T.C.
ERZİNCAN BİNALİ YILDIRIM ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
YAPAY ZEKA VE ROBOTİK ANABİLİM DALI

YAPAY ZEKA TEKNİKLERİ İLE YAZILIM TEST FAALİYETLERİNİN İNCELENMESİ VE YAZILIM KALİTE TAHMİNİ

DÖNE KARHAN

Danışman: Doç. Dr. Fulya ASLAY

TEZ JÜRİ ÜYELERİ
Dr. Ögr. Üyesi Hatice ARSLANTAŞ DALBOY
Doç.Dr. Fulya ASLAY
Doç.Dr. İsmail AKGÜL

YÜKSEK LİSANS TEZİ
ERZİNCAN, 2025

ÖZET

YAPAY ZEKA TEKNİKLERİ İLE YAZILIM TEST FAALİYETLERİNİN İNCELENMESİ VE YAZILIM KALİTE TAHMİNİ

Döne KARHAN

Yüksek Lisans Tezi, Erzincan Binali Yıldırım Üniversitesi, Fen Bilimleri Enstitüsü, Yapay Zeka ve Robotik Anabilim Dalı

Danışman: Doç. Dr. Fulya ASLAY

2025, 77 sayfa

Bu tez çalışması, yazılım test süreçlerinin kaliteyi artırmadaki belirleyici rolünü vurgulayarak, bu süreçlerin yapay zekâ ve makine öğrenmesi teknikleriyle nasıl daha verimli, öngörülebilir ve stratejik şekilde yönetilebileceğini göstermeyi amaçlamaktadır. Dijitalleşmenin hızlanmasıyla birlikte yazılım projelerinde işlevselliğin yanı sıra güvenilirlik, sürdürülebilirlik ve kullanıcı memnuniyeti gibi kalite unsurları önem kazanmakta; bu durum test süreçlerinin planlanması ve yürütülmesini yazılım kalitesinin ana unsurlarından biri hâline getirmektedir. Geleneksel yöntemlerin sınırlılıklarını aşmak için geliştirilen bu çalışmada, yazılım firmalarında görev yapan test mühendislerinden elde edilen anket verileri üzerinden çeşitli makine öğrenmesi algoritmaları uygulanmış, yazılım test süreçlerine ilişkin kalite düzeyi öngörülerek sınıflandırma modelleri oluşturulmuştur. Elde edilen bulgular, geliştirilen modellerin yazılım test süreçlerinde karar vericilere destek olabilecek güvenilir çıktılar ürettiğini ve bu yaklaşımın hem akademik literatürde hem de sektörel uygulamalarda somut katkılar sağlayabileceğini ortaya koymaktadır.

Anahtar Kelimeler: Yazılım testi, Kalite tahmini, Makine öğrenmesi, Yapay zekâ, Veri analizi

ABSTRACT

EXAMINATION OF SOFTWARE TESTING ACTIVITIES AND SOFTWARE QUALITY ESTIMATION WITH ARTIFICIAL INTELLIGENCE TECHNIQUES

Döne KARHAN

Master's Thesis, Erzincan Binali Yıldırım University, Institute of Science and Technology,
Department of Artificial Intelligence and Robotics
Advisor: Assoc. Prof. Dr. Fulya ASLAY
2025, 77 pages

This thesis emphasizes the decisive role of software testing processes in enhancing software quality and aims to demonstrate how these processes can be managed more efficiently, predictably, and strategically through artificial intelligence and machine learning techniques. With the accelerating pace of digitalization, not only functionality but also quality factors such as reliability, sustainability, and user satisfaction have gained prominence in software projects, making the planning and execution of testing processes a key determinant of software quality. To overcome the limitations of traditional methods, this study applied various machine learning algorithms to survey data collected from software test engineers working in the field, and developed classification models to predict the quality level of software testing processes. The findings reveal that the proposed models generate reliable outputs that can support decision-makers in managing software test processes, demonstrating that this approach can make concrete contributions to both the academic literature and sectoral practice.

Keywords: Software testing, Quality prediction, Machine learning, Artificial intelligence, Data analysis

TEŞEKKÜR

Yüksek lisans eğitimim süresince bilgi ve birikimiyle çalışmalarımı yönlendiren, akademik gelişimime değerli katkılar sağlayan danışmanım Doç. Dr. Fulya Aslay'a teşekkür ederim.

Her zaman yanımda olan, sevgisi ve desteğiyle beni bugünlere getiren kıymetli anneme ve babama teşekkürü borç bilirim.

Bu süreçte sabrı, anlayışı ve moral veren desteğiyle her zaman yanımda olan sevgili eşime gönülden teşekkür ederim.

Ayrıca, veri toplama sürecinde katkıda bulunan ve desteğini esirgemeyen değerli iş arkadaşlarıma da teşekkür ederim.

Döne KARHAN
Temmuz, 2025

İÇİNDEKİLER

TABLOLAR DİZİNİ

ŞEKİLLER DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

α	Cronbach Alpha Katsayısı (Güvenilirlik Katsayısı)
β	Momentum katsayısı
γ	Kernel fonksiyonunun etkisini belirleyen parametre
μ	Ortalama Değer
σ	Standart Sapma
Σ	Toplam Sembolü
%	Yüzde Sembolü
F_1	F1 Skoru (precision ve recall dengesini ifade eden performans metriği)
exp	Üstel fonksiyon
AI	Artificial Intelligence (Yapay Zekâ)
ANN	Artificial Neural Network (Yapay Sinir Ağı)
API	Application Programming Interface (Uygulama Programlama Arayüzü)
BT	Information Technologies (Bilgi Teknolojileri)
CNN	Convolutional Neural Network (Evremsel Sinir Ağı)
CSV	Comma-Separated Values (Virgülle Ayrılmış Değerler)
DBMS	Database Management System (Veritabanı Yönetim Sistemi)
DL	Deep Learning (Derin Öğrenme)
GUI	Graphical User Interface (Grafiksel Kullanıcı Arayüzü)
KLOC	Kilo Lines of Code (Bin Satır Kod)
KNN	K-Nearest Neighbors (K-En Yakın Komşu)
LR	Logistic Regression (Lojistik Regresyon)
LLM	Large Language Models (Büyük Dil Modelleri)
MAE	Mean Absolute Error (Ortalama Mutlak Hata)
MAPE	Mean Absolute Percentage Error (Ortalama Mutlak Yüzde Hata)
ML	Machine Learning (Makine Öğrenmesi)
MLP	Multi-Layer Perceptron (Çok Katmanlı Algılayıcı)
MSE	Mean Squared Error (Ortalama Kare Hata)
NLP	Natural Language Processing (Doğal Dil İşleme)
RMSE	Root Mean Square Error (Kök Ortalama Kare Hata)
RF	Random Forest (Rassal Orman Yöntemi)
SQA	Software Quality Assurance (Yazılım Kalite Güvencesi)

SPSS	Statistical Package for the Social Sciences
SVM	Support Vector Machine (Destek Vektör Makineleri)
TDD	Test Driven Development (Test Odaklı Geliştirme)
YSA	Artificial Neural Networks (Yapay Sinir Ağları)

1. GİRİŞ

Bilgi teknolojilerindeki hızlı ilerlemeler, yazılım sistemlerini bireylerin ve kurumların gündelik yaşamında vazgeçilmez bir konuma taşımıştır. Özellikle dijitalleşmenin yaygınlaşması ve kullanıcı taleplerinin çeşitlenmesi, yazılım projelerinin yalnızca işlevsel olarak yeterli olmasını değil; aynı zamanda güvenilir, kullanıcı odaklı ve sürdürülebilir olmasını da zorunlu kılmaktadır. Bu dönüşüm, yazılım geliştirme sürecinde kalite güvencesine duyulan ihtiyacı artırmakta; test faaliyetlerini ise bu sürecin ayrılmaz ve kritik bir parçası hâline getirmektedir.

Yazılım projelerinin başarısı, yalnızca teknik yeterlilikle değil; doğru planlama, etkin yönetim ve kullanıcı memnuniyetini esas alan çıktılarla doğrudan ilişkilidir. Ancak pratikte karşılaşılan zaman ve bütçe sınırlamaları, yazılım test süreçlerinin göz ardı edilmesine ya da yüzeysel olarak yürütülmesine neden olabilmektedir. Bu durum, yazılımın kullanım sürecinde hata üretme olasılığını artırmakta ve hem kullanıcı memnuniyetini hem de kurumsal itibarı olumsuz etkilemektedir. Oysa kaliteli bir yazılım ürününe ulaşmak, yalnızca kodlama sürecine değil; yazılım yaşam döngüsünün her aşamasına entegre edilmiş sistematik test uygulamalarına bağlıdır.

Yazılım testi; yazılımın gereksinimlere uygun çalışıp çalışmadığını denetlemek, olası hataları erken aşamalarda tespit etmek ve bu hataların etkilerini en aza indirgemek amacıyla yürütülen bütüncül bir süreçtir. Geleneksel test yöntemlerinin ötesinde, günümüzde yapay zekâ ve veri madenciliği gibi ileri analiz teknikleri, yazılım test süreçlerini daha etkili ve öngörülebilir hâle getirmek adına önemli imkânlar sunmaktadır. Özellikle büyük ölçekli yazılım projelerinde test faaliyetlerinin otomasyonu ve sınıflandırılması, maliyetlerin azaltılması, zamanın verimli kullanılması ve daha güvenilir sonuçların elde edilmesi açısından kritik rol oynamaktadır.

Literatürde yer alan çalışmalar, yazılım hatalarının projelere olan ekonomik etkilerini açıkça ortaya koymuştur. Örneğin Hayes (2002), 800 Bilgi Teknolojileri (BT) çalışanı ile gerçekleştirdiği çalışmada katılımcıların %97'sinin yazılım hatalarıyla karşılaştığını ve %90'ının bu hataların maliyet veya gelir kaybına neden olduğunu ifade etmiştir. Bu veriler, test süreçlerinin stratejik önemini ve test faaliyetlerinin başarısının yazılım kalitesi üzerindeki etkisini açıkça ortaya koymaktadır.

Yazılım hata tahmini alanındaki çalışmalarda özellikle Naive Bayes algoritmasının sınıflandırma başarımı öne çıkmaktadır. Çatal (2011), 1990-2009 yılları arasında yapılan 90

ayrı çalışmayı inceleyerek bu algoritmanın yüksek performans sağladığını ve yazılım kusur tahminlerinin otomatize edilmesi gerektiğini vurgulamıştır.

Bu tez çalışması, yazılım firmalarında yürütülen test faaliyetlerini çok boyutlu olarak analiz etmeyi, yapay zekâ teknikleri yardımıyla bu faaliyetleri sınıflandırmayı ve bu sınıflandırma sonuçlarına dayanarak yazılım kalite tahmini yapmayı amaçlamaktadır. Araştırmanın temel problemi, test mühendisleri ve analistlerin gerçekleştirdiği değerlendirmeler doğrultusunda test faaliyetlerinin sınıflandırılmasının mümkün olup olmadığını ortaya koymak ve bu sınıflandırma üzerinden yazılım kalitesine ilişkin öngörüler geliştirmektir. Bu amaç doğrultusunda, beşli Likert ölçeği (Büyüköztürk, 2017) kullanılarak toplanan veriler yapay zekâ ve veri madenciliği teknikleri ile analiz edilmiştir.

Araştırmada 90 katılımcıdan elde edilen çoktan seçmeli anket verileri çoğaltılarak 3849 satırlık bir veri seti oluşturulmuştur. Bu veri seti üzerinde veri ön işleme işlemleri (sayısal dönüşüm, Cronbach Alpha güvenilirlik analizi) gerçekleştirilmiş ve ardından Lojistik Regresyon, K-En Yakın Komşu (K-Nearest Neighbors – KNN), Destek Vektör Makineleri (Support Vector Machines – SVM), Naive Bayes, Rastgele Orman (Random Forest) ve Yapay Sinir Ağları (YSA) gibi denetimli öğrenme algoritmaları kullanılarak sınıflandırma ve kalite tahmini yapılmıştır. Model başarımları Doğruluk, F1 Skoru, Ortalama Mutlak Hata (Mean Absolute Error – MAE), Ortalama Kare Hata (Mean Squared Error – MSE), Kök Ortalama Kare Hata (Root Mean Squared Error – RMSE) ve Ortalama Mutlak Yüzde Hata (Mean Absolute Percentage Error – MAPE) gibi metrikler üzerinden değerlendirilmiştir.

Bu araştırma, yazılım test süreçlerinin yalnızca kalite güvencesi sağlama işleviyle sınırlı olmadığını; aynı zamanda veri analitiği ve yapay zekâ destekli karar verme süreçlerine katkı sunduğunu göstermeyi hedeflemektedir. Geliştirilen modellerin, mevcut projelerde test süreçlerinin değerlendirilmesinin yanı sıra; yeni geliştirilecek projelerde test başarısını ve yazılım kalitesini önceden tahmin edebilme potansiyeli bulunmaktadır. Bu bağlamda çalışma, geleneksel hata tahmini yöntemlerinden farklı olarak içerik temelli sınıflandırma yaklaşımı sunmakta ve sektörde uygulanabilir, esnek bir karar destek sistemi altyapısı önermektedir.

Çalışmanın kapsamı, yazılım firmalarında görev yapan uzmanların, sorumlu oldukları projelerdeki yazılım kalite metriklerine ilişkin değerlendirmelerine dayanmaktadır. Katılımcılar, test süreçleri ve proje performansını “çok düşük”ten, “çok yüksek”e kadar değişen

beşli ölçekle derecelendirerek proje kalitesini puanlamıştır. Bu yanıtlar temel alınarak geliştirilen sınıflandırma modelleri sayesinde, projelerin mevcut başarı düzeyleri analiz edilmiş ve benzer projeler için geleceğe yönelik öngörüler oluşturulmuştur. Kullanılan farklı algoritmalar ise modelleme sürecinin esnekliğini artırarak veriye dayalı bir kalite tahmini mekanizması sunmakta ve yazılım mühendisliği literatürüne somut katkılar sağlamaktadır.

Sonuç olarak bu tez, yazılım test süreçlerinin yapay zekâ destekli yöntemlerle değerlendirilmesine yönelik literatürdeki boşluğu doldurmayı hedeflemekte; yazılım kalitesinin sistematik olarak tahmin edilmesine yönelik bütüncül, veri temelli ve yenilikçi bir yaklaşım ortaya koymaktadır. Elde edilen bulguların hem akademik alanda hem de sektörel uygulamalarda yol gösterici olması beklenmektedir.

2. KAVRAMSAL ÇERÇEVE VE İLGİLİ ÇALIŞMALAR

2.1. İlgili Çalışmalar

Bu bölümde, yazılım yaşam döngüsünün önemli bir parçası olan test süreçlerinin, yapay zekâ ve makine öğrenmesi teknikleri ile nasıl desteklendiğine dair literatürde yer alan çalışmalar kapsamlı biçimde incelenmiştir. Mevcut çalışmalar, yazılım test faaliyetlerinin etkinliğini artırmak, kalite tahminlerini güçlendirmek ve süregelen problemlere çözüm üretmek amacıyla bu tekniklerin başarıyla kullanıldığını göstermektedir. Literatürdeki bu yaklaşımlar, tez çalışmasının teorik temelini oluşturmakta ve önerilen modelin geliştirilmesinde yönlendirici rol üstlenmektedir.

Yazılım test faaliyetlerinin, yalnızca hata bulmaya yönelik bir teknik uygulama olmanın ötesinde; yazılım kalitesini güvence altına alma, müşteri memnuniyetini sağlama ve ekonomik sürdürülebilirliği destekleme gibi çok boyutlu etkileri bulunmaktadır. Test süreçlerinin iyi planlanması, fonksiyonel ve fonksiyonel olmayan gereksinimlerin karşılanması açısından önemlidir. Nitekim Standish Grubu'nun raporları (1994–2012) da test süreçlerinin yazılım projelerinin başarısında belirleyici olduğunu ortaya koymaktadır. Basili ve Selby (1987), test yöntemlerini karşılaştırdıkları çalışmalarında, kod okuma tekniklerinin hata tespitinde üstünlüğüne dikkat çekerken; Gelperin ve Hetze (1988), test modellerini safha ve ömür devri modelleri olarak sınıflandırarak gereksinim ve tasarım hatalarının önlenmesindeki rollerini vurgulamıştır.

2000'li yıllarla birlikte test süreçlerinin kapsamı genişlemiş, kalite kontrol, test planlaması, kapsam belirleme ve verimlilik ölçümü gibi konular ön plana çıkmıştır (Whittaker, 2000; Tassey, 2002). Bu gelişmeleri destekleyen verilerden biri, InformationWeek'in 2002 tarihli anketidir. Ankete göre, 800 BT çalışanının %97'si yazılım hatalarıyla ilgili sorun yaşarken, %90'ı bu sorunların maliyet ve gelir kaybına neden olduğunu ifade etmiştir (Hayes, 2002).

Turhan ve Bener (2007), yazılımlarda mevcut hataların erken aşamada tespiti için kaynak kod ölçütlerine dayalı Bayes sınıflandırma temelli bir hata kestirim metodolojisi önermiş ve bu yaklaşımın test kaynak yönetiminde verimlilik sağladığını göstermiştir.

Martin ve arkadaşları (2007), Birleşik Krallık merkezli küçük ölçekli bir yazılım şirketinde yürüttükleri etnografik çalışmayla, yazılım test faaliyetlerinin kurumsal bağlamda nasıl şekillendiğini incelemişlerdir. Çalışmada detaylı biçimde ele alınan entegrasyon testi örneği aracılığıyla; test kapsamı, gereksinim testi ve senaryo tasarımı gibi süreçlerin organizasyonel koşullar ve kısıtlarla nasıl etkilendiği ortaya konmuştur. Yazarlar, yazılım testi araştırmaları ile pratik uygulamalar arasında ciddi bir kopukluk bulunduğunu belirtmiş ve bu boşluğun giderilmesi gerektiğine dikkat çekmişlerdir.

Beşli ve Çavdar (2010), test prosedürlerinin ayrıntılı uygulanmasının yazılım güvenilirliğini doğrudan artırdığını vurgulamışlardır. Kılıç ve Öztürk (2010) ise entegrasyon test süreçlerinin kurulum, modül uyumluluğu ve entegre sistemde işlevsellik açısından planlanmasının önemine değinmişlerdir.

Vural ve Sağıroğlu (2011), güvenlik testlerinin standartlara uygun şekilde yürütülmesinin yazılım dayanıklılığına katkı sunduğunu belirtmiş; Nasir ve Sahibuddin (2011) ise yazılım projelerinin başarısında gereksinimlerin net tanımlanması, gerçekçi zaman planlaması ve kullanıcı memnuniyetinin belirleyici faktörler olduğunu ifade etmişlerdir.

Atagören (2012), yazılım mühendisliğindeki gelişmelerin test süreçlerine olan etkisini, Standish Grubu'nun 1994–2011 verileriyle ilişkilendirerek ele almış; yazılım projelerindeki başarı oranlarındaki artışa dikkat çekmiştir.

Orso ve Rothermel (2014) ise testin, yazılım kalite güvencesi yöntemleri arasında en yaygın uygulamalardan biri haline geldiğini belirtmişlerdir.

Arora ve arkadaşları (2015), yazılım hata kestirimi konusunda mevcut açık problemlere dikkat çekmiş ve özellikle çapraz doğrulama gibi alanlarda daha geniş geçerliliğe sahip modellere ihtiyaç duyulduğunu vurgulamışlardır. Yaptıkları çalışmada, farklı projeler arasında hata kestirimi yapabilen modellerin geliştirilmesinin önemine değinilmiş ve bu alandaki mevcut yöntemlerin sınırlılıklarına işaret edilmiştir. Bu bağlamda, yazılım hata kestirimi modellerinin genellenebilirliğini artırmak için daha kapsamlı ve esnek yaklaşımların geliştirilmesi gerektiği belirtilmiştir.

Hourani ve arkadaşları (2019), yapay zekânın yazılım test süreçleri üzerindeki dönüştürücü etkisini ele almışlardır. Çalışmalarında, yapay zekâ destekli testlerin daha otomatik, tutarlı ve verimli hale geldiğini belirtmişlerdir. Ayrıca, yapay zekâ teknolojilerinin test süreçlerinde zaman tasarrufu sağladığı, hata tespitinde doğruluğu artırdığı ve genel olarak yazılım kalitesini iyileştirdiği ifade edilmiştir. Bu bulgular, yapay zekânın yazılım test süreçlerine entegrasyonunun, kalite güvencesi ve test verimliliği açısından önemli katkılar sunduğunu göstermektedir.

Yücalar ve Borandağ (2019), test olgunluk modelleri ile süreçlerin kalitesine katkı sağlandığını ortaya koymuş; aynı zamanda Uzun ve Koruyan (2019), hata tipine dayalı test raporlarının karar verme süreçlerinde önemli bir referans noktası sunduğunu ifade etmişlerdir. Yücalar ve Borandağ (2019) ayrıca, TMMI modelinin sistematik test süreçleri oluşturmadaki etkinliğine dikkat çekmiştir.
Garousi ve arkadaşları (2020) ise doğal dil işleme (NLP) destekli yazılım test yaklaşımlarına odaklandıkları sistematik derlemelerinde, bu yöntemlerin yazılım testine olan katkılarını kapsamlı biçimde değerlendirmiştir. Çalışmada, NLP tabanlı test metotlarının insan müdahalesini azaltarak test süreçlerinin verimliliğini artırabileceği vurgulanmıştır.

Lima ve Vergilio (2020), sürekli entegrasyon ortamlarında test durumlarının önceliklendirilmesine ilişkin çözüm arayışlarını ortaya koymuş; test önceliklendirmenin artan önemine ve çözülmemiş problemlere dikkat çekmiştir.

Zhao ve arkadaşları (2021), yazılım sektörünün küresel ölçekte ekonomik büyümenin temel itici güçlerinden biri haline geldiğini vurgulamaktadır. Araştırmacılar, sektördeki teknik yeniliklerin sürdürülebilir kalkınma açısından kritik öneme sahip olduğunu belirtmiş ve yazılım kalitesini bu bağlamda stratejik bir unsur olarak değerlendirmiştir. Yazılım testlerinin, kalite güvencesinin sağlanmasında en güvenilir yöntemlerden biri olduğunu ifade eden çalışma, geliştirici ekiplerin, test mühendislerinin ve bağımsız kalite güvence uzmanlarının rolünü açıkça ortaya koymaktadır. Yazılım testi süreçlerinin net bir şekilde tanımlanması ve bu süreçlerin standartlara uygun şekilde yürütülmesi, yazılım projelerinde başarı oranını doğrudan etkilemektedir.

Yazılım testi yalnızca hata bulma amacıyla değil, aynı zamanda yazılım sistemlerinin belirlenen gereksinimlere ne ölçüde uyum sağladığını değerlendirmek için de uygulanmaktadır.

Geleneksel yazılım geliştirme süreçlerinden farklı olarak, yapay zekâ (AI) ve makine öğrenmesi (ML) tabanlı modeller tümevarımsal yöntemlerle oluşturulmakta, bu da test ve doğrulama süreçlerine yeni bir perspektif kazandırmaktadır (De Silva ve Alahakoon, 2022).

De Silva ve Alahakoon (2022), yapay zekâ yaşam döngüsünü ele alırken; Salahat ve arkadaşları (2023), yazılım geliştirme projelerinde kalite güvence süreçlerinin yalnızca yazılımın doğruluğunu değil, aynı zamanda projenin genel hedeflere uygun yürütülmesini de güvence altına aldığını belirtmiştir. Araştırmada, yazılım analizinin derinleştirilmesi gerektiği vurgulanmış; mevcut uygulamalarda analiz yöntemlerinin yetersizliği, kullanıcı direnci ve operasyonel zorluklar gibi sorunların kaliteyi olumsuz etkilediği ifade edilmiştir. Ayrıca tip ve kalibrasyon süreçlerine dair araştırmaların hâlen yazılım testi alanında önemli bir geliştirme hedefi olarak varlığını sürdürdüğü belirtilmiştir.

Panwar ve Peddi (2023) tarafından yapılan sistematik haritalama çalışması, yazılım testi süreçlerinde makine öğrenmesi tekniklerinin kullanımına dair kapsamlı bir analiz sunmaktadır. Çalışmada, özellikle test vaka üretimi, test yürütme ve hata tahmini gibi alanlarda sınıflandırma ve regresyon tekniklerinin kullanımına odaklanılmıştır. Yazarlar, bu tekniklerin yazılım test süreçlerinin doğruluğunu, kapsayıcılığını ve maliyet etkinliğini artırma potansiyeline sahip olduğunu ortaya koymuştur. Ayrıca çalışmada, yazılım test süreçlerine entegre edilen öğrenme tabanlı algoritmaların, test mühendislerinin iş yükünü azalttığı ve test süresini kısalttığı da vurgulanmaktadır.

Wang ve arkadaşları (2023), yazılım testlerinde Büyük Dil Modellerinin (Large Language Models, LLM) kullanımına yönelik kapsamlı bir değerlendirme yapmışlardır. Bu çalışma, doğal dil işleme temelli modellerin test senaryolarının otomatik oluşturulmasında ve yazılım hatalarının semantik analizi yoluyla tespitinde yüksek performans sergilediğini göstermektedir. Özellikle ChatGPT benzeri büyük modellerin yazılımın gereksinim analizinden entegrasyon testine kadar birçok aşamada kullanılabileceği belirtilmiştir. Araştırma, yazılım testinde insan müdahalesini azaltan ve daha tutarlı sonuçlar üreten yapay zekâ çözümlerinin gelişen bir alan olduğuna dikkat çekmektedir.

Nama (2024) tarafından gerçekleştirilen çalışmada, yapay zekâ destekli otomasyonun test kapsamını genişletme, senaryo üretimini hızlandırma ve hata tahmini süreçlerinde isabet oranını artırma potansiyeli değerlendirilmiştir. Araştırmada özellikle regresyon testleri ve

sürekli entegrasyon ortamlarında yapay Zekâ tabanlı sistemlerin hata tespit oranlarını artırdığına ve test döngülerini kısalttığına yer verilmiştir. Ayrıca model temelli test otomasyonunun yazılım geliştirme sürecinin erken aşamalarında kaliteyi öngörme açısından avantaj sağladığı ifade edilmiştir.

Safaat ve Tjhin (2024) tarafından gerçekleştirilen karşılaştırmalı çalışmada, Katalon Studio kullanılarak yürütülen otomatik testlerin, manuel test yöntemlerine kıyasla daha hızlı ve verimli sonuçlar sağladığı ortaya konmuştur. Borsa işlemlerine yönelik bir uygulama üzerinde yürütülen dokuz farklı test senaryosu üzerinden yapılan karşılaştırmalar sonucunda, otomatik testlerin zaman açısından önemli bir avantaj sunduğu belirlenmiştir. Otomatik testlerin 3.177 saniyede tamamlandığı, buna karşın manuel testlerin 3.492 saniye sürdüğü gözlemlenmiştir. Bu sonuçlar, yazılım testlerinde otomasyonun zaman tasarrufu ve hızlı geri bildirim açısından değerli bir alternatif sunduğunu ortaya koymaktadır.

Khilari ve arkadaşları (2024), yazılım kalite güvencesinde kullanılan çeşitli makine öğrenimi algoritmalarının etkinliğini derinlemesine incelemiştir. Araştırmada, bu algoritmaların proje metriklerini analiz etme, kod karmaşıklığını değerlendirme, proje süresi ve geliştirici verimliliği gibi faktörleri tahmin etme potansiyeline sahip olduğu vurgulanmıştır. Ayrıca her algoritmanın farklı test türleri ve kalite kriterleri için farklı avantajlar sunduğu, doğru algoritma seçiminin yazılım kalitesini doğrudan artırabileceği ifade edilmiştir.

Tüm bu literatür değerlendirmeleri, yazılım testlerinin yalnızca hata tespiti değil; aynı zamanda kaliteyi güvence altına alma, süreçleri etkinleştirme ve proje başarısını artırma gibi temel işlevlere sahip olduğunu ortaya koymaktadır. Yapay zekâ destekli modellerin test süreçlerine entegrasyonu, gelecekte yazılım projelerinin başarısında belirleyici bir unsur olarak ön plana çıkmaktadır.

Mevcut literatürde yazılım testiyle ilgili çalışmaların büyük çoğunluğu, test eforu kestirimi ve hata tahmini üzerine yoğunlaşmıştır. Ancak test faaliyetlerinin sınıflandırılması yoluyla kalite tahmini yapılmasına yönelik araştırmalar oldukça sınırlıdır. Bu tez çalışması, bu boşluğu doldurmayı hedeflemektedir. Test faaliyetleri, yazılım kalitesine etkisi temel alınarak kalite sınıflarına ayrılacak; ardından bu sınıflandırmadan elde edilen örüntülerle, yeni gelen bir projede uygulanması planlanan test sürecinin başarı düzeyi öngörülmeye çalışılacaktır.

