleştirilmiş tedavi ve karar destek sistemlerinin doğruluğu açısından önemlidir. Örneğin, glikoz düzeyi, kan basıncı ya da EKG sinyalleri gibi biyomedikal

veriler anlık olarak izlenebilir ve kritik durumlarda sistem tarafından sağlık personeline otomatik uyarılar gönderilebilir. Bu tür yapıların temel amacı; müdahale süresini azaltmak, hızlı karar desteği sağlamak ve hasta yaşam kalitesini artırmaktır [15].

Bu tez çalışmasında geliştirilen mimari, Apache Kafka ve Zookeeper temelli dağıtık bir altyapı üzerine inşa edilmiştir. Laborant tarafından C# ile geliştirilen kullanıcı arayüzü üzerinden girilen hasta verileri, doğrudan Kafka Producer bileşeni aracılığıyla sistemin veri akışına iletilmektedir. Veriler, Kafka'nın FIFO (ilk giren ilk çıkar) prensibi doğrultusunda güvenli ve sıralı bir şekilde Java ortamında çalışan Kafka Consumer bileşeni tarafından anlık olarak dinlenmekte ve işlenmektedir. Bu event-driven mimari sayesinde, veri akışı kesintisiz ve yüksek performansla sağlanmakta; aynı zamanda gerçek zamanlı analizler otomatik olarak tetiklenmektedir. İşlenen tahmin sonuçları, Kafka Consumer tarafından doğrudan MSSQL veritabanına kaydedilmekte olup, ara dosya veya manuel müdahale gerektiren tamponlama mekanizmalarına ihtiyaç duyulmamaktadır. Bu yapı, sağlık alanında kritik olan düşük gecikme süresi, tutarlılık ve yüksek erişilebilirlik gibi gereksinimleri karşılayarak, modern ve sürdürülebilir bir gerçek zamanlı veri işleme platformu sunmaktadır.

Bu kapsamda geliştirilen mimari, gerçek zamanlı veri akışını uçtan uca yöneterek düşük gecikmeli, tutarlı ve esnek bir çözüm sunmaktadır. Sağlık gibi zaman hassasiyetinin yüksek olduğu kritik alanlarda, bu yapı hızlı ve güvenilir veri analitiği ile karar destek süreçlerinin etkin bir şekilde yürütülmesine olanak tanımakta; böylece hasta güvenliği ve tedavi kalitesinin artırılmasına doğrudan katkı sağlamaktadır.

2.5. Makine Öğrenmesi Yöntemleriyle Diyabet Teşhisi

Makine öğrenmesi (ML), son yıllarda veri madenciliği ve yapay zekâ alanlarında önemli ilerlemeler sağlayarak özellikle sağlık hizmetlerinde etkin bir araç haline gelmiştir. Diyabet gibi kronik hastalıkların erken teşhisi açısından, makine öğrenmesi modelleri karmaşık veriler içerisinden anlamlı örüntüler çıkartabilmesi sayesinde, geleneksel istatistiksel yaklaşımlara göre daha doğru ve esnek teşhis olanakları sunmaktadır [16].

Sınıflandırma problemleri kapsamında, bireylerin diyabet hastası olup olmadığını belirlemeye yönelik olarak birçok farklı makine öğrenmesi algoritması kullanılmaktadır. Bu algoritmalar arasında Naive Bayes, K-En Yakın Komşu (KNN),

Karar Ağaçları, Lojistik Regresyon, Destek Vektör Makineleri (SVM), Random Forest, AdaBoost ve topluluk (ensemble) modelleri ön plana çıkmaktadır [17][18]. Her bir algoritma kendi avantaj ve sınırlılıklarına sahip olsa da, ortak amaç doğrultusunda bireylerin tıbbi verileri üzerinden doğru sınıflandırmalar gerçekleştirmeye çalışırlar.

Diyabet teşhisine yönelik yapılan çalışmalarda sıklıkla kullanılan Pima Indians Diabetes veri seti, makine öğrenmesi yöntemlerinin karşılaştırmalı performans analizleri için bir benchmark görevi görmektedir. Literatürdeki çeşitli çalışmalarda, KNN gibi örüntü tanıma temelli yaklaşımların yüksek doğruluk oranları sunduğu, ensemble modellerin ise genelleme başarısını artırdığı rapor edilmiştir [18][19]. Ayrıca, algoritma performanslarının doğrudan veri ön işleme sürecine bağlı olduğu bilinmektedir. Özellikle eksik veri düzenlemesi, uç değer yönetimi, veri normalizasyonu ve sınıf dengesizliğinin giderilmesi gibi adımlar, model doğruluğunu önemli ölçüde etkilemektedir.

Tez kapsamında geliştirilen sistemde, veri ön işleme sürecinin ardından Java dili kullanılarak Naive Bayes, KNN, KNC, AdaBoost, Lojistik Regresyon, Ağırlıklı Ensemble ve Çoğunluk Ensemble algoritmaları kodlanmıştır. Apache Kafka üzerinden gelen hasta verileri, gerçek zamanlı olarak bu modellere iletilmiş ve elde edilen teşhis çıktıları sistemin diğer bileşenlerine entegre edilmiştir. Bu yaklaşım sayesinde yalnızca model doğruluğu değil, aynı zamanda gerçek zamanlı çalışabilirlik düzeyleri de değerlendirilmiştir.



3. VERİ SETİ ÖN İŞLEME SÜRECİ

Bu bölümde, çalışmada kullanılan Pima Indians Diabetes veri setinin yapısı açıklanmakta; eksik değer kontrolü, anlamsız değerlerin giderilmesi, uç değer analizi, sınıf dengesizliğinin giderilmesi ve sentetik veri üretimi gibi kapsamlı ön işleme adımları detaylı olarak ele alınmaktadır. Ayrıca, Kaba Kümeleme Teorisi'ne geçiş için gerekli olan kategorik dönüşüm süreci de açıklanacaktır.

3.1. Diyabet Hastalığı ve Risk Faktörleri

Bu çalışmada kullanılan veri seti, ABD'de yaşayan Pima Kızılderili kadınlarına ait tıbbi verilere dayanan Pima Indians Diabetes Database olup, literatürde diyabet teşhisine yönelik en yaygın kullanılan veri setlerinden biridir. Özellikle bireylerin glikoz düzeyleri, insülin değerleri, gebelik sayıları ve yaş gibi risk faktörlerini barındırması, bu veri setini sağlık bilişimi alanında referans bir kaynak hâline getirmiştir [1].

Veri seti toplamda 768 gözlem (satır) ve 9 sütundan oluşmaktadır. Sekiz sütun girdi (öz nitelik), bir sütun ise hedef (Outcome) değişkenidir. Tablo 3.1'de veri setinde yer alan öznitelikler ve bu özniteliklerin kısa açıklamaları verilmiştir.

Tablo 3.1. Pima Indians Veri Setindeki İlk Beş Kayıt.

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree Function	Age	Outcome
0	135.25	66.44	49.05	172.52	16.76	0.19	62	0
1	90.0	62.0	12.0	43.0	27.2	0.58	24	0
0	188.33	72.77	16.93	159.86	28.15	0.44	24	1
9	110.13	82.26	36.08	93.25	42.05	1.01	44	1
6	187.65	75.07	26.06	124.25	25.42	1.03	52	1

Veri setindeki her bir deęişken için bir analiz yapılmıř ve Tablo 3.2.'de sunulmaktadır. Bu tablo ve örnek kayıtlar, veri yapısının anlaşılmasını ve sonraki öniřleme süreci için temel bir referans nitelięi taşımaktadır.

Tablo 3.2. Pima Indians Veri Seti Deęişken Analizi

Deęişken	Açıklama
Pregnancies	Hastanın hamilelik sayısı
Glucose	Plazma glikoz konsantrasyonu
BloodPressure	Diyastolik kan basıncı (mm Hg)
SkinThickness	Triseps deri kalınlıęı (mm)
Insulin	İnsülin seviyesi (mu U/ml)
BMI	Vücut kitle indeksi (kg/m ²)
DiabetesPedigreeFunction	Ailede diyabet geçmiři oranı
Age	Hastanın yaşı
Outcome	Diyabet olup olmadıęını belirten deęişken (0 saęlıklı; 1 diyabet hastası)

Bu veri kümesinin, hem deęişken çeřitlilięi hem de sınıflandırma problemi içermesi nedeniyle birçok akademik çalışmada tercih edildięi görölmektedir. Özellikle “Outcome” sütunu, hedef deęişken olarak kullanılarak hastaların diyabet olup olmadıęının tahmini amacıyla deęerlendirilmekte ve veri madencilięi algoritmaları bu amaç doęrultusunda eęitilmektedir.

3.2. Eksik ve Aykırı Deęerlerin Analizi

Veri madencilięi uygulamalarında model başarısını belirleyen en kritik adımlardan biri, verinin **temizlięi** ve **kalitesidir**. Bu doęrultuda, çalışmada kullanılan Pima Indians Diabetes veri seti üzerinde kapsamlı bir öniřleme adımı gerçekleştirilmiřtir. Özellikle **eksik ve uç (aykırı) deęerlerin tespiti ve düzenlenmesi**, bu sürecin temelini oluřturmuřtur.

Veri setinde doğrudan NaN veya boş hücre bulunmamakla birlikte, **sıfır (0)** değeri içeren bazı özelliklerin tıbbi anlamda geçersiz olduğu belirlenmiştir. Örneğin; **Glucose, BloodPressure, SkinThickness, Insulin** ve **BMI** gibi biyolojik ölçümlerin sıfır olması mümkün olmadığından, bu değerler **eksik veri** olarak ele alınmıştır [20]. Veri ön işleme öncesi veri setindeki sıfır değerlerine ait dağılımlar Şekil 3.1.'de gösterilmiştir.

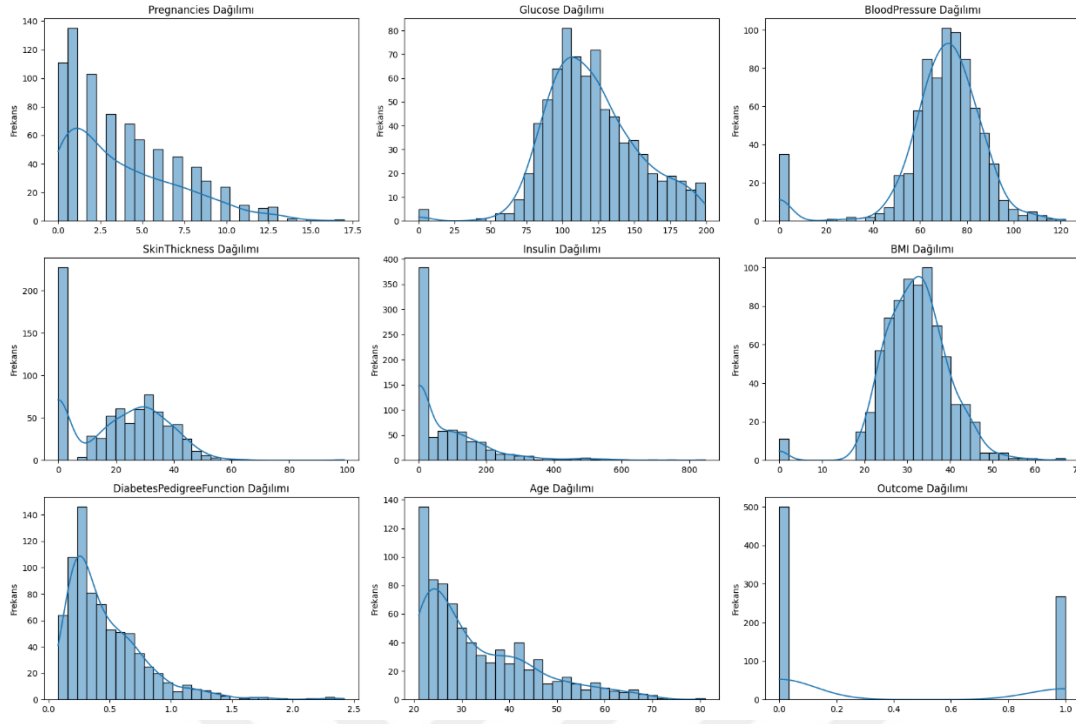
	0
Pregnancies	111
Glucose	5
BloodPressure	35
SkinThickness	227
Insulin	374
BMI	11
DiabetesPedigreeFunction	0
Age	0
Outcome	500

Şekil 3.1. Veri ön işleme öncesi sıfır değerlerinin dağılımı.

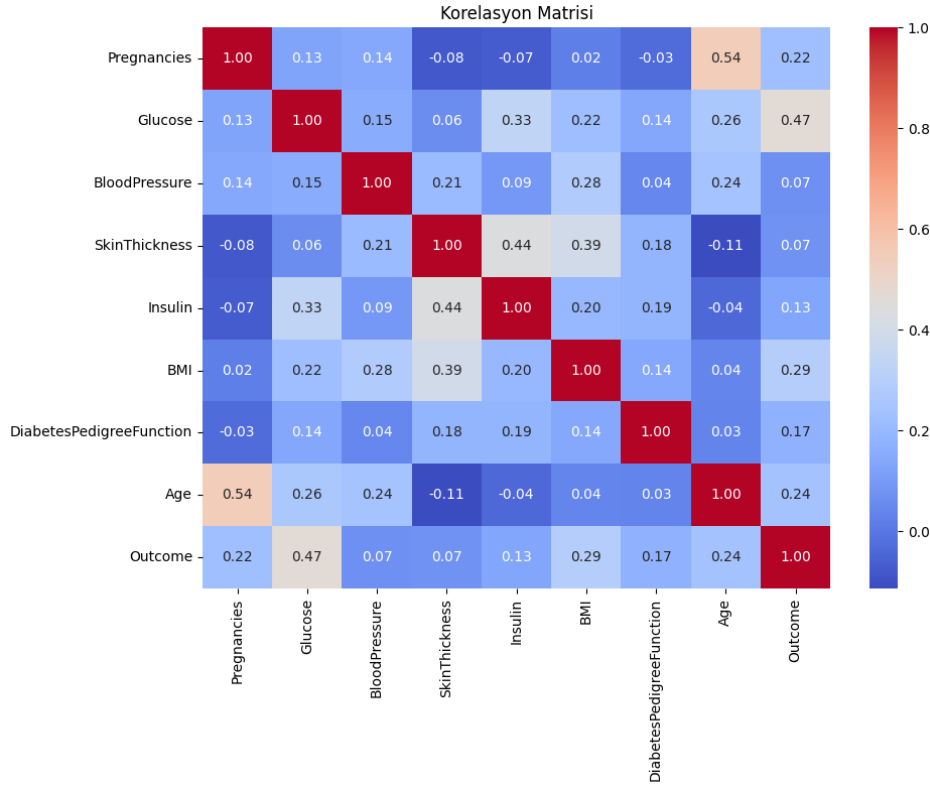
Veri setinin ilk haline ait değişken istatistikleri Tablo 3.3.'de, histogram grafikleri Şekil 3.2.'de ve Korelasyon Matrisi Şekil 3.3.'de gösterilmiştir.

Tablo 3.3. Eksik ve Aykırı Değer Analizi Öncesi Veri Setinin İstatistiksel Analizi.

	Kayıt Sayısı	Ortalama	Standart Sapma	En Düşük	1.Çeyrek	Medyan	3.Çeyrek	En Yüksek	Missing Values
Pregnancies	768.0	3.845	3.369	0.000	1.000	3.000	6.000	17.00	0
Glucose	768.0	120.894	31.972	0.000	99.000	117.000	140.250	199.00	0
BloodPressure	768.0	69.105	19.355	0.000	62.000	72.000	80.000	122.00	0
SkinThickness	768.0	20.536	15.952	0.000	0.000	23.000	32.000	99.00	0
Insulin	768.0	79.799	115.244	0.000	0.000	30.500	127.250	846.00	0
BMI	768.0	31.992	7.884	0.000	27.300	32.000	36.600	67.10	0
DiabetesPedigreeFunction	768.0	0.471	0.331	0.078	0.243	0.372	0.626	2.42	0
Age	768.0	33.240	11.760	21.000	24.000	29.000	41.000	81.00	0
Outcome	768.0	0.348	0.476	0.000	0.000	0.000	1.000	1.00	0



Şekil 3.2. Eksik ve Aykırı Değer Analizi öncesi veri setinin histogram grafiği.



Şekil 3.3. Eksik ve Aykırı Değer Analizi öncesi veri setinin korelasyon matrisi.

Bu tür eksik veriler, silinmek yerine istatistiksel yöntemlerle yeniden oluşturulmuştur. Literatürde yaygın olarak kullanılan **çeyreklik değerlerine dayalı IQR**

(**Interquartile Range**) yaklaşımı benimsenmiştir. Her bir değişken için Q1 ve Q3 değerleri hesaplanmış, bu değerler kullanılarak aykırı değer sınırları Şekil 3.4.’de ifade edilen formül yardımıyla elde edilmiştir:

$$\text{Alt sınır} = Q1 - 1.5 \times IQR,$$

$$\text{Üst sınır} = Q3 + 1.5 \times IQR$$

Şekil 3.4. IQR Yaklaşımı; Alt sınır ve Üst sınır formülü.

Bu sınırların dışına çıkan veriler, veri dağılımını bozmayacak şekilde **Q1-Q3 aralığında rastgele** değerlerle değiştirilmiştir. Böylece hem eksik hem de aykırı değerler, verinin doğal yapısına zarar vermeyecek şekilde yeniden düzenlenmiştir. Şekil 3.5’te ilk ön işleme sonucu olarak sıfır değerlerinin dağılımı ifade edilmiştir. Burada görüldüğü üzere Outcome ve Pregnancies değişkenleri dışındaki değişkenlerin sıfır değerine sahip olması yanlış olacağından uygun bir şekilde veri seti düzenlenmiştir.

	0
Pregnancies	111
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	500

Şekil 3.5. Veri ön işleme sonucu sıfır değerlerinin dağılımı.

Bu işlem için Python’un açık kaynaklı kütüphanelerinden **Pandas** ve **NumPy** kullanılmıştır. Ayrıca, Scikit-learn dokümantasyonunda önerilen stratejiler referans alınarak, eksik veri doldurma ve aykırı değer yönetimi birlikte yürütülmüştür [21]. Veri setindeki aykırı (uç değer) temizleme işlemine ait kod bloğu Şekil 3.6’da gösterilmiştir.

```

import numpy as np

# Uç değ r kontrol  yapılacak s tunlar (Outcome hari )
columns_to_check = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
                    'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']

for col in columns_to_check:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Sınırlar dıřında kalanları Q1-Q3 aralıęında rastgele değ rlerle deęiřtir
    mask_upper = df[col] > upper_bound
    mask_lower = df[col] < lower_bound

    df.loc[mask_upper, col] = np.random.uniform(Q1, Q3, size=mask_upper.sum())
    df.loc[mask_lower, col] = np.random.uniform(Q1, Q3, size=mask_lower.sum())

df.describe()

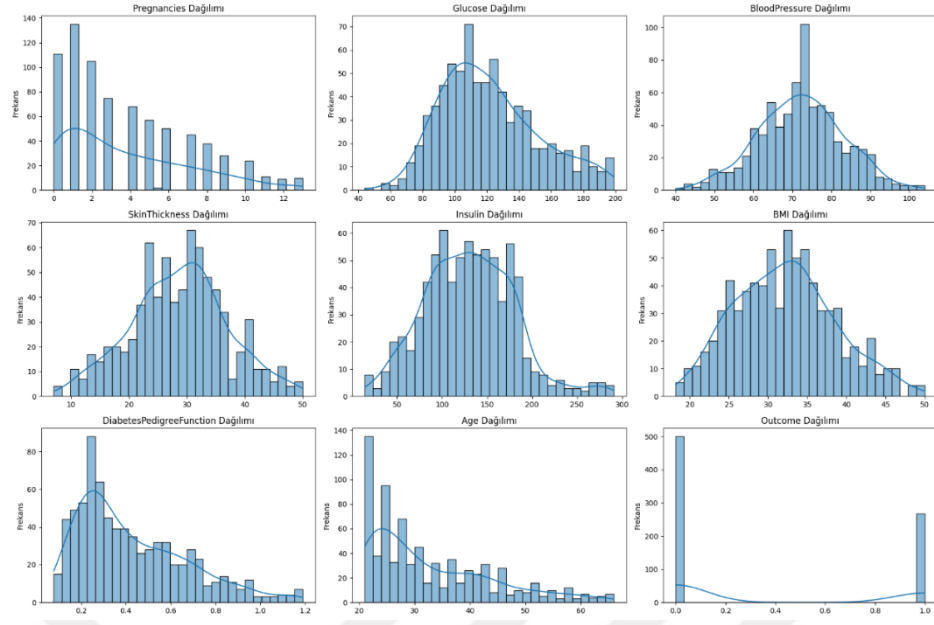
```

řekil 3.6. U  değ r temizlięi python kodu.

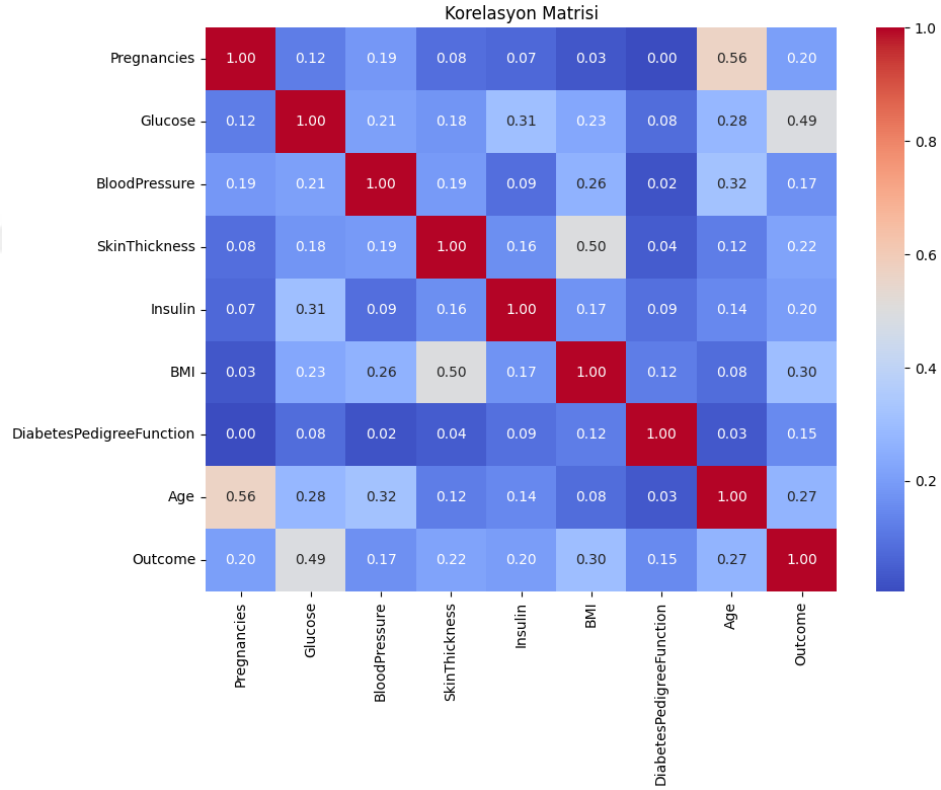
Tablo 3.4.'de eksik ve aykırı değ rlerin analizi sonucunda elde edilen veri setinin istatistiksel tablosuna, řekil 3.7.'de analiz sonucu elde edilen veri setinin histogram grafiklerine ve řekil 3.8.'de korelasyon matrisine yer verilmiřtir

Tablo 3.4. Eksik ve Aykırı Deęer Analizi Sonucu Oluřan Veri setinin İstatistiksel Analizi.

	Kayıt Sayısı	Ortalama	Standart Sapma	En D�ř�k	1.�eyrek	Medyan	3.�eyrek	En Y�ksek	Missing Values
Pregnancies	768.0	3.786	3.272	0.000	1.000	3.000	6.000	13.00	0
Glucose	768.0	121.663	30.449	44.000	99.700	117.000	140.250	199.00	0
BloodPressure	768.0	72.130	10.944	40.000	64.000	72.000	80.000	104.00	0
SkinThickness	768.0	28.696	8.310	7.000	23.000	29.000	33.943	50.00	0
Insulin	768.0	130.257	49.547	14.000	94.000	129.250	164.389	291.00	0
BMI	768.0	32.196	6.422	18.200	27.500	32.144	36.300	50.00	0
DiabetesPedigree Function	768.0	0.429	0.246	0.078	0.243	0.361	0.586	1.191	0
Age	768.0	32.805	11.061	21.000	24.000	29.000	40.000	66.00	0
Outcome	768.0	0.348	0.476	0.000	0.000	0.000	1.000	1.00	0



Şekil 3.7. Eksik ve Aykırı Değer Analizi sonrası oluşan veri setinin histogram grafiği.



Şekil 3.8. Eksik ve Aykırı Değer Analizi sonrası oluşan veri setinin korelasyon matrisi.

Bu çalışma kapsamında oluşturulan korelasyon matrisi, değişkenler arasındaki doğrusal ilişkinin gücünü ve yönünü incelemek için kullanılmıştır. Korelasyon analizi, özellikle bazı değişkenlerin diğerlerine olan etkisini değerlendirme ve potansiyel multicollinearity durumlarını tespit etme açısından önemlidir. Analiz sonucunda,

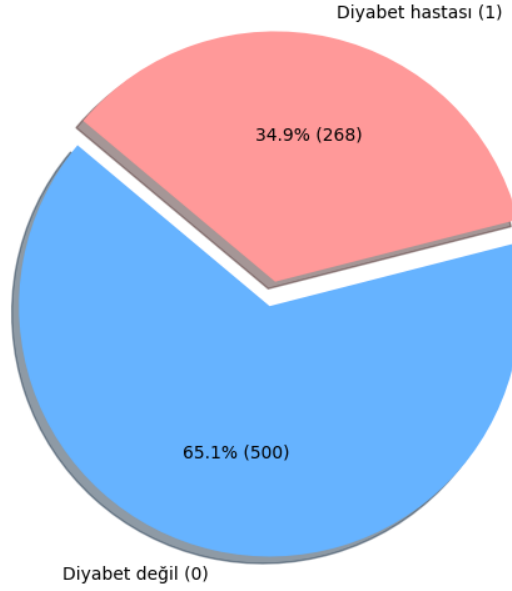
glikoz düzeyi ile diyabet sonucu arasında yüksek pozitif bir ilişki gözlemlenmiştir; bu durum, glikozun diyabet tahmininde belirleyici bir değişken olduğunu desteklemektedir. Korelasyon matrisi, veri analitiği süreçlerinde öznelitliklerin ilişkilerini kavramada güçlü bir istatistiksel araç olarak değerdendirilmektedir [22].

3.3. SMOTE ile Sınıf Dengesinin Sağlanması

Makine öğrenmesi algoritmaları, eğitim aşamasında verinin dengeli dağılımına oldukça duyarlıdır. Dengesiz veri setlerinde, özellikle azınlık sınıfına ait örneklerin yetersizliği nedeniyle model, çoğunluk sınıfı öğrenmeye yatkın hale gelir. Bu durum, modelin genelleme kabiliyetini düşürmekte ve özellikle azınlık sınıfa ait örneklerin doğru tahmin edilememesine neden olmaktadır. Bu problem “**sınıf dengesizliği**” (class imbalance) olarak tanımlanır ve sağlık verisi analizlerinde yaygın şekilde karşılaşılır.

Bu kapsamda SMOTE, özellikle tıbbi veri setlerinde sık karşılaşılan sınıf dengesizliği problemini azaltmak için yaygın biçimde kullanılan etkili bir yöntemdir. Literatürde de bu tekniğin uygulanabilirliğini gösteren çeşitli çalışmalar bulunmaktadır. Örneğin, Ebuka Eluzai Uduma tarafından gerçekleştirilen yüksek lisans tezinde SMOTE yöntemi kullanılarak Pima Indian Diabetes veri setinin dengelenmesi sağlanmıştır [23]. Benzer şekilde, Maganti Khyathi Sarayu ve çalışma arkadaşları da aynı veri setinde yapılan tahmin çalışmalarında özellik ölçeklendirme, tabakalı örnekleme ve SMOTE tabanlı aşırı örnekleme gibi veri hazırlama adımlarının sınıf dengesizliğini azaltmada kritik bir rol oynadığını göstermiştir [24].

Pima Indians Diabetes veri seti incelendiğinde, 500'ü sağlıklı (Outcome=0) ve 268'i diyabetli (Outcome=1) olmak üzere toplam 768 örnekten oluştuğu gözlemlenmiştir. Veri setinde hasta olmayan bireylerin oranı %65 civarındayken, hasta bireylerin oranı %35 seviyesindedir. Bu dengesizlik, sınıflandırma algoritmalarının diyabetli bireyleri tespit etmesini güçleştirmekte ve sistemin taraflı öğrenmesine neden olmaktadır. Şekil 3.9.'da dengesizlik açık bir şekilde görülmektedir.



Şekil 3.9. SMOTE öncesi Diyabet durum dağılımı.

Bu problemi gidermek amacıyla **SMOTE (Synthetic Minority Over-sampling Technique)** algoritması kullanılmıştır. SMOTE, azınlık sınıf örneklerinden yeni sentetik örnekler üretmeye dayanan bir **aşırı örnekleme** yöntemidir. Temel çalışma prensibi, azınlık sınıfına ait her bir örneğin k -komşuları içerisinde rastgele seçimler yaparak bu komşularla olan fark vektörü doğrultusunda yeni örnekler oluşturmaktır [25]. Bu çalışmada SMOTE algoritması, **varsayılan parametre olan $k=5$ komşuluk değeriyle** uygulanmıştır. Böylece mevcut azınlık sınıf örneklerinin etrafında benzer fakat benzersiz yeni kayıtlar üretilmiştir. Şekil 3.10.'da SMOTE algoritmasının uygulanaşına ait python kodu ve Şekil 3.11.'de SMOTE algoritması sonucunda elde edilen diyabet durum dağılımı grafiği görülmektedir.

```
# SMOTE Tekniği ile VERİSETİ DENGELEME

from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# Bağımlı (etiket) ve bağımsız değişkenleri ayıralım
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# SMOTE uygulayalım
smote = SMOTE(random_state=42) # SMOTE sınıfı varsayılan olarak k_neighbors'u 5 seçer
X_resampled, y_resampled = smote.fit_resample(X, y)

# SMOTE sonrası negatif değerleri kontrol edelim
for col in X_resampled.columns:
    if (X_resampled[col] < 0).any():
        # Q1 - Q3 aralığında rastgele değerlerle negatifleri değiştir
        Q1 = X_resampled[col].quantile(0.25)
        Q3 = X_resampled[col].quantile(0.75)
        random_values = np.random.uniform(Q1, Q3, size=(X_resampled[col] < 0).sum())
        X_resampled.loc[X_resampled[col] < 0, col] = random_values

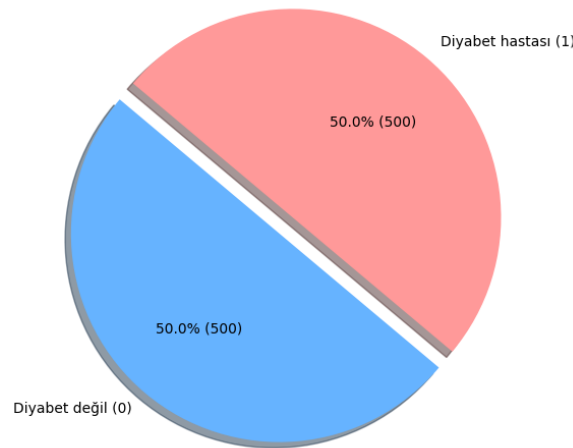
# Yeni veri setinin boyutuna bakalım
print("Orijinal sınıf dağılımı:\n", y.value_counts())
print("\nYeni (SMOTE sonrası) sınıf dağılımı:\n", y_resampled.value_counts())

# Yeni veri setini birleştirelim
df_resampled = pd.concat([X_resampled, y_resampled], axis=1)

Orijinal sınıf dağılımı:
Outcome
0    500
1    268
Name: count, dtype: int64

Yeni (SMOTE sonrası) sınıf dağılımı:
Outcome
1    500
0    500
Name: count, dtype: int64
```

Şekil 3.10. SMOTE algoritması python kodu.



Şekil 3.11. SMOTE sonrası Diyabet durum dağılımı.

Ayrıca SMOTE sonrası negatif değerlere sahip örnekler kontrol edilmiş ve bu tarz değerlerin ilgili sütunlarda tespit edilmesi durumunda, söz konusu negatifler rastgele

fakat mantıklı bir aralık olan Q1 ve Q3 çeyreklikleri arasında üretilen değerlerle değiştirilmiştir. Bu, sentetik verilerin anlamsal tutarlılığını korumak açısından önem arz etmektedir. Şekil 3.9’da ilgili kod bloğu ifade edilmiştir.

Bu uygulama sonucunda diyabetli birey sayısı 500’e çıkarılmış ve veri seti dengelenerek hem hasta hem de sağlıklı sınıf için eşit sayıda örnek içeren, toplamda 1000 gözlemden oluşan yeni bir veri seti elde edilmiştir.

Veri dengesizliğinin SMOTE ile giderilmesi sonucu elde edilen denge, özellikle ileri aşamalarda uygulanacak olan makine öğrenmesi algoritmalarının doğruluğunu ve duyarlılığını artırmıştır. Elde edilen dengeli veri seti, sonraki sentetik veri üretim ve analiz adımları için sağlam bir temel oluşturmuştur.

3.4. CTGAN ile Sentetik Veri Üretimi

Veri analitiği ve makine öğrenmesi çalışmalarında, veri miktarının artırılması hem model başarımı hem de genelleme kabiliyeti açısından kritik öneme sahiptir. Ancak, özellikle sağlık alanındaki veri setleri genellikle sınırlı sayıda gözlem içerdiğinden, gerçek veriye benzer nitelikte **sentetik veri üretimi** yaklaşımlarına ihtiyaç duyulmaktadır. Bu kapsamda, generatif modellerin başında gelen **CTGAN (Conditional Tabular Generative Adversarial Network)** yöntemi, yapay örnekler oluşturarak eğitim veri setini genişletmek amacıyla kullanılan etkin ve modern bir tekniktir [26].

Ayrıca, tıbbi veriler üzerinde GAN tabanlı modellerin uygulanabilirliğini inceleyen çalışmalar da literatürde yer almaktadır. Örneğin, Ebuka Eluzai Uduma tarafından gerçekleştirilen çalışmada, Pima Indian Diabetes veri setindeki sınıf dengesizliği problemi ele alınmış; SMOTE ve GAN tabanlı veri artırma teknikleri kullanılarak dengeli ve daha temsil gücü yüksek bir veri kümesi oluşturulmuştur [23]. Benzer şekilde, Sushma Jaiswal ve Priyanka Gupta tarafından yapılan bir çalışmada da Pima Indian Diabetes veri seti üzerinde çeşitli GAN modelleri kullanılarak sentetik veriler üretilmiş ve bu verilerin hem gerçek veriyle yüksek benzerlik gösterdiği hem de model performansını artırdığı ortaya konmuştur [27].

