YAZILIM TEST SÜREÇLERİNDE HATA KRİTİKLİK SEVİYELERİNİN MAKİNE ÖĞRENMESİ YÖNTEMLERİYLE BELİRLENMESİ

Adem ÖRPEK
241402205

YÜKSEK LİSANS TEZİ
Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı
Danışman: Dr. Öğr. Üyesi Önder ŞAHİNASLAN

İstanbul
T.C. Maltepe Üniversitesi
Lisansüstü Eğitim Enstitüsü
Haziran, 2026

YAZILIM TEST SÜREÇLERİNDE HATA KRİTİKLİK SEVİYELERİNİN MAKİNE ÖĞRENMESİ YÖNTEMLERİYLE BELİRLENMESİ

Adem ÖRPEK
241402205
ORCID: 0009-0004-5040-1600

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı
Danışman: Dr. Öğr. Üyesi Önder ŞAHİNASLAN

İstanbul
T.C. Maltepe Üniversitesi
Lisansüstü Eğitim Enstitüsü
Haziran, 2026



JÜRİ VE ENSTİTÜ ONAYI

ADEM ÖRPEK'ın "Yazılım Test Süreçlerinde Hata Kritiklik Seviyelerinin Makine Öğrenmesi Yöntemleriyle Belirlenmesi" başlıklı tezi 25.06.2026 tarihinde aşağıdaki jüri tarafından değerlendirilerek "Maltepe Üniversitesi Lisansüstü Eğitim ve Öğretim Yönetmeliği" nin ilgili maddeleri uyarınca Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans/Doktora/Sanatta Yeterlik tezi oy birliğiyle/oy çokluğuyla, başarılı/başarısız olarak kabul edilmiştir.

	Unvanı. Adı ve Soyadı	İmza
Üye (Tez Danışmanı)	Dr. Öğr. Üyesi Önder ŞAHİNASLAN Maltepe Üniversitesi	.....
Üye	Doç. Dr. Fatih YÜCALAR Manisa Celal Bayar Üniversitesi	.....
Üye	Dr. Öğr. Üyesi Erdal YAŞLICA Maltepe Üniversitesi	.....

Prof. Dr. Selva ERSÖZ KARAKULAKOĞLU
Enstitü Müdürü

TEŞEKKÜR

Bu tez çalışmam süresince beni her zaman destekleyen, aynı zamanda bu süreçte sınıf arkadaşlığı da yapan sevgili eşim Zeynep'e en içten teşekkürlerimi sunarım. Eğitim hayatım boyunca her zaman yanımda olan, desteklerini hiçbir zaman esirgemeyen değerli aileme ve tez çalışmam süresince bilgi ve katkılarıyla bana yol gösteren danışman hocam Sayın Dr. Öğr. Üyesi Önder ŞAHİNASLAN'a sonsuz teşekkür ederim.

Adem ÖRPEK

Haziran, 2026

ÖZET

YAZILIM TEST SÜREÇLERİNDE HATA KRİTİKLİK SEVİYELERİNİN MAKİNE ÖĞRENMESİ YÖNTEMLERİYLE BELİRLENMESİ

Adem Örpek

Yüksek Lisans Tezi

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı

Danışman: Dr. Öğr. Üyesi Önder Şahinaslan

Maltepe Üniversitesi Lisansüstü Eğitim Enstitüsü, 2026

Bu çalışmada, yazılım test süreçlerinde tespit edilen hata kayıtlarının kritiklik seviyelerinin makine öğrenmesi yöntemleriyle otomatik olarak belirlenmesi amaçlanmıştır. Yazılım geliştirme yaşam döngüsünde test süreci, yazılım kalitesinin sağlanması, hataların erken tespit edilmesi ve çözüm önceliklerinin doğru belirlenmesi açısından önemlidir. Ancak hata kritiklik seviyelerinin çoğunlukla test uzmanları tarafından manuel olarak atanması, tecrübe ve yorum farklılıkları gibi insan kaynaklı nedenlerle tutarsızlıklara yol açabilmektedir. Çalışma kapsamında, bir e-ticaret mobil uygulamasına ait yazılım test süreçlerinde oluşturulan hata kayıtları kullanılmıştır. Çalışmada Support Vector Machine, Naive Bayes, Random Forest, Decision Tree, XGBoost ve LightGBM algoritmaları karşılaştırılmıştır. Ayrıca sınıf dengesizliğinin model performansı üzerindeki etkisini azaltmak amacıyla Easy Data Augmentation yöntemi uygulanmıştır. Elde edilen sonuçlar, sınıf dengesizliğinin özellikle az örnekli Blocker sınıfının tahmin başarısını olumsuz etkilediğini göstermiştir. Easy Data Augmentation yöntemi sonrasında ise modellerin azınlık sınıflarındaki performansında belirgin iyileşmeler elde edilmiştir. Deneyler sonucunda, XGBoost algoritmasının hem Macro F1 hem de Blocker sınıfı F1-skoru açısından en başarılı model olduğu belirlenmiştir. Sonuç olarak, makine öğrenmesi yöntemlerinin hata kritiklik seviyelerinin daha tutarlı, hızlı ve objektif biçimde belirlenmesine katkı sağlayabileceği görülmüştür.

Anahtar Sözcükler: Yazılım Testi, Hata Kritiklik Seviyesi, Makine Öğrenmesi

ABSTRACT

DETERMINING DEFECT CRITICALITY LEVELS IN SOFTWARE TESTING PROCESSES USING MACHINE LEARNING METHODS

Adem Örpek
Master Thesis
Department of Computer Engineering
Master Program in Computer Enginnering with Thesis
Thesis Advisor: Assist. Prof. Dr. Önder Şahinaslan
Maltepe University Graduate School, 2026

In this study, it was aimed to automatically determine the criticality levels of defect records identified in software testing processes using machine learning methods. In the software development life cycle, the testing process is important for ensuring software quality, detecting defects at an early stage, and determining solution priorities accurately. However, the fact that defect criticality levels are mostly assigned manually by test specialists may lead to inconsistencies due to human-related factors such as experience and differences in interpretation. Within the scope of the study, defect records generated during the software testing processes of an e-commerce mobile application were used. In the study, Support Vector Machine, Naive Bayes, Random Forest, Decision Tree, XGBoost, and LightGBM algorithms were compared. In addition, the Easy Data Augmentation method was applied to reduce the effect of class imbalance on model performance. The results showed that class imbalance negatively affected the prediction performance, especially for the minority Blocker class. After Easy Data Augmentation, significant improvements were obtained in the performance of the models for minority classes. As a result of the experiments, the XGBoost algorithm was determined to be the most successful model in terms of both Macro F1 and the F1-score of the Blocker class. Consequently, it was observed that machine learning methods can contribute to determining defect criticality levels in a more consistent, faster, and objective manner.

Keywords: Software Testing, Defect Criticality Level, Machine Learning

İÇİNDEKİLER

JÜRİ VE ENSTİTÜ ONAYI ii

TEŞEKKÜR iii

ÖZET iv

ABSTRACT v

İÇİNDEKİLER vi

TABLOLAR LİSTESİ ix

ŞEKİLLER LİSTESİ xi

KISALTMALAR xii

1. GİRİŞ 1

1.1 Problem 7

1.2 Amaç 9

1.3 Önem 10

1.4 Varsayımlar 10

1.5 Sınırlıklar 11

1.6 Tanımlar 11

2. LİTERATÜR ARAŞTIRMASI 13

2.1 Yazılım Test Süreci Üzerine Yapılan Çalışmalar 13

2.2 Yazılım Test Sürecinde Makine Öğrenmesi Üzerine Yapılan Çalışmalar 15

3. YÖNTEM 24

3.1. Veri Toplama 26
3.2. Veri Temizleme ve Ön İşleme 28
3.3. Özellik Seçimi 31
3.4. Model Seçimi 32
3.4.1. Destek vektör makinesi 33
3.4.2. Naive Bayes 34
3.4.3. Rastgele orman 35
3.4.4. Karar Ağacı 36
3.4.5. XGBoost 37
3.4.6. LightGBM 38
3.5. Model Eğitimi 39
3.5.1. Destek Vektör Makinesi 40
3.5.2. Naive Bayes 44
3.5.3. Rastgele Orman 47
3.5.4. Karar Ağacı 50
3.5.5. XGBoost 53
3.5.6. LightGBM 56
3.6. Model Değerlendirme 59
3.7. Model İyileştirme 61
3.8. Model Dağıtım ve İzleme 61
4. BULGULAR VE TARTIŞMA 63

4.1 Bulgular 63
4.1.1 Destek vektör makinesi modeli sonuçları 63
4.1.2 Naive Bayes modeli sonuçları 64
4.1.3 Rastgele orman modeli sonuçları 65
4.1.4 Karar ağacı modeli sonuçları 66
4.1.5 XGBoost modeli sonuçları 67
4.1.6 LightGBM modeli sonuçları 68
4.1.7 Modellerin karşılaştırmalı değerlendirilmesi 69
4.2 Tartışma 75
5. SONUÇ ve ÖNERİLER 79
KAYNAKÇA 84
ÖZ GEÇMİŞ 92

TABLOLAR LİSTESİ

Tablo 1. Makine Öğrenmesi Tabanlı Hata Tahmini Çalışmalarının Özeti 18

Tablo 2. Hata Kayıtlarının Kritiklik Seviyesine Göre Dağılımı 27

Tablo 3. Veri Temizleme Süreci Sonrası Veri Dağılımı 31

Tablo 4. Özellik Seçimi Aşamasında Belirlenen Nitelikler 32

Tablo 5. Eğitim ve Test Veri Seti Ayrımının Şematik Gösterimi 39

Tablo 6. Eğitim ve Test Veri Setinin Sınıf Bazlı Dağılımı 39

Tablo 7. Dengesiz Veri Kümesi ile Eğitilen SVM Modelinin Performans Metrikleri .. 40

Tablo 8. EDA Sonrası Dengelenmiş Eğitim Veri Kümesi Dağılımı 44

Tablo 9. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen SVM Modelinin Performans Metrikleri 44

Tablo 10. Dengesiz Veri Kümesi ile Eğitilen NB Modelinin Performans Metrikleri ... 45

Tablo 11. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen NB Modelinin Performans Metrikleri 47

Tablo 12. Dengesiz Veri Kümesi ile Eğitilen RF Modelinin Performans Metrikleri 48

Tablo 13. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen RF Modelinin Performans Metrikleri 50

Tablo 14. Dengesiz Veri Kümesi ile Eğitilen DT Modelinin Performans Metrikleri.... 51

Tablo 15. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen DT Modelinin Performans Metrikleri 53

Tablo 16. Dengesiz Veri Kümesi ile Eğitilen XGBoost Modelinin Performans Metrikleri 54

Tablo 17. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen XGBoost Modelinin Performans Metrikleri 56

Tablo 18. Dengesiz Veri Kümesi ile Eğitilen LightGBM Modelinin Performans Metrikleri 57

Tablo 19. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen LightGBM Modelinin Performans Metrikleri 59

Tablo 20. Sınıflandırma Modellerinde Kullanılan Performans Ölçütleri ve Açıklamaları 59

Tablo 21. EDAsız Veri Seti Üzerinde Modellerin Genel Performans Karşılaştırması .. 69

Tablo 22. EDA Uygulanmış Veri Seti Üzerinde Modellerin Genel Performans Karşılaştırması 70

Tablo 23. EDA Öncesi ve Sonrası Model Performanslarının Karşılaştırılması 72

Tablo 24. Blocker Sınıfı İçin EDA Öncesi ve Sonrası F-1 Skorlarının Karşılaştırılması 73

Tablo 25. EDA Öncesi ve Sonrası En Başarılı Modellerin Karşılaştırılması 74

ŞEKİLLER LİSTESİ

Şekil 1. Yazılım Geliştirme Yaşam Döngüsü 2

Şekil 2. Makine Öğrenmesi Türleri 24

Şekil 3. Hata Kritiklik Seviyesi Tahmin Sürecinin Genel Akışı 25

Şekil 4. Veri Temizleme ve Ön İşleme Sürecinin Akış Diyagramı 30

Şekil 5. SVM Modeline Ait Örnek Sınıflandırma Yapısı 34

Şekil 6. Bayes Teoremi'nin Matematiksel İfadesi 35

Şekil 7. RF Sınıflandırma Şeması 36

Şekil 8. DT Temel Yapısı 37

Şekil 9. EDA Uygulaması ve SVM Model Eğitimi Kod Kesiti 43

Şekil 10. EDA Uygulanmış Eğitim Verisi ile NB Model Eğitimi Kod Kesiti 46

Şekil 11. EDA Uygulanmış Eğitim Verisi ile RF Model Eğitimi Kod Kesiti 49

Şekil 12. EDA Uygulanmış Eğitim Verisi ile DT Model Eğitimi Kod Kesiti 52

Şekil 13. EDA Uygulanmış Eğitim Verisi ile XGBoost Model Eğitimi Kod Kesiti 55

Şekil 14. EDA Uygulanmış Eğitim Verisi ile LightGBM Model Eğitimi Kod Kesiti 58

Şekil 15. Modellerin EDA Öncesi ve Sonrası Macro F1 Performans Karşılaştırması 80

Şekil 16. Modellerin EDA Öncesi ve Sonrası Blocker F1 Performans Karşılaştırması. 81

KISALTMALAR

Kısaltma		Türkçe	İngilizce
DT	:	Karar Ağacı	Decision Tree
EDA	:	Kolay Veri Artırma	Easy Data Augmentation
LIGHTGBM	:	Hafif Gradyan Artırma Makinesi	Light Gradient Boosting Machine
NB	:	Naive Bayes	Naive Bayes
NLP	:	Doğal Dil İşleme	Natural Language Processing
RF	:	Rastgele Orman	Random Forest
SDLC	:	Yazılım Geliştirme Yaşam Döngüsü	Software Development Life Cycle
SVM	:	Destek Vektör Makinesi	Support Vector Machine
XGBOOST	:	Aşırı Gradyan Artırma	Extreme Gradient Boosting

1. GİRİŞ

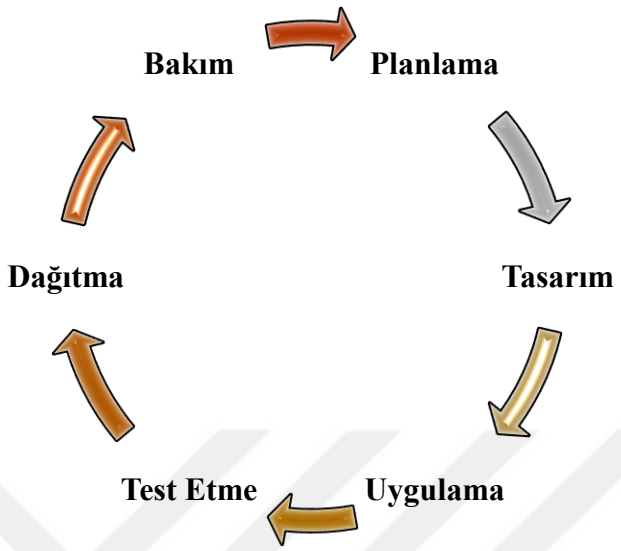
Teknolojinin hızla gelişmesiyle beraber, insanların günlük hayatta karşılaştıkları problemlerin çözülmesi ve iyileştirilmesi konusunda yazılımların katkısı oldukça artmıştır. Yazılımlar, iş süreçlerini daha verimli hale getirmekle kalmayıp aynı zamanda eğitim, iletişim, eğlence, sağlık gibi birçok farklı alanda da hayatı kolaylaştırma düzeyine gelmiştir. Bu denli gelişim gösteren yazılımlar sonucunda ise, insanların günlük yaşamlarının vazgeçilmez bir parçası haline gelmiş ve yer, zaman fark etmeden insanlar yazılımlara başvurur hale gelmiştir. Yazılımlar kullanıldıkça insan alışkanlıklarında değişimler başlamış ve tüm alanlarda yazılımlara duyulan ihtiyaç sürekli artış göstermiştir (Şahinaslan ve diğerleri, 2023).

Yazılım dünyasındaki bu hızlı dönüşüm, yazılım geliştirme standartlarını da önemli ölçüde değiştirmiştir. Artık yazılım geliştirme ekipleri, bir programın uçtan uca geliştirilmesini ve çalıştırılmasını sağlamakla kalmayıp, kullanıcıların ihtiyaçlarını karşılayan, güvenli, kullanıcı dostu ve verimli yazılımların üretilmesini temel bir hedef olarak belirlemişlerdir. Bu nedenle, yazılımların başarısı, büyük ölçüde kullanıcı ihtiyaçlarının doğru bir şekilde anlaşılmasına ve bu ihtiyaçlara yönelik uygun yazılım üretilmesine dayanmaktadır. Bu nedenle, yazılım geliştirme süreçlerinde yazılım kalitesinin sağlanması, sürecin en kritik aşamalarından biri haline gelmiştir. Yazılım kalitesinin sistemli ve kontrollü bir biçimde sağlanabilmesi için yazılım geliştirme süreçlerinin belirli bir metodoloji çerçevesinde işletilmesi gerekmektedir. Bu metodolojilerden en yaygın kullanılanı SDLC (Yazılım Geliştirme Yaşam Döngüsü, Software Development Life Cycle) yaklaşımıdır.

SDLC, yazılımın planlama, tasarım, geliştirme, test etme, dağıtım ve bakım aşamalarını içeren kapsamlı bir süreçtir. Bu döngü, yazılımın başlangıcından son kullanıcının kullanımına sunulana kadar geçen her aşamayı kapsar ve her bir aşama, yazılımın kalitesini iyileştirip, kullanıcı memnuniyeti sağlamak için kritik bir öneme sahiptir.

Platform fark etmeksizin yazılım geliştirme süreçlerinde SDLC, tüm proje ekiplerinin ortak bir amaç doğrultusunda çalışmasını sağlayan evrensel bir yöntem olarak benimsenmiştir. Bu sürecin her adımında sağlanan başarı sonucunda yazılımda büyük bir

kalite güvencesi sağlanmış olur. SDLC sürecinin adımları Şekil 1'de gösterilmektedir (AWS, 2025).



Şekil 1. Yazılım Geliştirme Yaşam Döngüsü

Başlangıç olan planlama aşaması, yazılım geliştirme sürecinin ilk ve oldukça kritik bir adımıdır. Bu aşamada, projenin amacı, hedefi, bu hedeflere ulaşmak için nasıl bir yol izleneceğinin belirlenmesi ve proje için kaynakların temin edilmesi sağlanır. Geliştirme ekibi projeyi değerlendirir, risk oluşabilecek durumları aktarır ve bu durumlara karşı alınabilecek olan önlemler hakkında beyin fırtınası yapılır. Planlamanın olabilecek en iyi şekilde yürütülmesi hem proje ekibinin motivasyonunu artırırken hem de diğer aşamaların daha sorunsuz şekilde ilerlemesine yardımcı olacağından oldukça önem verilmektedir. Kalite güvence gereksinimi, risk tanımı ve fizibilite raporu bu aşamanın en önemli çıktılarıdır (Gupta, 2021) .

Tasarım aşaması, planlanmış olan yazılımın nasıl hayata geçirileceğini belirleyen ve mimarisinin şekillendirildiği aşamadır. SDLC sürecinin en yaratıcı aşamasıdır (Dora ve Dubey, 2013). Bu süreçte yazılım mühendisleri, planlama aşamasında ortaya çıkarılan gereksinimlere göre yazılımın stratejisini belirler, kodlama için uygun araçları seçer ve görev paylaşımını modüllere göre gerçekleştirir. Analiz uzmanları, mevcut gereksinimlerin yazılımda nasıl gerçekleştirilebileceğini değerlendirir ve teknik dokümanlar oluşturur. Test mühendisleri ise, kullanıcı ihtiyaçlarına göre test

senaryolarının oluşturulmasını sağlar ve olası oluşabilecek hata durumlarını yazılım mühendisleri ile paylaşarak çıkabilecek hataların erken aşamada önüne geçilmesine yardımcı olur.

Uygulama aşaması, yazılım geliştirme sürecinde tasarlanan projenin artık yazılıma döküldüğü adımdır. Bu aşamadaki ilk hedef çözüm bileşenlerinin kodunu geliştirmek ve dokümantasyonunu oluşturmaktır (Sahu ve diğerleri, 2014). Bu aşamada yazılım mühendisleri, kullanıcı ihtiyaçlarını karşılayacak ve olabildiğince az hata çıkacak şekilde kodlama işlemini sağlarlar. Bu aşamada artık proje gerçek bir çıktı ile takip edilebilir hale gelir. Yazılım devam ettikçe oluşabilecek gereksinimler çıkartılır, engeller belirlenir ve engellerin kaldırılması için aksiyonlar alınır.

Çıktıların daha anlamlı ve izlenebilir olması için her bir modül küçük ekiplere ayrılabilir. Böylelikle yapılan zaman planlamasının doğru şekilde yürütülüp yürütülmediği de kolaylıkla anlaşılabilir olur. Analiz uzmanları geliştirilen yazılım ile kullanıcı gereksinimlerini sürekli olarak karşılaştırarak yazılım sürecinin doğru yolda ilerleyip ilerlemediğini kontrol eder. Test mühendisleri geliştirilen modüllerin işlevselliğini kontrol ederek ilgili yazılım için otomasyon testlerini gerçekleştirebilir.

Test etme, yazılım geliştirme sürecinde kaliteli bir yazılımın ortaya konması için en kritik ve dikkatlice yürütülmesi gereken aşamalardan biridir. Bu aşamada, yazılımın tüm fonksiyonları, tanımlanan kullanıcı gereksinimlerine ve tasarım dokümanlarına göre uygunlukları test edilir (Hossain, 2023). Test sürecinde aynı zamanda güvenlik ve performans kriterlerine göre de kontroller sağlanır. Bunun yanı sıra birim testleri, entegrasyon testleri, sistem testleri ve kabul testleri gibi farklı test seviyeleri uygulanarak yazılımın hem modül bazında hem de bütünsel olarak doğru çalışıp çalışmadığı doğrulanır.

Yazılımda olumsuz bir durum tespit edildiği durumda, hataya ait detaylı bir hata raporu test mühendisleri tarafından oluşturularak yazılım geliştirme ekibine iletilir ve sorunun giderilmesi talep edilir. Oluşturulan hata raporu genellikle hatanın başlığı, açıklaması, tekrarlanması için adımları, beklenen sonuç, gerçekleşen sonuç, sistem ortamı bilgileri, hata ekran görüntüleri veya log kayıtları gibi teknik detayları içerir. Ayrıca, hatanın

öncelik ve kritiklik seviyesi de raporda belirtilerek geliştirme sürecinde hangi sırayla çözüme alınması gerektiği belirlenir.

Yazılım testinin genel amacı olarak 4 temel madde bulunur. Bu maddeler aşağıdaki gibidir (Tuteja ve Dubey, 2012).

- Yazılımın doğru çalıştığını göstermek
- Hataları tespit etmek
- Gelecekteki sorunları önlemek
- Genel olarak kaliteyi artırmak

Tamamlanan yazılımın test edilmesi, yazılımda kaliteyi sağlamak ve yazılımın kullanıcı gereksinimlerine uygun olup olmadığını belirlemek amacıyla birçok süreci içermektedir. Bu süreç, yazılımda daha az hata ile karşılaşılması ve sistemin güvenli biçimde çalışması açısından kritik bir rol oynamaktadır. Ancak yazılım testinin önemi yalnızca basit uygulamaların doğruluğunu sağlamakla sınırlı değildir. Günümüzde yazılımlar; savunma sistemleri, havacılık ve uzay teknolojileri, sağlık sistemleri, finans ve kritik kamu hizmetleri gibi insan hayatını doğrudan etkileyen alanlarda da kullanılmaktadır. Bu tür yazılım sistemlerinde meydana gelebilecek bir yazılım hatası, maddi kayıpların dışında ciddi güvenlik risklerine ve hayati sonuçlara yol açabilmektedir. Bu nedenle test süreçleri yalnızca teknik bir kontrol mekanizması değil, aynı zamanda güvenlik ve risk yönetiminin temel bir bileşeni olarak değerlendirilmelidir.

Bu kapsamda, olası hata ve risklerin etkin biçimde yönetilebilmesi için test süreçlerinin belirli aşamalar doğrultusunda planlanması ve uygulanması gerekmektedir. Test sürecinin temel adımları test planlama, test tasarımı, test uygulama, hata tespiti ve raporlama, test sonuçlarının değerlendirilmesi ve dağıtım ve izleme aşamalarından oluşmaktadır.

Dağıtım aşaması, geliştirilmiş veya güncellenmiş yazılımı kullanıcıların kullanımına hazır hale getirme, kullanıcıya sunma evresidir (Şahinaslan ve arkadaşları, 2023). Yazılım ve test süreci sonrasında kullanıcıların sorunsuz şekilde kullanabilecekleri bir versiyon canlı ortama taşınmış olur. Bu taşınma bazen tüm kullanıcılara aynı anda olabilirken bazen de kademeli şekilde gözlem yapa yapa gerçekleştirilebilir. Kademeli

şekilde açılan versiyonda olası bir sorunda tüm kullanıcılara etki etmeden çözüm sağlanıp tekrar dağıtım işlemi gerçekleştirilebilir.

Yazılım dağıtıldıktan sonra, bakım aşaması devreye girer. Bu aşama, yazılımın kullanılabilirliğini sürdürebilmesi için önemlidir. Bakım sürecinde, yazılımda ortaya çıkan hatalar düzeltilir ve performans iyileştirmeleri yapılır. Ayrıca, kullanıcılar tarafından bildirilen sorunlar çözülür ve yazılımın yeni gereksinimlere göre güncellenmesi sağlanır (Hossain, 2023). Bakım aşaması, yazılımın ömrü boyunca devam eder ve yazılımın tüm yaşam döngüsünün en uzun süren aşamasıdır.

SDLC sürecinin her aşamasında kalite güvencesi ve sürekli iyileştirme prensiplerinin uygulanması, yazılımın başarısı için oldukça önemlidir. Özellikle yazılım test süreçleri, yazılımın kullanıcı beklentilerini karşıladığını doğrulamak ve yazılımın güvenilirliğini, işlevselliğini ve kullanılabilirliğini sağlamak için en etkili araçlardan biri olarak kabul edilmektedir. Yazılım testleri, yalnızca hataların tespit edilmesi ve düzeltilmesi için değil, aynı zamanda yazılımın genel kalitesinin artırılmasında da önemli bir rol oynamaktadır.

SDLC'nin uygulanan farklı modelleri, yazılım geliştirme sürecini çeşitli şekillerde organize eder ve proje gereksinimlerine göre esneklik sağlar. Bu modeller, yazılımın özelliklerini, teslimat sürelerini ve riskleri yönetme biçimini belirler. Her bir model, belirli avantajlar ve zorluklar sunarak farklı proje türleri için uygun olabilir. Yaygın olarak kullanılan bazı modeller aşağıdaki gibidir (Dora ve Dubey, 2013).

- Şelale Model
- V Model
- Spiral Model
- İteratif Model

Şelale Modeli, yazılım projelerinde kullanılan faaliyetlerin ardışık safhalar halinde icra edildiği en eski ve en temel modeldir (Gencer ve Kayacan, 2017). Bu modelde geliştirme süreci adım adım ve sıralı şekilde ilerler. Bir aşama tamamlanmadan bir sonraki aşamaya geçilmez. Süreç, gereksinim analizi ile başlar, ardından tasarım, kodlama, test ve bakım aşamaları ile tamamlanır. Şelale modelinin temel özelliği, başta belirlenen gereksinimlerin süreç boyunca değişmemesine odaklanmasıdır. Bu nedenle

dokümantasyon yoğun olup planlama ve takip oldukça sistemli bir şekilde ilerlemesi gerekmektedir. Şelale modeli, özellikle gereksinimleri net, kapsamı belirli olan ve küçük ölçekli projelerde etkili sonuçlar vermesi nedeniyle bu tarz projelerde kullanılması önerilmektedir.

