T.C.
İSTANBUL ÜNİVERSİTESİ-CERRAHPAŞA
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

DOKTORA TEZİ

MAKİNE ÖĞRENMESİ YÖNTEMLERİ ile YAZILIM HATA TAHMİNİ

Zeynep Behrin GÜVEN AYDIN

DANIŞMAN
Doç. Dr. Rüya ŞAMLI

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

İSTANBUL-2021

Bu çalışma, 5.03.2021 tarihinde aşağıdaki jüri tarafından Bilgisayar Mühendisliği Anabilim Dalı, Bilgisayar Mühendisliği Programında Doktora tezi olarak kabul edilmiştir.

Tez Jürisi

Doç. Dr. Rüya ŞAMLI(Danışman)
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Prof. Dr. Turgay Tugay BİLGİN
Bursa Teknik Üniversitesi
Mühendislik ve Doğa Bilimleri Fakültesi

Prof. Dr. Ahmet SERTBAŞ
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Prof. Dr. İlhami YAVUZ
Maltepe Üniversitesi
Mühendislik ve Doğa Bilimleri Fakültesi

Doç. Dr. Tarık KÜÇÜKDENİZ
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

20.04.2016 tarihli Resmi Gazete'de yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince; Bu Lisansüstü teze, İstanbul Üniversitesi-Cerrahpaşa'nın abonesi olduğu intihal yazılım programı kullanılarak Lisansüstü Eğitim Enstitüsü'nün belirlemiş olduğu ölçütlere uygun rapor alınmıştır.

Bu tez, TÜBİTAK 118E682 numaralı proje ile desteklenmiştir

ÖNSÖZ

Doktora eğitimime başladığım ilk günden bugüne hem akademik hayat, hem normal hayat ile ilgili kıymetli tecrübelerini dinlemekten her zaman zevk aldığım, bazen hocam, bazen ablam, bazen de arkadaşım olarak, kısacası bana her konuda rehber olan değerli tez danışmanım sayın Doç.Dr. Rüya ŞAMLI'ya, lisans öğrenimimden bu güne destek ve yardımlarını esirgemeyen hocam sayın Prof. Dr. Turgay Tugay BİLGİN'e, yine bu sürecin başından bugüne her konuda destek olan kıymetli hocam sayın Prof.Dr. İlhami YAVUZ'a, eğitim hayatım boyunca maddi manevi desteklerini esirgemeyen, her zaman yanımda olan en kıymetlilerim annem Füsun GÜVEN'e, babam Ferit GÜVEN'e, ablam Nazlı GÜVEN'e ve tabi yardımını ve sevgisini her daim yanımda hissettiğim sevgili eşim Erdoğan AYDIN'a en içten saygı, sevgi ve teşekkürlerimle…

Mart 2021 Zeynep Behrin GÜVEN AYDIN

İÇİNDEKİLER

Sayfa No

ÖNSÖZ ..... **iv**
İÇİNDEKİLER ..... **v**
ŞEKİL LİSTESİ ..... **viii**
TABLO LİSTESİ ..... **x**
SİMGE VE KISALTMA LİSTESİ ..... **xi**
ÖZET ..... **xiii**
SUMMARY ..... **xiv**
1. GİRİŞ **1**
1.1 TEZİN AMACI 2
1.2 TEZİN YAPILANDIRILMASI 3
2. GENEL KISIMLAR **4**
2.1. HATA KAVRAMI 4
2.2.YAZILIMDA HATA TAHMİNİ 5
2.3. YAZILIM METRİKLERİ 6
2.3.1.Yordamsal Kod Metrikleri 7
2.3.2.Nesneye Yönelik Kod Metrikleri 8
2.4. LİTERATÜR TARAMASI 9
3. MALZEME VE YÖNTEM **16**
3.1. NASA VERİ SETLERİ 16
3.2. DENEYSEL VERİ SETİ 18
3.3. DEĞERLENDİRME KRİTERLERİ 30
3.3.1. Doğruluk (Accuracy) 30
3.3.2. ROC (Reciever Operator Characteristics Curve–Alıcı Operatör Özellikleri Eğrisi) 30
3.3.3. AUC (Area Under the Curve - Eğri Altında Kalan Alan) 30
3.4. UYGULAMA PLATFORMLARI 31
3.4.1. WEKA Platformu 31
3.4.2. Google Colabarotary 31
3.5. YÖNTEMLER 32
3.5.1. BAYES 33

3.5.1.1. Bayes Ağı (Bayes Net) *33*
3.5.1.2. Naive Bayes *33*
3.5.1.3. Naive Bayes Çok Terimli Metin (Naive Bayes Multinomial Text) *33*
3.5.1.4. Naive Bayes Güncellenebilir (Naive Bayes Updateable) *33*
3.5.2. FONKSİYONLAR (FUNCTIONS) 33
3.5.2.1. Lojistik (Logistic) *33*
3.5.2.2. Çok Katmanlı Algılayıcı (Multilayer Perceptron) *34*
3.5.2.3. Stokastik Gradyan İniş (Stochastic Gradient Descent) *34*
3.5.2.4. Stokastik Gradyan İniş Metin (Stochastic Gradient Descent Text) *34*
3.5.2.5. Basit Lojistik (Simple Logistic) *35*
3.5.2.6. Sıralı Minimal Optimizasyon (Sequential Minimal Optimization) *35*
3.5.2.7. Oylanmış Algılayıcı (Voted Perceptron) *35*
3.5.3. LAZY 35
3.5.3.1. K-En Yakın Komşu Sınıflandırıcı (K-Nearest Neighbours Classifier) *35*
3.5.3.2. Kyıldız (Kstar –K)* *35*
3.5.3.3. Yerel Ağırlıklı Öğrenme (Locally Weighted Learning) *36*
3.5.4. META 36
3.5.4.1. İteratif Sınıflandırıcı Optimize Edici (Iterative Classifier Optimizer) *36*
3.5.4.2. Adaptif Yükseltme (AdaBoostM1) *36*
3.5.4.3. Öznitelik Seçici Sınıflandırıcı (Attribute Selected Classifier) *36*
3.5.4.4. Torbalama (Bagging) *36*
3.5.4.5. Regresyon Yoluyla Sınıflandırma (Classification via Regression) *36*
3.5.4.6. Maliyete Duyarlı Sınıflandırıcı (Cost Sensitive Classifier) *36*
3.5.4.7. Çapraz Doğrulama Parametre Seçimi (Cross Validation Parameter Selection) *37*
3.5.4.8. Filtrelenmiş Sınıflandırıcı (Filtered Classifier) *37*
3.5.4.9. Logi Boost *37*
3.5.4.10. Çok Sınıflı Sınıflandırıcı (Multi Class Classifier) *37*
3.5.4.11. Çok Sınıflı Sınıflandırıcı Güncellenebilir (Multi Class Classifier Updateable) *37*
3.5.4.12. Çoklu Şema (Multi Scheme) *37*
3.5.4.13. Rastgele Komite (Random Committee) *38*
3.5.4.14. Randomize Edilebilir Filtreli Sınıflandırıcı (Randomizable Filtered Classifier) *38*
3.5.4.15. Rastgele Alt Boşluk (Random Sub Space) *38*

3.5.4.16. İstifleme (Stacking) *38*
3.5.4.17. Oylama (Vote) *38*
3.5.4.18. Ağırlıklı Örnek İşleyici Sarmalayıcı (Weighted Instances Handler Wrappler) *38*
3.5.5. MISC – Giriş Eşlemeli Sınıflandırıcı (Input Mapped Classifier) 39
3.5.6. KURALLAR (RULES) 39
3.5.6.1. Karar Ağacı (Desicion Table) *39*
3.5.6.2. Tekrarlanan Artımlı Budama (Repeated Incremental Pruning) *39*
3.5.6.3. OneR *39*
3.5.6.4. Part *39*
3.5.6.5. ZeroR *40*
3.5.7. AĞAÇ (TREE) 40
3.5.7.1. Karar Kütüğü (Desicion Stump) *40*
3.5.7.2. Hoefding Ağacı (Hoefding Tree) *40*
3.5.7.3. J48 ..... *40*
3.5.7.4. Lojistik Model Ağacı (Logistic Model Tree) *40*
3.5.7.5. Rastgele Orman (Random Forest) *41*
3.5.7.6. Rastgele Ağaç (Random Tree) *41*
3.5.7.7. Rep Ağacı (Rep Tree) *41*
4. BULGULAR **42**
4.1. WEKA SİMÜLASYON SONUÇLARI 42
4.1.1 NASA Veri Setleri Simülasyon Sonuçları 42
4.1.2. Deneysel Veri Seti Simülasyon Sonuçları 62
4.2. PYTHON SİMÜLASYON SONUÇLARI 69
4.2.1 NASA Veri Setleri Simülasyon Sonuçları 69
4.2.2. Deneysel Veri Seti Simülasyon Sonuçları 79
4.3 WEKA ÖZNİTELİK SEÇİM METOTLARI İLE PERFORMANS DEĞERLENDİRMESİ 83
4.3.1. Cfs Subset Eval Öznitelik Seçim Algoritması 83
4.3.2. Temel Bileşen Öznitelik Seçim Algoritması 84
4.3.3. NASA Veri Setleri ve Deneysel Veri Seti ile Öznitelik Seçim Algoritması 84
5. TARTIŞMA VE SONUÇ **92**
KAYNAKLAR **95**
ÖZGEÇMİŞ **100**

ŞEKİL LİSTESİ

Sayfa No

ŞEKİL 3.1: 1-24 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERATÖR SAYILARI 20
ŞEKİL 3.2: 25-47 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERATÖR SAYILARI 20
ŞEKİL 3.3: 49-71 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERATÖR SAYILARI 21
ŞEKİL 3.4: 72-94 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERATÖR SAYILARI 21
ŞEKİL 3.5: 95-102 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERATÖR SAYILARI 21
ŞEKİL 3.6: 1-19 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI 22
ŞEKİL 3.7: 1-19 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 2 22
ŞEKİL 3.8: 1-19 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 3 22
ŞEKİL 3.10: 19-37 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI 23
ŞEKİL 3.11: 19-37 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 2 23
ŞEKİL 3.12: 19-37 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 3 24
ŞEKİL 3.13: 19-37 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 4 24
ŞEKİL 3.14: 37-55 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI 24
ŞEKİL 3.15: 37-55 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 2 25
ŞEKİL 3.16: 37-55 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 3 25
ŞEKİL 3.17: 37-55 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 4 25
ŞEKİL 3.18: 55-73 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI 26
ŞEKİL 3.19: 55-73 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 2 26
ŞEKİL 3.20: 55-73 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 3 26
ŞEKİL 3.21: 55-73 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 4 27
ŞEKİL 3.22: 73-91 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI 27
ŞEKİL 3.23: 73-91 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 2 27
ŞEKİL 3.24: 73-91 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 3 28
ŞEKİL 3.25: 73-91 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 4 28
ŞEKİL 3.26: 91-102 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI 28
ŞEKİL 3.27: 91-102 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 2 29
ŞEKİL 3.28: 91-102 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 3 29
ŞEKİL 3.29: 91-102 NUMARALI YAZILIM PROJE KODLARINDAKİ OPERAND SAYILARI - 4 29
ŞEKİL 4.1: JM1 VERİ SETİ SİMÜLASYON SONUÇLARI 45
ŞEKİL 4.2: JM1 VERİ SETİ SİMÜLASYON SONUÇLARI-2 46
ŞEKİL 4.3: JM1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ 47
ŞEKİL 4.4: JM1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ-2 48
ŞEKİL 4.5: KC1 VERİ SETİ SİMÜLASYON SONUÇLARI 49
ŞEKİL 4.6: KC1 VERİ SETİ SİMÜLASYON SONUÇLARI-2 50

ŞEKİL 4.7: KC1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ 51
ŞEKİL 4.8: KC1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ-2 52
ŞEKİL 4.9: CM1 VERİ SETİ SİMÜLASYON SONUÇLARI 53
ŞEKİL 4.10: CM1 VERİ SETİ SİMÜLASYON SONUÇLARI-2 54
ŞEKİL 4.11: CM1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ 55
ŞEKİL 4.12: CM1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ-2 56
ŞEKİL 4.13: PC1 VERİ SETİ SİMÜLASYON SONUÇLARI 57
ŞEKİL 4.14: PC1 VERİ SETİ SİMÜLASYON SONUÇLARI-2 58
ŞEKİL 4.15: PC1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ 59
ŞEKİL 4.16: PC1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ-2 60
ŞEKİL 4.17: DENEYSEL VERİ SETİ SİMÜLASYON SONUÇLARI 65
ŞEKİL 4.18: DENEYSEL VERİ SETİ SİMÜLASYON SONUÇLARI-2 66
ŞEKİL 4.19: DENEYSEL VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ 67
ŞEKİL 4.20: DENEYSEL VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ-2 68
ŞEKİL 4.21: JM1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ 70
ŞEKİL 4.22: JM1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ-2 71
ŞEKİL 4.23: KC1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ 72
ŞEKİL 4.24: KC1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ-2 73
ŞEKİL 4.25: CM1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ 74
ŞEKİL 4.26: CM1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ-2 75
ŞEKİL 4.27: PC1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ 77
ŞEKİL 4.28: PC1 VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ-2 78
ŞEKİL 4.29: DENEYSEL VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ 80
ŞEKİL 4.30: DENEYSEL VERİ SETİ SİMÜLASYON SONUÇLARI ROC EĞRİSİ-2 81

TABLO LİSTESİ

Sayfa No

TABLO 2.1: NASA'YA AİT VERİ SETİ ÖRNEĞİ. 5
TABLO 2.2: YAZILIM HATA TAHMİNİ LİTERATÜR ÇALIŞMALARI 14
TABLO 3.1: VERİ SETLERİNİN ÖZELLİKLERİ 16
TABLO 3.2: METRİK ÖZELLİKLERİ 17
TABLO 3.3: METRİK DETAYLARI 18
TABLO 3.4:DENEYSEL VERİ SETİ ÖZELLİKLERİ 19
TABLO 3.5:DENEYSEL VERİ SETİ METRİK ÖZELLİKLERİ 19
TABLO 4.1: WEKA NASA VERİ SETİ DOĞRULUK ORANLARI 43
TABLO 4.2: WEKA DENEYSEL VERİ SETİ DOĞRULUK ORANLARI 63
TABLO 4.3: PYTHON NASA VERİ SETİ DOĞRULUK ORANLARI 69
TABLO 4.4: PYTHON DENEYSEL VERİ SETİ DOĞRULUK ORANLARI 79
TABLO 4.6: JM1 VERİ SETİ ÖZNİTELİK SEÇİMİ İLE DOĞRULUK ORANI DEĞİŞİMİ 87
TABLO 4.7: KC1 VERİ SETİ ÖZNİTELİK SEÇİMİ İLE DOĞRULUK ORANI DEĞİŞİMİ 88
TABLO 4.8: CM1 VERİ SETİ ÖZNİTELİK SEÇİMİ İLE DOĞRULUK ORANI DEĞİŞİMİ 89
TABLO 4.9: PC1 VERİ SETİ ÖZNİTELİK SEÇİMİ İLE DOĞRULUK ORANI DEĞİŞİMİ 90
TABLO 4.10: DENEYSEL VERİ SETİ ÖZNİTELİK SEÇİMİ İLE DOĞRULUK ORANI DEĞİŞİMİ 91

SİMGE VE KISALTMA LİSTESİ

Kısaltmalar	Açıklama
AUC	:Area Under Curve – Eğri Altında Kalan Alan
ARFF	:Attribute Relationship File Format – Öznitelik İlişkisi Dosyası Biçimi
CBO	:Coupling Between Object Classes – Nesne Sınıfları Arasındaki Bağımlılık
CART	:Classification and Regression Tree – Sınıflandırma ve Regresyon Ağacı
Cfs	:Corelation-based Featue Selection- Korelasyon Tabanlı Özellik Seçici
CK	:Chidamber-Kemerer
DIT	:Depth of Inheritance Tree – Kalıtım Ağacının Derinliği
EM	:Expactation Maximation – Maksimum Beklenti
GRNN	:General Regression Neural Network – Genel Regresyon Sinir Ağları
GPU	:Graphics Procsessing Unit- Grafik İşlemci Birimi
Ibk	:Instance Based Learner – Örneğe Dayalı Öğrenme
IEEE	:Institute of Electrical and Electronics Engineers – Elektrik ve Elektronik Mühendisleri Enstitüsü
JDBC	:Java Derby Client – Java Derbi İstemcisi
LOC	:Lines of Code – Kod Satır Sayısı
LCOM	:Lack of Cohesion in Methods – Metotlardaki Uyum Eksikliği
LSR	:Least Square Residual – En Küçük Kare Kalıntı
LWL	:Locally Weighted Learning – Yerel Ağırlıklı Öğrenme
MBR	:Memory Based Reasoning – Belleğe Dayalı Akıl Yürütme
MLP	:Multilayer Perceptron – Çok Katmanlı Algılayıcılar
MOOD	:Metrics for Object Oriented Design – Nesneye Yönelik Tasarım için Metrikler
NASA	:National Aeronautics and Space Administration – Ulusal Havacılık ve Uzay Dairesi
NOC	:Number of Children – Alt Sınıf Sayısı
NB	:Naive Bayes
PD	:Probabilty of Detection – Tespit Olasılığı
PF	:Probabilty of False Alarm – Yanlış Alarm Olasılığı

PROMISE :Predictor Models in Software Engineering Models – Yazılım Mühendisliğinde Tahmin Modelleri

RvC :Regression via Classification – Sınıflandırma Yoluyla Regresyon

RFC :Response for a Class – Bir Sınıf için Çağrı Sayısı

ROC :Reciever Operator Characteristics Curve – Alıcı Operatör Özellikleri Eğrisi

SQL :Structured Query Language – Yapılandırılmış Sorgu Dili

SRS :Software Requirement Specification – Yazılım İsterler Belirtimi

SVM :Support Vector Machine – Destek Vektör Makinesi

WEKA :Waikato Environment for Knowledge Analysis – Bilgi Analizi için Waikato Ortamı

WMC :Weighted Method per Class – Sınıf İçindeki Metot Sayısı

QMOOD :Quality Model for Object Oriented Design – Nesneye Yönelik Tasarım İçin Kalite Modeli

YSA :Yapay Sinir Ağları

ÖZET

DOKTORA TEZİ

MAKİNE ÖĞRENMESİ YÖNTEMLERİ İLE YAZILIM HATA TAHMİNİ

Zeynep Behrin GÜVEN AYDIN

İstanbul Üniversitesi-Cerrahpaşa

Lisansüstü Eğitim Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

Danışman : Doç. Dr. Rüya ŞAMLI

Yazılım hatası, bir yazılım ürününde bir yazılım gereksinimini veya son kullanıcının ihtiyacını karşılamayan bir durum, başka bir ifadeyle hata, bir programın kusurlu çalışmasına ya da istenmeyen sonuçlar üretmesine neden olan kodlama veya mantıktaki bir kusurdur. Yazılım hataları, yazılım geliştirme aşamaları sırasında uygulanan testler sayesinde daha erken safhada tespit edilebilir ve düzeltilebilir. Yazılım hata tahmini konularında yapılan çalışmalarda daha çok makine öğrenmesi algoritmalarının ve istatistiksel yöntemlerin kullanıldığı görülmektedir. Bu tez çalışmasında, literatürde raporlanmış makine öğrenmesi algoritmaları ile kıyaslama gerçekleştirilmesi amacıyla NASA (National Aeronautics and Space Administration-Ulusal Havacılık ve Uzay Dairesi) Metrik Veri Programı tarafından oluşturulmuş, PROMISE (Predictor Models in Software Engineering–Yazılım Mühendisliğinde Tahmin Modelleri) deposunda yer alan NASA'ya ait projelerin 4 adet veri seti (JM1, KC1, CM1, PC1) üzerinde WEKA veri madenciliği aracında ve Google Colab ortamında Pyhton dili kullanılarak çeşitli makine öğrenmesi algoritmaları ve istatistiksel yöntemler gerçeklenmiş, performansları her veri seti için doğruluk (accuracy) oranları ve ROC (Reciever Operator Characteristics Curve – Alıcı Operatör Özellikleri Eğrisi) eğrisi altında kalan AUC (Area Under Curve- Eğri Altında Kalan Alan) değerleri ile karşılaştırılmıştır. Ayrıca çeşitli program kodlarından oluşan bir veri setinin metrikleri belirlenmiş, yeni oluşturulan bu veri seti üzerinde yine çeşitli veri madenciliği algoritmaları ile hata tahminlemesi gerçekleştirilmeye çalışılmıştır.

Mart 2021, 115. sayfa.

Anahtar kelimeler: Yazılım hata tahmini, makine öğrenmesi algoritmaları, WEKA, Python

SUMMARY

Ph.D. THESIS

SOFTWARE DEFECT PREDICTION WITH MACHINE LEARNING METHODS

Zeynep Behrin GÜVEN AYDIN

Istanbul University-Cerrahpasa

Institute of Graduate Studies

Department of Computer Engineering

Supervisor : Assoc. Prof. Dr. Rüya ŞAMLI

A software error is a situation in a software product that does not meet a software requirement or the need of the end user, in other words, an error is a flaw in coding or logic that causes a program to malfunction or produce unwanted results. Software errors can be detected and corrected at an earlier stage thanks to the tests applied during the software development stages. In studies on software error prediction, it is seen that machine learning algorithms and statistical methods are mostly used.

In this thesis study, 4 data sets of NASA projects (JM1, KC1, CM1, PC1) created by NASA (National Aeronautics and Space Administration-Metric Program) in the repository of PROMISE (Predictor Models in Software Engineering)) in order to compare machine learning algorithms reported in the literatüre was preferred. On these data sets, various machine learning algorithms and statistical methods were implemented using the WEKA data mining tool and Python language in Google Colab environment, their performances were compared with the accuracy rates and the AUC(Area Under Curve) values under the ROC (Reciever Operator Characteristics Curve) curve for each data set. In addition, the metrics of a data set consisting of various program codes were determined, and defect prediction was tried to be performed on this newly created data set with various data mining algorithms.

March 2021, 115 pages.

Keywords: Software defect prediction, machine learning algorithms, WEKA, Python

1. GİRİŞ

Yazılım, herhangi bir boyuttaki herhangi bir tür donanımda çalışan bilgisayar programını ve basılı veya elektronik ortamdaki her tür dokümanı içeren üründür. Mantık, veri, belge, insan ve program bileşenlerinin belirli bir üretim amacına yönelik olarak bir araya getirilmesi ve yönetilebilmesi için kullanılabilecek ve üretilen, yöntem, araç, bilgi ve belgelerin tümünü içerir.

Yazılım hem bir üründür hem de başka ürünler elde etmeye yarayan bir araçtır. Yazılım fiziksel bir ürün olmadığı için aşınmaz, ancak zamanla yetersizleşebilir. Yetersizlik sonucunda yazılımlar üzerinde yapılan değişiklikler yazılımlarda hataya sebep olabilir. Yazılım sistemlerinde kusurların çok büyük bölümü, az sayıda modül üzerinde gerçekleşir. Yazılım sistemleri mantıksal kusurlar nedeniyle bazı durumlarda başarısız olmaktadırlar. Bu başarısızlıkların sebebi incelendiğinde öncelikle karşılaşılan problemin, yazılım geliştirme sürecinde gerçekleştirilen test eylemleri ile yazılım içerisinde olabilecek olan hataların daha erken safhada bulunabileceği ve düzeltilebileceği görülmektedir. Bu sayede daha yüksek kalitede, daha az hatalı yazılımlar üretilerek yazılım kaynaklı problemlerin ortaya çıkması sağlanabilir. Çalışmalar incelendiğinde, kusur, hata ve eksiklik terimlerinin aslında aynı anlamı ifade eden terimler olarak kullanıldığı görülmektedir. Literatürde yapılan çalışmalarda, farklı terimler kullanılsa da elde edilecek modelde amaç, normalde bir bağımlı (dependent) değişkenin, n tane bağımsız (independent) değişkenin bulunduğu modelin kurulmasıdır. Hata tahmini çalışmalarında bağımlı değişken, modülün hataya yatkınlığını gösterirken, bağımsız değişken ise ürün veya süreç metrikleri olabilmektedir. Bu konuda yapılan çalışmaların büyük kısmında, ürün metrikleri kullanılmaktadır. Ürün metrikleri, sınıf ve metot seviyesinde metrikler olmak üzere iki çeşittir. Metot seviyesindeki metrikler, yordamsal veya nesneye yönelik programlamada da tahmin amaçlı kullanılabilirler. En sık kullanılan metot seviyesi metrikleri, Halstead ve McCabe metrikleridir. Sınıf seviyesinde metrikler ise, her çeşit nesneye yönelik metrik olabilir. Chidamber-Kemerer (CK) metrikleri, en sık kullanılan sınıf seviyesi metriklerindendir.

Bir yazılımın test aşamasında veya son kullanıcıda hataya sebep olan modüller, daha çok modüllerin çok ufak bir kısmını oluşturmaktadır. Bu tür modüllerden oluşan hatalı veri kümeleri “dengeli olmayan (imbalanced)” veri setleri olarak adlandırılır. Bu tür veri setleri

üzerinde uygulanan algoritmaların performansını ölçmek, diğer veri setlerinde gerçeklenen algoritmaların performansları ile karşılaştırmak, ya da literatürde yapılmış benzer çalışmaların performansları ile karşılaştırmak için doğruluk (accuracy) değerlendirme kriteri yeterli değildir. Dengeli olmayan veri setleri üzerinde, makine öğrenmesi algoritmaları ile ilgili çalışma yapanlar tarafından performans değerlendirmesinde doğruluk kriteri yeterli olmadığından G-ortalama1, G-ortalama2 ve F-ölçüm değerlerinin tamamına bakarak, veya sadece F-ölçüm değerine bakarak ya da ROC (Receiver Operating Characteristic- Alıcı Operatör Özellikleri Eğrisi) eğrisi altında kalan alandan hesaplanan AUC (Area Under Curve – Eğri Altında Kalan Alan) değerine bakarak yorum yapılır [1].

1.1 TEZİN AMACI

Bu tez çalışmasında:

1. Makine öğrenmesi algoritmalarının ve istatistiksel yöntemlerin kullanılması, bu yöntemler ile veri setlerinde yazılım metriklerinin kullanarak algoritmaların başarım analizinin yapılması,
2. Aynı veri setleri üzerinde farklı algoritma ve yöntemlerin performans analizlerinin yapılması ve sonuçların detaylı yorumlanması,
3. Farklı platformlarda aynı veri setleri üzerinde, makine öğrenmesi algoritmalarının performans analizlerinin yapılması,
4. Yazılım hata tahmini konusunda, literatürde yayınlanmış birçok yayında yer alan ve yazılım hata veri setlerinde hangi metriğin ne kadar önemli olduğunu ortaya koyan özellik seçimi ile hangi makine öğrenmesi algoritmalarında hangi özniteliklerin daha önemli olduğu ve hangi öznitelik seçimlerinde daha başarılı sonuçlar elde edildiğinin, doğruluk oranlarının arttırdığının tespit edilmesi,
5. Veri setlerinde yer alan hangi metrikler bir arada yer aldığında veya hangi metriklerin önemli, hangi metriklerin seçilen algoritma için önemsiz olduğunun bunun sonucu olarak da hangi algoritmaların başarı oranların daha yüksek olduğu belirlenmesi,

amaçlanmaktadır.

1.2 TEZİN YAPILANDIRILMASI

Tez çalışmasının kalan kısmı şu şekilde düzenlenmiştir:

- 2. Bölüm olan Genel Kısımlar Bölümü'nde hata kavramı, yazılımda hata tahmini, metrikler ve tez konusu kapsamında literatürde bugüne kadar yapılmış çalışmalar alt başlıklar halinde sunulmuştur.
- 3. Bölüm olan Malzeme ve Yöntem Bölümü'nde; tez çalışması kapsamında kullanılan veri setleri, uygulama platformları, makine öğrenmesi algoritmaları ve istatiksel yöntemler sunulmuştur.
- 4. Bölüm olan Bulgular Bölümü'nde yapılan çalışmalar üç kısımda incelenebilir. İlk kısımda, literatürdeki diğer çalışmalarla karşılaştırma yapılabilmesi amacıyla NASA Metrik Veri Programı kapsamında oluşturulmuş ve PROMISE havuzunda yer alan NASA'ya ait projelerin veri setleri üzerinde gerçekleştirilen simülasyon sonuçlarına yer verilmiştir. Bu veri setleri üzerinde WEKA veri madenciliği aracı ile çeşitli makine öğrenmesi algoritmaları ve istatistiksel yöntemler kullanılarak bu algoritmaların doğruluk oranları hesaplanmış ve bu değerlerden elde edilen ROC (Reciever Operator Characteristics Curve-Alıcı Operatör Özellikleri Eğrisi) eğrileri çizdirilmiştir. İkinci kısımda ise aynı veri setleri ve ek olarak oluşturulan deneysel veri seti üzerinde farklı platformlardan elde edilebilecek performans sonuçlarını karşılaştırmak amacıyla Google Colab ortamında Python programlama dilinde geliştirilen 10 makine öğrenmesi algoritmasının doğruluk değerleri hesaplanmış ve ROC eğrileri çizdirilmiştir. Üçüncü kısmında, aynı veri setleri ve ek olarak oluşturulan deneysel veri seti üzerinde öznitelik seçim metotları ile algoritmaların performansları hesaplanıp karşılaştırılmış, hangi özniteliklerin etkili hangilerinin biraz daha etkisiz olduğu değerlendirilmiştir.
- 5. Bölüm olan Tartışma ve Sonuç Bölümü'nde; her bir veri setinde gerçeklenen makine öğrenmesi algoritmalarının performans sonuçlarından, aynı veri setleri üzerinde farklı platformların performans karşılaştırılmasından bahsedilmiştir.

2. GENEL KISIMLAR

Bu bölümde, hata kavramı, yazılımda hata tahmini ve metrikler açıklanmış ve konu ile ilgili ayrıntılı bir literatür taraması sunulmuştur.

2.1. HATA KAVRAMI

Hata, kelime anlamı olarak yanlış, istemeyerek veya bilmeyerek yapılan yanlış, kusur, yanılma yanılgı, olarak ifade edilmektedir. Bilimde ve matematikte ise doğru bir değerden sapmayı belirten terim olarak ifade edilir. Literatür çalışmaları incelendiğinde ise hata kelimesinin yerine eksiklik (defect) ve kusur terimlerinin de birbirlerinin yerine kullanılabildiği görülmektedir. Eksiklik kelimesi; kusur, hata (error), arıza (failure) kelimelerini kapsayan daha genel bir ifade olarak literatürde kullanılmaktadır.

IEEE (Institute of Electrical and Electronics Engineers – Elektrik ve Elektronik Mühendisleri Enstitüsü), yazılım mühendisliği için hata (error), kusur (fault) ve arıza (failure) terimlerini şu şekilde tanımlamaktadır.

- *Hata:* "Hata, ürüne belirli kusurlar katan doğru olmayan ya da eksik insan eylemidir". Böylece, programcıların kod uygulama aşamasında, dikkatinden kaçırdığı noktalar hata olarak örneklenebilir.
- *Arıza:* "Arıza, program davranışı kullanıcı beklentilerinden farklı olduğu durumda ortaya çıkar." Yazılım İsterler Belirtimi (Software Requirement Specification – SRS) belgesinde belirtilen isterlerden farklı olduğu durumda, arıza durumunun ortaya çıktığı ifade edilebilir.
- *Kusur:* "Kusur, bir yazılım programı içinde arızaya götüren temel nedendir." Programcı tarafından kaynaklanan hata nedeniyle, program gitmemesi gereken bir duruma (state) gidebilir. Bunun sonucunda, ortaya çıkan durum yazılımın arıza ile sonlanmasına neden olabilir. Fakat her durumda arıza meydana gelmeyebilir.

Eksiklik (defect) kelimesi, IEEE tarafından tanımlamış olan ve üç terimin yerine de kullanılabilen daha genel bir anlam içeren bir sözcüktür [1].

2.2.YAZILIMDA HATA TAHMİNİ

Hata tahmini ile ilgili literatürde ele alınan çalışmalarda, hata, kusur ve eksiklik gibi farklı kelimeler kullanılsa da sonuç olarak elde edilmek istenilen modelde, bir adet bağımlı değişken ve *n* adet bağımsız değişkenin yer alması beklenmektedir. Burada kullanılan bir adet bağımlı değişken, modülün hata eğilimli olup olmadığını ifade etmektedir. Buna bir örnek olarak, NASA metrik veri programı tarafından oluşturulmuş PROMISE deposunda yer alan NASA'ya ait kusurlu ve kusursuz örnek bir veri seti için 21 bağımsız ve 1 bağımlı değişkenden oluşan 3 adet modüle ilişkin veriler Tablo 2.1'de sunulmaktadır:

Tablo 2.1: NASA'ya Ait Veri Seti Örneği.

11,3,1,1,49,215.22,0.07,14.67,14.67,3156.61,0.07,175.37,0,0,1,0,12,9,27,22,5, kusursuz
4,1,1,1,13,39,0.33,3,13,117,0.01,6.5,0,0,0,0,4,4,7,6,1, kusursuz
3,1,1,1,7,19.65,0.4,2.5,7.86,49.13,0.01,2.73,0,0,0,0,5,2,5,2,1, kusurlu

Diğer bağımsız değişkenler ise ürün veya süreç metrikleri olabilmektedir. Literatürde yapılan biçok çalışmada bu bağımsız değişkenlerin genellikle ürün metrikleri olduğu görülmüştür. Ürün metrikleri, sınıf ve metot düzeyindeki metrikler olmak üzere iki grupta tanımlanabilir. Sınıf seviyesinde olan metrikler, her türlü nesneye yönelik metrik olduğundan, bu sınıf seviyesinde olan metrikler, sadece nesneye yönelik programlama ile geliştirilen yazılımlar için kullanılabileceği aşikardır. Hata eğilimli modüllerin kestirimi için Chidamber-Kemerer sınıf seviyesindeki metrikler en çok kullanılanlardandır [2]. Metot seviyesinde olan metrikler; nesneye yönelik veya yordamsal (procedural) programlama paradigmasının kullanıldığı iki durumda tahmin ve kestirim amaçlı kullanılır. Bunun nedeni, bu iki paradigma içerisinde bu metotların yer alması gösterilebilir. Literatürde, McCabe ve Halstead metrikleri, metot seviyesindeki metriklerden en çok kullanılanlar olarak örnek verilebilir.

Hata veya kusur verileri, kullanıldığı modüllerde, sahada veya sistem testlerinde hataya sebep olup olmadığını ifade eden 0 veya 1 ile gösterilen verilerdir. Fakat, yazılım metrikleri ise sınıf, metot veya bileşen seviyesinde olan metriklerdir. Hata tahmini için yapılan çalışmalarda genellikle makine öğrenmesi algoritmaları ve istatistiksel yöntemler kullanılmıştır. Makine öğrenme algoritmaları aracılığıyla, metrik ve hata verilerinden ilk aşamada öğrenme süreci gerçekleştirilerek ve böylece yeni metriklerin oluştuğu durumda modüllerin kusurlu olup olmadığı hakkında tahmin yapılacaktır [1].

Yazılım mühendisliğinde en fazla bilinen “çevrimsel karmaşıklık” (cyclomatic complexity) metriği metot seviyesinde olan metriklere örnek olarak gösterilebilir. Sınıf seviyesindeki metriklere ise “Kalıtım ağacının boyutu” (depth of inheritance tree), “çocuklarının sayısı” (number of children) “sınıf içerisindeki metot sayısı” gibi metrikler örnek gösterilebilir.

Test aşamasında veya müşteride hataya yol açan modüller, genellikle bütün modüllerin çok küçük bir bölümüne karşı gelmektedir. Bundan dolayı; bu tür hata veri setlerine “dengeli olmayan” (imbalanced) veri kümeleri denilebilir. Dengeli olmayan veri setleri üzerinde kurulan modellerin veya çalıştırılan algoritmaların performansını karşılaştırmak için doğruluk (accuracy) değerlendirme kriterini kullanmak mümkün değildir. Makine öğrenmesi alanındaki araştırmacılar, bu tür veri kümeleri için farklı değerlendirme kriterleri sunmuşlardır. Bu kriterlere örnek verilecek olursa; ROC eğrisi alanından, hata matrisinin (confusion matrix) sadece diyagonaldeki değerlerini dikkate alınması veya sadece F-ölçüm değerini kullanarak ya da G-ortalama1, G-ortalama2 ve F-ölçüm değerlerinin hepsini bütünüyle kullanarak değerlendirme yapılması bu kriterlere örnek gösterilebilir [1].

Bu tez çalışmasında, literatürdeki diğer çalışmalarla karşılaştırma yapılabilmesi amacıyla NASA Metrik Veri Programı kapsamında oluşturulmuş ve PROMISE havuzunda yer alan NASA’ya ait projelerin veri setleri ve metrikler ile oluşturulan deneysel veri seti tercih edilmiştir.

2.3. YAZILIM METRİKLERİ

Bu tezde gerçekleştirilen uygulamalara ilişkin kullanılan yazılım metrik bilgileri, yordamsal kod metrikleri ve nesneye yönelik kod metrikleri başlıkları altında kısaca açıklanmıştır.

Yazılım metrikleri, kullanılan yazılımlardan farklı ölçümler gerçekleştirilerek sayısal değerler elde edilebilmektedir. Fakat elde edilen bu sayısal değerleri anlamlı kılan, bilgiye dönüşmüş halleri olan yazılım metrikleridir.

Kod Metrikleri, doğrudan kod üzerinden yapılan ölçümlere ilişkin metrikler olup aşağıdaki gibi açıklanan yordamsal kod metrikleri ve nesneye yönelik kod metrikleri olarak ifade edilebilir:

- *Yordamsal Kod Metrikleri:* Yordamsal Programlama yaklaşımı kullanılarak gerçekleştirilen yazılımların ölçülmesi için kullanılan metriklerdir. McCabe, Kod satır sayısı ve Halstead metrikleri bu alanda kullanılan metrikler olarak ifade edilebilir.

- *Nesneye Yönelik Kod Metrikleri:* Nesneye Yönelik Programlama yaklaşımı kullanılarak gerçekleştirilen yazılımların ölçülmesi için kullanılan metriklerdir. QMOOD (Quality Model for Object Oriented Design – Nesneye Yönelik Tasarım İçin Kalite Modeli), Chidamber-Kemerer metrik kümesi ve MOOD (Metrics for Object Oriented Design – Nesneye Yönelik Tasarım için Metrikler) metrik kümeleri bu kod metriklerine örnek olarak verilebilir [1].

2.3.1.Yordamsal Kod Metrikleri

NATO'nun katkılarıyla 1968 yılında düzenlenen ilk Yazılım Mühendisliği konferansında, Yazılım Krizi'nin ortaya çıktığı ve yazılım sistemlerinin gittikçe daha karmaşık hale geldiği ifade edilmiştir. Programlama alanındaki gelişmelerin ilerlemesiyle, daha yüksek soyutlama düzeyinde yazılım geliştirilmesini sağlamıştır. Bu değişimi; Assembly programlama dili, makine kodunun kullanılması, 3. nesil olarak ifade edilebilen C gibi yordamsal (procedural) diller, SQL (Structured Query Language – Yapılandırılmış Sorgu Dili) gibi 4. nesil diller, C++ gibi nesneye yönelik programlama dilleri, C gibi yordamsal programlama dillerinde geliştirilmiş kodlar için kullanılabilen metrikler açıklanmaktadır. Böylece, benzer şekilde nesneye yönelik programlamada da metotlar yer aldığı için buradaki metot seviyesindeki metrikleri o yaklaşımlarda kullanmaya imkân sağlamıştır. En çok bilinen ve kullanılan yordamsal kod metrikleri, McCabe ve Halstead metrikleridir.

McCabe Metrikleri: 1976 yılında McCabe, çevrimsel karmaşıklık (cyclomatic complexity) adını verdiği ve yazılımın kontrol akışını ölçen bir metod geliştirmiştir. Bu metod, metriklerin karmaşıklığını ölçerek bakım yapılabilirliği ve test süreci zor olan modülleri belirlemeyi sağlamaktadır. McCabe metrikleri içerisinde, en fazla popülerlik kazanan metrik çevrimsel karmaşıklık olup daha birçok McCabe metriği mevcuttur [1].

NASA'ya ait açık veri setlerinde, dört tane McCabe metriği kullanıldığından bu tez kapsamında bu metriklerden yararlanılmıştır. NASA veri setlerinde bu dört metrikten üç tanesi kullanılmış

olup bu metrikler; McCabe çevrimsel karmaşıklık, McCabe tasarım karmaşıklığı ve McCabe temel (essential) karmaşıklıktır.

Halstead Kod Metrikleri: Yazılım ölçümü konusunda Maurice Halstead, Thomas McCabe ile birlikte öncü isimler olarak verilebilir. Halstead, kendi yaklaşımına göre yazılım bilimi metriklerini (software science metrics), temel ve türetilmiş metrikler olarak iki temel gruba ayırmıştır [3].

Bu tez çalışmasında kullanılan, PROMISE havuzunda yer alan NASA açık veri setlerinde, 21 metrik özellik (feature) olarak yer almaktadır ve bu metrikler, hata tahmin modellerinin bağımsız değişkenlerini oluşturur. Yazılım modülünün hatalı olup olmadığını gösteren özellik ise en son sütunda yer almaktadır. 21 özelliğin 5 tanesi kod uzunluğu ile ilgili olup 4 tanesi primitif Halstead metriği, 8 tanesi türetilmiş Halstead metriği, 1 tanesi dallanma sayısını gösteren metrik (branch count), 3 tanesi McCabe metriğidir.

2.3.2.Nesneye Yönelik Kod Metrikleri

1990 yıllarında nesneye yönelik sistemler için birçok metrik önerilmiş olmasına rağmen bunlardan en çok kullanılanları Chidamber-Kemerer metrik grubudur. Birçok araç geliştiricisi ve araştırmacı bu metrikleri kullanmıştır. Bu tez kapsamında, Chidamber-Kemerer metrik kümesinden yararlanılan KC1 veri setinde yer alan metrikler aşağıda açıklanmaktadır:

- *WMC (Weighted Method per Class-Sınıfın Ağırlıklı Metot Sayısı):* WMC, bir sınıfta yer alan metot sayısıdır.
- *DIT (Depth of Inheritance Tree-Kalıtım Ağacının Derinliği):* DIT, bir sınıftan kalıtım ağacındaki kök elemana olan en uzak yolun uzaklığıdır.
- *RFC (Response for a Class-Sınıfın Tetiklediği Metot Sayısı):* RFC, bir sınıf içerisinde yer alan kalıtımdan gelen metot sayısı ile metot sayısının toplamıdır.
- *NOC (Number of Children-Alt Sınıf Sayısı):* NOC bir sınıftan türeyen sınıfların sayısı ya da sınıfın çocuklarının sayısıdır.
- *CBO (Coupling between Object Classes- Nesne Sınıfları Arasındaki Bağımlılık):* CBO, bir sınıfın kalıtım dışında bağlı olduğu farklı sınıfların sayısını ifade eder. A sınıfı B sınıfının özelliklerini (attribute) veya operasyonlarını kullanıyorsa, A sınıfının B sınıfına bağlı (coupled) olduğu söylenir.

- *LCOM (Lack of Cohesion in Methods- Metotlardaki Uyum Eksikliği):* LCOM her özellik (attribute) için o özelliği kullanan metotların yüzdesi belirlenip ortalaması alındıktan sonra bu değerin 100'den çıkarılması ile bulunan değerdir. Sınıf içi kohezyon yüksek olması gerekirken, sınıflar arası bağlılık (coupling) düşük olmalıdır [1].

2.4. LİTERATÜR TARAMASI

Bu tez çerçevesinde, yazılım hata tahmini ile ilgili yakın geçmişte yayınlanmış makine öğrenmesi algoritmaları ve hata tahmini konularını içeren çeşitli makaleler incelenmiştir. Gua vd. 2003 yılında, Venkata vd. 2006 yılında ve Elish ve Elish 2008 yılında, hata tahmin tekniklerinin tahmin doğruluğunun, yüksek yanlış sınıflandırma oranıyla %70 ile %85 arasında değişerek oldukça düşük olduğunu bulmuştur [4]. 2008 yılında yazılım hatası tahminiyle ilgili yapılan bir çalışmada hata tahmin modellerinin kapasitesini değerlendirebilecek uygun performans değerlendirme ölçütlerinin olmaması oldukça önemli bir endişe olarak görülmüştür [5]. Menzies vd. göre ise diğer bir endişe, önyargılı öğrenmeye yol açabilecek yazılım hatası veri setlerindeki hataların eşit olmayan dağılımı ile ilgilidir [6]. Hall vd. 2012 yılında yazılım mühendisliğinde hata tahmin performansı üzerine bir inceleme sunmuştur [7]. Bu çalışmanın amacı, yazılım hata tahmini performansında hata tahmin modeli, kullanılan yazılım ölçütleri, bağımlı değişkenler ve hata tahmin tekniklerinin bağlamını değerlendirmektir. Radjenovic vd. ise 2013 yılında hata tahmini için yazılım ölçütlerinin analizi ile ilgili bir inceleme çalışması sunmuştur. Bu çalışmada araştırmacılar tarafından en çok kullanılan nesneye yönelik metriklerin (%49), ardından geleneksel kaynak kod metriklerinin (%27) ve işlem ölçümlerinin (%24) en çok kullanılan ikinci ve üçüncü ölçümler olduğunu bulmuş ve geleneksel boyut veya karmaşıklık ölçütlerine kıyasla hata tahmini için nesne odaklı ve işlem metriklerini kullanmanın daha faydalı olduğu sonucuna varmış ve işlem metriklerinin yayın sonrası hataları tahmin etmede statik kod ölçütlerine kıyasla önemli ölçüde daha iyi sonuçlar ürettiğini görmüşlerdir [8]. Kamei ve Shihab (2016), yazılım hatası tahmini ve bileşenlerine kısa bir genel bakış sağlayan bir çalışma sunmuştur. Çalışma, yazılım hatası tahmininde yapılan başarıların altını çizdi ve bölgedeki mevcut eğilimleri tartışmıştır. Yazılım hatası tahmini için gelecekteki zorluklardan biri tanımlanmış ve tartışılmıştır. Bununla birlikte, çalışma, yazılım hatası tahmini

üzerine çeşitli çalışmaların ayrıntılarını sağlamadı. Ayrıca, yazılım hatası tahmini ile ilgili mevcut çalışmaların avantajları ve dezavantajları incelenmiştir [9].

Denaro anten konfigürasyon yazılımında lojistik regresyon tekniği ile, ticari ve 37 metrikle kusur eğilimli modülleri kestirmeye çalışmıştır [10]. Menzies ve Di Stefano 2004 yılında NASA açık veri setleri üzerinde lineer standart regresyon (LSR), model ağaçları, karar ağaçları, ROCKY ve Delphi detektörleriyle metot seviyesindeki metrikler ile kusur kestirimi üzerinde çalışmıştır. Bu çalışmada değerlendirme kriterleri olarak doğruluk, duyarlık, kesinlik, hassasiyet, çaba seçilmiştir [11]. Khoshgoftaar ve Seliya kusur kestirimi üzerine çalışmalarında (2004), büyük bir telekom sisteminde lojistik regresyon, CART (classification and regression trees), ağaç tabanlı sınıflandırma, C4.5 yöntemlerini kullanmışlardır. Değerlendirme kriteri olarak, yanlış sınıflamada beklenen maliyet kriteri kullanılmıştır [12].

2004 yılında Wang vd. yaptıkları çalışmada büyük bir telekom sisteminde, C dili ile geliştirilmiş Kümeleme Genetik Algoritması yöntemi ile ürün metrikleri kullanılarak yapay sinir ağı tabanlı kalite kestirim modelleri ile ilgili bir çalışma sunmuşlardır. Burada değerlendirme kriteri olarak doğruluk ve yapay sinir ağlarından kural elde edebilmek için Kümeleme Genetik Algoritması kullanılmış, Kümeleme Genetik Algoritmasından elde edilen kural kümesi kullanılarak yapılan kestirimdeki doğruluk oranı, yapay sinir ağı ile elde edilen doğruluk oranından daha düşük çıkmıştır [13]. Mahaweerawat vd. web sayfalarından elde edilen 3.000 satır C++ sınıf kodları üzerinde önce çok katmanlı algılayıcı ile var olan modüllerin kusur eğilimli olup olmadığını tespit etmeye çalışmışlardır. Burada değerlendirme kriteri olarak; doğruluk, Tip-I hatası, Tip-II hatası gibi kriterler kullanılmıştır [14]. Kaminsky ve Boetticher'in çalışmasında NASA veri setlerinden KC2 üzerinde genetik algoritmayla 21 adet metot seviyesindeki metrik ve değerlendirme kriteri olarak T-testi kullanılmıştır [15]. Kanmani vd. çalışmasında, öğrenci projeleri üzerinde 64 adet sınıf metriği kullanılarak GRNN (General Regression Neural Network- Genel Regresyon Sinir Ağı), yöntemiyle kalite kestirimi yapılmıştır. Sonuçlara bakıldığında nesneye yönelik metrikler ile oldukça iyi sonuçlar elde edilmiştir [16].

Zhong vd. NASA'nın KC2 veri seti üzerinde K-ortalama kümeleme ve Neural-Gas kümeleme algoritmalarını kullanarak ve değerlendirme kriteri olarak; ortalama karesel hata (mean squared error-MSE), ortalama saflık (average purity) ve zaman kullanarak kalite kestiriminde bulunmuşlardır [17]. Xing vd. çalışmasında, medikal görüntüleme yazılımı kullanılarak Destek

Vektör Makineleri ile 11 adet metot seviyesinde karmaşıklık metriği ile kalite kestirimi yapılmıştır. Tip-I ve Tip-II hatası, değerlendirme kriteri olarak kullanılmıştır. Çalışmaya göre; Destek Vektör Makineleri diskriminant analizinden ve sınıflama ağacından daha iyi sonuç vermiştir [18]. Koru ve Liu, NASA açık veri setleri üzerinde J48 ve KStar algoritmalarını kullanılarak metot ve sınıf seviyesindeki metrikler ile kusur kestiriminde modül büyüklüğünün etkisini araştırmışlardır. F-ölçüm değeri değerlendirme kriteri olarak seçilmiştir. Büyük bileşenlerde kusur tespitinin gerçekleştirilmesi gerektiği, küçük bileşenlerde de metot seviyesi metriklerindense sınıf seviyesindeki metriklerin kullanılması gerektiği önerilmiştir [19]. Khoshgoftaar vd. 2005 yılında yaptıkları bir çalışmada, iki gömülü yazılım sistemleri üzerinde, C4.5 karar ağacı, diskriminant analizi ve lojistik regresyon yöntemlerini kullanmışlardır. Yanlış sınıflandırmanın beklenen maliyeti kriteri değerlendirme kriteri olarak seçilmiştir. Çalışma sonunda; üç grup içeren sınıf etiketlerinin (yüksek-orta-düşük / kırmızı-sarı-yeşil) daha iyi sonuçlar verdiği görülmüştür [20].

