TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY ÖĞRENME İLE YAZILIM TEST EFORU KESTİRİMİ

YÜKSEK LİSANS TEZİ

Özgenil MERİÇ

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Doç. Dr. Ahmet Murat ÖZBAYOĞLU

EKİM 2020

Fen Bilimleri Enstitüsü Onayı

.....
Prof.Dr. Osman EROĞUL
Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

.....
Prof.Dr. Oğuz ERGİN
Anabilimdalı Başkanı

TOBB ETÜ, Fen Bilimleri Enstitüsü'nün **151111011** numaralı Yüksek Lisans öğrencisi **Özgenil MERİÇ** 'in ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"YAPAY ÖĞRENME İLE YAZILIM TEST EFORU KESTİRİMİ"** başlıklı tezi **12.10.2020** tarihinde aşağıda imzaları olan jüri tarafından kabul edilmiştir.

Tez Danışmanı: **Doç.Dr. Ahmet Murat ÖZBAYOĞLU**.....
TOBB Ekonomi ve Teknoloji Üniversitesi

Jüri Üyeleri: **Doç.Dr. Osman ABUL (Başkan)** .....
TOBB Ekonomi ve Teknoloji Üniversitesi

Prof.Dr. Ali YAZICI .....
Atılım Üniversitesi

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, alıntı yapılan kaynaklara eksiksiz atıf yapıldığını, referansların tam olarak belirtildiğini ve ayrıca bu tezin TOBB ETÜ Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırlandığını bildiririm.

Özgenil Meriç

ÖZET

Yüksek Lisans Tezi

YAPAY ÖĞRENME İLE YAZILIM TEST EFORU KESTİRİMİ

Özgenil Meriç

TOBB Ekonomi ve Teknoloji Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Doç.Dr. Ahmet Murat Özbayoğlu

Tarih: EKİM 2020

Yazılım Test dünyasındaki en önemli problemlerden bir tanesi yazılım test planları oluşturulurken test eforunun net bir şekilde belirlenememesidir. Projelerdeki yazılım test işçiliği için ayırılması gereken süre ve kaynak ihtiyacının doğru bir şekilde belirlenebilmesi, proje takvimlerinin oluşturulabilmesi ve kaynakların verimli bir şekilde kullanılabilmesi için önem arz etmektedir. Bu çalışmada yapay öğrenme algoritmaları kullanarak yazılım test eforu tahmini üzerine çeşitli yapay öğrenme modelleri önerilmiştir. Önerilen metot ile ASELSAN A.Ş. bünyesinde geliştirilen, Komuta Kontrol Kullanıcı Arayüzü Yazılımları ve Gömülü ve Gerçek Zamanlı Yazılımları doğrulamak için harcanan test eforu analiz edilerek, ileride yapılması planlanan test aktiviteleri için etkin bir test eforu tahmini yapılmaktadır. Yapılan test eforu tahminleri, şu anda kullanılmakta olan geleneksel yöntemler ile karşılaştırılarak önerilen yöntemin başarı değerlendirmesi de yapılmıştır.

Anahtar Kelimeler: Yazılım test eforu kestirimi, Yapay sinir ağları, Kaynak kullanımı, Makine öğrenmesi.

ABSTRACT

Master of Science

SOFTWARE TESTING EFFORT ESTIMATION WITH

MACHINE LEARNING

Özgenil Meriç

TOBB University of Economics and Technology
Institute of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Doç.Dr. Ahmet Murat Özbayoğlu

Date: October 2020

One of the main headlines of software test literature is the problem of not having a sound estimation of software test effort while scheduling a plan for the whole software development. The software test process time in software projects should be estimated timely in order to gather the required resources beforehand. In this work, using Machine Learning algorithms, we propose a new method of software effort estimation. Using the past experiences of software test efforts processed in ASELSAN in the areas of command center graphical user interfaces, embedded and real-time software test developments, we strive for better estimations. The new estimations are evaluated in comparison with the traditional methods.

Keywords: Software test effort estimation, Artificial neural networks, Source optimization, Machine learning.

TEŞEKKÜR

Her şeyden önce, lisansüstü çalışmalarım sırasında bana ilham veren danışman hocam Doç.Dr. Ahmet Murat Özbayoğlu'na sonsuz şükranlarımı sunmak istiyorum. Pozitifliği, arkadaşça yaklaşımı ve sonsuz desteği ile bana etkili bir araştırma ortamı sağladı. Onunla çalışmak benim için büyük bir onur ve ayrıcalıktı. Araştırma bursu desteği için tüm TOBB ETÜ ailesine teşekkür ederim. Tüm ASELSAN ailesine, teknik destek ve sağladıkları pozitif çalışma ortamı için teşekkür etmek istiyorum. Ayrıca tez komitemdeki Prof.Dr. Ali Yazıcı ve Doç.Dr. Osman Abul hocalarıma değerli katkıları için teşekkür etmek istiyorum. Son olarak, koşulsuz sevgi ve destekleri için aileme teşekkür etmek istiyorum.

İÇİNDEKİLER

YÖK

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

KISALTMALAR

SVM : Support Vector Machines
MLP : Multi Layer Perceptron
RMSE : Root Mean Square Error
MAPE : Mean Absolute Percentage Error
MAE : Mean Absolute Error
ANOVA : Tek Yönlü Varyans Analizi

1. GİRİŞ

Günümüz yazılım projelerinde yazılım test süreci, yazılımın olmazsa olmazları arasındadır. Yazılımcılar kendi kendilerine yaptıkları yazılım testlerinde sözdizimi (syntax) hatalarını yazılımı derlemeye bile gerek görmeden tespit edebilirken kendi yazdıkları yazılımdaki çalışma zamanı hatalarını ve mantık hatalarını görememeleri insani bir durumdur. Bu problemin çözümü yazılım test sürecinden geçer. Günümüz hızlı soluklu ve büyük çaplı Savunma Sanayi projelerinde, projelerin başlangıç ve teslimat süreleri arasındaki zamanlar gitgide azalsa da, kullanılan Savaş Yönetim Sistemleri'nin kontrol ve komuta edildiği yazılım sistemlerinde oluşabilecek en ufak hatalara bile engel olmak en ileri önceliktir. Bu nedenle proje takviminde, doğası gereği efor tespiti zor olan yazılım testine yeterli zaman ve kaynak ayrıldığına emin olunmalıdır. Bu çalışmada Savunma Sanayi yazılım test süreçlerinde uygulanabilecek, yazılım test süreçlerine harcanan test efor kestirimlerini yapabileceğimiz sistematik bir öznitelik araştırma sürecine yer verilmiştir. Savunma Sanayi yazılım test süreçlerinde uygulanabilecek bir yazılım test kestirimi altyapısı kazandırılmıştır.

Savunma sanayinde emniyet kritik sistemler geliştirilmektedir. Sistemlerde ve yazılımlarda çıkabilecek olası hatalar en üst seviyede önem arz etmektedir. Yazılım test faaliyetleri ile sadece geliştirilen yazılım içerisindeki hataların bulunması değil, yazılımın müşteri/sistem gereksinimlerine uygunluğunun kontrol edilmesi de amaçlanmaktadır. Dolayısıyla yazılım test faaliyetleri, yazılım geliştirme yaşam döngüsü içerisinde önemli bir parça olarak kendini göstermektedir.

Yazılım test yöneticileri açısından projelerde harcanacak olan yazılım test işgücünün doğru bir şekilde belirlenebilmesi, mevcut olan kaynak ve zamanı etkin kullanabilmek açısından oldukça önemlidir. Çoğu zaman, uzman görüşüne bağlı olarak belirlenen yazılım test eforunun gerçekçi tahmin edilememesi, test faaliyetlerinin yetersiz yapılmasına sebep olabilmektedir. Bu durum, geliştirilen yazılımın kalitesini önemli ölçüde etkilemektedir.

Yazılım metrikleri ve yazılım geliştirme eforu tahmini hakkında geniş bir araştırma yelpazesi olmasına rağmen, yazılım test eforu tahmininde araştırmaların, oldukça kısıtlı sayıda kaldığı ve gelişime açık bir alan olduğu görülmektedir.

Literatürde, bilişim teknolojileri yazılım projelerinde 40 temel yaygın kalite riski bulunduğu ifade edilmektedir. Ayrıca bu riskleri azaltmak için yazılım risklerini ve hatalarını bulma amaçlı doğrusal kademeli ayrımcılar tanıtılmıştır [18][24].

Her kuruma, projeye ve çalışan personel yelpazesine uygun bir model geliştirmek oldukça zordur. Bu nedenle çalışmamızda ASELSAN SST Sektör Başkanlığı bün-

yesinde geliştirilen Komuta Kontrol Yazılımları ve Gömülü ve Gerçek Zamanlı Yazılımlar için yapılan fonksiyonel kara kutu testlerin efor çalışmaları değerlendirilmiştir.

Bu çalışmayı ana hatları ile ifade etmek gerekirse: Öncelikle yazılım test ve yapay öğrenme algoritmalarının beraber kullanıldığı araştırma alanları hakkında geniş bir literatür araştırmasına yer verilmiştir. Bununla birlikte literatürde yazılım test efor kestirimi ile ilgili çalışmalar anlatılmıştır. Bir sonraki bölüm olan Yazılım Testi bölümünde, en çok kullanılan yazılım test metotlarına, yazılım mühendisliğinde sıklıkla kullanılan yazılım test planı ve süreçlerine yer verilmiştir. Yapay Öğrenme Modelleri bölümünde bu çalışmada kullanılan yapay öğrenme modelleri olan, Yapay Sinir Ağları, Rastgele Orman Algoritmaları ve Destek Vektör Makinelerinin teorileri kısaca açıklanmıştır. Bu çalışmanın yazılım dünyasına katkısı olarak görülebilecek Önerilen Çalışma bölümünde Savunma Sanayindeki projelerde kullanılabilecek Öznitelik Seçim sürecine, Kestirim Yöntemine ve Deneysel Sonuçlara yer verilmiştir. Ayrıca bu bölümde yer alan Öznitelik Araştırması kısmında da öznitelik korelasyon matrisleri oluşturulmuş ve aralarındaki ilişkilerin sonuca olan katkıları sunulmuştur. Son olarak Savunma Sanayi Projelerinde uygulanabilecek kendine has yeni bir öznitelik seçim stratejisi olan kestirim metodumuza bir örnek ile yer verilmiş ve gelecek çalışmalar planlanmıştır.

2. GEÇMİŞ ÇALIŞMALAR

Bu bölümde yazılım test ve yapay öğrenme algoritmalarının beraber kullanıldığı araştırma alanları hakkında geniş bir literatür araştırmasına yer verilmiştir. Literatürde yapılan çalışmalar yazılım test dünyasında açık olan problemlerin belirlenmesinde ve bu çalışmanın şekillenmesinde oldukça faydalı olmuştur.

Hourani [26], yapay zekanın yazılım testin geleceğine ışık tuttuğundan bahsetmiştir. Çalışmalarında ayrıca yazılım testlerinde yapay zeka kullanımı arttıkça çok daha tutarlı sonuçlar alınacağını, daha otomatik bir şekilde testlerin gerçekleştirilebileceğini ve yazılım geliştirme verimliliğinin de artacağını savunmuştur.

Durelli [16], makine öğrenme algoritmalarının, yazılım test faaliyetlerini iyileştirmek için nasıl kullanıldığını araştıran bir haritalama çalışması üzerinde durmuştur. 48 çalışma üzerinde yaptıkları araştırmada makine öğrenme algoritmalarının otomatik test senaryosu oluşturma ve iyileştirmesi, test odağı değerlendirmesi, test faaliyetleri efor tahmini gibi alanlarda temel olarak kullanıldığı belirtilmiştir. Bu çalışma ile yazılım test alanı ve makine öğrenme algoritmalarının nasıl kesiştiği ve literatürdeki mevcut durumları hakkında araştırmacıların bilgilendirilmesi amaçlanmıştır.

Turhan [43], yazılımlarda mevcut olan hataların olabilecek en erken aşamada tespit edilmesini sağlamak amacıyla bir hata kestirim metodolojisi üzerine çalışmıştır. Kaynak kod ölçütleri kullanılarak Bayes sınıflandırıcılar üzerinde çalışılmış ve geliştirilen modellerin kamuya açık veri setleri üzerinde performans değerlendirmesi yapılmıştır. Çalışmanın sonucunda yazılım test kaynak yönetimi üzerinde verimlilik sağlandığı vurgulanmıştır.

Briand [9], test senaryolarının iyileştirilmesi amacıyla yeniden yapılandırılması için çeşitli makine öğrenme modellerini kullanan bir metodoloji geliştirmiştir. Kara kutu test içeren bir vaka çalışmasında önerdikleri metodolojiyi kullanarak oldukça olumlu sonuçlar elde etmişlerdir.

Yazılım mühendisliğinde çoğunlukla gereksinim, test adımları, hata gibi metriklerin önem derecelerinin olmadığını ifade eden Zhang [48], yazılım mühendisliğinde uygulanan makine öğrenme metotlarının da bu ortamda sınırlandırıldığını gözlemlemiştir. Çalışmasında yazılım mühendisliğinin her alanına değer tabanlı (value-based) yazılım mühendisliği kavramını entegre ederek makine öğrenme algoritmalarının da bu alana kaydırılması gerektiğini savunmaktadır. Ayrıca, yazılım test verilerinin de değer tabanlı olarak üretilmesi için bir altyapı önermektedir. Önerilen altyapı ile beraber yazılım test verimliliğinin de artacağını savunmaktadır.

Cotroneo [13], geçmiş test deneyimlerini makine öğrenme algoritmalarına öğreterek, testler ilerledikçe test senaryolarının uyarlamalı olarak birleştirilmesi üzerine bir çalışma yapmıştır. Geliştirdiği metodolojiyi testler sırasında çevrimiçi olarak kullandığı için hata tespiti veriminde artış olduğunu deneyimlemiştir.

Zamli [3], test girdi uzayının örneklenebilmesi için kapsama kuvveti (variable strength) etkileşimli test senaryosu üreten bir strateji önermiştir. Bu stratejide, test senaryolarının en iyi boyutta olabilmesi için parçacık sürü optimizasyonu kullanılmıştır. Önerilen stratejinin özellikle ağ bileşenleri etkileşimlerinin test edilmesi gibi konularda başarılı olduğu gösterilmiştir.

Klasik Yazılım Hata Kestirim modellerindeki sınıf dengesizliği sorununu (class imbalance problem) çözmeye çalışan Pandey [37], makine öğrenme algoritmalarında, hata barındıran ve barındırmayan yazılım modüllerinin etkin bir şekilde ayrılmasını sağlayan özniteliklerin belirlenmesini sağlamıştır. Önerdikleri hata kestirim modelinde topluluk öğrenmesi (Ensemble Learning) ve derin temsil (Deep representation)'i birlikte kullanarak geleneksel modellere göre daha iyi bir başarım elde etmiştir.

Hunt [27], genetik algoritmalar kullanarak kontrol yazılımlarını test etmek üzerine çalışmıştır. Çalışma alanı için erken zamanlarda yapılan bu çalışmada insan tarafından yalnızca limitler belirlenip, yazılım girdileri ve çıktıları arasındaki bağlantı genetik algoritma tarafından değerlendirilip bir sonraki girdi bilgisine ulaşılmıştır. Çalışmada uygulanan hata arama modeli, tanımlanan hata senaryolarına göre kendini eğitmektedir.

Özakıncı [34], Erken yazılım hata kestirimleri üzerine çalışmıştır. Çalışmanın genelinde erken yazılım hata kestirim programlarının günümüzde yazılım kalitesine olumlu katkılarından bahsedilmiştir. Yapılan karşılaştırmalı çalışmada erken yazılım hata kestirimi için en çok kullanılabilir kestirim metotlarına, veri setlerine ve tasarım fazlarına yer verilmiştir.

Chi [12], sayısal çözümleme testleri için ilişki bazlı test durum önceliklendirmeleri üzerine çalışmıştır. Yazılım davranış haritalamalarının yanında önerilen yeni metot ile test kapsam alanlarının genişletilmesi önerilmiştir. Önerilen yeni metodun uygulanan yazılım büyüklüğü arttıkça daha iyi performans verdiği gözlemlenmiştir.

Lima [30], entegrasyonun devam ettiği yazılım projelerinde test durum önceliklendirmesi konusunda karşılaştırmalı çalışmalarda bulunmuştur. Devam eden projelerde test önceliklendirmenin son yıllarda ilgi çekmeye başladığını ve de çözüme açık birçok probleme sahip olduğuna dikkat çekmiştir.

Zhang [49], yapay sinir ağı kullanılarak emniyet kritik sistemler üzerindeki kontrol ve test yazılımları hakkında karşılaştırmalı bir çalışma yapmıştır. Faklı çalışma alanlarındaki güvenilirlik hayat döngülerini karşılaştırmalı tablolar halinde ortaya koymuştur. Genel geçer olarak kullanılabilecek endüstri uygulamaları değerlendirilerek, farklı alanlar için doğrulama adımları belirlenmiştir.

Bibi [8], yazılım hata tespiti için sınıflandırma ve sayısal çözümleme üzerinde çalışmıştır. Hata var/yok , 'Kaç tane hata var?' sorularına cevap aramanın yanında hataların olası nedenleri üzerinde de durulmuştur. Sonuç olarak farklı veri setleri üzerinde alınan sonuçlara yer verilmiştir. Kullanılan veri kümesinin sonuç üzerindeki etkinliğine dikkat çekilmiştir.

Arora [6], yazılım hata kestirimi konularındaki açık problemler üzerinde çalışmıştır. Hatalar üzerinde çeşitli parametrelerin değerlendirilebileceği üzerinde durulurken, performans üzerinde genel kabul görmüş bir çalışma olmadığını değerlendirmiştir. Farklı projeler arasında yapılan çapraz doğrulamaların ışığında, projeler arası kestirim yapabilecek daha geniş geçerliliği olan metotlara ihtiyaç olduğu değerlendirilmiştir.

Pandey [36], Bayes ağ sınıflandırmaları kullanarak yazılım hata kestirimlerinde bulunmuştur. Çalışmada, kullanılan veri setleri üzerinde yapılması gereken ön işlemenin önemi üzerinde durulmuştur. Çalışmanın sonucunda Bayesçi algoritmalar yerine kullanılacak rastgele orman algoritmaları veya destek vektör makinelerinin daha iyi sonuç verebileceği değerlendirilmiştir.

Gondra [23], yazılımlardaki hataya yatkınlık üzerine yapay öğrenme çalışmalarında bulunmuştur. Yazılım raporlarından alınan parametreler ve de hata raporları ile yapay sinir ağları üzerinde parametrelerin duyarlılık analizleri çıkartılmıştır. Büyük veri setleri kullanılarak yazılımların hatalı veya hatadan yoksun olduklarını bulmaya çalışan ikili sınıflandırma çalışmalarında bulunmuştur.

Xie [47], Yazılım kalitesi üzerinde kestirimler yapabilmek için test ve yapay öğrenme sınflandırıcılarını kullanmıştır. Yapay öğrenme ile yapılan testler ile elle yapılan testler arasında çapraz doğrulamalarda bulunulmuştur. Çalışmada önerilen yeni metot, açık kaynak kodlu yazılımlar üzerinde yapılan testlerde başarılı sonuçlara ulaşmıştır.

Xiao [45], yazılım testinden alınan geri dönüşler ile hata bulmak için entegre bir sistem üzerinde çalışmıştır. Geleneksel hata bulma metotlarına yazılım test sürecinde alınan geri dönüşleri ekleyerek yeni bir metot önermiştir. Önerilen yeni metot, geleneksel metotlara göre %40 oranında daha etkin sonuçlar vermiştir.

Alsolai [4], Yapay öğrenme metotları kullanarak, nesne tabanlı yazılımlar için ya-

zılım bakım kolaylığı hakkkında kestirimlerde bulunmuştur. Kullandığı metotta yazılım bakım kolaylığı ile eşleştirebileceği, yazılımın kullandığı veri kümesi ve çeşitli parametrelerin belirlendiği tümleşik ve yazılıma özel modellere yer verilmiştir. Çalışmanın sonucu olarak tümleşik modellerin yazılma has olarak hazırlanan modellere göre daha iyi sonuçlar verdikleri belirlenmiştir.

Garousi [44], doğal dil işleme kullanılarak yapılan yazılım test metotları hakkında geniş bir karşılaştırmalı çalışma yapmıştır. İncelediği tüm çalışmaların genel oylama metotları ile yazılım teste olan katkılarını ortaya koymuştur. Sonuç olarak yazılım test üzerinde harcanacak insan aktivitelerini azaltacak doğal dil işleme kullanılarak yapılan yazılım test metotları belirlenmiştir.

Literatürde yapay öğrenmenin yazılım test için araç olarak kullanıldığı birçok yenilikçi uygulama mevcuttur. Yazılım hatalarının yazılım test uzmanı davranışları izlenerek geri besleme alınıp hata kestirimleri yapıldığı uygulamaların [46] yanında, çeşitli hatalar içeren yazılımların yapay öğrenme algoritmalarına girdi olarak verilip hata kestirimlerinin yapıldığı çalışmalara da yer verilmiştir [31]. Ayrıca yazılım üzerinde hataya eğilimi ölçmek için yapay öğrenme algoritmaları ile karar verici etkinliği eğrilerinin (ROC curve) karşılaştırıldığı, ikili sınıflandırıcıların kullanıldığı karşılaştırmalı çalışma[1], yazılım test çalışmaları üzerinde yapay öğrenme algoritmalarının etkinliğini göstermiştir.

Literatürde araştırmaların genel olarak yazılım hata kestirimi, test girdi uzayının en iyi şekilde örneklenmesi, otomatik test senaryosu oluşturma gibi konularda yoğunlaştığı gözlemlenmiştir. Günümüz büyük çaplı yazılım projelerinde kaynak verimliliğini sağlamak için proje takvimlerinde önemli bir yeri olan yazılım test eforu kestirimi gibi konular hakkında sınırlı sayıda araştırma yapıldığı görülmüştür. Bu sebeple bu alanda yapılacak çalışmalarda açık olan problemlere cevap bulmanın yazılım test alanına önemli katkılar sağlayacağı düşünülmektedir. Yazılım test eforu ile ilgili yapılmış olan sınırlı sayıdaki çalışmaya aşağıda yer verilmiştir.