Bu yaklaşım klasik test eforu tahmini yöntemlerinden farklı olarak; içerik temelli sınıflandırmaya dayalı, benzerlik odaklı ve kaliteye dönük bir tahmin modeli geliştirmeyi amaçlamaktadır. Böylece, yazılım test faaliyetleri ve yapay zekâ teknikleri arasında özgün bir bağ kurularak hem literatüre katkı sağlanması hem de sektörel uygulamalarda kullanılabilir bir çözüm modeli ortaya konması hedeflenmektedir.

İlerleyen bölümlerde problem kapsamı ve kullanılan tekniklerle ilgili ayrıntılı bilgiler sunulacaktır.

2.2. Kavramsal Çerçeve

2.2.1. Yazılımda kalite

Yazılımda kalite, bir yazılım ürününün kullanıcı beklentilerini karşılaması, güvenilir biçimde çalışması ve sürdürülebilirliğinin sağlanması gibi temel kriterler doğrultusunda tanımlanır. Kaliteli bir yazılım ürünü, yalnızca doğru çalışmakla kalmaz; aynı zamanda değişen gereksinimlere uyum sağlayabilir, ölçeklenebilir ve uzun vadeli kullanıma elverişlidir. Bu düzeyde kaliteye ulaşmak için yazılım geliştirme sürecinin her aşamasında planlı, sistematik ve disiplinli bir yaklaşım benimsenmelidir.

Yazılım projelerinin kapsamı ve karmaşıklığı arttıkça, başarıya ulaşmak için süreç yönetimindeki resmiyet ve disiplinin de artması kaçınılmaz hale gelir. Özellikle küçük ölçekli projelerde sıklıkla göz ardı edilen süreç yönetimi, büyük ve karmaşık projelerde doğrudan kaliteyi belirleyen başlıca faktörlerden biri haline gelir. Plansız, denetimsiz ve rastlantısal yaklaşımlar, yazılımın hem işlevselliğini hem de güvenilirliğini riske atar.

Yazılım kalite güvencesi için dikkate alınması gereken temel süreçler şunlardır:

- Analiz ve Gereksinim Yönetimi: Yazılımın hedeflerinin net ve doğru şekilde tanımlanması, tüm proje sürecinin sağlıklı ilerlemesini sağlar.
- Tasarım ve Planlama: Yazılımın işlevsel ve teknik gereksinimlerini karşılayacak biçimde yapılandırılması, sağlam bir temel oluşturur.
- Test ve Gözden Geçirme: Hataların erken aşamalarda tespit edilerek düzeltilmesi hem zaman hem de maliyet açısından tasarruf sağlar.

Yazılım geliştirme süreçlerinin sürekli olarak iyileştirilmesi, yalnızca kaliteyi yükseltmekle kalmaz; aynı zamanda ekiplerin üretkenliğini artırır ve yazılımın kullanıcı ihtiyaçlarına uygunluğunu uzun vadede garanti altına alır. Bu bağlamda kalite yönetimi, yalnızca teknik bir gereklilik değil; aynı zamanda sürdürülebilir başarı için vazgeçilmez bir stratejik araçtır.

2.2.1.1. Kalite unsurları

Yazılım kalitesi, yalnızca yazılımın hatasız çalışmasıyla sınırlı değildir; aynı zamanda sistemin kullanıcı ihtiyaçlarını karşılaması, güvenilirliği, sürdürülebilirliği ve çevresel değişkenlere uyum sağlayabilmesi gibi çok boyutlu özellikleri içerir. Yazılım projelerinde kalite hem teknik hem de organizasyonel süreçlerin uyum içinde yürütülmesiyle elde edilebilir. Kaliteye yönelik bütünsel bir yaklaşım, yazılımın yalnızca geliştirme aşamasında değil, aynı zamanda işletme ve bakım süreçlerinde de başarısını sürdürebilmesini sağlar.

Yüksek kaliteli bir yazılım ürünü geliştirmek için dikkate alınması gereken temel faktörler aşağıda açıklanmıştır:

- Standartlara Uyum: Yazılımın, uluslararası geçerliliğe sahip kalite standartlarıyla uyumlu olarak geliştirilmesi kalite düzeyini doğrudan etkiler. Özellikle ISO/IEC 25010 standardı, yazılım ürünlerinde kaliteyi ölçmeye yönelik kapsamlı bir model sunar. Bu tür standartlar sayesinde yazılımın doğrulanabilirliği, güvenilirliği ve geniş çapta kabul görmesi sağlanır.

- Güvenlik ve Güvenilirlik: Yazılımın yetkisiz erişimlere, veri kayıplarına veya güvenlik açıklarına karşı korunaklı olması; kullanıcıların sisteme duyduğu güveni pekiştirir. Aynı şekilde sistemin hata toleransı, sürekli erişilebilirliği ve kesintisiz hizmet sunabilme kapasitesi de güvenilirlik göstergesidir.

- Kullanılabilirlik ve Kullanışlılık: Bir yazılımın kullanıcılar tarafından kolayca öğrenilebilir, kullanılabilir ve ihtiyaçlara uygun çözümler sunuyor olması, kullanıcı deneyimini doğrudan etkiler. Basit, sezgisel ve tutarlı bir arayüz tasarımı, yazılımın kabul edilebilirliğini artırır.

- Taşınabilirlik ve Esneklik: Farklı platformlar, cihazlar ya da işletim sistemlerinde sorunsuz çalışabilen bir yazılım, taşınabilirlik açısından başarılı kabul edilir. Esneklik ise yazılımın

değişen gereksinimlere ve çevresel koşullara kolayca uyum sağlayabilme yeteneğidir. Bu iki faktör, özellikle uzun vadeli yazılım bakım süreçlerinde kritik öneme sahiptir.

- Verimlilik ve Performans: Yazılımın, sistem kaynaklarını ne kadar etkin kullandığı ve verilen görevleri ne kadar sürede yerine getirdiği verimlilik ve performans göstergeleridir. Özellikle yüksek trafik alan sistemlerde, performans testleri ve optimizasyonlar kaliteyi belirleyen kilit unsurlardır.

- Dokümantasyon ve Eğitim: Yazılımın tüm süreçlerinin anlaşılabilir, güncel ve kapsamlı bir şekilde belgelenmesi sürdürülebilirliği destekler. Teknik dokümanlar, kullanıcı kılavuzları ve eğitim materyalleri sayesinde hem geliştirici ekip hem de son kullanıcı yazılımla daha etkin etkileşim kurabilir.

- Uyum ve Entegrasyon: Yazılımın dış sistemlerle ve mevcut altyapıyla entegre çalışabilme kabiliyeti, sistemin genel başarısını belirler. Özellikle kurumsal sistemlerde, diğer yazılım çözümleriyle uyumlu çalışabilmek büyük avantaj sağlar.

- İşlevsel Uygunluk: Yazılımın, kullanıcı veya kurum tarafından belirlenmiş gereksinimlere uygun olarak çalışması, işlevsel kaliteyi ifade eder. Beklenen işlevlerin eksiksiz ve doğru şekilde yerine getirilmesi, yazılımın başarısını artırır.

- Sürdürülebilirlik: Kaliteli yazılım, yalnızca bugünü değil, gelecekteki değişiklik ve bakım süreçlerini de destekleyebilmelidir. Modüler yapılar, temiz kodlama prensipleri ve sürekli entegrasyon süreçleri bu sürdürülebilirliği mümkün kılar.

- Ekip Yetenekleri ve Takım Dinamiği: Yazılım geliştirme sürecinde yer alan ekip üyelerinin bilgi, beceri ve deneyimi; projenin kalitesini doğrudan etkiler. Etkili bir ekip çalışması, açık iletişim ve güçlü koordinasyon, yazılım kalitesinin istikrarlı bir şekilde korunmasını sağlar.

- Karmaşıklık ve Tasarım Sadeliği: Yazılımın tasarımındaki karmaşıklık, hata olasılığını artırabilir. Bu nedenle sade, anlaşılır ve iyi yapılandırılmış bir mimari tercih edilmelidir. Yüksek karmaşıklık, bakım zorluklarına ve düşük sürdürülebilirliğe yol açabilir.

- Zaman ve Planlama: Kaliteli bir yazılım geliştirmek için yeterli zamanın ayrılması gereklidir. Zaman baskısı altında yürütülen projeler, test ve kalite güvence süreçlerinin ihmal edilmesine yol açabilir. Bu da yazılımın uzun vadeli başarısını riske atar.

Yazılım kalitesini yalnızca teknik başarım düzeyinde değil; aynı zamanda kullanıcı memnuniyeti, sürdürülebilirlik, çevresel uyum ve ekip performansı gibi çok yönlü ölçütlerle değerlendirmek gerekir. Bu bağlamda geliştirilen yazılım kalite modelleri, yapay zekâ destekli otomatik kalite tahmini sistemlerine de temel teşkil etmektedir. Yazılım test süreçlerinden elde edilen verilerin anlamlandırılması ve kalite öngörülerinin makine öğrenmesi algoritmalarıyla modellenmesi, gelecekte yazılım kalite yönetiminin daha öngörülebilir ve etkin hale gelmesini sağlayacaktır.

2.2.2. Yazılım testlerinin yazılım kalitesine etkisi

Yazılım test süreçleri, yazılım kalite güvencesinin sağlanmasında merkezi bir rol oynamakta ve geliştirme yaşam döngüsünün her aşamasında etkili bir araç olarak kullanılmaktadır. Test süreçleri, yalnızca teknik doğruluk sağlamaz; aynı zamanda kullanıcı memnuniyetini artırmak, sistem sürdürülebilirliğini garanti altına almak ve yazılımın beklenen işlevselliği güvence altına almak gibi hedeflere de hizmet eder.

Hem işlevsel hem de işlevsel olmayan test yaklaşımları, yazılımın gereksinimlere uygunluğunu değerlendirerek olası hataları erken evrede tespit etmeyi sağlar. Kalite kontrol ve test süreçlerinin yetersizliği, projelerde yüksek hata oranlarına, bütçe aşımlarına ve teslimat gecikmelerine yol açabilmektedir. Bu bağlamda yapılan akademik çalışmalar, yazılım test stratejilerinin kalite üzerindeki etkilerini açıkça ortaya koymaktadır.

Basili ve Selby (1987), yazılım test stratejilerini karşılaştırdıkları deneysel çalışmalarında, kod gözden geçirme yönteminin hata tespitinde diğer test tekniklerine kıyasla daha etkili olduğunu göstermiştir. Bu sonuç, test sürecinin yalnızca çalıştırılabilir test senaryolarıyla değil, aynı zamanda kaynak kodun yapısal analiziyle de desteklenmesi gerektiğini ortaya koymaktadır.
Gelperin ve Hetze (1988) ise yazılım test süreçlerini iki ana kategoriye ayırmıştır: safha modelleri ve yaşam döngüsü (ömür devri) modelleri. Her iki modelde de gereksinim, tasarım ve uygulama aşamalarında yapılacak testlerin kaliteyi artırıcı rolüne dikkat çekilmiştir. Bu modeller, test faaliyetlerinin yalnızca sonuç odaklı değil, süreç odaklı olarak da ele alınması gerektiğini vurgular.

Whittaker (2000), test süreçlerini modelleme, senaryo seçimi ve değerlendirme olmak üzere üç temel aşamada ele alarak, bu yapılandırmanın yazılımın güvenilirliğini artırmada kritik öneme sahip olduğunu belirtmiştir. Özellikle test senaryosu oluşturma ve seçim sürecinin, yazılımın risk taşıyan bileşenlerini erken evrede ortaya çıkarabildiği ifade edilmiştir.

Tassey (2002), yazılım test süreçlerinin eksikliği nedeniyle ortaya çıkan kalite kayıplarının maliyet artışı ve teslimat gecikmeleriyle sonuçlandığını belirtmiştir. Ayrıca test faaliyetlerinin yalnızca kodlama aşamasında değil, analiz ve tasarım evrelerinde de gerçekleştirilmesinin, yazılım geliştirme sürecine olan katkısını önemli ölçüde artıracağını vurgulamıştır.

Literatür genelinde, yazılım test süreçlerinin sadece hata tespitine yönelik bir faaliyet olmadığı; aynı zamanda kalite güvence mekanizması olarak sistematik, disiplinli ve stratejik bir şekilde yapılandırılması gerektiği sonucuna ulaşılmıştır. Test süreçleri; entegrasyon, güvenlik, süreç analizi ve olgunluk modelleri gibi farklı alanlarda yazılım kalitesine çok yönlü katkılar sunmaktadır. Bu süreçlerin her biri, yazılım yaşam döngüsünün farklı evrelerinde hatasız, güvenilir ve kullanıcı beklentilerini karşılayan ürünlerin ortaya konulmasına katkıda bulunmaktadır.

2.2.3 Yapay zekâ

Yapay zekâ (YZ), yalnızca akademik çevrelerde değil, toplumun genelinde de ilgi uyandıran teknolojik kavramlardan biridir. İlk kez 1956 yılında Dartmouth Konferansı'nda ortaya atılan bu kavram, günümüze kadar farklı alanlardan beslenerek gelişimini sürdürmüştür. Yapay zekâ, genel anlamda bilgisayarların veya makinelerin insan benzeri davranışlar sergileyebilme, öğrenme ve karar verme gibi bilişsel işlevleri yerine getirme yeteneği olarak tanımlanmaktadır (TÜBA, 2020).

Toplumun büyük bir kısmı yapay zekâyı, insan gibi hareket eden robotlarla özdeşleştirse de bu teknolojiyi gerçekten anlamak, öncelikle insan zihninin nasıl çalıştığını kavramaktan geçmektedir. Henüz beynin işleyişi tam anlamıyla açıklanamazken, bu karmaşık yapıyı dijital sistemlerle taklit etmeye çalışan yapay zekâ sistemleri hem bilim insanları hem de kamuoyu açısından merak uyandırmaktadır (Öztürk ve Akman, 2023).

Yapay zekâ, disiplinler arası bir araştırma alanıdır. Bilgisayar mühendisliği, yazılım, elektronik, matematik, felsefe, etik gibi farklı bilim alanlarını bir araya getirir. YZ sistemleri; algılama, öğrenme, problem çözme, tahmin ve karar verme gibi işlevleri insan zekâsına benzer biçimde yerine getirebilmek üzere tasarlanır (Yıldırım ve Küçükcan, 2023). Bu yeteneklerin kazandırılmasında en büyük rolü oynayan bileşen ise makine öğrenmesidir. Makine öğrenmesi, sistemlerin verilerden öğrenmesini ve öğrendiklerini yeni durumlara uygulamasını sağlar (Alpaydın, 2020).

Yapay zekânın uygulama alanları oldukça geniştir. Görüntü ve ses tanıma teknolojilerinden doğal dil işlemeye, karar destek sistemlerinden öneri motorlarına kadar pek çok teknolojide yapay zekâ kullanılmaktadır. Özellikle kameralar ve mikrofonlar aracılığıyla toplanan verilerin analiz edilmesi sayesinde, yapay zekâ sistemleri çevresel koşulları yorumlayabilir hale gelmiştir (Çevik ve Arslan, 2022).

Karar destek sistemleri de YZ'nin önemli uygulama alanlarından biridir. Bu sistemler, veriye dayalı karar alma süreçlerinde insanlara destek sunar (Koç ve Altun, 2023).

Yapay zekâ yalnızca teknik bir ilerleme değil; aynı zamanda etik, sosyal ve ekonomik boyutları olan bir dönüşüm sürecidir. Mahremiyet, algoritmik önyargı, kararların sorumluluğu gibi sorunlar, yapay zekânın gündeme getirdiği önemli tartışma başlıklarıdır (Floridi vd., 2018)

Buna rağmen, yapay zekâ teknolojileri sağlık, tarım, ulaşım, finans ve akıllı şehir planlaması gibi birçok alanda insan yaşamını kolaylaştırmaktadır. Yapay zekâ doğru şekilde kullanıldığında, tekrarlayan görevler otomatikleştirilebilir, riskler azaltılabilir ve daha hızlı, güvenilir karar alma süreçleri sağlanabilir (Yıldız ve Akbulut, 2021).

2.2.4 Yapay zekânın uygulama perspektifleri

Yapay zekâ teknolojileri, günümüzde yalnızca teorik bir kavram olmaktan çıkmış, çok çeşitli sektörlerde aktif biçimde uygulama alanı bulmuştur. Veri analitiği, karar destek sistemleri, otomasyon, görüntü işleme ve doğal dil işleme gibi yetenekleri sayesinde yapay zekâ, sağlık hizmetlerinden tarıma, ulaşım sistemlerinden eğitim teknolojilerine kadar geniş bir yelpazede dönüşüm yaratmaktadır (Floridi vd., 2018)

Yapay zekâ sistemlerinin bu kadar geniş bir alana entegre edilebilmesinin temelinde, büyük veriyle başa çıkabilme ve karmaşık örüntüleri yüksek doğrulukla tanıma yeteneği yatmaktadır. Son yıllarda yapılan çalışmalar, yapay zekâ destekli çözümlerin yalnızca verimliliği artırmakla kalmadığını, aynı zamanda insan kaynaklı hataları azalttığını ve karar süreçlerini daha öngörülebilir hale getirdiğini göstermektedir (Yıldız ve Akbulut, 2021).

Özellikle derin öğrenme, yapay sinir ağları ve destek vektör makineleri gibi modern algoritmaların gelişimiyle birlikte, yapay zekânın disiplinler arası uygulama alanları giderek genişlemekte ve bu teknolojiler, yazılım mühendisliğinden biyoinformatiğe kadar pek çok alanda stratejik bir rol üstlenmektedir. Bu gelişmeler, yapay zekânın yalnızca bir araç değil, aynı zamanda teknoloji temelli karar alma süreçlerinin merkezinde konumlandığını ortaya koymaktadır (Jordan ve Mitchell, 2015).

Bu bağlamda, yapay zekâ yalnızca teknik bir yenilik değil, aynı zamanda iş modelleri, toplumsal yapı ve bilgi yönetimi pratiklerini yeniden şekillendiren güçlü bir paradigma olarak değerlendirilmektedir.

2.2.4.1. Sağlıkta yapay zekâ

Günümüzde yapay zekâ (YZ), sağlık alanında yalnızca yardımcı bir araç değil, klinik süreçlerin temel bir parçası haline gelmiştir. Özellikle görüntüleme teknolojilerinde, YZ algoritmaları kanser, akciğer hastalıkları ve kalp rahatsızlıkları gibi birçok hastalığın erken teşhisinde yüksek doğruluk oranları sunmaktadır (Shaik vd., 2023). Bu sistemler; BT, MR ve ultrason görüntülerini değerlendirerek radyologlara destek olmakta, tanısal gecikmeleri azaltmaktadır.

2023 yılında yapılan sistematik bir analizde, YZ uygulamalarının en yaygın olarak radyoloji, gastroenteroloji ve kardiyoloji alanlarında kullanıldığı, bunun da görüntü tabanlı tanılara olan güveni artırdığı ortaya konmuştur (Horowitz, 2023). Bununla birlikte, uzaktan hasta izleme sistemlerinin gelişimi, kronik hastalıkların yönetiminde de yapay zekânın rolünü güçlendirmektedir. Bu sistemler; hasta verilerini gerçek zamanlı analiz ederek, doktorlara olası komplikasyonlar hakkında erken uyarılar sağlamaktadır (Shaik vd., 2023).

Amerikan Tabipler Birliği'nin (AMA) 2025 verilerine göre, ABD'de yapay zekâ kullanan hekim oranı 2023'te %38 iken, 2024 itibariyle bu oran %66'ya yükselmiştir. Bu durum,

hekimlerin klinik kararlarda YZ destekli sistemlere güven duymaya başladığını göstermektedir (American Medical Association, 2025)

Sağlık sektöründe yapay zekâ tabanlı karar destek sistemlerinin yaygınlaşması, sadece tanı süreçlerini hızlandırmakla kalmayıp, hasta güvenliğini artırma ve maliyetleri azaltma potansiyeli ile de dikkat çekmektedir.

2.2.4.2. Ulaşım ve lojistikte yapay zekâ

Yapay zekâ (YZ), ulaşım ve lojistik sektörlerinde operasyonel verimliliği artırmak, kaynak kullanımını iyileştirmek ve sürdürülebilir çözümler üretmek amacıyla yaygın olarak kullanılmaya başlanmıştır. Özellikle rota optimizasyonu, taşıma yönetimi ve talep tahmini gibi alanlarda YZ tabanlı sistemler ciddi performans iyileştirmeleri sağlamaktadır. Uber Freight'in 2025 yılında yayımladığı rapora göre, YZ destekli rota planlama algoritmaları sayesinde kamyonların boş seyir oranı %15'e kadar azaltılmış; bu da hem maliyetlerin düşürülmesine hem de karbon ayak izinin küçülmesine katkı sağlamıştır (Uber Freight, 2025).

Sadece büyük lojistik firmaları değil, küçük ve orta ölçekli işletmeler de YZ'den yararlanmaktadır. Özellikle depo içi operasyonlarda kullanılan YZ destekli otomasyon sistemleri, ürün toplama, stok yönetimi ve paketleme işlemlerinde hem hız hem de doğruluk kazandırmaktadır (Acropolium, 2025). Transport Logistic (2025) tarafından yayımlanan sektörel analizde, lojistik şirketlerinin %68'inin YZ tabanlı karar destek sistemlerini aktif olarak kullandığı, %42'sinin ise önümüzdeki iki yıl içinde bu sistemlere geçiş yapmayı planladığı belirtilmiştir.

Tüm bu gelişmeler, yapay zekânın ulaşım ve lojistik alanında sadece geçici bir yenilik değil, köklü bir dönüşüm aracı olduğunu göstermektedir. Akıllı lojistik sistemlerinin yaygınlaşması ile birlikte daha çevreci, hızlı ve ekonomik tedarik zincirleri inşa etmek mümkün hale gelmiştir.

2.2.4.3. Eğitimde uyarlanabilir sistemler

Yapay zekâ destekli uyarlanabilir öğrenme sistemleri, öğrencilerin bireysel ihtiyaçlarına göre özelleştirilmiş eğitim deneyimleri sunarak öğrenme süreçlerini daha etkili ve verimli hale getirmektedir. Bu sistemler, öğrencilerin performans verilerini analiz ederek, öğrenme

materyallerini ve öğretim stratejilerini dinamik bir şekilde uyarlamakta, böylece her öğrencinin kendi hızında ve tarzında öğrenmesini sağlamaktadır (Darad, 2024).

2023 yılında yapılan bir araştırma, kişiselleştirilmiş uyarlanabilir öğrenme sistemlerinin öğrencilerin akademik başarılarını ve öğrenme süreçlerini olumlu yönde etkilediğini ortaya koymuştur. Bu sistemler, öğrencilerin güçlü ve zayıf yönlerini belirleyerek, öğrenme materyallerini buna göre uyarlamakta ve öğrencilerin motivasyonunu artırmaktadır (Yalanskyi, 2024).

Ayrıca, yapay zekâ destekli uyarlanabilir öğrenme sistemleri, öğretmenlerin iş yükünü azaltmakta ve öğrencilere daha fazla bireysel destek sağlama imkânı sunmaktadır. Bu sistemler, öğrencilerin öğrenme süreçlerini sürekli izleyerek, öğretmenlere öğrencilerin ilerlemesi hakkında detaylı geri bildirimler sunmakta ve öğretim stratejilerini buna göre şekillendirmelerine yardımcı olmaktadır (Darad, 2024).

2.2.4.4. Tarımda akıllı uygulamalar

Tarım sektörü, yapay zekâ (YZ), nesnelerin interneti (IoT) ve robotik teknolojilerin entegrasyonu sayesinde önemli bir dönüşüm geçirmektedir. Bu teknolojiler, tarımsal üretimde verimliliği artırmak, kaynak kullanımını optimize etmek ve çevresel etkileri azaltmak amacıyla kullanılmaktadır (Hasan vd., 2022).

Özellikle hassas tarım uygulamalarında, YZ destekli sistemler toprak nemi, hava durumu ve bitki sağlığı gibi parametreleri gerçek zamanlı izleyerek, sulama ve gübreleme işlemlerini optimize etmektedir. Bu sayede, su ve gübre kullanımında tasarruf sağlanmakta, aynı zamanda ürün verimliliği artırılmaktadır (Gupta ve Pal, 2025).
Ayrıca, görüntü işleme ve makine öğrenmesi teknikleri kullanılarak, bitki hastalıkları ve zararlılar erken dönemde tespit edilebilmekte, bu da hastalıkların yayılmasını önleyerek ürün kayıplarını azaltmaktadır (Albanese vd., 2021).

Bununla birlikte, YZ tabanlı karar destek sistemleri, çiftçilere ekim zamanı, hasat dönemi ve pazarlama stratejileri konusunda öneriler sunarak, tarımsal karar alma süreçlerini iyileştirmektedir (Chaterji vd., 2020).

Ancak, bu teknolojilerin yaygınlaşması için altyapı eksiklikleri, yüksek maliyetler ve çiftçilerin dijital okuryazarlık düzeyleri gibi engellerin aşılması gerekmektedir. Bu nedenle, kamu ve özel sektör iş birliğiyle eğitim programları ve teşvik mekanizmalarının geliştirilmesi önem arz etmektedir (Hasan vd., 2022).

2.2.4.5. Finans ve güvenlikte yapay zekâ

Yapay zekâ (YZ) teknolojileri, finans sektöründe devrim niteliğinde değişikliklere yol açmaktadır. Özellikle, YZ'nin kredi değerlendirme süreçlerine entegrasyonu, finansal kurumların kredi risklerini azaltmalarına ve kârlılıklarını artırmalarına olanak tanımaktadır. Gelişmiş algoritmalar, yüksek risk taşıyan bireyleri tespit etmek ve krediye uygun kişileri değerlendirmek konusunda büyük katkılar sunmaktadır. Özellikle derin öğrenme ve makine öğrenimi teknikleri, geleneksel finansal kurumların hizmet veremediği bireyleri hedeflemek için etkili bir yöntemdir (Sarı, 2024).

Ayrıca, YZ destekli sistemler, dolandırıcılık tespiti ve önlenmesinde önemli bir rol oynamaktadır. YZ, anomalileri ve alışılmadık aktiviteleri belirleyerek, dolandırıcılıkları tespit ve önlemek için kullanılmaktadır. Bu sayede, finansal kurumlar, potansiyel riskli müşterileri tespit etmekte ve gerekli önlemleri alabilmektedir (Yıldız, 2022).

Ancak, YZ'nin finans sektöründe kullanımı bazı zorlukları da beraberinde getirmektedir. Özellikle, veri gizliliği ve güvenliği, şeffaflık ve hesap verebilirlik, adalet ve önyargı gibi etik sorunlar, YZ uygulamalarının etkinliğini sınırlayabilmektedir. Bu nedenle, finansal kurumlar ve düzenleyici otoriteler, YZ ve veri analitiği uygulamalarının etik anlayışına uygun olmasını sağlamak için çeşitli politikalar ve düzenlemeler geliştirmektedir (Şahin, 2024)

2.2.4.6. Kamu hizmetleri ve akıllı şehirler

Yapay zekâ (YZ) teknolojileri, kamu hizmetlerinin sunumunda ve akıllı şehirlerin yönetiminde önemli dönüşümler sağlamaktadır. Özellikle, YZ destekli sistemler sayesinde şehir yönetimleri, trafik akışını optimize edebilmekte, enerji tüketimini izleyebilmekte ve çevresel verileri analiz ederek sürdürülebilirlik hedeflerine katkıda bulunabilmektedir (Wolniak ve Stecuła, 2024).

Türkiye'de, Çevre ve Şehircilik Bakanlığı tarafından yayımlanan "2020-2023 Ulusal Akıllı Şehirler Stratejisi ve Eylem Planı" kapsamında, YZ tabanlı uygulamaların kamu hizmetlerine

entegrasyonu teşvik edilmektedir. Bu plan, veri ve uzmanlığa dayalı karar destek sistemleri aracılığıyla, vatandaşlara daha etkili ve verimli hizmet sunumunu amaçlamaktadır (Ün, 2022).