Koru ve Liu, NASA açık veri setleri üzerinde J48, K-Star, Rastgele Orman yöntemleri ile metot ve sınıf seviyesindeki metrikleri kullanarak kusur kestirimi modeli üzerine çalışmışlardır. Bu çalışmadaki değerlendirme kriteri F-ölçüm'dür. Çalışmada; yalnızca J48 algoritmasında değil, K-Star ve Rastgele Orman algoritmalarında da yüksek F-ölçüm değeri elde edildiği görülmüştür [21]. Challagulla ve diğerleri, NASA açık veri setleri üzerinde lineer regresyon, pace regresyonu, destek vektör regresyonu, sinir ağı, destek vektör lojistik regresyonu, Naive Bayes, örnek tabanlı öğrenme (instance based learning-IBL), J48 gibi yöntemler ile metot seviyesindeki metrikleri kusur kestiriminde kullanmışlardır [22]. Gyimothy vd. 2005 yılında Mozilla açık kaynaklı projesi üzerinde lojistik ve lineer regresyon, karar ağaçları ve sinir ağları yöntemleri ile sınıf seviyesindeki metrikler kullanarak, kusur kestirimi üzerine çalışmışlardır ve değerlendirme kriteri olarak; tamlık, hatasızlık, hassasiyet kriterlerini seçmişlerdir [23].

Tomaszewski vd. büyük bir telekom sistemi üzerinde tek ve çok değişkenli lineer regresyon yöntemlerini kullanarak metot ve sınıf seviyesindeki metrikler ile erken kusur kestirimi üzerinde çalışmış ve değerlendirme kriteri olarak "determinasyon katsayısı"nı tercih etmişlerdir [24]. Hassan ve Holt, 6 açık kaynaklı proje üzerinde kusur yatkınlı en çok olan ilk 10 bileşeni değerlendirmişlerdir. Metrik olarak ise; değiştirilme sıklığı, büyüklük, son zamanlarda değiştirilip değiştirilmediği bilgileri yer almaktadır. Değerlendirme için; isabet oranı (hit rate) ve ortalama kestirim yaşı (average prediction age) kriterleri seçilmiştir [25].

Ma vd. NASA açık veri setleri üzerinde çok sayıda makine öğrenmesi algoritmasının performansını incelemiş, metot seviyesinde metrikleri kullanmışlardır. Değerlendirme kriteri olarak; G-ortalama1, G-ortalama2, F-ölçüm, ve ROC eğrisi altında kalan alan seçilmiştir. G-ortalama1, G-ortalama2, F-ölçüm kriterlerine göre en iyi kusur kestiriminde ilk üçte yer alan algoritmaların olduğu görüşmüştür [26]. Challagulla ve arkadaşları NASA açık veri setleri üzerinde tekniğiyle 21 metot seviyesindeki metrik ile kusur kestirimi üzerinde çalışmışlardır. Değerlendirmde; PD, PF ve doğruluk kriterleri tercih edilmiştir [27]. Khoshgoftaar ve arkadaşları büyük bir telekom sistemi üzerinde kusurların kestirimi üzerine çalışmışlardır. Değerlendirme kriteri olarak; ortalama mutlak hata ve ortalama bağıl hata kullanılmıştır [28]. Nikora ve Munson, NASA'nın Jet İtici Güç Laboratuvarı'ndaki (Jet Propulsion Laboratory) Görev Veri Sistemi (Mission Data System) üzerinde metot seviyesindeki metriklerle kusur kestiricileri üzerinde çalışmışlardır [29].

Zhou ve Leung, NASA'nın KC1 veri setinde lojistik regresyon, Naive Bayes, Rastgele Orman, en yakın yöntemlerini kullanarak sınıf seviyesindeki metrikler ile yüksek ve düşük öncelikli kusurların kestirilmesi konusunda çalışma yapmışlar, değerlendirme kriteri olarak ise; tamlık, hatasızlık, hassasiyet kriterlerini tercih etmişlerdir [30]. Boetticher 2006 yılında, NASA açık veri setleri üzerinde J48 ve Naive Bayes yöntemleri ile metot seviyesindeki metrikleri kullanmış, değerlendirme kriteri olarak; PD, PF ve doğruluk kriterlerini seçmiştir [31]. Mertik vd. NASA veri setleri üzerinde Budanmış ve Budanmamış C4.5 (Prunned ve Unprunned C4.5), çoklu yöntem (multimethod), DVM teknikleri ile metot seviyesindeki metrikleri kullanarak yazılım kalite kestirimi üzerinde çalışmışlardır [32]. Bibi vd. çalışmasında, Finlandiya'daki ticari bir bankanın Pekka isimli veri setinde, sınıflandırma yoluyla regresyon tekniğiyle kusur kestirimi yapmıştır. Burada değerlendirme kriteri olarak; ortalama mutlak hata ve doğruluk kriterleri kullanılmıştır [33].

Gao ve Khoshgoftaar iki gömülü yazılım sisteminde Poisson regresyon modeli, sıfırla şişirilmiş Poisson modeli, negatif binomlu regresyon modeli, sıfırla şişirilmiş negatif binomlu model, Hurdle regresyon teknikleri ile 5 adet dosya seviyesindeki metriği kullanarak yazılım kusur kestirimi üzerine çalışmışlardır [34]. Li ve Reformat, NASA açık veri seti olan JM1 üzerinde metot seviyesindeki metrikler ile doğruluk kriterine göre yazılım kusur kestirimi üzerine çalışmışlardır [35]. Yang vd. yapay bir veri setinde "Bulanık Kendi Kendine Adapte Olabilen Öğrenme Kontrol Ağı" yöntemi ile karmaşıklık, kalıtım ağacının derinliği gibi metrikler ile yazılımın güvenirliği ve etkinliğini ölçmeyi amaçlamışlardır [36]. Pai ve Dugan NASA veri

setleri üzerinde doğrusal regresyon, Poisson regresyonu ve lojistik regresyon teknikleriyle modüllerin kusur eğilimliliğini araştırmışlardır. Metrik olarak, sınıf seviyesindeki CK metrikleri ve kod satır sayısı değerlendirme kriteri olarak duyarlık ise, kesinlik, hassasiyet, yanlış pozitif, yanlış negatif kriterleri seçilmiştir [37].

Wang vd. iki büyük telekom sisteminde genetik algoritma yöntemini kullanarak, 18 metot seviyesinde metriği ve 14 dosya seviyesinde metriği ile kalite kestiriminde bulunmuşlardır. Bu çalışmada değerlendirme kriteri olarak; Tip-I hatası, Tip-II hatası, yanlış sınıflandırma oranı seçilmiştir [38]. Seliya ve Khoshgoftaar, NASA açık veri setlerinden JM1'i eğitim verisi olarak, KC1, KC2, KC3 veri setlerini ise test verileri olarak seçmiştir. Daha sonra Beklenti Maksimizasyonu yöntemiyle, 13 tane metot seviyesindeki metrik ile kusur kestirimi üzerine çalışmışlardır [39]. Koru ve Liu, KOffice ve Mozilla açık kaynaklı projeleri üzerinde ağaç tabanlı model yöntemleri ile sınıf seviyesindeki metrikleri kullanmışlardır [40].

Cukic ve Ma, NASA'nın JM1 veri setinde WEKA'da yer alan 16 makine öğrenme algoritmasını kullanarak metot seviyesindeki metrikler ile kusur yatkınlığını araştırmışlardır. Kriter olarak, PD ve PF kullanılmıştır [41]. Tomaszewski vd. 2007 yılında, Ericsson'da geliştirilen iki yazılım sisteminde uzman görüşü tekniği ve tek değişkenli doğrusal regresyon yöntemini kullanarak sınıf ve bileşen seviyesindeki metrikler ile kusur kestirimi üzerine çalışmışlardır. Bu çalışmada değerlendirme kriteri olarak, doğruluk seçilmiştir [42]. Seliya ve Khoshgoftaar, NASA veri setleri üzerinde, k-ortalama algoritması kullanılarak 13 metot seviyesindeki metrik ile yazılım kalite analizi üzerine bir çalışma yürütmüşlerdir. Burada değerlendirme kriteri olarak; Tip-I hatası, Tip-II hatası, yanlış sınıflandırma oranı kullanılmıştır [43].

Olague vd. bir yazılım sisteminde çeşiti yöntemler ile CK, MOOD ve QMOOD metriklerini kullanmışlardır [44]. Binkley vd. Mozilla projesi ve bir ticari yazılım ürününde, kusur kestirimi üzerine çalışmışlardır [45]. Jiang vd. 2007 yılında, NASA veri setleri üzerinde çeşitli makine öğrenmesi teknikleriyle bazı metinlerden elde edilen metrikler ve kod metriklerini kullanarak kusur kestiriminde bulunmuşlardır [46]. Yang ve arkadaşları, yazılım projelerinden elde edilen veri setinde iki yazılım metrik analizi için Affinity Propagation(Yakın İlgi Yayılımı) adlı yeni bir kümeleme yöntemi önermişlerdir. Karşılaştırma için K-means yöntemi kullanılmıştır [47]. Marcus vd. yorumlar ve tanımlayıcılar gibi kaynak kodda gömülü olan yapılandırılmamış bilgilerin analizine dayalı olarak nesne tabanlı yazılım sistemlerindeki sınıfların bütünlüğü için yeni bir ölçü önermektedir. Kavramsal Sınıfların Uyumu olarak adlandırılan ölçü, bilişsel

psikoloji ve hesaplamalı dilbilimde metinsel tutarlılığı ölçmek için kullanılan mekanizmalardan esinlenmiştir [48].

Shafi vd. iki veri setinde otuz farklı tekniği karşılaştırmışlardır [49]. Riquelme vd. NASA'ya ait beş veri setinde iki dengeleme tekniği ile AUC değerlerini iyileştirdiğini tespit etmişlerdir [50]. Shatnawi ve Li, Eclipse'in üç projesinde hata eğilimli sınıfları bulmaya yönelik çalışmışlardır [51]. Arisholm vd. Java sisteminin yüksek hata olasılığına sahip parçalarını tanımlamak için öngörücü modeller oluşturmaya çalışan endüstriyel bir ortamda gerçekleştirilen bir çalışma yapmıştır. Birçok yöntem ve metrik seti kullanılmıştır [52]. Çatal ve Diri, Rastgele Ormanlar gibi makine öğrenmesine dayanan yüksek performanslı hata tahmin edicilerine ve Yapay Bağışıklık Sistemleri adı verilen yeni bir hesaplamalı zeka yaklaşımına dayanan algoritmalara odaklanmıştır. Rasgele Ormanlar, büyük veri setleri için en iyi tahmin performansını sağlarken, Naive Bayes, AUC değerlendirme parametresi açısından küçük veri setleri için en iyi tahmin algoritmasıdır [53]. Turhan ve Bener'in çalışmalarında NASA verilerinde yazılımdaki kusurları tahmin etme açısından, Naive Bayes algoritmasının diğer yöntemlerden daha iyi performans gösterdiğini ortaya koymuştur [54]. Hall vd. Ocak 2000'den Aralık 2010'a kadar yayınlanan 208 hata tahmin çalışmasını belirlemek için sistematik bir literatür taraması kullanmış, iyi performans gösteren modellerin, Naive Bayes veya Lojistik Regresyon gibi basit modelleme teknikleri olduğunu görmüştür [55]. Tablo 2.2'de bu çalışmalar özet halinde sunulmuştur.

Tablo 2.2: Yazılım Hata Tahmini Literatür Çalışmaları

Kaynak	Veri Set(ler)i	Yöntem(ler)
[4]	NASA'nın Çeşitli Veri Setleri	MBR
[5]	NASA'nın Çeşitli Veri Setleri	NB, Lojistik, J48,Bagging
[6]	NASA'nın Çeşitli Veri Setleri	
[7]	Bir Literatür Araştırması	Metrik araştırması
[8]	Bir Literatür Araştırması	Metrik araştırması
[9]	Bir Literatür Araştırması	Hata Tahmin Teknikleri
[10]	Bir Anten Konfigürasyon Sistemi Verileri	Lojistik Regresyon
[11]	NASA'nın Çeşitli Veri Setleri	LSR, Model Ağaçları
[12]	Bir Telekomünikasyon Sistemi Verileri	CART
[13]	Bir Telekomüminkasyon Sistemi Verileri	YSA
[14]	Web Sayfalarından Toplanan C ++ Verileri	MLP
[15]	NASA KC2 Veri Seti	
[16]	Pondicherry Üniversitesi Öğrenci Proje Verileri	GRNN
[17]	NASA KC2 Veri Seti	K-Ortalama
[18]	Bir Medikal Görüntü Yazılım Verileri	SVM
[19]	NASA'nın Çeşitli Veri Setleri	Karar Ağacı, K*
[20]	Çeşitli Gömülü Yazılım Sistemi Verileri	Karar Ağacı, Diskriminant Analizi, Lojistik Regresyon Analizi
[21]	NASA'nın Çeşitli Veri Setleri	Karar Ağacı, K*, Rastgele Ağaçlar

[22]	NASA'nın Çeşitli Veri Setleri	Lineer Regresyon, Pace Regresyonu, Destek vektör regresyonu, NB, Karar Ağacı
[23]	Mozilla Verileri	Lineer Regresyon, SVM, NB
[24]	Bir Telekominikasyon Sistem Verileri	Lojistik Regresyon, Lineer Regresyon, Karar Ağacı, YSA
[25]	Çeşitli Açık Kaynak Kodlu Proje Verileri	
[26]	NASA'nın Çeşitli Veri Setleri	Çok sayıda makine öğrenme algoritması
[27]	NASA'nın Çeşitli Veri Setleri	Çeşitli Yöntemler
[28]	Bir Telekominikasyon Sistem Verileri	Çeşitli Yöntemler
[29]	Jet Propulsion Laboratuvarı Verileri	
[30]	NASA KC1 Veri Seti	Lojistik Regresyon, NB, Rastgele Orman
[31]	NASA'nın Çeşitli Veri Setleri	Karar Ağacı ve NB
[32]	NASA'nın Çeşitli Veri Setleri	Karar Ağacı, SVM
[33]	Finlandiya Bankası Verileri	RvC
[34]	Çeşitli Gömülü Yazılım Sistemi Verileri	Poisson Regresyonları, Hurdle Regresyonları
[35]	NASA JM1 Veri Seti	
[36]	Yapay Bir Veri Seti	Bulanık Öğrenme Konrol Ağı
[37]	NASA'nın Çeşitli Veri Setleri	Lineer Regresyon, Poisson Regresyonu, Lojistik Regresyon
[38]	Bir Telekomünikasyon Sistem Verileri	GA
[39]	NASA JM1 Veri Seti	EM
[40]	KOffice ve Mozilla Verileri	Ağaç tabanlı model yöntemleri
[41]	NASA JM1 Veri Seti	Çeşitli makine öğrenmesi algoritmaları
[42]	Ericcson Sistem Verileri	Tek Değişkenli Doğrusal Regresyon
[43]	NASA'nın Çeşitli Veri Setleri	K-ortalama
[44]	Bir Yazılım Sistemi Verileri	Çeşitli makine öğrenmesi algoritmaları
[45]	Mozilla Verileri	Çeşitli makine öğrenmesi algoritmaları
[46]	NASA'nın Çeşitli Veri Setleri	NB, Lojistik Regresyon, Karar Ağacı, Rastgele Orman, IBk
[47]	Bir Yazılım Sistemi Verileri	Affinity Propagation Sınıflandırma Algoritması ve K- Ortalama
[48]	Bir Yazılım Sistemi Verileri	
[49]	Bir Yazılım Sistemi Verileri	
[50]	NASA'nın Çeşitli Veri Setleri	Dengeleme Teknikleri
[51]	Eclipse Projesine Ait Veriler	
[52]	Bir Java Sistemi Verileri	Çeşitli Yöntemler
[53]	NASA'nın Çeşitli Veri Setleri	Rasgele Orman, NB
[54]	NASA'nın Çeşitli Veri Setleri	NB ve çeşitli yöntemler
[55]	Çeşitli Literatür Çalışmaları	NB, Lojistik Regresyon

Burada LSR Least Square Residual (En Küçük Kare Kalıntı), CART Classification and Regression Tree (Sınıflandırma ve Regresyon Ağacı), YSA Yapay Sinir Ağları, MLP Multilayer Perceptrons (Çok Katmanlı Algılayıcı), GRNN General Regression Neural Network (Genel Regresyon Sinir Ağı), SVM Support Vector Machine (Destek Vektör Makinesi), MBR Memory Based Reasoning (Belleğe Dayalı Akıllı Yürütme), LWL Locally Weighted Learning (Yerel Ağırlıklı Öğrenme), Ibk Instance Based Learner (Örneğe Dayalı Öğrenme), RvC Regression via Classification (Sınıflandırma Yoluyla Regresyon), EM Expectation Maximization (Beklenti Maksimizasyonu) anlamına gelmektedir.

3. MALZEME VE YÖNTEM

Bu bölümde, tez çalışmasında kullanılan veri setleri, değerlendirme kriterleri, uygulama platformları ve yöntemler alt başlıklar halinde açıklanacaktır.

3.1. NASA VERİ SETLERİ

Tez çalışmasının bu bölümünde PROMISE veri tabanında yer alan ve NASA'ya ait yazılım projelerindeki bilgileri içeren JM1, PC1, KC1, CM1 adlı dört veri seti incelenmiştir. Literatürdeki diğer modellerle kıyaslama imkânı oluşturması, veri setlerinin tüm araştırmacılara ücretsiz ve açık bir şekilde sunulması, oldukça detaylı şekilde bilgilerin oluşturulmuş olmasından dolayı projelerdeki hata bilgileri ve metrikleri kullanılmıştır.

Veri setleri, metrik ölçüm değerlerini ve değişkenleri içerir. Veri setindeki kayıtların her birinin bir sınıf etiketi vardır, bu da bağlı olduğu yazılım modülünde bildirilen veya rapor edilmeyen bir hata olduğu anlamına gelir. Tablo 3.1'de, veri setlerinin adları, sahip oldukları özelliklerin sayısı, kayıt sayısı, kullanılan programlama dili ve içeriği verilmektedir. Veri setleri özellikleri de aşağıda sunulmuştur.

Tablo 3.1: Veri Setlerinin Özellikleri

Adı	Özellik Sayısı	Metod Sayısı	Programlama Dili	İçerik
JM1	22	10.885	C	Yer Sistemi Bilgileri
KC1	22	2.109	C++	Lokasyon Depo Yönetimi Bilgileri
CM1	22	498	C	Uzay Araç Bilgileri
PC1	22	1.109	C	Uçuşlardaki Dünya Yörünge Bilgileri

JM1: C dili ile geliştirilmiş, gerçek zamanlı bir projedir. 315.000 kod satırı ve 10.885 fonksiyon içermektedir. 8 yıllık hata bilgisi içermektedir ve modüllerde hata raporlandıkça düzeltmeler yapılarak güncellenmektedir. NASA'da yer alan en kapsamlı veri setlerindendir. Bu veri setindeki modüllerin %19'u test ya da sahada hataya neden olmuştur.

KC1: C++ dili ile geliştirilmiştir ve 750.000 kod satırı 2.109 modülden oluşmaktadır. 5 yıllık hata verisi içermekte olan veri setindeki modüllerin %15'i test ya da sahada hataya neden olmuştur. Kod satırı olarak JM1'den geniş olsa da modül ve içerdiği hata miktarı ile daha küçük bir veri setidir.

CM1: C dili ile geliştirilmiş, 20.000 kod satırı ve 498 modüle sahip, 2 yıllık hata bilgisi içeren ve modüllerinin %10'u hatalı olan bir projedir.

PC1: C dili ile geliştirilmiş, 40.000 satır ve 1109 modüle sahip, 3 yıllık hata bilgisi içeren ve bu modüllerin %7'si test ya da sahada hataya neden olan bir veri setidir [1].

Tablo 3.2 veri setlerinde kullanılan McCabe ve Halstead ölçümlerinin özelliklerini açıklamaktadır.

Tablo 3.2: Metrik Özellikleri

Özellik	Açıklama
loc	McCabe kod satır sayısı"
v(g)	McCabe "çevrimsel karmaşıklık"
ev(g)	McCabe "temel karmaşıklık"
iv(g)	McCabe "tasarım karmaşıklığı"
n	Halstead toplam operatör +operand
v	Halstead "yoğunluk"(volume)
l	Halstead "program uzunluğu"(level)
d	Halstead "zorluk"(difficulty)
i	Halstead "zeka"(intelligence)
e	Halstead "çaba"(effort)
b	Halstead "hata"(bug)
t	Halstead "zaman tahmincisi"(time)
lOCode	Halstead "satır sayısı"
lOComment	Halstead "yorum satır sayısı"
lOBlank	Halstead "boş satır sayısı"
uniq_Op	Benzersiz operatörler
uniq_Opnd	Benzersiz operandlar
total_Op	Toplam operatörler
total_Opnd	Toplam operandlar
branchCount	Akış Grafiğindeki Şube Sayısı
defects	{false,true} Modülde rapor edilmiş bir veya birden fazla hata sayısı

McCabe ölçümleri, temel karmaşıklık, döngüsel karmaşıklık, tasarım karmaşıklığı ve Kod Satır Sayısı'ndan (Line of Code - LOC) oluşan dört yazılım metriğini içerir. Siklomatik Karmaşıklık veya "$v(G)$", "doğrusal olarak bağımsız yolların" sayısını ölçer. Kümedeki hiçbir yol, bir programın "akış grafiği" aracılığıyla kümedeki diğer yolların doğrusal bir kombinasyonu değilse, bir yol kümesinin doğrusal bağımsız olduğu söylenir. Akış şeması, her düğümün bir program ifadesine karşılık geldiği ve her yayın bir ifadeden diğerine kontrol akışını gösterdiği yönlendirilmiş bir grafiktir. $v(G) = e - n + 2$ ile hesaplanır, burada G bir programın akış şemasıdır, e akış şemasındaki yay sayısıdır ve n akış şemasındaki düğümlerdir. Standart McCabes kuralları ($v(G) > 10$), hataya açık modülü tanımlamak için kullanılır.

Temel Karmaşıklık veya "$ev\,(G)$", "D yapılı primerler" olan tüm G alt akış grafiklerinin ayrıştırılmasıyla "azaltılabilen" bir akış şeması ölçüsüdür. Bu tür "D-yapılı primerler" bazen "tek girişli tek çıkışlı aşağı akış diyagramları" olarak anılır. "$ev\,(G)$", $ev\,(G) = v\,(G) - m$ kullanılarak hesaplanır; burada m, D-yapılı primerler olan "G" nin alt akış grafiklerinin sayısıdır.

Tasarım Karmaşıklığı veya "$iv\,(G)$", bir modülün azaltılmış akış şemasının döngüsel karmaşıklığıdır. Bir modülün "G" akış grafiği, tasarım modülleri arasındaki ilişkiyi etkilemeyen karmaşıklığı ortadan kaldırmak için küçültülmüştür. McCabe'ye göre, bu karmaşıklık ölçümü, modelleri hemen bağımlı modüllerine çağıran modülleri yansıtır.

Kod satırları McCabe'nin satır sayısı kurallarına göre ölçülür. Halstead metrikleri üç gruba ayrılır: temel ölçüler, türetilmiş ölçüler ve kod ölçü satırları [56]. Temel metrikler ve türetilmiş metrikler ve McCabe ve Halstead Metrics'in tanımları ve hesaplamaları hakkında detaylı bilgiler Tablo 3.3'de verilmiştir.

Tablo 3.3: Metrik Detayları

Temel Metrikler	**Açıklama**	**Türetilmiş Metrikler**	**Açıklama**
mu1	Benzersiz operatör sayısı	$P = volume = V = N * log2(mu)$	N uzunluğunda bir program yazmak için gereken mantıksal karşılaştırma sayısı
mu2	Benzersiz operand sayısı	$V^* = (2 + mu2') * log2(2 + mu2')$	Minimum uygulamadaki hacim
N1	Operatörlerin toplam oluşumu	$L = V^*/N$	Program uzunluğu
N2	Operandların toplam oluşumu	$D = 1/L$	Zorluk
length	$N = N1 + N2$	$L' = 1/D$	
vocabulary	$mu = mu1 + mu2$	$I = L' * V'$	Zeka
mu1' = 2	Potansiyel operator sayısı (Yalnızca fonksiyonun adı ve "geri dönüş" operatörü)	$E = V/L$	Program yazma çabası
mu2'	Potansiyel operand sayısı (modül argümanlarının sayısı)	$T = \frac{E}{18}s$	Programı yazmak için gerekli süre

3.2. DENEYSEL VERİ SETİ

Tez çalışmasının bu bölümünde, Bilgisayar Mühendisliği bölüm öğrencilerine ait C++ dilinde geliştirilen 102 tane yazılım projelerindeki program kodları kullanılarak elde edilmiş yazılım metrikleri deneysel bir veri seti haline getirilmiştir. Veri setindeki 102 proje kodu içinde 72

öğrencinin kodunda hata bulunmakta, 30 öğrencinin kodunda ise hata bulunmamaktadır. Bu veri setini oluşturan yazılım metriklerinin seçiminde NASA veri setlerinde ortak olan ve literatürde de sıklıkla kullanılan metrik ölçüm değerleri kullanılmıştır. Tablo 3.4'de deneysel veri setinin sahip oldukları özellikler, kullanılan programlama dili ve içeriği verilmektedir. Deneysel veri setini oluşturmak için NASA veri setlerinde de ortak olarak yer alan 14 metrik seçilmiştir. Tablo 3.5'de deneysel veri setine ait metriklerin özellikleri yer almaktadır.

Tablo 3.4: Deneysel Veri Seti Özellikleri

Adı	Özellik Sayısı	Satır Sayısı	Programlama Dili	İçerik
Öğrenci Veri Seti	14	9.142	C++	Öğrenci Yazılım Proje Kodları

Tablo 3.5: Deneysel Veri Seti Metrik Özellikleri

Özellik	Açıklama
loc	McCabe kod satır sayısı
n	Halstead Kelime Sayısı
N	Halstead Program Uzunluğu
v	Halstead "yoğunluk" (volume)
d	Halstead "zorluk" (difficulty)
e	Halstead "çaba"(effort)
b	Halstead "hata"(bug)
t	Halstead "zaman tahmincisi" (time)
lOComment	Halstead "yorum satır sayısı"
uniq_Op	Benzersiz operatörler
uniq_Opnd	Benzersiz operandlar
total_Op	Toplam operatörler
total_Opnd	Toplam operandlar
defects	{1,0} Modülde rapor edilmiş bir veya birden fazla hata sayısı

Veri setini oluşturma aşamasında öncelikle ilk adım olarak 102 adet yazılım projesi için her yazılım koduna ait operatör ve operand sayıları hesaplamıştır. Operatörler bir yazılım kodunda yer alan aritmetik ve mantıksal işlemleri gerçekleştiren (for, if, else if, while vs) gibi anahtar kelimeler ve semboller olarak adlandırılır. Operandlar ise, aritmetik ve mantıksal işlemlerde kullanılan diğer değişkenler ve sabit sayılara verilen addır. Yazılım proje kodlarında yer alan bu operatörler ve operandlar sayısal olarak toplanarak diğer yazılım metriklerinin hesaplanmasında temel girdi olarak rol oynar. Veri setini oluşturmak için 102 yazılım kodunda yer alan tüm operatör ve operandlar teker teker sayılarak kaç adet oldukları, isim etiketlendirmesi ile beraber bir dosyaya kaydedilmiştir. Öncelikle 102 tane yazılım proje

kodlarında kullanılan benzersiz (tekil) operatör sayısı ve toplam operatör sayıları hesaplanmış ve oluşturulmuş her bir projeyi temsilen proje kod numarası ile birlikte tabloya eklenmiştir. Daha sonra ise benzersiz (tekil) operand ve toplam operand sayısı hesaplanmış ve benzer şekilde proje kod numaraları ile birlikte tabloya eklenmiştir. Hesaplanan benzersiz operatör-toplam operatör ve benzersiz operand- toplam operand sayısal bilgileri deneysel veri seti için kulanılacak olan temel metriklerdir. Projelerde kullanılmış operatör ve operand sayısal bilgilerinin yer aldığı tablolarının ekran görüntüleri Şekil 3.1 ile Şekil 3.29 arasında verilmiştir.

Yazılım Proje Kod Numaraları	for	if	elseif	while	switch	return	char	int	:	,	(	)	[	]	{	}	+	-	&	%	cout	cin	break	*	**	"="	"=="	true	false	static	main	"++"	bool	include	iostream	stdlib	math	void	printf	scanf	string	const	!=	|	^	case	'	&&	Toplam Operatör Sayısı	Tekil Operatör Sayısı
C1	4	6				2	3	9	23	1	12	12	31	31	11	11		3			6			2	1	13	9	2	4		1		1	1	1									6	1			4	211	28
C2	7	3	2			1	2	6	40	10	13	13	43	43	15	15		1			15	2	4	2		10	1				1			1	1						1		8	1			4	265	28
C3	7	9				1	1	13	35	1	27	27	47	47	16	16		10			19			3		18		1	4		1	1	1	4	1								2	13	7			4	336	28
C4	10	5				1	1	6	31	3	19	19	31	31	12	12				1	4			1		15	8				1			3	1	1	1						1	13	3			4	238	27
C5	4	7	4	2		9	1	25	31	10	26	26	49	49	20	20		7		1	13			2		14	8				1	1		2	1	1		2				1	10	6	4			7	364	32
C6	4	2	3			1	2	8	30	4	14	14	41	41	11	11		8			9		4	2		15	9				1			1	1									3	1		2	4	246	27
C7	6	6	4	2		1	1	15	28	1	28	28	53	53	16	16		7		1	10			1		17	8				1	1		2	1	1						1	6	8	4			4	331	31
C8	3	3	3			6	2	4	20	11	28	28	40	40	10	10		4		1				3		11		4	1		1	2	1	2		1			10				1	5	2			4	261	30
C9	4	7	7			1	2	10	41	3	20	20	40	40	21	21		6			12	1		1		28	14				1			1	1								4	5	1			4	316	27
C10	5	3	7			1	2	12	18	4	20	20	58	58	8	8		4		1	11			3		9					1	1		5	1	1				1			11	7		1	8	289	29
C11	10	56				1		16	24	72	131	131	92	92	67	67			3	38						13	56				1			2		1		59	3				12	2			4	953	24
C12	11	7	3	2		6	1	14	36	16	41	41	53	53	27	27		3		6				1		22	8				1	1		2		1		9				1	13	4				410	28
C13	6	3	3				1	9	32		14	14	38	38	11	11		3			8	2	5			11	2				1	1		1	1									7	1			4	227	25
C14	10	56				1		16	24	72	133	133	92	92	67	67		2	3	38						13	56				1			2		1		59	3				12	2				955	24
C15	2	6					1	8	22	2	23	23	23	23	10	10		34		1	10			2		10	9	4	1		1		1	1	1			2			1		3	5			8	247	29
C16	4	8			1	5	2	8	19	3	15	15	27	27	14	14		6			7		4		1	9	4				1			2	1						3		8	4	4		216	27
C17	4	7					2	11	22	1	13	13	25	25	12	12		4			5			1	1	10	7				1	4		2	1	1						1	6	2				193	26
C18	6	13						3	29	7	21	21	30	30	20	20		29			13	2				10					1			1	1								1	9	1				268	21
C19	4	6				6	9	14	31	5	21	21	51	51	15	15		30			10			1		14	9	4	2		1		1	2	1			2					1	6	2			4	339	29
C20	7	3	3			1	1	13	27	1	18	18	37	37	16	16					7		4	2		14	10				1			3	1	2					3		10	3			4	262	27
C21	6	2	3				1	11	12	1	25	25	33	33	12	12		1			10			1		9	8				1			1	1								1	7	1			4	221	25
C22	4	3	3			1	3	8	21	4	14	14	41	41	13	13		30			5		5	1		11	9				1			2	1								1	6	2			4	261	27
C23	4	7		1	1	1		9	36	5	14	14	16	16	13	13		9				3	4			15		1	8		1		1	1	1								3	3	2	4		2	208	29
C24	13	15	5	1		10	16	65	103	40	54	54	91	91	40	40		1			15			4		70	25				2	1		1	1			2				3	1	15	1			7	287	30

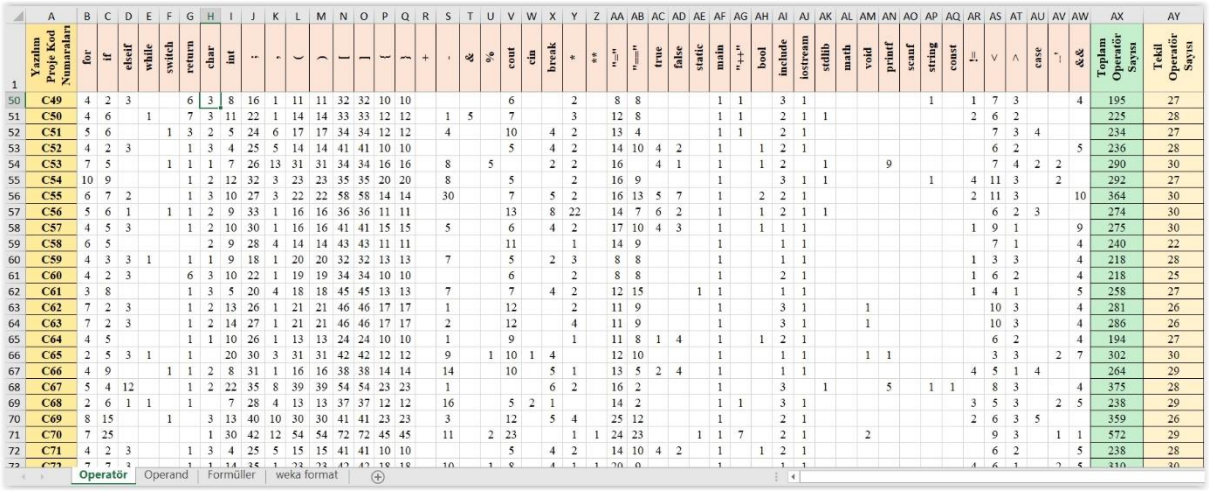
Operatör | Operand | Formüller | weka format

Şekil 3.1: 1-24 Numaralı Yazılım Proje Kodlarındaki Operatör Sayıları

Yazılım Proje Kod Numaraları	for	if	elseif	while	switch	return	char	int	:	,	(	)	[	]	{	}	+	-	&	%	cout	cin	break	*	**	"="	"=="	true	false	static	main	"++"	bool	include	iostream	stdlib	math	void	printf	scanf	string	const	!=	|	^	case	'	&&	Toplam Operatör Sayısı	Tekil Operatör Sayısı
C25	4	1	3			5	2	5	21	2	11	11	36	36	9	9		2			13			1		6					1			2	1								6	2			4	193	25
C26	4	4				1	1	10	19	1	11	11	24	24	10	10		1			9			1		7	7	1			1			2	1								6	2			4	172	25
C27	9	4	13			1	2	27	53	2	37	37	60	60	25	25		16			15		5	2		28	17	2	5		1	2	1	3	1	1					1	1	11	3			5	475	33
C28	2	5	1	1		5		5	24	3	16	16	37	37	7	7		16		1	5	2	1			8	10				1	1		5	1		1				2		1	7	5		2	5	240	32
C29	4	13	3			14	10	34	34	47	41	41	54	54	13	13		5			4		2	1		18	8	11	10		1		11	2	1						1	1	10	3			7	471	31
C30	4	2	3			1	1	10	23	2	13	13	39	39	11	11		8		1	5		4	1		11	9				1			1	1									1	1		2	4	222	28
C31	7	3	3			2	2	16	27	1	22	22	45	45	17	17		1			11			4		13	10				1			3	1									10	3			4	290	25
C32	10	9				1	2	12	32	3	23	23	35	35	20	20		8			5			2		16	9				1			3	1	1					1		4	11	3		2		292	27
C33	12	7	3			1	2	16	43	5	34	34	35	35	23	23		6			12			1		15	8				1			4	1								4	12	4		2	3	346	27
C34	4	6				1	3	5	26	5	15	15	41	41	11	11					11			1		14	1	4	2		1	5	1	4	1								1	8	4			4	246	28
C35	5	4	12			1	2	22	39	4	34	34	54	54	23	23		1			5		6	2		15	2				1	1		3	1	1					1	1	8	3			4	366	30
C36	4	2	3			1	1	10	23	2	13	13	39	39	11	11		8		1	5		4	1		11	9				1			1	1									1	1		2	4	222	28
C37	3	4	3					9	20	7	42	42	38	38	11	11		13	2	2			4	1		14	10				1			2		1			15	1				5	2		2	7	310	28
C38	4	4	3			1	1	8	21	1	17	17	44	44	13	13		4			9		1	2		11	10				1			1	1								1	5	1			4	242	27
C39	7	3	2			1	2	5	40	8	13	13	39	39	15	15		6					4	3		11	9				1			1	1									8	1			4	251	25
C40	4	2	3			1	2	5	19	2	11	11	40	40	9	9					8			1		12	1	2	4		1		1	2	1	1								6	2			4	204	28
C41	3	3	4			1	2	5	16	7	31	31	41	41	13	13		7			7			2		6	15			1	1			1	1									4	1			3	260	26
C42	4	5				1	1	10	26	1	1	13	13	24	24	10	10		1			9			1		12		2	4		1	1	1	2	1									6	2			4	189	27
C43	4	4	3			1	9	12	27	5	22	22	38	38	15	15					10		1	1		14	10	5	2		1			1	2	1			4				1	6	2			4	280	30
C44	8	14				1	2	12	31	1	24	24	38	38	21	21		4			5		6	1		16	10	2	6		1		1	1	1								4	6	1				300	28
C45	4	1	3			5	2	5	20	2	11	11	32	32	9	9					9			1		6					1			2	1									6	2			4	178	23
C46	6	2	3			6	1	8	17	2	19	19	32	32	10	10					6			3		8	8				1			2	1						3			7	2			4	212	25
C47	8	1	3			2		3	22	3	17	17	22	22	11	11					10	2				6	8				1			1	1									7	1			4	183	24
C48	7	7	1			2	1	15	31	2	20	20	37	37	16	16		8		1	6		4	1	1	16	9				1			1	1								2	7	1		2	4	277	30

Operatör | Operand | Formüller | weka format

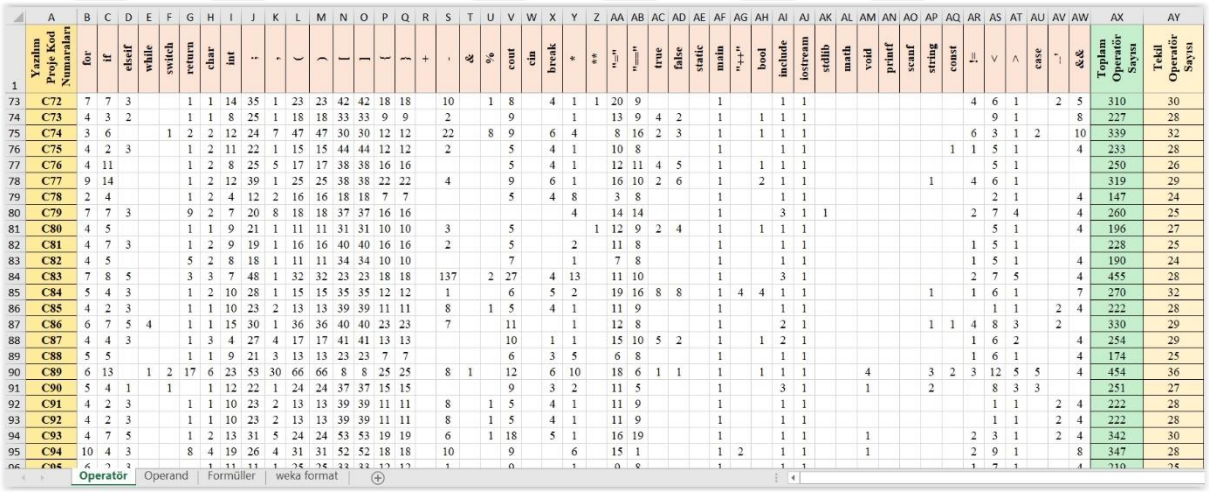
Şekil 3.2: 25-47 Numaralı Yazılım Proje Kodlarındaki Operatör Sayıları



Yazılım Proje Kod Numaraları	for	if	elseif	while	switch	return	char	int	:	,	(	)	[	]	{	}	+	-	&	%	cout	cin	break	*	**	"\n"	"==="	true	false	static	main	"++"	bool	include	iostream	stdlib	math	void	printf	scanf	string	const	!=	\/	^	case	_'	&&	Toplam Operatör Sayısı	Tekil Operatör Sayısı
C49	4	2	3			6	3	8	16	1	11	11	32	32	10	10					6			2		8	8				1	1		3	1						1		1	7	3			4	195	27
C50	4	6		1		7	3	11	22	1	14	14	33	33	12	12		1	5		7			3		12	8				1	1		2	1	1					2	6	2				225	28
C51	5	6			1	3	2	5	24	6	17	17	34	34	12	12		4			10		4	2		13	4				1	1		2	1							7	3	4				234	27
C52	4	2	3			1	3	4	25	5	14	14	41	41	10	10					5		4	2		14	10	4	2		1		1	2	1							6	2				5	236	28
C53	7	5			1	1	1	7	26	13	31	31	34	34	16	16		8		5			2	2		16		4	1		1		1	2		1			9			7	4	2	2		290	30
C54	10	9				1	2	12	32	3	23	23	35	35	20	20		8			5			2		16	9				1			3	1	1				1		4	11	3		2		292	27
C55	6	7	2			1	3	10	27	3	22	22	58	58	14	14		30			7		5	2		16	13	5	7		1		2	2	1							2	11	3			10	364	30
C56	5	6	1		1	1	2	9	33	1	16	16	36	36	11	11					13		8	22		14	7	6	2		1		1	2	1	1							6	2	3			274	30
C57	4	5	3			1	2	10	30	1	16	16	41	41	15	15		5			6		4	2		17	10	4	3		1		1	1	1							1	9	1			9	275	30
C58	6	5				2	9	28	4	14	14	43	43	11	11						11			1		14	9				1			1	1								7	1			4	240	22
C59	4	3	3	1		1	1	9	18	1	20	20	32	32	13	13		7			5		2	3		8	8				1			1	1							1	3	3			4	218	28
C60	4	2	3			6	3	10	22	1	19	19	34	34	10	10					6			2		8	8				1			2	1							1	6	2			4	218	25
C61	3	8				1	3	5	20	4	18	18	45	45	13	13		7			7		4	2		12	15			1	1			1	1							1	4	1			5	258	27
C62	7	2	3			1	2	13	26	1	21	21	46	46	17	17		1			12			2		11	9				1			3	1			1					10	3			4	281	26
C63	7	2	3			1	2	14	27	1	21	21	46	46	17	17		2			12			4		11	9				1			3	1			1					10	3			4	286	26
C64	4	5				1	1	10	26	1	13	13	24	24	10	10		1			9			1		11	8	1	4		1		1	2	1								6	2			4	194	27
C65	2	5	3	1		1		20	30	3	31	31	42	42	12	12		9		1	10	1	4			12	10				1			1	1			1	1				3	3		2	7	302	30
C66	4	9			1	1	2	8	31	1	16	16	38	38	14	14		14			10		5	1		13	5	2	4		1			1	1							4	5	1	4			264	29
C67	5	4	12			1	2	22	35	8	39	39	54	54	23	23		1					6	2		16	2				1			3		1			5		1	1	8	3			4	375	28
C68	2	6	1	1		1		7	28	4	13	13	37	37	12	12		16			5	2	1			14	2				1	1		3	1							3	5	3		2	5	238	29
C69	8	15			1		3	13	40	10	30	30	41	41	23	23		3			12		5	4		25	12				1			2	1							2	6	3	5			359	26
C70	7	25				1	1	30	42	12	54	54	72	72	45	45		11		2	23			1	1	24	23			1	1	7		2	1			2					9	3		1	1	572	29
C71	4	2	3			1	3	4	25	5	15	15	41	41	10	10					5		4	2		14	10	4	2		1		1	2	1								6	2			5	238	28

Operatör | Operand | Formüller | weka format

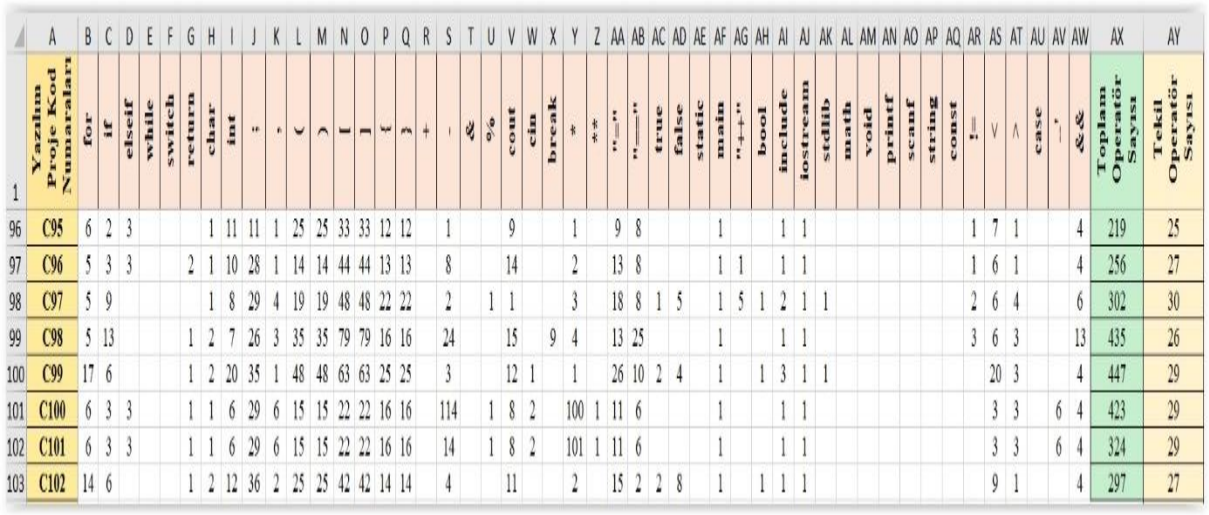
Şekil 3.3: 49-71 Numaralı Yazılım Proje Kodlarındaki Operatör Sayıları



Yazılım Proje Kod Numaraları	for	if	elseif	while	switch	return	char	int	:	,	(	)	[	]	{	}	+	-	&	%	cout	cin	break	*	**	"\n"	"==="	true	false	static	main	"++"	bool	include	iostream	stdlib	math	void	printf	scanf	string	const	!=	\/	^	case	_'	&&	Toplam Operatör Sayısı	Tekil Operatör Sayısı
C72	7	7	3			1	1	14	35	1	23	23	42	42	18	18		10		1	8		4	1	1	20	9					1		1	1							4	6	1		2	5	310	30
C73	4	3	2			1	1	8	25	1	18	18	33	33	9	9		2			9			1		13	9	4	2		1		1	1	1								9	1			8	227	28
C74	3	6			1	2	2	12	24	7	47	47	30	30	12	12		22		8	9		6	4		8	16	2	3		1		1	1	1							6	3	1	2		10	339	32
C75	4	2	3			1	2	11	22	1	15	15	44	44	12	12		2			5		4	1		10	8				1		1	1						1	1	5	1			4	233	28
C76	4	11				1	2	8	25	5	17	17	38	38	16	16					5		4	1		12	11	4	5		1		1	1	1								5	1				250	26
C77	9	14				1	2	12	39	1	25	25	38	38	22	22		4			9		6	1		16	10	2	6		1		2	1	1						1		4	6	1			7	319	29
C78	2	4				1	2	4	12	2	16	16	18	18	7	7					5		4	8		3	8				1			1	1								2	1			4	147	24
C79	7	7	3			9	2	7	20	8	18	18	37	37	16	16								4		14	14				1			3	1	1						2	7	4			4	260	25
C80	4	5				1	1	9	21	1	11	11	31	31	10	10		3			5			1	1	12	9	2	4		1		1	1	1								5	1			4	196	27
C81	4	7	3			1	2	9	19	1	16	16	40	40	16	16		2			5		2			11	8				1			1	1							1	5	1				228	25
C82	4	5				5	2	8	18	1	11	11	34	34	10	10					7			1		7	8				1			1	1							1	5	1			4	190	24
C83	7	8	5			3	3	7	48	1	32	32	23	23	18	18		137		2	27		4	13		11	10				1			3	1							2	7	5			4	455	28
C84	5	4	3			1	2	10	28	1	15	15	35	35	12	12		1			6		5	2		19	16	8	8		1	4	4	1	1						1		1	6	1			7	270	32
C85	4	2	3			1	1	10	23	2	13	13	39	39	11	11		8		1	5		4	1		11	9				1			1	1								1	1		2	4	222	28
C86	6	7	5	4		1	1	15	30	1	36	36	40	40	23	23		7			11			1		12	8				1			2	1						1	1	4	8	3		2		330	29
C87	4	4	3			1	3	4	27	4	17	17	41	41	13	13					10		1	1		15	10	5	2		1		1	2	1								1	6	2			4	254	29
C88	5	5				1	1	9	21	3	13	13	23	23	7	7					6		3	5		6	8				1			1	1								1	6	1			4	174	25
C89	6	13		1	2	17	6	23	53	30	66	66	8	8	25	25		8	1		12		6	10		18	6	1	1		1		1	1	1			4			3	2	3	12	5	5		4	454	36
C90	5	4	1		1		1	12	22	1	24	24	37	37	15	15					9		3	2		11	5				1			3	1			1				2		8	3	3			251	27
C91	4	2	3			1	1	10	23	2	13	13	39	39	11	11		8		1	5		4	1		11	9				1			1	1								1	1		2	4	222	28
C92	4	2	3			1	1	10	23	2	13	13	39	39	11	11		8		1	5		4	1		11	9				1			1	1								1	1		2	4	222	28
C93	4	7	5			1	2	13	31	5	24	24	53	53	19	19		6		1	18		5	1		16	19				1			1	1			1				2	3	1		2	4	342	30
C94	10	4	3			8	4	19	26	4	31	31	52	52	18	18		10			9			6		15	1				1	2		1	1			1				2	9	1			8	347	28

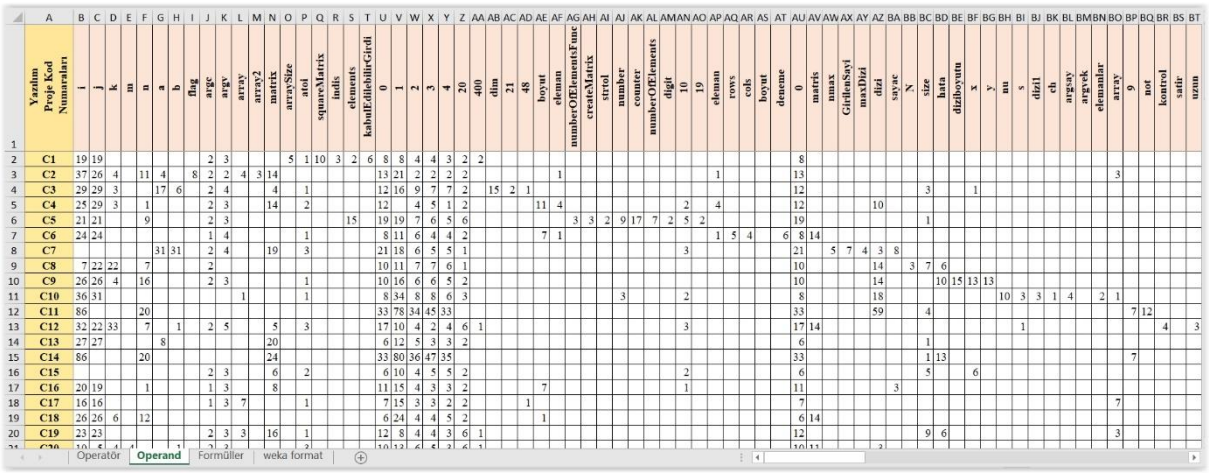
Operatör | Operand | Formüller | weka format