V Modeli, Şelale Modeline benzer olarak süreçlerin ardışık bir şekilde yürütüldüğü modeldir (Kute ve Thorat, 2014). Süreç, "V" harfi şeklinde ilerler. Bu modelde geliştirme aşamaları (sol kol) ile bu aşamalara karşılık gelen test aşamaları (sağ kol) birbirine paralel olarak planlanır. Yani her geliştirme fazı için henüz kodlama başlamadan bir test planı oluşturulur. Süreç, gereksinim analiziyle başlayıp sistem tasarımı ve mimari tasarımla devam ederken, bu aşamaların tam karşısında kabul testi, sistem testi ve entegrasyon testi gibi doğrulama adımları yer alır. V-Model'inin en büyük avantajı, hataların sürecin en sonunda değil, daha analiz ve tasarım aşamalarındayken tespit edilmesidir.

Spiral Model, yazılım geliştirme sürecini risk odaklı ve tekrarlı bir yapıda ele alan bir SDLC yaklaşımıdır. Bu modelde proje, her biri planlama, risk analizi, geliştirme ve müşteri değerlendirmesi aşamalarından oluşan döngüler halinde ilerler. Planlama ile başlayarak gereksinimler toplanır, olası riskler belirlenir ve sonraki aşamalarda çözümler sunulur (Agarwal ve arkadaşları, 2023). Her bir döngü sonunda çalışan bir ürün parçası ortaya çıkarılır. Bu şekilde hataların her aşamada erkenden tespit edilmesi amaçlanır. Özellikle büyük ölçekli ve belirsizliğin yüksek olduğu projelerde tercih edilen bu yaklaşım, yazılımın her adımda daha güvenli ve test edilmiş bir şekilde büyümesini sağlar.

İteratif Model, yazılımın tek seferde çıkarılması yerine küçük parçalar halinde geliştirilmesini amaçlar. Bu modelde proje, birden fazla yineleme sürecine bölünür ve her döngüde yazılımın çalışır bir parçası ortaya çıkarılır. İlk aşamada temel gereksinimler belirlenerek basit bir sürüm geliştirilir, ardından kullanıcı geri bildirimleri doğrultusunda sistem her döngüde geliştirilerek genişletilir. Böylece yazılımda eksikler erken aşamalarda fark edilerek düzeltilir. Özellikle karmaşık ve belirsiz projelerde tercih edilir çünkü sistem, süreç boyunca sürekli olarak iyileştirilir ve güncellenir.

1.1. Problem

Yazılım test süreçlerinde tespit edilen hataların doğru şekilde raporlanması kadar, bu hatalara ait kritiklik seviyesinin doğru belirlenmesi de yazılım kalitesi açısından büyük öneme sahiptir. Hata kritiklik seviyesi, bir yazılım hatasının sistem işleyişi, kullanıcı deneyimi, veri bütünlüğü ve iş süreçleri üzerindeki etkisini ifade eder. Bu seviye, hataların hangi sırayla çözüleceğini, geliştirme ekibinin kaynaklarını nasıl kullanacağını ve yazılımın canlı ortama ne ölçüde güvenli şekilde alınabileceğini doğrudan belirlemektedir. Bu nedenle kritiklik seviyesinin hatalı ya da tutarsız belirlenmesi, kritik hataların geç fark edilmesine, düşük öncelikli hatalara gereğinden fazla kaynak ayrılmasına ve yazılım kalitesinin olumsuz etkilenmesine neden olabilmektedir.

Yazılım test süreçlerinde hata kritiklik seviyeleri çoğu zaman test mühendisleri tarafından manuel olarak belirlenmektedir. Bu durum, değerlendirme sürecinin kişisel yorumlara, deneyim düzeyine, modül bilgisine, iş yüküne ve zaman baskısına bağlı olarak değişmesine yol açabilmektedir (Gomes ve arkadaşları, 2019). Aynı hatanın farklı test mühendisleri tarafından farklı kritiklik seviyelerinde değerlendirilmesi, hata yönetim sürecinde tutarsızlık oluşturmakta ve çözüm önceliklerinin doğru belirlenmesini zorlaştırmaktadır. Özellikle hızlı teslimat beklentisinin yüksek olduğu yazılım projelerinde, hataların yanlış önceliklendirilmesi geliştirme sürecinde zaman kaybına, kaynakların verimsiz kullanılmasına ve kritik sorunların geç çözülmesine neden olabilmektedir.

Test uzmanının yazılımda bir hata ile karşılaştığında genellikle seçebileceği kritiklik seviyeleri Blocker, Critical, Major ve Minor olarak ele alınmaktadır. Blocker seviyesindeki hatalar, ilgili modülün veya sistemin tamamen çalışamaz hale gelmesine neden olan ve test sürecinin devam etmesini engelleyen problemlerdir. Critical seviyesindeki hatalar, sistemin tamamen durmasına neden olmaz fakat, temel işlevlerin ciddi biçimde bozulmasına yol açan ve kullanıcı deneyimini, iş süreçlerini veya veri bütünlüğünü doğrudan etkiler. Major seviyesindeki hatalar, sistemin genel işleyişini tamamen durdurmasa da belirli fonksiyonların beklenen şekilde çalışmamasına neden olan problemlerdir. Minor seviyesindeki hatalar ise sistemin temel işlevselliğini doğrudan

etkilemeyen ancak kullanıcı deneyimini ve kalite algısını olumsuz etkileyebilecek görsel veya metinsel problemlerdir.

Bu kritiklik seviyelerinin manuel olarak belirlenmesi, hata kayıtlarının değerlendirilmesinde standartlaşma ihtiyacını ortaya çıkarmaktadır. Bir hata kaydının hangi kritiklik seviyesine ait olduğunun yanlış belirlenmesi, yalnızca ilgili hatanın çözüm sırasını değil, aynı zamanda yazılımın yayın planını, test kapanış kararlarını ve bakım süreçlerini de etkileyebilir. Örneğin, gerçekte yüksek kritiklik düzeyine sahip bir hatanın daha düşük seviyede değerlendirilmesi, yazılımın riskli şekilde canlı ortama alınmasına neden olabilir. Benzer şekilde düşük etkili bir hatanın yüksek kritiklikte değerlendirilmesi, geliştirme ekibinin zamanını ve kaynaklarını daha öncelikli sorunlardan uzaklaştırabilir.

Bu koşullar nedeniyle, geliştiricilerin doğru ciddiyet seviyelerine karar vermelerine yardımcı olmak için otomatik ciddiyet tahmin mekanizmalarına ihtiyaç duyulmaktadır (Dao ve arkadaşları, 2022). Günümüzde makine öğrenmesi yöntemleri, veri kümeleri içerisindeki örüntüleri öğrenerek sınıflandırma ve tahmin problemlerinde etkili çözümler sunabilmektedir. Yazılım test süreçlerinde oluşturulan hata raporları da başlık, açıklama ve hata türü gibi metinsel bilgiler içerdiğinden, bu veriler makine öğrenmesi modelleri tarafından analiz edilebilir niteliktedir. Bu nedenle hata kayıtlarının kritiklik seviyelerinin otomatik olarak tahmin edilmesi, manuel değerlendirmeden kaynaklanan tutarsızlıkların azaltılması açısından uygulanabilir bir çözüm yaklaşımı olarak değerlendirilmektedir.

Bu tez çalışmasının temel problemi, yazılım test süreçlerinde oluşturulan hata kayıtlarına ait kritiklik seviyelerinin manuel olarak belirlenmesinden kaynaklanan subjektiflik, tutarsızlık ve yanlış önceliklendirme riskidir. Bu problem doğrultusunda, hata kayıtlarında yer alan metinsel bilgilerden yararlanarak makine öğrenmesi yöntemleriyle hata kritiklik seviyelerinin otomatik, tutarlı ve daha objektif biçimde tahmin edilmesi hedeflenmektedir. Böylece yazılım test ve hata yönetim süreçlerinde insan kaynaklı değerlendirme farklılıklarının azaltılması, kritik hataların daha doğru önceliklendirilmesi ve yazılım kalitesinin artırılması amaçlanmaktadır.

1.2. Amaç

Bu tez çalışmasının amacı, yazılım test süreçlerinde hata tespiti ve raporlama aşamasında belirlenen hata kritiklik seviyelerinin makine öğrenmesi yöntemleri kullanılarak otomatik, doğru ve tutarlı biçimde tahmin edilmesini sağlamaktır. Yazılım hata kayıtlarında kritiklik seviyesinin manuel olarak belirlenmesi, değerlendirmeyi yapan kişiye bağlı olarak değişebilmekte ve insan kaynaklı hatalara ya da tutarsız önceliklendirmelere yol açabilmektedir. Bu nedenle çalışmada, hata kayıtlarının metinsel içeriklerinden yararlanılarak kritiklik seviyelerinin otomatik olarak sınıflandırılması amaçlanmaktadır.

Bu amaç doğrultusunda, yazılım hata kayıtlarının makine öğrenmesi yöntemleriyle kritiklik seviyelerine göre sınıflandırılıp sınıflandırılamayacağı araştırılacaktır. Ayrıca farklı makine öğrenmesi algoritmalarının hata kritiklik seviyesi tahminindeki performanslarının karşılaştırılması ve en başarılı yöntemin belirlenmesi hedeflenmektedir. Bunun yanında, veri setinde yer alan sınıf dengesizliğinin model performansı üzerindeki etkisinin incelenmesi ve veri artırma yaklaşımının özellikle az örnekli kritik hata sınıflarının tahmin başarısına katkısının değerlendirilmesi amaçlanmaktadır.

Çalışmada aşağıdaki araştırma sorularına yanıt aranacaktır:

1. Yazılım test sürecinde oluşan hata kayıtlarının kritiklik seviyeleri makine öğrenmesi yöntemleriyle otomatik olarak tahmin edilebilir mi?
2. Hata kritiklik seviyesi tahmininde kullanılan farklı makine öğrenmesi algoritmaları arasında en başarılı model hangisidir?
3. Sınıf dengesizliği, özellikle azınlıkta kalan kritik hata sınıflarının tahmin başarısını nasıl etkilemektedir?
4. Veri artırma yöntemi uygulandığında modellerin genel performansında ve sınıf bazlı başarılarında iyileşme sağlanabilir mi?

Bu araştırma soruları doğrultusunda çalışmada, hata kritiklik seviyelerinin otomatik tahminine yönelik uygun yöntemin belirlenmesi ve elde edilen bulguların yazılım test süreçlerine sağlayabileceği katkıların ortaya konulması hedeflenmektedir.

1.3. Önem

Yazılım test süreçlerinde hata raporlama ve kritiklik seviyesi belirleme, yazılımın kalitesini doğrudan etkileyen önemli aşamalardır. Hata tespiti ve kritiklik seviyesi belirleme, yazılım geliştirme sürecindeki çözüm sırasını, kaynak atamasını ve zaman yönetimini doğrudan etkilemektedir. Ancak, bu süreçte insandan kaynaklanan hatalar, yanlış sıralama ve önceliklendirme gibi sorunlara yol açabilmektedir. Hataların yanlış derecelendirilmesi, kritik sorunların gözden kaçmasına veya düşük öncelikli hataların ön plana çıkmasına neden olabilir. Bu durum, yazılımın kalitesini olumsuz etkileyebilir ve yazılım geliştirme sürecinde verimliliği düşürebilir. Bu nedenle, hata tespiti ve kritiklik seviyelerinin doğru bir şekilde belirlenmesi, yazılım projelerinin başarı oranını artıran temel faktörlerden biridir.

Bu tez çalışması, yazılım test süreçlerinde hata raporlama ve kritiklik seviyesi belirleme aşamalarındaki insan kaynaklı hataları en aza indirmeyi ve süreci daha objektif hale getirmeyi hedeflemektedir. Makine öğrenmesi yöntemlerinin uygulanması, hata raporlama sürecindeki tutarsızlıkları ortadan kaldırarak, yazılım hatalarının doğru bir şekilde kategorize edilmesini sağlayacaktır. Otomatikleştirilen bu sistem, yazılım geliştirme süreçlerinde daha verimli bir çözüm sıralaması oluşturulmasına yardımcı olacak ve yazılım kalitesini artıracaktır. Böylece, yazılım projelerinde daha hızlı ve doğru çözüm üretme kapasitesi artacak, insan hatalarından kaynaklanan riskler minimize edilecektir. Bu çalışma, yazılım geliştirme süreçlerinin daha verimli ve güvenilir hale getirilmesine önemli bir katkı sağlayacaktır.

1.4. Varsayımlar

Bu tez çalışmasında, yazılım test süreçlerinde hata raporlama ve kritiklik seviyesi belirleme aşamalarında makine öğrenmesi tabanlı bir modelin etkin bir şekilde uygulanacağı varsayılmaktadır. Modelin doğru bir şekilde eğitilmesi ve test verilerinin yeterli kaliteye sahip olması durumunda, insan kaynaklı hataların minimize edileceği ve hata raporlama sürecinin daha objektif bir hale geleceği öngörülmektedir. Ayrıca, hataların kritiklik seviyelerinin doğru şekilde sınıflandırılması ve önceliklendirilmesiyle yazılım geliştirme sürecinde daha hızlı ve verimli bir çözüm sıralaması oluşturulacağı varsayılmaktadır.

Bununla birlikte, bu çalışmanın başarılı olabilmesi için kullanılan makine öğrenmesi modelinin yeterli miktarda etiketlenmiş veriye sahip olması gerektiği kabul edilmektedir.

1.5. Sınırlıklar

Bu çalışma, yalnızca e-ticaret mobil platformunda yer alan bir mobil uygulama özelindeki hata raporlama ve kritiklik seviyesi belirleme süreçlerini ele almaktadır. Çalışmanın bulguları, bu spesifik platforma ait yazılım hatalarını ve raporlama süreçlerini içermektedir. E-ticaret uygulamaları, kullanıcı etkileşimi, ödeme sistemleri ve ürün yönetimi gibi özel modüllere sahip olduğundan, burada elde edilen sonuçların diğer yazılım platformları için genellenmesi sınırlıdır. Diğer yazılım alanlarında, özellikle farklı sektörlerdeki uygulamalarda bu modelin geçerliliği ve doğruluğu konusunda ek çalışmalar gerekmektedir.

Çalışma sürecinde, farklı yazılım projelerinden yeterli miktarda etiketlenmiş veriye erişim sağlanamaması nedeniyle, makine öğrenmesi modelinin daha geniş bir veri kümesiyle test edilmesi mümkün olmamıştır. Bu durum, elde edilen sonuçların yalnızca e-ticaret mobil uygulamalarıyla sınırlı kalmasına yol açmış, diğer sektörlerdeki uygulamalarda modelin performansını değerlendirmeyi engellemiştir. Bu sebeple, bu tezde kullanılan yöntemlerin başka alanlardaki yazılım test süreçlerinde uygulanabilirliği konusunda daha fazla araştırma yapılması gerektiği sonucuna varılmaktadır.

1.6. Tanımlar

Yapay Zeka: Bilgisayar sisteminin akıl yürütme, problem çözme, anlam çıkarma ve genelleme gibi insansı davranışları kullanarak belirli görevleri yerine getirebilmesidir (Arslan, 2010).

Makine Öğrenmesi: Bir makinenin, kendi kendine verilen örnek ve önceki çıkarımlarından yola çıkarak öğrenebileceği öğrenme biçimidir (Aylak ve arkadaşları, 2021).

Test Mühendisliği: Yazılımın kullanılabilirliği, güvenilirlik ve performans gibi kalite özelliklerini değerlendirmeye yönelik testlerin planlanması, tasarımı, yürütülmesi ve raporlanması sürecidir.

Bulgu (Defect/Bug): Yazılımın beklenmeyen bir sonuç üretmesine ve istenmeyen şekillerde davranmasına neden olan bir hata, kusur ya da arızadır.

Hata Raporu: Yazılım test sürecinde tespit edilen bir hataya ait başlık, açıklama, yeniden tekrarlama adımları, beklenen sonuç, gerçekleşen sonuç ve kritiklik seviyesi gibi bilgileri içeren kayıttır.

Sınıf Dengesizliği: Veri setinde bazı sınıflara ait örnek sayısının diğer sınıflara göre belirgin şekilde az veya fazla olması durumudur. Bu durum, modelin çoğunluk sınıflarına daha yatkın tahminler üretmesine neden olabilir.

Eğitim Veri Seti: Makine öğrenmesi modelinin öğrenme sürecinde kullandığı, örnek girişler ve bunlara ait doğru sınıf etiketlerinden oluşan veri kümesidir.

Test Veri Seti: Eğitilen makine öğrenmesi modelinin daha önce görmediği veriler üzerindeki başarısını ölçmek amacıyla kullanılan veri kümesidir.

Veri Ön İşleme: Ham verilerin makine öğrenmesi modellerinde kullanılabilir hale getirilmesi için temizlenmesi, düzenlenmesi ve gereksiz bilgilerden ayrıştırılması sürecidir.

2. LİTERATÜR ARAŞTIRMASI

Yazılım test süreçleri ve bu süreçlerde tespit edilen hataların yönetimi, yazılım kalitesi açısından büyük önem taşımaktadır. Özellikle hata kritiklik seviyesinin doğru şekilde belirlenmesi, hataların çözüm önceliğini etkileyen temel bir faktör olarak karşımıza çıkmaktadır. Son yıllarda, bu sürecin daha etkili ve otomatik hale getirilmesi için yapay zeka ve makine öğrenmesi tekniklerinden yararlanıldığı görülmektedir. Yapay zekanın yazılım testinde kullanımı üzerine yapılan çalışmalar, hata tespiti ve hata sınıflandırması gibi önemli sonuçlar ortaya koymuştur. Bu doğrultuda literatürdeki akademik ve uygulamalı çalışmalar incelenmiş, konuyla ilgili güncel yöntemler analiz edilmiştir.

2.1. Yazılım Test Süreci Üzerine Yapılan Çalışmalar

Yazılım test süreci üzerine yapılan çalışmalar incelendiğinde, test faaliyetlerinin yalnızca hata tespiti ile sınırlı olmadığı, aynı zamanda yazılım kalitesinin artırılması, sistem güvenliğinin sağlanması ve müşteri memnuniyetinin yükseltilmesi açısından da kritik bir rol oynadığı görülmektedir. Uzun ve Koruyan (2019), test sürecinde hata raporlamasının önemine dikkat çekerek, test boyunca elde edilmiş verilerin düzenli bir şekilde takip edilmesinin oluşabilecek risklere karşı önceden tedbir alınabileceğini ifade etmiştir. Başka bir çalışmada Kuday (2014), yazılım test sürecinde karşılaşılan hataların sistemli bir şekilde takip edilmesinin yazılım kalitesini doğrudan etkilediğini belirtmiş ve hata bildirimi, analiz, düzeltme, test etme ve kapatma aşamalarından oluşan bir hata yaşam döngüsünün ne kadar önemli bir gereklilik olduğunu vurgulamıştır. Ayrıca hata raporlarında yer alması gereken tüm bilgilerin açık ve anlaşılabilir bir şekilde belirtilmesi ekipler arası iletişimi güçlendirdiğini ifade etmiştir.

Öztürk (2022) tarafından gerçekleştirilen çalışmada ise özellikle bankacılık alanında yazılım test süreçlerinin kritik olduğu ele alınmıştır. Test planlama, analiz, koşum ve raporlama adımlarının dikkatlice yürütülmesi gerektiği belirtilmiştir. Bu yaklaşımı destekleyen Taley ve Pathak (2020), test süreçlerinin SDLC ile bütünsel olarak yürütülmesi gerektiğini ve test faaliyetlerinin gereksinim analizinden bakım sürecine kadar durmadan devam etmesinin önemini vurgulamıştır. Jamil ve arkadaşları (2016) ise yazılım testinin SDLC'nin en kritik aşamalarından biri olduğunu belirterek, test

faaliyetlerinin çoğu projede önemsenmeyerek geçiştirilmesinin maliyetleri artırdığını belirtmiş ve test mühendislerinin erken aşamalarda sürece aktif bir şekilde dahil edilmesinin gerektiğini belirtmiştir.

Literatür incelendiğinde test süreçlerinin sadece nasıl yürütüleceğine değil, aynı zamanda neden yürütüldüğüne yönelik yaklaşımlar da bulunmaktadır. Uddin ve Anand (2019), yazılım testinin amacının doğru anlaşılmasının, uygun test yöntemlerinin belirlenmesini kolaylaştıracağını ifade ederek "nasıl test edilir" sorusundan çok "neden test edilir" sorusuna odaklanmanın daha önemli olduğunu vurgulamıştır.

Hata yönetimi ve hata yaşam döngüsüne odaklanan çalışmalarda ise, hataların etkin bir biçimde incelenmesinin yazılım kalitesine doğrudan katkı sağladığı görülmektedir. Rana ve arkadaşları (2017), hata yaşam döngüsünü açılma, atanma, düzeltme, yeniden test etme ve kapatma aşamalarıyla ele alırken, farklı hata türlerini detaylandırmıştır. Çalışmada, kabul görmeyen hataların büyük bir kısmının eksik veya hatalı raporlamadan kaynaklandığı ortaya çıkarılmış ve standart hata rapor şablonlarının kullanılmasının bu problemi azaltacağını önermiştir.

Son yıllarda yapay zeka ve makine öğrenmesi tekniklerinin yazılım test süreçlerine entegrasyonu üzerine yapılan çalışmalar da bulunmaktadır. Islam ve arkadaşları (2023), bu alanda gerçekleştirdikleri literatür incelemesinde, manuel test süreçlerinin zaman alıcı ve maliyetli olduğunu ve insan hatasına açık olduğunu belirtmişlerdir. Buna karşılık makine öğrenmesi destekli çözümlerin test senaryosu üretimi, hata tahmini ve test edilecek modüllerin önceliklendirmesi gibi birçok süreci otomatikleştirebildiği ifade edilmiştir. Ancak çalışmada incelenen araştırma sayısının sınırlı olması bir kısıt olarak belirtilmiş ve daha kapsamlı çalışmaların gerekliliği vurgulanmıştır.

Yazılım test süreçlerine yönelik yeni yaklaşımlar da literatürde yer almaktadır. Özdemir ve Öztürk (2025), kuantum yazılım testine ait çalışmalarında klasik test yöntemlerinin kuantum sistemlerin karmaşık yapısını karşılamakta yetersiz kaldığını belirtmiş ve bu alanda yeni test yöntemlerine ihtiyaç duyulduğunu ifade etmiştir. Çalışmada farklı kuantum test araçları ve yöntemleri karşılaştırmış, güçlü ve zayıf yönleri analiz edilmiştir.

Genel olarak değerlendirildiğinde, literatürde yazılım test süreçlerinin sürekli takiplenerek izlenmesinin yazılım kalitesini artırdığı görülmektedir. Özellikle yapay zeka yardımlı test yaklaşımlarının gelecekte daha yaygın hale geleceği ve test süreçlerinin daha verimli yürütülmesine katkı sağlayacağı öngörülmektedir.

2.2. Yazılım Test Sürecinde Makine Öğrenmesi Üzerine Yapılan Çalışmalar

Yazılım test süreçlerinde makine öğrenmesi tekniklerinin kullanımı son yıllarda önemli bir araştırma alanı oluşturmuştur. Bu yönde yapılan çalışmalar incelendiğinde, hata kritiklik seviyesinin otomatik olarak tahmin edilmesine yönelik çeşitli çözümlerin geliştirildiği görülmektedir. Gomes ve arkadaşları (2019), açık kaynak kodlu yazılım projelerinde bu alandaki çalışmaları inceleyerek, kullanılan yöntemleri ve araştırma gerekliliklerini ortaya koymuştur. Çalışmada, çoğu yaklaşımın yapılandırılmamış metin verilerine dayandığı ve geleneksel makine öğrenmesi algoritmalarının yaygın olarak kullanıldığı belirtilmiştir. Ayrıca veri dengesizliği ve yüksek boyutluluk gibi problemlerin model performansını olumsuz yönde etkilediği vurgulanmıştır.

Brown ve arkadaşları (2022), blocker yazılım hatası tahmini alanında makine öğrenmesi tekniklerinin kullanımını inceleyen bir literatür taraması gerçekleştirmişlerdir. Çalışmada 2012–2022 yılları arasında yayımlanan çalışmalar Scopus, SpringerLink, IEEE Xplore, ACM Digital Library ve ScienceDirect veri tabanları üzerinden altı çalışma değerlendirilmiştir. İncelenen çalışmalarda Chromium, Eclipse, FreeDesktop, Mozilla, NetBeans, OpenOffice, Gentoo ve Fedora gibi açık kaynak kodlu projelerden elde edilen hata kayıtları kullanılmıştır. Blocker hata tahmini için Decision Tree, Random Forest, Bayes Network, XGBoost ve Deep Neural Network modellerinin kullanıldığı görülmüştür. Çalışmada ayrıca veri setlerinin önemli ölçüde dengesiz olduğu, bu nedenle Accuracy metriğinin tek başına yeterli olmadığı ve F1-score, precision ve recall gibi metriklerin kullanılmasının daha uygun olduğu belirtilmiştir.

Alharbi ve arkadaşları (2026), tarafından yapılan bir diğer çalışmada ise, yazılım bakım süreçlerinde hata raporlarının kritiklik seviyesinin otomatik olarak tahmin edilmesi konusu ele alınmıştır. Çalışmanın temel amacı, manuel hata kritiklik belirlenmesinden kaynaklanan tutarsızlıkları azaltmak ve yazılım ekiplerinin hata çözüm önceliklerinin daha verimli belirlemesine katkı sağlamaktır. 2019–2025 yılları arasında yayımlanan 27

çalışma incelenmiş, klasik makine öğrenmesi, derin öğrenme, transformer tabanlı modeller, büyük dil modelleri, grafik sinir ağları ve kaynak kod tabanlı yaklaşımlar karşılaştırılmıştır. Çalışmada, derin öğrenme ve özellikle transformer tabanlı yöntemlerin hata kritiklik tahmininde daha başarılı sonuçlar verdiği belirtilmiştir.

Bu alandaki çalışmalardan bir diğeri olan Lamkanfi ve arkadaşları (2010), hata raporlarının metinsel açıklamalarını kullanarak kritiklik seviyesini tahmin etmeyi amaçlayan bir model önermiştir. Mozilla, Eclipse ve GNOME projelerinden elde edilen veriler üzerinde yapılan incelemelerde, özellikle kısa hata özetlerinin model performansı açısından daha etkili olduğu belirlenmiştir. Benzer şekilde Tian ve arkadaşları (2012), bilgi geri getirim tabanlı k-en yakın komşu yöntemi ile hata kritiklik seviyesini tahmin etmeye çalışmış ve önerilen yaklaşımın özellikle zor sınıflandırılan durumlarda önceki yöntemlere göre daha başarılı sonuçlar verdiğini göstermiştir.

Literatürde önemli bir problem olarak öne çıkan veri dengesizliği konusu da farklı çalışmalarla ele alınmıştır. Roy ve Rossi (2017), bu problemi çözmek amacıyla maliyet duyarlı öğrenme yaklaşımlarını incelemiş ve sınıf ağırlıklandırma yöntemlerinin model performansını ciddi şekilde artırdığını ortaya koymuştur. Bu konuyu destekler nitelikte Mahmood ve arkadaşları (2015), veri setinin dengelenmesinin model başarısını doğrudan etkilediğini gözlemlemiştir. Çalışmada, özellikle azınlık sınıf oranının artırılmasının performans metriklerinde iyileşme sağladığı ifade edilmiştir.