Ancak, YZ'nin kamu hizmetlerinde kullanımı bazı zorlukları da beraberinde getirmektedir. Özellikle, veri gizliliği, şeffaflık ve hesap verebilirlik gibi etik konular, YZ uygulamalarının etkinliğini sınırlayabilmektedir. Bu nedenle, kamu kurumları ve düzenleyici otoriteler, YZ uygulamalarının etik standartlara uygunluğunu sağlamak için çeşitli politikalar ve düzenlemeler geliştirmektedir (Değirmenci, 2024).

2.2.4.7. Medya, sosyal ağlar ve dijital kültürde yapay zekâ

Yapay zekâ (YZ) teknolojileri, medya, sosyal ağlar ve dijital kültür alanlarında köklü dönüşümlere yol açmaktadır. Özellikle, YZ destekli algoritmalar, sosyal medya platformlarında kullanıcı deneyimlerini kişiselleştirerek içerik akışlarını şekillendirmekte ve bireylerin dijital etkileşimlerini yönlendirmektedir (Bilgici, 2023).

Medya sektöründe, YZ'nin haber üretim süreçlerine entegrasyonu, içerik oluşturma ve dağıtımında verimliliği artırmakta, ancak bu durum aynı zamanda etik ve güvenilirlik konularında yeni tartışmaları da beraberinde getirmektedir (Erdem, 2021). YZ'nin medya ve yayıncılık alanına etkisi üzerine yapılan analizler, bu teknolojinin sektördeki rolünü ve potansiyel risklerini ortaya koymaktadır.

Dijital kültür bağlamında, YZ'nin sanatsal üretim süreçlerine katkısı dikkat çekmektedir. Örneğin, YZ destekli sistemler, sanat eserlerinin oluşturulmasında yaratıcı süreçleri desteklemekte ve yeni ifade biçimlerinin ortaya çıkmasına olanak tanımaktadır (Anantrasirichai ve Bull, 2020). Bu durum, sanatın üretim ve tüketim dinamiklerini yeniden şekillendirmekte ve dijital kültürün evriminde önemli bir rol oynamaktadır.

Ancak, YZ'nin medya ve dijital kültür alanlarındaki etkileri, aynı zamanda bazı zorlukları da beraberinde getirmektedir. Özellikle, algoritmik önyargılar, veri gizliliği ve etik sorumluluklar gibi konular, YZ'nin bu alanlardaki uygulamalarında dikkate alınması gereken önemli hususlardır (Bilgici, 2023; Erdem, 2021).

2.2.5. Yazılım testlerinde kalite ve yapay zekâ

Yazılım test süreçleri, yazılım geliştirme yaşam döngüsünün kritik bir aşamasını oluşturmakta ve ürün kalitesinin sağlanmasında önemli bir rol oynamaktadır. Geleneksel test yöntemleri, manuel müdahalelere dayandığından zaman alıcı ve hata yapmaya açık olabilir. Bu bağlamda, yapay zekâ (YZ) ve makine öğrenimi (MÖ) tekniklerinin entegrasyonu, test süreçlerinin otomasyonu ve etkinliğinin artırılması açısından büyük bir potansiyel sunmaktadır (Baqar ve Khanda, 2024).

YZ destekli test araçları, geçmiş test verilerini analiz ederek otomatik test senaryoları oluşturabilir, test kapsamını genişletebilir ve hata tespitini daha hassas bir şekilde gerçekleştirebilir. Özellikle, derin öğrenme ve doğal dil işleme teknikleri, test senaryolarının otomatik olarak oluşturulması ve yazılım gereksinimlerinin analiz edilmesi süreçlerinde etkin bir şekilde kullanılmaktadır (Ramadan vd., 2024).

Ancak, YZ'nin yazılım test süreçlerine entegrasyonu bazı zorlukları da beraberinde getirmektedir. Özellikle, yüksek kaliteli eğitim verilerine duyulan ihtiyaç, model şeffaflığı ve otomasyon ile insan denetimi arasındaki denge gibi konular, YZ tabanlı test sistemlerinin etkinliğini sınırlayabilir (Baqar ve Khanda, 2024). Bu nedenle, YZ'nin yazılım test süreçlerinde etkin bir şekilde kullanılabilmesi için bu zorlukların üstesinden gelinmesi gerekmektedir.

2.2.5.1. Yapay zekâ destekli testlerde kalite göstergeleri

Yapay zekâ destekli yazılım test süreçleri, geleneksel test yöntemlerinin ötesine geçerek daha dinamik, öğrenebilen ve öngörü yapabilen bir test altyapısı sunmaktadır. Ancak bu altyapının başarısının değerlendirilmesinde kullanılacak kalite göstergeleri, klasik test metriklerinin ötesinde, yapay zekâya özgü parametreleri de içerecek şekilde çok boyutlu olarak ele alınmalıdır (Panichella, 2021).

Yapay zekâ destekli testlerin kalitesini ölçmek için sadece test kapsamı ya da hataları bulma oranı gibi metrikler yeterli olmayıp, aynı zamanda modelin doğruluğu, genelleme yeteneği, veri kalitesi ve model önyargısı (bias) gibi faktörler de dikkate alınmalıdır. Bu bağlamda, başlıca kalite göstergeleri şu şekilde sıralanabilir:

- Model Güvenilirliği ve Genellenebilirlik (Generalizability): Eğitim verileri dışında karşılaşılan senaryolarda modelin başarı düzeyi, test sisteminin güvenilirliğini doğrudan etkiler (Amershi vd., 2019).

- Modelin Açıklanabilirliği (Explainability): Yapay zekâ modellerinin karar alma süreçlerinin şeffaf olması, hem kalite güvence ekiplerinin değerlendirmelerini kolaylaştırır hem de sistem güvenilirliğini artırır (Amershi vd., 2019).

- Model Doğruluğu (Accuracy ve Precision/Recall): AI algoritmasının test senaryolarını doğru üretme ve hataları saptama başarısı bu göstergelerle ölçülür. Özellikle hata tahmini veya veri sınıflandırmada precision-recall dengesi büyük önem taşır (Panichella, 2021).

- Test Kapsamı (Test Coverage): Yapay zekâ destekli testlerin sistem gereksinimlerini ne derece kapsadığı, test kalitesinin temel göstergelerindendir. Derin öğrenme modelleri bazı durumlarda kapsamı sınırlı tutabilir, bu nedenle kapsama alanı özel olarak analiz edilmelidir.

- Veri Kalitesi ve Temsiliyeti: Kullanılan veri setlerinin güncelliği, çeşitliliği ve etiketleme doğruluğu, model performansını etkileyen önemli faktörlerdendir.

- Hata Yakalama Oranı (Defect Detection Rate): AI destekli testlerin hata tespit etme oranı, geleneksel test yöntemleriyle karşılaştırmalı olarak değerlendirilmelidir.

- Zaman ve Kaynak Verimliliği: Otomatikleştirilmiş yapay zekâ tabanlı test sistemlerinin geliştirme süreçlerinde zaman kazandırma potansiyeli, kaliteye dolaylı katkı sağlar.

Sonuç olarak, yapay zekâ destekli test sistemlerinde kalite göstergeleri hem klasik yazılım test metriklerini hem de AI tabanlı sistemlerin performans kriterlerini bütüncül olarak kapsamalıdır. Bu sayede geliştirilen sistemlerin hem teknik doğruluğu hem de saha uygulamalarındaki başarısı daha etkili şekilde değerlendirilebilir.

2.2.5.2. Yapay zekânın kalite ölçüm metodolojilerine katkısı

Yazılım test süreçlerinde kalite ölçümü, genellikle hata yoğunluğu, test kapsama oranı ve geri bildirim temelli kalite metrikleri gibi istatistiksel verilere dayanarak gerçekleştirilmektedir. Ancak bu geleneksel yöntemler, büyük ve dinamik veri setleriyle çalışan, sürekli evrilen yazılım projelerinde yeterince esnek ve öngörücü olamamaktadır. Bu noktada yapay zekâ (YZ), geçmiş test verilerini ve hata örüntülerini analiz ederek kalite ölçüm süreçlerine hem derinlemesine içgörü hem de kestirimci (predictive) analiz yetenekleri kazandırmaktadır. Böylece YZ destekli sistemler, yalnızca mevcut kalite düzeyini değerlendirmekle kalmayıp, olası riskleri önceden öngörerek proaktif müdahale imkânı da sunmaktadır (Jordan ve Mitchell, 2015).

YZ tabanlı sistemler, geçmiş test verilerini, hata raporlarını ve kullanıcı davranışlarını analiz ederek yazılımın gelecekteki hata olasılıklarını tahmin edebilmekte, bu da yazılım kalitesine yönelik daha stratejik kararların alınmasını sağlamaktadır. Kalite ölçüm sürecine katkı sağlayan bazı temel YZ yaklaşımları şunlardır:

- Tahmine Dayalı Kalite Değerlendirmeleri (Predictive Quality Assessment): Makine öğrenmesi algoritmaları, geçmiş hata kayıtları ve test metrikleri ile eğitilerek bir yazılım bileşeninin gelecekte hata üretme olasılığını hesaplayabilir (Panichella, 2021). Bu sayede kritik bileşenlere öncelik verilebilir ve kaynaklar daha verimli kullanılabilir.

- Sınıflandırma Tabanlı Hata Türü Analizi: Destek vektör makineleri (SVM) veya karar ağaçları gibi algoritmalar, tespit edilen hataları otomatik olarak kategorilere ayırarak kalite raporlamasını hızlandırır ve standardize eder.

- Doğal Dil İşleme (NLP) ile Geliştirilmiş Geri Bildirim Analizi: Kullanıcı yorumları, hata bildirimleri veya geliştirici açıklamaları gibi doğal dildeki metinler analiz edilerek yazılım kalitesine dair anlamlı göstergeler çıkarılabilir.

- Zaman Serisi ve Anomali Tespiti: Test verilerindeki zamana bağlı kalite değişimleri veya olağandışı hata artışları, zaman serisi analizi ve anomali tespiti algoritmaları kullanılarak belirlenebilir. Bu sayede erken uyarı sistemleri geliştirilebilir ve test süreçlerinde anlık müdahaleler sağlanabilir (Batarseh vd., 2021).

Yapay zekâ, bu katkıları sayesinde kalite ölçüm sürecini yalnızca geriye dönük bir analizden çıkararak proaktif bir izleme ve yönlendirme aracına dönüştürmektedir. Bu da hem yazılım güvenilirliğini artırmakta hem de toplam test maliyetlerini düşürmektedir.

2.2.5.3. Akademik literatürde yapay zekâ ile kalite iyileştirme yaklaşımları

Yapay zekâ tekniklerinin yazılım testi ve kalite güvencesi alanında kullanılmasına yönelik akademik çalışmalar son yıllarda artış göstermektedir. Bu çalışmalar hem test süreçlerinin otomasyonu hem de yazılım kalitesinin iyileştirilmesi açısından önemli katkılar sunmaktadır. Literatürde öne çıkan yaklaşımlar; makine öğrenmesi tabanlı hata tahmini, test önceliklendirme, otomatik test senaryosu üretimi ve doğal dil işleme destekli geri bildirim analizleri gibi alanlara yoğunlaşmaktadır.

Makine öğrenmesi tabanlı hata tahmini, yazılım bileşenlerinde hata oluşma olasılığını geçmiş veriler üzerinden öngörmeyi amaçlamaktadır. Panichella (2021), bu yöntemin, test kaynaklarının daha verimli kullanılmasına imkân tanıdığını ve yüksek riskli modüllere öncelik verilmesini sağladığını ifade etmiştir.

Destek vektör makineleri (SVM) ve Random Forest gibi sınıflandırma algoritmaları, hatalı kod bloklarını tespit etmede sıklıkla kullanılmaktadır. Zhang ve ark. (2020), bu modellerin yazılım hatalarını %80'e varan doğrulukla tahmin edebildiğini ve manuel test yükünü önemli ölçüde azalttığını vurgulamıştır.

Otomatik test senaryosu üretimi ise, yapay zekâ algoritmalarının kullanıcı hikâyeleri, kod tabanı veya hata kayıtlarını analiz ederek test senaryoları oluşturması sürecidir. Gao ve ark. (2019), bu yöntemin çevik geliştirme döngülerinde test sürekliliğini sağladığını belirtmiştir.
Doğal dil işleme (NLP) teknikleri, özellikle kullanıcı geri bildirimlerinden anlamlı kalite göstergeleri çıkarılmasında kullanılmaktadır. Hindle ve ark. (2016), hata raporları ve yorumlardan elde edilen metin analizlerinin, yazılımın kullanıcı memnuniyeti açısından zayıf yönlerini ortaya koyabildiğini göstermiştir.

Bu literatür çalışmalarından da anlaşılacağı üzere, yapay zekâ uygulamaları sadece test süreçlerini otomatikleştirmekle kalmayıp, aynı zamanda yazılım kalitesine dair daha bütüncül,

veri temelli bir bakış açısı geliştirilmesine olanak tanımaktadır. Böylece yazılım mühendisliği süreçleri hem daha öngörülebilir hem de daha sürdürülebilir bir yapıya kavuşmaktadır.

2.2.6. Makine öğrenmesi

Makine öğrenmesi (MÖ), sistemlerin açıkça programlanmaksızın verilerden örüntüler öğrenerek karar verebilmesini sağlayan bir yapay zekâ yaklaşımıdır. İnsan öğrenme sürecine benzer şekilde, MÖ algoritmaları da önceki deneyimlerden yola çıkarak yeni durumlara yönelik tahminler geliştirebilir. Bu süreçte model, bir veri seti üzerinden eğitilir; ardından sınıflandırma, kümeleme veya tahminleme gibi görevleri gerçekleştirebilecek duruma gelir (Gezmez, 2022). Özellikle büyük ve karmaşık veri kümeleriyle çalışılması gereken alanlarda MÖ, karar verme süreçlerini hızlandırmakta ve insan müdahalesini azaltarak hataları en aza indirmektedir (Keleş vd., 2020).

Yazılım test süreçleri bağlamında ele alındığında, makine öğrenmesi teknikleri hata tahmini, test veri üretimi, test önceliklendirme ve kalite değerlendirmesi gibi pek çok alanda etkili biçimde uygulanmaktadır. Bu sayede test süreçleri daha hızlı, otomatik ve esnek bir yapıya kavuşmaktadır (Meriç ve Özbayoğlu, 2021). MÖ tabanlı sistemler, geçmiş test sonuçları ve hata örüntülerini analiz ederek yeni gelen projelerdeki test faaliyetlerini sınıflandırabilir ve kaliteye dair kestirimlerde bulunabilir. Böylece yalnızca zaman ve kaynak tasarrufu sağlanmakla kalmaz, aynı zamanda içerik temelli test yaklaşımı sayesinde daha stratejik kalite öngörüleri yapılabilir (Zhang vd., 2020).

Makine öğrenmesi yöntemleri genel olarak üç temel yaklaşıma ayrılmaktadır: gözetimli öğrenme, gözetimsiz öğrenme ve pekiştirmeli öğrenme. Gözetimli öğrenmede algoritmalar, etiketlenmiş veriler üzerinden örüntüleri öğrenerek gelecekteki veriler için tahminlerde bulunur. Bu yöntem genellikle hata tahmini ve yazılım bileşenlerinin sınıflandırılması gibi görevlerde kullanılır. Gözetimsiz öğrenme ise etiketsiz verilerdeki yapısal benzerlikleri ve örüntüleri keşfetmeye odaklanır; kümelendirme (clustering) ve anomali tespiti bu yaklaşımın temel uygulamaları arasındadır. Pekiştirmeli öğrenmede ise model, çevreden aldığı ödül veya ceza sinyalleri doğrultusunda stratejik kararlar almayı öğrenir; yazılım test senaryosu üretimi ve regresyon testi gibi alanlarda bu yaklaşım ön plana çıkmaktadır (Onan ve Korukoğlu, 2016).

Sonuç olarak makine öğrenmesi, yalnızca geçmiş veriler üzerinden öğrenme yeteneği kazandırmakla kalmayıp; aynı zamanda yazılım test süreçlerinin daha tutarlı, ölçülebilir ve tahmin edilebilir hâle gelmesine katkı sunmaktadır. Bu yönüyle, yazılım kalitesini doğrudan etkileyen test faaliyetlerinin sınıflandırılması ve kalite tahmini gibi görevlerde MÖ temelli yaklaşımların kullanımı oldukça stratejik bir değer taşımaktadır.

2.2.6.1. Makine öğrenmesi türleri

Makine öğrenmesi yöntemleri, öğrenme biçimlerine göre temel olarak üç ana kategoriye ayrılmaktadır: gözetimli öğrenme, gözetimsiz öğrenme ve pekiştirmeli öğrenme. Bu sınıflandırma, modelin öğrenme sürecinde kullandığı veri yapısına ve geri bildirim mekanizmasına göre yapılmaktadır.

- Gözetimli Öğrenme (Supervised Learning): Gözetimli öğrenme, verilerin giriş ve çıkışlarının önceden tanımlandığı durumlarda kullanılan bir öğrenme yaklaşımıdır. Bu yöntemde model, etiketli verilerden faydalanarak girdi-çıktı ilişkisini öğrenir ve yeni girdilere karşılık gelebilecek çıktıları tahmin etmeye çalışır. Örneğin, bir yazılım modülünün daha önceki test sonuçlarına göre hatalı mı hatasız mı olduğu belirtilmişse, model bu örneklerden öğrenerek yeni bir modülün durumunu tahmin edebilir (Aydın ve Özkul, 2015). Yazılım testinde, hata türü sınıflandırması veya kod kalitesi tahmininde sıkça tercih edilir (Panichella, 2021).

- Gözetimsiz Öğrenme (Unsupervised Learning): Bu yaklaşımda, veriler herhangi bir etiket içermemektedir. Model, veriler arasındaki benzerlikleri veya örüntüleri kendiliğinden keşfetmeye çalışır. Özellikle verilerde gizli kümeleri belirlemek veya alışılmadık davranışları tespit etmek için kullanılır. Yazılım test süreçlerinde, benzer hata örüntülerinin gruplandırılması ya da anomali (sıra dışı durum) tespiti için uygundur. Örneğin, bir e-ticaret sisteminde sepete eklenen ürünler analiz edilerek kullanıcı davranışları kümelere ayrılabilir (Keskin ve Yıldız, 2018). Yazılım test verilerinde de benzer yaklaşımlar uygulanarak test senaryoları otomatik sınıflandırılabilir (Zhang vd., 2019).

- Pekiştirmeli Öğrenme (Reinforcement Learning): Pekiştirmeli öğrenme, modelin çevreyle etkileşimi sonucunda ödül ya da ceza gibi geri bildirimlerle öğrenmesini sağlayan bir yöntemdir. Model, hangi eylemlerin daha fazla ödül sağladığını deneyim yoluyla keşfeder.

Bu yaklaşımda doğrudan doğruya bir etiketli veri sunulmaz; sistem kendi hatalarından ve başarılarından öğrenir. Yazılım testinde, özellikle test senaryosu üretimi ve test akışlarının dinamik olarak düzenlenmesi gibi alanlarda kullanılabilir. Bu yöntem, bir çocuğun yürümeyi öğrenirken düşe kalka gelişim göstermesine benzetilebilir (Efe, 2021; Zhang, J. M. vd., 2019).

2.2.6.2. Makine öğrenmesi yöntem ve algoritmaları

Makine öğrenmesi, yazılım test süreçlerinin otomatikleştirilmesi, hataların erken tespiti ve yazılım kalitesinin öngörülmesi gibi birçok alanda etkin biçimde kullanılmaktadır. Bu teknikler, özellikle büyük miktarda test verisinin analiz edilmesi ve bu verilerden öğrenilen kalıplar aracılığıyla gelecekteki test senaryolarına yönelik çıkarımlar yapılması açısından önem arz etmektedir. Yazılım projelerinde geçmişte yaşanan test sonuçları, hatalar ve kalite metrikleri gibi veriler, makine öğrenmesi algoritmalarıyla işlenerek yeni durumlar için tahmin modelleri oluşturulabilir.

Bu bağlamda, Zhang ve arkadaşları (2020) tarafından yapılan bir çalışmada, yazılım bileşenlerinin hataya yatkınlıkları çeşitli makine öğrenmesi algoritmaları ile başarıyla tahmin edilmiş ve yazılım kalitesinin artırılmasına katkı sağlanmıştır. Benzer şekilde Panichella (2021), test veri setlerinin öğrenilmesiyle otomatik test senaryolarının oluşturulabildiğini ve bu sayede manuel test yükünün önemli ölçüde azaldığını ortaya koymuştur. Son olarak, Gao, ve arkadaşları (2019) tarafından geliştirilen sistemde, geçmiş test kayıtları analiz edilerek gelecekteki test adımlarının otomatik önerileri yapılabilmiş ve test sürekliliği sağlanmıştır.

Bu çalışmada da benzer biçimde, makine öğrenmesine dayalı yöntemler aracılığıyla yazılım test verilerinin sınıflandırılması ve yazılım kalitesine dair öngörülerde bulunulması hedeflenmektedir. Denetimli öğrenme temelli bu algoritmalar; hata eğilimlerinin belirlenmesi, riskli bileşenlerin saptanması ve daha etkili test stratejilerinin geliştirilmesi gibi birçok açıdan katkı sağlamaktadır.

2.2.7. Makine öğrenmesi algoritmaları

2.2.7.1. Lojistik regresyon (Logistic regression)

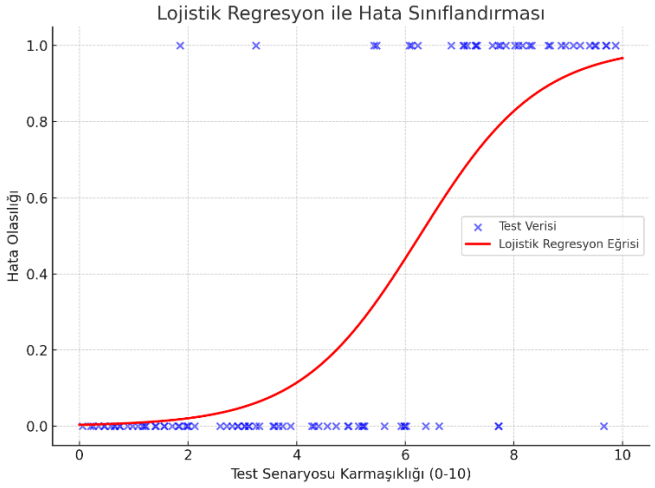
Lojistik regresyon, ikili (binary) ya da çok sınıflı (multinomial) sınıflandırma problemlerinde yaygın olarak kullanılan, istatistiksel bir modelleme yöntemidir. Bu yöntem, bağımlı

değişkenin olasılık değerini tahmin etmeye çalışır ve çıktılar genellikle "evet/hayır", "başarılı/başarısız" gibi kategorik değerlerdir. Doğrusal regresyondan farklı olarak, lojistik regresyonda çıktı sürekli değil; sınıflandırmaya yönelik olasılık değeri (0 ile 1 arasında) olarak ele alınır.

Modelin temelinde, doğrusal regresyon ile hesaplanan değerin bir sigmoid (lojistik) fonksiyon aracılığıyla dönüştürülmesi yer alır. Bu sayede tahmin edilen değerler, bir sınıfa ait olma olasılığı şeklinde yorumlanabilir. Sigmoid fonksiyonun matematiksel ifadesi şu şekildedir:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n)}} \quad (2.1)$$

Lojistik regresyon, özellikle sınıflandırma gerektiren yapay zekâ uygulamalarında; örneğin bir yazılım test senaryosunun "hatalı/hatasız" olarak etiketlenmesi ya da kullanıcı davranışlarının "riski var/yok" şeklinde ayrıştırılması gibi durumlarda sıklıkla tercih edilmektedir. Yazılım test süreçlerinde de test örneklerinin hata içerip içermediği gibi ikili karar gerektiren analizlerde etkili bir yöntemdir. Bu model sayesinde test verileri, geçmişte gözlemlenen sonuçlara göre sınıflandırılabilir ve hata olasılığı yüksek test örnekleri önceliklendirilebilir.



Şekil 1. Lojistik Regresyon Modeli (Hosmer vd., 2013)

Şekil 1'de görüldüğü gibi, karmaşıklık düzeyi arttıkça hata olasılığı önemli ölçüde artmaktadır. Bu durum, test senaryolarının önceliklendirilmesinde sayısal modellerin önemini ortaya koymaktadır.

Yüksek hesaplama verimliliği ve yorumlanabilirliği sayesinde lojistik regresyon, daha karmaşık makine öğrenmesi algoritmalarına geçmeden önce referans model olarak da kullanılmaktadır. Ayrıca, model çıktısı doğrudan olasılık temelli olduğundan, karar eşikleri ayarlanarak sistemin ihtiyacına göre esneklik sağlanabilir.

Lojistik regresyonun başarı düzeyi, değişkenler arasındaki ayrım gücüne ve model varsayımlarına olan uygunluğa bağlıdır. Özellikle çoklu değişkenli durumlarda, değişkenler arası korelasyon dikkatle analiz edilmeli ve gerektiğinde boyut indirgeme teknikleriyle desteklenmelidir (Hosmer vd., 2013).

2.2.7.2. Doğrusal regresyon (Linear regression)

Doğrusal regresyon (Linear Regression), bağımlı bir değişken ile bir veya birden fazla bağımsız değişken arasındaki doğrusal ilişkiyi modellemek için kullanılan temel bir istatistiksel yöntemdir. En sade formu olan basit doğrusal regresyonda, tek bir bağımsız değişkenin etkisi incelenirken, çoklu doğrusal regresyon modelinde birden fazla değişken bu etkiye dahil edilir. Modelin genel formu:

$$\boldsymbol{Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \epsilon} \qquad (\mathbf{2}.2)$$

Şeklinde olup; burada Y bağımlı değişkeni, $X_i$bağımsız değişkenleri, β_i regresyon katsayılarını ve ϵ hata terimini temsil eder.

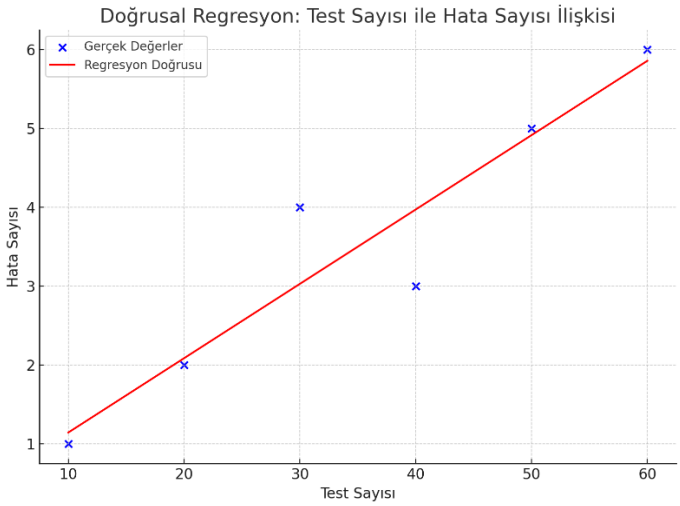
Bu yöntem, değişkenler arasındaki ilişkinin derecesini sayısal olarak ifade etmesi sayesinde hem açıklayıcı hem de tahmin edici analizlerde sıkça tercih edilmektedir. Doğrusal regresyonun avantajlarından biri, modelin yorumlanabilirliğinin oldukça yüksek olmasıdır. Her bir bağımsız değişkenin çıktıya olan katkısı doğrudan katsayılar üzerinden anlaşılabilir. Ancak bu yöntemin başarılı sonuçlar verebilmesi için bazı varsayımların sağlanması gerekir: doğrusal ilişki, sabit varyans (homoskedastisite), hata terimlerinin normalliği ve değişkenler arasında çoklu bağlantının (multicollinearity) düşük olması gibi.

Yazılım testi alanında doğrusal regresyon modelleri, test sonuçlarını etkileyen faktörlerin sayısal olarak analiz edilmesinde ve yazılım kalitesine etki eden unsurların öngörülmesinde kullanılmaktadır. Örneğin; bir yazılım projesinde hata sayısının, test kapsamı, geliştirici sayısı veya önceki sürümdeki hata oranları gibi bağımsız değişkenler ile açıklanması mümkündür. Bu

sayede kaliteyi düşüren etmenler sayısallaştırılarak geliştiricilere ve test mühendislerine yol gösterici veriler sağlanabilir.

Şekil 2'de gösterildiği üzere, doğrusal regresyon modeli kullanılarak test sayısı ile hata sayısı arasındaki ilişki incelenmiştir. Aşağıdaki grafik, hipotetik bir yazılım test projesine ait test sayısı (bağımsız değişken) ile tespit edilen hata sayısı (bağımlı değişken) arasındaki doğrusal ilişkiyi modellemektedir. Grafikte kırmızı çizgi, en küçük kareler yöntemi ile elde edilen regresyon doğrusunu temsil etmektedir.

