T.C.

İSTANBUL KÜLTÜR ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MAKİNE ÖĞRENMESİ YÖNTEMLERİ İLE YAZILIM HATA TAHMİNİ

YÜKSEK LİSANS TEZİ

Murat ÇETİNER

1700005575

Anabilim Dalı: Bilgisayar Mühendisliği

Programı: Bilgisayar Mühendisliği

Tez Danışmanı: Doç. Dr. Özgür Koray ŞAHİNGÖZ

AĞUSTOS 2020

T.C.

İSTANBUL KÜLTÜR ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MAKİNE ÖĞRENMESİ YÖNTEMLERİ İLE YAZILIM HATA TAHMİNİ

YÜKSEK LİSANS TEZİ

Murat ÇETİNER

1700005575

Anabilim Dalı: Bilgisayar Mühendisliği

Programı: Bilgisayar Mühendisliği

Tez Danışmanı: Doç. Dr. Özgür Koray ŞAHİNGÖZ

Diğer Jüri Üyeleri: Dr. Öğretim Üyesi Hakan AYDIN (Gelişim Üniversitesi)

Dr. Öğretim Üyesi Bahar İLGEN

AĞUSTOS 2020

ÖNSÖZ

"MAKİNE ÖĞRENMESİ YÖNTEMLERİ İLE YAZILIM HATA TAHİMİNİ" adlı yüksek lisans tez çalışmam süresince bilgi ve deneyimi ile çalışmalarımı yönlendiren ve desteğini esirgemeyen değerli tez danışmanım Doç. Dr. Özgür Koray Şahingöz'e, her durumda şartsız ve koşulsuz desteklerini ve sevgilerini benden esirgemeyen aileme, katkıda bulunan tüm hocalarıma ve arkadaşlarıma teşekkürlerimi sunarım.

İÇİNDEKİLER

ŞEKİL LİSTESİ

TABLO LİSTESİ

KISALTMALAR

SVM	: Destek Vektör Makinesi Algoritması
KNN	: K-En Yakın Komşu Algoritması
RF	: Rastgele Orman Algoritması
NB	: Naif Bayes Algoritması
IEEE	: Elektrik ve Elektronik Mühendisleri Enstitüsü
DT	: Karar Ağacı Algoritması
NN	: Sinir Ağları
GBC	: Gradyan Arttırıcı Sınıflandırması
PCA	: Temel Bileşenler Analizi
AC	: Adaboost Sınıflandırması
BA	: Bagging (Torbalama) Algoritması
MLP	: Çok Katmanlı Algılayıcı
CV	: Çapraz Doğrulama
PCA	: Temel Bileşen Analizi
CM	: Karışıklık Matrisi
LR	: Doğrusal Regresyon
AUC	: Eğri Altındaki Alan

Üniversite	**:**	**T.C. İstanbul Kültür Üniversitesi**
Enstitüsü	**:**	**Lisansüstü Eğitim Enstitüsü**
Anabilim Dalı	**:**	**Bilgisayar Mühendisliği**
Program	**:**	**Bilgisayar Mühendisliği**
Tez Danışmanı	**:**	**Doç. Dr. Özgür Koray ŞAHİNGÖZ**
Tez Türü ve Tarihi	**:**	**Yüksek Lisans – Ağustos 2020**

ÖZET

MAKİNE ÖĞRENMESİ YÖNTEMLERİ İLE YAZILIM HATA TAHMİNİ

Günümüzde artan şekilde devam eden kritik işlemler ve bu işlemlerin detaylı süreçleri büyük çoğunlukla yazılımlar ile geliştirilmektedir. Bu yüzden ürünün yazılımının kalitesi ve yazılımının kusursuz olması; yazılımı geliştiren, yazılımı takip eden, ortaya çıkan ürünü test eden veya ürünü kullanan herkes için önemli bir uzmanlaşma alanı haline gelmiştir. Kalite modeli, bir ürünün kalitesini değerlendirmede sisteminin temel yapı taşıdır. Kalite modeli, bir yazılım ürününün özelliklerini değerlendirirken hangi kalite özelliklerinin dikkate alınacağını belirler. Bir sistemin kalitesi, sistemin çeşitli paydaşlarının belirtilen ve ima edilen ihtiyaçlarını karşılama ve bu sayede değer sağlama derecesidir. Bu paydaşların ihtiyaçları (işlevsellik, performans, güvenlik, sürdürülebilirlik vb.), ürün kalitesini özellikler ve alt özellikler olarak kategorize eden kalite modelinde tam olarak temsil edilen şeydir. ISO / IEC 25010'da tanımlanan ürün kalite modeli sekiz kalite özelliğini içermektedir. Bunlar; fonksiyonel uygunluk, performans verimliliği, uygunluk, kullanılabilirlik, güvenilirlik, güvenlik, idame ve taşınabilirliktir.

Yazılım Mühendisliği kavramında, yazılım hatalarının tahmini, yazılım geliştirme yaşam döngüsünün en kritik ve pahalı aşamalarından biri olan yazılım sistemlerinin kalitesini artırmada hayati bir rol oynamaktadır. Günlük yaşamımızda yazılım sistemlerinin kullanımı artarken dolayısıyla bağımlılıkları ve karmaşıklıkları da artmaktadır ve bu da hatalar için uygun bir ortam sağlamaktadır. Yazılım hatalarından dolayı, yazılım yanlış sonuçlar ve davranışlar üretmektedir. Hatalardan

daha kritik olan şey ise bu hataların meydana gelmeden bulunmasıdır. Bu nedenle, yazılım hatalarının tespiti ve ayrıca tahmini, yazılım yöneticilerinin bakım ve test aşamaları için kaynakların verimli bir şekilde tahsis edilmesini sağlar. Literatürde, yazılım hatalarının tahmini için farklı öneriler vardır. Bu çalışmada, Karar Ağacı, Naif Bayes, K-En Yakın Komşu, Destek Vektör Makinesi, Rastgele Orman, Ekstra Ağaçlar, Adaboost, Gradient Boosting olarak toplamda 10 makine öğrenmesi algoritması üzerinde çalışılarak ve bu algoritmaları birbirleri ile karşılaştırarak makine öğrenme tabanlı yazılım hatası tahmin sistemleri hakkında karşılaştırmalı bir analiz yapılmıştır.

Günümüzde yazılım hata tahmini araştırmalarında kullanılan veri kümeleri dengeli veri ve dengesiz veri olarak adlandırılmaktadır. Dengeli olmayan veri birbirinden farklı sınıflarda kayıt sayılarının eşit olmadığı bir yapıda sınıf dağılımı olacak şekilde olmasıdır. Dengeli veri ise sınıf dağılımı eşit olan veri kümelerine denilmektedir.

Anahtar Kelimeler: Yazılım Hata Tahmini, Makine Öğrenmesi, Sınıflandırma, Phyton ile Sınıflandırma.

University : **T.C. İstanbul Kültür University**
Institute : **Institute of Graduate Studies**
Department : **Computer Engineering**
Program : **Computer Engineering**
Thesis Advisor : **Assoc. Prof. Özgür Koray ŞAHİNGÖZ**
Degree Awerded And Date : **MA – August 2020**

ABSTRACT

SOFTWARE DEFECT PREDICTION WITH MACHINE LEARNING METHODS

Today, increasingly ongoing critical processes and detailed processes of these processes are mostly developed by software. Therefore, the quality of the software and the flawless software of the product; It has become an important area of specialization for anyone who develops the software, follows the software, tests the resulting product, or uses the product. The quality model is the basic building block of the system in evaluating the quality of a product. The quality model determines which quality features to consider when evaluating the features of a software product. The quality of a system is the degree to which the system meets the stated and implied needs of the various stakeholders and thus provides value. The needs of these stakeholders (functionality, performance, safety, sustainability, etc.) are exactly what is represented in the quality model that categorizes product quality into features and sub-features. The product quality model defined in ISO / IEC 25010 includes eight quality characteristics: functional suitability, performance efficiency, suitability, usability, reliability, security, maintenance, and portability.

In the concept of Software Engineering, the prediction of software errors plays a vital role in improving the quality of software systems, one of the most critical and expensive stages of the software development life cycle. As the use of software systems increases in our daily life, their dependencies and complexities are also increasing, and this provides a suitable environment for errors. Due to software errors, the software produces incorrect results and behavior. More critical than errors is that these errors are found before they occur. Therefore, the detection, as well as the prediction of software errors, enables software administrators to efficiently allocate resources for maintenance and testing phases. In the literature, there are different

suggestions for the prediction of software errors. In this study, by working on a total of 10 machine learning algorithms such as Decision Tree, Naive Bayes, K-Nearest Neighbor, Support Vector Machine, Random Forest, Extra Trees, Adaboost, Gradient Boosting and comparing these algorithms with each other, machine learning-based software error prediction systems a comparative analysis was made about.

Data sets used in software error prediction research today are called balanced data and unbalanced data. Unbalanced data is that there is a class distribution in a structure where the number of records in different classes is not equal. Balanced data are data sets with the equal class distribution.

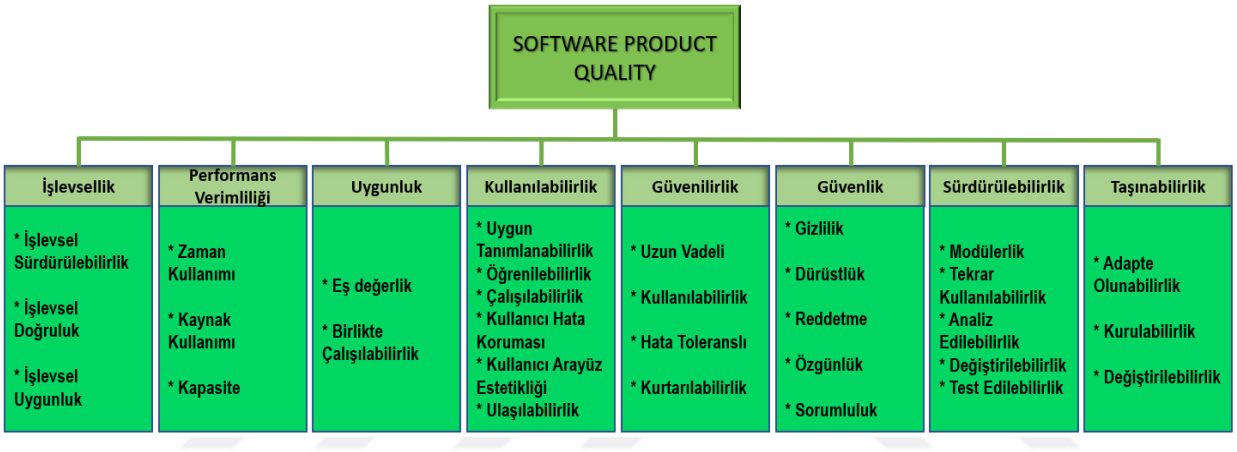
Keywords: Software Defect Prediction, Machine Learning, Classification, Classification with Pyhton.

1. GİRİŞ

Günümüzde artan şekilde devam eden kritik işlemler ve bu işlemlerin detaylı süreçleri büyük çoğunlukla yazılımlar ile geliştirilmektedir. Bir yazılım projesinin başarısını belirleyen ana faktör kalitesidir [1].

Yazılım kalitesi için birden fazla tanım bulunmakla birlikte bunlar icinde öne çıkanı; Yazılımın ne kadar iyi tasarlandığı ve çıkan ürünün bu tasarıma ne kadar uyduğudur [2].

Yazılım kalitesini ifade eden ISO / IEC 25010, Şekil 1.1'de gösterildiği gibi sekiz kalite özelliği içermektedir. [3].



Şekil 1.1- ISO 25010 Yazılım Ürün Kalitesi

1. **İşlevsellik (Fonksiyonel Uygunluk)**

Bu özellik, bir ürün veya sistemin belirli koşullar altında kullanıldığında belirtilen ve ima edilen ihtiyaçlarını karşılayan işlevleri sağlama derecesini temsil eder. Bu özellik aşağıdaki alt özelliklerden oluşmaktadır:

- ✓ **İşlevsel Sürdürülebilirlik:** İşlev setinin belirtilen tüm görevleri ve kullanıcı hedeflerini kapsama derecesidir.
- ✓ **İşlevsel Doğruluk:** Bir ürün veya sistemin gerekli hassasiyet derecesiyle doğru sonuçları sağlama derecesidir.
- ✓ **İşlevsel Uygunluk:** İşlevlerin, belirtilen görevlerin ve hedeflerin gerçekleştirilmesini kolaylaştırma derecesidir.

2. **Performans Verimliliği**

Bu özellik, belirtilen koşullar altında kullanılan kaynak miktarına göre performansı temsil eder. Bu özellik aşağıdaki alt özelliklerden oluşmaktadır:

- ✓ **Zaman Kullanımı:** Bir ürün veya sistemin, işlevlerini yerine getirirken yanıt ve işlem sürelerinin ve üretim hızlarının gereksinimleri karşılama derecesidir.

- ✓ **Kaynak Kullanımı:** Bir ürün veya sistem tarafından işlevlerini yerine getirirken kullanılan kaynakların miktarlarının ve türlerinin gereksinimleri karşılama derecesidir.
- ✓ **Kapasite:** Bir ürünün veya sistem parametresinin maksimum sınırlarının gereksinimleri karşılama derecesidir.

3. Uygunluk

Bir ürün, sistem veya bileşenin diğer ürünler, sistemler veya bileşenlerle bilgi alışverişinde bulunma ve aynı donanım veya yazılım ortamını paylaşırken gerekli işlevleri yerine getirme derecesidir. Bu özellik aşağıdaki alt özelliklerden oluşmaktadır:

- ✓ **Eşdeğerlik:** Bir ürünün, diğer ürünler üzerinde zararlı bir etki olmadan, diğer ürünlerle ortak bir ortamı ve kaynakları paylaşırken gerekli işlevlerini verimli bir şekilde yerine getirebilme derecesidir.

Birlikte Çalışabilirlik: İki veya daha fazla sistemin, ürünün veya bileşenin bilgi alışverişinde bulunabileceği ve takas edilen bilgileri kullanabileceği derecedir.

4. Kullanılabilirlik

Belirli bir kullanım bağlamında etkinlik, verimlilik ve memnuniyetle belirtilen hedeflere ulaşmak için belirli kullanıcılar tarafından bir ürünün veya sistemin kullanılma derecesidir. Bu özellik aşağıdaki alt özelliklerden oluşmaktadır:

- ✓ **Uygunluk Tanımlanabilirlik:** Kullanıcıların, bir ürün veya sistemin ihtiyaçlarına uygun olup olmadığını anlayabildikleri derecedir.
- ✓ **Öğrenilebilirlik:** Bir ürünün veya sistemin, belirli bir kullanım bağlamında ürün veya sistemi etkililik, verimlilik, risksizlik ve memnuniyetle kullanmayı öğrenme hedeflerine ulaşmak için belirli kullanıcılar tarafından kullanılma derecesidir.
- ✓ **Çalışılabilirlik:** Bir ürün veya sistemin, çalıştırmayı ve kontrol etmeyi kolaylaştıran niteliklere sahip olma derecesidir.
- ✓ **Kullanıcı Hata Koruması:** Bir sistemin, kullanıcıları hata yapmaya karşı koruma derecesidir.
- ✓ **Kullanıcı Arayüzü Estetikliği:** Bir kullanıcı arayüzünün, kullanıcı için hoş ve tatmin edici bir etkileşim sağlama derecesidir.
- ✓ **Erişilebilirlik:** Bir ürünün veya sistemin, belirli bir kullanım bağlamında belirli bir hedefe ulaşmak için en geniş özellik ve yeteneklere sahip kişiler tarafından kullanılma derecesidir.

5. Güvenilirlik

Bir sistem, ürün veya bileşenin belirli bir süre boyunca belirli koşullar altında belirli işlevleri

yerine getirme derecesidir. Bu özellik aşağıdaki alt özelliklerden oluşmaktadır:

- ✓ **Uzun Vadelilik:** Bir sistemin, ürünün veya bileşenin normal çalışma altında güvenilirlik ihtiyaçlarını karşılama derecesidir.
- ✓ **Kullanılabilirlik:** Bir sistemin, ürünün veya bileşenin operasyonel ve kullanım için gerektiğinde erişilebilir olma derecesidir.
- ✓ **Hata Toleranslılık:** Bir sistemin, ürünün veya bileşenin, donanım veya yazılım hatalarının varlığına rağmen amaçlandığı gibi çalıştığı derecedir.
- ✓ **Kurtarılabilirlik:** Bir kesinti veya arıza durumunda, bir ürünün veya sistemin doğrudan etkilenen verileri kurtarabileceği ve sistemin istenen durumunu yeniden kurabileceği derecedir.

6. Güvenlik

Kişilerin veya diğer ürünlerin veya sistemlerin, türlerine ve yetki düzeylerine uygun veri erişim derecesine sahip olması için bir ürün veya sistemin bilgileri ve verileri koruma derecesidir. Bu özellik aşağıdaki alt özelliklerden oluşmaktadır:

- ✓ **Gizlilik:** Bir ürün veya sistemin, verilere yalnızca erişim yetkisi verilenler tarafından erişilebilir olmasını sağlama derecesidir.
- ✓ **Dürüstlük:** Bir sistemin, ürünün veya bileşenin bilgisayar programlarına veya verilere yetkisiz erişimi veya değiştirilmesini önleme derecesidir.
- ✓ **Red Edilebilirlik:** Olayların veya eylemlerin daha sonra reddedilememesi için eylemlerin veya olayların gerçekleşmiş olduğunun kanıtlanma derecesidir.
- ✓ **Özgünlük:** Bir özne veya kaynağın kimliğinin iddia edilen kişi olduğunun kanıtlanabileceği derecedir.

Sorumluluk: Bir varlığın eylemlerinin benzersiz bir şekilde varlığa kadar izlenebilme derecesidir.

7. Sürdürülebilirlik

Bu özellik, bir ürünün veya sistemin iyileştirilmesi, düzeltilmesi veya ortamdaki ve gereksinimlerdeki değişikliklere uyarlanması için değiştirilebileceği etkinlik ve verimlilik derecesini temsil eder. Bu özellik aşağıdaki alt özelliklerden oluşmaktadır:

- ✓ **Modülerlik:** Bir sistem veya bilgisayar programının bir bileşende yapılan değişikliğin diğer bileşenler üzerinde minimum etkiye sahip olacağı şekilde ayrı bileşenlerden oluşma derecesidir.
- ✓ **Tekrar Kullanılabilirlik:** Bir varlığın birden fazla sistemde veya başka varlıkların oluşturulmasında kullanılma derecesidir.
- ✓ **Analiz Edilebilirlik:** Bir veya daha fazla parçasında amaçlanan bir değişikliğin bir

ürün veya sistem üzerindeki etkisini değerlendirmenin veya bir ürünü eksiklikler, arıza nedenleri için teşhis etmenin veya parçaları tanımlamanın mümkün olduğu etkililik ve verimlilik derecesidir.

- ✓ **Değiştirilebilirlik:** Bir ürün veya sistemin, kusurlara neden olmadan veya mevcut ürün kalitesini düşürmeden etkili ve verimli bir şekilde değiştirilebileceği derecedir.
- ✓ **Test Edilebilirlik:** Test kriterleri; bir sistemin, ürünün veya bileşenin ve testler için tespit edilebilme derecesidir.