Bu tez çalışmasında, Pima Indians Diabetes veri setinin **SMOTE ile dengelenmiş hali**, CTGAN modeli ile genişletilmiştir. Veri seti başlangıçta 1000 örnek (500 hasta, 500 sağlıklı birey) içerecek şekilde dengelenmiştir. Ancak, Apache Kafka üzerinden

gerçek zamanlı olarak aktarılabilecek veri hacminin artırılması ve makine öğrenmesi algoritmalarının daha geniş çapta eğitilebilmesi için bu veri setinin büyütülmesi hedeflenmiştir.

CTGAN modeli Python ortamında *sdv.tabular.CTGAN* modülü kullanılarak uygulanmıştır. Geliştirilen uygulamada, hem hasta (Outcome=1) hem de sağlıklı (Outcome=0) bireylere ait örnekler birbirinden ayrılmış ve **her sınıf için ayrı ayrı CTGAN modelleri** eğitilerek sentetik örnekler üretilmiştir. Bu ayırım sayesinde, üretilen verinin sınıf bazında daha tutarlı ve gerçekçi olması sağlanmıştır. Örneğin, sağlıklı bireylerin medikal profiliyle uyumsuz olacak şekilde anormal verilerin üretilmesi engellenmiştir. Şekil 3.12.'de CTGAN modelinin uygulandığı python kodu görülmektedir.

```
# Sentetik veri Üretimi
!pip install sdv

from sdv.single_table.ctgan import CTGAN
import pandas as pd
import numpy as np

# SMOTE sonrası veri setini birleştir (X_resampled ve y_resampled daha önce oluşturulmuş olmalı)
df_resampled = pd.concat([X_resampled, y_resampled], axis=1)

# Her sınıfı ayır
class_0 = df_resampled[df_resampled['Outcome'] == 0]
class_1 = df_resampled[df_resampled['Outcome'] == 1]

# CTGAN modelleri (metadata doğrudan kullanılmıyor, CTGAN kendi algılıyor)
model_0 = CTGAN(epochs=300)
model_1 = CTGAN(epochs=300)

# Eğitim
model_0.fit(class_0)
model_1.fit(class_1)

# Sentetik veri üret (her sınıftan 500)
synthetic_0 = model_0.sample(500)
synthetic_1 = model_1.sample(500)

# Birleştir ve karıştır
synthetic_balanced = pd.concat([synthetic_0, synthetic_1], ignore_index=True)
synthetic_balanced = synthetic_balanced.sample(frac=1, random_state=42).reset_index(drop=True)

# Original veri seti üzerinden referans al (negatif olamayacak sütunlar için)
columns_to_fix_negatives = ['Pregnancies', 'Glucose', 'BloodPressure',
                             'SkinThickness', 'Insulin', 'BMI',
                             'DiabetesPedigreeFunction', 'Age']

# Negatifleri rastgele Q1-Q3 aralığında düzelt
for col in columns_to_fix_negatives:
    Q1_orig = df[col].quantile(0.25)
    Q3_orig = df[col].quantile(0.75)

    negative_indices = synthetic_balanced[synthetic_balanced[col] < 0].index
    random_values = np.random.uniform(Q1_orig, Q3_orig, size=len(negative_indices))

    synthetic_balanced.loc[negative_indices, col] = random_values

# Negatif kontrolü
print("Negatif değer kontrolü (düzeltme sonrası):\n", (synthetic_balanced < 0).sum())

# Sınıf kontrolü
print("\nSınıf dağılımı:\n", synthetic_balanced['Outcome'].value_counts())

# Yeni dosyaya kaydet
synthetic_balanced.to_csv("sentetik_dengeli_veri.csv", index=False)
```

Şekil 3.12. CTGAN modeli uygulama python kodu.

Her sınıf için 500 adet yeni kayıt üretilerek, veri seti toplamda **2000 gözleme** çıkarılmıştır. Bu gözlemler daha sonra birleştirilmiş ve orijinal veri seti ile karşılaştırmalı olarak analiz edilmiştir. Korelasyon yapısı, dağılım grafikleri ve temel istatistiksel ölçümler karşılaştırılarak sentetik verilerin gerçek verilere yüksek oranda benzediği doğrulanmıştır.

```
# Sentetik veri sonucu oluşan veri setinin analizi

import matplotlib.pyplot as plt
import seaborn as sns

# Kaç tane değişken varsa ona göre satır/sütun düzeni oluştur
columns = df.columns
num_vars = len(columns)
rows = (num_vars // 3) + int(num_vars % 3 != 0) # 3 sütunlu grid

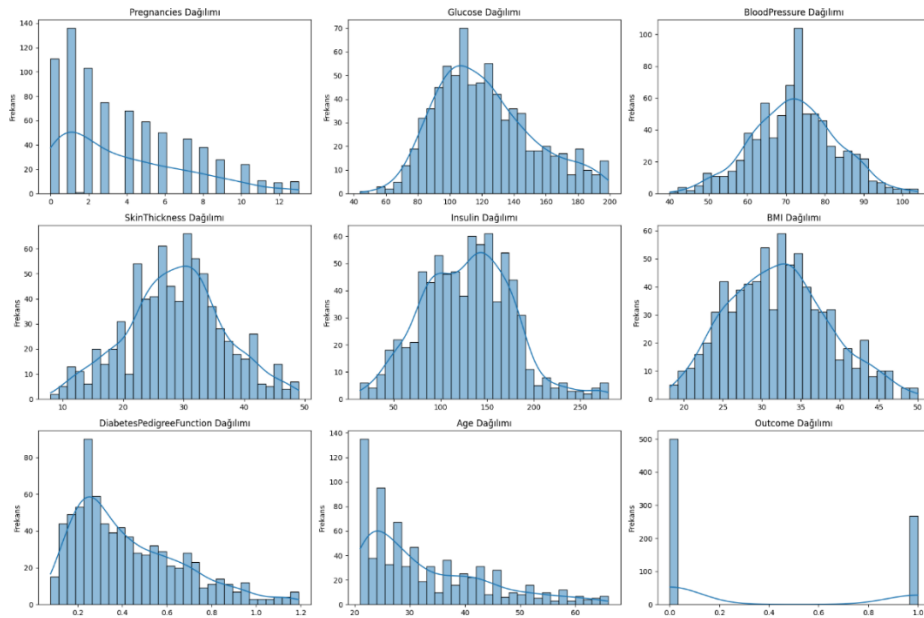
plt.figure(figsize=(18, rows * 4)) # Grafik boyutu dinamik

for idx, col in enumerate(columns):
    plt.subplot(rows, 3, idx + 1)
    sns.histplot(df[col], bins=30, kde=True)
    plt.title(f"{col} Dağılımı")
    plt.xlabel("")
    plt.ylabel("Frekans")

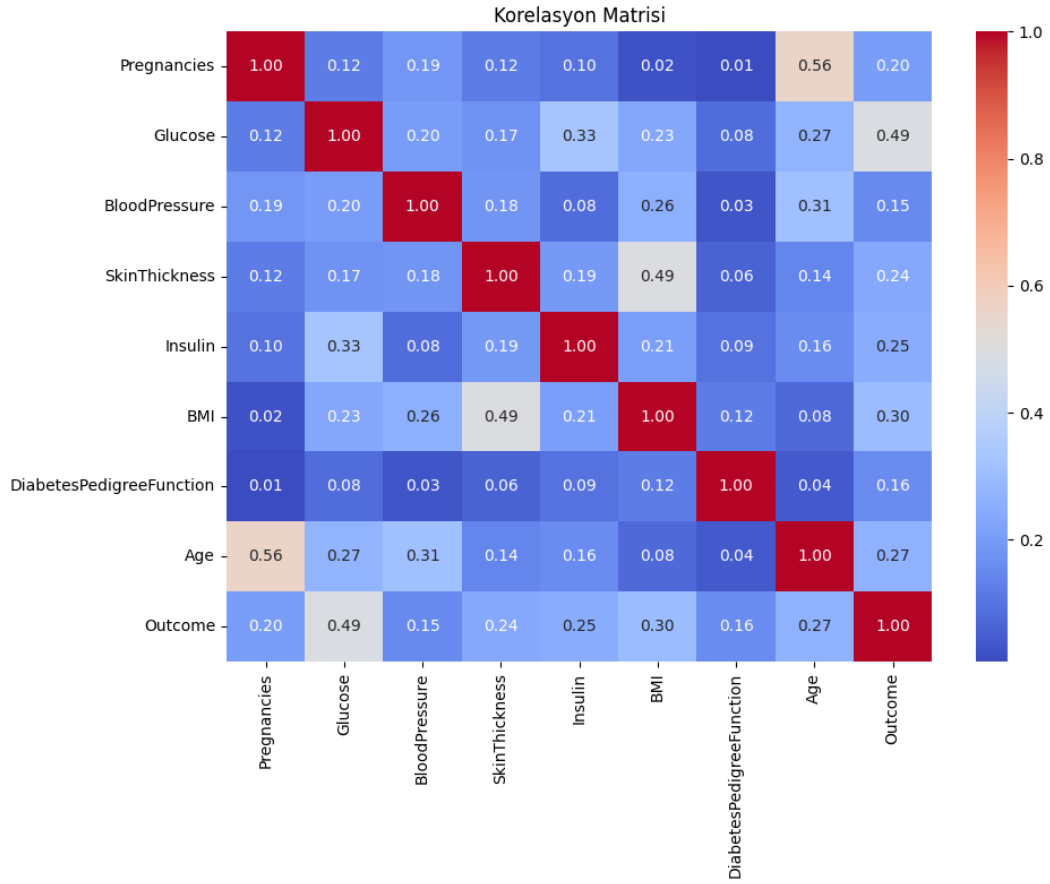
plt.tight_layout()
plt.show()

# Korelasyon matrisi
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Korelasyon Matrisi")
plt.tight_layout()
plt.show()
```

Şekil 3.13. CTGAN sonucu oluşan veri setinin analizini yapan python kodu.



Şekil 3.14. CTGAN sonrası oluşan veri setinin histogram grafiği.



Şekil 3.15. CTGAN sonrası oluşan veri setinin korelasyon matrisi.

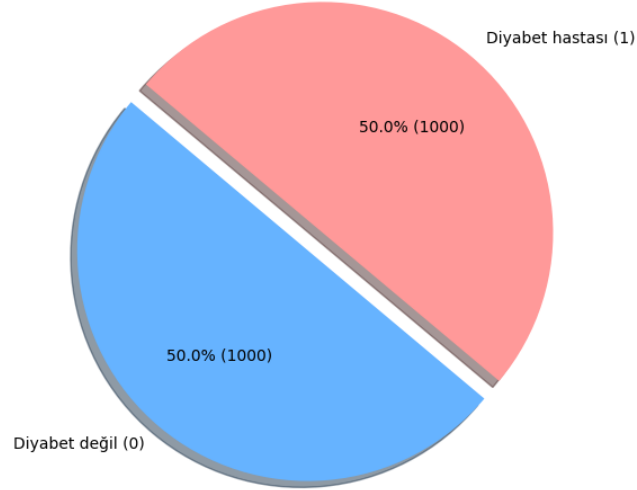
```
import matplotlib.pyplot as plt

# SMOTE sonrası sınıf dağılımı
sizes = df_combined['Outcome'].value_counts().values # [diyabet değil, diyabet]
labels = ['Diyabet değil (0)', 'Diyabet hastası (1)']
colors = ['#66b3ff', '#ff9999']
explode = (0.05, 0.05)

# Yüzde ve sayı birlikte gösterilecek özel fonksiyon
def autopct_format(pct):
    total = sum(sizes)
    count = int(round(pct * total / 100.0))
    return f'{pct:.1f}% ({count})'

# Pasta grafiğini çiz
plt.figure(figsize=(6, 6))
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct=autopct_format, shadow=True, startangle=140)
plt.axis('equal')
plt.show()
```

Şekil 3.16. CTGAN sonrası oluşan veri setinin diyabet dağılım grafiği python kodu.



Şekil 3.17. CTGAN sonrası oluşan veri setinin diyabet dağılımı.

Bu genişletilmiş veri seti, ilerleyen aşamalarda Kaba Kümeleme Teorisi ile karar kuralları çıkarımı ve makine öğrenmesi modellerinin eğitimi için kullanılmıştır. CTGAN modeliyle sağlanan veri zenginliği, sistemin daha fazla sayıda senaryo üzerinde eğitilmesini ve farklı hasta profilleri için daha kararlı tahminlerde bulunmasını mümkün kılmıştır.

3.5. Veri Setinin Eğitim ve Test Olarak Bölünmesi

Ön işleme ve sentetik veri üretimi aşamaları tamamlandıktan sonra, elde edilen 2000 örneklik dengeli veri seti, eğitim ve test süreçleri için ikiye ayrılmıştır. Bu ayrım, **%70 eğitim ve %30 test** oranına göre gerçekleştirilmiştir. Eğitim verisi, toplam 1400 kayıttan oluşmakta olup 700 diyabet hastası ve 700 sağlıklı birey içermektedir. Test verisi ise 600 kayıttan oluşmakta ve benzer şekilde 300 hasta ve 300 sağlıklı bireyi barındıracak şekilde dengelenmiştir.

Bu dengeli ayrım, sınıflar arası dengesizlik kaynaklı önyargıların önüne geçmeyi amaçlamakta ve makine öğrenmesi algoritmalarının daha adil bir değerlendirmeye tabi tutulmasına imkân vermektedir. Gerçek zamanlı sistemin performans testleri, yalnızca bu 600 verilik test kümesi üzerinden gerçekleştirilmiştir. Bu yaklaşım, sistemin gerçek klinik senaryolarda nasıl performans göstereceğini simüle etme açısından büyük önem taşımaktadır.

Veri ayrım sürecindeki denge, algoritmaların hem eğitimde öğrenme kabiliyetlerini artırmakta hem de test sırasında daha güvenilir tahminler üretebilmesini

sağlamaktadır. Bu yapı, sistemin genel başarımı ve klinik geçerliliği açısından kritik bir rol üstlenmektedir.

3.6. Kategorik Dönüşüm Süreci

Kaba Kümeleme Teorisi (Rough Set Theory), nitel (kategorik) veriler üzerinde çalışan bir bilgi keşfi yöntemidir. Bu teori, özellikle değişkenlerin değerlerini ayrık (discrete) biçimde ele alarak, veri setindeki örnekleri alt ve üst yaklaşımlar (lower and upper approximations) aracılığıyla analiz eder. Bu nedenle, sürekli (sayısal) değişkenler içeren veri setlerinde doğrudan uygulama yapılamamakta; öncelikle uygun bir kategorik dönüşüm süreci gerçekleştirilmesi gerekmektedir [28].

Pima Indians Diabetes veri seti, çoğunlukla sürekli nitelikteki değişkenlerden oluşmaktadır. Örneğin; glikoz düzeyi, insülin seviyesi, vücut kitle indeksi (BMI) veya yaş gibi değişkenler sayısal değerler içermekte olup bu haliyle doğrudan Kaba Kümeleme Teorisi'ne uygun değildir. Bu nedenle veri setinde yer alan her bir sürekli değişken, belirli aralıklara bölünerek kategorik sınıflar hâline dönüştürülmüştür.

Bu işlem için niceliksel değerleri **çeyrekliklere (quartile-based discretization)** göre ayıran bir yöntem tercih edilmiştir. Her bir değişken için:

- **Q1 (1. çeyreklik):** Veri kümesinin %25'lik alt sınır değeri
- **Q2 (medyan):** Orta değer (istatistiksel ortanca)
- **Q3 (3. çeyreklik):** Üst %25'lik sınır değeri

kullanılarak veri aralığı üç veya dört farklı sınıfa bölünmüştür. Örneğin, glikoz düzeyi değişkeni için şu şekilde bir dönüşüm yapılmıştır:

- **Düşük (Low):** $\text{Glikoz} \leq Q1$
- **Orta (Medium):** $Q1 < \text{Glikoz} \leq Q3$
- **Yüksek (High):** $\text{Glikoz} > Q3$

Bazı değişkenlerde daha fazla kategori gerekebileceği durumlar göz önünde bulundurularak esnek bir yaklaşım izlenmiş ve veri dağılımı analiz edilmiştir. Her bir değişkenin kategorik karşılığı, Python programlama dili ile oluşturulan özel bir dönüşüm algoritması sayesinde sistematik olarak üretilmiştir. Kullanılan kod parçacığında *pandas.cut()* ve *numpy.quantile()* fonksiyonları temel alınmış, böylece dönüşüm işlemi sırasında insan hatası minimize edilmiştir.

Ayrıca, kategorik dönüşümün ardından veri setinde oluşabilecek bilgi kaybını minimize etmek amacıyla, her bir değişkene ait yeni sınıflandırmalar görselleştirilmiş ve orijinal değerlerle karşılaştırmalı analiz yapılmıştır. Bu süreçte özellikle korelasyon ve karar kurallarının bozulmadığına dikkat edilmiştir.

3.6.1. Pregnancies sütunu için kategorikleştirme süreci

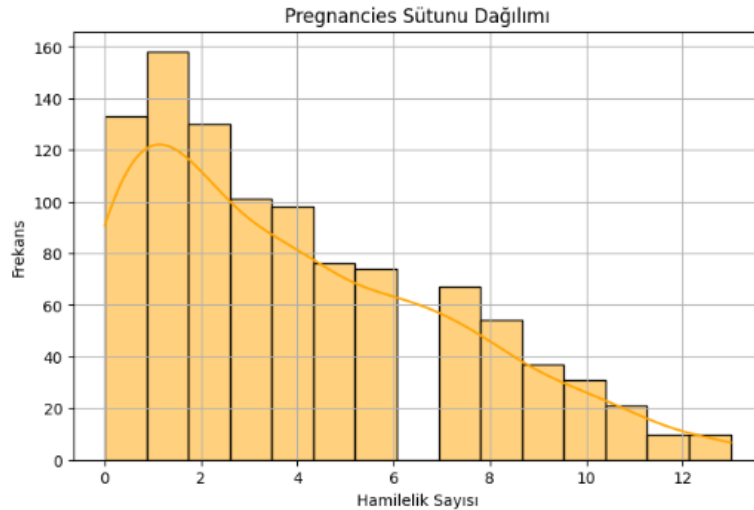
```
# Pregnancies sütununa ait temel istatistiksel bilgiler  
df["Pregnancies"].describe()
```

Pregnancies	
count	1000.000000
mean	3.962000
std	3.246397
min	0.000000
25%	1.000000
50%	3.000000
75%	6.000000
max	13.000000

dtype: float64

Şekil 3.18. Pregnancies sütununa ait temel istatistiksel bilgiler ve python kodu.

```
# Pregnancies dağılım grafiği (histogram)  
plt.figure(figsize=(8, 5))  
sns.histplot(df["Pregnancies"], bins=15, kde=True, color='orange', edgecolor='black')  
plt.title("Pregnancies Sütunu Dağılımı")  
plt.xlabel("Hamilelik Sayısı")  
plt.ylabel("Frekans")  
plt.grid(True)  
plt.show()
```



Şekil 3.19. Pregnancies sütunu histogram grafiği ve python kodu.

```
# Örnek kategori mantığı (Tablo 4'ten ilhamla):

# 0 → Hiç hamile kalmamış
# 1-3 → Düşük düzeyde hamilelik
# 4-6 → Orta düzeyde hamilelik
# 7 ve üzeri → Yüksek hamilelik

# Pregnancies değişkenini kategorik yapıya dönüştür
def categorize_pregnancies(x):
    if x == 0:
        return "Hiç hamile kalmamış"
    elif 1 <= x <= 3:
        return "Düşük düzeyde hamilelik"
    elif 4 <= x <= 6:
        return "Orta düzeyde hamilelik"
    else:
        return "Yüksek düzeyde hamilelik"

# Yeni sütun oluşturalım
df["Pregnancies_cat"] = df["Pregnancies"].apply(categorize_pregnancies)

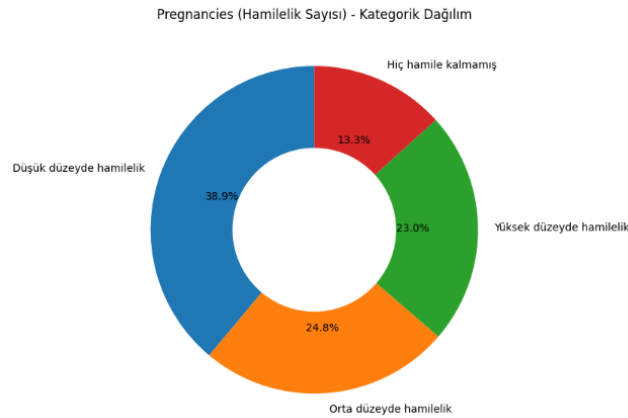
# Sonuçları görelim
df[["Pregnancies", "Pregnancies_cat"]].head(10)
```

	Pregnancies	Pregnancies_cat
0	6	Orta düzeyde hamilelik
1	1	Düşük düzeyde hamilelik
2	8	Yüksek düzeyde hamilelik
3	1	Düşük düzeyde hamilelik
4	0	Hiç hamile kalmamış
5	5	Orta düzeyde hamilelik
6	3	Düşük düzeyde hamilelik
7	10	Yüksek düzeyde hamilelik
8	2	Düşük düzeyde hamilelik
9	8	Yüksek düzeyde hamilelik

Şekil 3.20. Pregnancies sütunu kategorik yapı ve python kodu.

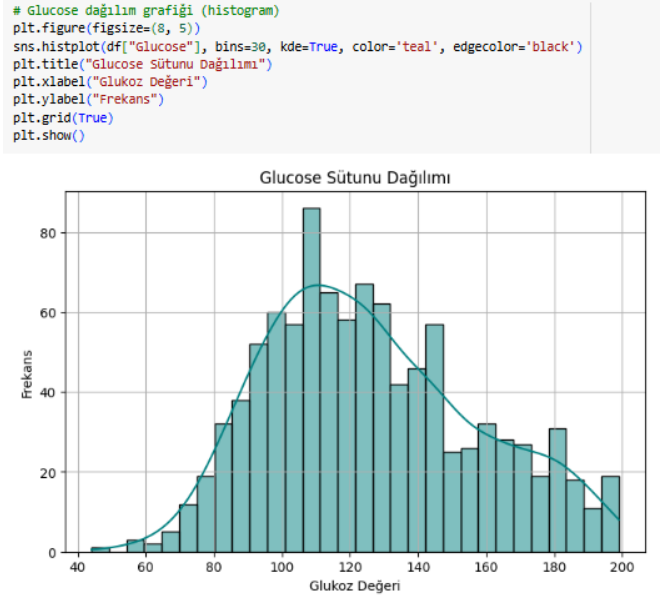
```
# Kategorik dağılımı görselleştir (Pasta Grafiği)

# Kategorik hamilelik sınıflarının dağılımını gör
plt.figure(figsize=(7, 7))
df["Pregnancies_cat"].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, wedgeprops=dict(width=0.5))
plt.title("Pregnancies (Hamilelik Sayısı) - Kategorik Dağılım")
plt.ylabel("") # y etiketi kaldır
plt.show()
```



Şekil 3.21. Pregnancies sütunu kategorik dağılım ve python kodu.

3.6.2. Glucose sütunu için kategorikleştirme süreci



Şekil 3.22. Glucose sütunu histogram grafiği ve python kodu.

```
# Kategori sınırlarını belirleyelim:
# Tıbbi çalışmalara ve gözleme göre:

# ≤ 90 → Düşük (Hipoglisemi riski)
# 91 - 120 → Normal
# 121 - 140 → Yüksek
# > 140 → Çok yüksek (hiperglisemi riski)

# Bu aralıklar, hem görsel dağılıma hem de tıbbi referanslara dayanır.

# Glucose değişkenini kategorik yapıya dönüştür
def categorize_glucose(x):
    if x <= 90:
        return "Düşük"
    elif 91 <= x <= 120:
        return "Normal"
    elif 121 <= x <= 140:
        return "Yüksek"
    else:
        return "Çok yüksek"

# Yeni sütun oluşturalım
df["Glucose_cat"] = df["Glucose"].apply(categorize_glucose)

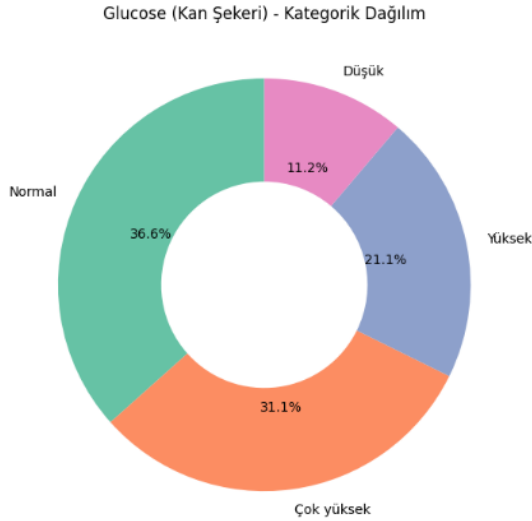
# Sonuçları görelim
df[["Glucose", "Glucose_cat"]].head(10)
```

	Glucose	Glucose_cat
0	148.0	Çok yüksek
1	85.0	Düşük
2	183.0	Çok yüksek
3	89.0	Düşük
4	137.0	Yüksek
5	116.0	Normal
6	78.0	Düşük
7	115.0	Normal
8	197.0	Çok yüksek
9	125.0	Yüksek

Şekil 3.23. Glucose sütunu kategorik yapı ve python kodu.

```
# Kategorik Glukoz dağılımı (Pasta Grafiği)

# Kategorik Glucose sınıflarının dağılımını gör
plt.figure(figsize=(7, 7))
df["Glucose_cat"].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, wedgeprops=dict(width=0.5), colors=sns.color_palette("Set2"))
plt.title("Glucose (Kan Şekeri) - Kategorik Dağılım")
plt.ylabel("")
plt.show()
```



Şekil 3.24. Glucose sütunu kategorik dağılım ve python kodu.

Diğer sütunlar içinde benzer şekilde analizler yapılarak kategorikleştirme işlemi gerçekleştirilmiştir ve sonuç olarak bir kategorik veri seti elde edilmiştir. Kategorik veri setine ait ilk beş kayıt Şekil 3.25.'te gösterilmiştir.

Elde edilen kategorik veri seti, C# dili ile geliştirilen masaüstü yazılım aracılığıyla Kaba Kümeleme analizi için kullanılmaya hazır hâle getirilmiştir. Bu dönüşüm süreci sayesinde, teorik gereklilikler sağlanmış ve analiz süreçlerinin doğru ve anlamlı sonuçlar üretmesi güvence altına alınmıştır.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	Orta düzeyde hamilelik	Çok yüksek	Normal	Orta	Normal	Obez	Orta Risk	İleri Yaşlı	Diyabet hastası
1	Düşük düzeyde hamilelik	Düşük	Normal	Orta	Normal	Fazla kilolu	Düşük Risk	Orta Yaşlı	Sağlıklı
2	Yüksek düzeyde hamilelik	Çok yüksek	Normal	Orta	Normal	Normal	Orta Risk	Orta Yaşlı	Diyabet hastası
3	Düşük düzeyde hamilelik	Düşük	Normal	Orta	Normal	Fazla kilolu	Düşük Risk	Genç	Sağlıklı
4	Hiç hamile kalmamış	Yüksek	Düşük	Orta	Normal	Obez	Orta Risk	Orta Yaşlı	Diyabet hastası

Şekil 3.25. Kategorik veri setine ait ilk beş kayıt.

4. KABA KÜMELEME TEORİSİ VE KARAR KURALLARI ÜRETİMİ

4.1. Kaba Kümeleme Teorisine Giriş

Kaba Kümeleme Teorisi (Rough Set Theory – RST), 1982 yılında Zdzisław Pawlak tarafından geliştirilmiş ve özellikle belirsizlik, eksik bilgi ve yaklaşıksal tanımlama gerektiren durumlarda güçlü bir çözüm yöntemi olarak öne çıkmıştır [29]. Teori, kesin sınırlarla tanımlanamayacak bilgi kümelerini yaklaşık olarak temsil etmeye olanak tanır. Bu yönüyle, bulanık mantık ve olasılıksal modellere alternatif bir yaklaşım sunar.

RST'nin temelinde, evren kümesindeki (U) nesneler bir özellik kümesi (A) ile tanımlanır. Bu özellikler aracılığıyla benzerlik ilişkileri kurulur ve nesneler ayrıştırılır. RST, klasik küme kavramının ötesinde, her küme için bir **alt yaklaşım (lower approximation)** ve bir **üst yaklaşım (upper approximation)** tanımlar. Alt yaklaşım, tamamen o kümeye ait olduğu kesin olarak bilinen nesneleri; üst yaklaşım ise o kümeye ait olabilecek potansiyel nesneleri ifade eder [28].

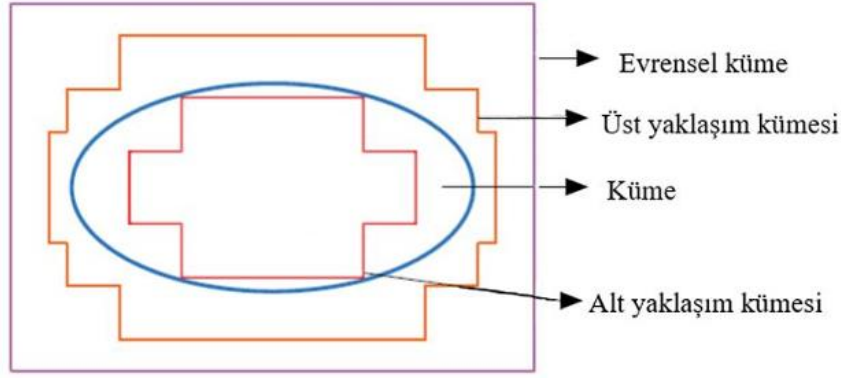
Bu iki yaklaşım arasında kalan fark, **sınır bölgesi (boundary region)** olarak tanımlanır ve belirsizliğin derecesini gösterir. Alt yaklaşım kümesi (1), Üst yaklaşım kümesi Şekil (2) ve Sınır Bölgesi (3) olacak şekilde Şekil 4.1.'de formül olarak gösterilmiştir. Ayrıca Şekil 4.2.'de Kaba kümeleme yaklaşımları küme boyutunda ele alınmıştır. Bu durum, verilerin bazı özellikleri hakkında net bir bilgiye ulaşamadığında, karar verme süreçlerinde önemli bir avantaj sağlar. Özellikle sağlık alanı gibi doğruluk ve yorumlanabilirliğin kritik olduğu disiplinlerde RST'nin sunduğu karar kuralları, açıklanabilir yapay zeka (XAI) yaklaşımlarına katkı sağlamaktadır [30].

$$\underline{BX} = \{x_i \in U \mid [x_i]_{IND(B)} \subset X\} \quad (1)$$

$$\overline{BX} = \{x_i \in U \mid [x_i]_{IND(B)} \cap X \neq \emptyset\} \quad (2)$$

$$BN(X) = \overline{BX} \setminus \underline{BX} \quad (3)$$

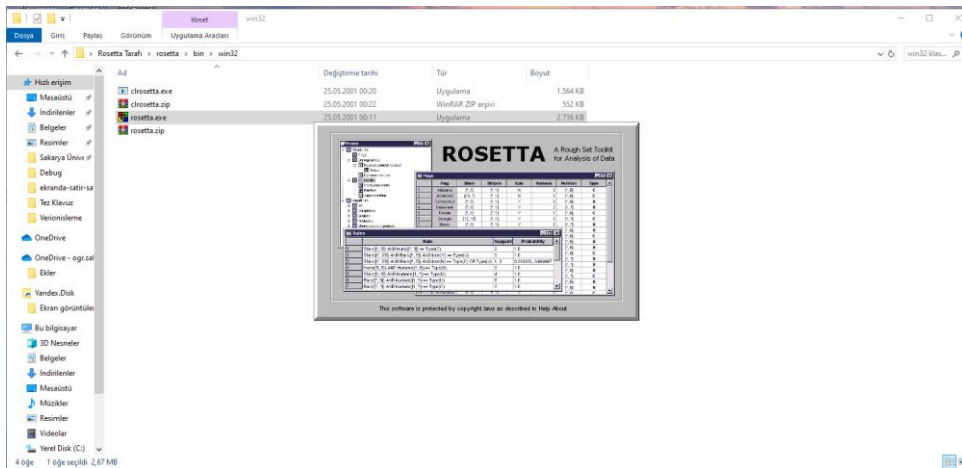
Şekil 4.1. Alt küme yaklaşımı, Üst küme yaklaşımı ve Sınır Bölgesi ifade biçimi.



Şekil 4.2. Kaba kümeleme yaklaşımları [12].

RST, veri madenciliği, makine öğrenmesi, desen tanıma ve karar destek sistemlerinde etkin şekilde kullanılmakta ve özellikle öznitelik indirgeme (attribute reduction), karar kuralı üretimi ve sınıflandırma işlemlerinde güçlü sonuçlar vermektedir [2]. Sağlık bilişimi gibi yüksek boyutlu ve hassas verilerin analizinde, Kaba Kümeleme Teorisi'nin yorumlanabilirliği, bu alanlarda yaygın şekilde tercih edilmesini sağlamıştır.

RST'nin uygulama alanlarında yaygın olarak kullanılan araçlardan biri, **ROSETTA** adlı açık kaynaklı yazılımdır. [31] İsveç Linköping Üniversitesi bünyesinde geliştirilen ROSETTA, kullanıcı dostu arayüzü ve sağlam algoritmik altyapısı sayesinde Kaba Kümeleme Teorisi'nin temel işlemlerini (özellikle **karar kuralları üretimi**) desteklemektedir. Yazılım, veri seti yükleme, öznitelik indirgeme, karar kuralları çıkarımı ve sınıflandırma gibi temel işlemleri otomatik olarak gerçekleştirmektedir. Ayrıca, algoritma seçimi (Johnson's reduct, genetic reduct, vb.) gibi parametrik özelleştirmelere de izin vermektedir [32].