Model sonuçları, test sayısının artmasıyla birlikte hata sayısının da arttığını göstermektedir. Bu doğrusal ilişki, yazılım test süreçlerinde kapsamın genişletilmesinin potansiyel hata tespitini artırabileceğini ve bu nedenle test kapsamının yazılım kalitesi açısından kritik bir unsur olduğunu desteklemektedir.



Şekil 2. Doğrusal Regresyon Modeli (James vd., 2021).

Doğrusal regresyon aynı zamanda makine öğrenmesi algoritmalarının temel yapıtaşlarından biridir. Basitliği, hesaplama maliyetinin düşüklüğü ve hızlı uygulanabilirliği nedeniyle, denetimli öğrenme modellerinde özellikle başlangıç analizi olarak sıkça kullanılmaktadır (James vd., 2021).

Sonuç olarak, doğrusal regresyon analizi hem teorik hem de pratik uygulamalarda etkili bir yöntem olarak öne çıkmakta; yazılım mühendisliği, kalite analizi ve hata tahmininde geniş bir kullanım alanı bulmaktadır.

2.2.7.3. K-en yakın komşu (K-nearest neighbors – KNN)

K-En Yakın Komşu (K-Nearest Neighbors – KNN) algoritması, örnek tabanlı öğrenmeye dayanan, denetimli bir makine öğrenmesi yöntemidir. İlk olarak Cover ve Hart (1967) tarafından önerilen bu yöntem, sınıflandırma ve regresyon görevlerinde yaygın biçimde kullanılmaktadır. Modelin temel prensibi, sınıflandırılması gereken bir gözlemin, eğitim veri kümesindeki "K" adet en yakın komşusunun sınıfına göre tahmin edilmesidir. Bu işlem sırasında genellikle Öklid mesafesi kullanılır; ancak Manhattan, Minkowski ve Mahalanobis gibi farklı mesafe ölçütleri de tercih edilebilir.

KNN, "tembel öğrenme" (lazy learning) yaklaşımını benimseyen bir algoritmadır. Yani klasik öğrenme algoritmalarından farklı olarak model eğitimi süreci yoktur. Bunun yerine tüm öğrenme işlemi, tahmin yapılacağı anda gerçekleşir. Bu özellik, algoritmanın eğitim süresini minimize ederken, tahmin sürecinde daha yüksek hesaplama yükü oluşturabilir (Mitchell, 1997).

KNN algoritmasının sınıflandırma başarısı, bazı hiperparametrelerin doğru şekilde seçilmesine bağlıdır:

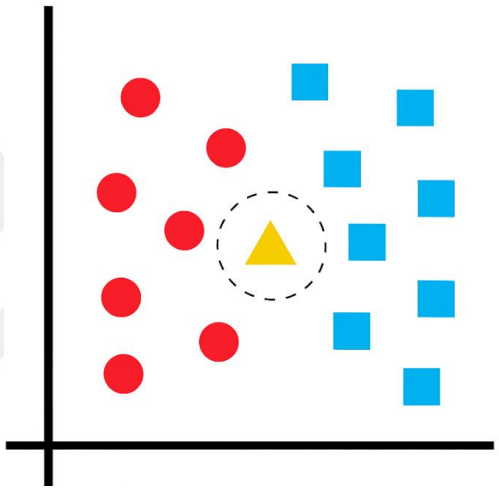
- K değeri: Kaç komşunun dikkate alınacağı; küçük K değerleri gürültüye duyarlıdır, büyük değerler ise fazla genellemeye neden olabilir.
- Mesafe ölçütü: Öklid, Manhattan, Minkowski gibi farklı metriklerin seçimi, verinin dağılımına bağlı olarak sonuçları etkiler.
- Ağırlıklı mesafe: Komşuların sınıflandırma kararına etkisi eşit mi olacak, yoksa yakınlıklarına göre ağırlıklandırılacak mı? Bu da modelin performansını etkileyen bir tercihtir.

Bu parametrelerin belirlenmesinde genellikle çapraz doğrulama (cross-validation) yöntemi kullanılır. Doğru parametre kombinasyonları ile optimize edilen KNN modelleri, özellikle düşük boyutlu ve iyi ayrılmış veri kümelerinde yüksek başarı sağlar (Umar vd., 2019).

KNN algoritması, yalnızca sınıflandırma değil, regresyon problemleri için de uyarlanabilir. Bu durumda, bir gözleme en yakın K komşunun hedef değişken değerleri ortalanarak tahmin yapılır. KNN regresyonu, veri dağılımının gözlemlenebilir olduğu ve ilişkilerin doğrusal olmadığı durumlarda da etkili sonuçlar verebilir.

Mitchell (1997), KNN'yi örnek tabanlı öğrenme yöntemleri arasında değerlendirerek, yorumlanabilirlik, sadelik ve geniş uygulama alanı nedeniyle makine öğrenmesinde güçlü bir alternatif olarak tanımlamaktadır.

Şekil 3'te, iki farklı sınıfa ait veri noktalarının KNN algoritması kullanılarak nasıl sınıflandırıldığını gösteren bir örnek sunulmaktadır. Bu örnekte, mavi renkli noktalar birinci sınıfa, kırmızı renkli noktalar ise ikinci sınıfa aittir. Sarı üçgenle temsil edilen yeni bir veri noktası, K=3 parametresiyle en yakın üç komşuya göre değerlendirilmiştir. Bu üç komşunun ikisi kırmızı, biri mavi sınıfa ait olduğundan, yeni veri noktası kırmızı sınıfa ait olarak sınıflandırılmıştır. KNN algoritması bu kararı, en yakın komşuların sınıf dağılımına göre verir.



Şekil 3. K-En Yakın Komşu Algoritması İçin İki Sınıflı Örnek

Bu tür bir yaklaşımda, sisteme yeni eklenen veri ile mevcut veriler arasındaki mesafe hesaplamaları Öklid, Manhattan, Minkowski veya Hamming gibi fonksiyonlar aracılığıyla yapılır.

- Öklid Uzaklığı: Öklid uzaklığı, iki nokta arasındaki doğrusal (geometrik) mesafeyi ölçen klasik bir yaklaşımdır. Genellikle sayısal veri kümelerinde tercih edilir. n-boyutlu bir uzayda iki vektör arasındaki mesafe aşağıdaki şekilde hesaplanır:

$$\boldsymbol{d_E(x,y) = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2}} \qquad (\mathbf{2}.3)$$

- Manhattan Uzaklığı: Manhattan (city-block) uzaklığı, iki nokta arasındaki mesafenin, eksenlere paralel adımların toplamı olarak tanımlanır. Özellikle sınıflandırma problemlerinde, veri değerlerinin dağılımına bağlı olarak Öklid'e alternatif olarak tercih edilir:

$$\boldsymbol{D = \sqrt{\sum_{i=1}^{k}}(x_i - y_i)^2} \qquad (\mathbf{2}.4)$$

- Minkowski Uzaklığı: Minkowski uzaklığı, Öklid ve Manhattan uzaklıklarının genelleştirilmiş bir versiyonudur. p parametresi farklı değerler aldığında diğer metriklere dönüşür. Genel formülü şu şekildedir:

$$\boldsymbol{D = \left(\sum_{i=1}^{k}\right)|x_i - y_i|^p)^{\frac{1}{p}}} \qquad (\mathbf{2}.5)$$

- Hamming Uzaklığı: Hamming uzaklığı, özellikle kategorik ya da ikili (binary) veri kümelerinde kullanılır. İki eşit uzunluktaki dizide karşılık gelen öğelerden farklı olanların sayısını ölçer. Bu mesafe, sıklıkla metin verilerinin ya da ikili özniteliklerin karşılaştırılmasında kullanılır:

$$\boldsymbol{D_H = \sum_{i=1}^{k}|x_i - y_i|} \qquad (\mathbf{2}.6)$$

KNN algoritmasında sınıflandırma işlemini etkileyen temel unsurlardan biri mesafe ölçüm yöntemleriyken, bir diğer önemli unsur ise "k" parametresinin doğru biçimde belirlenmesidir. Mesafe metrikleri veriler arası benzerliği ölçerken, k değeri ise karar sürecinde kaç komşunun dikkate alınacağını tanımlar.

Bu parametrenin uygun seçilmemesi, modelin sınıflandırma doğruluğunu olumsuz etkileyebilir. Çok küçük bir k değeri modelin gürültüye duyarlı olmasına yol açarken, çok büyük bir değer modelin aşırı genelleme yapmasına neden olabilir. Genellikle pratikte, k değeri veri kümesindeki örnek sayısının karekökü ya da öznitelik sayısının yaklaşık yarısı olarak önerilse de, en ideal değer veri setine özgü olarak belirlenmelidir.

Bu belirleme sürecinde yaygın olarak deneme-yanılma yöntemi veya çapraz doğrulama (cross-validation) stratejileri kullanılır. Deneme-yanılma yaklaşımında, farklı k değerleriyle model eğitilir ve her bir k için test verisi üzerindeki hata oranı gözlemlenerek en düşük hatayı veren değer seçilir. Öte yandan, çapraz doğrulama yönteminde veri kümesi eşit büyüklükte alt

parçalara bölünür; her bir parça sırasıyla test seti olarak kullanılır ve diğer parçalarla model eğitilir.

Bu işlemin sonucunda elde edilen ortalama hata oranları dikkate alınarak en uygun k değeri seçilir. Ayrıca veri setinin yapısı, sınıflar arası dağılım dengesi ve veri boyutu da k parametresi üzerinde belirleyici olabilir. Büyük veri kümelerinde daha yüksek k değerleri tercih edilebilirken, bu durum modelin duyarlılığını azaltabilir.

Özetle, k parametresinin belirlenmesi yalnızca sayısal denemelerle değil, aynı zamanda alan bilgisi ve uygulama hedefleri göz önünde bulundurularak yapılmalıdır. KNN algoritmasının basitliği ve yorumlanabilirliği avantaj sağlarken, performans açısından bu tür parametre ayarlarının özenle gerçekleştirilmesi gerekmektedir.

2.2.7.4. Naive bayes

Naive Bayes, istatistiksel temelli ve denetimli bir öğrenme algoritmasıdır. Temelinde Bayes teoremi yer alır ve sınıflandırma problemlerine basit ama etkili çözümler sunar. Özellikle metin madenciliği, spam filtreleme, duygu analizi ve sağlık verisi sınıflandırması gibi uygulama alanlarında sıklıkla tercih edilmektedir (Saylan ve Çınaroğlu, 2024).
Bayes teoremi, bir olayın belirli koşullar altında gerçekleşme olasılığını hesaplamaya dayanır. Genel formül aşağıdaki gibidir:

$$\boldsymbol{P(S_i \mid X) = \frac{P(X \mid S_i).P(S_i)}{P(X)}} \qquad \boldsymbol{(2.7)}$$

Burada:

- $P(S_i \mid X)$: X örneğinin S_i sınıfına ait olma olasılığıdır.
- $P(X \mid S_i)$: S_i sınıfına ait örneklerin X özelliklerini taşıma olasılığıdır.
- S_i: S_i sınıfının önsel olasılığıdır.
- P(X): Gözlemlenen X değerinin olasılığıdır.

Naive Bayes'in en belirgin varsayımı, tüm özniteliklerin birbirinden bağımsız olduğudur. Gerçek hayattaki verilerde bu varsayım her zaman geçerli olmasa da özellikle büyük veri kümelerinde düşük hesaplama maliyeti ve hızlı sonuç üretmesi nedeniyle yaygın olarak kullanılmaktadır (Baktir ve Atay, 2022).

Naive Bayes algoritmasının çeşitli türleri bulunmaktadır:

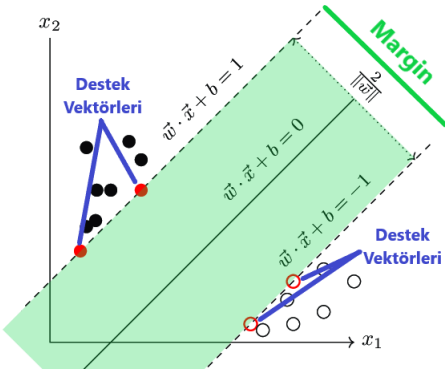
- Gaussian Naive Bayes: Sürekli özellikler için normal dağılım varsayımı yapar.
- Multinomial Naive Bayes: Kelime frekanslarının analiz edildiği metin sınıflandırma problemlerinde etkilidir.
- Bernoulli Naive Bayes: Özelliklerin ikili (var/yok) olarak tanımlandığı veri kümelerinde kullanılır.

Son yıllarda yapılan çalışmalar, Naive Bayes algoritmasının özellikle metin tabanlı veri setlerinde tatmin edici başarı oranlarına sahip olduğunu göstermektedir. Örneğin, spam e-posta sınıflandırma problemi üzerinde yapılan karşılaştırmalı analizlerde, Naive Bayes'in KNN ve Karar Ağaçları gibi yöntemlerle benzer düzeyde performans sergilediği ortaya konmuştur (Çınar, 2024; Baktir ve Atay, 2022).

Ancak, algoritmanın temelindeki "özellik bağımsızlığı" varsayımı nedeniyle, öznitelikler arasında yüksek korelasyon bulunduğunda sınıflandırma başarısı olumsuz etkilenebilir. Bu nedenle, veri ön işleme aşamasında dikkatli öznitelik seçimi yapılması önerilmektedir.

2.2.7.5. Destek vektör makineleri (Support vector machines – SVM)

Destek Vektör Makineleri (Support Vector Machines – SVM), denetimli öğrenme yaklaşımına dayalı güçlü ve esnek bir sınıflandırma algoritmasıdır. Bu yöntem, farklı sınıflara ait örnekleri birbirinden en iyi şekilde ayırabilecek hiper düzlemi (decision boundary) belirlemeyi hedefler. SVM'nin temel stratejisi, sınıflar arasında maksimum marjini sağlayacak karar sınırını bulmaktır. Bu sayede, model hem sınıflandırma başarısını artırır hem de genelleme kabiliyetini güçlendirir (Ayhan ve Erdoğmuş, 2014).



Şekil 4. İki Sınıfın SVM Hiper Düzlemi (Ayhan ve Erdoğmuş, 2014)

Veri kümeleri doğrusal olarak ayrılabilir olduğunda, birden fazla olası hiper düzlem çizilebilir. SVM, bu seçenekler arasından en geniş marjine sahip olanı seçerek optimal ayrımı sağlar. Ancak pek çok gerçek dünya problemi doğrusal bir şekilde sınıflandırılamaz. Bu tür durumlarda, SVM algoritması çekirdek (kernel) fonksiyonları yardımıyla veri noktalarını daha yüksek boyutlu bir uzaya dönüştürür. Böylece, giriş uzayında doğrusal olarak ayrılamayan örnekler, dönüştürülmüş uzayda doğrusal biçimde ayrılabilir hale gelir (Kavzoğlu ve Çölkesen, 2010). SVM'de yaygın olarak kullanılan çekirdek fonksiyonları şunlardır:

- Doğrusal çekirdek:

$$\mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i} \cdot \mathbf{x_j} \qquad (\mathbf{2.8})$$

- Polinom çekirdek:

$$\mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = (\boldsymbol{\gamma}(\mathbf{x_i} \cdot \mathbf{x_j}) + \mathbf{c})\mathbf{d} \qquad (\mathbf{2.9})$$

- Sigmoid çekirdek:

$$\mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{tanh}(\boldsymbol{\gamma}(\mathbf{x_i} \cdot \mathbf{x_j}) + \mathbf{c}) \qquad (\mathbf{2.10})$$

- Radyal Tabanlı Fonksiyon (RBF):

$$\mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{exp}(-\boldsymbol{\gamma}|\mathbf{x_i} - \mathbf{x_j}|\mathbf{2}) \qquad (\mathbf{2.11})$$

SVM'nin başarısı yalnızca çekirdek fonksiyonu seçimine değil, aynı zamanda hiperparametrelerin uygun ayarlanmasına da bağlıdır. C parametresi, sınıflandırma hatalarına

karşı ne kadar tolerans gösterileceğini belirler. Daha yüksek C değeri daha az hata toleransı sağlar ancak aşırı öğrenmeye (overfitting) yol açabilir. Gamma (γ) parametresi ise bir örneğin ne kadar geniş bir etki alanına sahip olacağını kontrol eder. Bu parametrelerin doğru kombinasyonu, modelin genel performansını önemli ölçüde etkiler (Zavuoğlu ve Çölkesen, 2010).

SVM, küçük ve orta ölçekli veri setlerinde yüksek doğruluk oranlarıyla çalışırken; aynı zamanda karmaşık karar sınırlarına sahip sınıflandırma problemlerinde de etkili çözümler sunar. Ayrıca, yalnızca sınıflandırma değil, regresyon analizi (SVR) ve aykırı değer tespiti gibi diğer makine öğrenmesi uygulamalarında da yaygın şekilde kullanılmaktadır (Meral ve Saraçlı, 2020).

SVM'nin temel başarısı, sınıfları ayıran karar sınırının, her iki sınıfa ait en yakın veri noktalarına (destek vektörlerine) olan mesafeyi maksimize etmesinden gelir. Bu strateji, modelin sadece eğitim verisine değil, aynı zamanda yeni örneklere karşı da daha yüksek doğrulukla genelleme yapabilmesini sağlar.

2.2.7.6. Rassal orman yöntemi (Random forest)

Rassal Orman (Random Forest, RF) algoritması, Leo Breiman tarafından geliştirilen ve karar ağaçları temelli bir topluluk (ensemble) öğrenme yaklaşımıdır. Bu yöntem, her biri eğitim verisinin rastgele bir alt kümesiyle eğitilen çok sayıda karar ağacından oluşur ve nihai tahmin, ağaçların çoğunluk oyuna göre yapılır (Breiman, 2001). Her ağaç, özelliklerin rastgele seçilen bir alt kümesi kullanılarak inşa edilir; bu rastgelelik, modelin genelleme gücünü artırırken, ağaçlar arası korelasyonu azaltır (Haddouchi ve Berrado, 2024).

Son yıllarda RF algoritması, çeşitli alanlardaki başarılı uygulamalarıyla dikkat çekmektedir. Sağlık alanında yapılan bir çalışmada, RF yöntemi kullanılarak EEG verileri üzerinden bipolar bozukluğa sahip bireylerin sınıflandırılması gerçekleştirilmiş ve %93'ün üzerinde doğruluk elde edilmiştir (Pettorruso vd., 2023).

Eğitim alanında ise algoritma, öğrencilerin akademik başarısını etkileyen faktörleri tahmin etmede kullanılmış ve öğrenme yönetim sistemleri üzerinden anlamlı öngörüler sunmuştur (Hu, Zhang ve Chen, 2023).

Finans sektöründe yapılan araştırmalar da RF algoritmasının kredi risk değerlendirmelerinde geleneksel yöntemlere kıyasla daha yüksek doğruluk sunduğunu ortaya koymuştur (Kuyoro, Sanusi ve Ajayi, 2022).

RF yönteminin temel avantajları; yüksek boyutlu verilerle başa çıkabilme, eksik verilerle çalışma esnekliği, aşırı değerlere karşı dayanıklılık ve değişken öneminin belirlenebilmesidir (Van der Laan ve Rose, 2022). Öte yandan, çok sayıda karar ağacının bir araya gelmesiyle oluşan yapı, yorumlanabilirliği azaltmakta ve modelin karar verme sürecini anlamayı zorlaştırmaktadır (Haddouchi ve Berrado, 2024).

Yapı içerisindeki her karar ağacı, veri setinin rastgele seçilen alt kümeleri üzerinde eğitilir. Ayrıca, her düğümde bölünme için dikkate alınan özellikler de rastgele seçilir. Bu rastgelelik, modelin aşırı öğrenmesini (overfitting) önlemeye yardımcı olur. Rassal Orman, hangi özelliklerin tahmin yaparken en önemli olduğunu belirlemeye yardımcı olabilir. Bu, modelin hangi girdilerin çıktıyı en çok etkilediğini anlamamıza olanak tanır. Bu yöntem, eksik verilere ve aşırı değerlere karşı oldukça sağlamdır. Ayrıca, modelin veri setindeki gürültüye karşı toleranslı olmasını sağlar. Yöntemin başarısı, ağaç sayısı, özellik sayısı ve ağaç derinliği gibi hiperparametrelerin uygun bir şekilde ayarlanmasına bağlıdır.

Rassal Orman yöntemi, geniş bir uygulama alanına sahiptir ve özellikle karmaşık veri yapılarına sahip problemlerde etkilidir. Ancak, modelin yorumlanabilirliği sınırlıdır çünkü çok sayıda karar ağacının birleşimi nedeniyle sonuçların arkasındaki mantığı anlamak zor olabilir.

2.2.7.7. Yapay sinir ağları

Yapay Sinir Ağları (YSA), insan beynindeki sinir hücrelerinin (nöronların) çalışma prensibinden esinlenerek geliştirilen, çok katmanlı ve çok bağlantılı matematiksel modellerdir. Bu yapılar, girdi verilerini işleyerek belirli örüntüleri öğrenebilmekte ve bu öğrenme sürecine dayanarak yeni veriler üzerinde tahminlerde bulunabilmektedir. Özellikle doğrusal olmayan ve karmaşık veri yapılarının modellenmesinde yüksek performans göstermeleri, YSA'ları modern makine öğrenmesi uygulamalarında vazgeçilmez hâle getirmiştir.
Bir yapay sinir ağı genel olarak; giriş katmanı, bir veya birden fazla gizli (hidden) katman ve çıkış katmanından oluşur. Her katmandaki düğümler (nöronlar), kendilerine gelen verileri

belirli bir aktivasyon fonksiyonu (örneğin sigmoid, ReLU) aracılığıyla işler. Ağırlıklar ve bias değerleri, öğrenme sürecinde güncellenerek modelin çıktılara daha isabetli tepkiler vermesi sağlanır.

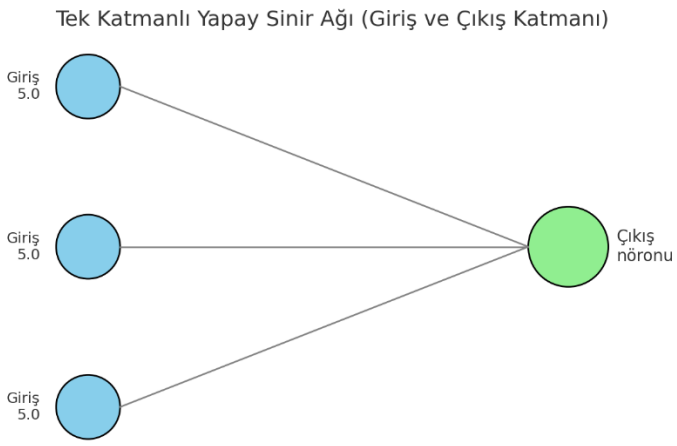
YSA'ların en önemli avantajlarından biri, etiketli büyük veri kümeleri ile eğitildiklerinde karmaşık ilişkileri yüksek doğrulukla öğrenebilmeleridir. Bu nedenle görüntü işleme, doğal dil işleme ve zaman serisi tahmini gibi alanlarda olduğu kadar, yazılım testi süreçlerinde de yaygın biçimde kullanılmaktadır. Özellikle yazılım sistemlerinden elde edilen log verileri, test sonuçları ve hata geçmişi gibi çok boyutlu ve hacimli veri kümeleri, yapay sinir ağları tarafından işlenerek; hata tahmini, test önceliklendirme ve otomatik test senaryosu üretimi gibi çeşitli çıktılar elde edilebilmektedir.

Yazılım test süreçlerinde YSA'nın sağladığı katkılar, test örneklerinin geçmiş performansına göre sınıflandırılması, sistem bileşenlerinin hata eğilimlerinin tahmin edilmesi ve manuel testlerin otomatikleştirilmesi gibi çeşitli kullanım senaryoları ile somutlaşmaktadır. Özellikle yazılım projelerinin büyüklüğü ve karmaşıklığı arttıkça, geleneksel yöntemlerin yetersiz kaldığı durumlarda YSA modelleri alternatif ve güçlü bir çözüm sunmaktadır.

Ancak YSA modellerinin eğitimi yüksek işlem gücü gerektirmekte ve öğrenme sürecinde kullanılan hiperparametrelerin dikkatle ayarlanmasını zorunlu kılmaktadır. Model karmaşıklığı arttıkça açıklanabilirliğin azalması ise, bu yöntemlerin "kara kutu" olarak eleştirilmesine neden olmaktadır (Goodfellow vd., 2016). Bu nedenle yazılım testinde kullanılacak YSA modellerinin, mümkün olduğunca sadeleştirilmiş yapılarla oluşturulması ve açıklanabilirlik kriterlerinin göz önünde bulundurulması önemlidir.

2.2.7.7.1. Tek katmanlı yapay sinir ağları

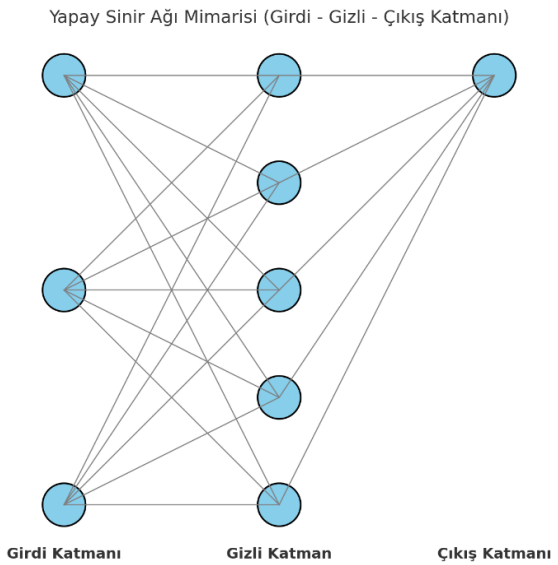
Yapay sinir ağlarının en basit formunu oluşturan tek katmanlı yapılar, yalnızca bir giriş katmanı ve doğrudan bir çıkış katmanından oluşmaktadır; bu ağlarda gizli katman bulunmaz. Sadece doğrusal olarak ayrılabilir problemlerin çözülebilmesine olanak tanır. Eğitim süreci hızlı olup, hesaplama maliyeti düşüktür. Ancak doğrusal olmayan ilişkiler söz konusu olduğunda yeterli performansı gösteremezler. Bu nedenle, yazılım testi gibi karmaşık örüntüler içeren uygulama alanlarında başarı düzeyleri sınırlıdır.



Şekil 5. Tek Katmanlı YSA

2.2.7.7.2. Çok katmanlı yapay sinir ağları

Giriş ve çıkış katmanları arasında bir ya da daha fazla gizli katman içeren çok katmanlı yapılar, doğrusal olmayan ilişkileri modelleyebilme yeteneğine sahiptir. Her gizli katman, önceki katmandan aldığı veriyi işleyerek bir sonraki katmana aktarır ve bu sayede çok katmanlı ağlar, veri içerisindeki yüksek boyutlu ve karmaşık örüntüleri başarılı şekilde öğrenebilir. Yazılım test verileri gibi çok değişkenli ve karmaşık yapılara sahip veri setlerinde bu modellerin üstün performans sergilediği görülmektedir.



Şekil 6. Çok Katmanlı YSA

Çok Katmanlı Algılayıcı (Multi-Layer Perceptron – MLP)

Yapay sinir ağlarının en yaygın kullanılan türlerinden biri olan Çok Katmanlı Algılayıcı (MLP), ileri beslemeli (feedforward) yapıya sahiptir. Giriş verileri, sırasıyla bir veya birden fazla gizli katmandan geçirilerek çıkış katmanına ulaşır. MLP modelleri, geriye yayılım (backpropagation) algoritmasını kullanarak hata değerlerini minimize etmek amacıyla ağ ağırlıklarını sürekli olarak günceller.

Yazılım testi bağlamında, MLP modelleri test senaryolarının başarılı ya da başarısız sonuçlarını sınıflandırmak, hata eğilimi yüksek modülleri tahmin etmek ve kalite kontrol süreçlerini otomatikleştirmek amacıyla yaygın olarak kullanılmaktadır.

- Toplama Fonksiyonunun Rolü: Sinir ağlarında her nöron, kendisine gelen sinyalleri belirli ağırlıklarla çarpar ve bunları toplar. Bu işlem "toplama fonksiyonu" aracılığıyla gerçekleştirilir. Genellikle lineer bir fonksiyon olan bu yapı, aktivasyon fonksiyonuna aktarılmadan önce girişlerin ağırlıklı toplamını hesaplar. Bu toplam, modelin öğrenme kapasitesi üzerinde doğrudan etkilidir.