Bu çalışmanın konusu olan yazılım test eforu [19] genel olarak Test Nokta Analizi (Test Point Analysis), Kullanım Durum Noktaları (Use Case Points) üzerlerine yapılan yapay öğrenme iyileştirmeleri üzerine yoğunlaşmaktadır. Çözüm için çeşitli doğrusal dönüşüm algoritmaları [40] kullanılsa da, farklı veri kümesine sahip sistemler için genel geçer bir sonuca ulaşılamadığı [39] ve sınırlı veri kümesiyle beraber zor bir kestirim problemine sahip olduğumuz işaret edilmektedir. Bununla birlikte test altyapısının zorluklarına göre de kestirim metotlarında farklılıklara açık olarak ihtiyaç olduğuna işaret edilmiştir [29]. COCOMO (Yapısal Masraf Modeli) [7] ile deneye dayalı pratik çözümler de bulunmaya çalışılmış fakat endüstri düzeyindeki çalışmalar için sonuçlara yer verilememiştir [7]. Endüstri ve akademi düzeyindeki çalışmaların farklılıkları, iki ana çalışma alanındaki iş birli-

ğinin gerekliliğine işaret etmektedir [2]. Yapılan çalışmalarda test eforu kestirimi için yalın yazılım kodunun kullanılmasının yanında aynı zamanda yazılım testin karmaşıklığı, yazılımın test edilebilir olması, işlenebilir çıktı alınabilecek kısım sayısı, üretkenlik, test stratejisi, tecrübe ve insan sayısı gibi birçok öznitelik yazılım test eforunu tespit için kullanılmıştır [2]. Güvenliği kritik olan sistemlerde daha çok tercih edilen test şekli olan elle yapılan testler için yapılan çalışmada [41], testler sırasında doğal dil işleme teknikleri kullanılarak test ile ilgili anahtar noktalar belirlenmiş ve buna uygun regresyon algoritmaları ile bir test eforu kestirimi önerilmiştir.

Literatürde yazılım test alanında yapılmış olan çalışmaları inceledikten sonra yazılım test eforu kestirimi hakkında oldukça kısıtlı sayıda çalışma olduğunu farketmemiz üzerine bu konuda yapay öğrenme modelleriyle bir çözüm getirmeyi amaçladık. Böylelikle yazılım testi ve yapay öğrenme konularının bir arada yer alacağı şekilde hem literatüre bir katkı sağlamayı hedefledik, hem de pratik olarak kullanılabilecek ve başarılı tahmin sonuçları verecek bir çözüm modeli geliştirmeyi hedefledik.

İlerleyen bölümde problem kapsamı ve kullanılan tekniklerle ilgili bilgiler verilecektir.

3. YAZILIM TESTİ

Bu çalışmada yer alan ASELSAN bünyesinde geliştirilen projeler V yazılım geliştirme modeli [21] [38] uygulanarak geliştirilmiştir. Yazılımların fonksiyonel gereksinimlerinin karşılanıp karşılanmadığını görmek ve yazılımlarda bulunan olası hataları tespit edebilmek için kara kutu test yaklaşımı kullanılmıştır.

Yazılım testin kalitesini belirleyen temel faktör her zaman yazılımın tanımında yatmaktadır. Temel olarak nihai amaç, süreç sonunda yazılımı hata kalmamış bir hale getirmek olarak algılanmaktadır. Fakat testin temel amacı yazılımın yaratılış amacındaki tüm fonksiyonlara başarı ile sahip olup olmadığının anlaşılması olmalıdır. Yapılan testlerin amacı yazılıma ve görevlerine olan güveni arttırmaktır. Bazen bu amaçlar karıştırılabilmektedir. Yazılım test sürecinin amacı yazılımın ürün olarak kalitesine değer katmaktır. Test ile katılan değer kendini yazılım kalitesi ve sistem güvenilirliği olarak göstermektedir. Güvenilirliğin arttırılması, hataların bulunması ve ortadan kaldırılması ile gerçekleşir. Bundan dolayı hiçbir yazılım testçi yazılımın çalıştığını göstermek için test yapmamalıdır. Her zaman yazılımın içerisinde hatalar olduğunu önkoşul olarak kabul etmeli ve olabildiğince fazla hata bulabilmek için çabalamalıdır. Ayrıca kabul edilmelidir ki, hiçbir yazılım hatasız değildir.

Tüm bu altyapı ile yazılım test tanımını tekrar yapmamız gerekir ise, "Yazılım testi, hataları bulmak için yazılımı koşturma sürecidir" diyebiliriz.

Kelime oyunu gibi görünse de yapılan işin tanımı verilen eforu etkinleştirmede ve başarısında anahtar rol oynamaktadır. Mizaç olarak hedef odaklı çalışmaya yönelik bu tutumun aynı zamanda çalışma temposunda pozitif etkileri de görülmektedir. Günümüz yazılım endüstrisinin dinamik ve zaman ile yarışan proje takvimlerinde maalesef önceden karar verilmiş son teslim tarihlerinin de etkisi ile hata olmadığını göstermek amaçlı test altyapıları ve kasıtsız olarak hatanın çıkma olasılığının az olacağı veri girdi setleri sıklıkla görülmektedir. Bu yaklaşım yazılıma hiçbir şey katmamakla beraber yazılımın olgunlaşmasına da katkıda bulunmayarak tüm süreci sabote etmektedir.

Test süreci birçok kez yıkıcı ve sadistik sıfatlar alabilmektedir. Bundan dolayı birçok kişi tarafından işlevi zor bir hal alabilmektedir. İnsan daha çok yapıcı bakmaya alışmış beşeri bir varlık olduğundan iş yeri sorumlulukları ile günlük hayat bakış açıları birbirlerinin içine geçebilmektedir.

Test tanımı aynı zamanda test vakalarının nasıl tasarlanacağının ve bunları kimlerin test etmesi gerektiği sorusuna da cevap bulmalıdır. Uygun test tanımını pekiştirmenin başka bir yolu da proje yöneticilerince sıklıkla kullanılan başarılı ve

başarısız terimleridir. Bu terimler çoğu kez test yazılım sahipleri ile proje yöneticileri tarafından birbirlerine ters anlamlarda kullanılmaktadır. Daha önce de bahsedildiği gibi hata bulma süreci yazılımı, daha iyi ve kararlı bir hale getirmek amaçlıdır. Bu amaca hizmet eden test yazılımları, test yazılımcısı ve herkes tarafından başarılı olarak adlandırılmalıdır. Giderilen her hata bu amaca hizmet eder. Herhangi bir hata bulunamadığı durumları 'sağlama' olarak değerlendirmek doğru değildir. İyi bir test yazılımının, yazılımı olgunlaştıramaması düşünülemez.

Durumu daha iyi anlatabilmek için bir örnek vermek gerekirse; hafif keyifsiz olduğu için veya uzun yaşam konusunda yaşamsal değerlerini yukarıya çekmek için doktora muayene olmak için gitmiş bir hastayı düşünelim. Doktorlar laboratuvar sonuçlarında herhangi bir sonuç bulamadığında tüm bu sürece başarılı demek, herkesin kabul ettiği gibi, doğru değildir. Hasta para ve zaman kaybetmiştir. Sağlığını geri kazanma veya uzun yaşam konusunda kendisine herhangi bir şey katamamıştır. Bu durumda hasta kendisini doktorun yeteneklerini sorgulama konusunda haklı bulabilir. Fakat laboratuvar sonuçları doğrultusundaki bulgular eşliğinde uygun bir tedavi ile hastanın iyileşme ve uzun yaşam yolculuğu başlayabilir. Tıp endüstrisinde doğru şekilde genel kabul bulmuş bu terminoloji tüm yazılım endüstrisi paydaşları tarafından da bu şekilde kabul edilip kullanılmalıdır.

Tanımdan gelen başka bir problem de yapılacak olan testlerin belirli bir süreç sonucunda hatalara sahip olmadığını göstermesidir. Psikolojik çalışmalar göstermiştir ki, insanlar mümkün olduğunu düşünmediği bir görevi dar bir zamanda üzerlerine aldıklarında görev konusunda hiçbir zaman başarı gösterememektedirler. Buna kıyasla görev konusunda motivasyonu iyi seviyede olan kişilere uzun süreler verilse bile, aynı dar zamanlar içerisinde görevlerinin büyük bir kısmını tamamladıkları gözlemlenmiştir. Devamlı hataları bulmak üzerine yapılan yazılım test çalışmaları, tüm yazılım projeleri için daha uygun bir felsefe içermektedir. Bu felsefe yukarıda bahsedilen psikolojik engeli de çalışanların üzerinden kaldırmaktadır [32].

Tanım olarak ele alınabilecek son problem: "Yazılımın, yapması gereken görevleri yerine getirmesi" dir. Kısaca, yazılım görev çıktılarını veremiyor ise hatalıdır diyebiliriz. Bununla birlikte, vermesi gereken çıktılarının yanında başka çıktılar da veriyor ise yine hatalıdır demeliyiz.

Özetlemek gerekir ise, başarılı bir test senaryosu yazılımın üzerinde hatalar bulmak üzere kendini evrimleştirir. Doğal olarak test yazılımının amacı yazılım üzerinde güven sağlamak ve yazılım altyapısını güçlendirmektir. Yazılımın gerekli görevleri ve yalnızca verilen görevleri yerine getirdiğiden emin olmak da önemli bir yazılım test çıktısıdır. Bu amaçlara ulaşmanın tek yolu itina ve çaba ile hataları aramaktır.

"Benim yazılımım mükemmel (hatadan yoksun) ! " diyen bir yazılım tasarımcısına yapılabilecek en büyük yardım çaba ve itina ile geçecek bir yazılım test sürecidir.

3.1 Test Metotları

3.1.1 Kara kutu testleri

Fonksiyonel test metotları ailesinin en çok kullanılan üyelerinden biri olan Kara Kutu Testleri [10] [17] [32], tüm iç mekanizmalara aldırmadan verilen veri kümesine göre alınan çıktıları ve icra koşullarını esas görür. Kara kutu testleri genel olarak sistem seviyesinde ve entegrasyon aşamalarında en çok kullanılan test metotlarından biridir. Yazılım test konseptine indirgendiğinde, özellikle raf ürünü olan ve kaynak kodlarının bulunmadığı 3. kişilerce yazılmış yazılımların sistem entegrasyonlarında yazılımın kararlılığı için oldukça faydalı ve kullanışlı bir metot olarak kendini göstermektedir. Kısaca görevleri:

1) Kara kutu testleri iç mekanizmalara aldırış göstermeden yalnızca verilen girdilere göre alınan çıktılara ve icra koşullarına dikkat eder.

2) Sistem içerisinde entegrasyon uyumluluğunu ve önceden karar verilmiş fonksiyonel gerekliliklerin yerine getirilip getirilmediğini test eder.

Kara kutu testlerinin temel amacı yazılımın verilen görevleri yerine getirip getirmediğini test etmektir. Bu test metodu birim testlerinde, entegrasyon testlerinde, sistem testlerinde, müşteri kabul testlerinde kullanılabilir. Fakat her bir seviye için ayrı gereklilikler belirlenmelidir. Örneğin entegrasyon aşamasındaki bir proje için kullanılan kara kutu testleri birbirleri ile entegre olan modüllerin fonksiyonel davranışlarına odaklanırken, sistem seviyesindeki kara kutu testleri sistem performansına ve güvenilirliğe odaklanmaktadır. Kara kutu testleri yazılım sürecinde farklı alanlarda farklı yaklaşımlar göstermekle birlikte uzun süredir kullanılmaktadır. Kullanılan teknikler kısaca 3 gruba ayrılabilir:

1) Kullanım bazlı kara kutu testleri: Rastgele testler veya istatistiksel testler.

2) Hata bazlı kara kutu testleri: Bu tekniği kullanan kara kutu testleri genel olarak hata bulmak üzere kendini evrimleştirir. Her bir iterasyon için, yazılım, kullanıcı arayüzüne hükmeden test sorumlusunun adımlarına göre şekillenir. Bu yazılım metodolojileri eşit ayırım testleri, kategori ayırım testleri, sınır değer analizleri, karar tablo testleri vb. gibi çeşitlere ayrılabilir.

3) Arıza bazlı kara kutu testleri: Üzerinde işlem gören sistemin hatalarını bulmak üzere evrimleşir. Bu tip karakutu testleri genel olarak üzerinde çalıştığı yazılımın kaynak kodlarına ihtiyaç duyar ve yazılım parçaları yerine geçebilecek değişken altyapılar oluşturur. Bu altyapıların özellikleri genel olarak arayüz yardımı ile farklı özellikler kazanabilir.

Yeni bir yazılım test süreci özgül isterlere göre hızlıca kara kutu testlerini esas alarak başlangıç bulabilir. Bununla birlikte aynı kısım için tekrar eden yazılım parça testleri yazılım test süreci için çok uygun bir durum değildir. Geleneksel kara kutu testlerini uygulama aşamasında çeşitli durumlar göz önüne alınmalıdır.

Kaynak koduna erişim: Genel olarak kullanılan raf ürünleri nedeni ile kaynak kod erişimleri mümkün olmamaktadır. Bundan dolayı kara kutu testleri bu gibi durumlar için en uygun seçenek haline gelmektedir. Kaynak koduna ulaşamamak yalnızca gerekli yazılım dilindeki orjinal algoritmaya ulaşımın mümkün olmamasının yanında aynı zamanda 3. kişilerce hazırlanmış yazılımın test logları, test ortamları, test metodolojileri ve doğrulama altyapılarına da ulaşımın mümkün olmadığı bir duruma bizi sürükler. Bu gibi durumlar kara kutu testlerinin doğrulama zamanları ve test etkinliklerinde üstün rol oynar.

Bileşen özellikleri, klasik yazılım sistemlerinde her bir iç modül için önceden tanımlanmıştır ve sistemin paydaşlarına açık olarak kendilerini göstermektedirler. Bu nedenle sistem paydaşlarının entegre olduğu dönemde iç modüllerin entegrasyonu hakkında herhangi bir şüpheye yer kalmaz. Bununla birlikte, parça tanımlı testlerin genel olarak iki detayı mevcuttur. 3. kişilerce yazılmış raf ürünleri kullanıldığından genel bir özelleşmeye tabii alt parçalar kullanılır. Tekrar kullanılabilir raf ürünleri bazı zamanlar parça olarak kullanıldığında gündemdeki yenilikleri takip edemeyip yazılım kökenli problemlere neden olabilmektedir. Ayrıca kullanıcılar genel olarak sistemlerine bu tip raf ürünlerini entegre etmeye çalıştıklarında seçim yapılırken bileşen özellik isterlerinde birebir uyum sağlamak oldukça zordur. Bu tip raf ürünü sağlayıcılarının sağlayabildikleri test altyapıları ve sistem mimarisi tarafından istenebilecek altyapıların uyumlarını sağlamak başka bir zorluk olarak kendilerini göstermektedir. Hazır raf ürünü kullanıcılarını bekleyen bu gibi zorluklar kara kutu test yazılımlarında da farklı boyutlarda değişimlere sebep olmaktadır.

Bileşenler test edilirken bileşen kullanıcıları iki adet yol seçebilir. Teste en aşağı seviyede başlamak veya bileşeni özelleştirmek ve özelleştirmenin yazılım bileşenini olumsuz etkilemeyeceğinden emin olmak. İlk seçenek için en büyük olumsuzluk gereken yüklü iş gücüdür. İkinci seçenekte bileşen yazılım kullanıcılarının, bileşen yazılım testlerini daha etkin kullanabilecekleri düşünülmektedir fakat yeterlilik konusunda ve test eforlarının yeniden kullanımı, bileşen kullanıcılarına

hangi formatlarda açılmasının doğru olduğu gibi yeni ve zorlayıcı sorular kendilerini göstermektedir.

Bileşen arayüzü: Geleneksel yazılım sistemlerinin aksine, bileşen bazlı yazılım türlerinde arayüz aracılığı ile oldukça esnek bir formda yazılım test sistemleri koşturulabilmektedir. Örneğin genele açık bir nesne aracılığı ile yazılımın genele açık verilerine kolayca ulaşım sağlanabilmektedir. Bileşen bazlı yazılım sistemlerinde arayüzler maalesef tek iletişim noktalarıdır. Böylelikle bileşenler arasındaki iletişim de arayüz üzerinden sağlanmaktadır. Bundan dolayı arayüz özellikleri yazılım test üzerinde anahtar rol oynamaktadır. Eğer doğru zaman verilirse ve testler çaba ile yürütülürse, etkin ve etkili bileşen testleri, arayüz üzerinden gerçekleştirilecek kara kutu testleri ile gerçekleştirilebilir.

Yazılım parçaları için, arayüzler tek etkileşim noktaları olmakla beraber kara kutu yazılım test altyapıları için çeşitli özel altyapılara izin verilmelidir. Bunlar arayüzün çeşitli parçalarının nasıl çağırılacağı ile başlar ve iç arayüzlerin doğrudan çağırılması için özel komutların varlığına kadar özelleştirilebilir. Ek bilgiler ve analiz verileri de arayüz tarafından tutulacak şekilde özelleşebilir.

Bununla birlikte bileşenlerin müşteri isteğine göre özelleştirilmesi ve sertifikasyon bilgilerinin bile eklenmesi yönünde gidilebilmektedir. Güçlü bir test altyapısı, yazılım test sonuçlarını almadan yazılıma olan güvenin artmasına yardımcı olmaz.

3.1.2 Rastgele test

Doğru rastgele test altyapısı için öncelikle operasyon öncesi profil oluşturulmalıdır. İyi bir operasyonel girdi profili oluşturmak da gerçek sistem kurulumu yapılmadan kullanıcı gerçek sistemleri ile çalışmaya başlamadan oldukça zordur. Rastgele testler ayrıca hazır yazılım blokları kullanılan sistemler tarafından da etkin bir şekilde kullanılabilir. Burada yapılan testler bazen yazılım bloklarının kendilerinde bazen de birleşen bütünün tamamına uygulanabilmektedir. Bazı hazır yazılım bloklarının tespitleri de yazılım blok ürünü ile birlikte gelmektedir.

Bu durumlarda da ek olarak yapılacak testlerin planlanması gerekmektedir. Ek olarak planlanan testlerin haritasını belirleyecek faktör daha önce de belirtildiği gibi operasyonel girdi dağılımı olarak kendini göstermektedir. Kullanılan hazır yazılım bloklarının yeni versiyonlarının çıkması durumunda, yeni bir veri kümesi oluşturup yeniden deneme testlerinin yapılması çoğunlukla standart prosedür haline gelmiştir.

Kullanılan kullanıcı profilinin değişmesi veya sistemin kullanım alanının değişmesi durumlarında ise profile göre yeniden test veri vaka setleri oluşturulup, test yapılmalıdır. Yapılmayan test alanları da ayrıca yeniden oluşturulabilir.

3.1.3 Kategori parça testleri

Kategori parça testleri orjinal fonksiyonel testler ve özellikleri ile başlar. Her bir alt yazılımın özellikleri ve detayları ile devam eden bir prosedürü takip eder. Kategori parça testleri genel olarak 7 adımı takip eder:

1) Fonksiyonel isterler üzerinden tüm yazılımı fonksiyonel birimlere ayırmak

2) İlgili parametreleri ve çevresel koşulları tanımlamak

3) İlgili kategori bilgilerini belirlemek

4) Her bir kategoriyi farklı seçenekler için kısımlara bölmek

5) Her bir kısım için parça özellikleri belirlemek

6) Sistem test iskeletini belirlemek

7) Gerekli test vakalarını belirlemek

3.1.4 Sınır değer testleri

Kategori parça testleriyle karşılaştırıldığında girdi alanları farklı, birbirleriyle bağlantısız parçalara bölmeye çalışılır. İdealde ise her bir parçanın hatayı ortaya çıkarma veya saklama olasılığı aynıdır. Fakat pratik olarak bu doğru değildir.

Sınır değerlerine daha yakın olan testlerde istatistiksel olarak hata oranı daha fazladır. Örneğin, pratik olarak 100 km/saat hız yapabilen bir aracın farlarının yanmaması durumu, 100 km/saat hızla giderken yapılan testlerde daha sıklıkla görülür. Araç dururken farların çalışmaması istatistiksel olarak daha az görülmektedir. Bundan dolayı sınır değer testlerinde bir elementten daha fazla sayıda testin aynı anda yapılması hata bulma olasılığını artıracaktır.

Sınır değer testleri hızları ve etkinlikleriyle özellikle yüksek riskli alanlarda göz doldurmaktadır. Kategori parça testleri yapılacak toplantı sayısını azaltma yönünde etkin iken, sınır değer testleri yapılan testlerin etkinliğini ve de yapılan testlerin güvende olmasını sağlar. Bundan dolayı iki stratejiyi beraber kullanmak oldukça avantajlıdır.

3.1.5 Karar tabloları testleri

Kategori parça testleri ve sınır değer testleri tüm kullanılan test kategorilerinin arasında bir korelasyon olmadığını öngörür. Diğer bir deyişle eğer analizde kullanılan değişkenler birbirleri ile ilişkili ise kullanılan sınır değerleri gerçekçi olmayabilir. Karmaşık senaryoların çeşitli mantıksal önermeler ile şekillendiği bazı yazılım sistemleri de karar tablosu bazlı test stratejilerine ihtiyaç duyar. Karar tabloları iki kısma sahiptir [32]:

1) Farklı durum kombinasyonlarının sebep olduğu sıralı çıktılar.

2) Farklı aksiyonların neden olduğu yeni durumlar.