8. Taşınabilirlik

Bir sistem, ürün veya bileşenin bir donanımdan, yazılımdan veya başka bir işletim veya kullanım ortamından diğerine aktarılabileceği etkinlik ve verimlilik derecesidir. Bu özellik aşağıdaki alt özelliklerden oluşmaktadır:

- ✓ **Adapte Olunabilirlik:** Bir ürün veya sistemin farklı veya gelişen donanım, yazılım veya diğer işletim veya kullanım ortamlarına etkili ve verimli bir şekilde uyarlanabileceği derecedir.
- ✓ **Kurulabilirlik:** Bir ürün veya sistemin belirli bir ortamda başarıyla kurulabileceği ve / veya kaldırılabileceği etkinlik ve verimlilik derecesidir.
- ✓ **Değiştirilebilirlik:** Bir ürünün, aynı ortamda aynı amaç için belirtilen başka bir yazılım ürününü değiştirebilme derecesidir.

Dünyada giderek sayısı artan şekilde veriler bulunmaktadır. Resim, yazı ve video gibi türlerden oluşan bu verilerin bir yöntemle anlamlandırılması gerekmektedir. Bu anlamlandırmayı matematiksel ve istatiktiksel olarak insanlar yapabilirler ancak insanların bir ömür boyu yapabileceği şeyleri bir makine çok kısa sürede gerçekleştirebilmektedir. Makineler bu verileri anlamlandırmaktırlar. Bu makinelerin kullandığı algoritmalara makine öğrenmesi algoritmaları denir.

1.1. Problem Tanımı

Bir problemin çözümünü üretebilmek için ilk olarak problemin ne olduğunun iyi bir şekilde anlaşılması gerekmektedir. Geliştirilen tüm yazılımlar muhtemel hataya açıktır. Diğer bir deyişle; iyi bir analiz ve iyi bir test aşamasından geçip, kullanıcı ile bütünleşmediği sürece de muhtemelen tüm yazılımlar hatalıdır.

Yazılım hata tahmini, en kullanışlı ve en düşük maliyetli işlemlerden biri olarak görülmektedir. Yazılımın kullanıcı ile buluştuktan sonra hatalar meydana getirmesi başta yazılımın kendi imajı olmak üzere yazılımın yayımcısı için de birçok kötü sonuca yol açabilmektedir. Yazılım hata tahmini, yazılımcılar tarafından geliştirilmekte olan ürünün

kalitesinin bağlı olduğu hayati bir aşama olarak görülebilmektedir. Bunun yanı sıra müşterilerin ürün kalitesi ile ilgili tepkisi tatmin edici olup gururlandırıcı bir hal alacaktır. Yazılım hata tahmini, yazılım geliştirme yaşam döngüsünün ve test aşamasının en yardımcı faaliyetlerinden biridir. Kusur eğilimli ve kapsamlı test gerektiren modülleri tanımlar. Bu şekilde, test kaynakları kısıtlamaları ihlal etmeden verimli bir şekilde kullanılabilmektedir [4].

1.2. Literatüre Katkıları

Geliştirilen model, günümüzde artık her alanda kullanımı yaygınlaşan yazılım ve bu yazılımların ortaya çıkardığı ürünlere katkı sağlayabilecektir. Kullanıcı makine öğrenme algoritmalarının sağladığı kolaylık sayesinde hem zamandan kazanacaktır hem de yazılımı erkenden test etme, hata tespit etme imkânı bulacaktır.

Katkı 1. Günümüzde hemen hemen her alanda kullanılan yazılımların hataya açık olma ihtimali vardır. Yazılımcının daha yapılan işi teslim etmeden belirli metrikler yardımıyla yazılımında hata olma ihtimalini test edebilmesi amaçlanmıştır.

Katkı 2. Makine öğrenmesi teknikleri, öngörülü yazılım modelleri oluşturmaya nasıl katkıda bulunur sorusunun cevabı amaçlanmıştır.

Katkı 3. Geliştirilen çalışma sayesinde, önemli bir konu olan maliyetin en aza indirilmesi amaçlanmaktadır. En az hata ile en fazla başarı oranı sağlamak bizim için temel amaçlardan biridir. Her yöntem her veride aynı efektif sonucu vermeyebilir. Bunun için hangi veri kümelerinde hangi metodolojinin yararlı olduğunun karşılaştırmalı analizinin çıkarılması amaçlanmıştır.

Katkı 4. Geliştirilen çalışmada amaçlardan biri de çok fazla miktarda veri işlenirken seçilen algoritmaların nasıl sonuç verdiğidir.

Katkı 5. Yapılan çalışmada diğer önemli bir katkı zaman kavramıdır. Hatayı önceden tespit etmek hatanın ilerde oluşturacağı sorunlarıda önceden tespit etmek olacağı için her anlamda maliyeti azaltacağı amaçlanmıştır.

1.3. Tezin Organizasyonu

Bu tez çalışması 5 bölümden oluşmaktadır.

- Birinci bölümde, problem tanımı yapılmıştır. Yapılan çalışma tanıtılmıştır, amacı ve önemi anlatılmıştır ve literatüre katkısından söz edilmiştir.
- İkinci bölümde, tez çalışmasının ana kaynağı olan makine öğrenmesi algoritmalarından ve kullanılan veri setlerinden bahsedilmiştir.
- Üçüncü bölümde, çalışmada kullanılan yöntemler, çalışmada kullanılan teoriler,

yaklaşımlardan ve bunların nasıl uygulandığından, amaçlarından bahsedilmiştir.

- Dördüncü bölümde, yapılan çalışma önerilen yöntemle ilgili detaylardan, bileşenlerden, yöntemin akış diyagramından detaylı bir şekilde bahsedilmiştir.
- Beşinci yani son bölümde ise yapılan testler ve sonuçları detaylı bir şekilde belirtilmiştir.

2. ÖN BİLGİLER ve LİTERATÜR ARAŞTIRMASI

Bu bölümde araştırması yapılan çalışmanın temel tanımları ve çalışma için yapılan literatür taraması sunulmuştur. İlk olarak yazılım hatası tanımının ne olduğu ve hata türlerinin neler olduğu anlatılmıştır. Daha sonra makine öğrenmesi tanımı yapılmıştır ve beraberinde tezde kullanılan makine öğrenmesi algoritmaları özetlenmiştir. Bu bölümün sonunda ise konuya ilişkin yapılmış olan akademik çalışmalardan kullanılanlar özetlenmiştir.

2.1. Yazılım Hata Tanımı ve Türleri

2.1.1. Yazılım Hata Tanımı

Yazılım mühendisliği alanında bir yazılım ürününün gerçek, beklenen veya istenen çıktısı arasında meydana gelen her türlü sapma ve uyumsuzluk genellikle yazılım hatası olarak adlandırılır. Bunlar genellikle bir geliştirici tarafından istenmeyen bir şekilde yazılım ürününe eklenir.

2.1.2. Yazılım Hata Türleri

Yazılım için hata türleri çok geniş bir kavram olup, dünyanın her yerindeki alanlarda birbirleri yerine kullanılmaktadır. Bu hata türleri yazılımın farklı yönlerini belirtip, temsil etmektedir. Bu terimler yazılım test süreçlerinin ayrılmaz bir parçasıdır ve bunları test edip tanımlamadan test ekibi yazılımın kalitesini, yazılımın etkinliğini, yazılımın işlevselliğini ve daha fazlasını doğrulayamamaktadır. Tüm bu tutarsızlıklar sistemi farklı şekilde etkiler ve hepsi farklı birer tanıma sahiptir.

Yazılım hatası tahmin yöntemi ise yazılım testi ve yazılım bakım yaşam döngülerinde yardımcı olan bir yaklaşımdır. Elektrik ve Elektronik Mühendisleri Enstitüsü'nün (IEEE) 104 standardına göre [5], yazılım hataları için farklı sınıflandırma türleri vardır. Bu türler arasındaki bağlantı Şekil 2.1'de gösterilmiştir.

Hata (Error): Bu tipteki hataların nedeni, yazılım geliştiricisinin yanlış anlaması veya yanlış yorumlamasıdır. Herhangi bir yanlış döngü, söz dizimi, anlamadaki karmaşıklık, yanlış değer hesabı, tasarım veya gereksinim faaliyetindeki hatalar, gerçek ve beklenen sonuçlar arasındaki tutarsızlıklar birer örnektir. Kod ve sistem incelemesi ile yazılım kalitesi artırılarak, sorunlar belirlenerek, uygun etki zaman planı çıkartılarak ve kapsamlı test yapılarak önlenebilmektedir.

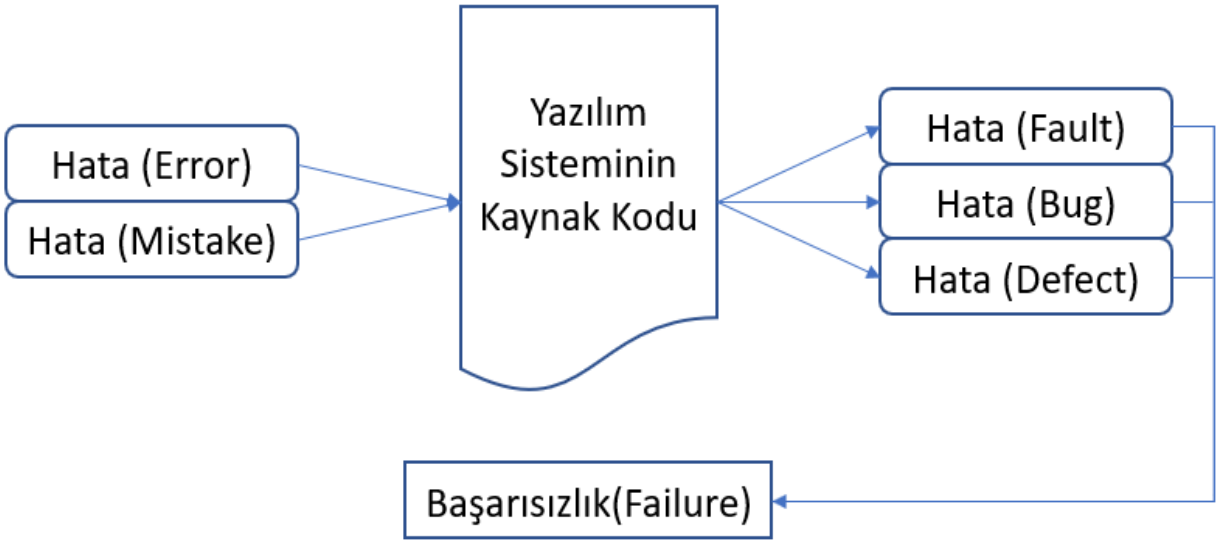
Arıza (Fault): Bir hata nedeniyle yazılım test ekibi tarafından bulunan tutarsızlıklardır. Yazılımın kullanımı esnasında bir hata meydana gelmesinin nedeni, gerçek ve beklenen değerlerin yanlış hesaplanmasından değil, yazılımdaki herhangi bir yerdeki hatanın belirlenmesidir. Ayrıca, yazılım sistemindeki bir hata yazılımın için amaçlanan işlevini gerçekleştirmesini engeller ve

sistemi beklenmedik bir şekilde hareket etmeye zorlamaktadır. Kapsamlı bir kod analizi veya işlevsel gereksinimlerin ölçülmesi ile önlenebilmektedir.

Hata (Bug): Bir yazılım sisteminin en ayrılmaz parçalarıdırlar. Bu hatalar yalnızca performansı etkilemekle kalmaz, aynı zamanda yazılımın beklenmedik şekilde davranmasına yol açar. Bu hatalar genellikle yazılımın kaynak kodunda yapılan hatalardan, tasarımındaki sorunlardan ve sisteme tanıtılan yanlışlıklardan meydana gelir. Test odaklı geliştirme ve iyi bir kod analizi ile önlenebilmektedirler.

Başarısızlık (Failure): Bir yazılım gerekli işlevlerini yerine getiremediğinde ve yeterli olmayan sonuçlarla karşılaştığında başarısızlık ve hata olarak adlandırılır. İnsan hataları, çevresel hatalar, kullanıcı hataları ve sistemsel sorunlar buna neden olabilmektedir. Test tekniğinden emin olunması ve tutarlı bir analiz sayesinde önlenebilmektedirler.

Hata (Defect): Yazılım sistemlerinin gerçek ve beklenen sonucu arasındaki farktır. Bazen kolayca düzeltilebilir. Bazen ise hayati sonuçlara yol açabilmektedirler. Sorunların bulunması için ise ciddi maliyet harcanması gerekebilmektedir. Tasarlanan süreçleri uygulayan yazılımcılar bu süreçteki hatalardan sorumludurlar. Yazılımın amaca uygun hizmet etmesinden yanı sıra yazılımın performansı da ciddi şekilde önemlidir. Verimli olabilecek türdeki program teknikleri benimsenerek, etkili ve doğru yazılım metodolojileri kullanılarak önlenebilmektedirler.



Şekil 2.1- Yazılım Hata Sınıflayıcıları Arasındaki İlişki

2.2. Makine Öğrenmesi

2.2.1. Makine Öğrenmesi Tanımı ve Amacı

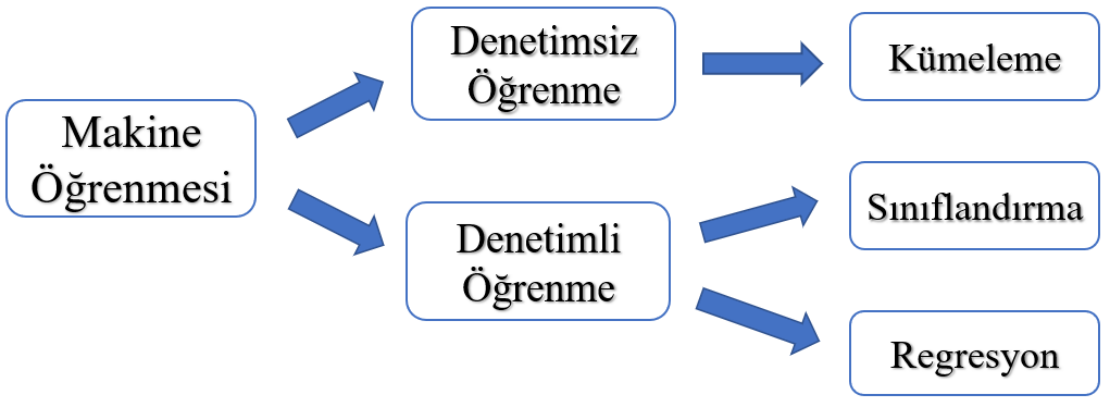
Makine öğrenimi, sistemlere açıkça programlanmadan deneyimden otomatik olarak öğrenme

ve geliştirme yeteneği sağlayan yapay zekanın bir uygulamasıdır. Yazılım mühendisleri, sistem geliştirme aşamalarını zaman ve maliyet tüketimlerini en aza indirgemek için makineleri kullanmaktadırlar [5]. Makine öğrenimi, verilere erişebilen ve bunları kendileri için kullanabilen bilgisayar programı geliştirilmesine odaklanır.

Öğrenme süresi, verilen örneklere dayanarak verilerdeki kalıpları aramak ve gelecekte daha iyi kararlar vermek için, doğrudan deneyim veya komutlar ile başlar. Makine öğrenmesinin birincil amacı, bilgisayarların insan yardımı olmadan otomatik olarak öğrenme sağlaması ve öğrendiklerini kullanması gereken an gelince eyleme dönüştürebilmesidir.

2.2.2. Makine Öğrenmesi Yöntemleri

Bu bölümde makine öğrenmesi algoritmalarındaki üç farklı öğrenme yöntemi anlatılmıştır. Şekil 2.2'de makine öğrenmeleri kategorize edilip gösterilmiştir.



Şekil 2.2- Makine Öğrenmesi Yöntemleri

Denetimli Öğrenme: Denetimli öğrenmenin ilk adımı iyi tanımlanmış olan geniş bir eğitim verisine sahip olmaktır [6]. Giriş ve çıkış verileri, gelecekteki veri işleme için bir öğrenme temeli sağlamak üzere etiketlenmiştir. Denetimli öğrenme terimi, bu algoritmanın öğretmen olarak düşünülebilecek bir eğitim veri kümesinden öğrenildiği fikrinden gelmektedir. Denetimli öğrenme problemleri; sınıflandırma ve regresyon olarak gruplandırılırlar.

Sınıflandırma problemi, çıktı değişkeninin "kırmızı" veya "mavi", "hastalık var" veya "hastalık yok" gibi kategori olması durumudur.

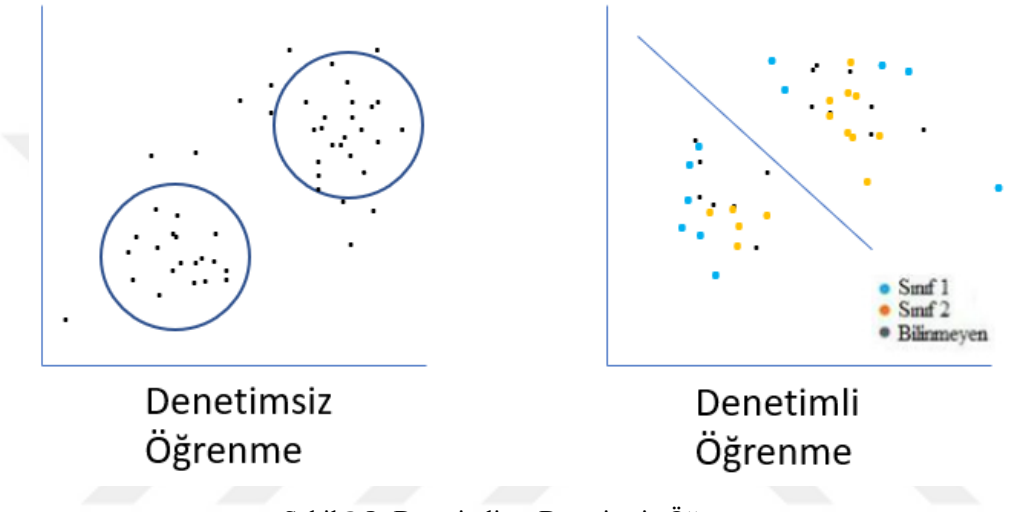
Regresyon problemi, çıktı değişkeninin "dolar" veya "ağırlık" gibi gerçek bir değer olmasıdır.

En yaygın kullanılan denetimli öğrenme algoritmaları; Destek Vektör Makineleri (SVM), Karar ağaçları (DT), K-En Yakın Komşu Algoritması (KNN), Naif Bayes (NB) ve Regülasyon olarak sıralanabilmektedir.

Denetimsiz Öğrenme: Yalnızca giriş verilerinin olduğu ve buna karşılık gelen çıkış

verilerinin olmadığı öğrenmedir. Denetimsiz öğrenmenin amacı, veriler hakkında daha fazla bilgi edinmek için verilerin temelini oluşturan yapıyı veya dağılımı modellemektir. Denetimli öğrenmenin aksine doğru cevapları yoktur ve tabiri caizse öğretmenleri yoktur. Denetimsiz öğrenme teknikleri rekabetçi öğrenme tekniklerdir [7].

Kümeleme problemi, satın alma davranışı yolu ile müşterileri gruplama gibi verilerden doğal gruplamaların keşfedilmek istenildiği problemlerdir. Şekil 2.3'de denetimli ve denetimsiz öğrenme farklı gösterilmiştir.