- Yaygın Aktivasyon Fonksiyonları: Aktivasyon fonksiyonları, bir nöronun çıktısını belirlemek için kullanılır. En sık kullanılanlar arasında sigmoid, tanh ve ReLU (Rectified Linear Unit) fonksiyonları yer alır.

  1. Sigmoid: Çıktıyı 0 ile 1 arasına sıkıştırarak özellikle ikili sınıflandırmalarda kullanılır.
  Formülü:

$$\boldsymbol{f(x) = \frac{1}{1 + e^{-x}}} \qquad \textbf{(2.12)}$$

  Bu yapı, çıkışı olasılık yorumu yapılabilecek bir aralığa getirir. Ancak büyük pozitif veya negatif girişlerde türevi çok küçük olduğundan, geri yayılımda gradyanların sönmesine neden olabilir.

2. ReLU: Negatif değerleri sıfıra eşitleyip pozitif değerleri olduğu gibi alır; derin ağlarda tercih edilir çünkü öğrenme hızını artırır.

Formülü:

$$\boldsymbol{f(x) = \max(0, x)} \qquad \boldsymbol{(2.13)}$$

ReLU, özellikle derin sinir ağlarında etkinliği sayesinde tercih edilir. Ancak negatif değerleri tamamen sıfıra indirdiği için bazı nöronların "ölmesi" riski vardır.

3. Tanh (Hiperbolik Tanjant): Çıktıyı -1 ile 1 arasında normalize eder ve merkezli yapısı nedeniyle bazı durumlarda sigmoid'e göre daha iyi sonuçlar verir.
Formülü:

$$\boldsymbol{f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} - e^{-x}}} \qquad \boldsymbol{(2.14)}$$

Tanh fonksiyonu, sigmoid'e göre daha güçlü gradyanlar sağlayabilir çünkü çıktılar -1 ile 1 arasında merkezlidir.

4. Leaky ReLU (Sızıntılı Doğrultucu Lineer Birim): Leaky ReLU, ReLU fonksiyonunun bir türevidir ve negatif değerler için sıfır yerine çok küçük bir gradyan değeri üretir. Böylece ReLU'nun en büyük dezavantajı olan "ölü nöron" (dead neuron) probleminin önüne geçilmesi hedeflenir. Özellikle derin katmanlı ağlarda bazı nöronların tamamen pasif kalmasını engellediği için performans artışı sağlar.
Formülü:

$$\boldsymbol{f(x) = max(0.001 \cdot x, x)} \qquad \boldsymbol{(2.15)}$$

Bu yapı sayesinde ağın negatif girdilere karşı duyarlılığı tamamen ortadan kalkmaz; bunun yerine çok küçük de olsa bir gradyan aktarımı devam eder (Bedi vd., 2022).

5. Softmax Fonksiyonu: Softmax, genellikle çok sınıflı sınıflandırma problemlerinde, özellikle çıkış katmanında kullanılan bir aktivasyon fonksiyonudur. Bu fonksiyon, her bir sınıfın karşılık geldiği logit (net giriş) değerini olasılık değerine çevirir ve tüm sınıfların toplam olasılığı 1 olacak şekilde normalize eder.
Formülü:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_i}} \quad (2.16)$$

Burada x_i i'inci sınıfın logit (ağırlıklı toplam) değerini; $\sum e^{x_i}$ ise tüm sınıfların üstel değerlerinin toplamını ifade eder. Softmax, özellikle çok sınıflı yazılım test senaryolarında (örneğin testlerin "başarılı", "şüpheli", "başarısız" gibi etiketlenmesinde) kullanıldığında yüksek sınıflandırma başarısı sağlar (Gao vd., 2020).

Yaygın olarak kullanılan aktivasyon fonksiyonları Tablo 1' de özetlenmiştir.

Tablo 1. Yaygın olarak kullanılan aktivasyon fonksiyonlarının özeti (Haykin, 2009)

Fonksiyon	Çıktı Aralığı	Avantajları	Sınırlılıkları
Sigmoid	(0, 1)	Olasılık yorumu yapılabilir, ikili sınıflama için uygundur.	Gradyan sönme problemi yaşanabilir.
Tanh	(-1, 1)	Çıkış merkezlidir, güçlü gradyan sağlar.	Aşırı uçlarda gradyan küçülür.
ReLU	[0, ∞)	Basit ve hesaplama açısından verimli.	Negatif girişlerde nöronlar ölebilir.
Leaky ReLU	≈(-∞, ∞)	Negatif değerlere duyarlıdır, ölü nöron problemini azaltır.	Negatif gradyanlar çok küçüktür.
Softmax	(0,1), toplamı 1	Çok sınıflı çıktılar için olasılık dağılımı sunar.	Yüksek hesaplama maliyeti.

Her fonksiyonun avantajları ve sınırlılıkları, ilgili sınıflandırma probleminin yapısına göre farklı performans gösterebilir.

Sinir ağı modellerinin başarısında önemli rol oynayan hiperparametrelerden biri olan epoch, tüm eğitim verisinin modele bir kez sunulması sürecini ifade eder. Genellikle model eğitimi, birden fazla epoch boyunca sürdürülür ve her bir epoch sonunda model, hata oranlarını azaltmak amacıyla ağırlıklarını günceller. Ancak epoch sayısının yetersiz olması öğrenmenin sınırlı kalmasına neden olurken, aşırı sayıda epoch uygulanması ise aşırı öğrenme (overfitting) riskini artırabilir.

Bu nedenle yazılım testi uygulamalarında, doğruluk ve genelleme başarısı dikkate alınarak optimal epoch sayısının belirlenmesi büyük önem taşır. Bununla birlikte, öğrenme sürecinin hassasiyetini belirleyen bir diğer parametre öğrenme katsayısıdır (learning rate). Bu katsayının yüksek olması, modelin dengesiz ve kararsız şekilde öğrenmesine; düşük olması ise öğrenmenin gereğinden fazla yavaşlamasına yol açabilir. Dolayısıyla, doğru öğrenme katsayısı seçimi modelin genel doğruluk oranını doğrudan etkiler.

Öte yandan, öğrenme sürecinin daha kararlı ilerlemesini sağlayan bir başka parametre olan momentum katsayısı, geçmiş ağırlık güncellemelerinden elde edilen yön bilgisini hesaba katarak modelin yerel minimumlara takılmadan global minimuma daha hızlı ve dengeli bir şekilde ulaşmasına katkı sağlar. Yazılım testine yönelik sinir ağı uygulamalarında, özellikle dengesiz veri kümeleriyle çalışıldığında momentumun sağladığı bu ivme, modelin kararlılığı ve doğruluğu açısından kritik bir avantaj sunmaktadır.

2.2.8. Hata metrikleri (Error metrics)

Makine öğrenmesi modellerinin doğruluğunu değerlendirmek için çeşitli hata metrikleri kullanılmaktadır. Bu metrikler, tahmin edilen değerler ile gerçek değerler arasındaki farkı sayısal olarak ifade ederek modelin performansını ölçmede önemli bir rol oynar. Bu bağlamda, özellikle regresyon temelli tahmin modellerinde sıklıkla tercih edilen dört temel hata metriği bulunmaktadır:

- 1. Ortalama Mutlak Yüzde Hata (Mean Absolute Percentage Error - MAPE): MAPE, tahminlerin gerçek değerlere göre ne oranda sapma gösterdiğini yüzde cinsinden ifade eder. Bu metrik genellikle kolay yorumlanabilir olması nedeniyle tercih edilir. Ancak gerçek değerlerin sıfıra yaklaşması durumunda metrik dengesiz sonuçlar verebilir.

$$\boldsymbol{MAPE = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|} \qquad (\mathbf{2.17})$$

Burada y_i gerçek değeri, $\hat{y}$ tahmin edilen değeri ve n toplam örnek sayısını temsil eder.

- 2. Ortalama Mutlak Hata (Mean Absolute Error - MAE): MAE, tüm tahmin hatalarının mutlak değerlerinin ortalamasını alarak modelin ortalama hata miktarını hesaplar. Basit ve anlaşılır yapısı sayesinde regresyon analizlerinde yaygın olarak kullanılır.

$$\boldsymbol{MAE = \frac{1}{n} \sum_{i=1}^{n} | y_i - \hat{y}_i |} \qquad \boldsymbol{(2.18)}$$

  Bu metrik, modelin genel hata seviyesini net bir şekilde sunar ve uç değerlerden MSE kadar etkilenmez.

- 3. Ortalama Kare Hatası (Mean Squared Error - MSE): MSE, tahmin hatalarının karelerinin ortalamasını alarak büyük hataları daha fazla cezalandıran bir metrik sunar. Bu özelliği sayesinde modelin istikrarı ve varyansı hakkında bilgi verir.

$$\boldsymbol{MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \qquad \boldsymbol{(2.19)}$$

  Modelin hatalarının karelerinin ortalaması alındığı için, tekil büyük hataların etkisi daha fazla olur.

- 4. Kök Ortalama Kare Hata (Root Mean Squared Error – RMSE): RMSE, MSE'nin karekökü alınarak elde edilir ve hata ölçüsünü orijinal veri birimleriyle sunar. Bu yönüyle yorumlanması MAE'ye benzer ancak büyük hatalara daha duyarlıdır.

$$\boldsymbol{RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}} \qquad \boldsymbol{(2.20)}$$

  Bu metrik, genellikle modelin genel hata seviyesini anlaşılır biçimde ifade etmesi açısından tercih edilmektedir.

Bu metrikler, geliştirilen modellerin hem doğruluğunu hem de güvenilirliğini ölçmek adına kritik öneme sahiptir ve bu çalışmada oluşturulan kalite tahmin modellerinin değerlendirilmesinde etkin olarak kullanılmıştır.

3. YÖNTEM

Bu tez çalışmasında, Türkiye'de faaliyet gösteren bir yazılım firmasında yürütülen test faaliyetlerinin sınıflandırılması ve yazılım kalite tahmini yapılması amacıyla yapay zekâ temelli bir yaklaşım benimsenmiştir. Çalışmanın temelinde, yazılım projelerinde gerçekleştirilen test uygulamalarının kalitesine ilişkin algının ölçülmesi ve bu algının yapay zekâ algoritmaları ile tahminlenebilir hale getirilmesi yer almaktadır. Bu doğrultuda, test mühendisleri ve yazılım uzmanlarının sürece dair deneyimlerinden yararlanılarak çok boyutlu bir veri seti oluşturulmuş ve bu veri seti üzerinden analizler gerçekleştirilmiştir.

Veriler, dijital ortamda uygulanan ve üç bölümden oluşacak şekilde yapılandırılmış bir anket aracılığıyla toplanmıştır. Anketin ilk bölümünde, katılımcıların görev aldıkları projelerdeki demografik ve mesleki profilleri belirlenmiştir. İkinci bölümde, projelerin genel çerçevesi çizilerek kullanılan geliştirme metodolojileri, tercih edilen yazılım test türleri, kullanılan yazılım dilleri, test araçları ve karşılaşılan zorluklara ilişkin bilgiler elde edilmiştir. Üçüncü bölümde ise, projelerde yürütülen yazılım test faaliyetleri ve yazılım kalite düzeyine dair değerlendirmeler beşli likert ölçeği kullanılarak ölçülmüş; katılımcılardan, test edilen yazılımların genel kalite düzeyini "çok yüksek", "yüksek", "orta", "düşük" ve "çok düşük" olarak derecelendirmeleri istenmiştir. Ayrıca proje başarısına yönelik olarak "Bu projede bulunan hataların ne kadarını tespit ettiğinizi düşünüyorsunuz?" ve "Bu projeyi genel olarak ne kadar başarılı buluyorsunuz?" gibi sorularla, test faaliyetlerinin yazılım üzerindeki etkisi nicel olarak sorgulanmıştır.

Anketin hazırlanma sürecinde sektörel dil farklılıkları ile teknik terimlerin anlaşılırlığı gözetilmiş; bu kapsamda özgün olarak hazırlanan sorular, bir ofiste görev yapan 15 test mühendisi üzerinde pilot bir uygulama ile test edilmiştir. Pilot uygulamadan elde edilen geri bildirimler doğrultusunda anket sorularının formatı gözden geçirilerek revize edilmiş; çoktan seçmeli ifade biçimleri eklenmiş ve böylece bir projede birden fazla yöntemin eşzamanlı olarak kullanılabilme durumu da kapsam altına alınmıştır. Bu düzenlemelerle, anketin hedef kitle tarafından açık, tutarlı ve doğru biçimde anlaşılması amaçlanmıştır. Pilot aşamanın ardından gerçekleştirilen asıl uygulama ile toplam 90 katılımcıdan veri toplanmıştır. Ancak bazı soruların çoktan seçmeli ve birden fazla yanıt içermesi nedeniyle, makine öğrenmesi algoritmalarının veri yapısına uygunluk sağlamak amacıyla satır çoğaltma (data augmentation) tekniği uygulanmış; her çoklu yanıt, tekil bir gözlem satırı olarak dönüştürülerek veri setindeki

toplam veri sayısı artırılmıştır. Bu yöntemle proje kapsamı sabit tutulurken, aynı projede birden fazla yöntemin uygulanabildiği durumlar ayrıştırılarak modele girdi olarak sunulmuş ve böylece sınıflandırma performansı ile analiz derinliği güçlendirilmiştir.

Elde edilen veriler üzerinde çeşitli veri ön işleme adımları gerçekleştirilmiştir. Bu adımlar arasında boş hücrelerin işlenmesi, metinsel ifadelerin temizlenmesi ve özellikle çoklu yanıtların analiz edilebilir hale getirilmesi amacıyla satır çoğaltmaları uygulanmıştır. Ardından, kategorik veriler Bayrow formülü kullanılarak sayısal formatlara dönüştürülmüş ve makine öğrenmesi algoritmalarına uygun hale getirilmiştir. Veri setinin güvenilirliğini değerlendirmek amacıyla Cronbach's Alpha katsayısı hesaplanarak verilerin içsel tutarlılığı sağlanmıştır.

Hazırlanan sayısal veri seti üzerinde Python programlama dili ve ilgili veri analizi kütüphaneleri kullanılarak çeşitli makine öğrenmesi teknikleri uygulanmıştır. Bu algoritmalar yardımıyla, geçmiş yazılım projelerindeki test yapılarına ilişkin başarı düzeyleri analiz edilerek, yeni projelerin test süreçlerine ilişkin kalite tahminlerinin yapılması hedeflenmiştir.
Modelleme sürecinde eğitim ve test verileri ayrılmış, her bir algoritma üzerinde çapraz doğrulama teknikleri uygulanarak başarım değerlendirmesi yapılmıştır.

Sonuç olarak bu yöntem, yazılım test süreçlerinin veriye dayalı şekilde sınıflandırılmasına ve öngörülebilir kalite modellerinin oluşturulmasına olanak tanımakta; yazılım firmalarının daha etkin test stratejileri geliştirmesine katkı sağlamaktadır.

3.1. Veri Toplama

Bu araştırmada ihtiyaç duyulan veriler, bir yazılım firmasında görev yapan yazılım test mühendisleri ve ilgili uzmanlara yöneltilen kapsamlı bir dijital anket aracılığıyla toplanmıştır. Anket formu, katılımcıların görev aldıkları projelere ilişkin olarak kullanılan geliştirme metodolojilerini, tercih edilen test tekniklerini, uygulanan yazılım test türlerini, ekip büyüklüğünü, proje süresini ve süreçte karşılaşılan zorlukları ayrıntılı şekilde sorgulamaktadır. Ayrıca proje içi iletişim, müşteri memnuniyeti ve genel başarı algısı gibi kaliteyi doğrudan etkileyen unsurlar da çoktan seçmeli ve kapalı uçlu sorularla ölçülmüştür. Katılımcılardan, kendi deneyimlerine dayanarak yer aldıkları projelerin genel yazılım kalite düzeyini de "çok yüksek"ten "çok düşük"e kadar uzanan beşli likert ölçeğiyle derecelendirmeleri istenmiştir. Böylece birden fazla boyutu kapsayan bu verilerden oluşan veri seti, projelerin teknik

özellikleri, yönetim süreçleri ve test faaliyetleri ile kalite algısı arasındaki ilişkilerin ayrıntılı biçimde analiz edilmesine imkân tanımaktadır.
Anket soruları ve seçeneklerin tamamı, tezin sonunda yer alan Ek A'da sunulmuştur.

Anket, Erzincan Binali Yıldırım Üniversitesi Fen ve Mühendislik Bilimleri Etik Kurulu'ndan gerekli etik onay alınarak dijital ortamda uygulanmıştır. Anket formunun hazırlanma aşamasında sektörel dil farklılıkları, teknik terimlerin katılımcılar tarafından doğru anlaşılması ve kavramsal tutarlılık unsurları dikkate alınarak sınırlı sayıda test mühendisiyle pilot bir uygulama gerçekleştirilmiş; elde edilen geri bildirimler doğrultusunda formda gerekli düzenlemeler yapılmıştır.

Veri toplama süreci sonucunda toplam 90 katılımcıdan geçerli yanıtlar elde edilmiş ve analiz aşamasına dahil edilmiştir.

3.2. Ön İşleme ve Dönüştürme

Veri toplama süreci sonrasında elde edilen ham veriler, analiz öncesinde çeşitli ön işleme adımlarından geçirilmiştir. Anketin bazı soruları çoktan seçmeli yapıda olduğundan, birden fazla seçenek işaretleyen katılımcıların yanıtları analiz edilebilir hale getirilmek üzere satır çoğaltma yöntemiyle yeniden yapılandırılmıştır. Bu işlemde her bir çoklu yanıt, ayrı bir gözlem satırı olarak ele alınmış ve veri setinde çoğaltılmıştır. Böylece başlangıçta 90 satırdan oluşan veri seti, kademeli çoğaltmalarla 3.849 analiz edilebilir satır içeren bir veri kümesine dönüştürülmüştür.

Veri çoğaltma işleminin tamamlanmasının ardından, veri setinde yer alan kategorik değişkenler sayısal forma dönüştürülmüştür. Bu dönüştürme sürecinde Bayrow formülü esas alınmış; katılımcıların yanıtları önceden tanımlı sayısal karşılıklara eşlenerek analiz edilebilir hale getirilmiştir. Örneğin, "çok düşük" yanıtı 1 puan, "çok yüksek" yanıtı ise 5 puan şeklinde ölçeklendirilmiştir. Bu sayede makine öğrenmesi algoritmalarının işleyebileceği, temiz ve yapılandırılmış bir veri seti oluşturulmuştur.

Hazırlanan bu veri kümesi, çalışmanın sonraki aşamalarında makine öğrenmesi tekniklerinin uygulanmasına temel oluşturmuş; test faaliyetlerinin sınıflandırılması ve yazılım kalite tahminine yönelik algoritmaların eğitilmesinde kullanılmıştır.

3.3. Veri Güvenilirliği ve Geçerleme

Veri setinin içsel tutarlılığını ve ölçüm aracının güvenirliğini değerlendirmek amacıyla Cronbach Alpha (α) katsayısı hesaplanmıştır. (Cronbach, 1951) Cronbach Alpha, özellikle anket formunda yer alan çok maddeli ölçüm yapılarında, maddelerin birbiriyle olan iç tutarlılık düzeyini belirlemek için kullanılan yaygın ve güvenilir bir istatistiksel yöntemdir.

Bu katsayı, aşağıdaki formül ile hesaplanmaktadır:

$$\alpha = \frac{k}{k-1}\left(1 - \frac{\Sigma_{i=1}^{k} \sigma_{Y_i}^2}{\sigma_X^2}\right) \qquad (3.1)$$

Burada:

- k: Ölçekteki madde sayısı,
- $\sigma_{Y_i}^2$: Her bir maddenin varyansı,
- σ_X^2: Toplam ölçek puanlarının varyansı

Cronbach Alpha katsayısı 0 ile 1 arasında bir değer alır. Değerin 0.70'in üzerinde olması, ölçeğin güvenilir olduğuna; 0,80 ve üzerindeki değerler ise yüksek güvenirliğe işaret etmektedir (Tavşancıl, 2002).

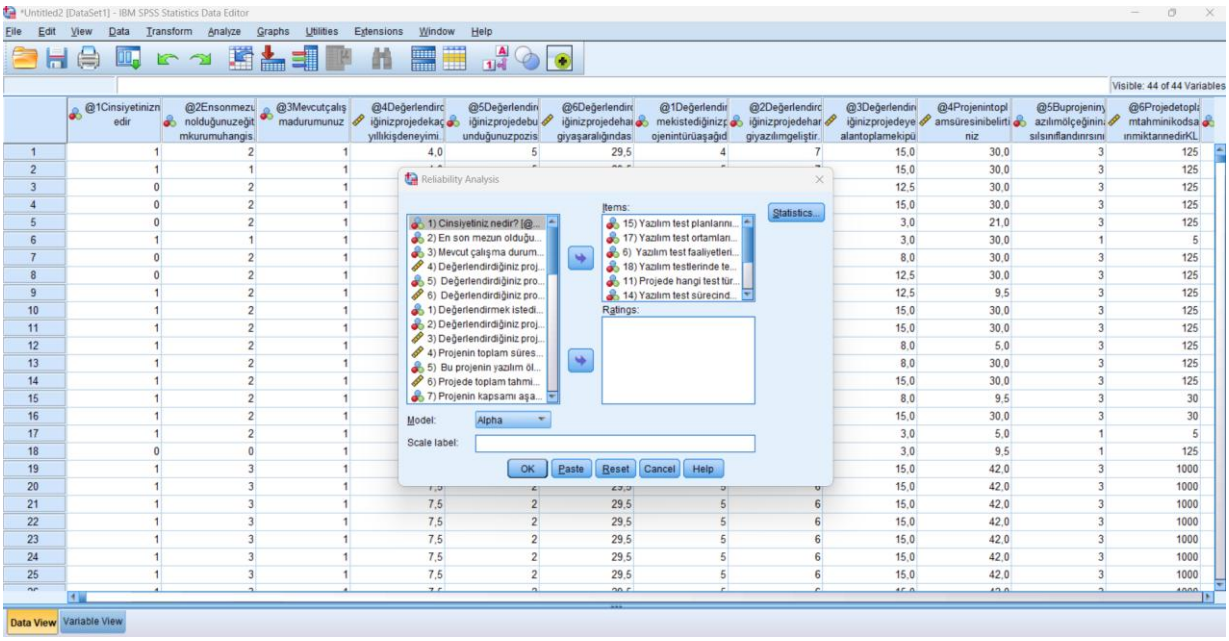
Bu çalışmada, anket sorularının büyük çoğunluğu yazılım test süreçlerinin yazılım kalitesi üzerindeki etkilerini bütüncül biçimde ölçmeyi amaçlayan, aynı kavramsal çatı altında gruplanmış maddelerden oluşmaktadır. Cronbach's Alpha iç tutarlılık katsayısı hesaplanırken, Ek A'da yer alan anket soruları arasından kalite boyutunu doğrudan, ölçülebilir ve nicel veriye dönüştürülebilir şekilde temsil eden toplam 10 madde seçilmiştir. Seçilen maddeler; planlama, hata tespiti, otomasyon, raporlama, süre yönetimi gibi test sürecinin temel bileşenlerini kapsamaktadır. Bu kapsamda, proje bazlı yanıt farklılıklarını azaltmak, test süreçlerinin bütüncül kalitesini homojen bir ölçekte yansıtmak ve analizde kavramsal tutarlılık sağlamak amacıyla, açıkça nicel değerlendirme imkânı sunan sorular tercih edilmiştir. Ölçekte yer almayan diğer sorular ise proje genel bilgisi, yöntem tercihi veya tanımlayıcı nitelikte olduğundan, iç tutarlılık katsayısına anlamlı katkı sağlamayacağı değerlendirilmiştir. Böylece

seçilen sorular, ölçeğin güvenilirliğini güçlendirmekte ve modelleme sürecinde anlamlı bir bağımsız değişken kümesi oluşturmaktadır.

Seçilen maddeler aşağıda sıralanmıştır:

- Yazılım test planlarının proje gereksinimlerini karşılayacak düzeyde olma derecesi
- Yazılım test ortamlarının uygunluğu ve kullanılabilirliği
- Test caselerinde bulunan kritik hata tespit oranı
- Tekrarlanan hataların sayısı
- Test faaliyetlerinin zamanında tamamlama oranı (hedefe göre)
- Test sürecindeki otomasyonun etkinliği
- Test caselerinin raporlanmasının projeye etkisi
- Test caseleri sayısı ve hata oranlarının geliştirmeye etkisi
- Test faaliyetlerinin planlanan süre içinde tamamlanma oranı
- Test faaliyetlerinin genel kalitesi (doğruluk, kapsam, etkinlik, hata tespiti)

Bu doğrultuda güvenirlik analizi, istatistiksel analizlerde yaygın olarak kullanılan SPSS (Statistical Package for the Social Sciences) yazılımı ile gerçekleştirilmiştir. SPSS arayüzünde "Analyze> Scale > Reliability Analysis" adımları izlenmiş; yukarıda belirtilen maddeler "Items" olarak seçilerek, Cronbach's Alpha katsayısı hesaplanmıştır. Analizler sırasında örneklem büyüklüğü (N=3849), veri yapısı ve ölçek türleri göz önünde bulundurulmuş; yalnızca geçerli (eksiksiz) veriler değerlendirmeye alınmıştır. Bu süreçte kullanılan SPSS arayüzüne ilişkin örnek Şekil 7'de sunulmuştur.

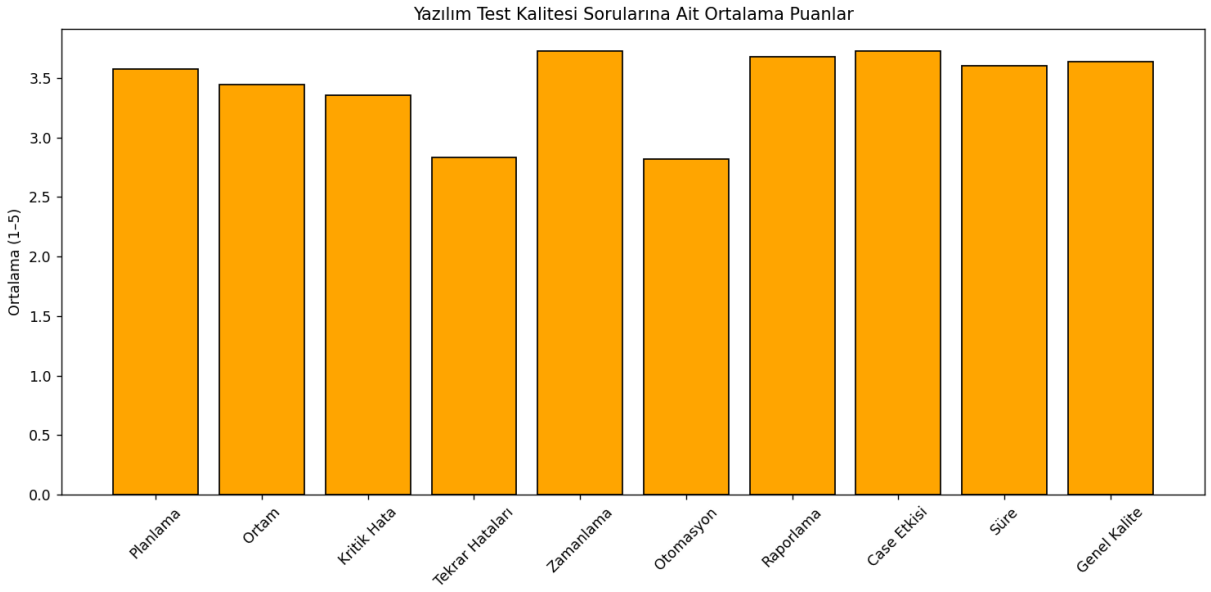


Şekil 7. SPSS ile Cronbach's Alpha Güvenirlik Analizi Ekranı

Yapılan analiz sonucunda elde edilen Cronbach Alpha katsayısı 0.760 olarak bulunmuştur. Bu sonuç, analizde kullanılan ölçeğin yeterli düzeyde iç tutarlılığa sahip olduğunu ve veri setinin modelleme süreci için güvenilir bir temel sunduğunu göstermektedir.

Bu çalışmada, yazılım test süreçlerinin kaliteye etkisini ölçmeye yönelik seçilen 10 bağımsız değişkenin beşli likert ölçeğine dayalı ortalama puanları hesaplanmış ve çubuk grafik formatında görselleştirilmiştir. Grafik, Python programlama dili kullanılarak hazırlanmış; veri işleme ve görselleştirme aşamalarında yaygın olarak kullanılan pandas (McKinney, 2010) ve matplotlib (Hunter, 2007) kütüphanelerinden yararlanılmıştır.