Örneğin a, b, c kenarlarına sahip bir üçgen problemini düşünelim. Verilen a, b, c kenar uzunlukları bir üçgen oluşturabilir mi? Cevap evet ise, ne tür bir üçgen oluşturabilir? Çizelge 3.1'de buna uygun karar tablosuna yer verilmiştir.

Bu problemlerin çıktıları, kenarlar arasındaki ilişkilere bağlı olduğu için girdileri ayrı parçalara bölmek oldukça zordur. Bundan dolayı sınır değerlerini atamak kolay olmayacaktır. Bu gibi problemlerde karar tablosu testleri bizlere kolaylık sağlamaktadır.

Çizelge 3.1: Karar Tablosuna bağlı test örneği.

Durum	$D_0 : a > b > c > 0$	T	T	T	T	T
	$D_1 : a < b + c$	F	T	T	T	T
	$D_2 : a = b$	-	T	T	F	F
	$D_3 : b = c$	-	T	F	T	F
Aksiyon						
	Üçgen değil	X				
	Çeşitkenar					X
	İkizkenar			X	X	
	Eşkenar		X			

3.1.6 Beyaz kutu testleri

IEEE Standartları'na [5] göre beyaz kutu testleri, aynı zamanda yapısal test veya saydam kutu testleri olarak da adlandırılır. Teste tabi tutulan iç mekanizmalara da erişimin mümkün olduğu sistem veya yazılım parçalarında kullanılır. Kara kutu testlerinden bu yanı ile ayrılır. Kara kutu testleri ile işlevsel olarak ayrıldığı başka bir yer de yazılım içerisinde gerçekleşebilecek hataları bulabilecek altyapıyı bize sunmasıdır. Örneğin, bir mantık hatası, oluşabilecek hataların %50'sine neden olabilir. Bunun da temel problem olarak tespiti kara kutu testinde mümkün değildir. Bunun için kullandığımız yazılım bileşenlerinin özellikleri yazılım test stratejimizde önemli rol oynamaktadır.

3.2 Yazılım Test Planı ve Süreci

Yazılım projelerinde kullanılan test planları genel olarak detaylı ve kompleks dokümanlardır. Planlama yaparken bazı soruların cevapları proje takvimleri ve de insan kaynağı için kritik öneme sahiptir.

1) Test amacı nedir? Test sonuçlarında ulaşılacak olan nedir? Test sırasında alınan riskler nelerdir?

2) Test edilecek olan nedir? Hangi kalemler, özellikler, prosedürler, fonksiyonlar, nesneler, kümeler ve alt sistemlerin test edilmeye ihtiyacı var?

3) Test sorumlulukları kimlerde olacak?

4) Nasıl test edilecek? Hangi stratejiler, metotlar, donanımlar, yazılımlar ve teknikler uygulanacak ? Hangi test sonuç dokümanları düzenlenecek?

5) Ne zaman test edilecek? Test için takvim ne olacak? Hangi tarihte hangi kalemlere ihtiyaç olacak?

6) Ne zaman test sonlandırılmalı? Her zaman tüm problemlerin çözümlendiği test sonuçları proje takvimine ve bütçesine uygunluk göstermeyebilmektedir. Proje ödemeleri, takvim, sözleşmeler, müşteri teslim tarihleri, özel koşullar bu sorunun cevabında etkili olmaktadır[11].

Planlama safhasında test takımı hangi dokümantasyonların test sürecinin temellerini oluşturacağına karar verir. Bu safhadaki temel aktivite ihtiyaç bulunan testlerin gözden geçirilmesidir. Yapılan testlerin tamamında aranan temel unsurlar bütünlük, açıklık ve tutarlılıktır. Bu unsurların temelde öncelik olarak kabul edilmesi başarılı bir test sürecinin anahtarıdır. Planlama safhası başlangıç olarak seçilmiş temel test dokümantasyonlarının kalite kontrolü olarak da görev görür.

Test tasarım sürecinin ilk çıktısı test tasarım temellerinin atılmasıdır. Test için ilk fırsat budur. Kusurlar ne kadar hızlı bulunursa onları düzeltmek de bir o kadar ucuz ve kolay olacaktır. Sonrasında bulunan kusurlar temel ve bağlı olduğu çeşitli entegrasyon fonksiyonlarıyla beraber testlerin yeniden yapılmasına neden olacağından önemli zaman kayıplarına neden olabilmektedir.

İlgili dokümanların seçimi, test vakalarına karşılık gelecek dokümantasyonları işaret etmelidir. Bununla birlikte test plan hazırlığı ve test safhaları çoğunlukla iç içe girmektedir. Genelde test plan hazırlığı yaşayan bir süreç olarak işlem ve işlev görmektedir. Bitmiş bir test planına eklentiler süreç içerisinde yapılabilmektedir.

Test gözden geçirmelerine genel olarak bir kontrol listesi eşlik eder. Bu kontrol listesi kullanılan tasarım tekniklerine göre farklılık gösterebilmektedir. Test planı ortaya çıkınca kullanılacak test tasarım teknikleri ile beraber hangi kontrol listesinin hangi test parçalarına uygulanacağı da ortaya çıkar.

Dokümantasyon değerlendirmesi, kontrol listelerinin hazırlığı sonrası test takımı her bir bulunan hata için rapor çıkarmaktadır. Sistem entegrasyonu tamamlanmadan bulunan hatalar yapılan test kalite temelini arttırmaktadır. Ayrıca sistem altyapısı ve büyüklüğü hakkında önemli bulgulara da varılır ve test planını şekillendirir.

Sonuçların raporlanması, bulunan hatalar dizisi sonrası test takımı tarafından test sonuç raporu başlığı altında bir doküman hazırlanır. Bu raporda verilen başlıklar sırasıyla kısaca: yapılan işin özeti, test temellerinin tanımı, test takımı ve görev tanımları, test sorumlusu ve de test sonuçlarıdır. İlk bulgular sonucundaki risk analizine de yer vermeli ve de var ise ilgili tavsiyeler ile son bulmalıdır. Son olarak kullanılan tüm kontrol listeleri de rapora ek olarak eklenmelidir [10].

Çoğu sistemler çeşitli alt sistem ve onun altında da bir veya birden fazla katmanlardan ve destek modüllerinden oluşur. Kullanıcıların arayüz ile yaptığı işlemler alt katmanlardaki diğer alt sistemler ile bağlantı kurup gerekli fonksiyonları yerine getirmeye çalışır. Sistemin alt katmanlarında kullanılan bağlantılar ne kadar fazla ise hataları bulma ve izole etmek bir o kadar zor olmaktadır.

Yazılım görevlerinden birini örneklemek gerekirse, ilk olarak yazılım arayüzü kullanıcıdan girdisini alır. Bu görev yazılımın çeşitli katmanlarında kullanılır, işlenir ve veri tabanına kayıt edilir. Daha sonra başka bir destek sistemi veya alt sistem tarafından da kullanılabilir. Eğer yapı süreç akış veya eşzamanlılık problemleri ile karşılaşılır ise bu hatanın yer tespiti ve yeniden tekrarlanması oldukça zor olmaktadır.

Yazılım test mimarisi tasarlanma safhasında yazılım test sorumlularının bir girdiyi sistem içerisinde nasıl takip edebilecekleri açık ve net olmalıdır. Örneğin bir girdi sonrası yardımcı bir görev yazılımı başka bir sunucuda işlem görüyor ise yazılım test uzmanının gerekli hak tanımları önceden düşünülmüş ve kontrol için gerekli altyapının tanımlanmış olması gereklidir. Eğer teklif edilen yazılım mimarisi bu gibi doğrulamalara izin vermiyor ise daha güvenilir ve teste uygun altyapıları mimariye dahil etmek gereklidir. Bu gibi iyi düşünülmemiş yazılım mimarileri yazılım test aşamasında ayrı alt sistem sorumlularının da dahil olduğu uzun ve yorucu süreçlere neden olmaktadır.

Üçüncü kişilerce yazılmış ürünleri ele almak gerekirse kaynak kodunun müsaitliği ve değiştirilebilirliği sorgulanmalıdır. Yazılım test altyapılarına kucak açan üçüncü kişilerce yazılmış alt sistemlerin kullanımı kritik önem taşımaktadır. Tüm sistemler için geçerli olan birşey vardır ki, akış mimarisinin şeffaf olmadığı yazılım iskeletleri güvenilir ve kararlı sonuçlar vermez.

Tüm bu tedbirler ile yazılım testi sırasında karşımıza çıkabilecek birçok olası zorluk ve süprizin aşılmasında kolaylık sağlanacaktır. Ayrıca eğer test mimarisinde tam olarak anlaşılmayan noktalar var ise test ekibi denemeler yapabilmek için prototip yazılım konusunda ısrarcı olmalıdır. Verilecek geri dönüşler yazılım kalitesinin yukarı çekilmesinde kesinlikle etkili olacaktır [17].

Bu bölümde yazılım test tanımı ve sürecinden bahsedilmiştir. Bu çalışmanın amacı olan yazılım test efor kestirimini, doğru bir şekilde yapabilmek için farklı yapay öğrenme algoritmalarına başvurulmuştur. Bir sonraki bölümde bu yapay öğrenme algoritmalarının teorilerine kısaca yer verilmeye çalışılmıştır.

4. YAPAY ÖĞRENME MODELLERİ

Pratik problemleri çözmek için makineleri kullanmadan önce girdi değerlerine göre alınması gereken çıktılar açık bir fonksiyon haline getirilmelidir. Yazılım test mimarisinde temel amaç bu fonksiyonu bilgisayarın anlayacağı dillerde sıralı yazılım talimatları haline getirmektir. Giriş ve çıkış talimatlarının belli olduğu öğrenme metodolojileri kümesine denetimli öğrenme denilmektedir. Örnek olarak sağlanan giriş-çıkış verilerine de genel olarak eğitim verileri denilmektedir.

Giriş-çıkış verileri bu şekilde ifade edildiğinde yapılacak olan sıralı işlemler bir başvuru çizelgesi ile çözülecek gibi düşünülse de tüm verilerinin üzerine eklenen bilgi gürültü faktöründen dolayı işimiz çok da kolay olmamaktadır. Tüm fonksiyonların temele indirgendiği ve açıkça tüm değişkenleri ile ifade edildiği giriş çıkış ilişkisi hedef aksiyon veya karar fonksiyon olarak ifade edilir. Bir çözüm olarak alınacak fonksiyon aday olarak seçilmiş bir çok hipotez arasından teste sokulup en iyi sonuçları veren arasından seçilir.

Basitçe eğitim verilerini girdi olarak alıp bir çok hipotez arasından en doğru olanı seçen bu yapıya yapay öğrenme algoritması denir.

Örnek olarak birçok kümeyi seçilen özellikleri ile ifade etmeye çalışıp, birbirinden ayırıp, tanımaya çalıştığımız algoritmalara sınıflandırma algoritmaları denir iken; örneğin protein tiplerini algılamak için yazılmış milyonlarca çıktı olasılığına sahip algoritmalara ise regresyon algoritmaları denir.

Çalışma sırasında kullanılan teorik altyapılara kısaca yer verilmeye çalışılacaktır. Gerekli altyapı ve fonksiyonlar kullanılan veri altyapısına uygun olarak düzenlenmiştir.

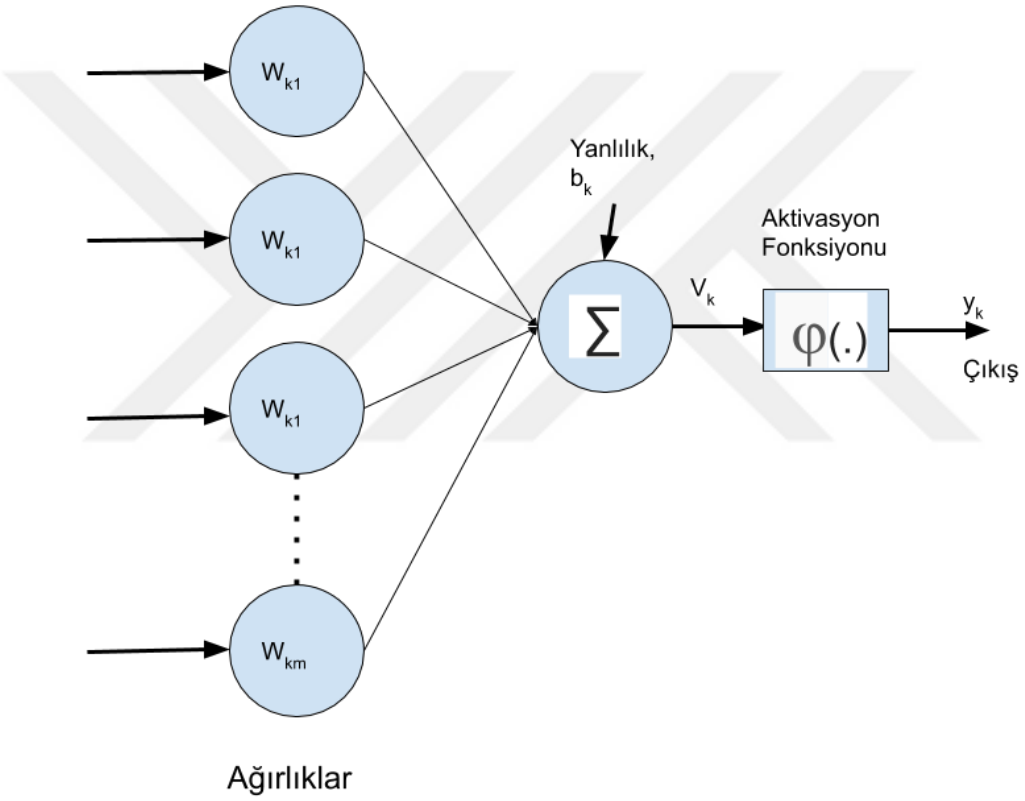
4.1 Yapay Sinir Ağları

Yapay Sinir Ağı, Geri Yayılım Algoritması [15], en basit ve en çok kullanılan denetimli metotlardan biridir. Bu algoritmadaki temel yaklaşım önce eğitilmemiş bir yapay sinir ağı ile başlayıp yapay sinir ağının değişim ağırlıklarını istenilen sonuca göre güncellemekten geçer.

İnsan beyninin geleneksel dijital bilgisayardan tamamen farklı bir şekilde işlediği bilindiğinden yapay sinir ağları üzerindeki çalışmalar her zaman ilgi çekici olmuştur. Çevresindeki ortama uyum sağlamak için gelişen bir sinir sistemi beynin esnekliğini anlatır. Bu konudaki araştırmalar insan beynindeki bu birimleri modellemeye çalışıp bilgiyi işleme alan nöron mimarileri üzerindedir. İnsan beynin-

deki yapıya benzer bu yapılara yapay sinir ağları denmektedir. Bu ağlar genellikle elektronik bileşenler kullanılarak uygulanır veya bilgisayar kullanarak dijital olarak yazılımda benzetimi yapılır.

Bir sinir ağı, bilgi işlem gücünü öncelikle kendi içinde büyük ölçüde paralel dağıtmış olan yapısından ve ikinci olarak öğrenme yeteneğinden alır. Bu iki yetenek sinir ağlarının bilgi işleme yeteneklerini belirler ve çözülemeyen karmaşık ve büyük ölçekli problemlere çözümler bulabilmesine elverir.



Şekil 4.1: Doğrusal olmayan ağ modeli

Şekil 4.1'in nöron modeli, b_k ile gösterilen harici olarak uygulanan bir girdi içerir. Bu girdi, sırasıyla pozitif veya negatif olmasına bağlı olarak aktivasyon fonksiyonunun net girişini artırma veya azaltma etkisine sahiptir.

Matematiksel olarak Şekil 4.1' da görülen ağı, Denklem 4.1, ve 4.2 ile gösterebiliriz.

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{4.1}$$

$$y_k = \phi(u_k + b_k) \tag{4.2}$$

$x_1, x_2, ..., x_m$ giriş sinyalleri,

$w_{k1}, w_{k2}, ..., w_{km}$ nöron k'nın ilgili ağırlıkları,

u_k doğrusal birleştirici,

b_k yanlılık sabiti,

$\phi(.)$ aktivasyon fonksiyonu,

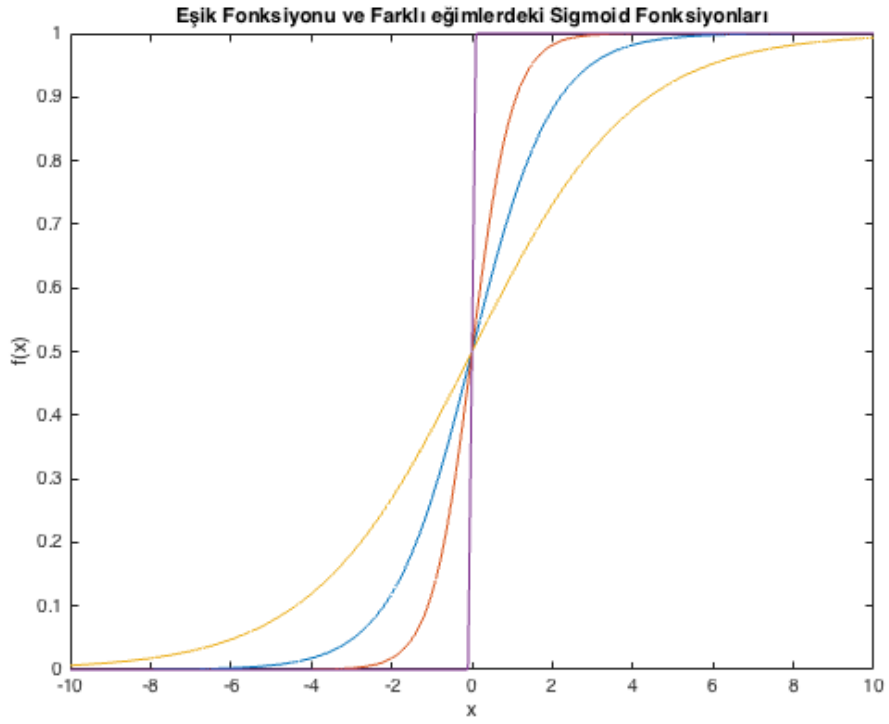
y_k nöronun sinyal çıktısıdır.

b_k nın kullanılması, u_k eklenen doğrusal bileştirici sayesinde Şekil 4.1'de de görüldüğü gibi Denklem 4.3, 4.4 olarak gösterilebilir.

$$v_k = u_k + b_k \tag{4.3}$$

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{4.4}$$

Şekil 4.1'de görülen ϕ aktivasyon fonksiyonu, işlenmiş olan sinyali iki şekilde değerlendirebilir: Eşik fonksiyonu ve sigmoid fonksiyonu. İlgili Fonksiyonlar Şekil 4.2'de gösterilmiştir.



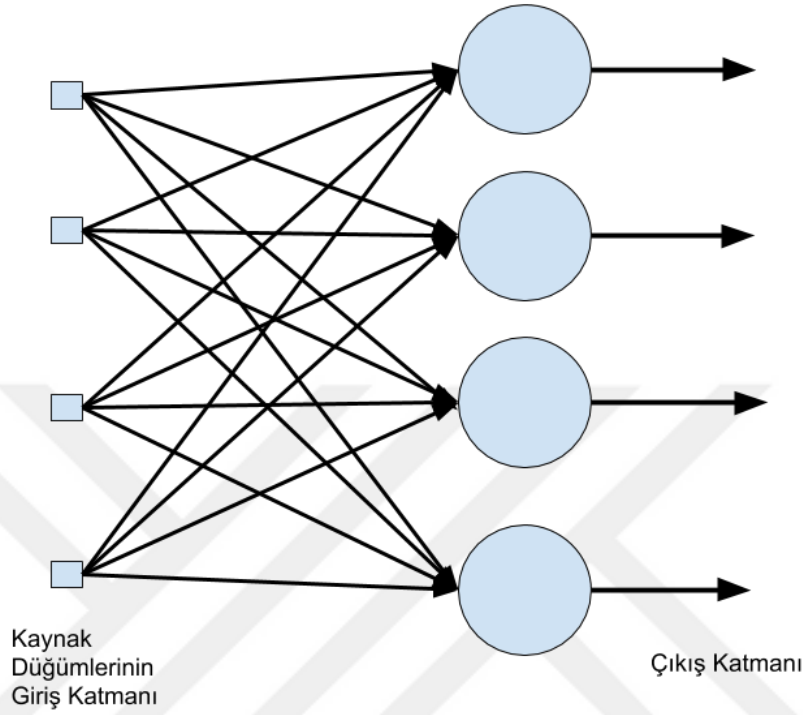
Şekil 4.2: Normalize Edilmiş Eşik fonksiyonu ve Değişen Eğimlere göre Sigmoid Fonksiyonu

4.1.1 Yapay sinir ağı mimarisi

Bir sinir ağının nöronlarının yapılandırılma şekli, ağı eğitmek için kullanılan öğrenme algoritması ile yakından bağlantılıdır. Yapay Sinir Ağı mimarilerinin temel olarak farklı üç sınıfından bahsedebiliriz:

Tek Katmanlı İleri Beslemeli Ağlar: Katmanlı bir ağın en basit formunda, doğrudan nöronların bir çıkış katmanına yansıtılan, ancak tam tersi olmayan bir kaynak düğümleri giriş katmanına sahibiz.

