T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MAKİNE ÖĞRENME ALGORİTMALARI KULLANILARAK YAZILIM HATA KESTİRİMİNİN İYİLEŞTİRİLMESİ

DOKTORA TEZİ

Ömer Faruk ARAR

Enstitü Anabilim Dalı	**:**	**BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ**
Tez Danışmanı	**:**	**Doç. Dr. Kürşat AYAN**

Temmuz 2016

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MAKİNE ÖĞRENME ALGORİTMALARI KULLANILARAK YAZILIM HATA KESTİRİMİNİN İYİLEŞTİRİLMESİ

DOKTORA TEZİ

Ömer Faruk ARAR

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ

Bu tez 14 / 07 /2016 tarihinde aşağıdaki jüri tarafından oybirliği/oyçokluğu ile kabul edilmiştir.

Doç. Dr.
İbrahim ÖZÇELİK
Jüri Başkanı

Yrd. Doç. Dr.
Gültekin ÇAĞIL
Üye

Doç. Dr.
Kürşat AYAN
Üye

Doç. Dr.
Pakize ERDOĞMUŞ
Üye

Yrd. Doç. Dr.
Pınar Onay DURDU
Üye

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim.

Ömer Faruk ARAR
14.07.2016

TEŞEKKÜR

Doktora eğitimim boyunca değerli deneyimlerinden yararlandığım, her konuda bilgi ve desteğini almaktan çekinmediğim, çalışmaların planlanmasından yazılmasına kadar tüm aşamalarında değerli zamanını ayıran, makale çalışmalarının yayınlamasında yol göstericilik yapan, teşvik eden, değerli danışman hocam Doç. Dr. Kürşat AYAN'a teşekkürlerimi sunarım.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

acc	: Doğruluk (accuracy)
AIRS	: Yapay Bağışıklık Tanıma Sistemi (Artificial Immune Recognition System)
AUC	: Eğri Altındaki Alan (Area Under Curve)
bal	: Balans (balance)
BID	: Başarısız İyileştirme Denemesi
CFS	: Korelasyon-tabanlı Özellik Seçimi (Correlation-based Feature Selection)
CK	: Chidamber ve Kemerer
CSANN-ABC	: Cost-Sensitive Artificial Neural Network with Artificial Bee Colony
ECM	: Beklenen Yanlış Sınıflandırma Maliyeti (Expected Cost of Misclassification)
FDNB	: Özellik Bağımlı Naive Bayes (Feature Dependent Naive Bayes)
FN	: Yanlış Negatif (True Negative)
FNR	: Yanlış Negatif Oranı (False Negative Rate)
FP	: Yanlış Pozitif (True Pozitif)
FPR	: Yanlış Pozitif Oranı (False Positive Rate)
g-mean	: Geometrik Ortalama
hm	: Hataya meyilli
hmo	: Hatalı modül oranı
hsm	: Hatasızlığa meyilli
kd	: Karar doğrusu
MDS	: Maksimum Devir Sayısı

NB	: Naive Bayes
NECM	: Normalize Edilmiş Beklenen Yanlış Sınıflandırma Maliyeti (Normalized Expected Cost of Misclassification)
pd	: Hatalı modül tespit oranı (probability of detection)
pf	: Yanlış alarm oranı (probability of false alarm)
QMOOD	: Quality Model for Object-Oriented Design
RF	: Random Forest
ROC	: Alıcı İşletim Karakteristik Eğrisi (Receiver Operating Characteristics)
THR	: Eşik Değeri (Threshold)
TN	: Doğru Negatif (True Negative)
TP	: Doğru Pozitif (True Pozitif)
TPR	: Doğru Pozitif Oranı (True Positive Rate)
VARL	: Kabul Edilebilir Risk Değeri (Value of an Acceptable Risk Level)
YAK	: Yapay Arı Kolonisi
YSA	: Yapay Sinir Ağı
YYD	: Yazılım Yaşam Döngüsü (Software Development Life Cycle)

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Makine öğrenmesi, Yazılım hata kestirimi, yazılım kalitesi, yazılım metrikleri, sınıflandırma

Yazılım sistemleri günlük yaşantımızda çok önemli bir role sahiptir ve her geçen gün kullanımı daha da yaygınlaşmaktadır. Makinelerin ve servislerin büyük çoğunluğu kendi içlerinde farklı türde yazılım içerirler. Yazılım geliştiriciler, günlük kullanımını yaygınlaştırmak ve rekabette geri kalmamak için mümkün olduğunca hızlı bir şekilde yazılımları geliştirmektedirler. Yazılım yaşam döngüsü; genellikle analiz, tasarım, kodlama, test ve kurulum safhalarından oluştur. Son kullanıcıya hatadan arındırılmış bir yazılım sunabilmek için test safhası etkili olarak yürütülmelidir. Yazılım metrikleri, kaynak kodun kalitesini yansıtmayı amaçlarlar ve içeriği ile ilgili niceliksel bilgi verirler. Her bir metrik kodun farklı bir yönünü değerlendirir. Kaynak kodun kalitesi seviyesi ile risk seviyesi arasında bir ilişki vardır. Son 20 yıllık dönemde, akademisyenler, yazılım hata kestirimi problemine giderek artan bir ilgi göstermişler, daha gürbüz bir kestirim için çeşitli makine öğrenmesi yaklaşımları uygulanmıştır. Bu çalışmada da bu problem için çeşitli makine öğrenmesi modelleri önerilmiştir. Yapay Sinir Ağı ve Yapay Arı Kolonisi kombinasyonu, Lojistik Regresyon-tabanlı Bender Metot ve Naive Bayes bu çalışmada kullanılan algoritmalardır. Önerilen yaklaşımlar, herkese açık NASA Metrik Veri Programı ve PROMISE havuzunda bulunan veri setlerine uygulanmıştır. İstatistiki olarak güvenilir sonuçlar elde etmek ve örneklem yanlılığını azaltmak için deneyler *n*-küme çapraz validasyon ile kurgulanmıştır. Performansı arttırmak için önerilen modellere özellik seçimi, normalizasyon ve ayrıklaştırma gibi çeşitli veri ön işleme teknikleri uygulanmıştır. Deneylerden elde edilen sonuçlar diğer çalışmalar ile karşılaştırılmıştır. Bu çalışma, özellikle, yazılım geliştiricileri ve test personelinin kullanımı yönüyle katkı yapmaktadır. Yazılım geliştiricileri, düzenlemeye ihtiyaç duyulan sınıf veya modülleri görürler; dolayısıyla, bu modüllerin kalitesinin arttırılmasına ve risk seviyelerinin azaltılmasına katkı yapmış olurlar. Test personeli, daha çok test yoğunlaşması gerektiren modülleri tespit eder ve bunun neticesinde modüllerin önceliklendirmesinin risk seviyelerine göre yapılması sağlanmış olunur.

USING MACHINE LEARNING ALGORITHMS TO IMPROVE SOFTWARE DEFECT PREDICTION

SUMMARY

Keywords: Machine learning, software defect prediction, software quality, software metrics, classification

Software systems play a significant role in our daily lives and are becoming more common day-by-day. Most machines and services have some type of built-in software. Developers aim to advance such software as quickly as possible in order to expand its everyday use and to stay competitive. The software development life cycle generally includes analysis, design, implementation, test and release phases. The testing phase should be operated effectively in order to release bug-free software to end users. Software metrics aim to exhibit the quality of source code and give insight to it quantitatively. Each metric assesses the code from a different aspect. There is a relationship between the quality level and the risk level of source code. In the last two decades, academicians have taken an increasing interest in the software defect prediction problem, several machine learning techniques have been applied for more robust prediction. In this study, different machine learning models for this problem are proposed. A combination of traditional Artificial Neural Network and the novel Artificial Bee Colony algorithm, Logistic Regression-based Bender Method and Naive Bayes are algorithms used in this study. The proposed approaches were applied to publicly available datasets from the NASA Metrics Data Program and PROMISE repository. Experiments were set up by using *n*-fold cross-validation to obtain a statistically reliable results and to reduce sampling bias. Different data preprocessing techniques such as feature selection, normalization and discretization were also applied to proposed models to increase the performance. Results revealed from experiments were compared with other studies. This study makes contributions primarily for use by software developers and testers. Software developers can see classes or modules that require revising; this, consequently, contributes to an increment in quality for these modules and a decrement in their risk level. Testers can identify modules that need more testing effort and can prioritize modules according to their risk levels.

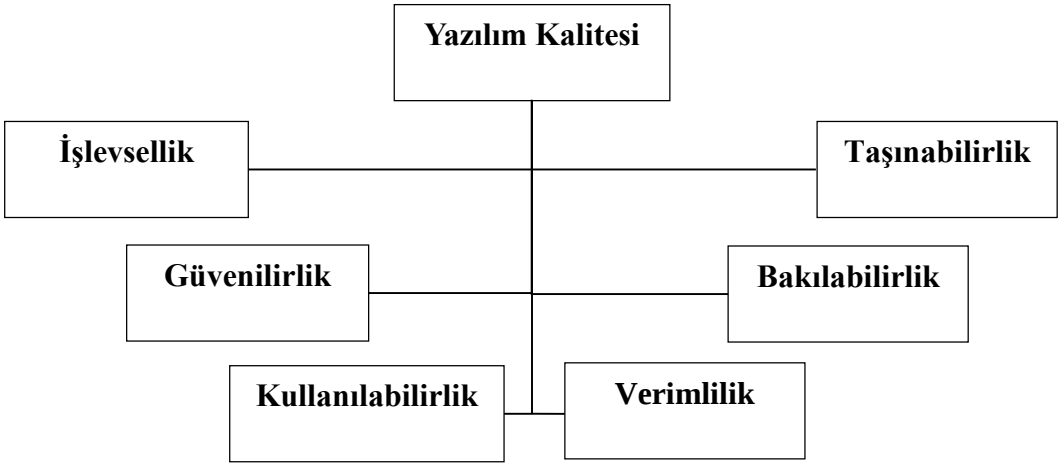
BÖLÜM 1. GİRİŞ

1.1. Yazılım Kalitesi

Crosby (1979)'ye göre yazılım kalitesi isterlere uygunluktur. Yazılımın başlangıç aşamasında isterler eksiksiz ve yeterli düzeyde ifade edilir. Eğer yazılım bu isterleri karşılıyorsa kaliteli yazılımdır.

Yazılım Kalitesi; geliştirici, yazılım mimarı ve son kullanıcı için faklı anlamlar ifade etmektedir. Son kullanıcı için; kaliteli bir yazılım, kendisinden beklenen görevleri tam ve verimli (performanslı) bir şekilde sağlayan yazılımdır. Örneğin, online bir alış-veriş sitesinden beklenen; güvenli ve kolay bir şekilde alış-verişin yapılabilmesidir. Yazılım geliştiricisi için kaliteli yazılım, bakımı kolay ve hataları hızlı çözülebilen yazılımdır. Yazılım mimarı için ise kaliteli yazılım, dokümantasyonu iyi yapılmış ve tekrar kullanılabilir birimleri içeren yazılımdır.

Uluslararası bir standart olan ISO 9126, yazılımda kalite ile ilgili standartları içermektedir. 1991 yılında yayınlanmış, 2001 ve 2004 yıllarında güncellenmiştir. Bu standardın birinci bölümü olan ISO 9126-1 yazılım kalitesini tanımlayan karakteristik ve bunların altında yer alan alt karakteristikleri içermektedir (Coallier, 2001). Bu standarda göre tanımlanmış olan karakteristikler Şekil 1.1.'de gösterilmiştir.



Şekil 1.1. ISO 9126'a göre yazılım kalite karakteristikleri

Bu karakteristiklerin tanımı ve bunların altında yer alan alt karakteristikler aşağıda belirtilmiştir (Erdemir ve ark., 2008):

- İşlevsellik: Yazılımın ihtiyaçları karşılama becerisi olarak tanımlanmaktadır. Uygunluk, doğruluk, birlikte çalışabilirlik ve güvenlik konuları bu kategori altında incelenmektedir.
- Güvenilirlik: Yazılımın düzgün çalışma halini muhafaza edebilme becerisi olarak tanımlanmaktadır. Olgunluk, hata toleransı ve geri kurtarma konuları bu kategori altında incelenmektedir.
- Kullanılabilirlik: Yazılımın kullanım kolaylığı sağlayan yetenekleri olarak tanımlanmaktadır. Öğrenebilme, anlaşılabilirlik, işletilebilirlik ve kullanıcı etkileşimi konuları bu kategori altında incelenmektedir.
- Verimlilik: Yazılımın ihtiyaç duyulan ölçüde yeterli performansla çalışabilme becerisi olarak tanımlanmaktadır. Zaman ve kaynak kullanımı konuları bu kategori altında incelenmektedir.
- Bakılabilirlik: Yazılımın değişiklik veya düzeltme isteklerine adaptasyon yeteneği olarak tanımlanmaktadır. Değiştirilebilirlik, test edilebilirlik, analiz edilebilirlik ve bağışıklık konuları bu kategori altında incelenmektedir.
- Taşınabilirlik: Yazılımın farklı çalışma ortamlarına uyum sağlayabilme yeteneği olarak tanımlanmaktadır. Adaptasyon yeteneği, yüklenebilirlik özellikleri, ortam değiştirme imkânı ve diğer yazılımlarla uyum konuları bu kategori altında incelenmektedir.

Yazılımda kaliteyi sağlamak için kodlama standartları ve yazılım metrikleri kullanılmaktadır. Bu standartlara ve metriklere yönelik verilerin üretildiği ve analiz edildiği çeşitli araçlar mevcuttur. Açık kaynak kodlu bir yazılım olarak Sonar, yazılımın kalitesini değerlendirmeye yönelik çeşitli metrikler sunmaktadır. Sonar haricinde; yazılım kalite analizinde Understand, SQuORE gibi farklı araçlar da kullanılmaktadır.

Kalite düşüklüğünü gösteren en iyi ölçüt, yazılımın testi sırasında veya canlı kullanımı sırasında bulunan hatalardır. Tez boyunca hata için şu tanımlama dikkate alınmıştır: "Bir hata bir yanlışlığın ortaya çıkmasıdır. Bir hata, karşılaşılması durumunda, sistemin çalışmasında aksaklığa sebep olabilir." (IEEE, 1990).

1.2. Yazılım Hata Kestirimi

Yazılım hata kestirimi ile amaçlanan; yazılım üretilmeden önce çeşitli metriklerin girdi olarak yer aldığı teknikler kullanarak yazılımın hatalı olup olmadığının tespitidir. Özellikle son 10 yılda bu alanda çok sayıda çalışma yapılmıştır. Yazılım hata kestirimi ile ilgili yapılan çalışmalar tezin ilerleyen bölümlerinde yer almaktadır.

1.3. Sınıflandırma

Teknolojinin ve bilgi sistemlerinin gelişmesi ile birlikte verinin işlenmesi, analiz edilmesi ve buna bağlı olarak çıkarımlar yapılması büyük önem taşımaktadır. Veri madenciliği, yüksek yoğunluklu verilerden bilginin çıkartılması işlemidir. Veri madenciliği işlemi ile veri tabanında mevcut olan veriler üzerinden ilişkilerin tespiti ve örüntülerin (pattern) çıkartımı yapılır. Veri sayısı ve ilişkilerin karmaşıklığından dolayı bu desenin veya desenlerin ortaya çıkarılması elektronik ortamda ve yardımcı algoritmalar ile mümkündür.

Veriler, sahip olunan ortak özelliklere göre belli bir sınıfı ifade ederler. Sınıflandırma, girdi olarak alınan verilere göre ait olunan kategorinin (sınıf) tespitidir.

1.4. Araştırma Soruları

Tez kapsamında üç ayrı (bağımsız) çalışma yapılmış ve yapılan çalışmalarda aşağıdaki ortak araştırma sorusuna cevaplar üretilmeye çalışılmıştır:

- Yazılım kalite metrikleri ile yazılımı oluşturan modüllerin hata kestirimi arasında ilişki kurulabilir mi?

Bu ortak araştırma sorusunun yanında üç çalışmada aşağıdaki detaylı araştırma soruları incelenmiştir:

1. Çalışma (Bölüm 2): "Maliyet-duyarlı sinir ağı kullanarak yazılım hata kestirimi":

- Yazılım hata kestirimi için maliyet-duyarlı bir öğrenme modeli geliştirilebilir mi?
- Yapay Sinir Ağı (YSA)'daki nöron bağlantılarının ağırlıklarının optimizasyonunda Yapay Arı Kolonisi Algoritması (YAK) kullanılabilir mi?

2. Çalışma (Bölüm 3): "Açık kaynak kodlu yazılımlar için eşik değerlerinin türetimi ve yazılım hata kestiriminde uygulanması":

- Metrik eşik değeri ile modülün hataya meyilliliği arasında bir ilişki var mıdır? Yani, bir modülün metrik değeri, türetilen eşik değerden daha büyükse, o modülün daha fazla hataya meyilli olduğu anlamına gelir mi?
- Türetilen eşik değerleri belli karakteristiklere sahip diğer açık kaynak kodlu yazılımlarda etkili bir şekilde kullanılabilir mi?
- Yazılım modüllerinin üç risk seviyesine bölünmesinin kestirim performansı üzerinde etkisi var mıdır?

3. Çalışma (Bölüm 4): Özellik bağımlı Naive Bayes yaklaşımı ve yazılım hata kestiriminde uygulanması:

- Veri önişleme adımlarının Naive Bayes'in yazılım hata kestirimine etkisi nasıldır?
- Naive Bayes'in özellik bağımsızlığı azaltıldığında yazılım hata kestirimine olumlu katkısı var mıdır?

1.5. Kullanılan Algoritmalar

Tez kapsamında üç ayrı çalışma yapılmış ve her bir çalışmada aşağıdaki makine öğrenmesi algoritmaları kullanılmıştır:

1. Çalışma (Bölüm 2): Maliyet Duyarlı Sinir Ağı Kullanarak Yazılım Hata Kestirimi:
   - Yapay Sinir Ağı
   - Yapay Arı Kolonisi

2. Çalışma (Bölüm 3): Açık kaynak kodlu yazılımlar için eşik değerlerinin türetimi ve yazılım hata kestiriminde uygulanması:
   - Lojistik Regresyon
   - Bender Metot
   - Yapay Arı Kolonisi

3. Çalışma (Bölüm 4): Özellik bağımlı Naive Bayes yaklaşımı ve yazılım hata kestiriminde uygulanması:
   - Naive Bayes

1.6. Kullanılan Veri Setleri

Tez kapsamında üç ayrı çalışma yapılmış ve her bir çalışmada aşağıdaki veri setleri kullanılmıştır. Bu veri setini oluşturan yazılım projelerinin hangi programlama dili ile kodlandığı da aşağıda yer almaktadır.

1. Çalışma (Bölüm 2): Maliyet Duyarlı Sinir Ağı Kullanarak Yazılım Hata Kestirimi:
   - NASA: KC1, KC2, CM1, PC1, JM1 – C, C++

2. Çalışma (Bölüm 3): Açık kaynak kodlu yazılımlar için eşik değerlerinin türetimi ve yazılım hata kestiriminde uygulanması:
   - PROMISE: ant, camel, jedit, log4j, lucene, poi, synapse, velocity, xalan, xerces – Java

3. Çalışma (Bölüm 4): Özellik bağımlı Naive Bayes yaklaşımı ve yazılım hata kestiriminde uygulanması:

- NASA: CM1, PC1, PC2, PC3, PC4, K3, KC4, MW1 – C, Java, Perl

1.7. Yayın Listesi

Bu tez boyunca üç ayrı çalışma yapılmış; bunlardan ikisi uluslararası bilimsel dergide yayınlanmış; biri ise değerlendirme aşamasındadır. Yayın listesi aşağıdadır:

- Bölüm 2: Arar, Ömer Faruk ve Ayan, Kürşat. "Software defect prediction using cost-sensitive neural network". Elsevier Applied Soft Computing 33 (2015)
- Bölüm 3: Arar, Ömer Faruk and Ayan, Kürşat. "Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies". Elsevier Expert Systems with Applications 61 (2016): 106-121.
- Bölüm 4: Arar, Ömer Faruk ve Ayan, Kürşat. “Feature dependent naive bayes Approach and its application to software defect prediction”. Elsevier Engineering Applications of Artificial Intelligence (değerlendirme aşamasında)

Tez kapsamında bu üç çalışma, detayları ile anlatılacaktır.

1.8. Tezin Amacı

Bu tez kapsamında çeşitli makine öğrenme algoritmaları ve veri önişleme adımları kullanılarak hata kestirim modelleri oluşturulmuştur. Bu modellerde girdi olarak yazılım kalite metrikleri kullanılmış ve çıktı olarak ise ilgili yazılım modülünün hataya meyilli olup olmadığı bilgisi üretilmiştir. Öğrenme modellerinin kurulmasında girdi ve çıktıların belli olduğu eğitim verileri kullanılmıştır. Modellerin validasyonu çıktıların belli olmadığı, sadece girdilerin mevcut olduğu test verileri kullanılarak yapılmıştır. Bu durum, oluşturulan modellerin gözetimli öğrenme olduğu anlamına gelmektedir. Öğrenme modellerinin kodlanmasında MATLAB ve Eclipse platformları kullanılmıştır.