Cheng ve arkadaşları (2020), açık kaynak yazılım projelerinde blocker hata tahmini için XGBoost tabanlı XGBlocker yaklaşımını önermiştir. Çalışmada Eclipse, FreeDesktop, NetBeans ve OpenOffice projelerinden elde edilen toplam 214.873 hata raporu kullanılmıştır. Veri setlerinde blocker hata oranının %7–9 aralığında olması nedeniyle sınıf dengesizliği bulunduğu belirtilmiştir. Yazarlar, XGBoost modelini eğitmiş, Random Forest, ELBlocker, AdaBoost, GBDT, CART ve Logistic Regression yöntemleriyle karşılaştırmıştır. Sonuçlarda XGBlocker yöntemi ortalama 0.767 F1-score değerine ulaşarak, literatürdeki güncel ELBlocker yöntemine göre daha başarılı performans göstermiştir.

Farklı makine öğrenmesi yöntemlerinin kullanıldığı çalışmalar da literatürde bulunmaktadır. Otoom ve arkadaşları (2016), AdaBoost tabanlı bir yaklaşım önererek

hata kritiklik tahmininde genel doğruluk oranını artırmış ve geleneksel modellere göre daha başarılı sonuçlar elde ettiğini açıklamıştır. Benzer şekilde Pushpalatha ve Mrunalini tarafından gerçekleştirilen çalışmada, Bagging yöntemi ile C4.5 algoritması karşılaştırılmış ve Bagging yaklaşımının daha yüksek doğruluk sağladığı belirlenmiştir.

Guo ve arkadaşları (2004), yazılım modüllerindeki hata eğilimini erken aşamada tahmin etmek için RF metodolojisini önermiştir. Beş gerçek NASA projesine ait veri seti üzerinde gerçekleştirilen çalışmada McCabe ve Halstead kaynak kod metrikleri girdi olarak kullanılmıştır. Sonuçlara göre Random Forest, özellikle büyük veri setlerinde daha yüksek doğruluk ve hata tespit oranı sağladığı paylaşılmıştır. Genel doğruluk %75–94 aralığında, hata tespit oranı ise %87'ye kadar çıkmıştır. Çalışma, yazılım kalite tahmininde RF modelinin güçlü ve uygulanabilir bir yöntem olduğunu göstermektedir.

Alenezi ve Banitaan (2013), hata raporlarının öncelik seviyesinin otomatik olarak tahmin edilmesini amaçlayan makine öğrenmesi tabanlı bir yaklaşım önermiştir. Çalışmada Eclipse ve Firefox açık kaynak projelerinden elde edilen hata raporları kullanılmıştır. Hata öncelikleri High, Medium ve Low olmak üzere üç sınıf belirlenmiştir. Araştırmada iki farklı özellik seti karşılaştırılmıştır. İlk özellik setinde hata raporlarının özet alanından çıkarılan metinsel özellikler kullanılmış, ikinci özellik setinde ise component, operating system ve severity gibi metadata bilgileri ele alınmıştır. Sınıflandırma aşamasında NB (Naive Bayes), DT (Karar Ağacı, Decision Tree) ve RF (Rastgele Orman, Random Forest) algoritmaları karşılaştırılmıştır. Sonuçlara göre metadata tabanlı ikinci özellik seti, metinsel özelliklere göre daha başarılı sonuçlar üretmiştir. Ayrıca RF ve DT modellerinin NB modeline göre daha yüksek performans gösterdiği belirtilmiştir.

Hufsa ve Javed (2012), açık kaynak yazılım projelerinde hata kayıtlarının otomatik sınıflandırılmasına yönelik bir sistem önermiştir. Bugzilla'dan alınan yaklaşık 29.000 Eclipse ve Firefox hata kaydı ile çalışma gerçekleştirilmiştir. Ön işleme sürecinde Bag of Words, TF-IDF ve Chi-Square özellik seçimi uygulanmıştır. Sınıflandırma aşamasında Multinomial Naive Bayes algoritması kullanılmıştır. Çalışmada NB tabanlı önerilen sistem, SVM kullanan önceki bir çalışma ile de karşılaştırılmıştır. Eğitim verisinin az olduğu durumlarda NB modelinin daha avantajlı olduğu belirtilmiştir. NB ve Chi-Square

kombinasyonu ile ortalama %83 doğruluk sağlarken, en yüksek oranda ise %86 doğruluk elde edilmiştir.

Prabha ve Shivakumar, 2020, yazılım hata tahmini için makine öğrenmesi kullanımlı bir çalışma önermiştir. Çalışmada, NB, RF, SVM (Destek Vektör Makinesi, Support Vector Machine) ve Yapay Sinir Ağları algoritmaları karşılaştırılmıştır. PCA (Temel Bileşen Analizi, Principal Component Analysis) yöntemi ile veri adeti azaltılarak model performansının arttırıldığı belirtilmiştir. PROMISE veri seti üzerinde yapılan deneyde, SVM modelinin yüksek doğruluk sağladığı ve yapay sinir ağlarının ise daha düşük hata oranına sahip olduğu sonucu çıkarılmıştır. Sonuç olarak, özellik seçimi ve veri boyutunun azaltılma tekniklerinin yazılım hata tahmini performansını önemli biçimde etkilediği belirtilmiştir.

Yazılım hata yönetimi süreçlerindeki otomasyon kullanımının artırılmasına yönelik çalışmalar da literatürde bulunmaktadır. Jalbert ve Weimer (2008), tekrarlayan yazılım hata kayıtlarını otomatik olarak tespit etmeye yönelik bir model önermiştir. Çalışmada, metinsel benzerlik ve kümeleme yöntemleri kullanılarak tekrar eden hataların belirlenebildiği ve bu sayede geliştiricilerin iş yükünün azaltılabileceği önerilmiştir. Benzer çalışma olarak Hooimeijer ve Weimer (2007), hata raporlarının kalitesini tahmin eden bir model geliştirerek, hangi hataların daha hızlı çözülebileceğini tahminlemeyi amaçlamışlardır. Bu çalışmada, hata raporundaki açıklama miktarı, ek dokümanlar gibi faktörlerin önemli olduğu belirtilmiştir.

Yazılım test süreçlerinde makine öğrenmesi kullanımına ilişkin incelenen çalışmalar, ele aldıkları problem alanı, kullandıkları yöntemler ve temel yaklaşımlar bakımından Tablo 1'de özetlenmiştir.

Tablo 1. Makine Öğrenmesi Tabanlı Hata Tahmini Çalışmalarının Özeti

Çalışma Konusu	**Yöntem, Algoritma, Mimari**	**Kaynak**	**Tez Çalışması ile İlgisi**
Açık kaynak yazılım projelerinde hata kritiklik seviyesi	Geleneksel makine öğrenmesi, yapılandırılmamış	Gomes ve arkadaşları (2019)	Hata kritiklik tahmini alanındaki eksiklikleri

tahminine yönelik çalışmalar sistematik olarak incelenmiş, kullanılan yöntemler ve araştırma boşlukları değerlendirilmiştir.	metin verisi, veri dengesizliği analizi		göstermesi nedeniyle bu tez çalışmasının literatürdeki yerini desteklemektedir.
Hata raporlarının metinsel açıklamaları kullanılarak kritiklik seviyesinin tahmin edilmesi amaçlanmıştır.	Metin tabanlı sınıflandırma, hata özeti ve açıklama analizi	Lamkanfi ve arkadaşları (2010)	Hata raporlarının metinsel içeriğinden kritiklik tahmini yapılabileceğini göstermesi nedeniyle tezdeki metin sınıflandırma yaklaşımıyla ilişkilidir.
Hata kritiklik seviyesinin tahmini için bilgi geri getirim tabanlı bir yaklaşım önerilmiştir.	Bilgi geri getirim, k-en yakın komşu yöntemi	Tian ve arkadaşları (2012)	Benzer hata kayıtlarından yararlanarak tahmin yapılması, tezde kullanılan metinsel benzerlik ve sınıflandırma yaklaşımını desteklemektedir.
Veri dengesizliği probleminin hata kritiklik tahmini üzerindeki etkisi ele alınmıştır.	Maliyet duyarlı öğrenme, sınıf ağırlıklandırma	Roy ve Rossi (2017)	Sınıf dengesizliğinin tahmin başarısını etkilediğini göstermesi nedeniyle tezdeki dengesiz veri

			problemine doğrudan katkı sağlamaktadır.
Veri setinin dengelenmesinin model başarısı üzerindeki etkisi değerlendirilmiştir.	Veri dengeleme, azınlık sınıf oranını artırma	Mahmood ve arkadaşları (2015)	Veri dengelemenin model performansına etkisini incelemesi, tezde EDA sonrası sonuçların değerlendirilmesiyle ilişkilidir.
Hata kritiklik tahmininde geleneksel modellere göre daha başarılı sonuçlar elde etmek amacıyla AdaBoost tabanlı bir yaklaşım önerilmiştir.	AdaBoost	Otoom ve arkadaşları (2016)	AdaBoost ile hata kritiklik tahmini yapılması, tezde farklı makine öğrenmesi algoritmalarının karşılaştırılmasını desteklemektedir.
Hata tahmininde farklı sınıflandırma yöntemleri karşılaştırılmıştır.	Bagging, C4.5	Pushpalatha ve Mrunalini	Farklı sınıflandırma yöntemlerinin hata tahmininde karşılaştırılması, tezdeki model karşılaştırma yaklaşımıyla benzerlik göstermektedir
Yazılım hata tahmini için farklı makine öğrenmesi algoritmaları	NB, RF, SVM, Yapay Sinir Ağları, PCA	Prabha ve Shivakumar (2020)	Farklı algoritmalar ve boyut azaltma tekniklerinin

karşılaştırılmış ve veri boyutu azaltma uygulanmıştır.			kullanılması, tezdeki çoklu model değerlendirme yaklaşımını desteklemektedir.
Hata takip sistemlerinde tekrarlayan hata kayıtlarının otomatik olarak tespit edilmesi amaçlanmıştır.	Metinsel benzerlik, kümeleme, duplicate hata tespiti	Jalbert ve Weimer (2008)	Tekrarlayan hata kayıtlarının tespitine odaklanması, tezde benzer kayıtların veri sızıntısını önlemek için gruplandırılmasıyla ilişkilidir.
Hata raporlarının kalitesi tahmin edilerek hangi raporların daha hızlı çözülebileceği incelenmiştir.	Hata raporu kalite tahmini üzerine model önerilmiştir.	Hooimeijer ve Weimer (2007)	Hata raporu kalitesinin çözüm sürecine etkisini incelemesi, tezde veri kalitesinin sınıflandırma başarısındaki önemini desteklemektedir.
Hata raporlarının kritiklik seviyesinin otomatik tahmini üzerine 2019–2025 yılları arasındaki çalışmalar incelenmiştir.	Klasik makine öğrenmesi, derin öğrenme, transformer tabanlı modeller, büyük dil modelleri, grafik sinir ağları, kaynak kod tabanlı yaklaşımlar	Alharbi ve arkadaşları (2026)	Güncel yöntemleri birlikte değerlendirmesi, tez çalışmasının klasik ve modern yaklaşımlar içindeki konumunu göstermektedir.

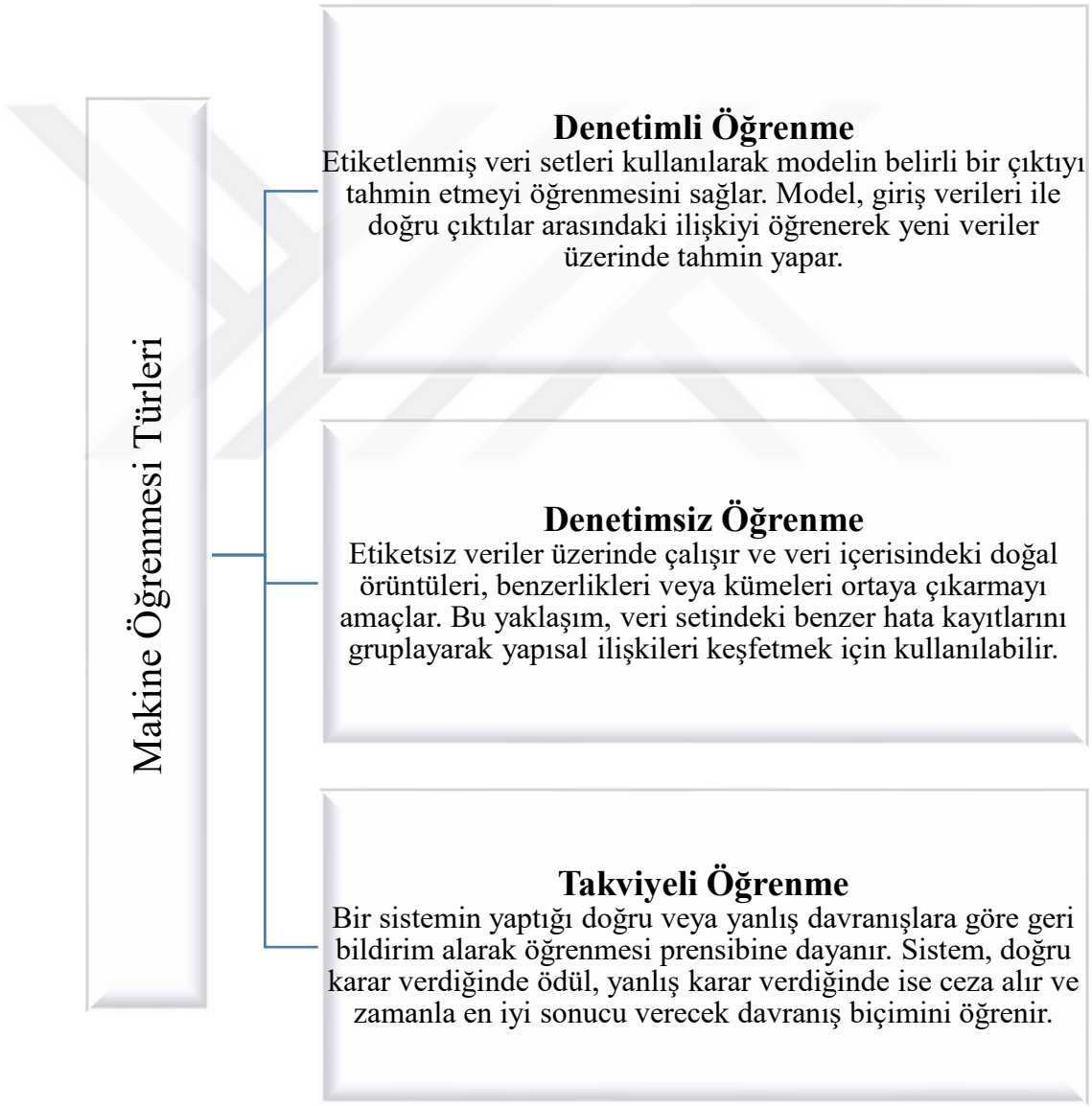
Yazılım modüllerinin hataya yatkın olup olmadığının tahmin edilmesi amaçlanmıştır. NASA veri setleri üzerinde hata içeren modüllerin erken tespit edilmesine yönelik sınıflandırma yapılmıştır.	Random Forest, lojistik regresyon, WEKA, See5, ROCKY karşılaştırması	Guo ve arkadaşları (2004)	Yazılım hatalarının erken tahmin edilebileceğini göstermesi, tezdeki karar destek odaklı sınıflandırma amacını desteklemektedir.
Açık kaynak yazılım projelerinde hata kayıtlarının otomatik olarak sınıflandırılması amaçlanmıştır. Eclipse ve Firefox Bugzilla depolarından elde edilen hata özetleri kullanılarak hataların ilgili sınıflara atanması hedeflenmiştir.	Metin madenciliği, Bag of Words, TF-IDF, Chi-Square özellik seçimi, Multinomial Naive Bayes	(Hufsa ve Javed, 2012)	Hata kayıtlarında TF-IDF ve metin madenciliği kullanılması, tezdeki metinsel özellik çıkarımı yaklaşımıyla doğrudan ilişkilidir.
Blocker hata tahmini üzerine yapılan çalışmalar sistematik olarak incelenmiş; kullanılan veri setleri, makine öğrenmesi yöntemleri, değerlendirme metrikleri ve araştırma boşlukları ele alınmıştır.	Sistematik literatür taraması, Decision Tree, Random Forest, Bayes Network, XGBoost, Deep Neural Network, sınıf dengesizliği analizi	Brown ve arkadaşları (2022)	Blocker hata tahmininde veri dengesizliği ve metrik seçimini vurgulaması, tezde Macro F1 ve sınıf bazlı değerlendirme yapılmasını desteklemektedir.

Açık kaynak yazılım projelerinde blocker bug'ların otomatik olarak tahmin edilmesi amaçlanmıştır. Eclipse, FreeDesktop, NetBeans ve OpenOffice hata raporları üzerinden blocker ve non-blocker hata ayrımı yapılmıştır.	XGBlocker, XGBoost, geliştirilmiş özellik çıkarımı, sınıf dengesizliği analizi	Cheng ve arkadaşları (2020)	XGBoost ile blocker hata tahmini yapılması, tezde XGBoost modelinin değerlendirilmesi açısından doğrudan ilişkilidir.
Hata raporlarının öncelik seviyesinin otomatik olarak tahmin edilmesi amaçlanmıştır. Eclipse ve Firefox hata raporları üzerinde High, Medium ve Low sınıfları tahmin edilmiştir.	Naive Bayes, Decision Tree, Random Forest, metinsel özellikler, metadata tabanlı özellikler	Alenezi ve Banitaan (2013)	Hata raporlarının öncelik seviyesinin tahmin edilmesi, tezdeki hata kritiklik seviyesi sınıflandırma problemiyle benzerlik göstermektedir.

Bu çalışmalar genel olarak değerlendirildiğinde, hata kritiklik seviyesi tahmininde farklı makine öğrenmesi ve derin öğrenme yöntemlerinin kullanıldığı, ancak veri dengesizliği, model genellenebilirliği ve azınlık sınıflarındaki düşük tahmin başarısının halen önemli problemler arasında yer aldığı görülmektedir. Bu tez çalışması, literatürdeki çalışmalardan farklı olarak bir e-ticaret mobil uygulamasına ait gerçek hata kayıtları üzerinde yürütülmüş ve hata kritiklik seviyeleri Blocker, Critical, Major ve Minor sınıfları kapsamında ele alınmıştır. Çalışmada yalnızca genel doğruluk başarısına odaklanılmamış, özellikle az örnekli Blocker sınıfının tahmin performansı ayrı olarak değerlendirilmiştir.

3. YÖNTEM

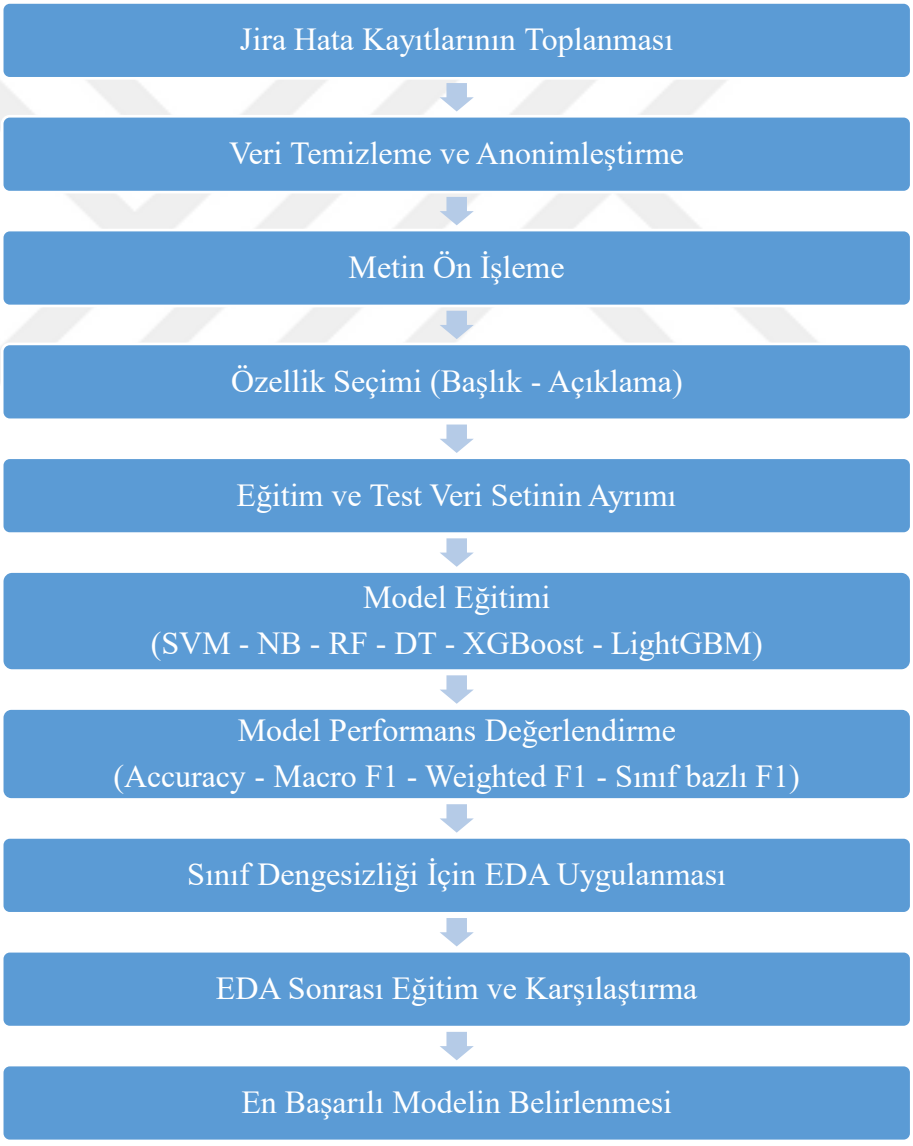
Bu çalışmada, yazılım testinde bulunan hataların kritiklik seviyelerinin tahmin edilmesinde makine öğrenmesi tekniklerinden yararlanılmıştır. Makine öğrenmesi, denetimli öğrenme**,** denetimsiz öğrenme ve takviyeli öğrenme olmak üzere üç ana türe ayrılmaktadır (Arslankaya ve Toprak, 2021). Bu 3 türe ait açıklamalar Şekil 2'de verilmiştir.



Şekil 2. Makine Öğrenmesi Türleri

Bu çalışma kapsamında yazılım hata kayıtları, önceden belirlenmiş kritiklik seviyeleri ile etiketlenmiş olduğundan problem denetimli öğrenme yaklaşımı çerçevesinde ele alınmıştır. Ayrıca, dört farklı kritiklik seviyesinin tahmin edilmesi hedeflendiği için problem çok sınıflı bir sınıflandırma problemi olarak tanımlanmıştır.

Bu çalışma kapsamında yazılım hata kayıtlarının sistematik biçimde analiz edilmesi ve kritiklik seviyelerinin tahmin edilmesi amacıyla bir makine öğrenmesi süreci uygulanmıştır. İzlenen sürecin genel akışı Şekil 2'de gösterilmektedir. Şekilde özetlenen iş akışı, veri setinin hazırlanmasından model performanslarının değerlendirilmesine kadar izlenen temel aşamaları göstermektedir.



Şekil 3. Hata Kritiklik Seviyesi Tahmin Sürecinin Genel Akışı

Şekil 3'te verilen genel iş akışı doğrultusunda, yazılım test süreçlerinde üretilen hata kayıtlarının işlenmesi ve kritiklik seviyelerine göre sınıflandırılması için izlenen temel adımlar aşağıda açıklanmıştır:

1. Veri Toplama: Modelin öğrenebilmesi için gerekli olan ham verilerin toplanmasıdır. Bu aşamada problemle ilgili tüm kayıtlar, dokümanlar veya veriler bir araya getirilir.
2. Veri Temizleme ve Ön İşleme: Toplanan verilerin kullanılabilir hale getirilmesidir. Hatalı, eksik veya gereksiz bilgiler çıkarılır, veriler düzenlenir ve modele uygun formata dönüştürülür.
3. Özellik Seçimi: Veri içindeki hangi bilgilerin model için önemli olduğunun belirlenmesidir. Gereksiz bilgiler çıkarılır, modelin daha doğru öğrenmesini sağlayacak anlamlı özellikler seçilir veya oluşturulur.
4. Model Seçimi: Problemi çözmek için uygun makine öğrenmesi algoritmasının belirlenmesidir. Sınıflandırma ya da tahmin yapmaya uygun model türü seçilir.
5. Model Eğitimi: Seçilen modelin, mevcut veriler kullanılarak öğrenme sürecinden geçirilmesidir.
6. Model Değerlendirme: Bu aşamada modelin ne kadar doğru tahmin yaptığı belirlenir ve performansı çeşitli değerlendirme ölçütleri kullanılarak analiz edilir.
7. Model İyileştirme: Modelin performansını artırmak için parametre ayarları yapılır ve çeşitli iyileştirmeler uygulanır.
8. Model Dağıtımı ve İzleme: Eğitilen modelin gerçek bir sistemde kullanılması ve performansının düzenli olarak takip edilmesidir.

3.1. Veri Toplama

Bu çalışmada kullanılan veriler, bir e-ticaret mobil uygulamasında yürütülen yazılım test süreçleri sırasında oluşturulan hata kayıtlarından elde edilmiştir. Yazılım testleri sonucunda tespit edilen hatalar, kurumsal yazılım geliştirme süreçlerinde kullanılan Jira platformu üzerinde test mühendisleri tarafından raporlanmış ve her bir hata için başlık, açıklama, oluşturulma tarihi, modül bilgisi, hata türü, öncelik ve kritiklik seviyesi gibi ayrıntılı bilgiler sistem tarafından saklanmıştır.

Çalışmanın amacı doğrultusunda, bu sistemde yer alan hata kayıtları veri setine dönüştürülmüş ve analiz sürecinde kullanılmak üzere dışa aktarılmıştır. Dışa aktarma işlemi sırasında yalnızca hata tanımıyla doğrudan ilişkili alanlar dikkate alınmıştır. Böylece oluşturulan veri seti, makine öğrenmesi modellerinin eğitimi için anlamlı ve açıklayıcı nitelikler içerecek biçimde yapılandırılmıştır. Verilerin tamamı analiz sürecinden önce anonimleştirilmiş, kişisel veya kurumsal bilgi içeren alanlar çalışma kapsamı dışında tutulmuştur. Veriler yalnızca teknik içerikli hata raporlarından oluştuğundan, gizlilik ilkeleri korunmuş ve etik araştırma standartlarına uygun hareket edilmiştir.

Dışa aktarılan hata kayıtlarını içeren veri seti toplam 1467 adettir. Bu hata kayıtlarının kritiklik seviyelerine göre dağılımı Tablo 2'de gösterilmektedir.