Şekil 3.4: 72-94 Numaralı Yazılım Proje Kodlarındaki Operatör Sayıları



Yazılım Proje Kod Numaraları	for	if	elseif	while	switch	return	char	int	:	,	(	)	[	]	{	}	+	-	&	%	cout	cin	break	*	**	"\n"	"==="	true	false	static	main	"++"	bool	include	iostream	stdlib	math	void	printf	scanf	string	const	!=	\/	^	case	_'	&&	Toplam Operatör Sayısı	Tekil Operatör Sayısı
C95	6	2	3				1	11	11	1	25	25	33	33	12	12		1			9			1		9	8				1			1	1							1	7	1			4	219	25
C96	5	3	3			2	1	10	28	1	14	14	44	44	13	13		8			14			2		13	8				1	1		1	1							1	6	1			4	256	27
C97	5	9					1	8	29	4	19	19	48	48	22	22		2		1	1			3		18	8	1	5		1	5	1	2	1	1						2	6	4			6	302	30
C98	5	13				1	2	7	26	3	35	35	79	79	16	16		24			15		9	4		13	25				1			1	1							3	6	3			13	435	26
C99	17	6				1	2	20	35	1	48	48	63	63	25	25		3			12	1		1		26	10	2	4		1		1	3	1	1							20	3			4	447	29
C100	6	3	3			1	1	6	29	6	15	15	22	22	16	16		114		1	8	2		100	1	11	6				1			1	1								3	3		6	4	423	29
C101	6	3	3			1	1	6	29	6	15	15	22	22	16	16		14		1	8	2		101	1	11	6				1			1	1								3	3		6	4	324	29
C102	14	6				1	2	12	36	2	25	25	42	42	14	14		4			11			2		15	2	2	8		1		1	1	1								9	1			4	297	27

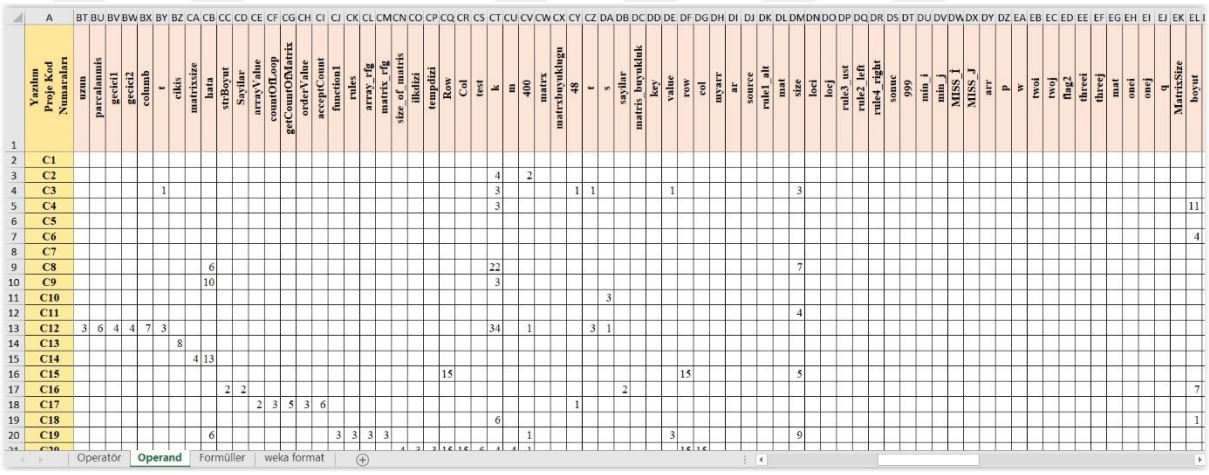
Şekil 3.5: 95-102 Numaralı Yazılım Proje Kodlarındaki Operatör Sayıları



Yazılım Proje Kod Numaraları	i	j	k	m	n	a	b	flag	arge	argv	array	array2	matrix	arraySize	atoi	squareMatrix	indis	elements	kabulEdilebilirGirdi	0	1	2	3	4	20	400	dim	21	48	boyut	eleman	numberOfElements	Fune	createMatrix	strtol	number	counter	numberOfElements	digit	10	19	eleman	rows	cols	boyut	deneme	0	matris	nmax	GirdenSayi	maxDizi	dizi	sayac	N	size	hata	diziboyutu	x	y	au	s	dizi1	eh	arguay	argvek	elemanlar	array	9	not	kontrol	satir	uzun
C1	19	19							2	3				5	1	10	3	2	6	8	8	4	4	3	2	2																				8																														
C2	37	26	4		11	4		8	2	2	4	3	14							13	21	2	2	2	2					1											1					13																										3					
C3	29	29	3			17	6		2	4			4		1					12	16	9	7	7	2		15	2	1																	12							3			1																		
C4	25	29	3		1				2	3			14		2					12		4	5	1	2				11	4											2		4				12					10																				
C5	21	21			9				2	3								15		19	19	7	6	5	6						3	3	2	9	17	7	2	5	2							19							1																		
C6	24	24							1	4					1					8	11	6	4	4	2				7	1													1	5	4		6	8	14																										
C7						31	31		2	4			19		3					21	18	6	5	5	1																	3					21		5	7	4	3	8																				
C8	7	22	22		7				2											10	11	7	7	6	1																						10					14		3	7	6																				
C9	26	26	4		16				2	3					1					10	16	6	6	5	2																						10					14			10	15	13	13																				
C10	36	31									1				1					8	34	8	8	6	3						3											2					8					18					10	3	3	1	4		2	1												
C11	86				20															33	78	34	45	33																						33					59		4																	7	12				
C12	32	22	33		7		1		2	5			5		3					17	10	4	2	4	6	1															3					17	14						1																	4		3
C13	27	27				8			1	4			20							6	12	5	3	3	2																						6							1																				
C14	86				20								24							33	80	36	47	35																						33							1	13																7				
C15									2	3			6		2					6	10	4	5	5	2																	2					6							5		6																		
C16	20	19			1				1	3			8							11	15	4	3	3	2				7													1					11						3																				
C17	16	16							1	3	7				1					7	15	3	3	2	2				1																		7																				7						
C18	26	26	6		12															6	24	4	4	5	2				1																	6	14																										
C19	23	23							2	3	3		16		1					12	8	4	4	3	6	1																						12							9	6																3				
C20	10	5	4	1			1		2	3					2					10	12	6	5	3	6	1																						10	11					3																				

Operatör | Operand | Formüller | weka format

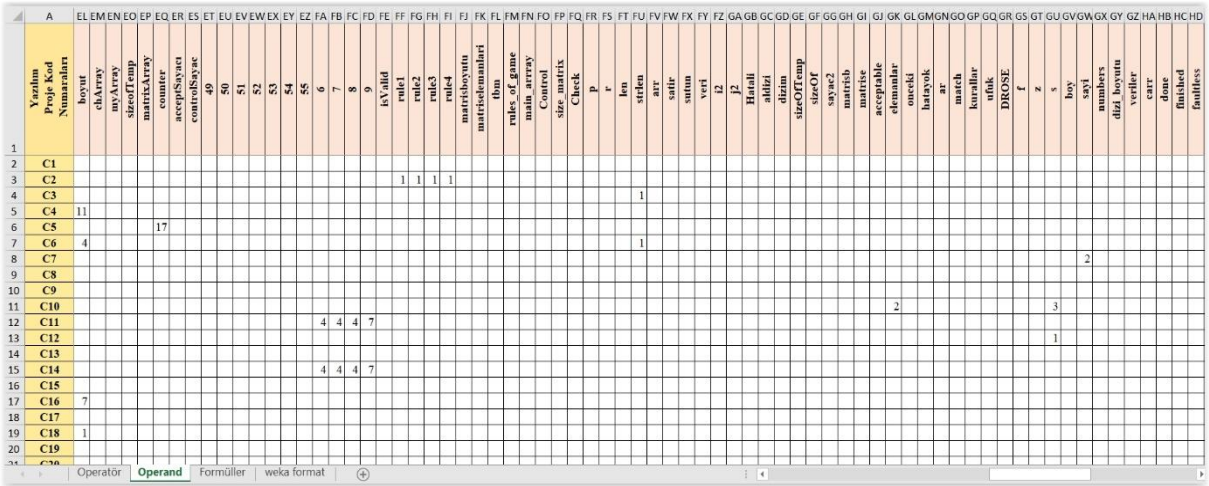
Şekil 3.6: 1-19 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları



Yazılım Proje Kod Numaraları	uzun	parcalanmis	gecici1	gecici2	columb	t	cikis	matrixsize	hata	strBoyut	Sayilar	arrayValue	countOfLoop	getCountOfMatrix	orderValue	acceptCount	function1	rules	array_rfg	matrix_rfg	size_of_matrix	ilkdizi	tempdizi	Row	Col	test	k	m	400	matrix	matrixboyutdegu	48	t	s	sayilar	matris_boyutlulk	key	value	row	col	myarr	ar	source	rule1_alt	mat	size	loci	locj	rule3_ust	rule2_left	rule4_right	sonuc	999	min_i	min_j	MISS_İ	MISS_J	arr	p	w	twoi	twoj	flag2	threei	threej	mat	onei	onej	q	MatrixSize	boyut
C1																																																																											
C2																										4		2																																																			
C3						1																				3					1	1							1																																					
C4																										3																																																	11
C5																																																																											
C6																																																																										4
C7																																																																											
C8								6																		22																																																		
C9								10																		3																																																		
C10																																	3																																											
C11																																																																											
C12	3	6	4	4	7	3																				34		1				3	1																																											
C13							8																																																																							
C14								4	13																																																																				
C15																								15																																																						
C16										2	2																																																																					
C17												2	3	5	3	6																																																																
C18																										6																																																					
C19								6									3	3	3	3								1																																																					
C20																					4	3	15	15	6	4	4	1																																																							

Operatör | Operand | Formüller | weka format

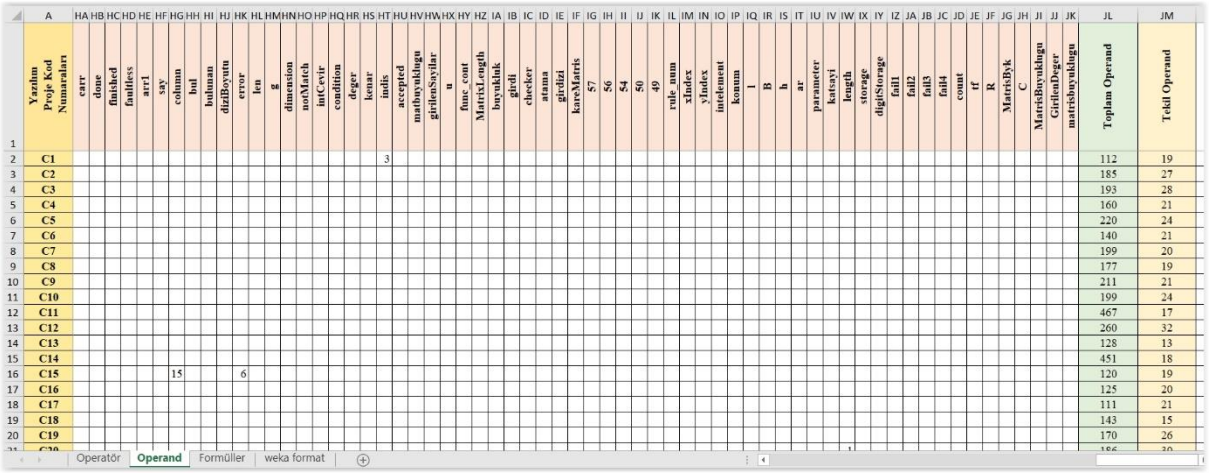
Şekil 3.7: 1-19 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları - 2



Yazılım Proje Kod Numaraları	boyut	elkArray	myArray	sizeofTemp	matrixArray	counter	acceptSayaci	controlSayac	49	50	51	52	53	54	55	6	7	8	9	isValid	rule1	rule2	rule3	rule4	matrisboyutu	matriselemanlari	ibm	rules_of_game	main_array	Control	size_matrix	Check	p	r	len	striln	arr	satir	sutun	veri	i2	j2	Hatali	aldizi	dizim	sizeOfTemp	sizeOf	sayac2	matrisb	matrise	acceptable	elemanlar	onceki	hatayok	ar	match	kurallar	ofok	DROSE	f	z	s	boy	sayi	numbers	dizi_boyutu	veriler	carr	done	finished	faultless
C1																																																																																	
C2																					1	1	1	1																																																												
C3																																						1																																													
C4	11																																																																																	
C5						17																																																																												
C6	4																																																																																	
C7																																																																																		
C8																																																																																		
C9																																																																																	
C10																																																																																	
C11																4	4	4	7																																																															
C12																																																																																	
C13																																																																																	
C14																4	4	4	7																																																														
C15																																																																																	
C16	7																																																																																	
C17																																																																																		
C18	1																																																																																	
C19																																																																																	
C20																																																																																	

Operatör | Operand | Formüller | weka format

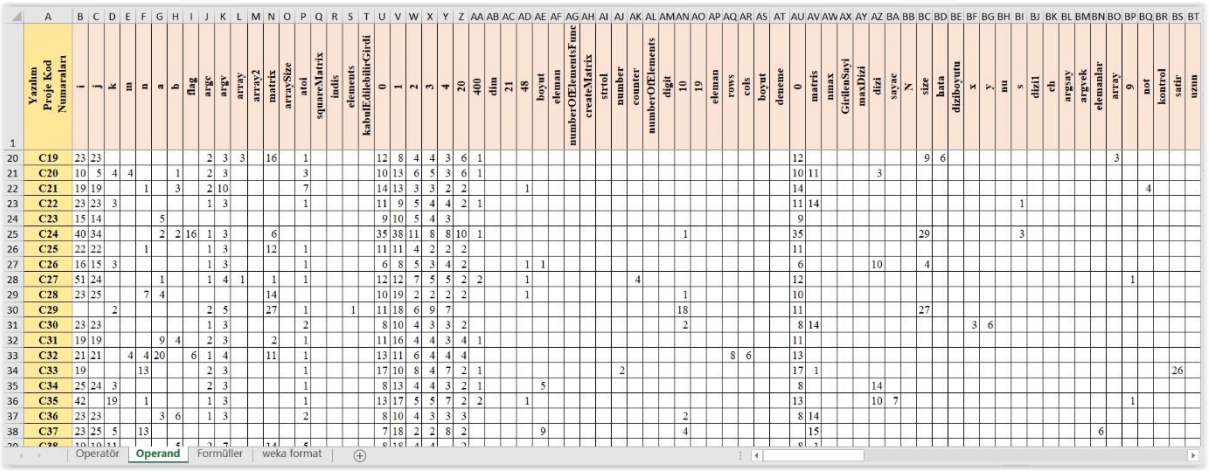
Şekil 3.8: 1-19 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları - 3



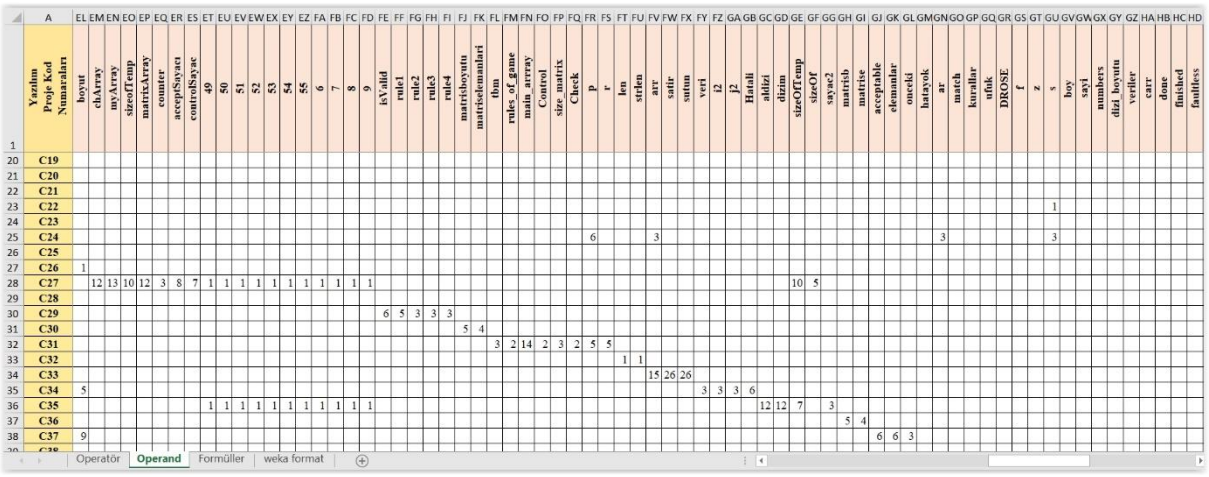
Yazılım Proje Kod Numaraları	carr	done	finished	faulties	arr1	say	column	bul	bulunan	diziBoyutu	error	len	g	dimension	notMatch	initCevir	condition	deger	kenar	indis	accepted	matboyuklugu	girilenSayilar	u	func_cont	MatrixLength	buyukluk	girdi	checker	atanan	girdizi	kareMatris	57	56	54	50	49	rule_num	xIndex	yIndex	intelement	konum	i	B	h	ar	parameter	katsayi	length	storage	digitStorage	fail1	fail2	fail3	fail4	count	tf	R	MatrisByk	C	MatrisBoyuklugu	GirilenDeger	matrisboyuklugu	Toplam Operand	Tekil Operand
C1																				3																																												112	19
C2																																																																				185	27
C3																																																																			193	28
C4																																																																			160	21
C5																																																																			220	24
C6																																																																			140	21
C7																																																																			199	20
C8																																																																			177	19
C9																																																																			211	21
C10																																																																			199	24
C11																																																																			467	17
C12																																																																			260	32
C13																																																																			128	13
C14																																																																			451	18
C15							15				6																																																											120	19
C16																																																																					125	20
C17																																																																					111	21
C18																																																																					143	15
C19																																																																					170	26

Operatör | Operand | Formüller | weka format

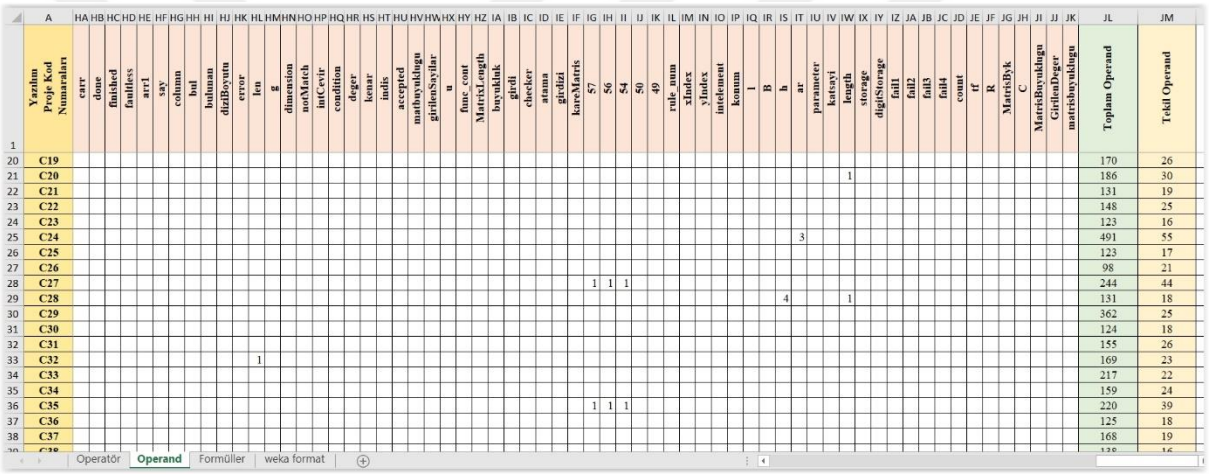
Şekil 3.9: 1-19 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları - 4



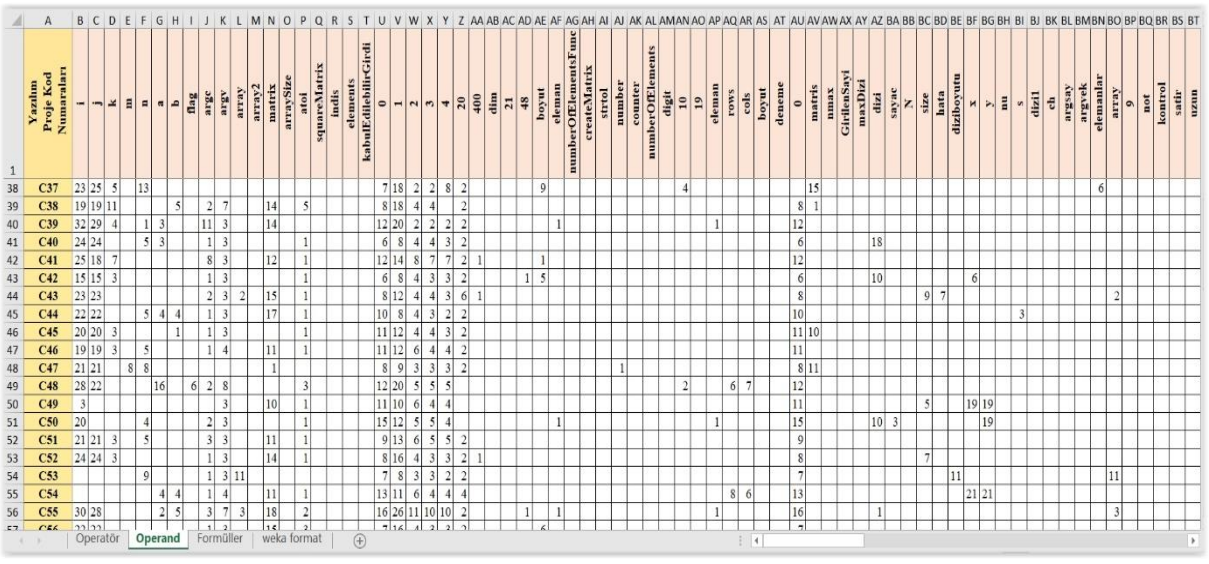
| Yazılım Proje Kod Numaraları | i | j | k | m | n | a | b | flag | arge | argc | array | array2 | matrix | arraySize | atoi | squareMatrix | indis | elements | kabulEdilebilirGirdi | 0 | 1 | 2 | 3 | 4 | 20 | 400 | dim | 21 | 48 | boyut | eleman | numberOfElementsFunc | createMatrix | strtol | number | counter | numberOfElements | digit | 10 | 19 | eleman | rows | cols | boyut | deneme | 0 | matris | nmax | GirilenSayi | maxDizi | dizi | sayac | N | size | hata | diziboyutu | x | y | nu | s | dizi1 | eh | argsay | argvek | elemanlar | array | 9 | not | kontrol | satir | uzun |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---



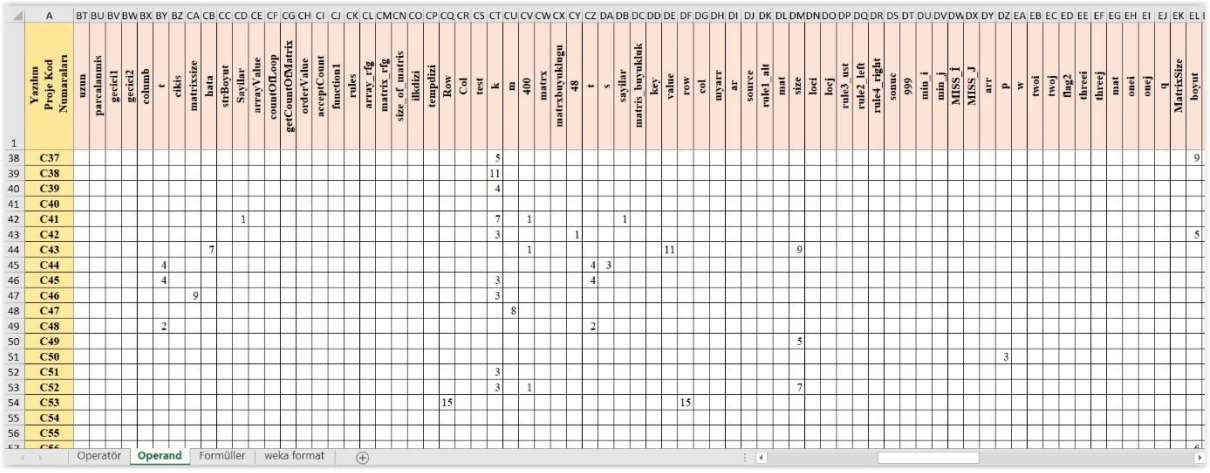
Şekil 3.12: 19-37 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları - 3



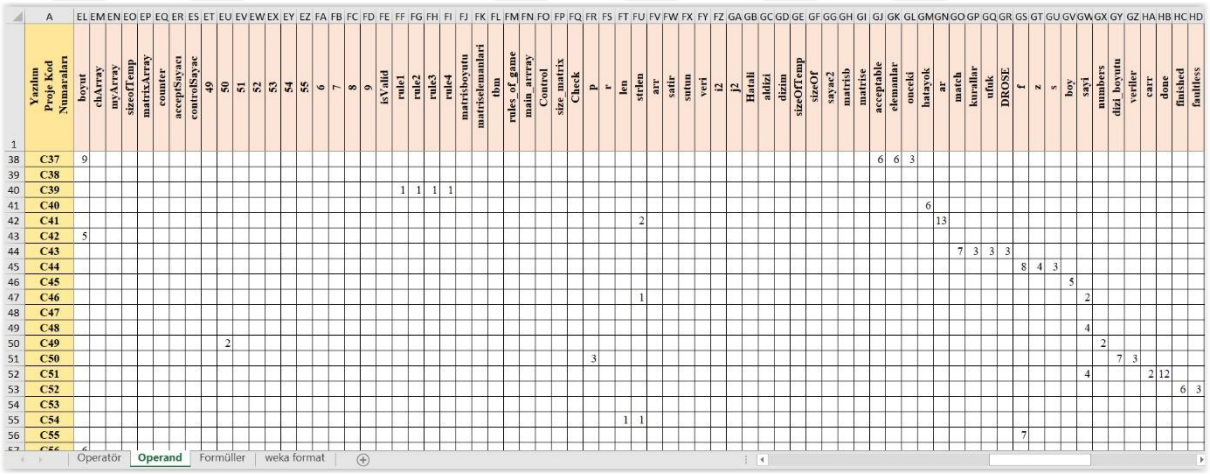
Şekil 3.13: 19-37 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları - 4



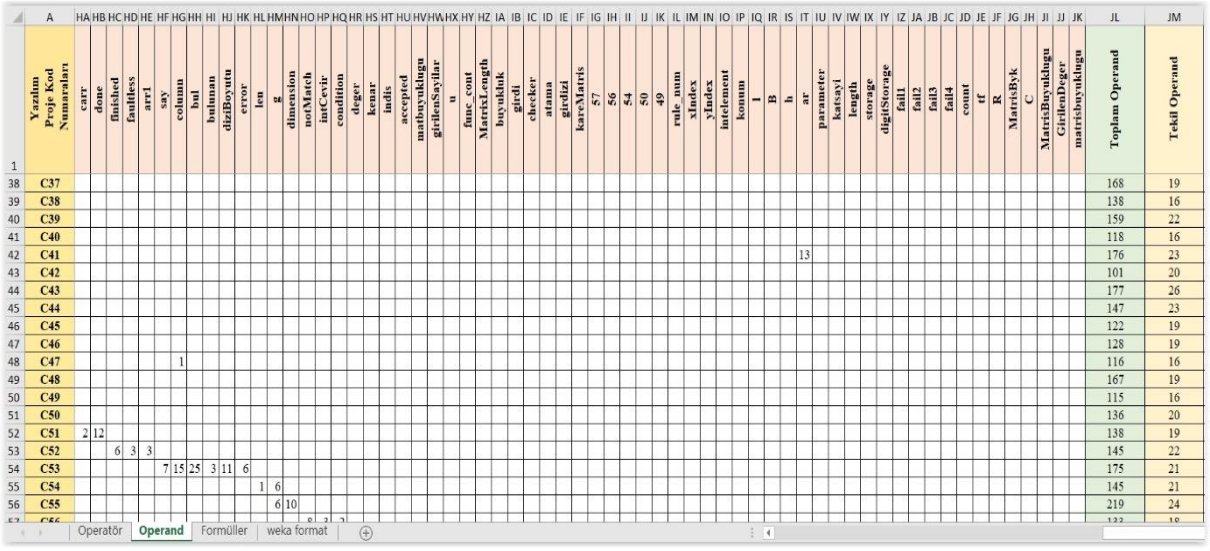
Şekil 3.14: 37-55 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları



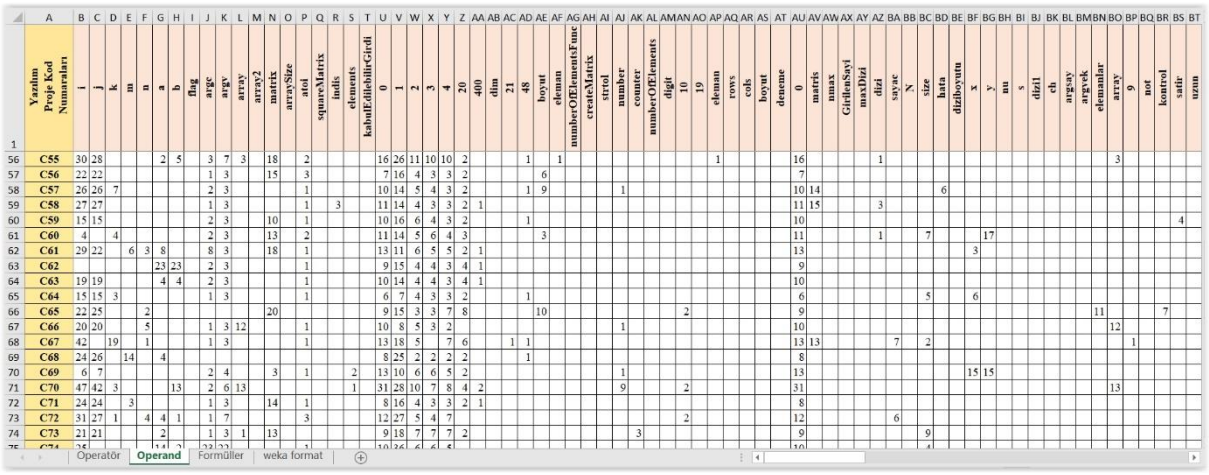
Şekil 3.15: 37-55 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları - 2



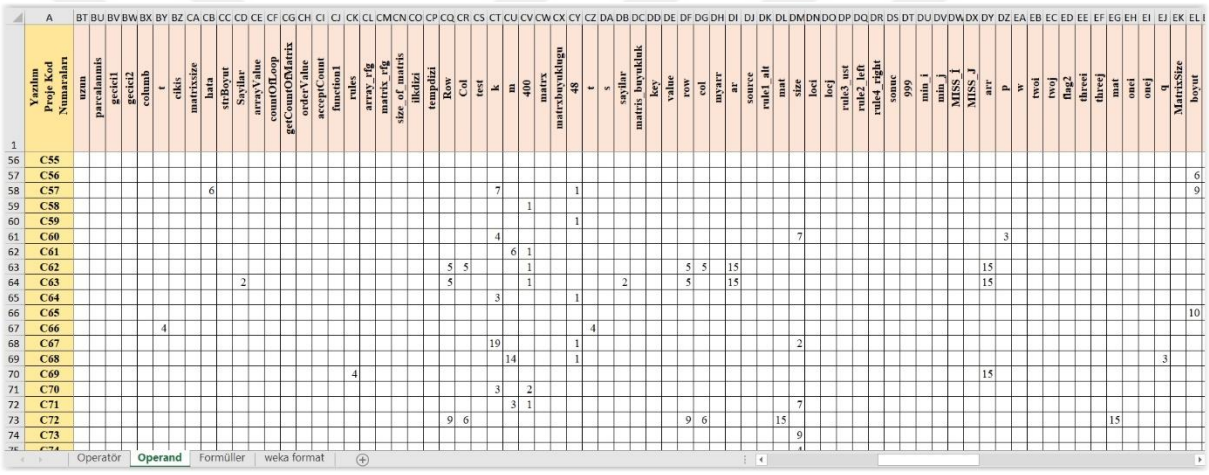
Şekil 3.16: 37-55 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları - 3



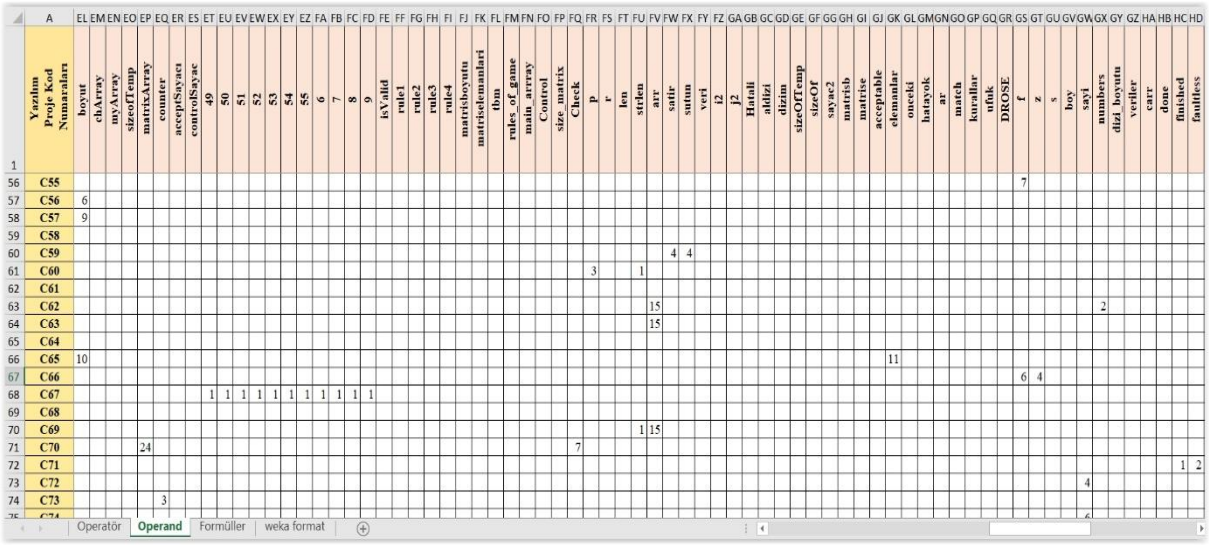
Şekil 3.17: 37-55 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları - 4



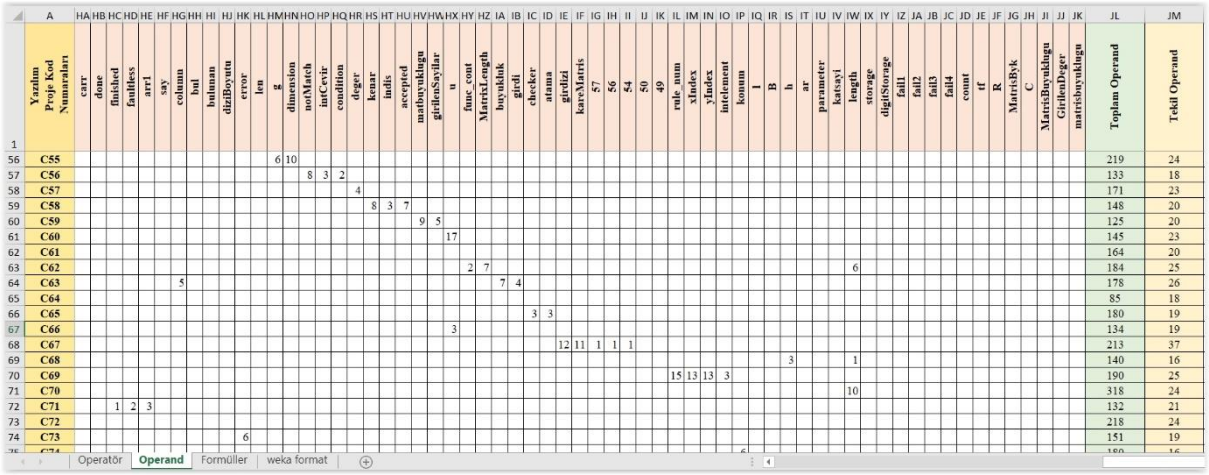
Şekil 3.18: 55-73 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları



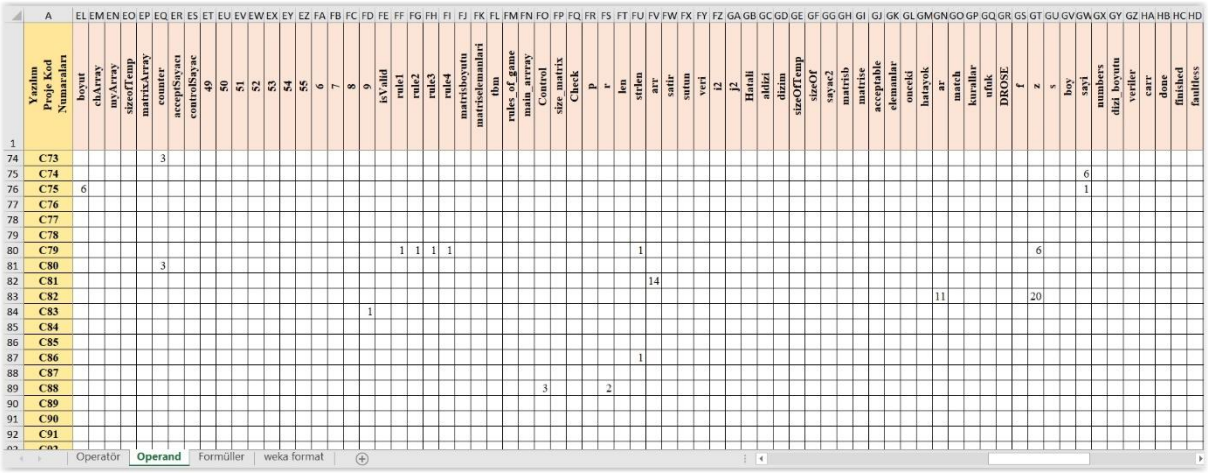
Şekil 3.19: 55-73 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları - 2



Şekil 3.20: 55-73 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları - 3



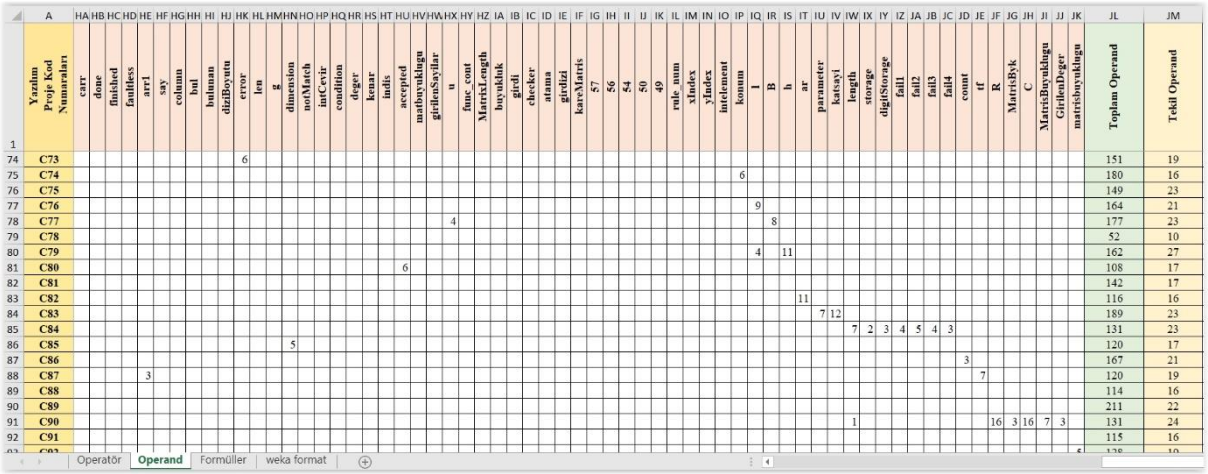
Yazılım Proje Kod Numaraları	carr	done	finished	faultless	arrl	say	column	bul	bulunan	diziBoyutu	error	len	g	dimension	notMatch	initCevir	condition	deger	kenar	indis	accepted	matBuyuklugu	girilenSayilar	u	func_cont	MatrisLength	buyukluk	girdi	checker	atama	girdizi	kareMatris	57	56	54	50	49	rule_num	xIndex	yIndex	inteleuset	konum	1	B	h	ar	parameter	katsayi	length	storage	digitStorage	fail1	fail2	fail3	fail4	count	tf	R	MatrisBsk	C	MatrisBuyuklugu	GirilenDeger	matrisBuyuklugu	Toplam Operand	Tekil Operand
C55												6	10																																																									219	24
C56														8	3	2																																																													133	18
C57																	4																																																													171	23
C58																		8	3	7																		



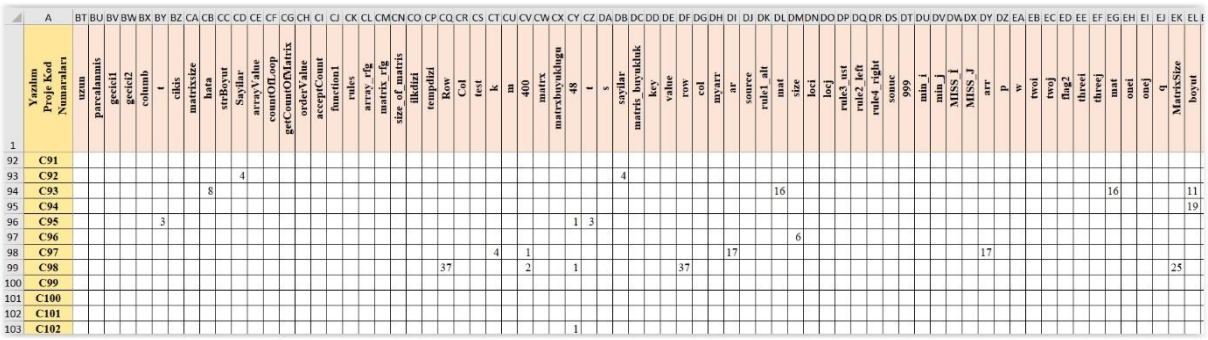
Yazılım Proje Kod Numaraları	boyut	chArray	myArray	sizeofTemp	matrixArray	counter	acceptSayacı	controlSayac	49	50	51	52	53	54	55	6	7	8	9	isValid	rule1	rule2	rule3	rule4	matrisboyutu	matriselemanları	tbm	rules_of_game	main_array	Control	size_matrix	Check	p	r	len	strlen	arr	satir	sutun	veri	i2	j2	Hatali	aldizi	dizim	sizeOfTemp	sizeOf	sayac2	matrisb	matrise	acceptable	elemanlar	onceki	hatayok	ar	match	kurallar	ufok	DROSE	f	z	s	boy	sayi	numbers	dizi_boyutu	veriler	carr	done	finished	faultless
C73						3																																																																								
C74																																																																																						
C75	6																																																																																					
C76																																																																																						
C77																																																																																						
C78																																																																																						
C79																					1	1	1	1													1																																																	
C80						3																																																																																		
C81																																						14																																																		
C82																																																																																										
C83																			1																																																																				
C84																																																																																								
C85																																																																																										
C86																																					1																																																		
C87																																																																																									
C88																																		3		2																																																		
C89																																																																																										
C90																																																																																										
C91																																																																																										

Operatör | Operand | Formüller | weka format

Şekil 3.24: 73-91 Numaralı Yazılım Proje Kodlarındaki Operand Sayıları - 3



| Yazılım Proje Kod Numaraları | carr | done | finished | faultless | arr1 | say | column | bul | bulunan | diziBoyutu | error | len | g | dimension | notMatch | intCevir | condition | deger | kenar | indis | accepted | matboyuklugu | girilenSayilar | u | func_cont | MatrixLength | buyukluk | girdi | checker | atama | girdizi | kareMatris | 57 | 56 | 54 | 50 | 49 | rule_num | xIndex | yIndex | inceleneut | konum | 1 | B | h | ar | parameter | katsayi | length | storage | digitStorage | fail1 | fail2 | fail3 | fail4 | count | tf | R | MatrisByk | C | MatrisBuyuklugu | GirilenDeger | matrisbuyuklugu | Toplam Operand | Tekil Operand |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---



Yazılım Proje Kod Numaraları	uzun	parcalanmis	gecici1	gecici2	columb	t	cikis	matrixsize	hata	strBoyut	Sayilar	arrayValue	countOfLoop	getCountOfMatrix	orderValue	acceptCount	function1	rules	array_rfg	matris_rfg	size_of_matris	ilkdizi	tempdizi	Row	Col	test	k	m	400	matrx	matrixboyutlugu	48	t	s	sayilar	matris_boyutluk	key	value	row	col	myarr	ar	source	rule1_alt	mat	size	loci	locj	rule3_ust	rule2_left	rule4_right	source	999	min_i	min_j	MISS_I	MISS_J	arr	p	w	twoi	twoj	flag2	threei	threej	mat	ouei	ouej	q	MatrixSize	boyut
C91		

3.3. DEĞERLENDİRME KRİTERLERİ

Hata tahmin modelleri için kullanılan veri setlerinde yer alan hatalı modüller, tüm yazılımların çok az bir kısmını kapsamaktadır. Bu tür veri setleri, dengesiz veri setleri olarak isimlendirilir. Bu yüzden, modellerin performanslarını kıyaslarken doğruluk (accuracy) değerini kullanmak yeterli değildir. Hata tahmin konusunda çalışma yapanlar birçok değerlendirme kriteri önermişlerdir. ROC eğrisi alanından, hata matrisinin (confusion matrix) sadece diyagonaldeki değerlerini dikkate alınması veya sadece F-ölçüm değerini kullanarak ya da G-ortalama1, G-ortalama2 ve F-ölçüm değerlerinin hepsini bütünüyle kullanarak değerlendirme yapılması bu kriterlere örnek gösterilebilir. Tez çalışmasında değerlendirme kriterleri olarak doğruluk ve ROC eğrileri kullanılmıştır.

3.3.1. Doğruluk (Accuracy)

Performans değerlendirme kriterlerinden en çok kullanılan doğruluk oranı, verilerin sınıflandırmadaki doğruluk başarısını ölçmektedir. Doğru tahmin edilen pozitif değerlerin (true positive - TP) ve doğru tahmin edilen negatif değerlerin (true negative - TN) toplamının tüm tahmin değerlerine oranı hesaplanarak bulunmaktadır. Formülü Denklem (3.1)'de verilmiştir.

$$Doğruluk = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.1}$$

3.3.2. ROC (Reciever Operator Characteristics Curve–Alıcı Operatör Özellikleri Eğrisi)

Ayrık veri setlerinde yer alan makine öğrenmesi algoritmalarının sınıflandırma başarısını ölçmektedir. Sınıfları birbirinden ayırt edebilme oranını göstermektedir. İki sınıf arasında ayrım yapabiliyorsa başarı oranının yüksek çıkması beklenir aksi takdirde oldukça düşüktür. ROC eğrisi, x ekseninde yanlış alarm olasılığını (probability of false alarm-PF) gösterirken y ekseninde tespit olasılığını (probability of detection-PD) göstermektedir.

3.3.3. AUC (Area Under the Curve - Eğri Altında Kalan Alan)

Bu kriter, ROC eğrisinin altında kalan alandır. Modelin performansını ölçmek için kullanılır. Diğer bir deyişle ROC eğirisinin altında kalan alan AUC metriğinin skorunu vermektedir [57].

AUC değeri 0 ve 1 arasında değer alabilmektedir. AUC değeri 1 olan sınıflayıcı mükemmel sınıflayıcı olarak adlandırılır.

3.4. UYGULAMA PLATFORMLARI

Tez çalışmasında WEKA (Waikato Environment for Knowledge Analysis - Bilgi Analizi için Waikato Ortamı) veri madenciliği aracı ve Google COLAB ortamında Python programlama dili kullanılmıştır.

3.4.1. WEKA Platformu

Veri madenciliği dalında yapılan çalışmaların ve yöntemlerin test ve analizlerinde Waikato Üniversitesince oluşturulan WEKA ismi sıkça geçmektedir. WEKA, bilgisayar bilimlerinin en önemli konularından birisi olan makine öğrenmesinde kullanılan uygulamalardan birisidir. Waikato Üniversitesi'nde açık kaynak kodlu olarak JAVA dilinde geliştirilmiştir.

WEKA değişkenleri sayısal ve nominal değerlerden oluşmaktadır. WEKA verileri basit bir dosyadan okumaktadır ve aynı zamanda veri tabanından veri de çekebilmektedir. Bir çeşit veri dosya türü olan ".arff" (Attribute Relationship File Format) WEKA'ya özel tasarlanmış bir dosya formatıdır. .arff formatlı dosyalar dışında metin tabanlı csv, c45, libsvm, svmlight ve Xarff gibi formatları da desteklemektedir. Ayrıca "jdbc" (Java Derby Client-Java Derbi İstemcisi) sürücüsü bulunan veri tabanlarına doğrudan bağlantı yapabilmektedir. WEKA'nın bir diğer özelliği de internet üzerinden http protokolünü kullanarak bu formatlara uygun dosyaları okuyabilme yeteneğine sahip olmasıdır [58]. WEKA'da makine öğrenmesi ve istatistikle ilgili birçok kütüphane bulunmaktadır. Bunlardan bazıları veri ön işlemesi (data preprocessing), ilkelleme (regression), sınıflandırma (classification), gruplandırma (clustering) ve özellik seçimidir (feature selection). Ayrıca bu işlemlerin sonucunu görsel olarak görmek için görüntüleme araçları bulunmaktadır [59].

3.4.2. Google Colabarotary

Google Colabarotary (Colab) platformu Google tarafından geliştirilen, kullanıcılara ücretsiz bulut hizmeti ve GPU (Graphics Processing Unit-Grafik İşlemci Birimi) imkânı vermektedir. Kullanıcılar böylece kendilerine tanınan belirli kapasite ile Python dilinde uygulama

geliştirebilmektedir. Kullanıcılar tarafından geliştirilen uygulamalar Google Drive'da depolanmaktadır. Colab, makine öğrenimi ve derin öğrenme algoritmaları için birçok kütüphaneye sahiptir. Diğer kütüphaneleri eklemek ve yüklemek oldukça kolay ve hızlıdır. Kullanıcıya Keras, OpenCv, XGBoots, PyTorch, Sklearn ve TensorFlow gibi birçok kütüphaneleri kullanma imkânı sunar. Böylece kullanıcı derin öğrenme uygulamaları geliştirilebilmektedir. Kullanıcının kendi Google Drive hesabındaki veya diğer Google Drive hesaplarındaki verilerle çalışmasına izin verir [57].