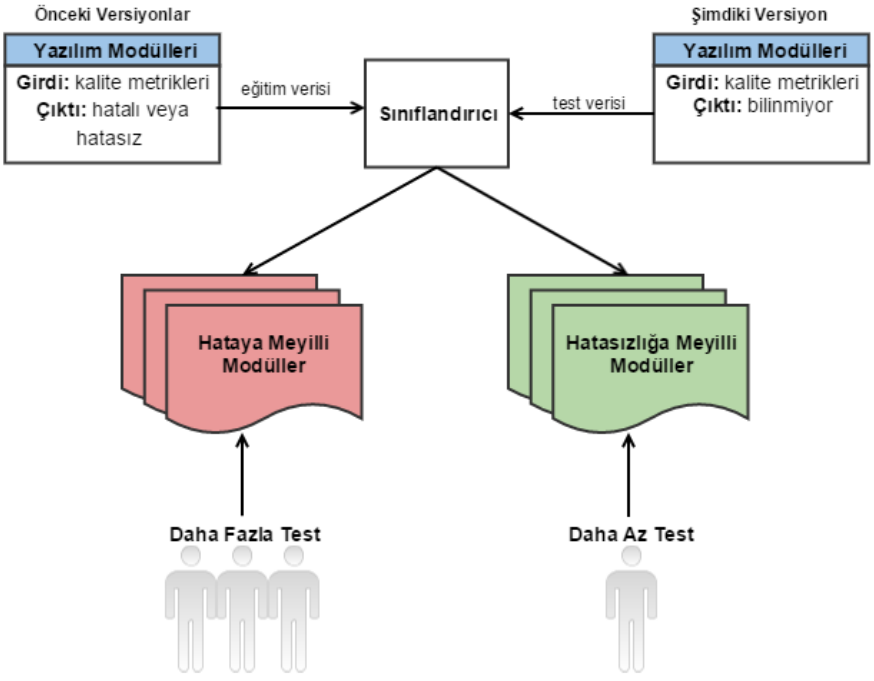
BÖLÜM 2. MALİYET-DUYARLI SİNİR AĞI KULLANARAK YAZILIM HATA KESTİRİMİ

2.1. Giriş

Yayınlanan bir rapora göre 2015 yılında dünya genelinde yazılıma toplamda 3,5 trilyon dolar harcanmıştır (Woods ve Meulen, 2016). Bir anket çalışması ise yazılım projelerindeki harcamanın, 2013 yılında %23'ünün ve 2015 yılında %35'inin kalite güvence ve test faaliyetlerine ayrıldığını göstermiştir (CapGemini, 2016). Küçük bir veri dönüştürme hatasından dolayı 125 milyon dolarlık NASA uzay aracının uzayda kaybolması, testin önemini ortaya koyan yaygın olarak bilinen bir örnektir (Michaels, 2008). Amerika Birleşik Devletleri Savunma Bakanlığı, yazılım hatalarından kaynaklı her yıl 4 milyar dolarlık harcama yapmaktadır (Dick ve ark., 2004). Bütün bu sayılar, yazılım kalite güvence ve testin önemini göstermekte; bu faaliyetlerin dikkatli ve etkili yürütülmesine ihtiyacı ortaya çıkarmaktadır. Bir yazılım hatasını gerçek ortamda kullanımdayken çözmek, geliştirme aşamasındayken tespit edip çözmekten 100 kat daha maliyetlidir (Pelayo ve Dick, 2007). Bütün bu gerçeklere rağmen, yazılım şirketlerinin sadece %30'u, projelerinde kalite güvence ve test faaliyetleri için yeterli bütçe ayırmaktadır (Michaels, 2008).

Yazılım sistemlerinin karmaşıklığı ve kullanıcıların beklentilerinin artması ile birlikte yazılım kalite disiplini ortaya çıkmıştır. ISO/IEC 9126, yazılım kalitesini değerlendiren uluslararası bir standarttır. Bu standarda göre yazılım ürünlerinin kalite karakteristikleri, dahili (internal) ve harici (external) metriklerden elde edilir. Standartta yer alan altı kalite karakteristiği şunlardır: fonksiyonellik (functionality), güvenilirlik (reliability), kullanılabilirlik (usability), etkinlik (efficiency), bakım yapılabilirlik (maintainability) ve taşınabilirlik (portability). Dahili metrikler, ürünün çalışma davranışına bakılmaksızın, kendi kaynak kodları dikkate alınarak ölçülür.

Diğer taraftan, harici metrikler ise ürünün davranışı ve çalışma şekliyle ilgilidir. Bu çalışmanın odağı dahili metriklerdir; yani, yazılım sisteminin kaynak kodlarıdır (davranış ve fonksiyonelliği değil). Yazılım hata kestirim faaliyetleri, yazılım kalite disiplini altında ele alınabilir. Daha kaliteli bir yazılım sistemi, daha az hataya meyilli yazılım demektir. Yazılım hata kestirim modelleri, yirmi yıldan fazla bir süredir araştırmacılar tarafından hatırı sayılır bir ilgi ile karşılanmıştır. Bu modeller, hataya meyilli yazılım modüllerinin, manuel teste geçmeden önce otomatik bir şekilde tespit edilmesini sağlar. Böylelikle, proje test ekibi, modelin çıktılarını baz alarak personel, zaman ve bütçelerini daha etkin kullanabilir. Hataya meyilli modüller, hatasızlığa meyilli modüllere göre daha fazla odak gerektirir. Hata kestirim modellerinin test faaliyetlerindeki genel rolü Şekil 2.1.'de gösterilmiştir. Modüllerin sınıflandırması, bir sonraki bölümlerde anlatılacak olan yazılım kalite metriklerine göre yapılır. Yazılım kalite metrikleri, modülün hataya meyillilik durumu ile ilgili bilgiler içerir (Basili ve ark., 1996). Bu alanda yayınlanmış araştırma makaleleri bu bağlantıyı ortaya koyan kanıtlardır.



Şekil 2.1. Test aktivitelerinde yazılım hata sınıflandırıcıların rolü

Birçok makine öğrenmesi algoritması, her bir sınıfın yanlış sınıflandırma maliyetinin eşit öneme sahip olduğunu varsayar. Ancak, genellikle azınlık (minority) sınıfın yanlış sınıflandırma maliyeti, çoğunluk (majority) sınıfın yanlış sınıflandırma maliyetinden daha yüksektir. Örneğin; hatalı bir modülün, hatasızlığa meyilli (hsm) modül olarak tahmin edilmesi daha pahalı onarma aktivitelerine sebep olabilir; çünkü, ilgili yazılım bu hatayı içerecek şekilde canlı sistemde kullanılmak üzere yayınlanmış olabilir. Diğer taraftan; hatasız bir modülün, hataya meyilli (hm) modül olarak tahmin edilmesi, projede gereksiz test faaliyetlerinin yapılmasına sebep olabilir; bu durum, genellikle bir önceki durumdan daha kabul edilebilirdir. Maliyet-duyarlı sınıflandırıcılar, bu iki durumu da dikkate alır ve toplam yanlış sınıflandırma maliyetini minimize etmeye çalışırlar (toplam yanlış sınıflandırma sayısını değil).

Bu bölümde yer alan çalışma ile aşağıdaki literatür katkıları yapılmıştır:

- "Yapay Arı Kolonisi (YAK) ile optimize edilmiş Yapay Sinir Ağı (YSA)" ile kurulmuş hibrit model, yazılım hata kestirimi (tahmini) alanına uygulanmıştır.
- Parametrik uygunluk (fitnes) fonksiyonu kullanılarak YSA'ya maliyet-duyarlılık özelliği kazandırılmıştır. Maliyet katsayılarının değiştirilmesi yoluyla, azınlık ve çoğunluk sınıfının, sınıflandırma performansı arasında seçim (trade-off) yapılabilir.

Önerilen hibrit modelin validasyonu için şu yöntemler kullanılmıştır:

- Yaygın olarak kullanılan; Alıcı İşletim Karakteristik Eğrisi (Receiver Operating Characteristics (ROC)), Eğri Altındaki Alan (Area Under Curve (AUC)), hatalı modül tespit oranı (probability of detection (pd)), yanlış alarm oranı (probability of false alarm (pf)), balance (bal) ve doğruluk (accuracy (acc)) performans sonuçları verilmiştir.
- AUC ölçütü baz alınarak diğer algoritmalarla performans karşılaştırması yapılmıştır.
- Farklı maliyet değerlerine göre pd, pf ve acc sonuçları verilmiştir.
- Farklı maliyet oranlarına göre Normalize Edilmiş Beklenen Yanlış Sınıflandırma Maliyeti (Normalized Expected Cost of Misclassification

(NECM)) sonuçları gösterilmiştir ve bu sonuçlar diğer bazı maliyet-duyarlı sinir ağları ile karşılaştırılmıştır.

Bu bölüm şu şekilde tasarlanmıştır: İlgili çalışmalar altbölüm 2.2'de özetlenmiştir. Altbölüm 2.3'de bu çalışmada kullanılan veri setleri açıklanmıştır. Altbölüm 2.4, algoritmalar ve önerilen kestirim modelini içermektedir. Performans ölçütleri ve sonuçlar altbölüm 2.5'de verilmiştir. Çalışma sonuçları altbölüm 2.6'da özetlenmiştir.

2.2. Kaynak Araştırması

Yazılım hata kestirim problemini çözmek üzere farklı istatistiki ve makine öğrenmesi algoritmaları kullanılmıştır. Random Forest (RF) (Guo ve ark., 2004), Yapay Bağışıklık Tanıma Sistemi (Artificial Immune Recognition System (AIRS)) (Catal ve Diri, 2009), Naive Bayes (NB) (Padberg ve ark., 2004; Zimmermann ve ark., 2007; Menzies ve ark., 2007), J48 (Koru ve Liu, 2005), ağaç tabanlı metotlar (Tree-based Methods) (Selby ve Porter, 1988; Khoshgoftaar ve ark., 1999; Guo ve ark., 2004), Durum bazlı çıkarım (Case-based Reasoning) (Khoshgoftaar ve Seliya, 2003), Destek Vektör Makinaları (Support Vector Machines) (Elish ve Elish, 2008), ve Lojistik Regresyon (Gyimothy ve ark., 2005; Olague ve ark., 2007) bu problemin çözümünde kullanılan bazı algoritmalardır. Menzies ve arkadaşları (2007), log filtreleme ile ön işlenmiş verinin NB'de kullanılmasının Doğru Pozitif Oranı (True Positive Rate (TPR)) ve Yanlış Pozitif Oranı (False Positive Rate (FPR)) baz alındığında en iyi sonucu verdiğini belirtmişlerdir. Malhotra (2015), 1991 ile 2013 arasında yazılım hata kestirimi amaçlı yapılan çalışmaları derlemiştir. Bu derleme çalışmasında kullanılan makine öğrenmesi ve istatistiki yöntemleri ele almış ve performanslarını karşılaştırmıştır.

Bu sınıflandırma algoritmalarının yanında, bazı optimizasyon algoritmaları da bu problem üzerinde uygulanmıştır. Bunlar: Genetik Programlama (GP) (Evett ve ark., 1998), Parçacık Sürüsü Optimizasyonu (Particle Swarm Optimization (PSO)) (Carvalho ve ark., 2010) ve Karınca Kolonisi Optimizasyonu (Ant Colony Optimization)'dur (Vandecruys ve ark., 2008). YSA da bir kaç çalışmada kullanılmış

ve performansı incelenmiştir (Neumann, 2002; Thwin ve Quah, 2005). Khoshgoftaar ve arkadaşları (1997) tarafından yapılan çalışma, yazılım kalite tahmininde sinir ağlarının kullanıldığı ilk çalışmalardan bir tanesidir. Bu çalışmada bir telekomünikasyon sistemi uygulama veri seti olarak kullanılmış ve öğrenme modelinde çıktı olarak modüllerin hm veya hsm olarak sınıflandırması yapılmıştır. Sinir ağlarının, istatistiki metotlardan daha başarılı olduğu sonucuna varmışlardır. Kanmani ve arkadaşları (2007) da nesne tabanlı yazılım kalite metriklerinin girdi olarak yer aldığı çalışmada sinir ağlarını kullanmışlardır. Sonuçlarını, iki farklı istatistiki metot ile karşılaştırmışlardır. Erturk ve Sezer (2015), Uyarlamalı Sinirsel Bulanık Çıkarsama Sistemi (Adaptive Neuro Fuzzy Inference System (ANFIS)) ile YSA, SVM yöntemlerinin performanslarını karşılaştırmışlar ve ANFIS'ın makul sonuçlar ürettiğini belirtmişlerdir.

Yazılım hataların büyük bir çoğunluğu, modüllerin küçük bir kısmında tespit edilir (Boehm ve Papaccio, 1988). Boehm, yazılım ürününde tespit edilen hataların yaklaşık %80'i modüllerin yaklaşık %20'si içerisinde bulunduğunu belirtmiştir (80:20 kuralı) (Boehm, 1987). Hatalı ve hatasız modüllerin dengesiz (unbalanced) dağılımı sınıflandırma performansını olumsuz etkilerler. Bu olumsuzluk azınlık sınıfında (hatalı modül) daha fazladır (Arisholm ve ark., 2010; Hall ve ark., 2011). Bu dağılım problemi, veri veya algoritma seviyesinde ele alınmıştır (Moser ve ark., 2008). Sınıf dağılımı dengelemek için veri seviyesinde, çeşitli örnek arttırım ve azaltım (oversampling ve undersampling) metotları uygulanmıştır (Zheng, 2010; Wang ve Yao, 2013). Rastsal örnek arttırım/azaltım ve SMOTE (Chawla ve ark., 2002) gibi bu metotlar kolay ve etkilidirler; ancak, problem uzayı ve kullanılan sınıflandırma algoritması, etkinliğini belirleyen önemli faktörlerdir (Estabrooks ve ark., 2004). Diğer taraftan, algoritma seviyesindeki metotlarda sınıf dengesizliği probleminin çözümünde, eğitim mekanizması değiştirilerek daha iyi azınlık sınıfı tespit doğruluğunu elde etme amaçlanır (Wang ve Yao, 2013). Buna iki örnek şunlardır: tek sınıf öğrenmesi (one-class learning) (Japkowicz ve ark., 1995) ve maliyet-duyarlı öğrenme algoritmaları (Wang ve Yao, 2013; Zhou ve Liu, 2006). Bu çalışmada önerilen model ile maliyet-duyarlı YSA kullanılarak sınıf dengesizliği algoritma

seviyesinde ele alınmıştır. Böylelikle, öğrenme modelinin önişlem adımı olarak örnek arttırım veya azaltımına ihtiyaç duyulmamıştır.

Bir kaç istisna dışında (Arisholm ve Briand, 2006; Zhou ve Liu, 2006) bu zamana kadar önerilen birçok hata kestirim modeli, yanlış sınıflandırma maliyetini dikkate almamışlardır. Ancak, gerçek dünya şartlarında, hatalı modülün yanlış sınıflandırılması, hatasız modülün yanlış sınıflandırmasına göre çok daha maliyetlidir. Bu yanlış sınıflandırmaların önem seviyesini belirlemek için katsayılar belirlenmiştir. Turney tarafından önerilen yöntemde maliyet-duyarlı sınıflandırma için Genetik Algoritma ile Karar Ağaçlarının hibrit kullanılmıştır (Turney, 1995). Örnek arttırım/azaltım ve eşik kaydırma (threshold moving) kullanarak sinir ağlarını maliyet-duyarlı bir hale dönüştürme çalışmaları olmuştur (Zhou ve Leung, 2006). Örnek arttırım ve örnek azaltım oranları, ilişkili maliyetlere göre belirlenmiştir. Her bir sınıftaki eğitim örneği sayısı, bağlantılı sınıfın tespit oranını etkiler. Optimal değer elde edilene kadar bir sinir ağı çıktısı için kullanılan eşik değer maliyet matrisi dikkate alınarak kaydırılır. Zhou ve Liu (2006), sinir ağını maliyet duyarlı hale dönüştürmek için eşik kaydırmanın iyi bir seçim olduğunu göstermişlerdir.

YAK, yakın zamanda sunulan bir optimizasyon algoritmasıdır (Karaboga ve Basturk, 2007). İlerleyen bölümlerde YAK algoritması detaylı bir şekilde açıklanacaktır. Nöron bağlantılarının ağırlık optimizasyonunda bu algoritma kullanılmıştır (Karaboga ve ark., 2007). Bu hibrit yaklaşım metodolojisi, UCI makine öğrenme veri setleri üzerinde sınıflandırma amacıyla uygulanmış ve bilinen diğer geleneksel ve evrimsel yöntemlerle karşılaştırılmıştır. Elde edilen sonuçlar, YAK algoritmasının, etkin bir şekilde sınıflandırma amacıyla, sinir ağlarının eğitilmesinde kullanılabileceğini göstermiştir (Karaboga ve Ozturk, 2009). Bu çalışmada, bu hibrit metodoloji baz alınmak suretiyle YAK'ın fitnes fonksiyonu değiştirilerek maliyet-duyarlı sinir ağı elde edilmiş ve başarılı bir şekilde yazılım hata kestirimi problemine uygulanmıştır.

2.3. Veri Setleri

Yazılım hata kestirim probleminde, NASA MDP veri setleri çok kullanılan kıyaslama (benchmarking) verileridir (Chapman ve ark., 2004). Buradaki yazılım projeleri; uzay uçuş kontrol, yer sistemleri için veri depolama yönetimi ve uzay aracı enstrümantasyonu amacıyla geliştirilmiştir. Bu veri havuzundan en çok kullanılan beş veri seti bu çalışmada kullanılmıştır. Her bir veri seti, kalite metrikleri girdi olmak üzere birçok yazılım modülünü içermektedir. Her bir modül, herhangi bir hata tespit edilmesi durumuna bakılarak atanan, hatalı veya hatasız çıktı etiketini içermektedir. Etiketleme işlemi, test aşamasından sonra manuel olarak yapılır. Tablo 2.1., bu çalışmada kullanılan beş veri setine ait karakteristikleri göstermektedir. Hatalı modüllerin %6.94 ile %20.49 arasında değişen yüzdesi, verilerin dengesiz dağılımda olduğunu göstermektedir. Bu NASA projeleri C/C++ dili kullanılarak geliştirilmiştir.

Tablo 2.1. Veri setlerinin temel özellikleri

Adı	Dil	Modül (#)	Hatasız (#)	Hatalı (#)	Hata Oranı (%)
KC1	C++	2109	1783	326	15,45
KC2	C++	522	415	107	20,49
CM1	C	505	449	49	9,83
PC1	C	1109	1032	77	6,94
JM1	C	10885	8779	2106	19,35

NASA veri setleri, McCabe (McCabe, 1976) ve (temel ve türetilmiş) Halstead (Halstead, 1977)'den 21 metot-seviyeli metrik bilgisini içerir. Bazı araştırmacılar, türetilmiş Halstead metriklerinin yazılım hata kestirimine bir katkısının olmadığı yönünde görüş sunmuşlar ve genellikle bunları çalışmalarında kullanmamışlardır. McCabe ve Halstead metrikleri, yazılım kalitesinin analiz edilmesinde yaygın olarak kullanılmaktadır. Bir modül; yordamsal (procedural) dillerde (C) prosedür/metot/fonksiyon olarak; nesne tabanlı dillerde (C++, Java) ise sınıf olarak temsil edilirler. McCabe metrikleri, modüldeki akış yollarını saymak suretiyle karmaşıklığı hakkında bilgiyi sunarlar. Halstead metrikleri ise modüldeki operatör ve işlenenleri hesaplamak suretiyle okunabilirliği hakkında bilgiyi sunarlar.

Karmaşıklığı fazla olan veya okunabilirliği az olan bir modülün yüksek olasılıkla hataya meyilli olduğu kabul edilir (Wang ve Yao, 2013). Her bir modül bazında toplanan bu 21 metot-seviyeli metrik Tablo 2.2'de özetlenmiştir.

Tablo 2.2. Çalışmada kullanılan metot-seviyeli metrikler

Tür	Metrik Kısaltması	Metrik Tanımı
McCabe	loc	Toplam kod satır sayısı (Total line of code)
	v(g)	Çevrimsel karmaşıklık (Cyclomatic complexity)
	ev(g)	Esas karmaşıklık (Essential complexity)
	iv(g)	Tasarım karmaşıklığı (Design complexity)
Türetilmiş Halstead	n	Toplam operatör ve işlenen sayısı (Total number of operators and operands)
	v	Hacim (Volume)
	l	Program uzunluğu = (v/n) (Program length)
	d	Zorluk = (1/l) (Difficulty)
	i	Zeka (Intelligence)
	e	Programı yazmak için efor = (v/l) (Effort to write program)
	b	Efor tahmini (Effort estimate)
	t	Zaman tahmincisi = E/18 saniye (Time estimator)
Temel Halstead	lOCode	Komut satır sayısı (Count of statement lines)
	lOComment	Yorum satır sayısı (Count of comment lines)
	lOBlank	Boş satır sayısı (Count of blank lines)
	lOCodeAndComment	Kod ve yorum satır sayısı (Count of code and comment lines)
	uniqOp	Özgün operatör sayısı (Number of unique operators)
	uniqOpnd	Özgün işlenen sayısı (Number of unique operands)
	totalOp	Toplam operatör sayısı
	totalOpnd	Toplam işlenen sayısı
	branchCount	Toplam dal sayısı

Bu çalışmada veri setindeki bütün metriklerin kullanımı yerine, özelliklerin seçilen bir alt kümesi girdi olarak kullanılmıştır. Bunun için, WEKA yazılımı (Hall ve ark., 2009) kullanılarak Korelasyon-tabanlı Özellik Seçimi (Correlation-based Feature Selection (CFS)) (Hall, 1999) tekniği uygulanmıştır. Aynı özellik seçim tekniği, farklı araştırmacıların çalışmalarında da yer almıştır (Elish ve Elish, 2008; Catal ve Diri, 2009). CFS'nin beş veri setinde uygulanmasından sonra seçilen özellikler (metrikler) Tablo 2.3'de gösterilmiştir. Menzies ve arkadaşları (2007) özellik seçim yönteminden ziyade sınıflandırıcı seçiminin önemini vurgulamışlardır, bu sebeple, özellik seçim tekniği bu çalışmanın ana odağı değildir. Ayrıca, Temel Bileşenler Analizi (Principal Component Analysis (PCA)) ve Bilgi Elde Etmeye Dayalı Özellik Değerlendirmesi (Info Gain Attribute Evaluation) gibi bazı diğer özellik seçim yöntemleri de deneylerde kullanılmış, ancak en iyi sonuçların CFS ile elde edildiği görülmüştür.

Tablo 2.3. Her bir veri seti için seçilen metrikler (Korelasyon-tabanlı Özellik Seçimi sonrasında)

Veri Seti	Metrik (#)	Seçilen Metrikler
KC1	8	v, d, i, lOCode, lOComment, lOBlank, uniqOpnd, branchCount
KC2	3	ev(g), b, uniqOpnd
CM1	7	loc, iv(g), i, lOComment, lOBlank, uniqOp, uniqOpnd
PC1	6	v(g), I, lOComment, lOCodeAndComment, lOBlank, uniqOpnd
JM1	8	loc, v(g), ev(g), iv(g), i, lOComment, lOBlank, lOCodeAndComment

Diğer taraftan; önerilen modelin CFS tekniği kullanılması durumundaki performansının, tüm 21 metrik kullanılması durumundaki performansına göre daha iyi olduğu görülmüştür. CFS'de arama parametresi olarak iki yönlü (bi-directional) arama seçilmiştir.