Tablo 2. Hata Kayıtlarının Kritiklik Seviyesine Göre Dağılımı

Kritiklik Seviyesi	**Adet**	**Oran**
Blocker	52	%3.5
Critical	359	%24.5
Major	647	%44.1
Minor	409	%27.9
Toplam	**1467**	**%100**

Tablo 2'de görüldüğü üzere, veri setinde yer alan hata kayıtları kritiklik seviyelerine göre dengeli bir dağılım göstermemektedir. Özellikle Blocker sınıfına ait kayıt sayısının diğer sınıflara göre oldukça düşük olduğu görülmektedir. Bu durum yalnızca bu çalışmaya özgü değildir. Valdivia Garcia ve Shihab (2014), Eclipse, FreeDesktop, Mozilla, NetBeans ve OpenOffice projelerinden elde edilen toplam 363.343 geçerli hata kaydını incelemiş ve bu projelerin çoğunda blocker bug oranının toplam hata kayıtları içinde %3'ün altında olduğunu belirtmiştir. Bu bulgu, blocker türündeki hataların yazılım hata veri setlerinde azınlık sınıfı oluşturabildiğini ve bu tez çalışmasında karşılaşılan sınıf dengesizliği durumunun literatürdeki benzer çalışmalarla uyumlu olduğunu göstermektedir.

3.2. Veri Temizleme ve Ön İşleme

Literatürde veri temizleme ve ön işleme adımları, ham verinin makine öğrenmesi modelleri tarafından öğrenilebilir ve analiz edilebilir bir yapıya dönüştürülmesi açısından önemli bir aşama olarak değerlendirilmektedir. Özellikle metin tabanlı sınıflandırma çalışmalarında noktalama işaretleri, özel karakterler, ilgisiz sayısal ifadeler, tekrar eden kayıtlar ve konu dışı içeriklerin temizlenmesiyle verinin model eğitimine daha uygun hale getirildiği görülmektedir. Örneğin Veziroğlu ve Bucak (2025), haber sınıflandırma sistemleri üzerine gerçekleştirdikleri çalışmalarında metinlerde yer alan noktalama işaretleri, özel karakterler vb. değerleri temizleyerek ham metin verilerini makine öğrenmesi algoritmaları için uygun bir yapıya dönüştürmüştür. Bu tez çalışmasında da literatürdeki benzer yaklaşımlar ve uygulama sürecinde edinilen deneyimler dikkate alınarak veri temizleme ve ön işleme akışı oluşturulmuştur.

Makine öğrenmesi modellerinin, daha doğru sonuçlar verebilmesi için kullanılan veri setinin temiz ve analiz edilmeye uygun bir duruma dönüştürülmesi gerekmektedir. Bu yapılan çalışmada, Jira platformu üzerinden elde edilen işlenmemiş hata kayıtları hem manuel yöntemlerle hem de Python tabanlı otomatik işlemlerle kapsamlı bir ön işleme süzgecinden geçirilmiştir. Bu işlemlerin amacı, veri setindeki gereksiz kirliliği ortadan kaldırmak, metinleri anlamlı bir yapıya kavuşturmaktır.

Öncelikle henüz işlenmemiş veri seti üzerinde manuel incelemeler yapılarak, tekrarlanan, hatalı veya anlamsız içerik bulunduran kayıtlar tespit edilmiş ve veri setinden silinmiştir. Bu aşamada özellikle hata detay açıklamalarında olan uygulama versiyon bilgileri, cihaz model bilgileri, fazla boşluklar, satır geçişleri ve okunamayan özel karakterler temizlenmiştir. Bununla beraber, veri setinde yer alan şirket adı, proje özel bilgileri, kullanıcı bilgileri, yazılım mühendislerinin isimleri, e-posta adresleri ve benzeri şirkete ait özel bilgiler tespit edilerek veri setinden çıkarılmış veya anonimleştirme işlemi sağlanmıştır. Böylece veri seti, herhangi bir kişi veya şirketi doğrudan hedeflememiş duruma gelmiştir.

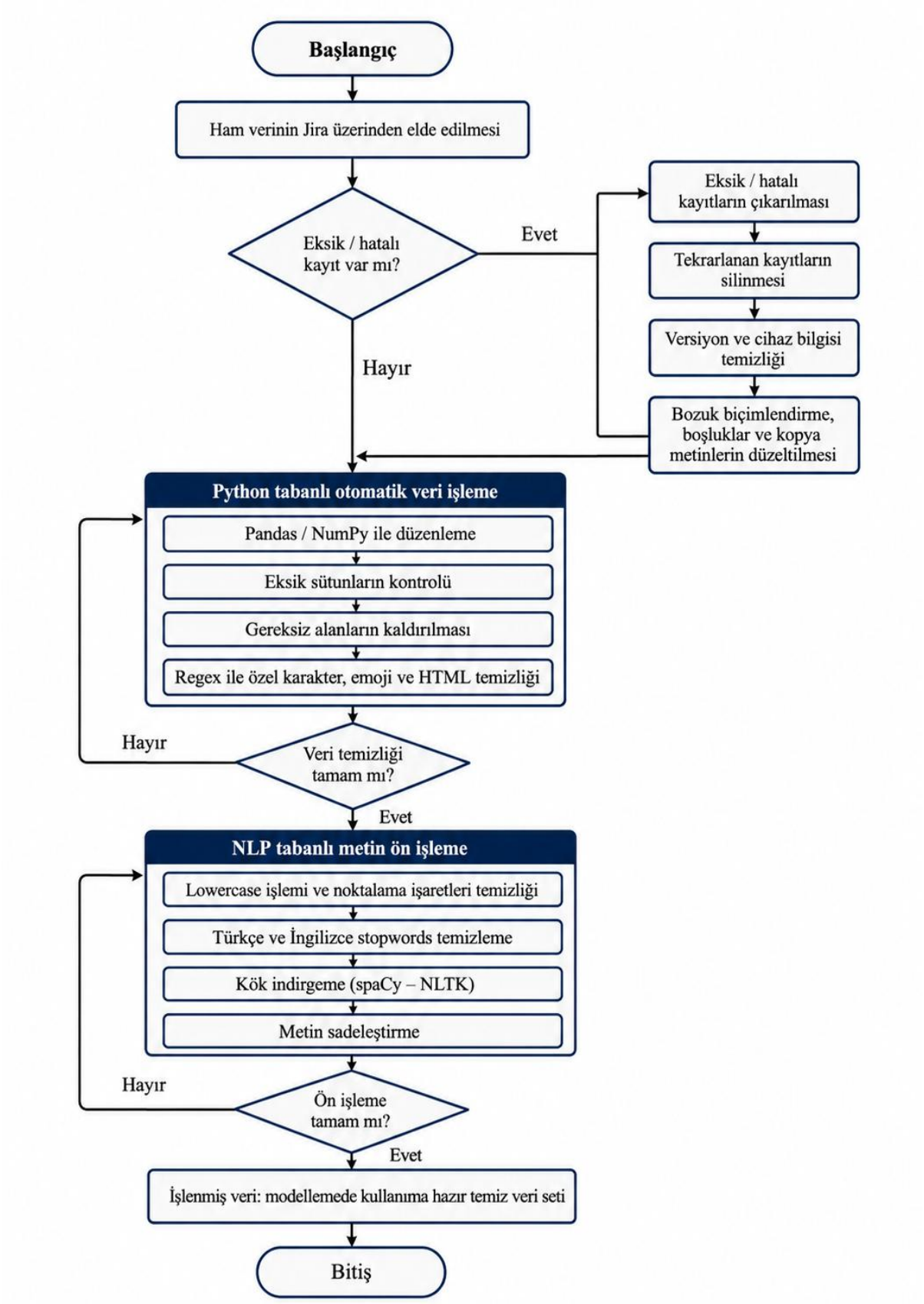
Manuel temizlik işleminden hemen sonra veri ön işleme aşaması Python ortamında sürdürülmüştür. Bu aşamada pandas ve numpy kütüphaneleri veri setinin yüklenmesi, eksik değerlerin kontrol edilmesi, gereksiz sütunların kaldırılması ve genel

düzenlemelerin yapılması için kullanılmıştır. Metinlerde manuel temizleme işleminde temizlenmemiş olabilecek özel karakterlerin, emojilerin, HTML etiketlerinin ve standart dışı sembollerin temizlenmesi için regular expressions kütüphanesinden yararlanılmıştır.

Doğal dil işleme adımlarında ise nltk, spaCy ve scikit-learn kütüphaneleri kullanılmıştır. Bu aşamada metinler küçük harfe dönüştürülmüş, noktalama işaretleri kaldırılmış, Türkçe ve İngilizce durak kelimeler (stop words) temizlenmiştir.

Tüm bu işlemler sonucunda veri seti hem içerik hem yapı açısından sadeleştirilmiş, ve makine öğrenmesi modellerinin eğitiminde doğrudan kullanılabilir bir formata dönüştürülmesi sağlanmıştır.

Bu çalışma kapsamında izlenen veri temizleme ve ön işleme süreci Şekil 4'te akış diyagramı şeklinde gösterilmektedir.



Şekil 4. Veri Temizleme ve Ön İşleme Sürecinin Akış Diyagramı

Tablo 3'te, veri temizleme işlemli sonrasında elde edilen veri setinin sınıf bazlı dağılımı yer almaktadır.

Tablo 3. Veri Temizleme Süreci Sonrası Veri Dağılımı

Kritiklik Seviyesi	**Adet**	**Oran**
Blocker	43	%3.8
Critical	295	%26.1
Major	439	%38.8
Minor	355	%31.3
Toplam	**1132**	**%100**

3.3. Özellik Seçimi

Veri temizleme sürecinin tamamlanmasının ardından, makine öğrenmesi modellerinin başarıyla eğitilebilmesi için hata kayıtlarından anlamlı, ayırt edici ve modele katkı sağlayan özelliklerin çıkarılması gerekmektedir. Bu doğrultuda, veri seti üzerinde özellik seçimi süreçleri yürütülmüştür.

Özellik seçimi sağlanmamış veri setinde kayıt Id, kayıt açılma tarihi, başlık, açıklama, öncelik seviyesi, kritiklik seviyesi, atanan kullanıcı ve kaydı oluşturan kullanıcı gibi çeşitli alanlar yer almaktaydı. Ancak bu alanlardan bir kısmı sınıflandırma problemine doğrudan katkı sağlamadığı için modelleme sürecine dahil edilmemesi gerektiği düşünülmüştür.

Bu çalışma kapsamında amaç, hata kaydının kritiklik seviyesinin yalnızca hata metnine dayalı olarak tahmin edilmesidir. Bu doğrultuda, özellik seçimi aşamasında modelin anlamlı ve ayırt edici bilgiyle beslenmesini sağlamak amacıyla yalnızca üç temel alan kullanılmıştır. Bu özellikler ve modeldeki işlevlerine ait açıklamalar Tablo 4'te gösterilmektedir.

Tablo 4. Özellik Seçimi Aşamasında Belirlenen Nitelikler

Özellik	**Açıklama**
Başlık	Hatanın kısa ve özet tanımını içerdiği için, ilgili problemin türüne ilişkin önemli ipuçları barındırmaktadır.
Açıklama	Hatanın detaylı anlatımını sunduğundan, modelin öğrenme aşamasında en fazla bilgi içeren niteliktir.
Kritiklik Seviyesi	Çalışmanın hedef sınıfı olup, makine öğrenmesi modellerinin tahmin edeceği değişkendir.

Sonuç olarak, bu özellik seçimi süreci ile yalnızca çalışmanın amacına hizmet eden, alanlar kullanılarak sade, anlamlı ve etkili bir veri oluşturulmuştur.

3.4. Model Seçimi

Bu çalışmada hata kayıtlarının kritiklik seviyesini otomatik olarak tahmin edebilmek için farklı öğrenme yaklaşımlarını temsil eden SVM, NB, DT, RF, XGBoost (Aşırı Gradyan Artırma, Extreme Gradient Boosting) ve LightGBM (Hafif Gradyan Artırma Makinesi, Light Gradient Boosting Machine) modelleri tercih edilmiştir. Model seçimi yapılırken yalnızca literatürdeki başarıları değil, veri setinin yapısı da dikkate alınmıştır. Özellikle veri kümesinde belirgin bir sınıf dengesizliği bulunması, farklı algoritmaların aynı veri seti üzerindeki performanslarının karşılaştırmalı olarak değerlendirilmesini gerekli kılmıştır.

Literatürde NB sınıflandırıcının hızlı tahminler yapabilen, basit ve etkili sınıflandırma algoritmalarından biri olduğu ifade edilmektedir (Kayakuş ve Yiğit Açıkgöz, 2023). Bu doğrultuda NB, hızlı ve anlaşılır bir temel sınıflandırma modeli sunması nedeniyle çalışmada referans model olarak kullanılmıştır. SVM ise istatistiksel öğrenme teorisi ve yapısal risk minimizasyonu ilkelerine dayanan, teorik altyapısı güçlü bir örüntü sınıflandırma tekniği olması ve metin tabanlı sınıflandırma problemlerinde ortaya çıkan

yüksek boyutlu özellik uzaylarında etkili sonuçlar verebilmesi nedeniyle değerlendirilmiştir (Harman, 2021).

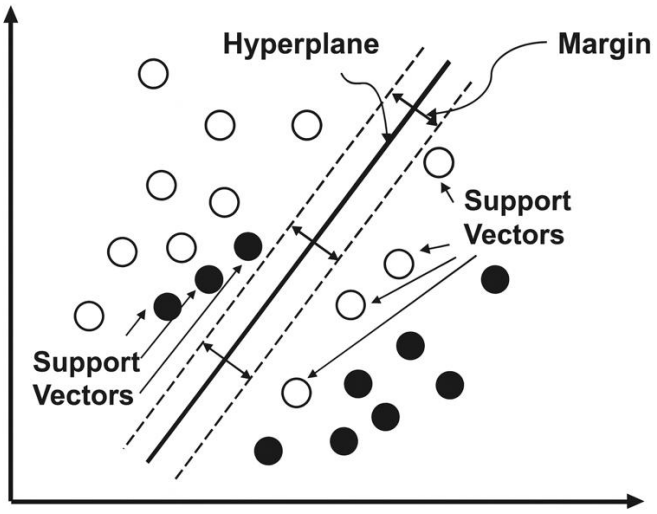
DT, basit ve anlaşılır yapıya sahip, karar kuralları üzerinden sınıflandırma yapan bir algoritmadır. Literatürde DT modelinin küçük veri setlerinde oldukça iyi çalışabildiği, ancak aşırı öğrenmeye yatkın olabileceği ifade edilmektedir (Sevimli Deniz, 2024). Bu nedenle DT, çalışmada yorumlanabilir bir temel sınıflandırma modeli olarak değerlendirilmiştir. RF ise hızlı, esnek ve yüksek boyutlu verileri analiz edebilecek kadar güçlü bir algoritma olarak değerlendirilmektedir (Şirin, 2020). Bu nedenle çalışmada, karar ağacı tabanlı yapısı ve topluluk öğrenmesi avantajı sayesinde daha kararlı sınıflandırma sonuçları üretebilmesi amacıyla tercih edilmiştir.

Literatürde XGBoost'un sınıflandırma problemlerinde hızlı ve yüksek doğruluk oranları sunabildiği, LightGBM'in ise büyük veri setleri ve çok boyutlu özellikler üzerinde hızlı ve etkili sonuçlar üretebildiği belirtilmektedir (Eşidir, 2025). Bu nedenle bu çalışmada, hata kayıtlarının kritiklik seviyesini tahmin etmede güçlü topluluk öğrenmesi yaklaşımlarını değerlendirebilmek amacıyla XGBoost ve LightGBM modelleri tercih edilmiştir.

3.4.1. Destek vektör makinesi

SVM algoritması, istatistiksel öğrenme teorisine dayanan gözetimli makine öğrenme algoritmalarından biridir (Wendong ve arkadaşları, 2017). Temel amacı, farklı sınıflara ait örnekleri en geniş marjine sahip hiper düzlem aracılığıyla birbirinden ayırmaktır. Bu yaklaşım hem doğru sınıflandırmayı maksimize etmeyi hem de modelin yeni veriler üzerindeki genellenebilirlik performansını artırmayı hedefler. SVM'in matematiksel temelleri, sınıflar arasındaki optimal ayrım sınırının belirlenmesini sağlayan yapısal risk minimizasyonuna dayanmaktadır. Ayrıca SVM, doğrusal olarak ayrılabilen veri kümelerinde etkin bir şekilde çalışırken, doğrusal olmayan ilişkilerin bulunduğu durumlarda çekirdek fonksiyonları (kernel) aracılığıyla veri uzayını daha yüksek boyutlara dönüştürür ve karmaşık karar sınırlarını modelleyebilir. Bu özellikleri nedeniyle metin sınıflandırma, veri analizi, görüntü işleme ve hata tespiti gibi yüksek boyutlu ve çok değişkenli veri yapılarının bulunduğu birçok alanda yaygın olarak tercih

edilen bir yöntem haline gelmiştir. Şekil 5'te SVM modeline ait örnek sınıflandırma yapısı gösterilmektedir.



Şekil 5. SVM Modeline Ait Örnek Sınıflandırma Yapısı

Şekilde, doğrusal bir SVM modelinin sınıflandırma yapısı gösterilmektedir. Veri noktaları iki sınıfa ayrılmış olup, sınıfları ayıran hiper düzlem kalın çizgiyle ifade edilmiştir. Hiper düzleme paralel kesikli çizgiler, modelin sınıflar arasındaki en geniş ayrım mesafesini temsil eden marjinleri göstermektedir. Marjin üzerinde yer alan ve karar sınırının konumunu doğrudan belirleyen kritik örnekler ise destek vektörleri olarak işaretlenmiştir. Bu yapı, SVM'in karar fonksiyonunu yalnızca marjin sınırındaki örneklere dayanarak oluşturduğunu ve böylece yüksek genellenebilirlik performansı elde ettiğini göstermektedir.

3.4.2. Naive Bayes

NB sınıflandırıcısı veri madenciliği, makine öğrenmesi, duygu analizi gibi alanında yaygın kullanıma sahip denetimli öğrenme sınıfında bir algoritmadır (Şahinaslan ve arkadaşları, 2022). Bayes Teoremi'ne dayanarak bir örneğin hangi sınıfa ait olduğuna ilişkin olasılık tahmini yapar. NB modelleri, her bir kelimenin belirli bir sınıfta görülme olasılığını hesaplayarak bir dokümanın en olası sınıfını belirlemeyi amaçlar. Bu yaklaşım, hesaplama maliyetinin düşük olması, modelin kolay yorumlanabilir olması ve büyük veri kümelerinde hızlı çalışabilmesi gibi avantajlar sunmaktadır. Ayrıca modelin parametre

sayısı kısmen az olduğundan, eğitim süresi diğer birçok makine öğrenmesi yöntemine kıyasla oldukça kısadır. NB sınıflandırıcısının kullanımı, hava durumu tahmin hizmetleri, müşteri kredi değerlendirmeleri, sağlık durumu sınıflandırmaları gibi birçok başarılı gerçek yaşam uygulaması bulunmaktadır (Yang F.-J. , 2018). NB sınıflandırıcısının temelini oluşturan Bayes Teoremi'nin matematiksel ifadesi Formül 1'de verilmiştir (Aytaç ve arkadaşları, 2018).

$$P(y_j|\mathrm{X}) = \frac{P(X|y_j)P(y_j)}{P(X)} \quad (1)$$

Şekil 6. Bayes Teoremi'nin Matematiksel İfadesi

Bayes Teoremi, bir olayın gerçekleşme olasılığının, başka bir olayın gerçekleşme olasılığına bağlı olarak nasıl değiştiğini belirtir.

P(y | X): X koşulunda y'nin olma olasılığıdır.

P(X | y): y koşulunda, X'in olma olasılığıdır.

P(y): y'nin başlangıçtaki olasılığıdır.

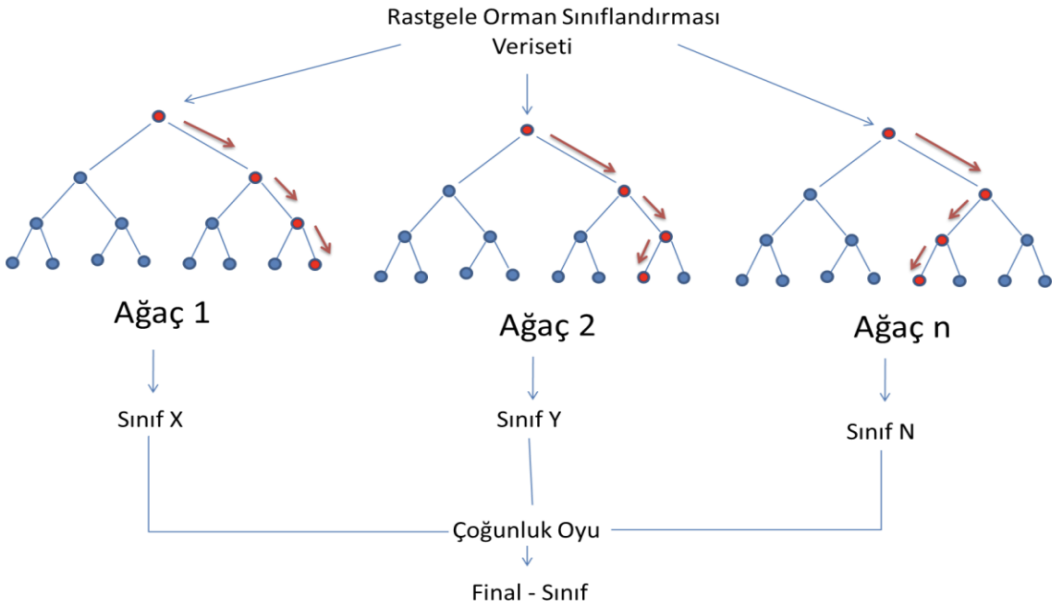
P(X): X bilgisinin genel olasılığıdır.

3.4.3. Rastgele orman

RF, sınıflandırma ve regresyon problemlerinde yaygın olarak kullanılan bir makine öğrenmesi yöntemidir. Eğitim aşamasında çok sayıda karar ağacının oluşturulmasına ve bu ağaçlardan elde edilen çıktıların bir araya getirilmesine dayanan RF, topluluk öğrenme yöntemleri arasında yer almaktadır (Örpek ve arkadaşları, 2026). Algoritmayı popüler yapan en önemli özelliklerinden biri, kapsamlı hiperparametre kestirimi gerektirmeden iyi sonuçlar üretebilmesi ve hem regresyon hem de sınıflandırma problemlerine uygulanabilmesidir (Özbay ve Altunbey Özbay, 2021). Eğitim sürecinde her bir karar ağacı, bootstrap örnekleme yöntemi ile oluşturulan rastgele veri alt kümeleri üzerinde

eğitilir. Ayrıca her düğüm bölünmesinde tüm özellikler yerine rastgele seçilmiş bir alt özellik kümesinin değerlendirilmesi sağlanır.

Bu iki mekanizma, modeller arasında çeşitlilik oluşturarak aşırı öğrenme riskini azaltır ve modelin genellenebilirliğini artırır. RF algoritması, girdi değişkeninin çok olduğu büyük verilerde iyi performans göstermesinin yanı sıra eksik verilerde de yüksek tahminler gerçekleştirmesi bakımından büyük kolaylık sağlamaktadır (Özlüer Başer ve arkadaşları, 2021). Sınıflandırma sürecinde ise her bir karar ağacının ürettiği sınıf tahminleri çoğunluk oylaması ile birleştirilmekte ve nihai sınıf etiketi belirlenmektedir. Bu süreç, çok sayıdaki karar ağacından elde edilen tahminlerin bir araya getirilmesi yoluyla nihai kararın oluşturulmasını ifade etmekte olup, görsel olarak Şekil 7'de sunulmuştur (Khan ve arkadaşları, 2021).

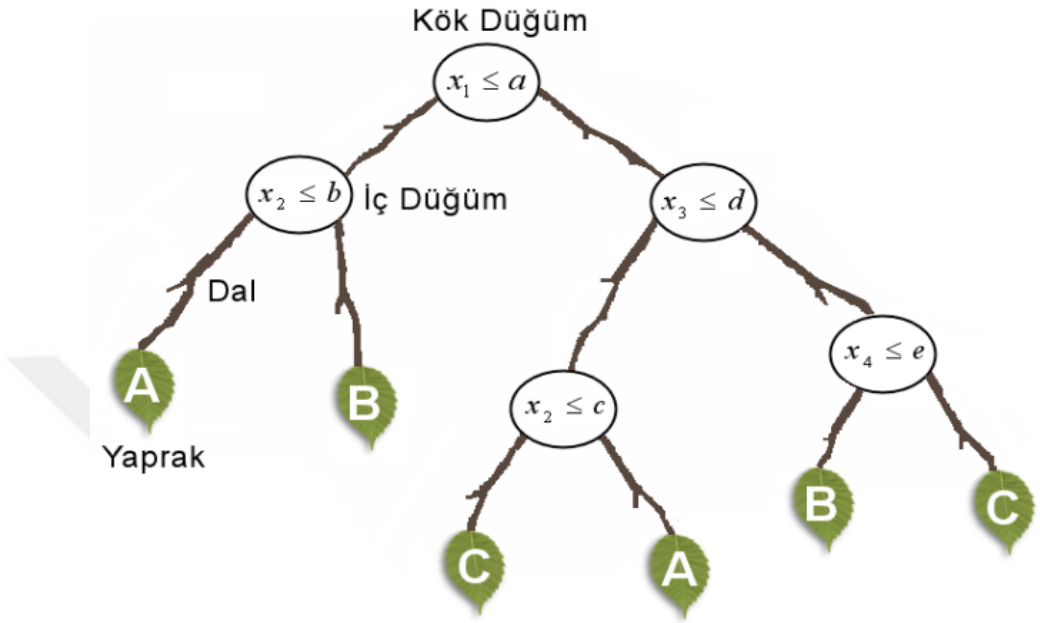


Şekil 7. RF Sınıflandırma Şeması

3.4.4. Karar Ağacı

DT sınıflandırma problemlerinde yaygın olarak kullanılan makine öğrenmesi algoritmalarından biridir. Veri madenciliğinde, sınıflandırma ve regresyon problemlerinin çözümünde yaygın olarak kullanılmaktadır. Bu algoritmanın avantajı, oluşturulmasının ve yorumlanmasının kolay olmasıdır (Özlüer Başer ve arkadaşları, 2021).

Bir DT'nin temel yapısı Şekil 8'de gösterilmektedir (Kavzoğlu ve Çölkesen, 2010). Ağaç yapısı, verileri içeren bir kök düğümü, iç düğümler (dallar) ve uç düğümlerden (yapraklar) oluşur.



Şekil 8. DT Temel Yapısı

Sınıflandırma işlemi kök düğümde başlar ve burada bir özellik için belirlenen eşik değere göre bir karar testi uygulanır. Test sonucu örneğin doğruysa sol dala, yanlışsa sağ dala yönlendirilir. Her iç düğümde benzer şekilde başka bir özellik için yeni bir eşik testi yapılır ve örnek, test sonucuna göre ilgili dala aktarılır. Bu süreç, örnek bir yaprak düğüme ulaşana kadar devam eder. Yaprak düğümler (A, B, C) modelin ürettiği nihai sınıf etiketini temsil eder. Dolayısıyla, kökten yaprağa izlenen her yol, "eğer–ise" biçiminde ifade edilebilen bir karar kuralı dizisini oluşturur.

3.4.5. XGBoost

XGBoost algoritması, gradyan artırma yaklaşımını temel alan, karar ağaçlarıyla çalışan bir makine öğrenmesi algoritmasıdır (Tekin ve Yaman, 2023). Algoritma ilk olarak Tianqi Chen ve Carlos Guestrin tarafından 2011 senesinde önerilmiştir (Chen ve Guestrin, 2016). XGBoost modelinde, karar ağaçları ardışık olarak eğitilmektedir. İlk ağaç bir tahmin üretir ve sonraki ağaçlar önceki modellerin yaptığı tahmin hatalarını azaltmayı hedefleyerek öğrenme sürecine katkı sağlar. Bu yapı sayesinde model

performansını iyileştirir ve en iyi seçilen tahmin, tüm ağaçların çıktılarının toplamı olarak elde edilir. XGBoost'un düzenlileştirme (regularization) özelliği modelin karmaşıklığını kontrol altına alarak aşırı öğrenmenin önüne geçmeyi amaçlamaktadır.