Tez çalışmasında Python dilinde geliştirilen uygulamalar için OpenCV, Sklearn, Numpy gibi birçok kütüphaneyi destekleyen ve ücretsiz GPU desteği sağlayan Google Colab platformu kullanılmıştır. Çalışmada kullanılan algoritmalar scikit-learn kütüphanesi kullanılarak yüklenmiştir. Verileri Google Colab'e aktarmak için, veri setleri. csv uzantılı veri setleri önce Google Drive'a yüklenmiştir. Veri setlerine erişim için Google Drive ile Colab arasında gerekli bağlantı kurulmuştur. Veri setlerini sınıflandırmak ve gerekli performans oranlarını hesaplamak için çeşitli kütüphaneler kullanılmıştır.

Sklearn: Skit-Learn'ün kısaltması olan Python kitaplığıdır. Sınıflandırma, kümeleme, ön işleme, regresyon ve model seçimi gibi işlemlerde kullanılan bu kütüphane, veri madenciliği alanında oldukça önemlidir.
NumPy: N boyutlu diziler, karmaşık fonksiyonlar, matematiksel dönüşümler, mantıksal ve istatistiksel işlemler, sıralama, seçim gibi sayısal işlemlerde kullanılan temel Python kütüphanesidir.
Matlibplot: Python dilinde yazılmış 2 boyutlu çizim kütüphanesidir. Grafik arayüzü oluşturma, 2 boyutlu görseller üzerinde işlem yapma, görüntü verilerini yeniden ölçeklendirme, grafik, histogram, çizelge gibi sonuç görselleştirme işlemlerinde kullanılabilmektedir [57].

3.5. YÖNTEMLER

Bu kısımda tez çalışmasında kullanılan yöntemler açıklanacaktır. Toplamda 45 adet kullanılmasından dolayı yöntemler okunabilirliği arttırmak açısından önce ana başlıklara ayrılmış daha sonrasında alt başlıklar halinde maddelendirilmiştir.

3.5.1. BAYES

3.5.1.1. Bayes Ağı (Bayes Net)

Bayes Ağı, çeşitli arama algoritmaları ve kalite ölçüleri kullanan bir öğrenme yöntemidir. Veri yapıları (ağ yapısı, koşullu olasılık dağılımları vb.) ve K2 ve B gibi Bayes Ağı öğrenme algoritmalarında ortak olan tesisler sağlar.

3.5.1.2. Naive Bayes

Tahminci sınıflarını kullanan bir Bayes sınıflandırıcısıdır. Sayısal tahminci kesinlik değerleri, eğitim verilerinin analizine göre seçilir. UpdateableClassifier işlevine ihtiyaç varsa NaiveBayesUpdateable sınıflandırıcısını kullanmak önerilir. NaiveBayesUpdateable sınıflandırıcısı, buildClassifier sıfır eğitim örnekleriyle çağrıldığında sayısal öznitelikler için varsayılan 0,1 duyarlığını kullanır.

3.5.1.3. Naive Bayes Çok Terimli Metin (Naive Bayes Multinomial Text)

Metin verileri için çok terimli Naive Bayes yöntemidir. Doğrudan (ve yalnızca) metin nitelikler üzerinde çalışan bir algoritmadır. Diğer girdi özellikleri türleri kabul edilir ancak eğitim ve sınıflandırma sırasında göz ardı edilir.

3.5.1.4. Naive Bayes Güncellenebilir (Naive Bayes Updateable)

Tahminci sınıflarını kullanan bir Naive Bayes sınıflandırıcısıdır ve Naive Bayes'in güncellenebilir sürümüdür. Bu sınıflandırıcı, buildClassifier sıfır eğitim örnekleriyle çağrıldığında sayısal öznitelikler için varsayılan 0,1 duyarlığını kullanır.

3.5.2. FONKSİYONLAR (FUNCTIONS)

3.5.2.1. Lojistik (Logistic)

Denklem (3.2) ile verilen çok terimli lojistik regresyon modeli oluşturma ve kullanma sınıfıdır. L'nin küçültüldüğü B matrisini bulmak için, $m(k-1)$ değişkenlerinin optimize edilmiş değerlerini aramak için Quasi-Newton Yöntemi kullanılır. Optimizasyon prosedürünü kullanmadan önce, matris B'yi bir $m(k-1)$ vektörüne sıkıştırılmıştır.

M özniteliğine sahip n örnek için k sınıfı varsa, hesaplanacak parametre matrisi B bir $m(k-1)$matrisi olacaktır. Son sınıf haricinde j sınıfı için olasılık $P_j(X_i) = e^{X_i B_j} / (\sum_{j=1}^{k-1} e^{X_i B_j} + 1)$'dır. Son sınıfın olasılığı ise, $1 - (\sum_{j=1}^{k-1} P_j(X_i)) = 1/(\sum_{j=1}^{k-1} e^{X_i B_j} + 1)$'dır. (Negatif) çok terimli log-olabilirlik böylece aşağıdaki gibi olur:

$$L = -\sum_{i=1}^{n} \left\{ \sum_{j=1}^{k-1} Y_{ij} \, ln\left(P_j(X_i)\right) + \left(1 - \sum_{j=1}^{k-1} Y_{ij}\right) ln\left(1 - \sum_{j=1}^{k-1} P_j(X_i)\right) \right\} + ridge(B^2) \tag{3.2}$$

3.5.2.2. Çok Katmanlı Algılayıcı (Multilayer Perceptron)

Örnekleri sınıflandırmak için çok katmanlı bir algılayıcı öğrenmek için geri yayılımı kullanan bir sınıflandırıcıdır. Ağ elle oluşturulabilir veya basit bir buluşsal yöntem kullanılarak kurulabilir. Ağ parametreleri eğitim süresi boyunca izlenebilir ve değiştirilebilir. Bu ağdaki düğümlerin tümü sigmoiddir. (sınıfın sayısal olduğu durumlar hariç, bu durumda çıkış düğümleri eşiksiz doğrusal birimler haline gelir)

3.5.2.3. Stokastik Gradyan İniş (Stochastic Gradient Descent)

Çeşitli doğrusal modelleri (ikili sınıf SVM) , ikili sınıf lojistik regresyon, kare kayıp, Huber kaybı ve epsilon duyarsız kayıp doğrusal regresyon) öğrenmek için stokastik gradyan inişini uygular. Küresel olarak tüm eksik değerleri değiştirir ve nominal özellikleri ikili olanlara dönüştürür. Ayrıca tüm öznitelikleri normalleştirir, böylece çıktıdaki katsayılar normalleştirilmiş verilere dayanır. Sayısal sınıf öznitelikleri için kare, Huber veya epsilon-duyarsız kayıp işlevi kullanılmalıdır. Epsilon'a duyarsız ve Huber kaybı çok daha yüksek bir öğrenme oranı gerektirebilir.

3.5.2.4. Stokastik Gradyan İniş Metin (Stochastic Gradient Descent Text)

Metin verileri üzerinde doğrusal ikili sınıf SVM veya ikili sınıf lojistik regresyon öğrenmek için stokastik gradyan inişini uygular. Doğrudan (ve yalnızca) metin nitelikleri üzerinde çalışır. Diğer girdi özellikleri türleri kabul edilir ancak eğitim ve sınıflandırma sırasında göz ardı edilir.

3.5.2.5. Basit Lojistik (Simple Logistic)

Doğrusal lojistik regresyon modelleri oluşturmak için bir sınıflandırıcıdır. Temel öğrenenler olarak basit regresyon işlevlerine sahip LogitBoost, lojistik modelleri uydurmak için kullanılır. Gerçekleştirilecek optimum LogitBoost yinelemesi sayısı çapraz doğrulanmıştır ve bu da otomatik öznitelik seçimine yol açar.

3.5.2.6. Sıralı Minimal Optimizasyon (Sequential Minimal Optimization)

John Platt'ın bir destek vektörü sınıflandırıcı eğitimi için sıralı minimum optimizasyon algoritmasını uygular. Bu uygulama, küresel olarak tüm eksik değerleri değiştirir ve nominal özellikleri ikili olanlara dönüştürür. Ayrıca varsayılan olarak tüm öznitelikleri normalleştirir. (Bu durumda çıktıdaki katsayılar orijinal verilere değil normalleştirilmiş verilere dayanmaktadır - bu sınıflandırıcıyı yorumlamak için önemlidir.) Çok sınıflı problemler ikili sınıflandırma kullanılarak çözülür. Uygun olasılık tahminlerini elde etmek için, kalibrasyon modellerini destek vektör makinesinin çıktılarına uyduran seçeneği kullanmak gerekir. Çok sınıflı durumda, tahmin edilen olasılıklar Hastie ve Tibshirani'nin ikili birleştirme yöntemi kullanılarak birleştirilir.

3.5.2.7. Oylanmış Algılayıcı (Voted Perceptron)

Freund ve Schapire tarafından oylanan algılayıcı algoritmasının uygulanmış halidir. Global olarak tüm eksik değerleri değiştirir ve nominal özellikleri ikili olanlara dönüştürür.

3.5.3. LAZY

3.5.3.1. K-En Yakın Komşu Sınıflandırıcı (K-Nearest Neighbours Classifier)

K-en yakın komşular sınıflandırıcısıdır. Çapraz doğrulamaya dayalı olarak uygun K değerini seçebilir. Mesafe ağırlıklandırması da yapabilir.

3.5.3.2. Kyıldız (Kstar –K)*

$K*$, örnek tabanlı bir sınıflandırıcıdır, yani bir test örneğinin sınıfı, bazı benzerlik işlevleriyle belirlendiği üzere, ona benzer eğitim örneklerinin sınıfına dayanır. Entropi tabanlı bir mesafe işlevi kullanması açısından diğer örnek tabanlı öğrencilerden farklıdır.

3.5.3.3. Yerel Ağırlıklı Öğrenme (Locally Weighted Learning)

Yerel ağırlıklı bir öğrenme yöntemidir. Örnek ağırlıkları atamak için örnek tabanlı bir algoritma kullanır ve bunlar daha sonra belirli bir WeightedInstancesHandler sınıfı tarafından kullanılır. Sınıflandırma (örn. Naive Bayes kullanarak) veya regresyon (örn. Doğrusal regresyon kullanarak) yapabilir.

3.5.4. META

3.5.4.1. İteratif Sınıflandırıcı Optimize Edici (Iterative Classifier Optimizer)

Çapraz doğrulama veya yüzde bölünmüş değerlendirme kullanarak belirli yinelemeli sınıflandırıcının yineleme sayısını optimize eder.

3.5.4.2. Adaptif Yükseltme (AdaBoostM1)

Adaboost M1 yöntemini kullanan nominal bir sınıflandırıcı yöntemidir. Sınıflandırma yapmada kullanılan algoritmaların, daha iyi bir doğruluk oranı elde etmek için performanslarını arttırmayı amaçlamıştır. Yalnızca nominal sınıf sorunları çözülebilir. Çoğunlukla performansı önemli ölçüde iyileştirir, ancak bazen aşırı uyum sağlar.

3.5.4.3. Öznitelik Seçici Sınıflandırıcı (Attribute Selected Classifier)

Eğitim ve test verilerinin boyutu, bir sınıflandırıcıya aktarılmadan önce nitelik seçimi ile azaltılır.

3.5.4.4. Torbalama (Bagging)

Varyansı azaltmak için bir torbalama sınıflandırıcısıdır.

3.5.4.5. Regresyon Yoluyla Sınıflandırma (Classification via Regression)

Regresyon yöntemlerini kullanarak sınıflandırma yapmak için kullanılan bir sınıflandırıcıdır. Sınıf ikilileştirilir ve her sınıf değeri için bir regresyon modeli oluşturulur.

3.5.4.6. Maliyete Duyarlı Sınıflandırıcı (Cost Sensitive Classifier)

Temel sınıflandırıcısını maliyete duyarlı hale getiren bir meta sınıflandırıcıdır. Maliyet duyarlılığını tanıtmak için iki yöntem kullanılabilir: Her sınıfa atanan toplam maliyete göre

eğitim örneklerini yeniden ağırlıklandırma veya minimum beklenen yanlış sınıflandırma maliyetiyle sınıfı tahmin etmek (en olası sınıf yerine). Temel sınıflandırıcının olasılık tahminlerini iyileştirmek için genellikle Torbalı sınıflandırıcı kullanılarak performans iyileştirilebilir.

3.5.4.7. Çapraz Doğrulama Parametre Seçimi (Cross Validation Parameter Selection)

Herhangi bir sınıflandırıcı için çapraz doğrulama ile parametre seçimi gerçekleştirme sınıfıdır.

3.5.4.8. Filtrelenmiş Sınıflandırıcı (Filtered Classifier)

Rasgele bir filtreden geçirilen veriler üzerinde rastgele bir sınıflandırıcı çalıştırma sınıfıdır. Sınıflandırıcı gibi, filtrenin yapısı da özellikle eğitim verilerine dayanır ve test örnekleri, yapılarını değiştirmeden filtre tarafından işlenir. Eşit olmayan örnek ağırlıkları veya öznitelik ağırlıkları varsa ve filtre veya sınıflandırıcı bunlarla başa çıkamıyorsa, örnekler ve / veya öznitelikler, filtreye veya sınıflandırıcıya (uygun olduğu şekilde) iletilmeden önce ağırlıklara dayalı olarak değiştirme ile yeniden örneklenir.

3.5.4.9. Logi Boost

İlave lojistik regresyon gerçekleştirmek için bir sınıflandırıcıdır. Bu sınıf, temel öğrenci olarak bir regresyon şeması kullanarak sınıflandırma gerçekleştirir ve çok sınıflı problemleri çözebilir.

3.5.4.10. Çok Sınıflı Sınıflandırıcı (Multi Class Classifier)

2 sınıflı sınıflandırıcılarla çok sınıflı veri setlerini işlemek için bir meta sınıflandırıcısıdır. Bu sınıflandırıcı ayrıca, daha yüksek doğruluk için hata düzeltme çıktı kodlarını uygulayabilir.

3.5.4.11. Çok Sınıflı Sınıflandırıcı Güncellenebilir (Multi Class Classifier Updateable)

İki sınıflı sınıflandırıcılarla çok sınıflı veri setlerini işlemek için bir meta sınıflandırıcısıdır. Bu sınıflandırıcı ayrıca, daha yüksek doğruluk için hata düzeltme çıktı kodlarını uygulayabilir. Temel sınıflandırıcı, güncellenebilir bir sınıflandırıcı olmalıdır.

3.5.4.12. Çoklu Şema (Multi Scheme)

Eğitim verileri veya eğitim verilerindeki performans üzerinde çapraz doğrulama yöntemi kullanarak birkaç sınıflandırıcı arasından bir sınıflandırıcı seçmek için kullanılan bir

sınıflandırıcıdır. Performans, yüzde doğru (sınıflandırma) veya ortalama kare hataya (regresyon) göre ölçülür.

3.5.4.13. Rastgele Komite (Random Committee)

Randomize edilebilir temel sınıflandırıcılar topluluğu oluşturmak için bir sınıflandırıcıdır. Her temel sınıflandırıcı, farklı bir rasgele sayı çekirdeği kullanılarak oluşturulur (ancak aynı verilere dayalı olarak). Nihaî tahmin, bireysel temel sınıflandırıcılar tarafından üretilen tahminlerin düz bir ortalamasıdır.

3.5.4.14. Randomize Edilebilir Filtreli Sınıflandırıcı (Randomizable Filtered Classifier)

FilteredClassifier'ın, modeli randomize edilebilir bir filtreyle, daha spesifik olarak, RandomProjection ve temel sınıflandırıcı olarak IBk ile başlatan basit bir varyantıdır. Bunun dışında ve iki temel şemadan en az birinin Randomizable arayüzü uyguladığını kontrol ederek, Randomizable'ı da uygulayan FilteredClassifier ile tam olarak aynı işlevselliği uygular.

3.5.4.15. Rastgele Alt Boşluk (Random Sub Space)

Bu yöntem, eğitim verilerinde en yüksek doğruluğu koruyan ve karmaşıklık arttıkça genelleme doğruluğunu iyileştiren karar ağacı tabanlı bir sınıflandırıcıdır. Sınıflandırıcı, özellik vektörünün bileşenlerinin alt kümelerinin sözde rastgele seçilmesiyle sistematik olarak oluşturulmuş birden çok ağaçtan, yani rastgele seçilen alt uzaylarda oluşturulan ağaçlardan oluşur.

3.5.4.16. İstifleme (Stacking)

Yığınlama yöntemini kullanarak birkaç sınıflandırıcıyı birleştirir. Sınıflandırma veya regresyon yapabilir.

3.5.4.17. Oylama (Vote)

Sınıflandırma için farklı olasılık tahmin kombinasyonları mevcuttur.

3.5.4.18. Ağırlıklı Örnek İşleyici Sarmalayıcı (Weighted Instances Handler Wrappler)

Ağırlıklı örnek desteğini etkinleştirmek için herhangi bir sınıflandırıcı etrafında genel bir sarmalayıcıdır. Temel sınıflandırıcı eka.core.WeightedInstancesHandler arabirimini

uygulamıyorsa ve 1.0'dan farklı örnek ağırlıkları varsa, ağırlıklarla yeniden örneklemeyi kullanır. Varsayılan olarak, örnek ağırlıklarını işleyebiliyorsa eğitim verileri temel sınıflandırıcıya aktarılır. Bununla birlikte, yeniden örnekleme kullanımını ağırlıklarla da zorlamak mümkündür.

3.5.5. MISC – Giriş Eşlemeli Sınıflandırıcı (Input Mapped Classifier)

Bir sınıflandırıcının birlikte oluşturulduğu eğitim verileri ile gelen test örneklerinin yapısı arasında bir eşleme oluşturarak uyumsuz eğitim ve test verilerini ele alan sarmalayıcı sınıflandırıcısıdır. Gelen örneklerde bulunmayan model öznitelikleri eksik değerleri alır, dolayısıyla sınıflandırıcının daha önce görmediği gelen nominal öznitelik değerleri de alır. Yeni bir sınıflandırıcı eğitilebilir veya bir dosyadan mevcut bir sınıflandırıcı yüklenebilir.

3.5.6. KURALLAR (RULES)

3.5.6.1. Karar Ağacı (Desicion Table)

Basit bir karar tablosu ile çoğunluk sınıflandırıcısı oluşturmak için kullanılan bir sınıflandırıcıdır.

3.5.6.2. Tekrarlanan Artımlı Budama (Repeated Incremental Pruning)

Bu sınıf, IREP'nin optimize edilmiş bir versiyonu olarak William W. tarafından önerilen, önerme kuralı öğrenen, Hata Azaltma Üretmek için Tekrarlanan Artımlı Budama'yı (Repeated Incremental Pruning - RIPPER) uygular.

3.5.6.3. OneR

OneR sınıflandırıcı oluşturma ve kullanma sınıfı; başka bir deyişle, tahmin için minimum hata özniteliğini kullanarak sayısal öznitelikleri ayırır.

3.5.6.4. Part

Part karar listesi oluşturma sınıfıdır. Böl ve fethet yöntemini kullanır. Her yinelemede kısmî bir C4.5 karar ağacı oluşturur ve "en iyi" yaprağını bir kural haline getirir.

3.5.6.5. ZeroR

ZeroR sınıflandırıcı oluşturma ve kullanma sınıfıdır. Ortalamayı (sayısal bir sınıf için) veya modu (nominal bir sınıf için) tahmin eder.

3.5.7. AĞAÇ (TREE)

3.5.7.1. Karar Kütüğü (Desicion Stump)

Bir karar güdüsü oluşturma ve kullanma sınıfıdır. Genellikle bir yükseltme algoritması ile birlikte kullanılır. Regresyon (ortalama kare hatasına göre) veya sınıflandırma (entropiye dayalı) yapar. Eksik, ayrı bir değer olarak kabul edilir.

3.5.7.2. Hoefding Ağacı (Hoefding Tree)

Bir Hoefding Ağacı, dağıtım üreten örneklerin zamanla değişmediği varsayıldığında, büyük veri akışlarından öğrenme yeteneğine sahip, her an artan bir karar ağacı indüksiyon algoritmasıdır. Hoefding ağaçları, küçük bir örneğin en uygun ayırma özelliğini seçmek için genellikle yeterli olabileceği gerçeğinden yararlanır. Bu fikir matematiksel olarak Hoefding sınırı tarafından desteklenir; bu, önceden belirlenmiş bir kesinlik dahilinde bazı istatistikleri tahmin etmek için gereken gözlem sayısını nicelendirir.

Hoefding Ağacı'nın diğer artımlı karar ağacı öğrenenler tarafından paylaşılmayan teorik olarak çekici bir özelliği, sağlam performans garantilerine sahip olmasıdır. Hoefding sınırını kullanarak, onun çıktısının sonsuz sayıda örnek kullanarak artımlı olmayan bir öğrenicinin çıktısının asimptotik olarak neredeyse aynı olduğunu gösterebilir.

3.5.7.3. J48

Budanmış veya budanmamış bir karar ağacı üretmek için bir sınıflandırıcıdır.

3.5.7.4. Lojistik Model Ağacı (Logistic Model Tree)

Yapraklarda lojistik regresyon işlevlerine sahip sınıflandırma ağaçları olan 'lojistik model ağaçları' oluşturmak için kullanılan bir sınıflandırıcıdır. Algoritma, ikili ve çok sınıflı hedef değişkenleri, sayısal ve nominal öznitelikleri ve eksik değerleri ele alabilir.

3.5.7.5. Rastgele Orman (Random Forest)

Rastgele ağaçlardan oluşan bir orman inşa etmek için kullanılan bir sınıflandırıcıdır.

3.5.7.6. Rastgele Ağaç (Random Tree)

Her düğümde *K* rastgele seçilen öznitelikleri dikkate alan bir ağaç oluşturmak için kullanılan bir sınıflandırıcıdır. Budama yapmaz. Ayrıca, bir uzatma kümesine (geri uyum) dayalı olarak sınıf olasılıklarının (veya regresyon durumunda hedef ortalamanın) tahminine izin verme seçeneğine de sahiptir.

3.5.7.7. Rep Ağacı (Rep Tree)

Hızlı karar ağacı öğrenicisidir. Bilgi kazanımı / varyansını kullanarak bir karar / gerileme ağacı oluşturur ve bunu daha az hata ile budama (arka plan ile) kullanarak kurur. Sayısal özellikler için değerleri yalnızca bir kez sıralar. Eksik değerler, karşılık gelen örnekleri parçalara bölerek ele alınır [60].

4. BULGULAR

Bu tezde gerçekleştirilen performans analizleri, doğruluk, ROC eğrileri ve AUC kriterleri dikkate alınarak gerçekleştirilmiştir. Geliştirme ortamı olarak. WEKA veri madenciliği aracı ve Google COLAB platformunda Python dili kullanılmıştır. Bilgisayar donanımının simülasyon sonuçlarını etkilememesi için her birinde aynı bilgisayar sistemi kullanılmıştır. Bu sistemin özellikleri aşağıdaki gibidir:

- ✓ Intel® Core i7-7700HQ, 2.8GHz İşlemci
- ✓ 16 GB RAM
- ✓ 64 Bit Windows 10 Pro İşletim Sistemi.

4.1. WEKA SİMÜLASYON SONUÇLARI

4.1.1 NASA Veri Setleri Simülasyon Sonuçları

Tez çalışmasının bu bölümünde literatürde en çok kullanılan JM1, KC1, CM1, PC1 veri setleri WEKA veri madenciliği yazılımı bünyesinde 10 katlı çapraz geçerleme kuralı ile Bayes Net, Naive Bayes, Naive Bayes Multinomial, Naive Bayes Multinominal Text, Naive Bayes Multinomial Updateable, Naive Bayes Updatable, Logistic, Multilayer Perceptron, Stochastic Gradient Descent Text, Simple Logistic, Sequential Minimal Optimization, Voted Perceptron, K-Nearest Neighbours Classifier, K*, Locally Weighted Learning, Iterative Classifier Optimizer, Adaboost1, Attribute Selected Classifier, Bagging, Classification via Regression, Cross Validation Parameter Selection, Filtered Classsifier, Logi Boost, Multi Class Classifier, Multi Class Classifier Updatable, Multi Scheme, Random Comittee, Randomizable Filtered Classifer, Random Sub Space, Stacking, Vote, Weighted Instances Handler Wrapper, Input Mapped Classifier, Desicion Table, Repeated Incremental Pruning, OneR, Part, ZeroR, Desicion Stump, Hoeffding Tree, J48, Random Forest, Random Tree, Rep Tree algoritmalarıyla gerçeklenmiş, doğruluk sonuçları hesaplanmış ve tablo halinde (Tablo 4.1) sunulmuştur.

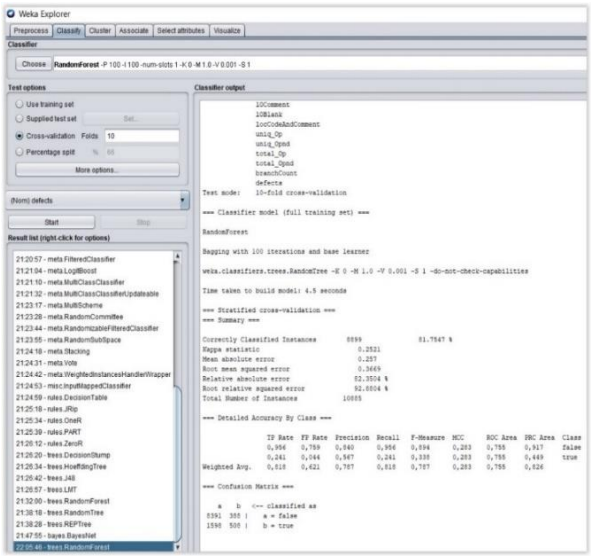
Tablo 4.1'e bakıldığında Random Forest, Sequential Minimal Optimization, Bagging, Simple Logistic algoritmalarının %85 üzerinde bir başarı oranıyla tüm veri setlerinde en başarılı sonuçları verdiği, Voted Perceptron, Naive Bayes Multinominal Text, Stochastic Gradient Descent Text, OneR algoritmalarının ise doğruluk oranının tüm veri setlerinde oldukça düşük olduğu görülmektedir. Her veri seti için oldukça fazla sayıda algoritma ile işlem yapıldığından

her algoritmaya dair simülasyon sonuçları vermek yerine her veri setinde en yüksek doğruluk değerini veren 10 algoritmanın simülasyon sonuçlarının verilmesi tercih edilmiştir. JM1 veri seti için en yüksek 10 doğruluk oranı Random Forest, Random Sub Space, Logistic, Logistic Model Tree, Classification Via Regression, Bagging, Filtered Classifier, Repeated Incremental Pruning, Random Committee ve Multilayer Perceptron algoritmalarında elde edilmiştir ve Şekil 4.1 ve Şekil 4.2.'de bu algoritmaların simülasyon sonuçları, Şekil 4.3 ve Şekil 4.4.'de ise ROC eğrileri verilmiştir, KC1 veri seti için en yüksek 10 doğruluk oranı Random Forest, Bagging, Multilayer Perceptron, Simple Logistic, Logistic, Classification Via Regression, Logi Boost, Naive Bayes Multinomial, Naive Bayes Multinomial Updateable, Iterative Classifier Optimizer algoritmalarında elde edilmiştir ve Şekil 4.5 ve Şekil 4.6 bu algoritmaların simülasyon sonuçları, Şekil 4.7 ve Şekil 4.8.'de ROC eğrileri verilmiştir, CM1 veri seti için en yüksek 10 doğruluk oranı Naive Bayes Multinomial Text, Filtered Classifier, Adaboost1, Cross Validation Parameter Selection, Multi Scheme, Random Sub Space, Stacking, Vote, Voted Perceptron Stochastic Gradient Descent Text algoritmalarında elde edilmiştir ve Şekil 4.9 – Şekil 4.10.'da bu algoritmaların simülasyon sonuçları ve Şekil 4.11 ve 4.12.'de ROC eğrileri verilmiştir, PC1 veri seti için Bagging, Random Sub Space, Random Forest, Part, Rep Tree, Random Committe, Multilayer Perceptron, Filtered Classifier, Attribute Selected Classifier, Repeated Incremental Pruning algoritmalarında elde edilmiştir ve Şekil 4.13 ve Şekil 4.14'de bu algoritmaların simülasyon sonuçları, Şekil 4.15 ve Şekil 4.16'da ise ROC eğrileri verilmiştir.

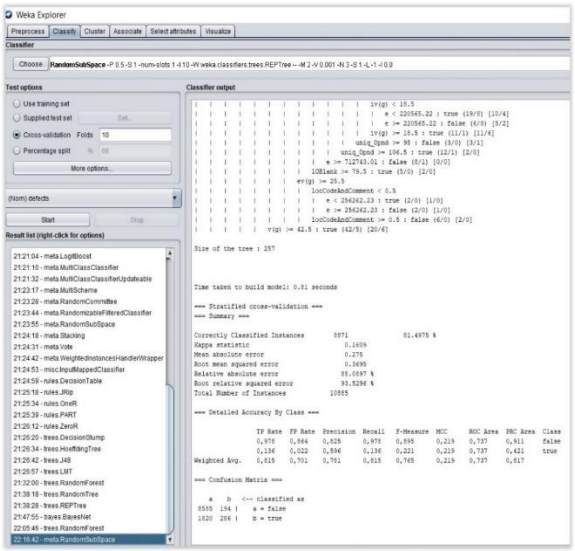
Tablo 4.1: WEKA NASA Veri Seti Doğruluk Oranları

Algoritma	JM1 (%)	KC1 (%)	CM1 (%)	PC1 (%)
BAYES				
Bayes Net	68,05	69,89	64,65	74,39
Naive Bayes	80,42	82,36	85,34	89,17
Naive Bayes Multinomial		85,34	70,68	90,53
Naive Bayes Multinominal Text	80,65	84,54	90,16	93,05
Naive Bayes Multinomial Updateable		85,34	70,68	90,53
Naive Bayes Updatable	80,42	82,36	85,34	89,17
FUNCTIONS				
Logistic	81,35	85,68	88,35	92,42
Multilayer Perceptron	80,95	85,91	87,55	93,59
Stochastic Gradient Descent	80,77	85,20	89,55	93,05
Stochastic Gradient Descent Text	80,65	84,54	90,16	93,05
Simple Logistic	81,12	85,72	89,15	92,60
Sequential Minimal Optimization	80,72	84,77	89,55	92,96

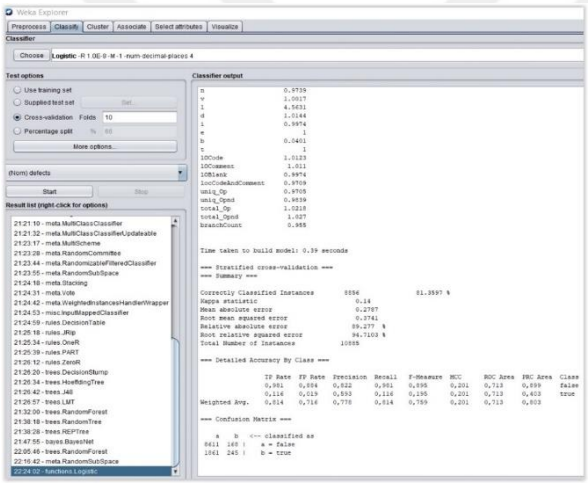
Algoritma	JM1 (%)	KC1 (%)	CM1 (%)	PC1 (%)
Voted Perceptron	52,21	83,73	90,16	92,60
LAZY				
K-Nearest Neighbours Classifier	76,97	84,40	84,73	92,06
K*	78,56	83,97	87,14	91,79
Locally Weighted Learning	80,65	71,72	89,75	93,23
META				
Iterative Classifier Optimizer	80,89	85,25	89,15	93,05
Adaboost1	80,79	84,96	90,16	93,05
Attribute Selected Classifier	80,86	84,30	89,35	93,41
Bagging	81,19	86,01	89,75	94,13
Classification via Regression	81,24	85,58	89,35	93,14
Cross Validation Parameter Selection	80,65	84,54	90,16	93,05
Filtered Classsifier	81,12	84,87	90,16	93,50
Logi Boost	80,89	85,39	88,95	93,14
Multi Class Classifier	81,35	85,68	88,35	92,42
Multi Class Classifier Updatable	80,77	85,20	89,55	93,05
Multi Scheme	80,65	84,54	90,16	93,05
Random Comittee	81,03	85,39	87,75	93,59
Randomizable Filtered Classifer	75,26	81,31	85,54	89,45
Random Sub Space	81,49	85,49	90,16	93,86
Stacking	80,65	84,54	90,16	93,05
Vote	80,65	84,54	90,16	93,05
Weighted Instances Handler Wrapper	80,65	84,54	90,16	93,05
MISC				
Input Mapped Classifier	80,65	84,54	90,16	93,05
RULES				
Desicion Table	80,90	84,87	89,15	92,87
Repeated Incremental Pruning	81,04	84,54	89,35	93,32
OneR	79,39	83,68	88,35	92,87
Part	80,74	84,82	88,75	93,68
ZeroR	80,65	84,54	90,16	93,05
TREE				
Desicion Stump	80,65	84,54	90,16	93,05
Hoeffding Tree	80,71	84,54	90,16	93,05
J48	79,50	84,54	87,95	93,32
LMT	81,24	85,72	89,15	92,42
Random Forest	81,75	86,67	88,75	93,68
Random Tree	75,47	82,69	84,33	91,07
Rep Tree	80,67	85,11	89,15	93,59
ORTALAMA	***81,09***	***67,45***	***82,40***	***88,43***



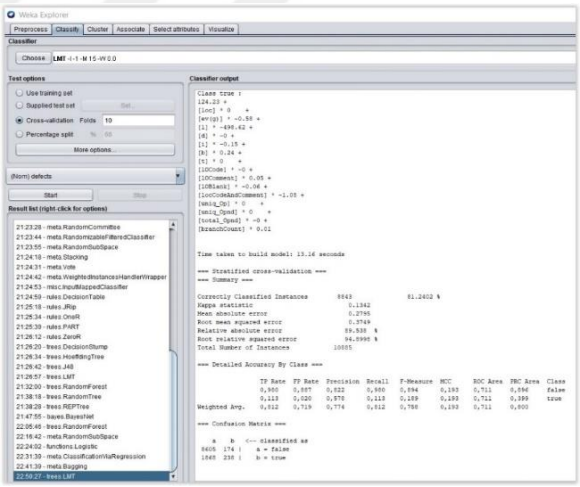
a) Random Forest Algoritması Simülasyon Sonuçları



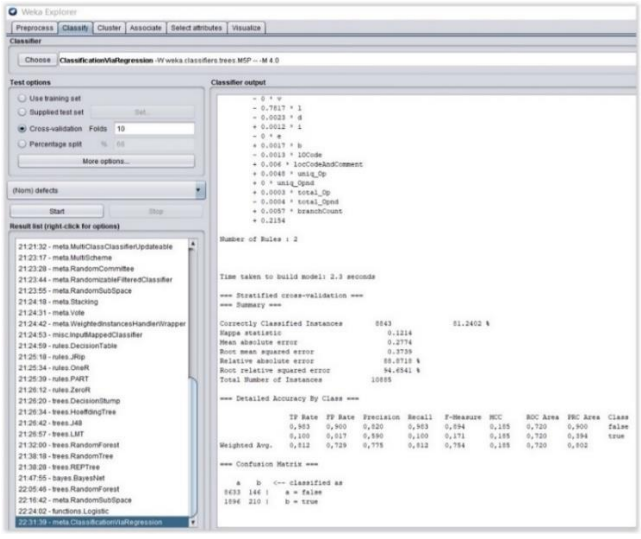
b) Random Sub Space Algoritması Simülasyon Sonuçları



c) Logistic Algoritması Simülasyon Sonuçları

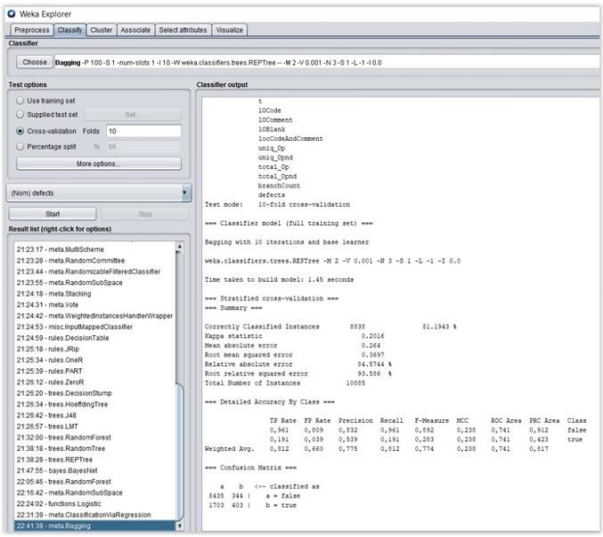


d) Logistic Model Tree Algoritması Simülasyon Sonuçları

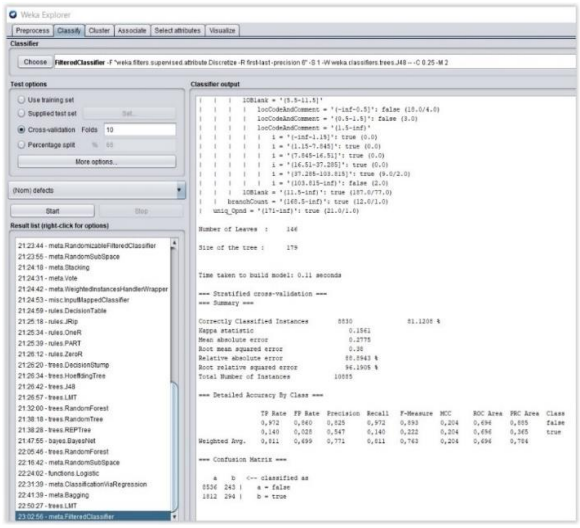


e) Classification Via Regression Algoritması Simülasyon Sonuçları

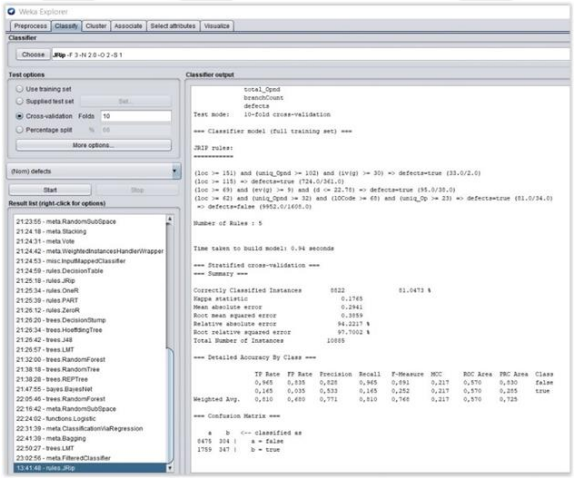
Şekil 4.1: JM1 Veri Seti Simülasyon Sonuçları



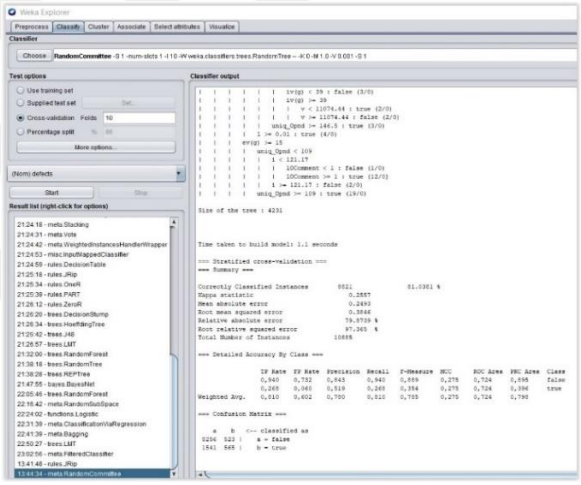
a) Bagging Algoritması Simülasyon Sonuçları



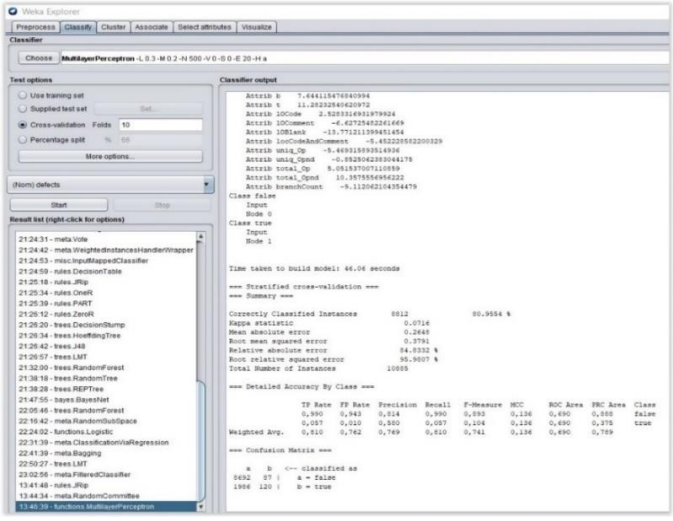
b) Filtered Classifier Algoritması Simülasyon Sonuçları



c) Repeated Incremental Pruning Algoritması Simülasyon Sonuçları

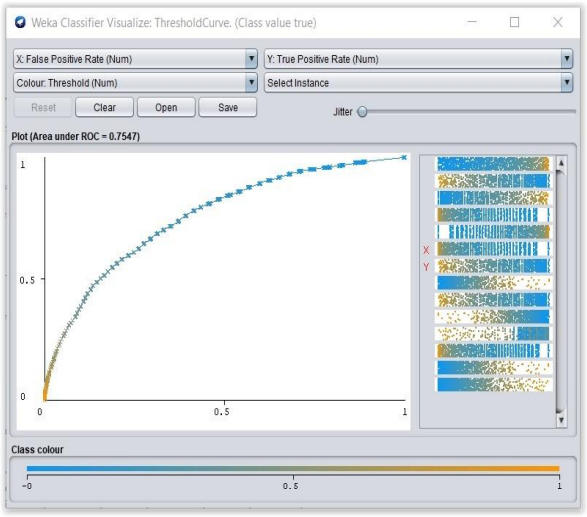


d) Random Committee Algoritması Simülasyon Sonuçları

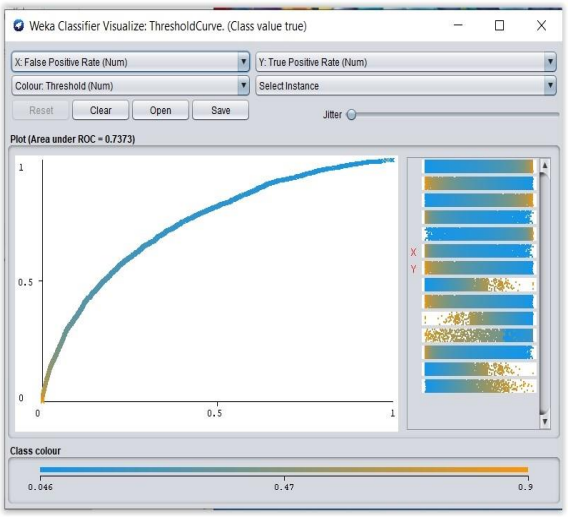


e) Multilayer Perceptron Algoritması Simülasyon Sonuçları

Şekil 4.2: JM1 Veri Seti Simülasyon Sonuçları-2



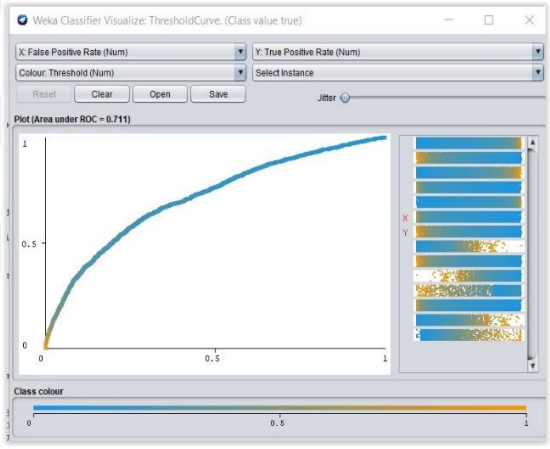
a) Random Forest Algoritması Simülasyon Sonuçları ROC Eğrisi



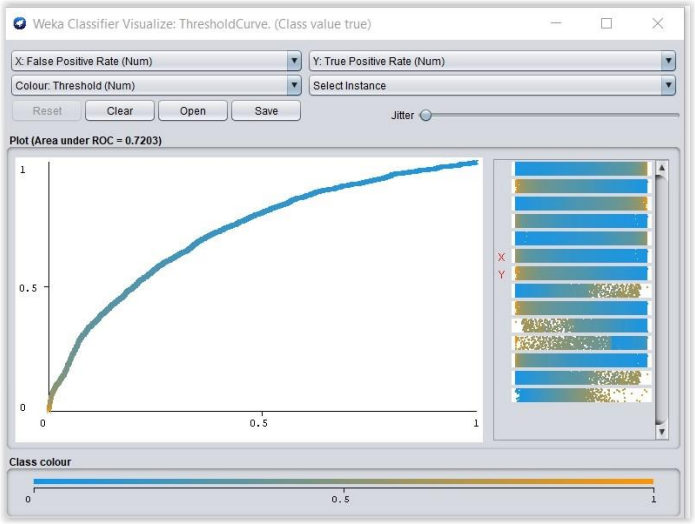
b) Random Sub Space Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Logistic Algoritması Simülasyon Sonuçları ROC Eğrisi

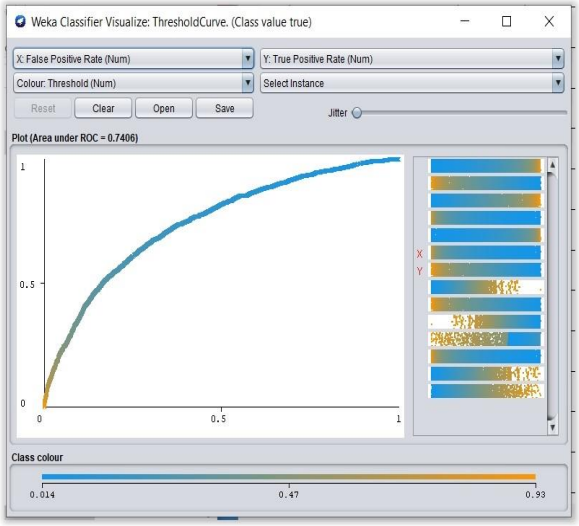


d) Logistic Model Tree Algoritması Simülasyon Sonuçları ROC Eğrisi

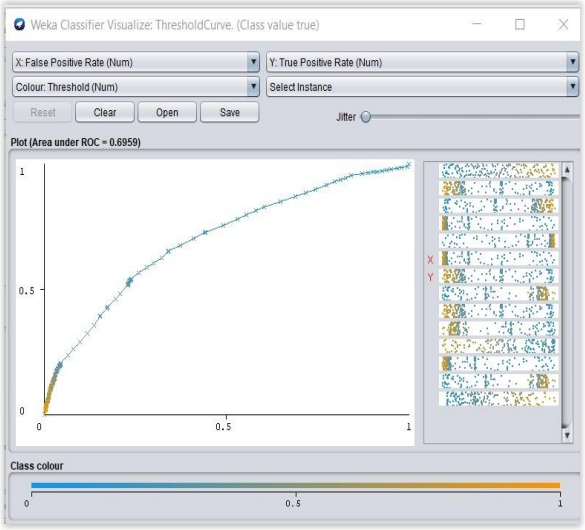


e) Classification Via Regression Algoritması Simülasyon Sonuçları ROC Eğrisi

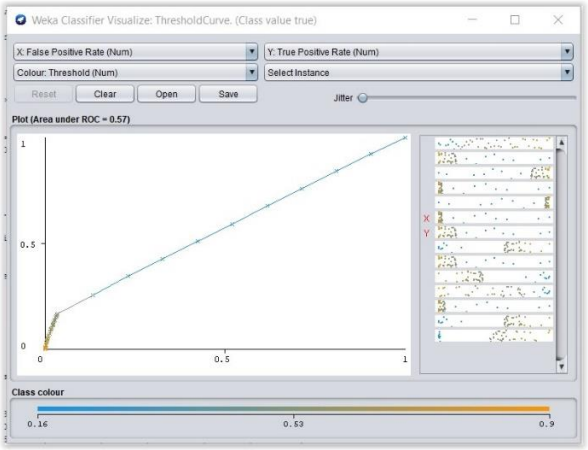
Şekil 4.3: JM1 Veri Seti Simülasyon Sonuçları ROC Eğrisi



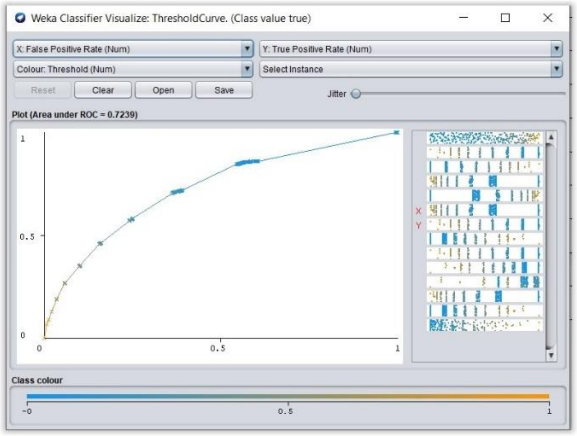
a) Bagging Algoritması Simülasyon Sonuçları ROC Eğrisi



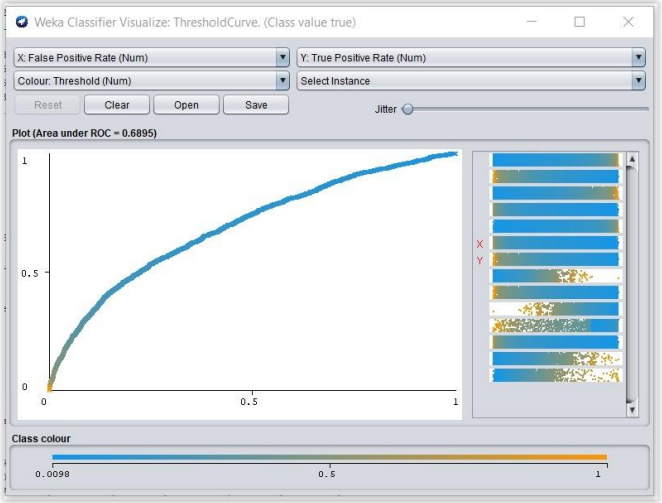
b) Filtered Classifier Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Repeated Incremental Pruning Algoritması Simülasyon Sonuçları ROC Eğrisi