Bu görsel analiz, test mühendislerinin yazılım test süreçlerine ilişkin kalite algılarını bütüncül biçimde yansıtarak, hangi test unsurlarının daha yüksek ya da daha düşük düzeyde değerlendirildiğini açıkça göstermektedir. Elde edilen bulgular, modelleme sürecinde öncelikli alanların belirlenmesi açısından yol gösterici niteliktedir ve karşılaştırmalı eğilimler Şekil 8'de detaylı biçimde sunulmaktadır.



Şekil 8. Yazılım Test Kalitesi Sorularına Ait Ortalama Değer Grafiği

Şekil 8'de her bir çubuk, ilgili anket maddesine katılımcılar tarafından verilen puanların ortalamasını temsil etmektedir. Böylece yazılım test süreçlerinin hangi boyutlarının daha yüksek veya düşük algılandığı somut biçimde ortaya konmakta; bu veriler, modelleme aşamasına yön verecek önemli ipuçları sağlamaktadır.

3.4. Veri Analizi İçin Programlama Dili Seçimi

Veri analiz süreci, yalnızca istatistiksel testlerle sınırlı kalmamış; aynı zamanda veri görselleştirme, ön işleme ve sayısallaştırma gibi çok boyutlu işlemleri de kapsamıştır. Bu nedenle analiz sürecinin her aşamasında esnek, modüler ve güçlü araçlara ihtiyaç duyulmuştur. Anket verilerinin güvenirlik analizleri ve temel istatistiksel değerlendirmeleri için SPSS yazılımı tercih edilmiştir.

Bununla birlikte, Python programlama dili yalnızca veri işleme ve modelleme değil, aynı zamanda temel grafiksel çıktılar aracılığıyla verinin daha iyi yorumlanmasını da mümkün kılmıştır. Veri analizi sürecine katkı sağlayan bu görsel yaklaşımlar, özellikle test süreçlerine ilişkin algıların anlaşılmasında kullanılmıştır. Python'un açık kaynak yapısı ve pandas, matplotlib gibi kütüphaneleri sayesinde veri görselleştirme işlemleri etkili biçimde gerçekleştirilmiş; bu sayede analizlerin yorumlanabilirliği artırılmıştır.

Bu kapsamda kullanılan veri seti, 3.2. Ön İşleme ve Dönüştürme bölümünde ayrıntılı şekilde açıklandığı üzere çoğaltma ve sayısallaştırma adımlarından geçirilmiş; çoktan seçmeli yanıtlar yeniden yapılandırılmış ve Bayrow formülü kullanılarak sayısal karşılıklar atanmıştır. Bu ön işleme süreci sayesinde veri, Python tabanlı analiz tekniklerine uygun, temiz ve yapılandırılmış bir forma dönüştürülmüştür.

Sonuç olarak, bu çalışmada Python programlama dili ve SPSS yazılımı birlikte kullanılarak hem istatistiksel güvenirlik testleri hem de ileri düzey veri analizi süreçleri başarıyla yürütülmüştür. Nitekim, Python'un özellikle büyük ve karmaşık veri setlerinin modellenmesinde sağladığı avantajlar literatürde de sıklıkla vurgulanmakta ve bu dilin veri bilimi alanında standart haline geldiği ifade edilmektedir (Müller ve Guido, 2016).

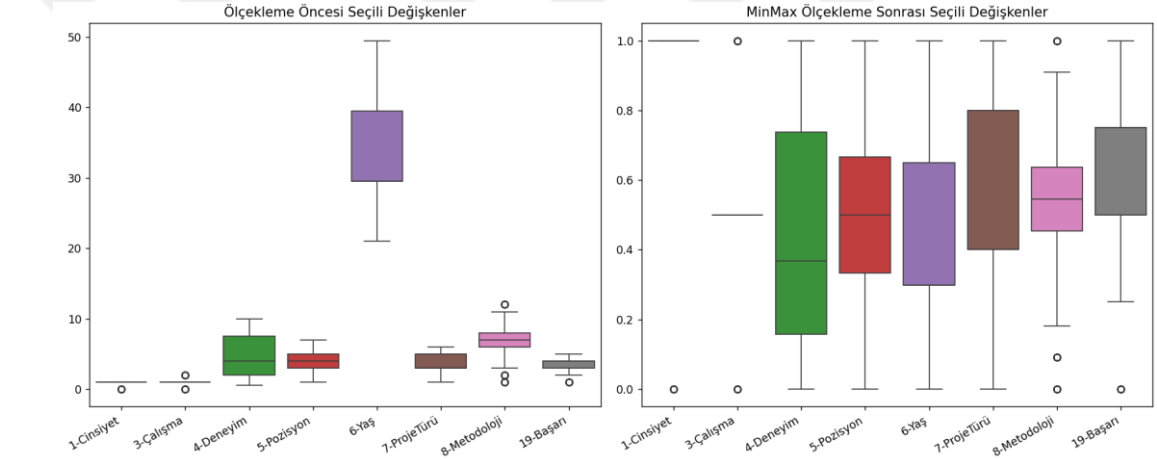
3.5. Veri Kümesinin Hazırlanması

Modelleme çalışmalarına temel oluşturmak amacıyla, bu tez çalışmasında kullanılan anket verileri üzerinde kapsamlı bir veri ön işleme süreci yürütülmüştür. İlk aşamada, anket formundan elde edilen çoktan seçmeli, sıralı ve kategorik değişkenler, Bayrow formülü kullanılarak sayısal formatlara dönüştürülmüştür. Örneğin, cinsiyet değişkeni 'kadın=0', 'erkek=1' biçiminde kodlanmış; eğitim durumu, yaş aralığı ve pozisyon gibi değişkenler ise sıralı yapıya uygun şekilde derecelendirilmiştir. Bu dönüşüm sürecinin sonucunda 'Sayısal Çeviri' başlıklı yeni bir veri seti oluşturulmuş ve çoktan seçmeli sorularda uygulanan veri çoğaltımı sayesinde toplam 3.849 gözlem birimine ulaşılmıştır.

Veri kümesinin hazırlanmasında üç temel ön işlem adımı izlenmiştir:

- 1. Eğitim/Test Ayrımı (Train/Test Split): İlk olarak, objektif şekilde değerlendirebilmek amacıyla veri seti, Python'da train_test_split fonksiyonu kullanılarak eğitim ve test kümelerine ayrılmıştır. Bu işlemde yaygın olarak tercih edilen %70 eğitim – %30 test oranına ek olarak, %80 eğitim – %20 test ve %60 eğitim – %40 test oranları da uygulanarak farklı bölme senaryoları incelenmiştir. Elde edilen sonuçlar karşılaştırıldığında, hata metrikleri açısından oranlar arasında belirgin bir fark olmadığı, modelin farklı veri bölme oranlarına karşı stabil bir performans sergilediği görülmüştür. Eğitim verileri makine öğrenmesi algoritmalarını eğitmek amacıyla kullanılırken, test verileri ise modelin daha önce görmediği yeni veriler üzerindeki tahmin performansını değerlendirmek için ayrılmıştır.

- 2. Veri Normalizasyonu / Standartlaştırma (Scaling): İkinci aşamada, modelin özellikle mesafe tabanlı algoritmalar (örneğin KNN, SVM) tarafından daha etkili kullanılabilmesi için değişkenler arasında ortak bir ölçekte karşılaştırma yapılması gerekmiştir. Bu nedenle veri kümesine MinMaxScaler (Min-Max Ölçekleme) veya StandardScaler (Standart Ölçekleme) gibi ölçekleme teknikleri uygulanmıştır. Bu çalışmada MinMaxScaler kullanılarak tüm değişkenler 0 ile 1 aralığına normalize edilmiştir. Bu işlem sayesinde büyük sayısal farklılıklara sahip değişkenlerin model üzerinde baskın etkiler oluşturması engellenmiş ve her değişken eşit önemle değerlendirilmiştir. Şekil 9'da ilk sekiz değişkenin ölçekleme öncesi ve sonrası dağılımı gösterilmiştir.

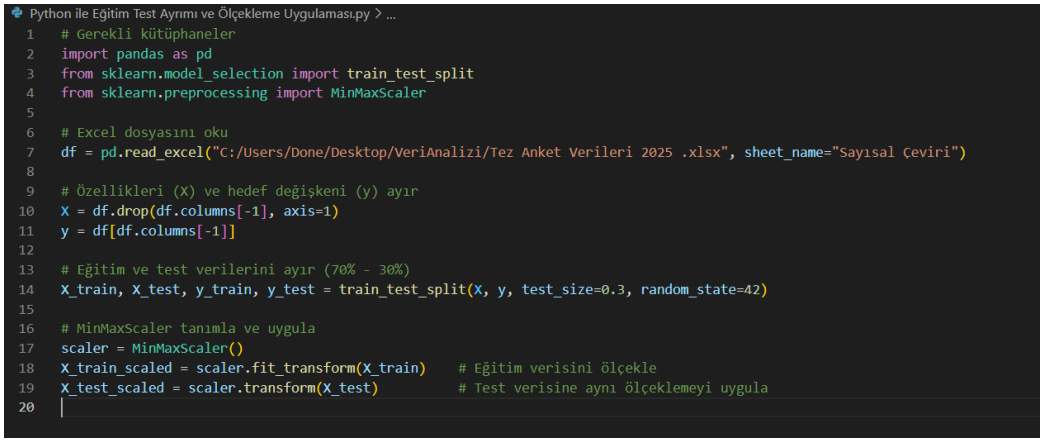


Şekil 9. Seçili Değişkenlerde MinMax Ölçekleme Öncesi ve Sonrası Dağılım

- 3. Eksik Veri Temizliği: Üçüncü ve son aşamada eksik veri temizliği gerçekleştirilmiştir. Veri setinde eksik veya boş bırakılan hücreler kontrol edilmiş; uygun durumlarda ortalama gibi istatistiksel yöntemlerle doldurulmuş, gerekli görülen durumlarda ise model performansını olumsuz etkilememesi adına veri setinden çıkarılmıştır. Bu adım, veri kalitesinin korunması ve algoritmaların sağlıklı şekilde çalışabilmesi açısından kritik bir rol oynamaktadır.

Tüm bu işlemler Python ortamında gerçekleştirilmiş ve veri ölçekleme adımı scaler.fit_transform() ve scaler.transform() komutlarıyla tamamlanmıştır. Şekil 10 eğitim/test ayrımı ve normalizasyon sürecine ait Python uygulama ekran görüntüsü sunulmuştur. Bu

yapılandırılmış veri seti, modelleme aşamasında yüksek doğruluk ve genellenebilirlik sağlayacak şekilde hazırlanmış ve sonraki süreçler için sağlam bir temel oluşturmuştur.



Şekil 10. Python ile Eğitim Ayrımı ve Ölçekleme

Sonuç olarak, bu bölümde gerçekleştirilen veri kümesi hazırlama işlemleri, makine öğrenmesi algoritmalarının daha dengeli, doğru ve genellenebilir sonuçlar üretmesine zemin hazırlamıştır. Eğitim ve test ayrımı, veri ölçekleme ve temizlik adımları sayesinde oluşturulan yapılandırılmış veri seti, sonraki modelleme süreçleri için güçlü bir temel sunmuştur.

3.6. Modelleme Süreci ve Algoritmaların Uygulanması

Yazılım projelerindeki test süreçlerinin yazılım kalitesi üzerindeki etkisini tahmin edebilmek amacıyla, bu çalışmada çeşitli makine öğrenmesi algoritmaları uygulanmıştır (KNN, RF, Naive Bayes, SVM, Lojistik Regresyon ve YSA). Anket verileri üzerinden türetilen bağımsız değişkenler kullanılarak, yazılım test süreçlerinin kalite tahminine olan katkısı sınıflandırma odaklı yöntemlerle değerlendirilmiştir. Bu kapsamda hem doğrusal hem de doğrusal olmayan veri örüntülerine uygun çeşitli algoritmalar test edilmiş ve karşılaştırmalı analizler gerçekleştirilmiştir.

Modelleme sürecinde kullanılan veri seti, 3.3. Veri Güvenilirliği ve Geçerleme bölümünde detaylandırıldığı üzere, Cronbach's Alpha analizi ile içsel tutarlılığı doğrulanmış değişkenlerden oluşturulmuştur. Bu değişkenler, yazılım testi sürecine ilişkin ölçülebilir nitelikteki temel unsurlar olan planlama, test ortamı uygunluğu, kritik hata tespiti, otomasyon uygulamaları, zaman yönetimi ve raporlama gibi bileşenleri içermektedir. Böylece, kalite

düzeyini etkileyen çok boyutlu faktörlerin modele bütüncül biçimde entegre edilmesi sağlanmıştır.
Böylece modelleme için kullanılacak verilerin güvenilirliği sağlanmış, eğitim ve test kümeleri de bu doğrultuda yapılandırılmıştır.

Modelin doğruluk ve tahmin gücü, sadece genel başarı oranlarıyla değil, detaylı performans metrikleriyle de değerlendirilmiştir. Bu kapsamda, sınıflandırma problemlerinin başarımını ölçmek için F1 skoru öne çıkan temel ölçütlerden biri olarak kullanılmıştır. Özellikle F1 skoru, modelin hem *precision* (kesinlik) hem de *recall* (duyarlılık) performansını birlikte değerlendiren dengeli bir metriktir. F1 skoru, yanlış olumlu (false positive) ve yanlış olumsuz (false negative) sınıflamaların birlikte dikkate alındığı durumlarda, modelin genel başarımını daha adil biçimde yansıtır. Aşağıdaki formül yardımıyla hesaplanan bu metrik, özellikle dengesiz veri kümeleriyle çalışırken öne çıkan değerlendirme kriterlerinden biridir.

$$\mathbf{F1} = \mathbf{2} \cdot \frac{\mathbf{Precision} \cdot \mathbf{Recall}}{\mathbf{Precision} + \mathbf{Recall}} \qquad (\mathbf{3.2})$$

Bu bağlamda, her bir algoritmanın performansı doğruluk, F1 skoru ve hata metrikleri (MAE, MSE, RMSE, MAPE) temelinde analiz edilmiştir.

Aşağıda, bu çalışmada kullanılan makine öğrenmesi yöntemlerinin kısa açıklamaları ile sınıflandırmaları Tablo 2'de sunulmaktadır:

Tablo 2. Yazılım kalite tahmininde kullanılan makine öğrenmesi yöntemleri (Geron, 2019)

Yöntem Adı	Türü	Açıklama
Lojistik Regresyon	Sınıflandırma	Olasılık temelli; evet/hayır gibi ikili sınıflamalar için kullanılır.
KNN(K-Nearest Neighbors)	Sınıflandırma/ Regresyon	Yeni verileri benzer komşularına bakarak tahmin eder.
Naive Bayes	Sınıflandırma	Olasılık ve Bayes teoremi temelinde çalışır.
SVM(SupportVector Machines)	Sınıflandırma	En iyi ayrım sınırını bulur; küçük ama karmaşık veri için uygundur.
Random Forest	Sınıflandırma/ Regresyon	Karar ağaçlarının çoğul kararına dayanır (ensemble).
Yapay Sinir Ağları (YSA)	Sınıflandırma/ Regresyon	İnsan beyninden esinlenmiştir; güçlü örüntü tanıma sağlar.

Bu çalışmada kullanılan makine öğrenmesi algoritmalarının uygulanmasında, 3.4. Veri Analizi İçin Programlama Dili Seçimi bölümünde de detaylandırıldığı üzere, Python programlama dili tercih edilmiş; ilgili analizler kapsamında scikit-learn, pandas ve numpy gibi yaygın kütüphanelerden faydalanılmıştır. Modelleme sürecinde, eğitim ve test veri setlerinin ayrımı gerçekleştirilmiş; ayrıca çapraz doğrulama (cross-validation) yöntemi uygulanarak modellerin genellenebilirlik düzeyi değerlendirilmiştir. Elde edilen sonuçlar doğrultusunda, her bir algoritmanın yazılım kalite tahmini üzerindeki etkisi karşılaştırmalı olarak analiz edilmiştir.

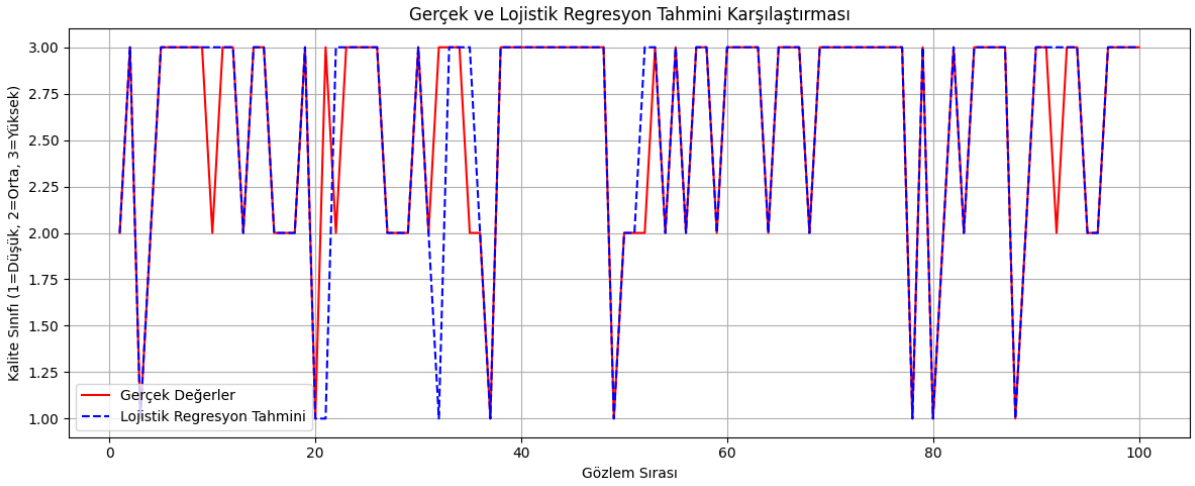
Modelleme sürecinde kullanılan veri kümesi, 3.5. Veri Kümesinin Hazırlanması bölümünde de belirtildiği üzere, öncelikli olarak %70 eğitim – %30 test oranında bölünmüş; bununla birlikte, modellerin kararlılığını test etmek amacıyla %80–%20 ve %60–%40 şeklinde farklı bölme oranları da uygulanarak sonuçlar arasındaki tutarlılık analiz edilmiştir. Elde edilen bulgular, hata metrikleri açısından farklı oranlar arasında belirgin bir performans farkı olmadığını, modelin değişen veri bölme oranlarına karşı stabil bir sınıflandırma başarımı sergilediğini göstermektedir.

4. BULGULAR

Bu çalışma kapsamında kullanılan her bir makine öğrenmesi algoritmasının yazılım test süreçlerine ilişkin kalite düzeyini tahmin etme başarımı; Ortalama Mutlak Hata (MAE), Ortalama Kare Hata (MSE), Kök Ortalama Kare Hata (RMSE) ve Ortalama Mutlak Yüzde Hata (MAPE) gibi hata metrikleri aracılığıyla ayrı ayrı değerlendirilmiştir. Böylelikle her bir modelin tahmin değerlerinin gerçek kalite puanlarına yakınlık düzeyi nesnel olarak karşılaştırılmıştır.

Lojistik Regresyon algoritması, yazılım test faaliyetlerinin kalite düzeyini öngörmede yüksek doğruluk oranı ve sınıf içi tutarlılığı ile öne çıkmıştır. Model uygulaması Python programlama diliyle yürütülmüş; veri işleme, eğitim-test bölme, model eğitimi ve hata metriklerinin hesaplanması işlemleri scikit-learn kütüphanesi yardımıyla gerçekleştirilmiştir. Çalışmada model parametreleri varsayılan ayarlarda tutulmuş, iterasyon sınırı max_iter = 1000 olarak belirlenmiş ve bu sayede lojistik regresyon algoritmasının veri üzerinde tam uyum sağlaması hedeflenmiştir.

Lojistik regresyon modeli ile elde edilen tahmin değerlerinin gerçek verilerle karşılaştırması Şekil 11'de sunulmuştur. Bu grafik, ilk 100 gözlem üzerinde modelin tahmin başarımını görsel olarak ortaya koymakta; kırmızı çizgi gerçek değerleri, mavi çizgi ise lojistik regresyon tahminlerini temsil etmektedir.

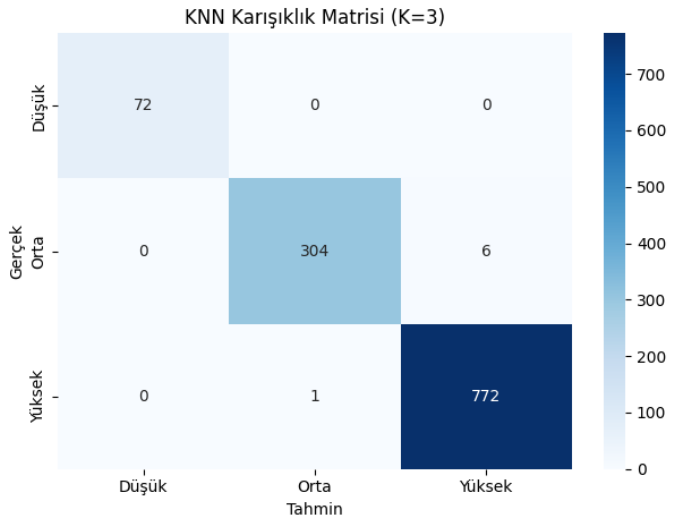


Şekil 11. Lojistik Regresyon Yöntemi ile Tahminleme

Elde edilen bulgular doğrultusunda, lojistik regresyon modelinin genel doğruluk oranı %94,20, ağırlıklı ortalama F1 skoru ise 0,941 olarak hesaplanmıştır. Bu sonuçlar, modelin hem sınıf bazında hem de genel veri kümesi üzerinde yüksek düzeyde bir tahmin performansı sergilediğini ortaya koymaktadır. Ayrıca MAE (0.0615), MSE (0.0684), RMSE (0.2615) ve MAPE (%2,76) değerlerinin düşük seviyelerde olması, modelin öngörüde bulunduğu değerlerin gerçek değerlere oldukça yakın olduğunu desteklemektedir. Lojistik Regresyon algoritması, elde edilen düşük hata oranları ve yüksek başarı ölçütleriyle yazılım projelerinin kalite düzeyini tahmin etmeye yönelik güvenilir bir karar destek aracı olarak değerlendirilmektedir.

K-En Yakın Komşu (KNN) algoritması, kalite düzeyini sınıflandırmada üstün bir performans göstermiştir. Çeşitli k değerleri (k=3, 5, 7 ve 9) denenmiş, k=3 optimum parametre olarak belirlenmiştir.

Elde edilen bulgulara göre, en yüksek doğruluk ve F1 skoru k=3 değerinde gözlemlenmiş; k değeri büyüdükçe başarı oranında sınırlı düzeyde bir azalma meydana gelmiştir. Bu durum, veri setinde daha düşük k değerlerinin komşuluk ilişkilerini daha dengeli yansıttığını ve sınıf ayırımını daha hassas biçimde sağladığını göstermektedir.



Şekil 12. KNN Karışıklık Matrisi

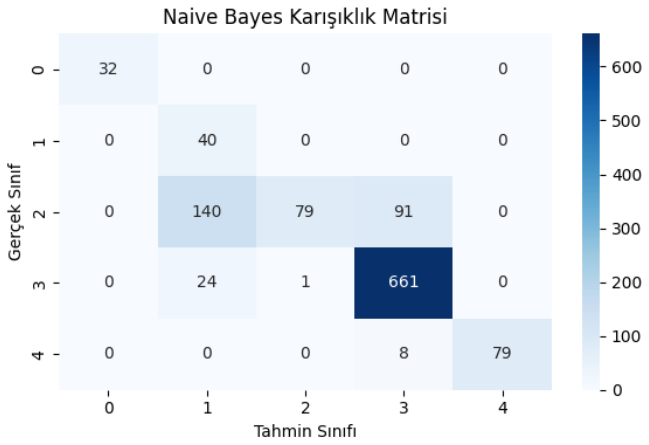
Şekil 12'de sunulan karışıklık matrisi, optimum k=3 parametresiyle oluşturulan KNN modelinin sınıf bazlı tahmin doğruluğunu göstermektedir. Matris incelendiğinde, model

özellikle ‘Yüksek’ ve ‘Orta’ kalite sınıflarında yüksek düzeyde doğru sınıflandırma gerçekleştirmiş; yanlış sınıflandırmalar minimum düzeyde kalmıştır.

Bu senaryoda modelin genel doğruluk oranı %99,39 ve ağırlıklı F1 skoru 0,9939 olarak elde edilmiştir. Sayısal hata metrikleri son derece düşüktür: MAE (0.0061), MSE (0.0061), RMSE (0.0779), MAPE (%0.29). Bu bulgular, KNN algoritmasının sınıf bazlı örüntüleri yüksek isabet oranı ile öngörebildiğini ve özellikle veri yoğunluğu yüksek kümelerde hata oranını minimum düzeyde tutabildiğini ortaya koymaktadır.

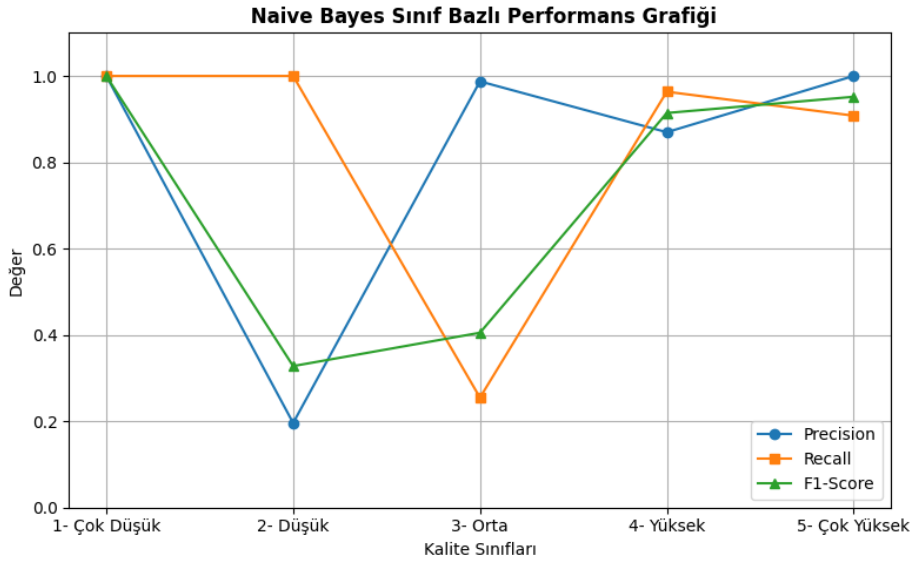
Naive Bayes algoritması, yazılım test süreçlerine ilişkin kalite düzeyi tahmininde %78,61 doğruluk oranı ve 0,7874 ağırlıklı F1 skoru ile değerlendirilmiştir. Özellikle MAPE değeri %12,14 ile diğer algoritmalara kıyasla oldukça yüksektir. MAE (0.2528), MSE (0.3307) ve RMSE (0.5751) gibi diğer hata metrikleri de bu sonucu desteklemektedir. Bu durum, Naive Bayes algoritmasının koşulsuz bağımsızlık varsayımına dayanması ve bu varsayımın gerçek veri setindeki bağımlılıkları tam olarak temsil edememesiyle açıklanabilir. Yazılım test süreçlerinde yer alan kriterler arasında doğal olarak bulunan çok değişkenli ilişkiler, bu modelin sınıflandırma başarısını sınırlamıştır.

Modelin genel başarımı, doğruluk oranı, MAE, MSE, RMSE ve MAPE gibi temel hata metrikleriyle değerlendirilmiştir. Ayrıca elde edilen sınıf tahminlerinin performansı, Şekil 13’te sunulan karışıklık matrisi yardımıyla sınıf bazında ayrıntılı olarak incelenmiştir. Karışıklık matrisi, modelin her bir kalite düzeyine ilişkin doğru ve hatalı sınıflandırma durumlarını göstermekte, sınıf içi öngörü başarısını sayısal olarak ortaya koymaktadır. Buna ek olarak precision (kesinlik), recall (duyarlılık) ve F1 skoru (kesinlik ile duyarlılığın harmonik ortalaması) metriklerinin sınıf bazlı dağılımı Şekil 14’te çizgi grafik biçiminde sunularak modelin kalite düzeyi tahmin performansı görselleştirilmiştir.