XGBoost'un başarısının arkasındaki temel etkenlerden biri, farklı kullanım senaryolarında sunduğu yüksek ölçeklenebilirlik ve hesaplama verimliliğidir. Algoritma, optimize edilmiş öğrenme ve sistem tasarımı sayesinde tek bir makine üzerinde dahi birçok yaygın makine öğrenmesi çözümüne kıyasla önemli ölçüde daha hızlı çalışabilmekte ve büyük ölçekli veri kümeleri üzerinde etkin biçimde uygulanabilmektedir (Chen ve Guestrin, 2016).

3.4.6. LightGBM

LightGBM, gradyan artırma çatısı altında çalışan karar ağaçları tabanlı yeni nesil topluluk öğrenme algoritmalarındandır (Üstüner ve arkadaşları, 2020). Algoritma, Microsoft Research tarafından geliştirilmiş ve ilk olarak 2017 yılında tanıtılmıştır. LightGBM'in temel amacı, büyük ölçekli veri kümeleri üzerinde yüksek doğruluk, düşük bellek kullanımı ve hızlı eğitim süresi sağlamaktır.

LightGBM, GOSS (Gradyan Tabanlı Tek Taraflı Örnekleme, Gradient-based One-Side Sampling) ve EFB (Özel Özellik Birleştirme, Exclusive Feature Bundling) adı verilen iki yeni teknikten faydalanmaktadır. GOSS yöntemi, gradyan değeri yüksek olan örneklerin tamamını korurken, gradyan değeri düşük olan örneklerden rastgele bir alt küme seçerek hesaplama maliyetini azaltmaktadır. EFB tekniği ise, nadiren aynı anda aktif olan seyrek özellikleri tek bir özellik altında birleştirerek bellek kullanımını düşürmekte ve eğitim sürecini hızlandırmaktadır. Bu teknikler sayesinde LightGBM, yüksek eğitim hızı ve düşük bellek kullanımı ile öne çıkmaktadır (Üstüner ve Balık Şanlı, 2019).

LightGBM, klasik GBDT (Gradyan Artırmalı Karar Ağaçları, Gradient Boosting Decision Tree) yaklaşımlarına kıyasla, özellikle hesaplama verimliliği ve ölçeklenebilirlik açısından önemli iyileştirmeler sunmaktadır. Bu özellikleri sayesinde LightGBM, hem akademik çalışmalarda hem de endüstriyel uygulamalarda yaygın biçimde tercih edilen bir boosting algoritması haline gelmiştir.

3.5. Model Eğitimi

Model eğitimi aşamasında, ön işleme ve özellik çıkarımı adımlarından geçirilmiş yazılım hata kayıtları kullanılarak sınıflandırma modelleri eğitilmiştir. Veri kümesi, modelin genelleme performansını gözlemlemek amacıyla sınıf dağılımı korunacak şekilde %85 eğitim ve %15 test olacak biçimde iki alt kümeye ayrılmıştır. Test veri kümesi, deney süreci boyunca sabit tutulmuş ve herhangi bir veri artırma veya dengeleme işlemine tabi tutulmamıştır. Eğitim ve test veri kümeleri sırasıyla 962 ve 170 kayıttan oluşmaktadır.

Eğitim ve test veri setlerinin ayrılma yapısı Tablo 5'te, sınıf bazlı dağılımları ise Tablo 6'da gösterilmektedir.

Tablo 5. Eğitim ve Test Veri Seti Ayrımının Şematik Gösterimi

Veri Kümesi	**Adet**	**Oran**
Eğitim	962	%85
Test	170	%15

Tablo 6. Eğitim ve Test Veri Setinin Sınıf Bazlı Dağılımı

Kritiklik Seviyesi	**Eğitim Veri Seti Adet**	**Test Veri Seti Adet**
Blocker	36	7
Critical	251	44
Major	373	66
Minor	302	53

Eğitim ve test veri setlerinin sınıf bazlı dağılımları incelendiğinde, Blocker sınıfının hem eğitim hem de test veri kümesinde oldukça düşük bir orana sahip olduğu görülmektedir. Bu durum, modelin özellikle yüksek öneme sahip Blocker seviyesindeki hataları öğrenmesini zorlaştıran temel problemlerden biri olarak değerlendirilmiştir.

Model eğitimi yapılan bilgisayar, Apple M2 işlemcili, 8 GB RAM'e sahip ve macOS Sequoia 15.2 versiyonlu MacBook Air (2022) bilgisayarı üzerinden gerçekleştirilmiştir. Model geliştirme sürecinde Python 3.12.8 programlama dili kullanılmıştır.

3.5.1. Destek Vektör Makinesi

Bu çalışmada, yazılım hata kayıtlarının kritiklik seviyelerinin otomatik olarak sınıflandırılabilmesi amacıyla SVM modeli kullanılmıştır. Model, hata kayıtlarına ait Summary ve Description alanlarından elde edilen metin verileri üzerinden eğitilmiş, metinler TF-IDF yöntemi ile sayısal vektörlere dönüştürülerek sınıflandırma işlemi gerçekleştirilmiştir. Veri kümesi, sınıf dağılımı korunacak şekilde %85 eğitim ve %15 test olacak biçimde iki alt kümeye ayrılmıştır. Test veri kümesi deney süreci boyunca sabit tutulmuş ve herhangi bir veri artırma ya da dengeleme işlemine tabi tutulmamıştır.

SVM modelinin yapılandırılması sürecinde, belirli parametre değerlerinden ziyade, model performansını doğrudan etkileyen temel parametreler üzerinde durulmuştur. Bu kapsamda özellikle kernel türü, regularizasyon parametresi (C) ve metin temsiline ilişkin TF-IDF yapılandırmaları farklı kombinasyonlar halinde değerlendirilmiş ve her bir yapılandırma için elde edilen sonuçlar karşılaştırılarak en yüksek genel performansı ve sınıf bazlı dengeyi sağlayan parametre seti tercih edilmiştir.

İlk aşamada, SVM modeli herhangi bir sentetik veri üretimi uygulanmaksızın, orijinal ve dengesiz eğitim veri kümesi üzerinde eğitilmiştir. Bu senaryoda elde edilen performans metrikleri Tablo 7'de sunulmaktadır.

Tablo 7. Dengesiz Veri Kümesi ile Eğitilen SVM Modelinin Performans Metrikleri

Sınıf	**Precision**	**Recall**	**F1-Skoru**	**Support**
Blocker	0.0000	0.0000	0.0000	7
Critical	0.8293	0.7727	0.8000	44
Major	0.7887	0.8485	0.8175	66
Minor	0.8448	0.9245	0.8829	53
Accuracy	—	—	0.8176	170
Genel Ortalama (Macro)	0.6157	0.6364	**0.6251**	170
Genel Ortalama (Weighted)	0.7842	0.8176	**0.7997**	170

Sınıf dengesizliğinin olumsuz etkisini gidermek için yapılan literatür araştırmalarında, küçük veri setleri üzerinde gerçekleştirilen benzer çalışmalarda EDA (Kolay Veri Artırma, Easy Data Augmentation) temelli sentetik veri üretim yönteminin başarılı sonuçlar verdiği gözlemlenmiştir. EDA, sınıf dengesizliğini azaltmaya yönelik olarak geliştirilmiş, metin tabanlı artırma işlemlerine dayanan bir sentetik veri üretim yaklaşımıdır. Metin verisi üzerinde rastgele kelime silme, kelime değiştirme, kelime ekleme ve kelime yer değiştirme gibi basit ancak etkili işlemlerle, mevcut örneklerin anlam bütünlüğünü koruyarak yeni varyasyonlar üreten bir veri artırma tekniği olarak tanımlanmaktadır (Wei ve Zou, 2019). Bu çalışmada EDA yönteminin tercih edilmesinin temel nedeni, metin sınıflandırma problemlerinde az örnek içeren sınıfları desteklerken doğrudan metin verisi üzerinde uygulanabilmesi, kolay uygulanabilir olması ve ek model gereksinimi oluşturmamasıdır. Sınıf dengesizliği probleminin giderilmesinde literatürde kullanılan yöntemlerden biri de SMOTE (Sentetik Azınlık Aşırı Örnekleme Tekniği, Synthetic Minority Over-sampling Technique) yöntemidir. SMOTE, dengesiz veri setlerinde azınlık sınıfına ait örnek sayısını artırmak amacıyla kullanılan bir veri dengeleme yöntemidir. Bu yöntem, azınlık sınıfındaki örnekleri doğrudan kopyalamak yerine, bu örneklerin en yakın komşuları arasındaki ilişkiden yararlanarak özellik uzayında sentetik yeni örnekler üretir (Nitesh ve arkadaşları, 2002). Ancak SMOTE, ham veri üzerinde değil, özellik vektörleri üzerinde çalışan bir yöntemdir. Bu çalışmada kullanılan veri kümesi metin tabanlı hata kayıtlarından oluştuğu için, kelime düzeyinde veri artırma yapabilen ve metinsel yapıyı daha doğrudan koruyan EDA yöntemi tercih edilmiştir.

EDA yöntemi aşağıdaki dört temel işlemden biri rastgele seçilerek uygulanır.

1. Eş Anlamlı Kelime Değiştirme: Cümleden rastgele *n* adet, durak kelime (stop word) olmayan kelime seçilir. Bu kelimelerin her biri, sözlükteki eş anlamlılarından rastgele seçilen bir kelime ile değiştirilir.
2. Rastgele Ekleme: Cümledeki durak kelime olmayan rastgele bir kelimenin eş anlamlısı bulunur ve bu kelime cümlenin rastgele bir konumuna eklenir. Bu işlem *n* kez tekrarlanır.
3. Rastgele Yer Değiştirme: Cümledeki iki kelime rastgele seçilir ve birbirleriyle yer değiştirilir. Bu işlem *n* kez uygulanır.

4. Rastgele Silme: Cümledeki her bir kelime, belirlenen *p* olasılığıyla rastgele silinir. (Wei ve Zou, 2019)

Bu çalışmada EDA yöntemi, yalnızca veri kümesinde ciddi dengesizlik gösteren Blocker sınıfına uygulanmıştır. Critical, Major ve Minor sınıflarının eğitim veri kümesinde yeterli sayıda gerçek örnekle temsil edilmesi nedeniyle bu sınıflar üzerinde herhangi bir sentetik veri üretimi gerçekleştirilmemiştir. Blocker sınıfının eğitim veri kümesindeki örnek sayısı 36'dan 250'ye yükseltilmiştir. EDA işleminin yalnızca eğitim veri kümesi üzerinde uygulanması, test veri kümesinin sabit tutulması ve ardından SVM modelinin TF-IDF tabanlı olarak eğitilmesine ilişkin kod kesiti Şekil 9'da verilmiştir. Kod kesitinde, veri setinin eğitim ve test olarak ayrılması, Blocker sınıfına yönelik EDA uygulaması ve model performansının sabit test verisi üzerinde değerlendirilmesi gösterilmektedir.

```python
def eda_to_blocker_target(X_train, y_train, target=250):
    blocker_idx = y_train[y_train == "Blocker"].index
    current_count = len(blocker_idx)
    needed_count = target - current_count

    X_new, y_new = [], []

    for _ in range(needed_count):
        selected_idx = np.random.choice(blocker_idx)
        augmented_text = augment_text(X_train.loc[selected_idx])
        X_new.append(augmented_text)
        y_new.append("Blocker")

    X_train_eda = pd.concat([X_train, pd.Series(X_new)], ignore_index=True
    y_train_eda = pd.concat([y_train, pd.Series(y_new)], ignore_index=True

    return X_train_eda, y_train_eda


df = pd.read_excel("yl_bug_data.xlsx")

df["text"] = df["Summary"].fillna("") + " " + df["Description"].fillna("")
X = df["text"]
y = df["Severity"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.15, random_state=42, stratify=y
)

X_train_eda, y_train_eda = eda_to_blocker_target(
    X_train, y_train, target=250
)

model = Pipeline([
    ("tfidf", TfidfVectorizer(ngram_range=(1, 2), max_df=0.9)),
    ("classifier", LinearSVC(random_state=42, C=0.5))
])

model.fit(X_train_eda, y_train_eda)
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Macro F1:", f1_score(y_test, y_pred, average="macro"))
print("Weighted F1:", f1_score(y_test, y_pred, average="weighted"))
```

Şekil 9. EDA Uygulaması ve SVM Model Eğitimi Kod Kesiti

EDA işlemi sonrası eğitim veri kümesinin sınıf bazlı dağılımı Tablo 8'de sunulmaktadır.

Tablo 8. EDA Sonrası Dengelenmiş Eğitim Veri Kümesi Dağılımı

Kritiklik Seviyesi	**Adet**	**Oran**
Blocker	250	%21.26
Critical	251	%21.36
Major	373	%31.72
Minor	302	%25.68
Toplam	**1176**	**%100**

Bu dengeli ancak kontrollü veri artırma stratejisiyle eğitilen SVM modelinin performans metrikleri Tablo 9'da gösterilmektedir.

Tablo 9. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen SVM Modelinin Performans Metrikleri

Sınıf	**Precision**	**Recall**	**F1-Skoru**	**Support**
Blocker	0.7500	0.4286	0.5455	7
Critical	0.8947	0.7727	0.8293	44
Major	0.7887	0.8485	0.8175	66
Minor	0.8421	0.9057	0.8727	53
Accuracy	—	—	0.8294	170
Genel Ortalama (Macro)	0.8189	0.7389	**0.7662**	170
Genel Ortalama (Weighted)	0.8312	0.8294	**0.8266**	170

3.5.2. Naive Bayes

Modelin performansını karşılaştırmalı olarak değerlendirebilmek ve farklı sınıflandırma algoritmalarının sınıf dengesizliğine karşı duyarlılığını inceleyebilmek amacıyla, aynı veri kümesi NB algoritması ile de deneysel sürece tabi tutulmuştur. Bu çalışmada, scikit-learn kütüphanesinde yer alan ve metin sınıflandırma problemlerinde yaygın olarak kullanılan Multinomial NB modeli tercih edilmiştir. Model, TF-IDF yöntemi ile sayısal

vektörlere dönüştürülmüş metin temsilleri üzerinden, Bayes Teoremi'ne dayalı olasılık hesaplamaları ile sınıflandırma gerçekleştirmektedir.

NB algoritması, kelimelerin birbirinden bağımsız olduğu varsayımına dayanan yapısı sayesinde yüksek boyutlu metin veri kümelerinde hesaplama açısından verimli sonuçlar üretebilmektedir. Modelde kullanılan alpha parametresi ise Laplace düzeltmesi sağlayarak düşük frekanslı kelimelerden kaynaklanan sıfır olasılık problemini azaltmayı amaçlamaktadır. Bu çalışmada, farklı alpha değerleri denenmiş ve modelin genel performansını ve sınıf bazlı dengeyi en iyi sağlayan yapılandırma tercih edilmiştir.

NB modeli ilk olarak herhangi bir veri artırımı işlemi uygulanmadan, orijinal ve belirgin biçimde dengesiz olan veri kümesi üzerinde eğitilmiştir. Elde edilen sonuçlar Tablo 10'da sunulmuştur.

Tablo 10. Dengesiz Veri Kümesi ile Eğitilen NB Modelinin Performans Metrikleri

Sınıf	**Precision**	**Recall**	**F1-Skoru**	**Support**
Blocker	0.0000	0.0000	0.0000	7
Critical	0.7805	0.7273	0.7529	44
Major	0.7761	0.7879	0.7820	66
Minor	0.7742	0.9057	0.8348	53
Accuracy	—	—	0.7765	170
Genel Ortalama (Macro)	0.5827	0.6052	**0.5924**	170
Genel Ortalama (Weighted)	0.7447	0.7765	**0.7587**	170

Dengesiz veri seti ile elde edilen sonuçlar doğrultusunda, az temsil edilen Blocker sınıfının performansını iyileştirmek amacıyla yalnızca eğitim veri kümesi üzerinde EDA yöntemi uygulanmıştır. Bu kapsamda Blocker sınıfına ait örnek sayısı artırılmış ve diğer sınıflarda herhangi bir sentetik veri üretimi gerçekleştirilmemiştir. Test veri kümesi deney süreci boyunca sabit tutulmuştur. EDA işleminin yalnızca eğitim veri kümesi üzerinde uygulanması ve ardından NB modelinin CountVectorizer tabanlı olarak eğitilmesine ilişkin kod kesiti Şekil 10'da verilmiştir.

```
df = pd.read_excel("yl_bug_data.xlsx")

df["text"] = df["Summary"].fillna("") + " " + df["Description"].fillna("")
X = df["text"]
y = df["Severity"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.15, random_state=42, stratify=y
)

X_train_eda, y_train_eda = apply_eda_blocker_only(
    X_train, y_train, target=250
)

model = Pipeline([
    ("vectorizer", CountVectorizer(
        ngram_range=(1, 2),
        max_df=0.95
    )),
    ("classifier", MultinomialNB(alpha=1.5))
])

model.fit(X_train_eda, y_train_eda)
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Macro F1:", f1_score(y_test, y_pred, average="macro"))
print("Weighted F1:", f1_score(y_test, y_pred, average="weighted"))
```

Şekil 10. EDA Uygulanmış Eğitim Verisi ile NB Model Eğitimi Kod Kesiti

EDA sonrası dengelenmiş eğitim veri kümesi kullanılarak NB modeli yeniden eğitilmiş ve elde edilen performans metrikleri Tablo 11'de sunulmuştur.

Tablo 11. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen NB Modelinin Performans Metrikleri

Sınıf	Precision	Recall	F1-Skoru	Support
Blocker	0.3750	0.4286	0.4000	7
Critical	0.8649	0.7273	0.7901	44
Major	0.7656	0.7424	0.7538	66
Minor	0.7869	0.9057	0.8421	53
Accuracy	—	—	0.7765	170
Genel Ortalama (Macro)	0.6981	0.7010	**0.6965**	170
Genel Ortalama (Weighted)	0.7819	0.7765	**0.7762**	170

3.5.3. Rastgele Orman

Modelin performansını karşılaştırmalı olarak inceleyebilmek ve farklı algoritmaların sınıf dengesizliğinden nasıl etkilendiğini gözlemleyebilmek amacıyla aynı veri kümesi RF ile de ayrı bir deneysel süreçten geçirilmiştir. Bu çalışmada RF modeli, scikit-learn kütüphanesi tarafından sağlanan RandomForestClassifier fonksiyonu kullanılarak oluşturulmuştur. Model, metin verilerinden elde edilen özellikleri temsil etmek amacıyla TF-IDF yöntemiyle dönüştürülmüş kelime ağırlıklarını girdi olarak kullanmakta ve bu özellikler üzerinden çok sayıda karar ağacını topluluk öğrenme yaklaşımıyla eğitmektedir.

RF modeli, sınıf olasılıklarına dayalı basit varsayımlar yerine çoklu karar ağaçlarının birleşik gücünü kullanması sayesinde, özellikle karmaşık karar sınırlarına sahip veri kümelerinde ve sınıf dağılımının dengesiz olduğu senaryolarda daha kararlı bir performans sergileyebilmektedir. Bu çalışmada RF modeli, herhangi bir veri artırımı veya dengeleme işlemi uygulanmadan, orijinal ve dengesiz veri kümesi üzerinde eğitilmiş, elde edilen performans metrikleri Tablo 12'de sunulmuştur.

Tablo 12. Dengesiz Veri Kümesi ile Eğitilen RF Modelinin Performans Metrikleri

Sınıf	Precision	Recall	F1-Skoru	Support
Blocker	0.0000	0.0000	0.0000	7
Critical	0.8919	0.7500	0.8148	44
Major	0.8088	0.8333	0.8209	66
Minor	0.7692	0.9434	0.8475	53
Accuracy	—	—	0.8118	170
Genel Ortalama (Macro)	0.6175	0.6317	**0.6208**	170
Genel Ortalama (Weighted)	0.7847	0.8118	**0.7938**	170

Dengesiz veri seti ile elde edilen sonuçlar doğrultusunda, az temsil edilen Blocker sınıfının performansını iyileştirmek amacıyla yalnızca eğitim veri kümesi üzerinde EDA yöntemi uygulanmıştır. Bu kapsamda Blocker sınıfına ait örnek sayısı artırılmış ve diğer sınıflarda herhangi bir sentetik veri üretimi gerçekleştirilmemiştir. Test veri kümesi deney süreci boyunca sabit tutulmuştur. EDA işleminin yalnızca eğitim veri kümesi üzerinde uygulanması ve ardından RF modelinin TF-IDF tabanlı olarak eğitilmesine ilişkin kod kesiti Şekil 11'de verilmiştir.

```
df = pd.read_excel("yl_bug_data.xlsx")

df["text"] = df["Summary"].fillna("") + " " + df["Description"].fillna("")
X = df["text"]
y = df["Severity"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.15, random_state=42, stratify=y
)

X_train_eda, y_train_eda = apply_eda_blocker_only(
    X_train, y_train, target=250
)

model = Pipeline([
    ("tfidf", TfidfVectorizer(
        ngram_range=(1, 2),
        max_features=2000,
        max_df=0.9
    )),
    ("classifier", RandomForestClassifier(
        n_estimators=300,
        random_state=42,
        n_jobs=-1
    ))
])

model.fit(X_train_eda, y_train_eda)
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Macro F1:", f1_score(y_test, y_pred, average="macro"))
print("Weighted F1:", f1_score(y_test, y_pred, average="weighted"))
```

Şekil 11. EDA Uygulanmış Eğitim Verisi ile RF Model Eğitimi Kod Kesiti

EDA sonrası dengelenmiş eğitim veri kümesi kullanılarak RF modeli yeniden eğitilmiş ve elde edilen performans metrikleri Tablo 13'te sunulmuştur.

Tablo 13. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen RF Modelinin Performans Metrikleri

Sınıf	Precision	Recall	F1-Skoru	Support
Blocker	0.3750	0.4286	0.4000	7
Critical	0.9412	0.7273	0.8205	44
Major	0.8060	0.8182	0.8120	66
Minor	0.8033	0.9245	0.8596	53
Accuracy	—	—	0.8118	170
Genel Ortalama (Macro)	0.7314	0.7246	**0.7230**	170
Genel Ortalama (Weighted)	0.8224	0.8118	**0.8121**	170

3.5.4. Karar Ağacı

Modelin performansını karşılaştırmalı olarak inceleyebilmek ve farklı algoritmaların sınıf dengesizliğinden nasıl etkilendiğini gözlemleyebilmek amacıyla aynı veri kümesi DT algoritması ile de ayrı bir deneysel süreçten geçirilmiştir. Bu çalışmada DT modeli, scikit-learn kütüphanesi tarafından sağlanan DecisionTreeClassifier fonksiyonu kullanılarak oluşturulmuştur. Model, metin verilerinden elde edilen özellikleri temsil etmek amacıyla TF-IDF yöntemiyle dönüştürülmüş kelime ağırlıklarını girdi olarak kullanmakta ve bu özellikler üzerinden hiyerarşik bir karar yapısı inşa etmektedir.

DT algoritması, sınıflandırma sürecini ardışık ve kural tabanlı bölünmeler aracılığıyla gerçekleştirmektedir. Her bir iç düğümde, seçilen bir öznitelik için belirlenen eşik değere göre veri kümesini daha homojen alt kümelere ayırmaktadır. Bu bölünmeler, sınıf saflığını en üst düzeye çıkarmayı hedefleyen ölçütler doğrultusunda belirlenmektedir. Bu çalışmada kullanılan modelde, düğüm bölünmelerinin belirlenmesinde varsayılan olarak Gini safsızlık ölçütü (criterion="gini") tercih edilmiş, ağaç yapısının karmaşıklığını sınırlamak amacıyla maksimum derinlik (max_depth), minimum bölünme örnek sayısı (min_samples_split) ve yaprak düğüm başına minimum örnek sayısı (min_samples_leaf) gibi hiperparametreler kontrol altında tutulmuştur.

Bu çalışmada DT modeli, herhangi bir veri artırımı veya sınıf dengeleme işlemi uygulanmadan, orijinal ve belirgin biçimde dengesiz olan veri kümesi üzerinde eğitilmiştir. Elde edilen performans metrikleri Tablo 14'te sunulmuştur.

Tablo 14. Dengesiz Veri Kümesi ile Eğitilen DT Modelinin Performans Metrikleri

Sınıf	**Precision**	**Recall**	**F1-Skoru**	**Support**
Blocker	0.2000	0.1429	0.1667	7
Critical	0.7255	0.8409	0.7789	44
Major	0.7619	0.7273	0.7442	66
Minor	0.8235	0.7925	0.8077	53
Accuracy	—	—	0.7529	170
Genel Ortalama (Macro)	0.6277	0.6259	**0.6244**	170
Genel Ortalama (Weighted)	0.7486	0.7529	**0.7492**	170

Dengesiz veri seti ile elde edilen sonuçlar doğrultusunda, az temsil edilen Blocker sınıfının performansını iyileştirmek amacıyla yalnızca eğitim veri kümesi üzerinde EDA yöntemi uygulanmıştır. Bu kapsamda Blocker sınıfına ait örnek sayısı artırılmıştır ve diğer sınıflar üzerinde herhangi bir sentetik veri üretimi gerçekleştirilmemiştir. Test veri kümesi deney süreci boyunca sabit tutulmuştur. EDA işleminin yalnızca eğitim veri kümesi üzerinde uygulanması ve ardından DT modelinin TF-IDF tabanlı olarak eğitilmesine ilişkin kod kesiti Şekil 12'de verilmiştir.

```
df = pd.read_excel("yl_bug_data.xlsx")

df["text"] = df["Summary"].fillna("") + " " + df["Description"].fillna("")
X = df["text"]
y = df["Severity"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.15, random_state=42, stratify=y
)

X_train_eda, y_train_eda = apply_eda_blocker_only(
    X_train, y_train, target=250
)

vectorizer = TfidfVectorizer(
    ngram_range=(1, 2),
    min_df=1
)

X_train_eda_vec = vectorizer.fit_transform(X_train_eda)
X_test_vec = vectorizer.transform(X_test)

model = DecisionTreeClassifier(
    random_state=42,
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1
)

model.fit(X_train_eda_vec, y_train_eda)
y_pred = model.predict(X_test_vec)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Macro F1:", f1_score(y_test, y_pred, average="macro"))
print("Weighted F1:", f1_score(y_test, y_pred, average="weighted"))
```

Şekil 12. EDA Uygulanmış Eğitim Verisi ile DT Model Eğitimi Kod Kesiti

EDA sonrası dengelenmiş eğitim veri kümesi kullanılarak DT modeli yeniden eğitilmiş ve elde edilen performans metrikleri Tablo 15'te sunulmuştur.

Tablo 15. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen DT Modelinin Performans Metrikleri

Sınıf	Precision	Recall	F1-Skoru	Support
Blocker	0.2727	0.4286	0.3333	7
Critical	0.8250	0.7500	0.7368	44
Major	0.7313	0.7424	0.7368	66
Minor	0.8269	0.8113	0.8190	53
Accuracy	—	—	0.7529	170
Genel Ortalama (Macro)	0.6640	0.6831	**0.6687**	170
Genel Ortalama (Weighted)	0.7665	0.7529	**0.7585**	170

3.5.5. XGBoost

Modelin performansını karşılaştırmalı olarak inceleyebilmek ve farklı algoritmaların sınıf dengesizliğinden nasıl etkilendiğini gözlemleyebilmek amacıyla, aynı veri kümesi bu çalışmada XGBoost algoritması kullanılarak da ayrı bir deneysel sürece tabi tutulmuştur. XGBoost modeli, Python ortamında XGBoost kütüphanesi tarafından sağlanan XGBClassifier sınıfı kullanılarak oluşturulmuştur. Modelin girdileri, hata raporlarından elde edilen metinsel verilerin TF-IDF yöntemi ile sayısal vektörlere dönüştürülmesiyle elde edilen özelliklerden oluşmaktadır.