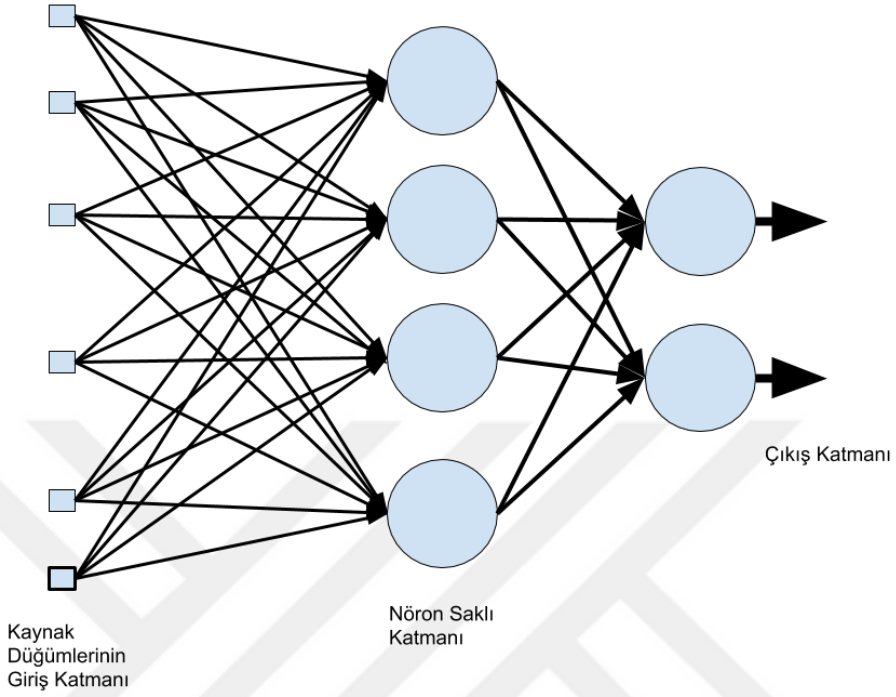
Çok Katmanlı İleri Beslemeli Ağlar: hesaplama düğümlerine karşılık gelen gizli nöronlar veya gizli birimler olarak adlandırılan bir veya daha fazla gizli katmanın varlığı ile diğerlerinden ayrılır. Gizli terimi sinir ağının bu kısmının, ağın girişinden veya çıkışından doğrudan görülmediği gerçeğini ifade eder. Gizli nöronların işlevi, harici giriş ve ağ çıkışı arasında ağa yararlı şekillerde müdahale etmektir. Bir veya daha fazla gizli katman ekleyerek, ağ girdisinden daha yüksek dereceli istatistikler çıkartılabilir.



Şekil 4.3: Bir katmanlı yapay sinir ağı

Ağın giriş katmanındaki kaynak düğümleri, ikinci katmandaki hesaplama düğümlerine uygulanan giriş sinyallerini oluşturan aktivasyon desenine giriş vektörü olarak görev görür. İkinci katmanın sinyalleri üçüncü katmana giriş olarak kullanılır ve bu kurgu ile ağ altyapısı devam eder. Şekil 4.4 deki mimari grafik, tek bir gizli katman durumunda çok katmanlı ileri beslemeli bir sinir ağının yerleşimini göstermektedir.

Tekrarlayan sinir ağları, geri besleme döngülerine sahip olması nedeniyle kendisini ileri beslemeli sinir ağlarından ayırır. Geri besleme döngülerinin varlığı, ağın öğrenme yeteneği ve performansı üzerinde derin bir etkiye sahiptir.



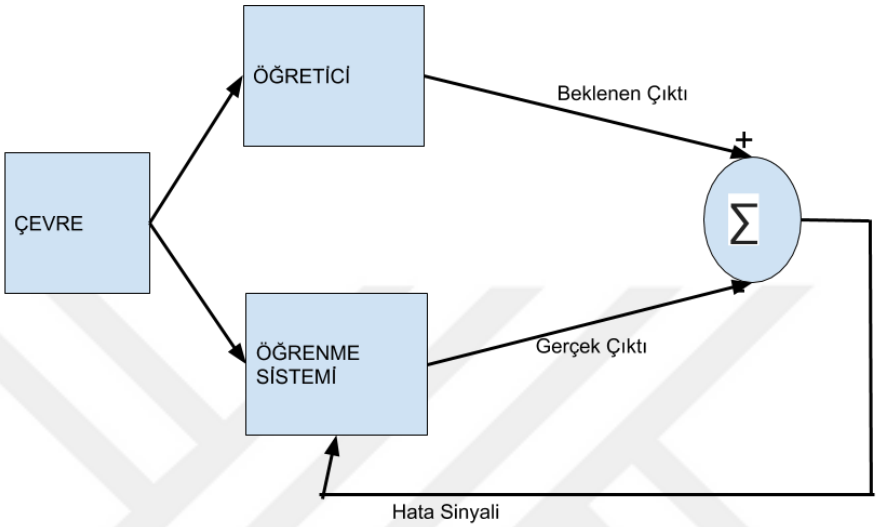
Şekil 4.4: Tam bağlamlı, ileri beslemeli, bir gizli katmanlı, bir çıkış katmanlı yapay sinir ağı

4.1.2 Öğrenme süreci

Genel anlamda, sinir ağları işleyişindeki öğrenme süreçlerini denetimli ve denetimsiz olarak iki kategoride değerlendirebiliriz. İkinci öğrenme şeklini, denetimsiz öğrenme ve pekiştirici öğrenme olarak iki alt kategoriye ayırabiliriz. Bu öğrenme tipleri insanlarda ve yapay öğrenme dillerinde paralel stratejilere sahiptirler.

Şekil 4.5 'de, bu öğrenme biçimini gösteren bir blok diyagram gösterilmektedir. Kavramsal olarak, eğitim verilerinin çevre hakkında bilgi sahibi olduğunu düşünebiliriz. Bu bilgi bir dizi girdi-çıktı örneği ile temsil edilir. Bunun neticesinde istenen cevap için optimum sinir ağı tarafından gerçekleştirilecek sonuç eğitilir. Bu eğitim, eğitim vektörü ve hata sinyallerinin birleşik etkisi ile şekil bulur. Hata sinyali, istenen yanıt ile ağın gerçek yanıtı arasındaki fark olarak tanımlanır. Bu hatayı minimize etmek için aynı adım birden çok kez tekrarlanır ve belirli bir hata ölçüsünün altında optimum olduğu varsayılır. Bu şekilde ortam bilgisi eğitilmiş sinir ağına aktarılmış olur. Bu aşama sonunda eğitilmiş yapay sinir ağının çevre

ile etkileşimi sonrası kendisinin karar verebileceği bir tecrübeye ulaşılmış olunur.



Şekil 4.5: Geri beslemeli denetimli öğrenme blok diagramı

Denetimsiz veya kendi kendine organize olan öğrenmede, öğrenme sürecini denetleyecek harici bir eğitmen yoktur. Ağın gerek duyduğu temsil kalitesinin görevden bağımsız ölçüsü, öğrenmek ve ağın serbest parametrelerini buna göre optimize etmektir. Belirli bir görevden bağımsız önlemler için, ağ ayarlandıktan sonra girdi verilerinin istatistiksel düzenlilikleri, ağ dahili yeteneği geliştirir. Girdinin özelliklerini kodlar ve böylece girdi temsillerini otomatik olarak sınıflar.

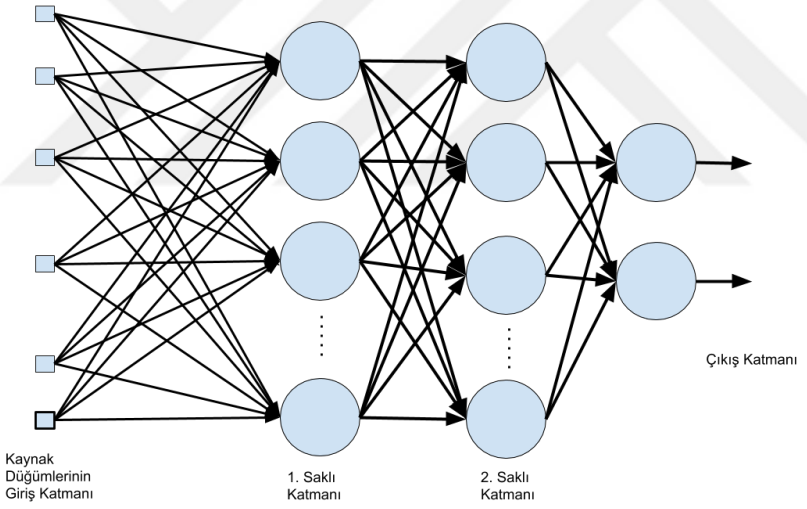
4.1.3 Çok katmanlı algılama

Tek katmanlı bir sinir ağı, doğrusal olarak ayrılabilir desenlerin sınıflandırılması ile sınırlıdır. Algılayıcının ve en küçük ortalama kare algoritmasının (LMS) pratik sınırlamalarının üstesinden gelmek için, çok katmanlı algılayıcı olarak bilinen bir sinir ağı yapısına ihtiyaç vardır. Çok katmanlı algılayıcıların temel özellikleri aşağıdaki gibidir:

1) Ağdaki her nöronun modeli, ayırt edilebilen doğrusal olmayan bir aktivasyon fonksiyonu içerir.

2) Ağ, hem giriş hem de çıkış düğümlerinden gizlenen bir veya daha fazla katman içerir.

3) Ağ, derecesi ağın sinaptik ağırlıklarıyla belirlenen yüksek derecede bağlantı gösterir.



Şekil 4.6: İki gizli katmanlı, yapay sinir ağı gösterimi

Çok katmanlı algılayıcıların eğitimi için popüler bir yöntem, LMS algoritmasını özel bir durum olarak içeren geri yayılma algoritmasıdır. Eğitim iki fazda ilerler:

1) İleri fazda, ağın sinaptik ağırlıkları sabitlenir ve giriş sinyali, çıkışa ulaşana kadar ağdan katman katmana yayılır. Değişiklikler ağdaki nöronların aktivasyon potansiyelleri ve çıktıları ile sınırlıdır.

2) Geriye doğru fazda, ağın çıkışı istenen bir yanıtla karşılaştırılarak bir hata sinyali üretilir. Sonuçta ortaya çıkan hata sinyali ağ üzerinden tekrar katman katman

olarak yayılır, ancak bu sefer yayılma geri yönde gerçekleştirilir. Bu ikinci aşamada, ağın sinaptik ağırlıklarında ardışık ayarlamalar yapılır. Çıktı katmanı için ayarlamaların hesaplanması basittir, ancak gizli katmanlar için çok daha zordur.

4.1.4 Geri yayılım algoritması

Geri yayılım sinir ağı; çok katmanlı, ileri beslemeli sinir ağıdır ve literatürde bilinen en yaygınıdır. Aynı zamanda çok katmanlı sinir ağlarının denetimli eğitimi için kullanılan en basit ve en genel yöntemlerden biri olarak kabul edilmektedir [25]. Geri yayılım metodolojisi, ağırlık değerlerini dahili olarak ayarlayarak giriş ve çıkış arasındaki doğrusal olmayan ilişkiye dayanarak çalışır. Bu şekilde eğitim örüntülerine dahil olmayan girdiler için genelleştirilebilir.

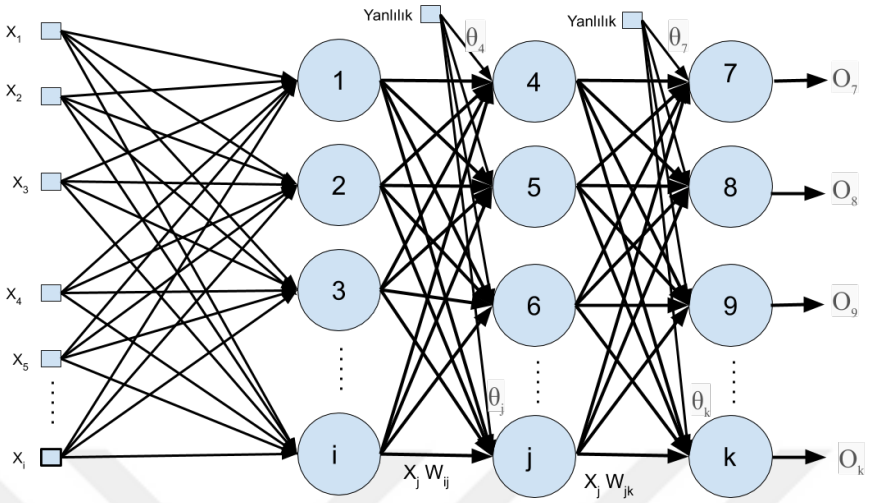
Genel olarak, Geri Yayılım ağının eğitim ve test olmak üzere iki aşaması vardır. Eğitim aşaması sırasında, ağ örnek girişlerini ve doğru sınıflandırmaları gösterir. Örneğin, girdi bir yüzün kodlanmış bir resmi olabilir ve çıktı kişinin adına karşılık gelen bir kodla temsil edilebilir.

Birçok öğrenme algoritması gibi bir sinir ağı, giriş ve çıkışların kullanıcı tanımlı bir şemaya göre kodlanmış olması gerekir. Şema, bir ağ eğitildikten sonra, tamamen yeni bir ağ oluşturmadan şemanın değiştirilemeyeceği şekilde ağ mimarisi oluşturacaktır. Benzer şekilde ağ yanıtını kodlamanın birçok biçimi vardır.

Şekil 4.7'de geri yayılım algoritması sinir ağının topolojisini ve giriş katmanını, iki gizli katmanı ve bir çıkış katmanını gösterilmektedir. Geri yayılım sinir ağlarının birden fazla gizli katmanı olabileceğine dikkat edilmelidir.

Sinir ağlarının işlemleri iki adıma ayrılabilir: İleri beslemeli ve Geri yayılımlı. İleri besleme adımında, bir giriş vektörü giriş katmanına uygulanır ve etkisi, bir çıktı üretilene kadar ağ üzerinden katman halinde yayılır. Ağın gerçek çıkış değeri daha sonra beklenen çıkışla karşılaştırılır ve çıkış düğümlerinin her biri için bir hata sinyali hesaplanır. Tüm gizli düğümler, bir dereceye kadar, çıkış katmanında görülen hatalara katkıda bulunur. Çıkış hata sinyalleri, katkıda bulunan gizli katmandaki her düğüme geri iletilir. Bu işlem daha sonra, ağdaki her düğüm için tekrarlanır. Genel hataya görece olarak katkısı tanımlanan bir hata sinyali alana kadar katman katman tüm düğümler tekrarlanır.

Her düğüm için hata sinyali belirlendikten sonra; hatalar ağ tarafından tüm eğitim modellerinin kodlanmasına izin verene kadar düğümler tarafından her bağlantı ağırlığının değerini güncellemek için kullanılır. Geri yayılım algoritması, delta kuralı veya gradyan inişi denilen bir teknik kullanarak ağırlık alanındaki hata iş-



Şekil 4.7: İki gizli katman ile geri yayılım sinir ağı

levinin minimum değerini arar. Hata fonksiyonunu en aza indiren ağırlıkların verilen probleme bir çözüm olduğuna karar verilir.

Ağ davranışı, bir veri kümesi olarak gösterilen ve bunları önceden tanımlanmış sınıflara sınıflandırması istenen bir insana benzer. Bir insan gibi, örneklerin sınıflara nasıl uyduğu hakkında hipotezler ortaya çıkacaktır. Bunlar daha sonra ağ tahminlerinin ne kadar doğru olduğunu görmek için doğru çıktılara karşı test edilir. En son hipotezdeki köklü değişiklikler, ağırlıklardaki büyük değişimlerle gösterilir ve küçük değişimler, hipotezde küçük ayarlamalar olarak görülebilir.

Sinir ağını genelleme konusundaki hususlar yetersiz eğitim ve aşırı eğitim verileriyle ilişkili problemlerdir. Sinir ağı karmaşık bir veri kümesindeki bir deseni tespit edecek kadar karmaşık olmadığında yetersiz eğitim oluşabilir. Bu genellikle çözümü kesin olarak temsil edemeyecek kadar az gizli düğümü olan ağların sonucudur. Öte yandan, aşırı eğitim, çok karmaşık bir ağla sonuçlanabilir ve bu da eğitim verisi aralığının çok ötesinde tahminlerle sonuçlanabilir. Çok fazla gizli düğümü olan ağlar, çözüme aşırı uyma eğilimi gösterecektir. Burada amaç, soruna iyi bir çözüm sağlayacak gizli düğüm sayısına sahip sinir ağları oluşturmaktır.

Giriş katmanına belirli bir eğitim modeli beslendiğinde, gizli katmandaki j'ninci düğüm için girdinin ağırlıklı toplamı Denklem 4.5 olarak verilmiştir.

$$Net_j = \sum w_{i,j} x_j + \theta_j \tag{4.5}$$

Denklem 4.5 ağ girdisini hesaplamak için kullanılır. Terim, her zaman 1 çıkış değerine sahip bir yanlılık düğümünden ağırlıklı değer ile kullanılır. Yanlılık düğümü, gizli katmandaki ve çıkış katmanındaki her bir nörona "sahte bir giriş" olarak kabul edilir ve girişin sıfır olduğu durumların üstesinden gelmek için kullanılır. Herhangi bir giriş deseninin sıfır değeri varsa, sinir ağı bir yanlılık düğümü olmadan eğitilemez.

Bir nöronun çıkış değeri aktivasyon fonksiyonundan geçer. Aktivasyon fonksiyonundan elde edilen değer, nöronun çıkışını belirler ve ona bağlanan bir sonraki katmandaki nöronlar için giriş değeri olur.

Geri yayılım algoritması için gereksinimlerden biri olan aktivasyon fonksiyonunun ayırt edilebilir özelliğe ihtiyaç duyduğundan dolayı en çok kullanılan aktivasyon fonksiyonlarından biri de Sigmoid denklemidir.

$$O_j = x_k = 1/(1 + e^{-Net_j}) \tag{4.6}$$

Benzer şekilde denklem 4.5 ve 4.6, çıkış katmanındaki düğüm k için çıkış değerini belirlemek için kullanılır.

Çıkış düğümünün (k) gerçek etkinleştirme değeri O_k ise ve k düğümü için beklenen hedef çıktı t_k ise, gerçek çıktı ile beklenen çıktı arasındaki fark şu şekilde verilir:

$$\Delta_k = t_k - O_k \tag{4.7}$$

Çıkış katmanındaki k düğümü için hata sinyali şu şekilde hesaplanabilir:

$$\delta_k = \Delta_k O_k(1 - O_k) = (t_k - O_k)O_k(1 - O_k) \tag{4.8}$$

$\Delta_k O_k(1 - O_k)$ sigmoid fonksiyonunun türevidir.

Delta kuralıyla, ağırlık giriş düğümü j ve çıkış düğümü k'daki değişiklik, düğüm k'daki hata ile j düğümünün aktivasyonu ile çarpılır.

Çıkış düğümü, k ve düğüm k arasındaki ağırlığı değiştirmek için kullanılan denklem $w_{j,k}$,

$$\Delta w_{j,k} = 1_\Gamma \delta_k x_k \tag{4.9}$$

$$w_{j,k} = w_{j,k} + \Delta w_{j,k} \tag{4.10}$$

$\Delta w_{i,j}$, j ve k düğümleri arasındaki ağırlıktaki değişim, 1_Γ öğrenme oranıdır.

Öğrenme oranı, ağırlıklardaki göreceli değişikliği gösteren nispeten küçük bir sabittir. Öğrenme oranı çok düşükse, ağ çok yavaş öğrenir ve öğrenme oranı çok yüksekse, ağ minimum ağırlık etrafında Şekil 4.8 deki gibi salınabilir. Genellikle öğrenme oranı çok düşüktür, örneğin 0,01 gibi. Geri yayılım algoritmasında yapılan bazı değişiklikler, öğrenme işlemi sırasında öğrenme hızının büyük bir değerden yavaş yavaş düşmesine izin verir. Bunun birçok avantajı vardır. Ağın, optimum ağırlık kümesinden uzak bir durumda başladığı varsayıldığı için, eğitim başlangıçta hızlı olacaktır. Öğrenme ilerledikçe, öğrenme oranı en uygun noktaya yaklaştıkça, öğrenme oranı azalır. Öğrenme oranını en uygun noktaya yakınlaşırken azaltmak, sinir ağı sonucunun en iyi noktayı aşma olasılığını azaltırken en iyi çözüme ulaşmaya teşvik eder. Bununla birlikte, öğrenme sürecinde sistem en uygun noktaya yakın bir şekilde başlarsa, başlangıçta salınabilir. Ancak öğrenme hızı azaldıkça bu etki zamanla azalır.

Denklem 4.9 'te x_k değişkeninin k düğümünün giriş değeri olduğu ve j düğümünden çıkan değerle aynı değer olduğuna dikkat edilmelidir.

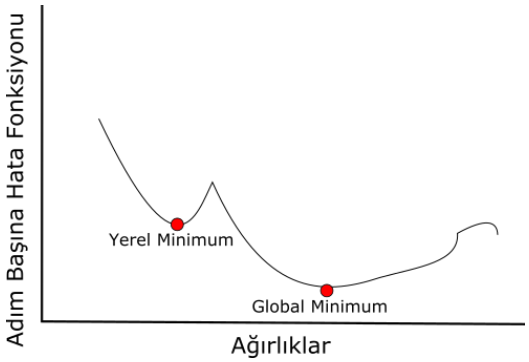
Ağırlıkların güncellenmesi sürecini iyileştirmek için Denklem 4.9'da bir değişiklik yapılır:

$$\Delta w_{i,j}^n = 1_\Gamma \delta_k x_k + \Delta w_{i,j}^{n-1} \mu \tag{4.11}$$

Burada, n'inci yineleme sırasındaki ağırlık güncellemesi, (n-1) 'in yinelemesiyle çarpılan bir momentum terimi ile belirlenir. Momentum teriminin tanıtılması, ağırlık değişikliklerinin daha büyük adımlarla aynı yönde devam etmesini teşvik ederek öğrenme sürecini hızlandırmak için kullanılır. Ayrıca, momentum terimi öğrenme sürecinin yerel bir minimum seviyeye yerleşmesini önler. Tipik olarak, momentum terimi 0 ile 1 arasında bir değere sahiptir.