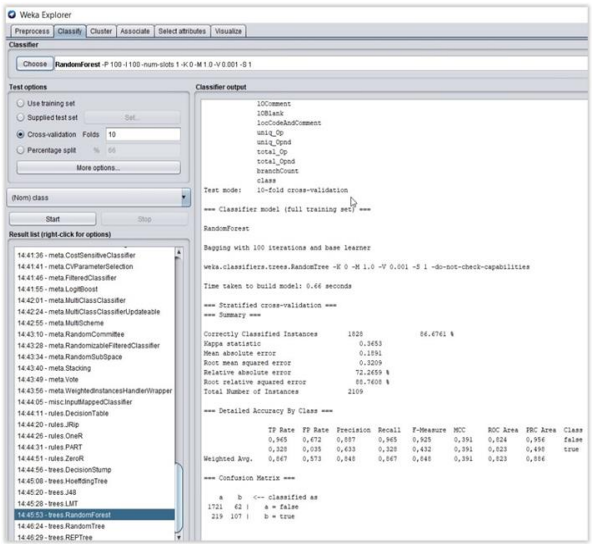


d) Random Committee Algoritması Simülasyon Sonuçları ROC Eğrisi

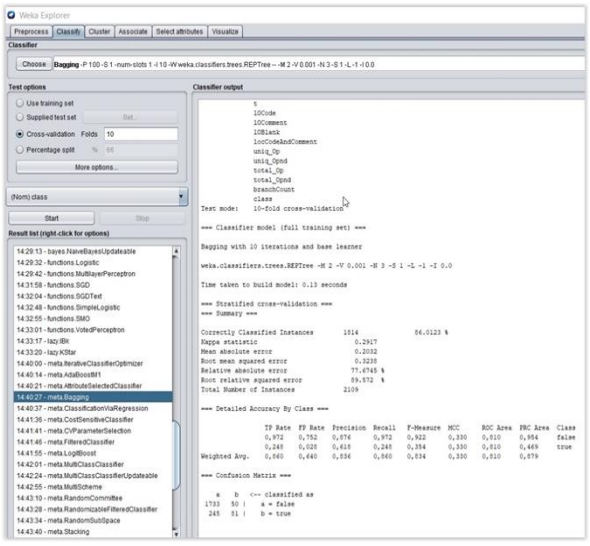


e) Multilayer Perceptron Algoritması Simülasyon Sonuçları ROC Eğrisi

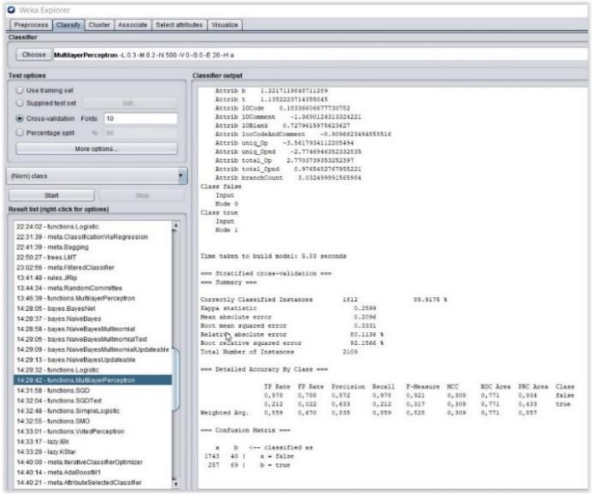
Şekil 4.4: JM1 Veri Seti Simülasyon Sonuçları ROC Eğrisi-2



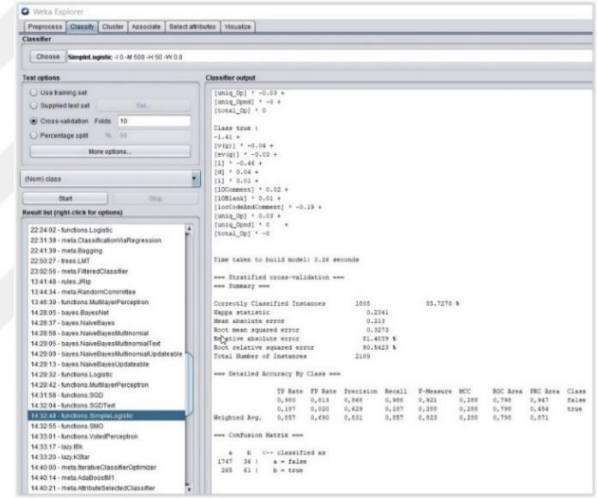
a) Random Forest Algoritması Simülasyon Sonuçları



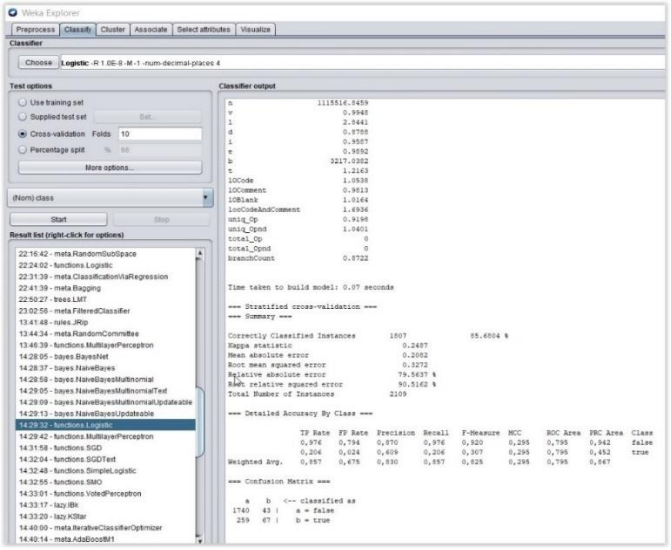
b) Bagging Algoritması Simülasyon Sonuçları



c) Multilayer Perceptron Algoritması Simülasyon Sonuçları

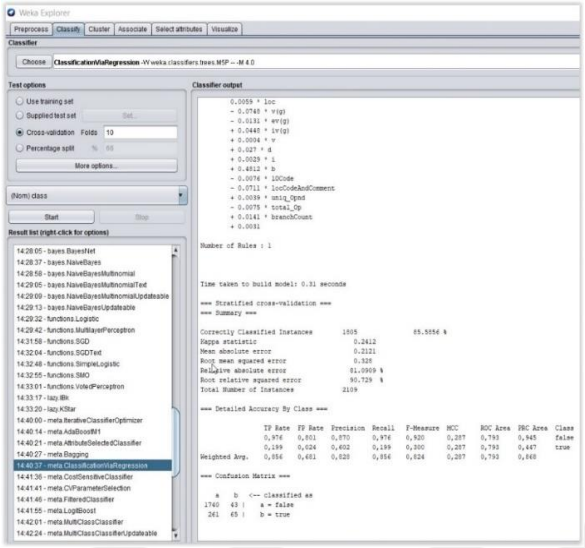


d) Simple Logistic Algoritması Simülasyon Sonuçları

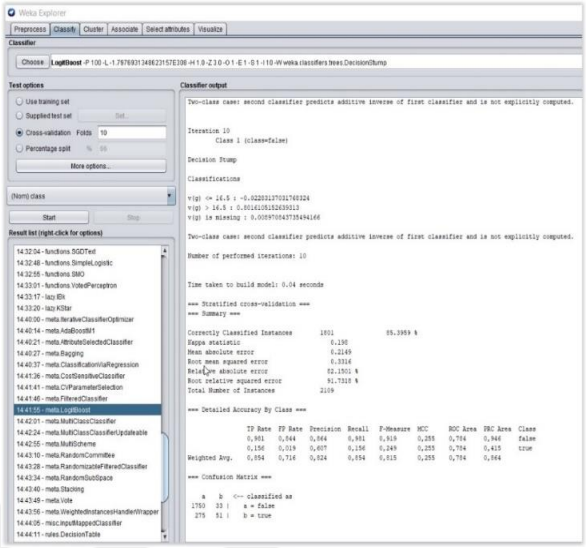


e) Logistic Algoritması Simülasyon Sonuçları

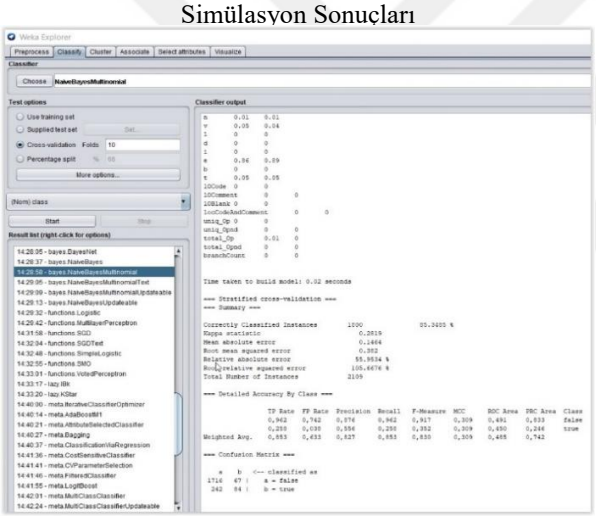
Şekil 4.5: KC1 Veri Seti Simülasyon Sonuçları



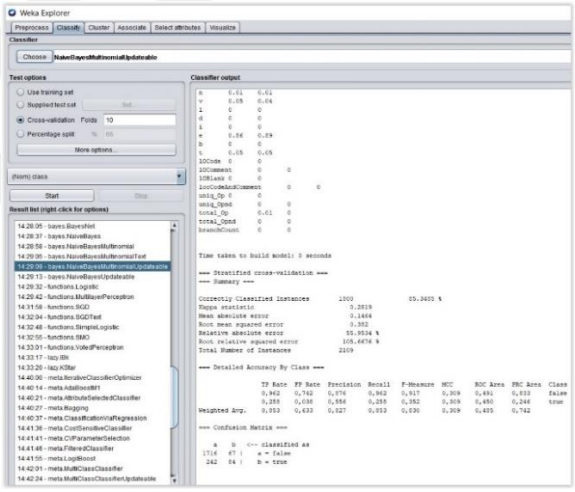
a) Classification Via Regression Algoritması Simülasyon Sonuçları



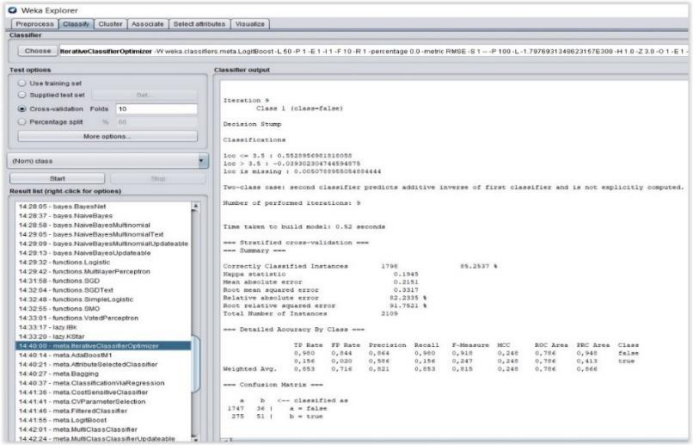
b) Logi Boost Algoritması Simülasyon Sonuçları



c) Seti Naive Bayes Multinomial Algoritması Simülasyon Sonuçları

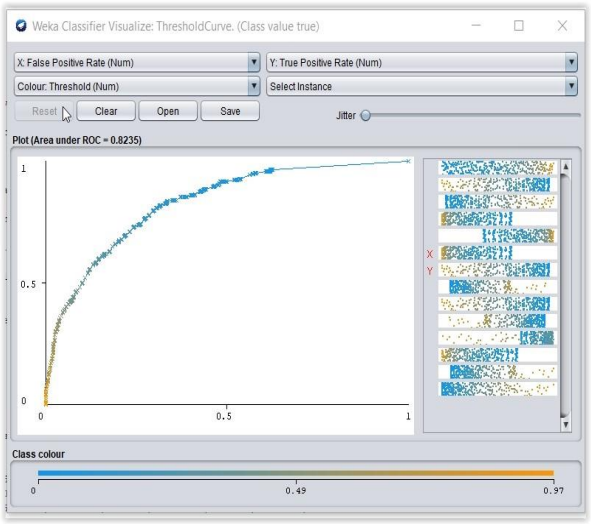


d) Naive Bayes Multinomial Updateable Algoritması Simülasyon Sonuçları

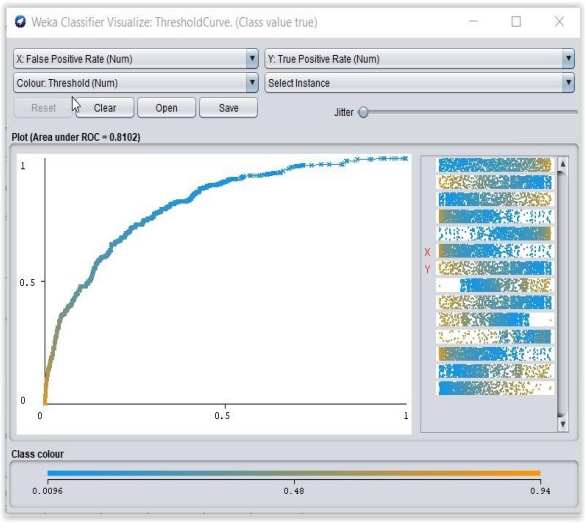


e) Iterative Classifier Optimizer Algoritması Simülasyon Sonuçları

Şekil 4.6: KC1 Veri Seti Simülasyon Sonuçları-2



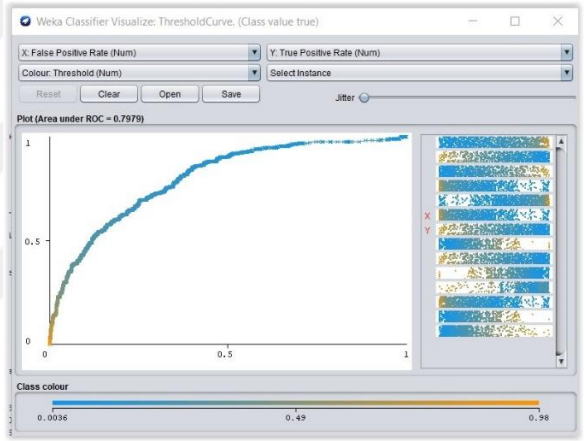
a) Random Forest Algoritması Simülasyon Sonuçları ROC Eğrisi



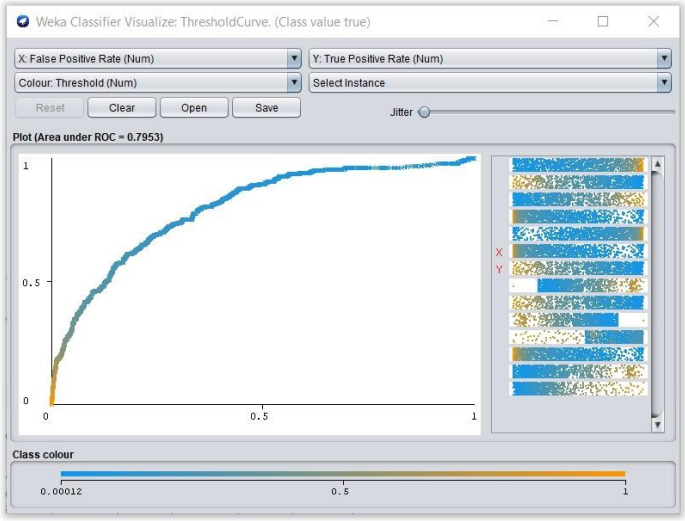
b) Bagging Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Multilayer Perceptron Algoritması Simülasyon Sonuçları ROC Eğrisi

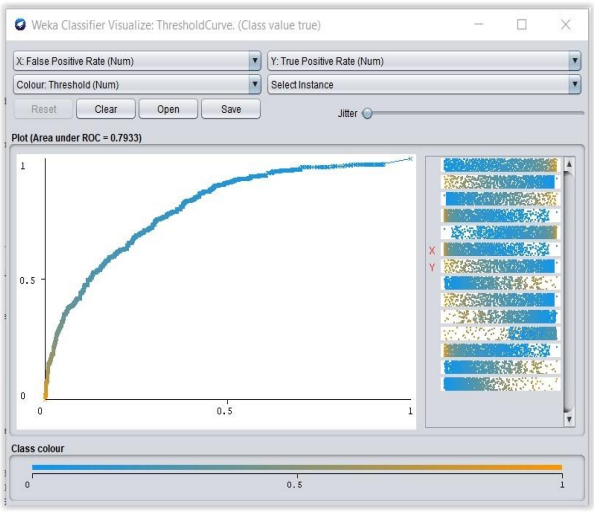


d) Simple Logistic Algoritması Simülasyon Sonuçları ROC Eğrisi

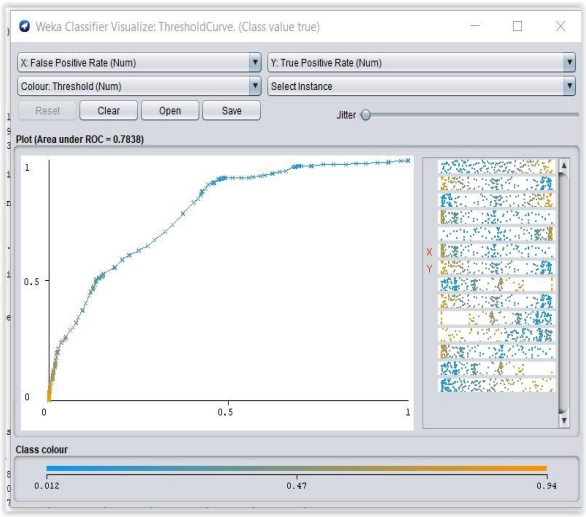


e) Logistic Algoritması Simülasyon Sonuçları ROC Eğrisi

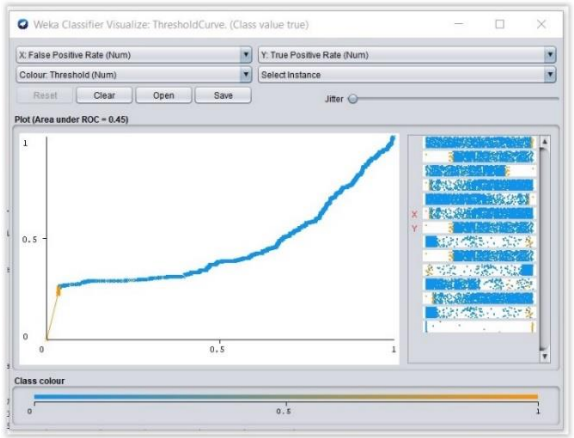
Şekil 4.7: KC1 Veri Seti Simülasyon Sonuçları ROC Eğrisi



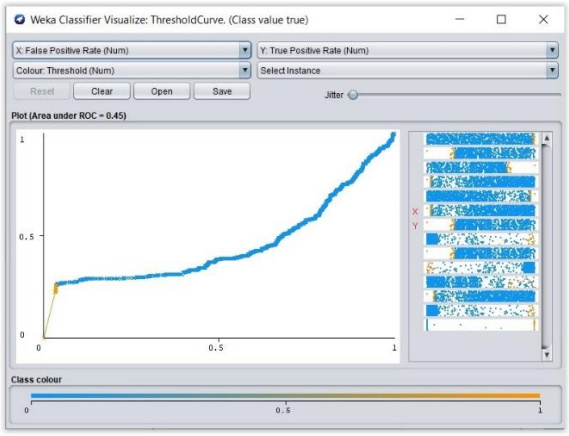
a) Classification via Regression Algoritması Simülasyon Sonuçları ROC Eğrisi



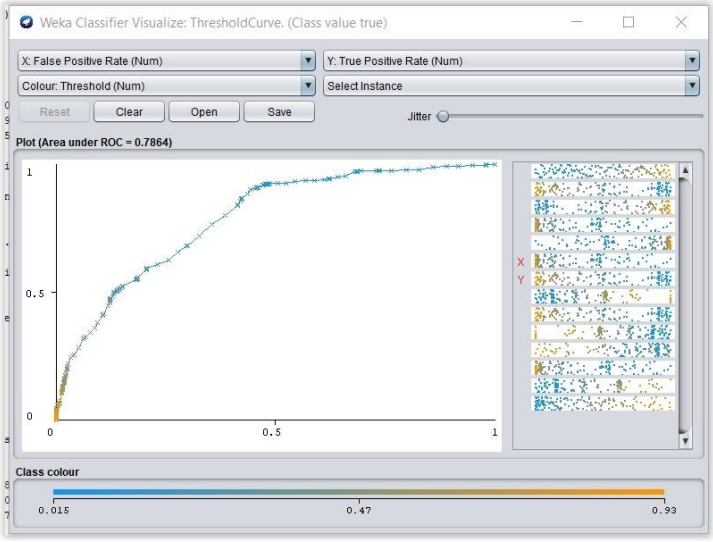
b) Logi Boost Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Naive Bayes Multinomial Algoritması Simülasyon Sonuçları ROC Eğrisi

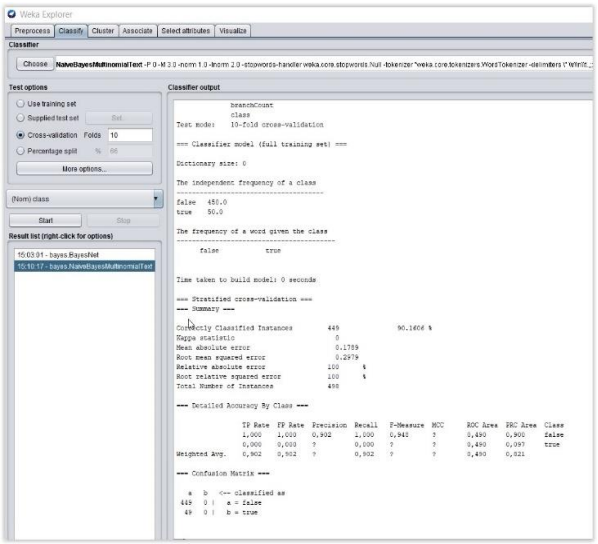


d) Naive Bayes Multinomial Updateable Algoritması Simülasyon Sonuçları ROC Eğrisi

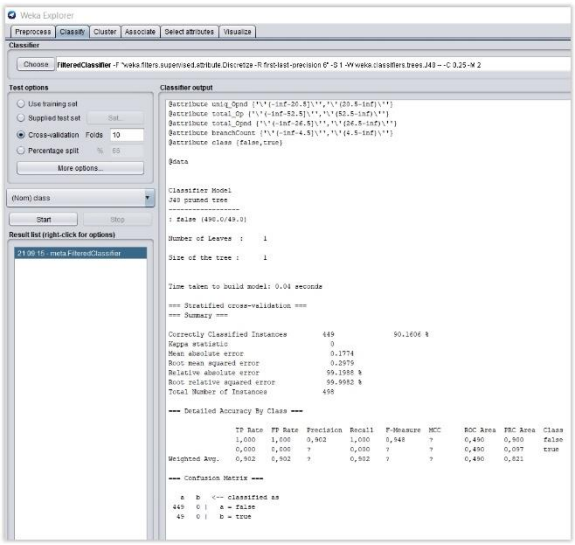


e) Iterative Classifier Optimizer Algoritması Simülasyon Sonuçları ROC Eğrisi

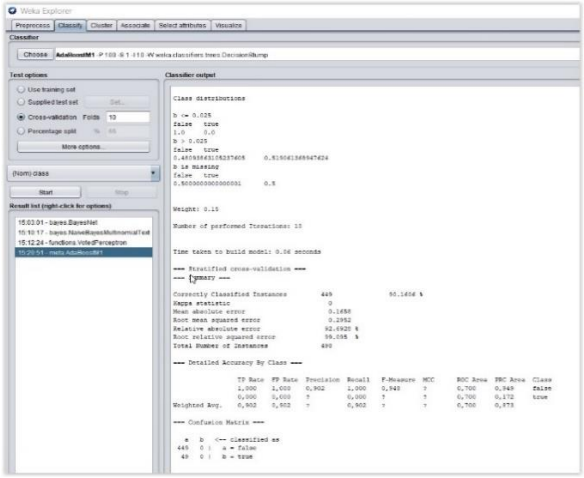
Şekil 4.8: KC1 Veri Seti Simülasyon Sonuçları ROC Eğrisi-2



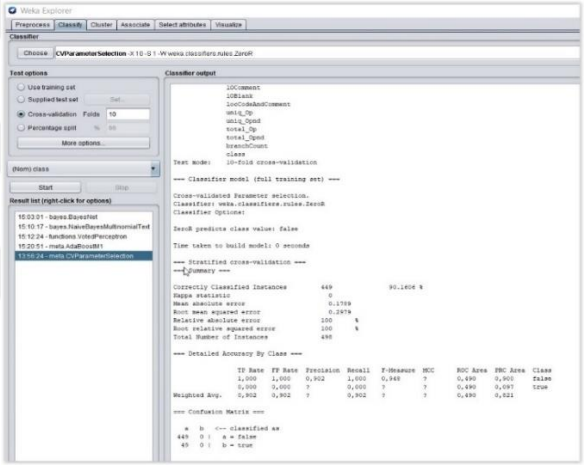
a) Naive Bayes Multinomial Text Algoritması Simülasyon Sonucları



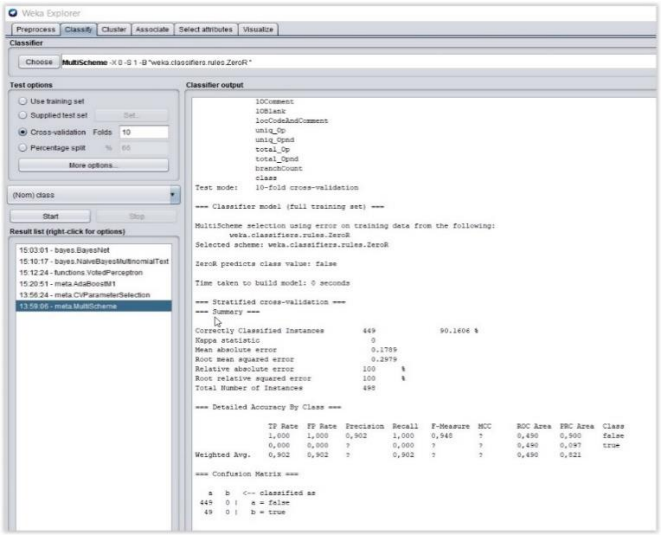
b) Filtered Classifier Algoritması Simülasyon Sonucları



c) AdaboostM1 Algoritması Simülasyon Sonuçları

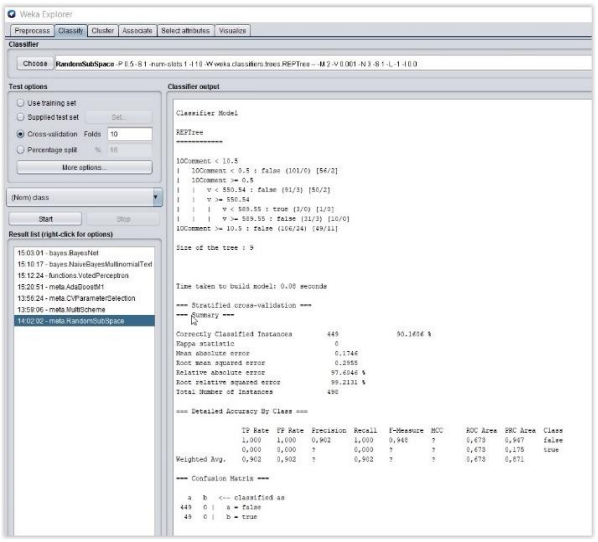


d) Cross Validation Parameter Selection Algoritması Simülasyon Sonuçları

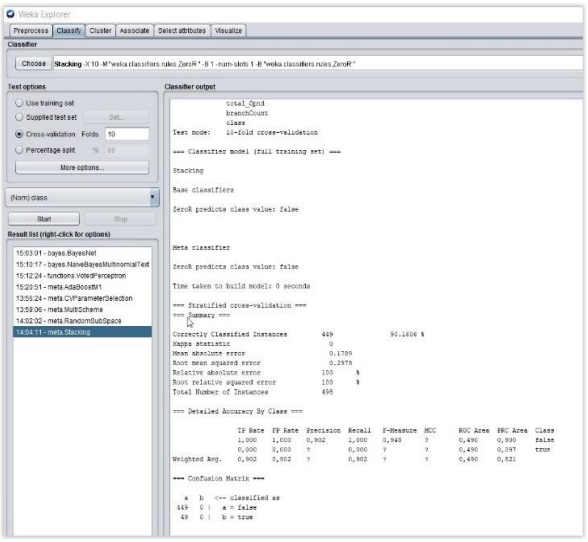


e) Multi Scheme Algoritması Simülasyon Sonuçları

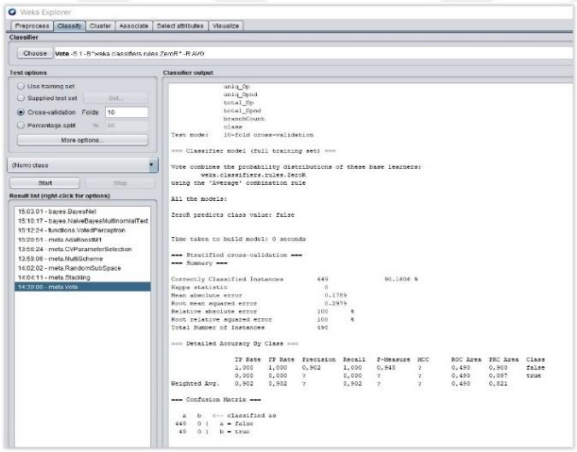
Şekil 4.9: CM1 Veri Seti Simülasyon Sonuçları



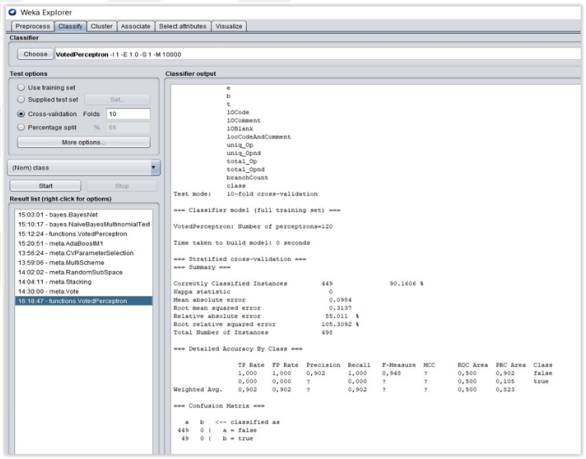
a) Random Sub Space Algoritması Simülasyon Sonuçları



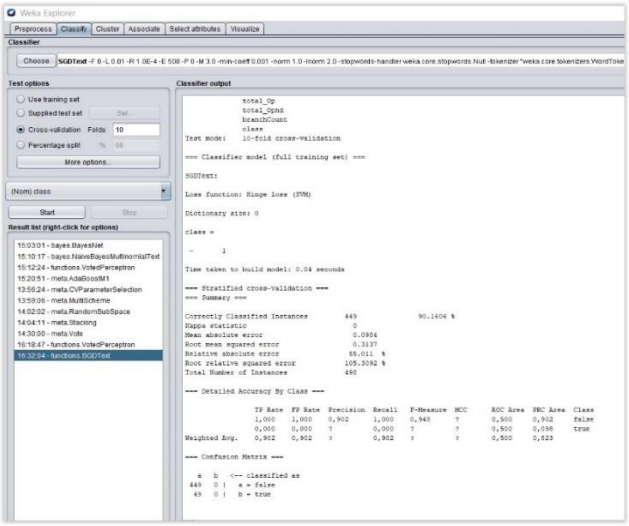
b) Stacking Algoritması Simülasyon Sonuçları



c) Vote Algoritması Simülasyon Sonuçları

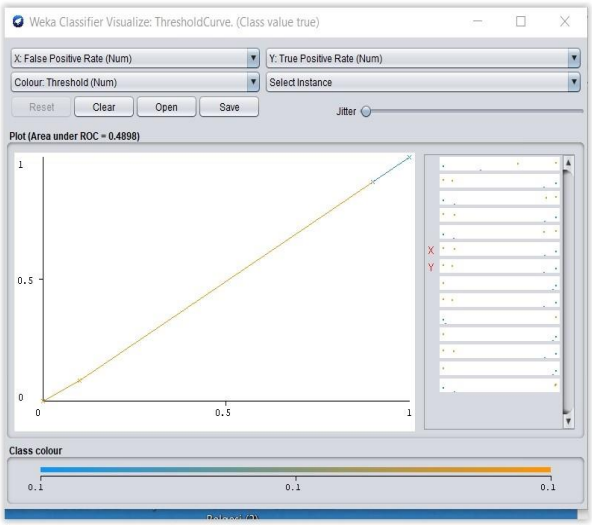


d) Voted Perceptron Algoritması Simülasyon Sonuçları

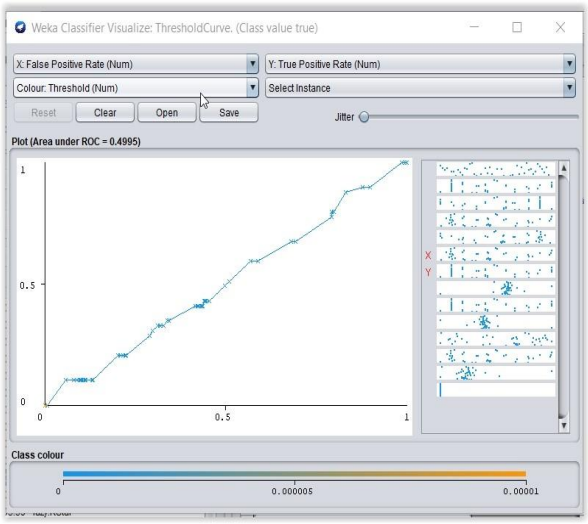


e) Stochastic Gradient Descent Text Algoritması Simülasyon Sonuçları

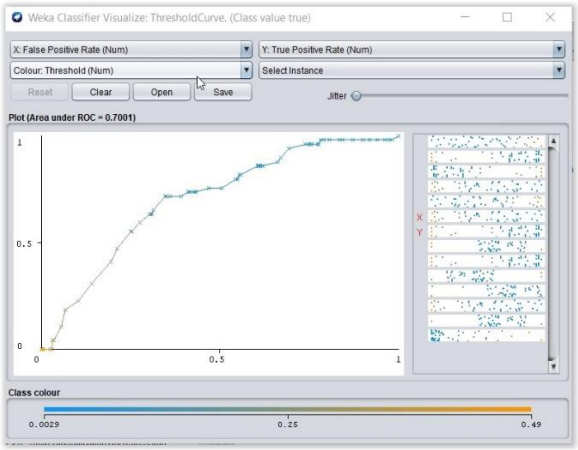
Şekil 4.10: CM1 Veri Seti Simülasyon Sonuçları-2



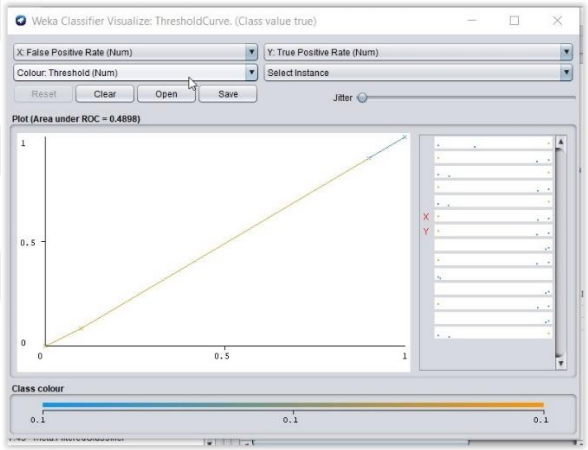
a) Naive Bayes Multinomial Text Algoritması Simülasyon Sonuçları ROC Eğrisi



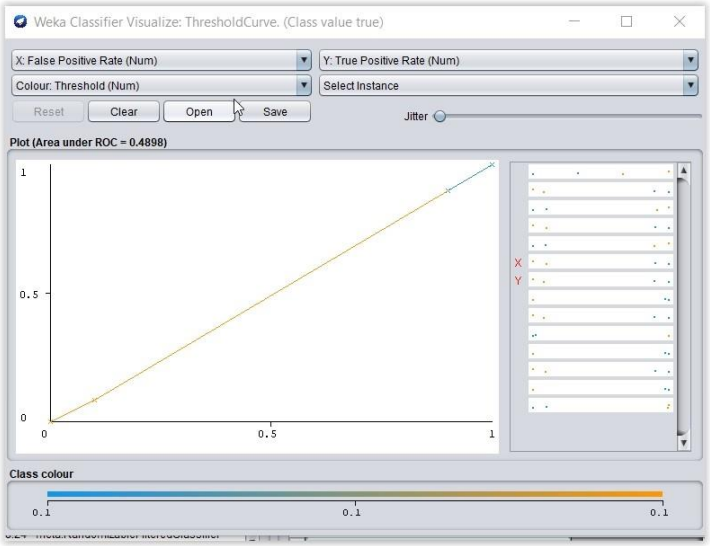
b) Voted Perceptron Algoritması Simülasyon Sonuçları ROC Eğrisi



c) AdaboostM1 Algoritması Simülasyon Sonuçları ROC Eğrisi

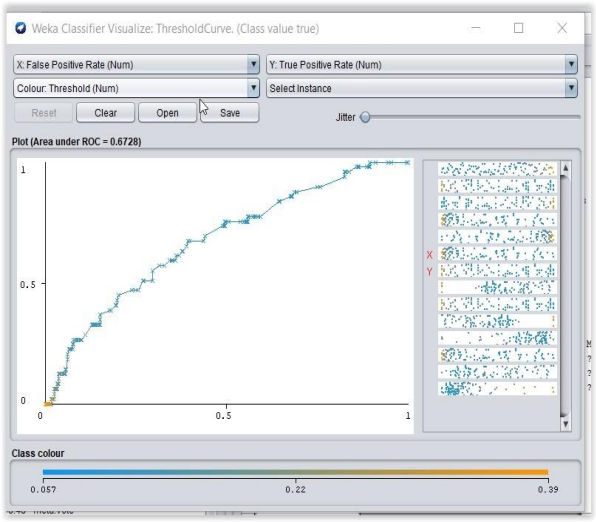


d) Cross Validation Parameter Selection Algoritması Simülasyon Sonuçları ROC Eğrisi

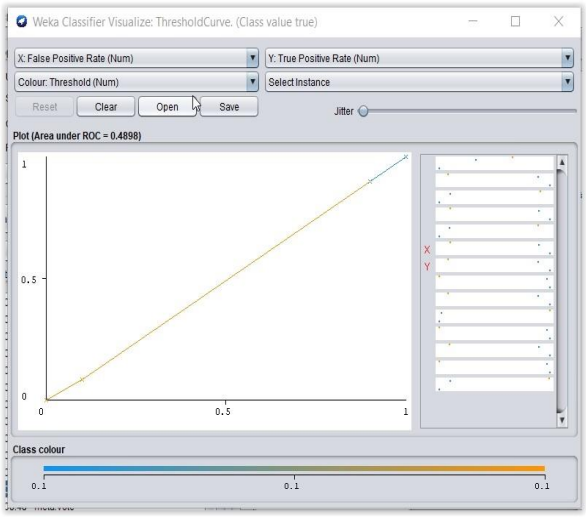


e) Multi Scheme Algoritması Simülasyon Sonuçları ROC Eğrisi

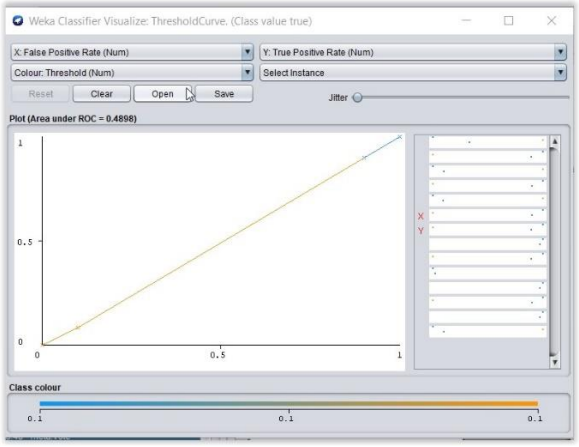
Şekil 4.11: CM1 Veri Seti Simülasyon Sonuçları ROC Eğrisi



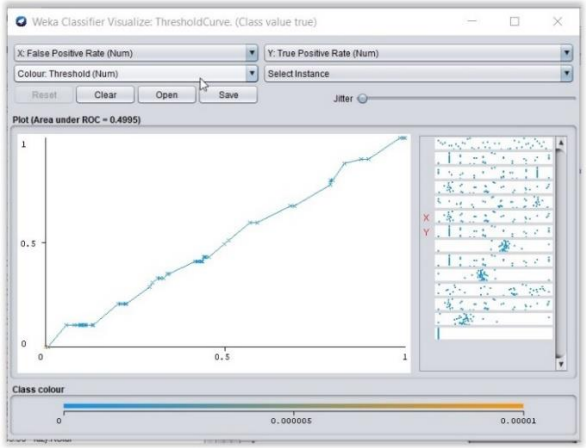
a) Random Sub Space Algoritması Simülasyon Sonuçları ROC Eğrisi



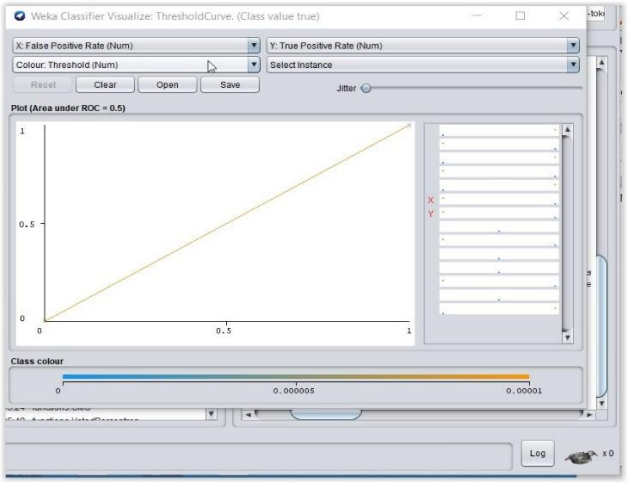
b) Stacking Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Vote Algoritması Simülasyon Sonuçları ROC Eğrisi

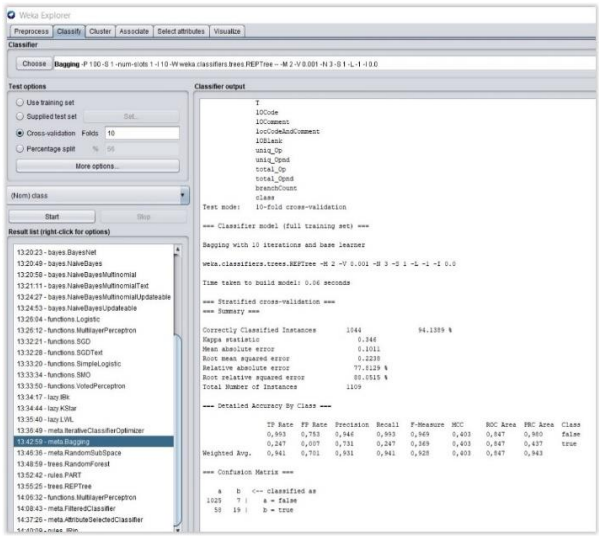


d) Voted Perceptron Algoritması Simülasyon Sonuçları ROC Eğrisi

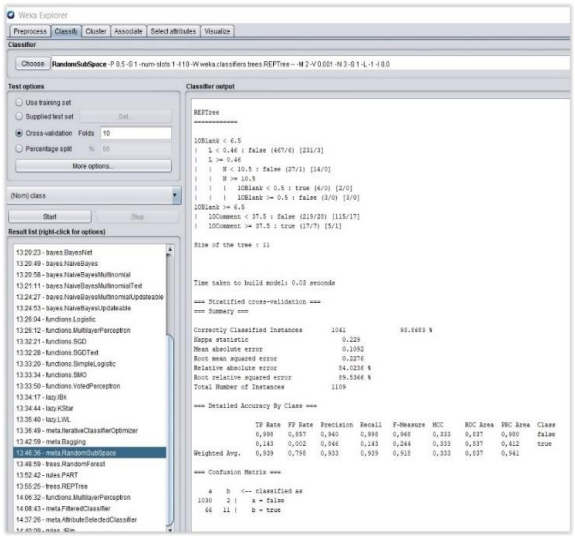


e) Stochastic Gradient Descent Text Algoritması Simülasyon Sonuçları ROC Eğrisi

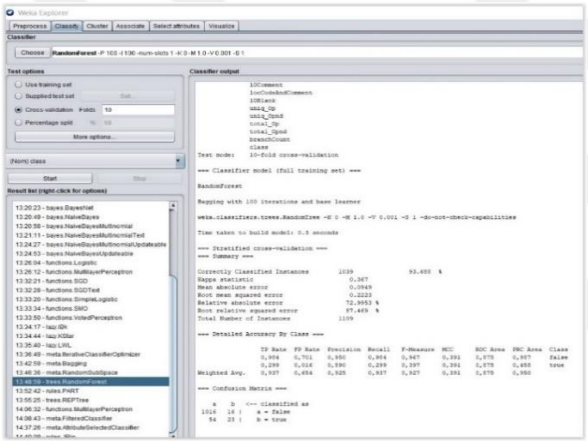
Şekil 4.12: CM1 Veri Seti Simülasyon Sonuçları ROC Eğrisi-2



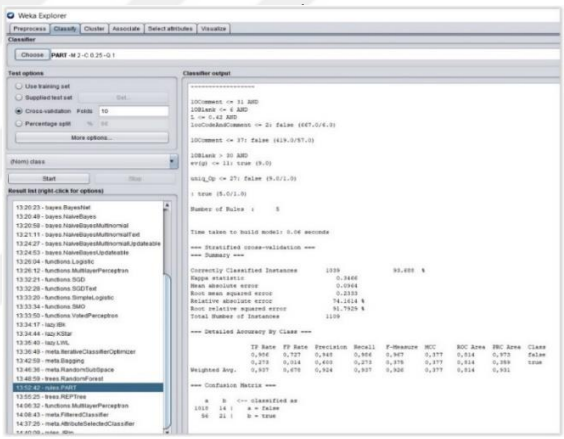
a) Bagging Algoritması Simülasyon Sonuçları



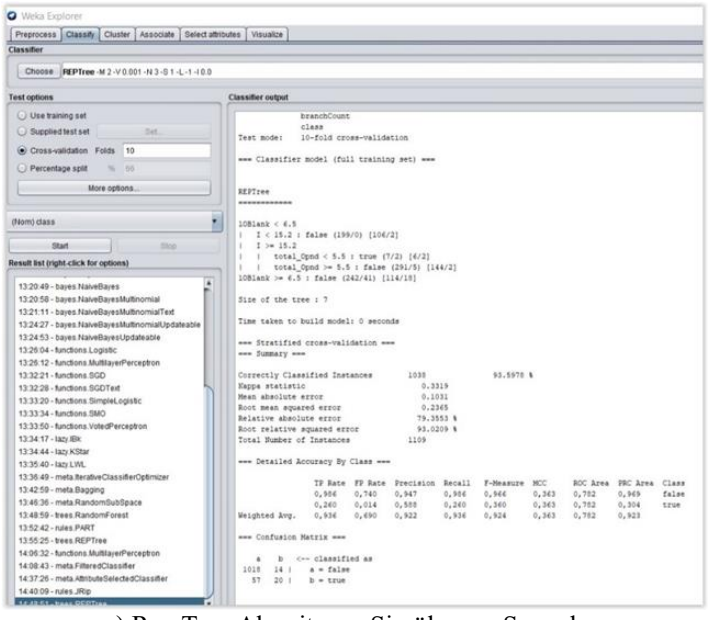
b) Random Sub Space Algoritması Simülasyon



c) Random Forest Algoritması Simülasyon Sonuçları

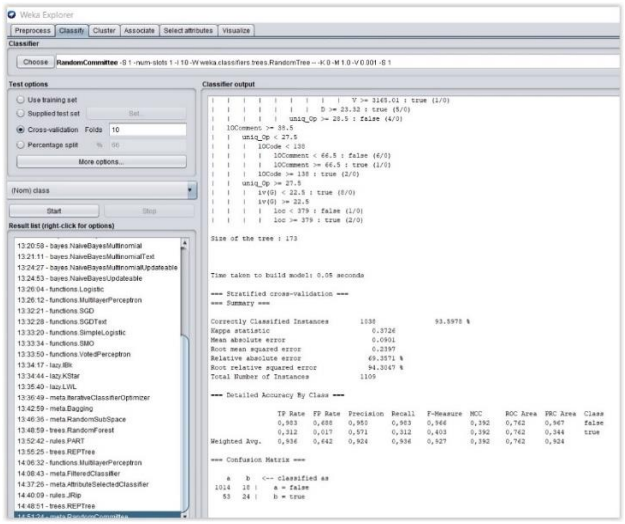


d) Part Algoritması Simülasyon Sonuçları

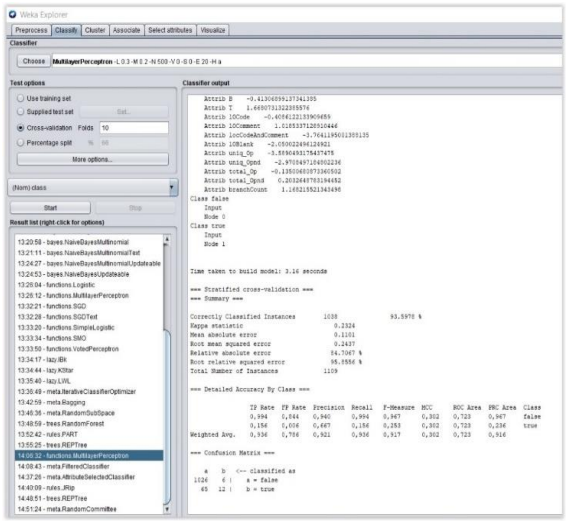


e) Rep Tree Algoritması Simülasyon Sonuçları

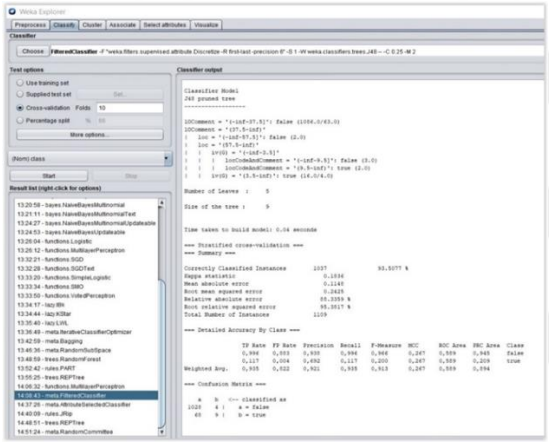
Şekil 4.13: PC1 Veri Seti Simülasyon Sonuçları



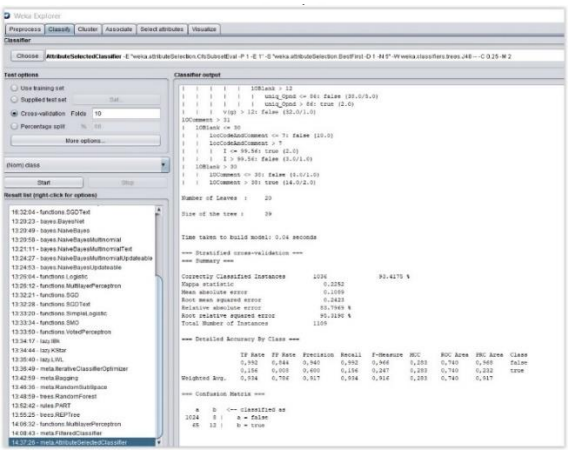
a) Random Committe Algoritması Simülasyon Sonuçları