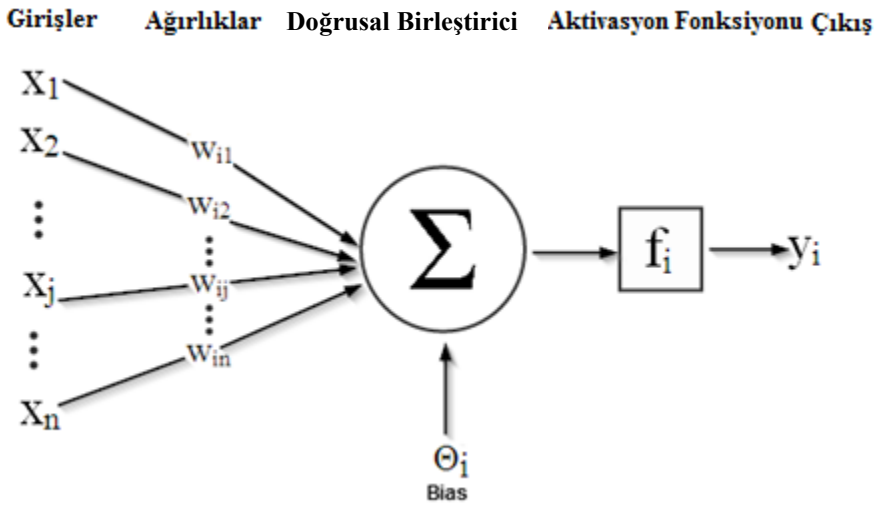
2.4. Önerilen Sistem

2.4.1. Yapay sinir ağı

Bir sinir ağı, insan beyninden ilham alınarak geliştirilmiş muhakeme tabanlı bir modeldir (Yao, 1993). YSA, nöron olarak adlandırılan temel bilgi işleme ünitelerinden

oluşur. Ağ, üç katmandan oluşur: bir giriş katmanı, en az bir gizli katman ve bir çıkış katmanı. Giriş verisi (sinyali), katmandan-katmana prensibine göre ileri yönlü olarak iletilir. Nöronlar arası bağlantılar, ağırlıklarla temsil edilirler. Bu ağırlıklar, ilgili nöronun önemini (veya gücünü) yansıtırlar. Bir sinir ağının öğrenme işlemi, tekrarlayan ağırlık düzenlemeleri ile yapılır.

Her bir nöron, önceki katmandaki nöronlardan gelen girdileri alır ve bu girdiler ilgili ağırlık değerleri ile çarpılır. Sonrasında, bu çarpımların sonuçları toplanır. Bir nöron, yeni bir aktivasyon seviyesini bu toplamdan hesaplar ve çıktı olarak bunu bir sonraki katmana iletir. Bir çıktı sinyali, ağın nihai çözümü veya başka bir nöronun girdisi olabilir. Genel olarak bir nöron bileşenleriyle birlikte Şekil 2.2.'de gösterilmiştir.



Şekil 2.2. Bileşenleri ile birlikte bir nöron

Bir aktivasyon fonksiyonu adım (step), işaret (sign), sigmoid veya doğrusal fonksiyon olabilir. Aktivasyon fonksiyonunun seçimi ağdan beklenen göreve (sınıflandırma veya regresyon) göre yapılır. Bir nöronun çıktısı aşağıdaki (2.1) denklemi ile hesaplanır.

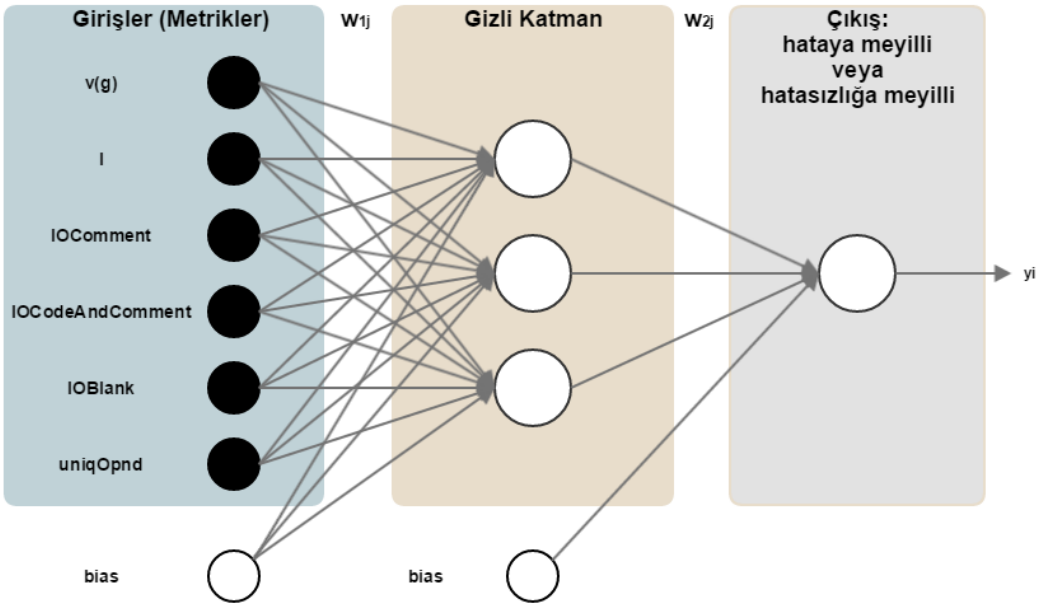
$$y_i = f_i\left(\sum_{j=1}^{n} w_{ij}x_j + \theta_i\right) \tag{2.1}$$

burada; y_i, nöronun çıktısını; n, bu nörona olan toplam girdi sayısını; x_j, j no'lu girişi, w_{ij}, bu nöron ile j no'lu giriş arasındaki ağırlığı ve θ_i ise nörünün bias değerini

belirtmektedir. f_i, bu katmanın aktivasyon fonksiyonunu temsil etmektedir. Aktivasyon fonksiyonu genellikle sigmoid ve gauss gibi doğrusal olmayan fonksiyonlardır. Bu durum YSA'nın doğrusal olmayan ilişkileri modellemesini sağlar. Yazılım kalite metrikleri ile modüllerin hataya meyillilik durumları arasındaki bağlantı doğrusal değildir ve karmaşıktır, bu nedenle YSA yazılım hata kestirimi problemi için uygun bir seçimdir.

Ağın optimizasyon hedefi; ağdaki bütün ağırlık değerlerini (bütün w_{ij}'ler) optimize etmek suretiyle, hata fonksiyonunu minimize etmektir. Her bir iterasyondaki ağ hatası farklı ölçütlerle hesaplanır. Bu ölçütlerden bazıları şunlardır: kök ortalama kareli hata (root mean squared error), ortalama mutlak hata (mean absolute error), göreli mutlak hata (relative absolute error) ve kök göreli kareli hata (root relative squared error). Bu hata ağ üzerinde geriye doğru iletilir ve hatayı minimize etmek üzere ağırlıklar düzenlenir. Bu iterasyonlar bir durma kriteri sağlanana kadar devam eder. Maksimum iterasyon sayısı veya minimum hata değeri durma kriteri olarak kullanılmaktadır.

PC1 veri seti için oluşturulan sinir ağı Şekil 2.3.'de gösterilmiştir.



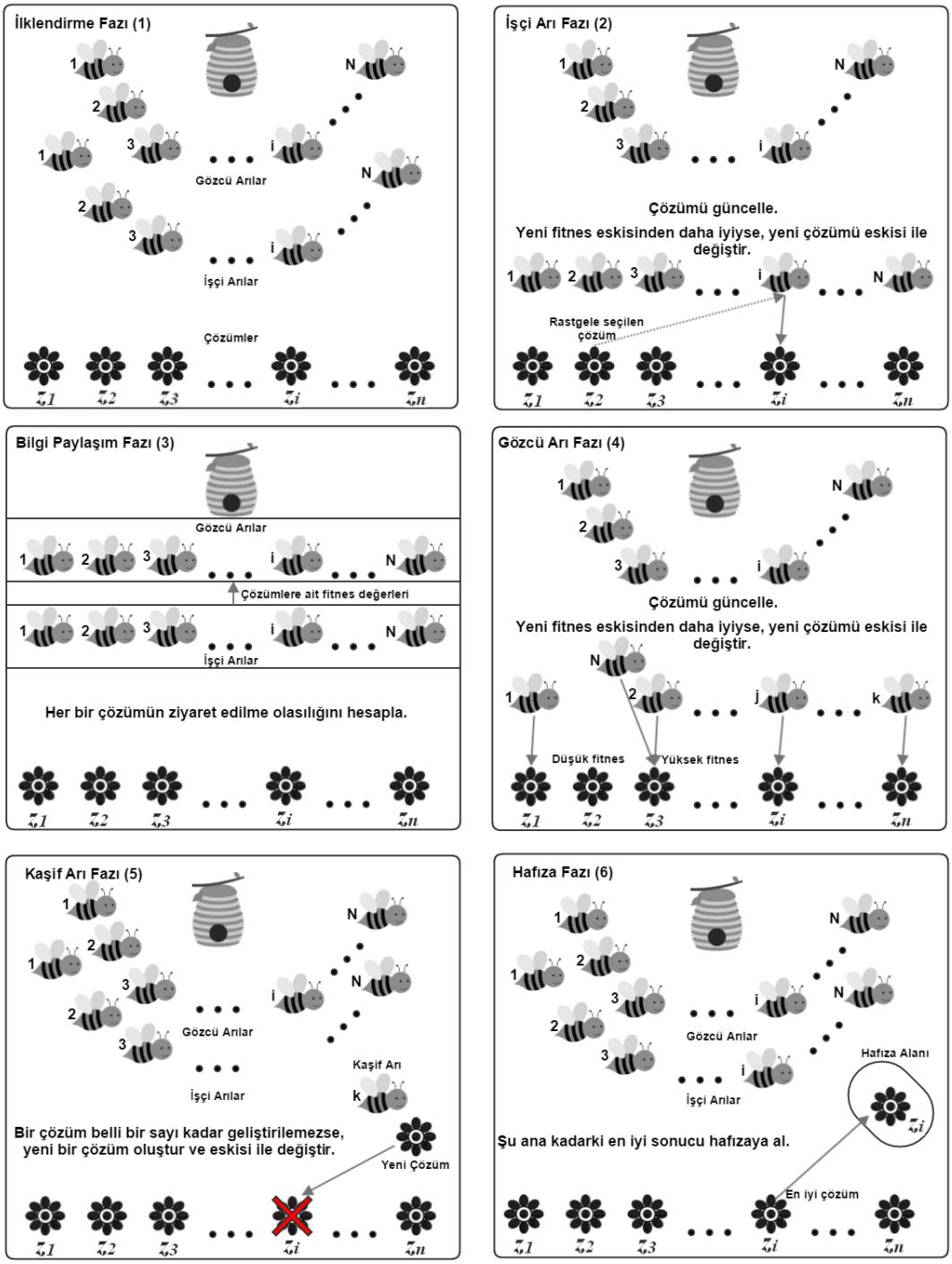
Şekil 2.3. PC1 veri seti için sinir ağı yapısı

Bu çalışmada ağırlık güncellemeleri YAK algoritması ile yapılmıştır ve maliyet-duyarlılığı sağlayan özel bir hata fonksiyonu oluşturulmuştur. Öğrenme modeli, ileri beslemeli ağ mimarisine sahip ve gözetimli öğrenme metodu ile eğitilen bir yapıya sahiptir.

2.4.2. Yapay arı kolonisi

YAK algoritması, bal arılarının yiyecek arama davranışını simüle eder. Karaboga tarafından sayısal optimizasyon problemlerinin çözümü için önerilmiştir (Karaboga ve Basturk, 2007). Bu algoritma aynı zamanda gözetimli (supervised) (Karaboga ve Ozturk, 2009) ve gözetimsiz (unsupervised) (Karaboga ve Ozturk, 2011) öğrenme amacıyla da kullanılır. YAK algoritması, Genetik Algoritma, Diferansiyal Evrim (Differential Evolution) ve PSO gibi çok bilinen diğer popülasyon tabanlı algoritmalarla karşılaştırıldığında rekabetçi sonuçlar vermiştir (Karaboga ve Akay, 2009). Güncel birçok çalışmada, YAK algoritması destekli YSA başarıyla kullanılmış ve dereceli alçalma (gradient descent) ve Levenberg-Mardquardt gibi geleneksel geri beslemeli algoritmalara göre daha iyi sonuçlar vermiştir (Karaboga ve ark., 2007).

Sürü tabanlı YAK optimizasyon algoritmasında bulunan üç tip arı popülasyonu (işçi, gözcü ve kaşif), en iyi nektar kaynağı (çözüm) için yiyecek kaynaklarını (çözüm adayları) kullanır. İşçi arılar, nektar miktarını (fitnes) arttırmak için yiyecek kaynaklarını ziyaret ederler; gözcü arılar, yiyecek kaynakları hakkında bilgi almak için dans alanında işçi arıları beklerler. Belli bir sayıda girişimden sonra bir yiyecek kaynağındaki nektar miktarı arttırılamazsa, sorumlu işçi arı bu yiyecek kaynağını terk eder ve kaşif arıya dönüşür. YAK algoritmasının gösterimi Şekil 2.4.'de verilmiştir. Önceki çalışmalarda, YAK algoritmasının açıklaması akış şeması, sözde kod (pseudo-code) veya normal metin olarak yapılmıştır; bu çalışma ile ilk defa grafiksel anlatım kullanılmıştır. Bu kullanım türü algoritmanın çalışma mekanizmasının anlaşılmasını kolaylaştırmaktadır.



Şekil 2.4. YAK algoritmasının fazlar halinde çalışma mekanizması

Faz 1: Her bir yiyecek kaynağı için sadece bir işçi arı atanır. İşçi arı ile gözcü arı sayıları eşittir. *N* adet çözüm, rastsal olarak arılar için oluşturulur. Her bir çözüm z_i (*i*=1,2,...,*N*), *D*-boyutlu bir vektör içerir; *D*, problem uzayındaki optimize edilmesi amaçlanan parametre sayısını temsil etmektedir. Faz 1'den sonra, çözümleri

geliştirmek ve en iyi çözüme ulaşmak için İşçi Arı Fazı (Faz 2), Bilgi Paylaşım Fazı (Faz 3), Gözcü Arı Fazı (Faz 4), Kaşif Arı Fazı (Faz 5) ve Hafıza Fazı (Faz 6) Maksimum Devir Sayısı (MDS) kadar çalıştırılır.

Faz 2: Bir işçi arı, kendisine karşılık gelen yiyecek kaynağında (çözüm), rastsal olarak seçilen bir komşunun parametre değerini dikkate alarak modifikasyonlar yapar. Yeni bir parametre değeri (z'_{ij}) denklem (2.2)'deki gibi oluşturulur.

$$z'_{ij} = z_{ij} + \emptyset_{ij}(z_{ij} - z_{kj}) \tag{2.2}$$

burada; $j\epsilon\{1,2,\dots,D\}$, rastsal olarak seçilen parametre numarasını ve $k\epsilon\{1,2,\dots,N\}$, rastsal olarak seçilen mevcut çözümden (i) farklı olan başka bir çözümün numarasını gösterir. Örneğin, z_{kj}, k numaralı çözümün j numaralı parametre değerini temsil eder. $\emptyset_{ij}$, -1 ile +1 arasında gerçel (real) bir değerdir. Yeni nektar miktarı (fitnes değeri) değişen parametre değeri kullanılarak hesaplanır. Eğer yeni fitnes değeri eskisinden daha iyi ise eskisi unutulur ve yenisi hafızaya alınır. Aksi takdirde, eski parametre değeri değiştirilmez ve Başarısız İyileştirme Denemesi (BID) bir arttırılır. Başka bir deyişle, çözüm adayı ile eski çözüm arasındaki seçim işleminde, açgözlü (greedy) bir seçim yaklaşımı uygulanır. Denklem (2.2)'de de görüldüğü üzere z_{kj} parametresi z_{ij} parametresine yaklaştıkça, z_i çözümündeki düzensizlik de azalır. Bu sebeple, arama optimum çözüme yaklaştıkça, eş zamanlı olarak adım büyüklüğü de azalır (Karaboga ve Ozturk, 2009).

Faz 3: Faz 2'nin tamamlanmasından sonra işçi arılar, yiyecek kaynaklarındaki nektar miktarını (çözümlerin fitnes değerini) gözcü arılarla paylaşmak üzere dans alanına geri dönerler. Bu fazda gözcü arılar tarafından kaynakların ziyaret edilme olasılığı denklem (2.3)'de gösterilen formül ile hesaplanır.

$$p_i = \frac{fit_i}{\sum_{n=1}^{N} fit_n} \tag{2.3}$$

burada; fit_i , i çözümünün fitnes değeridir.

Faz 4: Gözcü arılar, önceki fazda hesaplanan yiyecek kaynaklarının olasılık değerlerine göre bu kaynakları ziyaret ederler. Düşük fitnes değerine sahip yiyecek kaynakları gözcü arılar tarafından hiç ziyaret edilmeyebilir; diğer taraftan, yüksek fitnes değerine sahip yiyecek kaynakları ise birden fazla arı tarafından ziyaret edilebilir. Gözcü arılar, Faz 2'deki modifikasyonun aynısını uygularlar (Denklem (2.2)).

Faz 5: YAK algoritmasının bir diğer parametresi, yiyecek kaynağını ne zaman terk edeceğini belirleyen limit değeridir. Eğer, BID değeri önceden tanımlanmış bir limit değerine ulaşırsa, kaşif arı bu yiyecek kaynağı ile değiştirilmek üzere yeni bir yiyecek kaynağı oluşturur. Yeni çözüm denklem (2.4)'e göre oluşturulur.

$$z_i^j = z_{min}^j + rand(0,1)(z_{max}^j - z_{min}^j) \tag{2.4}$$

burada; terkedilen kaynak, z_i ve z_{max}^j ve z_{min}^j ise oluşturulacak olan parametrelerin üst ve alt sınırlarını ifade etmektedir. Bu alt ve üst sınır değerleri, parametre güncellemelerinin yapıldığı Faz 2 ve Faz 4'de de kullanılmaktadır.

Faz 6: En iyi fitnes değerine sahip çözüm hafızaya alınır.

Yukarıdaki açıklamalardan anlaşılacağı üzere, orijinal YAK algoritması üç kontrol parametresine sahiptir: koloni büyüklüğü (2x*N*), terk etme limiti ve Maksimum Devir Sayısı (MDS). Bullinaria ve ark (2014), koloni büyüklüğü ve terk etme limitinin elde edilen sonuçlar üzerinde küçük bir etkisinin olduğunu belirtmişlerdir. Diğer taraftan, MDS ve arama uzayının alt/üst sınır değerlerinin performansa etkisinin büyük olduğunu belirtmişlerdir (Bullinaria ve AlYahya, 2014). Karaboga ve Ozturk (2009), bütün veri setlerinde, koloni büyüklüğü olarak 30, terk etme limiti olarak 1000 ve alt/üst sınır değerleri olarak da [-2, +2] kullanmışlardır. Fakat her bir veri seti için bu parametrelerin optimize edilmesi ile daha iyi sonuçlar elde edilebilmekte ve alt/üst sınır değerlerinin tamamen kaldırılmasının da performansa etkisi büyük olabilmektedir (Karaboga ve Akay, 2009). Bu araştırmacıların görüşlerini dikkate

alarak, bu çalışmada da YAK algoritmasının kontrol parametreleri (özellikle alt/üst sınır değerleri ve MDS) her bir NASA veri setine göre optimize edilmeye çalışıldı.