Gizli katmandaki j düğümü için hata sinyali şu şekilde hesaplanabilir:

$$\delta_k = (t_k - O_k) O_k \sum (w_{j,k} \delta_k) \tag{4.12}$$



Şekil 4.8: Fonksiyonun yerel ve global mimumum gösterimi

Toplam terimi, çıkış katmanındaki tüm düğümler (k) için ağırlıklı hata sinyallerini toplar. Giriş düğümü i, çıkış düğümü j, düğüm ağırlığı $w_{i,j}$ olarak verildiğinde, Denklem 4.13 :

$$\Delta w_{i,j}^{n} = 1_{\Gamma}\delta_j x_j + \Delta w_{i,j}^{n-1}\mu, \tag{4.13}$$

$$w_{i,j} = w_{i,j} + \Delta w_{i,j}, \tag{4.14}$$

Evrensel hata, sinir ağına sunulan tüm desenler üzerindeki çıkış düğümlerindeki hatanın en aza indirilmesinin arzu edildiği bir yakınsamayla türetilir. Tüm denklemler için hata fonksiyonunu (E) hesaplamak için Denklem 4.15 kullanılır.

$$E = \frac{1}{2}\sum(\sum(t_k - O_k)^2) \tag{4.15}$$

İdealde sinir ağı doğru bir şekilde eğitildiğinde hata fonksiyonu sıfır değerine sahip olmalıdır. Ancak bu sayısal olarak gerçekçi değildir.

Yapay sinir ağları en eski ve en çok kullanılan yapay öğrenme algoritmalarından biridir. Genel geçer bir yapay öğrenme algoritması olduğundan dolayı farklı alanlarda birçok uygulaması vardır [35]. Bu çalışmada Yapay Sinir Ağlarından, Geri Yayılım Algoritması kullanılmıştır.

4.2 Rastgele Orman Algoritmaları

Rastgele Orman Algoritmaları [15], toplu öğrenim algoritmalarından biridir. Eğitim sırasında birden fazla karar ağacı ile pek çok değişken sahibi istatistiksel mo-

dellere uygun olarak sonuç kümelenmeleri yapılandırılır. Temel olarak bu sonuç kümelenmelerinin ortalaması veya küme olarak ayrımlarıyla sonuca karar verilir. Rastgele Orman Algoritmaları çeşitlerinde anahtar özelliklerin verildiği [28] kaynağından faydalanabilir.

Bu denetimli yapay öğrenme algoritması, geçmiş verileri inceler ve kestirimci bir anlayışla trendler oluşturmaya çalışır. Sınıflandırıcı olarak karar ağaçları kullanır. Rastgele Orman Algoritması rastgele karar ağaçları oluşturur. Rastgelelik iki şekilde ifade edilebilir:

1) Torbalamanın yapıldığı sırada seçilen örneklerin rastgele seçilmesi.

2) Her bir karar ağacı için seçilen niteliklerin rastgele seçilmesi.

Her bir karar ağacı sınıflandırıcısının gücü ve birbirleri arasındaki korelasyon, rastgele orman sınıflandırma algoritmasının sonuçlarındaki hata yüzdelerinin temel göstergeleridir.

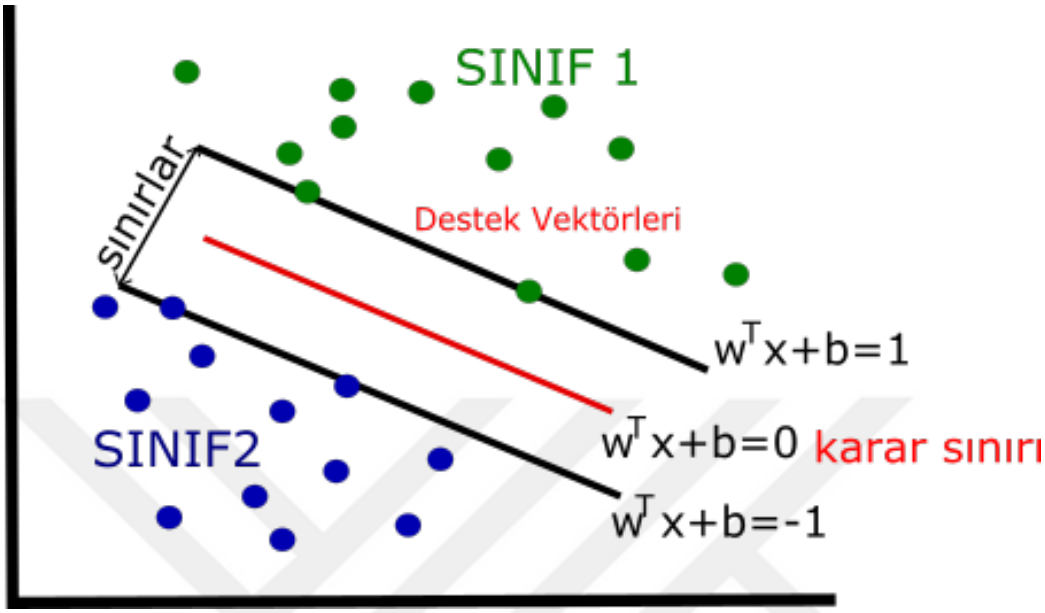
Rastgele Orman Algoritması, büyük veri setlerinde etkin bir şekilde çalışır. Girdi değişmesine gerek duymadan binlerce veriyi işleyebilir, önemli değişkenlerin tahminlerini verir ve orman büyümesi ilerledikçe sabit bir genelleştirme hatası üretir. Eksik verileri tahmin etmek için etkili bir metoda sahiptir. Büyük veri oranları eksilse bile sınıf popülasyonunun dengesiz olma durumlarına karşı veri kümelerinde sınıf hatasını dengeleme yöntemleri vardır. Rastgele Orman Algoritması; çok iş parçacıklı, çok çekirdekli ve paralel mimariler kullanarak paralel uygulamalara ön ayak olmuştur.

Rastgele Orman Algoritması, yukarıda belirtilen özellikleri nedeniyle birçok yeni sınıflandırma ve tahmin uygulamasında kullanılmaktadır. Literatürdeki çalışmalar 5 ayrı ana sınıfta toplanmıştır. Bunlar hassasiyet odaklı, performans odaklı, online uygulamalar, veriye özel ve sade olarak kendini göstermektedir [28].

4.3 Destek Vektör Makineleri

Destek Vektör Makineleri (SVM) [14] [15] [22], verilen eğitim setini, farklı boyutlarda noktalar olarak temsil edip, temsil edilen örnekler arasında açıkları kullanarak karar vermeye çalışır. Basit bir örnek olarak, iki küme arasındaki sınıflandırmayı yapmaya çalıştığınızda ikili doğrusal sınıflandırıcı ile çalışılır. Bu noktada farklı tanıları sınıflandıracak niteliksel özellikler çok önemli rol oynamaktadır. Destek Vektör Makineleri sınıflandırmalarındaki temel amaç uygun bir hesaplama zorluğu ile çok boyutlu düzlemler arasında ayırıcı hiper düzlemler oluşturabil-

mektir.



Şekil 4.9: SVM Örneği

Bundan sonraki bölümde basitçe 2 boyutlu bir düzlem üzerinden gidilerek SVM ve özellikleri örneklendirilmeye çalışılacaktır. Bulunan hiper düzlemler iki sınıf arasında bir sınır teşkil etmektedir, $w^T x + b = 0$. Bu sınır noktasının üstündeki ve altındakileri sırasıyla +1,-1 olarak etiketleyebilmek için $f\ (x) = sign(w^T\ x+b)$ kul-lanılabilir. Şekil 4.9 'da da görüldüğü gibi sınıflandırma için kullanılan 3 doğru da birbirine paraleldir. Bu da w ve b parametrelerinin paylaşıldığını göstermektedir. Bu iki doğru arasındaki mesafeyi bulmak için $w^T\ x\ +\ b\ =\ -1$ üzerinde x_1 noktası seçelim. $w^T\ x\ +b\ =\ +1$ üzerindeki x_1 noktasına en yakın noktaya x_2 diyelim. $x_2\ =\ x_1\ +\lambda\ *w$ olarak gösterilebilir. w her zaman her iki doğruya da diktir. Bu denkleme göre $\lambda * w$ iki noktayı birbirine bağlayan doğru parçasıdır. $\lambda\ *\ ||w||$ ise iki paralel doğru olarak ifade edilmiş olan sınırlar arasındaki mesafedir. $w^T x_2 + b = 1$ iken $x_2 = x_1 + \lambda * w$'dir.

$$w^T (x_1 + \lambda * w) + b = 1, \tag{4.16}$$

$$w^T x_1 + b + \lambda * w^T w = 1, \tag{4.17}$$

$$w^T x_1 + b = -1, \tag{4.18}$$

$$-1 + \lambda * w^T w = 1, \tag{4.19}$$

$$\lambda * w^T w = 2, \tag{4.20}$$

$$\lambda = \frac{2}{w^T w}, \tag{4.21}$$

$$\lambda = \frac{2}{||w||^2}. \tag{4.22}$$

Aradaki mesafe ise; $\lambda * w = \frac{2}{||w||^2} * ||w|| = \frac{2}{||w||} = \frac{2}{\sqrt{ww^T}}$ 'dir. Sınıflandırma yaparken aradaki mesafeyi en büyük seçmek yapılacak olan hata olasılığını en aza indirgemek için gereklidir. Bundan dolayı problemimizi ikinci dereceden denklemler halinde ifade edebiliriz.

$$min_{w,b} \frac{w^T w}{2} \tag{4.23}$$

$$y_i(w^T x_i + b) \geq 1, (\forall x_i). \tag{4.24}$$

Yanlış sınıflandırılmış bir eğitim seti veya niteliği iyi olmayan bir sınıflandırma, doğrusal olarak ayrılamayacak bir duruma neden olabilir. Örneğin, farklı türde balıkları ayırmaya çalışırken pullarının olup olmamasına bakmak gibi. Bu durumda gevşek değişkenler tanımlayabiliriz.

$\varepsilon_i \geq 0$ her bir x_i için
Bu durumda yeni doğrusal denklemimiz,

$$min_{w,b,\varepsilon} \frac{w^T w}{2} + C\Sigma_i \varepsilon_i \tag{4.25}$$

$$y_i(w^T x_i + b) \geq 1 - \varepsilon_i, \varepsilon_i \geq 0, \forall x_i \tag{4.26}$$

Veri vektörlerimizi daha yüksek dereceden boyutlara haritalamak veri vektörlerimizin ayrılabilmesi için kullanılan bir metottur. Bu durumda x_i , $\phi(x_i)$ ile değiştirildiğinde yeni denklemlerimiz,

$$min_{w,b,\varepsilon} \frac{w^T w}{2} + C\Sigma_i \varepsilon_i \tag{4.27}$$

$$y_i(w^T \phi(x_i) + b) \geq 1 - \varepsilon_i, \varepsilon_i \geq 0, \forall x_i \tag{4.28}$$

Lagrange çarpanları bir denkleme ait olan değişkenlerin tepe ve minimumlarının bulunması için kullanışlı bir metodolojidir. $y_i(w^T \phi(x_i) + b)$, 1'e olabildiğince yakın olmalıdır. Bu durum $max_{a_i}\ a_i[1 - y_i(w^T \phi(x_i) + b)]$, $a_i \geq 0$ ile sağlanmaktadır.

$y_i(w^T \phi(x_i) + b) \geq 1$ olduğunda $[1 - y_i(w^T \phi(x_i) + b)]$ negatife gittiği için $a_i = 0$ olduğu zaman yukarıdaki durum sağlanmaktadır. Diğer taraftan $y_i(w^T \phi(x_i) + b) < 1$ oldugunda $[1 - y_i(w^T \phi(x_i) + b)]$ pozitif bir değer olduğu için $a_i => \infty$ ile sağlanmaktadır. Bu şartlarda yapılan her yanlış sınıflandırma cezalandırılmakta ve doğru yapılan her sınıflandırmaya ceza verilmemektedir.

Bundan dolayı denklem,

$$min_{w,b}[\frac{w^T w}{2} + \sum_i max_{a_i>0} a_i[1 - y_i(w^T \phi(x_i) + b)]] \tag{4.29}$$

Bundan dolayı Denklem 4.29 üzerinde a değişkeninin sonsuza gitmesini engellemek için sınırlamak faydalı olacaktır $0 \leq a_i \leq C$. Lagrange denklemlerindeki eşli

problemi oluşturabilmek için max ve min değişimini yapıp yeni denklemi oluşturalım:

$$max_{a\geq 0}[min_{w,b}J(w,b;a)], \tag{4.30}$$

$$J(w,b;a) = \frac{w^T w}{2} + \sum_i a_i[1 - y_i(w^T \phi(x_i) + b)]] \tag{4.31}$$

Optimizasyon problemi çözdüğümüz için,

$$\frac{\partial J}{\partial w} = 0; \sum_i a_i y_i \phi(x_i) \tag{4.32}$$

$$\frac{\partial J}{\partial b} = 0; \sum_i a_i y_i = 0 \tag{4.33}$$

Yerine koyup basitlestirdikten sonra,

$$min_{w,b}J(w,b;a) = \sum_i a_i - \frac{1}{2}\sum_{i,j} a_i a_j y_j \phi(x_i)^T \phi(x_j) \tag{4.34}$$

Bu durumda Lagrange Eşli Denklemlerimiz:

$$max_{a\geq 0}[\sum_i a_i - \frac{1}{2}\sum_{i,j} a_i a_j y_j \phi(x_i)^T \phi(x_j)] \tag{4.35}$$

, $\sum_i a_i y_i = 0$ ve $0 \leq a_i \leq C$ tabiidir.

Genel olarak çok boyutlu uzaylarda çalıştığımızdan dolayı yukarıdaki denklemde verilen $\phi(x_i)^T \phi(x_j)$ denklemini hesaplamak zorlu olabilmektedir. Bunun için alt uzaylarda x_i ve x_j üzerinde skalar çarpıma denk gelecek özel çekirdek denklemleri kullanılmaktadır. Bu çarpımı $K(x_i, x_j)$ olarak ifade edebiliriz.

Yeniden denklemi ifade edecek olursak:

$$max_{a\geq 0}[\sum_{i} a_i - \frac{1}{2}\sum_{i,j} a_i a_j y_j K(x_i, x_j)] \tag{4.36}$$

Yeni bir x noktasını değerlendirmek istediğimizde tek yapmamız gereken, yukarıdan öğrendiğimiz optimal a_i parametrelerini değerlendirip denklemi hesaplamaktır.

$$f(x) = sign(w^T x + b) = \sum_{i} a_i y_i K(x_i, x) + b \tag{4.37}$$

$$w = \sum_{i} a_i y_i \phi(x_i) \tag{4.38}$$

Burada son olarak not edilmesi gereken a_i nin yalnizca $\phi(x_i)$ üzerinde sıfır olmayan değerlere sahip olmasıdır. Bu noktalar karar sınırını oluşturmaktadır. Bundan dolayı basitçe x_i üzerinde toplama yapmaktayız.

Bu çalışmada Hiperparametre Regresyonu ve Klasik Gaussian regresyonları kullanılmıştır. Bu metodolojilerin detaylarına [15]'den ulaşılabilir.

5. ÖNERİLEN ÇALIŞMA

Önerilen Çalışma bölümünde kullanılan veri kümesi, öznitelik seçim süreci ve uygulanan yöntem anlatılmaktadır.

5.1 Problem Tanımı

Savunma sanayi yazılım geliştirme projelerinde yazılım test süreci oldukça büyük bir zaman alabilmektedir. Yazılım test süreci doğası gereği iş gücü kaynağı tahmini zor bir problem olarak tüm yazılım sistemlerinde kendini göstermektedir. İnsan performans faktörü, projelerin büyüklüğüne göre yazılım zorluklarının artışı gibi sebeplerden dolayı test sürecinde ne kadar kaynak kullanılacağını doğru kestirilebilmek imkansızdır. Bu nedenle günümüz hızlı soluklu savunma sanayi projelerinde yazılım test kaynak ataması için doğru bir tahmin yapmak proje takvimi için oldukça önemlidir.

Bu çalışmada, ASELSAN bünyesinde gerçekleştirilmiş proje verileri kullanılarak yeni yazılım test projeleri için işgücü tahmini yapabilecek bir yapay öğrenme algoritması geliştirmek amaçlanmıştır. Kullanılan veri kümesi ile birlikte öznitelik seçim sürecine bir sonraki bölümde detaylı olarak yer verilmiştir.

5.2 Kullanılan Veri Kümesi ve Öznitelik Seçim Süreci

Bu çalışmada kullanılan veri kümesi, ASELSAN bünyesinde 2011-2019 yılları arasında geliştirilmiş olan 8 farklı projeye ait 31 yazılım konfigürasyon birimi (YKB) için gerçekleştirilmiş olan test sonuçları analiz edilerek oluşturulmuştur. Veri kümesinde 85 adet test verisi örneği bulunmaktadır. 85 adet test verisinin 68 tanesi kullanıcı arayüz yazılımlarına ait test verisi 17 tanesi ise gömülü sistem yazılımlarına ait test verilerinden oluşmaktadır. Her bir veri örneği, bir YKB için yapılmış olan kara kutu fonksiyonel test sonuç raporlarının analiz edilmesiyle oluşturulmuştur.

Yazılım Test Sonuç raporlarında; gerçekleştirilen test sayısı, testlerin kimler tarafından yapıldığı, kullanılan yazılımlar/simülatörler ve test yazılımları, kullanılan donanımlar, testlerin başlangıç ve bitiş tarihleri, testlerin gerçekleştirilmesi için harcanan süre gibi bilgiler bulunmaktadır. Bu bilgiler kullanılarak, geçmiş verilere dayalı olarak yazılım test eforu kestirimi yapan bir yapay öğrenme metodu geliştirilmesi amaçlanmıştır.

Çizelge 5.1'de görülebileceği gibi çalışmanın ilk aşamalarında gerçekleştirilen test sayısı, simülatör/test yazılımı kullanım bilgisi ve yazılımın tipi (Kullanıcı Arayüz Yazılımları/Gömülü ve Gerçek Zamanlı Yazılımlar) bilgilerinin yazılım testinin eforunu etkileyen önemli değişkenler olduğu düşünülerek bu değişkenlerin öznitelik olarak belirlendiği bir veri kümesi oluşturulmuştur (Veri kümesi-1) ve bu veri kümesi yapay öğrenme çalışmalarında kullanılmıştır. Çizelge 5.2 ve 5.3'de de Veri kümesi-1'de yer alan özniteliklerin alabileceği değerlere yer verilmiştir. Kullanılan yapay öğrenme algoritmalarının, bu öznitelikler ile gerçek değerlere göre %139 oranında hatalı tahminler ürettiği görülmüştür. Geliştirilen modele bağlı tahmin sonuçlarının çok etkili olmadığı, belirlenen özniteliklerin problem üzerinde iyi bir ayrım yapmaya yeterli olmadığı gözlenmiştir.

Çizelge 5.1: Veri kümesi-1, İlk Kullanılan Öznitelikler

1-Gerçekleştirilen Test SayısıI
2-Yazılımın Tipi
3-Simülatör/Test Yazılımı Kullanım Durumu

Çizelge 5.2: Yazılımın Tipi Öznitelik Vektörünün Alabileceği Değerler

YAZILIMIN TİPİ	**DEĞER**
Kullanıcı Arayüz Testleri	0
Gömülü Sistemler Testleri	1

Çizelge 5.3: Simülatör/Test Yazılımı Kullanım Durumu Öznitelik Vektörünün Alabileceği Değerler

SİMÜLATÖR/TEST YAZILIMI KULLANIM DURUMU	**DEĞER**
Kullanım Yok	0
Kullanım Var	1

Veri Kümesi-1'de kullanılan özniteliklerin yeterli derecede ayırıcı özelliğe sahip olmaması nedeni ile, yazılım test eforunu etkileyen ve geliştirilen modele öznitelik olarak girdi sağlayabilecek diğer özelliklerin belirlenebilmesi için bir çalışma yapılarak elimizdeki veriler yeniden düzenlenmiştir (Veri Kümesi-2). Testi gerçekleştiren test ekibinin deneyimi ve test edilen yazılımın zorluk derecesinin de test eforunu etkileyen özellikler arasında olması gerektiği değerlendirilmiştir. Test ekibinin deneyiminin harcanan test eforunu doğrudan etkileyeceği, deneyimli bir ekiple yapılan yazılım testleri için daha az efor harcanacağı varsayımı üzerinde durulmuştur. Yazılımın zorluk derecesinin de test eforu üzerinde önemli etkisi olduğu değerlendirilmiştir. Test edilen yazılımın zorluk ve karmaşıklık seviyesi

arttıkça harcanan test eforunun da artacağı düşünülmüştür. Yazılım Test Sonuç Raporlarında mevcut olmayan test ekibinin deneyimi ve yazılımın zorluk seviyesi bilgileri, her bir veri örneği için tek tek uzman görüşleri alınarak değerlendirilmiş ve veri kümesine bir girdi, geliştirilen modele de öznitelik olarak eklenmiştir. Veri kümesi-2'de kullanılan özniteliklere Çizelge 5.4'de yer verilmiştir. Yeni eklenen özniteliklerin alabileceği değerlere 5.5 ve 5.6'da yer verilmiştir. Kullanılan yapay öğrenme algoritmalarının, bu aşamada kullanılan öznitelikler ile gerçek değere göre %82 oranında hatalı tahminler ürettiği gözlenmiştir. Geliştirilen bu modelin ölçüm sonuçları değerlendirildiğinde; önceki modele göre daha iyi performans alındığı fakat hala gerçekte kullanılabilir olarak değerlendirilemeyeceği görülmüştür.