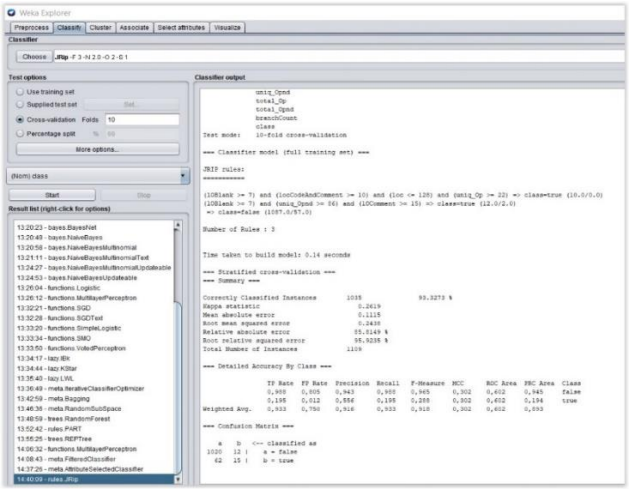
b) Multilayer Perceptron Algoritması Simülasyon



c) Filtered Classifier Algoritması Algoritması Simülasyon Sonuçları

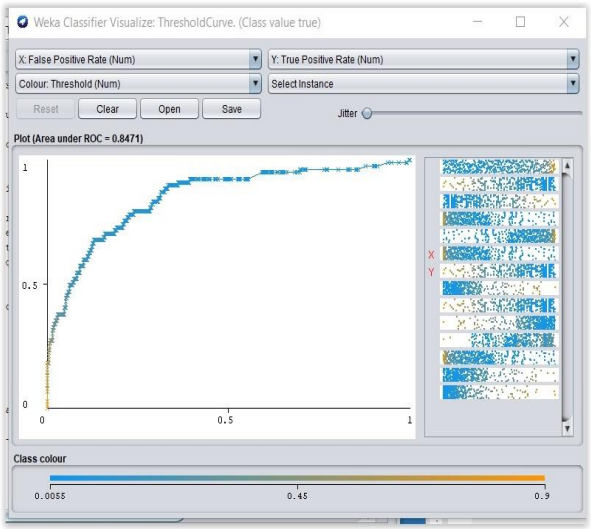


d) Attribute Selected Classifier Algoritması Simülasyon Sonuçları

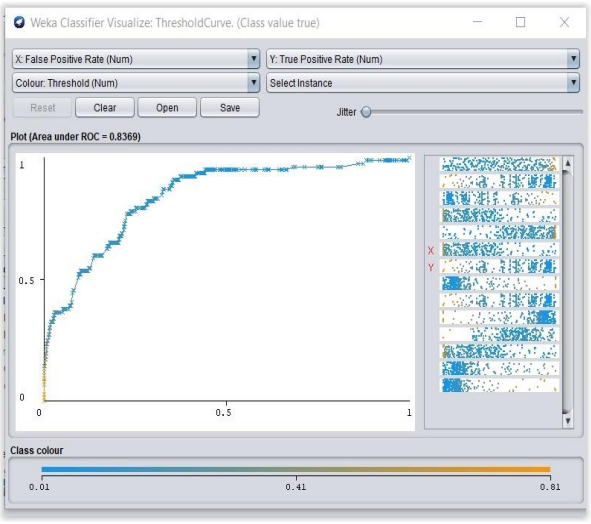


e) Repeated Incremental Pruning Algoritması Simülasyon Sonuçları

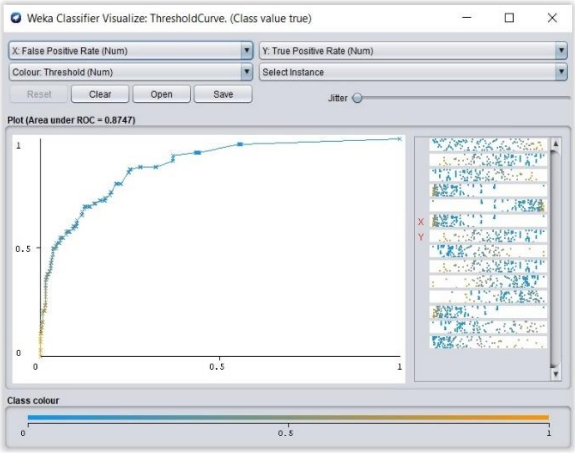
Şekil 4.14: PC1 Veri Seti Simülasyon Sonuçları-2



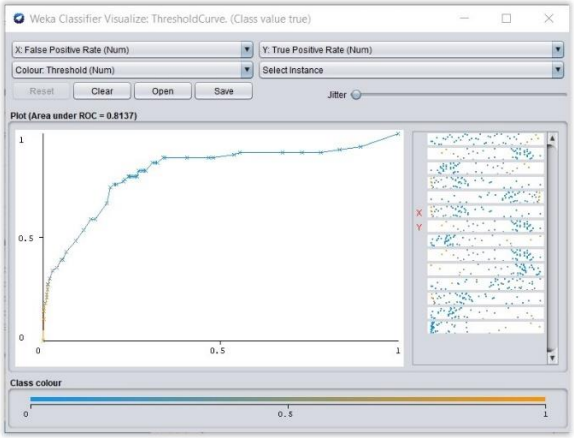
a) Bagging Algoritması Simülasyon Sonuçları ROC Eğrisi



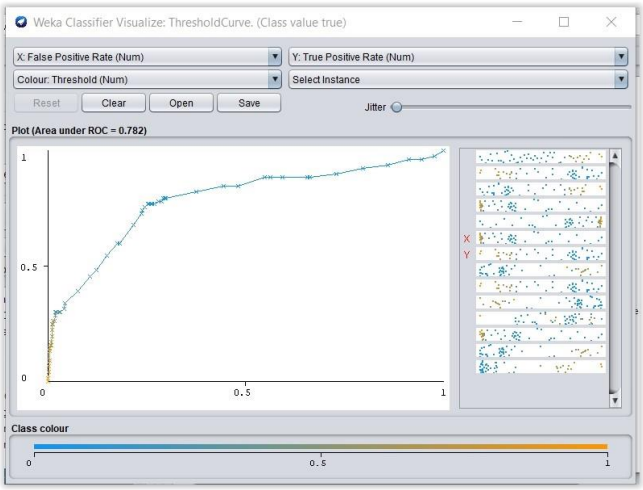
b) Random Sub Space Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Random Forest Algoritması Simülasyon Sonuçları ROC Eğrisi

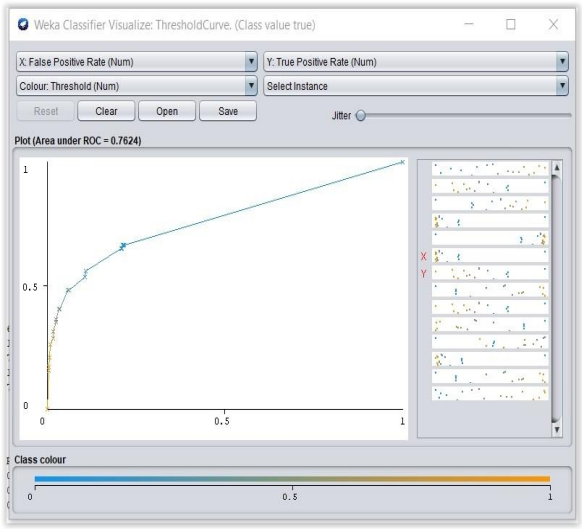


d) Part Algoritması Simülasyon Sonuçları ROC Eğrisi

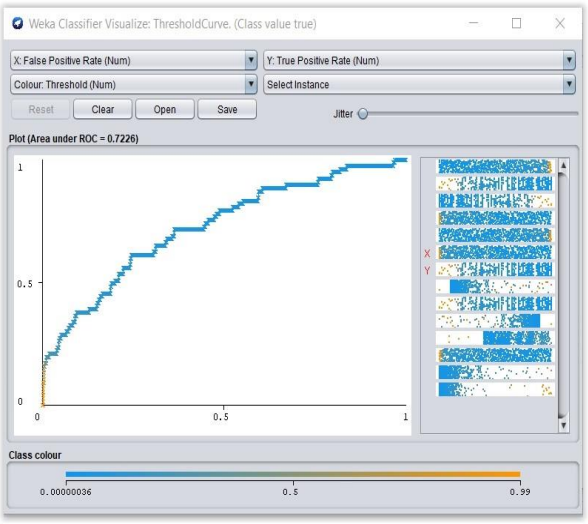


e) Rep Tree Algoritması Simülasyon Sonuçları ROC Eğrisi

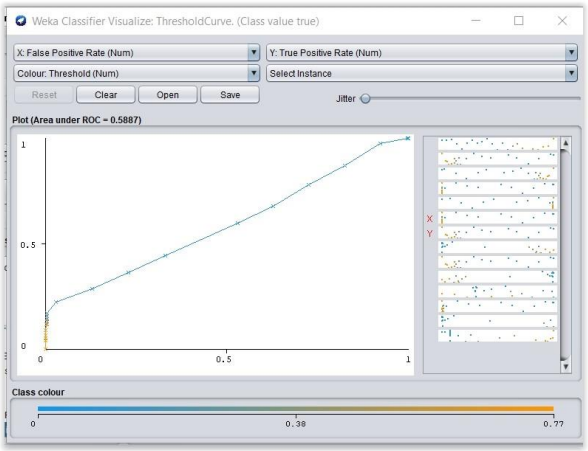
Şekil 4.15: PC1 Veri Seti Simülasyon Sonuçları ROC Eğrisi



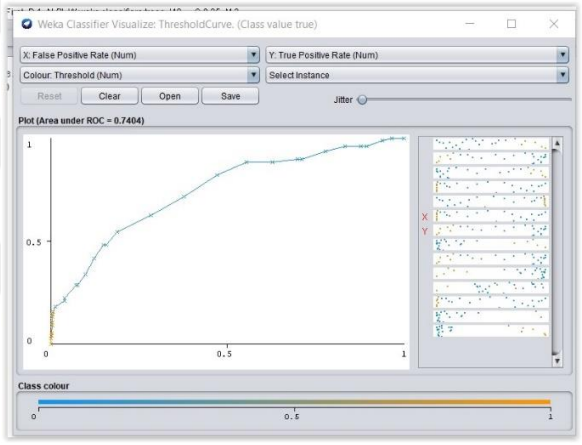
a) Random Committe Algoritması Simülasyon Sonuçları ROC Eğrisi



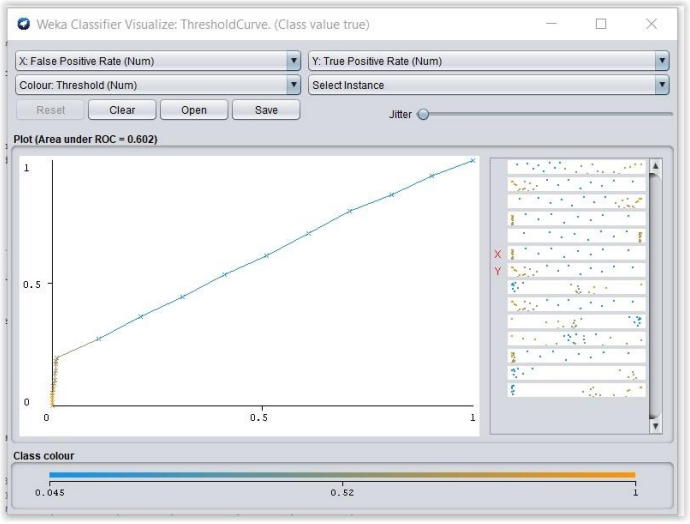
b) Multilayer Perceptron Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Filtered Classifier Algoritması Simülasyon Sonuçları ROC Eğrisi



d) Attribute Selected Classifier Algoritması Simülasyon Sonuçları ROC Eğrisi



e) Repeated Incremental Pruning Algoritması Simülasyon Sonuçları ROC Eğrisi

Şekil 4.16: PC1 Veri Seti Simülasyon Sonuçları ROC Eğrisi-2

JM1'de uygulanan 46 algoritmaya bakıldığında en yüksek doğruluk oranının %81,75 ile Random Forest olduğu, en düşük doğruluk oranının ise %52,21 ile Voted Perceptron olduğu görülmüştür. JM1'de en yüksek doğruluk oranını veren 10 algoritmanın simülasyon sonuçları Şekil 4.1 ve Şekil 4.2'de verilmiştir. Bu sonuçlar incelendiğinde Random Forest algoritmasının %81,75 ile en yüksek doğruluk oranını aldığı, Random Forest algoritmasında ROC eğrisi altında kalan alandan hesaplanan AUC değerinin 0,75 olduğu görülmüştür. Diğer dokuz algoritmanın doğruluk oranları %81,49 ile %80,95 arasında azaldığı görülmektedir. AUC değerleri de 0,73 ile 0,68 arasında azalmıştır. JM1'deki en yüksek doğruluk oranını veren 10 algoritmanın AUC değerlerinin de yüksek olması bu algoritmalarda tahmin etme başarımının daha yüksek olduğunu ispatlamaktadır. KC1'de uygulanan 46 algoritmanın tamamına bakıldığında en yüksek doğruluk oranının %86,67 ile Random Forest, en düşük doğruluk oranının ise % 69,86 ile Bayes Net olduğu görülmüştür. KC1'de en yüksek doğruluk oranını veren 10 algoritmanın simülasyon sonuçları Şekil 4.5 ve Şekil 4.6'da verilmiştir. Sonuçlar incelendiğinde Random Forest'ın %86,67 ile en yüksek doğruluk oranını aldığı, Random Forest algoritmasında ROC eğrisi altında kalan alandan hesaplanan AUC değerinin 0,82 olduğu görülmüştür. Diğer dokuz algoritmanın da doğruluk oranlarının %86,01 ile %85,24 arasında azaldığı görülmektedir. AUC değerleri de Naive Bayes Multinomial ve Naive Bayes Multinomial Updateable algoritmaları hariç 0,82 ile 0,78 arasındadır. AUC değerlerinin de yüksek olması bu algoritmalarda tahmin etme başarımının daha yüksek olduğunu ispatlamaktadır. CM1'de uygulanan 46 algoritmanın tamamına bakıldığında en yüksek doğruluk oranının %90,16 ile Naive Bayes Multinomial Text, en düşük doğruluk oranının ise %64,65 ile Bayes Net olduğu görülmüştür. CM1'de en yüksek doğruluk oranını veren 10 algoritmanın simülasyon sonuçları Şekil 4.9 ve Şekil 4.10'da verilmiştir. Sonuçlar incelendiğinde Naive Bayes Multinomial Text'in %90,16 ile en yüksek doğruluk oranını aldığı, Naive Bayes Multinomial Text algoritmasında ROC eğrisi altında kalan alandan hesaplanan AUC değerinin 0,49 olduğu görülmüştür. Diğer dokuz algoritmanın da doğruluk oranları %90,16 çıkmıştır. AUC değerleri de 0,70 ile 0,49 arasında değişmektedir. PC1 veri setinde uygulanan 46 makine öğrenmesi algoritmasının tamamına bakıldığında en yüksek doğruluk oranının %94,13 ile Bagging algoritması olduğu görülmüştür. En düşük doğruluk oranının ise %74,39 ile Bayes Net algoritması olduğu görülmüştür. PC1'de en yüksek doğruluk oranını veren 10 algoritmanın simülasyon sonuçları Şekil 4.13 ve Şekil 4.14'de verilmiştir. Sonuçlar incelendiğinde Bagging'in %94,13 ile en yüksek doğruluk oranını aldığı, burada ROC eğrisi altında kalan alandan hesaplanan AUC değerinin 0,84 olduğu görülmüştür. Diğer dokuz

algoritmanın da doğruluk oranları %93,86 ile %93,32 arasında azaldığı görülmektedir. AUC değerleri de 0,84 ile 0,58 arasında azalmıştır. PC1'de en yüksek doğruluk oranını veren 10 algoritmanın AUC değerlerinin de yüksek olması bu algoritmalarda tahmin etme başarımının daha yüksek olduğunu ispatlamaktadır. JM1, KC1, CM1 ve PC1 kullanılarak WEKA'da gerçeklenen 46 algoritmaya ait simülasyon sonuçları Şekil 4.1 ile Şekil 4.16 arasında verilmiştir. JM1ve KC1 veri setinde Random Forest algoritması en yüksek doğruluk oranını vermiştir. CM1'de Naive Bayes Multinomial Text, PC1'de ise Bagging algoritması en yüksek doğruluk oranlarını vermiştir. Voted Perceptron %52,21 ile JM1'de en düşük doğruluk oranını alırken, KC1'de %69,86 ile Bayes Net algoritması en düşük doğruluk oranını almıştır. CM1'de %64,65 ile Bayes Net algoritması en düşük doğruluk oranını almıştır. PC1'de %74,39 ile en düşük doğruluk oranını yine Bayes Net almıştır. PC1'de Bayes Net ile elde edilen en düşük doğruluk oranının JM1, KC1, CM1'de elde edilen en düşük doğruluk oranların aksine biraz daha yüksek olduğu görülmektedir. JM1'de gerçeklenen tüm doğruluk oranlarının ortalamasının %81,09, KC1'de gerçeklenen tüm algoritmaların ortalamasının %67,45, CM1'de gerçeklenen tüm algoritmaların ortalamasının %82,40, PC1'de gerçeklenen tüm doğruluk oranlarının ortalaması %88,43 olarak hesaplanmıştır. Ortalama doğruluk oranlarına bakıldığında en yüksek oranın %88,43 ile PC1 olduğu görülmektedir. Sonuçlar incelendiğinde, Random Forest, Sequential Minimal Optimization, Bagging, Simple Logistic algoritmalarının %85 üzerinde bir başarı oranıyla tüm veri setlerinde en başarılı sonuçları verdiği, Voted Perceptron, Naive Bayes Multinominal Text, Stochastic Gradient Descent Text, OneR algoritmalarının ise doğruluk oranının tüm veri setlerinde oldukça düşük olduğu görülmektedir.

4.1.2. Deneysel Veri Seti Simülasyon Sonuçları

Tez çalışmasının bu bölümünde ise, 102 öğrenciye ait yazılım proje kodlarından hesaplanarak oluşturulan öğrenci veri seti üzerinde WEKA veri madenciliği yazılımı bünyesinde 10 katlı çapraz geçerleme kuralı ile Bayes Net, Naive Bayes, Naive Bayes Multinomial, Naive Bayes Multinominal Text, Naive Bayes Multinomial Updateable, Naive Bayes Updatable, Logistic, Multilayer Perceptron, Stochastic Gradient Descent Text, Simple Logistic, Sequential Minimal Optimization, Voted Perceptron, K-Nearest Neighbours Classifier, K*, Locally Weighted Learning, Iterative Classifier Optimizer, Adaboost1, Attribute Selected Classifier, Bagging, Classification via Regression, Cross Validation Parameter Selection, Filtered Classsifier, Logi Boost, Multi Class Classifier, Multi Class Classifier Updatable, Multi Scheme, Random Comittee, Randomizable Filtered Classifer, Random Sub Space, Stacking, Vote, Weighted

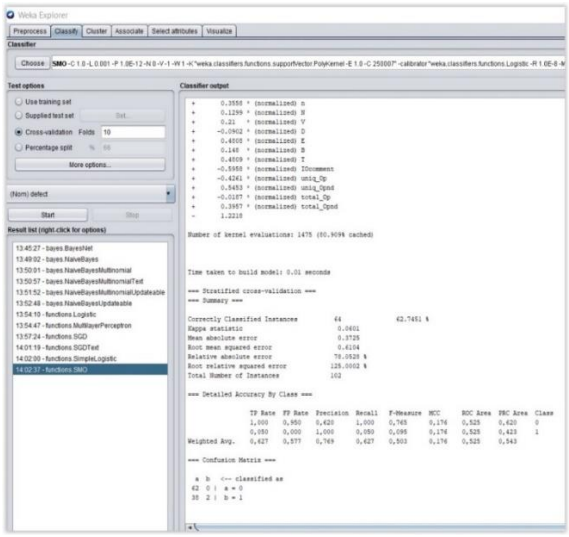
Instances Handler Wrapper, Input Mapped Classifier, Desicion Table, Repeated Incremental Pruning, OneR, Part, ZeroR, Desicion Stump, Hoeffding Tree, J48, Random Forest, Random Tree, Rep Tree algoritmalarıyla gerçeklenmiş, doğruluklar hesaplanmış ve sunulmuştur (Tablo 4.2).

Tablo 4.2: WEKA Deneysel Veri Seti Doğruluk Oranları

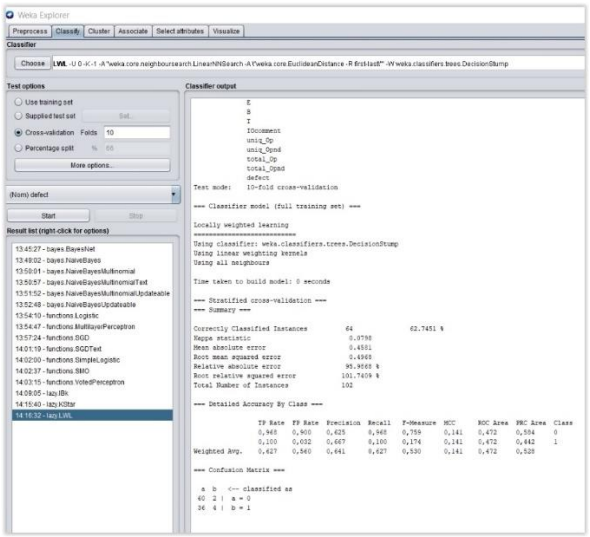
Algoritma	**Deneysel Veri Seti Doğruluk Oranı (%)**
BAYES	
Bayes Net	60,78
Naive Bayes	60,78
Naive Bayes Multinomial	59,80
Naive Bayes Multinominal Text	60,78
Naive Bayes Multinomial Updateable	59,80
Naive Bayes Updatable	60,78
FUNCTIONS	
Logistic	55,88
Multilayer Perceptron	52,94
Stochastic Gradient Descent	61,76
Stochastic Gradient Descent Text	60,78
Simple Logistic	57,84
Sequential Minimal Optimization	62,75
Voted Perceptron	56,86
LAZY	
K-Nearest Neighbours Classifier	51,96
K*	49,01
Locally Weighted Learning	62,75
META	
Iterative Classifier Optimizer	60,78
Adaboost1	60,78
Attribute Selected Classifier	59,80
Bagging	49,01
Classification via Regression	59,80
Cross Validation Parameter Selection	60,78
Filtered Classsifier	60,78
Logi Boost	50,98
Multi Class Classifier	55,88
Multi Class Classifier Updatable	61,76
Multi Scheme	60,78
Random Comittee	45,09
Randomizable Filtered Classifer	45,09
Random Sub Space	60,78

Algoritma	Deneysel Veri Seti Doğruluk Oranı (%)
Stacking	60,78
Vote	60,78
Weighted Instances Handler Wrapper	60,78
MISC	
Input Mapped Classifier	60,78
Desicion Table	59,80
Repeated Incremental Pruning	55,88
OneR	46,07
Part	61,76
ZeroR	60,78
TREE	
Desicion Stump	60,78
Hoeffding Tree	57,84
J48	61,76
LMT	57,84
Random Forest	48,03
Random Tree	50,98
Rep Tree	58,82
ORTALAMA	***48,52***

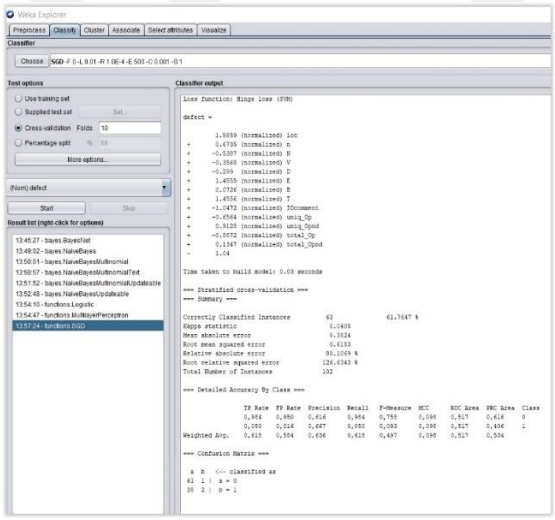
Tablo 4.2'ye genel olarak bakıldığında deneysel veri setinde elde edilen doğruluk oranlarının NASA veri setlerine göre oldukça düşük olduğu görülmüştür. En yüksek doğruluk oranı %62,75 ile Sequential Minimal Optimization ve Locally Weighted Learning algoritmalarında elde edilmiştir. %45,09 doğruluk oranı ile Random Commitee, Randomizable Filtered Classifier ve OneR algoritmalarının ise doğruluk oranının tüm veri setlerinde oldukça düşük olduğu görülmektedir. NASA'ya ait veri setlerinin simülasyon sonuçlarında olduğu gibi deneysel veri setinde de oldukça fazla sayıda algoritma ile işlem yapıldığından her algoritmaya dair simülasyon sonuçları vermek yerine deneysel veri setinde en yüksek doğruluk değerini veren 10 algoritmanın simülasyon sonuçlarının verilmesi tercih edilmiştir. En yüksek 10 doğruluk oranı Sequential Minimal Optimization, Locally Weighted Learning, Stochastic Gradient Descent, Multi Class Classifier, Part, J48, Bayes Net, Naive Bayes, Naive Bayes Multinomial, Naive Bayes Multinominal Text, Naive Bayes Multinomial Updateable, Naive Bayes Updatable algoritmalarında elde edilmiştir ve Şekil 4.17 ve Şekil 4.18'de bu algoritmaların simülasyon sonuçları, Şekil 4.19 ve Şekil 4.20'de ise ROC eğrileri verilmiştir.



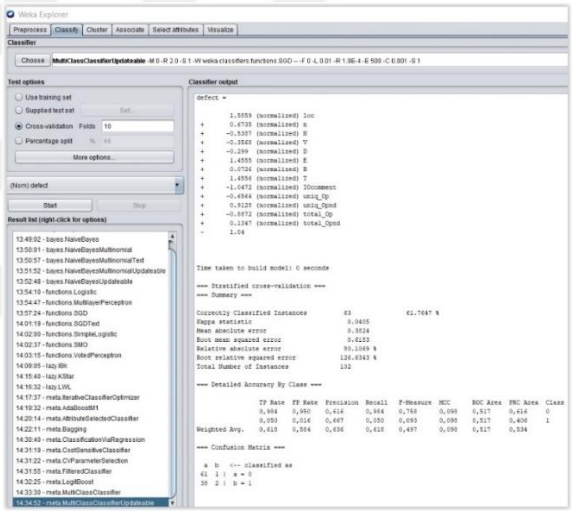
a) SMO Algoritması Simülasyon Sonuçları



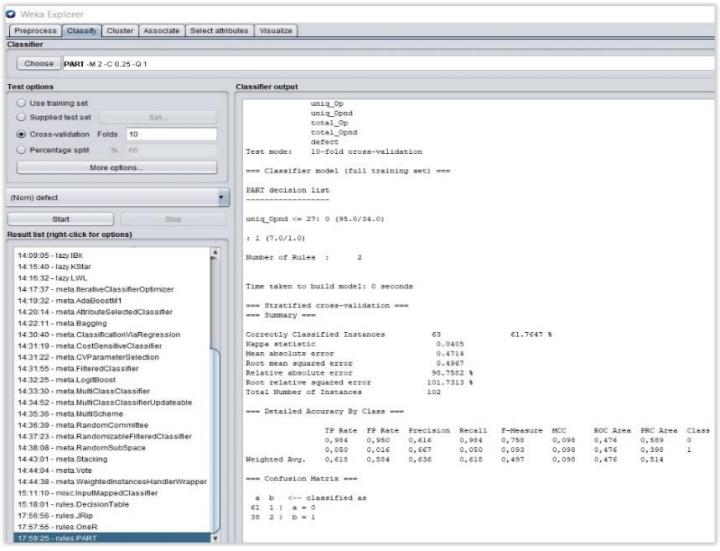
b) LWL Algoritması Simülasyon Sonuçları



c) SGD Algoritması Simülasyon Sonuçları

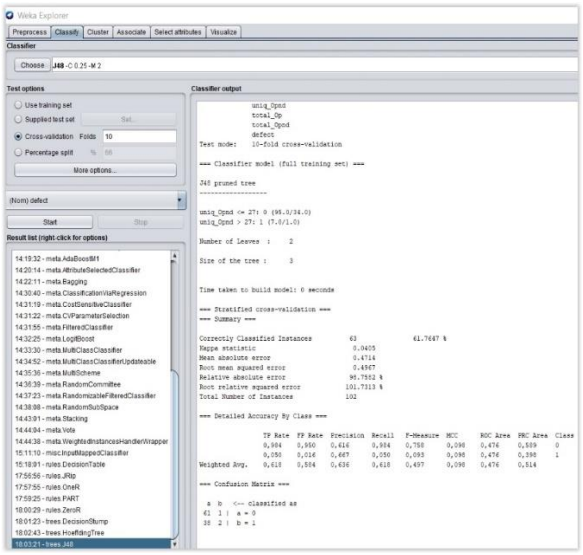


d) Multi Class Classifier Updateable Algoritması Simülasyon Sonuçları

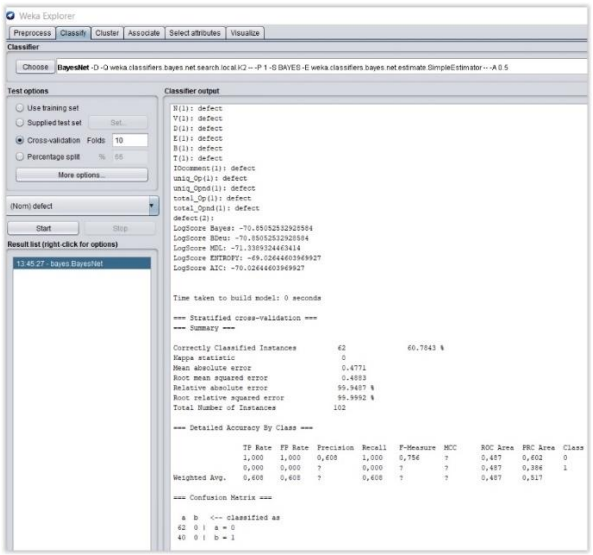


e) PART Algoritması Simülasyon Sonuçları

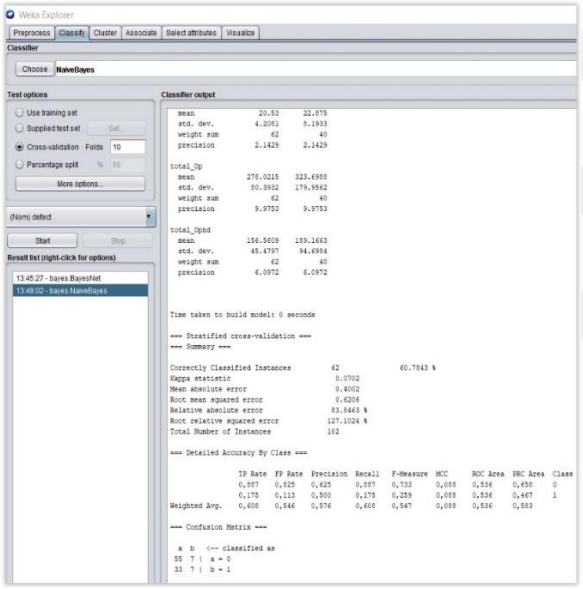
Şekil 4.17: Deneysel Veri Seti Simülasyon Sonuçları



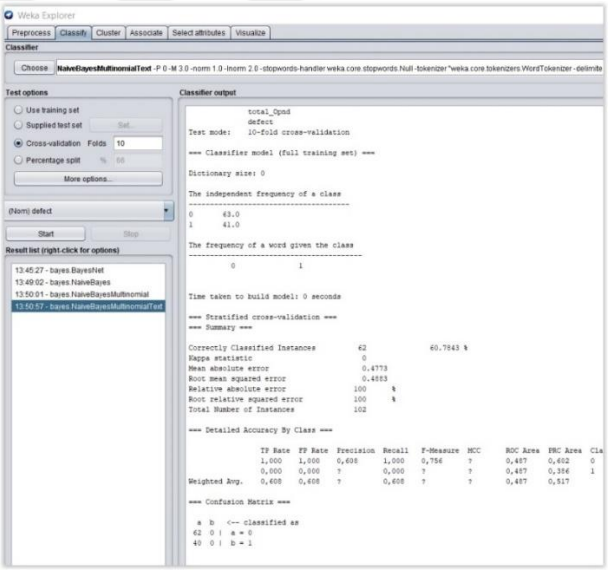
a) J48 Algoritması Simülasyon Sonuçları



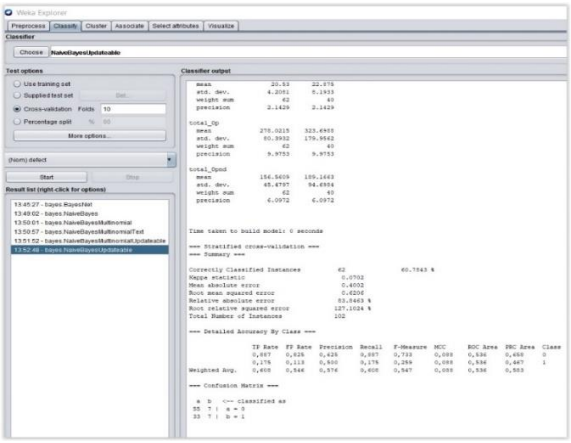
b) Bayes Net Algoritması Simülasyon Sonuçları



c) Naive Bayes Algoritması Simülasyon Sonuçları

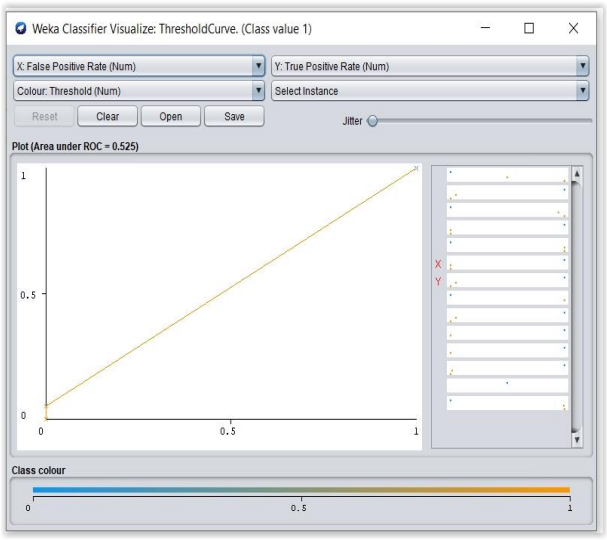


d) Naive Bayes Multinomial Text Algoritması Simülasyon Sonuçları

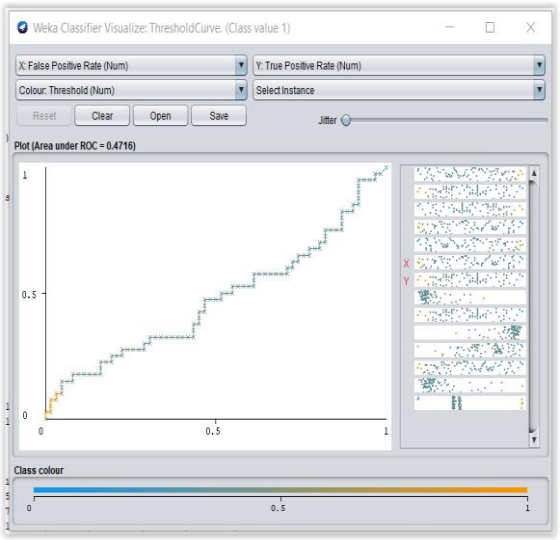


e) Naive Bayes Updateable Algoritması Simülasyon Sonuçları

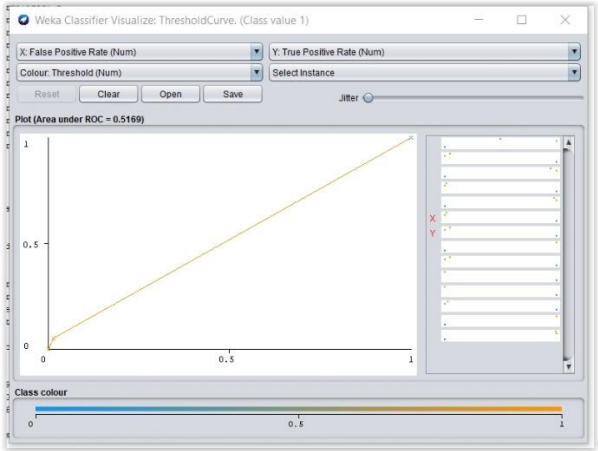
Şekil 4.18: Deneysel Veri Seti Simülasyon Sonuçları-2



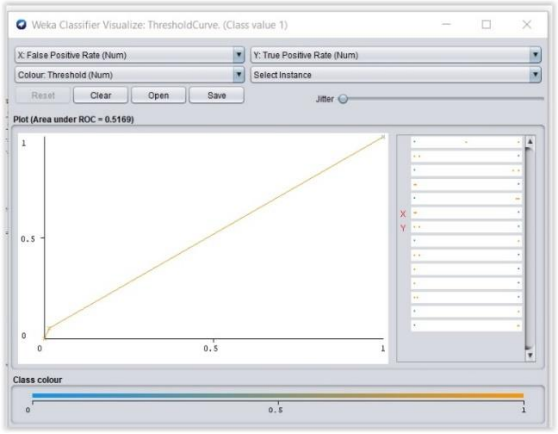
a) SMO Algoritması Simülasyon Sonuçları ROC Eğrisi



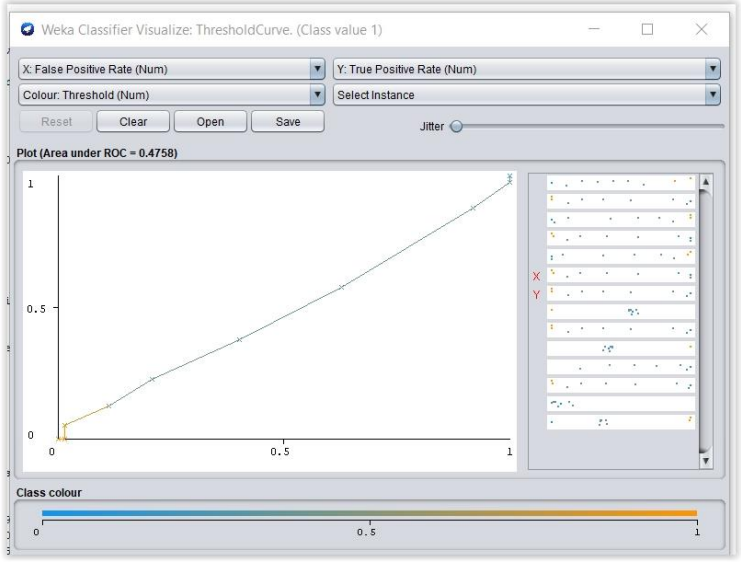
b) LWL Net Algoritması Simülasyon Sonuçları ROC



c) SGD Algoritması Simülasyon Sonuçları ROC Eğrisi

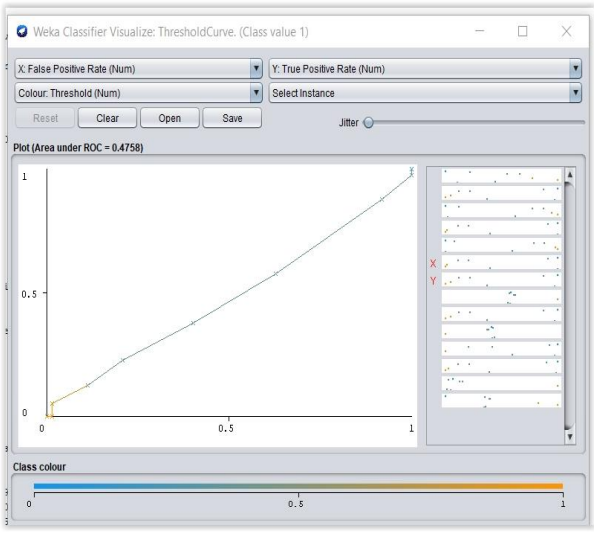


d) Multi Class Classifier Updateable Algoritması Simülasyon Sonuçları ROC Eğrisi

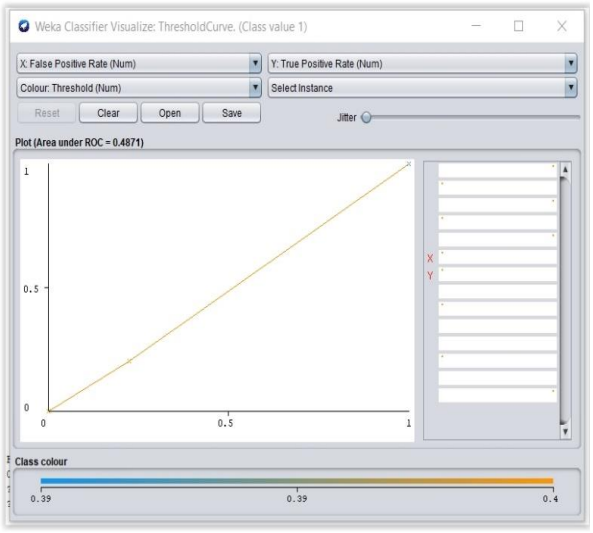


e) PART Algoritması Simülasyon Sonuçları ROC Eğrisi

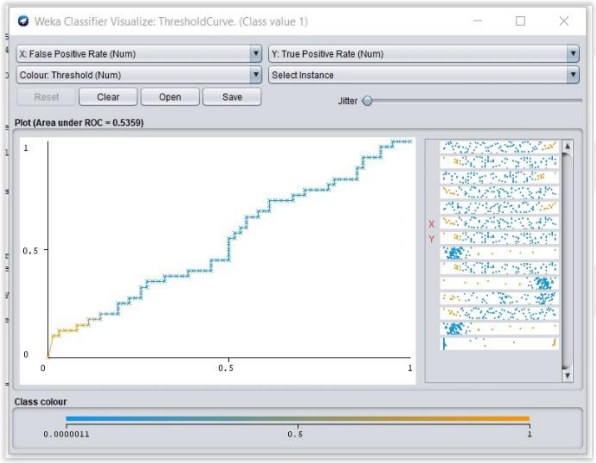
Şekil 4.19: Deneysel Veri Seti Simülasyon Sonuçları ROC Eğrisi



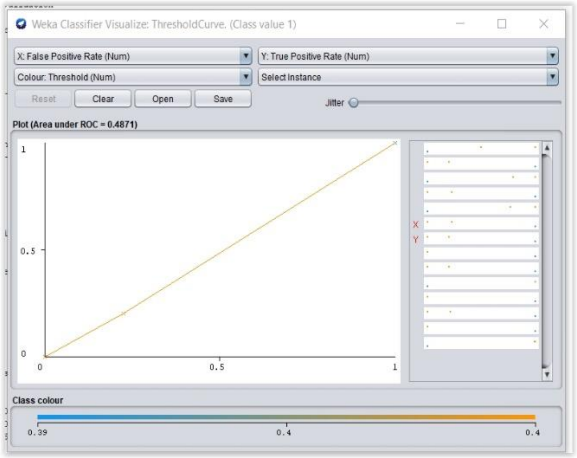
a) J48 Algoritması Simülasyon Sonuçları ROC Eğrisi



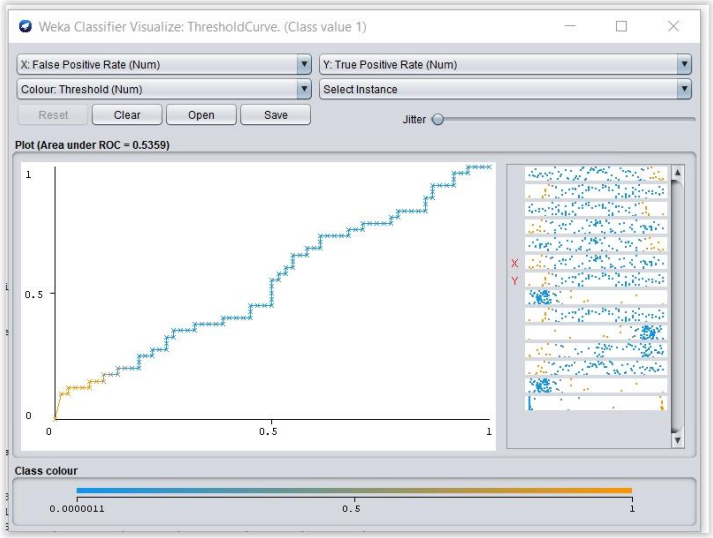
b) Bayes Net Algoritması Simülasyon Sonuçları ROC



c) Naive Bayes Algoritması Simülasyon Sonuçları ROC Eğrisi



d) Naive Bayes Multinomial Text Updateable Algoritması Simülasyon Sonuçları ROC Eğrisi



e) Naive Bayes Updateable Algoritması Simülasyon Sonuçları ROC Eğrisi

Şekil 4.20: Deneysel Veri Seti Simülasyon Sonuçları ROC Eğrisi-2

Deneysel set ile WEKA'da gerçeklenen 46 algoritma incelendiğinde en yüksek doğruluğun %62,75 ile SMO ve LWL'de, en düşük doğruluğun %45,09 ile Random Commitee ve Randomizable Filtered'da olduğu görülmüştür. En yüksek doğruluk veren 10 algoritmanın sonuçları Şekil 4.17-Şekil 4.20'de verilmiştir. SMO ve LWL'nin %62,75 ile en yüksek doğruluk oranını aldığı, SMO'da ROC eğrisi altında kalan alandan hesaplanan AUC değerinin 0,52 iken, LWL'de ROC altında kalan alandan hesaplanan AUC değerinin 0,47 olduğu görülmüştür. Diğer algoritmaların doğruluk oranlarının %62,75 ile %60,78 arasında AUC değerlerinin de 0,52 ile 0,48 arasında azaldığı görülmüştür. Deneysel sette gerçeklenen tüm algoritmaların doğruluk oranlarının ortalaması alındığında %48,52 olarak hesaplanmıştır.

4.2. PYTHON SİMÜLASYON SONUÇLARI

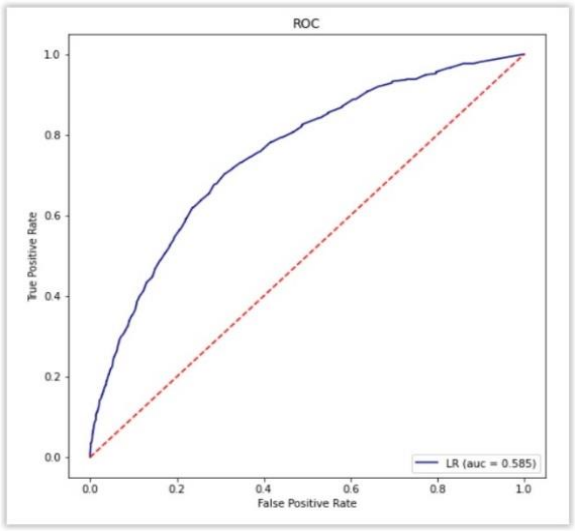
4.2.1 NASA Veri Setleri Simülasyon Sonuçları

WEKA platformunda NASA setleri için ortalama olarak en yüksek doğruluk oranını sunan 10 algoritma (RF, AdaBoost, K-NN, DT, NB, SVM, LR, NN, Gradient Boosting ve Bagging) Python scikit-learn kullanılarak uygulanmış ve değerler 10 kat çapraz doğrulama kullanılarak bulunmuştur. Tablo 4.3'de algoritmaların NASA veri setleri için doğruluk oranları sunulmuştur.

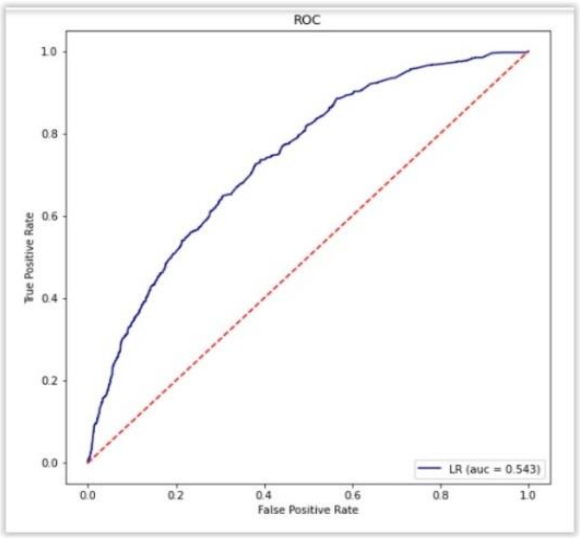
Tablo 4.3: Python NASA Veri Seti Doğruluk Oranları

Algoritma	**JM1 (%)**	**KC1 (%)**	**CM1 (%)**	**PC1 (%)**
Random Forest	81	85	87	94
AdaBoost	81	84	89	93
K-NN	79	85	85	93
Decision Tree	74	82	81	90
Naive Bayes	80	66	85	92
SVM	81	85	88	93
Logistic Regression	81	86	88	94
Neural Network	81	87	87	91
Gradient Boosting	81	85	88	94
Bagging	80	85	90	91
ORTALAMA	***79.9***	***83***	***86.8***	***92.5***

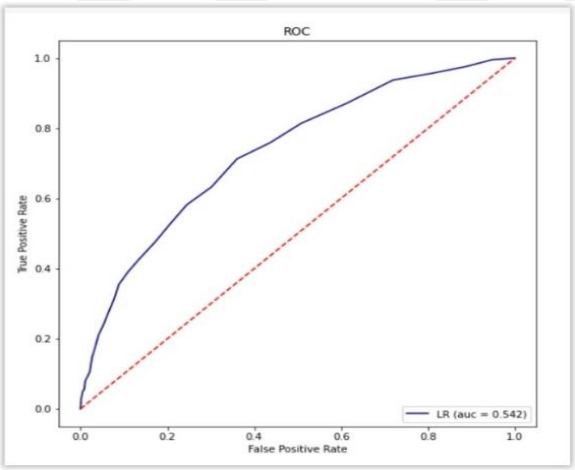
JM1'de en yüksek doğruluk oranı %81 ile RF, AdaBoost, SVM, LR, NN, Gradient Boosting ve Bagging'de; KC1'de %87 ile NN'de, CM1'de %90 ile Bagging'de görülmüştür. PC1'de RF, LR ve Gradient Boosting %94 doğruluk oranına ulaşmıştır. PC1, 10 algoritmada da en yüksek doğruluk oranı ortalamasına sahiptir. JM1'de 10 algoritmanın simülasyon sonuçlarından elde edilen ROC eğrileri Şekil 4.21 - Şekil 4.22'de verilmiştir.