Şekil 2.3- Denetimli ve Denetimsiz Öğrenme

Yarı Denetimli Öğrenme: Büyük miktarda giriş verisine ve sadece bazı çıkış verilerinin etiketlendiği yöntemlere yarı denetimli öğrenme denir. Bu problemler denetimli ve denetimsiz öğrenme arasındadır. Örnek olarak sınıflandırma ve regresyon verilebilmektedir. Etiketlenmemiş verilerin nasıl modelleneceği hakkında varsayımlar yapan diğer yöntemlerin uzantılarıdırlar.

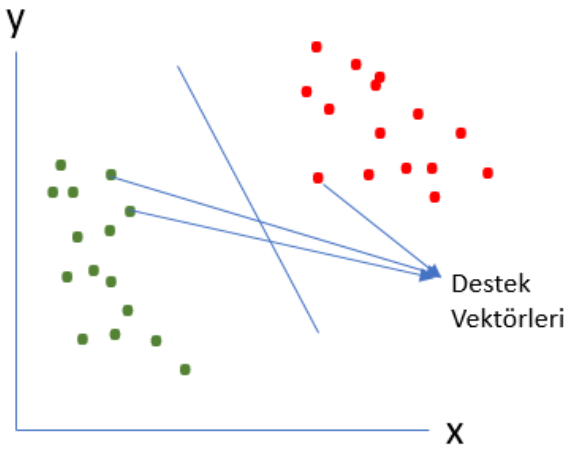
2.2.3. Yararlanılan Makine Öğrenmesi Algoritmaları

2.2.3.1. Destek Vektör Makinesi Algoritması (SVM)

SVM hem sınıflandırma hem de regresyon problemleri için kullanılabilen denetimli bir makine öğrenme algoritmasıdır. Ancak çoğunlukla sınıflandırma problemlerinde kullanılmaktadır. SVM temel olarak verileri iki sınıftan en uygun şekilde ayırmak için kullanılır. Bunun için karar sınırları veya başka bir deyişle hiper düzlemleri belirler. Boyut sayısı örnek sayısından fazla olduğunda etkilidir. Kararın yeni verilere dirençli olması için sınır çizgisinin iki sınıfın sınır çizgisine en yakın olması gerekir. Bu sınır çizgisine en yakın noktalara destek noktaları denir [8]. Şekil 2.4'de gösterilmiştir.

- Net bir ayrılma marjı ile gerçekten iyi çalışmaktadır.

- Yüksek boyutlu alanlarda etkilidirler.
- Boyut sayısının örnek değer sayısından fazla olduğu durumlarda etkilidirler.
- Karar işlevlerinde eğitim noktalarının bir alt kümesini kullanır bu nedenle bellek kullanmada verimlidirler.
- Veri seti daha fazla gürültüye sahip olduğunda yani hedef sınıfların çakıştığı durumlarda iyi performans göstermemektedir.



Şekil 2.4- Destek Vektörleri

2.2.3.2. Karar Ağacı Algoritması (DT)

Karar ağaçları, yazılım hata modüllerini kurallar kullanarak sınıflandırmaktadır. Karar ağacında bulunan temel bileşenler; karar düğümü, dallar ve yapraklardır. Karar ağacı içindeki girdi alanı karşılıklı olarak farklı bölgelere bölünmektedir ve veri noktalarını karakterize etmek için her bölgeye bir değer veya bir etiket atanmaktadır. DT mekanizması şeffaftır ve karar ağacının nasıl yapıldığını görmek için karar ağacı yapısı takip edilebilmektedir. Çoğu karar ağaçları yapım algoritması iki aşamadan oluşmaktadır. İlk aşamada, çok büyük boyutlu bir ağaç inşa edilir ve daha sonra aşırı uyum sorunlarını önlemek için ağaç ikinci adımda budanır. Sonra budama ağacı sınıflandırma amaçları için kullanılır [9]. Karar ağaçları, rasgele orman, gradyan artırma gibi yöntemler her türlü veri bilimi probleminde yaygın olarak kullanılmaktadır. Şekil 2.5'de basit bir karar ağacı yapısı gösterilmiştir.

Karar ağaçları ile kullanılan yaygın terimler:

Kök Düğümü: Tüm popülasyonu veya numuneyi temsil eder. İki veya daha fazla homojen kümeye bölünür.

Bölme: Bir düğümü iki veya daha fazla alt düğüme bölme işlemidir.

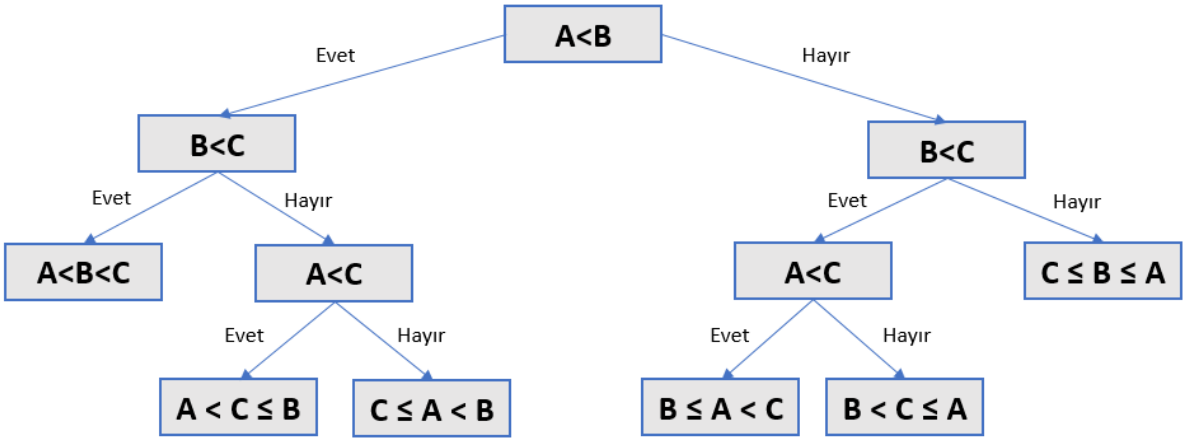
Karar Düğümü: Bir alt düğüm diğer alt düğümlere ayrıldığında, buna karar düğümü denir.

Yaprak / Terminal Düğümü: Bölünmeyen düğümlere verilen isimdir.

Budama: Bir karar düğümünün alt düğümleri kaldırıldığında, bu işleme budama denir.

Dal / Alt Ağaç: Tüm ağacın alt bölümüne dal veya alt ağaç denir.

Üst / Alt Düğüm: Alt düğümlere bölünmüş bir düğüme alt düğümlerin üst düğümü denir, alt düğümler ise üst düğümün alt öğesidir.



Şekil 2.5- Karar Ağacı Yapısı

2.2.3.3. Naif Bayes Algoritması (NB)

NB sınıflandırma algoritması, Matematikçi Thomas Bayes'in adını taşıyan bir sınıflandırma algoritmasıdır. Olasılık ilkeleri ile tanımlanan bir dizi hesaplama ile sisteme sunulan veri sınıfını belirlemeyi amaçlamaktadır. NB sınıflandırmasında, sistem belirli miktarda veri sağlar. Öğretim için sunulan bir veri sınıfı olmalıdır. Öğretilen veriler üzerinde olasılık işlemleri ile sisteme sunulan yeni test verileri, daha önce elde edilen olasılık değerlerine göre çalıştırılır ve verilen test verileri hangi kategoride belirlenir. Ne kadar çok veri öğretilirse, test verilerinin gerçek kategorisini belirlemek o kadar kesin olabilmektedir [10]. Bayes kuralı denklem 2.1'de gösterilmiştir.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (2.1)$$

P (A/B) = B olayı gerçekleştiğinde A olayının gerçekleşme olasılığıdır.

P (A) = A olayının gerçekleşme olasılığıdır.

P (B/A) = A olayı gerçekleştiğinde B olayının gerçekleşme olasılığıdır.

P (B) = B olayının gerçekleşme olasılığıdır.

Algoritmanın çalışma şekli, bir eleman için her durumun olasılığını hesaplar ve olasılık değeri en yüksek olana göre sınıflandırır. Az bir eğitim verisiyle çok başarılı işler

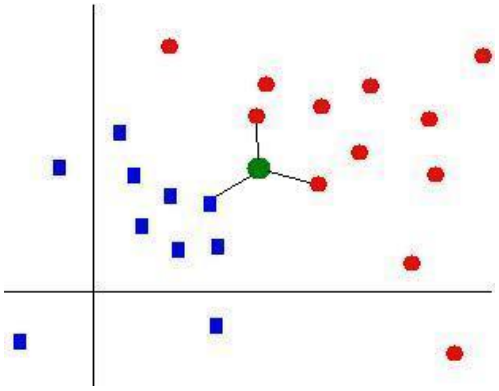
çıkartabilmektedir. Test kümesindeki bir değerin eğitim kümesinde gözlemlenemeyen bir değeri varsa olasılık değeri olarak sıfır verir yani tahminleme yapamaz. Bu durum genellikle "Sıfır Frekans" adıyla bilinir. Bu durumu çözmek için ise düzeltme teknikleri kullanılmaktadır.

2.2.3.4. K-En Yakın Komşu Algoritması (KNN)

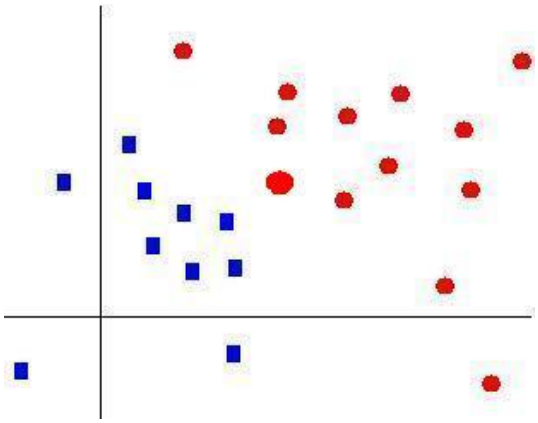
Sınıflandırma sürecinde k değeri, bakılacak eleman sayısını belirler. Algoritma bir eğitim verisi içerir. Yeni bir değer geldiğinde, mesafeler k değerine göre hesaplanır ve yeni değer bir kümeye eklenir. Mesafe hesaplama sürecinde Öklid uzaklığı ve Manhattan uzaklığı gibi mesafe hesaplama yöntemleri kullanılmaktadır. Algoritma beş adımdan oluşur.

1. Önce k değeri belirlenir.
2. Diğer nesnelerden hedef nesneye Öklid mesafeleri (Manhattan uzaklığı, Minkowski uzaklığıda olabilir) hesaplanır.
3. Mesafeler sıralanır ve minimum mesafeye bağlı olarak en yakın komşular bulunur.
4. En yakın komşu kategorileri birleştirilir.
5. En uygun komşu kategorisi seçilir.

Büyük bir eğitim setine sahip olmak ve k değerini uygun şekilde seçmek gerekmektedir [10]. Şekil 2.6'da gösterildiği gibi k değeri 3 için yeni bir eleman sınıflandırılması istenildiğinde eski sınıflandırılmış en yakın 3 eleman ele alınır. Bu elemanlar hangi sınıfa dahil ise yeni eleman da o sınıfa dahil edilir. Şekil 2.7'de ise yeni gelen elemanın dahil olduğu sınıf gösterilmiştir.



Şekil 2.6- K değeri 3 Seçilen KNN Sınıflandırma Başlangıcı

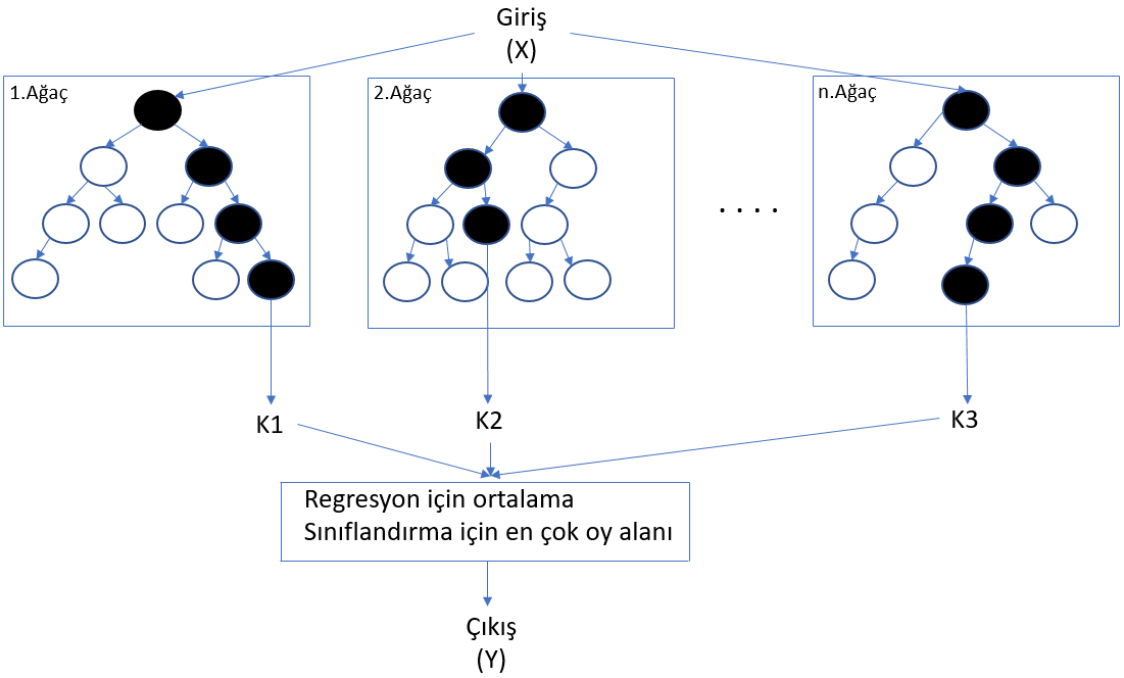


Şekil 2.7- K değeri 3 Seçilen KNN Sınıflandırma Sonucu

2.2.3.5. Rastgele Orman Algoritması (RF)

Rastgele Orman Algoritması, sınıflandırma işlemi sırasında çoklu DT üreterek sınıflandırma değerini artırmayı amaçlayan bir algoritmadır. Bireysel olarak bir karar ormanı oluşturmak için DT'ler oluşturulur. Buradaki DT, bağlı oldukları veri kümesinden rastgele seçilen alt kümelerdir [11]. Çok kısa sürede sonuçlanır.

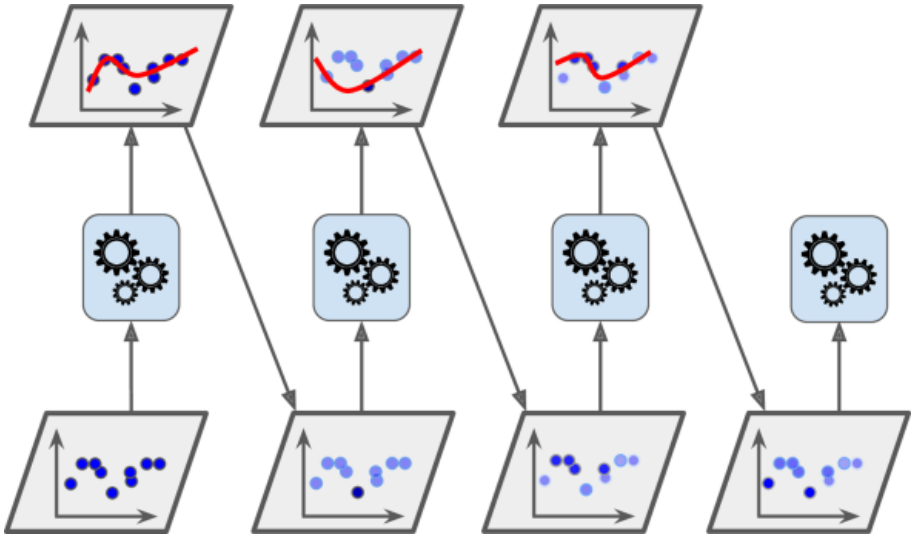
Geleneksel yöntemlerden biri olan karar ağaçlarının en büyük problemlerinden biri aşırı öğrenme yani veriyi ezberlemesidir. RF modeli bu problemi çözmek için hem veri setinden hem de öznitelik setinden rassal olarak 10'larca 100'lerce farklı alt setler seçer ve bunları eğitir. Bu yöntemle 100'lerce karar ağacı oluşturulur ve her bir karar ağacı bireysel olarak tahminde bulunur. Günün sonunda problem regresyonsa karar ağaçlarının tahminlerinin ortalamasını, problem sınıflandırmaysa tahminler arasında en çok oy alanı seçilir.



Şekil 2.8- n Tane DT İçeren RF Modeli

2.2.3.6. Ekstra Ağaç Algoritması (ET)

Bir RF'de bir ağacı büyütülmek istendiğinde, her düğümdeki rastgele özellik alt kümesini böldüğü düşünülür. En iyi eşikleri aramak yerine, her özellik için rastgele eşikler kullanarak ağaçları daha da rastgele hale getirmek mümkündür. Basitçe yüksek oranda randomize ağaç topluluğu olarak adlandırılır [12]. ET, birçok karar ağacından gelen tahminleri birleştiren bir topluluk makine öğrenme algoritmasıdır. ET algoritması, eğitim veri kümesinden çok sayıda eşi görülmemiş karar ağacı oluşturarak çalışır. Tahminler, regresyon durumunda karar ağaçlarının tahmini ortalanarak veya sınıflandırma durumunda çoğunluk oylaması kullanılarak yapılır. Eğitim veri kümesinin bir önyükleme örneğinden her karar ağacını geliştiren kümeleme ve rastgele ormanın aksine, Ekstra Ağaçlar algoritması her karar ağacına tüm eğitim veri kümesine uyar.



Şekil 2.9- ET Algoritması Ağaç Modeli

2.2.3.7. Artırma Algoritması (AC)

Bir tahmin edicinin; kendinden önce gelen tahmin ediciyi düzeltmesi için var olan yollardan biri, kendinden önce gelen tahmin edicinin eksik öğrendiği eğitim verilerine daha fazla dikkat etmesidir. Bu yöntem Adaboost algoritmasında kullanılır. Bir Adaboost sınıflandırıcısı oluşturabilmek için, ilk olarak sınıflandırıcı eğitim seti üzerinde eğitilir ve tahmin yapılır. Daha sonra, yanlış sınıflandırılan eğitim verilerinin "Göreli Ağırlığı" (Relative Weight) artırılır. İkinci sınıflandırıcı, bu arttırılmış ağırlıklar ile eğitilir ve tekrar tahmin yapılır. Ağırlıklar yine güncellenir ve bu şekilde devam edilir. Tüm tahmin ediciler eğitildiğinde tahminler, tahmin edicilerin doğruluk oranlarına göre ağırlıkları dikkate alınarak bagging ile yapılır [9].

2.2.3.8. Gradyan Artırma Algoritması (GBC)

AC ve GBC sınıflandırıcıları, makine öğrenme algoritmaları tarafından elde edilen doğruluğu artırmak için uygulanır [12]. GBC bir kayıp işlevine bağlıdır. Özel bir kayıp işlevi kullanılabilir ve birçok standartlaştırılmış kayıp işlevi GBC tarafından desteklenir, ancak kayıp işlevi farklılaştırılabilir olmaktadır. Sınıflandırma algoritmaları sık sık logaritmik kaybı kullanırken, regresyon algoritmaları kareli hataları kullanabilmektedir. GBC, artırma algoritması her eklenmede yeni bir kayıp işlevi türetmek zorunda değildir, aksine herhangi bir farklı tanımlanabilir kayıp işlevi sisteme uygulanabilmektedir. GBC'nin iki gerekli bölümü daha vardır: zayıf öğrenen ve katkı maddesi bileşenidir. GBC, karar ağaçlarını zayıf öğrenenler olarak kullanır. Regresyon ağaçları zayıf öğrenenler için kullanılır ve bu regresyon ağaçları gerçek değerler çıkar. Çıktılar gerçek değerler olduğundan, modele yeni öğrenenler eklendikçe, tahminlerdeki hataları düzeltmek için regresyon ağaçlarının çıktıları bir araya getirilebilmektedir.