Çizelge 5.4: Veri Kümesi-2'de Kullanılan Öznitelikler

1-Gerçekleştirilen Test Sayısı
2-Yazılımın Tipi
3-Simülatör/Test Yazılımı Kullanım Durumu
4-Test Ekibinin Deneyimi
5-Yazılımın Zorluk Derecesi

Çizelge 5.5: Test Ekibinin Deneyimi Öznitelik Vektörünün Alabileceği Değerler

TEST EKİBİNİN DENEYİMİ	**DEĞER**
Deneyimli	3
Deneyimli ve Deneyimsiz Karışık	2
Deneyimsiz	1

Çizelge 5.6: Yazılımın Zorluk Derecesi Öznitelik Vektörünün Alabileceği Değerler

YAZILIMIN ZORLUK DERECESİ	**DEĞER**
Basit	1
Orta	2
Zor	3
Çok Zor	4

Son olarak Nageswaran'ın [33] Kullanım Durum Noktaları'nı (Use Case Points) test efor tahmine uyarladığı teknikten esinlenilerek yapay öğrenme algoritmalarına girdi olabilecek parametreler tekrar değerlendirildi. Savunma Sanayi projeleri özelinde, kendi verilerimize uygun olarak değerlendirdiğimiz, yazılım test eforunu etkileyen 15 parametreye karar verildi. Çizelge 5.7'de de görülebileceği gibi

veri uzayını azaltmak amacı ile bu belirlenen 15 parametreyi 5 farklı öznitelik altında gruplandırarak Veri Kümesi-3 oluşturulmuştur. Çalışmaya bu veri kümesi ile devam edilmiştir.

Çizelge 5.7: Veri Kümesi-3'de Kullanılan Öznitelikler

1-Gerçekleştirilen test senaryosu sayısı
2-Test edilen yazılımın tipi
3-Test senaryolarının zorluk seviyesi
4-Teknik faktörler
5-Test ekibinin tecrübesi

Çizelge 5.8: Teknik Faktörler

TEKNİK FAKTÖRLER	**AĞIRLIK**
Test Otomasyonu	5
Test Verisi Kullanma	5
Donanım Üzerinde Test İhtiyacı	6
Test Başlangıcında Donanım Altyapısının Kurulması İhtiyacı	7
Dağıtık Sistemlerde Çalışma	4
Karmaşık Kullanıcı Arayüzü	5

Çizelge 5.9: Testlerin Zorluk Seviyesi

ZORLUK SEVİYESİ	**AĞIRLIK**	**AÇIKLAMA**
Basit	1	Basit kullanıcı arayüzü işlemleri
Orta	2	Konfigürasyon dosyası işlemleri, veri tabanı kontrolü
Zor	3	Simülatör ya da başka bir yazılım/sistem kullanımı, dosya yükleme işlemleri, uzun süren test adımları

Parametrelerin hesaplanması için belirlenen ağırlık değerleri, veri kümesinde kullanılan projeler göz önünde bulundurularak ve uzman görüşleri alınarak karar verilmiştir. Veri kümesinde bulunan her bir veri için de uzman görüşü alınarak parametre değerleri hesaplanmıştır.

Bu çalışmada, yazılım test eforunu doğrudan etkileyen, yapay öğrenme modellerine girdi olan 5 özniteliği incelersek:

Gerçekleştirilen Test Senaryosu Sayısı: Bir yazılım için bir test döngüsü sırasında yürütülen test senaryolarının sayısıdır.

Test Edilen Yazılımın Tipi: Çizelge 5.2'de de görülebileceği gibi çalışmada kullanılan veriler, Kullanıcı Arayüz Yazılımları ya da Gömülü ve Gerçek Zamanlı Yazılımlar olarak işaretlenmiştir.

Test Senaryolarının Zorluk Derecesi: Çizelge 5.9'da görülebileceği gibi Basit olarak sınıflandırılan testler, yazılımdaki basit kullanıcı arayüzü işlemleridir. Orta olarak sınıflandırılan testler, kullanıcı arayüzü işlemlerine nazaran daha uzun sürede gerçekleştirilen, veri tabanı kontrolü, konfigürasyon dosyası işlemleri gibi testlerdir. Zor olarak sınıflandırılan testler ise, simülatör ya da başka sistemlerin kullanıldığı, dosya yükleme/yazma işlemlerini içeren ya da çok sayıda yazılımın gereksiniminin doğrulandığı test maddelerini içeren testler örnek gösterilebilir.

Test senaryoları yüzdesel olarak basit, orta ve zor olarak sınıflandırılarak, ağırlıkları ile çarpılıp, toplam değerleri hesaplanmıştır. yapay öğrenme algoritmalarına da öznitelik olarak, hesaplanan toplam değer girdi olarak verilmiştir. Örneğin, 100 test senaryosu olan bir yazılım test sürecinde, test senaryolarının %60'ı basit, %30'u orta, %10'u zor olarak uzman görüşü alınarak sınıflandırılmış olsun. Çizelge 5.10'de görülebileceği gibi Denklem 5.1 kullanılarak her bir yüzdeliğin ağırlıkları ile çarpılıp toplam değer normalize edilerek test senaryolarının zorluk seviyesi hesaplanmaktadır.

Çizelge 5.10: Zorluk Seviyesi Özniteliği Örneği

Seviye	**Yüzde**	**Ağırlık**	**Toplam**
Basit	60	1	60
Orta	30	2	60
Zor	10	3	30
Zorluk Seviyesi			1.5

$$ZorlukSeviyesi = \frac{\%Basit * 1 + \%Orta * 2 + \%Zor * 3}{100} \tag{5.1}$$

Teknik Faktörler: Çizelge 5.8'de görülen teknik faktörler içerisinde yer alan parametrelerin test eforunu ciddi şekilde etkilediği bilinmektedir. Özellikle Gömülü ve Gerçek Zamanlı Yazılım testlerinin donanım üzerinde yapılma ihtiyacı test eforunu arttırmaktadır. Bu testlerde donanımsal altyapının kurulma süresi, toplam test süresi içerisinde en büyük kalemdir. Bu sebeple bu iki parametrenin ağırlığının yüksek olması değerlendirilmiştir.

Test otomasyonu kullanımının maliyetinin yüksek olduğu bilinmektedir. Her ne kadar uzun vadede düşünüldüğünde test eforunda maliyet azaltıcı bir etmen olsa da, toplamda test otomasyonu gerektiren testler yüksek bir efor gerektirmektedir.

Test verisi kullanılan testler, birden çok verinin ya da veri kümesinin ayrı ayrı test edilerek beklenen çıktıyı karşılayıp karşılamadığını test eder. Dolayısıyla bu testler yüksek test eforu gerektiren testlerdir.

Dağıtık sistemlerde gerçekleştirilen testler, birden çok sistemde çalışmayı gerektirdiği ve test ortamının karmaşıklığını arttırdığı için normalden fazla test eforu gerektirir. Ayrıca dağıtık test ortamlarında diğer sistemlerden kaynaklı problem çıkma olasılığı da fazla olduğu için test eforunda artış yaşanabilir.

Karmaşık kullanıcı arayüzü olan yazılımların test eforunu olumsuz yönde etkilediği bilinmektedir. Yazılım ve sistemlerin karmaşıklık seviyesi arttıkça test eforunda da artış yaşanmaktadır.

Teknik faktörler içerisinde yer alan parametrelerin ağırlıkları veri kümesinde kullanılan projeler göz önünde bulundurularak ve uzman görüşleri alınarak karar verilmiştir. Veri kümesinde bulunan her bir veri için de uzman görüşü alınarak parametre değerleri hesaplanmıştır. Örneğin, bir yazılım test sürecinde, test ortamında donanımsal testlerin olduğu, donanım test ortamının kurulması gerekliliği, dağıtık sistemlerden oluşan bir test ortamının olduğu ve yazılımın karmaşık bir kullanıcı ara yüzüne sahip olduğu değerlendirilmiş olsun. Çizelge 5.11'de teknik faktörler için hesaplanacak olan teknik faktör değerine Denklem 5.2 kullanılarak ulaşılmıştır. Burada TF, teknik faktörü; TFa teknik faktör ağırlığını, K ise kullanımı ifade etmektedir.

Çizelge 5.11: Teknik Faktör Hesaplama Örneği

Faktörler	**Ağırlık**	**Kullanım**	**Toplam**
Test Otomasyonu	5	0	0
Test verisi kullanma	5	0	0
Donanım üzerinde test ihtiyacı	6	1	6
Donanım setup'ının kurulması ihtiyacı	7	1	7
Dağıtık sistemlerde çalışma	4	1	4
Karmaşık Kullanıcı arayüzü	5	1	5
Teknik Faktör			2.2

$$TF = \frac{\sum(TFa * K)}{10} \tag{5.2}$$

Test Ekibinin Deneyimi: Yazılım projelerinde, ilgili projelerde deneyim arttıkça test eforunun da olumlu yönde etkilenmesi beklenmektedir. Çizelge 5.5'de görülebileceği gibi, çalışmada kullanılan veriler, testleri gerçekleştiren test ekibinin deneyimine göre deneyimli, deneyimsiz ve deneyimli/deneyimsiz karışık olarak işaretlenmiştir.

Çalışmada kullanılan veri kümesi ve öznitelikler hakkında özet bilgiler Çizelge 5.12 'de yer almaktadır.

Çizelge 5.12: Veri Kümesi Hakkında Özet Bilgiler

Veri Kümesi	85 adet veri örneği	-17 Adet Gömülü Sistemler Yazılımları -68 Adet Kullanıcı Arayüzü Yazılımları
Veri Kümesi kullanımı	85 adet test verisi örneği	Veri kümesindeki tüm veriler K-Bölme (K-Fold) yöntemi ile test verisi içerisinde yer aldı.
Öznitelikler (Girdi)	5 adet Öznitelik	-Gerçekleştirilen Test Senaryosu Sayısı -Test Edilen Yazılımın Tipi -Test Senaryolarının Zorluk Seviyesi -Teknik Faktörler -Test Ekibinin Tecrübesi
Çıktı	1 Parametre	-Yazılım Test Eforu (saat)

5.3 Yöntem

Bu çalışmada, kara kutu test koşma döngüsü, yani kara kutu test senaryosu paketinin tam koşulabilmesi için gereken test çabasını tahmin etme problemi üzerine çalışılmaktadır. Bu problemin çözümünde geleneksel yöntemler kullanmak yerine yapay öğrenme metotları ile geçmiş veriler analiz ederek yeni bir efor kestirimi yapılmaktadır.

Yapay öğrenme için kullanılan özniteliklere Çizelge 5.7 'de yer verilmiştir. Özniteliklerin anlam ve detaylarına bir önceki bölümden ulaşılabilir.

Bu çalışmada test eforu tahmini problemine çözüm olabilecek 6 model (Çizelge 5.13) üzerinde çalışılmış ve sonuçları karşılaştırılmıştır.

Çizelge 5.13: Çalışılan Modeller

1) Levenberg-Marquardt Geri Yayılım Algoritması (geleneksel MLP (MultiLayer Perceptron) ağı ve ikinci derece Levenberg-Marquardt Geri Yayılım Algoritması)
2) Bayesci Geri Yayılım Algoritması (geleneksel MLP ağı ve Bayesci Geri Yayılım Algoritması)
3) Rastgele Orman Regresyonu
4) Destek Vektör Makinesi (SVM) Regresyonu
5) Gauss Çekirdek Fonksiyonu ile Destek Vektör Makinesi (SVM) Regresyonu
6) Otomatik Hiper-parametre Optimizasyonu ile Destek Vektör Makinesi (SVM) Regresyonu

Çalışmada önerilen yapay öğrenme yöntemlerinde eğitim ve test sonuçlarının daha verimli hale gelmesi ve daha iyi bir öğrenmenin yapılabilmesi için yüksek varyasyon içeren özniteliklere logaritmik normalizasyon işlemi uygulanmıştır. Bu çalışmada normalizasyon işlemi uygulanan parametreler Gerçekleştirilen Test Sayısı ve Harcanan Saat'tir. Normalizasyon işlemi Denklem 5.3'de verilmiştir.

$$x' = ln(x) \tag{5.3}$$

Yapay Öğrenme tekniklerinin performansını etkileyen en önemli sorunlardan bir tanesi de kullanılan veri kümeleri içerisindeki gürültülü verilerdir. Veri kümeleri içerisindeki uygun olmayan ve gereksiz değişkenleri ortadan kaldırarak öğrenme algoritmalarının performansını arttıracak birçok yöntem bulunmaktadır. Bizim veri kümemizde olduğu gibi veri örneklerinin sayısının nispeten az olduğu durumlarda, uygun olmayan ve gereksiz değişkenlerin getirdiği gürültü nedeniyle değişken seçim sürecinin önemi daha da artmaktadır.

Değişken seçiminin birçok avantajı vardır: Daha iyi veri anlama ve görselleştirme, bilgi toplama ve depolama için daha basit altyapı sunumu ve öğrenme makinelerini eğitmek ve çalıştırmak için gereken süreyi azaltma [40]. Bu çalışmada, veri kümesinde bulunan veriler içerisinde alakasız ve gereksiz verilerin ayıklanması için bir çalışma yapılmıştır. Veri kümesinde bulunan bütün veri örnekleri hem eğitim veri kümesinde hem de test veri kümesi içerisine eklenerek eğitim ve test veri kümelerindeki performanslarına bakılmıştır. Bütün modellerde test veri kümesinde kötü performans sergileyen veriler, eğitim veri kümesi içerisine alındığında sistemin genel performansı izlenmiştir. Bu verileri veri kümesinden çıkarttığımızda sistemin genel performansında artış gözlenmiştir.

Sistemlerin performansını ölçmek için birçok formül kullanılmaktadır. Bu çalışmada yapay öğrenme yöntemlerinin performansını ölçmek ve yöntemleri karşılaştırmak için literatürde en çok kullanılan formüller olan RMSE, MAPE ve MAE kullanılmıştır.

Sonuçlar bölümünde kullanılan istatistiksel standart denklemler kısaca Denklem 5.4,5.5,5.6'de gösterildiği gibi hesaplanmıştır. Denklemlerde ifade edilen y_i, gerçek yazılım test eforunu(saat), $y_{iTAHMIN}$ ise bu çalışmada önerilen kestirim sonucunu ifade etmektedir.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(y_{iTAHMIN} - y_i)^2}{n}} \tag{5.4}$$

$$MAPE = \frac{100}{n} \sum | \frac{y_{iTAHMIN} - y_i}{y} | \tag{5.5}$$

$$MAE = \sum_{i=1}^{n} | \frac{y_{iTAHMIN} - y_i}{n} | \tag{5.6}$$

ASELSAN'da test eforu tahmini için ortalama bir değer hesaplanmaktadır. Bu hesaplama herhangi parametrik bir tahmin içermemektedir. Hesaplama formulü doğrusal bir sonuca tabiidir. İnsan faktörü ve yapılan iş zorluğu yazılım test efor tahminini etkilememektedir.

Ortalama olarak bir test mühendisinin günde 25 test gerçekleştirebileceği baz alınarak adam/saat olarak bir tahmin yapılır.

Bu çalışmada maalesef sınırlı sayıda veri örneği vardır. Çeşitli güvenlik ve bilgi korunum politikaları dolayısı ile bu durumun üstesinden gelinememiştir.

Literatürdeki çeşitli çalışmalar [40][39][7], veri altyapılarında kurumdan kuruma farklılıkların görüldüğünü göstermektedir. Fakat ortak özellikleri, yapılan tüm çalışmaların, veri, ortam ve organizasyona özel altyapıları olmalarıdır. Veri kümesindeki sınırlı sayı, problem sonuçlandırmadaki kesin sonuçlandırmalara engel olmaya devam etmektedir.

Bu çalışmada kullanılan yapay öğrenme metodlarına girdi olarak kullanılan parametrelere Çizelge 5.14 'de yer verilmiştir.

Test sonuçlarını alırken aşağıda bulunan adımlar uygulanmıştır:

Çizelge 5.14: Yapay Öğrenme Modellerinin Girdi Parametreleri

METOT	**PARAMETRE**
MLP-Geri Yayılım Algoritması-LM,BR	5 girdi, 1 gizli katman 10 nöron
SVM Regresyonu-Gaussian	10 K-bölmeli Gauss çekirdek fonksiyonu
SVM Regresyonu-Hiperparametre Optimizasyonu	Hiperparametre (Box constraint, Kernel scale, Epsilon) optimizasyonları
Rastgele Orman Algoritması	100 Regresyon Ağacı

1- Veri kümesi, rastgele olarak %84 oranında eğitim seti, %16 oranında test seti olarak ayırılmıştır.
2- Geri Yayılım Algoritmalarında çapraz doğrulama yapıldığı için burada %70 oranında eğitim seti, %15 oranında çapraz doğrulama ve %15 oranında test verisi olarak ayrılmıştır.
3- Geri Yayılım Algoritmasında daha sağlıklı ölçümler elde edebilmek için her bir veri kümesi 10'ar kez koşturulup, test eforu tahmini sonuçlarının ortalaması alınmıştır.
4- Veri kümesindeki tüm veriler K-Bölmeli Çapraz doğrulama (K-Fold) yapılarak tüm verilerin test verisi içerisinde kullanılması sağlandı.
5- Her yapay öğrenme yöntemi için sonuçlar kayıt altına alınarak performansları ölçümlenmiştir.
6- Bütün yöntemlerin performansları karşılaştırılmıştır.

5.4 Deneysel Sonuçlar

Son olarak, Çizelge 5.15 'de verilen yapay öğrenme yöntemlerinin test eforu tahmini problemi üzerinde sonuç performansları karşılaştırılmıştır. Karşılaştırma tablosundan da görülebileceği üzere SVM regresyonu RMSE ve MAE için değerlendirildiğinde tüm metotlara göre en iyi performansı sergilemiştir. Bununla birlikte temel ve kullanışlı bir metot olan Bayesci Geri Yayılım Algoritması MAPE için değerlendirildiğinde en iyi performansı sergilemiştir. Bütün değerlendirmeler içinde en kötü tahmin Gauss Çekirdek fonksiyonu ile Destek Vektör Makinesi (SVM) Regresyonunda yapılmıştır.

Çizelge 5.16 'da çalışmada kullanılan veri kümesi üzerinde farklı metotların yaptığı tahminin, ASELSAN'da yapılan mevcut tahminlerle kıyaslandığında gerçek test eforlarına yakınlığı performans yüzdeleri olarak verilmiştir. Örneğin MLP-BR modeli test ettiği verilerin %73.626 'sında ASELSAN'ın yaptığı tahminden

Çizelge 5.15: Yapay Öğrenme Modellerinin Performans Karşılaştırmaları

METOT	**RMSE (saat)**	**MAPE**	**MAE (saat)**
MLP-Geri Yayılım Algoritması-LM	134,554	0,446	60,044
MLP-Geri Yayılım Algoritması-BR	133,500	0,393	54,262
Rastgele Orman Algoritması	138,791	0,470	61,310
SVM Regresyonu	94,271	0,423	46,299
SVM Regresyonu-Gaussian	167,145	0,627	65,312
SVM Regresyonu-Hiperparametre Optimizasyonu	95,889	0,409	46,700
ASELSAN Tahmini	154,774	0,605	81,142

daha başarılı bir tahminde bulunmuştur. Çizelgedeki hata metriklerinin büyük bir veri kümesi karşısında birbirini daha iyi takip etmeleri beklenmektedir.

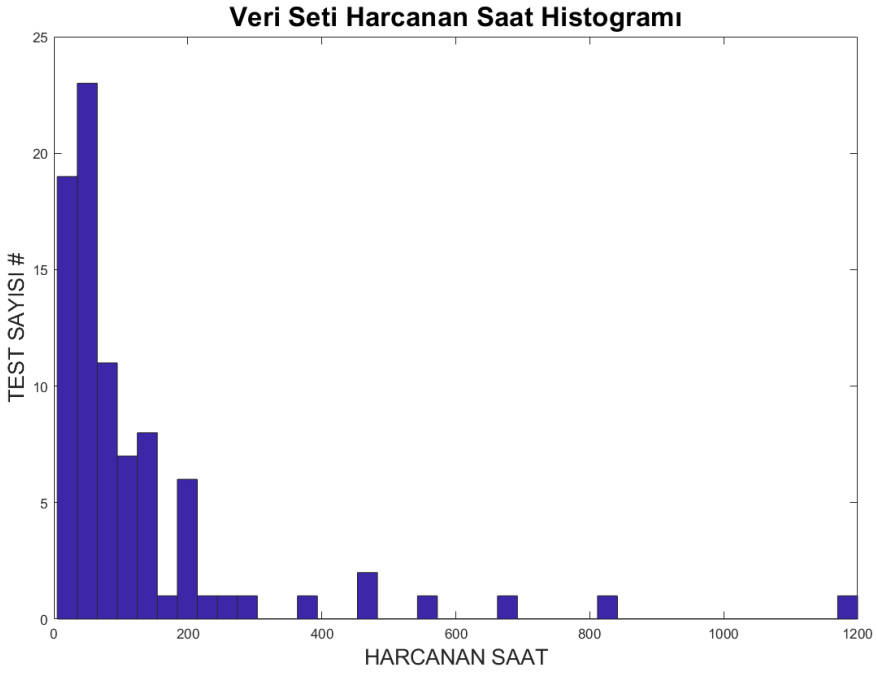
Çizelge 5.16: Mevcut Tahmine Göre Performans Karşılaştırması

METOT	**PERFORMANS(%)**
MLP-Geri Yayılım Algoritması-LM	%67,033
MLP-Geri Yayılım Algoritması-BR	%73,626
Rastgele Orman Algoritması	%70,330
SVM Regresyonu	%73,626
SVM Regresyonu-Gaussian	%58,242
SVM Regresyonu-Hiperparametre Optimizasyonu	%73,626

SVM algoritmalarının, veri kümelerinde yeterli öznitelik olduğunda, en iyi sonuçları verdiği literatürde gösterilmiştir [42]. Bunun yanında çalışmamızda sınırlı veri kümesi ile birlikte Bayesci Geri Yayılım Algoritması (BR) test verilerimiz üzerinde SVM ile birlikte en iyi sonuçları vermiştir. Özellikle çok boyutlu öznitelikler ile birlikte hesaplama karmaşıklığını çözmede etkili olan SVM algoritmaları, öznitelik dağılımının daha seyrek olduğu ve kişi performanslarını içeren karmaşık veri kümesinde beklenildiği gibi Gaussian dağılım kabulü yapan model dışında en iyi performanslara ulaşmıştır. BR algoritması aşırı öğrenmeye (overfitting) daha bağışıklı olduğundan çapraz doğrulama metotlarına ihtiyaç duyulmamıştır. Bayesci Geri Yayılım Algoritmalarının karmaşık veri kümelerinde, daha iyi performans vermesi de beklenilen bir durumdur. Geleneksel Geri Algoritmalarında sabit bir ağırlık vektörü bulunurken Bayesci Geri Yayılım Algoritmalarda bir olasılık dağılımı kullanılmaktadır.