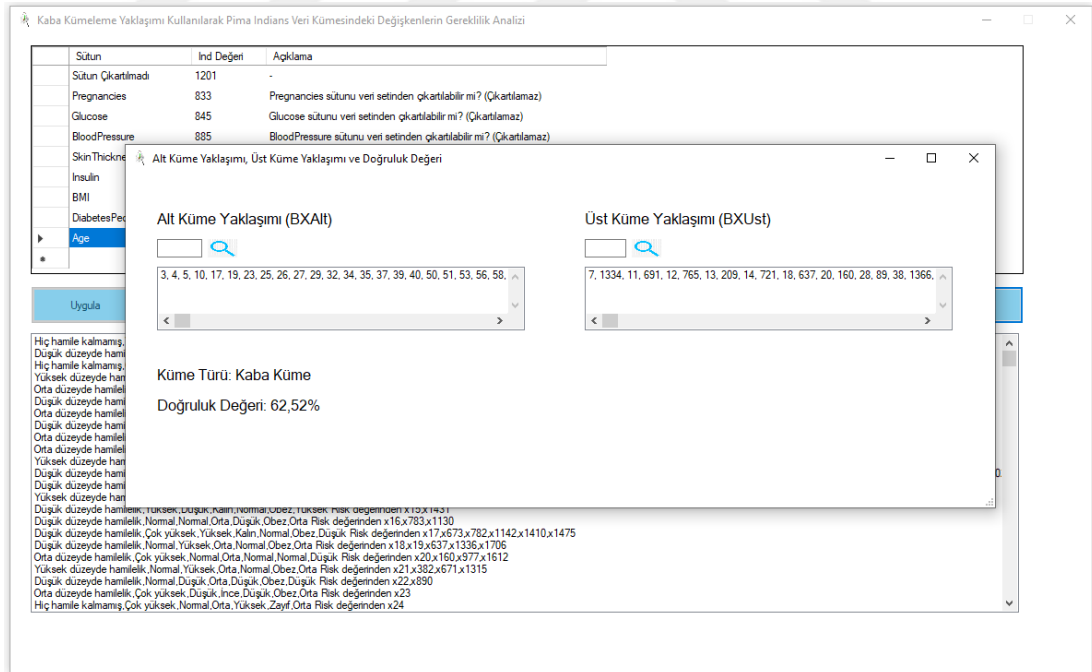


Şekil 4.3. Rosetta yazılımına ait bir ekran görüntüsü.

Ancak ROSETTA'nın bazı sınırlılıkları da bulunmaktadır. Örneğin; **alt ve üst yaklaşım kümeleri, gereklilik analizi, doğruluk oranı hesaplaması** gibi daha ayrıntılı kaba kümeleme metriklerini içermemektedir. Ayrıca yazılımın çıktıları genellikle sadece karar kuralları ile sınırlı kalmakta ve gerçek zamanlı sistemlerle entegrasyon için uygun bir yapı sunmamaktadır.

Bu tez kapsamında gerçekleştirilen analizlerde, ROSETTA yazılımı referans alınarak oluşturulan karar kuralları bu yazılım aracılığıyla da test edilmiş ve doğrulanmıştır. Ancak yukarıda belirtilen sınırlamalar nedeniyle daha bütüncül bir kaba kümeleme analizi yapılabilmesi amacıyla, **C# programlama dili kullanılarak özel bir masaüstü yazılım geliştirilmiştir**. Bu yazılım ile ROSETTA'nın sunduğu karar kurallarına ek olarak aşağıdaki analiz bileşenleri de desteklenmiştir:

- Alt ve üst yaklaşım kümelerinin hesaplanması (Şekil 4.4.'te gösterilmiştir.)
- Gereklilik analizi ile özniteliklerin önem derecesinin belirlenmesi
- Karar kurallarının sadeleştirilmesi ve mantıksal ifadelerle dönüştürülmesi
- Doğruluk oranlarının sayısal olarak hesaplanması



Şekil 4.4. Veri setine ait alt küme yaklaşımı, üst küme yaklaşımı, küme türü ve doğruluk değeri analizi.

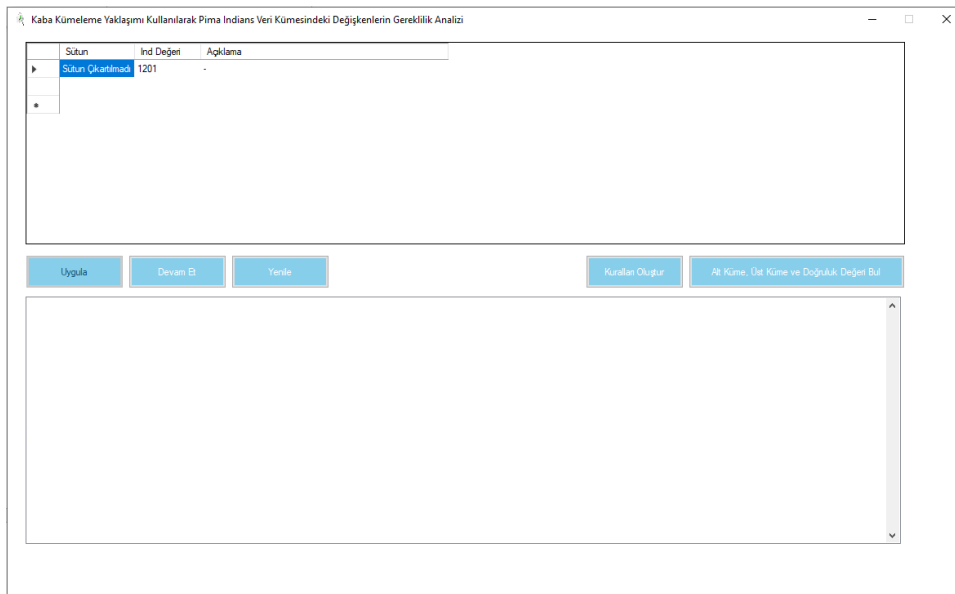
Geliştirilen sistem, ROSETTA ile üretilen karar kurallarını başarıyla üretmiş ve böylece teorik doğruluk bakımından doğrulanabilirlik elde edilmiştir. Bununla birlikte, sistemde yer alan ilave analiz modülleri sayesinde kaba kümeleme teorisinin

uygulamalı yönleri daha ayrıntılı ve esnek biçimde değerlendirilmiştir. Bu yaklaşım, yalnızca karar destek sistemlerine katkı sağlamakla kalmayıp, aynı zamanda literatürdeki açık kaynaklı araçların da sınırlarını ortaya koyan özgün bir yazılım geliştirme örneği sunmaktadır.

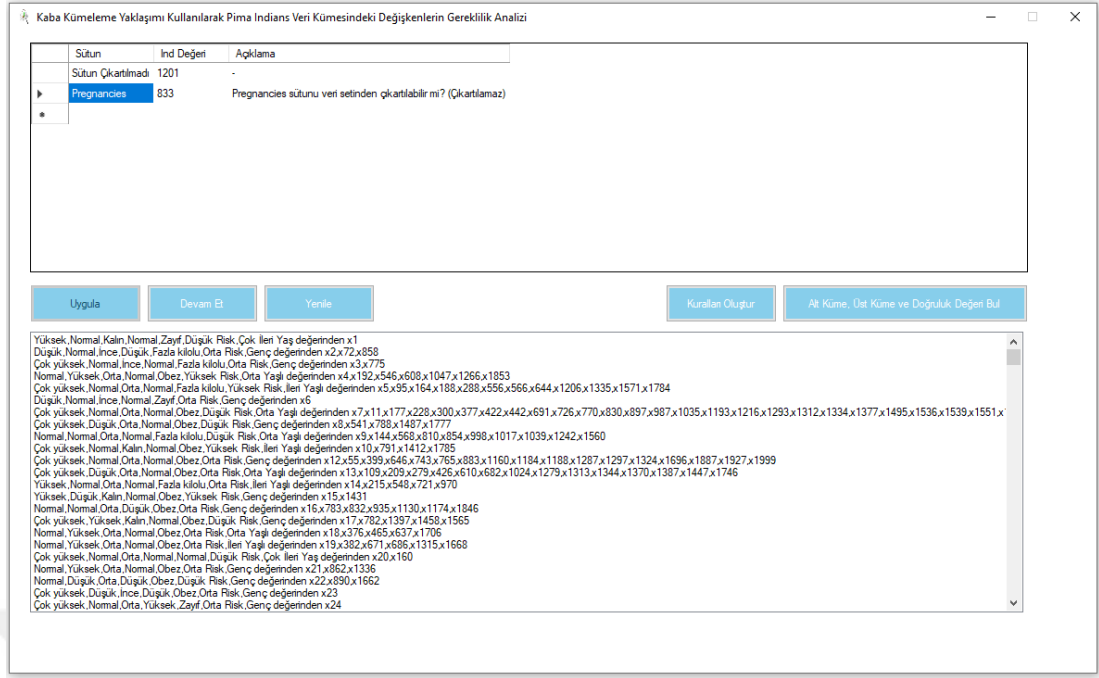
4.2. Öznitelik İndirgeme Süreci

Kaba Kümeleme Teorisi'nin temel uygulamalarından biri olan öznitelik indirgeme (attribute reduction), karar verme sürecine katkı sağlamayan, yani anlamsal olarak fazlalık oluşturan özniteliklerin elimine edilmesini hedeflemektedir. Bu işlem sırasında, veri setindeki ayrılmazlık (indiscernibility) ilişkisi esas alınır. Ayrılmazlık ilişkisi, belirli bir öznitelik kümesine göre ayırt edilemeyen nesneleri belirlemek amacıyla kullanılır ve böylece hangi özniteliklerin karar sınıfları üzerindeki ayrıştırıcı etkisinin daha düşük olduğu analiz edilebilir [29].

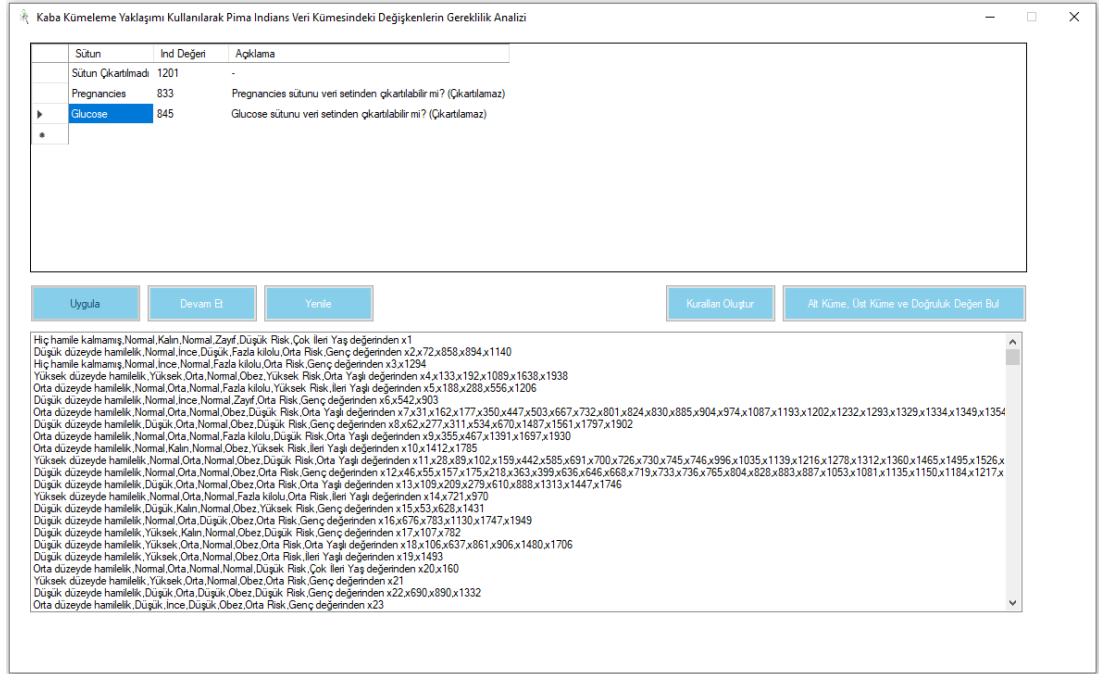
Bu çalışmada, daha önce kategorik hale dönüştürülmüş olan veri seti üzerinde öznitelik indirgeme süreci detaylı biçimde yürütülmüştür. İlk aşamada, tüm özniteliklerin bulunduğu veri setinin ayrılmazlık ilişki değeri (indiscernibility count) hesaplanmış ve bu değer 1201 olarak elde edilmiştir ve Şekil 4.5.'te görülmektedir. Ardından, her bir öznitelik veri setinden tek tek çıkarılarak aynı ilişki değeri tekrar hesaplanmıştır. Şekil 4.6., Şekil 4.7., ... ve Şekil 4.13'de her bir öznitelik veri setinden çıkartıldığında elde edilen ayrılmazlık ilişki değerleri görsel olarak sunulmuştur.



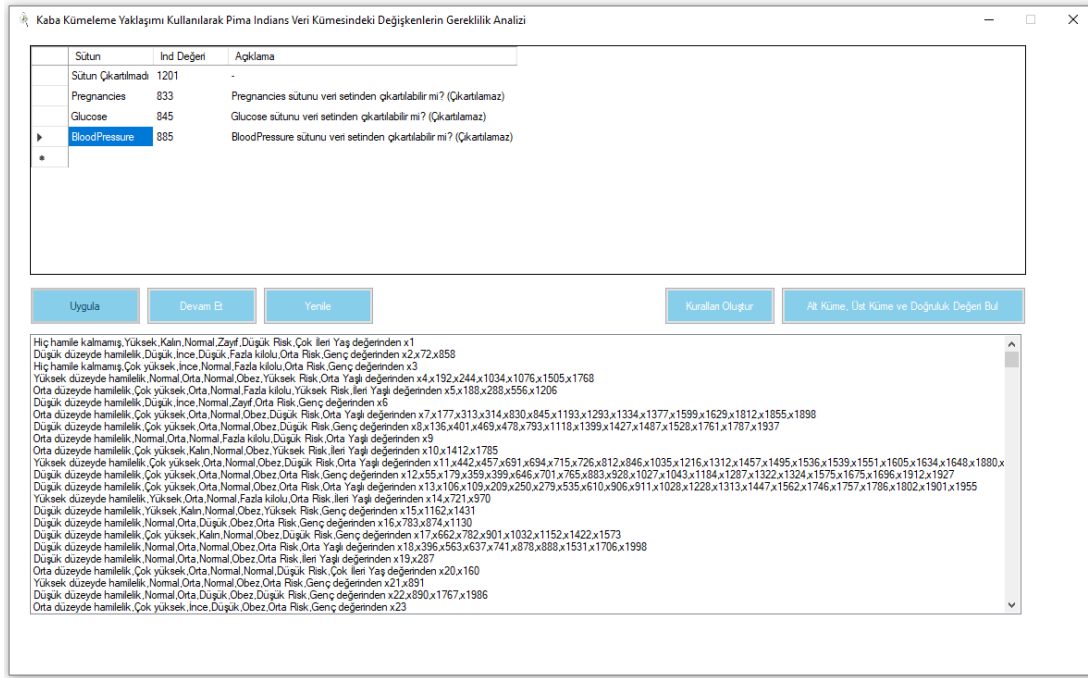
Şekil 4.5. Veri setinin hiçbir öznitelik çıkartılmadığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.



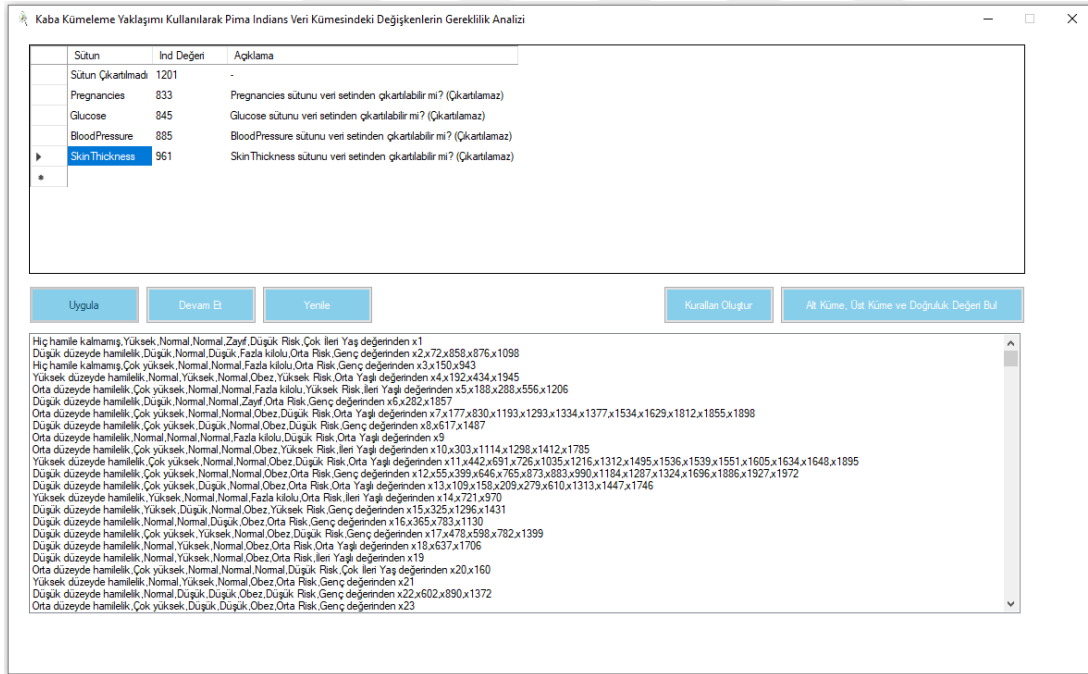
Şekil 4.6. Veri setinden Pregnancies öz niteliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.



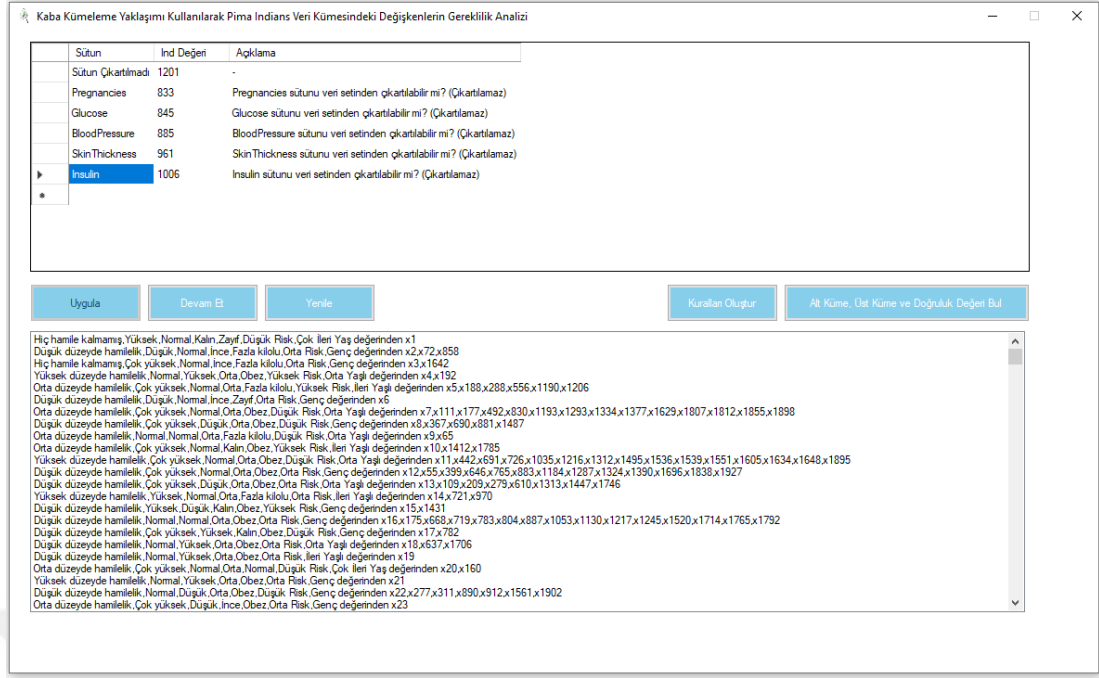
Şekil 4.7. Veri setinden Glucose öz niteliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.



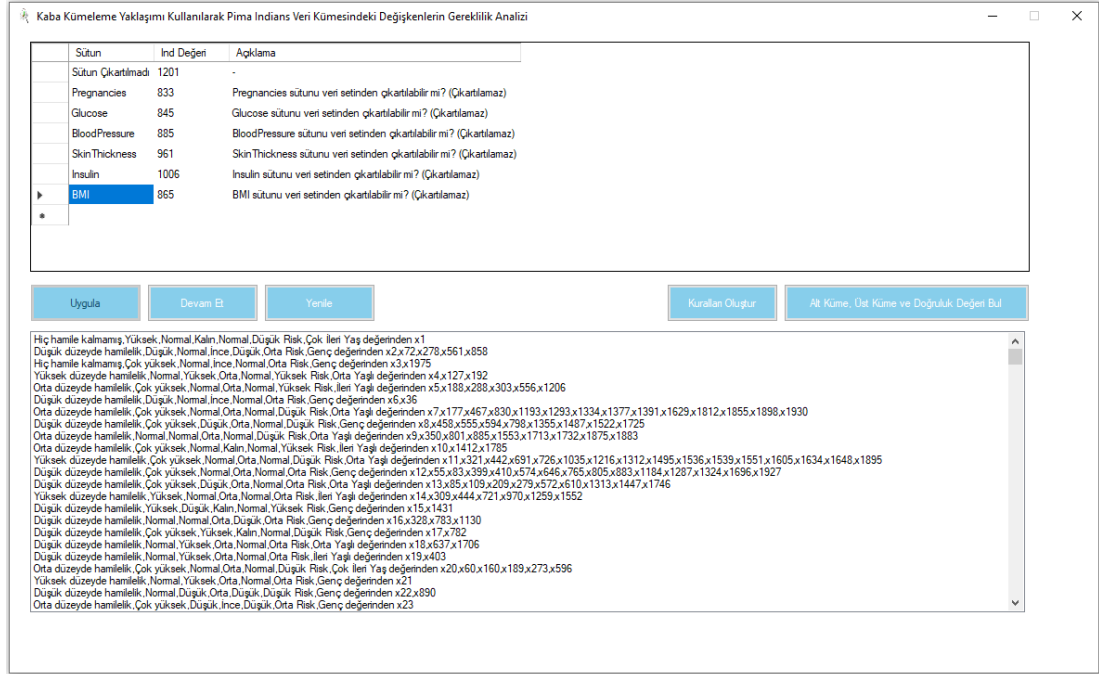
Şekil 4.8. Veri setinden BloodPressure özneteliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.



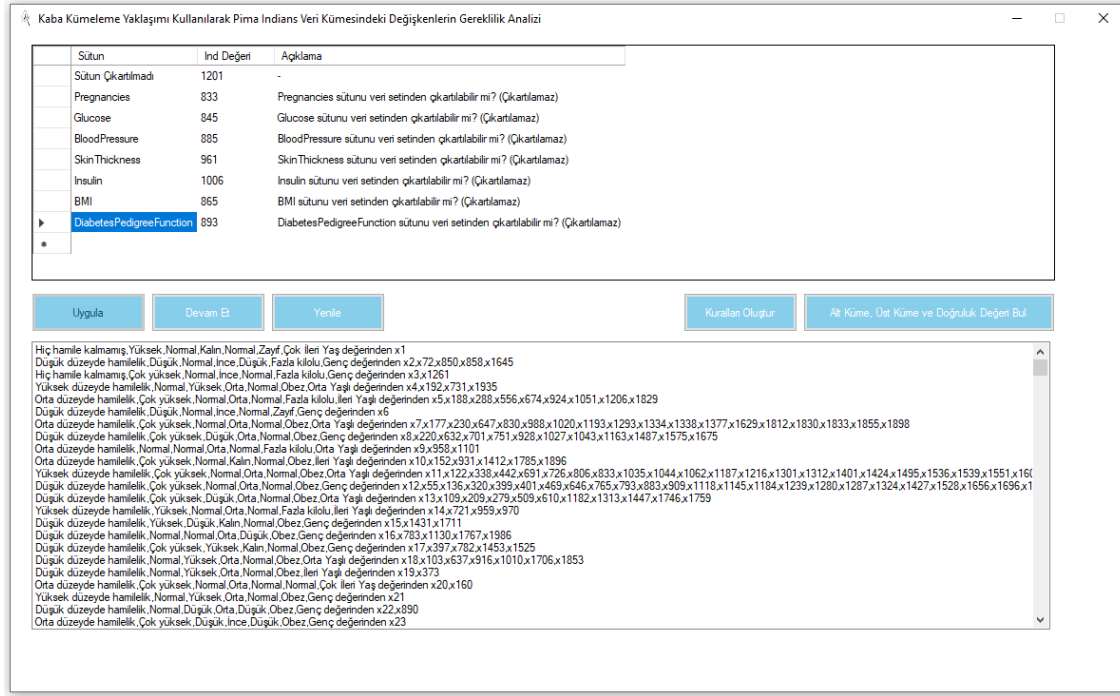
Şekil 4.9. Veri setinden SkinThickness özneteliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.



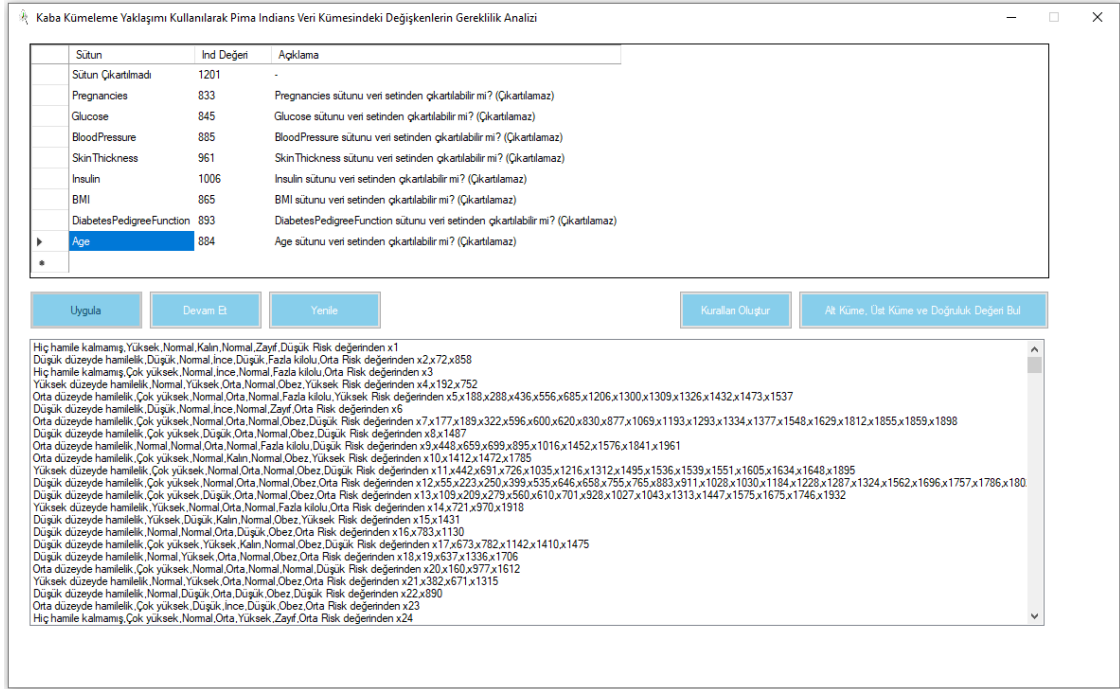
Şekil 4.10. Veri setinden Insulin özneteliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.



Şekil 4.11. Veri setinden BMI özneteliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.



Şekil 4.12. Veri setinden DiabetesPedigreeFunction özneliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.



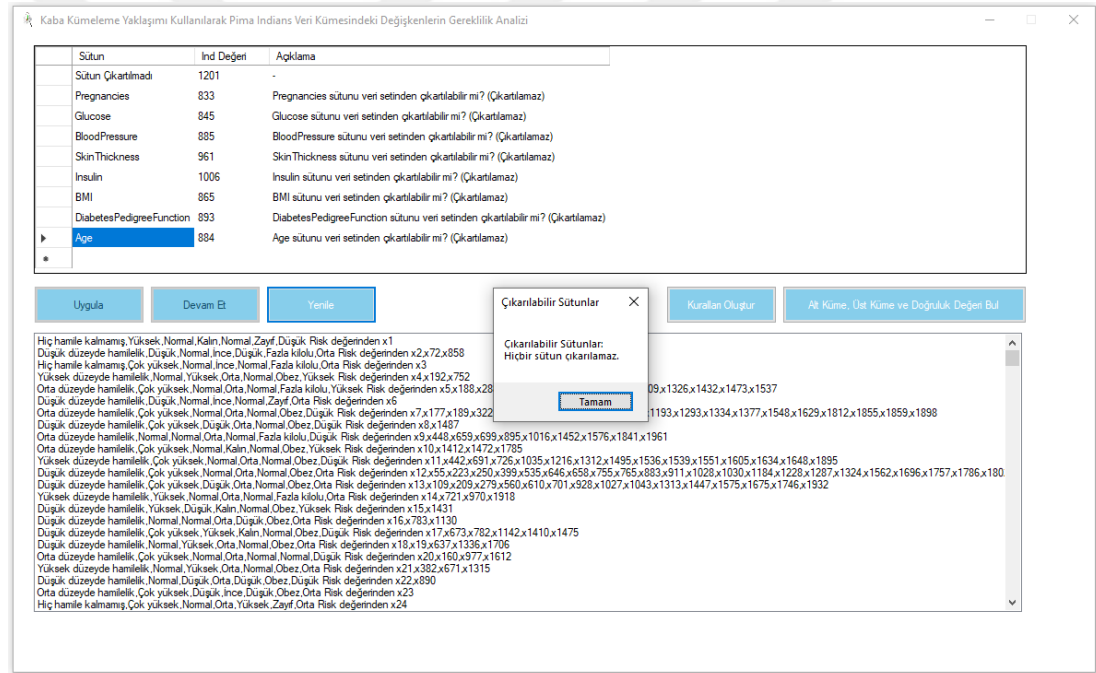
Şekil 4.13. Veri setinden Age özneliği çıkartıldığında elde edilen ayrılmazlık ilişkisine ait ekran görüntüsü.

Elde edilen sonuçlar şu şekildedir:

- Pregnancies: 833
- Glucose: 845

- BloodPressure: 885
- SkinThickness: 961
- Insulin: 1006
- BMI: 865
- Diabetes Pedigree Function: 893
- Age: 884

Kaba Kümeleme Teorisi'ne göre bir öznitelik yalnızca, çıkarıldığında veri setinin ayrılmazlık değerinde bir azalma meydana getirmiyorsa yani orijinal değer ile aynı kalıyorsa, elenebilir [28]. Yukarıdaki bulgulara göre, hiçbir öznitelik çıkarıldığında bu eşitlik sağlanamamış ve dolayısıyla tüm özniteliklerin karar verme sürecinde anlamlı olduğu sonucuna ulaşılmıştır.



Şekil 4.14. Veri setinden çıkartılabilecek özniteliklere ait ekran görüntüsü.

Bu sonuçlar, veri setindeki tüm özniteliklerin sınıflandırma ve teşhis süreçleri açısından değerli olduğunu ve indirgeme yapılmaksızın karar kurallarının oluşturulması gerektiğini göstermektedir. Bu bulgu, özellikle küçük veri setleri üzerinde çalışan sistemlerde maksimum bilgi kullanımı açısından önemli bir avantaj sağlamaktadır [30].

4.3. Karar Kurallarının Elde Edilmesi

Kaba Kümeleme Teorisi'nin en temel ve uygulamaya dönük çıktılarından biri, karar kurallarının (decision rules) oluşturulmasıdır. Bu kurallar, belirli öznitelik kombinasyonları gerçekleştiğinde hangi karar sınıfına (örneğin hasta veya sağlıklı) ait olunabileceğini tanımlayan ve “Eğer ... ise ...” (If ... then ...) yapısında ifade edilen mantıksal ifadelerdir. Bu bağlamda karar kuralları, veri temelli karar destek sistemlerinin omurgasını oluşturarak uzman sistemlerin geliştirilmesine katkı sağlar [29].

Bu çalışmada, veri seti üzerinde yapılan öznitelik indirgeme süreci sonucunda herhangi bir değişkenin elenmesine gerek duyulmamıştır. Bu nedenle karar kuralları, veri setinin tam hâli kullanılarak üretilmiştir. İlk aşamada reduct'lar (en az sayıda öznitelikle aynı ayırım gücünü sağlayan alt kümeler) hesaplanmış ve bu reduct'lar yardımıyla fark edilebilirlik matrisi (discernibility matrix) oluşturulmuştur. Discernibility matrisi, birbirinden ayırt edilmesi gereken örnek çiftlerini analiz ederek hangi özniteliklerin bu ayırımı sağladığını ortaya koyar [30].

Geliştirilen C# tabanlı yazılım aracılığıyla gerçekleştirilen analiz sonucunda, toplamda **1319 adet karar kuralı** elde edilmiştir. Bu karar kuralları, sistemin teşhis gücünü açıklamakta ve diyabet gibi karmaşık bir hastalığın sınıflandırılmasında önemli katkılar sunmaktadır. Kurallar, örneğin:

Eğer Glucose = High ve BMI = Medium ise Outcome = 1 (Hasta)

Eğer Glucose = Normal ve Insulin = Low ise Outcome = 0 (Sağlıklı)

şeklinde özetlenebilir. Elde edilen bu kurallar Şekil 4.15'te örneklenmiş olup, sistemin karar verme altyapısını oluşturur.