Bu çalışmada kullanılan XGBoost modelinde, öğrenme sürecinin kontrol altında tutulması ve aşırı öğrenmenin azaltılması amacıyla çeşitli hiperparametreler dikkate alınmıştır. Modelde kullanılan ağaç sayısı (n_estimators), oluşturulacak toplam karar ağacı sayısını belirlemekte olup, modelin öğrenme kapasitesini doğrudan etkilemektedir. Öğrenme oranı (learning_rate), her bir ağacın modele katkı düzeyini ölçeklendirmekte ve modelin daha kademeli bir şekilde öğrenmesini sağlamaktadır. Ağaçların karmaşıklığını sınırlamak amacıyla maksimum derinlik (max_depth) parametresi kullanılmıştır. Böylece aşırı karmaşık ağaç yapılarının oluşması engellenmiştir.

Ayrıca, her bir düğümde bölünme yapılabilmesi için gereken minimum örnek sayısını belirleyen min_child_weight parametresi ile, bölünme sonrasında elde edilen yaprak düğümlerin daha güvenilir istatistikler içermesi amaçlanmıştır. Subsample ve colsample_bytree parametreleri aracılığıyla, her bir ağacın eğitimi sırasında kullanılan örneklerin ve özniteliklerin bir alt kümesi seçilerek modelin genelleme yeteneği artırılmıştır. XGBoost'un öğrenme hedefinde yer alan düzenlileştirme (regularization) terimleri kapsamında, yaprak ağırlıklarını cezalandıran L2 düzenlileştirme katsayısı (reg_lambda) kullanılarak model karmaşıklığı kontrol altında tutulmuştur.

Bu çalışmada XGBoost modeli, herhangi bir veri artırımı veya sınıf dengeleme yöntemi uygulanmaksızın, orijinal ve belirgin biçimde dengesiz veri kümesi üzerinde eğitilmiştir. Elde edilen performans sonuçları Tablo 16'da sunulmuştur.

Tablo 16. Dengesiz Veri Kümesi ile Eğitilen XGBoost Modelinin Performans Metrikleri

Sınıf	**Precision**	**Recall**	**F1-Skoru**	**Support**
Blocker	0.6000	0.4284	0.5000	7
Critical	0.7551	0.8409	0.7957	44
Major	0.8033	0.7424	0.7717	66
Minor	0.8364	0.8679	0.8519	53
Accuracy	—	—	0.7941	170
Genel Ortalama (Macro)	0.7487	0.7200	**0.7298**	170
Genel Ortalama (Weighted)	0.7928	0.7941	**0.7917**	170

Dengesiz veri seti ile elde edilen sonuçlar doğrultusunda, az temsil edilen Blocker sınıfının performansını iyileştirmek amacıyla yalnızca eğitim veri kümesi üzerinde EDA yöntemi uygulanmıştır. Bu kapsamda Blocker sınıfına ait örnek sayısı artırılmıştır ve diğer sınıflar üzerinde herhangi bir sentetik veri üretimi gerçekleştirilmemiştir. Test veri kümesi deney süreci boyunca sabit tutulmuştur. EDA işleminin yalnızca eğitim veri kümesi üzerinde uygulanması ve ardından XGBoost modelinin TF-IDF tabanlı olarak eğitilmesine ilişkin kod kesiti Şekil 13'te verilmiştir.

```
df = pd.read_excel("yl_bug_data.xlsx")

df["text"] = df["Summary"].fillna("") + " " + df["Description"].fillna("")
X = df["text"]
y = df["Severity"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.15, random_state=42, stratify=y
)

X_train_eda, y_train_eda = apply_eda_blocker_only(
    X_train, y_train, target=250
)

label_encoder = LabelEncoder()
label_encoder.fit(["Blocker", "Critical", "Major", "Minor"])

y_train_eda_enc = label_encoder.transform(y_train_eda)
y_test_enc = label_encoder.transform(y_test)

vectorizer = TfidfVectorizer(
    ngram_range=(1, 2),
    min_df=1
)

X_train_eda_vec = vectorizer.fit_transform(X_train_eda)
X_test_vec = vectorizer.transform(X_test)

model = XGBClassifier(
    objective="multi:softmax",
    num_class=4,
    random_state=42,
    n_estimators=400,
    max_depth=6,
    learning_rate=0.1,
    tree_method="hist",
    eval_metric="mlogloss"
)

model.fit(X_train_eda_vec, y_train_eda_enc)
y_pred = model.predict(X_test_vec)

print("Accuracy:", accuracy_score(y_test_enc, y_pred))
print("Macro F1:", f1_score(y_test_enc, y_pred, average="macro"))
print("Weighted F1:", f1_score(y_test_enc, y_pred, average="weighted"))
```

Şekil 13. EDA Uygulanmış Eğitim Verisi ile XGBoost Model Eğitimi Kod Kesiti

EDA sonrası dengelenmiş eğitim veri kümesi kullanılarak XGBoost modeli yeniden eğitilmiş ve elde edilen performans metrikleri Tablo 17'de sunulmuştur.

Tablo 17. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen XGBoost Modelinin Performans Metrikleri

Sınıf	Precision	Recall	F1-Skoru	Support
Blocker	0.6667	0.5714	0.6154	7
Critical	0.8043	0.8409	0.8222	44
Major	0.8095	0.7727	0.7907	66
Minor	0.8364	0.8679	0.8519	53
Accuracy	—	—	0.8118	170
Genel Ortalama (Macro)	0.7792	0.7632	**0.7700**	170
Genel Ortalama (Weighted)	0.8107	0.8118	**0.8107**	170

3.5.6. LightGBM

Bu çalışmada, yazılım hatalarının kritiklik seviyesinin tahmin edilmesi amacıyla LightGBM algoritması kullanılmıştır. Model eğitimi öncesinde, sınıflandırma performansını ve genelleme yeteneğini artırmak amacıyla çeşitli hiperparametreler belirlenmiş ve kontrol altında tutulmuştur. LightGBM, gradyan artırma yaklaşımına dayalı olarak ardışık biçimde eğitilen karar ağaçlarından oluştuğundan, modelin karmaşıklığı ve öğrenme süreci bu hiperparametreler aracılığıyla düzenlenmektedir.

Modelde kullanılan öğrenme oranı (learning_rate), her bir ağacın modele olan katkı düzeyini belirlemekte olup, küçük değerler tercih edilerek aşırı öğrenmenin önüne geçilmesi amaçlanmıştır. Ağaç sayısı (n_estimators), öğrenme oranı ile değerlendirilmiş ve modelin yeterli temsil gücüne ulaşmasını sağlayacak biçimde belirlenmiştir. Karar ağaçlarının yapısal karmaşıklığını sınırlamak amacıyla maksimum ağaç derinliği (max_depth) ve yaprak düğüm başına minimum örnek sayısı (min_data_in_leaf) parametreleri kullanılmıştır. Bu parametreler, özellikle küçük veya dengesiz veri kümelerinde modelin aşırı uyum göstermesini engellemede kritik rol oynamaktadır.

LightGBM'in histogram tabanlı bölünme yaklaşımı kapsamında kullanılan özellik sayısı (num_leaves) parametresi, ağaçların öğrenme kapasitesini doğrudan etkilemektedir. Bu

çalışmada, num_leaves değeri, max_depth ile uyumlu olacak şekilde sınırlandırılarak model karmaşıklığının kontrol edilmesi hedeflenmiştir. Ayrıca, her bir ağacın eğitimi sırasında rastgele seçilen örnek oranını belirleyen alt örnekleme oranı (subsample) ve rastgele seçilen özellik oranını belirleyen özellik alt örnekleme oranı (colsample_bytree) parametreleri kullanılarak, modelin genelleme performansının artırılması amaçlanmıştır.

Son olarak, LightGBM tarafından sağlanan düzenlileştirme (regularization) mekanizmaları kapsamında L1 (lambda_l1) ve L2 (lambda_l2) ceza terimleri kullanılarak modelin aşırı öğrenmeye karşı dayanıklılığı artırılmıştır. Bu hiperparametrelerin beraber kullanılmasıyla, LightGBM modelinin sınıflar arasında dengeli bir sınıflandırma performansı sağlaması hedeflenmiştir.

Bu çalışmada LightGBM modeli, herhangi bir veri artırımı veya sınıf dengeleme yöntemi uygulanmadan, orijinal ve dengesiz veri kümesi üzerinde eğitilmiştir. Elde edilen performans sonuçları Tablo 18'de sunulmuştur.

Tablo 18. Dengesiz Veri Kümesi ile Eğitilen LightGBM Modelinin Performans Metrikleri

Sınıf	**Precision**	**Recall**	**F1-Skoru**	**Support**
Blocker	0.4000	0.2857	0.3333	7
Critical	0.8605	0.8409	0.8506	44
Major	0.8615	0.8485	0.8550	66
Minor	0.8596	0.9245	0.8909	53
Accuracy	—	—	0.8471	170
Genel Ortalama (Macro)	0.7454	0.7249	**0.7324**	170
Genel Ortalama (Weighted)	0.8417	0.8471	**0.8436**	170

Dengesiz veri seti ile elde edilen sonuçlar doğrultusunda, az temsil edilen Blocker sınıfının performansını iyileştirmek amacıyla yalnızca eğitim veri kümesi üzerinde EDA yöntemi uygulanmıştır. Bu kapsamda Blocker sınıfına ait örnek sayısı artırılmıştır ve diğer sınıflar üzerinde herhangi bir sentetik veri üretimi gerçekleştirilmemiştir. Test veri kümesi deney süreci boyunca sabit tutulmuştur. EDA uygulanmış eğitim veri kümesi kullanılarak LightGBM modelinin TF-IDF tabanlı olarak eğitilmesine ilişkin kod kesiti Şekil 14'te verilmiştir.

```
df = pd.read_excel("yl_bug_data.xlsx")

df["text"] = df["Summary"].fillna("") + " " + df["Description"].fillna("")
X = df["text"]
y = df["Severity"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.15, random_state=42, stratify=y
)

X_train_eda, y_train_eda = apply_eda_blocker_only(
    X_train, y_train, target=250
)

vectorizer = TfidfVectorizer(
    ngram_range=(1, 2),
    min_df=1
)

X_train_eda_vec = vectorizer.fit_transform(X_train_eda)
X_test_vec = vectorizer.transform(X_test)

model = LGBMClassifier(
    objective="multiclass",
    num_class=4,
    boosting_type="gbdt",
    n_estimators=250,
    learning_rate=0.10,
    num_leaves=31,
    random_state=42,
    n_jobs=-1
)

model.fit(X_train_eda_vec, y_train_eda)
y_pred = model.predict(X_test_vec)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Macro F1:", f1_score(y_test, y_pred, average="macro"))
print("Weighted F1:", f1_score(y_test, y_pred, average="weighted"))
```

Şekil 14. EDA Uygulanmış Eğitim Verisi ile LightGBM Model Eğitimi Kod Kesiti

EDA sonrası dengelenmiş eğitim veri kümesi kullanılarak LightGBM modeli yeniden eğitilmiş ve elde edilen performans metrikleri Tablo 19'da sunulmuştur.

Tablo 19. EDA Sonrası Dengelenmiş Veri Kümesi ile Eğitilen LightGBM Modelinin Performans Metrikleri

Sınıf	Precision	Recall	F1-Skoru	Support
Blocker	0.6667	0.2857	0.4000	7
Critical	0.8163	0.9091	0.8602	44
Major	0.8710	0.8182	0.8438	66
Minor	0.8571	0.9057	0.8807	53
Accuracy	—	—	0.8471	170
Genel Ortalama (Macro)	0.8028	0.7297	**0.7462**	170
Genel Ortalama (Weighted)	0.8441	0.8471	**0.8413**	170

3.6. Model Değerlendirme

Eğitim sürecinde, sınıflandırma sonuçlarının değerlendirilmesinde accuracy**,** precision**,** recall ve F1-skoru gibi temel performans ölçütleri kullanılmıştır. Bu performans ölçütlerinin tanımlamaları Tablo 20'de gösterilmektedir.

Tablo 20. Sınıflandırma Modellerinde Kullanılan Performans Ölçütleri ve Açıklamaları

Ölçüt	Açıklama
Accuracy (Doğruluk Oranı)	Modelin tüm test verisi üzerinde doğru sınıflandırdığı örneklerin oranını ifade eder.
Precision (Kesinlik)	Model tarafından belirli bir sınıfa ait olduğu tahmin edilen örneklerin gerçekten o sınıfa ait olma oranıdır.
Recall (Duyarlılık / Bulma Oranı)	Gerçek örneklerin model tarafından ne kadar doğru bulunabildiğini gösterir.

F1-Skoru	Precision ve recall değerlerinin harmonik ortalamasıdır ve sınıf dengesizliği durumlarında daha güvenilir sonuç verir.
Support (Destek Sayısı)	Test kümesindeki ilgili sınıfa ait örnek sayısını ifade etmektedir.
Macro-F1	Her sınıfa eşit ağırlık vererek F1 skorlarının aritmetik ortalamasını alır. Sınıf dengesizliği bulunan veri setlerinde daha sağlıklı bir karşılaştırma sağlar.
Weighted-F1	Her sınıfın F1 skorunun, o sınıfın support değeri ile ağırlıklandırılmış ortalamasıdır. Veri kümesinin gerçek dağılımını dikkate almaktadır.

Bu çalışmada geliştirilen sınıflandırma modellerinin performanslarının değerlendirilmesi amacıyla çeşitli istatistiksel ölçütler kullanılmıştır. Veri setinde sınıf dengesizliği bulunması nedeniyle model başarımının yalnızca doğruluk (Accuracy) metriği üzerinden değerlendirilmesinin yeterli olmayacağı öngörülmüştür. Bu nedenle değerlendirme sürecinde Accuracy, Precision, Recall, F1-Score, Macro F1 ve Weighted F1 metrikleri birlikte kullanılmıştır.

Accuracy metriği, modelin toplam doğru tahmin oranını vermektedir. Fakat dengesiz veri setlerinde çoğunluk olan sınıflar doğru tahmin edildiğinde yüksek doğruluk elde edilebilmekteyken azınlık sınıflardaki başarısızlıklar bu metrikte yeterince yansımamaktadır. Bu sebeple sınıflar arası dengeyi daha adil şekilde ölçümleyebilmek adına Macro F1 metriği burada temel değerlendirme ölçütü olarak ele alınmıştır. Macro F1, her bir sınıf için hesaplanan F1 skorlarının eşit ağırlıklı ortalamasını alarak azınlık sınıfların performansını daha görünür hale getirmektedir. Weighted F1 ise sınıf örnek

sayısı dikkate alınarak hesaplanmakta olup genel performansın dağılımını yansıtmaktadır.

Ayrıca azınlık olan Blocker sınıfının performansını daha detaylı inceleyebilmek adına sınıf bazlı F1 skorları da değerlendirilmiştir. Kritik hata tespitinin yazılım geliştirme süreçlerinde büyük öneme sahip olmasıyla, Blocker sınıfına ait performans sonuçları model karşılaştırmalarında dikkatle incelenmiştir.

Bu kapsamda elde edilmiş performans sonuçları tablolar ile sunulmuş ve seçilen modeller hem genel metrikler hem de sınıf bazlı ölçütler açısından karşılaştırılma yapılarak değerlendirilmeye alınmıştır.

3.7. Model İyileştirme

Model eğitimi ve elde edilen ilk performans değerlendirmelerden sonra, modellerin başarısını yükseltme amacıyla çeşitli iyileştirme çalışmaları yapılmıştır. Bu kapsamda, özellikle sınıf dengesizliğinin performans üzerindeki etkisini azaltmak amacıyla veri artırma ve dengeleme çözümleri uygulanmıştır. Azınlık sınıfların daha iyi temsil edilmesi sağlanarak modellerin sınıflar arası öğrenme dengesi iyileştirilmiştir.

Bununla beraber, kullanılan makine öğrenmesi algoritmalarının temel hiper parametrelerinde ayarlamalar gerçekleştirilmiştir. Parametre ayarlama sürecinde farklı değerler ile kombinasyonlar denenmiş ve performans metrikleri karşılaştırılarak en iyi sonuç veren ayar belirlenmiştir. Özellikle sınıflar arası performansı daha adil şekilde belirleyen Macro F1 metriği, iyileştirme sürecinde en önemli değerlendirme ölçütü olarak belirlenmiştir.

Gerçekleştirilen iyileştirme çalışmaları sonucunda modellerin genel performansında ve özellikle azınlık sınıflardaki tahmin başarımında artış gözlemlenmiştir. Bu süreç, modelin genelleme yeteneğinin artırılmasına ve daha dengeli sonuçlar elde edilmesine katkı sağlamıştır.

3.8. Model Dağıtım ve İzleme

Bu çalışma kapsamında geliştirilen makine öğrenmesi modelleri deneysel ortamda eğitilmiş ve performansları test veri seti üzerinde değerlendirilmiştir. Gerçek zamanlı bir

üretim ortamına dağıtım işlemi gerçekleştirilmemiştir. Ancak elde edilen sonuçlar, modelin yazılım hata takip sistemlerine entegre edilmesi durumunda otomatik kritiklik tahmini amacıyla kullanılabileceğini göstermektedir.

Modelin gerçek bir yazılım geliştirme sürecine entegre edilmesi halinde, yeni oluşturulan hata kayıtlarının otomatik olarak analiz edilmesi ve tahmini kritiklik seviyesinin öneri olarak sunulması mümkün olacaktır.

4. BULGULAR VE TARTIŞMA

4.1 Bulgular

Bu bölümde, yazılım hata kayıtlarının makine öğrenmesi yöntemleri ile sınıflandırılması sürecinde elde edilen deneysel bulgular sistematik bir biçimde sunulmaktadır. Çalışma kapsamında kullanılan her model, öncelikle sınıf dengesizliği bulunan ham veri seti üzerinde eğitilmiş ve performans sonuçları analiz edilmiştir. Ardından, sınıf dengesizliğini azaltmak amacıyla uygulanan EDA yöntemi sonrasında aynı modeller yeniden eğitilerek performans değişimleri incelenmiştir.

Bu doğrultuda, öncelikle modeller tek tek ele alınarak EDA öncesi ve sonrası performansları detaylı biçimde değerlendirilmiş, sonrasında ise tüm modeller karşılaştırmalı olarak analiz edilerek genel bir model değerlendirmesi yapılmıştır.

4.1.1 Destek vektör makinesi modeli sonuçları

Bu bölümde, SVM modelinin sınıf dengesizliği bulunan veri seti üzerindeki performansı ve EDA uygulaması sonrasında elde edilen sonuçları ayrıntılı olarak incelenmektedir. İlk olarak dengesiz veri seti ile elde edilen performans metrikleri değerlendirilmiş, ardından veri artırma sürecinin model başarımına olan etkisi analiz edilmiştir.

Elde edilen sonuçlara göre, dengesiz veri kümesi ile eğitilen SVM modelinin genel doğruluk oranı %81,76, weighted-F1 skoru ise 0,7997 olarak hesaplanmıştır. Sınıf bazlı performans sonuçları incelendiğinde, Major, Minor ve Critical sınıflarında yüksek ve dengeli sonuçlar elde edildiği görülmektedir. Bununla beraber, Blocker sınıfı için precision, recall ve F1-skoru 0,0000 olarak hesaplanmıştır.

Bu durum, modelin çoğunluk sınıfları başarılı şekilde öğrenmesine rağmen, az veri olan Blocker sınıfını ayırt edemediğini açıkça göstermektedir. Eğitim veri kümesinde Blocker sınıfının yalnızca 36 veri olması, modelin bu sınıfa ait örüntüleri öğrenmesini zorlaştırmıştır. Bu sonuçlara göre, sınıf dengesizliğinin model performansı üzerindeki etkisini düşürmek hedeflenerek Blocker sınıfına EDA yöntemi uygulanmıştır.

EDA işlemi sonrası tekrar eğitilen modelin sonuçları incelendiğinde, genel doğruluk oranının %82,94'e yükseldiği, weighted-F1 skorunun 0,8266, macro-F1 skorunun ise 0,7662 olarak sonuçlandığı görülmektedir. En önemli iyileşme Blocker sınıfında gerçekleşmiştir. EDA öncesinde 0/7 doğru tahmin ederken, EDA sonrası 3/7 örnek doğru tahmin edilerek sınıflandırılmıştır. Blocker sınıfının recall değeri 0,0000'dan 0,4286'ya yükselmiştir.

Bu sonuçlar, EDA uygulamasının modelin genel doğruluk artışından çok, azınlık sınıfın performansını iyileştirmesi açısından etkili olduğunu göstermektedir.

4.1.2 Naive Bayes modeli sonuçları

Bu bölümde NB modelinin dengesiz veri seti üzerinde eğitilmiş performansı ve EDA uygulaması sonrasında elde edilen sonuçlar karşılaştırma yapılarak değerlendirilmiştir.

Sınıflar arası dengesizlik bulunan veri kümesi ile eğitilen NB modelinin genel doğruluk oranı %77,65, weighted-F1 skoru ise 0,7587 olarak hesaplanmıştır. Major, Minor ve Critical sınıflarında kısmen dengeli sonuçlar elde edilmiş olmakla beraber, Blocker sınıfı için precision, recall ve F1-skoru değerleri 0,0000 olarak hesaplanmıştır. Test veri kümesinde 7 adet Blocker örneğinin hiçbirinin doğru sınıflandırılamamış olması, modelin az örnekli sınıfları ayırt etmede yetersiz kaldığını göstermektedir.

NB modeli EDA uygulanarak yeniden eğitildiğinde genel doğruluk oranının %77,65 seviyesinde korunduğu, macro-F1 skorunun ise 0,5924'ten 0,6965'e yükseldiği görülmektedir. Özellikle sınıf dengesizliğini daha iyi yansıtan macro-F1 metriğindeki bu artış, veri artırma işleminin sınıflar arası öğrenme dengesini iyileştirdiğini göstermektedir.

En göze çarpan iyileşme Blocker sınıfında gerçekleşmiştir. EDA öncesinde 0/7 doğru tahmin yapılırken, EDA sonrasında 3/7 örnek doğru tahminlenerek sınıflandırılmıştır. Blocker sınıfının recall değeri 0,00'dan 0,4286'ya, F1-skoru ise 0,00'dan 0,40'a yükselmiştir.

Bu sonuçlar, NB modelinin sınıf dengesizliğinden önemli ölçüde etkilendiğini, ancak veri artırma yönteminin azınlık sınıfının öğrenilebilirliğini yüksek düzeyde artırdığını göstermektedir.

4.1.3 Rastgele orman modeli sonuçları

Bu bölümde RF modelinin dengesiz veri seti üzerindeki performansı ve EDA uygulaması sonrasında elde edilen sonuçlar karşılaştırmalı olarak değerlendirilmiştir.

Dengesiz veri kümesi ile eğitilen RF modelinin genel doğruluk oranı %81,18 (Accuracy=0,8118), weighted-F1 skoru 0,7938 ve macro-F1 skoru 0,6208 olarak hesaplanmıştır. Major, Minor ve Critical sınıflarında görece dengeli performans değerleri elde edilmiş olmakla birlikte, Blocker sınıfına ait precision, recall ve F1-skoru değerleri 0,0000 olarak hesaplanmıştır. Test veri kümesindeki 7 Blocker örneğinin hiçbirinin doğru sınıflandırılamamış olması, modelin az temsil edilen sınıflara karşı yetersiz kaldığını göstermektedir.

EDA uygulanarak model yeniden eğitildiğinde genel doğruluk oranının %81,18 seviyesinde korunduğu, macro-F1 skorunun 0,6208'den 0,7230'e yükseldiği ve weighted-F1 skorunun 0,7938'den 0,8121'e çıktığı görülmektedir. Macro-F1 metriğindeki artış, iyileşmenin özellikle sınıflar arası dengeye yansıdığını göstermektedir.

En belirgin iyileşme Blocker sınıfında gözlemlenmiştir. EDA öncesinde 0/7 doğru tahmin yapılırken, EDA sonrasında 3/7 örnek doğru sınıflandırılmıştır. Blocker sınıfının recall değeri 0,00'dan 0,4286'ya, F1-skoru ise 0,00'dan 0,4000'e yükselmiştir.

Diğer sınıflarda belirgin bir performans kaybı gözlemlenmemiştir. Critical ve Minor sınıflarında sınırlı iyileşmeler görülürken, Major sınıfında çok küçük bir düşüş meydana gelmiştir ancak modelin genel performans dengesi korunmuştur.

Bu bulgular, RF modelinin dengesiz veri setinde genel doğruluk açısından güçlü bir performans sergilemesine rağmen azınlık sınıfı öğrenmekte zorlandığını, EDA tabanlı hedefli veri artırma yaklaşımının ise az temsil edilen Blocker sınıfının öğrenilebilirliğini anlamlı düzeyde artırdığını göstermektedir.

4.1.4 Karar ağacı modeli sonuçları

Bu bölümde DT modelinin dengesiz veri seti üzerindeki performansı ve EDA uygulaması sonrasında elde edilen sonuçlar karşılaştırmalı olarak değerlendirilmiştir.

Dengesiz veri kümesi ile eğitilen DT modelinin genel doğruluk oranı %75,29 (Accuracy=0,7529), weighted-F1 skoru 0,7492 ve macro-F1 skoru 0,6244 olarak hesaplanmıştır. Major, Minor ve Critical sınıflarında görece dengeli performans değerleri elde edilmiş olmakla birlikte, Blocker sınıfında belirgin bir performans düşüklüğü gözlemlenmiştir. Model, test kümesindeki 7 Blocker örneğinin yalnızca 1'ini doğru sınıflandırabilmiş. Blocker sınıfı için recall=0,1429 ve F1-skoru=0,1667 olarak hesaplanmıştır.

EDA uygulanarak model yeniden eğitildiğinde genel doğruluk oranının %75,29 seviyesinde korunduğu, macro-F1 skorunun 0,6244'ten 0,6687'ye yükseldiği ve weighted-F1 skorunun 0,7492'den 0,7585'e çıktığı görülmektedir. Macro-F1 metriğindeki artış, veri artırma işleminin sınıflar arası performans dengesini iyileştirdiğini göstermektedir.

En belirgin iyileşme Blocker sınıfında ortaya çıkmıştır. EDA öncesinde 1/7 doğru tahmin yapılırken, EDA sonrasında 3/7 örnek doğru sınıflandırılmıştır. Blocker sınıfının recall değeri 0,1429'dan 0,4286'ya, F1-skoru ise 0,1667'den 0,3333'e yükselmiştir.

Diğer sınıflarda belirgin bir performans kaybı gözlemlenmemiştir. Major sınıfında sınırlı bir düşüş görülse de Critical ve Minor sınıflarında dengeli performans korunmuştur. Bu sonuçlar, EDA uygulamasının çoğunluk sınıflarda ciddi bir performans kaybına yol açmadan azınlık sınıfın öğrenilebilirliğini artırdığını göstermektedir.