Şekil 13. Naive Bayes Karışıklık Matrisi

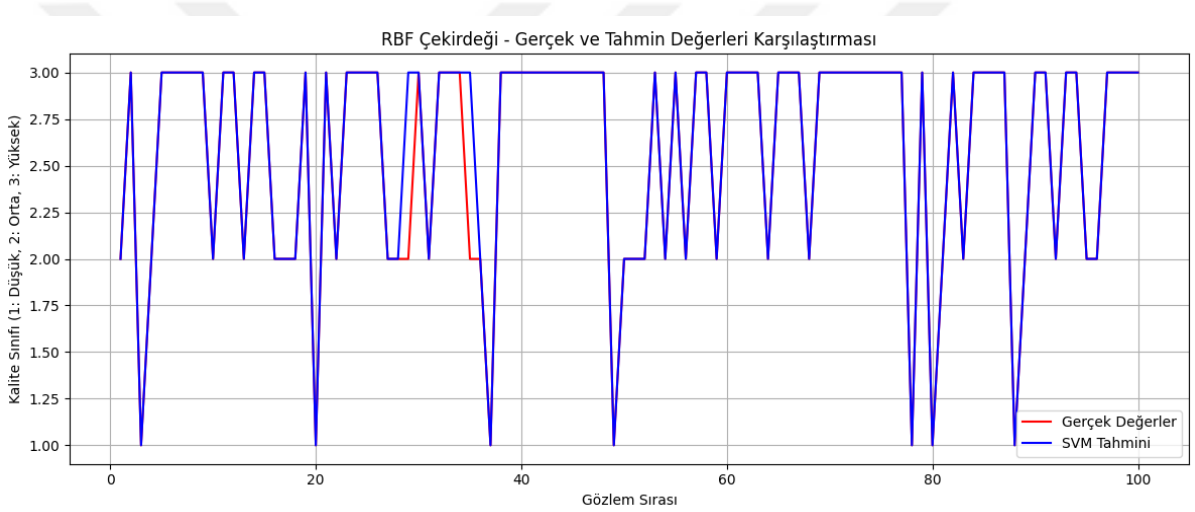
Gaussian Naive Bayes algoritmasıyla elde edilen sınıf tahminleri, gerçek sınıf değerleriyle karşılaştırılarak sınıf içi doğru ve yanlış sınıflandırma durumları ile tahminlerin örtüşme düzeyini göstermektedir.



Şekil 14. Naive Bayes Sınıf Bazlı Performans Grafiği

Şekil 14'te precision (kesinlik), recall (duyarlılık) ve F1 skoru değerlerinin sınıflar bazında karşılaştırmalı dağılımı sunularak, modelin sınıflar arasındaki öngörü performansı istatistiksel olarak görsel biçimde ifade edilmektedir.

Destek Vektör Makineleri (SVM) algoritması, farklı çekirdek (kernel) türleri üzerinden uygulanmış ve elde edilen performanslar karşılaştırılmıştır Doğrusal çekirdeği, %97,75 doğruluk oranı ve 0,9776 ağırlıklı F1 skoru ile yüksek sınıflandırma başarısı sunmuştur. Polinom çekirdeği, %95,93 doğruluk oranı ve 0,9582 ağırlıklı F1 skoru ile doğrusal olmayan örüntülerin ayrımında etkili olmuştur. Doğrusal çekirdeği için MAE 0,0294, MAPE %1,18; Polinom çekirdeği için ise MAE 0,0407, MAPE %2,03 olarak ölçülmüştür. Üç senaryo içerisinde en başarılı sonuçlar, %98,87 doğruluk oranı ve 0,9887 ağırlıklı F1 skoru ile RBF çekirdeği tarafından elde edilmiştir. Hata metrikleri de bu durumu desteklemektedir: MAE (0.0113), MSE (0.0113), RMSE (0.1061) ve MAPE (%0.56). RBF çekirdeği, karmaşık örüntüleri daha iyi öğrenme yeteneğine sahip olduğundan, yazılım test süreçlerinde yer alan çok boyutlu ilişkileri başarıyla modellemiştir.



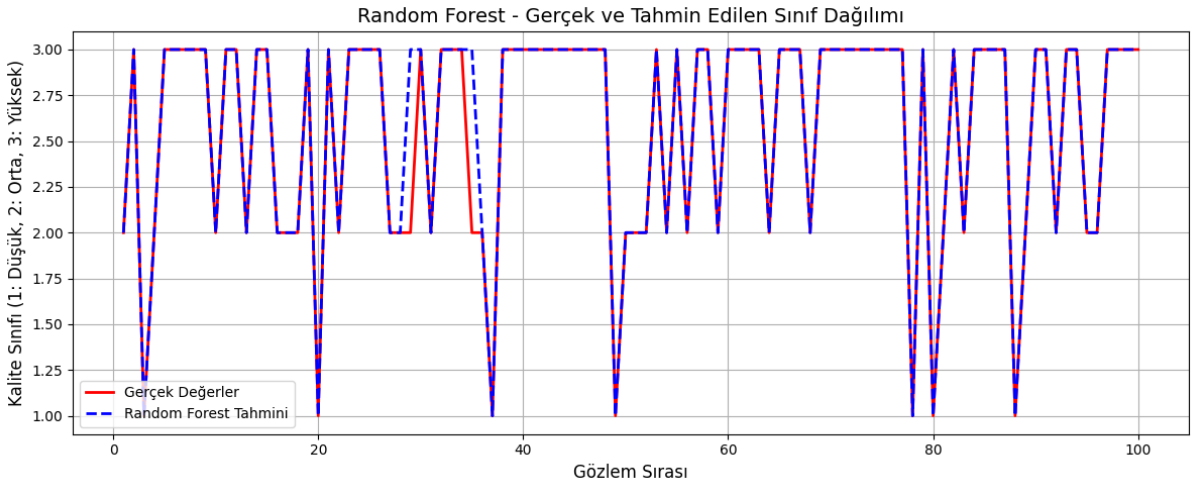
Şekil 15. SVM Yöntemi ile Tahminleme

Şekil 15'te sunulan çizgi grafiği, SVM algoritması ile elde edilen tahmin değerlerinin gerçek kalite sınıfı değerleriyle örnek bazında örtüşme düzeyini göstermektedir. Grafik incelendiğinde, gözlem sırası boyunca modelin sınıf bazlı tahminlerinin, gerçek sınıf etiketleri ile büyük ölçüde uyumlu olduğu görülmektedir. Sadece sınıf atamalarının değil, sınıflar arası geçişlerin de doğru şekilde modellenmesi, SVM yönteminin yazılım test süreçlerine ilişkin kalite düzeylerini istatistiksel olarak güvenilir biçimde öngörebildiğini ortaya koymaktadır.

Random Forest algoritmasının parametreleri belirlenirken;

- n_estimators=100 değeri, ormandaki ağaç sayısını temsil etmekte olup; optimum denge için 100 ağaç yeterli görülmüştür. Çok daha yüksek değerler işlem süresini artırmakla birlikte performansa anlamlı katkı sağlamadığı için tercih edilmemiştir.
- max_depth=5 parametresi, her bir karar ağacının maksimum derinliğini sınırlamak amacıyla ayarlanmıştır. Böylece, ağaçların aşırı derinleşerek veri setini hatasız ezberlemesi engellenmiş, aşırı öğrenmenin önüne geçilmiştir.
- max_features='sqrt' ayarı, her düğüm bölme işlemi için kullanılacak değişken sayısını kontrol etmekte; karekök yaklaşımı, özellik çeşitliliğini koruyarak modelin genelleme yeteneğini artırmaktadır.

Alternatif parametreler de test edilmiş; max_depth parametresi kaldırıldığında ağaçların kontrolsüz derinleştiği ve karar alanlarının karmaşıklaştığı gözlemlenmiş; bu durumun modelin test kümesinde genelleme başarısını düşürdüğü görülmüştür. Benzer şekilde, daha düşük ağaç sayısı kullanıldığında model varyansı artmış ve istikrar azalmıştır. Bu nedenle, mevcut parametreler optimum denge gözetilerek seçilmiştir.



Şekil 16. Random Forest Yöntemi ile Tahminleme

Modelin sınıflandırma yetkinliğini örnek bazında göstermek amacıyla, Şekil 16'da 100 gözleme ait gerçek sınıf değerleri ile Random Forest tahminleri karşılaştırmalı olarak çizgi grafikte sunulmuştur. Grafik, sınıf etiketlerinin sayısal dönüştürülmesiyle (1: Düşük, 2: Orta, 3: Yüksek) oluşturulmuş, iki dağılımın örtüşme düzeyi modelin tahmin doğruluğunu görsel olarak desteklemektedir.

Random Forest algoritması, %98,44 doğruluk oranı ve 0,9843 F1 skoru ile güçlü bir sınıflandırma performansı ortaya koymuştur. Özellikle sınıf bazında yapılan analizlerde "Düşük", "Orta" ve "Yüksek" sınıflarında %100'e yakın doğru sınıflandırmalar gerçekleştirilmiştir. Hata metrikleri ise MAE (0.0156), MSE (0.0156), RMSE (0.1248) ve MAPE (%0,78) şeklindedir. Özellik önem sıralaması grafikleri incelendiğinde, modelin özellikle test planlarının kalitesi ve otomasyon etkinliği gibi değişkenlere yüksek önem atfettiği gözlemlenmiştir. Bu da modelin karar verme mekanizmasının anlamlı örüntüler üzerinden gerçekleştiğini göstermektedir.

Yapay Sinir Ağları (YSA) modeli ise %70 eğitim-%30 test veri seti bölmesiyle yüksek bir tahmin başarımı göstermiştir. Modelin R^2 skoru 0,9862 olarak hesaplanmış; bu değer YSA'nın tahminlerinin gerçek kalite değerleriyle örtüştüğünü kanıtlamaktadır. MAE 0,0347; MSE 0,0084; RMSE 0,0918 ve MAPE %1,01 olarak belirlenmiştir. Ek sınıflandırma analizine göre tahmin edilen değerlerin sınıf bazlı dönüşümü neticesinde modelin yaklaşık genel doğruluk oranı %99,31; ağırlıklı F1 skoru ise 0,9931 olarak bulunmuştur.

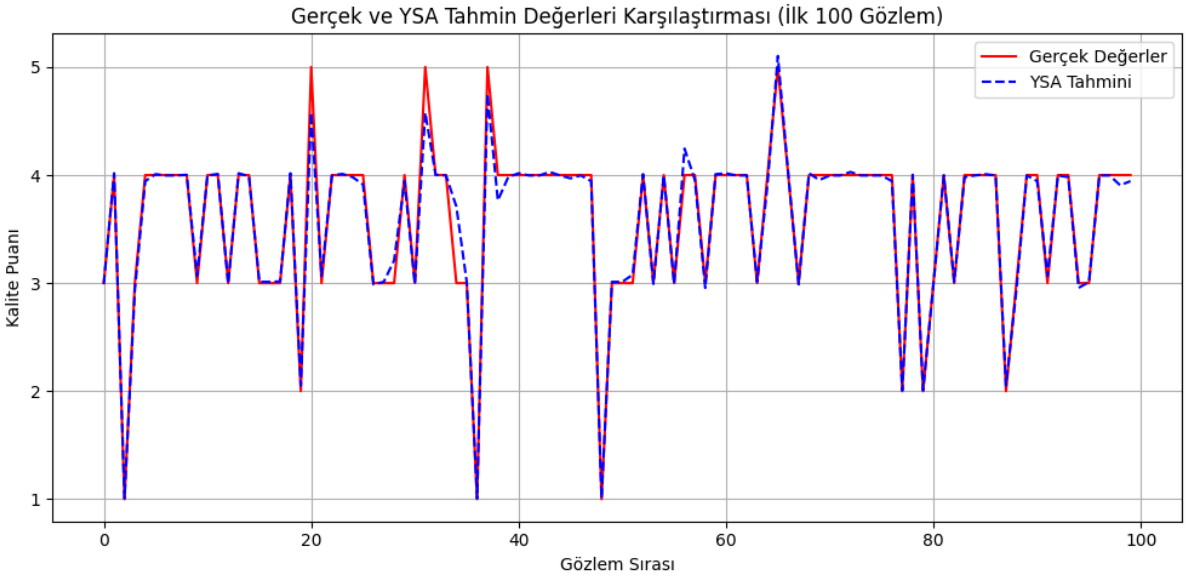
Model, çok katmanlı algılayıcı (MLPRegressor) yapısı temelinde oluşturulmuş ve YSA mimarisi, veri setinin boyutu ve karmaşıklığı göz önünde bulundurularak iki gizli katmanla yapılandırılmıştır. İlk katmanda 50, ikinci katmanda 25 nöron kullanılarak (hidden_layer_sizes= (50, 25)) hem modelin veri örüntülerini yeterli derinlikte öğrenebilmesi hem de hesaplama yükünün makul düzeyde kalması hedeflenmiştir.

Çalışmada, farklı katman sayısı ve nöron kombinasyonları (ör. (25, 10), (100, 50)) denenmiş; tek katmanlı veya çok yüksek nöron sayılı yapıların aşırı öğrenmeye yatkın olduğu, düşük nöronlu yapıların ise karmaşık örüntüleri tam olarak yakalayamadığı gözlemlenmiştir. Bu nedenle iki katmanlı ve dengeli bir nöron sayısı tercih edilmiştir.

Aktivasyon fonksiyonu olarak ReLU (Rectified Linear Unit) tercih edilmiştir. ReLU, doğrusal olmayan örüntülerin etkin biçimde öğrenilmesini sağlamakta ve geri yayılım sırasında gradyan sönümlemesi sorununu azaltarak eğitim sürecini hızlandırmaktadır.

Epoch (max_iter) değeri belirlenirken 500, 750, 1000 gibi farklı iterasyon sayıları test edilmiş; düşük iterasyonlarda modelin hata fonksiyonunu yeterince minimize edemediği, yüksek

iterasyonlarda ise doğrulama hatasında anlamlı bir iyileşme görülmediği tespit edilmiştir. Bu nedenle 1000 epoch değeri optimum öğrenme süresi ve genelleme başarımı sağlamaktadır.



Şekil 17. YSA Yöntemi ile Tahminleme

Şekil 17 Yapay Sinir Ağları yöntemi ile elde edilen tahmin değerlerinin, ilk 100 gözleme ilişkin gerçek kalite puanlarıyla örtüşme düzeyini gösteren çizgi grafik.

Elde edilen tüm bulgular, yazılım test süreçlerine ilişkin kalite düzeyinin tahmin edilmesinde farklı makine öğrenmesi algoritmalarının güçlü birer öngörü aracı olarak kullanılabileceğini ortaya koymaktadır. Özellikle YSA, RBF çekirdeği ile uygulanan SVM ve K-En Yakın Komşu (KNN) algoritmalarının yüksek doğruluk oranları ve düşük hata metrikleri ile karar destek sistemlerinde etkin biçimde uygulanabileceği anlaşılmaktadır.

Tablo 3'te, incelenen tüm modellerin genel doğruluk, F1 skoru ve temel hata metriklerine ilişkin özet bulgular yer almaktadır:

Tablo 3. Tahmin Modellerine Ait Hata Metrikleri

Algoritma Adı	Genel Doğruluk (%)	F1 Skoru	MAPE (%)	MAE	MSE	RMSE
Lojistik Regresyon	94,20	0,941	2.76	0.0615	0.0684	0.2615
K-Nearest Neighbors	99,39	0,9939	0,29	0.0061	0.0061	0.0779
Naive Bayes	78,61	0,7874	12.14	0.2528	0.3307	0.5751
Destek Vektör Makinesi	98,87	0.9887	0.56	0.0113	0.0113	0.1061
Random Forest	98,44	0.9843	0,78	0,0156	0,0156	0,1248
Yapay Sinir Ağı (YSA)	99,31	0.9931	1.01	0.0347	0.0084	0.0918

Bu kapsamda, Tablo 3'te sunulan karşılaştırmalı hata metrikleri modellerin genel başarı durumunu özetlemektedir. Ayrıca görsel karşılaştırmalar, grafiksel dağılımlar ve karışıklık matrisi analizleri aracılığıyla, her bir algoritmanın tahmin performansı detaylı biçimde değerlendirilmiştir. Sonuç olarak, elde edilen bulgular, yazılım test süreçlerinde yapay zekâ temelli sınıflandırma yöntemlerinin etkili bir karar destek mekanizması olarak kullanılabileceğini ve özellikle YSA ile KNN modellerinin yüksek öngörü gücü ile öne çıktığını göstermektedir.

5. TARTIŞMA VE SONUÇ

Yazılım test süreçleri, günümüz yazılım projelerinde yalnızca hata tespiti amacıyla değil, aynı zamanda yazılım kalitesinin sürdürülebilir şekilde yönetilmesi ve proje başarısının artırılması açısından stratejik bir rol üstlenmektedir. Etkin biçimde planlanan ve yürütülen test faaliyetleri; hata tespit oranı, kapsam, doğruluk ve zamanlama gibi çok boyutlu kalite ölçütleri aracılığıyla yazılım ürününün güvenilirliğine ve başarısına doğrudan katkı sağlamaktadır.

Bu çalışmada, yazılım test mühendislerinden anket yoluyla elde edilen veriler doğrultusunda, yazılım projelerine ilişkin kalite düzeylerinin öngörülmesine yönelik makine öğrenmesi temelli sınıflandırma modelleri geliştirilmiştir. Uygulama kapsamında toplanan veriler Cronbach Alpha katsayısı ile güvenilirlik analizine tabi tutulmuş, Bayrow formülü ile sayısallaştırılmış ve veri çoğaltma teknikleri ile modellemeye uygun hale getirilmiştir. Kalite düzeyi hedef değişkeni, düşük, orta ve yüksek olmak üzere sınıflandırılarak, sınıflandırma algoritmaları için uygun etiket yapısına dönüştürülmüştür.

Modelleme sürecinde; Lojistik Regresyon, K-En Yakın Komşu (KNN), Naive Bayes, Destek Vektör Makineleri (SVM), Random Forest ve Yapay Sinir Ağı (YSA) algoritmaları kullanılmıştır. Her bir model; genel doğruluk oranı, F1 skoru gibi sınıflandırma metrikleri ile Ortalama Mutlak Hata (MAE), Ortalama Kare Hata (MSE), Kök Ortalama Kare Hata (RMSE) ve Ortalama Mutlak Yüzde Hata (MAPE) gibi hata metrikleri üzerinden değerlendirilmiştir. Modeller, %70 eğitim – %30 test bölünmesi ile eğitilmiş; bazı algoritmalarda farklı bölme oranları uygulanarak model kararlılığı da gözlemlenmiştir.

Elde edilen bulgulara göre, en yüksek performans değerleri KNN algoritmasında elde edilmiştir. KNN algoritması %99,39 doğruluk oranı, %0,29 MAPE, %0,0061 MAE gibi son derece düşük hata metrikleri ile dikkat çekmiş; özellikle dengeli ve ayrışabilir sınıf yapılarında yüksek doğruluk ve düşük sapma sağlamıştır. YSA modeli ise %99,31 doğruluk oranı, 0,9931 F1 skoru ve %1,01 MAPE değeri ile hem genel başarı hem de düşük hata oranı açısından başarılı bir model olarak öne çıkmıştır. SVM algoritması, RBF çekirdeği ile uygulandığında %98,87 doğruluk ve %0,56 MAPE değeri ile yüksek sınıflandırma başarısı sunmuştur. Random Forest algoritması %98,44 doğruluk ve %0,78 MAPE değeri ile başarılı bir tahmin performansı sergilemiştir. Lojistik Regresyon algoritması ise %94,20 doğruluk oranı

ve %2,76 MAPE değeri ile klasik modelleme yöntemleri arasında iyi bir başarı ortaya koymuştur.

Naive Bayes algoritması %78,61 doğruluk oranı, %0,7874 F1 skoru ve %12,14 MAPE değeri ile en düşük başarıyı göstermiştir. Bu sonuç, modelin temel varsayımı olan öznitelikler arası koşulsuz bağımsızlık ilkesinin, yazılım test süreçlerine ilişkin çok boyutlu ve yüksek etkileşimli verileri temsil etmede yetersiz kalmasından kaynaklanmaktadır. Naive Bayes modeli sınıf ayrımını tam olarak yansıtamamakta, bu da yüksek hata oranlarına ve düşük sınıflandırma başarısına neden olmaktadır.

Sonuç olarak, yazılım test süreçlerinden elde edilen verilerin, makine öğrenmesi algoritmalarıyla analiz edilerek kalite düzeylerinin nesnel biçimde tahmin edilebileceği ortaya konulmuştur. Bu bağlamda, KNN en iyi başarımı yakalarken YSA ve RBF çekirdekli SVM algoritmaları; yüksek doğruluk, düşük hata oranları ve güçlü sınıf ayrımı yetenekleriyle, yazılım sektöründe karar destek sistemlerine entegre edilebilecek nitelikte modeller olarak önerilmektedir.

5.1 Öneriler

- Yazılım kalite tahmini süreçlerinde, KNN algoritması yüksek doğruluk oranı ve düşük hata değerleri nedeniyle öncelikli olarak tercih edilebilir. KNN'e benzer şekilde başarılı sonuçlar veren YSA ve SVM algoritmaları da alternatif olarak kullanılabilir.
- Anket tabanlı veri toplama yaklaşımı, yazılım testi gibi öznel süreçlerin sayısallaştırılmasında uygulanabilir ve farklı kurumsal yapılara kolayca uyarlanabilir.
- Sınıflandırma modellerinin, kurumsal yazılım test yönetim araçlarına entegre edilmesi, manuel kalite takip süreçlerinin otomasyonuna katkı sağlayacaktır.
- Yapay zekâ destekli otomatik kalite sınıflandırma uygulamaları, yazılım test ekiplerinin karar verme süreçlerine entegre edilerek operasyonel verimlilik artırılabilir.
- Gelecek çalışmalarda, hata yönetim sistemlerinden veya gerçek zamanlı test raporlarından sağlanacak verilerle daha derinlemesine ve dinamik modellemeler yapılabilir.
- Özellik seçimi, hiperparametre optimizasyonu gibi ileri seviye ayarlamalarla model başarımının artırılması, daha esnek ve güvenilir sonuçlar sunulmasına katkı sağlayacaktır.

5.2. Kısıtlar

- Bu tez çalışması, yalnızca beşli Likert ölçekli anket verilerine dayandığı için test süreçlerine ilişkin değerlendirmeler öznel kullanıcı yanıtlarıyla sınırlı kalmıştır. Gerçek zamanlı test çıktıları, sistem hataları veya hata log'ları gibi daha nesnel veri kaynakları modelleme sürecine dâhil edilmemiştir.
- Daha çeşitli ve büyük örneklem grupları ile model genelleme kapasitesi artırılabilir.
- Geliştirilen modeller, herhangi bir yazılım projesine ön değerlendirme aracı olarak entegre edilmemiştir. İleride bu modeller gereksinim analizi ve test planlamasında öngörüsel destek aracı olarak kullanılabilir.
- Modelleme sürecinde yalnızca temel düzey hiperparametre ayarları tercih edilmiştir. Gelişmiş optimizasyon teknikleriyle (Grid Search, Random Search, vb.) daha yüksek başarımlara ulaşılabilir.

Bu tez çalışması, yazılım test süreçlerinin yapay zekâ destekli sınıflandırma algoritmaları aracılığıyla sistematik biçimde analiz edilebileceğini ve yazılım kalite tahminine yönelik uygulanabilir modeller geliştirilebileceğini ortaya koymuştur. Elde edilen bulguların hem akademik literatüre katkı sunması hem de sektörel düzeyde karar destek sistemlerine entegre edilerek fayda sağlaması beklenmektedir.

KAYNAKÇA

Acropolium. (2025). "Ulaşım ve lojistikte yapay zekâ: 6 kullanım örneği". https://acropolium.com/blog/use-cases-of-ai-in-transportation-logistics-are-they-relevant-for-your-business/ adresinden 1 Haziran 2025 tarihinde edinilmiştir.

Albanese, A., Nardello, M. ve Brunelli, D. (2021). "Deep neural network inference for pest detection on edge devices for precision agriculture". *IEEE Transactions on Circuits and Systems for Video Technology*.

Alpaydın, E. (2020). *Introduction to Machine Learning*. Massachusetts Institute of Technology, Cambridge: The MIT Press.

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician, 46*(3), 175–185.

American Medical Association. (2025). *2 in 3 physicians are using health AI—up 78% from 2023*. https://www.ama-assn.org/practice-management/digital-health/2-3-physicians-are-using-health-ai-78-2023 adresinden 1 Haziran 2025 tarihinde edinilmiştir.

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., ... ve Zimmermann, T. (2019). "Software engineering for machine learning: A case study". *IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*, 291–300.

Anantrasirichai, N. ve Bull, D. R. (2021). "Yaratıcı endüstrilerde yapay zekâ: Bir inceleme". *Artificial Intelligence Review*, 55, 589–656.

Arora, A., Soni, D. ve Soni, A. (2015). "A survey on software defect prediction models". *International Journal of Computer Applications*, 116(12), 1–5.

Atagören, Ç. (2012). *Yazılım mühendisliği projelerinde hata ölçümü, kök neden analizleri ve örnek bir vaka incelemesi* (Doktora tezi). Yükseköğretim Kurulu Ulusal Tez Merkezi'nden edinilmiştir.

Aydın, M. ve Özkul, E. (2015). "Makine öğrenmesi algoritmalarının sınıflandırma performanslarının karşılaştırılması". *Sakarya Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, 19(3), 295–303.

Ayhan, H. ve Erdoğmuş, N. (2014). "Destek Vektör Makineleri ve Finansal Tahminler Üzerine Bir Uygulama". *Osmangazi Üniversitesi İİBF Dergisi*, 9(1), 1–21.

Baqar, M. ve Khanda, R. (2024). "The Future of Software Testing: AI-Powered Test Case Generation and Validation". *arXiv preprint arXiv:2409.05808*. https://arxiv.org/abs/2409.05808 adresinden 1 Haziran 2025 tarihinde edinilmiştir.

Baktir, N. ve Atay, Y. (2022). “Makine öğrenmesi yaklaşımlarının spam-mail sınıflandırma probleminde karşılaştırmalı analizi”. *Bilişim Teknolojileri Dergisi*, 15(3), 349–354.

Basili, V. R. ve Selby, R. W. (1987). “Comparing the effectiveness of software testing strategies”. *IEEE Transactions on Software Engineering*, 13(12), 1278–1296.

Batarseh, F. A., Freeman, L. ve Huang, C. H. (2021). “A survey on artificial intelligence assurance”. *Journal of Big Data*, 8(1), 1–36.

Bedi, J. S., Kinger, R. ve Sharma, V. (2022). “A novel hybrid deep learning model using leaky ReLU and batch normalization for image classification”. *Neural Computing and Applications*, 34(13), 10755–10766.

Beşli, O. ve Çavdar, İ. H. (2010). “Veri ambarı yazılım geliştirme sürecinde test”. XII. Akademik Bilişim Konferansı’nda sunulan bildiri, Muğla.

Büyüköztürk, Ş. (2017). Sosyal bilimler için veri analizi el kitabı: İstatistik, araştırma deseni, SPSS uygulamaları ve yorum. Ankara: Pegem Akademi Yayıncılık.

Breiman, L. (2001). “Random forests”. *Machine Learning*, 45(1), 5–32.

Bilgici, C. (2023). “Yapay zekâ ve algoritmik kültür bağlamında sosyal medya deneyiminin geleceği”. *Yeni Medya*, 7(3), 45–62.

Canaz Sevgen, S. (2022). *Kitlesel değerlemede makine öğrenme algoritmaları* (Doktora tezi). Ankara Üniversitesi, Fen Bilimleri Enstitüsü, Gayrimenkul Geliştirme ve Yönetimi Anabilim Dalı, Ankara.

Caruana, R. ve Niculescu-Mizil, A. (2006). “An empirical comparison of supervised learning algorithms”. 23. Uluslararası Makine Öğrenimi Konferansı Bildiri Kitabı, 161–168.

Chaterji, S., DeLay, N., Evans, J., Mosier, N., Engel, B., Buckmaster, D. ve Chandra, R. (2020). “Geniş ölçekli dijital tarım için yapay zekâ: Teknikler, politikalar ve zorluklar”.

Cover, T. M. ve Hart, P. E. (1967). “Nearest neighbor pattern classification”. *IEEE Transactions on Information Theory*, 13(1), 21–27.

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika*, 16(3), 297–334. https://doi.org/10.1007/BF02310555

Çevik, A. ve Arslan, M. (2022). “Görüntü işleme ve yapay zekâ destekli sistemlerde ses tanıma uygulamaları”. *Yapay Zekâ Uygulamaları Dergisi*, 5(1), 21–35.