2.4.3. Önerilen sınıflandırıcı modeli

Bu çalışmada yazılım hata kestirim problemi için YAK algoritması destekli maliyet-duyarlı bir YSA önerilmiştir. YSA'yı maliyet-duyarlı bir öğreniciye dönüştürmek için farklı bir hata fonksiyonu kullanılmıştır. Beklenen Yanlış Sınıflandırma Maliyeti (Expected Cost of Misclassification (NECM)) (Johnson ve Wichern, 1992), maliyetleri ve hatalı modül oranını dikkate alan tekil bir formüldür (Denklem (2.5)). C_{FP}, Yanlış Pozitif (FP) hatası (hatasız bir modülü, yanlış olarak hataya meyilli şeklinde sınıflandırma) ile ilgili maliyeti temsil etmektedir; C_{FN} ise Yanlış Negatif (FN) hatası (hatalı bir modülü, yanlış olarak hatasızlığa meyilli şeklinde sınıflandırma) ile ilgili maliyeti temsil etmektedir. P_{hsm} ve P_{hm} sırasıyla eldeki eğitim verilerindeki hsm ve hm modül yüzdelerini göstermektedir. FPR ve FNR'ya ait yanlış sınıflandırma maliyetlerini ayrı ayrı belirlemek pratikte kolay değildir. Denklem (2.6)'da gösterilen ECM formülü, C_{FP}'ye göre normalize edilir ve Normalize Edilmiş Yanlış Sınıflandırma Maliyeti (Normalized Expected Cost of Misclassification (NECM)) elde edilmiş olur (Khoshgoftaar ve Seliya, 2004). Yazılım ekipleri tarafından maliyet oranının belirlenmesi, ayrık olarak maliyetlerin belirlenmesinden daha kolaydır. Bu çalışmada kullanılan YAK algoritmasının hedefi NECM değerini minimize etmektir.

$$\text{NECM} = \text{FPR} \times P_{hsm} + \frac{C_{FN}}{C_{FP}} \times \text{FNR} \times P_{hm} \tag{2.5}$$

$$\text{ECM} = C_{FP} \times \text{FPR} \times P_{hsm} + C_{FN} \times \text{FNR} \times P_{hm} \tag{2.6}$$

Algoritmanın ana fonksiyonunun sözde kodu Şekil 2.5.'de verilmiştir. *RunABC*; *EmployedBeesFly*, *OnlookerBeesFly* ve *ScoutBeeFly* gibi diğer fonksiyonları çağıran ana fonksiyondur. Veri setinin yüklenmesinin yanı sıra YAK algoritmasının değişken tanımlamaları 1 ile 7. satır arasında yapılır. Veri seti, çapraz-validasyonda kullanılmak üzere eğitim ve test olarak ikiye bölünür. Çapraz-validasyonun detayları sonraki bölümlerde verilmiştir. Yazılım metriklerinin değer aralıkları çok farklı olduğundan

bir normalizasyon önişlemine ihtiyaç duyulur. Menzies ve arkadaşları (2007), log-filtreleme neticesinde ön işlenmiş veri ile iyi sonuçlar elde etmişlerdir. Ancak, bu çalışma kapsamında yapılan deneylerde, min-max (0-1) normalizasyonun, log-filtrelemeye göre daha iyi sonuçlar verdiği görülmüştür. Bazı çalışmalarda, normalizasyon, eğitim ve test verilerine ayrı olarak uygulanmaktansa bütün veri setine uygulanmaktadır. Bu durum, sınıflandırıcı performansını etkileyen yanlış bir kullanımdır. Bu çalışmada her bir metriğe ait minimum ve maksimum değerler eğitim veri setinden elde edilir ve test setindeki her bir örnek bu değer baz alınarak normalize edilir. 8 ile 17. satırlar arasında; sinir ağının optimal ağırlıklarını bulmak için bir eğitim fazı uygulanır ve algoritmanın performansı bu ağırlıklar kullanılarak test veri seti üzerinde hesaplanır (19. ve 20. satırlar).

```
function RunABC
      */ INITIALIZATION
1     initialize the variables of the ANN */ inputs, hidden neurons, output
2     initialize the variables of the ABC
3     */ solution  size (N), Foods, MCN, upper/lower bounds, limit, Fitness
      */ Foods  include weights and it is N x D size matrix
      */ Fitness  includes fitness values of each  Food
4     FIT = 0     */ Failed Improvement Trial
5     load the dataset
6     shuffle the dataset
7     split dataset as trainSet and testSet (cross-validation)
      */ TRAIN
8     repeat
9        normalize all metrics in the range [0.0, 1.0]  */ trainSet
10       initialize Foods  */  Random  weight values between upper and lower bounds
11       [Foods, Fitness] = EmployedBeesFly ()
12       prob = Fitness / sum (Fitness)
13       [Foods, Fitness] = OnlookerBeesFly ()
14       [Foods, Fitness] = ScoutBeeFly ()
15       i = find ( max(Fitness) )
16       bestSol = Foods (i)
17    until MCN times
      */ TEST
18    normalize all metrics in the range [0.0, 1.0]  */ testSet
19    [tp, tn, fp, fn] = RunANN (testSet, bestSol)
20    calculate performance results (AUC, fr, pd)
```

Şekil 2.5. Hibrit sınıflandırıcının ana fonksiyonunun sözde kodu

RunANN fonksiyonu, belirlenen ağırlıklara göre ağı koşturmak için kullanılır ve ağın hata değeri hesaplanır. Hata değeri NECM formülü ile hesaplanır (Denklem (2.6)). Kullanıcının algoritmadan beklentisine göre C_{FN} ve C_{FP}'nin maliyet oranı tanımlanır. Maliyet oranı C_{FN}'nin önemi ile doğru orantılıdır; yani, yüksek oran, C_{FN}'nin öneminin fazla olduğunu, düşük oran ise C_{FN}'nin öneminin düşük olduğunu göstermektedir. Şekil 2.6.'da *RunANN* fonksiyonunun sözde kodu verilmiştir. Ağın çıktısı (*y2*) 0.5'den büyükse, o modülün hm olduğu; aksi durumda ise hsm olduğu varsayılır.

```
function RunANN
    Input : data, sol
    Output : error, tp, tn, fp, fn

1   y = calculate output of the network with logsig activation functions using sol weights
2   y2 = round (y)
3   calculate tp, tn, fp, fn by comparing data output and y2
4   error = apply (6)
```

Şekil 2.6. Sinir ağının sözde kodu

EmployedBeeFly fonksiyonu, YAK algoritmasının Faz 2'sine karşılık gelmektedir. Bu fonksiyonun sözde kodu Şekil 2.7.'de verilmiştir.

```
function EmployedBeesFly
    Input : Foods, Fitness
    Output : Foods, Fitness

1   for i = 1 to N
2       sol = Foods(i)
3       newWeight = apply (2)
4       error = RunANN (trainSet, sol) */ sol include also newWeight
5       newFitness = 1 / error
6       if newFitness > oldFitness
7               Foods(i) = sol  */ Change current Food with newWeight value
8               FIT(i) = 0
9               Fitness(i) = newFitness
10      else */ No improvement on current Food
                */ No change in Foods and Fitness
11              FIT(i)++
12      end
13  end
```

Şekil 2.7. İşçi arı fazının sözde kodu

YAK algoritmasının Faz 3'ü, ana fonksiyonun 12. satırında kodlanmıştır.

OnlookerBeesFly fonksiyonu, YAK algoritmasının Faz 4'üne karşılık gelmektedir. Bu fonksiyonun sözde kodu Şekil 2.8.'de verilmiştir.

```
function OnlookerBeesFly
    Input : Foods, Fitness
    Output : Foods, Fitness

1   i = 0; t = 0
2   while t < N
3       if rand < prob(i)  */ rand is a real value between 0 and 1
4               t++
5               sol = Foods(i)
6               newWeight = apply (2)
7               error = RunANN (trainSet, sol) */ sol include also newWeight
8               newFitness = 1 / error
9               if newFitness > oldFitness
                        */ Change current Food with newWeight value
10                      Foods(i) = sol
11                      FIT(i) = 0
12                      Fitness(i) = newFitness
13              else */ No improvement on current Food
                        */ No change in Foods and Fitness
14                      FIT(i)++
15              end
16              i++
17      end

18      if i == N +1
19              i = 1
20      end
21  end
```

Şekil 2.8. Gözcü arı fazının sözde kodu

Faz 5'in sözde kodu (*ScoutBeeFly* fonksiyonu) Şekil 2.9.'da verilmiştir.

```
function ScoutBeeFly
    Input : Foods, Fitness
    Output : Foods, Fitness

1   i = find (max(FIT))
2   if FIT(i) > limit
3       newFood = create a Food
        */ random weight values between upper and lower bounds
4       Foods (i) = newFood
5       error = RunANN (trainSet, sol) */ sol include also newWeight
6       newFitness = 1 / error
7       Fitness(i) = newFitness
8   end
```

Şekil 2.9. Kaşif arı fazının sözde kodu

Bütün çözüm adayları içerisinden en iyisinin hafızaya alınması ana fonksiyonun 15 ve 16. satırlarında yapılır (YAK algoritmasının Faz 6'sı). YSA'nın eğitiminin tamamlanması ile birlikte elde edilen nöron bağlantılarının ağırlık değerleri ile sınıflandırıcı modelinin testi yapılır.

2.5. Deneyler ve Araştırma Bulguları

2.5.1. Performans ölçütleri

İkili sınıflandırma problemlerinde (örneğin; hm veya hsm çıktılı) tahmin modelinin performansının değerlendirilmesi genellikle Tablo 2.4.'de gösterilen hata matrisi kullanılarak yapılır. Tabloda hm modülleri pozitif (EVET) ve hsm modülleri ise negatif (HAYIR) olarak temsil edilmiştir. Matris, olası dört sonucu içermektedir: Eğer hatalı bir modül, hm olarak tahmin edildiyse, Doğru Pozitif (TP); hsm olarak tahmin edildiyse FN olarak işaretlenir. Diğer taraftan, hatasız bir modül, hsm olarak tahmin edildiyse, Doğru Negatif (TN); hm olarak tahmin edildiyse FP olarak işaretlenir. Emam ve arkadaşları, Bu dört çıktıyı kullanarak oluşturulan farklı performans ölçütlerini açıklamışlardır (El Emam ve ark., 2001). Bu çalışmada sonuçların değerlendirilmesinde şu ölçütler kullanılmıştır: hatalı modül tespit oranı (probability of detection (pd)), yanlış alarm oranı (probability of false alarms (pf)), balans (bal) ve doğruluk (accuracy (acc)). Bu ölçütlerin yanı sıra, yazılım hata kestirimi alanında ROC

ve AUC kullanımı da çok yaygındır. Bu sebeple, bu ölçütlere göre de değerlendirme yapılmış ve diğer çalışmalarla karşılaştırma yapılmıştır (Catal ve Diri, 2009; Sun ve ark., 2012; Wang ve Yao, 2013).

Tablo 2.4. Hata matrisi

	EVET (Tahmin Edilen)	**HAYIR (Tahmin Edilen)**
EVET (Gerçek)	TP (Doğru Pozitif)	FN (Yanlış Negatif)
HAYIR (Gerçek)	FP (Yanlış Pozitif)	TN (Doğru Negatif)

Duyarlılık olarak da bilinen pd değeri, doğru olarak sınıflandırılan hatalı modül yüzdesi olup denklem (2.7) ile hesaplanır. Aynı zamanda FNR'nin tamlayanıdır.

$$pd = \text{duyarlılık} = \frac{\text{TP}}{\text{TP+FN}} = 1 - \text{FNR} \tag{2.7}$$

pf değeri, yanlış olarak sınıflandırılan hatasız modül oranını belirtmekte olup formülü denklem (2.8) ile verilmiştir.

$$pf = \text{FPR} = \frac{\text{FP}}{\text{FP+TN}} \tag{2.8}$$

bal değeri, (pf, pd) noktasından ROC eğrisinde arzu edilen nokta olan (0,1)'e olan normalize edilmiş öklid uzaklığına eşit olup formülü denklem (2.9) ile verilmiştir (Jiang ve ark., 2008).

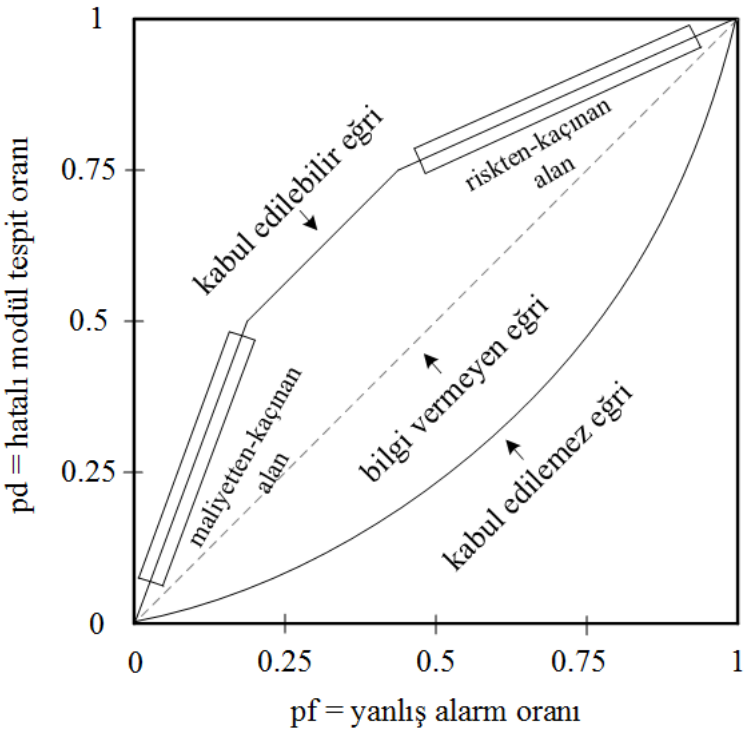
$$bal = 1 - \frac{\sqrt{(1-pd)^2+(0-pf)^2}}{\sqrt{2}} \tag{2.9}$$

acc formülü, basit bir şekilde, doğru olarak sınıflandırılan modüllerin oranı olup formülü denklem (2.10) ile gösterilmiştir.

$$acc = \frac{\text{TP+TN}}{\text{FP+FN+TP+TN}} \tag{2.10}$$

Dengesiz sınıf dağılımı gösteren veri setlerinde acc ölçütünün kullanımı uygun değildir (Nickerson ve ark., 2001). Örneğin, PC1 veri seti %6.94 hatalı modül içerir. İlkel bir sınıflandırıcı, bütün örnekleri hsm olarak tahmin etse dahi performansı %93.06 olmaktadır. Bu sebeple, bu durum dikkate alınarak bu ölçüt kullanılmamalıdır.

ROC eğrisi, Şekil 2.10.'da gösterildiği üzere *y*-ekseni pd ve *x*-ekseni pf olmak üzere 2 boyutlu bir gösterim türüdür. ROC eğrisi, bütün muhtemel değerlerin farklı eşik değerleri ile ayrılması sonucunda oluşturulur (Fawcett, 2006). Her bir ROC eğrisi, bütün örnekleri hsm olarak tahmin edildiği anlamına gelen (0, 0) ve bütün örneklerin hm olarak tahmin edildiği anlamına gelen (1, 1) noktasından geçer (Menzies ve ark., 2007). (0, 0)'dan (1,1)'e olan doğru, bilgi içermez, sınıflandırma yönüyle bir anlam ifade etmez. Grafiğin sol üst alanına denk gelen noktalar en çok kabul edilebilir olanlardır; yani, yüksek pd ile birlikte düşük pf elde edildiği anlamına gelmektedir. Dengesiz veri setlerindeki ve farklı yanlış sınıflandırma maliyetlerindeki direnci, ROC eğrilerinin avantajlarıdır (Provost ve Fawcett, 2001). Yazılım hata veri seti bu karakteristikleri sergiler, dolayısıyla bu eğri türünün bu alanda kullanımı uygundur (Khoshgoftaar ve Seliya, 2004). AUC, karşılaştırmalar yapabilmek amacıyla ROC eğrisinin altında kalan alanın hesaplanması ile türetilen sayısal bir değerdir. Dengesiz veri setlerinde AUC yaygın olarak kullanılan bir ölçüttür. Jin Huang ve Ling (2005), dengeli dağılmış ve dengesiz dağılmış veri setlerinde AUC'un, acc'ye göre istatistiki açıdan daha ayırt edici olduğunu deneysel olarak göstermişlerdir.



Şekil 2.10. Riskten-kaçınan ve maliyetten-kaçınan bölgeleri bulunduran örnek ROC eğrileri

Hata kestirim modellerinin ana hedefi, yanlış alarm üretimlerinden kaçınmanın yanında, mümkün olduğunca fazla sayıda hatalı modüllerin tespit edilmesidir (yüksek pd ve düşük pf değeri). Ancak, denklem (2.5)'deki hata fonksiyonunda C_{FN} ve C_{FP} katsayılarının değiştirilmesi ile pd ve pf performansı arasında bir seçim yapılır. Farklı işletmeler farklı hedefler tercih ederler. Yüksek pd'nin yanında, yüksek pf değeri emniyet-kritik veya riskten-kaçınan sistemler için hala pratik olarak kullanışlı olabilmektedir. Yani, bu tip sistemlerde hatasız bir modülün yanlış sınıflandırılması, hatalı modülün yanlış sınıflandırmasına göre çok daha az önemlidir ($C_{FP} << C_{FN}$) (Song ve ark., 2011). Diğer taraftan, test faaliyetleri için sınırlı zaman/bütçeye sahip olan maliyetten-kaçınan yazılım projelerindeki beklenti mümkün olduğunca düşük pf elde edilmesidir. Yani, bu tip projelerde hatasız bir modülün yanlış sınıflandırılması, hatalı modülün yanlış sınıflandırılmasından daha önemlidir ($C_{FN} < C_{FP}$). Şekil 2.10'da gösterilen ROC eğrisinde riskten-kaçınan ve maliyetten-kaçınan alanların gösterimi yapılmıştır (Menzies ve ark., 2010).

2.5.2. Deneyler

Önerilen sınıflandırıcının performansını değerlendirmek için yaygın olarak kullanılan *N*-küme çapraz validasyon (Kohavi, 1995) yöntemi uygulanmıştır. Bu yöntem ile veri seti, her biri eşit örnek sayısına sahip *N* kümeye bölünür. Algoritma *N* defa koşturulur; her bir koşumda (*N*-1) alt küme, sınıflandırıcı eğitiminin gerçekleştirilmesinde; geriye kalan tek küme ise testinin yapılmasında kullanılır. Küme sayısı olarak genellikle 10 seçilmektedir. Ancak, veri setindeki hata oranı çok düşükse, bazı alt kümeler azınlık sınıfından hiç örnek bulundurmayabilir. Bu sorunun üstesinden gelmek için *N* sayısı, Tablo 2.5.'de verildiği üzere, veri setindeki hatalı modül oranına (hmo) göre tayin edilmiştir. Bu yaklaşım ile, KC1, KC2 ve JM1 veri setinin kullanıldığı deneyler için *N*=10 olarak; CM1 ve PC1 veri setleri için ise *N*=5 olarak hesaplanmıştır. Hall ve Holmes (2003), tarafından yapılan çalışmayı temel alarak, istatistiki olarak güvenilir sonuçlar elde etmek için deneyler 10 defa tekrarlanmıştır. Bu çalışmanın sonuç kısmındaki performans ölçüt değerleri, 100 veya 50 koşumdan elde edilmiştir. Fisher ve ark., veri setindeki örneklerin sıralamasının bazı sınıflandırıcıların başarılarını etkileyebildiklerini; performanslarını önemli ölçüde arttırıp, azaltabildiğini belirtmişlerdir (Fisher, 1992). Bu sebeple, bu çalışmada algoritmanın her koşumundan önce örneklerin sıralaması rastsal olarak değiştirilmiştir.

Tablo 2.5. Hatalı modül oranına bağlı olarak küme sayıları

Koşul	Küme Sayısı (#)
hmo < 10	5
10 <= hmo < 15	7
hmo >= 15	10

YSA'nın temel yapısında, gizli katmandaki nöron sayısı ve gizli katman ile çıkış katmanındaki aktivasyon fonksiyonu bulunur. Bu değişkenler, deneme yanılma metodu ile belirlenmiştir. Gizli katman ile çıkış katmanında aktivasyon fonksiyonu olarak sigmoid fonksiyonu [0, 1] seçilmiştir. Gizli katmandaki nöron sayısı ise 3 olarak belirlenmiştir. Deneyler, 2'den 20'ye kadar farklı gizli katman nöron sayısı kullanılarak gerçekleştirilmiş ve belirlenen değer, beş veri setinde de en iyi sonucu vermiştir.

YAK algoritmasının kontrol parametreleri, otomatik bir yaklaşım kullanılarak belirlenmiştir. Her bir deney koşumunda farklı kontrol parametre seti ayarlanmış ve sonuçlar bir dosyaya kaydedilmiştir. Kontrol parametre seti otomatik olarak her iterasyonda değiştirilmiştir. Böylelikle en iyi sonucu veren parametrelerin tespiti yapılabilmiştir. Sinir ağı, doğası gereği, yavaş bir algoritmadır. Bir deneyi 100 defa tekrarlamak, YAK algoritmasının en iyi kontrol parametre setini tespit etmek ve YSA kullanımı kaynak kodun defalarca çalıştırılmasını gerektirmekte ve toplamda saatlerce zaman alabilmektedir. Bu otomatik koşum yaklaşımı ile emek ve zaman etkin bir şekilde farklı parametre kombinasyonları denenebilmektedir. Öğrenme modelinin oluşturulmasında MATLAB platformu kullanılmış olup kodlamalar bu platform üzerinde gerçekleştirilmiştir.

YAK algoritmasının kontrol parametrelerinin değerleri Tablo 2.6.'da verilmiştir. Dikkat edilirse, bu çalışmada belirlenen kontrol parametreleri orijinal YAK algoritmasından farklıdır. Bullinaria ve AlYahya (2014), UCI makine öğrenmesi veri setleri için optimal parametreleri tespit etmek suretiyle "Optimize Edilmiş YAK" elde etmişlerdir. Bu çalışmada ise yazılım hata veri setleri için "Optimize Edilmiş YAK" elde edilmiş oldu.

Tablo 2.6. YAK algoritmasının kontrol parametreleri

Parametre	KC1	KC2	CM1	PC1	JM1
Koloni Büyüklüğü	40	40	40	40	40
Üst / Alt Sınır	1,0 / -3,0	1,0 / -2,8	1,0 / -3,5	1,0 / -4.0	1,0 / -2,8
Limit	100	100	100	100	100
MDS	310	70	250	520	310

2.5.3. Araştırma bulguları

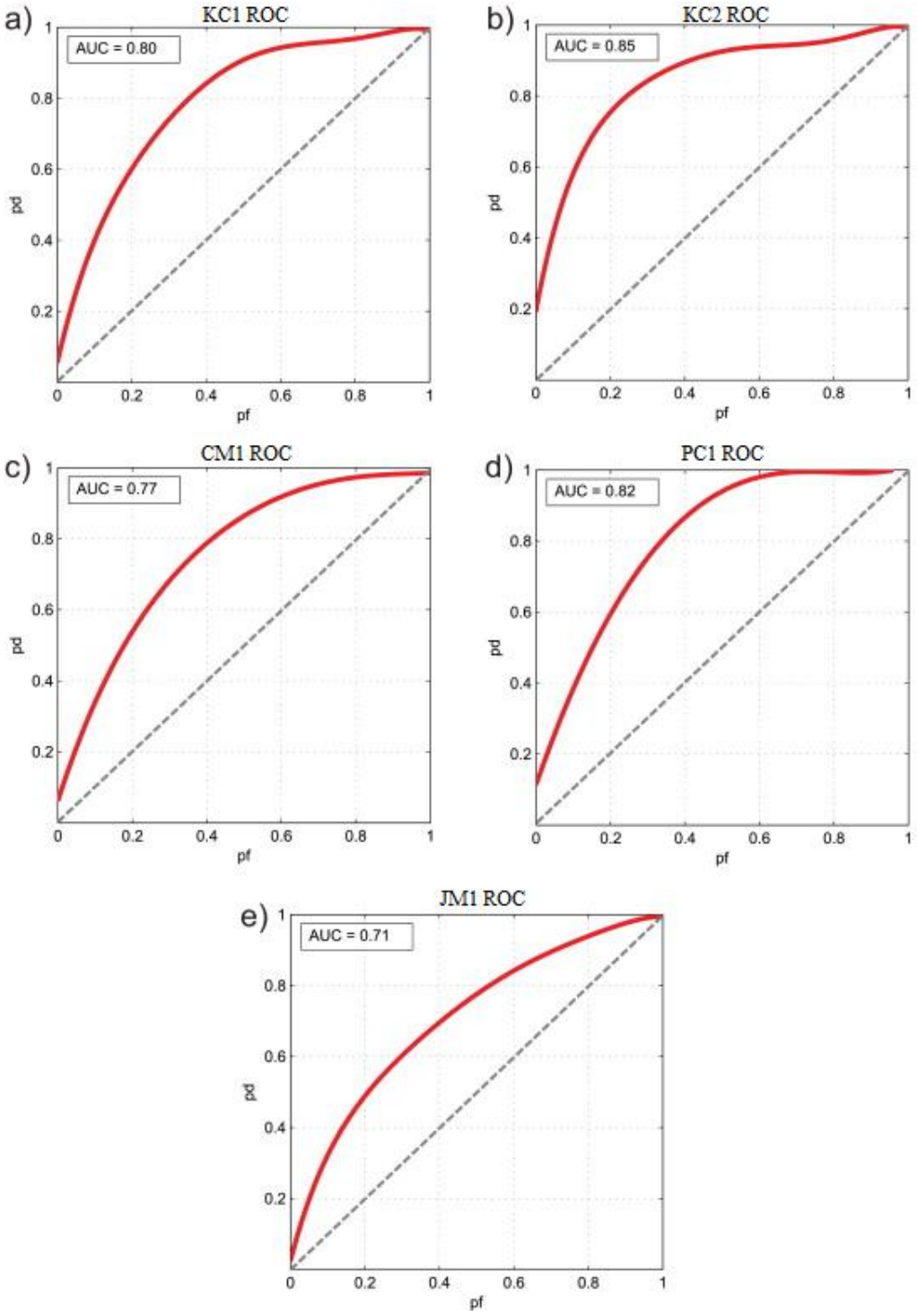
Önerilen sınıflandırıcının performansını daha etkin bir şekilde aktarabilmek için, bu bölüm iki kısma ayrılmıştır: ilki maliyet-duyarlılık dikkate alınmaksızın ve diğeri ise maliyet-duyarlılık dikkate alınarak. Her bir kısımda, elde edilen sonuçlar diğer maliyet-duyarlı ve maliyet-duyarlı olmayan çalışmalarla karşılaştırılmıştır.