Kuralları	Kuralları	Sonuç
x1	Pregnancies<Hç hantle kulnangı ve Glucose.Yüksek ve BloodPressure.Normal ve SkinThickness.Kalı ve Insulin.Normal ve BMI.Zayıf ve DiabetesPedigreeFunction.Düşük Risk ve Age.Çok İler Yaşı	Sağlık
x101x141x1705	Pregnancies.Ota düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Normal ve SkinThickness.Kalı ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Yüksek Risk ve Age.İler Yaşı	Diyabet hastası
x100	Pregnancies.Yüksek düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Normal ve SkinThickness.İnce ve Insulin.Yüksek ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.Gerç	Sağlık
x1001	Pregnancies.Hç hantle kulnangı ve Glucose.Çok yüksek ve BloodPressure.Yüksek ve SkinThickness.Ota ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.Ota Yaşı	Diyabet hastası
x1002	Pregnancies.Düşük düzeyde hantle ve Glucose.Normal ve BloodPressure.Düşük ve SkinThickness.İnce ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık
x1003	Pregnancies.Düşük düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Normal ve SkinThickness.Kalı ve Insulin.Normal ve BMI.Fada kılı ve DiabetesPedigreeFunction.Düşük Risk ve Age.Ota Yaşı	Diyabet hastası
x1005x1609	Pregnancies.Yüksek düzeyde hantle ve Glucose.Normal ve BloodPressure.Normal ve SkinThickness.Ota ve Insulin.Düşük ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.Ota Yaşı	Diyabet hastası
x1007x1417	Pregnancies.Ota düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Düşük ve SkinThickness.Ota ve Insulin.Normal ve BMI.Zayıf ve DiabetesPedigreeFunction.Yüksek Risk ve Age.Gerç	Sağlık
x1008	Pregnancies.Hç hantle kulnangı ve Glucose.Çok yüksek ve BloodPressure.Düşük ve SkinThickness.Ota ve Insulin.Normal ve BMI.Fada kılı ve DiabetesPedigreeFunction.Düşük Risk ve Age.Ota Yaşı	Diyabet hastası
x101x1865	Pregnancies.Hç hantle kulnangı ve Glucose.Çok yüksek ve BloodPressure.Düşük ve SkinThickness.Ota ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.Gerç	Diyabet hastası
x1011x1679	Pregnancies.Düşük düzeyde hantle ve Glucose.Düşük ve BloodPressure.Normal ve SkinThickness.Ota ve Insulin.Normal ve BMI.Fada kılı ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık
x1012	Pregnancies.Ota düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Düşük ve SkinThickness.İnce ve Insulin.Yüksek ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık
x1018x1895	Pregnancies.Ota düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Düşük ve SkinThickness.İnce ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık
x102x155x495x700x730x745x746x1078x...	Pregnancies.Düşük düzeyde hantle ve Glucose.Normal ve BloodPressure.Normal ve SkinThickness.İnce ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.Ota Yaşı	Diyabet hastası
x1021	Pregnancies.Ota düzeyde hantle ve Glucose.Normal ve BloodPressure.Normal ve SkinThickness.Kalı ve Insulin.Yüksek ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.İler Yaşı	Diyabet hastası
x1022	Pregnancies.Yüksek düzeyde hantle ve Glucose.Düşük ve BloodPressure.Yüksek ve SkinThickness.Ota ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Yüksek Risk ve Age.İler Yaşı	Diyabet hastası
x1024x1344	Pregnancies.Ota düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Düşük ve SkinThickness.Ota ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.Ota Yaşı	Diyabet hastası
x1026	Pregnancies.Yüksek düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Düşük ve SkinThickness.Ota ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Yüksek Risk ve Age.Çok İler Yaşı	Diyabet hastası
x1029	Pregnancies.Düşük düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Normal ve SkinThickness.İnce ve Insulin.Düşük ve BMI.Orta ve DiabetesPedigreeFunction.Yüksek Risk ve Age.Gerç	Sağlık
x103x1916x1010	Pregnancies.Düşük düzeyde hantle ve Glucose.Normal ve BloodPressure.Yüksek ve SkinThickness.Ota ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.Ota Yaşı	Diyabet hastası
x1030	Pregnancies.Düşük düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Normal ve SkinThickness.Ota ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.İler Yaşı	Diyabet hastası
x1031	Pregnancies.Düşük düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Yüksek ve SkinThickness.Ota ve Insulin.Yüksek ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.Gerç	Sağlık
x1032	Pregnancies.Ota düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Düşük ve SkinThickness.İnce ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık
x1036	Pregnancies.Ota düzeyde hantle ve Glucose.Yüksek ve BloodPressure.Normal ve SkinThickness.Kalı ve Insulin.Yüksek ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.Ota Yaşı	Sağlık
x1039	Pregnancies.Hç hantle kulnangı ve Glucose.Normal ve BloodPressure.Normal ve SkinThickness.Ota ve Insulin.Normal ve BMI.Fada kılı ve DiabetesPedigreeFunction.Düşük Risk ve Age.Ota Yaşı	Diyabet hastası
x104	Pregnancies.Düşük düzeyde hantle ve Glucose.Normal ve BloodPressure.Normal ve SkinThickness.Kalı ve Insulin.Düşük ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık
x1042	Pregnancies.Düşük düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Normal ve SkinThickness.İnce ve Insulin.Düşük ve BMI.Fada kılı ve DiabetesPedigreeFunction.Ota Risk ve Age.Ota Yaşı	Diyabet hastası
x1045	Pregnancies.Ota düzeyde hantle ve Glucose.Düşük ve BloodPressure.Düşük ve SkinThickness.İnce ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Yüksek Risk ve Age.Gerç	Sağlık
x1046	Pregnancies.Hç hantle kulnangı ve Glucose.Çok yüksek ve BloodPressure.Normal ve SkinThickness.Ota ve Insulin.Normal ve BMI.Normal ve DiabetesPedigreeFunction.Ota Risk ve Age.Gerç	Sağlık
x1049	Pregnancies.Ota düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Normal ve SkinThickness.Ota ve Insulin.Yüksek ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık
x105	Pregnancies.Hç hantle kulnangı ve Glucose.Normal ve BloodPressure.Normal ve SkinThickness.Ota ve Insulin.Normal ve BMI.Zayıf ve DiabetesPedigreeFunction.Ota Risk ve Age.Gerç	Sağlık
x1050	Pregnancies.Hç hantle kulnangı ve Glucose.Normal ve BloodPressure.Yüksek ve SkinThickness.Ota ve Insulin.Normal ve BMI.Fada kılı ve DiabetesPedigreeFunction.Düşük Risk ve Age.İler Yaşı	Sağlık
x1052x1213	Pregnancies.Düşük düzeyde hantle ve Glucose.Yüksek ve BloodPressure.Yüksek ve SkinThickness.İnce ve Insulin.Düşük ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.Gerç	Sağlık
x1054	Pregnancies.Düşük düzeyde hantle ve Glucose.Yüksek ve BloodPressure.Düşük ve SkinThickness.Ota ve Insulin.Düşük ve BMI.Fada kılı ve DiabetesPedigreeFunction.Ota Risk ve Age.Gerç	Sağlık
x1056	Pregnancies.Hç hantle kulnangı ve Glucose.Düşük ve BloodPressure.Normal ve SkinThickness.Kalı ve Insulin.Düşük ve BMI.Orta ve DiabetesPedigreeFunction.Yüksek Risk ve Age.Gerç	Sağlık
x1058	Pregnancies.Ota düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Yüksek ve SkinThickness.Ota ve Insulin.Düşük ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.Gerç	Sağlık
x105x106	Pregnancies.Düşük düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Yüksek ve SkinThickness.Ota ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.Ota Yaşı	Diyabet hastası
x1060	Pregnancies.Ota düzeyde hantle ve Glucose.Normal ve BloodPressure.Normal ve SkinThickness.Ota ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.Çok İler Yaşı	Sağlık
x1063	Pregnancies.Hç hantle kulnangı ve Glucose.Çok yüksek ve BloodPressure.Yüksek ve SkinThickness.Ota ve Insulin.Normal ve BMI.Zayıf ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık
x1064	Pregnancies.Hç hantle kulnangı ve Glucose.Çok yüksek ve BloodPressure.Yüksek ve SkinThickness.İnce ve Insulin.Yüksek ve BMI.Zayıf ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık
x1065	Pregnancies.Düşük düzeyde hantle ve Glucose.Normal ve BloodPressure.Yüksek ve SkinThickness.İnce ve Insulin.Normal ve BMI.Zayıf ve DiabetesPedigreeFunction.Düşük Risk ve Age.Ota Yaşı	Sağlık
x1066	Pregnancies.Düşük düzeyde hantle ve Glucose.Çok yüksek ve BloodPressure.Normal ve SkinThickness.Kalı ve Insulin.Normal ve BMI.Zayıf ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık
x107	Pregnancies.Düşük düzeyde hantle ve Glucose.Normal ve BloodPressure.Yüksek ve SkinThickness.İnce ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık
x1072	Pregnancies.Düşük düzeyde hantle ve Glucose.Normal ve BloodPressure.Normal ve SkinThickness.İnce ve Insulin.Düşük ve BMI.Orta ve DiabetesPedigreeFunction.Yüksek Risk ve Age.Gerç	Sağlık
x1073	Pregnancies.Düşük düzeyde hantle ve Glucose.Düşük ve BloodPressure.Düşük ve SkinThickness.İnce ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Ota Risk ve Age.Gerç	Sağlık
x1074	Pregnancies.Ota düzeyde hantle ve Glucose.Düşük ve BloodPressure.Normal ve SkinThickness.İnce ve Insulin.Normal ve BMI.Fada kılı ve DiabetesPedigreeFunction.Yüksek Risk ve Age.Ota Yaşı	Sağlık
x1075	Pregnancies.Düşük düzeyde hantle ve Glucose.Düşük ve BloodPressure.Düşük ve SkinThickness.İnce ve Insulin.Normal ve BMI.Orta ve DiabetesPedigreeFunction.Düşük Risk ve Age.Gerç	Sağlık

Şekil 4.15. Geliştirilen yazılım ile elde edilen karar kuralları örnek ekran görüntüsü.

Ayrıca, oluşturulan karar kurallarının doğruluğunu test etmek ve geliştirilen yazılımın güvenilirliğini sınamak amacıyla açık kaynaklı bir karar destek yazılımı olan **ROSETTA** kullanılmıştır. ROSETTA, Kaba Kümeleme Teorisi temelinde çalışan bir veri madenciliği aracıdır ve kullanıcı dostu arayüzü ile karar kuralları üretiminde sıkça tercih edilmektedir [31]. Aynı veri seti ROSETTA’ya aktarılarak reduct’lar ve karar kuralları hesaplanmış, ardından geliştirilen sistemden elde edilen çıktılar ile karşılaştırılmıştır.

Yapılan karşılaştırma sonucunda, ROSETTA ile geliştirilen yazılımın çıktılarının tutarlı olduğu gözlemlenmiştir. Bu durum, geliştirilen yazılımın doğru ve güvenilir bir biçimde çalıştığını ortaya koymakta ve karar kuralları üretiminde ROSETTA ile benzer bir başarıya sahip olduğunu göstermektedir. Şekil 4.15 ve Şekil 4.16’da, sırasıyla geliştirilen yazılım ve ROSETTA yazılımı ile elde edilen karar kurallarına ait örnek çıktılar sunulmuştur.



Şekil 4.16. Rosetta yazılımı ile elde edilen karar kuralları örnek ekran görüntüsü.

Sonuç olarak, karar kurallarının başarıyla elde edilmesi, hem veri setinin mantıklı bir şekilde yapılandırıldığı hem de geliştirilen sistemin bilimsel kurallar doğrultusunda işlediğini ortaya koymuştur. Elde edilen kurallar, diyabetin erken teşhisinde kullanılabilecek önemli bir karar destek altyapısı sunmaktadır. Ayrıca bu yaklaşım, sağlık sektöründe uzman sistemlerin geliştirilmesine yönelik uygulamalı bir örnek teşkil etmektedir [28].

5. GERÇEK ZAMANLI VERİ ANALİTİĞİ MİMARİSİ

Günümüzde veri analitiği uygulamalarının çoğu, yalnızca statik veri kümeleriyle değil, aynı zamanda gerçek zamanlı (real-time) veri akışlarıyla çalışmak zorundadır. Bu bağlamda, sağlık bilişimi gibi kritik alanlarda anlık veri işleyebilen, ölçeklenebilir ve **event-driven** dağıtık mimariler tasarlamak ve uygulamak büyük önem arz etmektedir. Bu tez çalışmasında geliştirilen gerçek zamanlı teşhis sistemi, hasta verilerinin anlık olarak toplanması, işlenmesi ve tahmin sonuçlarının hem görsel arayüz hem de veritabanı yoluyla erişilebilir hale getirilmesi üzerine inşa edilmiştir.

Geliştirilen mimari aşağıdaki bileşenlerle yapılandırılmıştır:

- **Apache Kafka**, sistemdeki anlık veri iletimini **güvenilir ve sıralı FIFO (First-In, First-Out) kuyruklama prensibi** ile gerçekleştiren merkezi bileşen olarak görev yapmaktadır.
- **Apache Zookeeper**, Kafka altyapısının tutarlılık, süreklilik ve koordinasyonunu sağlayarak sistemin kararlı çalışmasını temin etmektedir.
- **C# programlama dili** ile geliştirilen Kafka Producer bileşeni, kullanıcı arayüzünden alınan hasta verilerini doğrudan Kafka'ya ileterek gerçek zamanlı veri akışını **event-driven** yapıda sağlamaktadır.
- **Java programlama dili** ile geliştirilen Kafka Consumer bileşeni, Kafka üzerinden gelen verileri alarak analiz ve tahmin işlemlerini gerçek zamanlı olarak gerçekleştirmekte; tahmin sonuçlarını doğrudan MSSQL veritabanına kaydetmektedir.
- Gerçek zamanlı teşhis sürecinde her hasta verisi **tekil ve sıralı şekilde işlenmekte** olup, çok iş parçacıklı yapıya yer verilmemiştir. Bu tasarım tercihi, veri akışının **daha güvenli, izlenebilir ve düşük gecikmeli** şekilde yönetilmesini sağlamaktadır.
- **C# tabanlı hastane otomasyon sistemi**, Kafka Producer bileşeni aracılığıyla kullanıcıdan alınan hasta verilerini gerçek zamanlı olarak Kafka'ya iletmekte ve böylece veri akışının ilk adımını sağlamaktadır.

Çok iş parçacıklı yapı ise yalnızca **performans karşılaştırmaları ve algoritma doğruluk analizleri** sırasında kullanılmıştır. Gerçek zamanlı sistemdeki analiz işlemleri, bilinçli olarak **tekil iş parçacığı** ile yürütülmüş ve sistemin kararlılığı önceliklendirilmiştir.

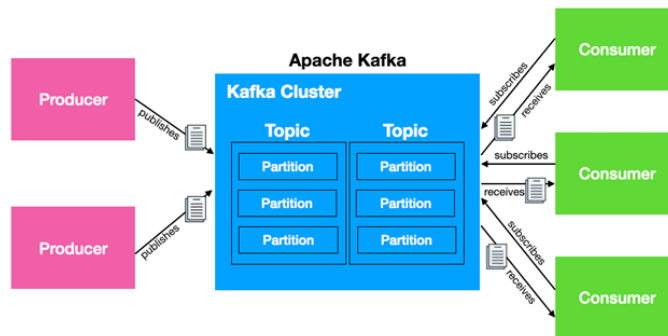
Bu bölümde, söz konusu bileşenlerin işleyişi ayrıntılı olarak ele alınacaktır.

5.1. Gerçek Zamanlı Mimari Bileşenleri

5.1.1. Apache kafka

Apache Kafka, büyük hacimli veri akışlarını gerçek zamanlı olarak işlemek için geliştirilmiş açık kaynaklı bir dağıtık mesajlaşma sistemidir. Kafka, üretici (producer) ile tüketici (consumer) arasında “topic” adı verilen aracı yapılar üzerinden veri iletimini gerçekleştirir. Bu tezde kullanılan Kafka mimarisi, sağlık gibi zaman duyarlılığı yüksek bir alanda anlık kararlar alabilmek için tasarlanmış **event-driven (olay tetiklemeli)** bir yapıya sahiptir. C# diliyle geliştirilen Kafka Producer bileşeni, kullanıcı arayüzünden girilen hasta verilerini doğrudan belirlenen topic üzerinden Apache Kafka’ya iletmektedir. Java ortamında sürekli dinleme konumunda bulunan Kafka Consumer bileşeni ise bu verileri **anlık olarak algılayarak**, makine öğrenmesi algoritmalarıyla analiz etmekte ve tahmin sonuçlarını doğrudan MSSQL veritabanına kaydetmektedir.

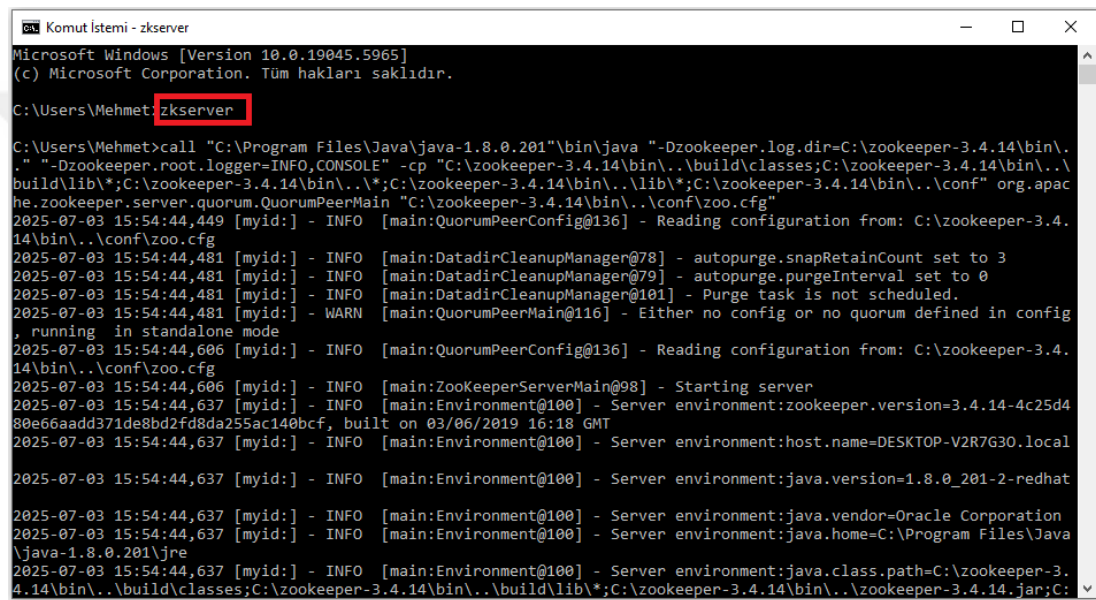
Kafka sisteminde veriler partition adı verilen alt yapılara bölünerek depolanır. Bu yapı, verilerin paralel olarak işlenmesini ve yüksek performans elde edilmesini sağlar. Kafka ayrıca veri kaybını önlemek amacıyla offset yapısını kullanır. Bu sayede herhangi bir kesinti durumunda sistem, verileri kaldığı yerden işlemeye devam edebilir [33].



Şekil 5.1. Apache Kafka yapısı [34].

5.1.2. Apache zookeeper

Apache Zookeeper, dağıtık sistemlerde yapılandırma yönetimi, senkronizasyon ve koordinasyon görevlerini gerçekleştiren bir hizmettir. Kafka bileşenleri arasında iletişimi sağlamak, broker'ların durumunu izlemek ve topic-partition eşlemesini yönetmek gibi işlevlerde Zookeeper kritik bir rol üstlenir. Bu tez kapsamında kullanılan Kafka mimarisinde, Zookeeper yalnızca Kafka'nın arka plan servisi olarak kullanılmış, doğrudan veri akışında aktif rol oynamamıştır. Ancak Kafka'nın tutarlı ve kararlı bir şekilde çalışabilmesi için Zookeeper bileşeninin kesintisiz hizmet vermesi gerekmektedir [35].



```
Komut İstemi - zkserver
Microsoft Windows [Version 10.0.19045.5965]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\Mehmet>zkserver

C:\Users\Mehmet>call "C:\Program Files\Java\java-1.8.0.201\bin\java" "-Dzookeeper.log.dir=C:\zookeeper-3.4.14\bin\." "-Dzookeeper.root.logger=INFO,CONSOLE" -cp "C:\zookeeper-3.4.14\bin\..\build\classes;C:\zookeeper-3.4.14\bin\..\build\lib\*;C:\zookeeper-3.4.14\bin\..\lib\*;C:\zookeeper-3.4.14\bin\..\conf" org.apache.zookeeper.server.quorum.QuorumPeerMain "C:\zookeeper-3.4.14\bin\..\conf\zoo.cfg"
2025-07-03 15:54:44,449 [myid:] - INFO [main:QuorumPeerConfig@136] - Reading configuration from: C:\zookeeper-3.4.14\bin\..\conf\zoo.cfg
2025-07-03 15:54:44,481 [myid:] - INFO [main:DatadirCleanupManager@78] - autopurge.snapRetainCount set to 3
2025-07-03 15:54:44,481 [myid:] - INFO [main:DatadirCleanupManager@79] - autopurge.purgeInterval set to 0
2025-07-03 15:54:44,481 [myid:] - INFO [main:DatadirCleanupManager@101] - Purge task is not scheduled.
2025-07-03 15:54:44,481 [myid:] - WARN [main:QuorumPeerMain@116] - Either no config or no quorum defined in config, running in standalone mode
2025-07-03 15:54:44,606 [myid:] - INFO [main:QuorumPeerConfig@136] - Reading configuration from: C:\zookeeper-3.4.14\bin\..\conf\zoo.cfg
2025-07-03 15:54:44,606 [myid:] - INFO [main:ZooKeeperServerMain@98] - Starting server
2025-07-03 15:54:44,637 [myid:] - INFO [main:Environment@100] - Server environment:zookeeper.version=3.4.14-4c25d480e66aadd371de8bd2fd8da255ac140bcf, built on 03/06/2019 16:18 GMT
2025-07-03 15:54:44,637 [myid:] - INFO [main:Environment@100] - Server environment:host.name=DESKTOP-V2R7G30.local
2025-07-03 15:54:44,637 [myid:] - INFO [main:Environment@100] - Server environment:java.version=1.8.0_201-2-redhat
2025-07-03 15:54:44,637 [myid:] - INFO [main:Environment@100] - Server environment:java.vendor=Oracle Corporation
2025-07-03 15:54:44,637 [myid:] - INFO [main:Environment@100] - Server environment:java.home=C:\Program Files\Java\java-1.8.0.201\jre
2025-07-03 15:54:44,637 [myid:] - INFO [main:Environment@100] - Server environment:java.class.path=C:\zookeeper-3.4.14\bin\..\build\classes;C:\zookeeper-3.4.14\bin\..\build\lib\*;C:\zookeeper-3.4.14\bin\..\zookeeper-3.4.14.jar;C:
```

Şekil 5.2. Apache Zookeeper ayağa kaldırma komutu.

5.1.3. Gerçek zamanlı veri akışı ve tahmin sonuçlarının yönetimi

Bu tez çalışmasında geliştirilen gerçek zamanlı veri analitiği mimarisi, hasta verilerinin anlık olarak işlenmesi ve tahmin sonuçlarının kullanıcılara hızlı ve güvenilir biçimde ulaştırılması amacıyla doğrudan Apache Kafka altyapısı üzerinden yürütülmektedir. Sistem, C# programlama dili ile yazılmış Kafka Producer bileşeni sayesinde laborant tarafından girilen hasta verilerini anında Kafka'ya aktarırken, Java tabanlı Kafka Consumer bileşeni bu verileri alıp analiz sürecini başlatmaktadır.

Veri akışında performans ve süreklilik öncelikli tutulmuş, böylece önceden kullanılan ara katmanlı dosya tabanlı yapılardan vazgeçilmiştir. Kafka Producer, veritabanı ya da ara dosya sorgulamadan doğrudan gerçek zamanlı veri üreticisi olarak hareket

etmekte; veri iletiminde gecikmeler minimize edilerek yüksek hız ve ölçeklenebilirlik sağlanmaktadır.

Tahmin süreci tamamlandıktan sonra, Kafka Consumer bileşeni tarafından üretilen sonuçlar, geçici buffer yapıları veya dosya sistemleri yerine doğrudan veritabanına kaydedilmektedir. Böylece hastane otomasyon sisteminde, tahmin verilerine ulaşabilmek için yalnızca bir SQL sorgusu yazmak yeterli olmaktadır. Bu yalın ve doğrudan yapı, hem bakım kolaylığı hem de sistemin içsel şeffaflığı açısından önemli avantajlar sağlamaktadır.

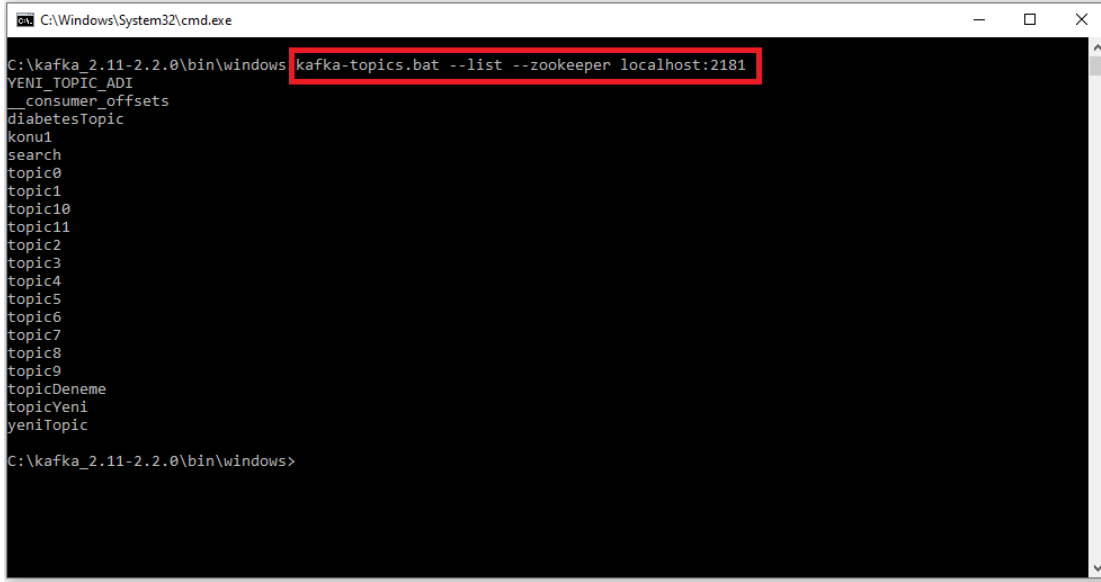
Ayrıca, sistemdeki her veri girişi bir olay (event) olarak değerlendirilmekte ve Kafka'nın olay tetiklemeli (event-driven) mimarisi sayesinde her bir ölçüm verisi anlık olarak analiz sürecini başlatmaktadır. Bu yapı, gelen verilerin sırasını koruyan **FIFO (First-In First-Out)** mekanizması ile birlikte çalışarak, işlenmemiş verilerin ardışık şekilde analiz edilmesini garanti altına almaktadır. Böylece aynı anda birden fazla hasta verisi sisteme girmiş olsa dahi, verilerin doğru sırayla ve karışıklık olmadan analiz edilmesi sağlanmaktadır.

Geliştirilen mimaride, sistem bileşenleri arasında herhangi bir doğrudan bağımlılık yaratılmaksızın, her bir katmanın yalnızca kendi sorumluluğuna odaklanması sağlanmıştır. Bu **gevşek bağlılık (loosely coupled architecture)** prensibi, bileşenler arası etkileşimi asgari düzeyde tutarken sistemin yeniden yapılandırılabilirliğini artırmakta ve bakım süreçlerini kolaylaştırmaktadır. Aynı zamanda, verilerin gerçek zamanlı işlenmesi sayesinde, kullanıcılar için sunulan tahmin sonuçları güncel ve anlık karar destek kapasitesine sahiptir.

Sonuç olarak, Kafka tabanlı bu olay odaklı mimari; düşük gecikme süresi, sıralı veri işleme, sistemler arası bağımsızlık ve yüksek performans gibi nitelikleriyle özellikle sağlık sektörü gibi zaman hassasiyetinin yüksek olduğu alanlarda **güvenilir ve ölçeklenebilir bir çözüm** sunmaktadır. Mimari tasarım, yalnızca mevcut işlevsellik için değil, aynı zamanda gelecekteki potansiyel modüllerle yapılacak entegrasyonlar için de esnek bir altyapı oluşturmaktadır.

5.2. Java ile Streaming Producer ve Consumer Yapısı

Bu çalışmada, diyabet hastalığının erken teşhisi için geliştirilen gerçek zamanlı karar destek sistemi, **Apache Kafka** kullanarak çift yönlü veri akışı sağlamakta; veri üretim



```
C:\Windows\System32\cmd.exe
C:\kafka_2.11-2.2.0\bin\windows kafka-topics.bat --list --zookeeper localhost:2181
YENI_TOPIC_AD
__consumer_offsets
diabetesTopic
konu1
search
topic0
topic1
topic10
topic11
topic2
topic3
topic4
topic5
topic6
topic7
topic8
topic9
topicDeneme
topicYeni
yeniTopic
C:\kafka_2.11-2.2.0\bin\windows>
```

Şekil 5.5. Mevcut topicleri listeleme komutu.

5.2.1. Producer bileşeni

C# programlama dili kullanılarak geliştirilen hastane otomasyon sistemindeki arayüz aracılığıyla, laborantın girdiği hasta ölçüm verilerini doğrudan Apache Kafka'ya gönderen bir Producer bileşeni olarak görev yapmaktadır. Ölçüm verileri Kafka'ya gerçek zamanlı olarak JSON formatında iletilmektedir.

Producer bileşeni, verileri arka planda anlık olarak Kafka broker'larına aktarır ve bu sayede dosya sistemine veya ara katmanlara ihtiyaç olmadan veri iletimini sağlar. Gönderilen her mesaj, Kafka tarafından benzersiz bir offset ile işaretlenerek sistem genelinde sıralı ve güvenilir bir veri akışı garanti edilir. Ayrıca, acks=all, retries ve linger.ms gibi kritik ayarlar sayesinde mesaj teslimat güvenliği üst düzeye çıkarılır; bu da yüksek trafikli ortamda veri kaybını önler ve sistemin kararlılığını artırır. [14]

Bu doğrudan Kafka entegrasyonu, gerçek zamanlı veri işleme sürecinin performansını maksimize ederken, sistem bileşenleri arasındaki bağımlılığı azaltarak daha esnek ve ölçeklenebilir bir mimari oluşturulmasına olanak tanır.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Confluent.Kafka; // Kafka ile iletişim kurmak için gerekli kütüphane
using Newtonsoft.Json; // Nesneleri JSON formatına çevirmek için kullanılan kütüphane

namespace data_gonder_sa
{
    // Bu sınıf, Kafka'ya veri (mesaj) göndermek için kullanılır.
    public class KafkaProducer
    {
        // Kafka sunucusunun adresi (localhost = bu bilgisayar)
        // ":9092" Kafka'nın varsayılan port numarasıdır.
        private static readonly string bootstrapServers = "localhost:9092";

        // Mesajların gönderileceği konu (topic) adı.
        private static readonly string topicName = "diabetesTopic";

        // Bu metod, parametre olarak aldığı "Person" nesnesini Kafka'ya gönderir.
        public static void SendToKafkaConsumer(Person person)
        {
            // Kafka Üretici (producer) yapılandırması oluşturuluyor.
            // BootstrapServers: Kafka'ya nasıl bağlanacağını söyler.
            var config = new ProducerConfig
            {
                BootstrapServers = bootstrapServers
            };

            // Producer (veri Üretici) nesnesi oluşturuluyor.
            // Null: Mesajın anahtar (key) kısmı kullanılmayacak.
            // string: Mesajın değeri (value) kısmı string (metin) olacak.
            using (var producer = new ProducerBuilder<Null, string>(config).Build())
            {
                // Person nesnesi JSON formatına dönüştürülüyor.
                // Örneğin: { "Pregnancies": "1.5", "Age": 25 }
                string json = JsonConvert.SerializeObject(person);

                // Kafka'ya gönderilecek mesaj nesnesi oluşturuluyor.
                var message = new Message<Null, string>
                {
                    Value = json
                };

                try
                {
                    // Mesaj belirtilen "topic"e gönderiliyor.
                    producer.Produce(topicName, message, deliveryReport =>
                    {
                        // Bu kısım, gönderim tamamlandığında çalışır.
                        if (deliveryReport.Error.IsError)
                        {
                            // Eğer hata varsa, hata nedeni ekrana yazdırılır.
                            Console.WriteLine("Hata: " + deliveryReport.Error.Reason);
                        }
                        else
                        {
                            // Başarılı olursa, gönderilen mesaj ekrana yazdırılır.
                            Console.WriteLine("Kafka'ya gönderildi: " + deliveryReport.Value);
                        }
                    });
                }
                catch (Exception ex)
                {
                    // Flush: Mesajın gerçekten Kafka'ya ulaştığından emin olmak için bekler. (Aksi halde uygulama bitmeden mesaj ulaşmayabilir.)
                    producer.Flush(TimeSpan.FromSeconds(5));
                }
            }
        }
    }
}

```

Şekil 5.6. Producer tarafı kod yapısı.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace data_gonder_sa
8  {
9      public class Person
10     {
11         public string Pregnancies { get; set; }
12         public string Glucose { get; set; }
13         public string BloodPressure { get; set; }
14         public string SkinThickness { get; set; }
15         public string Insulin { get; set; }
16         public string BMI { get; set; }
17         public string Diabetes { get; set; }
18         public string Age { get; set; }
19         public string Outcome { get; set; }
20     }
21 }
22

```

Şekil 5.7. Producer tarafı kod yapısı (Devamı-1).

```

39 private void btn_kaydet_Click(object sender, EventArgs e)
40 {
41     Person newPerson = new Person
42     {
43         Pregnancies = txt_pregnancies.Text,
44         Glucose = txt_Glucose.Text,
45         BloodPressure = txt_BloodPressure.Text,
46         SkinThickness = txt_SkinThickness.Text,
47         Insulin = txt_Insulin.Text,
48         BMI = txt_BMI.Text,
49         Diabetes = txt_DiabetesPedigreeFunc.Text,
50         Age = txt_Age.Text,
51         Outcome = "-1"
52     };
53
54     KafkaProducer.SendToKafkaConsumer(newPerson);
55     MessageBox.Show("Veri Kafka Consumer'a gönderildi!");
56
57     /**Data Eklendikten sonra --- Biraz bekleyelim Makine Öğrenmesi buna bir tahminleme gerçekleştirsin.*/
58     Thread.Sleep(1000); // 1000 milisaniye bekle (1 saniye)
59

```

Şekil 5.8. Producer tarafı kod yapısı (Devamı-2).

5.2.2. Consumer bileşeni

Java ile geliştirilen Kafka Consumer bileşeni, tekil bir KafkaConsumer nesnesi olarak yapılandırılmıştır ve sistemde yalnızca bir adet tüketici (consumer) aktif olarak görev yapmaktadır. Bu bileşen, diabetesTopic başlığını düzenli olarak dinleyerek her 500 ms'de bir poll() işlemi gerçekleştirmekte ve yeni veri olup olmadığını kontrol etmektedir.

Kafka üzerinden alınan her JSON mesajı, Gson kütüphanesi yardımıyla Person nesnesine dönüştürülür. Ardından bu veri, herhangi bir ek ön işleme adımı uygulanmaksızın sırasıyla beş farklı makine öğrenmesi algoritmasına (Naive Bayes, KNN, KNC, Lojistik Regresyon ve AdaBoost) gönderilir.

Gerçek zamanlı teşhis sisteminin temel amacı olan **kararlılık** ve **güvenilirlik** ilkeleri doğrultusunda, tüm bu tahmin işlemleri **tekil iş parçacığı (single-threaded)** yapı ile, **sıralı ve deterministik** bir şekilde yürütülmektedir. Bu tasarım tercihi, özellikle ardışık tahmin işlemlerinde oluşabilecek zamanlama problemlerini ortadan kaldırmakta ve her bir hastaya ait verinin işlem bütünlüğünü koruyarak analiz edilmesini sağlamaktadır.

Tekil algoritmalarla elde edilen tahmin sonuçlarına ek olarak, bu çalışmada **iki farklı topluluk (ensemble) modeli** de geliştirilmiştir. Bunlardan ilki, **çoğunluk oylaması (majority voting)** esasına dayanan basit ama etkili bir yöntemle çalışmakta; her algoritmanın tahmin sonucu bir oy olarak kabul edilerek, en fazla oyu alan sınıf sistemin nihai tahmini olarak belirlenmektedir. İkinci topluluk modeli ise, **ağırlıklı oylama (weighted voting)** yöntemine dayanmaktadır. Bu modelde, her algoritmaya

doğruluk performanslarına göre ağırlıklar atanmakta ve tahmin süreci bu ağırlıklı oyların toplamına göre yönlendirilmektedir.