Genel olarak değerlendirildiğinde, DT modeli dengesiz veri setinde genel doğruluk açısından kabul edilebilir bir performans sergilemesine rağmen az temsil edilen Blocker sınıfında düşük duyarlılık göstermiştir. EDA sonrası uygulanan hedefli veri artırma stratejisi, Blocker sınıfı performansını anlamlı biçimde iyileştirmiş ve sınıflar arası performans dengesini güçlendirmiştir.

4.1.5 XGBoost modeli sonuçları

Bu bölümde XGBoost modelinin dengesiz veri seti üzerindeki performansı ve EDA uygulaması sonrasında elde edilen sonuçlar karşılaştırmalı olarak değerlendirilmiştir.

Dengesiz veri kümesi ile eğitilen XGBoost modelinin genel doğruluk oranı 0,7941, macro-F1 skoru 0,7298 ve weighted-F1 skoru 0,7917 olarak hesaplanmıştır. Modelin tüm sınıflarda dengeli bir performans sergilediği görülmektedir. Minor sınıfında F1-skoru 0,8519, Critical sınıfında 0,7957 ve Major sınıfında 0,7717 olarak ölçülmüştür.

Azınlık sınıf olan Blocker için model, test kümesindeki 7 örneğin 3'ünü doğru sınıflandırmıştır ve precision=0,6000, recall=0,4286 ve F1-skoru=0,5000 olarak hesaplanmıştır. Bu sonuç, sınıf dengesizliğine rağmen modelin Blocker sınıfına ait örüntüleri belirli ölçüde öğrenebildiğini göstermektedir.

EDA uygulanarak model yeniden eğitildiğinde genel doğruluk oranının 0,8118'e yükseldiği, macro-F1 skorunun 0,7298'den 0,7700'a, weighted-F1 skorunun ise 0,7917'den 0,8107'ye çıktığı görülmektedir. Macro-F1 metriğindeki artış, sınıflar arası performans dengesinin güçlendiğini göstermektedir.

En belirgin iyileşme Blocker sınıfında gözlemlenmiştir. EDA öncesinde 3/7 doğru tahmin yapılırken, EDA sonrasında 4/7 örnek doğru sınıflandırılmıştır. Blocker sınıfının recall değeri 0,4286'dan 0,5714'e, F1-skoru ise 0,5000'dan 0,6154'e yükselmiştir. Precision değerindeki artış da modelin Blocker tahminlerini daha güvenilir biçimde gerçekleştirdiğini göstermektedir.

Diğer sınıflarda performans korunmuş veya sınırlı iyileşmeler gözlemlenmiştir. Bu durum, veri artırma yaklaşımının çoğunluk sınıflarda olumsuz bir etki yaratmadan azınlık sınıf performansını artırdığını göstermektedir.

Genel olarak değerlendirildiğinde, XGBoost modeli dengesiz veri setinde dahi yüksek ve dengeli bir performans sergilemiştir. EDA tabanlı hedefli veri artırma yaklaşımı ile kullanıldığında ise özellikle azınlık sınıf olan Blocker üzerinde anlamlı bir iyileşme sağlamıştır. Bu bulgular, gradyan artırma temelli topluluk yöntemlerinin ve hedefli veri

artırma stratejilerinin sınıf dengesizliği içeren metin sınıflandırma problemlerinde etkili bir yaklaşım sunduğunu göstermektedir.

4.1.6 LightGBM modeli sonuçları

Bu bölümde LightGBM modelinin dengesiz veri seti üzerindeki performansı ve EDA uygulaması sonrasında elde edilen sonuçlar karşılaştırmalı olarak değerlendirilmiştir.

Dengesiz veri kümesi ile eğitilen LightGBM modelinin genel doğruluk oranı 0,8471, macro-F1 skoru 0,7324 ve weighted-F1 skoru 0,8436 olarak hesaplanmıştır. Modelin Critical, Major ve Minor sınıflarında oldukça yüksek ve dengeli bir performans sergilediği görülmektedir. Özellikle Minor sınıfında recall değerinin 0,9245 olması, modelin bu sınıfa ait örneklerin büyük bölümünü doğru biçimde yakalayabildiğini göstermektedir.

Azınlık sınıf olan Blocker için model, test kümesindeki 7 örneğin 2'sini doğru sınıflandırmıştır ve F1-skoru 0,3333 olarak hesaplanmıştır. Bu sonuç, sınıf dengesizliğinin azınlık sınıf performansını sınırladığını göstermektedir.

EDA uygulanarak model yeniden eğitildiğinde genel doğruluk oranının 0,8471 seviyesinde korunduğu, macro-F1 skorunun 0,7324'ten 0,7462'ye yükseldiği ve weighted-F1 skorunun 0,8436'dan 0,8413'e sınırlı bir değişim gösterdiği görülmektedir. Accuracy değerinin korunması, veri artırma işleminin genel performansı olumsuz etkilemediğini göstermektedir.

Blocker sınıfında recall değeri sabit kalmakla birlikte precision değerinin 0,4000'dan 0,6667'ye yükselmesi, modelin Blocker tahminlerinde daha isabetli hale geldiğini göstermektedir. F1-skorunun 0,3333'ten 0,4000'a yükselmesi, azınlık sınıf performansında sınırlı ancak anlamlı bir iyileşmeye işaret etmektedir.

Diğer sınıflarda performans büyük ölçüde korunmuş ve özellikle Critical sınıfında recall değerinde artış gözlemlenmiştir. Bu durum, EDA uygulamasının çoğunluk sınıflar üzerinde olumsuz bir etki yaratmadan sınıflar arası performans dengesini güçlendirdiğini göstermektedir.

Genel olarak değerlendirildiğinde, LightGBM modeli dengesiz veri setinde yüksek doğruluk oranı ve güçlü sınıf performansı sergilemiştir. EDA sonrası ise sınıflar arası performans dengesini daha da iyileştirmiştir. Özellikle macro-F1 metriğindeki artış, modelin tüm sınıflar üzerinde daha dengeli bir öğrenme gerçekleştirdiğini ortaya koymaktadır.

4.1.7 Modellerin karşılaştırmalı değerlendirilmesi

Yukarıdaki alt bölümlerde, her bir modelin dengesiz veri seti üzerindeki performansı ve EDA uygulanması sonrası elde edilen sonuçları ayrı ayrı değerlendirilmiştir. Ancak model seçiminde yalnızca tek bir modele ait bulgular değil, tüm modellerin ortak bir çerçevede karşılaştırılması gerekmektedir. Bu nedenle bu bölümün devamında, EDA öncesi ve sonrası senaryolarda elde edilen sonuçlar, genel performans metrikleri ve azınlık sınıf başarısı açısından karşılaştırmalı olarak sunulmuştur.

Öncelikle, EDA uygulanmadan elde edilen genel performans metrikleri Tablo 21'de sunulmakta ve modellerin sınıf dengesizliği karşısındaki davranışları karşılaştırılmaktadır.

Tablo 21. EDAsız Veri Seti Üzerinde Modellerin Genel Performans Karşılaştırması

Model	Accuracy	Macro F1	Weighted F1
SVM	0.8176	0.6251	0.7997
NB	0.7765	0.5924	0.7587
RF	0.8118	0.6208	0.7938
DT	0.7529	0.6244	0.7492
XGBoost	0.7941	0.7298	0.7917
LightGBM	0.8471	0.7324	0.8436

Tablo incelendiğinde, dengesiz veri seti üzerinde eğitilen modellerin Accuracy değerlerinin birbirine görece yakın olduğu görülmektedir. Ancak Macro F1 değerleri dikkate alındığında modeller arasında belirgin performans farkları ortaya çıkmaktadır. Özellikle LightGBM (0.7324) ve XGBoost (0.7298) modellerinin Macro F1 açısından diğer modellere kıyasla daha başarılı olduğu görülürken, NB modelinin sınıf bazlı dengeyi sağlamada daha zayıf kaldığı anlaşılmaktadır. Bu durum, sınıf dengesizliği

bulunan veri setlerinde yalnızca doğruluk oranının model performansını değerlendirmek için yeterli olmadığını açıkça göstermektedir.

Weighted F1 değerleri incelendiğinde ise modellerin çoğunun görece yüksek sonuçlar ürettiği görülmektedir. Bu durum, veri setinde örnek sayısı fazla olan sınıfların model performansını baskıladığını ve az örnekli sınıflardaki başarısızlığın bu metrikte yeterince yansımadığını göstermektedir. Özellikle SVM ve RF modellerinin Accuracy ve Weighted F1 değerleri yüksek olmasına rağmen Macro F1 skorlarının daha düşük kalması, modellerin az örnekli hata sınıflarını öğrenmekte zorlandığını ortaya koymaktadır.

Bu bulgular, dengesiz veri setlerinde model başarımının değerlendirilmesinde Macro F1 metriğinin daha ayırt edici bir ölçüt olduğunu ve sınıf bazlı performansın göz önünde bulundurulmasının gerekliliğini ortaya koymaktadır.

EDA uygulanmadan önce elde edilen sonuçların ardından, veri setine EDA tabanlı veri artırma ve dengeleme işlemi uygulanmış ve aynı modeller yeniden eğitilmiştir. Bu aşamada amaç, sınıf dengesizliğinin azaltılması ile model performansında meydana gelen değişimin gözlemlenmesidir. Tablo 22'de, EDA sonrası veri seti üzerinde eğitilen modellerin genel performans metrikleri sunulmakta ve modeller arası karşılaştırma yapılmaktadır.

Tablo 22. EDA Uygulanmış Veri Seti Üzerinde Modellerin Genel Performans Karşılaştırması

Model	**Accuracy**	**Macro F1**	**Weighted F1**
SVM	0.8294	0.7662	0.8266
NB	0.7765	0.6965	0.7762
RF	0.8118	0.7230	0.8121
DT	0.7529	0.6687	0.7585
XGBoost	0.8118	0.7700	0.8107
LightGBM	0.8471	0.7462	0.8413

Tablo 22 incelendiğinde, EDA uygulanmış veri seti üzerinde modellerin Macro F1 değerlerinde belirgin bir artış olduğu görülmektedir. Bu durum, veri artırma ve dengeleme işleminin özellikle sınıf dengesizliğinin etkisini azaltarak modellerin sınıflar

arasında daha dengeli öğrenme yapmasına katkı sağladığını göstermektedir. Macro F1 açısından en yüksek performansın XGBoost modelinde (0.7700) elde edildiği, bunu SVM (0.7662) ve LightGBM (0.7462) modellerinin izlediği görülmektedir. Buna karşılık DT modeli Macro F1 değerinde (0.6687) diğer modellere göre daha düşük bir performans sergilemiştir.

Genel doğruluk (Accuracy) değerleri açısından LightGBM modelinin 0.8471 ile en yüksek değere ulaştığı görülmekle birlikte, sınıf dengesizliğinin etkisini daha iyi yansıtan Macro F1 metriğinde LightGBM'nin XGBoost ve SVM'nin gerisinde kaldığı dikkat çekmektedir. Weighted F1 değerleri incelendiğinde ise en yüksek sonucun LightGBM modelinde (0.8413) elde edildiği, ardından SVM (0.8266) ve RF (0.8121) modellerinin geldiği görülmektedir. Bu bulgular, EDA sonrasında modellerin genel performanslarının arttığını ancak model seçiminde yalnızca Accuracy veya Weighted F1 değerlerine değil, sınıflar arası dengeyi daha iyi temsil eden Macro F1 değerine de öncelik verilmesi gerektiğini ortaya koymaktadır.

EDA uygulanmadan önce ve uygulandıktan sonra elde edilen performans sonuçlarının daha açık biçimde karşılaştırılabilmesi amacıyla Tablo 23'te hazırlanmıştır. Bu tabloda, her model için Accuracy, Macro F1 ve Weighted F1 değerlerinin EDA öncesi ve sonrası durumları birlikte sunulmaktadır. Böylece veri artırma sürecinin model performansı üzerindeki etkisi doğrudan gözlemlenebilmektedir.

Tablo 23. EDA Öncesi ve Sonrası Model Performanslarının Karşılaştırılması

Model	Accuracy(Önce)	Accuracy(Sonra)	Macro F1(Önce)	Macro F1(Sonra)	Weighted F1(Önce)	Weighted F1(Sonra)
SVM	0.8176	0.8294	0.6251	0.7662	0.7997	0.8266
NB	0.7765	0.7765	0.5924	0.6965	0.7587	0.7762
RF	0.8118	0.8118	0.6208	0.7230	0.7938	0.8121
DT	0.7529	0.7529	0.6244	0.6687	0.7492	0.7585
XGBoost	0.7941	0.8118	0.7298	0.7700	0.7917	0.8107
LightGBM	0.8471	0.8471	0.7324	0.7462	0.8436	0.8413

Tablo 23 incelendiğinde, EDA uygulanmasının özellikle Macro F1 değerleri üzerinde belirgin ve tutarlı bir iyileşme sağladığı görülmektedir. Tüm modellerde Macro F1 skorlarının artmış olması, veri artırma ve dengeleme sürecinin sınıflar arası öğrenme dengesini olumlu yönde etkilediğini göstermektedir. Bu artış en belirgin biçimde SVM modelinde gözlemlenmiş olup Macro F1 değeri 0.6251'den 0.7662'ye yükselmiştir. Benzer şekilde NB, RF, DT ve XGBoost modellerinde de Macro F1 değerlerinde dikkat çekici artışlar meydana gelmiştir. LightGBM modelinde ise başlangıçta da yüksek olan Macro F1 değeri EDA sonrası daha da artarak 0.7462 seviyesine ulaşmıştır.

Accuracy değerleri incelendiğinde ise bazı modellerde belirgin bir değişim gözlenmezken, SVM ve XGBoost modellerinde artış olduğu görülmektedir. Bu durum, veri dengeleme işleminin genel doğruluk oranına her zaman doğrudan yansımadığını ancak sınıf bazlı öğrenme performansını artırdığını göstermektedir. Weighted F1 değerleri de Macro F1'e benzer şekilde çoğu modelde artış göstermiştir. Özellikle SVM, RF ve XGBoost modellerinde Weighted F1 skorlarının yükselmesi, modellerin hem çoğunluk hem de azınlık sınıflarda daha dengeli tahminler ürettiğini ortaya koymaktadır.

Bu bulgular, EDA'nın model başarımına olan katkısının en doğru şekilde Macro F1 metriği üzerinden gözlemlenebildiğini ve veri dengesizliği probleminin giderilmesinin sınıf bazlı performans üzerinde doğrudan etkili olduğunu açıkça göstermektedir.

Genel performans metrikleri EDA'nın modellere olan olumlu etkisini ortaya koymakla birlikte, veri artırma sürecinin en belirgin etkisi az örnekli sınıflarda görülmektedir. Bu durumu daha açık biçimde gösterebilmek amacıyla, Tablo 24'te yalnızca Blocker sınıfına ait F1-skorlarının EDA öncesi ve sonrası değerleri sunulmuştur. Bu tablo, veri dengesizliğinin model öğrenme süreci üzerindeki etkisini ve EDA'nın bu etkiyi nasıl azalttığını doğrudan ortaya koymaktadır.

Tablo 24. Blocker Sınıfı İçin EDA Öncesi ve Sonrası F-1 Skorlarının Karşılaştırılması

Model	**Blocker F1 (Önce)**	**Blocker F1 (Sonra)**
SVM	0.0000	0.5455
NB	0.0000	0.4000
RF	0.0000	0.4000
DT	0.1667	0.3333
XGBoost	0.5000	0.6154
LightGBM	0.3333	0.4000

Tablo 24 incelendiğinde, EDA uygulanmasının Blocker sınıfı üzerindeki etkisinin son derece belirgin olduğu görülmektedir. EDA öncesinde SVM, NB ve RF modellerinin bu sınıfa ait hiçbir örneği doğru şekilde sınıflandıramadığı ve F1-skorlarının 0.0000 olduğu görülürken, EDA sonrasında bu değerlerin sırasıyla 0.5455, 0.4000 ve 0.4000 seviyelerine yükseldiği dikkat çekmektedir. Benzer şekilde, DT modelinin F1-skoru 0.1667'den 0.3333'e, XGBoost modelinin 0.5000'dan 0.6154'e ve LightGBM modelinin ise 0.3333'ten 0.4000'e yükselmiştir.

Bu bulgular, veri dengesizliğinin en yoğun hissedildiği az örnekli sınıflarda modellerin öğrenme kapasitesinin EDA uygulanmadan önce son derece sınırlı olduğunu, ancak veri artırma ve dengeleme işlemi sonrasında modellerin bu sınıfa ait örüntüleri öğrenebilir hale geldiğini açıkça göstermektedir. Özellikle EDA öncesinde tamamen başarısız olan

modellerin EDA sonrasında anlamlı F1-skorlarına ulaşması, veri artırma yönteminin sınıf bazlı performans üzerindeki doğrudan ve güçlü etkisini ortaya koymaktadır.

EDA öncesi ve sonrası elde edilen performans sonuçları birlikte değerlendirildiğinde, veri dengesizliğinin model seçiminde belirleyici bir rol oynadığı görülmektedir. Bu nedenle, EDA uygulanmayan ve uygulanan senaryolar için en uygun model ayrı ayrı belirlenmiştir. Tablo 25'te, her iki durum için modellerin kritik performans göstergeleri birlikte sunularak karşılaştırmalı bir değerlendirme yapılmıştır.

Tablo 25. EDA Öncesi ve Sonrası En Başarılı Modellerin Karşılaştırılması

Model	**Macro F1 (Önce)**	**Blocker F1 (Önce)**	**Macro F1 (Sonra)**	**Blocker F1 (Sonra)**	**EDA Öncesi Değerlendirme**	**EDA Sonrası Değerlendirme**
SVM	0.6251	0.0000	0.7662	0.5455	Çoğunluk sınıflarda başarılı, Blocker sınıfını tamamen öğrenemedi.	Sınıflar arası denge sağlandı, Blocker başarımı belirgin şekilde arttı.
NB	0.5924	0.0000	0.6965	0.4000	Sınıf dengesizliğinden en fazla etkilenen model.	Dengeli öğrenme sağladı ancak genel başarı sınırlı kaldı.
RF	0.6208	0.0000	0.7230	0.4000	Az örnekli sınıflarda zayıf, çoğunluk sınıflarda güçlü.	Daha dengeli sonuçlar üretti ancak üst modellerin gerisinde kaldı.
DT	0.6244	0.1667	0.6687	0.3333	Dengesiz veri karşısında kararsız ve düşük performans.	Kısmi iyileşme sağladı ancak genel performans düşük kaldı.
XGBoost	0.7298	0.5000	0.7700	0.6154	Sınıflar arası en dengeli öğrenmeyi	Hem Macro F1 hem Blocker başarımı

					sağlayan model.	açısından en başarılı model.
LightGBM	0.7324	0.3333	0.7462	0.4000	Genel metriklerde yüksek ancak sınıf dengesinde sınırlı.	Yüksek genel performansını koruyarak dengeli sonuçlar üretti.

Tablo 25 incelendiğinde, EDA uygulanmadan önce ve uygulandıktan sonra modellerin performanslarında belirgin farklılıklar oluştuğu görülmektedir. EDA öncesinde Macro F1 ve Blocker F1 değerleri dikkate alındığında LightGBM ve XGBoost modellerinin diğer modellere göre daha dengeli sonuçlar ürettiği anlaşılmaktadır. Özellikle LightGBM modelinin EDA öncesi en yüksek Macro F1 değerine sahip olduğu görülmektedir.

EDA uygulandıktan sonra ise tüm modellerde Macro F1 ve Blocker F1 değerlerinde artış meydana gelmiş, ancak bu artışın en belirgin şekilde XGBoost modelinde gerçekleştiği dikkat çekmektedir. XGBoost modeli, EDA sonrası hem Macro F1 hem de Blocker F1 değerleri açısından en yüksek performansı sergileyerek sınıflar arası en dengeli öğrenmeyi sağlayan model olmuştur. Bu tablo, EDA uygulanmasının model davranışı ve model seçimi üzerinde doğrudan etkili olduğunu açık biçimde göstermektedir.

4.2 Tartışma

Bu çalışmada elde edilen bulgular, yazılım hata kayıtlarının sınıflandırılmasında veri dengesizliğinin model performansı üzerinde belirleyici bir etkiye sahip olduğunu açık biçimde ortaya koymaktadır. Özellikle Macro F1 ve Blocker sınıfına ait F1 skorları birlikte değerlendirildiğinde, modellerin gerçek sınıflandırma kapasitesinin yalnızca Accuracy metriği üzerinden değerlendirilemeyeceği anlaşılmaktadır. Bu bulgu, Gomes ve arkadaşları (2019) ile Brown ve arkadaşları (2022) tarafından vurgulanan veri dengesizliği problemleriyle örtüşmektedir. Brown ve arkadaşları (2022), blocker bug tahmini çalışmalarında veri setlerinin önemli ölçüde dengesiz olduğunu ve bu nedenle Accuracy metriğinin tek başına yeterli olmadığını belirtmiştir. Bu tez çalışmasında da benzer şekilde değerlendirme yalnızca doğruluk metriğiyle sınırlandırılmamış, Macro F1, Weighted F1 ve sınıf bazlı F1 skorları birlikte yorumlanmıştır.

EDA uygulanmadan önce elde edilen sonuçlar incelendiğinde, modellerin Accuracy değerlerinin görece yüksek olmasına rağmen Macro F1 skorlarının belirgin şekilde daha düşük olduğu görülmektedir. Bu durum, modellerin çoğunluk sınıflarda başarılı tahminler üretirken az örnekli sınıflarda ciddi başarım kaybı yaşadığını göstermektedir. Özellikle SVM, NB ve RF modellerinde Blocker sınıfı için F1 skorunun 0.0000 olması, bu modellerin kritik hata sınıfını tamamen göz ardı ettiğini ortaya koymaktadır.

Bu sonuç, dengesiz veri setlerinde doğruluk oranının yanıltıcı olabileceğini ve modelin gerçek genelleme kapasitesini yansıtmadığını desteklemektedir. Weighted F1 değerlerinin görece yüksek kalması da çoğunluk sınıfların performans üzerindeki baskın etkisini doğrulamaktadır. Buna karşılık XGBoost ve LightGBM modellerinin EDAsız senaryoda dahi daha yüksek Macro F1 değerlerine ulaşması, boosting tabanlı yöntemlerin karmaşık örüntüleri ve sınıflar arası sınırları daha etkin modelleyebildiğini göstermektedir. Bu çalışmada elde edilen XGBoost sonuçları, literatürde topluluk öğrenmesi tabanlı yöntemlerin hata tahmini ve hata kritiklik sınıflandırması problemlerinde etkili olduğunu gösteren çalışmalarla genel olarak uyumludur. Cheng ve arkadaşları (2020), açık kaynak yazılım projelerinde blocker bug tahmini için XGBoost tabanlı XGBlocker yaklaşımını önermiş ve bu yöntemin mevcut ELBlocker yaklaşımına göre daha başarılı sonuçlar verdiğini göstermiştir. Bu yönüyle, bu tez çalışmasında XGBoost modelinin yüksek performans göstermesi, blocker bug tahmininde XGBoost'un etkili olabileceğini gösteren literatür bulgularıyla da örtüşmektedir. Özellikle AdaBoost ve Bagging gibi yöntemlerin başarılı sonuçlar verdiği çalışmalar, topluluk öğrenmesi yaklaşımlarının yazılım hata kayıtlarından anlamlı örüntüler çıkarma konusunda etkili olabileceğini göstermektedir. Bu tez çalışmasında XGBoost modelinin yüksek performans göstermesi de bu literatür bulgularını desteklemektedir. Ancak veri seti, sınıf dağılımı ve değerlendirme metrikleri çalışmalar arasında farklılık gösterdiğinden sonuçlar doğrudan birebir karşılaştırılmamış, genel eğilimler üzerinden yorumlanmıştır.

EDA uygulanması sonrasında tüm modellerde Macro F1 değerlerinin artmış olması, veri artırma ve dengeleme sürecinin sınıf bazlı öğrenmeyi iyileştirdiğini açık biçimde göstermektedir. Özellikle SVM modelinde Macro F1 skorunun 0.6251'den 0.7662'ye yükselmesi, lineer ayırıcıların yeterli temsil gücüne sahip olduğu ancak az örnekli sınıflarda veri eksikliği nedeniyle sınır çizmekte zorlandığı şeklinde yorumlanabilir.

NB modelinde de anlamlı bir iyileşme gözlemlenmiş olmakla birlikte, modelin bağımsızlık varsayımı nedeniyle karmaşık bağlamsal ilişkileri yakalamakta sınırlı kaldığı görülmektedir. DT modelinde ise kısmi bir iyileşme sağlanmasına rağmen genel performansın diğer modellere göre düşük kalması, tek ağaç yapısının aşırı öğrenmeye yatkınlığı ve genelleme kapasitesinin sınırlı olmasıyla açıklanabilir.

RF modelinde Macro F1 artışı daha dengeli bir öğrenme sürecine işaret etmekle birlikte, boosting yöntemlerine kıyasla daha sınırlı kalmıştır. Bu durum, bagging yaklaşımının varyansı azaltmada başarılı olmasına rağmen, az örnekli sınıfların sınırlarını hassas biçimde modellemede boosting kadar etkili olmayabileceğini göstermektedir.

Çalışmanın en önemli çıktılarından biri, Blocker sınıfındaki iyileşmedir. EDAsız senaryoda bazı modellerin bu sınıfı tamamen öğrenememesi, yazılım projelerinde kritik hataların tespit edilememesi anlamına gelebilecek ciddi bir risk oluşturmaktadır. EDA sonrasında tüm modellerde Blocker F1 skorlarının artması, veri dengesizliği probleminin doğrudan giderilmesinin kritik hata tespiti açısından hayati olduğunu göstermektedir.

Özellikle XGBoost modelinin Blocker F1 skorunu 0.5000'dan 0.6154'e yükseltmesi, modelin kritik hata örüntülerini daha sağlam biçimde temsil edebildiğini göstermektedir. Bu bulgu, Cheng ve arkadaşları (2020) tarafından önerilen XGBoost tabanlı XGBlocker yaklaşımının blocker bug tahmininde başarılı sonuçlar vermesiyle de paralellik göstermektedir. Bu sonuç, pratik yazılım geliştirme ortamlarında kritik hataların erken tespiti açısından önemli bir avantaj sağlamaktadır.

Çalışma kapsamında belirlenen araştırma sorularına elde edilen bulgular doğrultusunda yanıt verilmiştir. Sonuçlar, yazılım test sürecinde oluşan hata kayıtlarının makine öğrenmesi yöntemleriyle otomatik olarak kritiklik seviyelerine göre tahmin edilebildiğini göstermektedir. Modeller arası karşılaştırmada özellikle XGBoost ve LightGBM modellerinin öne çıktığı, XGBoost modelinin ise genel performans ve Blocker sınıfındaki başarımı açısından daha dengeli sonuçlar ürettiği görülmektedir. Bununla birlikte, sınıf dengesizliğinin özellikle azınlıkta kalan kritik hata sınıflarının tahmin başarısını olumsuz etkilediği belirlenmiştir. EDA uygulanması sonrasında Macro F1 ve sınıf bazlı F1 skorlarında görülen iyileşmeler, veri artırma yaklaşımının model performansına olumlu katkı sağladığını ortaya koymaktadır. Bu nedenle çalışmada elde edilen bulgular hem

araştırma sorularına yanıt sunmakta hem de hata kritiklik seviyesi tahmininde veri dengesi, model seçimi ve sınıf bazlı değerlendirme metriklerinin birlikte ele alınması gerektiğini göstermektedir.