2.5.3.1. Araştırma bulguları (maliyet-duyarlılık dikkate alınmaksızın)

Bu kısımdaki sonuçlar, FP ve FN maliyetleri olmaksızın elde edilmiştir. Diğer bir değişle, FP ve FN maliyetlerinin eşit ağırlığa sahip olduğu kabul edilmiştir (C_{FN}/C_{FP}=1). Eğitim verisindeki hatalı ve hatasız modül oranları (P_{hm}ve P_{hsm}) da ihmal edilmiştir. Bu varsayımlar ile YAK algoritması hata fonksiyonu denklem (2.11)'deki duruma dönüşmüştür.

$$\mathrm{NECM} = \mathrm{FPR} \times 0.5 + \mathrm{FNR} \times 0.5 \tag{2.11}$$

Bu kısımdaki sonuçlar maliyet duyarlılık dikkate alınmadan elde edilmiştir. Bu çalışmada, daha önce de bahsedildiği gibi NASA veri havuzunda bulunan beş veri seti kullanılmıştır. Önerilen sınıflandırıcı, her bir veri seti için Şekil 2.11.'deki ROC eğrilerini üretmiştir. Çapraz validasyondaki her bir alt küme koşumu ile birlikte bir ROC eğrisi üretilmiştir; yani, eğer çapraz validasyondaki küme sayısı 10 olarak belirlendiyse 100 adet ROC eğrisi üretilmiş olur (algoritma koşum sayısı da 10 olduğundan). Bu eğriler, 6. dereceden polinom tabanlı eğri-uydurma (curve-fitting) teknikleri kullanılarak tek bir eğriye dönüştürülmüştür. Maliyet değerlerinin değişiminin ROC üzerindeki etkisi çok küçüktür; çünkü, bu tip grafikler doğası gereği farklı eşik değerleri için pd, pf çiftlerini içerirler. Diğer bir değişle, aynı sınıf çıktısına sahip örneklerin ne kadar iyi gruplandırıldığını gösterirler.



Şekil 2.11. ROC eğrileri a) KC1 b) KC2 c) CM1 d) PC1 ve e) JM1

Tablo 2.7.'de her bir veri seti için AUC, pd, pf, bal ve acc performans sonuçları gösterilmiştir. Bütün AUC değerleri 0.5'den büyüktür, bu durum önerilen sınıflandırıcının kabul edilebilir sonuçlara sahip olduğu anlamına gelir. KC1 ve KC2 veri setleri, 0.80 ve 0.85 AUC değerleri ile diğer veri setlerinden daha iyi sonuçlara

sahiptirler. Algoritma koşumunun 10 defa tekrarlanması sonucunda hesaplanan standart sapma değeri ± işaretinden sonra verilmiştir. Sonuçlar algoritmanın gürbüzlüğünü göstermektedir. CM1 ve PC1 veri setleri AUC bazında maksimum standart sapma değerlerine sahiptirler (sırasıyla 0.0156 ve 0.0083). Bunun sebebi, bu iki veri setinin diğerlerine göre daha dengesiz sınıf dağılımı göstermeleridir.

Tablo 2.7. Beş NASA veri setinde elde edilen standart sapma ile AUC, pd, pf, bal ve acc sonuçları

Veri Seti	AUC	pd (%)	pf (%)	bal (%)	acc (%)
KC1	0,80 ± 0,0019	79 ± 0,93	33 ± 0,67	72 ± 0,56	69 ± 0,60
KC2	0,85 ± 0,0068	79 ± 1,26	21 ± 0,60	79 ± 0,61	79 ± 0,46
CM1	0,77 ± 0,0156	75 ± 2,93	33 ± 1,17	71 ± 1,11	68 ± 0,09
PC1	0,82 ± 0,0083	89 ± 2,31	37 ± 1,84	73 ± 1,07	65 ± 1,63
JM1	0,71 ± 0,0021	71 ± 1,22	41 ± 1,47	64 ± 0,44	61 ± 0,97
Ortalama	0,79	78,6	33,0	71,8	68,4

Önerilen sınıflandırıcının, diğer algoritmalarla AUC bazında karşılaştırması Tablo 2.8.'de yapılmıştır. En iyi performansı gösteren sonuç, koyu renkli olarak işaretlenmiştir. NB ve RF sonuçları, Wang ve Yao (2013)'nun çalışmasından; C4.5 sonuçları, Sun ve arkadaşlarının (2012) çalışmasından; Immunos ve AIRS sonuçları ise Catal ve Diri (2009)'nin çalışmasındaki 3 no'lu deneyden buraya aktarılmıştır. 3 no'lu deneyde kullanılan metrik kümesinin aynısı bu çalışmada da kullanılmıştır. Immunos ve AIRS, insan bağışıklık sisteminden ilham alınarak geliştirilen sınıflandırma algoritmalarıdır. Bütün bu sonuçlar, modifikasyon yapılmamış (orijinal) algoritmalardan alınmıştır. Aynı algoritmaları kullanan diğer çalışmalarda da aşağı yukarı benzer sonuçlar elde edilmiştir. Başarı sıralaması, her bir veri setindeki sonuçlara göre yapılmıştır. Önerilen algoritma; KC1, KC2 ve CM1 veri setlerinde en iyi sonucu, PC1 ve JM1 veri setlerinde ise en iyi ikinci sonucu üretmiştir. KC1 veri setinde, önerilen algoritma ile birlikte RF algoritması da en iyi sonucu vermiştir. Her bir algoritmanın her bir veri seti için başarı sıralaması toplanmış ve veri seti sayısına bölünerek ortalama başarı sıralaması elde edilmiştir. Tablonun son sütununda ise sınıflandırma algoritmalarının ortalama başarı sıralaması verilmiştir. Daha düşük başarı sıralaması, ilgili veri seti için daha iyi performans anlamına gelmektedir.

Önerilen sınıflandırıcı, 1.4 değeri ile en iyi ortalama başarı sırasını elde etmiştir. RF de 1.6 ile yakın bir performans sıralaması elde etmiştir. NB ise 2.6 ile bunlardan sonra gelmektedir. Diğer algoritmalar ise zayıf performans sonuçları elde etmişlerdir.

Tablo 2.8. Önerilen sınıflandırıcının, AUC bazında farklı sınıflandırıcılar ile karşılaştırması

Sınıflandırıcı	KC1	KC2	CM1	PC1	JM1	Ort. Sıralama
Önerilen	**0,80**	**0,85**	**0,77**	0,82	0,71	**1,4**
NB	0,79	0,82	0,75	0,70	0,68	2,6
RF	**0,80**	0,82	0,74	**0,85**	**0,75**	1,6
C4.5	0,64	0,67	0,53	0,68	0,61	5,0
Immunos	0,71	0,73	0,63	0,64	0,63	4,0
AIRS	0,60	0,67	0,53	0,58	0,56	5,6

Parametrik olmayan Friedman test kullanılarak, önerilen sınıflandırıcı, istatistiki anlamlılık (statistical significance) yönüyle diğer algoritmalarla karşılaştırılmıştır. Vandecruys ve arkadaşları (2008) da yaptıkları çalışmada bu istatistiki yöntemi kullanmışlardır. Friedman testi, anlamlılık düzeyinin altında bir değer döndürmüştür. Bu, en az iki sınıflandırıcı performansının anlamlı bir şekilde birbirinden farklı olduğunu ifade etmektedir. Sonrasında, Conover (1999)'un çalışmasına dayanarak ikili karşılaştırmalar yapılmıştır. Tablo 2.9.'da bu karşılaştırmaların sonuçları gösterilmiştir. Tabloda da görüldüğü üzere; önerilen sınıflandırıcının performansı, RF haricindeki diğer sınıflandırıcılardan anlamlı bir şekilde farklıdır (%99 anlamlılık düzeyi baz alındığında). Fakat bu durum önerilen sınıflandırıcının, RF'den daha kötü olduğu anlamına gelmektedir.

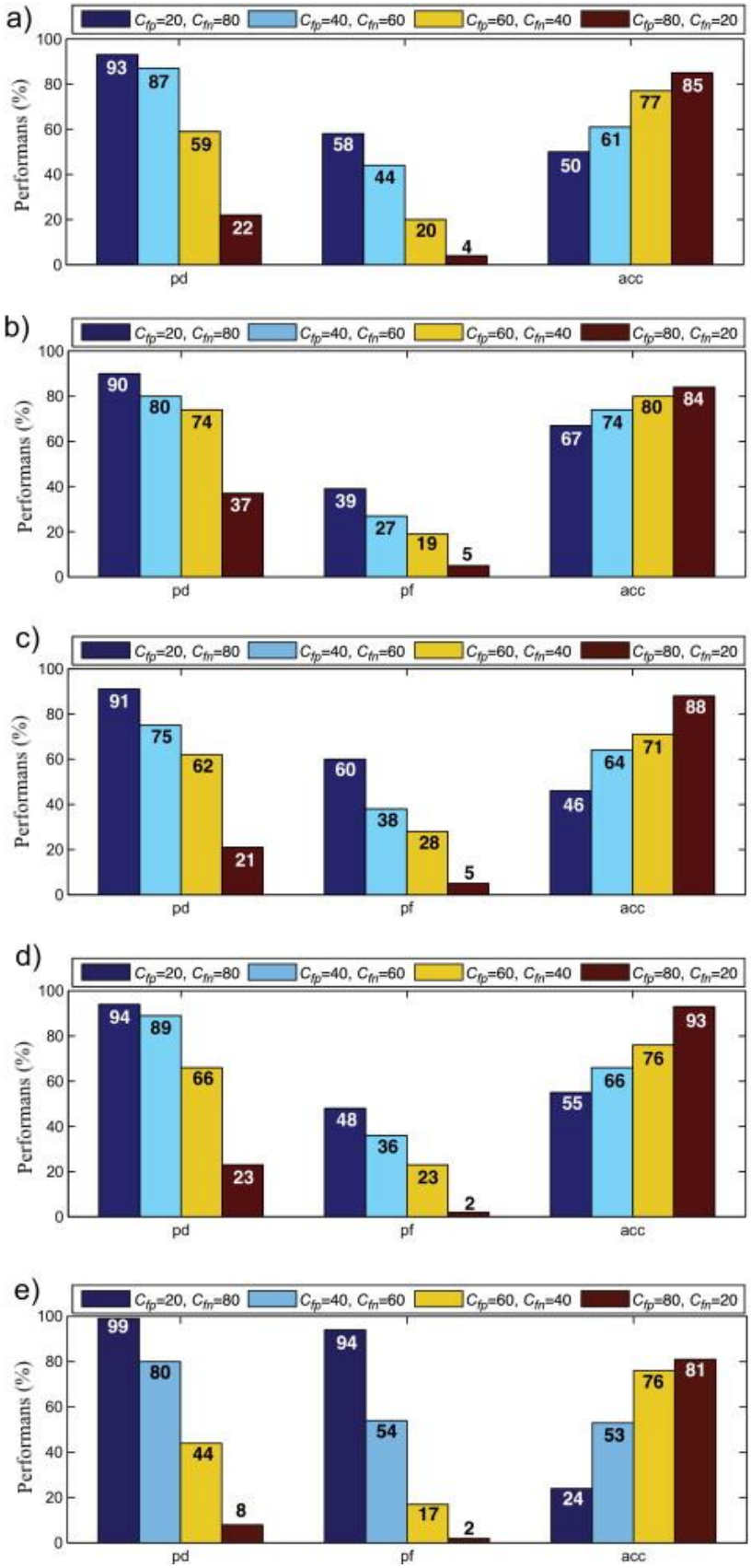
Tablo 2.9. Sınıflandırıcı performanslarının Friedman test karşılaştırması

I	J	Farklar (I-J)
Önerilen	NB	1.2 **
	RF	0.3
	C4.5	3.5 ***
	Immunos	2.7 ***
	AIRS	4.3 ***

* $p<0.05$ ** $p<0.01$ *** $p<0.001$

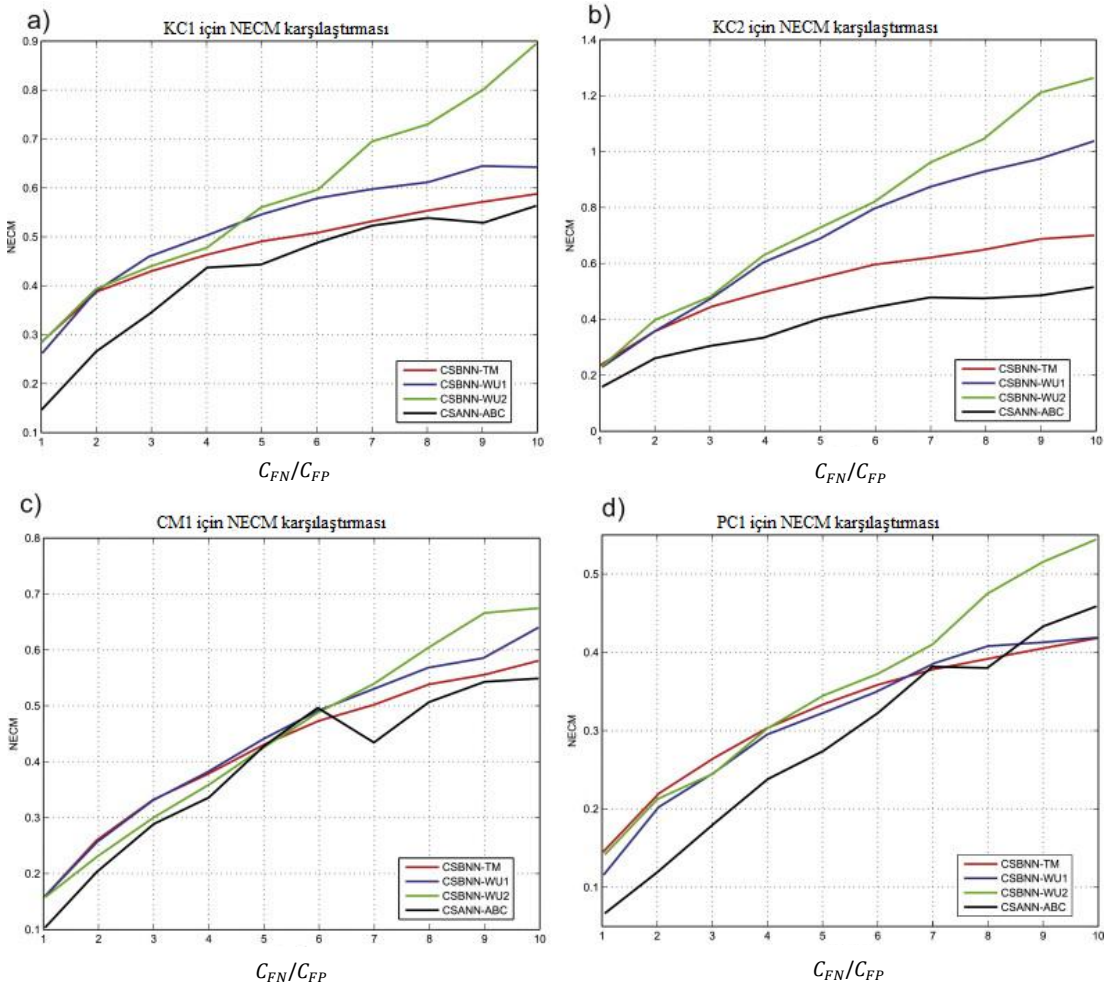
2.5.3.2. Araştırma bulguları (maliyet-duyarlılık dikkate alınarak)

Önerilen sistemin avantajların biri de maliyet-duyarlılık özelliğine sahip olmasıdır. Bu kısımda maliyet-duyarlılık özelliği dikkate alınması durumundaki sonuçlar sunulmaktadır. Özel maliyet oranları (C_{FN}/C_{FP}) kullanılarak, farklı (pd, pf) ikilileri elde edilebilir. Şekil 2.12.'de dört farklı maliyet oranlarına göre pd, pf ve acc sonuçları sunulmuştur. Şekilden de görüldüğü üzere; FN maliyeti (C_{FN}) azaldıkça (düşük maliyet oranlarında), pd performansı da azalır. Aynı şekilde, FP maliyeti (C_{FP}) arttıkça (yüksek maliyet oranlarında), beklenildiği gibi pf performansı da artar. Düşük pf değerleri, yanlış alarm bakımından iyi performans anlamına gelmektedir. pf performansı ile acc performansı arasında direk bir ilişki vardır. Çünkü, çoğunluk sınıfını negatif (hatasız modül) olanlar oluşturmaktadır. Düşük pf değeri, negatif sınıflardaki hatanın az olduğunu anlamına gelmektedir; dolayısıyla acc sonucunun iyileşmesini sağlamaktadır.



Şekil 2.12. Farklı maliyet değerlerine göre pd, pf ve acc sonuçları a) KC1 için b) KC2 için c) CM1 için d) PC1 için ve e) JM1 için

NECM, maliyet-duyarlı algoritmalarda kullanılan tekil bir performans ölçütüdür. Önerilen algoritmadaki sonuçlar, Zheng (2010)'in çalışmasında geçen değişik türdeki sinir ağları ile karşılaştırılmıştır. Çalışmasında iki farklı sinir ağı kullanmıştır: Eşik-Kaydırmalı Maliyet-Duyarlı Hızlandırılmış Sinir Ağı (Cost-Sensitive Boosting Neural Networks with Threshold-Moving (CSBNN-TM)) ve Ağırlık-Güncellemeli Maliyet-Duyarlı Hızlandırılmış Sinir Ağı (Cost-Sensitive Boosting Neural Networks with Weight-Updating (CSBNN-WU)). Orijinal ağırlık güncelleme işleminde iki modifikasyon yapmıştır. Bu modifiye edilmiş algoritmalar, CSBNN-WU1 ve CSBNN-WU2 olarak isimlendirilmiştir. Önerilen algoritmanın, bu üç maliyet-duyarlı algoritma ile NECM baz alınarak karşılaştırması Şekil 2.13.'de verilmiştir. Zheng'in çalışmasında JM1 veri seti yer almamıştır; dolayısıyla, karşılaştırmalar diğer dört veri setine göre yapılmıştır. Önerilen algoritma, (Cost-Sensitive Artifical Neural Network with Artifical Bee Colony) CSANN-ABC olarak adlandırılmış, Şekil 2.13.'de bu ad ile gösterilmiştir. Farklı maliyet oranlarına (1'den 10'a kadar) göre NECM sonuçları Şekil 2.13.'de gösterilmiştir. *x*-ekseni maliyet oranlarını ve *y*-ekseni ise ilişkili NECM sonuçlarını göstermektedir. Önerilen algoritma, dört veri setinde ve hemen hemen bütün maliyet oranlarında en iyi sonucu vermiştir. Sadece, CM1 veri setinde C_{FN}/C_{FP}= 6 olduğunda ve PC1 veri setinde C_{FN}/C_{FP}= 7 olduğunda daha kötü sonuçlar vermiştir. NECM çıktısının düşük olması, sonucun iyi olduğu anlamına gelmektedir.



Şekil 2.13. Önerilen algoritmanın NECM bazında diğer algoritmalarla karşılaştırması a) KC1 için b) KC2 için c) CM1 için ve d) PC1 için

YAK algoritması, sınıflandırıcı performansını etkileyen farklı kontrol parametreleri içermektedir. MDS artarken performansın da arttığı görülmüş, ancak belli bir değere ulaştığında dengeye ulaşmıştır. Diğer bir değişle, MDS'nin sürekli artışı, performans artışını sağlamamaktadır. Ayrıca, MDS'nin performans üzerindeki etkisinin çok az olduğu söylenebilir. Arama uzayındaki alt ve üst sınır değerleri, YAK'ın diğer önemli parametreleridir. Tablo 2.6.'da görüldüğü üzere, alt ve üst sınır değerleri simetrik değildir (0 orta değer olarak kabul edildiğinde). Bu durum, YAK'ın kullanıldığı diğer çalışmalardan farklılık göstermektedir. Diskriminant noktasının 0 olduğu varsayılırsa, negatif aralık pozitif aralığa göre daha geniştir. Bu durum, algoritmayı dengesiz sınıf dağılımlı veri setlerine karşı daha güçlü hale getirmiştir. Veri setindeki dengesizlik oranı ile alt sınır değeri arasında ilişki vardır. Örneğin, PC1 en dengesiz dağılımlı veri setidir ve en düşük alt sınır değerine sahiptir. Diğer taraftan, KC2 ve JM1 veri seti

dengesizlik dağılımı en az olan veri setleridir ve alt sınır değerleri en yüksektir. Bullinaria ve AlYahya (2014), UCI makine öğrenmesi veri setlerinde sınır değerlerinin olmadığı "Optimize Edilmiş UABC" kullanılmasının daha iyi sonuçlar verdiğini iddia etmişlerdir. Ancak, bu çalışmada yazılım hata veri setlerinde sınır değerleri kullanılmadığında performansın çok düşük çıktığı görülmüştür.