Topluluk modellerinin entegrasyonu, yalnızca performans artışı sağlamakla kalmayıp, aynı zamanda tekil modellerin yapısal eksikliklerini de dengelemeyi amaçlamaktadır. Örneğin; parametrik varsayımlara dayanan Naive Bayes ile örnek yoğunluğu duyarlılığı yüksek olan KNN algoritmaları, bazı durumlarda birbirine zıt kararlar üretebilirken, ensemble yaklaşımı bu tür çelişkileri minimize eden daha **genellenebilir bir tahmin stratejisi** ortaya koymaktadır.

Bu sayede, sistemin hem bireysel hem de kolektif öğrenme modellerinden yararlanması mümkün olmuş; gerçek zamanlı teşhis mimarisi, sadece yüksek doğruluk oranı değil, aynı zamanda **algoritmik çeşitlilik, tutarlılık ve karar güvenilirliği** gibi yönlerden de güçlendirilmiştir.

Ayrıca Consumer bileşeni algoritmaların ürettiği tahmin sonuçlarını, doğrudan veritabanına kaydeder.

```
StreamingConsumer.java x
1 // JSON olarak Kafka'ya gelen veriyi okuyabilmek için ilgili library
2 import com.google.gson.Gson;
3 // Kafka Consumer libs
4 import org.apache.kafka.clients.consumer.ConsumerConfig;
5 import org.apache.kafka.clients.consumer.ConsumerRecord;
6 import org.apache.kafka.clients.consumer.ConsumerRecords;
7 import org.apache.kafka.clients.consumer.KafkaConsumer;
8 import org.apache.kafka.common.serialization.StringDeserializer; // Düzgün deserializer sınıfını içe aktar
9 // Spark MachineLearning libs
10 import org.apache.spark.ml.classification.NaiveBayesModel;
11 import org.apache.spark.ml.feature.StringIndexer;
12 import org.apache.spark.ml.feature.StringIndexerModel;
13 import org.apache.spark.ml.feature.VectorAssembler;
14 // Spark Sql and Spark Streaming API libs
15 import org.apache.spark.sql.*;
16 import org.apache.spark.sql.streaming.StreamingQuery;
17 import org.apache.spark.sql.streaming.StreamingQueryException;
18 import org.apache.spark.sql.types.DataTypes;
19 import org.apache.spark.sql.types.StructField;
20 import org.apache.spark.sql.types.StructType;
21 // Java IO-Input_Output libs
22 import java.io.BufferedWriter;
23 import java.io.FileWriter;
24 import java.io.IOException;
25 import java.util.*;
26 // Java time libs
27 import java.time.Duration;
```

Şekil 5.9. Consumer tarafı kullanılan kütüphaneler.

```

public class StreamingConsumer
{
    public static void main(String[] args) throws StreamingQueryException, IOException
    {
        /*
         * Bu class; KAFKA'ya anlik olarak aktarilan datalarin anlik olarak cekilip MODEL'e gidyirilmesi islemi yapilmaktadir.
         */

        // Kafka Producer ve Consumer haberlesmesi icin topic
        String topicName="diabetesTopic";

        // Kaydedilen model yukleniyor
        NaiveBayesModel loadedModel = NaiveBayesModel.Load( path: "models/naive_bayes_model");

        /* Datalari Anlik Olarak KAFKA'dan Cekme ve ModeLe Uygulama Asamalari*/
        // Kafka Consumer ayarları
        Properties config=new Properties(); // Consumer ayarlamalarını yapabilmek için ilgili properties sınıfından nesne turetildi
        config.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092"); // baglanacağı server/port yolu
        config.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, new StringDeserializer().getClass().getName()); // Producer tarafından serialize edilen key deserialize ediliyor
        config.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, new StringDeserializer().getClass().getName()); // Producer tarafından serialize edilen data deserialize edili
        config.put(ConsumerConfig.GROUP_ID_CONFIG, "groupName1"); //group1 yerine istedigini ismi verebilirsin. Bu yalnızca consumer grubunu temsil ediyor.
        config.put(ConsumerConfig.CLIENT_ID_CONFIG, "clientName1"); // grup içerisindeki hangi client consume yapıyor
        //config.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest"); // En eski kayıtları al

        // Simdi verileri ekrana yazma asamasına gecelim
        // Burada bir KAFKA CONSUMER olusturduk ve KONFIGÜRASYON islemlerini yukarıdaki gibi yap demis olduk.
        KafkaConsumer<String, String> kafkaConsumer=new KafkaConsumer<String, String>(config);
        // Simdi bu CONSUMER'in bir TOPIC'e SUBSCRIBE-ABONE olmasi lazim.
        kafkaConsumer.subscribe(Arrays.asList(topicName)); // topic topic'ine abone oldu

        // dizayn ayari icin BASLIK ekrana bir kere yazilsin:
        int baslikKontrol=0;

        // Topic'ten gelen datalari dinliyor
        while(true)
        {

```

Şekil 5.10. Consumer tarafı kod yapısı.

```

// dizayn ayari icin BASLIK ekrana bir kere yazilsin:
int baslikKontrol=0;

// Topic'ten gelen datalari dinliyor
while(true)
{
    // Gerçek zamanlı olarak verileri çekiyor.
    ConsumerRecords<String, String> records = kafkaConsumer.poll(Duration.ofMillis(100)); // 100 ms bekle

    for (ConsumerRecord<String, String> rec : records) // Anlik olarak gelen datalari aliyoruz
    {
        String satir = rec.value(); // Okunan satir
        //System.out.println("Gelen veri: " + satir); // Satir satir(Anlik) gelen veriyi kontrol et

        // Veriler JSON formatında gelmişti. Json Formatında gelen bu verileri normale cevirelim
        List<Person> modellist = new ArrayList<Person>();
        Gson gson = new Gson();
        Person person = gson.fromJson(satir, Person.class);

        // Gelen verileri alalım ve MODELe(Algoritmalara) uygun formata getirerek uygulayalım.
        // [DATA]
        String p1 = person.getPregnancies(), p2 = person.getGlucose(), p3 = person.getBloodPressure(), p4 = person.getSkinThickness(),
        p5 = person.getInsulin(), p6 = person.getBMI(), p7 = person.getDiabetes(), p8 = person.getAge(), p9 = person.getOutcome();
        Double p1_ = Double.valueOf(p1);
        Double p2_ = Double.valueOf(p2);
        Double p3_ = Double.valueOf(p3);
        Double p4_ = Double.valueOf(p4);
        Double p5_ = Double.valueOf(p5);
        Double p6_ = Double.valueOf(p6);
        Double p7_ = Double.valueOf(p7);
        Double p8_ = Double.valueOf(p8);
        Double p9_ = Double.valueOf(p9);

```

Şekil 5.11. Consumer tarafı kod yapısı (Devamı-1).

```

// Tek veri:
Row singleData = RowFactory.create(p1_, p2_, p3_, p4_, p5_, p6_, p7_, p8_, p9_);

// Sema tanımlanıyor
StructType schema = DataTypes.createStructType(new StructField[]{
    DataTypes.createStructField( name: "Pregnancies", DataTypes.DoubleType, nullable: false),
    DataTypes.createStructField( name: "Glucose", DataTypes.DoubleType, nullable: false),
    DataTypes.createStructField( name: "BloodPressure", DataTypes.DoubleType, nullable: false),
    DataTypes.createStructField( name: "SkinThickness", DataTypes.DoubleType, nullable: false),
    DataTypes.createStructField( name: "Insulin", DataTypes.DoubleType, nullable: false),
    DataTypes.createStructField( name: "BMI", DataTypes.DoubleType, nullable: false),
    DataTypes.createStructField( name: "DiabetesPedigreeFunction", DataTypes.DoubleType, nullable: false),
    DataTypes.createStructField( name: "Age", DataTypes.DoubleType, nullable: false),
    DataTypes.createStructField( name: "Outcome", DataTypes.DoubleType, nullable: false)
});

// Tek veriyi DataFrame'e donusturuyoruz
List<Row> data = Arrays.asList(singleData);
Dataset<Row> raw_data = sparkSession.createDataFrame(data, schema);

// Sutun listesi
String[] headerList = {"Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"}; // Verisetinde
//List<String> headers = Arrays.asList(headerList); // Yukarida array seklinde bulunan Basliklari List e attik, islem yapmasi daha kolay
List<String> headersResult = new ArrayList<>(); // Donustürölmüş sütunları tutacak

// Tüm sütunlara StringIndexer uygulamak
for (String h : headerList) {
    if (h.equals("Outcome")) {
        // Outcome için kaydedilen label indexer'ı yükle
        StringIndexerModel indexerModel = StringIndexerModel.load( path: "models/indexer_label").setHandleInvalid("keep"); // Unseen değerler için
        raw_data = indexerModel.transform(raw_data);
        headersResult.add("label");
    } else {
        // Diğer sütunlar için kaydedilen indexer'ı yükle
        StringIndexerModel indexerModel = StringIndexerModel.load( path: "models/indexer_" + h.toLowerCase()).setHandleInvalid("keep"); // Unseen değerler için
        raw_data = indexerModel.transform(raw_data);
        headersResult.add(h.toLowerCase() + "_cat");
    }
}
}

```

Şekil 5.12. Consumer tarafı kod yapısı (Devamı-2).

```

System.out.println("-----");
System.out.println("Data:"+p1_+" "+p2_+" "+p3_+" "+p4_+" "+p5_+" "+p6_+" "+p7_+" "+p8_);
//System.out.println("*****");
//---Naive Bayes Algoritması Baslangic*****
NaiveBayesAlgorithm naiveBayesAlgorithm=new NaiveBayesAlgorithm(p1_,p2_,p3_,p4_,p5_,p6_,p7_,p8_);
int predictionBayes= naiveBayesAlgorithm.naiveBayesTahmin();
String tahminBayesAlgoritması = Integer.toString(predictionBayes);
System.out.println("Naive Bayes Tahmin:"+tahminBayesAlgoritması);
double dogrulukOraniBayes= naiveBayesAlgorithm.dogrulukOraniHesapla();
//System.out.println("Naive Bayes Doğruluk Oranı:"+naiveBayesAlgorithm.dogrulukOraniHesapla());
//---Bayes Algoritması Bitis

//System.out.println("*****");
//---KNN Algoritması Baslangic*****
KNNAlgorithm knnAlgorithm=new KNNAlgorithm(p1_,p2_,p3_,p4_,p5_,p6_,p7_,p8_);
int predictionKNN = knnAlgorithm.knnUygula( k: 3); // K=3
String tahminKNNAlgoritması = Integer.toString(predictionKNN);
System.out.println("KNN Tahmin:"+tahminKNNAlgoritması);
double dogrulukOraniKNN= knnAlgorithm.dogrulukOraniHesapla( k: 3);
//System.out.println("KNN Doğruluk Oranı:"+knnAlgorithm.dogrulukOraniHesapla(3));
//---KNN Algoritması Bitis

//System.out.println("*****");
//---KNC Algoritması Baslangic*****
KNCAlgorithm kncAlgorithm=new KNCAlgorithm(p1_,p2_,p3_,p4_,p5_,p6_,p7_,p8_);
int predictionKNC = kncAlgorithm.kncUygula();
String tahminKNCAlgoritması = Integer.toString(predictionKNC);
System.out.println("KNC Tahmin:"+tahminKNCAlgoritması);
double dogrulukOraniKNC= kncAlgorithm.dogrulukOraniHesapla();
//System.out.println("KNC Doğruluk Oranı:"+kncAlgorithm.dogrulukOraniHesapla());
//---KNC Algoritması Bitis

//---LogisticRegression Algoritması Baslangic*****
// Lojistik regresyon algoritmasını başlat
LogisticRegressionAlgorithm lojistikRegresyon = new LogisticRegressionAlgorithm(p1_,p2_,p3_,p4_,p5_,p6_,p7_,p8_);
// Modeli eğit

```

Şekil 5.13. Consumer tarafı kod yapısı (Devamı-3, Tüm kodlar paylaşılmamıştır.).

5.2.3. Sistem entegrasyonu ve gerçek zamanlılık

Bu tez kapsamında geliştirilen sistemde, tahmin süreci yalnızca Kafka üzerinden alınan verilerin anlık olarak analiz edilmesiyle sınırlı kalmamakta; elde edilen sonuçların kullanıcıya ulaştırılması ve sistem veritabanına kalıcı olarak kaydedilmesiyle tamamlanmaktadır. Java tabanlı Kafka Consumer bileşeni tarafından analiz edilen her hasta verisi, ilgili tahmin çıktılarıyla birlikte doğrudan MSSQL veritabanına kaydedilmektedir. Bu kayıt işlemi sayesinde, sistemin C# tabanlı hastane otomasyon arayüzü, yalnızca bir SQL sorgusu ile en güncel tahmin sonuçlarına erişebilmekte ve bunları kullanıcıya anlık olarak sunabilmektedir.

Gerçek zamanlılık ilkesine tam uyum sağlanabilmesi adına, sistemde önceden kullanılan **CSV dosyası tabanlı geçici tampon yapılar** tamamen terk edilmiştir. Böylece veriler, üretimden tüketime kadar herhangi bir ara katman veya manuel kontrol süreci olmaksızın, **uçtan uca (end-to-end)** bir akış içerisinde ilerlemektedir. Bu, sadece işlem hızını artırmakla kalmamış, aynı zamanda hata kaynaklarını minimize ederek sistem güvenilirliğini de yükseltmiştir.

Ayrıca, Apache Kafka'nın **FIFO (First-In-First-Out)** kuyruk yapısı, veri sırasının korunmasını ve işlem bütünlüğünün sağlanmasını garanti altına almaktadır. Bu yapı, özellikle sağlık verileri gibi zaman duyarlı bilgilerin sırasıyla işlenmesi gerektiği durumlar için büyük avantaj sunmaktadır. Öte yandan, Kafka'nın **offset yönetimi** sayesinde, tüketici (consumer) tarafında ani bir kesinti veya sistem hatası yaşansa dahi, veri kaybı yaşanmamakta ve işlem kaldığı noktadan itibaren devam ettirilebilmektedir [36]. Bu da sistemin **yüksek erişilebilirlik (high availability)** ve **hata toleransı (fault tolerance)** özelliklerini desteklemektedir.

Gerçek zamanlı sistemde her bir hasta verisi, Kafka Consumer tarafından **tekil ve senkron bir biçimde** işlenmektedir. Bu tasarım tercihi, özellikle eş zamanlı çoklu veri girişlerinde bile kontrol edilebilirlik ve izlenebilirliği koruyarak **deterministik davranışı** mümkün kılmaktadır. Performans açısından daha yüksek işlem kapasitesi sunabilecek çok iş parçacıklı yapı ise, yalnızca sistem testleri ve performans karşılaştırmaları gibi senaryolarda kullanılmak üzere yedekte tutulmuştur. Gerçek zamanlı teşhis sürecinde **kararlılık ve güvenilirlik**, performanstan daha öncelikli olarak değerlendirilmiştir.

5.2.4. Güvenilirlik ve yüksek performans

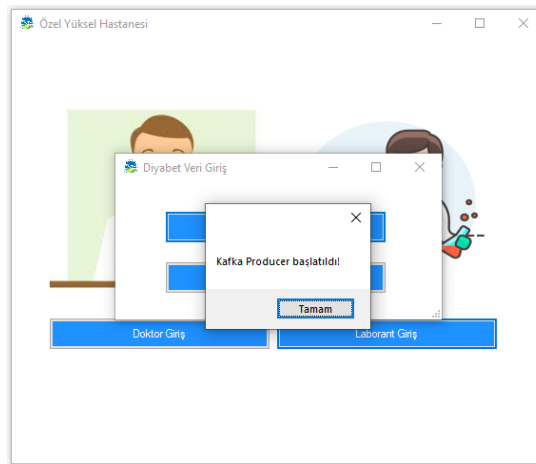
Kafka'nın bölümlendirme (partitioning) yapısı sayesinde veri paralellik bazında işlenebilir, offset ile takip sayesinde veri kaybı veya sıra dışı işlemler önlenir. Producer garantili veri teslimi sağlarken, Consumer ise kuyruk yapısı ve offsetler sayesinde kesintilere aldırış etmeden tahminlemelerini üretebilir. Bu sayede **doğru, kesintisiz ve güvenilir gerçek zamanlı bir işleme mimarisi** elde edilmiştir.

5.2.5. Gerçek zamanlı veri akış süreci

Bu çalışmada geliştirilen mimari içerisinde, Kafka Producer ve Kafka Consumer bileşenlerinin başlatılmasının ardından gerçek zamanlı veri akışı süreci başlatılmaktadır. Öncelikle Kafka Consumer, belirli bir **topic** üzerinden veri dinlemeye hazır hale getirilir. Ardından Kafka Producer, laborant tarafından hastane otomasyon sistemine bir kayıt girildiği anda yine aynı topic üzerinden Kafka Consumer'a iletimini sağlar (Şekil 5.15 ve Şekil 5.16).

```
Run StreamingConsumer x
25/09/19 13:44:02 INFO ContextCleaner: Cleaned accumulator 57
25/09/19 13:44:02 INFO ContextCleaner: Cleaned accumulator 52
25/09/19 13:44:02 INFO ContextCleaner: Cleaned accumulator 59
25/09/19 13:44:02 INFO ContextCleaner: Cleaned accumulator 54
25/09/19 13:44:02 INFO ContextCleaner: Cleaned accumulator 64
25/09/19 13:44:02 INFO ContextCleaner: Cleaned accumulator 78
25/09/19 13:44:02 INFO ContextCleaner: Cleaned accumulator 69
25/09/19 13:44:02 INFO ContextCleaner: Cleaned accumulator 75
25/09/19 13:44:02 INFO ContextCleaner: Cleaned accumulator 55
25/09/19 13:44:02 INFO BlockManagerInfo: Removed broadcast_3_piece0 on DESKTOP-V2R7630.mshome.net:53390 in memory (size: 21.9 KB, free: 886.8 MB)
25/09/19 13:44:02 INFO ContextCleaner: Cleaned accumulator 61
25/09/19 13:44:02 INFO Metadata: Cluster ID: p-P22KEQ5JWwF8g0SpWgA
25/09/19 13:44:02 INFO AbstractCoordinator: [Consumer clientId=clientName1, groupId=groupName1] Discovered group coordinator DESKTOP-V2R7630.mshome.net:9092 (id: 2147483647 rack: null)
25/09/19 13:44:02 INFO AbstractCoordinator: [Consumer clientId=clientName1, groupId=groupName1] Revoking previously assigned partitions []
25/09/19 13:44:02 INFO AbstractCoordinator: [Consumer clientId=clientName1, groupId=groupName1] (Re-)joining group
25/09/19 13:44:02 INFO AbstractCoordinator: [Consumer clientId=clientName1, groupId=groupName1] (Re-)joining group
25/09/19 13:44:02 INFO AbstractCoordinator: [Consumer clientId=clientName1, groupId=groupName1] Successfully joined group with generation 11
25/09/19 13:44:02 INFO AbstractCoordinator: [Consumer clientId=clientName1, groupId=groupName1] Setting newly assigned partitions: topic5-1, topic5-0, topic5-2
```

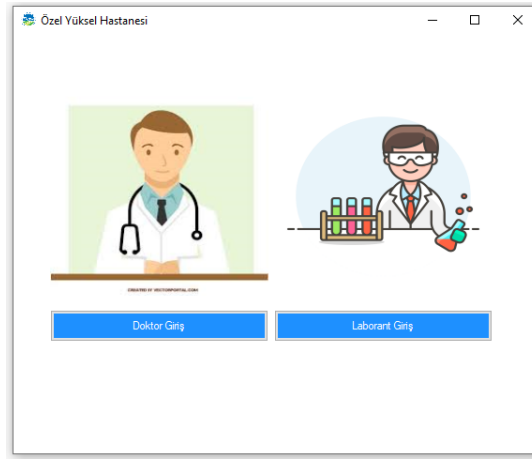
Şekil 5.14. Kafka Consumer başlatıldı.



Şekil 5.15. Kafka Producer başlatıldı.

Bu veri akışının sağlıklı şekilde ilerleyebilmesi için hastane otomasyon arayüzünden sisteme yeni bir hasta verisi girilir. Bu veri, laborant tarafından kullanıcı arayüzü

üzerinden girildikten sonra Kafka Producer aracılığıyla ilgili konfigürasyonlar ile Consumer'e iletilir (Şekil 5.17 ve 5.18).



Şekil 5.16. Hastane Otomasyon Sistemi arayüzü.

Veri Giriş | Tek Data

Tahlil Verisi

Hamilelik sayısı: 2

Glikoz seviyesi: 61

Kan basıncı: 66

Vücut kitle indeksi: 25.2

Cilt kalınlığı (mm cinsinden): 39

İnsülin seviyesi: 179

Dişabet Fonksiyonu: (Ailede dişabet öyküsü fonk.) 0.268

Yaş: 38

Verileri Verile | Tahlil Giriş | Kayıtları Güncelle

Veri Kafka Consumer'a gönderildi!

Tamam

Hasta ID	Hamilelik Sayısı	Glikoz	Kan Basıncı	Deri Kalınlığı	İnsülin	Vücut Kitle İndeksi	KNN Tahmini	KNC Tahmini	Lojistik Tahmin	AdaBoost Tahmini	Ağırlık Ensemble Tahmini	Çoklu Ensemble Tahmini	Gerçek Sonuç
5	2	135	40	40	178	5.6	0.262	21	Sağlıklı	Sağlıklı	Hasta	Hasta	Hasta
6	3	66	66	29	95	34.0	0.232	32	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	Sağlıklı
7	3	100	47	21	172	2.0	0.788	28	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	Sağlıklı
8	4	141	63	17	172	50.4	0.293	43	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	Sağlıklı
129	7	113	72	21	178	39.3	0.013	40	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	Sağlıklı
17	4	170	69	24	176	40.0	0.232	32	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	Hasta
107	4	57	57	29	180	37.9	0.444	24	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı
97	4	199	61	44	650	33.2	0.612	32	Sağlıklı	Hasta	Sağlıklı	Hasta	Sağlıklı
64	5	159	68	29	179	39.7	0.599	45	Sağlıklı	Hasta	Sağlıklı	Hasta	Hasta
182	6	53	37	8	688	34.6	0.013	22	Sağlıklı	Sağlıklı	Hasta	Sağlıklı	Sağlıklı
80	3	96	71	38	167	37.6	0.688	39	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	Sağlıklı
66	6	143	48	43	177	31.4	0.291	40	Sağlıklı	Hasta	Hasta	Hasta	Hasta
67	2	147	59	32	174	37.2	0.013	31	Sağlıklı	Hasta	Sağlıklı	Hasta	Hasta
195	6	102	84	19	174	42.5	0.463	27	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	Sağlıklı
184	5	131	43	29	174	38.9	0.403	25	Sağlıklı	Sağlıklı	Hasta	Sağlıklı	Sağlıklı

Şekil 5.17. Hastane Otomasyonuna örnek hasta tahlil verisi girişi.

Yazılan bu veri Kafka Producer tarafından anlık olarak algılanır ve Kafka Consumer bileşenine iletilir (Şekil 5.18).

```
25/09/19 13:48:42 INFO AbstractCoordinator: [Consumer clientId=clientName1, groupId=groupName1] Successfully joined group with generation 13
25/09/19 13:48:42 INFO ConsumerCoordinator: [Consumer clientId=clientName1, groupId=groupName1] Setting newly assigned partitions: topic5-1, topic5-0, topic5-2
-----
Data:2.0,61.0,66.0,39.0,179.0,25.2,0.268,38.0
```

Şekil 5.18. Kafka Producer, girilen veriyi Consumer'a iletti.

Consumer bileşeni bu veriyi analiz eder, sonuçları hesaplar ve veritabanına kaydeder (Şekil 5.19).

```
Run StreamingConsumer x
25/09/19 13:48:42 INFO ContextCleaner: Cleaned accumulator 62
25/09/19 13:48:42 INFO ContextCleaner: Cleaned accumulator 59
25/09/19 13:48:42 INFO BlockManagerInfo: Removed broadcast_3_piece0 on DESKTOP-V2R7630.mshome.net:53516 in memory (size: 21.9 MB, free: 886.8 MB)
25/09/19 13:48:42 INFO ContextCleaner: Cleaned accumulator 71
25/09/19 13:48:42 INFO ContextCleaner: Cleaned accumulator 73
25/09/19 13:48:42 INFO Metadata: Cluster ID: p-P22KEQ5jWwWf8g0SpWgA
25/09/19 13:48:42 INFO AbstractCoordinator: [Consumer clientId=clientName1, groupId=groupName1] Discovered group coordinator DESKTOP-V2R7630.mshome.net:9092 (id: 2147483647 rack: null)
25/09/19 13:48:42 INFO ConsumerCoordinator: [Consumer clientId=clientName1, groupId=groupName1] Revoking previously assigned partitions []
25/09/19 13:48:42 INFO AbstractCoordinator: [Consumer clientId=clientName1, groupId=groupName1] (Re-)joining group
25/09/19 13:48:42 INFO AbstractCoordinator: [Consumer clientId=clientName1, groupId=groupName1] (Re-)joining group
25/09/19 13:48:42 INFO AbstractCoordinator: [Consumer clientId=clientName1, groupId=groupName1] Successfully joined group with generation 13
25/09/19 13:48:42 INFO ConsumerCoordinator: [Consumer clientId=clientName1, groupId=groupName1] Setting newly assigned partitions: topic5-1, topic5-0, topic5-2
-----
Data:2.0,61.0,66.0,39.0,179.0,25.2,0.268,38.0
Naive Bayes Tahmin:0
KNN Tahmin:0
KNC Tahmin:0
Logistic Tahmin: 0
AdaBoost Tahmin: 0
Çoğunluk Ensemble Tahmin: 0
Ağırlıklı Ensemble Tahmin: 0
Bağlantı başarılı!
Kayıt başarıyla veritabanına eklendi!
|
```

Şekil 5.19. Consumer veriyi analiz etti ve çıktıyı kaydetti.

Son olarak elde edilen tahmin sonuçlarını hem hastane otomasyon sisteminde hem de veritabanında anlık olarak görüntüleyebiliriz (Şekil 5.20 ve 5.21).

Tahsil Verisi

Hamilelik sayısı:

61

Glikoz seviyesi:

66

Kan basıncı:

25.2

Vücut kitle indeksi:

39

Cilt kalınlığı (mm cinsinden):

179

İnsülin seviyesi:

0.268

Dişabet Fonksiyonu:

38

(Ailede dişabet öyküsü fonk.)

Yaş:

Verileri Yenile

Tahsil Giriş

Kayıtları Güncelle

Hasta ID	Hamilelik Sayısı	Glikoz	Kan Basıncı	Deri Kalınlığı	İnsülin	Vücut Kitle İndeksi	Dişabet Fonksiyonu	Yaş	Bayes Tahmini	KNN Tahmini	KNC Tahmini	Logistik Tahmin	AdaBoost Tahmini	Ağırlıklı Ensemble Tahmini	Çoğunluk Ensemble Tahmini	Gerçek Sonuç
779	7	152	73	31	173	41.5	0.516	36	Sağlıklı	Hasta	Hasta	Sağlıklı	Hasta	Hasta	Hasta	
496	3	96	71	38	167	37.6	0.688	39	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	
932	2	139	65	30	282	42.1	1.037	21	Sağlıklı	Sağlıklı	Hasta	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	
686	3	120	49	15	178	46.7	0.013	24	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	
768	2	147	59	32	174	37.2	0.013	31	Sağlıklı	Hasta	Hasta	Sağlıklı	Hasta	Hasta	Hasta	
515	1	59	45	15	115	36.0	0.047	21	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	
550	2	190	70	28	174	43.7	0.074	27	Sağlıklı	Hasta	Hasta	Sağlıklı	Hasta	Hasta	Hasta	
121	6	82	63	37	175	44.9	0.566	27	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	
616	0	154	61	28	171	44.9	0.38	43	Sağlıklı	Hasta	Hasta	Hasta	Hasta	Hasta	Hasta	
529	0	62	65	18	170	34.8	0.204	23	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	
477	3	92	60	31	360	40.8	0.958	24	Sağlıklı	Sağlıklı	Hasta	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	
230	2	61	66	39	179	25.2	0.268	38	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	
82	2	61	66	39	179	25.2	0.268	38	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	Sağlıklı	

Şekil 5.20. Tahmin sonucu arayüzde görüntülendi.

id	person_id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree	Age	BayesPrediction	KNNPrediction	KNCPrediction	LogisticPrediction	AdaBoostPred...	AğırlıklıEmm...	ÇokluKarar...	True
10000	5	2	135	40	40	178	5.6	0.725	29	1	0	0	1	1	1	1	NAİL
10021	6	3	66	66	29	95	34.0	0.262	21	0	0	0	0	0	0	0	NAİL
10022	7	3	100	47	21	172	2.0	0.232	32	0	1	0	0	0	0	0	NAİL
10023	8	4	141	63	17	172	50.4	0.788	28	0	1	1	0	0	0	0	NAİL
10025	129	7	113	72	21	178	39.3	0.913	40	0	1	1	0	0	0	0	NAİL
10027	17	4	170	68	24	176	40.0	0.293	43	0	1	1	0	0	1	1	NAİL
10028	107	4	57	57	29	180	37.9	0.444	24	0	0	0	0	0	0	0	NAİL
10029	97	4	199	61	44	650	33.2	0.612	32	0	0	1	0	1	0	1	NAİL
10030	64	5	159	68	29	179	39.7	0.599	45	0	1	1	0	1	1	1	NAİL
10031	162	6	53	37	8	688	34.8	0.013	22	0	0	1	0	0	0	0	NAİL
10032	80	3	96	71	38	167	37.6	0.688	39	0	1	0	0	0	0	0	NAİL
10033	66	6	143	48	43	177	31.4	0.291	40	0	1	1	1	1	1	1	NAİL
10034	67	2	147	59	32	174	37.2	0.013	31	0	1	1	0	1	1	1	NAİL
10035	195	6	102	84	19	174	42.5	0.463	27	0	1	0	0	1	0	0	NAİL
10036	184	5	131	43	29	174	38.9	0.403	25	0	0	1	0	1	0	0	NAİL
10037	523	0	154	61	28	171	44.9	0.30	43	0	1	1	1	1	1	1	NAİL
10038	329	0	69	75	39	170	41.6	0.042	28	0	0	0	0	0	0	0	NAİL
10039	779	7	152	73	31	173	41.5	0.516	36	0	1	1	0	1	1	1	NAİL
10040	496	3	96	71	38	167	37.6	0.688	39	0	1	0	0	0	0	0	NAİL
10041	932	2	139	65	30	282	42.1	1.037	21	0	0	1	0	1	0	0	NAİL
10042	686	3	120	49	15	178	46.7	0.013	24	0	1	0	0	1	0	0	NAİL
10043	768	2	147	59	32	174	37.2	0.013	31	0	1	1	0	1	1	1	NAİL
10044	515	1	59	45	15	115	36.0	0.047	21	0	0	0	0	0	0	0	NAİL
10045	550	2	190	70	28	174	43.7	0.074	27	0	1	1	0	1	1	1	NAİL
10046	121	6	82	63	37	175	44.9	0.566	27	0	0	0	0	0	0	0	NAİL
10047	616	0	154	61	28	171	44.9	0.30	43	0	1	1	1	1	1	1	NAİL
10048	529	0	62	65	18	170	34.8	0.394	23	0	0	0	0	0	0	0	NAİL
10049	477	3	92	60	31	360	40.8	0.958	24	0	0	1	0	0	0	0	NAİL
10050	239	2	61	68	39	179	25.2	0.268	38	0	0	0	0	0	0	0	NAİL
10051	82	2	61	66	39	179	25.2	0.268	38	0	0	0	0	0	0	0	NAİL
10052	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL	NAİL

Şekil 5.21. Tahmin sonucu veritabanına kaydedildi.

Bu süreç sayesinde sistemde uçtan uca bir **gerçek zamanlı veri işleme ve tahmin altyapısı** sağlanmıştır. Kafka Producer ve Consumer yapılarının koordineli çalışması, hastane otomasyon sisteminin veriye anlık erişimi ve Java ile geliştirilmiş makine öğrenmesi algoritmaları bu yapının temel bileşenlerini oluşturmuştur.

5.3. Makine Öğrenmesi Algoritmalarının Java ile Uygulanması

Bu bölümde, diyabet hastalığını tahmin etmek amacıyla geliştirilen makine öğrenmesi modelleri ve bu modellerin Java programlama dili kullanılarak nasıl uygulandığı detaylı bir şekilde açıklanmıştır. Çalışmada Naive Bayes, K-En Yakın Komşu (KNN), K-En Yakın Merkez (KNC), Lojistik Regresyon ve AdaBoost algoritmaları bireysel olarak değerlendirilmiş; ardından bu algoritmaların çıktıları kullanılarak Ensemble (Topluluk) yöntemleriyle daha sağlam tahmin sistemleri oluşturulmuştur.

Bu algoritmalarından bazıları, algoritmanın test aşamasındaki tahmin sürecini hızlandırmak ve performansının kontrol edilebilirliğini ölçmek amacıyla çoklu iş parçacığı (multi-threading) mimarisiyle kodlanmıştır. Bu yapı sayesinde, test verisi üzerindeki tahmin işlemleri, Java'nın *ExecutorService*, *Callable*, *Future* gibi sınıfları aracılığıyla paralel biçimde gerçekleştirilmiş, böylece test sürecindeki işlem süresi önemli ölçüde azaltılmıştır. Özellikle Bayes ve ADA algoritmalarında bu yapı aktif olarak kullanılmıştır. Diğer yandan, Ensemble model gibi bazı algoritmalarda ise işlem serisel olarak gerçekleştirilmiştir.

Öte yandan, bu tez çalışmasında geliştirilen gerçek zamanlı karar destek sistemi, hastane otomasyon sisteminden gelen her hasta verisini tekil olarak işleyecek şekilde yapılandırılmıştır. Bu nedenle, gerçek zamanlı tahmin sürecinde çok iş parçacıklı yapı

kullanılmamıştır çünkü sistem gerçek zamanlı veri akışı sağladığından her bir verinin gerçek zamanlı olarak analiz edilmesi gerekmektedir. Bu durumda her bir hasta için anlık olarak kayıtlar analiz edildiğinde çoklu iş parçacıklı bir yapıya başvurulduğunda sistem gecikmelere yol açmakta ve real-time dışına çıkılmasına neden olmaktadır. Bu doğrultuda her bir hasta verisi sisteme ulaştığında, Kafka Consumer bileşeni bu veriyi tekil bir iş parçası ile analiz ederek tahmin sonucunu üretmiştir. Bu tercih, canlı veri akışının daha güvenli, izlenebilir ve düşük gecikmeli şekilde yönetilmesini sağlamıştır. Dolayısıyla çok iş parçacıklı yapı, yalnızca performans testleri ve doğruluk karşılaştırmaları sırasında kullanılmış; gerçek zamanlı ortamda ise analiz işlemleri tekil iş parçası ile gerçekleştirilmiştir.