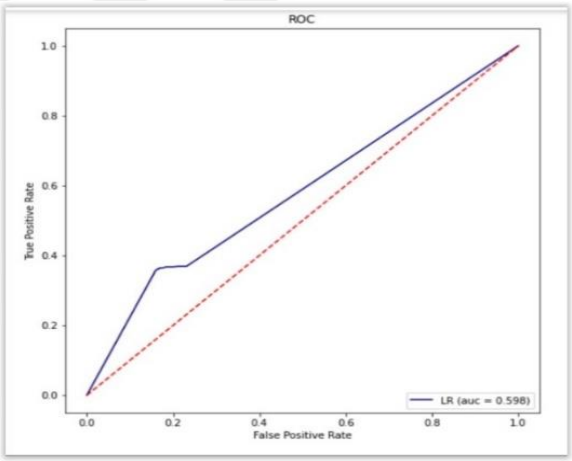
a) Random Forest Algoritması Simülasyon Sonuçları ROC Eğrisi



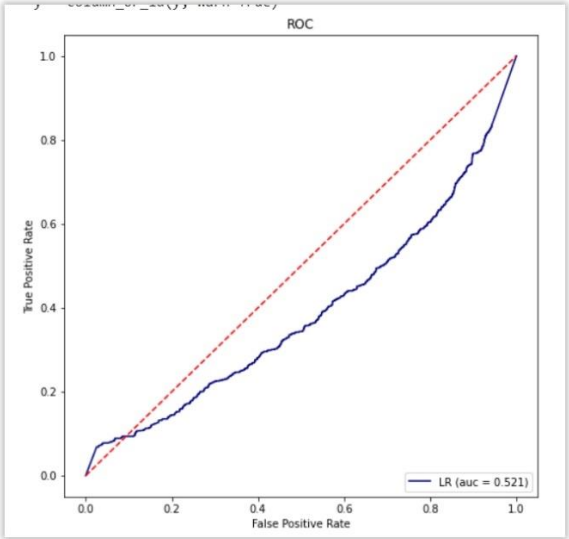
b) Adaboost Algoritması Simülasyon Sonuçları ROC Eğrisi



c) K-NN Algoritması Simülasyon Sonuçları ROC Eğrisi

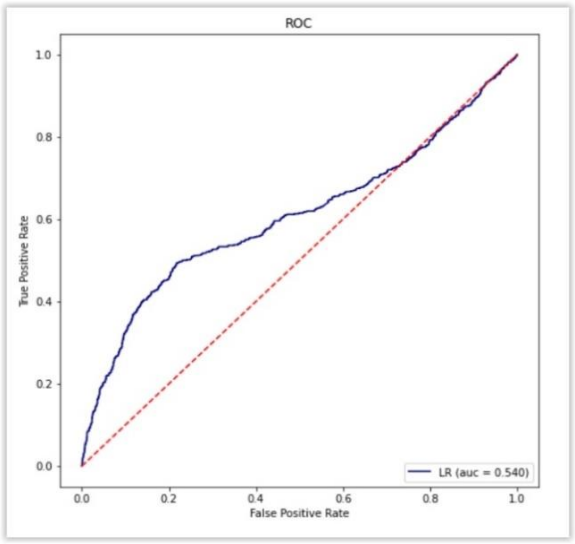


d) Desicion Tree Algoritması Simülasyon Sonuçları ROC Eğrisi

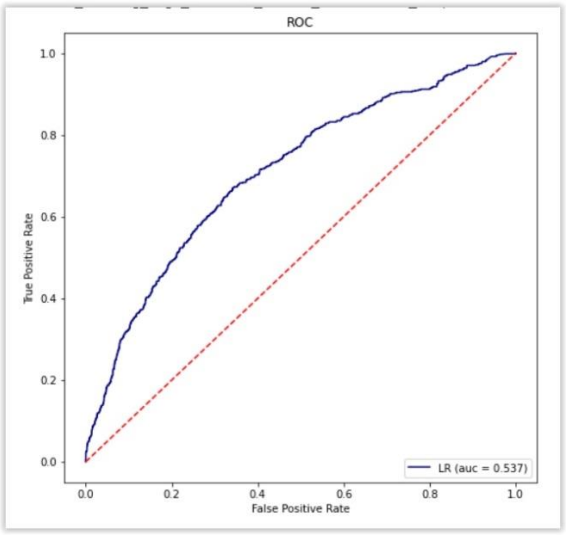


e) Naive Bayes Algoritması Simülasyon Sonuçları ROC Eğrisi

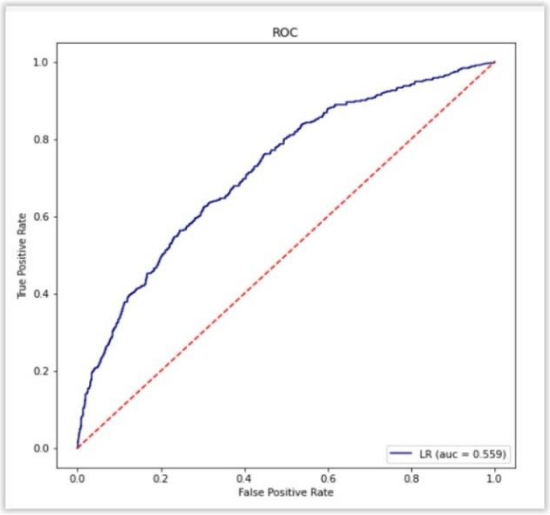
Şekil 4.21: JM1 Veri Seti Simülasyon Sonuçları ROC Eğrisi



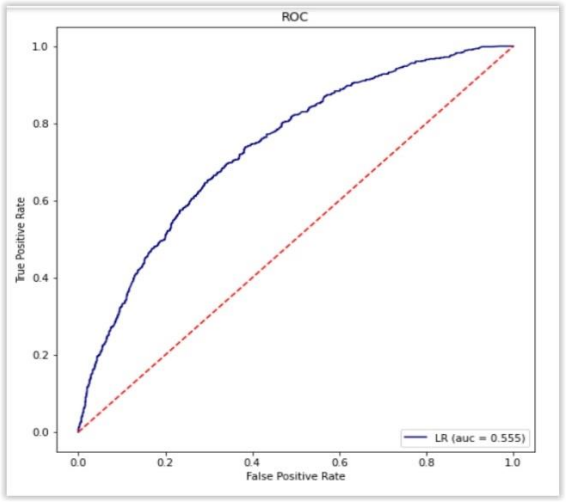
a) SVM Algoritması Simülasyon Sonuçları ROC Eğrisi



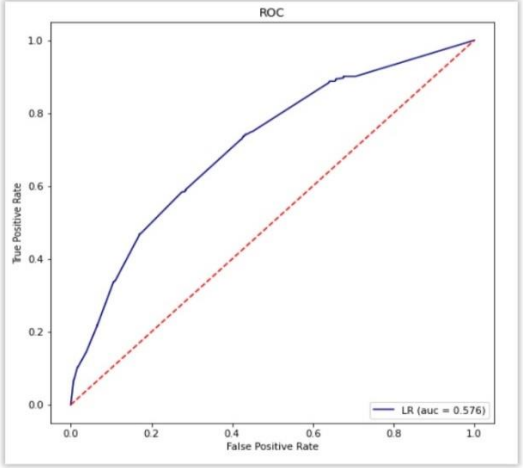
b) Logistic Regression Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Neural Network Algoritması Simülasyon Sonuçları ROC Eğrisi

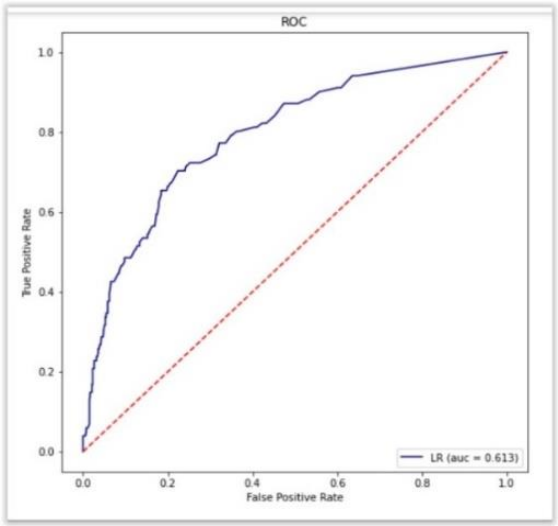


d) Gradient Boosting Algoritması Simülasyon Sonuçları ROC Eğrisi

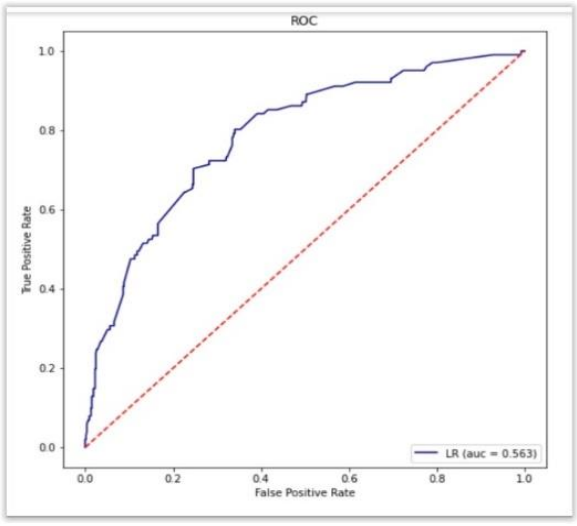


e) Bagging Algoritması Simülasyon Sonuçları ROC Eğrisi

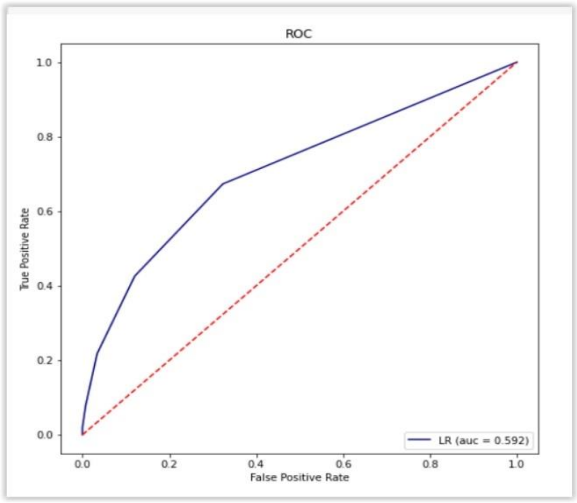
Şekil 4.22: JM1 Veri Seti Simülasyon Sonuçları ROC Eğrisi-2



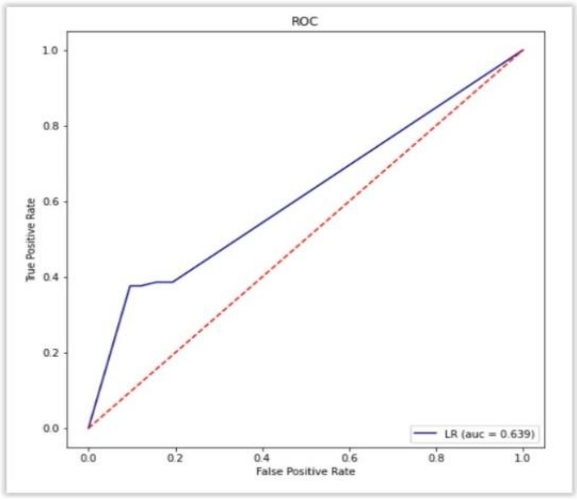
a) Random Forest Algoritması Simülasyon Sonuçları ROC Eğrisi



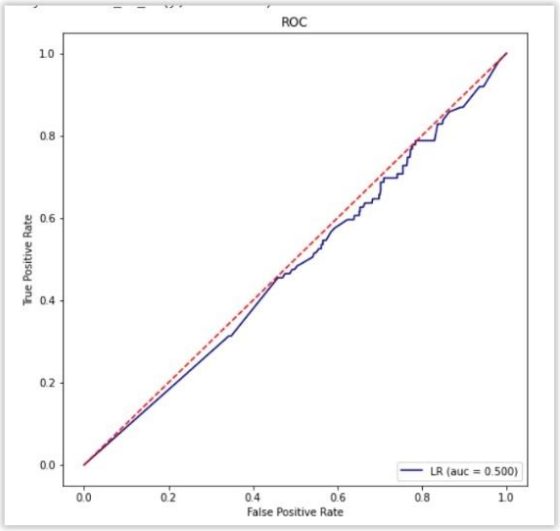
b) AdaBoost Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Seti K-NN Algoritması Simülasyon Sonuçları ROC Eğrisi

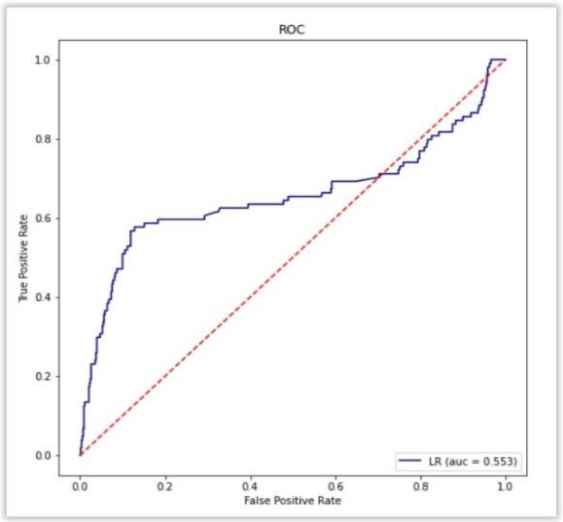


d) Desicion Tree Algoritması Simülasyon Sonuçları ROC Eğrisi

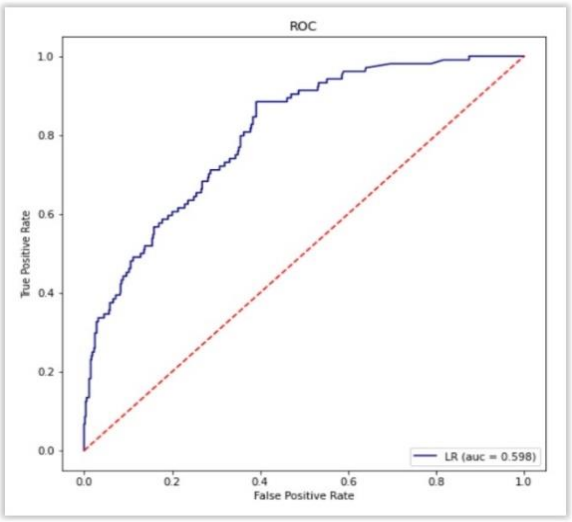


e) Naive Bayes Algoritması Simülasyon Sonuçları ROC Eğrisi

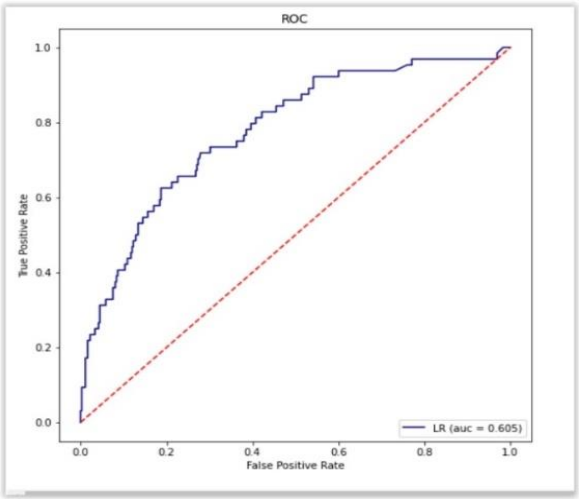
Şekil 4.23: KC1 Veri Seti Simülasyon Sonuçları ROC Eğrisi



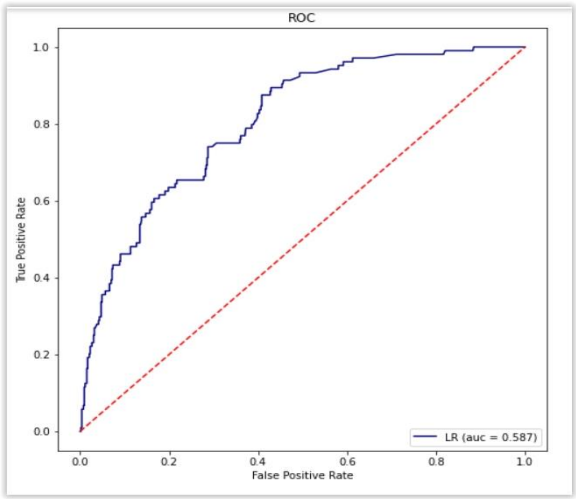
a) SVM Algoritması Simülasyon Sonuçları ROC Eğrisi



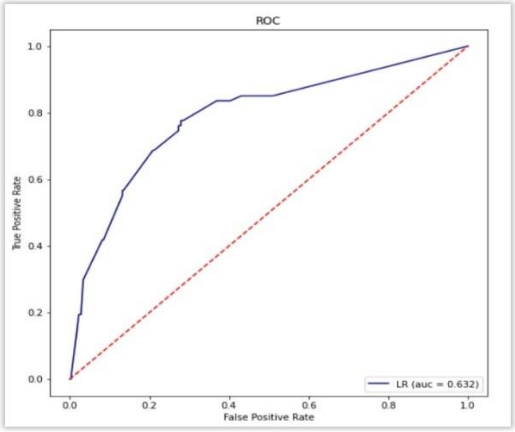
b) Logistic Regression Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Neural Network Algoritması Simülasyon Sonuçları ROC Eğrisi

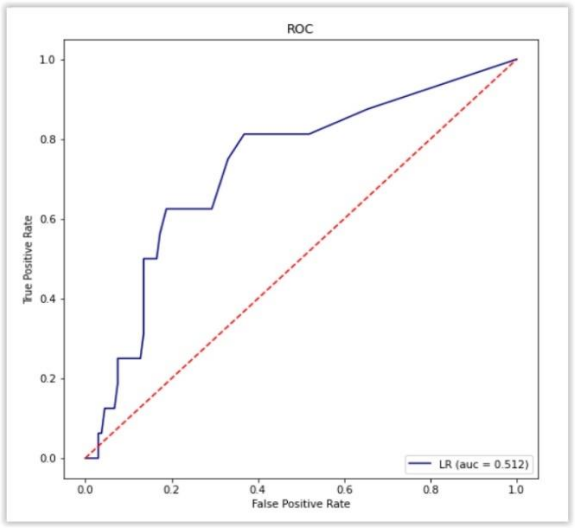


d) Gradient Boosting Algoritması Simülasyon Sonuçları ROC Eğrisi

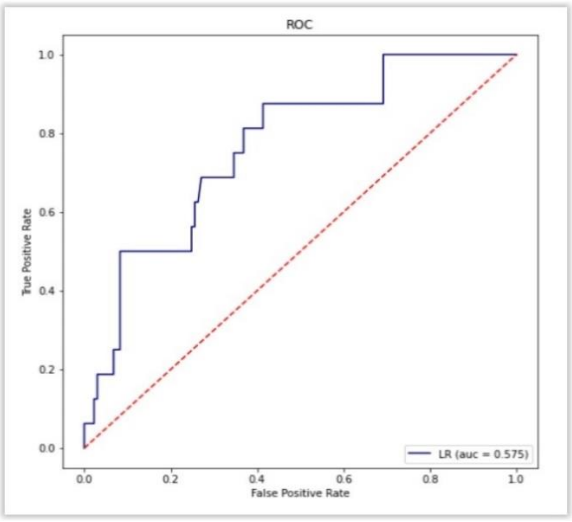


e) Bagging Algoritması Simülasyon Sonuçları ROC Eğrisi

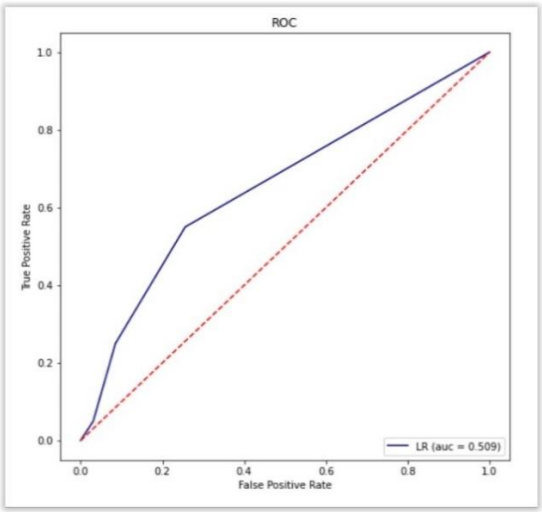
Şekil 4.24: KC1 Veri Seti Simülasyon Sonuçları ROC Eğrisi-2



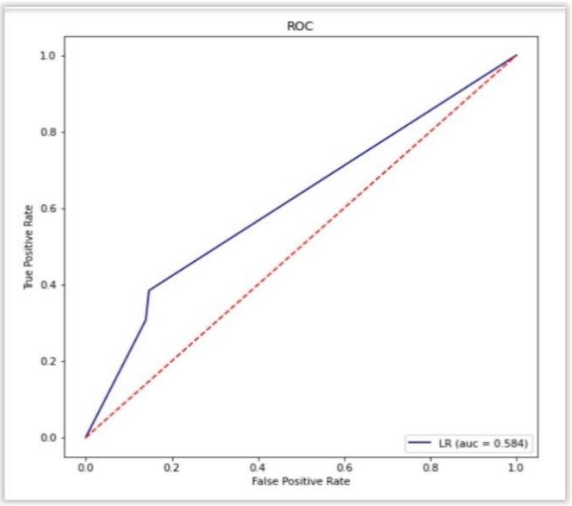
a) Random Forest Algoritması Simülasyon Sonuçları ROC Eğrisi



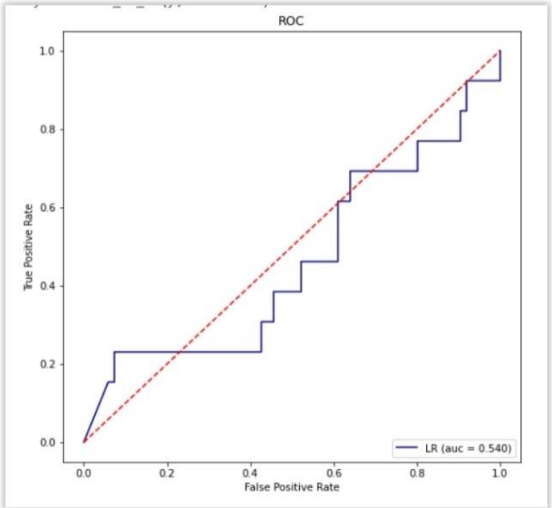
b) AdaBoost Algoritması Simülasyon Sonuçları ROC Eğrisi



c) K-NN Algoritması Simülasyon Sonuçları ROC Eğrisi

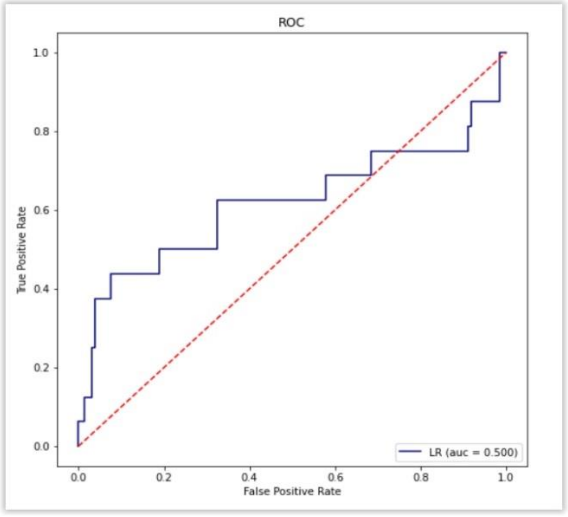


d) Desicion Tree Algoritması Simülasyon Sonuçları ROC Eğrisi

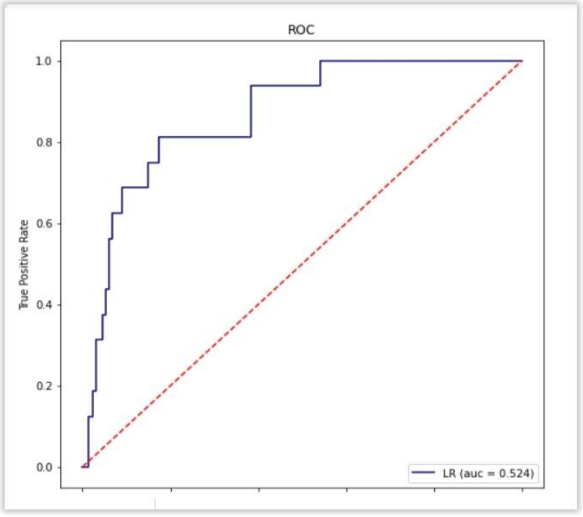


e) Naive Bayes Algoritması Simülasyon Sonuçları ROC Eğrisi

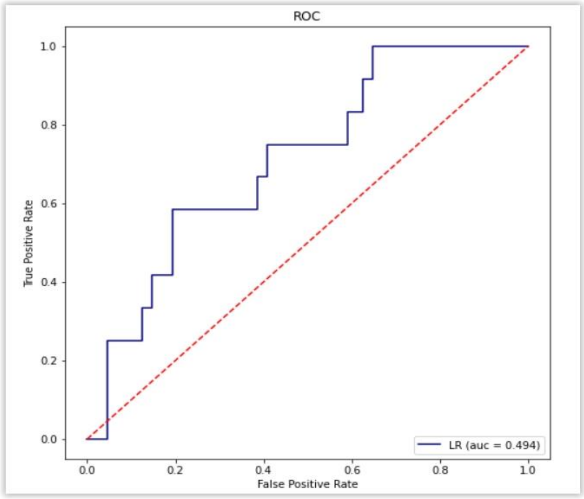
Şekil 4.25: CM1 Veri Seti Simülasyon Sonuçları ROC Eğrisi



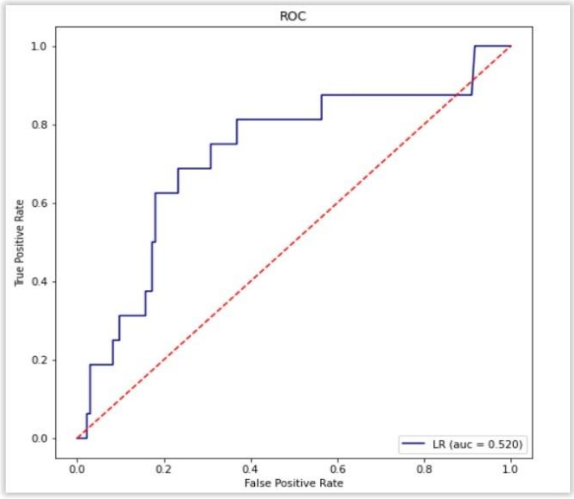
a) SVM Algoritması Simülasyon Sonuçları ROC Eğrisi



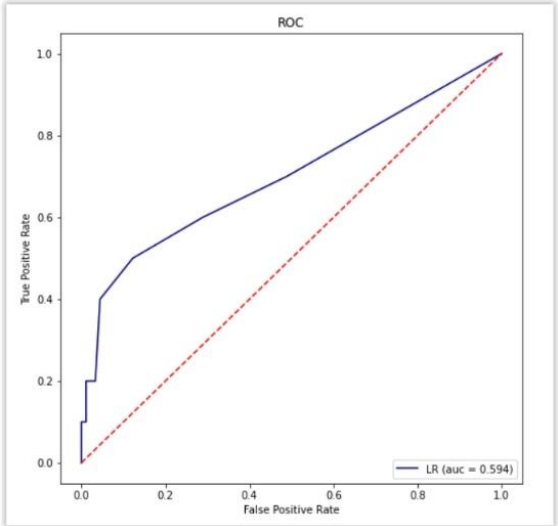
b) Logistic Regression Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Neural Network Algoritması Simülasyon Sonuçları ROC Eğrisi



d) Gradient Boosting Algoritması Simülasyon Sonuçları ROC Eğrisi



e) Bagging Algoritması Simülasyon Sonuçları ROC Eğrisi

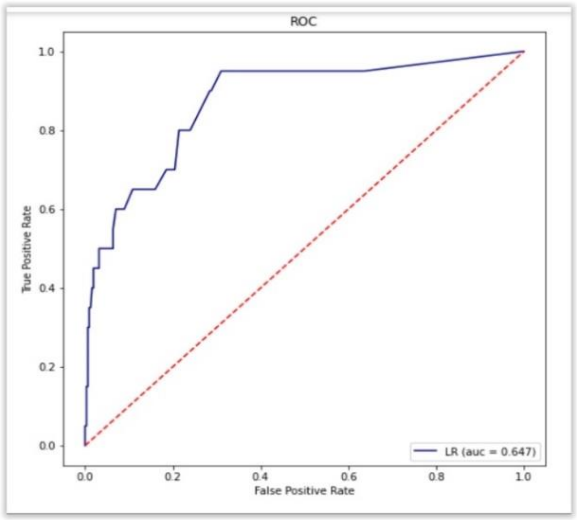
Şekil 4.26: CM1 Veri Seti Simülasyon Sonuçları ROC Eğrisi-2

JM1'e uygulanan 10 algoritmaya bakıldığında en yüksek doğruluk oranının %81 ile Random Forest, AdaBoost, SVM, Logistic Regression, Neural Network ve Gradient Boosting algoritmaları olduğu, en düşük doğruluk oranının ise %74 ile Desicion Tree algoritması olduğu görülmüştür. Elde edilen simülasyon sonuçları Şekil 4.21 ve Şekil 4.22'de verilmiştir. Algoritmalardan elde edilen ROC eğrisi altında kalan alandan hesaplanan AUC değerleri 0,598 ile 0,521 arasında azalarak değişmektedir. KC1 veri setinde gerçeklenen 10 öğrenme algoritmasının simülasyon sonuçlarından elde edilen ROC eğrileri Şekil 4.23 ve Şekil 4.24'de verilmiştir.

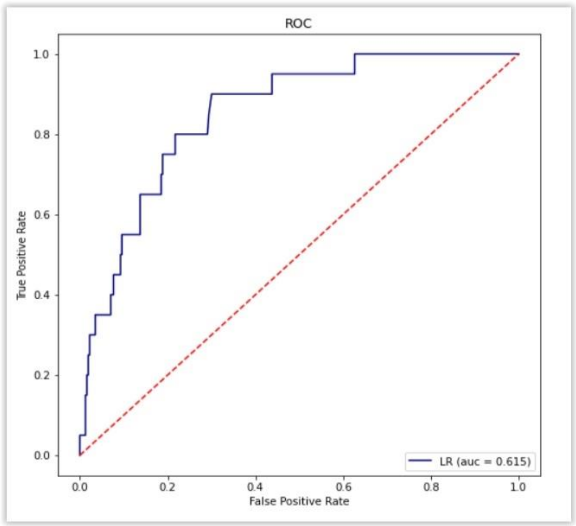
KC1 veri setinde uygulanan 10 makine öğrenmesi algoritmasının tamamına bakıldığında en yüksek doğruluk oranının %87 ile Neural Network algoritması olduğu görülmüştür. Ancak yine de diğer algoritmalarda elde edilen doğruluk oranlarının da Neural Network algoritmasından elde edilen doğruluk oranlarına çok yakın olduğu görülmüştür. En düşük doğruluk oranının ise %66 ile Naive Bayes algoritması olduğu görülmüştür. KC1 veri setinde 10 algoritmadan elde edilen simülasyon sonuçları Şekil 4.23 ve Şekil 4.24'de verilmiştir. Algoritmalardan elde edilen ROC eğrisi altında kalan alandan hesaplanan AUC değerleri 0,500 ile 0,639 arasında azalarak değişmektedir.

CM1 veri setinde gerçeklenen 10 öğrenme algoritmasının simülasyon sonuçlarından elde edilen ROC eğrileri Şekil 4.25 ve Şekil 4.26'da verilmiştir.

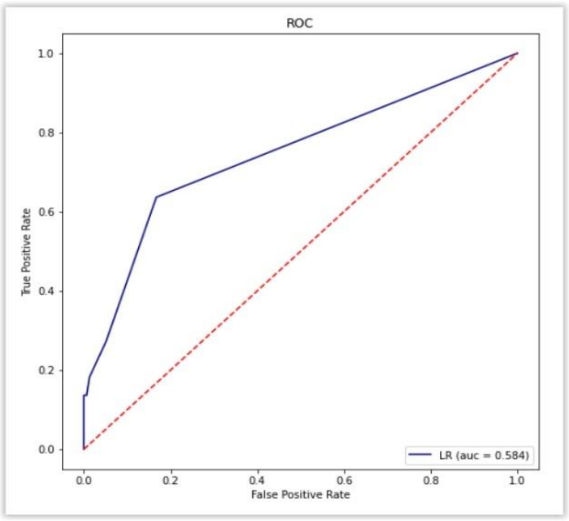
Tablo 4.3'de görüldüğü gibi CM1 veri setinde Bagging algoritması uygulaması sonucunda doğruluk oranı %90 olarak çıkmıştır. Bu sonuç, örneklerin %90'ının doğru sınıflandırıldığı anlamını ifade etmektedir. Şekil 4.26'da ise Bagging algoritmasına ait simülasyon sonucunda görüldüğü gibi ROC eğrisi altında kalan AUC olarak belirtilen değer 0.594 olarak hesaplanmıştır. CM1 veri setinde uygulanan 10 makine öğrenmesi algoritmasının tamamına bakıldığında en yüksek doğruluk oranının %90 ile Bagging algoritması olduğu görülmüştür. En düşük doğruluk oranının ise %81 ile Desicion Tree algoritması olduğu görülmüştür. CM1 veri setinde 10 algoritmadan elde edilen simülasyon sonuçları Şekil 4.25 ve Şekil 4.26'da verilmiştir. Algoritmalardan elde edilen ROC eğrisi altında kalan alandan hesaplanan AUC değerleri 0,594 ile 0,500 arasında azalarak değişmektedir. PC1 veri setinde gerçeklenen 10 öğrenme algoritmasının simülasyon sonuçlarından elde edilen ROC eğrileri Şekil 4.27 ve Şekil 4.28'de verilmiştir.



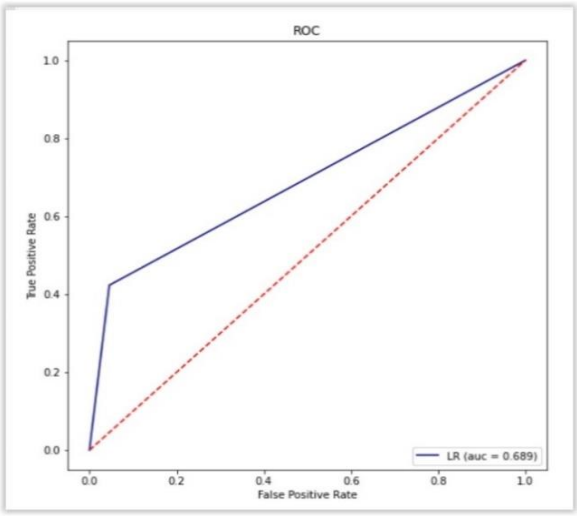
a) Random Forest Algoritması Simülasyon Sonuçları ROC Eğrisi



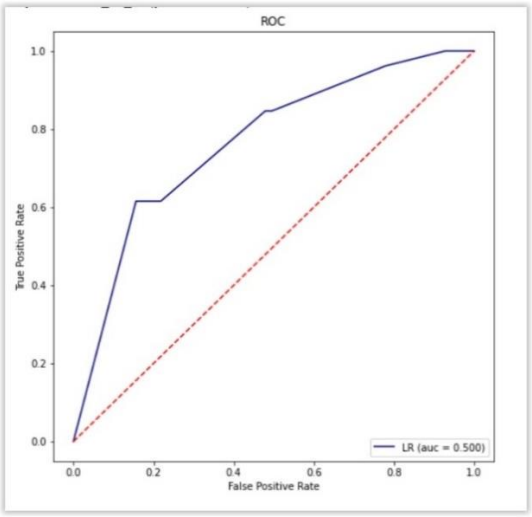
b) AdaBoost Algoritması Simülasyon Sonuçları ROC Eğrisi



c) K-NN Algoritması Simülasyon Sonuçları ROC Eğrisi

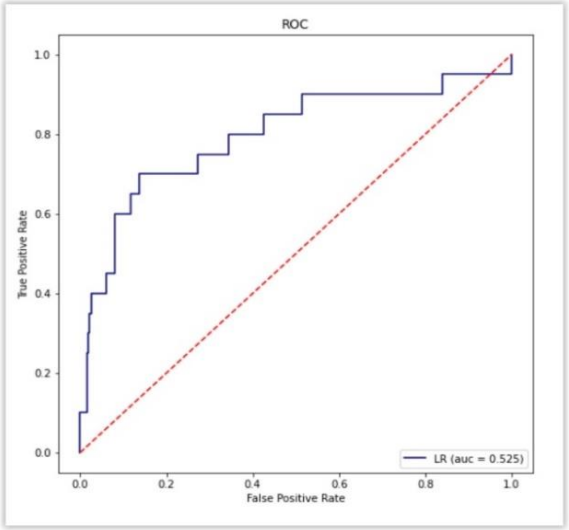


d) Desicion Tree Algoritması Simülasyon Sonuçları ROC Eğrisi

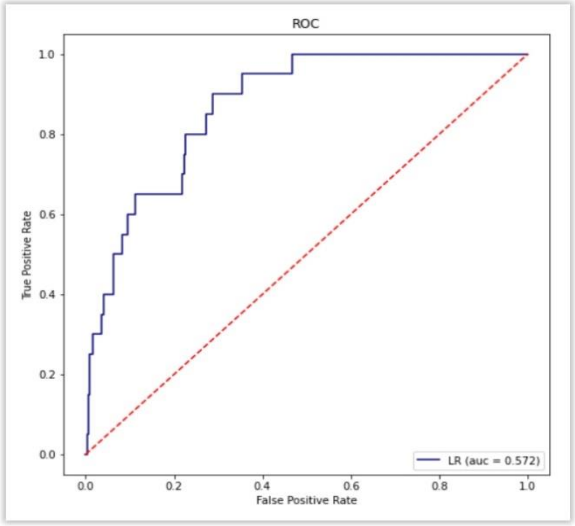


e) Naive Bayes Algoritması Simülasyon Sonuçları ROC Eğrisi

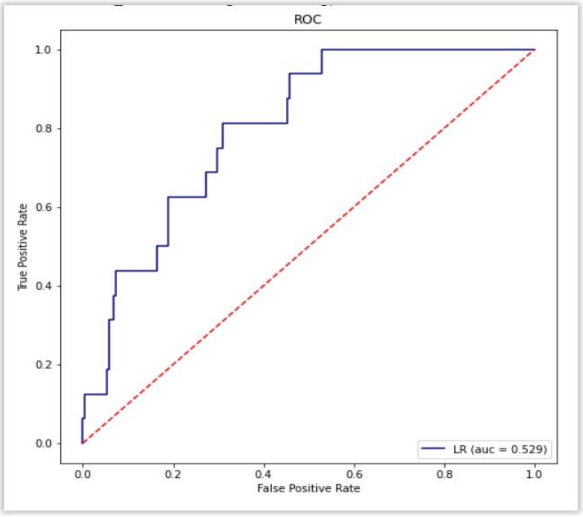
Şekil 4.27: PC1 Veri Seti Simülasyon Sonuçları ROC Eğrisi



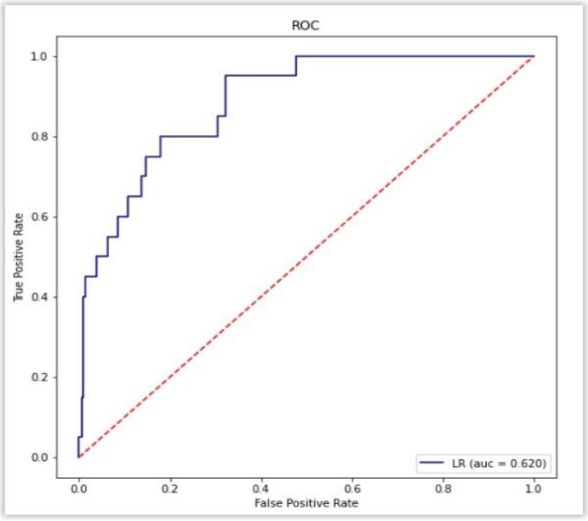
a) SVM Algoritması Simülasyon Sonuçları ROC Eğrisi



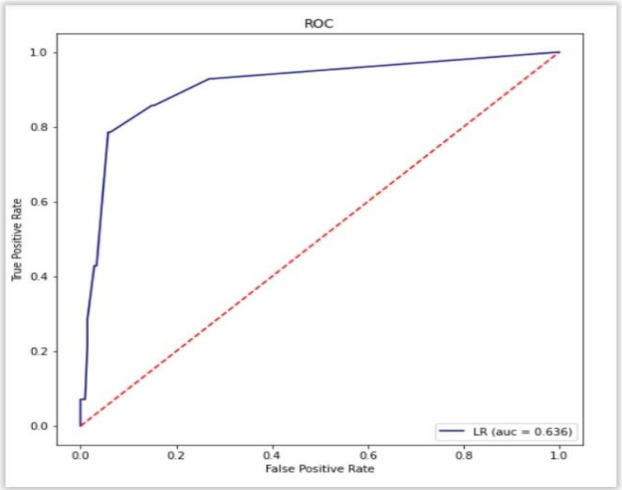
b) Logistic Regression Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Neural Network Algoritması Simülasyon Sonuçları ROC Eğrisi



d) Gradient Boosting Algoritması Simülasyon Sonuçları ROC Eğrisi



e) Bagging Algoritması Simülasyon Sonuçları ROC Eğrisi

Şekil 4.28: PC1 Veri Seti Simülasyon Sonuçları ROC Eğrisi-2

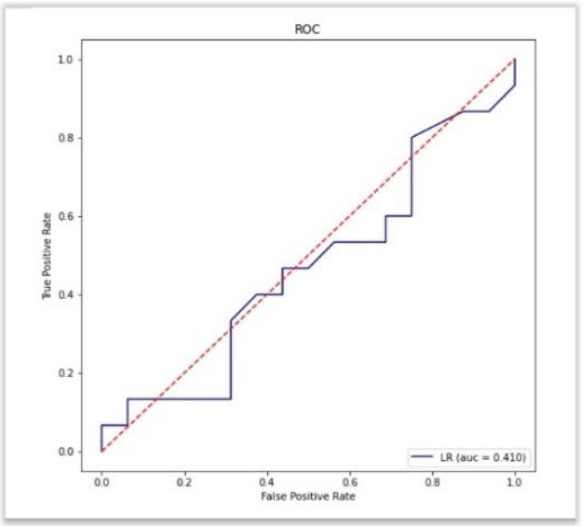
PC1'de uygulanan 10 algoritmaya bakıldığında en yüksek doğruluk oranının %94 ile RF, LR ve Gradient Boosting olduğu, en düşük doğruluk oranının ise %90 ile Desicion Tree olduğu görülmüştür. PC1'de 10 algoritmadan elde edilen simülasyon sonuçları Şekil 4.27 ve Şekil 4.28'de verilmiştir. ROC eğrisi altında kalan alandan hesaplanan AUC değerleri 0,689 ile 0,500 arasında azalarak değişmektedir. JM1, KC1, CM1 ve PC1 kullanılarak Python'da gerçeklenen 10 algoritmaya ait simülasyon sonuçları Şekil 4.21-Şekil 4.28'de verilmiştir. JM1'de RF, SVM, LR, NN ve Gradient Boosting algoritmaları %81 ile en yüksek doğruluk oranını vermiştir. KC1'de %87 doğruluk oranı ile NN, CM1'de %90 doğruluk oranı ile Bagging, PC1'de %94 doğruluk oranı ile RF, LR, Gradient Boosting en yüksek doğruluk oranlarını vermiştir. DT %74 ile JM1'de en düşük doğruluk oranını alırken, KC1'de %66 ile NB, CM1'de %81 ile DT en düşük doğruluk oranını almıştır. PC1'de ise, %90 ile en düşük doğruluk oranını yine DT algoritması almıştır. JM1'de doğruluk oranlarının ortalaması %79,9, KC1'de doğruluk oranlarının ortalaması %83 olarak hesaplanmıştır. CM1'de doğruluk oranlarının ortalaması %86,8, PC1'de doğruluk oranlarının ortalaması ise %92,5 olarak hesaplanmıştır. Ortalama doğruluk oranlarına bakıldığında en yüksek oranın %92,5 ile PC1 olduğu görülmektedir.

4.2.2. Deneysel Veri Seti Simülasyon Sonuçları

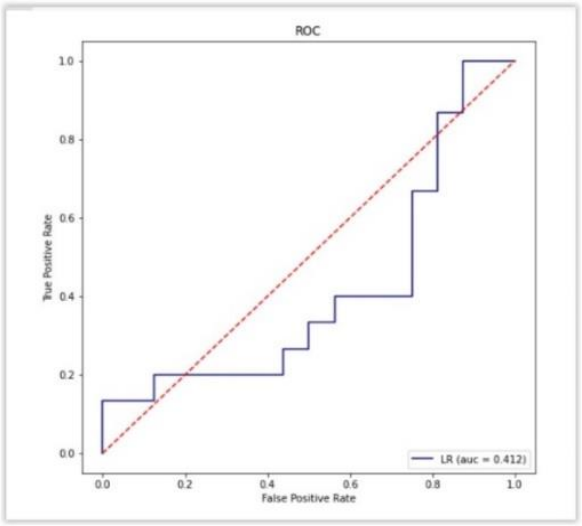
WEKA'da hata tahmini konularında başarılı sonuçlar sunan 10 algoritma (RF, AdaBoost, K-NN, DT, NB, SVM, LR, NN, Gradient Boosting ve Bagging) seçilmiş tüm algoritmalar Python scikit-learn kullanılarak uygulanmış ve performanslar 10 kat çapraz doğrulama yöntemi ile bulunmuştur. Tablo 4.4'de bu algoritmaların her veri seti için doğruluk oranları sunulmuştur. Deneysel veri setinde en yüksek doğruluk oranı %74 ile LR'de, en düşük doğruluk oranı ise %32 ile Gradient Boosting'de görülmüştür. Deneysel Veri Setinde gerçeklenen 10 algoritmanın simülasyon sonuçlarından elde edilen ROC eğrileri Şekil 4.29 ve Şekil 4.30'da verilmiştir.

Tablo 4.4: Python Deneysel Veri Seti Doğruluk Oranları

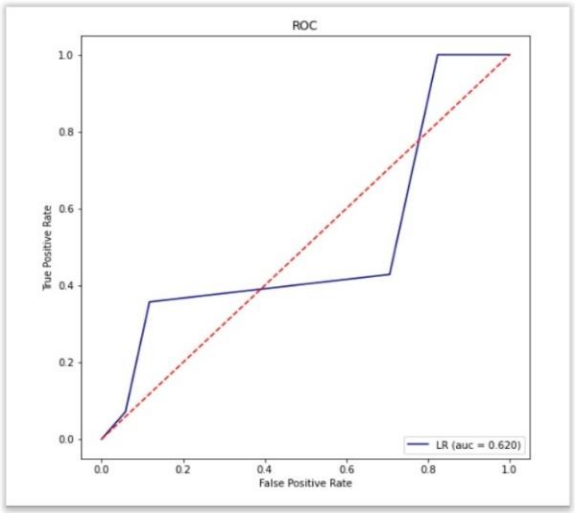
Algoritma	**Deneysel Veri Seti**
Random Forest	45
AdaBoost	45
K-NN	65
Decision Tree	39
Naive Bayes	70
SVM	71
Logistic Regression	74
Neural Network	71
Gradient Boosting	32
Bagging	43
ORTALAMA	***55,5***



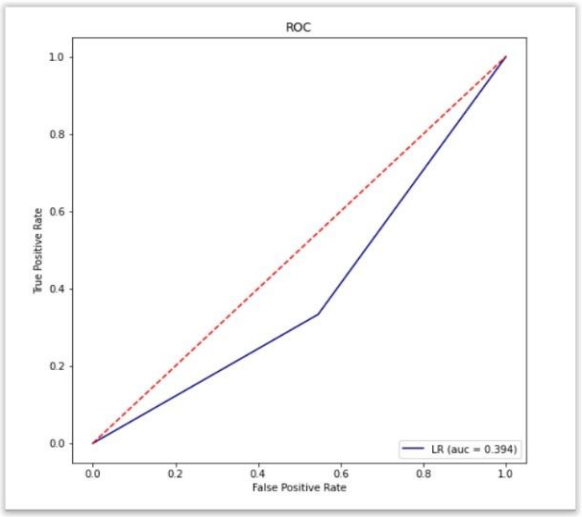
a) Random Forest Algoritması Simülasyon Sonuçları ROC Eğrisi



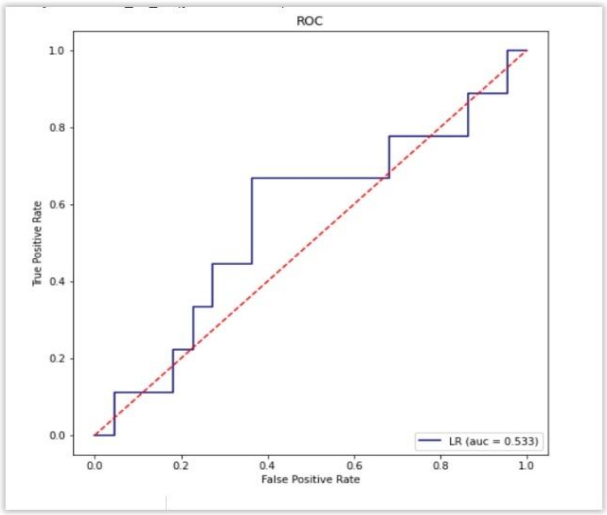
b) AdaBoost Algoritması Simülasyon Sonuçları ROC Eğrisi



c) K-NN Algoritması Simülasyon Sonuçları ROC Eğrisi

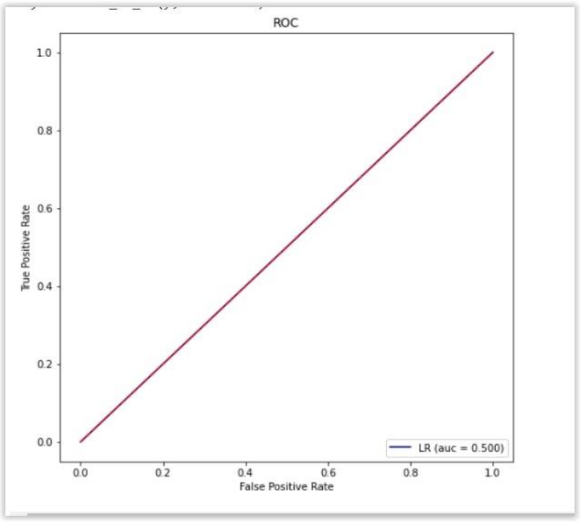


d) Desicion Tree Algoritması Simülasyon Sonuçları ROC Eğrisi

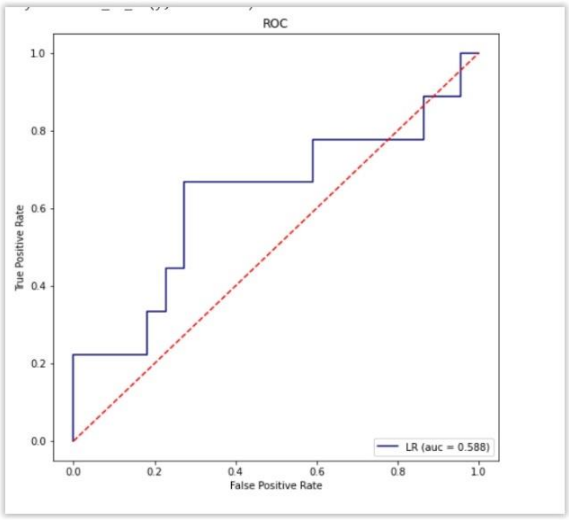


e) Naive Bayes Algoritması Simülasyon Sonuçları ROC Eğrisi

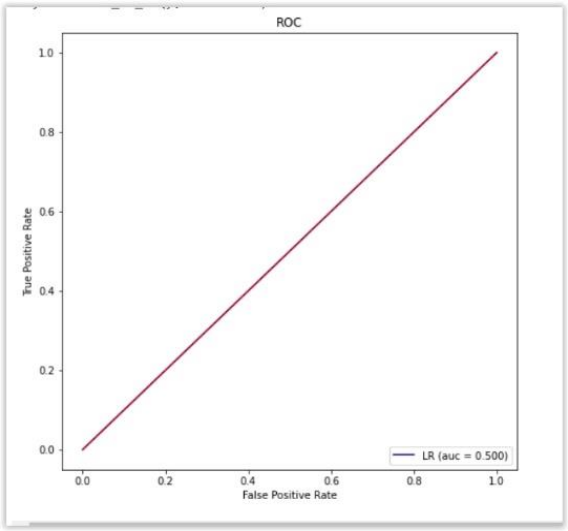
Şekil 4.29: Deneysel Veri Seti Simülasyon Sonuçları ROC Eğrisi



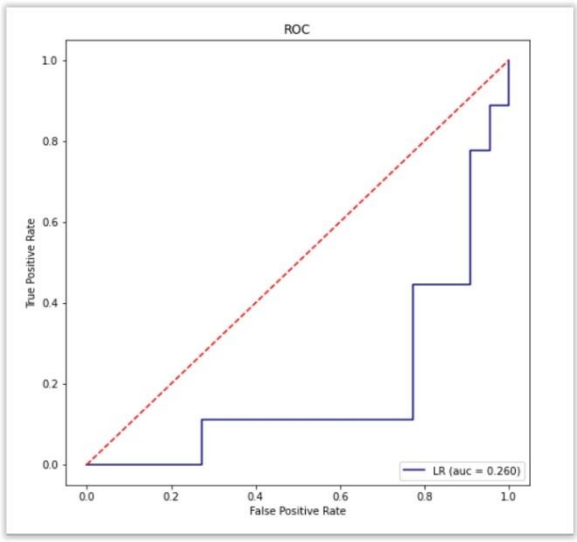
a) SVM Algoritması Simülasyon Sonuçları ROC Eğrisi



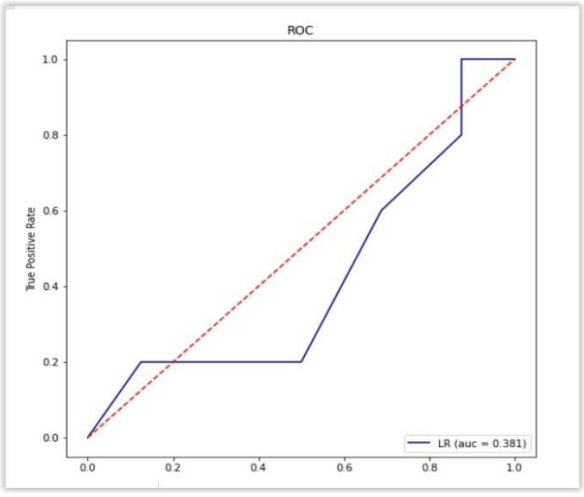
b) Logistic Regression Algoritması Simülasyon Sonuçları ROC Eğrisi



c) Neural Network Algoritması Simülasyon Sonuçları ROC Eğrisi



d) Gradient Boosting Algoritması Simülasyon Sonuçları ROC Eğrisi



e) Bagging Algoritması Simülasyon Sonuçları ROC Eğrisi

Şekil 4.30: Deneysel Veri Seti Simülasyon Sonuçları ROC Eğrisi-2

Deneysel veri setinde uygulanan 10 makine öğrenmesi algoritmasının tamamına bakıldığında en yüksek doğruluk oranının %74 ile Lojistik Regresyon algoritmasının olduğu görülmüştür. En düşük doğruluk oranının ise %32 ile Gradient Boosting algoritması olduğu görülmüştür. Deneysel veri setinde 10 algoritmadan elde edilen simülasyon sonuçları Şekil 4.29 ve Şekil 4.30'da verilmiştir. Algoritmalardan elde edilen ROC eğrisi altında kalan alandan hesaplanan AUC değerleri 0,62 ile 0,26 arasında azalarak değişmektedir. Hem literatürde hem de diğer veri setlerinde gerçeklenen makine öğrenmesi algoritmalarında oldukça yüksek doğruluk oranı sonucunu veren Random Forest algoritması, deneysel veri setinde %45 ile oldukça düşük bir doğruluk oranı elde edilmiştir. Gradient Boosting algoritması da benzer şekilde NASA veri setlerinde ve literatürde yüksek doğruluk oranına sahipken deneysel veri setinde %32 doğruluk oranı ile 10 algoritma içinden en düşük orana sahip olmuştur. Tez kapsamında kullanılan veri setleri üzerinde gerçeklenen en iyi ve en kötü algoritmalar toplu halde Tablo 4.5'de sunulmuştur.