2.6. Sonuç

Bu çalışmada yazılım hata kestirim problemi için hibrit bir sınıflandırma yöntemi önerilmiştir. YSA nöron bağlantı ağırlıkları, arıların yiyecek arama davranışını modelleyen YAK algoritması ile optimize edilmiştir. Yeni bir hata fonksiyonu oluşturulması ile YSA'ya parametrik maliyet-duyarlılık özelliği eklenmiştir. Pozitif ve Negatif sınıfların yanlış sınıflandırma maliyeti ilişkili katsayılar ile belirlenir. Bu maliyet değerlerinin pd, pf ve acc performansına etkisi vardır. Bu yöntem ile riskten-kaçınan ve maliyetten-kaçınan işletmeler, kendi yazılım projelerinin ihtiyaçlarına göre en uygun katsayıları belirleyebilmeleri sağlanmıştır.

Önerilen sınıflandırıcının performansı beş NASA veri seti kullanılarak diğer algoritmalar ile karşılaştırılmış ve elde edilen sonuçlar performansının diğerlerinden daha iyi olduğunu göstermiştir.

BÖLÜM 3. AÇIK KAYNAK KODLU YAZILIMLAR İÇİN YAZILIM METRİK EŞİK DEĞERLERİNİN TÜRETİMİ VE HATA KESTİRİMİNDE UYGULANMASI

3.1. Giriş

Yazılım sistemleri günlük yaşantımızda çök önemli bir role sahiptir ve her geçen gün kullanımı daha da yaygınlaşmaktadır. Makinelerin ve servislerin büyük çoğunluğu kendi içlerinde farklı türde yazılım içerirler. Yazılım geliştiriciler, günlük kullanımını yaygınlaştırmak ve rekabette geri kalmamak için mümkün olduğunca hızlı bir şekilde yazılımları geliştirmektedirler. Yazılım geliştirmenin her bir safhası etkin bir şekilde yürütülmelidir. Test, en önemli safhalardan bir tanesidir ve dikkatli bir şekilde ele alınmalıdır. Etkin olmayan bir şekilde işletilen test safhası, çeşitli problemler doğurabilir. Yazılımda mevcut olan bir hatanın test ekibi tarafından tespit edilememesinin maliyeti, canlı kullanım sırasında ağır olabilir. Devam eden bir servisi sekteye uğratabilir veya ilgili yazılımla bağlantılı diğer sistemlerde arızalara sebep olabilir. Eğer, bu sistem, banka yazılımı veya uzay aracı gibi karmaşık bir sistem ise aksaklığın sebep olduğu maliyet de çok daha ağır olacaktır. Örneğin, 125 milyon dolara geliştirilen NASA uzay aracı görevi sırasında uzayda kayboldu. Bu durumun sebebinin küçük bir veri dönüştürme hatası olduğu ortaya çıktı (Michaels, 2008). Daha güncel bir gelişme olarak; yeni tasarlanmış Airbus A400M tipi askeri kargo uçağı test uçuşu sırasında düştü. Teknik uzmanlar düşüş sebebinin hatalı motor kontrol yazılımı olduğunu keşfettiler (Flottau ve Osborne, 2015). Amerika Birleşik Devletleri Savunma Bakanlığının, yazılım hatalarından kaynaklı yaptığı harcama miktarı her yıl 4 milyar dolardır (Dick ve ark, 2004).

Test işlemlerinin etkinliğini arttırmak için son on yılda akademisyenler tarafından otomatik hata tespit ve erken uyarı sistemleri üzerinde yoğunlaşma olmuştur. Yazılım modüllerinin hataya meyilli (hm) ve hatasızlığa meyilli (hsm) olarak sınıflandırıldığı bir liste sunulmasını sağlayan böyle bir hata tespit sistemi, yazılım projelerindeki test ekiplerinin faaliyetlerini kolaylaştıracaktır. Böylelikle, test personeli, hsm modüllerden ziyade hm modüllere yoğunlaşır. Yazılım sisteminin dahili (internal) kalite özellikleri, ilgili modülün hataya meyillilik durumu hakkında ipuçları verir. Kalite ile hataya meyillilik arasındaki bu ilişki bir sonraki kısımda referansları yapılan ve bu çalışmada yer bulmayan birçok çalışma ile doğrulanmıştır. Yazılım metrikleri, yazılım sisteminin dahili kalitesini yansıtmayı amaçlar. Farklı metrik kümeleri önerilmiştir. Bunlar arasında en yaygın olarak kullanılanları şunlardır: McCabe (McCabe, 1976), Halstead (Halstead, 1977) ve Chidamber & Kemerer (CK) (Chidamber ve Kemerer, 1994). McCabe ve Halstead metrikleri, kaliteyi metot-seviyesinde yansıtırlar; CK metrikleri ise kaliteyi sınıf-seviyesinde yansıtırlar. Farklı metrik grupları, yazılımı farlı yönlerden değerlendirirler. Bu metriklere ait eşik değerlerin türetiminin; yazılım geliştiricileri, test personeli ve müşteriye katkısı olacaktır. Bunun neticesinde yazılım geliştiricileri, versiyon yayınından önce eşik değerini aşan modüllere yoğunlaşabilirler. Diğer taraftan, yazılım kalitesini değerlendirecek düzeyde teknik bilgiye genellikle sahip olmayan müşteriler, bu eşik değerlerinin yardımıyla teslim edilen yazılımın kalitesi hakkında fikir sahibi olurlar. Eşik değerleri, bir örneği genellikle iki grup şeklinde kategorilendirmek için kullanılırlar; böylelikle, her örnek kategorisine göre ele alınır. Henderson-Sellers (1996), eşik değerlerinin pratik yararlarını vurgulamışlar ve şu tanımlamayı yapmışlardır: "Belirli bir dahili metrik değeri, önceden belirlenmiş eşik değerini aştığı zaman bir alarm oluşturulur".

Bu çalışmada, bundan sonraki geri kalan kısımlarda kaynak çalışma olarak adlandırılacak olan Shatnawi (2010)'nin çalışmasının replikasyonu sunulmuştur. Bu replikasyonda, daha büyük boyutta veri seti ve metrikler kullanılarak kaynak çalışma genişletilmiştir. Ayrıca, kaynak çalışmada yapılan deneylerde projeler ayrık olarak kullanılmıştır; yani, eğitim seti ve test setinin seçimi aynı proje içerisinde yapılmıştır. Bu sebeple, türetilen eşik değerleri proje bağımlıdır. Diğer proje ekipleri, kaynak

çalışmanın çıktısı olan eşik değerlerinden faydalanamayabilirler. Bu replikasyon çalışmasında ise belli bir proje grubu için üniversal eşik değerleri türetilmesine yönelik deneyler yapılmıştır. Böylelikle, benzer karakteristiklere sahip diğer projeler de bu çalışmanın çıktısından yararlanabilirler. Çalışma yapısı kurgulanırken, Carver (2010) tarafından replikasyon çalışmaları için önerilen standartlara uyulmuştur.

Bu bölüm şu şekilde yapılandırılmıştır: Kaynak çalışma altbölüm 3.1'de anlatılmıştır. Altbölüm 3.2., kaynak çalışmasını; altbölüm 3.3 ise, replikasyon çalışmasını; hedefler, hipotezler ve bağlamlarıyla birlikte ele alır. Altbölüm 3.4. , önceden yapılan ilgili çalışmaları içeren kaynak araştırmasını özetler. Bu çalışmada kullanılan veri setleri ve metrikler altbölüm 3.5'de aktarılmıştır. Altbölüm 3.6., kullanılan algoritmaların yanında eşik değeri türetim modelinin açıklamasını içermektedir. Altbölüm 3.7'de, çalışmanın bulguları bulunmaktadır. Geçerliliği tehdit eden faktörler altbölüm 3.8'de anlatılmıştır. Çalışma sonuçlar ise altbölüm 3.9'da özetlenmiştir.

3.2. Kaynak Çalışma

Bu altbölümde kaynak çalışma kısaca tanıtılmıştır. Kaynak çalışma ile ilgili diğer detaylar, ilgili altbölümlerde anlatılmıştır; örneğin, kaynak çalışmada kullanılan veri setleri ve metrikler aynı başlıklı altbölümde aktarılmıştır. Shatnawi (2010), çalışmasında Bender metodu (Bender, 1999) tabanlı Lojistik Regresyon kullanmıştır. Eclipse 2.0'ı eğitim veri seti ve Eclipse 2.1'i ise test veri seti olarak kullanmıştır. Çalışmasında analiz ettiği araştırma sorusu; CK metrikleri için türetilen eşik değerlerin hataya meyillilik yönüyle etkisinin incelenmesidir. Altı CK metriği içerisinden üç tanesi için performans limiti ve istatistiki anlamlılık testini geçerek, etkili eşik değerine sahip olduğu sonucuna ulaşmıştır. Ancak, aynı modeli, farklı iki açık kaynak kodlu sisteme (Mozilla ve Rhino) uyguladığında az sayıda pozitif eşik değeri elde etmiştir, çoğu negatif eşik değeri olmuştur. Shatnawi, önerilen modelin bütün açık kaynak kodlu sistemlerde kullanılacak şekilde genelleştirilemeyeceği sonucunu ifade etmiştir.

3.3. Replikasyon Çalışması

3.3.1. Araştırma sorusu

Bu çalışmadaki araştırma sorusu kaynak çalışma ile temel olarak aynıdır. Bu çalışmadaki ana hedef olan, benzer diğer yazılım sistemlerinde de uygulanabilecek etkili üniversal eşik değerlerinin araştırılması; kaynak çalışmadan farklılık göstermektedir.

3.3.2. Hipotezler

Bu çalışmanın amacı, yazılım metriklerine ait etkili üniversal eşik değerleri olup olmadığının deneysel olarak sorgulanmasıdır. Böylelikle, bu çalışmada her bir metrik için test edilecek olan hipotez[1] ve sıfır hipotez aşağıdaki gibidir:

Hipotez: Metrik eşik değeri ile modülün hataya meyilliliği arasında bir ilişki vardır; yani, bir modülün metrik değeri, türetilen eşik değerden daha büyükse, o modül daha fazla hataya meyillidir.

Sıfır Hipotez: Metrik eşik değeri ile modülün hataya meyilliliği arasında bir ilişki yoktur; yani, bir modülün metrik değeri, türetilen eşik değerden daha büyükse, o modül daha fazla hataya meyilli olabilir veya olmayabilir.

3.3.3. Deney düzeni

Yazılım modülleri hataya meyillilik grubu olarak üç kategoriye ayrılmıştır: sıfır hataya meyilli (0hm), 1+hataya meyilli (1+hm) ve 3+hataya meyilli (3+hm). 1+hm yazılım modülü, o yazılımın koşumu sırasında en az 1 hataya sahip olma potansiyelindeki modülü temsil eder. Diğer taraftan; 3+hm yazılım modülü, yazılım koşumu sırasında en az 3 hataya sahip olma potansiyelindeki modülü temsil eder. Bu şekilde yapılan üç seviyeli etiketleme yaklaşımı ile geliştirici firma; kodlama ve test aktivitelerini

[1] Sıfır hipoteze karşılık gelen alternatif hipotezlerdir.

önceliklendirme konusunda fikir sahip olur. Test bakış açısından; firma kaynaklarını daha etkili bir şekilde tahsis eder. Sınırlı kaynak olması durumunda, sadece 3+hm modüller dikkate alınır. Yeterli personel ve zaman durumunda ise, 1+hm modüller de dikkate alınır. Ayrıca, modüllerin risk durumunu belirlemede, üç seviyeli etiketleme, başka çalışmalarda da kullanılmıştır. Bu çalışmada, hata sayılarının dağılımı baz alınarak, üç hataya meyillilik grubu tanımlanmıştır. Sıfır hataya sahip modüller oldukça fazladır (%76); 1 ile 3 hata sayısına sahip modüllerin oranı daha azdır (%21) ve 3'den fazla hataya sahip modüller ise azdır (%5).

PROMISE[2] veri havuzundan (Jureczko ve Madeyski, 2010; Menzies ve ark., 2012) açık kaynak kodlu yazılım sistemleri, deney veri seti olarak kullanılmıştır. Her bir yazılım metriğinin eşik değerinin türetilmesinde, Bender metodu tabanlı bir Lojistik Regresyon modeli uygulanmıştır. Deneylerin değerlendirilmesinde kullanılan performans ölçütü ise, Doğru Pozitif Oranı (TPR) ile Doğru Negatif Oranının (TNR) geometrik ortalamasıdır (*g-mean*).

3.3.4. Kaynak çalışmada yapılan değişiklikler

Bu çalışmadaki eşik değeri türetim modelinin oluşturulmasında, kaynak çalışmadaki model temel alınmıştır. Kaynak çalışmaya göre bu çalışmadaki katkılar ve farklı yönler aşağıda listelenmiştir:

- Üniversal eşik değerlerine ulaşmak için 10 farklı açık kaynak kodlu sistem kullanılmış ve eşik değerlerin validasyonu, 27 veri seti üzerinde yapılmıştır. Kaynak çalışma: öğrenme modeli, farklı açık kaynak kodlu sistemlere ayrı ve bağımsız olarak uygulanmıştır.
- Bu çalışmada kullanılan metrik sayısı daha kapsamlıdır. CK metrik grubunun yanı sıra farklı metrik grupları da kullanılmıştır. Kaynak çalışma: sadece altı CK metriğini içerir.
- Üç risk seviyesi (0hm, 1+hm ve 3+hm) kullanılmıştır. Bu kullanım durumunun pratik katkısının olması beklenmektedir. Kaynak çalışma: iki risk seviyesi kullanılmıştır (hataya meyilli ve hatasızlığa meyilli)

[2] Bütün veri setleri açıklamaları ile birlikte http://openscience.us/repo/ adresinde bulunmaktadır.

3.4. Kaynak Araştırması

3.4.1. Hata kestirimi

Yazılım hatalarının kestirimi amacıyla birçok makine öğrenmesi tekniği kullanılmıştır. Şu teknikler bu problem için kullanılmıştır: Random Forest (Guo ve ark., 2004), Yapay Bağışıklık Sistemi (Catal ve Diri, 2009), Naive Bayes (Padberg ve ark., 2004; Menzies ve ark., 2007; Turhan ve ark., 2013), J48 (Koru ve Liu, 2005), ağaç-tabanlı metotlar (Selby ve Porter, 1988; Khoshgoftaar ve ark., 1999; Guo ve ark., 2004), durum-tabanlı muhakeme (Khoshgoftaar ve Seliya, 2003), Bayesian Ağı (Pérez-Miñana ve Gras, 2006), ve Yapay Sinir Ağı (Zheng, 2010; Arar ve Ayan, 2015). Menzies ve arkadaşları (2007), log-filtreleme veri önişlemi ile birlikte Naive Bayes kullanımının diğer öğrenme modellerine göre en iyi performansı verdiği sonucuna varmışlardır. Destek Vektör Makineleri (Elish ve Elish, 2008) ve Lojistik Regresyon (Olague ve ark., 2007; Gao ve ark., 2011) gibi istatistiki yöntemler de bu probleme uygulanmış olup dikkat çekici sonuçlar elde edilmiştir. Birden fazla metrik kombinasyonu yerine metriklerin tekil (bağımsız) olarak etkisinin analiz edilmesinde tek değişkenli Lojistik Regresyon kullanılmıştır. Böylece, her bir metriğin hataya meyillilik ile bağlantısının kuvveti ortaya çıkmıştır. Gyimothy ve arkadaşları (2005) ve Zhou ve Leung (2006)'in çalışmaları çok değişkenli ve tek değişkenli Lojistik Regresyonun kullanıldığı iki çalışma örneğidir. Hall ve arkadaşları (2011), yaptıkları kapsamlı literatür taraması ile 2000 yılından 2010'a kadar yapılan çalışmaları analiz etmişlerdir. Geliştirdikleri kriterleri kullanarak, bağlamsal ve metodolojik bilgi eksikliği olan birçok çalışmayı analizin sonraki aşamalarına geçirmemişler, elemişlerdir. Sonrasında, geriye kalan 36 çalışmanın niceliksel ve niteliksel sonuçlarını sentezlemişlerdir. Sonuç kısmında, şunu ifade etmişlerdir: "birçok hata kestirim çalışması yapılmış olmasına rağmen; bağlam, metodoloji ve performansının kapsamlı olarak raporlandığı ve güvenilir metodolojinin kullanıldığı daha fazla çalışmaya ihtiyaç duyulmaktadır". Yapılan bu çalışmanın motivasyon noktalarından birisi de yazarların yaptıkları bu tespittir.

3.4.2. Metrik eşik değeri türetimi

Yazılım hata kestirim çalışmalarına olan yoğun ilgi ile kıyaslandığında, metrik eşik değeri türetim çalışmaları çok daha azdır. Genellikle, metrik eşik değerleri, kodlama tecrübelerine dayanarak kıdemli yazılım uzmanları tarafından belirlenir (Oliveira ve ark., 2014). Ancak, makine öğrenmesi tekniklerine dayalı çeşitli metrik eşik değeri çıkartım yöntemleri de önerilmiştir. Bender ve ROC metotları bunlar arasında en popüler olanlardır. Shatnawi (2010), çalışmasında Bender Metot tabanlı Lojistik Regresyon kullanmıştır. Catal ve arkadaşları (2011), C++ dili kodlanmış genel erişime açık NASA veri setleri üzerinde ROC tabanlı eşik değeri değerlendirme yöntemini kullanmışlardır. Bu veri setleri çoğunlukla metot-seviyeli metrikler içerirler. Sánchez-González ve arkadaşları (2012), iki tekniği birlikte uygulayarak iş süreci modelleri için eşik değerler elde etmişlerdir. Sonrasında, performans sonuçlarını, duyarlılık (recall), kesinlik (precision) ve f-ölçütü (f-measure) yönüyle karşılaştırmışlardır. İki deney kümesinde ROC eğrisi tabanlı yaklaşımın kullanımı ile daha güvenilir sonuçlar elde etmişlerdir. Shatnawi ve arkadaşları (2010), farklı Eclipse sürümlerine ait veri setlerini kullanarak, ikili ve ordinal kategorizasyon eşik değerlerini bulmak için ROC kullanmışlardır. Ancak, türetilen eşik değerleri ile ikili kategorizasyon için kabul edilebilir sonuçlar üretilememiştir.

Diğer istatistiki ve veri madenciliği teknikleri de aynı amaç için kullanılmıştır. Ferreira ve arkadaşları (2012), yazılım modüllerini metrik değerlerine göre iyi, normal ve kötü olarak sınıflandırmışlardır. Metriklerin eşik değerlerini, Java dili ile kodlanmış 40 açık kaynak kodlu yazılım programına ait verilerin istatistiki dağılımını baz alarak belirlemişlerdir. Veri, en iyi uyuma sahip olasılıksal dağılıma (Poisson veya Weibull) dönüştürülmüştür. Kategorizasyonu, her metriğin frekansına göre yapmışlardır. Ferreira ve arkadaşları (2012), türetilen bu eşik değerlerinin karar vermeye güçlü olarak katkıda bulunabileceğini; ancak, uzmanların görüşlerinin yerine geçemeyeceğini belirtmişlerdir. Ferreira ve arkadaşları, kötü olarak kategorilendirilmiş modülleri; gerçekten olası tasarım kusurlarına ve yapısal kodlama problemlerine sahip olup olmadığını tespit etmek için manuel olarak kontrol etmişlerdir. Bu inceleme sonucunda, önerilen eşik değerlerinin yardımıyla, potansiyel

olarak problemli olan modellerin tespitinin yapılabileceği ortaya çıkmıştır. Erni ve Lewerentz (1996), metrik değerlerinin ortalama ve standart sapmalarını baz alarak eşik değerlerini bulmaya yönelik bir yaklaşım önermişlerdir. İki eşik değerini belirlemek için iki formül oluşturmuşlardır: $T_{min} = \mu - \sigma$ ve $T_{max} = \mu + \sigma$ (μ ortalama sembolü ve σ standart sapma sembolüdür). Herbold ve arkadaşları (2011), kullandıkları Diktörtgensel Öğrenme (Rectangle Learning) adındaki başka bir makine öğrenmesi yöntemiyle, farklı programlama dilleri için ayrı deneylerle eşik değerleri türetmişlerdir. Morasca ve Lavazza (2016), metrik eşik değerlerini elde etmek için öncelikle eğitim verisindeki metrik değerlerinden dağılım eğrisini oluşturmuşlardır. Bu eğrinin eğim ve anlık değişim noktalarına göre eşik değerlerini belirlemişlerdir. Veado ve arkadaşları (2016), TDTool adını verdikleri, açık kaynak kodlu eşik değeri türetim yazılımı ile literatürde geçen üç çalışmayı (Alves ve ark., 2010; Ferreira ve ark., 2012; Oliveira ve ark., 2014), araştırmacıların ve proje geliştiricilerin kullanımı için geliştirmişlerdir. Rosenberg ve arkadaşları (1999), Yazılım Güvence Teknoloji Merkezi (Software Assurance Technology Center)'nin NASA projelerinde kullanımı için önerdikleri CK metriklerine ait eşik değerlerini ele almışlardır.