Şekil 5.1, veri kümesindeki harcanan saat (target) veri histogramını göstermektedir. Veriler incelendiğinde yapay öğrenme modeli hedefinin yaklaşık %90'ının

200 saatin altında olduğu görülmektedir. Bununla birlikte 1200 saati bulan testler de veri kümemizde bulunmaktadır. Daha önce de belirtildiği gibi bu çalışmaya uygun yalnızca 85 adet yazılım test verisi sağlanabilmiştir. Veri kümesine eklenebilecek yeni veriler ile histogramda görülen boşlukların doldurulup daha iyi sonuçlara ulaşılabileceği düşünülmektedir.

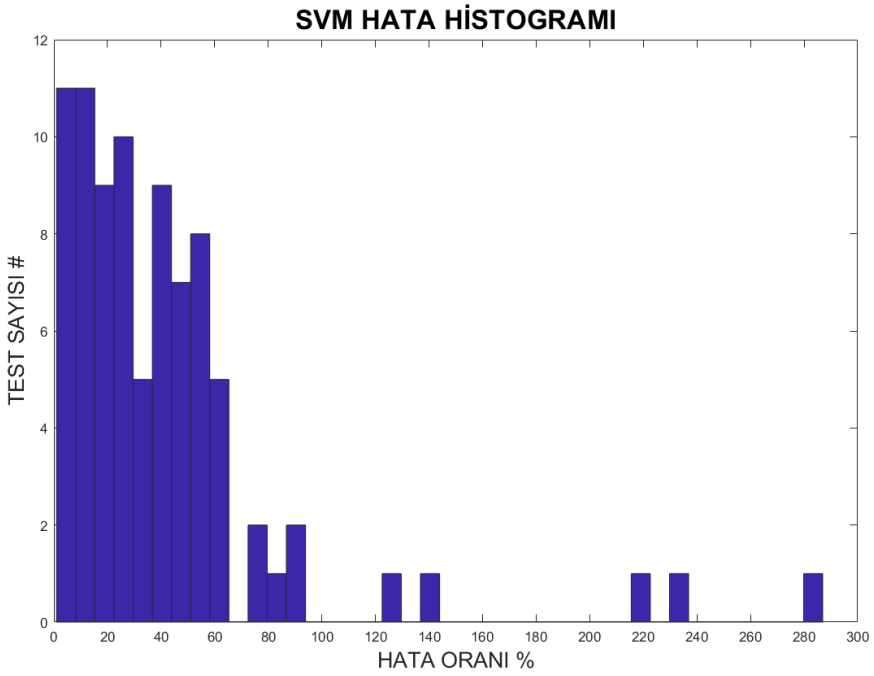


Şekil 5.1: Veri Kümesi harcanan saat histogramı

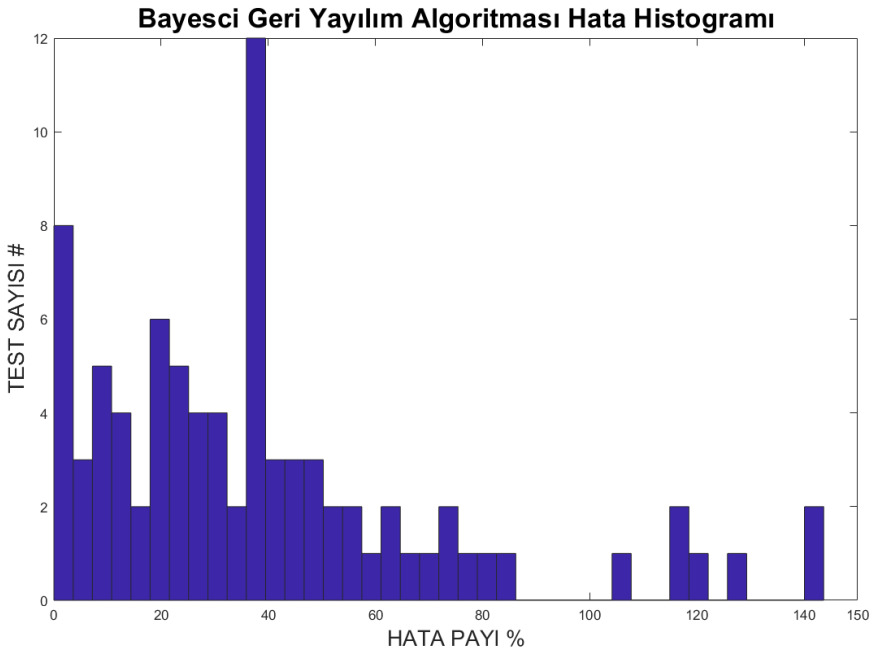
Şekil 5.2, SVM regresyonu için % hata histogramını göstermektedir. %60 a kadar hata histogramında benzer sayılarda test verisi görünse de eğitilen SVM modeli %280'e varan hatalar yapabilmiştir. Hata yüzdeleri yüksek olan veriler incelendiğinde, genel olarak testler için harcanan saatin 60 saatin altında olduğu verilerde daha büyük yüzdeliklerde hataların olduğu görülmüştür. SVM algoritması doğası gereği harcanan saatin 1200'lere vardığı veri setlerindeki özniteliklere göre bir yaklaşım sergileme doğasında olduğundan, tüm veri kümesi istatistiklerinde bir sapmaya neden olmaktadır. En büyük sapma da doğal olarak harcanan saatin düşük olduğu değerlerde görülmektedir.

Şekil 5.3 Bayesci Geri Yayılım Algoritması için % hata histogramını göstermektedir. Bu algoritmanın sonuçları en fazla % 140 lık hatalara neden olmuştur. Hata yapılan veriler incelendiğinde yüzdelik hataların genele yayıldığı görülmektedir.

Şekil 5.4 Rastgele Orman Algoritması % hata histogramını göstermektedir. Bu al-

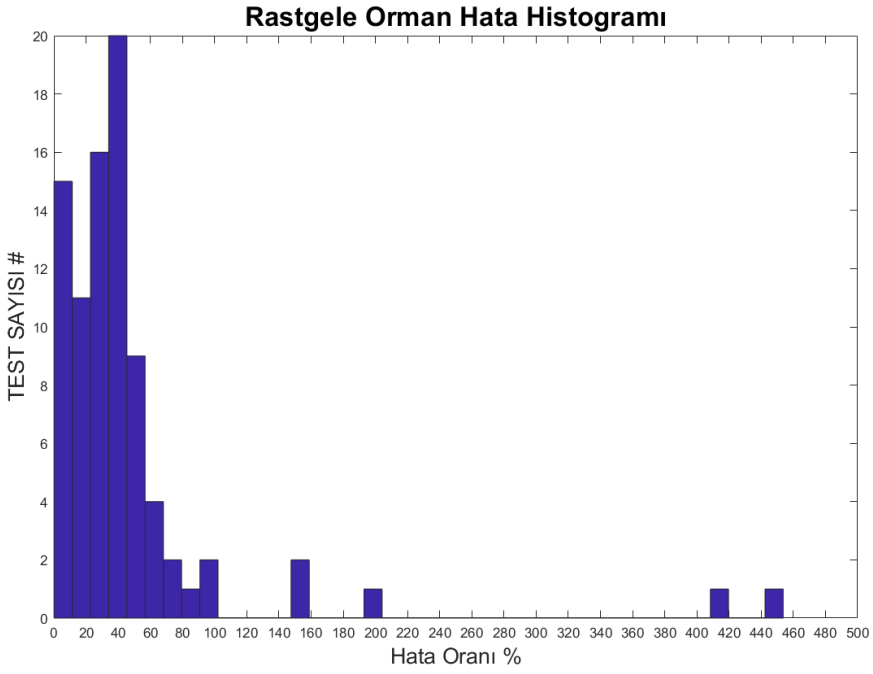


Şekil 5.2: SVM % hata histogramı



Şekil 5.3: Bayesci Geri Yayılım Algoritması % hata histogramı

goritmanın sonuçları incelendiğinde tüm veri kümesinin %90 lık bir kısmında%70 ve altında hata ile sonuç bulunulduğu görülmüştür. Bu değer veri kümesinin %50 sinde %30 hatanın altında kalmıştır. Fakat Rastgele Orman Algoritması aşırı öğrenmeye (overfitting) yatkın olduğundan bazı verilerde %600'e varan hatalar yapabilmiştir.

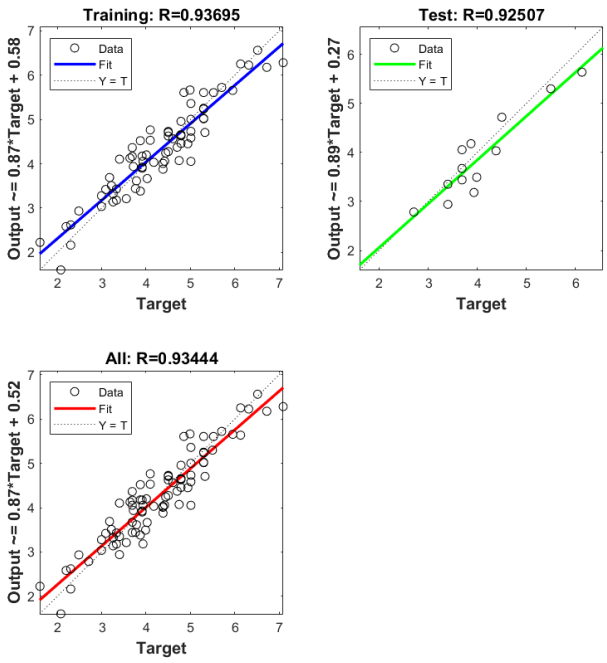


Şekil 5.4: Rastgele Orman Algoritması % hata histogramı

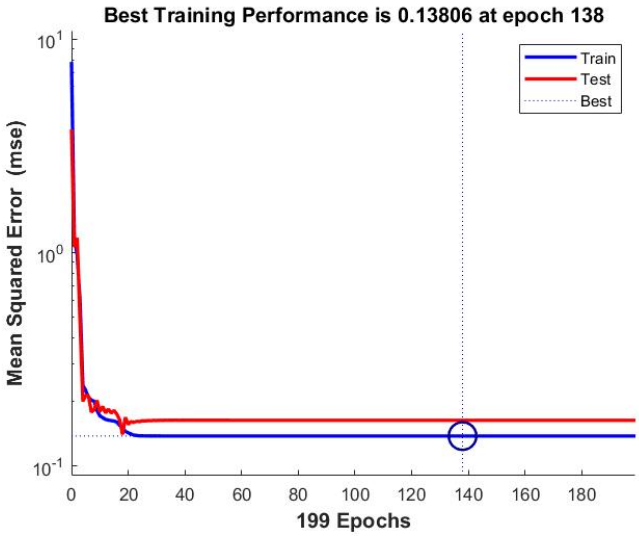
Şekiller 5.5 ve 5.6'de Bayesci Geri Yayılım Algoritmasının, Şekiller 5.7 ve 5.8'de ise Levenberg-Marquardt Geri Yayılım Algoritmasının performans ve hata grafikleri verilmiştir. BR ve LM (Levenberg-Marquardt) Geri Yayılım Algoritmaları karşılaştırıldığında, LM Geri Yayılım Algoritmasının tahmin performansı BR Geri Yayılım Algoritmasının biraz daha altında kalmıştır.

Şekiller 5.5 ve 5.7 'de Levenberg-Marquardt ve Bayesci Geri Yayılım algoritmaları için eğitim ve test performansından birer örnek görülmektedir.

Şekiller 5.6 ve 5.8 'de Levenberg-Marquardt ve Bayesci Geri Yayılım algoritmaları için iterasyon-hata grafikleri için birer örnek görülmektedir. Beklenildiği üzere iterasyon sayısı arttıkça öğrenmenin daha iyi seviyelere çıktığı, başarımın arttığı grafiklerden de görülmektedir. Veri kümesi üzerinde uyguladığımız Levenberg-Marquard Geri Yayılım algoritmasında iterasyon sayısı ortalama 10 iken Bayesci

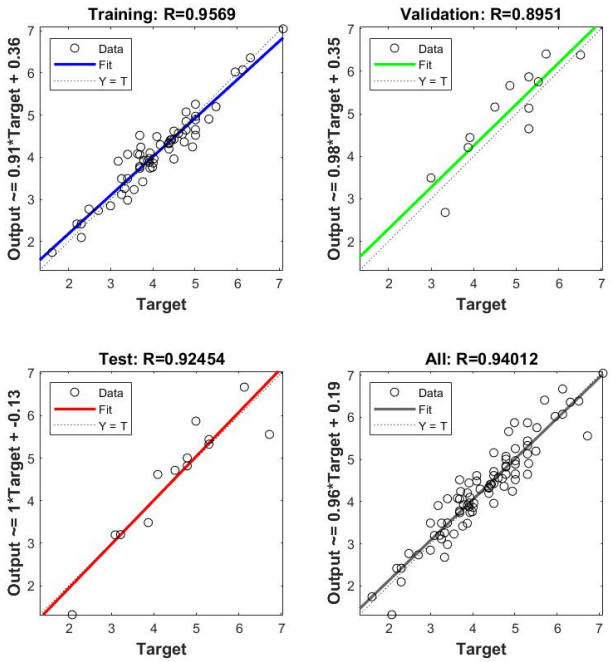


Şekil 5.5: Bayesci Geri Yayılım Algoritması için eğitim ve test performansından bir örnek



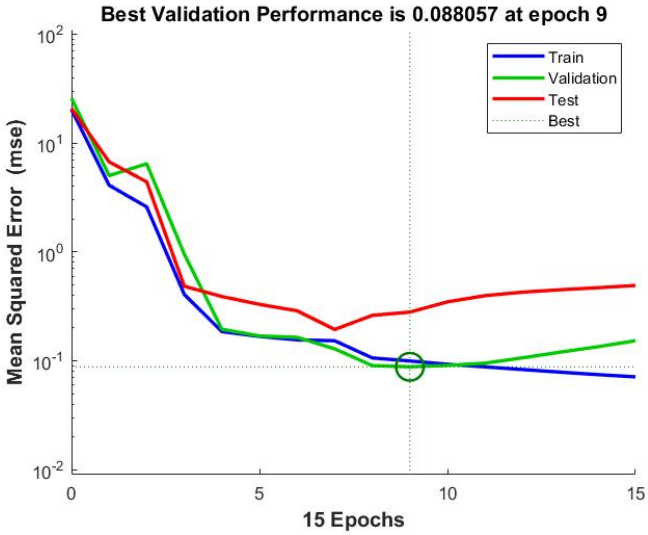
Şekil 5.6: Bayesci Geri Yayılım Algoritması için iterasyon/mse grafiği

Geri Yayılım algoritmasında ise önceden tanımlı iterasyon sayısına ulaşıldığında (1000) algoritma durmaktadır.



Şekil 5.7: Levenberg-Marquardt Geri Yayılım Algoritması için eğitim, çapraz doğrulama ve test performansından bir örmek

Bu çalışmada kullanılan metotların ortalamaları arasındaki farkın anlamlılığını test etmek için Tek Yönlü Varyans Analizi (ANOVA) kullanılarak Şekil 5.9a, 5.9b ve Şekil 5.10 oluşturulmuştur. ANOVA, istatistik biliminde çoklu gruplar için kullanılan varyans analizinin başlıca tekniğidir [20]. ANOVA sonuçlarına göre çalışmada kullanılan yapay öğrenme metotları arasında anlamlı bir farklılığın olmadığı (Sig (p) > 0.05) fakat Gauss Çekirdek fonksiyonu ile Destek Vektör Makinesi (SVM) Regresyonu için bir farklılık olduğu görülmüştür. ASELSAN 'da kullanılan mevcut tahminler ile çalışmada kullanılan metotlar arasında ise beklenildiği gibi anlamlı bir farklılık çıkmıştır (Sig (p) < 0.05).



Şekil 5.8: Levenberg-Marquardt Geri Yayılım Algoritması için İterasyon/MSE grafiği

5.5 Öznitelik Araştırması

Çalışmanın bu kısmında yazılım test eforu tahmininde en etkili öznitelikleri bulmak ve yazılım test eforunu en çok etkileyen faktörler hakkında bilgi edinmek istenmiştir. Bu amaçla özniteliklerin tek başlarına sonuç üzerinde etkisi ve bir öznitelik olmadığında sonuca olan etkileri incelenmiştir. Çizelge 5.17'de görülebileceği üzere 'Gerçekleştirilen Test Sayısı' özniteliğinin yazılım test eforu tahmini üzerinde en yüksek etkiye sahip olduğu görülmüştür. Diğer özniteliklerin de kestirim sonucuna olumlu yönde etkileri olduğu fakat 'Gerçekleştirilen Test Sayısı' özniteliği kadar güçlü bir etkilerinin olmadığı görülmüştür. Çizelge 5.17'de yapılan çalışmada Tüm Özniteliklerin var olduğu kestirim sonucu %42.05'lik Yapay Öğrenme Ortalama Yüzde Hata Sonucu vermiştir.

Çizelge 5.18'de öznitelik vektörleri içerisinde en önemlisi ile başlanarak teker teker yeni öznitelikler eklenmiştir. Sadece Gerçekleştirilen Test Sayısı, Teknik Faktörler ve Zorluk Seviyesi özniteliklerinin kullanıldığı yapay öğrenme metodu ile tüm özniteliklerin olduğu Ortalama Hata Yüzdelerine çok yakın sonuçlara ulaşılmıştır. Bunun nedeni sahip olduğumuz veri kümesinde yazılım tiplerindeki veri sayısı dengesizliği ve bu özniteliklerin problemi iyi modelleyebilmiş olmasıdır. Ayrıca sahip olduğumuz veri kümesinde, farklı yazılım tiplerindeki veri sayısı dengesizliği, bu öznitelik çıkartıldığında %0,43 oranında daha iyi sonuçların ortaya çıkmasına sebep olmuştur. İlerideki çalışmalarda bu dengenin sağlanabile-

Paired Samples Test									
		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
					Lower	Upper			
Pair 1	Random_Forest - BackProp_BR	9,35	87,31	9,47	-9,48	28,19	0,99	84	0,33
Pair 2	Random_Forest - BackProp_LM	-0,31	99,58	10,80	-21,78	21,17	-0,03	84	0,98
Pair 3	Random_Forest - SVM	4,73	81,07	8,79	-12,76	22,22	0,54	84	0,59
Pair 4	Random_Forest - SVM_Gaussian	34,49	122,56	13,29	8,05	60,92	2,59	84	0,01
Pair 5	Random_Forest - SVM_Opt	4,29	85,09	9,23	-14,06	22,64	0,47	84	0,64
Pair 6	BackProp_BR - BackProp_LM	-9,66	47,06	5,10	-19,81	0,49	-1,89	84	0,06
Pair 7	BackProp_BR - SVM	-4,62	60,78	6,59	-17,74	8,49	-0,70	84	0,49
Pair 8	BackProp_BR - SVM_Gaussian	25,13	107,66	11,68	1,91	48,35	2,15	84	0,03
Pair 9	BackProp_BR - SVM_Opt	-5,07	51,67	5,60	-16,21	6,08	-0,90	84	0,37
Pair 10	BackProp_LM - SVM	5,04	70,84	7,68	-10,24	20,32	0,66	84	0,51
Pair 11	BackProp_LM - SVM_Gaussian	34,79	128,75	13,96	7,02	62,56	2,49	84	0,02

(a) 1.Kısım

Paired Samples Test									
		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
					Lower	Upper			
Pair 12	BackProp_LM - SVM_Opt	4,59	64,90	7,04	-9,40	18,59	0,65	84	0,52
Pair 13	SVM - SVM_Gaussian	29,76	110,80	12,02	5,86	53,65	2,48	84	0,02
Pair 14	SVM - SVM_Opt	-0,44	19,70	2,14	-4,69	3,81	-0,21	84	0,84
Pair 15	SVM_Gaussian - SVM_Opt	-30,20	112,90	12,25	-54,55	-5,85	-2,47	84	0,02
Pair 16	ASELSAN_Mevcut_Tahmin - Random_Forest	-68,01	117,05	12,70	-93,25	-42,76	-5,36	84	0,00
Pair 17	ASELSAN_Mevcut_Tahmin - BackProp_BR	-58,65	106,16	11,51	-81,55	-35,76	-5,09	84	0,00
Pair 18	ASELSAN_Mevcut_Tahmin - BackProp_LM	-68,31	119,71	12,98	-94,13	-42,49	-5,26	84	0,00
Pair 19	ASELSAN_Mevcut_Tahmin - SVM	-63,28	79,82	8,66	-80,49	-46,06	-7,31	84	0,00
Pair 20	ASELSAN_Mevcut_Tahmin - SVM_Gaussian	-33,52	82,46	8,94	-51,31	-15,73	-3,75	84	0,00
Pair 21	ASELSAN_Mevcut_Tahmin - SVM_Opt	-63,72	89,47	9,70	-83,02	-44,42	-6,57	84	0,00

(b) 2.Kısım

Şekil 5.9: Tüm modeller için ANOVA Analiz sonucu

Paired Samples Test									
		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the				
					Lower	Upper			
Pair 1	ASELSAN_Mevcut_Tahmin - Random_Forest	-68,01	117,05	12,70	-93,25	-42,76	-5,36	84	0
Pair 2	ASELSAN_Mevcut_Tahmin - BackProp_BR	-58,65	106,16	11,51	-81,55	-35,76	-5,09	84	0
Pair 3	ASELSAN_Mevcut_Tahmin - BackProp_LM	-68,31	119,71	12,98	-94,13	-42,49	-5,26	84	0
Pair 4	ASELSAN_Mevcut_Tahmin - SVM	-63,28	79,82	8,66	-80,49	-46,06	-7,31	84	0
Pair 5	ASELSAN_Mevcut_Tahmin - SVM_Gaussian	-33,52	82,46	8,94	-51,31	-15,73	-3,75	84	0
Pair 6	ASELSAN_Mevcut_Tahmin - SVM_Opt	-63,72	89,47	9,70	-83,02	-44,42	-6,57	84	0

Şekil 5.10: ASELSAN tahmini ve tüm modellerin ikili ANOVA karşılaştırılması

ceği daha büyük veri kümeleri ile çalışıp daha fazla sayıda yazılım tipine sahip olan veri kümeleri ile çalışılması öngörüldüğünden bu fark görmezden gelinmiştir.