GBC modelinin katkı bileşeni, ağaçların zaman içinde modele eklenmesinden gelir ve bu durumda var olan ağaçlar manipüle edilmezse, değerleri sabit kalır. Verilen parametreler arasındaki hatayı en aza indirmek için inişe benzer bir yordam kullanılır. Bu, hesaplanan kaybı alarak ve bu kaybı azaltmak için iniş gerçekleştirerek yapılır. Daha sonra, ağacın parametreleri artık kaybı azaltmak için değiştirilir. Yeni ağacın çıktısı daha sonra modelde kullanılan önceki ağaçların çıktısına eklenir. Bu işlem, daha önce belirtilen ağaç sayısına ulaşılıncaya veya kayıp belirli bir eşiğin altına düşürülene kadar yinelenir.

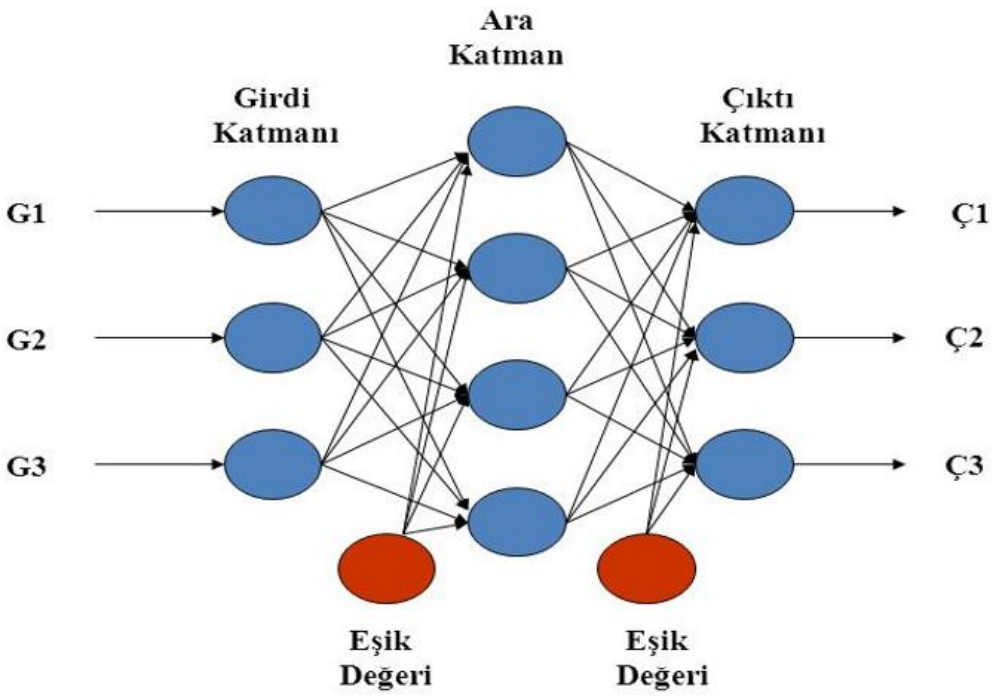
2.2.3.9. Bagging (Örnekleme) Algoritması (BA)

Mevcut bir eğitim setinden yeni eğitim setleri türeterek temel öğreneni yeniden eğitmeyi amaçlayan bir yöntemdir. Örnekleme, yerine koyarak yapılır. Eğitim seti, Bagging'teki n örnekten oluşan eğitim setinden n örnekle değiştirilerek rastgele seçim yapılarak üretilir. Seçilen her numune eğitim setine geri konur. Bazı örnekler yeni eğitim setine dahil edilmezken, diğerleri birden fazla yer almaktadır. Rastgele değiştirilerek seçilen eğitim setleri ile başarılı öğrenenleri ayırma sağlamak üzere eğitilir. Böylece kolektif başarı elde edilir.

2.2.3.10. Çok Katmanlı Algılayıcı Algoritması (MLP)

MLP derin bir yapay sinir ağıdır. Birden fazla algılayıcıdan oluşur. Sinyali almak için bir giriş katmanı, giriş hakkında bir karar veya tahmin yapan bir çıkış katmanı ve bu ikisi arasında MLP'nin gerçek hesaplama motoru olan keyfi sayıda gizli katmandan oluşur. Bir gizli katmana sahip MLP'ler, herhangi bir sürekli işleve yaklaşabilir. Çok katmanlı algılayıcılar genellikle denetimli öğrenme problemlerine uygulanır.

Denetimli bir öğrenme yaklaşımıdır ve yapay bir sinir ağı modelinden oluşur [13]. Bu yaklaşımdaki girdi haritası veri kümeleri uygun çıktı kümesiyle eşleşir. Tamamen, her bir düğümde birden çok düğüm katmanından ve sonrakinden oluşan yönlendirilmiş bir grafiğin MLP'sine bağlıdır. Her giriş düğümüne doğrusal olmayan aktivasyon fonksiyonuna sahip bir nöron denir. Gizli katmanın sigmoidal birimleri işlevler hakkında bilgi edinir. Eğitim amaçlı olarak MLP, backpropagation (geri yayılım) adı verilen bir teknik kullanır.



Şekil 2.10- MLP Çalışma Modeli

2.3. Literatür Araştırması

Bu bölümde tez çalışmasında yapılan literatür araştırmasına ve ilgili çalışmalara değinilmiştir.

Koru ve Liu, hatalı yazılım modülleri yazılım hatalarına neden olup geliştirme zamanını ve bakım maliyetlerini artırdığı ve bu sebepten dolayı müşteri memnuniyetini azalttığının çalışmasını sunmuşlardır. Etkili yazılım hata tahmin modelleri geliştiricilerin yazılıma daha çok odaklanmasına yardımcı olabilir, hatalı olabilecek modüllerde yazılım kalite güvence faaliyetlerini ve bulunan kaynakların daha verimli kullanımını arttırmak yazılım kalitesini artırmıştır. Bu modeller genellikle, hatalar ve değişiklikler gibi risk faktörleri ile ilişkilendirilmiş olan kaynak kodundan elde edilen statik ölçütleri kullanmışlardır. Önerdikleri tahmin modelini ise DT algoritması ile tasarlamışlardır [14].

Pelayo ve Dick, yazılımın karmaşıklığı ve bu karmaşıklık yüzünden, yazılım güvenilirliğini artırmanın son derece zor bir iş olduğunu söylemişlerdir. İşletim sırasında hangi modüllerin hata vereceğini tahmin etmeye odaklanan yazılım hatası tahmin problemini incelemişlerdir. Çok sayıda çalışma, makine öğrenimini yazılım hatası tahminine uyguladılar; ancak, hata tahmini veri kümelerindeki çarpıklık genellikle öğrenme algoritmalarına zarar vermiştir. Ortaya çıkan sınıflandırıcılar çoğu zaman hatalı azınlık sınıfını asla tahmin etmemiştir. Bu sorunun makine öğreniminde iyi bilindiğini ve genellikle dengesiz veri kümelerinden öğrenme olarak adlandırıldığını söylemişlerdir. Yazılım hatası tahmininde çok az ilgi gören dengesiz verileri

öğrenmek için yaygın olarak kullanılan bir teknik olan sınıf düzenini incelemişlerdir. Deneyim olarak ise azınlık sınıfından örnekleri aşırı örnekleme yöntemi olan SMOTE tekniğine odaklanmışlardır. DT'nin, G-ortalama sınıflama başarısını ölçtüler. Amaçları, SMOTE'nin hataya açık modüllerin tanınmasını iyileştirip iyileştiremeyeceğini ve ne pahasına olduğunu belirlemektir. Deneyleri, SMOTE yeniden örneklemeden sonra daha dengeli bir sınıflandırmaya sahip olduğumuzu göstermiştir. G-ortalama sınıflandırma doğruluğunda en az %23'lük bir iyileşme bulmuşlardır [15].

Menzies ve diğ. [16] Çalışmalarında hata tahmini yapabilmek için kartil grafiği üzerinde durmuşlardır. Kartil grafiği herhangi bir veri setini dört eşit parçaya ayıran üç değere denilmektedir. Veri madenciliği yöntemlerini statik kodlar ile yazılım hata tahmin yöntemi geliştirmişlerdir.

Lessmann ve diğ. [17] Yazılım hata tahmini alanında, hataya açık modüllerin zamanında tanımlanmasını sağlamak için kod özniteliklerinden tahmini sınıflandırma modelleri oluşturarak yazılım kalitesini ve yazılımın test verimliliğini artırmaya çalışmışlardır. Bu amaç için çeşitli sınıflandırma modelleri değerlendirilmiştir. Bununla birlikte bir sınıflandırıcının diğerine üstünlüğü ve genel olarak metrik tabanlı sınıflandırmanın yararlılığı ile ilgili tutarsız bulgular nedeniyle çalışmalar arasında yakınsamayı iyileştirmek ve deneysel sonuçlara olan güveni daha da artırmak için daha fazla araştırmaya ihtiyaç olduğunu söylemişlerdir. Yanlılık için üç potansiyel kaynağı göz önünde bulundurmuşlardır: sınıflandırıcıları bir veya az sayıda tescilli veri setinde karşılaştırmak, kavramsal olarak yazılım hatası tahmini ve çapraz çalışma karşılaştırmaları için uygun olmayan doğruluk göstergelerine güvenmek ve son olarak, istatistiksel test prosedürlerinin sınırlı kullanımı ampirik bulguları güvence altına alınmasıdır. Bu sorunları gidermek için, karşılaştırmalı yazılım hata tahmin deneyleri için bir önerilmiş ve NASA Metrics Data deposundan alınan 10 halka açık veri seti üzerinden 22 sınıflandırıcının büyük ölçekli deneysel karşılaştırmasında uygulamışlardır. Genel olarak, metrik tabanlı sınıflandırmanın yararlı olduğu görüşünü destekleyen cazip bir tahmine dayalı doğruluk derecesi gözlemlenmiştir. Bununla birlikte, sonuçları belirli sınıflandırma algoritmasının öneminin, ilk 17 sınıflandırıcı arasında önemli performans farklılıkları tespit edilemediği için daha önce varsayılandan daha az olabileceğini göstermektedir.

Çatal ve Diri, makalelerinde ölçümler, yöntemler ve veri kümelerine özel bir çalışma ile önceki yazılım hatası tahmin çalışmalarının sistematik bir incelemesini sağlamışlardır. İncelemelerinde, dergilerindeki 74 yazılım hatası tahmin kağıdını ve çeşitli konferans raporlarını kullanmışlardır. İnceleme sonuçlarına göre, herkese açık veri setlerinin kullanım yüzdesi önemli ölçüde arttı ve makine öğrenimi algoritmalarının kullanım yüzdesi 2005 yılından bu yana biraz

artmıştır. Çalışmalarını 2009 yılında yapmışlardır ve doğal olarak veriler bu yıllar arasını kapsamaktadır. Ayrıca, araştırma sonuçlarına göre yöntem düzeyinde ölçümler hata tahmini araştırma alanında hala en baskın ölçümlerdir ve makine öğrenimi algoritmaları hata tahmini için hala en popüler yöntemlerdir. Yazılım hatası tahmini alanında çalışan araştırmacılar, daha iyi hata tahmin edicileri oluşturmak için herkese açık veri kümelerini ve makine öğrenimi algoritmalarını kullanmaya devam etmelidir demişlerdir. Sınıf seviyesinin kullanım yüzdesi kabul edilebilir seviyelerin ötesindedir ve yazılım yaşam döngüsünün tasarım aşamasında hataları daha erken tahmin etmek için şu anda olduğundan çok daha fazla kullanılmasını belirtmişlerdir [18].

Gupta ve diğ. [19] Yazılım kalitesi, bir yazılım bileşeninin veya sistemin belirtilen gereksinimleri ve özellikleri karşılama derecesi olarak tanımlanır. Kalite, sistemlerin geliştirilmesinde en önemli gereksinimlerden biri haline gelmiştir. Önceki araştırmalar, yazılım ölçütlerine dayalı yazılım kalitesi modellerinin yararlı doğrulukta öngörü sağlayabileceğini göstermiştir. Bir yazılım kalitesi tahmin modeli, yazılım geliştirme ekibinin olası yazılım kusurlarını izlemesine ve tespit etmesine olanak tanır. Hata içermesi muhtemel olan yazılım alt sistemlerini (modüller, bileşenler, sınıflar veya dosyalar) tanımlarlar. Farklı araştırmacılar, yazılım ürünlerinin kalitesini ölçmeye yardımcı olmak için tasarım yazılımı kalite tahmin modellerini önermişlerdir ancak her kuruluş gereksinimlerine göre farklı kalite tahmin modelleri kullanmaktadır. Makalelerinde, farklı yazılım kalitesi tahmin modellerinin karşılaştırmalı bir değerlendirmesini sunmuşlardır ve performanslarını analiz etmişlerdir. Bu çalışma aynı zamanda kalite için istenen seviyeleri elde etmek için en iyi yaklaşımı seçmeye yardımcı olur ve ayrıca yazılım kalitesi tahmini için yumuşak bir hesaplama yaklaşımının kullanılmasını önerir. Yazılım kalitesi tahminindeki sorunları tartışmışlardır. Yazılım güvenilirliği, yazılım endüstrisinde gittikçe daha önemli hale geldiğini göstermişlerdir, yazılımın geliştirilmesinde erken dönemde hataları bulmak için çeşitli teknikler gerekmektedir. İstatistiksel modeller tamamen geçmiş verilere dayanmaktadır ve bu nedenle bu modellerde şeffaflık mevcut değildir. Bu sorunları mevcut modellerde araştırmışlardır. Mevcut kalite modelleme tekniklerini inceledikten sonra, tek bir kalite modelinin tüm gereksinimlerimizle baş edemeyeceği sonucuna varmışlardır. Ancak, birkaç yaklaşım bu gereksinimlerin bazılarını karşılamaktadır. Burada bir kalite tahmin modeli oluşturmak için NN, durum tabanlı kurallar, regresyon ağacı, kural tabanlı sistemler, çoklu lineer regresyon ve bulanık sistemlerini incelemişlerdir. Teknolojilerin bir arada kullanılması, bulanık ve kural tabanlı sistemler gibi ayrı ayrı ve özel olarak kullanılan her teknolojiye kıyasla etkili sorun çözme ile sonuçlanmıştır.

Hall ve diğ. [20] Yazılımın kodunda hataların nerede meydana gelebileceğine dair doğru tahmin, doğrudan test çabasına yardımcı olabilir, maliyetleri azaltabilir ve yazılım kalitesini

artırabilir fikri ile çalışmalarına başlamışlardır. Amaçları, modellerin kullanılan bağımsız değişkenlerin ve uygulanan modelleme tekniklerinin hata tahmin modellerinin performansını nasıl etkilediğidir. Yöntem olarak Ocak 2000'den Aralık 2010'a kadar yayınlanan 10 yıllık süreçteki çalışmaları araştırıp kendi çalışmalarını belirlemek için sistematik bir literatür taraması yapmışlardır. Ve 208 adet kaynak incelemişlerdir. Yeterli bağlamsal ve metodolojik bilgiyi bildiren 36 çalışmanın nicel ve nitel sonuçlarını kaydederek ve uyguladıkları kriterlere göre sentezlemişlerdir. Sonuç olarak iyi performans gösteren modeller, NB veya LR gibi basit modelleme tekniklerine dayanma eğilimindedir. İyi performans gösteren modeller tarafından bağımsız değişkenlerin kombinasyonları kullanılmıştır. C4.5 ise NB ve LR algoritmalarına göre daha kötü sonuç vermiştir.

Wang ve Yao, yazılım testini kolaylaştırmak ve yazılım test maliyetlerinden tasarruf etmek için çıktıkları yolda yazılım modüllerindeki hataları tahmin etmeyi çok çeşitli makine öğrenimi yöntemleri ile çalışmışlardır. Ne yazık ki, bu tür verilerin dengesiz doğası, böyle bir görevin öğrenme zorluğunu artırmaktadır görüşünü öne sürmüşlerdir. Sınıf dengesizliği öğrenimi, dengesiz dağılımlarla sınıflandırma problemlerinin üstesinden gelmede uzmanlaşmıştır; bu hata tahmini için yardımcı olabilir, ancak şimdiye kadar derinlemesine araştırılmamıştır. Çalışmalarında, sınıf dengesizliği öğrenme yöntemlerinin daha iyi çözümler bulmak için yazılım hatası tahminine nasıl ve ne şekilde fayda sağlayabileceği konusunu incelemişlerdir. Yeniden örnekleme teknikleri, eşik hareketleri ve topluluk algoritmaları dahil olmak üzere farklı sınıf dengesizliği öğrenme yöntemlerini araştırmışlardır. İnceledikleri yöntemler arasında AdaBoost.NC, denge, G-ortalama ve AUC dahil olmak üzere ölçümler açısından en iyi genel performansı göstermektedir. Algoritmanın performansını daha da iyileştirmek ve yazılım hatası tahmininde kullanımını kolaylaştırmak için AdaBoost.NC'nin, eğitim sırasında parametresini otomatik olarak ayarlayan dinamik bir sürümünü önermişlerdir. Herhangi bir parametreyi önceden tanımlamaya gerek kalmadan, orijinal AdaBoost.NC'den daha etkili ve verimli olduğu gösterilmiştir. Çıkan sonuçları RF ve NB algoritmaları ile karşılaştırmışlardır [21].

Paramshetti ve Phalke, yazılım hata tahmini yazılım kalitesinin iyileştirilmesinde önemli bir rol oynar ve yazılım testi için zaman ve maliyetin azaltılmasına yardımcı olur mantığı ile çalışmalarını ele almışlardır. Makine öğrenimi, yeni verilere maruz kaldığında büyümeyi ve değişmeyi kendilerine öğretebilen bilgisayar programlarının geliştirilmesine odaklanır. Bir makinenin performansı önceki sonuçlara göre iyileştirme yeteneğinden anlaşılabilmektedir. Makine öğrenimi, insan öğreniminin verimliliğini artırır, insanlar tarafından bilinmeyen yeni şeyler veya yapılar keşfeder ve bir belgede önemli bilgiler bulur. Bu amaçla, gereksiz hatalı verileri veri kümesinden çıkarmak için farklı makine öğrenimi teknikleri kullanılır. Yazılım hatası

tahmini, bir yazılım projesi planlanırken oldukça önemli bir yetenek olarak görülür ve bu karmaşık sorunu yazılım ölçümleri ve hata veri setlerini kullanarak çözmek için çok daha fazla çaba gerekir. Metrikler, sayısal değer arasındaki ilişkidir ve yazılıma uygulandığı için kusurları tahmin etmek için kullanılır. Amaçları yazılım hatalarını tahmin etmek için mevcut teknikleri anlamaktır. Makaleleri, yazılım hatası tahmini için çeşitli makine öğrenimi tekniklerinin bir sonuçlarını sunar. Sonuçlardan, yazılım hatasının gerçekten de yazılım mühendisliğinde önemli bir sorun olduğu gözlemlenebilir. Farklı makine öğrenme tekniklerini kullanarak yazılım hatası modülü tahmini, yazılım geliştirme sürecinin kalitesini artırmaktır. Yazılım yöneticileri bu tekniği kullanarak kaynakları etkin bir şekilde tahsis eder. Yazılımdaki hataları tahmin etmek için NN, SVM, DT, birliktelik kuralları ve kümeleme ile ilgili avantaj ve sınırlamaları liste halinde sundular. Karışık tasarımına rağmen DT diğer algoritmalara göre daha iyi sonuç vermiştir [22].