Yukarıda özetlenen çeşitli çalışmalar tarafından türetilen eşik değerleri Tablo 3.1.'de gösterilmiştir. İkinci sütun, kaynak çalışmadan elde edilen eşik değerleridir. İlgili çalışmalarda istatistiki anlamlılığı olmayan metrikler, tabloda U (Uygulanamaz) ile ifade edilmiştir. Bazı çalışmalar tablodaki bütün metrikleri içermemişlerdir (tire ile gösterilmiştir); diğer taraftan, bazı çalışmalar tablodaki metriklerden daha fazlasını içermişlerdir. Sadece bu çalışmada da kullanılan metrikler tabloya aktarılmıştır, diğerleri aktarılmamıştır.

Tablo 3.1. Farklı çalışmalarda türetilen eşik değerleri [1]

	Shatnawi (2010)	Ferreira ve ark. (2012)	Herbold ve ark. (2011) [2]	Rosenberg ve ark. (1999)
Veri seti	Eclipse	40 açık kaynaklı sistem	Eclipse	Yok
Metot	Bender	Veri Dağılımları	Rectangle Learning	-
Dil	Java	Java	Java	Genel
CBO	9	-	5	5
RFC	40	-	98	100
WMC [3]	20	-	99	100
DIT	U	2	U	2-5
NOC	U	-	U	-
LCOM	U	20	-	-
LOC	-	-	-	-
Ca		20	-	-

[1] Bu çalışma kapsamında kullanılan tüm metriklerin detayları ilerleyen kısımlarda anlatılacaktır.
[2] Eşik değerleri, aynı zamanda C ve C++ dilleri için de türetilmiştir. Bu çalışma ile bağlantılı olmasından dolayı sadece Java metrikleri buraya aktarılmıştır.
[3] WMC hesaplaması araçlara göre değişebilmektedir: Bazıları her bir metot için 1 değerini kabul etmekte, bazıları ise McCabe'in metot bazındaki hesaplamasını yapmaktadır. Bu değişkenlikten dolayı, bu metriğe göre bir kıyaslama yapılmayacaktır.

Bir eşik değeri belirleme metodolojisinin bazı gereksinimleri karşılaması gerekir: i) Geniş temsile sahip kıyaslamalı veri seti üzerinde kurulmalıdır, görüşlere göre değil. ii) Veri, istatistiki olarak analiz edilmelidir. Diğer bir deyişle, dağılım ve skalalar dikkate alınmalıdır. iii) Anlaşılır, sade, tekrarlanabilir ve kolayca diğer sistemlerde kullanılabilir olmalıdır (Alves ve ark., 2010; Sánchez-González ve ark., 2012). Bu üç şartın da bu çalışmada karşılanmasına dikkat edilmiştir.

3.4.3. Üniversal modeller (çapraz-proje hata kestirimi)

Bu çalışmadaki deneylerde incelenen projelere benzeyen diğer projelerde de kullanılabilecek üniversal eşik değerleri elde etmek, bu çalışmanın amaçlarından biridir. Hata kestirim çalışmaları, genellikle öğrenme modelini kurarken yazılımın önceki versiyonlarına ait geçmiş verileri kullanırlar ve elde edilen modeli gelecek versiyon verilerine uygularlar. Bu kullanım stratejisi, proje-içi (within-project) hata kestirimi olarak adlandırılmaktadır. Ancak, ilk versiyonun geliştirilmesi aşamasında olan veya geçmiş verileri bulundurmayan projeler için bu yaklaşım uygulanamaz. Bu sebeple, geçmiş verilere ihtiyaç duymayan yeni stratejilerin tanıtılmasına ihtiyaç

duyulmuştur (Jureczko ve Madeyski, 2010). Çapraz-proje (cross-project) hata kestirim stratejisi; diğer projelerin verileri ile eğitimi yapılan öğrenme modelinin, sınırlı geçmiş veriye sahip başka bir projede başarılı tahmin performansı elde etmesini amaçlar. Birçok çalışmada bu senaryonun olabilirliği ele alınmıştır (Canfora ve ark., 2013; Zimmermann ve ark., 2009; He ve ark., 2015). Çapraz-proje hata kestirim problemi iki sebepten dolayı ilgi uyandırmaktadır: i) Sınırlı geçmiş veri bulunduran projeler tarafından kullanılabilir. ii) Üniversal hata kestirim modelleri geliştirilmiş olur. Projeler heterojen karakteristikler sergiledikleri için bu problem zorludur ve kestirim doğruluğu her zaman parlak olmamaktadır (Canfora ve ark., 2013). Ancak, Koru (2005) tarafından yapılan çalışmada, genel kullanıma özgü hata kestirim modellerinin kurulabileceği ve bu tarz modellerin oldukça tatminkar sonuçlar verebileceği belirtilmiştir. Bu yöndeki bir görüş, bu problem üzerindeki girişimleri teşvik etmiş olmaktadır. Ryu ve Baik (2016) tarafından yapılan çalışmada çok amaçlı Naive Bayes yöntemi çapraz-proje hata kestiriminde kullanılmış ve proje-içi hata kestirim modelleri ile kıyaslanabilir düzeyde performans değerlerine ulaşılmıştır. Ryu ve ark. (2016) tarafından yapılan çalışmada, VCB-SVM adı verilen öğrenme modeli ile Destek Vektör Makinesi algoritması geliştirilmiş ve çapraz-proje hata kestirimine uygulanmıştır. Gerçekleştirilen iki ayrı deneyde NASA ve SOFTLAB veri setleri kullanılmıştır. İlk deneyde, öğrenme modelinin eğitiminde NASA veri seti ve testinde ise SOFTLAB verisi kullanılmıştır. İkinci deneyde ise öğrenme modelinin eğitiminde her bir NASA veri seti kullanılmış ve testinde ise kalan NASA veri setleri kullanılmıştır. Elde edilen sonuçlar, yaygın olarak kullanılan diğer yöntemlerle karşılaştırılmış ve başarılı sonuçlar elde edildiği görülmüştür.

3.5. Metrikler ve Veri Setleri

3.5.1. Metrikler

Yazılım metrikleri, yazılım modüllerinin niteliksel karakteristiklerini niceliksel olarak yansıtırlar. Metrikler, bir modülün daha anlaşılabilir bir yol ile anlatılmasını sağlarlar. C ve Ada gibi yapısal dillerde, bir metot/fonksiyon, modül olarak ele alınır; Java ve C++ gibi nesne tabanlı dillerde ise bir sınıf, modül olarak ele alınır. Ancak, veri

toplama yaklaşımı veya çalışmanın yapısı dikkate alınarak, nesne tabanlı dillerde metot/fonksiyonlar da modül olarak değerlendirmeye alınabilirler. Bir metodun özelliklerini temsil eden metrikler, metot-seviyeli metrikler olarak adlandırılır; diğer taraftan, bir sınıfın özelliklerini yansıtan metrikler ise sınıf-seviyeli metrikler olarak adlandırılırlar. Diğer bir değişle, sınıf-seviyeli metriklerin değerleri ilgili sınıfın kaynak kodunu kullanarak hesaplanırlar. Bu çalışmanın odağı nesne tabanlı dillerdir, dolayısıyla sınıf-seviyeli metrikler kullanılmıştır. Bölüm boyunca genellikle "modül" kavramı kullanılmıştır; yani, bir modül bir Java sınıfını ifade etmektedir. "Sınıf" kavramı sınıflandırma problemlerinde başka bir maksat için kullanılmaktadır, bu kavramın kullanılması ifade karmaşıklığına sebep olabilirdi. Ayrıca, Fenton ve Pfleeger (1998), metrikleri iki kategoriye bölmüşlerdir: dahili (internal) ve harici (external). Dahili metrikler; davranışına bakılmaksızın, ürünün veya kaynağın kendisi dikkate alınarak hesaplanır. Harici metrikler ise sadece ürünün davranışı (fonksiyonelliği) ile alakalıdır. Bu çalışmanın odağı dahili metriklerdir; yani, yazılım sistemlerinin kaynak kodlarıdır (davranış veya fonksiyonelliği değil).

CK, nesne tabanlı programlama paradigmalarını ölçen bir metrik kümesidir (Chidamber ve Kemerer, 1994). Bu set, birçok çalışma (Catal ve Diri, 2009b; Radjenović ve ark., 2013) ile deneysel olarak doğrulanmış altı adet yaygın olarak bilinen metrik içerir. CK metriklerinin yanı sıra, bu çalışmanın deneylerinde, eşik değerlerinin türetilmesi amacıyla diğer metriklerin de geçerlilik sonuçları analiz edilmiştir. Radjenović ve arkadaşları (2013), yaptıkları sistematik literatür taraması ile farklı metrikleri kullanan yazılım hata kestirim çalışmalarını kapsamlı olarak derlemişlerdir. Bu çalışma kapsamında kullanılan metriklerin listesi Tablo 3.2.'de verilmiştir. Çevrimsel Karmaşıklık (Cyclomatic Complexity (CC)) haricindeki metrikler sınıf-seviyeli metriklerdir. Bu sebeple, CC'yi sınıf-seviyeli hale dönüştürmek için bu metrikten MAX_CC ve AVG_CC metrikleri türetilmiştir.

Tablo 3.2. Bu çalışmada kullanılan yazılım kalite metrikleri (Jureczko ve Spinellis, 2010)

Kısa Adı	Metrik Grubu	Açıklama
WMC	CK (Chidamber and Kemerer, 1994)	*Sınıfın Ağırlıklı Metot Sayısı (Weighted Methods per Class):* Bir sınıf içerisindeki bütün metotların karmaşıklarının toplamıdır. Eğer her bir metodun karmaşıklığı 1 olarak kabul edilirse, sınıftaki toplam metot sayısıdır.
DIT	CK	*Kalıtım Ağacının Derinliği (Depth of Inheritance Tree):* Sınıftan kalıtım ağacının köküne olan uzaklıkların en fazla olanıdır.
NOC	CK	*Alt Sınıf Sayısı (Number of Children):* Sınıftan türetilen alt sınıf sayısıdır.
CBO	CK	*Nesne Sınıfları Arası Bağımlılık (Coupling Between Object Classes):* Sınıfın bağımlı olduğu toplam sınıf sayısıdır (Afferent couplings + Efferent couplings). Bağımlılık, başka sınıfın metodunun çağrılması, değişkenine erişim veya dönüş tipine erişim şeklinde olabilir.
RFC	CK	*Sınıfın Tetiklediği Metot Sayısı (Response For a Class):* Sınıfın nesnesinden potansiyel olarak alınan bir mesaja karşılık aynı sınıf içerisinde tetiklenen metot sayısı. Kolaylık açısından, metrik değeri, mevcut lokal metot sayısı ile lokal metotlar tarafından çağrılan metot sayısı toplanarak hesaplanır.
LCOM	CK	*Metotların Uyumluluğu (Lack of Cohesion in Methods):* Sınıfın uyumu, sınıf içerisindeki metotların birbirleriyle ne kadar güçlü ilişkiye sahip olduğudur. Sınıf içerisindeki bazı metot çiftleri, en az bir ortak değişken paylaşırlar, bazı çiftler arasında ise değişken paylaşımı olmaz. Sonraki durumu sağlayan çift sayısının, önceki durumu sağlayan çift sayısından çıkartılması ile bu metrik değerinin hesaplaması yapılır. Bu metriğin yüksek değerleri; sınıf içerisinde ayrık fonksiyonelliklerin olduğunu gösterir.
Ca	Martin (1994)	*İçe Bağımlılık (Afferent couplings):* Mevcut sınıfa bağımlı diğer sınıfların sayısıdır.
Ce	Martin (1994)	*Dışa Bağımlılık (Efferent couplings):* Mevcut sınıfın bağımlı olduğu diğer sınıfların sayısıdır.
LCOM3	Henderson-Sellers (1996)	*Metotların Uyumluluğu (Lack of Cohesion in Methods):* Aşağıdaki formüle göre sınıf içi uyum seviyesini hesaplanır: *LCOM3 = (m - sum(mA)/a) / (m-1)* *m: Sınıftaki metot sayısı* *a: Sınıftaki değişken sayısı* *mA: Bir değişkene erişen metot sayısı* *sum(mA): Her değişken bazındaki hesaplanan mA değerlerinin toplamı* Bu metriğin değeri 0 ile 2 arasında değişmektedir. 0, sınıfın iyi uyumlu olduğunu ve 1 ise kötü uyumlu olduğunu gösterir. Bazı değişkenlerin hiç kullanılmaması veya sadece diğer sınıflar tarafından kullanılması durumunda 1 ile 2 arasında bir değer alır (Zimmermann ve ark., 2009).
NPM	QMOOD (Bansiya ve Davis, 2002)	*Public Metot Sayısı (Number of Public Methods):* Sınıf içerisindeki public tipindeki metot sayısıdır. Basitçe, "public" anahtar sözcüğüne sahip metot sayılarının toplanması ile hesaplanır.
DAM	QMOOD	*Veriş Erişim Metriği (Data Access Metric):* Sınıf içerisindeki private ve protected tipindeki değişken sayısının toplam değişken sayısına oranıdır. Bu metrik sınıfın kapsülleme (encapsulation) seviyesini gösterir.
MOA	QMOOD	*Birleşim Ölçütü (Measure of Aggregation):* Değişkenlerin tiplerine göre parça-bütün ilişkisini hesaplar. Kullanıcı tanımlı sınıflara ait veri değişkenlerinin toplanması şeklinde hesaplanır.

Tablo 3.2. (Devamı)

Kısa Adı	Metrik Grubu	Açıklama
MFA	QMOOD	*Fonksiyonel Soyutlama Ölçütü (Measure of Functional Abstraction):* Üst sınıftan kalıtımı yapılan metot sayısının, o sınıftaki metot sayısına oranıdır (kalıtımı yapılan sınıftaki metot sayısı da dahil edilmelidir).
CAM	QMOOD	*Sınıf Metotları Arası Uyum (Cohesion Among Methods of Class):* metotların kullandığı parametre listesi baz alınarak metotlar arasındaki ilişkinin (bağın) seviyesini gösterir.
IC	Tang (Tang ve ark., 1999)	*Kalıtım Bağlantısı (Inheritance Coupling):* Mevcut sınıfın türetildiği ana sınıf sayısının toplamıdır. Bir sınıf aşağıdaki durumlardan biri gerçekleşirse ana sınıfı ile bağlantılıdır: - Türetilen metot, tekrar düzenlenmiş metodu çağırırsa - Türetilen metot, yeni veya tekrar düzenlenmiş başka bir metoda ait en az bir değişkeni kullanırsa - Türetilen metot, yeni veya tekrar düzenlenmiş başka bir metot tarafından çağrılırsa
CBM	Tang	*Metotlar Arası Bağlantı (Coupling Between Methods):* Türetilen metodun bağlı olduğu yeni/tekrar düzenlenmiş metot sayısıdır. Bağlı olma durumu, IC metriğinde listelenen üç durumdan en az birinin var olmasıyla gerçekleşir.

3.5.2. Veri setleri

Yapılan deneylerde, PROMISE (PRedictOr Models In Software Engineering) veri havuzundan (Jureczko ve Madeyski, 2010; Menzies ve ark., 2012) 37 veri seti kullanılmıştır. Bu veri setleri, Java tabanlı açık kaynak kodlu 10 sistemin birden fazla sürümünden oluşturulmuştur. Bu veri setleri, 20 metrik değeri ve çıktı olarak ilgili modülde bulunan hata sayısını içermektedir. Modüllerin metrik değerleri, ckjm[3] aracı (Spinellis, 2005) kullanılarak hesaplanmıştır. ckjm aracı, derlenmiş Java dosyalarının baytkodunu işleyerek, nesne tabanlı yazılım metriklerini hesaplar. Bir başka araç olan BugInfo[4] ise değişiklik (commit) geçmişini, kurallı ifadeler (regular expression) ile analiz eder ve yüklenen bir değişikliğin (commit) hata giderme aktivitesi olup olmadığına karar verir. Kaynak kodda yapılan bir güncellemeye ait yüklenen bir değişiklik aşağıda verilen kurallı ifadeye uyuyorsa, bu güncelleme bir hata giderme aktivitesi olarak hesaba katılır. Sonrasında, BugInfo, yüklenen bu değişiklikle alakalı bütün modüllerin hata sayısını bir arttırır.

[3] ckjm aracı, http://www.spinellis.gr/sw/ckjm/ adresinde bulunmaktadır.

[4] BugInfo aracı, https://kenai.com/projects/buginfo adresinde bulunmaktadır.

.* [bB][uU][gG][fF][iI][xX]
| [bB][uU][gG][zZ][iI][lL][lL][aA] .*

10 açık kaynak kodlu sistemin ilk versiyonları bir dosya halinde olacak şekilde birleştirilir ve eğitim veri seti olarak kullanılır. Kalan 27 veri seti ise türetilen eşik değerlerinin etkinliğini doğrulamak için test veri seti olarak kullanılır. Bu sistemlerin; açıklamaları, versiyon bilgileri ve KLOC (Bin Kod Satır Sayısı) olarak büyüklüğü Tablo 3.3.'de listelenmiştir. Tabloda görüldüğü üzere; sistemlerin test versiyonlarının sayısı aynı değildir. Bu eşit olmayan dağılım validasyon safhasını etkilememektedir; çünkü, her bir veri seti eşit ağırlığa sahip ayrı bir set olarak değerlendirilmiştir.

Tablo 3.3. Bu çalışmada kullanılan veri setleri

Adı	Açıklaması	Eğitim Seti olarak Kullanılan Versiyonlar	Eğitim Seti Büyüklüğü (KLOC)	Test Seti olarak Kullanılan Versiyonlar	Test Seti Büyüklüğü (KLOC)
ant	Yazılım inşası (build) süreçlerini otomatikleştirir	1.3	37	1.4, 1.5, 1.6, 1.7	54,87,113,208
camel	Entegrasyon kütüphanesi; Java sınıfları arasında kural tabanlı iletişimi sağlar.	1.0	33	1.2, 1.4, 1.6	66,98,113
Jedit	Yazılım kodlarının yazılmasını sağlayan gelişmiş bir metin düzenleme aracıdır.	3.2	128	4.0, 4.1, 4.2, 4.3	144,153,170,202
log4j	Sistemin çalışması sırasında tanımlı mesajların kaydedilmesini sağlayan bir loglama kütüphanesidir.	1.0	21	1.1, 1.2	19,38
lucene	Gelişmiş metin arama motoru kütüphanesidir.	2.0	50	2.2, 2.4	63,102
poi	Microsoft Ofis formatlarında dosya okuma ve yazmayı sağlayan Java kütüphanesidir.	1.5	55	2.0, 2.5, 3.0	93,119,129

Tablo 3.3. (Devamı)

Adı	Açıklaması	Eğitim Seti olarak Kullanılan Versiyonlar	Eğitim Seti Büyüklüğü (KLOC)	Test Seti olarak Kullanılan Versiyonlar	Test Seti Büyüklüğü (KLOC)
synapse	Düz metin, binary ve JSON gibi formatlar arasında haberleşmeyi sağlayan yüksek performanslı bir kütüphanedir (Enterprise Service Bus (ESB)).	1.0	28	1.1, 1.2	42,53
velocity	Bir formattan başka bir formata dönüştürmeyi sağlayan Java tabanlı şablon motorudur.	1.4	51	1.5, 1.6	53,57
xalan	XML dokümanlarını farklı doküman tiplerine dönüştürmeyi sağlayan XSLT işlemcisidir.	2.4	225	2.5, 2.6, 2.7	388,412,428
xerces	XML dosyalarının işleyen bir parser kütüphanesidir.	1.2	159	1.3, 1.4	167,141

Eğitim verisi ve öğrenme algoritması, kestirim performansını etkileyen iki temel faktördür (He ve ark., 2011). Eğitim verisinin rastgele bir şekilde seçilmesi; büyük olasılıkla, yanlış sonuçlar ve kararlara sebebiyet verecek kötü bir öğrenme modelinin ortaya çıkmasına neden olacaktır (Zimmermann ve ark., 2009). Nagappan ve arkadaşları (2006), hata kestirim modellerinin ortak (üniversal) kullanımı üzerinde çalışmış ve deneylerini beş Microsoft projesi üzerinde gerçekleştirmişlerdir. Bütün projelerde uygulanabilecek tek bir metrik kümesine ulaşamamakla birlikte kestirim modellerinin eğitim ve uygulamasının, aynı veya benzer projelerde yapılması durumunda, daha doğru sonuçlar verdiğini bulmuşlardır (Nagappan ve ark., 2006). Bu sebeple, projelerin seçilmesi ve eğitim veri setlerinin oluşturulması dikkatli bir şekilde yapılmalıdır. Geçmiş çalışmalar, bazı karakteristiklere göre projelerin gruplandırılabileceğini göstermiştir (Jureczko ve Madeyski, 2010). Bu çalışmada aşağıdaki ortak karakteristikler dikkate alınarak 10 proje seçilmiştir:

- Açık kaynak kodlu
- Aynı yazılım dili (Java)

- Benzer alanlar (birçoğu metin işleme ile ilgili)
- Orta büyüklükte geliştirme ekibi (yazılım geliştirici sayısı 25'den az)
- Sınıf sayısı bakımından orta büyüklük (101 ve 1000 arasında sınıf sayısı)
- Benzer firmalar tarafından geliştirme (çoğu Apache)

Ayrıca, bu çalışmada kullanılan veri seti grubunu kullanan birçok geçmiş çalışma da bulunmaktadır (Jureczko ve Madeyski, 2010; Canfora ve ark., 2013; Herbold, 2013; Turhan ve ark., 2013; Shatnawi, 2015). Yukarıda listelenen karakteristiklere sahip diğer projeler, bu çalışmadan elde edilen bulguları kullanabilirler. Kaynak çalışmada, Eclipse, ana veri seti olarak kullanılmış olmakla birlikte bu yazılım yukarıdaki karakteristiklere uymamaktadır. Bu sebeple, bu çalışmada oluşturulan öğrenme modelinde ve türetilen eşik değerlerin uygulanmasında bu veri seti kullanılmamıştır.