Çizelge 5.19'da Yapay Öğrenme Modellerine özniteliklerin ikili kombinasyonlarını girdi olarak verdiğimizde alınan sonuçlar görülebilir. Gerçekleştirilen Test Sayısının en önemli öznitelik olduğu bu çizelgede de doğrulanmıştır. Çizelge 5.20'de Yapay Öğrenme metotlarına girdi olan özniteliklerin birbirleri ile olan korelasyonları görülmektedir. Çıkan korelasyon matrisine göre Yazılımın Tipi ve Teknik Faktörler özniteliklerinin birbiri ile ilişkili olduğu görülmektedir. Gömülü ve Gerçek Zamanlı Yazılımlarda donanım üzerinde test, donanım ortamının kurulma ihtiyacının sıklıkla görülmesi ve dağıtık sistemlerde çalışma ihtiyacının çok olduğu bilinmektedir. Bu da Teknik Faktörler öznitelik değerini arttırmaktadır. Bu sebeple de Yazılım Tipi ve Teknik Faktörler arasında korelasyon yüksek çıkmaktadır. Diğer özniteliklerin birbirleri ile ilişkili olmadığı, bağımsız oldukları söylenebilir. Çizelge 5.17'deki sonuca paralel olarak Çizelge 5.20'de de 'Gerçekleştirilen Test Sayısı' özniteliğinin sonuç (Gerçekte Harcanan Saat) üzerinde en büyük korelasyona sahip olduğu görülebilir.

Çizelge 5.20'de aynı zamanda Teknik Faktörler ile Harcanan Saat üzerinde de 0.363'lük bir korelasyon bulunmuştur. Bunun nedeni Teknik Faktör değerinin artması ile daha karmaşık ve birbirleri ile haberleşmesi gereken daha fazla altsiste-

Çizelge 5.17: Öznitelik önem araştırması - Yapay Öğrenme Ortalama Yüzde Hata Sonuçları

	Sadece Bu Öznitelik Var	**Sadece Bu Öznitelik Yok**
Gerçekleştirilen Test Sayısı	%52.05	%123,01
Yazılımın Tipi	%119.24	%41,62
Zorluk Seviyesi	%144.09	%49,48
Teknik Faktörler	%115.19	%49,86
Test Ekibinin Tecrübesi	%120.12	%43,24

Çizelge 5.18: Öznitelik Ekleme Evrimi ile Yapay Öğrenme Ortalama Yüzde Hata Sonuçları

Öznitelikler	**Ortalama Yüzde Hata (MAPE)**
G.Test Sayısı	%52.05
G.Test Sayısı + Teknik Faktörler	%47.76
G.Test Sayısı + Zorluk Seviyesi	%57.67
G.Test Sayısı + Yazılım Tipi	%50.43
G.Test Sayısı + Yazılım Tipi + Teknik Faktörler	%48.39
G.Test Sayısı + Teknik Faktörler + Zorluk Seviyesi	%42.31
G.Test Sayısı + Teknik Faktörler + Zorluk Seviyesi + Yaz. Tipi	%43.24
G.Test Sayısı + Teknik Faktörler + Zorluk Seviyesi + Tecrübe	%41.62
Tüm Öznitelikler	%42.05

min bulunduğu, donanım üzerinde test ihtiyacının ve test donanımı kurulumlarının fazlalaştığı yazılım test altyapılarına ihtiyaç duyulmasıdır.

Çalışmanın bu bölümü, Gerçekleştirilen Test Sayısı özniteliğinin yazılım test eforunda sonuca en etkin olan faktör olduğunu açıkça göstermektedir. Teknik Faktörler ve Zorluk Seviyesi öznitelikleri de eklendiğinde, Çizelge 5.18'de de görülebileceği gibi yazılım test eforu kestiriminde kullanılan tüm özniteliklerin çıktılarına yakın bir yapay öğrenme sonucuna ulaşılmıştır. Her ne kadar Gerçekleştirilen Test Sayısı faktörü ortalama bir değer alabilmek için gerekli altyapıyı sağlıyor gibi görünse de, yazılım gereksinimlerini doğrulamak için karar kılınmış testlerin teknik faktör kriterleri ve testlerin zorluk seviyeleri, test eforunun doğru olarak kestirilebilmesi için önemli bir katkıda bulunmaktadır.

Çizelge 5.19: İkili Öznitelik Vektörlü Yapay Öğrenme Ortalama Yüzde Hata Sonuç Matrisi - *1-Gerçekleştirilen Test Sayısı, 2-Yazılım Tipi 3-Test senaryolarının zorluk seviyesi, 4-Teknik Faktörler, 5- Test Ekibinin Tecrübesi*

Öznitelikler	**1**	**2**	**3**	**4**	**5**
1	52.05	50.43	57.67	47.76	49.85
2		119.24	139.43	116.54	125.87
3			144.09	118.97	133.95
4				115.19	118.15
5					120.12

Çizelge 5.20: Öznitelik Korelasyon Matrisi - *1-Gerçekleştirilen Test Sayısı, 2-Yazılım Tipi 3-Test senaryolarının zorluk seviyesi, 4-Teknik Faktörler, 5-Test Ekibinin Tecrübesi 6-Harcanan Saat (Sonuç)*

Öznitelikler	**1**	**2**	**3**	**4**	**5**	**6**
1	1	0.2115	-0.3350	-0.0035	0.1451	0.5603
2		1	0.2527	0.7243	-0.3339	0.0642
3			1	0.2643	-0.2957	0.0850
4				1	-0.2587	0.3630
5					1	-0.1249
6						1

6. SONUÇ VE ÖNERİLER

Yazılım test planlaması yapılırken, ihtiyaç duyulan yazılım test işgücünün gerçekçi bir şekilde belirlenmesi, değerli olan zaman ve işgücü kaynağını etkin kullanabilmek için oldukça önemlidir. Bu çalışmada yazılım test eforlarını daha iyi bir oranla tespit edebileceğimiz, ASELSAN içerisinde daha iyi bir test efor kestirimi için kullanılabilecek en iyi yapay öğrenme metotları ve öznitelikleri belirlenmiştir. Kestirim özniteliklerinin sistematik olarak sağlanabileceği bir altyapı ile savunma sanayi proje zaman kıstaslarını daha doğru sağlayabilecek dinamik temel taşlar atılmıştır. Ayrıca yazılım test eforunu etkileyen faktörler belirlenerek efor tahmini üzerinde hangi özniteliklerin en önemli olduğuna karar verilmiştir.

Bu çalışma savunma sanayisi bünyesinde geliştirilen sistemler/yazılımlar özelinde gerçekleştirilmiştir. Seçilen öznitelikler savunma sanayisinde geliştirilen yazılımların altyapılarına ve ihtiyaçlarına göre belirlenmiştir. Her kuruma, projeye ve çalışan personel yelpazesine uygun bir model geliştirmek oldukça zordur. Her ku-

rumun kendi sektörüne ve çalışan yelpazesine uygun özniteliklerini belirlemesi daha uygundur.

Yazılım ve yazılım testin temelinde insan ve insanı etkileyen diğer çevresel faktörler bulunmaktadır. Performans temelinde bunlar olduğu için yazılım test dünyasında literatüre de geçen hiçbir efor tahmini gerçeği düşük bir hata payı ile kestirememektedir. Bu çalışmada gerçeğe en yakın şekilde yazılım test sürecini modelleyebilmek ve yazılım test eforu tahmini yapabilmek amaçlanmıştır. Önerilen öznitelik seçim stratejisi ile yapılan efor kestirimi proje planlarında daha verimli bir yazılım test sürecine izin vermektedir. Yapay öğrenme algoritmaları için kullanılabilecek öznitelik seçim sürecinde 'Gerçekleştirilecek Test Sayısı' yanında Savunma Sanayi Yazılım Projelerinde kullanılabilecek 'Teknik Faktör' hesaplama altyapısı kazandırılmıştır.

ASELSAN'da uygulanan mevcut test eforu tahmin yöntemi, yalnızca gerçekleştirilecek olan test sayısını baz almaktadır. Bu çalışmada test sayısı girdisine ilave olabilecek ve test eforunu etkileyen diğer faktörler de eklenerek daha gerçekçi bir yazılım test eforu tahmini yapılması sağlanmıştır.

Yazılım test eforunu etkileyen faktörlerin test sayısı, yazılımın tipi, gerçekleştirilen testlerin zorluk oranı, donanımsal altyapı ihtiyacı, test verisi ya da test otomasyonu kullanımı, dağıtık ve karmaşık sistemlerde çalışma ve test ekibinin deneyim seviyesi olduğu belirlenmiştir. Bu faktörler içerisinde test eforunu en çok etkileyen faktörün de test sayısı olduğu gözlenmiştir.

Gerçekleştirilen test sayısının en önemli öznitelik olması beklenen bir durumdur. Fakat savunma sanayi projelerinde kullanılan altyapıların her geçen gün kendilerini yenilemeleri, yeni donanımlara ve test düzeneklerine ihtiyaç duymaları nedeni ile kullanılan teknolojiyi ve de yapılacak işin platformunu belirlemek için gösterge görevini gören Teknik Faktörler özniteliği eklenmiştir. Ayrıca uzman görüşleri alınarak atanan Testlerin Zorluk Seviyesi özniteliği öğrenme periyoduna ihtiyacın duyulacağı karmaşık sistemler için önemli bir ayırıcıdır. Bu üç öznitelik beraber kullanıldıklarında günümüz yazılım test süreçleri için kullanılabilecek yazılım test eforu kestirimleri yapılabilmektedir.

Veri kümesinde kullandığımız sınırlı sayıdaki yazılım tipi genişletilerek ASELSAN Yazılım Test Müdürlüğü sorumluluğundaki tüm yazılım tiplerinin eklenmesi planlanmaktadır.

Bu çalışmada uyguladığımız yapay öğrenme metotları arasında veri kümesi üzerinde en iyi performansı sergileyen yöntem en iyileştirilerek ASELSAN süreçlerine uygulanacak ve gelecek projelerde test planlaması safhasında daha etkili test

eforu tahmini yapmak için kullanılması planlanmaktadır. Önerilen kestirim yazılımının kullanım senaryosu şu şekilde düşünülebilir: Gerçekleştirilmesi Planlanan Test Sayısı ve Yazılım Tipi girilir. Planlanan Test Senaryolarının Zorluk Seviyesi hesaplanır. Teknik Faktörler için Çizelge 5.8'de ve Denklem 5.2'de verildiği gibi Teknik Faktör değeri hesaplanır. Testlerin hangi deneyim seviyesinde bir ekip ile gerçekleştirileceği girilir. Öznitelikler geçmişte kullanılan tüm veriler ile eğitilmiş Yapay Öğrenme Algoritmasına girdi olarak kullanılıp test eforu kestirimi yapılır.

Veri kümesinde, sistemin performansını bozan verilerin çıkartılması için analiz çalışması yapılmıştır. Bu analize ek olarak bozuk veya gereksiz olduğu tespit edilerek çıkartılan bu veriler hakkında bir kök neden analizi yapılarak bu verilerde hataya sebep olan durumlar tespit edilecek ve bundan sonraki projeler için hazırlanacak olan Yazılım Test Raporlarına test eforu verisinin daha doğru şekilde girilmesi için süreç çalışması yapılacaktır.

Diğer projelerden toplanan yeni verilerin eklenmesi ile birlikte nispeten küçük olan veri kümemizin genişletilmesi ve uygulanan modeller üzerinde çeşitli optimizasyonlar yapılarak daha yüksek başarım elde edilebileceği öngörülmektedir.

KAYNAKLAR

[1] **Boucher, A., Badri, M.** Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison. *Information and Software Technology 96* (2018), 38–67.

[2] **Adali, E., Karagöz, N.A.** Software test effort estimation. *43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (2017).

[3] **Ahmed, B.S. and Zamli, K.Z.** A variable strength interaction test suites generation strategy using particle swarm optimization. *The Journal of Systems and Software 84* (2011), 2171–2185.

[4] **Alsolai, H., Roper, M.** A systematic literature review of machine learning techniques for software maintainability prediction. *Information and Software Technology 119*, 106214 (2020).

[5] **ANSI/IEEE std** 610.12. Ieee standard glossary of software engineering terminology.

[6] **Arora, I., Tetarwal, V., Saha, A.** Open issues in software defect prediction. *Procedia Computer Science 46* (2015), 906–912.

[7] **Bhattacharya, P., Ranjan, P., and Prasad, S.B.** Software test effort estimation using particle swarm optimization. *Proceedings of the International Conference on Information Systems Design and Intelligent Applications* (2012).

[8] **Bibi, S., Tsoumakas, G., Stamelos, I., Vlahavas, I.** Regression via classification applied on software defect estimation. *Expert Systems with Applications 34* (2008), 2091–2101.

[9] **Briand, L. C., Labiche Y., Bawar, Z.** Using machine learning to refine black-box test specifications and test suites. *The Eighth International Conference on Quality Software*, 10176809 (2008).

[10] **BROEKMAN, B., NOTENBOOM, E.** *Testing Embedded Software*. Addison-Wesley, Edinburgh Gate Harlow CM20 2JE, 2003.

[11] **BURNSTEIN, I.** *PRACTICAL SOFTWARE TESTING, A PROCESS-ORIENTED APPROACH*. Springer, 175 Fifth Avenue, New York, NY 10010, USA, 2002.

[12] **CHI, J., QU, Y., ZHENG, Q.** Relation-based test case prioritization for regression testing. *The Journal of Systems and Software 163*, 110539 (2020).

[13] **COTRONEO, D., PIETRANTUONO, R., RUSSO, S.** A learning-based method for combining testing techniques. *35th International Conference on Software Engineering (ICSE)*, 13799206 (2013).

[14] **CRISTIANINI, N.,TAYLOR, J.S.** *An Introduction to Support Vector Machines*. Cambridge University Press, University Printing House, Cambridge CB2 8BS , United Kingdom, 2014.

[15] **DUDA, R.O., HART, P.E., STORK, D.G.** Pattern classification. *Second Edittion ISBN-13: 978-0471056690* (2001).

[16] **DURELLI, V.H.S. AND DURELLI, R.S. AND BORGES, S.S.** Machine learning applied to software testing a systematic mapping study. *IEEE Transactions on Reliability 68*, 3 (2019), 1189–1212.

[17] **DUSTIN, E.** *Effective Software Testing*. Addison-Wesley, Edinburgh Gate Harlow CM20 2JE, 2002.

[18] **ELZAMLY, A., HUSSIN, B., SALLEH, N.M.** Top fifty software risk factors and the best thirty risk management techniques in software development lifecycle for successful software projects. *International Journal of Hybrid Information Technology 9*, 6 (2016), 11–32.

[19] **FELIPE, N.F.** A comparative study of three test effort estimation methods. *Revista Cubana de Ciencias InformÃ¡ticas 8* (2014).

[20] **FIELD,A.** *Discovering Statistics using IBM SPSS Statistics, 5th*. SAGE Publications Ltd, 2017.

[21] **FIRESMITH, D.** Using v models for testing. *SEI Technical Report* (2013).

[22] **GAVRILOV, Z.** Svm tutorial. *MIT Lecture Notes on Machine Learning* (2020).

[23] **GONDRA, I.** Applying machine learning to software fault-proneness prediction. *The Journal of Systems and Software 81* (2008), 186–195.

[24] **GROSS, H.** *Component-Based Software Testing with UML.* Springer, Springer-Verlag Berlin Heidelberg 2005, 2005.

[25] **HAYKIN, S.** *Neural Networks, A Comprehensive Foundation*. Pearson Prentice Hall, 2005.

[26] **HOURANI, H., HAMMAD, A., LAFI, M.** The impact of artificial intelligence on software testing. *Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, 18691627 (2019).

[27] **HUNT, J.** Testing control software using a genetic algorithm. *Contributed Paper 0952-1976* (1995).

[28] **KULKARNI, V.Y., SINHA, P.K.** Random forest classifiers :a survey and future research directions. *International Journal of Advanced Computing 36*, 1 (2013).

[29] **KUSHWAHA, D.S., MISRA, A.K.** Software test effort estimation. *ACM SIGSOFT Software Engineering Notes 33*, 3 (May 2008).

[30] **LIMA, J.A.P., VERGILIO, S. R.** Test case prioritization in continuous integration environments: A systematic mapping study. *Information and Software Technology 121*, 106268 (2020).

[31] **MALHOTRA, R.** Comparative analysis of statistical and machine learning methods for predicting faulty modules. *Applied Soft Computing 21* (2014), 286–297.

[32] **MYERS, G.J.** *The Art of Software Testing, Second Edition*. John Wiley Sons, Inc., Hoboken, New Jersey., 2004.

[33] **NAGESWARAN, S.** Test effort estimation using use case points. *Quality Week June* (2001).

[34] **OZAKINCI, R., TARHAN, A.** Early software defect prediction a systematic map and review. *The Journal of Systems and Software 144* (2018), 216–239.

[35] **PALIWAL, M., KUMAR, A.A.** Neural networks and statistical techniques: A review of applications. *Expert Systems with Applications 36*, 1 (2019), 2–17.

[36] **PANDEY, S.K., MISHRA, R.B., TRIP, A.K.** Software bug prediction prototype using bayesian network software bug prediction prototype using bayesian network. *International Conference on Computational Intelligence and Data Science 132* (2018), 1412–1421.

[37] **PANDEY, S.K., MISHRA, R.B., TRIPATHI, A.K.** Bpdet: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Systems With Applications 144*, 113085 (2020).

[38] **PRESSMAN, R.S.** *Software Engineering: A Practitioner s Approach*. McGraw-Hill, 2009.

[39] **SILVA, D.G.** Machine learning methods and asymmetric cost function to estimate execution effort of software testing. *Third International Conference on Software Testing Verification and Validation* (2010).

[40] **SRIVASTAVA, P.R.** Estimation of software testing effort using fuzzy multiple linear regression. *Int. J. Software Engineering 1*, 2/3/4 (2015).

[41] **TAHVILI, S., AFZAL, W., SAADATMAND, M. AND BOHLIN, M.** Espret: A tool for execution time estimation of manual test cases. *The Journal of Systems and Software 146* (2018), 26–41.

[42] **TOMBUL, H., OZBAYOGLU, M., OZBAYOGLU, E.** . Computational intelligence models for piv based particle (cuttings) direction and velocity estimation in multi-phase flows. *Journal of Petroleum Science and Engineering 172* (2019), 547–558.

[43] **TURHAN, B., BENER, A.** Yazilim hata kestirimi iÃ§in kaynak kod olcutlerine dayali bayes siniflandirmasi. *1st National Symposium on Software Engineering* (2007).

[44] **VAHID GAROUSI, SARA BAUER, MICHAEL FELDERER**. Nlp-assisted software testing: A systematic mapping of the literature. *Information and Software Technology 126*, 106321 (2020).

[45] **XIAO, P., LIU, B., WANG, S.** Feedback-based integrated prediction: Defect prediction based on feedback from software testing process. *The Journal of Systems and Software 143* (2018), 159–171.

[46] **XIAO, P., LIU, B., WANG, S.** Feedback-based integrated prediction: Defect prediction based on feedback from software testing process. *Journal of Systems and Software 143* (2018), 159–171.

[47] **Xie, X. and Ho, J.W.K. and Murphy, C. and Kaiser, G.** Testing and validating machine learning classifiers by metamorphic testing. *The Journal of Systems and Software 84* (2011), 544–558.

[48] **Zhang, D.** Machine learning in value-based software test data generation. *18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, 9308531 (2006).

[49] **Zhang, J., Li, J.** Testing and verification of neural-network-based safety-critical control software: A systematic literature review. *Information and Software Technology 123*, 106296 (2020).

ÖZGEÇMİŞ

Ad-Soyad : Özgenil MERİÇ

Uyruğu : TC

Doğum Tarihi ve Yeri : 14.06.1987 Eskişehir

E-posta : omeric@etu.edu.tr

ÖĞRENİM DURUMU:

• **Lisans** : 2010, Dokuz Eyül Üniversitesi, Müh. Fakültesi, Elektrik-Elektronik Mühendisliği

MESLEKİ DENEYİM VE ÖDÜLLER:

Yıl	**Yer**	**Görev**
2010-2011	VODAFONE A.Ş.	RF Planlama ve Optimizasyon Mühendisi
2012-2015	MİKES A.Ş.	Yazılım Test Mühendisi
2015-2020	ASELSAN A.Ş.	Yazılım Test Mühendisi

YABANCI DİL: İngilizce

TEZDEN TÜRETİLEN YAYINLAR, SUNUMLAR VE PATENTLER:

- **Meriç, Ö.**, Özbayoğlu, A.M., 2020. Makine Öğrenme ile Yazılım Test Eforu Kestirimi, 3rd International Conference on Data Science and Applications (ICONDATA'20), June 25-28, Istanbul, TURKEY ...

DİĞER YAYINLAR, SUNUMLAR VE PATENTLER:

- **Meriç, Ö.**, Üçüncü, E., Madencan, R., Öztarak, H., 2016. ASELSAN Ateş Destek Projelerinde Test Otomasyonu ile Doğrulama, SAVTEK, 12-14 Ekim, Ankara, Turkey.