Bu bağlamda, geliştirilen sistemde tahmin sonuçları doğrudan MSSQL veritabanına kaydedilmekte ve bu sayede tüm bileşenlerin gerçek zamanlı sonuçlara anlık erişimi mümkün hale gelmektedir. Bu yaklaşım, **dosya tabanlı tampon yapılarının neden olabileceği ek gecikme, senkronizasyon karmaşası ve bakım maliyetlerini ortadan kaldırmakta**; aynı zamanda sistemin **tam anlamıyla otomatikleşmiş ve tetiklemeli (event-driven)** bir yapıda çalışmasını sağlamaktadır. Böylece, analiz tamamlandığında herhangi bir ek işlem gerekmeksizin sonuçlar doğrudan erişilebilir ve kalıcı hale gelmektedir.

Uygulanan algoritmaların Java dili ile manuel şekilde yazılması, kullanıcıya hem algoritmaların iç işleyişine tam hâkimiyet sağlamış hem de kodun esnekliğini artırmıştır. Hazır kütüphaneler (örneğin Scikit-learn gibi) yerine bu yapının tercih edilmesi, sistemi bağımsız ve özelleştirilebilir hâle getirmiştir (örneğin ensemble modellerinin elde edilmesi). Bu yaklaşım, literatürde önerilen makine öğrenmesi modellerinin gerçek zamanlı ortamlarda kendi yazılım altyapımızla nasıl entegre edilebileceğini göstermesi bakımından önemlidir [37].

Her algoritma, eğitim-veri (train) ve test-veri (test) ayrımı yapılarak test edilmiş, elde edilen doğruluk oranları Ensemble modellerin ağırlıklı hesaplamasında kullanılmıştır. Ağırlıklı Ensemble modeli, her algoritmanın başarı oranını göz önünde bulundurarak daha güvenilir bir tahmin üretmeyi amaçlamıştır. Örneğin, KNN ve KNC algoritmalarının başarı oranı daha yüksek olduğundan bu algoritmaların tahminleri ağırlıklı modelde daha fazla katkı sağlamıştır.

Yapılan bu uygulamalar, gerçek zamanlı veri akışını yakalayabilen, CPU dostu ve özelleştirilebilir makine öğrenmesi çözümleri üretmek için önemli bir örnek sunmaktadır.

5.3.1. Naive bayes

Naive Bayes algoritması, olasılık teorisine dayalı basit ancak etkili bir sınıflandırma yöntemidir. Özellikle yüksek boyutlu veri kümelerinde ve sınıf sayısı az olan problemler için hızlı ve güvenilir tahminler sunar. Diyabet teşhisi gibi ikili sınıflandırma problemlerinde sıklıkla tercih edilmektedir. Bu algoritma, her bir öznitelik ile sınıf etiketi arasındaki koşullu olasılıkları hesaplayarak tahmin işlemini gerçekleştirir. Naive varsayımı ise tüm özniteliklerin birbirinden bağımsız olduğunu kabul etmesidir [38].

Bu çalışmada, Naive Bayes algoritması Java programlama dili ile kodlanmıştır. Algoritmanın eğitim aşamasında, her bir sınıf için özniteliklerin ortalama ve varyans değerleri hesaplanmış; test aşamasında ise Bayes teoremi kullanılarak her örnek için sınıf olasılıkları belirlenmiş ve en yüksek olasılığa sahip sınıf etiketi tahmin edilmiştir. Kodlama sürecinde herhangi bir hazır kütüphane kullanılmamış, tüm hesaplamalar manuel olarak gerçekleştirilmiştir.

Ayrıca, algoritmanın tahmin süreci daha verimli hâle getirilmek amacıyla çoklu iş parçacığı (multi-threading) yaklaşımı uygulanmıştır. Test_Bayes_Forking.java dosyasında görüldüğü üzere, test verileri belirli parçalara bölünmüş ve bu parçalar eşzamanlı olarak işlenmiştir. Java'nın ExecutorService, Callable, Future gibi sınıfları kullanılarak işlem süresi kayda değer biçimde azaltılmıştır. (Şekil 5.22, 5.23, 5.24 ve 5.25)

Naive Bayes algoritmasının özellikle düşük bellek gereksinimi ve yüksek işlem hızı gibi avantajları, onu gerçek zamanlı tahminleme sistemlerinde tercih edilebilir hâle getirmiştir. Bu çalışmada da gerçek zamanlı veri akışından gelen hasta bilgileri için hızlı ve etkin sınıflandırma sonuçları üretmiştir.

```

1 import java.io.BufferedReader; // Dosya okumak için
2 import java.io.FileReader; // Dosya okumak için
3 import java.util.ArrayList; // Dinamik dizi kullanmak için
4 import java.util.List; // Liste kullanmak için
5 import java.util.concurrent.Callable; // Çoklu iş parçacığı için
6 import java.util.concurrent.ExecutorService; // İş parçacığı havuzunu yönetmek için
7 import java.util.concurrent.Executors; // ExecutorService nesnelerini oluşturmak için
8 import java.util.concurrent.Future; // Gelecekteki sonuçları temsil etmek için
9
10 public class Test_Bayes_Forking
11 {
12     //*****Ensemble kısmında bazı verilere erişmemiz gerekiyor bunun için tanımlamalar Başlangıç
13     // dogrulukOraniAda'yi static olarak tanımla---Ensemble kısmında bu veriyi kullanıyoruz---
14     public static double dogrulukOraniBayes = 0.0; // 3 usages
15     // Statik liste - tahmin verileri
16     public static List<Integer> tahminBayes = new ArrayList<>(); // 4 usages
17
18     // Getter metodu ekle
19     public static double getDogrulukOraniBayes() { // 1 usage
20         return dogrulukOraniBayes;
21     }
22     public static List<Integer> getTahminBayes() { // 1 usage
23         return tahminBayes; // Getter metodu
24     }
25     //*****Ensemble kısmında bazı verilere erişmemiz gerekiyor bunun için tanımlamalar Bitti
26
27     // Main Class
28     public static void main(String[] args)
29     {
30         int satirSay = 0; // İşlem gören satır sayısını tutan değişken
31
32         String red = "\u001B[31m"; // Kırmızı renk için ANSI escape kodu
33         String blue = "\u001B[34m"; // Mavi renk için ANSI escape kodu
34         String reset = "\u001B[0m"; // Renk sıfırlamak için
35
36         // Geçen süreyi hesaplamak için başlangıç zamanı kaydedilir
37         long startTime = System.currentTimeMillis();
38
39         String csvFile = "test.csv"; // Okunacak CSV dosyasının adı
40         String line; // Dosyadan okunan her bir satırı tutmak için kullanılacak değişken
41         String delimiter = ","; // CSV dosyası için ayırıcı karakter
42

```

Şekil 5.22. Naive Bayes Algoritması uygulama kodu.

```

// Gerçek değerler ve tahminler için listeler tanımlanır
List<Integer> gercekDeğerler = new ArrayList<>(); // Gerçek etiket değerlerini saklamak için liste
//List<Integer> tahminBayes = new ArrayList<>(); // Naive Bayes algoritması ile yapılan tahminleri saklamak için liste

System.out.println(red + "Bayes Algoritması Başlatıldı!" + reset); // Programın başladığını konsola yazdırır

try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) { // CSV dosyası okunur
    // Dosyanın ilk satırını okuyarak başlıkları atla
    if ((line = br.readLine()) != null) {
    }

    // İş parçacığı havuzunu oluştur
    ExecutorService executorService = Executors.newFixedThreadPool(4); // 4 iş parçacığı kullanarak bir havuz oluştur
    List<Future<Integer>> futures = new ArrayList<>(); // Gelecekteki sonuçları tutacak liste

    // Dosyanın geri kalanını oku ve tahminleri yap
    while ((line = br.readLine()) != null) { // Dosya sonuna kadar satırları oku
        String[] values = line.split(delimiter); // Satırı ayırıcı karakterle böl
        int outcome = Integer.parseInt(values[8]); // Sonuç değerini al
        gercekDeğerler.add(outcome); // Gerçek değerleri listeye ekle

        // Özellikleri saklayacak bir dizi oluştur
        double[] features = new double[8]; // 8 özellik için dizi
        for (int i = 0; i < 8; i++) { // İlk 8 sütun değerlerini diziye aktar
            features[i] = Double.parseDouble(values[i]);
        }

        // Her satır için bir Callable (iş) oluştur ve iş parçacığı havuzuna ekle
        futures.add(executorService.submit(new Test_Bayes_Forking.NaiveBayesTask(features)));
    }

    // Tahminleri al
    for (Future<Integer> future : futures) { // Her işin sonucunu al
        tahminBayes.add(future.get()); // Tahmin sonuçlarını listeye ekle
        satirSay++; // İşlem gören satır sayısını artır
        System.out.println(satirSay + ".satır bitti"); // İşlemin tamamlandığını yazdır
    }

    executorService.shutdown(); // İş parçacığı havuzunu kapat
}

```

Şekil 5.23. Naive Bayes Algoritması uygulama kodu (Devamı-1).

```

        executorService.shutdown(); // İş parçacığı havuzunu kapat

        //*****Doğru ve yanlış tahmin sayısı yazdırılıyor
        // Doğru ve yanlış tahminleri saymak için sayacı tanımla
        int dogruTahmin = 0;
        int yanlışTahmin = 0;

        // Tahminleri ve gerçek değerleri karşılaştır
        for (int i = 0; i < gercekDegerler.size(); i++) {
            if (gercekDegerler.get(i).equals(tahminBayes.get(i))) {
                dogruTahmin++; // Doğru tahmin sayısını artır
            } else {
                yanlışTahmin++; // Yanlış tahmin sayısını artır
            }
        }

        // Sonuçları ekrana yazdır
        System.out.println("Doğru tahmin sayısı: " + dogruTahmin);
        System.out.println("Yanlış tahmin sayısı: " + yanlışTahmin);
        //*****Bitis

        // Doğruluk oranını hesapla
        dogrulukOraniBayes = calculateAccuracy(gercekDegerler, tahminBayes); // Doğruluk oranını hesapla
        System.out.print(red + "Doğruluk Oranı: " + reset); // Doğruluk oranlarını yazdır
        System.out.println(red + " " + reset + dogrulukOraniBayes); // Bayes algoritmasının doğruluk oranını yazdır

        long endTime = System.currentTimeMillis(); // İşlem bitiş zamanını al
        long elapsedTime = endTime - startTime; // Geçen süreyi hesapla
        System.out.println(blue + "Toplam süre: " + (elapsedTime / 1000.0) + " saniye" + reset); // Toplam süreyi yazdır
    } catch (Exception e) { // Herhangi bir hata durumunda
        e.printStackTrace(); // Hata izini yazdır
    }
} // main

// Tahmin için kullanılan iş sınıfı
static class NaiveBayesTask implements Callable<Integer> { // Callable arayüzünü implement eden NaiveBayesTask sınıfı 1 usage
    private double[] features; // Özellikleri tutacak dizi 9 usages

    public NaiveBayesTask(double[] features) { this.features = features; // Özellikleri diziye aktar }

}

```

Şekil 5.24. Naive Bayes Algoritması uygulama kodu (Devamı-2).

```

        System.out.println(blue + "Toplam süre: " + (elapsedTime / 1000.0) + " saniye" + reset); // Toplam süreyi yazdır
    } catch (Exception e) { // Herhangi bir hata durumunda
        e.printStackTrace(); // Hata izini yazdır
    }
} // main

// Tahmin için kullanılan iş sınıfı
static class NaiveBayesTask implements Callable<Integer> { // Callable arayüzünü implement eden NaiveBayesTask sınıfı 1 usage
    private double[] features; // Özellikleri tutacak dizi 9 usages

    public NaiveBayesTask(double[] features) { this.features = features; // Özellikleri diziye aktar }

    @Override
    public Integer call() throws Exception { // İş parçasının gerçekleştireceği işlem
        // NaiveBayesAlgorithm sınıfından bir nesne oluşturun ve tahmin yap
        NaiveBayesAlgorithm naiveBayesAlgorithm = new NaiveBayesAlgorithm(features[0], features[1], features[2],
            features[3], features[4], features[5],
            features[6], features[7]);
        return naiveBayesAlgorithm.naiveBayesTahmin(); // Tahmin sonucunu döndür
    }
}

// Doğruluk oranını hesaplama
private static double calculateAccuracy(List<Integer> gercekDegerler, List<Integer> tahminBayes) { // Gerçek ve tahmin değerleriyle doğruluğu hesapla 1 usage
    int correctPredictions = 0; // Doğru tahmin sayısını tutan değişken
    for (int i = 0; i < gercekDegerler.size(); i++) { // Tüm gerçek değerleri kontrol et
        if (gercekDegerler.get(i).equals(tahminBayes.get(i))) { // Eğer tahmin doğruysa
            correctPredictions++; // Doğru tahmin sayısını artır
        }
    }
    return (double) correctPredictions / gercekDegerler.size() * 100; // Yüzde cinsinden doğruluk oranını döndür
}
}

```

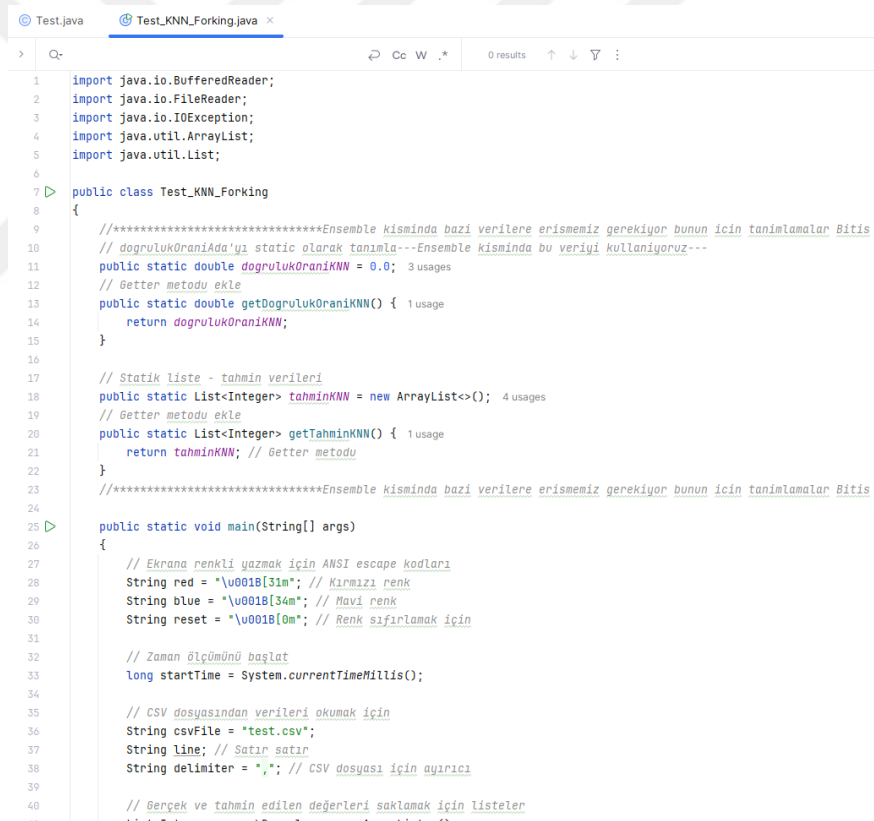
Şekil 5.25. Naive Bayes Algoritması uygulama kodu (Devamı-3).

5.3.2. KNN ve KNC

Bu çalışmada, diyabet hastalığının sınıflandırılmasına yönelik olarak kullanılan algoritmalarından biri **En Yakın Komşu (K-Nearest Neighbors – KNN)** algoritmasıdır. KNN, örüntü tanıma ve sınıflandırma problemlerinde sıkça tercih

edilen, gözetimli bir makine öğrenmesi yöntemidir. Algoritma, sınıflandırılması gereken bir örneği, eğitim verisindeki en yakın k komşusuna bakarak etiketler. Bu komşular genellikle öklidyen mesafeye göre belirlenir. Çoğunluk sınıfına ait etiket, tahmin sonucu olarak atanır [39].

Gerçekleştirilen uygulamada, her satır hasta verisini temsil eden bir vektör olarak değerlendirilmiştir. Her vektör için ayrı ayrı, eğitim veri seti üzerindeki örneklerle mesafesi hesaplanarak en yakın k komşu belirlenmiş ve bu komşuların sınıflarına göre sınıflandırma işlemi yapılmıştır. Geliştirilen Java uygulamasında, test verileri CSV formatında dışarıdan okunmakta ve her bir satır için k=3 komşuya göre sınıflandırma yapılmaktadır. Tahmin edilen etiketler bir listeye kaydedilerek doğruluk oranı hesaplanmaktadır. KNN algoritması, sınıflandırma sürecinde parametre olarak sadece k değeri alacak şekilde tasarlanmıştır.



```
1 import java.io.BufferedReader;
2 import java.io.FileReader;
3 import java.io.IOException;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 public class Test_KNN_Forking
8 {
9     //*****Ensemble kisminda bazi verilere erismemiz gerekiyor bunun için tanımlamalar Bitis
10    // dogrulukOraniAda'yi static olarak tanımla--Ensemble kisminda bu veriyi kullanıyoruz---
11    public static double dogrulukOraniKNN = 0.0; // 3 usages
12    // Getter metodu ekle
13    public static double getDogrulukOraniKNN() { 1 usage
14        return dogrulukOraniKNN;
15    }
16
17    // Statik liste - tahmin verileri
18    public static List<Integer> tahminKNN = new ArrayList<>(); // 4 usages
19    // Getter metodu ekle
20    public static List<Integer> getTahminKNN() { 1 usage
21        return tahminKNN; // Getter metodu
22    }
23    //*****Ensemble kisminda bazi verilere erismemiz gerekiyor bunun için tanımlamalar Bitis
24
25    public static void main(String[] args)
26    {
27        // Ekranı renkli yazmak için ANSI escape kodları
28        String red = "\u001B[31m"; // Kırmızı renk
29        String blue = "\u001B[34m"; // Mavi renk
30        String reset = "\u001B[0m"; // Renk sıfırlamak için
31
32        // Zaman ölçümünü başlat
33        long startTime = System.currentTimeMillis();
34
35        // CSV dosyasından verileri okumak için
36        String csvFile = "test.csv";
37        String line; // Satır satır
38        String delimiter = ","; // CSV dosyası için ayırıcı
39
40        // Gerçek ve tahmin edilen değerleri saklamak için listeler
```

Şekil 5.26. KNN Algoritması uygulama kodu.

```

// Gerçek ve tahmin edilen değerleri saklamak için listeler
List<Integer> gercekDegerler = new ArrayList<>();
//List<Integer> tahminKNN = new ArrayList<>();

System.out.println(red + "KNN Algoritması Başlatıldı!" + reset);

try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
    // Dosyanın ilk satırını okuyarak başlıkları atla
    if ((line = br.readLine()) != null) {}

    // Dosyanın geri kalanını oku
    while ((line = br.readLine()) != null) {
        String[] values = line.split(delimiter); // Satırı ayır

        // Değerleri parse et
        int pregnancies = Integer.parseInt(values[0]);
        double glucose = Double.parseDouble(values[1]);
        double bloodPressure = Double.parseDouble(values[2]);
        double skinThickness = Double.parseDouble(values[3]);
        double insulin = Double.parseDouble(values[4]);
        double bmi = Double.parseDouble(values[5]);
        double diabetesPedigreeFunction = Double.parseDouble(values[6]);
        int age = Integer.parseInt(values[7]);
        int outcome = Integer.parseInt(values[8]);

        // Gerçek değerleri kaydet
        gercekDegerler.add(outcome);

        // KNN algoritmasını başlat ve tahmin yap
        KNNAlgorithm knnAlgorithm = new KNNAlgorithm(
            (double) pregnancies, (double) glucose, (double) bloodPressure,
            (double) skinThickness, (double) insulin, bmi, diabetesPedigreeFunction, (double) age
        );

        int predictionKNN = knnAlgorithm.knnUygula(k: 3); // k=3
        tahminKNN.add(predictionKNN);
    }
}

```

Şekil 5.27. KNN Algoritması uygulama kodu (Devamı-1).

```

// Tahminlerin doğruluğunu kontrol et ve doğru tahmin sayısını hesapla
int correctPredictions = 0;
for (int i = 0; i < gercekDegerler.size(); i++) {
    if (gercekDegerler.get(i).equals(tahminKNN.get(i))) {
        correctPredictions++;
    }
}

//*****Dogru ve yanlis tahmin sayisi yazdiriliyor
// Dogru ve yanlis tahminleri saymak için sayacı tanımla
int dogruTahmin = 0;
int yanlisTahmin = 0;

// Tahminleri ve gerçek değerleri karşılaştır
for (int i = 0; i < gercekDegerler.size(); i++) {
    if (gercekDegerler.get(i).equals(tahminKNN.get(i))) {
        dogruTahmin++; // Dogru tahmin sayısını artır
    } else {
        yanlisTahmin++; // Yanlis tahmin sayısını artır
    }
}

// Sonuçları ekrana yazdır
System.out.println("Dogru tahmin sayısı: " + dogruTahmin);
System.out.println("Yanlis tahmin sayısı: " + yanlisTahmin);
//*****Bitis

// Doğruluk oranını hesapla
dogrulukOraniKNN = (double) correctPredictions / gercekDegerler.size() * 100;

System.out.println(red + "Dogruluk Oranı: " + reset + dogrulukOraniKNN + "%");

// Zaman ölçümünü bitir ve ekrana saniye cinsinden yazdır
long endTime = System.currentTimeMillis();
double elapsedTimeInSeconds = (endTime - startTime) / 1000.0; // Saniye cinsine çevir
System.out.println(blue + "Toplam süre: " + elapsedTimeInSeconds + " saniye" + reset);
} catch (IOException e) {}

```

Şekil 5.28. KNN Algoritması uygulama kodu (Devamı-2).

Bununla birlikte, benzer şekilde geliştirilen **KNC (K-Nearest Centroid)** algoritması da çalışmada kullanılmıştır. KNC, her sınıfa ait örneklerin ortalamasını (merkezini) belirleyerek, sınıflandırılması gereken örneğin bu merkezlere olan uzaklıklarını hesaplamak suretiyle sınıflandırma yapmaktadır [40, 41]. Geliştirilen uygulamada her sınıfa ait vektörlerin ortalaması alınmış, ardından sınıflandırılacak örnek ile bu ortalamalar arasındaki mesafe ölçülerek örnek en yakın merkeze ait sınıfa atanmıştır.

KNC algoritması, özellikle veri setinin büyük olduğu durumlarda işlem süresini önemli ölçüde azaltmaktadır çünkü her bir örnek yerine yalnızca sınıf merkezleri üzerinden hesaplama yapılmaktadır. Çalışmada her iki algoritmanın performansı karşılaştırılmış ve tahmin sonuçları sistemde daha sonra ensemble algoritmaları için kullanılmıştır.

```
Test.java x Test_KNC_Forking.java x
1 import java.io.BufferedReader;
2 import java.io.FileReader;
3 import java.io.IOException;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 public class Test_KNC_Forking
8 {
9     //*****Ensemble kısmında bazı verilere erişmemiz gerekiyor bunun için tanımlamalar Başlangıç
10    // dogrulukOraniAda'yi static olarak tanımla--Ensemble kısmında bu veriyi kullanıyoruz---
11    public static double dogrulukOraniKNC = 0.0; 3 usages
12    // Getter metodu ekle
13    public static double getDogrulukOraniKNC() { 1 usage
14        return dogrulukOraniKNC;
15    }
16
17    // Statik liste - tahmin verileri
18    public static List<Integer> tahminKNC = new ArrayList<>(); 4 usages
19    // Getter metodu ekle
20    public static List<Integer> getTahminKNC() { 1 usage
21        return tahminKNC; // Getter metodu
22    }
23    //*****Ensemble kısmında bazı verilere erişmemiz gerekiyor bunun için tanımlamalar Bitis
24
25    // Main class
26    public static void main(String[] args)
27    {
28        // Ekranı renkli yazmak için ANSI escape kodları
29        String red = "\u001B[31m"; // Kırmızı renk
30        String blue = "\u001B[34m"; // Mavi renk
31        String reset = "\u001B[0m"; // Renk sıfırlamak için
32
33        // Zaman ölçümünü başlat
34        long startTime = System.currentTimeMillis();
35
36        // CSV dosyasından verileri okumak için
37        String csvFile = "test.csv";
38        String line; // Satır satır
39        String delimiter = ","; // CSV dosyası için ayırıcı
40
41        // Gerçek ve tahmin edilen değerleri saklamak için listeler
42        List<Integer> gercekDeğerler = new ArrayList<>();
```

Şekil 5.29. KNC Algoritması uygulama kodu.

```

System.out.println(red + "KNC Algoritması Başlatıldı!" + reset);

try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
    // Dosyanın ilk satırını okuyarak başlıkları atla
    if ((line = br.readLine()) != null) {}

    // Dosyanın geri kalanını oku
    while ((line = br.readLine()) != null) {
        String[] values = line.split(delimiter); // Satırı ayır

        // Değerleri parse et
        int pregnancies = Integer.parseInt(values[0]);
        double glucose = Double.parseDouble(values[1]);
        double bloodPressure = Double.parseDouble(values[2]);
        double skinThickness = Double.parseDouble(values[3]);
        double insulin = Double.parseDouble(values[4]);
        double bmi = Double.parseDouble(values[5]);
        double diabetesPedigreeFunction = Double.parseDouble(values[6]);
        int age = Integer.parseInt(values[7]);
        int outcome = Integer.parseInt(values[8]);

        // Gerçek değerleri kaydet
        gercekDegerler.add(outcome);

        // KNC algoritmasını başlat ve tahmin yap
        KNCAlgorithm kncAlgorithm = new KNCAlgorithm(
            (double) pregnancies, (double) glucose, (double) bloodPressure,
            (double) skinThickness, (double) insulin, bmi, diabetesPedigreeFunction, (double) age
        );

        int predictionKNC = kncAlgorithm.kncUygula(); // k=3
        tahminKNC.add(predictionKNC);
    }

    // Tahminlerin doğruluğunu kontrol et ve doğru tahmin sayısını hesapla
    int correctPredictions = 0;
    for (int i = 0; i < gercekDegerler.size(); i++) {
        if (gercekDegerler.get(i).equals(tahminKNC.get(i))) {
            correctPredictions++;
        }
    }
}

```

Şekil 5.30. KNC Algoritması uygulama kodu (Devamı-1).

```

    }
}

//*****Doğru ve yanlış tahmin sayısı yazdırılıyor
// Doğru ve yanlış tahminleri saymak için sayaçlar tanımla
int dogruTahmin = 0;
int yanlışTahmin = 0;

// Tahminleri ve gerçek değerleri karşılaştır
for (int i = 0; i < gercekDegerler.size(); i++) {
    if (gercekDegerler.get(i).equals(tahminKNC.get(i))) {
        dogruTahmin++; // Doğru tahmin sayısını artır
    } else {
        yanlışTahmin++; // Yanlış tahmin sayısını artır
    }
}

// Sonuçları ekrana yazdır
System.out.println("Doğru tahmin sayısı: " + dogruTahmin);
System.out.println("Yanlış tahmin sayısı: " + yanlışTahmin);
//*****Bitiş

// Doğruluk oranını hesapla
dogrulukOraniKNC = (double) correctPredictions / gercekDegerler.size() * 100;

System.out.println(red + "Doğruluk Oranı: " + reset + dogrulukOraniKNC + "%");

// Zaman ölçümünü bitir ve ekrana saniye cinsinden yazdır
long endTime = System.currentTimeMillis();
double elapsedTimeInSeconds = (endTime - startTime) / 1000.0; // Saniye cinsine çevir
System.out.println(blue + "Toplam süre: " + elapsedTimeInSeconds + " saniye" + reset);

} catch (IOException e) {
    e.printStackTrace();
} catch (NumberFormatException e) {
    System.out.println("Bir sayı format hatası oluştu: " + e.getMessage());
}

} // main end!

```

Şekil 5.31. KNC Algoritması uygulama kodu (Devamı-2).

Bu algoritmaların çıktıları, daha sonra çoğunluk oylaması ve ağırlıklı oylama yöntemleri ile birleştirilmiş, karar destek sisteminde daha sağlam tahminler yapılması

sağlanmıştır. Doğruluk oranları da bu süreçte hesaplanarak diğer algoritmalarla karşılaştırılmıştır.

5.3.3. Logistic regression

Bu çalışmada lojistik regresyon algoritması, diyabet tanısı için sınıflandırma amacıyla Java programlama dili kullanılarak uygulanmıştır. Lojistik regresyon, özellikle ikili sınıflandırma problemlerinde sıklıkla tercih edilen, doğrusal olmayan bir modelleme yöntemidir. Modelin çıktısı, verilen özelliklerin hastalığa etkisini hesaplayan bir olasılık değeri olarak yorumlanır [42].

Uygulanan sistemde, test.csv adlı veri seti işlenerek her bir hasta örneği için tahmin işlemi yapılmıştır. Lojistik regresyon modeli, çoklu iş parçacığı (multi-threading) yapısıyla geliştirilmiştir. Bu yapı sayesinde, veri setindeki satırlar dört farklı iş parçacığına dağıtılarak eş zamanlı olarak işlenmiş ve böylece işlem süresi önemli ölçüde azaltılmıştır. Kodda bu yapı ExecutorService sınıfı ile gerçekleştirilmiş ve her iş parçacığı kendi sorumlu olduğu veri aralığını işleyerek tahmin işlemini gerçekleştirmiştir.

Tahmin sürecinde her satırdaki veriler parse edilerek LogisticRegressionAlgorithm sınıfına gönderilmiş ve bu sınıfın eğitimModeliniKur() metodu yardımıyla model eğitilmiş, ardından tahminYap() metodu ile hasta bireylerin diyabet olup olmadığı belirlenmiştir. Tüm tahminler eş zamanlı olarak tahminLogistic adlı listeye kaydedilmiştir. Ayrıca tahmin sonuçları gerçek değerlerle karşılaştırılarak modelin doğruluk oranı hesaplanmıştır. Kodda doğru ve yanlış tahmin sayıları ayrı ayrı raporlanmakta, ardından genel doğruluk yüzdesi dogrulukOraniLogistic değişkenine atanmaktadır.

Bu mimariyle çalışan lojistik regresyon modeli, tek iş parçacıklı modellere göre belirgin ölçüde daha hızlı çalışmakta ve büyük veri kümeleriyle başa çıkmada etkinlik sağlamaktadır. Ayrıca, çoklu iş parçacıklı yapı sayesinde CPU kaynaklarının daha verimli kullanılması mümkün hale gelmektedir.

Literatürde lojistik regresyon, sağlık verisi sınıflandırmasında sıklıkla tercih edilmiştir. Özellikle diyabet, kalp hastalıkları ve hipertansiyon gibi rahatsızlıklarda hastalık riskini öngörmek amacıyla kullanılmaktadır [43]. Bu çalışmada da benzer şekilde, eğitilmiş model hastane otomasyon sistemiyle entegre edilerek gerçek zamanlı hasta verileri üzerinde tahminleme gerçekleştirmiştir. Böylece, her yeni veri girişinden

sonra model tarafından üretilen tahminler sistem tarafından değerlendirilmiş, sonuçlar hem kullanıcı arayüzünde görselleştirilmiş hem de MSSQL veritabanına kaydedilmiştir.