Aleem ve diğ. [9] Göre makine öğrenimi yaklaşımları, daha az bilgiye sahip problemleri çözmede iyidir. Çoğu durumda, yazılım alanındaki sorunlar, çeşitli koşullara bağlı olan ve buna göre değişen bir öğrenme süreci olarak karakterize edilir. Tahmine dayalı bir model, makine öğrenimi yaklaşımları kullanılarak oluşturulur ve bunları hatalı olan, hatalı olmayan modüller olarak sınıflandırır. Makine öğrenimi teknikleri, geliştiricilerin sınıflandırmadan sonra yararlı bilgileri almalarına yardımcı olur ve verileri farklı perspektiflerden analiz etmelerini sağlar. Makine öğrenimi tekniklerinin yazılım hatası tahmini açısından yararlı olduğu kanıtlanmıştır. Çalışmalarında, yazılım modüllerinin halka açık veri setlerini kullanılmıştır ve yazılım hata tahmini için farklı makine öğrenimi tekniklerinin karşılaştırmalı bir performans analizini sağlamışlardır. Sonuçların çoğu öğrenme yöntemleri için yazılım hata veri kümelerinde iyi performans gösterdi. Doğruluk, F ölçümü tablolar ile farklı algoritmalar için çeşitli veri kümelerinde gösterilmiştir. Elde ettikleri sonuçlara göre yazılım hata sınıflandırması için NB algoritması %83,47 ile çeşitli veri kümelerinin ortalama doğruluğunu göstermiştir. Doğruluk sonuçlarının %95'in üzerinde olduğu MC1, PC2 ve PC5 veri kümelerinde gerçekten iyi performans göstermiştir. En kötü performans, doğruluğun %50'den az olduğu PC3 veri setinde görülebilmektedir. MLP algoritması ayrıca MC1 ve PC2 veri setinde iyi performans gösterdi ve çeşitli veri kümelerinde %89,14 genel doğruluk elde etti. SVM ve BA, diğer makine öğrenimi yöntemlerine kıyasla gerçekten iyi performans gösterdi ve yaklaşık %89 genel doğruluk elde etti. Adaboost %88,59 doğruluk elde etti, BA %89,39, DT %88,47 civarında doğruluk elde etti, RF %89.08. Seçilen tüm veri kümelerinde MLP, SVM ve BA performansı, diğer makine öğrenimi yöntemlerine kıyasla iyiydi. Çeşitli veri kümelerinde 0.82 olan KNN yöntemiyle elde edilen en kötü F ölçümüdür. Yazılım yaşam döngüsünün daha önceki bir aşamasında yazılım hatalarının belirlenmesi, yazılım kalite güvence önlemlerinin yönlendirilmesine yardımcı olur ve ayrıca

yazılımın yönetim sürecini iyileştirir. Etkili bir hatanın tahmini, tamamen iyi bir tahmin modeline bağlıdır. Çalışmaları, bir hatanın tahmini için kullanılabilecek farklı makine öğrenimi yöntemlerini kapsamaktadır. Çeşitli yazılım veri setlerinde farklı algoritmaların performansı analiz edilmiştir. Çoğunlukla SVM, MLP ve torbalama teknikleri, hatanın veri kümelerinde iyi performans göstermiştir. Hatanın tahminine yönelik uygun yöntemi seçmek için uzmanlar, veri kümelerinin türü, sorun alanı, veri kümelerindeki belirsizlik veya projenin doğası gibi çeşitli faktörleri göz önünde bulundurmalıdır. Daha doğru sonuçlar elde etmek için birden fazla teknik birleştirilebilir kanısına varmışlardır.

Magal ve Jacob, çalışmalarında yazılım hatası tahmini için özellik seçimi tabanlı bir RF modeli önermiştir. Korelasyona dayalı özellik alt kümesi seçim tekniği, yazılım modüllerindeki hataları tahmin etmeye yardımcı olan önemli özellikleri seçmek için kullanılmıştır. Yeni bir özellik alt kümesi kullanılarak geliştirilen sınıflandırıcının performansında, özellik kümesinin tamamı üzerine kurulu sınıflandırıcıyla karşılaştırıldığında önemli bir fark vardı. Bu çalışma, hatalı yazılım modülleri için önerilen modelin tahmin performansını değerlendirmiş ve ayrıca dört PROMISE veri kümesini kullanarak beş istatistiksel ve makine öğrenimi yaklaşımına karşı karşılaştırmalı bir analiz gerçekleştirmiştir. Deneysel sonuçlar, önerilen yaklaşımın öngörü yeteneğinin diğer yaklaşımlarla daha iyi veya en azından karşılaştırılabilir olduğunu ortaya koymuştur. Araştırma, korelasyona dayalı özellik alt kümesi seçimi (CF alt kümesi) tabanlı RF yaklaşımının hatalı yazılım modüllerini tahmin etmedeki etkinliğini ortaya koymakta ve önerilen modelin, önemli özniteliklere dayalı olarak hatalı yazılımı tahmin etmede yararlı olabileceğini önermektedir. Yazılım geliştirmenin doğruluğunu ve kalitesini iyileştirmek için, yazılım geliştirmede toplanan çok sayıda hata verisini analiz etmek ve tahmin etmek için veri madenciliği teknikleri kullanılır. Doğruluğu artırmak için, daha iyi doğruluk sağlayan RF algoritmasının bir parçası olarak uygun özellik seçim algoritması dahil edilerek mevcut algoritma iyileştirilir. PC1 veri setinin sonucundan elde edilen geliştirilmiş RF performansı %94.545, PC2 veri seti %98.561, PC3 veri seti %89.676 ve PC4 veri seti için %90.625'tir. Gelecekteki geliştirmelerinin ise bir sistem olarak geliştirilmiş RF tasarımı ve bunu sağlık güvenlik tehditleri gibi diğer araştırma alanlarındaki potansiyel hedefleri tahmin etmek için kullanmak olacağını belirtmişlerdir [23].

Prasad ve diğ. [8] Yazılım kalitesi, yazılım ürünlerinin arzu edilen özelliklerini tanımlayan bir çalışma ve uygulama alanıdır. Performans, herhangi bir kusur olmaksızın mükemmel olmalıdır. Yazılım kalite ölçütleri, ürün, süreç ve projenin kalite yönlerine odaklanan yazılım ölçütlerinin bir alt kümesidir. Yazılım hata tahmin modeli, kusurların erken tespitine yardımcı olur ve bunların verimli bir şekilde ortadan kaldırılmasına ve çeşitli ölçütlere dayalı kaliteli bir yazılım sistemi üretilmesine katkıda bulunur düşünceleri ile yazdıkları makale de temel amaç olarak

geliştiricilerin veri madenciliği tekniklerini kullanarak mevcut yazılım ölçütlerine dayalı kusurları belirlemelerine ve böylece yazılım kalitesini iyileştirmelerine yardımcı olmayı edinmişlerdir. Çalışmalarında, literatürdeki yazılım ölçütleri kullanılarak yazılım hata tahmini için kullanılan çeşitli sınıflandırma teknikleri gözden geçirilmiştir.

Batista ve diğ. [24] Mevcut öğrenme sistemleri ile elde edilen performansı etkileyebilecek birkaç husus vardır. Bu yönlerden birinin, bir sınıfa ait eğitim verilerindeki örneklerin diğer sınıftaki örneklerden çok daha fazla olduğu bir sınıf dengesizliği ile ilgili olduğu bildirilmiştir. Az ama önemli bir olayı anlatan gerçek dünya verilerinde bulunan bu durumda, öğrenme sistemi azınlık sınıfı ile ilgili kavramı öğrenmekte güçlükler yaşayabilir. Çalışmalarında, 13 UC Irvine veri setinde sınıf dengesizliği sorununu ele almak için üçü yazarlar tarafından önerilen 10 yöntemi içeren geniş bir deneysel değerlendirme gerçekleştirmişlerdir. Deneyleri, sınıf dengesizliğinin sistematik olarak öğrenme sistemlerinin performansını engellemediğine dair kanıt sağlamaktadır. Aslında sorun, sınıf çakışması gibi diğer karmaşık faktörlerin varlığında çok az azınlık sınıfı örneği olan öğrenmeyle ilgili görünmektedir. Önerilen yöntemlerinden ikisi, daha iyi tanımlanmış sınıf kümeleri üretmek için bilinen bir aşırı örnekleme yöntemini veri temizleme yöntemleriyle birleştirerek bu koşullarla doğrudan ilgilenir. Karşılaştırmalı deneyleri, genel olarak aşırı örnekleme yöntemlerinin, AUC altındaki alanı dikkate alarak, yetersiz örnekleme yöntemlerinden daha doğru sonuçlar verdiğini göstermektedir. Bu sonuç, daha önce literatürde yayınlanan sonuçlarla çelişiyor gibi görünmüştür. Önerilen yöntemlerinden ikisi, Smote + Tomek ve Smote + ENN, az sayıda olumlu örnekle veri setleri için çok iyi sonuçlar vermiştir. Dahası, çok basit bir aşırı örnekleme yöntemi olan rastgele aşırı örnekleme, daha karmaşık aşırı örnekleme yöntemlerine karşı çok rekabetçidir. Aşırı örnekleme yöntemleri çok iyi performans sonuçları sağladığından, aşırı örneklenmiş verilerden kaynaklanan karar ağaçlarının söz dizimsel karmaşıklığını da ölçmüşlerdir. Sonuçlar bu ağaçların genellikle orijinal verilerden elde edilenlerden daha karmaşık olduğunu göstermektedir. Rastgele aşırı örnekleme, araştırılan aşırı örnekleme yöntemleriyle karşılaştırıldığında, genellikle indüklenen kuralların ortalama sayısında en küçük artışı ve Smote + ENN, kural başına ortalama koşul sayısındaki en küçük artışı sağlamıştır.

Çatal ve diğ. [18] 2009 yılında yayınlanan çalışmalarında yazılımdaki test süreçlerini daha iyi bir seviyeye getirmek için yapay bağışıklık keşfetme sistemlerini temel alan bir hata tahmin etme modeli önermişlerdir. Bu model ile Halstead, McCabe yöntem-seviye ve diğer bazı sınıf-seviye metriklerini kullandılar elde edilen sonuçlarda yapılan çalışmanın C4.5 algoritmasına göre daha yüksek performanslı olduğu ifade edilmiştir.

Tomar ve Agarwal, yazılım hatası belirleyicileri, yazılım ürünlerinin yüksek kalitesini etkin

bir şekilde sürdürmek için kullanışlıdır. Hatalı yazılım modüllerinin erken tahmini, yazılım geliştiricilerin mevcut kaynakları yüksek kaliteli yazılım ürünleri sunmak için tahsis etmelerine yardımcı olabilir. Yazılım hata tahmin sisteminin amacı, genel performansı etkilemeden mümkün olduğunca çok sayıda hatalı yazılım modülü bulmaktır. Bir yazılım hata tahmin edicisinin öğrenme süreci, yazılım modüllerinin hatalı ve kusurlu olmayan sınıflar arasında dengesiz dağılımı nedeniyle zordur. Hatalı yazılım modüllerinin yanlış sınıflandırma maliyeti, genellikle hatalı olmayanların yanlış sınıflandırılmasından çok daha yüksek maliyete neden olur. Bu nedenle, yanlış sınıflandırma maliyeti konusunu da göz önünde bulundurarak, Ağırlıklı En Küçük Kareler İkiz Destek Vektör Makinesi (WLSTSVM) kullanarak bir yazılım hata tahmin sistemi geliştirmişlerdir. Bu sistem, hatalı sınıfların veri örneklerine daha yüksek yanlış sınıflandırma maliyetleri ve kusurlu olmayan sınıfların veri örneklerine göre daha düşük olarak maliyet atar. Sekiz yazılım hatası tahmin veri seti üzerinde yapılan deneyler, önerilen hata tahmin sisteminin geçerliliğini kanıtlamıştır. Sonuçların önemi, parametrik olmayan Wilcoxon işaretli sıra testi kullanılarak yapılan istatistiksel analiz yoluyla test edilmiştir [25].

Çatal ve Diri, yazılım kalite mühendisliği, test etme, resmi doğrulama, inceleme, hata toleransı ve yazılım hatası tahmini gibi çeşitli kalite güvence faaliyetlerinden oluşur. Şimdiye kadar birçok araştırmacı, makine öğrenimi ve istatistiksel teknikleri kullanarak birkaç hata tahmin modeli geliştirdi ve doğruladı. Modellerin performansını iyileştirmek için farklı türde yazılım ölçütleri ve çeşitli özellik azaltma teknikleri kullanılmıştır. Ancak, bu çalışmalar veri kümesi boyutunun, ölçüm setinin ve özellik seçim tekniklerinin yazılım hatası tahmini için etkisini araştırmadı. Bu çalışma, RF gibi makine öğrenmesine dayanan yüksek performanslı hata tahmin edicilerine ve Yapay Bağışıklık Sistemleri adı verilen yeni bir hesaplamalı zekâ yaklaşımına dayanan algoritmalara odaklanmıştır. Tahmine dayalı modelleri tekrarlanabilir, çürütülebilir ve doğrulanabilir hale getirmek için PROMISE havuzundaki halka açık NASA veri kümelerini kullandık. Araştırma soruları, veri seti boyutu, ölçüm seti ve özellik seçim tekniklerinin etkilerine dayanıyordu. Bu sorulara cevap verebilmek için yedi test grubu oluşturulmuştur. Ek olarak, beş genel NASA veri kümesinin her biri için dokuz sınıflandırıcı incelenmiştir. Bu çalışmaya göre, RF, büyük veri kümeleri için en iyi tahmin performansını sağlarken, NB, AUC değerlendirme parametresi açısından küçük veri kümeleri için en iyi tahmin algoritmasıdır. Yapay Bağışıklık Tanıma Sistemleri (AIRS2Parallel) algoritmasının paralel uygulaması, yöntem düzeyindeki ölçümler kullanıldığında en iyi yapay bağışıklık sistemleri paradigmasına dayalı algoritmadır [26].

Karataş ve diğ. [27] Ağlara yapılan saldırıları tespit edip bunları önleyebilmek için kurulabilecek sistemde siber güvenliğin çok kritik bir rol oynadığını söylemişlerdir.

Çalışmalarında, en yeni eğitim ve sınıflandırma tekniklerinden biri derin öğrenme ortaya çıkmıştır. Bu nedenle, çalışmalarında izinsiz giriş tespiti ve derin öğrenme algoritmalarının çeşitli yönlerine genel bir bakış ile derin öğrenme tabanlı saldırı tespit sistemlerinin kısa bir araştırmasının yapılması amaçlanmışlardır. Ayrıca, bu çalışmada özellikleri ve eksiklikleriyle halka açık bazı veri kümeleri hakkında listeler ve ayrıntılar vermişlerdir.

Baykal ve diğ. [28] Büyük veri çağında derin öğrenme kavramı ile ilgili tespitlerde bulunmuşlardır. Doğru sonuçlar elde etmelerine rağmen veriyi eğitme sürecinin zaman aldığını göstermişlerdir. Bu nedenle bazı hızlandırıcı teknolojiler kullanma veya programlara ihtiyaç olduğunu söylemişlerdir. GPU ve CPU performanslarını karşılaştırmışlardır.

Tian ve diğ. [29] Bilgi teknolojinin gelişmesi ile yazılımın toplumun her alanına etki ettiğini göz önünde bulundurarak, yazılım kalitesindeki sorunlara dikkat çekmişlerdir. Yazılım kalitesindeki sorunların büyük kayıplara yol açtığını savunarak, makine öğrenmesi teknolojilerinin kullanılması gerektiğini savunmuşlardır. Makalelerinde farklı makine öğrenmesi algoritmalarını kullanarak avantajlarını ve dezavantajlarına değinmişlerdir.

Li ve diğ. [30] CoForest ve ACoForest dahil olmak üzere örneklemeye dayalı yarı denetimli kusur tahmin algoritmaları önermişlerdir. CoForest'e dayalı ve aktif öğrenmeyi kullanan ACoForest, en çok ihtiyaç duyulan örnekleri otomatik olarak seçer, böylece yarı denetimli öğrenmenin performansını ve verimliliğini artırır. Araştırma sonuçları, önerilen CoForcst ve ACoForest'in NB, LR ve DT gibi geleneksel öğrenme algoritmalarından daha iyi etkiler elde ettiğini göstermektedir.

2.4. Veri Seti

Veri, en kısa tanımı ile işlenmiş bilgilerdir. Veri; ölçüm, deney, gözlem, sayım veya araştırma yolu gibi yöntemler ile elde edilebilir. Örneğin, internet üzerinde yapılan her hareket birer veridir ve kaydı tutulmaktadır. Günümüzde, veri diye adlandırdığımız işlenmiş bilgilerin çokluğu git gide büyük verileri oluşturmuştur. Bu büyük verilere “big data” ismi verilir ve bu başlık altında birçok çalışma yapılmaktadır.

2.4.1. Yararlanılan Veri Setleri

Veri setleri, bir konu ile ilgili ham verilerin saklandığı yapılara verilen isimdir. Herkesin kolayca erişebildiği veri setleri her geçen gün artmaktadır. Buna yazılım hata tahmini için oluşturulan veri setleri de dahildir. Bunlardan biri, uzay araştırmacısı olan NASA’nın sahip olduğu halka açık olarak kullanılabilen PROMISE veri setidir. Bu çalışmada PROMISE veri setinden CM1, KC1, KC2, JM1 ve PC1 veri kümeleri kullanılmıştır [31].

Bu veri kümelerinin özellikleri Tablo 2.1'de gösterilmiştir. Kullanılan veri kümelerinin bazı ölçüm değerleri ve değişkenleri mevcuttur. Veri kümesinin her kaydı, ilgili yazılım modülü ile alakalı bilinen veya geriye doğru bildirilen hatalı veya hatasız olduğunu bildiren sınıf etiketi içerir.

Hata olması, ilgili yazılım modülünde veya diğer bağlantılı modüllerde değişiklik veya güncelleme yapılması gerektiğini ifade etmektedir [14]. Tüm veri kümelerinin kaynak tarihleri 2 Aralık 2004'tür ve donörü Tim Menzies'dir. Hiçbir veri kümesinin eksik özniteliği yoktur [31].

CM1, JM1 ve PC1 veri seti C yazılım dili ile yazılmış bir NASA uzay aracı iken, KC1 ve KC2 ise C++ yazılım dili ile geliştirilmiştir.

En fazla modül sayısı 10885 modül ile JM1 iken, en az modül ise 498 modül bulunduran CM1 veri setindedir.

Tüm veri setlerinde 22 adet özellik bulunmaktadır. 21 özellik yazılımdaki metrikleri 22. özellik ise o metriklere bağlı sonucu göstermektedir.

En çok hatalı veri sayısı JM1 veri setindedir ve sayısı 2106'dır. En az hata sayısı ise 49 hata ile CM1 veri setinde bulunmaktadır.

En fazla hatasız veri sayısı JM1 veri setindedir ve sayısı 8779'dur. En az hatasız veri sayısı ise 415 ile KC2 veri setinde bulunmaktadır.