Tablo 4.5: Veri Setleri En iyi ve En Kötü Algoritmalar

Veri Seti	**Platform**	**En Yüksek Doğruluk Oranı (%)**	**En iyi Algoritma(lar)**	**En Düşük Doğruluk Oranı (%)**	**En Kötü Algoritma(lar)**
JM1	WEKA	81,75	Random Forest	52,21	Voted Perceptron
JM1	Python	81	Random Forest, AdaBoost, SVM, Logistic Regression, Neural Network, Gradient Boosting	74	Desicion Tree
KC1	WEKA	86,67	Random Forest	69,89	Bayes Net
KC1	Python	87	Neural Network	66	Naive Bayes
CM1	WEKA	90,16	Naive Bayes Multinomial Text, Filtered Classifier, AdaboostM1, Cross Validation Parameter Selection, Multi Scheme, Random Sub Space, Stacking, Vote, Voted Perceptron Stochastic Gradient Descent Text	64,65	Bayes Net
CM1	Python	90	Bagging	81	Desicion Tree
PC1	WEKA	94,13	Bagging	74,39	Bayes Net
PC1	Python	94	Random Forest, Logistic Regression Gradient Boosting	90	Desicion Tree
Deneysel Veri Seti	WEKA	62,75	Sequential Minimal Optimization Locally Weighted Learning	45,09	Random Comittee Randomizable Filtered Classifer
Deneysel Veri Seti	Python	74	Logistic Regression	32	Gradient Boosting

4.3 WEKA ÖZNİTELİK SEÇİM METOTLARI İLE PERFORMANS DEĞERLENDİRMESİ

Yazılım hata tahmini konusunda literatürde yayınlanmış birçok yayında yer alan öznitelik seçimi konusu araştırılmıştır. Öznitelik seçimi, genellikle ilgisiz ve gereksiz öznitelikleri azaltarak sınıflandırıcının doğruluğu arttırmak amaçlı kullanılır ve veri setindeki boyutların azaltılması için verilerin ön işlemden geçirilmesi aşamalarındandır. Veriler üzerinde herhangi bir dönüşüm yapmadan, mevcut özelliklerin bir alt kümesini seçmektedir.

Özniteliklerin önemini bulmak için öznitelik seçim algoritmaları kullanılır. Öznitelik seçimi, hangi makine öğrenmesi algoritmalarında hangi özniteliklerin daha önemli olduğu ve hangi öznitelik seçimlerinde doğruluk oranlarının daha başarılı sonuçlar verdiğini ortaya koymaktadır. Özellik seçiminde kullanılan tek bir yöntem yoktur. Kullanılacak olan yöntem veri setinin durumuna göre değişkenlik gösterebilir. Farklı öznitelik seçim algoritmaları literatürde mevcuttur. Bunlar, Cfs Subset Eval, Korelasyon Niteliği, OneR, Kazanç Oranı, Bilgi Kazancı, relief-F ve Temel Bileşen gibi algoritmalardır.

Tez çalışmasında farklı özellik çıkarım metotları denenerek, seçilen en uygun iki tanesi olan Cfs Subset Eval algoritması ve Temel Bileşen özellik seçim metotları ile veri setleri üzerinde işlemler gerçekleştirilmiştir.

4.3.1. Cfs Subset Eval Öznitelik Seçim Algoritması

Cfs Subset öznitelik seçim algoritması, en iyi öznitelikleri veri setindeki değişkenleri korelasyon yardımı ile bulur. Bu algoritma aralarında düşük korelasyonlu, sınıf etiketleri arasında ise yüksek korelasyonlu öznitelikleri seçer. Bu algoritmada, yüksek korelasyonlu özellikler veri setinden çıkarılır çünkü ilişkisiz özellikler daha iyi sınıflandırma başarısı ortaya çıkarır. Algoritma, özniteliklerin öngörü yeteneği ve fazlalık derecesi temelinde önemini ölçer. Daha az karşılıklı korelasyona sahip olan ancak hedef sınıfla yüksek oranda ilişkili olan alt gruplar tercih edilir. Cfs Subset bir arama algoritması kullanmaktadır. Algoritmada nitelikler arasından, sınıf etiketi ile en iyi ilişkiye sahip olanların belirlenmesi sağlanmaktadır. Bu durumda belirlenen özellik grubu sınıf etiketi ile yüksek bir bağlantı içermekte fakat diğer niteliklerin daha önemsiz bir duruma sahip olduğu tespit edilmektedir. Dolayısıyla sınıf etiketi ile yüksek bağlantıya sahip olan nitelik grubunun kullanımının başarımı artırması beklenmektedir [61]. Tez çalışmasında veri setleri üzerinde Cfs Subset Eval algoritması ile birlikte Greedy Stepwise arama algoritması birlikte kullanılmıştır.

4.3.2. Temel Bileşen Öznitelik Seçim Algoritması

Verilerin temel bileşen analizi ve dönüşümünü gerçekleştirir. Ranker arama algoritmasıyla birlikte kullanılır. Veri setinin, kovaryans matrisinin veya tekil değer çıkarımının yöntemi ile elde edilen basitleştirilmiş halidir. Algoritma, sadece veri setlerini sadeleştirmez, verilerin birbiri ile olan ilişkisini de ortaya koyar. Böylece verilerin sonuca etkilerinin ağırlıklarının hesaplanması için de kullanılır. Amacı, orijinal öznitelikler alanını özniteliklerin ilişkisiz olduğu yeni bir alana dönüştürerek çok sayıda ilişkili öznitelik içeren veri setinin boyutluluğunu azaltmaktır. Algoritma daha sonra orijinal veri seti ile yenisi arasındaki değişimi sıralar.

4.3.3. NASA Veri Setleri ve Deneysel Veri Seti ile Öznitelik Seçim Algoritması

Tez çalışmasının bu bölümünde NASA veri setleri ve deneysel veri seti üzerinde öznitelik seçimi algoritmaların, metriklerden oluşan veri setlerinde yer alan hangi özellikler bir arada kullanıldığında veya hangi metriklerin önemli, hangi metriklerin o algoritma için önemsiz olduğunun bunun sonucu olarak da hangi algoritmaların başarı oranlarının daha yüksek olduğu öznitelik seçim algoritmalarıyla belirlenmiştir. JM1 Veri Seti Öznitelik Seçimi ile Doğruluk Oranı Değişimi Tablo 4.6'da, KC1 Veri Seti Öznitelik Seçimi ile Doğruluk Oranı Değişimi Tablo 4.7'de, CM1 Veri Seti Öznitelik Seçimi ile Doğruluk Oranı Değişimi Tablo 4.8.'de, PC1 Veri Seti Öznitelik Seçimi ile Doğruluk Oranı Değişimi Tablo 4.10'da verilmiştir.

Tablo 4.6- 4.10 arasındaki tablolarda sırasıyla * : Cfs Subset Eval+Best First (Korelasyon Tabanlı Özellik Seçici + Önce En İyi (Doğruluk Oranı %) (Özellik Seçimi Olmadan), ** : Cfs SubsetEval+Greedy Stepwise (Korelasyon Tabanlı Özellik Seçici + Açgözlü) (Doğruluk Oranı %) ve ***: Principal Componenets+Ranker (Temel Bileşen+ Sıralayıcı) (Doğruluk Oranı %) ifade etmektedir.

JM1 veri seti için, Cfs Subset Eval ve Greedy Stepwise algoritmaları birlikte seçildiğinde loc, $v(g), ev(g), iv(g), i,\ lOcomment, lOblank, lOcodeandcomment$ ve defect öznitelikleri seçilip makine öğrenmesi algoritmaları gerçeklendiğinde bazı algoritmalarda doğruluk oranlarında elde edilen değişim Tablo 4.6.'da verilmiştir. Tablo incelendiğinde, doğruluk oranlarında bir iyileşme oranı görülürken, bazı algoritmalarda çok az bir fark görülmektedir. JM1 veri seti için, Principal Components ve Ranker algoritmaları birlikte seçildiğinde $loc, v(g), ev(g), iv(g), n, v, l, d$ ve defect öznitelikleri seçilip makine öğrenmesi algoritmaları gerçeklendiğinde bazı algoritmalarda doğruluk oranlarında elde edilen değişim Tablo 4.6.'da

verilmiştir. Tablo incelendiğinde, bazı algoritmalarda doğruluk oranlarında bir iyileşme oranı görülürken, bazı algoritmaların doğruluk oranlarında çok az bir fark görülmektedir.

KC1 veri seti için, Cfs Subset Eval ve Greedy Stepwise algoritmaları birlikte seçildiğinde $v, d, i, lOcode, lOcomment,\ lOblank, uniqopnd, branchcount$ ve defect öznitelikleri seçilip makine öğrenmesi algoritmaları gerçeklendiğinde bazı algoritmalarda doğruluk oranlarında elde edilen değişim Tablo 4.7'de verilmiştir. Doğruluk oranlarında bir iyileşme oranı görülürken, bazı algoritmalarda çok az bir fark görülmektedir.

KC1 veri seti için, Principal Components ve Ranker algoritmaları birlikte seçildiğinde $loc, v(g), ev(g), iv(g), n, v, l, d$ ve defect öznitelikleri seçilip makine öğrenmesi algoritmaları gerçeklendiğinde bazı algoritmalarda bazı algoritmalarda doğruluk oranlarında elde edilen değişim Tablo 4.7'de verilmiştir. Tablo incelendiğinde, doğruluk oranlarında bir iyileşme oranı görülürken, bazı algoritmalarda çok az bir fark görülmektedir.

CM1 veri seti için, Cfs Subset Eval ve Greedy Stepwise algoritmaları birlikte seçildiğinde $loc, i(v), i, lOcomment, lOblank, uniqop, uniqopnd,$ ve defect öznitelikleri seçilip makine öğrenmesi algoritmaları bazı algoritmalarda doğruluk oranlarında elde edilen değişim Tablo 4.8'de verilmiştir. Tablo incelendiğinde, doğruluk oranlarında bir iyileşme oranı görülürken, bazı algoritmalarda çok az bir fark görülmektedir. CM1 veri seti için, Principal Components ve Ranker algoritmaları birlikte seçildiğinde $loc, v(g), ev(g), iv(g), n, v, i$ ve defect öznitelikleri seçilip makine öğrenmesi algoritmaları gerçeklendiğinde bazı algoritmalarda doğruluk oranlarında elde edilen değişim Tablo 4.8'de verilmiştir. Tablo incelendiğinde, doğruluk oranlarında bir iyileşme oranı görülürken, bazı algoritmalarda çok az bir fark görülmektedir.

PC1 veri seti için, Cfs Subset Eval ve Greedy Stepwise algoritmaları birlikte seçildiğinde v(g), $i, lOcomment, lOcodeandcomment, lOblank, uniq_opnd$ ve defect öznitelikleri seçilip makine öğrenmesi algoritmaları gerçeklendiğinde bazı algoritmalarda doğruluk oranlarında elde edilen değişim Tablo 4.9'da verilmiştir. Tablo incelendiğinde, doğruluk oranlarında bir iyileşme oranı görülürken, bazı algoritmalarda çok az bir fark görülmektedir.

PC1 veri seti için, Principal Components ve Ranker algoritmaları birlikte seçildiğinde $loc, v(g), ev(g), iv(g), n, v, i$ ve defect öznitelikleri seçilip makine öğrenmesi algoritmaları gerçeklendiğinde bazı algoritmalarda doğruluk oranlarında elde edilen değişim Tablo 4.9'da

verilmiştir. Tablo incelendiğinde, doğruluk oranlarında bir iyileşme oranı görülürken, bazı algoritmalarda çok az bir fark görülmektedir.

JM1, KC1, CM1 ve PC1 veri setleri üzerinde farklı özellik çıkarım metotları test edildiğinde, tüm veri setlerinde yer alan 22 özellik, en uygun metot olan Cfs Subset Eval algoritması ve Principal Component özellik seçim metotlarının seçilmesiyle birlikte 8 özellliğe düşürülmüştür. Daha sonra WEKA platformunda 46 adet sınıflandırma algoritmasının doğruluk oranları hesaplanmıştır. Tüm veri setlerinde doğruluk oranlarında en iyi değişim Bayes Net, Voted Perceptron, K* ve Random Forest algoritmalarında görülmüştür.

Öznitelik seçimlerinde $loc, v(g), lOcomment, lOblank$ öznitelikleri hem Cfs Subset Eval metotunda hem de Principal Component metotunda tüm makine öğrenmesi algoritmalarının doğruluk oranlarını ölçmek için kesinlikle olması gereken en önemli özellik olarak belirlenmiştir.

Deneysel veri seti için, Cfs Subset Eval ve Greedy Stepwise algoritmaları birlikte seçildiğinde sadece *loc* özelliğinin seçilmesi durumu ortaya çıkmıştır. Ancak *loc* özelliğinin tek başına seçilmesi veri seti için makine öğrenmesi algoritmalarının doğruluk oranı hesabında mantıklı bir yaklaşım olmaz. Bu yüzden Cfs Subset Eval ve Greedy Stepwise öznitelik seçim algoritmaları deneysel veri setinde kullanılmamıştır.

Deneysel veri setinde, Principal Components ve Ranker algoritmaları birlikte seçildiğinde loc, n, N, V ve defect öznitelikleri seçilip makine öğrenmesi algoritmaları gerçeklendiğinde bazı algoritmalarda doğruluk oranlarında elde edilen değişim Tablo 4.10'da verilmiştir. Tablo incelendiğinde, bazı algoritmalarda doğruluk oranlarında bir iyileşme oranı görülürken, bazı algoritmaların doğruluk oranlarında çok az bir fark görülmektedir. Tabloda sırasıyla * : Cfs Subset Eval+Best First (Korelasyon Tabanlı Özellik Seçici + Önce En İyi (Doğruluk Oranı %) (Özellik Seçimi Olmadan), ve **: Principal Componenets+Ranker (Temel Bileşen+ Sıralayıcı) (Doğruluk Oranı %) ifade etmektedir.

Tablo 4.6: JM1 Veri Seti Öznitelik Seçimi ile Doğruluk Oranı Değişimi

Algoritma	*	**	***
Bayes Net	68,05	75,71	72,00
Naive Bayes	80,42	80,41	80,31
Naive Bayes Multinomial		71,74	78,62
Naive Bayes Multinominal Text	80,65	80,65	80,65
Naive Bayes Multinomial Updateable		71,74	78,62
Naive Bayes Updatable	80,42	80,41	80,31
Logistic	81,35	80,95	81,08
Multilayer Perceptron	80,95	81,10	81,12
Stochastic Gradient Descent	80,77		80,77
Stochastic Gradient Descent Text	80,65	80,65	80,65
Simple Logistic	81,12	80,86	81,13
Sequential Minimal Optimization	80,72	80,65	80,68
Voted Perceptron	52,21	79,55	80,87
K-Nearest Neighbours Classifier	76,97	76,52	76,86
K*	78,56	80,96	80,91
Locally Weighted Learning	80,89	80,65	80,65
Iterative Classifier Optimizer	80,89	80,95	81,01
Adaboost1	80,79	80,65	80,82
Attibute Selected Classifier	80,86	81,0	81,14
Bagging	81,19	81,11	81,22
Classification via Regression	81,24	80,80	81,14
CVParameter Selection	80,65	80,65	80,65
Filtered Classsifier	81,12	81,13	81,40
Logi Boost	80,89	80,65	81,01
Multi Class Classifier	81,35	80,95	81,08
Multi Class Classifier Updatable	80,77	80,71	80,77
Multi Scheme	80,65	80,65	80,65
Random Comittee	81,03	79,54	79,95
Randomizable Filtered Classifer	75,26	76,15	75,50
Random Sub Space	81,49	81,31	81,43
Stacking	80,65	80,65	80,65
Vote	80,65	80,65	80,65
Weighted Instances Handler Wrapper	80,65	80,65	80,65
Input Mapped Classifier	80,65	80,65	80,65
Desicion Table	80,90	80,80	80,97
Repeated Incremental Pruning	81,04	81,15	81,16
OneR	79,39	79,88	79,71
Part	80,74	80,98	81,25
ZeroR	80,65	80,65	80,65
Desicion Stump	80,65	80,65	80,65
Hoeffding Tree	80,71	80,46	80,65
J48	79,50	81,00	80,93
LMT	81,24	80,91	81,20
Random Forest	81,75	81,20	81,19
Random Tree	75,47	75,66	75,06
Rep Tree	80,67	81,17	80,76

Tablo 4.7: KC1 Veri Seti Öznitelik Seçimi ile Doğruluk Oranı Değişimi

Algoritma	*	**	***
Bayes Net	69,89	75,77	76,57
Naive Bayes	82,36	82,40	82,88
Naive Bayes Multinomial	85,34	83,40	82,83
Naive Bayes Multinominal Text	84,54	84,54	84,54
Naive Bayes Multinomial Updateable	85,34	83,40	82,83
Naive Bayes Updatable	82,36	82,40	82,88
Logistic	85,68	85,20	84,63
Multilayer Perceptron	85,91	85,72	85,01
Stochastic Gradient Descent	85,20	84,44	84,49
Stochastic Gradient Descent Text	84,54	84,54	84,54
Simple Logistic	85,72	85,15	85,06
Sequential Minimal Optimization	84,77	84,49	84,54
Voted Perceptron	83,73	84,40	84,54
K-Nearest Neighbours Classifier	84,40	84,30	83,30
K*	83,97	86,29	86,53
Locally Weighted Learning	71,72	84,73	84,54
Iterative Classifier Optimizer	85,25	85,30	85,82
Adaboost1	84,96	84,92	84,49
Attibute Selected Classifier	84,30	84,16	84,11
Bagging	86,01	85,01	84,87
Classification via Regression	85,58	85,44	85,30
CVParameter Selection	84,54	84,54	84,54
Filtered Classsifier	84,87	84,54	85,15
Logi Boost	85,39	85,30	85,72
Multi Class Classifier	85,68	85,20	84,63
Multi Class Classifier Updatable	85,20	84,44	84,49
Multi Scheme	84,54	84,54	84,54
Random Comittee	85,39	84,82	84,68
Randomizable Filtered Classifer	81,31	83,92	83,07
Random Sub Space	85,49	85,20	85,11
Stacking	84,54	84,54	84,54
Vote	84,54	84,54	84,54
Weighted Instances Handler Wrapper	84,54	84,54	84,54
Input Mapped Classifier	84,54	84,54	84,54
Desicion Table	84,87	84,92	85,25
Repeated Incremental Pruning	84,54	85,06	84,58
OneR	83,68	83,68	82,78
Part	84,82	84,87	84,63
ZeroR	84,54	84,54	84,54
Desicion Stump	84,54	84,54	84,54
Hoeffding Tree	84,54	84,54	84,54
J48	84,54	84,68	84,54
LMT	85,72	85,15	85,15
Random Forest	86,67	85,44	85,58
Random Tree	82,69	82,78	83,45
Rep Tree	85,11	84,54	84,54

Tablo 4.8: CM1 Veri Seti Öznitelik Seçimi ile Doğruluk Oranı Değişimi

Algoritma	*	**	***
Bayes Net	64,65	75,50	74,89
Naive Bayes	85,34	86,54	86,74
Naive Bayes Multinomial	70,68	85,34	86,14
Naive Bayes Multinominal Text	90,16	90,16	90,16
Naive Bayes Multinomial Updateable	70,68	85,34	86,14
Naive Bayes Updatable	85,34	86,54	86,74
Logistic	88,35	89,95	88,75
Multilayer Perceptron	87,55	89,15	89,55
Stochastic Gradient Descent	89,55	90,16	90,16
Stochastic Gradient Descent Text	90,16	90,16	90,16
Simple Logistic	89,15	89,75	89,75
Sequential Minimal Optimization	89,55	90,16	90,16
Voted Perceptron	90,16	89,35	90,16
K-Nearest Neighbours Classifier	84,73	83,13	81,52
K*	87,14	85,14	86,54
Locally Weighted Learning	89,75	89,55	90,16
Iterative Classifier Optimizer	89,15	88,75	90,16
Adaboost1	90,16	90,16	90,16
Attibute Selected Classifier	89,35	89,35	90,16
Bagging	89,75	89,35	89,35
Classification via Regression	89,35	89,75	88,95
CVParameter Selection	90,16	90,16	90,16
Filtered Classsifier	90,16	90,16	90,16
Logi Boost	88,95	88,55	89,55
Multi Class Classifier	88,35	89,95	88,75
Multi Class Classifier Updatable	89,55	90,16	90,16
Multi Scheme	90,16	90,16	90,16
Random Comittee	87,75	88,15	86,94
Randomizable Filtered Classifer	85,54	84,33	84,13
Random Sub Space	90,16	90,16	90,16
Stacking	90,16	90,16	90,16
Vote	90,16	90,16	90,16
Weighted Instances Handler Wrapper	90,16	90,16	90,16
Input Mapped Classifier	90,16	90,16	90,16
Desicion Table	89,15	89,15	90,16
Repeated Incremental Pruning	89,35	87,95	89,95
OneR	88,35	88,35	88,35
Part	88,75	88,75	89,15
ZeroR	90,16	90,16	90,16
Desicion Stump	90,16	90,16	90,16
Hoeffding Tree	90,16	90,16	90,16
J48	87,95	89,35	89,55
LMT	89,15	88,55	89,15
Random Forest	88,75	88,55	88,35
Random Tree	84,33	84,73	84,13
Rep Tree	89,15	89,55	90,16

Tablo 4.9: PC1 Veri Seti Öznitelik Seçimi ile Doğruluk Oranı Değişimi

Algoritma	*	**	***
Bayes Net	74,39	87,19	85,12
Naive Bayes	89,17	89,99	89,72
Naive Bayes Multinomial	90,53	88,99	88,54
Naive Bayes Multinominal Text	93,05	93,05	93,05
Naive Bayes Multinomial Updateable	90,53	88,99	88,54
Naive Bayes Updatable	89,17	89,99	89,72
Logistic	92,42	93,14	92,87
Multilayer Perceptron	93,59	93,05	92,96
Stochastic Gradient Descent	93,05	93,05	93,05
Stochastic Gradient Descent Text	93,05	93,05	93,05
Simple Logistic	92,60	92,87	92,87
Sequential Minimal Optimization	92,96	93,05	93,05
Voted Perceptron	92,60	93,05	93,05
K-Nearest Neighbours Classifier	92,06	91,07	91,43
K*	91,79	92,42	92,33
Locally Weighted Learning	93,23	93,05	93,05
Iterative Classifier Optimizer	93,05	93,05	92,96
Adaboost1	93,05	93,05	93,05
Attibute Selected Classifier	93,41	92,96	92,96
Bagging	94,13	92,96	92,87
Classification via Regression	93,14	93,14	92,96
CVParameter Selection	93,05	93,05	93,05
Filtered Classsifier	93,50	93,05	93,05
Logi Boost	93,14	92,96	92,78
Multi Class Classifier	92,42	93,14	92,87
Multi Class Classifier Updatable	93,05	93,05	93,05
Multi Scheme	93,05	93,05	93,05
Random Comittee	93,59	92,51	92,33
Randomizable Filtered Classifer	89,45	92,15	90,44
Random Sub Space	93,86	93,14	93,14
Stacking	93,05	93,05	93,05
Vote	93,05	93,05	93,05
Weighted Instances Handler Wrapper	93,05	93,05	93,05
Input Mapped Classifier	93,05	93,05	93,05
Desicion Table	92,87	93,05	93,05
Repeated Incremental Pruning	93,32	92,51	93,14
OneR	92,87	92,60	92,60
Part	93,68	93,14	93,14
ZeroR	93,05	93,05	93,05
Desicion Stump	93,05	93,05	93,05
Hoeffding Tree	93,05	93,05	93,05
J48	93,32	92,96	92,87
LMT	92,42	92,87	92,87
Random Forest	93,68	93,32	93,23
Random Tree	91,07	90,89	91,52
Rep Tree	93,59	92,78	93,05

Tablo 4.10: Deneysel Veri Seti Öznitelik Seçimi ile Doğruluk Oranı Değişimi

Algoritma	*	**
BAYES		
Bayes Net	60,78	60,78
Naive Bayes	60,78	60,78
Naive Bayes Multinomial	59,80	57,84
Naive Bayes Multinominal Text	60,78	60,78
Naive Bayes Multinomial Updateable	59,80	57,84
Naive Bayes Updatable	60,78	60,78
FUNCTIONS		
Logistic	55,88	58,82
Multilayer Perceptron	52,94	63,72
Stochastic Gradient Descent	61,76	60,78
Stochastic Gradient Descent Text	60,78	60,78
Simple Logistic	57,84	58,82
Sequential Minimal Optimization	62,75	59,80
Voted Perceptron	56,86	60,78
LAZY		
K-Nearest Neighbours Classifier	51,96	44,11
K*	49,01	42,15
Locally Weighted Learning	62,75	60,78
META		
Iterative Classifier Optimizer	60,78	59,80
Adaboost1	60,78	55,88
Attibute Selected Classifier	59,80	60,78
Bagging	49,01	48,03
Classification via Regression	59,80	62,74
CVParameter Selection	60,78	60,78
Filtered Classsifier	60,78	60,78
Logi Boost	50,98	55,88
Multi Class Classifier	55,88	58,82
Multi Class Classifier Updatable	61,76	60,78
Multi Scheme	60,78	60,78
Random Comittee	45,09	57,84
Randomizable Filtered Classifer	45,09	40,19
Random Sub Space	60,78	60,78
Stacking	60,78	60,78
Vote	60,78	60,78
Weighted Instances Handler Wrapper	60,78	60,78
MISC		
Input Mapped Classifier	60,78	60,78
Desicion Table	59,80	60,78
Repeated Incremental Pruning	55,88	57,84
OneR	46,07	53,92
Part	61,76	60,78
ZeroR	60,78	60,78
TREE		
Desicion Stump	60,78	60,78
Hoeffding Tree	57,84	60,78
J48	61,76	60,78
LMT	57,84	59,80
Random Forest	48,03	50,62
Random Tree	50,98	58,82
Rep Tree	58,82	59,80

5. TARTIŞMA VE SONUÇ

Bu tez çalışmasında, yazılım hata tahmini konusunda farklı veri setleri üzerinde çeşitli makine öğrenmesi algoritmalarının farklı platformlarda performanslarının ölçümleri yapılmış ve değerlendirilmiştir. Bu amaçla, öncelikle yazılım hata tahmini ile ilgili detaylı literatür taraması gerçekleştirilmiştir. Literatürde bu konuda yapılan çalışmaların NASA açık veri setleri olan JM1, KC1, CM1, PC1 üzerinde en iyi performansları sergilediği için tez çalışmasının ilk bölümünde bu veri setleri kullanılmıştır. Yapılan literatür çalışmalarında, bu veri setlerinde en başarılı sonuçları Rastgele Orman, Naive Bayes, J48 algoritmalarının ortaya çıkardığı görülmüştür.

Tez çalışmasındaki uygulamaların birinci bölümünde, literatür çalışmalarına paralel olması amacıyla JM1, KC1, CM1, PC1 NASA açık veri setleri üzerinde 46 makine öğrenmesi algoritmasının WEKA veri madenciliği aracında uygulaması gerçeklenmiştir. Algoritmaların performanslarının değelendirmesi için doğruluk ve ROC eğrileri altında kalan AUC değerleri kriterleri tercih edilmiştir. Elde edilen sonuçlar incelendiğinde, JM1ve KC1 veri setinde Rastgele Orman algoritmasının, CM1 veri setinde Naive Bayes Multinomial Text algoritmasının, PC1 veri setinde ise Bagging algoritmasının en yüksek doğruluk oranlarını verdiği görülmüştür. %52,21 ile JM1 veri setinde Voted Perceptron algoritması, KC1 veri setinde %69,86 ile Bayes Net algoritması, CM1 veri setinde %64,65 ile Bayes Net algoritması, PC1 veri setinde ise, %74,39 ile en düşük doğruluk oranını yine Bayes Net algoritması sunmuştur. Tüm veri setlerinde, Rastgele Orman, Sequential Minimal Optimization, Bagging, Simple Logistic algoritmalarının %85 üzerinde bir başarı oranıyla en başarılı sonuçları verdiği, Voted Perceptron, Naive Bayes Multinominal Text, Stochastic Gradient Descent Text, OneR algoritmalarının ise doğruluk oranının oldukça düşük olduğu görülmektedir.

Uygulamaların birinci bölümünde ayrıca Bilgisayar Mühendisliği bölümü öğrencilerinin yazılım proje kodlarının çeşitli yazılım metrikleri hesaplanarak oluşturulmuş deneysel bir veri seti üzerinde 46 adet makine öğrenmesi algoritmasının WEKA veri madenciliği aracında uygulaması gerçeklenmiştir. Simülasyon sonuçlarına bakıldığında, en yüksek doğruluk oranını %62,75 ile SMO ve LWL algoritmaları vermiştir. En düşük doğruluk oranının ise %45,09 ile Random Commitee ve Randomizable Filtered algoritmalarının olduğu görülmüştür. Deneysel veri setinde incelendiğinde SMO algoritmasında ROC eğrisi altında kalan alandan hesaplanan AUC değerinin 0,52 iken, aynı doğruluk oranına sahip LWL algoritmasında ROC eğrisi altında kalan alandan hesaplanan AUC değerinin 0,47 olduğu görülmüştür. Diğer dokuz algoritmanın

da doğruluk oranlarının %62,75 ile %60,78 arasında azaldığı görülmektedir. Buna paralel şekilde AUC değerleri de 0,52 ile 0,48 arasında azalmıştır. Deneysel veri setinde gerçeklenen tüm makine öğrenmesi algoritmalarının doğruluk oranlarının ortalaması alındığında %48,52 olarak hesaplanmıştır.

Tez çalışmasındaki uygulamaların ikinci bölümünde ise NASA veri setleri ve deneysel veri setleri üzerinde Google Colab ortamında Python dilinde 10 makine öğrenmesi algoritmasının performansı değerlendirilmiştir. NASA veri setleri üzerinde gerçeklenen algoritmaların sonuçları incelendiğinde, JM1 ve veri setinde Random Forest, SVM, Logistic Regression, Neural Network ve Gradient Boosting algoritmaları %81 ile en yüksek doğruluk oranını vermiştir. KC1 veri setinde %87 doğruluk oranı ile Neural Network algoritması, CM1 veri setinde %90 doğruluk oranı ile Bagging algoritması, PC1 veri setinde ise %94 doğruluk oranı ile Random Forest, Logistic Regression, Gradient Boosting algoritmaları en yüksek doğruluk oranlarını vermiştir. JM1 veri setinde %74 ile Desicion Tree algoritması, KC1 veri setinde %66 ile Naive Bayes algoritması, CM1 veri setinde %81 ile Desicion Tree algoritması, PC1 veri setinde %90 ile yine Desicion Tree algoritması en düşük doğruluk oranını sunmuştur. JM1 veri setinde gerçeklenen tüm makine öğrenmesi algoritmalarının doğruluk oranlarının ortalaması alındığında %79,9, KC1 veri setinde gerçeklenen tüm makine öğrenmesi algoritmalarının doğruluk oranlarının ortalaması %83 olarak hesaplanmıştır. CM1 veri setinde gerçeklenen tüm makine öğrenmesi algoritmalarının doğruluk oranlarının ortalaması %86,8, PC1 veri setinde gerçeklenen tüm makine öğrenmesi algoritmalarının doğruluk oranlarının ortalaması ise %92,5 olarak hesaplanmıştır. Ortalama doğruluk oranlarına bakıldığında en yüksek oranın %92,5 ile PC1 veri seti olduğu görülmektedir.

Tez çalışmasının ikinci bölümünde ayrıca aynı platform üzerinde deneysel veri seti ile makine öğrenmesi algoritmaları simüle edilmiştir. Sonuçlar incelendiğinde hem literatürde hem de diğer veri setlerinde gerçeklenen makine öğrenmesi algoritma performanslarında oldukça yüksek doğruluk oranı sonucunu veren Random Forest algoritması, deneysel veri setinde %45 ile oldukça düşük bir doğruluk oranı elde etmiştir. Gradient Boosting algoritması da benzer şekilde NASA veri setlerinde ve literatürde yüksek doğruluk oranına sahipken deneysel veri setinde %32 doğruluk oranı ile 10 algoritma içinden en düşük orana sahip olmuştur. En yüksek doğruluk oranının %74 ile Lojistik Regression algoritmasının olduğu görülmüştür. En düşük doğruluk oranının ise %32 ile Gradient Boosting algoritması olduğu görülmüştür.

Algoritmalardan elde edilen ROC eğrisi altında kalan alandan hesaplanan AUC değerleri 0,62 ile 0,26 arasında azalarak değişmektedir.

Tez çalışmasında uygulamaların üçüncü bölümünde, literatürde de oldukça fazla kullanılan öznitelik seçim metotları NASA veri setleri üzerinde ve deneysel veri setleri üzerinde kullanılmıştır. NASA'ya ait JM1, KC1, CM1 ve PC1 veri setleri üzerinde farklı özellik çıkarım metotları test edilmiş, en iyi sonuçları Cfs Subset Eval algoritması ve Temel Bileşen özellik seçim metotlarının seçilmesiyle olduğu görülmüştür. Veri setlerindeki 22 özellik bu metotların uygulanması sonucunda 8 özelliğe düşürülmüş, daha sonra WEKA platformunda 46 adet sınıflandırma algoritmasının doğruluk oranları hesaplanmıştır. Sonuçlara bakıldığında tüm veri setlerinde doğruluk oranlarındaki en iyi değişimin Bayes Net, Voted Perceptron, K* ve Random Forest algoritmalarında olduğu görülmüştür. Öznitelik seçimlerinde $loc, v(g)$, $IOcomment, IOblank$ öznitelikleri hem CfsSubsetEval metotunda hem de Temel Bileşen metotunda tüm makine öğrenmesi algoritmalarının doğruluk oranlarını ölçmek için kesinlikle olması gereken en önemli özellik olarak belirlenmiştir.

Deneysel veri setleri üzerinde yapılan farklı özellik çıkarım metotlarında ise, sadece *loc* özelliği seçilemeyeceği için Cfs Subset Eval algoritmasında öznitelik seçimi uygulanamayacağı görülmüştür. Bu yüzden yalnızca Temel Bileşen özellik seçim metotları seçilmiştir. Veri setlerindeki 14 özellik bu metotun Ranker algoritması ile birlikte uygulanması sonucunda 5 özelliğe düşürülmüş, daha sonra WEKA platformunda 46 adet sınıflandırma algoritmasının doğruluk oranları hesaplanmıştır. Sonuçlara bakıldığında deneysel veri setlerinde doğruluk oranlarındaki en iyi değişimin Multilayer Perceptron, OneR ve Random Tree algoritmalarında olduğu görülmüştür.

NASA veri setleri ve deneysel veri setleri üzerinde uygulanan tüm öznitelik seçim metotlarında yazılım metriklerine ait loc, n, v ve *defect* özelliklerinin kesinlikle olması gerektiği görülmüştür. Her veri setini oluşturan yazılım metriklerinin hesaplamasında loc(kod satır sayısı), n(tekil operatör ve tekil operand sayısı toplamı), v(program hacmi) ve defect(hata olup olmadığı) özelliklerinin oldukça önemli olduğu açıkça belli olmuştur.

Bu tez çalışması, yazılım hata tahmini ve makine öğrenmesi algoritmaları konusunda kapsamlı bir performans değerlendirmesi sunmuştur. Bu konuda çalışmak isteyen araştırmacılara yardımcı olacağı düşünülmektedir.

KAYNAKLAR

[1]. Çatal Ç., 2008, Yazılım Kusur Kestirimi Probleminde Yapay Bağışıklık Sistemlerinin Uygulanması, Doktora, Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü.

[2]. Chidamber, S. R. ve Kemerer, C. F., (1994), "A Metrics Suite for Object-Oriented Design", 227, IEEE Transactions On Software Engineering, 20(6), 476-493.

[3]. Halstead, M., (1977), Elements of Software Science, Elsevier North-Holland, New York.

[4]. Venkata UB, Bastani BF, Yen IL (2006) A unified framework for defect data analysis using the mbr technique. Proceedings of 18th IEEE international conference on tools with artificial intelligence, 39–46.

[5]. Jiang, Y., Cukic, B. ve Yan, M., 2008 Techniques for evaluating fault prediction models. Empirical Software Engineering, 13(5), 561–595.

[6]. Menzies T, Milton Z, Burak T, Cukic B, Jiang Y, Bener et al (2010) Defect prediction from static code features: current results, limitations, new approaches. Automated Software Engineering 17(4), :375–407.

[7]. Hall T, Beecham S, Bowes D, Gray D, Counsell S (2012) A systematic review of fault prediction performance in software engineering. IEEE Transactions on Software Engineering 38(6),1276–1304.

[8]. Radjenovic D, Hericko M, Torkar R, Zivkovic A (2013) Software fault prediction metrics: a systematic literature review. Information and Software Technology 55(8):1397–1418.

[9]. Kamei Y, Shihab E (2016) Defect prediction: accomplishments and future challenges. In: Proceeding of 23rd international conference on software analysis, evolution, and reengineering, 33–45.

[10]. Denaro, G., Pezzè, M. ve Morasca, S., 2003. Towards industrially relevant faultproneness models, International Journal of Software Engineering and Knowledge Engineering, 13(4), 395–417.

[11]. Menzies, T. ve Di Stefano, J.S., 2004. How good is your blind spot sampling policy? Proceedings of IEEE international symposium on high-assurance systems engineering, 129–138.

[12]. Khoshgoftaar, T.M. ve Seliya, N., 2004. Comparative assessment of software qualityclassification techniques: An empirical case study. Empirical Software Engineering, 9(3), 229–257.

[13]. Wang, Q., Yu, B. ve Zhu, J., 2004. Extract rules from software quality prediction model based on neural network. Proceedings of IEEE international conference on tools with artificial intelligence, 191– 195.

[14]. Mahaweerawat, A., Sophatsathit, P., Lursinsap, C. ve Musilek, P., 2004. Fault prediction in object-oriented software using neural network techniques. Proceedings of InTech conference, 27–34.

[15]. Kaminsky, K. ve Boetticher, G. D., 2004. How to predict more with less, defect prediction using machine learners in an implicitly data starved domain. Proceedings of world multiconference on systemics, cybernetics and informatics.

[16]. Kanmani, S., Uthariaraj, V.R., Sankaranarayanan, V. ve Thambidurai, P., 2004. Object oriented software quality prediction using general regression neural networks. SIGSOFT Software Engineering Notes, 29(5), 1–6.

[17]. Zhong, S., Khoshgoftaar, T. ve Seliya, N., 2004. Unsupervised learning for expertbased software quality estimation. Proceedings of IEEE international symposium on high assurance systems engineering, 149–155.

[18]. Xing, F., Guo, P. ve Lyu, M.R., 2005. A novel method for early software quality prediction based on support vector machine. Proceedings of IEEE international symposium on software reliability engineering, 213–222.

[19]. Koru, A.G. ve Liu, H. 2005. An investigation of the effect of module size on defect prediction using static measures. Proceedings of Workshop on predictor models in software engineering, 1–5.

[20]. Khoshgoftaar, T. M., Seliya, N. ve Gao, K., 2005. Assessment of a new three-group software quality classification technique: An empirical case study. Empirical Software Engineering, 10(2), 183–218.

[21]. Koru, A.G. ve Liu, H., 2005. Building effective defect prediction models in practice. IEEE Software, 22(6), 23–29.

[22]. Challagulla, V.U.B., Bastani, F.B., Yen, I.L. ve Paul, R. A., 2005. Empirical assessment of machine learning based software defect prediction techniques. Proceedings of IEEE international workshop on object-oriented real-time dependable systems, 263–270.

[23]. Gyimothy, T., Ferenc, R. ve Siket, I., 2005. Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Transactions on Software Engineering, 31(10), 897–910.

[24]. Tomaszewski, P., Lundberg, L. ve Grahn, H., 2005. The accuracy of early fault prediction in modified code. Proceedings of Conference on software engineering research and practice in Sweden, 57–63.

[25]. Hassan, A.E. ve Holt, R.C., 2005. The top ten list: Dynamic fault prediction. Proceedings of IEEE international conference on software maintenance, 263–272.

[26]. Ma, Y., Guo, L. ve Cukic, B., 2006. A statistical framework for the prediction of faultproneness. Advances in machine learning application in software engineering. Idea Group Inc. 237–265.

[27]. Challagulla, V.U.B., Bastani, F.B. ve Yen, I., 2006. A unified framework for defect data analysis using the MBR technique. Proceedings of IEEE international conference on tools with artificial intelligence, 39–46.

[28]. Khoshgoftaar, T. M., Seliya, N. ve Sundaresh, N., 2006. An empirical study of predicting software faults with case based reasoning. Software Quality Journal, 14(2), 85–111.

[29]. Nikora, A. Ve Munson, J.C., 2006. Building high-quality software fault predictors. Software-Practice and Experience, 36(9), 949–969.

[30]. Zhou, Y. ve Leung, H., 2006. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. IEEE Transactions on Software Engineering, 32(10), 771–789.

[31]. Boetticher, G., 2006. Improving credibility of machine learner models in software engineering. Advanced machine learner applications in software engineering. Series on software engineering and knowledge engineering.

[32]. Mertik, M., Lenic, M., Stiglic, G. ve Kokol, P., 2006. Estimating software quality with advanced data mining techniques. Proceedings of International conference on software engineering advances, 19.

[33]. Bibi, S., Tsoumakas, G., Stamelos, I. ve Vlahvas, I. (2006). Regression via classification applied on software defect estimation. Expert Systems with Applications, 34(3), 2091–2101.

[34]. Gao, K. ve Khoshgoftaar, T. G., 2007. A comprehensive empirical study of count models for software fault prediction. IEEE Transactions for Reliability, 56(2), 223–236

[35]. Li, Z. ve Reformat, M., 2007. A practical method for the software fault-prediction. Proceedings of IEEE international conference on information reuse and integration, 659–666.

[36]. Yang, B., Yao, L. ve Huang, H.Z., 2007. Early software quality prediction based on a fuzzy neural network model. Proceedings of International conference on natural computation, 760–764.

[37]. Pai, G. P. ve Dugan, J.B., 2007. Empirical analysis of software fault content and fault proneness using Bayesian methods. IEEE Transactions on Software Engineering, 33(10), 675–686.

[38]. Wang, Q., Zhu, J. ve Yu, B., 2007. Feature selection and clustering in software quality prediction. Proceedings of 11th international conference on evaluation and assessment in software engineering.

[39]. Seliya, N. ve Khoshgoftaar, T.M., 2007. Software quality estimation with limited fault data: A semi supervised learning perspective. Software Quality Journal,15(3), 327–344

[40]. Koru, A.G. ve Liu, H., 2007. Identifying and characterizing change-prone classes in two large-scale open-source products. Journal of Systems and Software, 80(1), 63–73.

[41]. Cukic, B. ve Ma, Y., 2007. Predicting fault-proneness: Do we finally know how? In Reliability analysis of system failure data.

[42]. Tomaszewski, P., Håkansson, J., Grahn, H. ve Lundberg, L., 2007. Statistical models vs. expert estimation for fault prediction in modified code – An industrial case study. Journal of Systems and Software, 80(8), 1227–1238.

[43]. Seliya, N. ve Khoshgoftaar, T.M., 2007. Software quality analysis of unlabeled program modules with semisupervised clustering. IEEE Transactions on Systems, Man, and Cybernetics, 37(2), 201–211.

[44]. Olague, H.M., Etzkorn, L.H., Gholston, S. ve Quattlebaum, S., 2007. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. IEEE Transactions on Software Engineering, 33(6), 402–419.

[45]. Binkley, D., Feild, H., Lawrie, D. ve Pighin, M., 2007. Software fault prediction using language processing. Testing: Industrial and academic conference. Practice and research techniques. Cumberland Lodge, Windsor, UK: IEEE Press.

[46]. Jiang, Y., Cukic, B. ve Menzies, T., 2007. Fault prediction using early lifecycle data. Proceedings of IEEE international symposium on software reliability, 237–246.

[47]. Yang, B., Yin, Q., Xu, S. ve Guo, P., 2008. Software quality prediction using affinity propagation algorithm. Proceedings of International Joint Conference on Neural Networks, 1891–1896.

[48]. Marcus, A., Poshyvanyk, D. ve Ferenc, R., 2008. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. IEEE Transactions Software Engineering, 34(2), 287–300.

[49]. Shafi, S., Hassan, S.M. Arshaq, A., Khan, M.J. ve Shamail, S., 2008. Software quality prediction techniques: A comparative analysis. Proceedings of international conference on emerging technologies, 242–246.

[50] Riquelme, J.C., Ruiz, R., Rodríguez, D. ve Moreno, J., 2008. Finding defective modules from highly unbalanced datasets. Actas del 8_ taller sobre el apoyo a la decisión en ingeniería del software, 67–74.

[51]. Shatnawi, R. ve Li, W., 2008. The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. Journal of Systems and Software, 81(11), 1868–1882.

[52]. Arisholm, E., Briand, L.C. ve Johannessen, E.B., 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems and Software, 83(1), 2–17.

[53]. Catal, C. ve Diri, B., 2009. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Sciences, 179(8), 1040–1058.

[54]. Turhan, B. ve Bener, A., 2009. Analysis of Naive Bayes' assumptions on software fault data: An empirical study. Data Knowledge Engineering, 68(2), 278–290.

[55]. Hall, T., Beecham, S., Bowes, D., Gray, D. ve Counsell, S., 2012. A Systematic Literature Review on Fault Prediction Performance in Software Engineering, Proceedings of IEEE Transactions on Software Engineering, 38(6), 1276–1304.

[56] Promise Software Engineering Repository, http://promise.site.uottawa.ca/ SERepository/datasets-page.html. [Ziyaret Tarihi:09.04.2019.]

[57]. Varol Malkoçoğlu Berika A., 2020, Akut Lenfoblastik Lösemi Hücrelerinin Derin Öğrenme Yöntemleri İle Sınıflandırılması, Yüksek Lisans, Ondokuz Mayıs Üniversitesi Lisansüstü Eğitim Ensitütüsü.

[58]. Karslı Burcu Ö., 2019, Makine Öğrenme Yöntemleri İle Karaciğer Hastalığının Teşhisi, Yüksek Lisans, Ağrı İbrahim Çeçen Üniversitesi Fen Bilimleri Enstitüsü.

[59]. Altınsoy F., 2019, Uzaktan Eğitim Öğrencilerinin Başarılarının Yapay Zeka Teknikleri ile Tahmini,Yüksek Lisans Tezi, Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü.

[60]. WEKA The Workbench for machine learning**,** http://old-www.cms.waikato. ac.nz/ml/weka. [Ziyaret Tarihi: Kasım 2019].

[61]. Gümüşçü, İ. B. Aydilek ve R. Taşaltın, "Mikro-dizilim Veri Sınıflandırmasında Öznitelik Seçme Algoritmalarının Karşılaştırılması," Harran Üniversitesi Mühendislik Dergisi, cilt 1, sayı 1, s. 1-7, 2016.