Sonuç olarak, lojistik regresyon algoritması bu çalışmada çoklu iş parçacıklı yapıyla etkili şekilde uygulanmış, yüksek veri işlem kapasitesi ve klinik karar destek katkısı bakımından başarılı sonuçlar vermiştir.

```
Test.java Test_Logistic_Forking.java x
1 import java.io.BufferedReader;
2 import java.io.FileReader;
3 import java.io.IOException;
4 import java.util.ArrayList;
5 import java.util.List;
6 import java.util.concurrent.ExecutorService;
7 import java.util.concurrent.Executors;
8 import java.util.concurrent.TimeUnit;
9
10 public class Test_Logistic_Forking
11 {
12     //*****Ensemble kısmında bazı verilere erişmemiz gerekiyor bunun için tanımlamalar Bitis
13     // dogrulukOraniLogistic'i static olarak tanımla---Ensemble kısmında bu veriyi kullanıyoruz---
14     public static double dogrulukOraniLogistic = 0.0; 5 usages
15     // Getter metodu ekle
16     public static double getDogrulukOraniLogistic() { 1 usage
17         return dogrulukOraniLogistic;
18     }
19
20     // Statik liste - tahmin verileri
21     public static List<Integer> tahminLogistic = new ArrayList<>(); 5 usages
22     // Getter metodu ekle
23     public static List<Integer> getTahminLogistic() { 1 usage
24         return tahminLogistic; // Getter metodu
25     }
26     //*****Ensemble kısmında bazı verilere erişmemiz gerekiyor bunun için tanımlamalar Bitis
27
28 public static void main(String[] args) throws IOException
29 {
30     // Ekranı renkli yazmak için ANSI escape kodları tanımlanıyor
31     String red = "\u001B[31m"; // Kırmızı renk için
32     String blue = "\u001B[34m"; // Mavi renk için
33     String reset = "\u001B[0m"; // Renk sıfırlamak için
34
35     // Zaman ölçümünü başlat
36     long startTime = System.currentTimeMillis();
37
38     // CSV dosyasından veri okumak için dosya yolu ve ayırıcı tanımlanıyor
39     String csvFile = "test.csv";
40     String delimiter = ",";
41
42     // Gerçek değerleri ve tahmin sonuçlarını tutmak için listeler
```

Şekil 5.32. Logistic Regression Algoritması uygulama kodu.

```

// Gerçek değerleri ve tahmin sonuçlarını tutmak için listeler
List<Integer> gercekDegerler = new ArrayList<>();
List<Integer> tahminLogistic = new ArrayList<>();

// İş parçacıkları oluşturmak için ExecutorService kullanılıyor
int numberOfThreads = 4; // Kullanılacak iş parçacığı sayısı
ExecutorService executorService = Executors.newFixedThreadPool(numberOfThreads); // İş parçacığı havuzu

// Algoritmanın başladığını bildiren mesaj
System.out.println(red + "Logistic Regression Algoritması Başlatıldı!" + reset);

// CSV dosyasını okuma ve veriyi iş parçacıklarıyla paralel işleme
try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
    // Dosyanın ilk satırını okuyarak başlıkları atla
    br.readLine();

    // Satırları tutmak için bir liste oluşturuluyor
    List<String> lines = new ArrayList<>();
    String line;

    // Dosyadaki tüm satırları bir listeye aktar
    while ((line = br.readLine()) != null) {
        lines.add(line);
    }

    // Her iş parçacığına verilecek satır sayısını hesapla
    int chunkSize = lines.size() / numberOfThreads; // Örnek: 750/4=187

    // Her iş parçacığına belirli bir veri aralığı tahsis ediliyor
    for (int i = 0; i < numberOfThreads; i++) {
        final int start = i * chunkSize; // İş parçacığının başlangıç indeksi
        final int end = (i == numberOfThreads - 1) ? lines.size() : (i + 1) * chunkSize; // Son iş parçacığı için kalan satırlar

        // İş parçacığını başlat
        executorService.submit(() -> {
            // Belirli aralıktaki satırları işle
            for (int j = start; j < end; j++) {
                // Satırı ayır ve değerlere dönüştür
                String[] values = lines.get(j).split(delimiter);
                int pregnancies = Integer.parseInt(values[0]);
            }
        });
    }
}

```

Şekil 5.33. Logistic Regression Algoritması uygulama kodu (Devamı-1).

```

String[] values = lines.get(j).split(delimiter);
int pregnancies = Integer.parseInt(values[0]);
double glucose = Double.parseDouble(values[1]);
double bloodPressure = Double.parseDouble(values[2]);
double skinThickness = Double.parseDouble(values[3]);
double insulin = Double.parseDouble(values[4]);
double bmi = Double.parseDouble(values[5]);
double diabetesPedigreeFunction = Double.parseDouble(values[6]);
int age = Integer.parseInt(values[7]);
int outcome = Integer.parseInt(values[8]);

// Gerçek değeri ekleyin
synchronized (gercekDegerler) {
    gercekDegerler.add(outcome);
}

// Logistic regresyon algoritmasını çalıştır
LogisticRegressionAlgorithm lojistikRegresyonThread = new LogisticRegressionAlgorithm(
    (double) pregnancies,
    (double) glucose,
    (double) bloodPressure,
    (double) skinThickness,
    (double) insulin,
    bmi,
    diabetesPedigreeFunction,
    (double) age
);

try {
    lojistikRegresyonThread.egitimModeliniKur(); // Modeli kur
} catch (IOException e) {
    throw new RuntimeException(e); // Hata durumunda istisna fırlat
}

// Tahmin yap ve sonucu ekle
int predictionLogistic = lojistikRegresyonThread.tahminYap();
synchronized (tahminLogistic) {
    tahminLogistic.add(predictionLogistic);
}

```

Şekil 5.34. Logistic Regression Algoritması uygulama kodu (Devamı-2).

```

        executorService.submit() -> {

            // Konsola bilgi ver
            System.out.println("Satır " + (j + 1) + " bitti."); // Hangi satırın işleme alındığını belirt
        }
    });

    // Executor'ün iş parçacıklarının tamamlanmasını bekle
    executorService.shutdown(); // İş parçacıklarını kapat
    executorService.awaitTermination(1, TimeUnit.HOURS); // 1 saat bekle
    // eğer iş parçacığı için tanımlanan bu süre-1 saat- asılırsa program sonlanır

} catch (IOException | InterruptedException e) {
    e.printStackTrace(); // Hata durumunda uyarıyı yazdır
}

// Doğru tahmin sayısını ve toplam test verisini kullanarak doğruluk oranını hesapla
int dogruTahminler = 0; // Doğru tahmin sayısını başlat
for (int i = 0; i < gercekDegerler.size(); i++) {
    if (gercekDegerler.get(i).equals(tahminLogistic.get(i))) {
        dogruTahminler++; // Doğru tahminleri say
    }
}

//*****Doğru ve yanlış tahmin sayısı yazdırılıyor
// Doğru ve yanlış tahminleri saymak için sayaçlar tanımla
int dogruTahmin = 0;
int yanlışTahmin = 0;

// Tahminleri ve gerçek değerleri karşılaştır
for (int i = 0; i < gercekDegerler.size(); i++) {
    if (gercekDegerler.get(i).equals(tahminLogistic.get(i))) {
        dogruTahmin++; // Doğru tahmin sayısını artır
    } else {
        yanlışTahmin++; // Yanlış tahmin sayısını artır
    }
}

// Sonuçları ekrana yazdır

```

Şekil 5.35. Logistic Regression Algoritması uygulama kodu (Devamı-3).

```

// Sonuçları ekrana yazdır
System.out.println("Doğru tahmin sayısı: " + dogruTahmin);
System.out.println("Yanlış tahmin sayısı: " + yanlışTahmin);
//*****Bitiş

// Doğruluk oranını hesapla ve ekrana yazdır
dogrulukOraniLogistic = (double) dogruTahminler / gercekDegerler.size(); // Doğruluk oranı (örneğin, 0.532 gibi)
dogrulukOraniLogistic = dogrulukOraniLogistic * 100; // Yüzdeye çevir
System.out.println(red + "Doğruluk Oranı:" + reset + dogrulukOraniLogistic); // Sonucu ekrana yazdır

// Zaman ölçümünü bitir ve ekrana yazdır
long endTime = System.currentTimeMillis();
double elapsedTimeInSeconds = (endTime - startTime) / 1000.0; // Geçen süreyi hesapla
System.out.println(blue + "Toplam Süre: " + elapsedTimeInSeconds + " saniye" + reset); // Toplam süreyi ekrana yazdır
}
}

```

Şekil 5.36. Logistic Regression Algoritması uygulama kodu (Devamı-4).

5.3.4. AdaBoost

AdaBoost (Adaptive Boosting), zayıf sınıflandırıcıların topluca değerlendirilmesiyle güçlü bir sınıflandırıcı elde etmeyi amaçlayan bir topluluk (ensemble) öğrenme yöntemidir. AdaBoost algoritması, her yinelemde hatalı sınıflandırılan örneklerle daha fazla ağırlık vererek yeni zayıf sınıflandırıcıların bu örnekler üzerinde daha iyi performans göstermesini sağlar. Böylece, her bir zayıf öğrenicinin çıktısı uygun ağırlıklarla birleştirilerek nihai tahmin yapılır [44].

Bu çalışmada, AdaBoost algoritması Java dili kullanılarak manuel biçimde geliştirilmiştir. Geliştirilen yapıda, temel sınıflandırıcılar olarak basit karar kurallarına dayanan bileşenler kullanılmış ve bu sınıflandırıcılar üzerinden iteratif şekilde tahmin

yapılmıştır. Her iterasyon sonucunda örnek ağırlıkları güncellenerek hatalı sınıflandırılan örnekler daha öncelikli hale getirilmiştir. Belirli sayıda iterasyon sonrasında algoritma final kararını vermektedir.

Kod yapısında BufferedReader sınıfı ile test verisi satır satır okunmakta ve her bir satır için AdaBoostAlgorithm sınıfı aracılığıyla tahminleme yapılmaktadır. Bu tahmin sonuçları tahminAdaBoost listesinde tutulmakta, aynı anda gerçek değerlerle karşılaştırılarak doğruluk oranı hesaplanmaktadır. Elde edilen doğruluk oranı doğrulukOraniAda değişkenine atanarak sistemin başarımı değerlendirilmiştir. Geliştirilen yapı tek iş parçacıklı olarak çalışmakta olup, her işlem sırayla yürütülmektedir.

Özellikle eğitim verilerindeki dengesizlik problemleri için AdaBoost güçlü bir alternatif sunmaktadır. Literatürde yapılan çalışmalarda Boosting algoritmalarının doğası gereği aşırı öğrenmeye (overfitting) karşı daha dirençli olduğu ve sınıflar arasında ayrımı güçleştiren durumlarda iyi performans gösterdiği bilinmektedir [45].

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class Test_ADA_Forking
{
    //*****Ensemble kısmında bazı verilere erişmemiz gerekiyor bunun için tanımlamalar Baslangic
    // dogrulukOraniAda'yi static olarak tanımla
    public static double dogrulukOraniAda = 0.0; 3 usages
    // Statik liste - tahmin verileri
    public static List<Integer> tahminAdaBoost = new ArrayList<>(); 3 usages
    // Statik liste -- gercek degerler [bir kerelik cekildi]
    public static List<Integer> gercekDegerler = new ArrayList<>(); 4 usages

    // Getter metodu ekle
    public static double getDogrulukOraniAda() { return dogrulukOraniAda; }
    public static List<Integer> getTahminAdaBoost() { return tahminAdaBoost; } // Getter metodu
    public static List<Integer> getGercekDegerler() { return gercekDegerler; } // Getter metodu
    //*****Ensemble kısmında bazı verilere erişmemiz gerekiyor bunun için tanımlamalar Bitti

    //Main Class
    public static void main(String[] args)
    {
        // Ekrana renkli yazmak için
        String red = "\u001B[31m"; // Kırmızı renk için ANSI escape kodu
        String blue = "\u001B[34m"; // Mavi renk için ANSI escape kodu
        String reset = "\u001B[0m"; // Renk sıfırlamak için

        // Zaman ölçümünü başlat
        long startTime = System.currentTimeMillis();

        // CSV Dosyasından data okumak için
        String csvFile = "test.csv";
        String line; // satır satır
        String delimiter = ","; // CSV dosyası için ayırıcı // Split işlemi için parser virgül
    }
}
```

Şekil 5.37. AdaBoost Algoritması uygulama kodu.

```

AdaBoostAlgorithm adaBoost= null;

System.out.println(red+"ADA Algoritması Başlatıldı!"+"reset");

try (BufferedReader br = new BufferedReader(new FileReader(csvFile)))
{
    // Dosyanın ilk satırını okuyarak başlıkları atla
    if ((line = br.readLine()) != null) {}

    // Dosyanın geri kalanını oku
    while ((line = br.readLine()) != null)
    {
        String[] values = line.split(delimiter); // Satırı ayır

        // Değerleri parse et
        int pregnancies = Integer.parseInt(values[0]);
        double glucose = Double.parseDouble(values[1]);
        double bloodPressure = Double.parseDouble(values[2]);
        double skinThickness = Double.parseDouble(values[3]);
        double insulin = Double.parseDouble(values[4]);
        double bmi = Double.parseDouble(values[5]);
        double diabetesPedigreeFunction = Double.parseDouble(values[6]);
        int age = Integer.parseInt(values[7]);
        int outcome = Integer.parseInt(values[8]);

        Double p1_ = Double.valueOf(pregnancies);
        Double p2_ = Double.valueOf(glucose);
        Double p3_ = Double.valueOf(bloodPressure);
        Double p4_ = Double.valueOf(skinThickness);
        Double p5_ = Double.valueOf(insulin);
        Double p6_ = Double.valueOf(bmi);
        Double p7_ = Double.valueOf(diabetesPedigreeFunction);
        Double p8_ = Double.valueOf(age);
        Double p9_ = Double.valueOf(outcome);

        // original degerini ekledik
        gercekDegerler.add(outcome);

        // simdi algoritmaları calistiralim

```

Şekil 5.38. AdaBoost Algoritması uygulama kodu (Devamı-1).

```

//--AdaBoost Algoritması Baslangic*****
adaBoost = new AdaBoostAlgorithm(p1_,p2_,p3_,p4_,p5_,p6_,p7_,p8_);
int predictionADA = adaBoost.adaBoostUygula( IterasyonSayisi: 10); // Iterasyon sayısı örneği
//--AdaBoost Algoritması Bitis

tahminAdaBoost.add(predictionADA);
} // while End--- CSV bitti

//*****Dogru ve yanlış tahmin sayısı yazdırılıyor
// Doğru ve yanlış tahminleri saymak için sayacı tanımla
int dogruTahmin = 0;
int yanlışTahmin = 0;

// Tahminleri ve gerçek değerleri karşılaştır
for (int i = 0; i < gercekDegerler.size(); i++) {
    if (gercekDegerler.get(i).equals(tahminAdaBoost.get(i))) {
        dogruTahmin++; // Doğru tahmin sayısını artır
    } else {
        yanlışTahmin++; // Yanlış tahmin sayısını artır
    }
}

// Sonuçları ekrana yazdır
System.out.println("Doğru tahmin sayısı: " + dogruTahmin);
System.out.println("Yanlış tahmin sayısı: " + yanlışTahmin);
//*****Bitis

// -----
dogrulukOraniAda=adaBoost.dogrulukOraniHesapla( IterasyonSayisi: 10);
System.out.print(red+"Dogruluk Orani:"+"reset");
System.out.println(red+" "+reset+dogrulukOraniAda);

// Zaman ölçümünü bitir ve ekrana yazdır
long endTime = System.currentTimeMillis();
long elapsedTimeInMilliseconds = endTime - startTime; // Geçen süreyi milisaniye cinsinden hesapla
double elapsedTimeInSeconds = (endTime - startTime) / 1000.0; // Saniye cinsine çevir
System.out.println(blue+"Toplam süre: " + elapsedTimeInSeconds + " milisaniye"+"reset");
} catch (IOException e) {
    e.printStackTrace();
}

```

Şekil 5.39. AdaBoost Algoritması uygulama kodu (Devamı-2).

5.3.5. Ensemble algoritmalar

Gerçek zamanlı tahmin mimarisinde her makine öğrenmesi algoritmasının tekil olarak performansı incelenirse de, sistem genelinde daha yüksek doğruluk elde edebilmek amacıyla birden fazla modelin çıktısının birleştirildiği ensemble yöntemlerinden faydalanılmıştır. Bu çalışmada iki farklı ensemble yaklaşımı uygulanmıştır: **Ağırlıklı Oylama Algoritması** ve **Çoğunluk Oylaması Algoritması**.

5.3.5.1. Ensemble ağırlıklı oylama algoritması

Bu yöntemde, her algoritmanın doğruluk oranı üzerinden bir ağırlık katsayısı belirlenmiş ve tahmin sonuçlarına bu oranlar yansıtılmıştır. Örneğin doğruluk oranı yüksek olan modellerin tahminlerine daha fazla önem verilmiş; düşük doğruluk oranına sahip modellerin tahminleri ise kararda daha az etkili olmuştur. Böylece genel tahmin sistemine daha güvenilir modellerin baskın katkı sağlaması amaçlanmıştır. Bu yaklaşım, özellikle heterojen modellerin başarı oranları birbirinden farklı olduğunda, bireysel hataları azaltarak genelleme yeteneğini artırmak için etkili bir yöntemdir [46].

```
Test.java Test_Ensemble_Model_Agırlıklı.java
5 public class Test_Ensemble_Model_Agırlıklı
6 {
7     public static void main(String[] args) throws IOException {
8         // ----- Ağırlıklı Ensemble -----
9         double toplamDogrulukOrani=dogrulukOraniBayes+dogrulukOraniKNN+dogrulukOraniKNC+dogrulukOraniLogistic+dogrulukOraniAda;
10        double agırlıkBayes=dogrulukOraniBayes/toplamDogrulukOrani;
11        double agırlıkKNN=dogrulukOraniKNN/toplamDogrulukOrani;
12        double agırlıkKNC=dogrulukOraniKNC/toplamDogrulukOrani;
13        double agırlıkLogistic=dogrulukOraniLogistic/toplamDogrulukOrani;
14        double agırlıkAda=dogrulukOraniAda/toplamDogrulukOrani;
15
16        double[] agırlıklar = {agırlıkBayes, agırlıkKNN, agırlıkKNC, agırlıkLogistic, agırlıkAda};
17
18        List<Integer> agırlıklıTahminEnsembleList = new ArrayList<>();
19
20        for (int i = 0; i < tahminBayesList.size(); i++) {
21            double toplamPuan = 0.0;
22
23            // Her modelin tahmininin ağırlıklı toplamı hesaplanıyor
24            toplamPuan += agırlıklar[0] * tahminBayesList.get(i);
25            toplamPuan += agırlıklar[1] * tahminKNNList.get(i);
26            toplamPuan += agırlıklar[2] * tahminKNCList.get(i);
27            toplamPuan += agırlıklar[3] * tahminLogisticList.get(i);
28            toplamPuan += agırlıklar[4] * tahminAdaBoostList.get(i);
29
30            // Nihai tahmin i ise, 1 döndür; aksi halde 0
31            agırlıklıTahminEnsembleList.add(toplamPuan >= 0.5 ? 1 : 0);
32        }
33
34        // Doğruluk oranını hesapla
35        int dogruTahminSayisiAgırlıklıEnsemble = 0;
36        for (int i = 0; i < agırlıklıTahminEnsembleList.size(); i++) {
37            if (agırlıklıTahminEnsembleList.get(i).equals(gercekDeğerlerList.get(i))) {
38                dogruTahminSayisiAgırlıklıEnsemble++;
39            }
40        }
41
42        //*****Dogru ve Yanlış Tahmin Sayısı
43        // Dogru ve yanlış tahmin sayısı
44        int dogruTahminSayisiAgırlıklı = 0;
45        int yanlışTahminSayisiAgırlıklı = 0;
46
47        for (int i = 0; i < agırlıklıTahminEnsembleList.size(); i++) {
48            if (agırlıklıTahminEnsembleList.get(i).equals(gercekDeğerlerList.get(i))) {
49                dogruTahminSayisiAgırlıklı++;
50            } else {
51                yanlışTahminSayisiAgırlıklı++;
52            }
53        }
54
55        // Sonuçları ekrana yazdır
56        System.out.println("Dogru Tahmin Sayısı: "+dogruTahminSayisiAgırlıklı);
57        System.out.println("Yanlış Tahmin Sayısı: "+yanlışTahminSayisiAgırlıklı);
58        //*****Dogru ve Yanlış Tahmin Sayısı Bitiş
59
60        // Doğruluk oranını hesapla
61        double dogrulukOraniAgırlıklıEnsemble = (double) dogruTahminSayisiAgırlıklıEnsemble / gerçekDeğerlerList.size() * 100;
62        // Doğruluk oranını yazdır
63        System.out.printf("%sEnsemble [Ağırlık Oylaması] Doğruluk Oranı: %s %.2f%%\n", red, reset, dogrulukOraniAgırlıklıEnsemble);
64    }
65 }
```

Şekil 5.40. Ensemble Ağırlık Algoritması uygulama kodu.

5.3.5.2. Ensemble çoğunluk oylaması algoritması

Bu yöntemde ise tüm modeller eşit ağırlıkla değerlendirilmiştir. Her algoritmanın belirli bir örnek için yaptığı tahminler toplanmış ve en fazla sayıda önerilen sınıf sonucu nihai karar olarak kabul edilmiştir. Örneğin beş algoritmadan üçü bir hastayı "Diyabet" olarak sınıflandırırken, ikisi "Diyabet Değil" tahmininde bulunuyorsa, sistem bu hastayı "Diyabet" olarak değerlendirmiştir. Bu yaklaşım, bireysel model hatalarının dengelenmesini sağlar ve aşırı uç tahminlerin etkisini azaltır [46].

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class EnsembleModel {
5
6     public static int cogunlukTahminle(int tahminBayes, int tahminKNN, int tahminKNC, int tahminLogistic, int tahminAdaBoost)
7     {
8         int sayi1 = 0; // 1 olarak tahmin edilenlerin sayısı
9         int sayi0 = 0; // 0 olarak tahmin edilenlerin sayısı
10
11         // Tahminleri bir listeye ekle
12         List<Integer> tahminler = new ArrayList<Integer>();
13         tahminler.add(tahminBayes);
14         tahminler.add(tahminKNN);
15         tahminler.add(tahminKNC);
16         tahminler.add(tahminLogistic);
17         tahminler.add(tahminAdaBoost);
18
19         // Tahminler üzerinde döngü başlat
20         for (int tahmin : tahminler) {
21             if (tahmin == 1) {
22                 sayi1++; // Eğer tahmin 1 ise, sayi1'i artır
23             } else {
24                 sayi0++; // Aksi takdirde (tahmin 0 ise), sayi0'ı artır
25             }
26         }
27
28         // Eğer 1 olarak yapılan tahminler 0 olarak yapılan tahminlerden fazlaysa
29         if (sayi1 > sayi0) // Çoğunluk tahmini 1
30         {
31             return 1;
32         }
33         else if (sayi1==sayi0) // Esit sayıdaysa --- KNN ne ise cevap o olsun
34         {
35             if (tahminKNN==1)
36                 return 1;
37             else
38                 return 0;
39         }
40         else // Çoğunluk tahmini 0
41         {
42             return 0;
43         }
44     }
45
46     public static double hesaplaDogruOrani(List<Integer> gercekSonuclar, List<Integer> tahminler)
47     {
48         int dogruTahminSayisi = 0;
49
50         for (int i = 0; i < gercekSonuclar.size(); i++) {
51             if (gercekSonuclar.get(i).equals(tahminler.get(i))) {
52                 dogruTahminSayisi++;
53             }
54         }
55
56         // Doğruluk oranı
57         return (double) dogruTahminSayisi / gercekSonuclar.size();
58     }
59 }
```

Şekil 5.41. Ensemble Çoğunluk Algoritması uygulama kodu.

Her iki yöntem de Java diliyle özel olarak geliştirilmiş olup, gerçek zamanlı veri akışıyla entegre çalışmaktadır. Algoritmaların tahmin çıktıları ayrı ayrı toplandıktan sonra, bu iki ensemble yöntemiyle birleştirilmekte ve MSSQL veri tabanına sonuçlar kaydedilmektedir.

5.4. Tahmin Sonuçlarının Kayıt ve Görselleştirilme Süreci

Bu tez kapsamında geliştirilen gerçek zamanlı teşhis sisteminde, makine öğrenmesi modelleri tarafından üretilen tahmin sonuçları, analiz süreci tamamlandıktan hemen sonra **Kafka Consumer bileşeni tarafından doğrudan MSSQL veritabanına kaydedilmektedir**. Böylece, tüm işlem zinciri içerisinde herhangi bir dosya tabanlı tamponlama yapısına ihtiyaç duyulmadan, sonuçlar sistem bileşenleri arasında **senkronize, tutarlı ve düşük gecikmeli** şekilde iletilmektedir.

Bu yapı, sistemin **tam anlamıyla tetiklemeli (event-driven)** bir işleyişe sahip olmasını sağlamaktadır. Kafka Consumer, belirli aralıklarla gelen verileri analiz ettikçe sonuçlar eşzamanlı biçimde veritabanına yazılmakta; böylece herhangi bir manuel tetikleme ya da dışsal kontrol mekanizmasına ihtiyaç duyulmadan, tüm süreç otomatikleşmektedir. Veritabanına yazılan sonuçlara, kullanıcı arayüzü üzerinden anlık olarak erişilebilmekte; her tahmin işlemi sonucunda, C# tabanlı otomasyon sistemi ilgili veriyi doğrudan sorgulayarak ekranda göstermektedir.

Bu yaklaşım, geçmişte dosya tabanlı yapılarda karşılaşılabilen **senkronizasyon sorunları, dosya bütünlüğü riskleri ve bakım zorluklarını** ortadan kaldırmaktadır. Veritabanı tabanlı çözüm, hem **yüksek erişim hızı** hem de **veri tutarlılığı** açısından sistemin güvenilirliğini artırmakta, aynı zamanda **analitik çıktılar ve istatistiksel karşılaştırmalar** için sorgulanabilir ve genişletilebilir bir altyapı sunmaktadır.

Bu yapı sayesinde, sistem sadece anlık tahmin üretmekle kalmamakta, aynı zamanda bu tahminleri sürdürülebilir, izlenebilir ve geliştirilebilir bir yapıda sunarak, sağlık bilişimi alanındaki karar destek sistemlerine **yüksek güvenilirlikte, otomatik ve genişletilebilir** bir katkı sağlamaktadır.

Şekil 5.42’de, sistem tarafından gerçekleştirilen tahmin sürecine ait örnek doğruluk çıktılarından alınan ekran görüntüsüne yer verilmiştir. Bu ekranlar, modelin işleyişinin doğru olduğunu belgelemek amacıyla geliştirme aşamasında alınmıştır:

```

"C:\Program Files\Java\java-1.8.0.201\bin\java.exe" ...
Algoritmalar Çalıştırılıyor...

ADA Algoritması Başlatıldı!
Doğru tahmin sayısı: 366
Yanlış tahmin sayısı: 234
Doğruluk Oranı: 61.0
Toplam süre: 4.733 milisaniye

KNN Algoritması Başlatıldı!
Doğru tahmin sayısı: 483
Yanlış tahmin sayısı: 117
Doğruluk Oranı: 80.5%
Toplam süre: 1.529 saniye

KNC Algoritması Başlatıldı!
Doğru tahmin sayısı: 399
Yanlış tahmin sayısı: 201
Doğruluk Oranı: 66.5%
Toplam süre: 1.388 saniye

Logistic Regression Algoritması Başlatıldı!
Satır 1 bitti.
Satır 151 bitti.
Satır 301 bitti.
Satır 451 bitti.
Satır 2 bitti.
Satır 302 bitti.
Satır 452 bitti.
Satır 152 bitti.
Satır 3 bitti.
Satır 303 bitti.
...
Satır 450 bitti.
Satır 300 bitti.
Doğru tahmin sayısı: 361
Yanlış tahmin sayısı: 239
Doğruluk Oranı: 60.16666666666667
Toplam süre: 18.73 saniye

Bayes Algoritması Başlatıldı!
1.satır bitti
2.satır bitti
3.satır bitti
4.satır bitti
5.satır bitti
6.satır bitti
...
598.satır bitti
599.satır bitti
600.satır bitti
Doğru tahmin sayısı: 474
Yanlış tahmin sayısı: 126
Doğruluk Oranı: 79.0
Toplam süre: 1.046 saniye

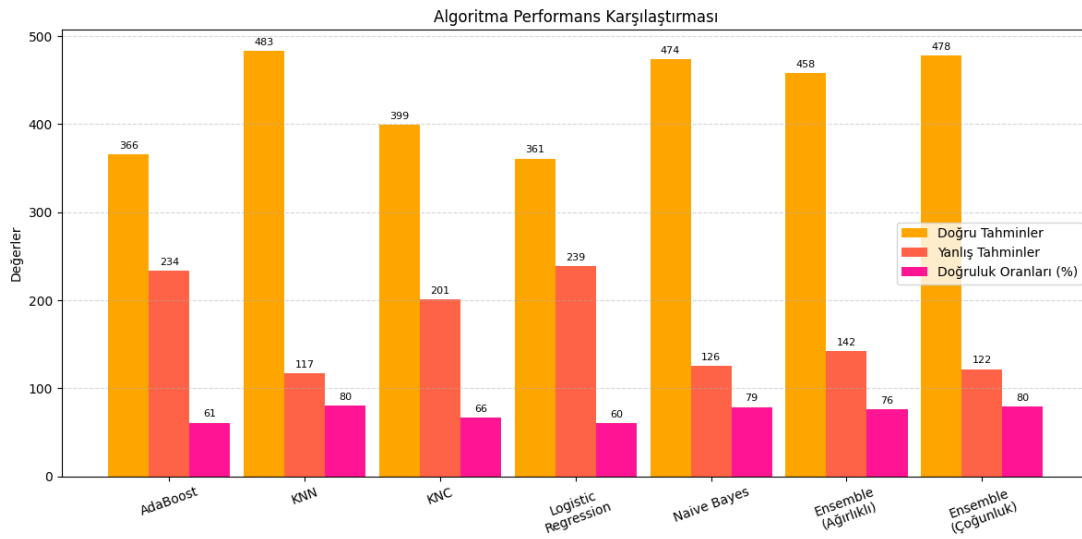
Doğru Tahmin Sayısı: 458
Yanlış Tahmin Sayısı: 142
Ensemble [Ağırlık Oylaması] Doğruluk Oranı: 76,33%

Doğru Tahmin Sayısı: 478
Yanlış Tahmin Sayısı: 122
Ensemble [Çoğunluk Oylaması][KNN-KNC-Logistic-ADA] Doğruluk Oranı: 79,67%

```

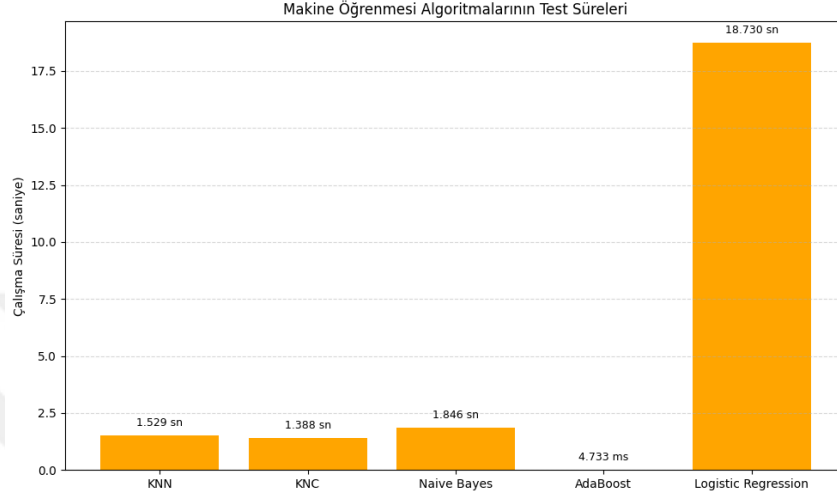
Şekil 5.42. Algoritmaların doğruluk çıktıları.

Makine öğrenmesi algoritmalarının diyabet teşhisindeki performanslarını karşılaştırmak ve sistemin etkinliğini ortaya koymak amacıyla, her algoritma için doğru ve yanlış tahmin sayıları ile birlikte doğruluk oranları görsel olarak sunulmuştur. Şekil 5.43'te sekiz farklı sınıflandırma yöntemine ait doğruluk oranları karşılaştırmalı olarak gösterilmiştir.



Şekil 5.43. Algoritmaların performans karşılaştırması.

Sistemin gerçek zamanlı veri akışına verdiği tepki süresi de performans açısından kritik bir faktördür. Bu nedenle, her bir algoritmanın test verisi üzerindeki işlem süresi de ölçülerek görsel olarak karşılaştırılmıştır. Her algoritma, 600 test verisi ile test edilmiş; geçen süreler saniye cinsinden kaydedilmiştir. Şekil 5.44, bu işlem sürelerini sunmaktadır.



Şekil 5.44. Algoritmaların test süreleri.

Bu görsel analizler, yalnızca doğruluk oranlarının değil, aynı zamanda sistem yanıt süresi gibi gerçek zamanlı kullanım açısından kritik metriklerin de değerlendirilmesine olanak tanımaktadır. Özellikle KNN algoritması, hem yüksek doğruluk oranı (%80.5) hem de oldukça kısa sürede sonuç üretmesiyle öne çıkmıştır. Buna karşın bazı algoritmalar, daha yüksek işlem süresiyle daha düşük doğruluk sağlamış; bu da model seçimi yapılırken yalnızca doğruluğun değil, hız ve kaynak verimliliği gibi kriterlerin de dikkate alınması gerektiğini göstermektedir.

5.5. MSSQL Veritabanı Entegrasyonu

Tahmin sonuçlarının kalıcı olarak saklanması, sistemin geriye dönük analizlere açık olması ve veri bütünlüğünün sağlanması amacıyla bu çalışmada Microsoft SQL Server (MSSQL) tabanlı bir veritabanı yapısı kullanılmıştır. MSSQL, ACID (Atomicity, Consistency, Isolation, Durability) garantisi sunan güçlü bir ilişkisel veritabanı yönetim sistemi (RDBMS) olup, büyük veri ortamlarında bile güvenli ve tutarlı veri saklama yetenekleriyle öne çıkmaktadır [47].

Geliştirilen sistemde, **Kafka Consumer bileşeni hem analiz hem de tahmin sonuçlarının kalıcı olarak kaydedilmesi** görevlerini doğrudan üstlenmektedir. Tüm

tahmin süreci tamamlandıktan sonra, elde edilen sonuçlar anında **MSSQL veritabanına kaydedilmekte** ve böylece ayrı bir tamponlama veya dosya tabanlı ara katmana ihtiyaç duyulmamaktadır.

Bu mimari tercih, sistemin **tamamen tetiklemeli (event-driven)** bir şekilde çalışmasını sağlamakta; her yeni hasta verisi Kafka tarafından alındığında analiz süreci başlamakta, sonuç elde edildiği anda veritabanı işlemi otomatik olarak gerçekleşmektedir. Böylece analiz ve kayıt adımları arasında herhangi bir gecikme ya da manuel işlem ihtiyacı oluşmadan, süreç baştan sona otomatikleşmiş bir akışla yürütülmektedir.

Bu yapı aynı zamanda **gevşek bağlılık (loosely coupled)** ilkesine uygun olarak tasarlanmış olup, bileşenlerin birbirine sıkı bağımlılıklar olmadan çalışmasına imkân tanımaktadır. Kafka Consumer bileşeni analiz sürecini tamamlarken aynı anda başka bileşenlerin çalışmasını engellemez; benzer şekilde, C# tabanlı kullanıcı arayüzü de veritabanından doğrudan sorgu yaparak tahmin sonuçlarını anlık olarak görüntüleyebilir.

Veritabanı temelli bu çözüm, geçmişte kullanılan geçici dosya sistemlerinin doğurabileceği senkronizasyon zorluklarını ortadan kaldırmakta; sistemin **izlenebilirliği, güvenilirliği ve sürdürülebilirliği** açısından önemli avantajlar sunmaktadır.

Veri tabanı yapısı aşağıda detaylandırılmıştır:

- **Veritabanı Adı:** vt_ozelYukselH
- **Tablo:** tbl_hastaData

Bu tabloda hem hastaya ait ölçüm değerleri hem de farklı makine öğrenmesi algoritmalarına ait tahmin sonuçları yer almaktadır. Ayrıca, gerçek etiket (Outcome) bilgisi de kayıt altına alınarak model doğruluğu izlenebilmektedir.

Tablo 5.1. Hastalık Sonuçlarının Tutulduğu Tablo.

Sütun Adı	Açıklama
person_id	Hastanın sistemdeki benzersiz ID'si
Pregnancies, Glucose...	Hastaya ait medikal ölçüm verileri
BayesPrediction, KNNPrediction...	Her bir algoritmanın tahmin sonucu
True	Gerçek sonuç (diyabet hastalığı var/yok)

- **Tablo:** tbl_hastaBilgi

Hastanın temel demografik bilgileri ile birlikte, varsa görsel verileri (fotoğraf) ayrı bir tabloda saklanmaktadır. Bu sayede sağlık geçmişi ve kişisel bilgiler birbiriyle ilişkilendirilebilir yapıdadır.

Tablo 5.2. Hasta Bilgileri Tablosu.

Sütun Adı	Açıklama
hasta_id	Hastaya ait benzersiz kimlik
tcNo, adi, soyadi, dogumTarihi, mailAdresi	Kişisel ve iletişim bilgileri
fotograf (varbinary), dosyaYoluFotograf	Hastanın fotoğrafı ve görsel yolu

Bu yapı sayesinde:

- **Tahmin tamamlandığında** doğrudan tbl_hastaData tablosuna kayıt edilir.
- **Geçmiş veriler**, person_id veya hasta_id üzerinden sorgulanabilir; istatistiksel analizler, görselleştirmeler veya harici raporlama araçları ile entegre analizler yapılabilir.
- **Veritabanı yapısı**, sistemin sürdürülebilirliği, geçmişe dönük kayıtların denetlenebilirliği ve gerektiğinde üçüncü taraf sağlık yazılımlarıyla entegrasyonu için uygun altyapıyı sağlamaktadır.



6. SONUÇLAR VE DEĞERLENDİRME

6.1. Ön İşleme Sürecinin Etkileri

Makine öğrenmesi algoritmalarının başarısı, büyük ölçüde verinin kalitesine ve doğru ön işleme tekniklerinin uygulanmasına bağlıdır. Gerçek dünyadan elde edilen ham veriler genellikle eksik, hatalı, tutarsız ya da dengesiz olabilir. Bu nedenle, modelin eğitilmesinden önce uygulanan veri ön işleme adımları; modelin doğruluk, genelleme ve kararlılık düzeyini doğrudan etkiler [48].