Tüm bu istatistiklere göre ise en fazla hata oranı %20,5 olan KC2 veri setindedir. En az hata oranı ise %6,94 oran ile PC1 veri setindedir.

Tablo 2.1-Veri Seti Bilgisi

ADI	DİLİ	AÇIKLAMA	MODÜL SAYISI	ÖZELLİK SAYISI	HATALI VERİ SAYISI	HATASIZ VERİ SAYISI	HATA ORANI (%)
CM1	C	NASA uzay aracı aleti	498	22	49	449	9.83
KC1	C++	Zemin verilerinin alınması ve işlenmesi için depolama yönetimi	2109	22	326	1783	15.45
KC2	C++	Zemin verilerinin alınması ve işlenmesi için depolama yönetimi	522	22	105	415	20.50
JM1	C	Gerçek zamanlı kestirim sistemi	10885	22	2106	8779	19.35
PC1	C	Dünya yörüngesindeki uydu için uçuş yazılımı	1109	22	77	1032	6.94

2.4.2. Yararlanılan Veri Setlerindeki Metrikler

Tüm metrikler açıklamalarıyla birlikte Tablo 2.2' de gösterilmiştir. Çalışmada kullanılan veri kümesi McCabe metrikleri, Halstead metrikleri ve diğer metriklerden oluşmaktadır. Her ne kadar yöntem düzeyinde metrikler Halstead ve McCabe tarafından 1970'lerde önerilmiş olsada, günümüzde hala yaygın olarak kullanılmaktadır.

Bu çalışmada kullanılan; 4 adet McCabe metriği yöntem düzeyine karşılık gelmektedir. Yöntem düzeyinde metrikler, programlama kavramlarına odaklanan metriklerdir. Bu metriklerin nesne tabanlı kaynak kodundan toplanması kolaydır ve hataya eğilimli modüllerin sistem testi veya saha testleri sırasında hata olması muhtemeldir [26].

Kullanılan 11 adet Halstead metriği ise, çoğunlukla programla ilgili sayısal değerlere sahip metriklerdir. Bu metriklere tüm yazılımlardan kolayca erişilebilir. Hatalar metriği bir Boolean değeridir. Diğer tüm metrikler sayısaldır [31].

Siklomatik karmaşıklık, bir programın karmaşıklığını belirtmek için kullanılan bir yazılım ölçüsüdür. Bir programın kaynak kodu aracılığıyla doğrusal olarak bağımsız yolların sayısının nicel bir ölçüsüdür. 1976 yılında Thomas J. McCabe, Sr. tarafından geliştirilmiştir [32].

Tablo 2.2-Veri Seti Metrik Bilgisi

METRİK	AÇIKLAMASI
loc	McCabe'nin satır kod sayısı
v(g)	McCabe "siklomatik karmaşıklık"
ev(g)	McCabe "temel karmaşıklık"
iv(g)	McCabe "tasarım karmaşıklığı"
n	Halstead toplam operatör + işlenenler
v	Halstead "boyut"
l	Halstead "program uzunluğu"
d	Halstead "zorluk"
i	Halstead "bilgi"
e	Halstead "efor"
b	Halstead “tespit edilen hatalar”
t	Halstead'in zaman tahmincisi
lOCode	Halstead'in satır sayısı
lOComment	Halstead'in yorum satırı sayısı
lOBlank	Halstead'in boş satır sayısı
lOCodeAndComment	Kod satırları ve yorumlar
uniq_Op	Benzersiz operatörler
uniq_Opnd	Eşsiz işlenenler

total_Op	Toplam operatörler
total_Opnd	Toplam işlenenler
branchCount	Akış grafiği
defects	Modülde bildirilen bir veya daha fazla hata var / yok

loc - Kod satırları: McCabe'nin satır sayma kurallarına göre ölçülür.

v(g) - Siklomatik karmaşıklık: Doğrusal olarak bağımsız yolların sayısını ölçer. Kümedeki hiçbir yol, bir programın akış grafiği aracılığıyla kümedeki diğer yolların doğrusal bir birleşimi değilse, bir dizi yolun doğrusal olarak bağımsız olduğu söylenir. Akış grafiği, her düğümün bir program deyimine karşılık geldiği ve her yay, bir deyimden diğerine kontrol akışını gösterdiği yönlendirilmiş bir grafiktir. "v (g) = e - n + 2" formülü ile hesaplanır, burada "g" bir programın akış grafiğidir, "e" akış grafiğindeki yay sayısıdır ve "n" ise akış grafiğindeki düğümlerdir. Standart McCabe kuralları ("v(g)" > 10), hataya açık modülü tanımlamak için kullanılır.

ev(g) - Temel karmaşıklık: Bir akış grafiğinin, D-yapılı primler olan tüm g alt akış grafiklerinin ayrıştırılmasıyla azaltılabileceği ölçektir. Bu tür D-yapılı asallara bazen uygun tek girişli tek çıkışlı alt akış grafikleri olarak da atıfta bulunulur. "ev (g) = v (g) - m" formülü kullanılarak hesaplanır. Burada “m”, "g" nin D-yapılı asal sayıları olan alt akış grafiklerinin sayısıdır.

iv(g) - Tasarım karmaşıklığı: Bir modülün azaltılmış akış grafiğinin döngüsel karmaşıklığıdır. Bir modülün akış grafiği "g", tasarım modülleri arasındaki karşılıklı ilişkiyi etkilemeyen herhangi bir karmaşıklığı ortadan kaldırmak için küçültülür. McCabe'ye göre, bu karmaşıklık ölçümü, modelleri çağıran modülleri hemen alt modüllerine yansıtıyor.

3. YÖNTEM

Yapılan çalışma birçok basamaktan oluşmaktadır. Bunlardan ilki veri toplanması adımıdır. Veriler daha önceki bölümlerde de anlatıldığı üzere PROMISE açık kaynaklı veri seti kütüphanesinden bir araya getirilmiştir. Veri toplanırken verinin tutarlı olması, makine öğrenmesi yöntemleri ile yazılım hata tahmini amacına uygun hizmet edebilecek olması göz önünde bulundurmuştur.

İkinci adım olarak toplanan bu veriler bir araya getirildikten sonra standart normalizasyon işlemine tabii tutulmuştur.

Normalize edilmiş olan veriler Çapraz Doğrulama yöntemi kullanılarak makine öğrenmesi algoritmaları tarafından sırasıyla eğitilmiştir. Daha önceki bölümlerde bahsedildiği gibi 10 adet makine öğrenmesi algoritması kullanılmıştır. Çıkan sonuçlar daha sonra test için tahsis edilen veriler ile karşılaştırılarak Karışıklık Matrisine yansıtılmıştır.

Tüm bu hesaplamalar bittikten sonra elde edilen değerler çıktıya yazılır ve sistem kendisine şu soruyu sorar: "Temel Bileşen Analizi kullanıldı mı?". Yanıt hayır ise sistem tüm veri kümeleri için normalleştirme adımından itibaren aynı işlemleri gerçekleştirir. Yanıt evet ise sistem sonuçların karşılaştırma adımına geçer.

3.1. Normalizasyon İşlemi

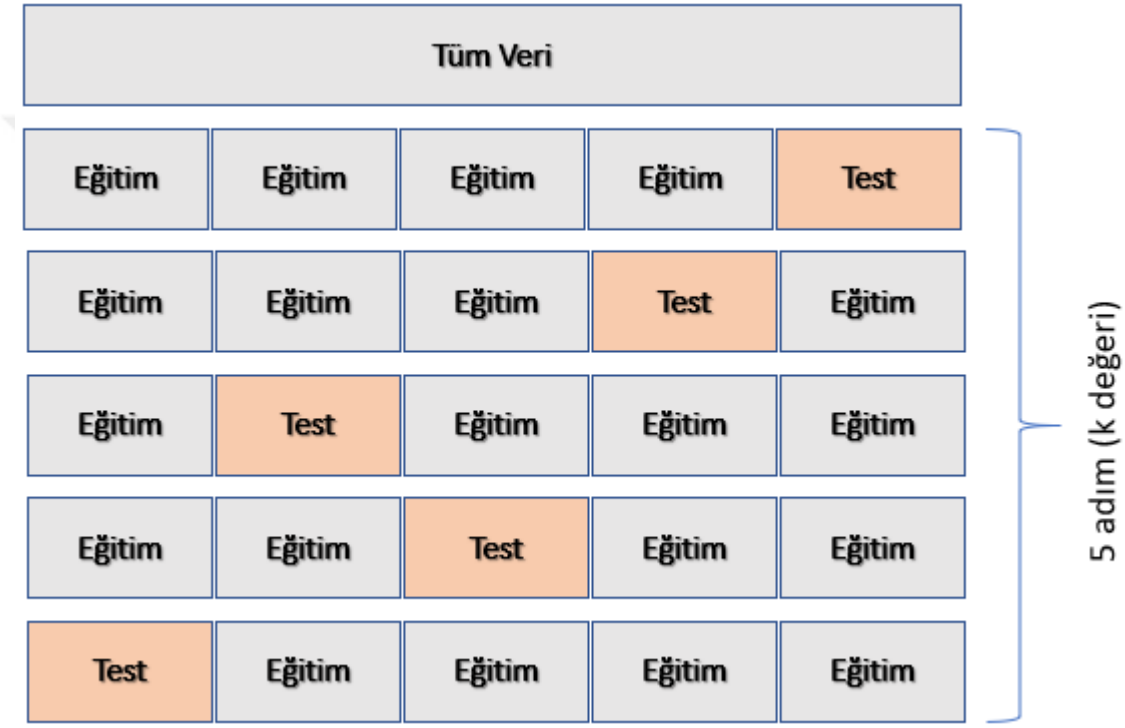
Normalizasyon yapılmasının amacı; veri setlerindeki parametreler arasında veri bütünlüğünü sağlamak, gereksiz veri tekrarını önleyerek verilerdeki bozulmaları önlemek, uygulama yöntemi değiştirmiş olunsa bile veri setindeki bütünlüğü korumak, performansı artırmak yani veri tekrarını en aza indirmektir. Bu çalışmada kullanılan normalizasyon işlemi bu amaçlara tam anlamıyla hizmet etmiştir. Çalışmada yapılan normalize işlemi denklem 3.1'de gösterilmiştir. Formüldeki X değeri mevcut parametrenin değeridir. X_{yeni} ise yeni hesaplanacak değerdir. Yani normalize edilmiş değer de denilebilir. X_{min} değeri bu parametre başlığı altındaki en küçük değerdir. X_{max} değeri ise bu parametre başlığı altındaki en büyük değerdir. X'in mevcut değerinin minimum değer ile farkının, maksimum değer ile minimum değer arasındaki farka oranı yeni X değerini vermektedir.

$$X_{yeni} = (X - X_{min}) / (X_{max} - X_{min}) \quad (3.1)$$

3.2. Çapraz Doğrulama (CV)

CV, makine öğrenimi modellerinin başarılarını değerlendirmek için kullanılan bir yöntemdir. Bu yöntemde, veri seti eğitim ve test setine ayrılmıştır ve bu işlem için seçilen yöntem

modelin başarısını önemli ölçüde etkilemektedir [17]. Bu çalışmada CV kullanımının nedeni; CV sayesinde, modelin doğruluğu için çok gerekli olan modelin tüm koşullarında eğitilir. Verilerin daha iyi kullanılmasına yardımcı olur ve algoritma performansı hakkında daha fazla bilgi verir. Bu yöntem kullanılmadan önce bir k değeri seçilir. Bu çalışmada, k değeri 5 seçilmiştir. Bunun anlamı veri kümesi 5 eşit parçaya bölünmüştür; test verileri her seferinde test edilen verinin test içinde Şekil 3.1'de gösterildiği gibi kullanıldığından emin olmak için bir adım kaydırır. Sonuçta her işlemden ayrı olarak üretilir. Bu sonuçların aritmetik ortalaması alınarak genel bir hata veya başarı metriği sonucu elde edilmiştir.



Şekil 3.1- Kullanılan CV Modeli

3.3. Karışıklık Matrisi (CM)

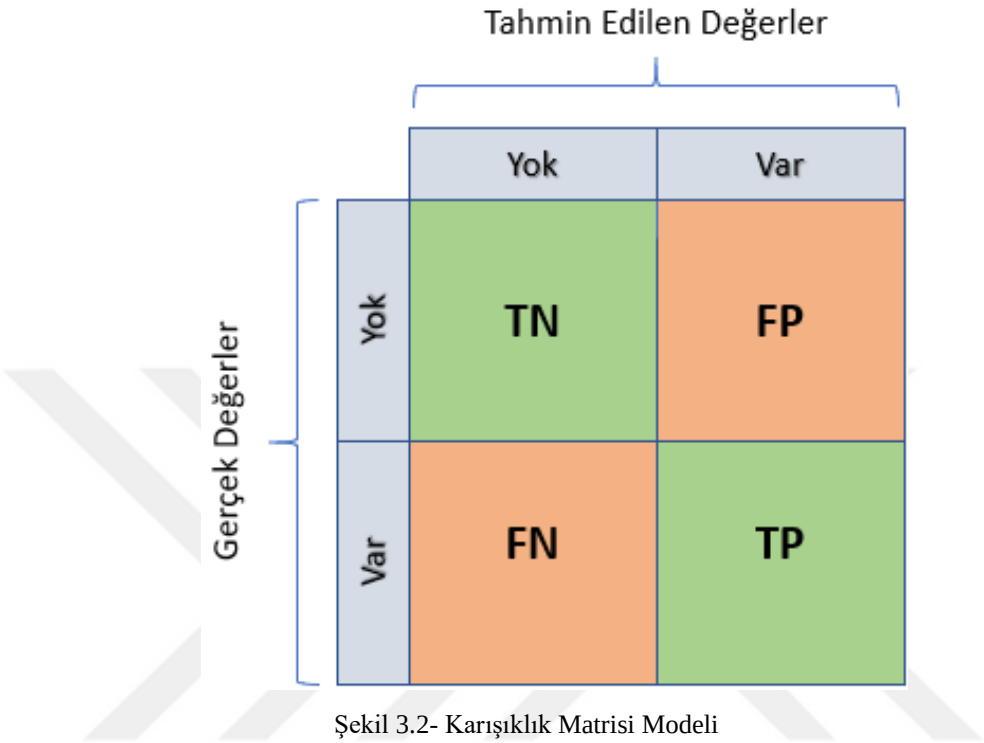
Karışıklık matrisi, makine öğreniminde önceden belirlenmiş hedef veri kümelerinden elde edilen modellerin performansını değerlendirmek için en yaygın kullanılan yöntemdir [24]. Matriste pozitif ve negatif veriler değerlendirme sonuçlarına göre gösterilir. Gelinen aşamada matris ile gösterim alınan sonuçları analiz etme açısından yarar sağlamıştır. Karışıklık matris modeli Şekil 3.2'de gösterilmiştir.

• **Gerçek Pozitif (TP):** Test verilerindeki değer, model değerlerinin sınıfıyla aynıdır. Doğru sınıflandırma yapılmıştır.

• **Yanlış Negatif (FN):** Test verilerindeki değer, model tarafından üretilen sınıftan farklıdır. Yanlış sınıflandırma yapılmıştır.

• **Yanlış Pozitif (FP):** Gerçek değer negatiftir ancak pozitif olarak sınıflandırılmıştır. Yanlış sınıflandırma yapılmıştır.

• **Gerçek Negatif (TN):** Gerçek değer negatiftir ve negatif olarak sınıflandırılmıştır. Doğru sınıflandırma yapılmıştır.



Şekil 3.2- Karışıklık Matrisi Modeli

Karışıklık matrisi ile mevcut veri kümesinin; doğruluk (accuracy), kesinlik (precision) ve duyarlılık (recall) değerleri hesaplanmıştır. Doğruluk değeri denklem 3.2'de, kesinlik değeri denklem 3.3'de, duyarlılık değeri ise denklem 3.4'deki formüller ile hesaplanmıştır.

$$Doğruluk\ (Accuracy) = (TP + TN) / (TP + TN + FP + FN) \quad (3.2)$$

$$Kesinlik\ (Precision) = TP / (TP + FP) \quad (3.3)$$

$$Duyarlılık\ (Recall) = TP / (TP + FN) \quad (3.4)$$

3.4. Temel Bileşen Analizi (PCA)

PCA, daha az değişkenli ve minimum veri kaybına sahip çok değişkenli bir veri kümesindeki bilgileri açıklayan matematiksel bir tekniktir. Başka bir deyişle PCA, çok sayıda birbirine bağlı değişken içeren veri kümesindeki veri kümesini koruyarak veri kümesinin boyutunun daha küçük bir boyuta küçültülmesini sağlayan bir dönüştürme tekniğidir [33].

PCA, büyük boyutlu veri kümelerindeki boyutsallığı azaltır. Teknik, boyut küçültme işlemindeki veri kümesindeki değişken sayısını azaltmayı amaçlamaktadır. Dönüşümden sonra elde edilen değişkenlere ilk değişkenlerin ana bileşenleri denir. İlk temel bileşen olarak, varyans değeri en büyük ile seçilir ve diğer temel bileşenler varyans değerlerini azaltmak için sıralanmaktadır.

4. ÖNERİLEN MODEL

Yapılan çalışmada, daha önceden geliştirilen çalışmalara göre daha iyi sonuçların sağlanması ve tutarlılık amaçlanmıştır. Bu amaca hizmet edebilmek için hangi veri setleri kullanıldığı daha önce Bölüm 2'de bahsedilmişti. Hangi makine öğrenmesi algoritmaları ve hangi metodolojilerin kullanıldığından bu bölümde bahsedilmiştir. Yapılan çalışma sayesinde oldukça yararlı geri dönüşümler sağlanacaktır. Bu bölümde sistemin genel yapısı, kullanılan özellikler ve çalışmanın genel işleyişi anlatılacaktır. Daha sonra tez çalışmasında geliştirilen modelde kullanılan ide ve kütüphanelerin bilgisi verilmiştir.

4.1. Kullanılan Veri Setleri

Veri kümesi bölümünde anlatıldığı gibi; CM1, KC1, KC2, JM1 ve PC1 veri setleri PROMISE veri deposundan kullanılmıştır. PROMISE deposundaki çalışmalar bu alandaki araştırmacılar tarafından yoğun olarak kullanılan UCI Makine Öğrenim Deposu'ndan ilham alınmıştır [31].

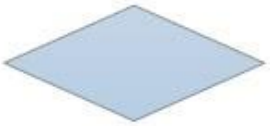
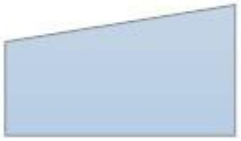
4.2. Modelin Akış Diyagramı

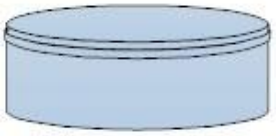
Algoritmalar problemlerin çözümünün en basit, en net ve en sıralı biçimde belirtilmiş halidir. Problemlerin çözümünde bir algoritma geliştirildikten sonra bu algoritma; metinlerle, sözde kodlarla veya akış diyagramları ile sunulur.

Akış diyagramları, akış şeması olarak da bilinmektedir. Akış diyagramları algoritmaların belirli şekiller ve simgeler ile ifade edilmiş halidir. Bu yüzden algoritma sunumlarında yaygınlıkla kullanılır. Açık ve anlaşılması kolay şemalarda karmaşık süreçleri belgelemek, araştırmak, planlamak, geliştirmek ve iletişim kurmak için çok sayıda alanda yaygın olarak kullanılmaktadır. Akış diyagramları geliştirilen bir algoritmayı şekilsel olarak ifade etmekte ve anlaşılabilirliğini kolaylaştırmaktadır. Akış diyagramlarını çeşitli şekiller barındırmaktadır. Bu şekiller işlevlerin simgesel gösterimleridir.