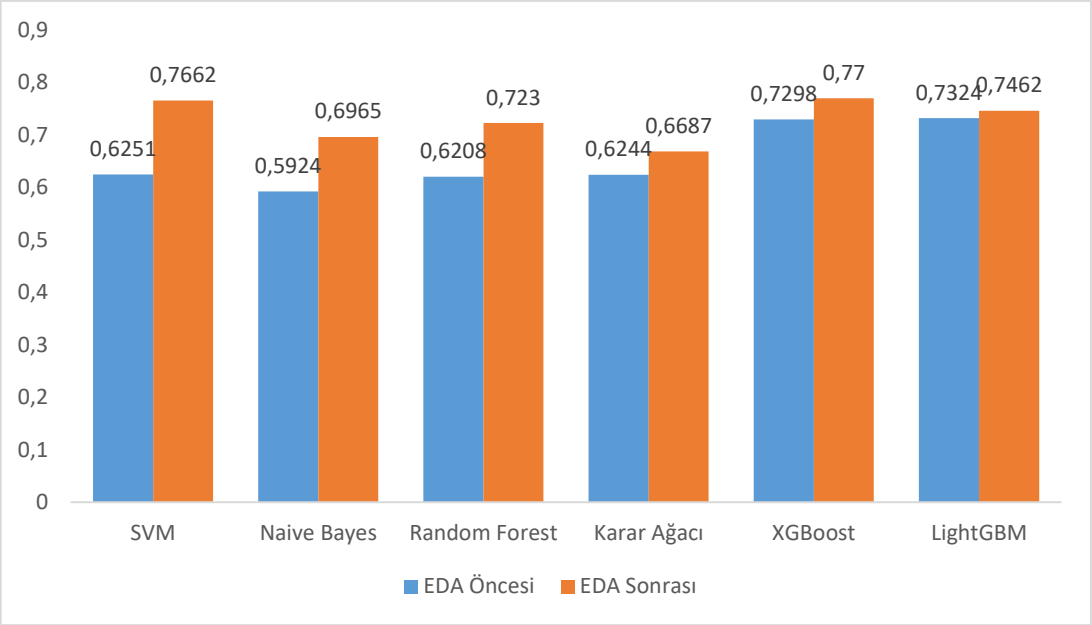
5. SONUÇ ve ÖNERİLER

Sonuçlar:

Bu çalışmada, yazılım hata kayıtlarının makine öğrenmesi yöntemleri ile kritiklik seviyelerine göre sınıflandırılması problemi ele alınmış ve veri dengesizliğinin model performansı üzerindeki etkisi sistematik olarak incelenmiştir. Elde edilen bulgular, dengesiz veri setlerinde yalnızca doğruluk (Accuracy) metriğine dayalı model değerlendirmesinin yanıltıcı olabileceğini açık biçimde ortaya koymuştur. Özellikle Macro F1 ve az örnekli sınıflara ait F1 skorları dikkate alındığında, modellerin gerçek sınıflandırma kapasiteleri daha net biçimde anlaşılmaktadır.

EDA uygulanmadan önce elde edilen sonuçlar, çoğu modelin çoğunluk sınıflarda yüksek doğruluk üretmesine rağmen az örnekli ve kritik öneme sahip Blocker sınıfını öğrenmekte ciddi şekilde zorlandığını göstermiştir. Bazı modellerin bu sınıf için F1 skorunun sıfır olması, veri dengesizliğinin kritik hata tespiti açısından önemli bir risk oluşturduğunu ortaya koymaktadır. Bu durum, yazılım geliştirme süreçlerinde kritik hataların gözden kaçmasına neden olabilecek bir zayıflığa işaret etmektedir.

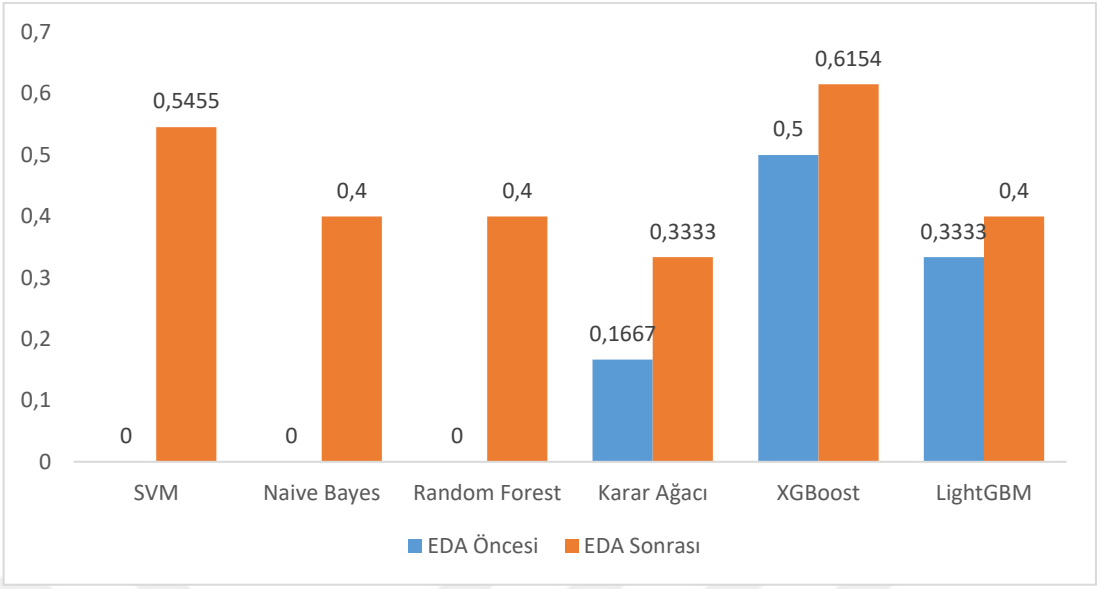
EDA tabanlı veri artırma ve dengeleme işlemi sonrasında ise tüm modellerde özellikle Macro F1 değerlerinde anlamlı artışlar gözlemlenmiştir. Bu iyileşme, veri dengesizliğinin azaltılmasının sınıflar arası öğrenme dengesini güçlendirdiğini ve az örnekli sınıfların temsil gücünü artırdığını göstermektedir. Çalışmada kullanılan modellerin EDA öncesi ve sonrası Macro F1 performanslarının genel görünümünü sunmak amacıyla elde edilen sonuçlar Şekil 15'te görselleştirilmiştir.



Şekil 15. Modellerin EDA Öncesi ve Sonrası Macro F1 Performans Karşılaştırması

Şekil 15 incelendiğinde, EDA uygulamasının tüm modellerde Macro F1 performansını artırdığı görülmektedir. En belirgin artış SVM modelinde ortaya çıkarken, EDA sonrasında en yüksek Macro F1 değerine XGBoost modeli ulaşmıştır. Bu durum, veri artırma ve dengeleme işleminin sınıf dengesizliği bulunan hata kritiklik seviyesi tahmin probleminde modellerin sınıflar arası öğrenme başarısını olumlu yönde etkilediğini göstermektedir.

EDA uygulamasının az örnekli ve kritik öneme sahip Blocker sınıfı üzerindeki etkisini daha açık biçimde göstermek amacıyla Blocker F1 değerleri Şekil 16'da karşılaştırmalı olarak sunulmuştur.



Şekil 16. Modellerin EDA Öncesi ve Sonrası Blocker F1 Performans Karşılaştırması

Şekil 16 incelendiğinde, EDA uygulamasının Blocker sınıfı üzerinde belirgin bir iyileşme sağladığı görülmektedir. EDA öncesinde SVM, NB ve RF modellerinin Blocker sınıfında F1 skorunun 0,0000 olması, sınıf dengesizliğinin kritik hata sınıflarının öğrenilmesini zorlaştırdığını göstermektedir. EDA sonrasında ise tüm modellerde Blocker F1 değerlerinin artması, veri artırma işleminin az örnekli kritik sınıfların öğrenilmesine katkı sağladığını ortaya koymaktadır. En yüksek Blocker F1 değeri EDA sonrasında XGBoost modelinde elde edilmiştir.

Model bazlı değerlendirme sonucunda, EDA uygulanmış senaryoda XGBoost modelinin hem Macro F1 hem de Blocker F1 açısından en dengeli ve yüksek performansı sergilediği belirlenmiştir. LightGBM modeli genel doğruluk ve Weighted F1 açısından güçlü sonuçlar üretmekle birlikte, sınıflar arası dengeyi daha iyi yansıtan Macro F1 ve Blocker F1 değerleri açısından XGBoost modelinin gerisinde kalmıştır. SVM modeli ise EDA sonrasında belirgin bir performans artışı göstermiştir. Bu bulgular doğrultusunda, yazılım hata kritiklik seviyesi tahmin problemlerinde özellikle veri artırma uygulanmış senaryolarda XGBoost modelinin tercih edilebilir bir yaklaşım olduğu sonucuna ulaşılmıştır.

Elde edilen sonuçlar, yazılım kalite süreçlerinde veri odaklı karar destek mekanizmalarının kullanılmasının hata yönetiminde önemli katkılar sağlayabileceğini göstermektedir. Özellikle sınıf dengesizliği bulunan veri setlerinde, yalnızca genel

doğruluk oranına odaklanmanın yeterli olmadığı; Macro F1 ve sınıf bazlı F1 skorlarının birlikte değerlendirilmesinin modellerin gerçek başarısını daha doğru biçimde ortaya koyduğu görülmektedir.

Öneriler:

Bu çalışma kapsamında elde edilen sonuçlar doğrultusunda, hata kritiklik seviyesi tahmini üzerine yapılacak akademik çalışmalarda model başarısının yalnızca Accuracy metriği ile değerlendirilmemesi önerilmektedir. Özellikle dengesiz veri setlerinde Macro F1, Weighted F1 ve sınıf bazlı F1 skorlarının birlikte yorumlanması daha doğru sonuçlar sağlayacaktır. Bunun yanında, Blocker gibi az örnekli ancak yazılım süreci açısından yüksek etkiye sahip sınıfların ayrı olarak değerlendirilmesi önemlidir.

Veri dengesizliği bulunan hata kayıtlarında EDA gibi veri artırma yöntemlerinin kullanılması, özellikle az örnekli kritik sınıfların öğrenilmesine katkı sağlayabilir. Bu nedenle gelecek çalışmalarda, veri artırma ve dengeleme yöntemlerinin farklı modeller üzerindeki etkisinin karşılaştırmalı olarak incelenmesi önerilmektedir. Bu çalışma kapsamında EDA sonrasında XGBoost modelinin öne çıkması, bu modelin benzer hata kritiklik seviyesi tahmini problemlerinde dikkate alınabilecek güçlü bir alternatif olduğunu göstermektedir.

Gelecek çalışmalarda, bu tez kapsamında kullanılan modellerin farklı sektörlerden ve farklı yazılım projelerinden elde edilecek daha geniş hata kayıtları üzerinde yeniden eğitilmesi ve performanslarının karşılaştırılması önerilmektedir. Böylece geliştirilen yaklaşımın yalnızca e-ticaret mobil uygulaması özelinde değil, farklı yazılım test süreçlerinde de ne ölçüde genellenebilir olduğu değerlendirilebilir. Ayrıca farklı makine öğrenmesi, derin öğrenme ve transformer tabanlı modellerin aynı veri yapıları üzerinde karşılaştırılması, hata kritiklik seviyesi tahmini alanında daha kapsamlı sonuçlar elde edilmesine katkı sağlayabilir.

Uygulama açısından değerlendirildiğinde, geliştirilen yaklaşımın Jira gibi hata takip sistemlerine eklenti veya harici servis olarak entegre edilmesi önerilmektedir. Böyle bir entegrasyonda, hata kaydı oluşturulduğunda başlık, açıklama, hata adımları ve ilgili modül bilgileri model tarafından analiz edilerek önerilen kritiklik seviyesi otomatik

olarak sunulabilir. Bu yapı, test mühendisinin kararını tamamen ortadan kaldıran bir sistem olarak değil, insan denetimli bir karar destek mekanizması olarak kullanılmalıdır.

Bu tür bir entegrasyon sayesinde hata önceliklendirme süreci daha hızlı ve tutarlı hale getirilebilir. Test uzmanı model tarafından önerilen kritiklik seviyesini kabul edebilir, değiştirebilir veya manuel olarak farklı bir seviye atayabilir. Bu geri bildirimler ilerleyen süreçte modelin yeniden eğitilmesinde kullanılabilir. Böylece sistem, zaman içinde kurumun hata yönetim yapısına uyum sağlayan sürdürülebilir bir karar destek aracına dönüştürülebilir.

Genel olarak değerlendirildiğinde, bu tez çalışmasında elde edilen bulgular, veri dengesizliği dikkate alınarak geliştirilen makine öğrenmesi modellerinin yazılım test ve hata yönetimi süreçlerinde karar destek aracı olarak kullanılabileceğini ortaya koymaktadır.

KAYNAKÇA

Agarwal, A., Agarwal, A., Verma, D., Tiwari, D., ve Pandey, R. (2023). A review on software development life cycle. International Journal of Scientific Research in Computer Science. *Engineering and Information Technology*, 384-388.

Alenezi, M., ve Banitaan, S. (2013). Bug Reports Prioritization: Which Features and Classifier to Use? *2013 12th International Conference on Machine Learning and Applications* (s. 112-116). Miami, FL, USA: IEEE.

Alharbi, S., Alsaeedi, A., ve Yafooz, W. (2026). Bug Severity Prediction: A Comprehensive Review and Preliminary Results using Pre-trained Transformers. *Proceedings of the 5th International Conference on Communication, Computing and Electronics Systems* (s. 2073-2078). Coimbatore, India: IEEE.

Arslan, K. (2010). Eğitimde Yapay Zeka ve Uygulamaları. *Batı Anadolu Eğitim Bilimleri Dergisi*, 71-88.

Arslankaya, S., ve Toprak, Ş. (2021). Makine Öğrenmesi ve Derin Öğrenme Algoritmalarını Kullanarak Hisse Senedi Fiyat Tahmini. *International Journal of Engineering Research and Development*, 178-192.

AWS, A. (2025, 6 1). *SDLC (Yazılım Geliştirme Yaşam Döngüsü) nedir?* 2025, 6 1AWS AMAZON: https://aws.amazon.com/tr/what-is/sdlc/ adresinden alındı

Aylak, B., Oral, O., ve Yazıcı, K. (2021). Yapay Zeka ve Makine Öğrenmesi Tekniklerinin Lojistik Sektöründe Kullanımı. *El-Cezerî Fen ve Mühendislik Dergisi*, 74-93.

Aytaç Arpacı, S., ve Kalıpsız, O. (2018). Yazılım Hata Sınıflandırmasında Farklı Naive Bayes Tekniklerin Kıyaslanması. *Niğde Ömer Halisdemir Üniversitesi Mühendislik Bilimleri Dergisi*, 1-13.

Brown, S., Weyori, B., Adekoya, A., Kudjo, P., ve Mensah, S. (2022). Predicting Blocking Bugs with Machine Learning Techniques: A Systematic Review. *(IJACSA) International Journal of Advanced Computer Science and Applications*, 674-683.

Chen, T., ve Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (s. 785-794). Association for Computing Machinery.

Cheng, X., Liu, N., Guo, L., Xu, Z., ve Zhang, T. (2020). Blocking Bug Prediction Based on XGBoost with Enhanced Features. *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)* (s. 902-911). Madrid, Spain: IEEE.

Dao, A.-H., Yue-Lin, Y., ve Cheng-Zen, Y. (2022). Severity Prediction for Bug Reports using Tree-based Ensemble Models: A Comparative Study. *2nd IEEE International Conference on Software Engineering and Artificial Intelligence* (s. 01-05). Xiamen, China: IEEE.

Dora, S. K., ve Dubey, P. (2013). Software Development Life Cycle (SDLC) Analytical Comparison and Survey on Traditional And Agile Methodology. *National Monthly Refereed Journal of Research in Science & Technology*, 22-30.

Eşidir, K. (2025). Makine Öğrenimi Modelleri ile Yetişkin Eğitimi Analizi: Modellerin Karşılaştırmalı Performansı. *Elektronik Sosyal Bilimler Dergisi*, 946-964.

Gencer, C., ve Kayacan, A. (2017). Yazılım Proje Yönetimi: Şelale Modeli ve Çevik Yöntemlerin Karşılaştırılması. *Bilişim Teknolojileri Dergisi*, 335 - 352.

Gomes, L. A., Torres, R. d., ve Côrtes, M. L. (2019). Bug report severity level prediction in open source software: A survey and research opportunities. *Information and Software Technology, 115*, 58-78.

Gupta, A. (2021). Comparative Study of Different SDLC Models. *International Journal for Research in Applied Science and Engineering Technology*, 73-80.

Harman, G. (2021). Destek Vektör Makineleri ve Naive Bayes Sınıflandırma Algoritmalarını Kullanarak Diabetes Mellitus Tahmini. *Avrupa Bilim ve Teknoloji Dergisi*, 7-13.

Hooimeijer, P., ve Weimer, W. (2007). Modeling bug report quality. *In Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering* (s. 34-43). IEEE.

Hossain, M. (2023). Software development life cycle (SDLC) methodologies for information systems project management. *International Journal For Multidisciplinary Research*.

Hufsa, M., ve Javed, M. (2012). An Automated Approach for Software Bug Classification. *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems* (s. 414-419). IEEE.

Islam, M., Khan, F., Alam, S., ve Hasan, M. (2023). Artificial Intelligence in Software Testing: A Systematic Review. *2023 IEEE Region 10 Conference (TENCON)* (s. 524-529). Thailand: IEEE.

Üstüner, M., ve Balık Şanlı, F. (2019). Polarimetric Target Decompositions and Light Gradient Boosting Machine for Crop Classification: A Comparative Evaluation. *ISPRS International Journal of Geo-Information*, 97.

Üstüner, M., Abdikan, S., Bilgin, G., ve Balık Şanlı, F. (2020). Hafif Gradyan Artırma Makineleri ile Tarımsal Ürünlerin Sınıflandırılması. *Türk Uzaktan Algılama ve CBS Dergisi*, 97-105.

Jalbert, N., ve Weimer, W. (2008). Automated Duplicate Detection for Bug Tracking Systems. *International Conference on Dependable Systems & Networks: Anchorage* (s. 24-27). Alaska: IEEE.

Jamil, M. A., Arif, M., Abubakar, N. S., ve Ahmad, A. (2016). Software testing techniques: A literature review. *2016 6th International Conference on Information and Communication Technology for The Muslim World* (s. 177-182). Indonesia: IEEE.

Kavita, S., Raj Shree, P., ve Rajeev, K. (2014). Risk Management Perspective in SDLC. *International Journal of Advanced Research in Computer Science and Software Engineering, 4*(3), 1247-1251.

Kavzoğlu, T., ve Çölkesen, İ. (2010). Karar Ağaçları İle Uydu Görüntülerinin Sınıflandırılması. *Harita Teknolojileri Elektronik Dergisi*, 36-45.

Kayakuş, M., ve Yiğit Açıkgöz, F. (2023). Twitter'da Makine Öğrenmesi Yöntemleriyle Sahte Haber Tespiti. *Abant Sosyal Bilimler Dergisi*, 1017-1027.

Khan, M., Qayoom, A., Nizami, M., Siddiqui, M., Wasi, S., ve Syed, K.-U.-R. (2021). Automated prediction of good dictionary examples (GDEX): A comprehensive experiment with distant supervision, machine learning, and word embedding-based deep learning techniques. *Complexity*.

Kuday, G. (2014). Yazılım Mühendisliği Yöntemleriyle Yazılım Test Süreci. *Yöktez*(Tez no: 389639). [Yüksek Lisans Tezi, İstanbul Haliç Üniversitesi] https://tez.yok.gov.tr/UlusalTezMerkezi/

Kute, S., ve Thorat, S. (2014). A review on various software development life cycle (SDLC) models. *International Journal of Research in Computer and Communication Technology*, 778–779.

Lamkanfi, A., Demeyer, S., Giger, E., ve Goethals, B. (2010). Predicting the severity of a reported bug. *Proceedings - International Conference on Software Engineering*, (s. 1 - 10).

Lan, G., Yan, M., Bojan, C., ve Harshinder, S. (2004). Robust Prediction of Fault-Proneness by Random Forests. *15th International Symposium on Software Reliability Engineering* (s. 417-428). Saint-Malo, France: IEEE.

Mahmood, Z., Bowes, D., Lane, P., ve Hall, T. (2015). What is the impact of imbalance on software defect prediction performance? *In Proceedings of the 11th international conference on predictive models and data analytics in software engineering*, (s. 1-4).

Nitesh, C., Kevin, B., Lawrence, H., ve Philip, K. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 321–357.

Otoom, A., Al-Shdaifat, D., Hammad, M., ve Abdallah, E. (2016). Severity Prediction of Software Bugs. *2016 7th International Conference on Information and Communication Systems (ICICS)* (s. 92-95). Irbid: IEEE.

Örpek, A., Örpek, Z., ve Şahinaslan, Ö. (2026). Classification of Unstructured Text for RPA: A Comparative Machine Learning Analysis. *Boletim Da Sociedade Paranaense De Matemática, 44*(5), 1-12.

Özbay, E., ve Altunbey Özbay, F. (2021). Derin Öğrenme ve Sınıflandırma Yaklaşımları ile BT görüntülerinden Covid-19 Tespiti. *Dicle Üniversitesi Mühendislik Fakültesi Mühendislik Dergisi*, 211-219.

Özdemir, F., ve Öztürk, S. (2025). Kuantum Yazılım Test Teknikleri Ve Araçları Üzerine Bir İnceleme. *Journal of Information Systems and Management Research*, 90-104.

Özlüer Başer, B., Yangın, M., ve Sarıdaş, S. (2021). Makine Öğrenmesi Teknikleriyle Diyabet Hastalığının Sınıflandırılması Yıl 2021, Cilt: 25 Sayı: 1, 112 - 120, 20.04.2021. *Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, 112-120.

Öztürk, İ. (2022). Bankacılık Sektöründe Yazılım Test Süreçlerinin Önemi ve Yöntemleri. *Bilgisayar Bilimleri Ve Teknolojileri Dergisi*, 23-29.

Prabha, L., ve Shivakumar, N. (2020). Software Defect Prediction Using Machine Learning Techniques. *Proceedings of the Fourth International Conference on Trends in Electronics and Informatics* (s. 728-733). Tirunelveli: IEEE.

Pushpalatha, M., ve Mrunalini, M. (2016). Predicting the severity of bug reports using classification algorithms. *International Conference on Circuits, Controls, Communications and Computing (I4C)* (s. 1-4). Bangalore: IEEE.

Rana, A., Rawat, A., ve Bijalwan, A. (2017). Process of finding defects in software testing. *Proceedings of the Second International Conference on Research in Intelligent and Computing in Engineering*, (s. 297-300).

Rossi, B., ve Singha Roy, N. (2017). Cost-Sensitive Strategies for Data Imbalance in Bug Severity Classification: Experimental Results. *43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2017)* (s. 426–429). Gothenburg: IEEE.

Sahu, K., Shree, R., ve Rajeev, K. (2014). Risk management perspective in SDLC. *International Journal of Advanced Research in Computer Science and Software Engineering*.

Sevimli Deniz, S. (2024). İnternet Bağımlılığı Skorlarının Tahmininde Farklı Makine Öğrenme Algoritmalarının Karşılaştırılması. *Süleyman Demirel Üniversitesi Sosyal Bilimler Enstitüsü Dergisi*, 273-286.

Şahinaslan, Ö., Şahinaslan, E., ve Dalyan, H. (2022). Naive Bayes Sınıflandırıcısı Kullanılarak YouTube Verileri Üzerinden Çok Dilli Duygu Analizi. *Bilişim Teknolojileri Dergisi*, 221-229.

Şahinaslan, E., Şahinaslan, Ö., ve Nusret, A. (2023). Yazılım Dağıtım Sürecinin Otomatikleştirilmesine İlişkin Uygulamalı Bir Çalışma. *International Journal of Management Information Systems and Computer Science*, 41 - 67.

Şirin, S. (2020). Rastgele Orman Algoritmaları İle Otel Özellikleri Analizi. *Uluslararası Yönetim Bilişim Sistemleri ve Bilgisayar Bilimleri Dergisi*, 123-132.

Taley, D. S., ve Pathak, B. (2020). Comprehensive study of software testing techniques and strategies: A review. *International Journal of Engineering Research & Technology*, 817-822.

Tekin, R., ve Yaman, O. (2023). Akıllı Ev Sistemleri için XGBoost Tabanlı Saldırı Tespit Yöntemi. *Zeki Sistemler Teori ve Uygulamaları Dergisi*, 152-158.

Tian, Y., Lo, D., ve Sun, C. (2012). Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction. *2012 19th Working Conference on Reverse Engineering* (s. 215-224). Kingston: IEEE.

Tuteja, M., ve Dubey, G. (2012). A Research Study on importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models. *International Journal of Soft Computing and Engineering (IJSCE), 2*(3), 251-257.

Uddin, A., ve Anand, A. (2019). Importance of Software Testing in the Process of Software Development. *International Journal for Scientific Research & Development*|.

Uzun, B., ve Koruyan, K. (2019). Yazılım Test Sürecinde Durum Raporlamasına Genel Bakış ve Yaklaşımlar. *Yönetim Bilişim Sistemleri Dergisi, 5*(1), 52-63.

Valdivia Garcia, H., ve Shihab, E. (2014). Characterizing and predicting blocking bugs in open source projects. *Proceedings of the 11th Working Conference on Mining Software Repositories* (s. 72–81). New York, USA: Association for Computing Machinery.

Veziroğlu, M., ve Bucak, İ. (2025). Haber Sınıflandırma Sistemlerinde Naive Bayes ve Makine Öğrenmesi Algoritmaları Arasında Performans Karşılaştırması. *Iğdır Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, 57-70.

Wei, J., ve Zou, K. (2019). EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. *EMNLP-IJCNLP 2019.* ArXiv.

Wendong, Y., Zhengzheng, L., ve Bo, J. (2017). A multi-factor analysis model of quantitative investment based on GA and SVM. *2017 2nd International Conference on Image, Vision and Computing (ICIVC).* Chengdu, China: IEEE.

Yang, F.-J. (2018). An implementation of Naive Bayes classifier. *In 2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, 301-306.

Yang, G., Zhang, T., ve Lee, B. (2014). Towards Semi-automatic Bug Triage and Severity Prediction Based on Topic Model and Multi-feature of Bug Reports. *2014 IEEE 38th Annual Computer Software and Applications Conference* (s. 97-106). Vasteras: IEEE.

ÖZ GEÇMİŞ

Adem ÖRPEK

Eğitim

Yüksek Lisans	2024	Maltepe Üniversitesi, Bilgisayar Mühendisliği
Lisans	2019	Anadolu Üniversitesi, Yönetim Bilişim Sistemleri
Ön Lisans	2016	Marmara Üniversitesi, Bilgisayar Programcılığı

İş

2018-2020	Vodafone	Operasyon Uzmanı
2020-2021	Vakıf Katılım Bankası	Yazılım Test Uzmanı
2021- Halen	LCWaikiki	Yazılım Test Mühendisi

Yayınlar

Örpek, Z., Örpek, A., ve Şahinaslan, Ö. (2026). Classification of unstructured text for RPA: A comparative machine learning analysis. *Boletim da Sociedade Paranaense de Matemática*, 44(5).

Örpek, A., Şahinaslan, Ö.(2026). Mobil E-Ticaret Uygulamalarında Yazılım Hatalarının Modül Bazlı Dağılımının İncelenmesi. *II. International Graduate Students Interdisciplinary Symposium (IGSIS)*

Örpek, A., Esin, E. M. (2025). İşletmeler için kurye verimlilik simülasyonu. *Bursa 4. Uluslararası Matematik ve Mühendislik Kongresi*.