Çınar, A. (2024). “Multi-class classification with the Gaussian Naive Bayes algorithm”. *Journal of Data Applications*, 2, 1–13.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297.

Darad, K. (2024). “Yapay zekâ destekli uyarlanabilir öğrenme: Kişiselleştirilmiş eğitimin geleceği”.

De Silva, D. ve Alahakoon, D. (2022). “An artificial intelligence life cycle: From conception to production”. *Patterns*, 3(10), 100489.

Değirmenci, S. D. (2024). “Yapay Zekâ ve Kamu Hizmetleri Çalıştayı-23.11.2024”. *Publicus*, (2), 251–271.

Efe, M. (2021). “Yapay zekâda pekiştirmeli öğrenme algoritmalarının karşılaştırılması”. *Afyon Kocatepe Üniversitesi Fen Bilimleri Dergisi*, 21(1), 12–22.

Erdem, B. K. (2021). “Yapay zekânın medya ve yayıncılık alanına etkisi”. *TRT Akademi*, 6(13), 896–903.

Floridi, L., Cowls, J., Beltrametti, M., Chatila, R., Chazerand, P., Dignum, V., ... ve Vayena, E. (2018). “AI4People—An ethical framework for a good AI society: Opportunities, risks, principles, and recommendations”. *Minds and Machines*, 28(4), 689–707.

Gao, J., Bai, X. ve Tsai, W. T. (2019). “Automated test case generation using artificial intelligence techniques”. *IEEE Software*, 36(2), 44–51.

Gao, C., Xu, J. ve Yu, Y. (2020). “Softmax regression based software fault prediction using code metrics”. *IEEE Access*, 8, 113307–113316.

Garousi, V., Bauer, S. ve Felderer, M. (2020). “NLP-assisted software testing: A systematic mapping of the literature”. *Information and Software Technology*, 126, 106321.

Gelperin, D. ve Hetze, B. (1988). “The growth of software testing”. *Communications of the ACM*, 51(6).

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.

Gezmez, A. (2022). *Makine öğrenmesi ve yazılım test süreçleri: Temel kavramlar ve uygulamalar*. İstanbul: Akademik Yayıncılık.

Goodfellow, I., Bengio, Y. ve Courville, A. (2016). *Deep Learning*. Cambridge: MIT Press.

Gupta, G. ve Pal, S. K. (2025). “Hassas tarımda yapay zekâ uygulamaları”. *Discover Agriculture*, 3(61).

Haddouchi, M. ve Berrado, A. (2024). “A survey and taxonomy of methods interpreting random forest models”. *arXiv preprint arXiv:2407.12759*. https://arxiv.org/abs/2407.12759 adresinden 1 Haziran 2025 tarihinde edinilmiştir.

Hasan, M. M., Islam, M. U. ve Sadeq, M. J. (2022). “Yapay zekâ, IoT ve robotik ile gelişmiş tarımın teknolojik adaptasyonuna doğru: Kapsamlı bir inceleme”. *Artificial Intelligence and Smart Agriculture Technology*, 21–42. https://arxiv.org/abs/2202.10459 adresinden 1 Haziran 2025 tarihinde edinilmiştir.

Haykin, S. (2009). *Neural Networks and Learning Machines* (3rd ed.).

Hindle, A., Barr, E. T., Gabel, M., Su, Z. ve Devanbu, P. (2016). "On the naturalness of software". *Communications of the ACM*, 59(5), 122–131.

Horowitz, B. T. (2023). "The current state of AI in healthcare and where it's going". *HealthTech Magazine*. https://healthtechmagazine.net/article/2023/01/ai-healthcare-trends adresinden 1 Haziran 2025 tarihinde edinilmiştir.

Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression* (3rd ed.). Wiley.

Hourani, H., Hammad, A. M. ve Lafi, M. (2019). "The impact of artificial intelligence on software testing". *International Journal of Advanced Computer Science and Applications*, 10(4), 565–570.

Hu, Y., Zhang, L. ve Chen, J. (2023). "Utilizing random forest algorithm for early detection of academic failure in open learning environments". *Journal of Educational Data Mining*, 15(1), 45–60.

Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment*. Computing in Science & Engineering, 9(3), 90–95. Erişim adresi: https://matplotlib.org adresinden 1 Haziran 2025 tarihinde edinilmiştir.

ISTQB. (2018). *Foundation Level Syllabus Version 2018*. International Software Testing Qualifications Board. Retrieved from https://www.istqb.org/ adresinden 1 Haziran 2025 tarihinde edinilmiştir.

James, G., Witten, D., Hastie, T. ve Tibshirani, R. (2021). *An Introduction to Statistical Learning with Applications in R* (2. baskı). New York: Springer.

Jordan, M. I. ve Mitchell, T. M. (2015). "Machine learning: Trends, perspectives, and prospects". *Science*, 349(6245), 255–260.

Kavzoğlu, T. ve Çölkesen, İ. (2010). "Destek vektör makineleri ile uydu görüntülerinin sınıflandırılmasında kernel fonksiyonlarının etkilerinin incelenmesi". *Harita Teknolojileri Elektronik Dergisi*, 2(1), 1–9.

Keleş, M., Öztürk, F. ve Ayaz, R. (2020). "Makine öğrenmesi temelli sistemlerin karar süreçlerine etkisi: Kavramsal bir değerlendirme". *Bilgi Teknolojileri ve Uygulamaları Dergisi*, 11(2), 145–158.

Keskin, B. ve Yıldız, B. (2018). "Makine öğrenmesi yöntemleri ve uygulama alanları: Derinlemesine bir inceleme". *Bilgisayar ve Yazılım Bilimleri Dergisi*, 6(2), 104–113.

Khan, M. E. (2011). "Different approaches to black box testing technique for finding errors". *International Journal of Software Engineering & Applications*, 2(4), 31–40.

Khilari, S., Bhamango, B., Dabade, T., Kale, R. ve Khade, M. (2024). "Software quality assurance using machine learning algorithms". *International Journal of Software Engineering & Applications*, 15(1), 78–91.

Kılıç, E. ve Öztürk, S. (2010). "Büyük ölçekli yazılım projelerinde entegrasyon testleri". 2. Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu'nda sunulan bildiri, İstanbul.

Koç, M. ve Altun, Y. (2023). "Karar destek sistemlerinde yapay zekâ uygulamaları: Bir literatür taraması". *Bilgi Yönetimi ve Teknolojileri Dergisi*, 8(2), 50–64.

Kuyoro, A. O., Sanusi, O. A. ve Ajayi, O. T. (2022). "Dynamic effectiveness of random forest algorithm in financial credit risk management for improving output accuracy and loan classification prediction". *Information Systems and Informatics*, 27(5), 15–25.

Martin, D., Rooksby, J., Rouncefield, M. ve Sommerville, I. (2007). "'Good' organisational reasons for 'bad' software testing: An ethnographic study of testing in a small software company". *Proceedings of the 29th International Conference on Software Engineering*, 602–611.

McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. Proceedings of the 9th Python in Science Conference, 51–56. Erişim adresi: https://pandas.pydata.org adresinden 1 Haziran 2025 tarihinde edinilmiştir.

McLaren, M., Vogt, R., Baker, B. ve Sridharan, S. (2010). "Experiments in SVM-based speaker verification using short utterances". *17. Odyssey Konferansı'nda sunulan bildiri*.

Melnik, G., Meszaros, G. ve Bach, J. (2009). *Acceptance Test Engineering Guide*. Redmond, WA: Microsoft Press.

Meral, M. ve Saraçlı, S. (2020). "Makine öğrenmesi yöntemleri ile sınıflandırma problemleri". *İstatistik Araştırmaları Dergisi*, 13(1), 21–38.

Meriç, Ö. ve Özbayoğlu, A. M. (2021). "Yapay öğrenme ile yazılım test eforu tahmini". *Veri Bilimi Dergisi*, 4(1), 38–44.

Mitchell, T. (1997). *Machine Learning*. New York: McGraw-Hill.

Müller, A. ve Guido, S. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. Sebastopol, CA: O'Reilly Media, Inc.

Myers, G. J., Sandler, C. ve Badgett, T. (2012). *The Art of Software Testing* (3. baskı). Canada: John Wiley ve Sons, Inc.

Nama, P. (2024). "Enhancing test coverage and predictive analysis for improved software quality". *World Journal of Advanced Engineering Technology and Sciences*, 3(2), 52–61.

Nasir, M. H. N. ve Sahibuddin, S. (2011). “Critical success factors for software projects: A comparative study”. *Scientific Research and Essays*, 6(10), 2174–2186.

Ngah, A., Munro, M. ve Abdallah, M. (2017). “An Overview of Regression Testing”. *Journal of Telecommunication, Electronic and Computer Engineering*, 9(3–5), 45–54.

Onan, A. ve Korukoğlu, S. (2016). “Makine öğrenmesi yöntemleri ve uygulama alanları: Sistematik bir inceleme”. *Yazılım Mühendisliği ve Bilgi Sistemleri Dergisi*, 4(1), 22–36.

Orso, A. ve Rothermel, G. (2014). “Software testing: A research travelogue (2000–2014)”. *Proceedings of the Future of Software Engineering*, 117–132.

Osherove, R. (2013). *The Art of Unit Testing* (2. baskı). United States of America: Manning Publications.

Öztürk, M. ve Akman, N. (2023). “Yapay zekânın gelişimi ve insan zekâsı ile ilişkisi: Nörobilimsel bir yaklaşım”. *Bilim ve Teknoloji Araştırmaları Dergisi*, 9(1), 15–29.

Panichella, A. (2021). “Automated software testing: A machine learning-based approach”. *Journal of Systems and Software*, 175, 110886.

Panwar, A. ve Peddi, P. (2023). “Implementation of software testing using machine learning: A systematic mapping study”. *SSRN Electronic Journal*.

Patton, R. (2000). *Software Testing* (1. baskı). Indiana: Sams Publishing.

Ramadan, A., Yasin, H. ve Pektaş, B. (2024). *Yazılım testinde yapay zekâ ve makine öğrenmesinin rolü*. arXiv. https://arxiv.org/abs/2409.02693 adresinden 1 Haziran 2025 tarihinde edinilmiştir.

Pettorruso, M., Martinotti, G., Di Giannantonio, M. ve De Berardis, D. (2023). “Application of the random forest algorithm for accurate bipolar disorder classification using EEG data”. *Life*, 15(3), 394.

Safaat, N. ve Tjhin, V. U. (2024). “A comparative study of manual and automated testing using Katalon Studio”. *International Journal of Advanced Computer Science and Applications*, 15(4), 134–141.

Salahat, M., Abu-Tayeh, B., Al-Ghazzawi, K. ve Tarawneh, M. (2023). “Software testing issues improvement in quality assurance”. 2nd International Conference on Business Analytics for Technology and Security (ICBATS 2023) kapsamında sunulan bildiri.

Sarı, S. (2024). “Bankacılıkta yapay zekâ uygulamaları”. *Journal of Emerging Economies and Policy*, 9(Özel Sayı), 246–263.

Shaik, T., Tao, X., Higgins, N., Li, L., Gururajan, R., Zhou, X. ve Acharya, U. R. (2023). “Remote patient monitoring using artificial intelligence: Current state, applications, and

challenges". *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2), e1485.

Srinivas, N. ve Jagruthi, D. (2012). "Black box and white box testing techniques – a literature review". *International Journal of Embedded Systems and Applications*, 2(4), 29–37.

Steegmans, E., Bekaert, P., Devos, F., Delanote, G., Smeets, N., van Dooren, M. ve Boydens, J. (2003). *Black & White Testing: Bridging Black Box Testing and White Box Testing*. KU Leuven, Bilgisayar Bilimleri Bölümü, Belçika.

Saylan, B. ve Çınaroğlu, S. (2024). "Metin madenciliği ve makine öğrenmesi teknikleri ile sağlık hizmetleri pazarlamasına yönelik Twitter verilerinin analizi". *Bilişim Teknolojileri Dergisi*, 17(2), 109–121.

Sokolova, M. ve Lapalme, G. (2009). "A systematic analysis of performance measures for classification tasks". *Information Processing & Management*, 45(4), 427–437.

Şahin, S. (2024). "Finans sektöründe yapay zekâ, makine öğrenmesi ve büyük veri kullanımı: Fırsatlar, zorluklar ve politika yapıcılar için çıkarımlar". *Finans Ekonomi ve Sosyal Araştırmalar Dergisi*, 9(4), 364–381.

Tassey, G. (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing*. RTI for National Institute of Standards and Technology.

Tavşancıl, E. (2002). *Tutumların Ölçülmesi ve SPSS ile Veri Analizi*. Ankara: Nobel Yayın Dağıtım.

The Standish Group International Inc. (2013). *CHAOS Manifesto 2013: Think Big, Act Small*. Boston, MA: The Standish Group International Inc. https://www.standishgroup.com/sample_research_files/CM2013-8%2B9.pdf adresinden 1 Haziran 2025 tarihinde edinilmiştir.

Transport Logistic. (2025). "Lojistikte yapay zekâ: Uygulama alanları ve fırsatlar". https://transportlogistic.de/... adresinden 20 Nisan 2025 tarihinde edinilmiştir.

TÜBA. (2020). *Yapay Zekâ Raporu*. Türkiye Bilimler Akademisi Yayınları.

Uber Freight. (2025). "Uber Freight, yapay zekâyla kamyon taşımacılığını nasıl optimize ediyor". *Business Insider*. https://www.businessinsider.com/... adresinden 1 Haziran 2025 tarihinde edinilmiştir.

Umar, A. I., Salim, N. ve Khan, A. (2019). "An improved K-nearest neighbor algorithm for classification problems". *International Journal of Advanced Computer Science and Applications*, 10(7), 176–182.

Uzun, B. ve Koruyan, K. (2019). "Yazılım test sürecinde durum raporlamasına genel bakış ve yaklaşımlar". *Yönetim Bilişim Sistemleri Dergisi*, 5(1), 52–63.

Ün, A. (2022). “Yapay zekâ uygulamalarının kamu yönetimindeki rolü ve önemi”. *Kamu Yönetimi Dergisi*, 55(1), 123–138.

Van der Laan, M. J. ve Rose, S. (2022). “Using random forest to identify longitudinal predictors of health in a 30-year cohort study”. *Scientific Reports*, 12, 14632.

Vocke, H. (2018). *The Practical Test Pyramid*. https://martinfowler.com/articles/practical-test-pyramid.html adresinden 1 Haziran 2025 tarihinde edinilmiştir.

Vural, Y. ve Sağıroğlu, Ş. (2011). “Kurumsal bilgi güvenliğinde güvenlik testleri ve öneriler”. *Gazi Üniversitesi Mühendislik ve Mimarlık Fakültesi Dergisi*, 26(1), 89–103.

Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S. ve Wang, Q. (2024). “Software Testing with Large Language Models: Survey, Landscape, and Vision”. *IEEE Transactions on Software Engineering*, 50(4), 911–936.

Whittaker, J. A. (2000). “What is software testing? And why is it so hard?”. *IEEE Software*, 17(1), 70–79.

Yalanskyi, A. (2024). “Yapay zekâ destekli uyarlanabilir öğrenme: Kişiselleştirilmiş eğitimin geleceği”. *eLearning Industry*. https://elearningindustry.com/understanding-adaptive-learning-how-ai-is-revolutionizing-personalized-education adresinden 1 Haziran 2025 tarihinde edinilmiştir.

Zavuoğlu, M. ve Çölkesen, E. (2010). “SVM parametrelerinin performans üzerindeki etkisi”. *Akademik Bilişim Konferansı Bildirileri*, 537–542.

Yıldız, F. ve Akbulut, T. (2021). “Yapay zekâ teknolojilerinin toplumsal etkileri ve etik sorumluluklar”. *Sosyal Bilimler ve Teknoloji Dergisi*, 3(2), 113–128.

Yıldız, A. (2022). “Finans alanında yapay zekâ teknolojisinin kullanımı: Sistematik literatür incelemesi”. *Pamukkale Üniversitesi Sosyal Bilimler Enstitüsü Dergisi*, 52, 47–66.

Yıldırım, B. ve Küçükcan, B. F. (2023). *Yapay Zekâ: Disiplinlerarası Yaklaşımlar*. İstanbul: Akademik Yayıncılık.

Young, M. ve Pezzè, M. (2008). *Software Testing and Analysis: Process, Principles and Techniques* (1. baskı). Hoboken, NJ: John Wiley & Sons, Inc.

Yücalar, F. ve Borandağ, E. (2019). “Yazılım projelerinde kalitenin arttırılması: TMMi”. *AURUM Mühendislik Sistemleri ve Mimarlık Dergisi*, 3(2).

Zhang, H. (2004). The Optimality of Naive Bayes. *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004)*.

Zhang, Y., Wang, S. ve Lo, D. (2020). “Predicting defective software components using machine learning algorithms”. *Empirical Software Engineering*, 25, 2932–2961.

Zhang, J. M., Harman, M., Ma, L. ve Liu, Y. (2022). “Machine learning testing: Survey, landscapes and horizons”. *IEEE Transactions on Software Engineering*, 48(1), 1–36.

Zhao, Y., Hu, Y. ve Gong, J. (2021). “Research on international standardization of software quality and software testing”. *Proceedings of the 2021 IEEE/ACIS 20th International Fall Conference on Computer and Information Science (ICIS Fall)*, 56–62.

Wallace, D. R. (1986). *An Overview of Computer Software Acceptance Testing*. National Bureau of Standards Special Publication 500-136.

Wolniak, R. ve Stecuła, K. (2024). “Akıllı şehirlerde yapay zekâ: Uygulamalar, engeller ve gelecek yönelimleri”. *Smart Cities*, 7(3), 1346–1389.

EKLER

Ek A. Anket Formu

1. Bölüm: Demografik Özellikler

Bu bölümde demografik özellikler yer almaktadır.

1) Cinsiyetiniz nedir?
    a. Kadın
    b. Erkek

2) En son mezun olduğunuz eğitim kurumu hangisidir?
    a. Lise
    b. Ön Lisans
    c. Lisans
    d. Lisansüstü

3) Çalışma durumunuz?
    a. Part-time çalışıyorum
    b. Tam zamanlı çalışıyorum

4) Çalıştığınız firmada kaç yıllık iş deneyiminiz bulunmaktadır?
    a. 0-1
    b. 1-3
    c. 3-5
    d. 5-10
    e. 10+

5) Çalıştığınız firmada bulunduğunuz pozisyon nedir?
    a. Proje Yöneticisi
    b. Yazılım Geliştirici (Developer)
    c. İş Analisti
    d. Yazılım Test Mühendisi
    e. Operasyon Uzmanı
    f. Veri Tabanı Yöneticisi
    g. Diğer…

6) Hangi yaş aralığındasınız?
    a. 18-24
    b. 25-34
    c. 35-44
    d. 45-54
    e. 55+

2. Bölüm: Proje Özellikleri

Değerlendirmek istediğiniz proje ile ilgili aşağıdaki soruları cevaplayınız.

1. Değerlendirmek istediğiniz projenin türü nedir?
    a) Veri Analizi ve İş Zekâsı (BI) Projeleri
    b) Mobil Uygulama Geliştirme Projeleri
    c) Web Uygulaması ve ya Web Sitesi Geliştirme Projeleri
    d) IoT (Nesnelerin İnterneti) Cihaz ve Sistem Geliştirme Projeleri
    e) Telekomünikasyon Sistemleri (Özellikle OSS/BSS Geliştirme)
    f) Diğer…

2. Değerlendirdiğiniz projede hangi yazılım geliştirme metodolojisi kullanılmaktadır?
    a) Şelale Modeli (Waterfall Model)
    b) V-Model (V-Şeklinde Model)
    c) Artırımlı Model (Incremental Model)
    d) Prototip Model
    e) Helezonik (Spiral) Model
    f) DevOps
    g) Scrum (Çevik Model - Agile)
    h) Kanban (Çevik Model - Agile)
    i) Extreme Programming (XP)
    j) Lean (Yalın Çevik Model)
    k) Hibrit Model
    l) Diğer…

3. Değerlendirdiğiniz projede yer alan toplam ekip üyelerinin sayısı nedir?
    a) 1-5 kişi
    b) 6-10 kişi
    c) 11-14 kişi
    d) 15+ kişi

4. Projenin toplam süresini belirtiniz.
    a) 0-3 ay
    b) 4-6 ay
    c) 7-12 ay
    d) 13-18 ay
    e) 19-24 ay
    f) 24 + ay

5. Bu projenin yazılım ölçeğini nasıl sınıflandırırsınız?
    a) Küçük ölçekli
    b) Orta ölçekli
    c) Büyük ölçekli

6. Projede toplam tahmini kod satırı miktarı nedir? (KLOC – Kilo Lines of Code => 1 KLOC = 1000 Satır Kod)
    a) 1-10 KLOC
    b) 11–50 KLOC
    c) 51–200 KLOC
    d) 201–500 KLOC
    e) 501–1000 KLOC
    f) 1000+ KLOC
    g) Fikrim yok / Tahmin edemiyorum

7. Projenin kapsamı aşağıdaki kategorilerden hangisine daha yakındır?
    a) Dar kapsam (Tek bir modül veya özellik)
    b) Orta kapsam (Birden fazla modül veya özellik)
    c) Geniş kapsam (Tüm sistem veya büyük bir platform)

8. Proje bütçesini nasıl tanımlarsınız?
    a) Çok Düşük maliyetli
    b) Düşük maliyetli
    c) Orta Maliyetli
    d) Yüksek Maliyetli
    e) Çok Yüksek Maliyetli

9. Proje başlarken risk analizi yaptınız mı?
    a) Evet
    b) Hayır
    c) Kısmen

10. Projenin başarısını nasıl ölçtünüz?
    a) Teslimat tarihi
    b) Bütçe uyumu
    c) Kalite standartları
    d) Kullanıcı memnuniyeti
    e) Diğer…

11. Projede hangi test türlerini kullandınız?
    a) Birim Testi (Unit Test)
    b) Entegrasyon Testi
    c) Sistem Testi
    d) Kabul Testi
    e) Performans Testi
    f) Diğer…
    g) Test Sürecine Dahil Olmadım

12. Projede kullanılan programlama dili nedir?
    a) Java
    b) Python
    c) JavaScript
    d) PHP
    e) Go
    f) Kotlin
    g) Ruby
    h) Swift
    i) TypeScript
    j) C++
    k) C#
    l) Diğer…

13. Proje çerçevesini (framework veya kütüphane) belirtiniz?
    a) Spring
    b) Angular
    c) React
    d) Vue.js
    e) Django
    f) Flask
    g) ASP.NET
    h) Ruby on Rails
    i) Laravel
    j) Hibernate
    k) Express.js
    l) Diğer…

14. Projede kullanılan veri tabanını yönetim sistemlerini belirtiniz?
    a) MySQL
    b) PostgreSQL
    c) Microsoft SQL Server
    d) Oracle Database
    e) MongoDB
    f) SQLite
    g) Firebase Realtime Database
    h) Cassandra
    i) MariaDB
    j) Redis
    k) DynamoDB
    l) Diğer…

15. Proje boyunca ekip içi iletişim düzeyini nasıl değerlendirirsiniz?
    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

16. Proje süresince ekip üyeleri ve kullanıcılar arasındaki iletişim düzeyini nasıl değerlendirirsiniz?
    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

17. Proje sonucunda hedeflenen beklentilere ulaşma düzeyini belirtiniz
    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

18. Bu projede bulunan hataların ne kadarını tespit ettiğinizi düşünüyorsunuz?
    a) 0-10
    b) 11-20
    c) 21-30
    d) 31-40
    e) 41-50
    f) 51-60
    g) 61-70
    h) 71-80
    i) 81-90
    j) 91-100

19. Bu projeyi genel olarak ne kadar başarılı buluyorsunuz?

(Aşağıdaki başarı kriterlerini göz önünde bulundurarak değerlendiriniz:Hedeflere ulaşma, zamanında tamamlama, kalite, ekip uyumu, müşteri memnuniyeti...)

    a) 0-10
    b) 11-20
    c) 21-30
    d) 31-40
    e) 41-50
    f) 51-60
    g) 61-70
    h) 71-80
    i) 81-90
    j) 91-100

3. Bölüm: Lütfen bu bölümdeki soruları, dana önceki bölümlerde cevaplandırdığınız proje özelinde yanıtlamaya devam ediniz.

1. Yazılım test faaliyetleri için planlanan süre aralığını nasıl değerlendiriyorsunuz?
    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

2. Yazılım test faaliyetleri için belirlenen plan süresi ile gerçekleşen süre arasındaki farkı (sapmayı) nasıl değerlendirirsiniz?

    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

3. Yazılım test faaliyetleri için planlanan insan kaynağı (personel sayısı ve yeterliliği) sizce yeterli miydi?
(*Lütfen insan kaynağının test sürecindeki iş yükünü karşılama düzeyini değerlendiriniz*.)
    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

4. Yazılım test faaliyetleri için planlanan insan kaynağı ile fiilen kullanılan insan kaynağı arasındaki sapma oranını nasıl değerlendirirsiniz?

(Lütfen planlanan personele göre artış ya da azalma durumunu göz önünde bulundurarak cevaplayınız.)

    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

5. Yazılım test faaliyetleri için planlanan test faaliyet gün sayısından sapma oranını nasıl değerlendirirsiniz?
    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

6. Yazılım test faaliyetleri için oluşturulan test caselerinde bulunan kritik hata tespit oranlarını nasıl değerlendirirsiniz?
   (Kritik hata tespiti, canlıya çıktıktan sonra önemli problemlere yol açabilecek hataların test süreçlerinde ne kadar tespit edildiğini ifade eder.)
   a) Çok Düşük
   b) Düşük
   c) Orta
   d) Yüksek
   e) Çok Yüksek

7. Yazılım test faaliyetleri için oluşturulan test caselerinin sayısını ve hata oranlarının geliştirmeye etkisini nasıl değerlendirirsiniz?

   a) Çok Düşük
   b) Düşük
   c) Orta
   d) Yüksek
   e) Çok Yüksek

8. Projede Yazılım test faaliyetlerinin test caselerinin raporlanmasının projeye etkisini nasıl değerlendirirsiniz?

   a) Çok Düşük
   b) Düşük
   c) Orta
   d) Yüksek
   e) Çok Yüksek

9. Projede planlanan yazılım test faaliyetleri için ayrılan bütçenin yeterliliğini nasıl değerlendirirsiniz?
   a) Çok Düşük
   b) Düşük
   c) Orta
   d) Yüksek
   e) Çok Yüksek

10. Yazılım Yazılım test faaliyetlerinin kalitesini nasıl değerlendirirsiniz?
    (Lütfen test sürecindeki doğruluk, kapsam, hata tespiti oranı ve testlerin etkinliği gibi kriterleri göz önünde bulundurarak yanıtlayınız.)

    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

11. Yazılım test faaliyetlerinin planlanan hedefler doğrultusunda zamanında tamamlanma oranını nasıl değerlendirirsiniz?

    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

12. Yazılım test faaliyetlerinde ortaya çıkan hataların çözüm sürecini nasıl değerlendirirsiniz?

    a) Yavaş ve etkisiz
    b) Orta seviyede
    c) Hızlı ve etkili

13. Yazılım test raporlarının yöneticilere ve ilgili paydaşlara sunulma şeklini nasıl değerlendirirsiniz?

    a) Karışık ve anlaşılmaz
    b) Orta seviyede
    c) Açık ve anlaşılır

14. Yazılım test sürecindeki otomasyonun etkinliğini nasıl değerlendirirsiniz?

    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

15. Yazılım test planlarının proje gereksinimlerini karşılayacak düzeyde olma derecesini nasıl değerlendirirsiniz?

    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

16. Yazılım testlerindeki hata tespit ve raporlama sürecinin etkinliğini nasıl değerlendirirsiniz?

    a) Çok Düşük
    b) Düşük
    c) Orta
    d) Yüksek
    e) Çok Yüksek

17. Yazılım test ortamlarının uygunluğunu ve kullanılabilirliğini nasıl değerlendirirsiniz?

a) Çok Düşük
b) Düşük
c) Orta
d) Yüksek
e) Çok Yüksek

18. Yazılım testlerinde tekrarlanan hataların sayısını nasıl değerlendirirsiniz?

a) Çok Düşük
b) Düşük
c) Orta
d) Yüksek
e) Çok Yüksek

19. Yazılım test faaliyetlerinin planlanan süre içinde tamamlanma oranını nasıl değerlendirirsiniz?

a) Çok Düşük
b) Düşük
c) Orta
d) Yüksek
e) Çok Yüksek