Bilgisayar programlama dilinde kullanılan programlama kodları bu şekiller ile basit olarak gösterilerek programı yazan programcı yanı sıra başka bir programcı tarafından incelendiğinde anlaşılabilirliğini kolaylaştırmak veya programcı uzun süre önce yazdığı program kodunu unutmuş ise hatırlamasına yardımcı olmak amacına da sahiptir. Örneğin; Büyük resmi düşünmeyi organize etmeye ve kodlama zamanı geldiğinde bir rehber sağlamaya yardımcı olabilmektedir [34]. Tablo 4.1'de akış diyagramlarında kullanılan semboller gösterilmiştir.

Tablo 4.1-Akış Diyagramı Sembolleri

Bağlantı Sembolü: Akış yönü ve semboller arasındaki bağlantıyı gösterir	
Başlangıç / Bitiş Sembolü: Sonlandırıcı, işlemin nerede başladığını veya nerede bittiğini gösterir	
Eylem veya Süreç Sembolü: Bir işlemi, süreci veya aktiviteyi ifade eder	
Belge Sembolü: Basılı bir belge veya raporu temsil eder	
Birden Fazla Belge Sembolü: İşlemde birden çok belgeyi temsil eder	
Karar Sembolü: Bir karar veya dallanma noktasını temsil eder	
Giriş / Çıkış Sembolü: Giriş – çıkış yani sisteme giren veya sistemden gönderilen materyal, bilgileri temsil eder	
Manuel Giriş Sembolü: Bilgilerin sisteme manuel olarak girmesi istendiği bir adımı temsil eder	
Hazırlık Sembolü: Süreçte başka bir adımın ön hazırlık veya kurulumunu temsil eder. Devam etmeden önce bir işlemin yapılması, ayarlanması veya değiştirilmesi gerektiğini gösterir	

Bağlayıcı Sembolü: Akışın, eşleşen bir sembolün yerleştirildiği yerde devam ettiğini gösterir	
Veya Sembolü: İşlem akışının ikiden fazla dalda devam ettiğini gösterir	
Alt Yordam Sembolü: Daha büyük bir süreç içinde gömülü belirli bir görevi gerçekleştiren eylemler dizisini belirtir	
Manuel Döngü Sembolü: Manuel olarak durdurulana kadar tekrarlamaya devam edecek bir dizi komut dizisini belirtir	
Döngü Sınırı Sembolü: Bir döngüün durması gereken noktayı belirtir	
Gecikme Sembolü: İşlemde gecikme olduğunu gösterir	
Veri Depolama – Saklama Sembolü: Verilerin saklandığı bir adımı belirtir	
Veritabanı Sembolü: Arama ve sıralama için izin veren standart bir yapıya sahip bilgi listesini gösterir	

Dahili Depolama Sembolü: Yazılım tasarım akış şemalarında kullanılan bir program sırasında bilgilerin bellekte saklandığını belirtir	
Ekran Sembolü: Bilgileri gösteren bir adımı belirtir	

Yapılan çalışmada kullanılan akış diyagramı Şekil 4.1'de gösterilmektedir. Şemanın adımları şu şekildedir:

1. Adım: Başla işlemi ile akış başlatılır.

2. Adım: Sisteme hatalı ve hatasız veriler barındıran kaynaklardan elde edilerek bir araya getirilen yazılım hata veri setlerinin girişi sağlanır.

3. Adım: Sisteme girişi yapılan veriler işlenir.

4. Adım: Verilerin normalizasyon işlemi yapılır.

5. Adım: Çapraz doğrulama (CV) yöntemi ile makine öğrenmesi algoritmaları eğitilir.

6. Adım: Eğitim sonucu karışıklık matrisleri ile gösterilir ve izlenir.

7. Adım: Yapılan işlemlerde 'Temel Bileşen Analizi (PCA) kullanıldı mı?' sorusu sorulur.

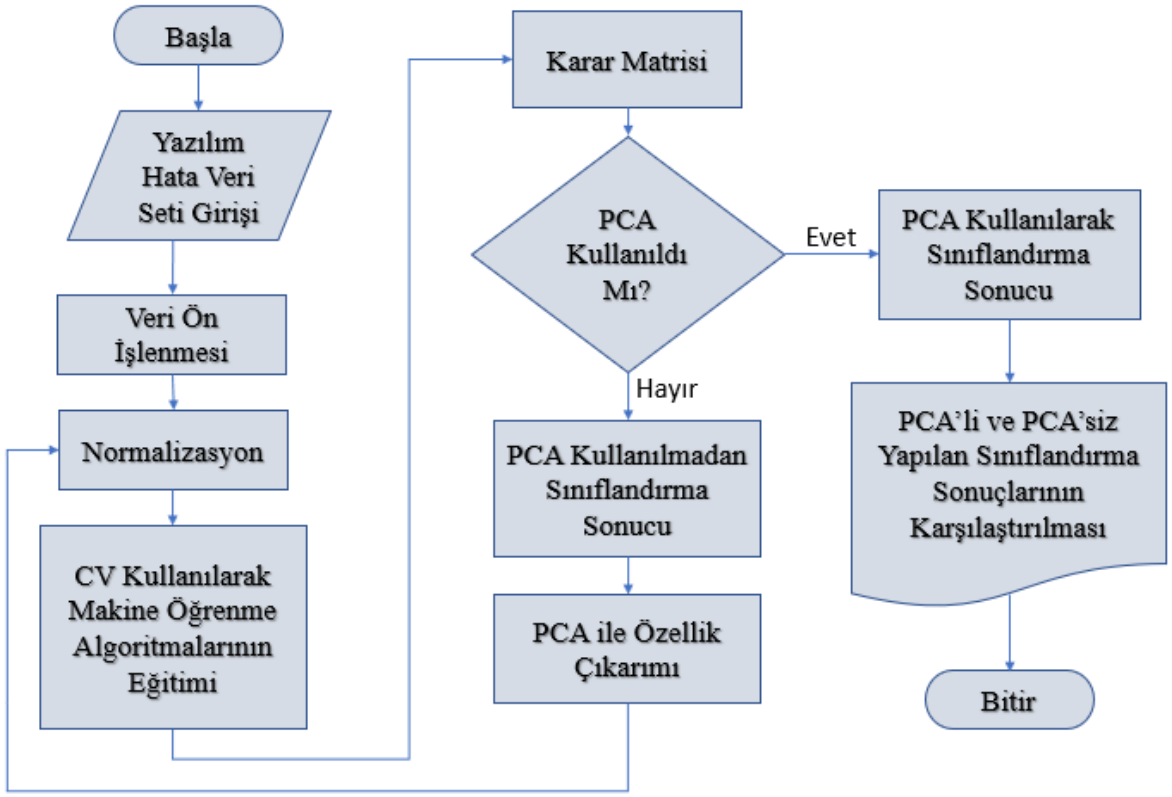
8. Adım: 7. Adımda sorulan sorunun cevabı hayır ise PCA'siz performans analizi yapılır. Eğer sorunun cevabı evet ise 10.adıma atlanır.

9. Adım: PCA yöntemi ile özellik çıkarımı yapılır ve 4. Adıma atlanır.

10. Adım: PCA'li performans analizi yapılır.

11. Adım: PCA'li ve PCA'siz olarak işlenen performans analizleri karşılaştırılır.

12. Adım: Bitir işlemi ile akış sona erdirilir.



Şekil 4.1- Geliştirilen Modelin Akış Diyagramı

4.3. Modelde Kullanılan Teknoloji

Tez çalışması yazılım olarak Python dili kullanılmıştır. İde olarak ise Anaconda Navigator vasıtası ile Speyder 3.7 kullanılmıştır. Python'un tercih edilmesinin sebebi kolaylıkla verileri analiz edebilme yeteneğidir ve ayrıca kullanışlı arayüzü, tercih edilmesinin nedenleri arasındadır. Python, sahip olduğu geniş kütüphanelerle yazılım sürecini normale göre daha hızlanmasını sağlar. Scipy ve Numpy gibi bilimsel hesaplamalardan Django gibi web geliştirme araçlarına kadar gerekli doküman desteği sağlamaktadır.

Anaconda platformu veri analizi için gerekli kaynak kütüphanelerini içinde barındırdığından dolayı süreci hızlandırmıştır. Kullanılan kodlama ve mimari tamamen tensorflow alt yapısına uygundur.

Kullanılan kütüphaneler:

- Makine öğrenmesi algoritmaları için sckit-learn,
- Dizi işlemleri için numpy,
- Veriyi görselleştirmek için mathplotlib-seaborn,
- Veriyle işlemler yapmak için pandas kütüphaneleri kullanılmıştır.

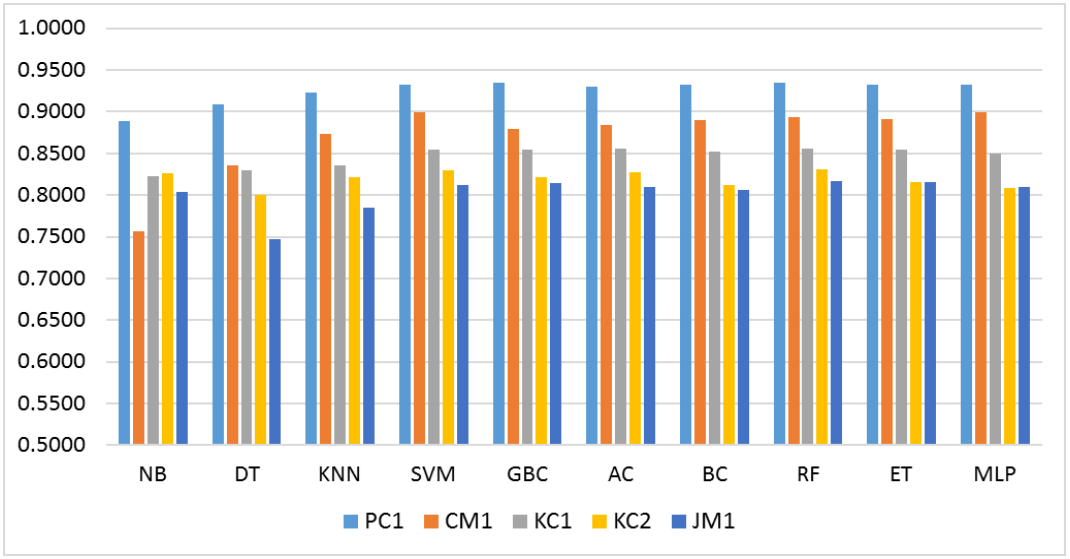
5. TEST SONUÇLARI VE DEĞERLENDİRMELER

Bu bölümde tüm veri kümelerinin PCA yöntemi kullanılmadan ve PCA yöntemi kullanılarak, önerilen sistem tarafından yapılan testlerin tüm sonuçları açıklanmakta ve grafiklerde gösterilmektedir. Bu çalışmada, 21 olan özellik sayısı PCA kullanılarak 15'e düşürülmüştür. Bu indirgeme sonucunda daha iyi sonuçlar gözlemlenmiştir. Sistem içinde aynı işlemleri PCA ile tamamladıktan sonra, sistem çıkışları karşılaştırmaya başlanmıştır.

Sonuç – 1

PCA yöntemi kullanılmadan sırasıyla PC1, CM1, KC1, KC2 ve JM1 veri kümelerine göre sistemde kullanılan 10 algoritmanın doğruluk sonuçlarının grafiği Şekil 5.1'de gösterilmektedir. Oluşan grafiğe göre sonuçlar;

- PC1 veri seti için en yüksek sonucu 0.9351 ile SVM algoritması verirken, en düşük sonucu 0.8891 ile NB algoritması vermiştir.
- CM1 veri seti için en yüksek sonucu 0.8996 ile SVM ve MLP algoritmaları verirken, en düşük sonucu ise 0.7570 ile NB algoritmasında vermiştir.
- KC1 veri seti için en yüksek sonucu 0.8559 ile AC algoritması verirken, en düşük sonucu 0.8227 ile NB algoritması vermiştir.
- KC2 veri seti için en yüksek sonucu 0.8314 ile RF algoritması verirken, en düşük sonucu 0.8008 ile DT algoritması vermiştir.
- JM1 veri seti için en yüksek sonucu 0.8168 ile RF algoritması verirken, en düşük sonucu 0.7467 ile DT algoritması vermiştir.

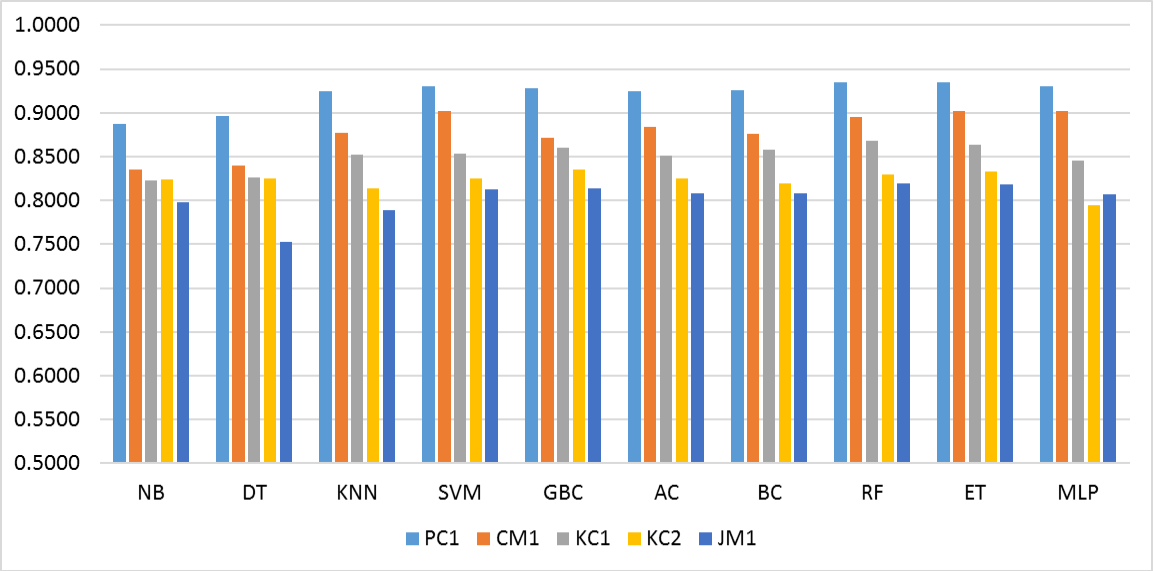


Şekil 5.1- PCA Yöntemi Kullanılmadan Doğruluk Sonuçları Grafiği

Sonuç – 2

PCA yöntemi kullanılarak sırasıyla PC1, CM1, KC1, KC2 ve JM1 veri kümelerine göre sistemde kullanılan 10 algoritmanın doğruluk sonuçlarının grafiği Şekil 5.2'de gösterilmiştir. Oluşan grafiğe göre sonuçlar;

- PC1 veri seti için en yüksek sonucu 0.9351 ile RF ve ET algoritmaları verirken, en düşük sonucu ise 0.8873 ile NB algoritmasında vermiştir.
- CM1 veri seti için en yüksek sonucu 0.9016 ile SVM, ET ve MLP algoritmaları verirken, en düşük sonucu ise 0.8353 ile NB algoritmasında vermiştir.
- KC1 veri seti için en yüksek sonucu 0.8677 ile RF algoritması verirken, en düşük sonucu 0.8227 ile NB algoritması vermiştir.
- KC2 veri seti için en yüksek sonucu 0.8352 ile GBC algoritması verirken, en düşük sonucu 0.7952 ile MLP algoritması vermiştir.
- JM1 veri seti için en yüksek sonucu 0.8199 ile RF algoritması verirken, en düşük sonucu 0.7529 ile DT algoritması vermiştir.

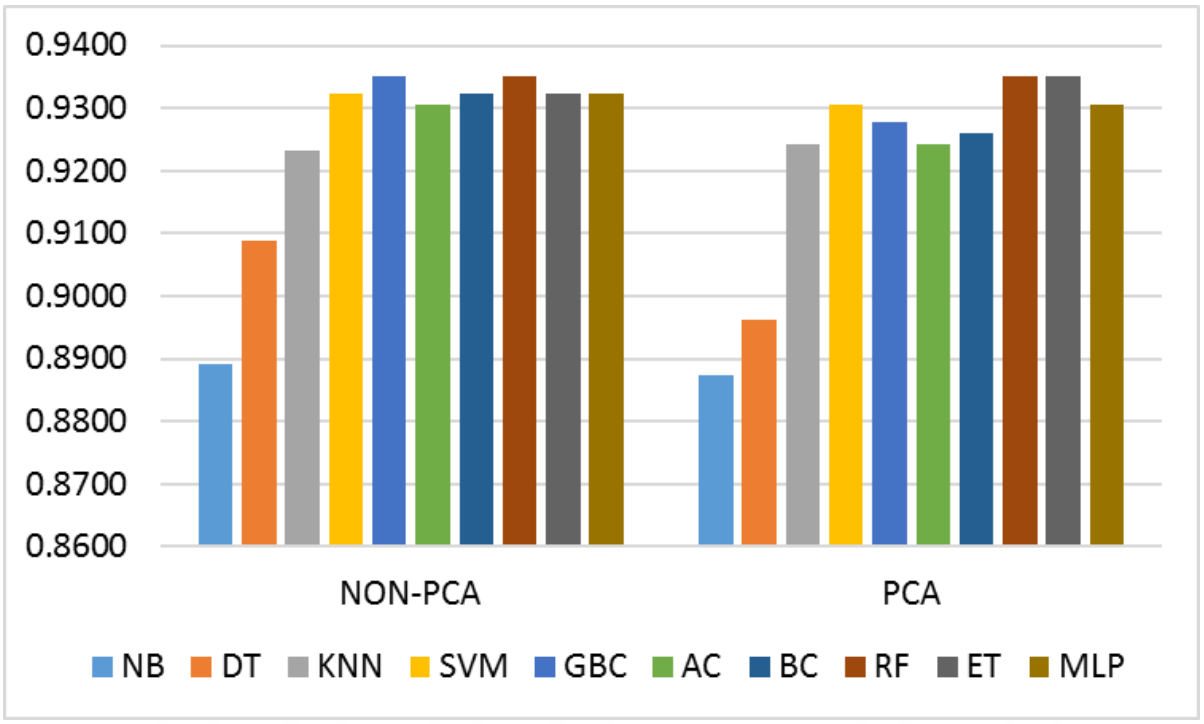


Şekil 5.2- PCA Yöntemi Kullanılan Doğruluk Sonuçları Grafiği

Sonuç – 3

PCA yöntemi kullanılarak ve kullanılmayarak PC1 veri kümesi için kullanılan algoritmalarda verilen doğruluk sonuçları Şekil 5.3'de gösterilmektedir. Oluşan grafiğe göre sonuçlar;

- PCA yöntemi kullanılmadan en yüksek sonucu 0.9351 ile RF ve GBC algoritmaları verirken, en düşük sonucu ise 0.8891 ile NB algoritmasında vermiştir.
- PCA yöntemi kullanılarak en yüksek sonucu 0.9351 ile RF ve ET algoritmaları verirken, en düşük sonucu ise 0.8873 ile NB algoritmasında vermiştir.