Bu çalışmada kullanılan veri seti üzerinde kapsamlı bir ön işleme süreci gerçekleştirilmiştir. Süreç içerisinde:

- Eksik veriler analiz edilerek uygun istatistiksel yöntemlerle doldurulmuş,
- Anlamsız ve gürültülü değerler temizlenmiş,
- Uç değerlerin etkisini azaltacak normalizasyon adımları uygulanmış,
- SMOTE (Synthetic Minority Over-sampling Technique) yöntemiyle dengesiz sınıf dağılımı giderilmiş,
- CTGAN (Conditional Tabular GAN) algoritması ile veri artırımı sağlanarak, modelin öğrenme yetisi genişletilmiştir [49].

Literatürde yapılan bazı önceki çalışmalarda, veri ön işleme süreci sınırlı tutulmuş veya tamamen göz ardı edilmiştir. Doğrudan ham veri ile model eğitimi gerçekleştirilmiş, bu da modelin genelleme gücünü düşürmüş ve doğruluk oranlarının suni olarak yüksek kalmasına neden olmuştur. Oysa bu çalışmada uygulanan detaylı ön işleme adımları, algoritmaların daha gerçekçi ve güvenilir sonuçlar üretmesini sağlamıştır.

Özellikle SMOTE ve CTGAN gibi ileri düzey tekniklerin uygulanması, sınıf dengesizliğini gidererek azınlık sınıfların daha doğru şekilde temsil edilmesini sağlamıştır. Böylelikle, sadece yüksek doğruluk oranı değil, aynı zamanda modelin farklı sınıfları ayırt edebilme yeteneği de artmıştır. Bu durum, özellikle sağlık alanında kritik hastalıkların erken tanısında son derece önemlidir.

Bununla birlikte, kaba kümeleme (Rough Set Theory) algoritması üzerinden yapılan özellik indirgeme çalışması sonucunda veri setinde anlamlı bir özellik elenmesi gerçekleşmemiştir. Başka bir ifadeyle, reduct kümesi orijinal özniteliklerle birebir örtüşmüştür. Bu nedenle, kaba kümeleme indirgeme açısından doğrudan etkili olmamakla birlikte, karar kuralları üretimi ve örüntü keşfi açısından katkı sunmuştur.

Sonuç olarak, yapılan ön işleme adımları veri kalitesini artırmış ve geliştirilen makine öğrenmesi algoritmalarının gerçekçi ve genellenebilir modeller oluşturmaya olanak tanımıştır. Ancak bu sürecin doğrudan doğruluk oranını yükseltme amacı taşımadığı, modelin güvenilirliğini ve veri temsiliyetini artırmayı hedeflediği unutulmamalıdır.

6.2. Kaba Kümeleme Sonuçlarının Analizi

Kaba Kümeleme Teorisi (Rough Set Theory - RST), belirsizlik ve eksik bilgi içeren veri kümelerinde örüntü çıkarımı ve karar kuralları üretimi için etkili bir yöntemdir. Zdzisław Pawlak tarafından geliştirilen bu teori, özellikle özellik seçimi ve kural tabanlı sınıflandırma gibi karar destek süreçlerinde sıkça kullanılmaktadır [29].

Bu çalışmada, gerçek zamanlı veri analitiği sürecine dâhil edilen RST yaklaşımı, özellikle karar kuralları üretimi ve veri örüntülerinin anlaşılması amacıyla uygulanmıştır. C# diliyle geliştirilen bir arayüz aracılığıyla, veri seti üzerinde **alt ve üst yaklaşım kümeleri, çekirdek küme (core) ve redukt (reduct)** hesaplamaları gerçekleştirilmiştir. Ancak uygulanan RST analizinde, elde edilen **redukt kümesi**, orijinal özellik kümesiyle birebir aynı çıkmıştır. Bu durum, veri setinde herhangi bir öznitelik indirgeme ihtiyacı olmadığını ve mevcut özniteliklerin tamamının sınıflandırma açısından ayırt edici bilgi taşıdığını göstermektedir.

Kaba kümeleme teorisi bu bağlamda, doğrudan öznitelik sayısını azaltmamış olsa da, özellikle **kural tabanlı örüntülerin çıkarılması** açısından katkı sunmuştur. Karar tabloları üzerinden elde edilen **karar kuralları**, veri kümesinde hangi öznitelik kombinasyonlarının belirli sonuçlara (örneğin, diyabet teşhisi) yol açtığını açıkça ifade etmiş ve sistemin anlaşılabilirliğini artırmıştır.

Örneğin, şu tür kurallar elde edilmiştir:

- Eğer Glukoz düzeyi “Yüksek” ve Yaş “İleri” ise → Diyabet Hastasıdır.
- Eğer Vücut Kitle İndeksi “Normal” ve Gebelik sayısı “Düşük” ise → Diyabet Hastası Değildir.

Bu tip kurallar, sadece makine öğrenmesi modellerinin kararlarını desteklemekle kalmamış, aynı zamanda uzman sistemlerin geliştirilmesi için de zemin oluşturmıştır.

RST'nin karar destek sistemleri ile entegre şekilde kullanımı, özellikle tıbbi tanı sistemlerinde hem doğruluk hem de açıklanabilirlik (explainability) açısından büyük avantaj sağlar. Nitekim, çalışmanın yazılım bileşenleri arasında yer alan karar kuralı üretim modülü, geliştirilen hastane otomasyon sistemi ile paralel çalışarak yorumlanabilirlik düzeyini artırmıştır.

Sonuç olarak, kaba kümeleme teorisi bu çalışmada indirgeme fonksiyonu açısından sınırlı bir fayda sağlamış olsa da, karar kuralları çıkarımı ve örüntü analizi açısından önemli bir rol oynamıştır. Gelecekte daha yüksek hacimli ve çeşitlilik içeren veri kümeleriyle yapılacak analizlerde, RST'nin indirgeme fonksiyonu da daha belirgin bir şekilde ortaya çıkabilir.

6.3. Gerçek Zamanlı Tahminleme Performansı

Bu çalışmada geliştirilen gerçek zamanlı veri işleme mimarisi, klasik statik veri analizi yaklaşımlarından farklı olarak canlı veri akışı üzerinden tahminleme gerçekleştirmektedir. Uygulama kapsamında, C# ile geliştirilen bir hastane otomasyon sistemi, hastaya ait verileri Apache Kafka üzerinden Java tabanlı makine öğrenmesi algoritmalarına gerçek zamanlı olarak iletmektedir. Kafka tarafından veriler işlenip tahmin sonuçları elde edildikten sonra sonuçlar doğrudan MSSQL veritabanına kaydedilmektedir.

Gerçek zamanlı mimari, veri iletiminde Apache Kafka'nın Producer-Consumer yapısını temel almakta; Java tarafında ise Logistic Regression, KNN, KNC gibi algoritmalar ile sınıflandırma gerçekleştirilmektedir. Bu algoritmalarından bazıları, özellikle test ve değerlendirme sürecinde çok iş parçacıklı (multi-threaded) mimari ile çalışacak şekilde geliştirilmiş olup; bu sayede 600 test verisinin eş zamanlı olarak analiz edilmesi sağlanmış ve algoritmaların performans açısından da karşılaştırılabilmesine olanak tanımıştır.

Ancak, geliştirilen sistemin nihai amacı algoritmaların başarımları değil; algoritmaları birer araç olarak kullanılıp gerçek zamanlı sistemi ön plana çıkartmaktır. Bu nedenle gerçek zamanlı veri akışı sırasında sistem tekil (single-threaded) veri işleme yaklaşımını benimsemektedir. Yani, her bir hasta verisi bağımsız olarak analiz

edilmekte ve sonuçlar sistem tarafından anlık olarak kaydedilmektedir. Bu tercih, çok iş parçacıklı yapının getirdiği karmaşayı önlemek, işlem sıralamasını korumak ve analiz süreçlerini daha kontrol edilebilir hâle getirmek amacıyla yapılmıştır. Sistem, her hasta için sıralı ve güvenilir bir teşhis üretimi hedeflemektedir.

Gerçek zamanlılığın sağlıklı biçimde sürdürülebilmesi amacıyla iş bölümlemesi yapılmıştır. Kafka Producer, C# hastane otomasyonuna girilen hasta verilerini doğrudan Kafka Consumer'a iletmektedir. Kafka Consumer ise gelen verileri işlemekte ve elde ettiği tahmin verilerini MSSQL veritabanına kaydetmektedir. Bu görev paylaşımı, sistemin eş zamanlı şekilde çalışmasına olanak tanımakta; analiz ve kayıt işlemlerinin ayrı bileşenlerce yürütülmesini sağlayarak darboğazları önlemektedir.

Özetle; analiz süreci ve veritabanına kayıt süreci Kafka Consumer bileşeni tarafından gerçekleştirilmektedir. Bu yapı aynı zamanda, Kafka bileşenlerinin kuyruktaki diğer verileri işlemesini geciktirmemekte; veri akışı kesintisiz biçimde devam etmektedir. Ayrıca dosya tabanlı sistemlere göre, **veritabanı temelli bu yapı uzun süreli kullanımda ortaya çıkabilecek dosya yönetimi sorunlarını, bellek tüketimini ve senkronizasyon karmaşasını önleyerek sistem kaynaklarının daha verimli kullanılmasını sağlamaktadır.** Süreçlerin otomatik ve tetiklemeli (event-driven) bir şekilde gerçekleşmesi, sistemdeki tüm bileşenlerin gerçek zamanlılık ilkesine uygun biçimde senkronize çalışmasına imkân tanımakta ve bu da özellikle sağlık gibi zaman kritik uygulamalarda önemli bir avantaj sunmaktadır..

Tablo 6.1'de, algoritmaların bireysel olarak performansları ele alınmış, test doğruluk oranları elde edilmiş ve 600 adet test verisi için elde edilen işlem süreleri sunulmuştur. Bu değerler, sistemin doğruluk ölçümüne ve algoritmaların işlem süresi açısından değerlendirilmesine olanak tanımaktadır:

Tablo 6.1. Gerçek Zamanlı Tahminleme Mimarisinde Kullanılan Algoritmaların Performans Sonuçları.

Algoritma	Doğru Tahmin Sayısı	Yanlış Tahmin Sayısı	Doğruluk Oranı (%)	Süre (saniye)
KNN	483	117	80.50	1.529
KNC	399	201	66.50	1.388
Logistic Regression	361	239	60.50	18.73
Naive Bayes	474	126	79.00	1.046
AdaBoost	366	234	61.00	0.004
Ensemble – Ağırlık Oylaması	458	142	76.33	-
Ensemble – Çoğunluk Oylaması	478	122	79.67	-

Tablo 6.1'de görüldüğü üzere, ensemble yaklaşımları, bireysel makine öğrenmesi algoritmalarına kıyasla genel olarak daha yüksek doğruluk oranları sunmuştur. Özellikle çoğunluk oylaması yöntemi, %79.67 doğruluk oranı ile en başarılı model olarak öne çıkmıştır. KNN algoritması ise %80.50'lik bireysel başarı oranı ile dikkat çekmiştir.

Bu doğruluk oranları, bazı literatür çalışmalarındaki sonuçlarla karşılaştırıldığında rekabetçi seviyededir. Ancak bu çalışmanın öncelikli amacı, yalnızca yüksek doğruluk oranlarına ulaşmak değil; gerçek zamanlı veri işleme mimarisi altında çalışan, pratikte uygulanabilir ve tam entegre bir karar destek sistemi tasarlamaktır. Bu kapsamda, sadece algoritma başarımı değil, **veri iletimi, analiz ve kayıt işlemlerinin optimize edilmiş şekilde ayrıştırılması**, sistem mimarisinin temel hedeflerinden biri olmuştur.

Elde edilen sonuçların değerlendirilmesinde aşağıdaki faktörler etkili olmuştur:

- Bu çalışmada statik veri analizinden ziyade gerçek zamanlı veri akışı ile tahmin gerçekleştirilmiştir. Bu durum, klasik model eğitim–test senaryolarına göre daha zorlu bir ortam sunmaktadır.

- Veri setine kapsamlı ön işleme uygulanmış; anlamsız ve eksik veriler temizlenmiş, sınıf dengesizliği SMOTE ile giderilmiş ve CTGAN ile sentetik örnekler üretilmiştir.
- Literatürdeki bazı çalışmaların aksine, bu sistemde veri seti 2000 örneğe çıkarılmış ve %70–%30 oranında eğitim/test ayrımı yapılmıştır. Gerçek zamanlı test süreci yalnızca bu test verisi (600 kayıt) üzerinden yürütülmüştür.
- Makine öğrenmesi algoritmalarının bazıları (ör. Logistic Regression, Naive Bayes) test süreçlerinde çok iş parçacıklı (multi-threaded) yapıyla kodlanmıştır. Ancak, gerçek zamanlı hasta verilerinde tekil iş parçacıklı yapı tercih edilmiştir. Bu mimari, sistemin kararlılığını, işlem sırasını ve güvenilirliğini artırmıştır.
- Ayrıca sistemde, sınıflandırma aşamasından önce veri setine Kaba Kümeleme (Rough Set Theory) uygulanmış ve karar kuralları üretilmiştir. Bu işlem sınıflandırma doğruluğunu doğrudan etkilememekle birlikte, sistemin yorumlanabilirliğini ve şeffaflığını artırmıştır [30], [46].

Sonuç olarak, geliştirilen sistem; Apache Kafka tabanlı veri iletim altyapısı, Java ile entegre edilmiş makine öğrenmesi algoritmaları ve C# ile geliştirilen hastane otomasyon arayüzü aracılığıyla, uçtan uca entegre ve gerçek zamanlı çalışan bir tahmin mimarisi olarak başarıyla yapılandırılmıştır. **Event-driven (olaya dayalı) yapısı sayesinde tüm süreçler otomatik biçimde tetiklenmekte; analiz ve veritabanı kayıt işlemleri birbirinden ayrılarak sistem bileşenleri arasında eş zamanlılık sağlanmaktadır.** Dosya tabanlı geçici tampon yapılar yerine doğrudan veritabanı etkileşimi tercih edilerek gecikme süreleri azaltılmış, veri işleme süreci daha güvenilir ve izlenebilir hâle getirilmiştir. Klinik senaryolar üzerinde test edilen bu mimari, özellikle karar destek sistemleri kapsamında **hız, doğruluk ve sürdürülebilirlik** açısından önemli katkılar sunmaktadır.

7. GENEL SONUÇLAR VE ÖNERİLER

7.1. Çalışmanın Katkıları

Bu tez çalışması, sadece geleneksel makine öğrenmesi uygulamalarıyla sınırlı kalmayıp, gerçek zamanlı veri işleme, dağıtık sistemler, veri kalitesi, sistem optimizasyonu ve karar destek sistemleri gibi çok boyutlu konuları bütünsel bir yaklaşımla ele alarak sağlık bilişimi alanına önemli katkılar sunmuştur. Katkılar aşağıdaki şekilde özetlenebilir:

- **Gerçek Zamanlı ve Dağıtık Mimari Kurulumu:** Apache Kafka, Apache Zookeeper ve Java tabanlı Producer–Consumer yapısıyla çalışan gerçek zamanlı bir veri akış mimarisi başarıyla kurulmuştur. C# ile geliştirilen hastane otomasyon sistemi üzerinden alınan hasta verileri anlık olarak Java tabanlı sınıflandırma modellerine iletilmiş ve tahmin sonuçları MSSQL veri tabanına kaydedilmiştir.
- **Makine Öğrenmesi Algoritmalarının Elle Kodlanması:** Logistic Regression, Naive Bayes, KNN, KNC ve AdaBoost algoritmaları, Java programlama dili kullanılarak manuel olarak geliştirilmiştir. Bu algoritmalar, sistem testleri ve doğruluk karşılaştırmaları sırasında çok iş parçacıklı (multi-threaded) yapı ile çalıştırılmış; böylece algoritmaların eğitim ve test süreleri optimize edilmiştir. Ancak, gerçek zamanlı analiz sürecinde her bir veri örneği tekil iş parçacığında sıralı olarak işlenmektedir. Çünkü sistemin temel amacı yüksek doğruluklu modeller üretmekten ziyade, gerçek zamanlı ve uçtan uca entegre çalışan bir karar destek mimarisi kurmaktır. Bu nedenle, eşzamanlı veri işleme gibi yapıların kullanımı, gerçek zamanlılık gereksinimlerine ters düşerek sistemin gecikme sürelerini artırmakta ve izlenebilirliği zorlaştırmaktadır. Gerçek zamanlı sistemde, Kafka üzerinden gelen her hasta verisi zaten birer birer (FIFO mantığıyla) kuyruklanmakta ve işlem sırasını beklemektedir. Bu bağlamda, paralel iş parçacıkları kullanılarak aynı verinin birden fazla algoritma ile aynı anda analiz edilmesi, teorik olarak işlem süresini azaltıyor gibi görünse de, sistemin tepki süresini artırmakta ve kaynak

kullanımını verimsizleştirmektedir. Bu nedenle, analiz sürecinin senkron biçimde yürütülmesi sistemin genel performansı, kararlılığı ve sürdürülebilirliği açısından daha uygun bulunmuştur. Algoritmaların bireysel olarak manuel şekilde geliştirilmesi, aynı zamanda bu yapıların üzerinde topluluk (ensemble) modeller kurulmasına olanak tanımış; çoğunluk oylaması (majority voting) ve ağırlıklı oylama (weighted voting) yaklaşımları ile elde edilen sonuçlar, sistemin karar doğruluğunu artırmıştır.

- **Ensemble Yöntemlerin Gerçek Zamanlı Uygulaması:** Ağırlıklı oylama ve çoğunluk oylaması gibi ensemble modeller, bireysel algoritmaların çıktılarından yararlanarak doğruluk oranlarını iyileştirmiştir. Bu yapı, hem tahmin başarımını artırmış hem de daha sağlam karar sistemleri geliştirilmesine katkı sunmuştur.
- **Ön İşleme ve Veri Kalitesi Duyarlılığı:** Eksik verilerin düzeltilmesi, aykırı verilerin temizlenmesi, sınıf dengesinin SMOTE ile sağlanması ve CTGAN ile sentetik veri üretimi gerçekleştirilmiştir. Bu sayede modellerin daha dengeli ve gerçekçi sonuçlar üretmesi sağlanmıştır.
- **Veri Setinin Dengeli Ayrımı ve Performans Değerlendirmesi:** 2000 örnekten oluşan veri seti %70 eğitim – %30 test oranında dengeli şekilde bölünmüş, test verileri üzerinden model başarımı ve işlem süresi karşılaştırmalı olarak görselleştirilmiştir.
- **Kaba Kümeleme ile Karar Açıklanabilirliği:** Rough Set Theory ile karar kuralları çıkarılmış, sistemin açıklanabilirliği artırılmıştır. Böylece sadece “ne” sorusuna değil, “neden” sorusuna da cevap üretilebilmiştir.
- **Dağıtık Sistemlerin Etkin Kullanımı:** Kafka’nın FIFO tabanlı kuyruğu, tekil Consumer yapısı ve Zookeeper ile senkronize edilmiş sistem düzeni, veri kayıpsızlığı, işlem sıralaması ve sistem sürekliliği açısından önemli bir yapı ortaya koymuştur.

Bu yönleriyle çalışma, akademik araştırmaların yanı sıra gerçek dünyada uygulanabilecek, ölçeklenebilir ve anlaşılabilir bir karar destek sistemine örnek teşkil etmektedir.

7.2. Gelecek Çalışmalar için Öneriler

Bu çalışma, sağlıkta dijitalleşme ve veri odaklı karar verme sistemlerine yönelik sağlam bir temel sunmuştur. Ancak daha ileri düzey iyileştirmelerle sistemin kapsamı genişletilebilir. Aşağıda bazı öneriler sıralanmıştır:

- **Yeni Algoritmaların Entegrasyonu:** Random Forest, XGBoost, Support Vector Machines gibi daha gelişmiş modeller veya derin öğrenme yaklaşımları sisteme entegre edilerek sınıflandırma başarımı artırılabilir.
- **Otomatik Model Ayarı (Hyperparameter Tuning):** Grid Search, Random Search veya Bayes Optimizasyonu gibi tekniklerle algoritmaların hiperparametreleri otomatik optimize edilebilir.
- **Web Tabanlı Görselleştirme Araçları:** Gerçek zamanlı tahmin sonuçlarının Grafana, Kibana veya React tabanlı özel dashboardlarla sunulması, sistem çıktılarının daha anlaşılır hale gelmesini sağlayacaktır.
- **Uyarı ve Alarm Sistemleri:** Belirli eşiklerin aşıldığı durumlarda sistemin SMS, e-posta veya sistem bildirimi yoluyla sağlık personelinin uyarılması sağlanabilir.
- **Gerçek Hasta Verisi ile Uygulama:** Anonimleştirilmiş ve etik kurul onaylı gerçek hasta verileri ile yapılan testler, sistemin klinik geçerliliğini artıracaktır.
- **Mobil Uygulama Geliştirme:** Geliştirilen sistemin bir mobil uygulama ile desteklenmesi, doktorların anlık tahmin sonuçlarına mobil cihazlardan erişimini sağlayarak karar süreçlerini hızlandırabilir.
- **Veri Ayrımı Stratejileri:** Çapraz doğrulama, zaman serisi testi gibi alternatif veri bölme stratejileri ile sistemin genel başarımı daha farklı senaryolarda ölçülebilir.
- **Karar Açıklanabilirliğinin Artırılması:** RST'ye ek olarak SHAP, LIME gibi açıklanabilir yapay zeka (XAI) araçlarının kullanımı, sistemin şeffaflığını ve etik sorumluluğunu güçlendirecektir [50].

Bu öneriler doğrultusunda yapılacak çalışmalar, hem bilimsel derinliği artıracak hem de gerçek dünya uygulamaları için daha fonksiyonel çözümler geliştirilmesine olanak sağlayacaktır.



KAYNAKLAR

- [1] World Health Organization. (2023, Nisan 4). Diabetes. <https://www.who.int/news-room/fact-sheets/detail/diabetes>
- [2] Doğan, M., & Telçeken, S. (2004). *Kaba Kümeler Teorisi Yardımı İle Büyük Veri Topluluklarının Analizi*. <https://doi.org/10.13140/2.1.1664.5924>
- [3] Smith, J. W., Everhart, J. E., Dickson, W. C., Knowler, W. C., & Johannes, R. S. (1988). Using the adap learning algorithm to forecast the onset of diabetes mellitus. *Proceedings of the Annual Symposium on Computer Application in Medical Care*, 261-265. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2245318/>
- [4] Yüksel, M., & Demirci, H. (2025, Mayıs 16–17). Pima Indians verisetinin Kaba Kümeleme Teorisi kullanılarak indirgenmesi ve karar kurallarının çıkartılması. *14. IMASCON Uluslararası Marmara Fen ve Sosyal Bilimler Kongresi*, Kocaeli, Türkiye.
- [5] American Diabetes Association. (2022). Classification and diagnosis of diabetes. *Diabetes Care*, 45(Supplement_1), S17–S38. <https://doi.org/10.2337/dc22-S002>
- [6] International Diabetes Federation. (2021). *IDF diabetes atlas* (10th ed.). <https://diabetesatlas.org/>
- [7] Haux, R. (2006). Health information systems—Past, present, future. *International Journal of Medical Informatics*, 75(3-4), 268-281. <https://doi.org/10.1016/j.ijmedinf.2005.08.002>
- [8] Raghupathi, W., & Raghupathi, V. (2014). Big data analytics in healthcare: Promise and potential. *Health Information Science and Systems*, 2(1), 3. <https://doi.org/10.1186/2047-2501-2-3>
- [9] Zhang, Y., Qiu, M., Tsai, C.-W., Hassan, M. M., & Alamri, A. (2017). Healthcps: Healthcare cyber-physical system assisted by cloud and big data. *IEEE Systems Journal*, 11(1), 88-95. <https://doi.org/10.1109/JSYST.2015.2460747>
- [10] Pawlak, Z. (1982). Rough sets. *International Journal of Computer and Information Sciences*, 11(5), 341–356. <https://doi.org/10.1007/BF01001956>
- [11] Komorowski, J., Polkowski, L., & Skowron, A. (1998). Rough sets: A tutorial. In S. K. Pal & A. Skowron (Eds.), *Rough fuzzy hybridization: A new trend in decision making* (pp. xx–xx). Springer.
- [12] Taşci, M., & Yildirim, M. T. (2020). Uçuş verilerinin kaba küme yöntemi kullanılarak indirgenmesi. *Ejons International Journal on Mathematic, Engineering and Natural Sciences*, 4(15), 625-637. <https://doi.org/10.38063/ejons.297>
- [13] Warren, J., & Marz, N. (2015). *Big Data: Principles and best practices of scalable realtime data systems*. Simon and Schuster.

- [14] Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A distributed messaging system for log processing. *Proceedings of the NetDB Workshop*.
- [15] Ristevski, B., & Chen, M. (2018). Big data analytics in medicine and healthcare. *Journal of Integrative Bioinformatics*, 15(3), 1–10. <https://doi.org/10.1515/jib-2017-0030>
- [16] Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., ... Dean, J. (2019). A guide to deep learning in healthcare. *Nature Medicine*, 25(1), 24–29. <https://doi.org/10.1038/s41591-018-0316-z>
- [17] Kumar, Y., Singh, S., & Singh, M. (2021). Prediction of diabetes using classification algorithms. *Procedia Computer Science*, 167, 2464–2471. <https://doi.org/10.1016/j.procs.2020.03.302>
- [18] Kavakiotis, I., Tsave, O., Salifoglou, A., Maglaveras, N., Vlahavas, I., & Chouvarda, I. (2017). Machine learning and data mining methods in diabetes research. *Computational and Structural Biotechnology Journal*, 15, 104–116. <https://doi.org/10.1016/j.csbj.2016.12.005>
- [19] Sisodia, D., & Sisodia, D. S. (2018). Prediction of diabetes using classification algorithms. *Procedia Computer Science*, 132, 1578–1585. <https://doi.org/10.1016/j.procs.2018.05.122>
- [20] Johnson, A. E. W., Pollard, T. J., & Mark, R. G. (2020). Reproducibility in critical care: A mortality prediction case study. In *Open access machine learning in healthcare*. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7051842/>
- [21] Scikit-learn Documentation. (2024). *Imputation of missing values*. <https://scikit-learn.org/stable/modules/impute.html>
- [22] Buda, M., Maki, A., & Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106, 249–259. <https://doi.org/10.1016/j.neunet.2018.07.011>
- [23] Uduma, E. E., & Nnanna, A. (n.d.). *Enhancing diabetes prediction through GAN and SMOTE-based data augmentation*. Academia.edu. https://www.academia.edu/download/123814482/Ebuka_Thesis.pdf
- [24] Maganti Khyathi Sarayu, Shaik Ayesha Bhanu, Karanam Deekshitha, Maduri Meghana, & Joseph, I. T. (2024). Diabetes Prediction using SMOTE and Machine Learning. *Proceedings of the IEEE International Conference on ICICI*, 15–20. <https://doi.org/10.1109/icici62254.2024.00011>
- [25] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16, 321–357. <https://doi.org/10.1613/jair.953>
- [26] Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling tabular data using conditional GAN. *arXiv*. <https://arxiv.org/abs/1907.00503>
- [27] Jaiswal, S., & Gupta, P. (2023). GLSTM: A novel approach for prediction of real & synthetic PID diabetes data using GANs and LSTM classification model. *International Journal of Experimental Research and Review*, 30, 32–45. <https://doi.org/10.52756/ijerr.2023.v30.004>

- [28] Pawlak, Z., & Skowron, A. (2007). Rudiments of rough sets. *Information Sciences*, 177(1), 3–27. <https://doi.org/10.1016/j.ins.2006.06.006>
- [29] Pawlak, Z. (2002). Rough sets and intelligent data analysis. *Information Sciences*, 147(1–4), 1–12. [https://doi.org/10.1016/S0020-0255\(02\)00197-4](https://doi.org/10.1016/S0020-0255(02)00197-4)
- [30] Pawlak, Z., & Słowiński, R. (1994). Decision analysis using rough sets. *International Transactions in Operational Research*, 1(1), 107–114. [https://doi.org/10.1016/0969-6016\(94\)90050-7](https://doi.org/10.1016/0969-6016(94)90050-7)
- [31] Rosetta Toolkit. (2024). *Open source rough set data mining software*. <https://github.com/komorowskilab/R.ROSETTA>
- [32] Komorowski, J., Polkowski, L., & Skowron, A. (1999). ROSETTA: A rough set toolkit for analysis of data. In *Proceedings of the International Conference on Rough Sets and Current Trends in Computing (RSCTC'99)* (pp. 605–606).
- [33] Apache Kafka Documentation. (2023). *Kafka architecture overview*. <https://kafka.apache.org/documentation/>
- [34] Singh, K. (2023). *Effortlessly manage Apache Kafka with Docker Compose: A YAML-powered guide*. <https://collabnix.com/effortlessly-manage-apache-kafka-with-docker-compose-a-yaml-powered-guide/>
- [35] Apache ZooKeeper Documentation. (2023). *Introduction to ZooKeeper*. <https://zookeeper.apache.org/doc/current/zookeeperOver.html>
- [36] Confluent. (2024). *Apache Kafka documentation: Developer portal*. <https://developer.confluent.io>
- [37] Schimpf, C., & Castellani, B. (2020). Complex-it: A case-based modelling and scenario simulation platform for social inquiry. *Journal of Open Research Software*, 8(1), 25. <https://doi.org/10.5334/jors.298>
- [38] Zhang, H. (2004). The optimality of naive Bayes. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004)* (pp. 562–567). <https://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf>
- [39] Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3), 175–185. <https://www.jstor.org/stable/2685209>
- [40] Mehta, S., Shen, X., Gou, J., & Niu, D. (2018). A new nearest centroid neighbor classifier based on K local means using harmonic mean distance. *Information*, 9(9), 234. <https://doi.org/10.3390/info9090234>
- [41] Also useful for data reduction – DataScienceCentral.com. (2017, August 15). *Data Science Central*. <https://www.datasciencecentral.com/nice-generalization-of-the-k-nn-clustering-algorithm>
- [42] Şerbetçi, A. (2016). Sıralı lojistik regresyon analizi ile istatistik ve ekonometri derslerinde başarıyı etkileyen faktörlerin belirlenmesi: Atatürk Üniversitesi İktisadi ve İdari Bilimler Fakültesi öğrencileri üzerine bir uygulama. *Kahramanmaraş Sütçü İmam Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi*, 3(1), 89–110. <https://dergipark.org.tr/tr/pub/ksuiibf/issue/10266/125917>

- [43] Yu, W., Liu, T., Valdez, R., Gwinn, M., & Khoury, M. J. (2010). Application of support vector machine modeling for prediction of common diseases: The case of diabetes and pre-diabetes. *BMC Medical Informatics and Decision Making*, 10(1), 16. <https://doi.org/10.1186/1472-6947-10-16>
- [44] Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 119–139. https://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic_generalization.pdf
- [45] Zhou, Z.-H. (2012). *Ensemble methods: Foundations and algorithms*. CRC Press. <https://tjzhifei.github.io/links/EMFA.pdf>
- [46] Pintelas, P., & Livieris, I. E. (2020). Special issue on ensemble learning and applications. *Algorithms*, 13(6), 140. <https://doi.org/10.3390/a13060140>
- [47] TechTarget. (n.d.). SQL Server. *SearchDataManagement*. Eriřim: <https://www.techtarget.com/searchdatamanagement/definition/SQL-Server>
- [48] Rahman, M. G., & Davis, D. N. (2013). Addressing the class imbalance problem in medical datasets. *International Journal of Machine Learning and Computing*, 3(2), 224.
- [49] Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling tabular data using conditional GAN. *arXiv*. <https://arxiv.org/abs/1907.00503>
- [50] Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems* (Vol. 30). https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf

ÖZGEÇMİŞ

Ad-Soyad : Mehmet YÜKSEL

ÖĞRENİM DURUMU:

- Lisans** : 2024, Sakarya Üniversitesi, Bilgisayar ve Bilişim Bilimleri Fakültesi, Bilgisayar Mühendisliği Bölümü
- Yükseklisans** : Devam Ediyor, Sakarya Üniversitesi, Fen Bilimleri Enstitüsü, Bilişim Sistemleri Mühendisliği

MESLEKİ DENEYİM VE ÖDÜLLER:

- 2017 yılında Sa-tech Otomasyon şirketinde yaz stajı yaparak C# dili ile stok takip yazılımı geliştirdi.
- 2021 yılında aynı firmada haftada 2 gün olacak şekilde Bahar dönemi stajını tamamladı (30 iş günü).
- 2021 yılında Hacettepe Üniversitesi Başkent OSB Teknik Bilimler MYO'yu birincilikle bitirerek Başarı Belgesi aldı.
- 2021 yılında Hacettepe Üniversitesi Mezunlar Derneği (HUMED) tarafından Üstün Başarı Belgesi verildi.
- 2023 yılında 6th USSCIENCE Applied Sciences Congress'te sözlü bildiri sunumu gerçekleştirerek Yazar Katılım Sertifikası aldı.
- 2024 yılında Turkcell Global Bilgi'de UMDE (Uygulamalı Mühendislik Deneyimi Eğitimi) kapsamında 3 ay tam zamanlı çalıştı.
- 2024 yılında IMASCON 2024 Autumn kongresinde sözlü sunum gerçekleştirdi ve Yazar Katılım Sertifikası aldı.
- 2025 yılında IMASCON 2025 Spring kongresinde sözlü sunum gerçekleştirdi ve Yazar Katılım Sertifikası aldı.
- 2025 yılında aynı kongrede Oturum Başkanı olarak görev yaptı ve Oturum Başkanlığı Sertifikası aldı.

TEZDEN TÜRETİLEN ESERLER:

- Yüksel, M. ve Demirci, H. (2025, 16-17 Mayıs). Pima Indians Verisetinin Kaba Kümeleme Teorisi Kullanılarak İndirgenmesi ve Karar Kurallarının Çıkartılması. *14.Uluslararası Marmara Fen ve Sosyal Bilimler Kongresi*. Kocaeli, Turkey.

DİĞER ESERLER:

- Yüksel, M. ve Sevin, A. (2023, 15-17 Aralık). Form Tabanlı İnsan Kaynakları Yönetim Sistemi Tasarımı, *Uluslararası 6.Usbilim Uygulamalı Bilimler Kongresi*. Ankara, Turkey.
- Yüksel, M. ve Demirci, H. (2024, 29-30 Kasım). Akıllı Ev Sistemlerinde Kullanıcı Deneyimi ve Kullanılabilirlik: Arayüz Tasarımları ve Sesli Komutlar Üzerine Bir İnceleme. *13.Uluslararası Marmara Fen ve Sosyal Bilimler Kongresi*. Kocaeli, Turkey.
- Yüksel, M. ve Demirci, H. 2025. The Role of Blockchain Technology in Supply Chain and Logistics Sector: A Literature Review for the Period 2015-2024, *Düzce University Journal of Science and Technology*, 13(3), 1331-1345, <https://doi.org/10.29130/dubited.1658171>