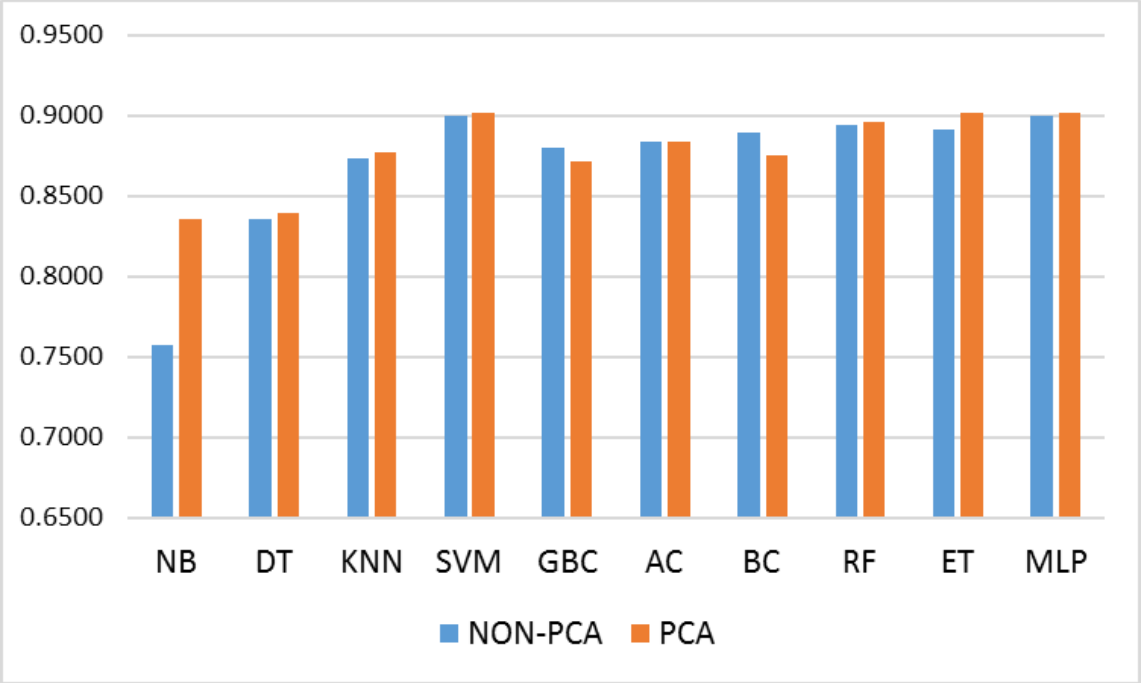


Şekil 5.3- PC1 Veri Seti İçin Doğruluk Sonuçları Grafiği

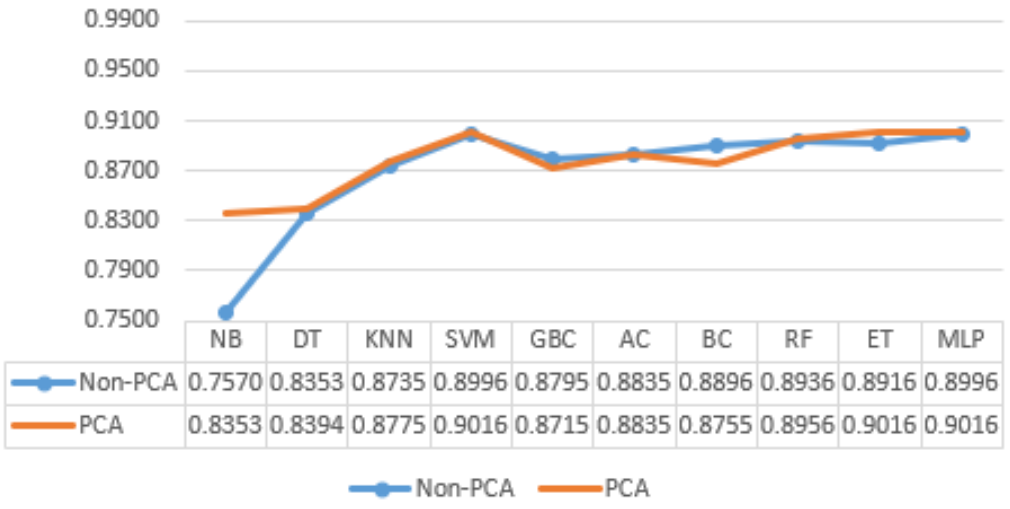
Sonuç – 4

PCA yöntemi kullanılarak ve kullanılmayarak, CM1 veri seti için 10 algoritmada verilen doğruluk sonuçları Şekil 5.4'te ve Şekil 5.5'de gösterilmektedir. Oluşan grafiğe göre sonuçlar;

- PCA yöntemi kullanılmadan en yüksek sonucu 0.8996 ile SVM ve MLP algoritmaları verirken, en düşük sonucu ise 0.7570 ile NB algoritmasında vermiştir.
- PCA yöntemi kullanılarak en yüksek sonucu 0.9016 ile SVM, MLP ve ET algoritmaları verirken, en düşük sonucu ise 0.8353 ile NB algoritmasında vermiştir.



Şekil 5.4- CM1 Veri Seti İçin Doğruluk Sonuçları Grafiği



Şekil 5.5- CM1 Veri Seti İçin Doğruluk Sonuçları Grafiği-2

Sonuç – 5

Araştırmada kullanılan algoritmalar; veri kümelerindeki ortalama değerler, PCA yöntemi kullanılmadan Tablo 5.1'de gösterilmiştir. Örneğin, değerler hesaplanırken PC1 veri kümesinin doğruluk değeri, tüm algoritmalarda doğruluk sonuçlarının toplamlarının aritmetik ortalaması olarak alınmıştır. Oluşan grafiğe göre sonuçlar;

- PC1 veri setinin doğruluk değeri en yüksek çıkarken, JM1 veri setinin doğruluk değeri ise en küçük çıkmıştır.

- Duyarlılık sıralaması; KC2>KC1>PC1>JM1>CM1 şeklinde sıralanmıştır.
- Keskinlik değerinse ise en yüksek veri seti KC2 çıkarken, en düşük ise CM1 veri seti çıkmıştır.

Tablo 5.1-PCA Yöntemi Kullanılmadan Veri Seti Bazlı Metrikler

Veri Seti	Doğruluk	Duyarlılık	Kesinlik	TP	TN	FP	FN
PC1	**0.925**	0.429	0.234	18	1008	59	24
CM1	0.871	0.241	0.143	7	427	42	22
KC1	0.846	0.506	0.258	84	1701	242	82
KC2	0.820	0.594	0.383	41	387	66	28
JM1	0.802	0.470	0.191	401	8324	1702	453

Sonuç – 6

Araştırmada kullanılan algoritmalar; veri kümelerindeki ortalama değerler, PCA yöntemi kullanılarak Tablo 5.2'de gösterilmiştir. Örneğin, değerler hesaplanırken PC1 veri kümesinin doğruluk değeri, tüm algoritmalarda doğruluk sonuçlarının toplamlarının aritmetik ortalaması olarak alınmıştır. Oluşan grafiğe göre sonuçlar;

- PC1 veri setinin doğruluk değeri en yüksek çıkarken, JM1 veri setinin doğruluk değeri ise en küçük çıkmıştır.
- Duyarlılık sıralaması; KC2>KC1> JM1>PC1>CM1 şeklinde sıralanmıştır.
- Keskinlik değerinse ise en yüksek veri seti KC2 çıkarken, en düşük ise CM1 veri seti çıkmıştır.

Tablo 5.2-PCA Yöntemi Kullanılarak Veri Seti Bazlı Metrikler

Veri Seti	Doğruluk	Duyarlılık	Kesinlik	TP	TN	FP	FN
PC1	**0.922**	0.342	0.151	12	1011	65	21
CM1	0.878	0.246	0.114	6	432	43	17
KC1	0.850	0.531	0.267	87	1706	239	77

KC2	0.823	0.606	0.387	41	388	66	27
JM1	0.803	0.475	0.190	400	8335	1703	442

Sonuç – 7

Çalışmada PCA yöntemi kullanılmadan, algoritmalara göre ortalama değerler Tablo 5.3'te gösterilmektedir. Oluşan grafiğe göre sonuçlar;

- Doğruluk sıralamasında hemen hemen tüm algoritmalar birbirlerine yakın sonuçlar üretirken en yüksek değeri RF algoritması vermiştir. En düşük sonucu ise DT algoritması vermiştir.
- Duyarlılık sonuçlarında da doğruluk sıralamasında olduğu gibi hemen hemen tüm algoritmalar birbirlerine yakın sonuçlar üretirken en yükse değeri MLP algoritması vermiştir. En düşük sonucu ise yine DT algoritması vermiştir.
- Keskinlik sonuçlarında ise DT algoritması en yüksek sonucu verirken, MLP algoritması en düşük sonucu vermiştir.

Tablo 5.3-PCA Yöntemi Kullanılmadan Algoritma Bazlı Metrikler

Veri Seti	**Doğruluk**	**Duyarlılık**	**Kesinlik**	**TP**	**TN**	**FP**	**FN**
NB	0.811	0.436	0.237	126	2328	406	163
DT	0.775	0.360	0.360	192	2152	341	340
KNN	0.806	0.429	0.295	157	2282	375	209
SVM	0.830	0.632	0.086	46	2464	486	27
GBC	0.831	0.570	0.166	88	2425	444	67
AC	0.828	0.543	0.151	81	2424	452	68
BC	0.824	0.504	0.248	132	2361	400	130
RF	**0.833**	0.564	0.247	132	2390	401	102
ET	0.831	0.550	0.242	129	2386	403	105
MLP	0.827	0.671	0.034	18	2482	514	9

Sonuç – 8

Çalışmada PCA yöntemi kullanılarak, algoritmalara göre ortalama değerler Tablo 5.4'te gösterilmektedir. Oluşan grafiğe göre sonuçlar;

- Doğruluk sıralamasında hemen hemen tüm algoritmalar birbirlerine yakın sonuçlar üretirken en yüksek değeri RF algoritması vermiştir. En düşük sonucu ise DT algoritması vermiştir.
- Duyarlılık sonuçlarında da doğruluk sıralamasında olduğu gibi hemen hemen tüm algoritmalar birbirlerine yakın sonuçlar üretirken en yükse değeri SVM algoritması vermiştir. En düşük sonucu ise yine MLP algoritması vermiştir.
- Keskinlik sonuçlarında ise KNN algoritması en yüksek sonucu verirken, MLP algoritması en düşük sonucu vermiştir.

Tablo 5.4-PCA Yöntemi Kullanılarak Algoritma Bazlı Metrikler

Veri Seti	**Doğruluk**	**Duyarlılık**	**Kesinlik**	**TP**	**TN**	**FP**	**FN**
NB	0.810	0.431	0.240	128	2322	404	169
DT	0.779	0.372	0.373	199	2157	334	334
KNN	0.811	0.443	0.279	149	2304	384	187
SVM	0.830	0.602	0.111	59	2452	473	39
GBC	0.831	0.567	0.170	91	2422	442	69
AC	0.826	0.519	0.169	90	2408	442	84
BC	0.826	0.516	0.235	125	2374	407	117
RF	**0.837**	0.599	0.238	127	2406	406	85
ET	0.836	0.590	0.231	123	2406	409	86
MLP	0.823	0.000	0.000	0	2491	532	0

6. SONUÇLAR

Yazılım hatası tahmini, Yazılım Mühendisliği alanındaki en önemli süreçlerden biridir. Hatasız yazılım veya minimum hata ile yazılımı oluşturmak için yazılımdaki (özellikle modüllerindeki) hataların doğru tahmini hayati öneme sahiptir. Uygun tahmin modellerinin kullanılmasıyla yazılım yöneticileri çalışanları için verimliliği artırabilir. Sadece test için değil, aynı zamanda yazılım sistemlerinin bakımı için de daha fazla zaman ayırabilir. Böylece harcanan maliyet de daha iyi seviyede olacaktır.

Literatürde bu kritik işlemler için farklı yaklaşımlar kullanılmaktadır. Yazılım hatalarının tahmini için Karar Ağacı Algoritması, Naif Bayes Sınıflandırması, K-En Yakın Komşu Algoritması, Destek Vektör Makinesi, Rastgele Orman, Ekstra Ağaçlar, Adaboost Sınıflandırması, Gradyan Arttırma, Torbalama ve Çok Katmanlı Algılayıcılar olarak 10 makine öğrenme yönteminin kullanılmasını tercih edilmiştir. PROMISE deposundan CM1, KC1, KC2, JM1 ve PC1 ortak veri kümeleri kullanılmıştır. Karşılaştırmalı çalışmalar, kullanılan algoritmaların PC1 veri kümesi için daha iyi ortalama doğruluk oranlarına sahip olduğunu ve PCA yaklaşımı ile Random Forest öğrenme modellerinin, kullanılan veri kümeleri için daha iyi ortalama performans verdiğini göstermiştir. Ulaşılan performans değerleri, önerilen modellerin yazılım hatalarının tahmini için iyi bir doğruluğa sahip olduğunu göstermektedir.

KAYNAKÇA

[1] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero and M. Zelkowitz, "What we have learned about fighting defects," Los Alamitos, CA, USA, 2002.

[2] R. S. Pressman, Software Engineering: A Practitioner's Approach, 7. ed., 2005.

[3] "ISO 25000," [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. [Accessed 25 Mayıs 2020].

[4] I. Arora, V. Tetarwal and A. Saha, "Open Issues in Software Defect Prediction," in *International Conference on Information and Communication Technologies (ICICT 2014)*, Delhi, 2015.

[5] F. Akmel, E. Birihanu and B. Siraj, "A Literature Review Study of Software Defect Prediction using Machine Learning Techniques," *International Journal of Emerging Research in Management & Technology*, vol. 6, no. 6, pp. 300-306, 2017.

[6] A. M. Karimi, J. S. Fada, J. Lui, J. L. Braid, M. Koyuturk and R. H. French, "Feature Extraction, Supervised and Unsupervised Machine Learning Classification of PV Cell Electroluminescence Images," in *IEEE Conference*, Waikoloa Village, HI, USA, USA, 2018.

[7] M. da Silva Ferreira, L. F. Vismari, P. S. Cugnasca, J. R. de Almeida, J. B. Camargo and G. Kallemback, "A Comparative Analysis of Unsupervised Learning Techniques for Anomaly Detection in Railway Systems," in *IEEE*, Boca Raton, FL, USA, USA, 2019.

[8] M. C. Prasad, L. Florence and A. Arya, "A Study on Software Metrics based Software Defect Prediction using Data Mining and Machine Learning Techniques," *International Journal of Database Theory and Application*, vol. 8, no. 3, pp. 179-190, 2015.

[9] S. Aleem, L. F. Carpetz and F. Ahmed, "BENCHMARKING MACHINE LEARNING TECHNIQUES FOR SOFTWARE DEFECT DETECTION," *International Journal of Software Engineering & Applications (IJSEA)*, vol. 6, no. 3, pp. 11-23, May 2015.

[10] M. Anbu and G. S. Anandha Mala, "Feature selection using firefly algorithm in software defect prediction," *Springer Science Business Media, LLC 2017*, pp. 10925-10934, 2017.

[11] X. Yang, K. Tang and X. Yao, "A Learning-to-Rank Approach to Software Defect Prediction," pp. 234 - 246, 2015.

[12] A. K. Verma, S. Pal and S. Kumar, “Prediction of Skin Disease Using Ensemble Data Mining Techniques and Feature Selection Method - a Comparative Study,” in *Applied Biochemistry and Biotechnology*, Springer US, 2020, pp. 341-359.

[13] R. Collobert and S. Bengio, “Links between Perceptrons, MLPs and SVMs,” in *Proceedings of International Conference on Machine Learning (ICML)*, 2004.

[14] A. G. Koru and H. Liu, “Building effective defect-prediction models in practice,” pp. 23-29, Jan 2005.

[15] L. Pelayo and S. Dick, “Applying Novel Resampling Strategies To Software Defect Prediction,” in *Fuzzy Information Processing NAFIPS '07*, San Diego, USA, 2007.

[16] T. Menzies, J. Greenwald and A. Frank, “Data Mining Static Code Attributes to Learn Defect Predictors,” *IEEE Transactions on Software Engineering*, pp. 2-13, Jan 2007.

[17] S. Lessmann, B. Baesens, C. Mues and S. Pietsch, “Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings,” *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485-496, Aug 2008.

[18] C. Catal and B. Diri, “A systematic review of software fault prediction studies,” *Expert Systems with Applications*, vol. 36, no. 4, pp. 7346-7354, May 2009.

[19] D. Gupta, V. K. Goyal ve H. Mittal, «Comparative Study of Soft Computing Techniques for Software Quality Model,» *International Journal of Software Engineering Research & Practices*, cilt 1, no. 1, pp. 33-37, Jan 2011.

[20] T. Hall, S. Beecham, D. Bowes, D. Gray and S. Counsell, “A Systematic Literature Review on Fault Prediction Performance in Software Engineering,” *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276-1304, Jan 2012.

[21] S. Wang and X. Yao, “Using Class Imbalance Learning for Software Defect Prediction,” *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434-443, 2013.

[22] P. Paramshetti and D. A. Phalke, “Survey on Software Defect Prediction Using Machine Learning Techniques,” *International Journal of Science and Research (IJSR)*, pp. 1394-1397, Dec 2014.

[23] K. R. Magal and S. G. Jacob, “Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques,” *International Journal of Computer Applications,* vol. 117, no. 23, pp. 18-23, May 2015.

[24] G. E. A. P. .. A. Batista, R. C. Prati and M. C. Monard, *A study of the behavior of several methods for balancing machine learning training data,* 1 ed., vol. 6, New York, NY: New York, NY, USA: ACM, 2004, pp. 20-29.

[25] D. Tomar and S. Agarwal, *Prediction of Defective Software Modules Using Class Imbalance Learning,* 2016, pp. 1-12.

[26] C. Catal and B. Diri, *Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem,* Kocaeli, 2009, pp. 1040-1058.

[27] G. Karataş, Ö. Demir and Ö. K. Şahingöz, “Deep Learning in Intrusion Detection Systems,” in *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*, Ankara, 2018.

[28] S. I. Baykal, D. Bulut and Ö. K. Şahingöz, “Comparing deep learning performance on BigData by using CPUs and GPUs,” in *2018 Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT)*, İstanbul, 2018.

[29] T. Zhang, X. Jing, Z. Sun, Y. Zhang and Y. Yan, “Software Defect Prediction based on Machine Learning Algorithms,” in *2019 IEEE 5th International Conference on Computer and Communications (ICCC) Computer and Communications (ICCC), 2019 IEEE 5th International Conference,* 2019.

[30] G. Luo, Y. Ma ve K. Qin, «Active learning for software defect prediction,» *IEICE TRANSACTIONS on Information and Systems,* cilt 95, no. 6, pp. 1680-1683, 2012.

[31] “Promise datasets page,” NASA, [Online]. Available: http://promise.site.uottawa.ca/SERepository/datasets-page.html. [Accessed 12 Nisan 2020].

[32] “Wikipedia,” Wikimedia Foundation, [Online]. Available: https://en.wikipedia.org/wiki/Cyclomatic_complexity. [Accessed 11 Temmuz 2020].

[33] X. Jin ve R. Bie, «Random Forest and PCA for Self-Organizing Maps Based Authomatic Music Genre Discrimination,» %1 içinde *Conference on Data Mining*, Las Vegas, Nevada, 2006.

[34] Recep, “Yazılım Bilişim,” [Online]. Available: https://www.yazilimbilisim.net/algoritma/akis-diyagrami-nedir/. [Accessed 18 Haziran 2020].