3.5.3. Eğitim seti detayları

Bu çalışmada kullanılan eğitim veri seti oluşturma yaklaşımı Liu ve arkadaşları (2010) ile Turhan ve arkadaşları (2013)'nın çalışmalarına dayanmaktadır. Liu ve arkadaşları (2010), şu hipotez üzerinde çalışmışlardır: tek bir sistem kullanmaktansa birden fazla yazılım sisteminin eğitim işleminde kullanılması, öğrenme modeline ek bilgi sağlar ve kestirim performansını geliştirir. Bu araştırmacılar, NASA veri havuzundaki yedi veri setini kullanmışlar ve eğitim/test verilerinin oluşturulmasında farklı stratejiler uygulamışlardır. Bu stratejilerden biri, çoklu veri seti temelli sınıflandırıcı (multiple datasets baseline classifier) olarak adlandırılmıştır. Bu stratejide, test seti olarak bir sistem kullanılmış, geriye kalan altı sistem ise tek bir eğitim seti oluşturmak için birleştirilmiştir (Liu ve ark., 2010). Turhan ve arkadaşları (2013), tarafından yapılan çalışma ise karma proje (mixed project) verisi yaklaşımına dayanmaktadır. Bu yaklaşımda, proje-içi hata kestirimin performansını arttırmak için diğer projelerin verilerini de kullanmışlardır. Modellerini, proje-içi ve çapraz-proje verilerinin karmasından oluşturmuşlar ve sınırlı verinin olduğu durumlarda, karma proje hata kestiriminin kabul edilebilir performans ürettiği sonucuna varmışlardır. Bu sebeple, bu çalışmada da yazılım metriklerinin eşik değerlerinin türetilmesinde karma proje verileri kullanılmıştır. Genelde, yazılımın ilk versiyonuna ait veriler, analiz etmek

üzere mevcuttur; diğer versiyonlar ise sonradan yayınlanırlar. Bu sebeple, bu çalışmada da modeli daha gerçekçi yapmak için sistemlerin ilk versiyonları eğitim modelinin oluşturulmasında kullanılmıştır.

Her bir eğitim setindeki (sistemlerin ilk versiyonları) hata yüzdeleri Tablo 3.4.'de verilmiştir. Modül sayısı, ilgili setteki sınıf sayısı anlamına gelmektedir. "1+hatalı (%)" sütunu, en az bir hata (1 dahil) tespit edilen modüllere karşılık gelmektedir. Aynı şekilde; "3+hatalı (%)" sütunu ise en az üç hata (3 dahil) tespit edilen modüllere karşılık gelmektedir. Tekil eğitim setlerinin birleştirilmesinden sonra, toplamda 2819 modül elde edilmiştir; bunların %26.00'sı "1+hatalı" sınıfında ve %5.14'ü ise "3+hatalı" sınıfındadır.

Tablo 3.4. Veri setlerinde hata bulunma yüzdeleri

Veri seti	Modül Sayısı (#)	1+hatalı (%)	3+hatalı (%)
ant	125	16,00	3,02
camel	339	3,83	0,00
jedit	272	33,09	15,07
log4j	135	25,19	4,44
lucene	195	46,67	13,85
poi	237	59,49	17,72
synapse	157	10,19	0,64
velocity	196	75,00	6,63
xalan	723	15,21	1,24
xerces	440	16,14	0,45
Toplam	2819	26,00	5,14

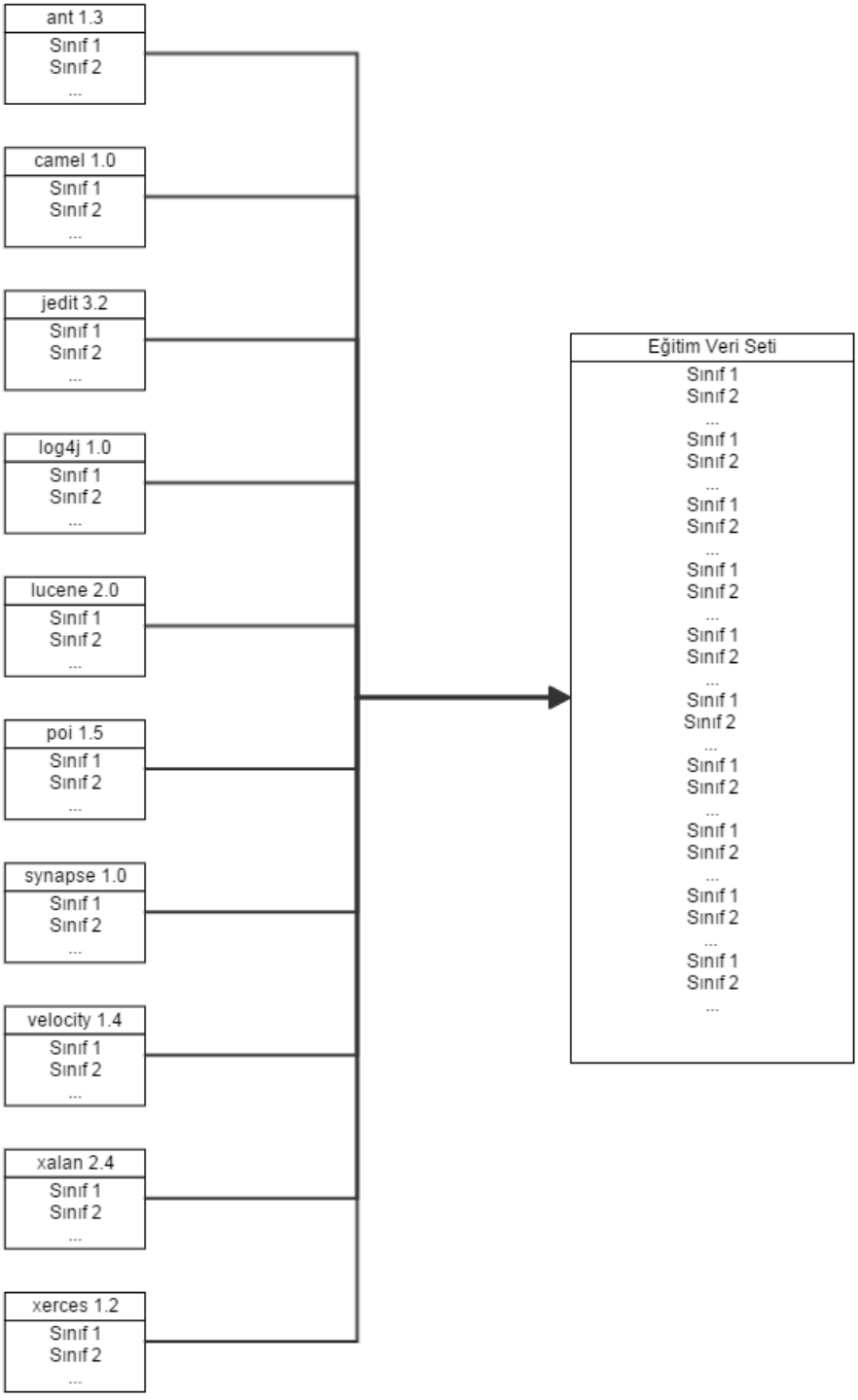
Eğitim seti baz alınarak, 20 yazılım metriğinin istatistiki karakteristikleri Tablo 3.5.'de sunulmuştur. Son sütun haricindeki bütün sütunlar, birleştirilmiş tek eğitim setinden elde edilmiştir. Şekil 3.1.'de de gösterildiği üzere, eğitim veri seti, yazılım sistemlerinin ilk versiyonlarına ait verilerin birleştirilmesi ile oluşturulur. Son sütun (St. Sap. 2), bağımsız ele alınan 10 eğitim veri setine ait metrik ortalamalarının farkını gösterir. Kartiller arası bölge (interquartile range (IQR)) sütunu, 25. persentil değeri (1. kartil) ile 75. persentil değeri (3. kartil) arasındaki farkı gösterir. Bu değerler, verinin nasıl dağılım gösterdiği hakkında fikir verirler. "N>0" sütunu, ilgili metrik değeri sıfırdan büyük olan modül sayısını göstermektedir. Tabloda görüldüğü üzere;

CK metrikleri arasında, DIT ve NOC'un değişkenliği en düşüktür. Modüllerin %75'den fazlası, 0 NOC değerine sahiptir (ortalama, 0; IQR, 0; ve $N>0$ ise sadece 330'dur). DIT'ın medyan değeri sadece 2'dir ve maksimum değeri ise 8'dir. Bu düşük değişkenlik diğer çalışmalarda da bulunmuştur (Basili ve ark., 1996; Zhou ve Leung, 2006; Jureczko ve Madeyski, 2010; Shatnawi, 2010; Herbold ve ark., 2011). Diğer taraftan; LCOM değişkenliği, 450.15'lik standart sapma ile yüksektir. Bu metriğin büyük değerleri 4. kartilde dağılmıştır (modüllerin %50'si içerisinde en büyük değer, 5; IQR, sadece 33; ve ortalama değer ise yaklaşık 90). LCOM'un bu dağılımı, kaynak çalışma ile benzerlik sergilemektedir. Bu istatistiklerden dolayı, Shatnawi (2010) çalışmasında bu metriğe yer vermemiş olmakla birlikte bu çalışmada kullanılmıştır; çünkü, bu metriğin değişkenliği, DIT ve NOC kadar düşük değildir. NPM haricindeki bütün QMOOD (Quality Model for Object-Oriented Design) metrikleri de düşük değişkenliğe sahiptir. Bütün bu sebeplerden dolayı, deneyin ileriki safhalarında şu metrikler yer almamıştır: DIT, NOC, LCOM3, DAM, MOA, MFA, CAM, IC ve CBM.

Tablo 3.5. Eğitim setinin istatistiki bilgileri [(1)]

Metrik	*N*	*N*>0	Min	Medyan	Max	Ort.	St. Sap. 1	IQR	Çarpıklık	St. Sap. 2
WMC	2819	2781	0	6	399	10,49	15,59	9	8,73	2,19
DIT	2819	2813	0	2	8	2,15	1,39	2	1,28	0,42
NOC	2819	330	0	0	100	0,53	3,25	0	15,76	0,14
CBO	2819	2633	0	6	185	10,48	14,96	9	4,68	2,87
RFC	2819	2781	0	18	487	26,82	32,86	26	4,45	6,08
LCOM	2819	2052	0	5	11469	90,20	450,15	33	12,98	47,26
Ca	2819	2166	0	2	184	5,19	13,11	3	6,06	1,38
Ce	2819	2350	0	4	93	5,90	7,43	7	3,11	2,02
LCOM3	2819	2690	0	0,87	2	1,12	0,68	1,36	0,25	0,18
NPM	2819	2687	0	5	169	8,10	10,97	7	4,57	1,94
DAM	2819	1556	0	0,5	1	0,47	0,46	1	0,09	0,19
MOA	2819	960	0	0	38	0,80	1,93	1	7,20	0,25
MFA	2819	1554	0	0,43	1	0,43	0,42	0,9	0,15	0,11
CAM	2819	2770	0	0,42	1	0,47	0,24	0,3	0,66	0,03
IC	2819	1142	0	0	5	0,59	0,86	1	1,55	0,21
CBM	2819	1142	0	0	30	1,68	3,16	2	3,14	0,91
AMC	2819	2463	0	13,60	780	22,67	35,79	22,16	8,92	6,28
MAX_CC	2819	2648	0	2	236	4,08	8,69	3	12,91	1,46
AVG_CC	2819	2648	0	1	22	1,30	1,17	0,75	4,91	0,25
LOC	2819	2795	0	107	23350	281,21	770,55	243	14,00	106,8

[(1)] Eğitim seti çalışma kapsamında seçilen yazılım sistemlerinin ilk versiyonlarının birleştirilmesi ile elde edilir. Sadece son sütun, birleştirilmemiş (bağımsız) eğitim setleri dikkate alınarak hesaplanmıştır.



Şekil 3.1. Eğitim veri setinin, yazılım sistemlerinin ilk versiyonlarının tek dosyada birleştirilmesi ile oluşturulması

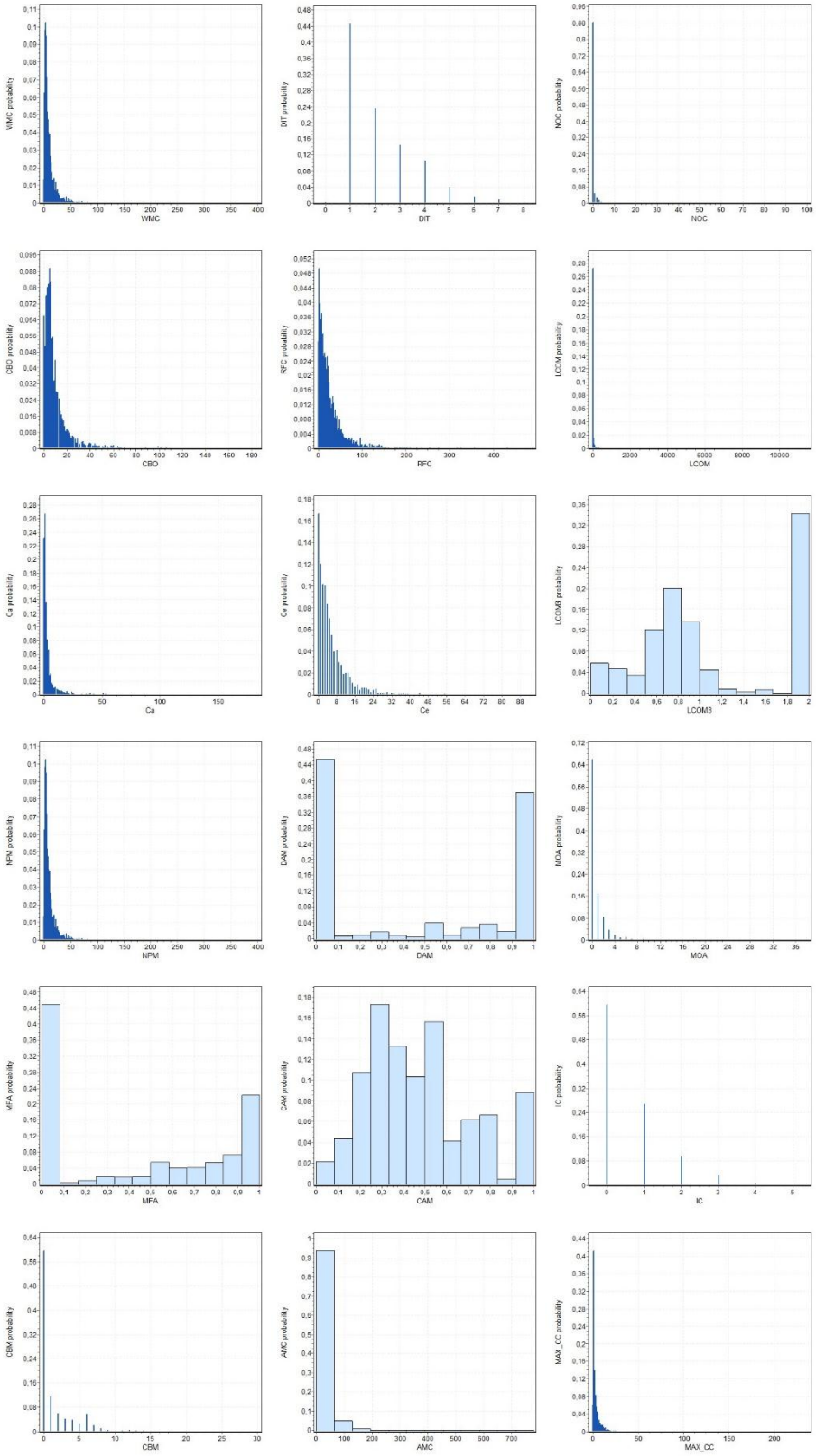
Tablo 3.5.'deki St. Sap. 2 sonuçları baz alındığında, 10 bağımsız eğitim setine ait metrik değerlerinin benzer istatistiki karakteristikler gösterdiği görülmüştür. Her bir metrik için St. Sap. 2 değeri denklem (3.1) ve (3.2) ile hesaplanmaktadır:

$$\text{Mean2} = \frac{\sum_{i=1}^{N} \mu_i}{N} \tag{3.1}$$

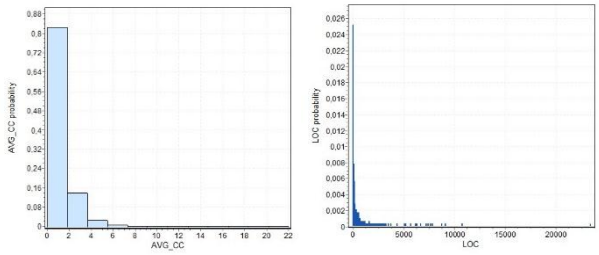
$$\text{St. Sap. 2} = \sqrt{\frac{\sum_{i=1}^{N} (\mu_i - Mean2)^2}{N-1}} \tag{3.2}$$

burada; *N*, bağımsız eğitim seti sayısını (10); μ ortalamayı; ve μ_i ise *i* no'lu eğitim veri setinde ilgili metriğin ortalamasını göstermektedir. St. Sap. 2 ve ortalama sütunları birlikte okunmalıdır. Yüksek bir ortalama değerine sahip metriğin, yüksek bir St. Sap. 2 değerine sahip olması muhtemeldir. Tabloda görüldüğü üzere; veri setleri, birbirlerinden çok farklı olmamakla birlikte aynı da değillerdir. Bu durum, üniversal eşik değerleri türetmek için iyi olabilir. Diğerlerinden farklı olarak; LCOM ve LOC metrikleri, sırasıyla 47.26 ve 106.8 standart sapma değerleri ile daha fazla değişiklik göstermektedir.

Yazılım metrikleri dengesiz dağılımlıdır; yani, veriler genellikle normal dağılım göstermezler. Yazılım metriklerinin dağılımının incelendiği birçok çalışma yapılmıştır. Bu çalışmaların ortak bulgusu, yazılım metrik verilerinin kuvvet yasasını (power law) takip ettiğidir (Baxter ve ark., 2006; Ferreira ve ark., 2012; Shatnawi ve Althebyan, 2013). Bir kuvvet yasası dağılımı, uzun kuyrukludur (heavy-tailed); yani, rastsal değişkenlerin yüksek değerleri çok seyrek, daha düşük değerler ise oldukça sıktır. Kuvvet yasası, olasılık dağılımının özel bir türüdür: Rastsal bir X değişkeninin x değerini alma olasılığı, x'in negatif gücü ile doğru orantılıdır; yani, $P(X = x) \propto cx^{-k}$ (Ferreira ve ark., 2012). Her bir metriğin dağılımı, olasılık dağılım fonksiyonu (pdf) bazında Şekil. 3.2'de gösterilmiştir. Bu fonksiyon ile rastsal bir x değerinin alacağı olasılık değeri gösterilir. Ayrıca, çarpıklık (skewness) gibi nümerik metotlar, verinin normalitesi hakkında fikir verirler. Tablo 3.5.'in sondan bir önceki sütunu çarpıklık ölçütünü içerir. Çarpıklık ölçütü, olasılık dağılımının asimetri derecesini ölçer. Eğer çarpıklık sıfırdan büyükse, dağılım sağ-kuyrukludur (right-tailed); yani, sol taraftaki örnek sayısı daha fazladır. Normal dağılımlı bir veri seti, sıfıra yakın bir çarpıklık değerine sahiptir (Shatnawi ve Althebyan, 2013).



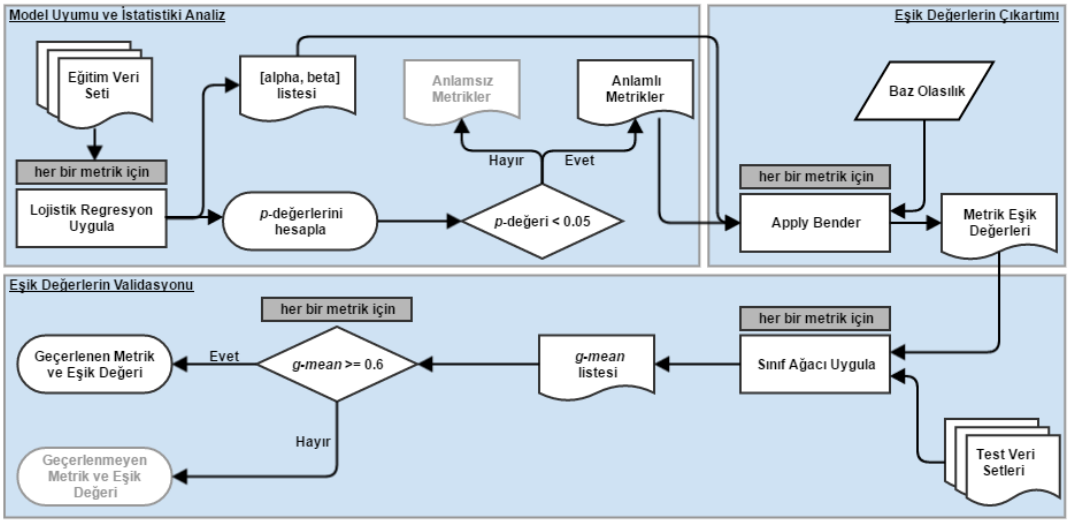
Şekil 3.2. Olasılık dağılım fonksiyonu baz alınarak her bir metriğin dağılımı



Şekil 3.2. (Devamı)

3.6. Eşik Değeri Türetim Modeli

Bu çalışmadaki eşik değeri türetim modeli, Shatnawi'nin (2010) çalışmasını esas almaktadır. Akış diyagramı şeklinde gösterimi Şekil 3.3.'de verilmiştir. Bu akış diyagramı, modelin tasarımını daha anlaşılır bir şekilde göstermektedir. Eşik değeri türetimi üç adımdan oluşmaktadır: (I) Model Uyumu ve İstatistiki Analiz, (II) Eşik Değerlerin Türetimi ve (III) Eşik Değerlerin Validasyonu. Birinci adımda; eğitim verilerine göre her bir metrik için öğrenme modelleri oluşturulur ve bu modeller, ilgili metriğin istatistiksel anlamlılığı olup olmadığını kontrol etmek için anlamlılık testine tabi tutulurlar. İkinci adımda; Bender Metot kullanılarak anlamlı metrikler için eşik değerleri türetilir. En son adımda ise; Sınıflandırma Ağacı (Classification Tree) performans sonuçlarına göre türetilen eşik değerlerinin geçerli (valid) veya geçersiz (invalid) olması durumu incelenir. Birleştirilmiş eğitim veri seti, birinci ve ikinci adımda girdi olarak kullanılır ve sonrasında üçüncü adımda öğrenme modelinin ve eşik değerlerinin performansının gözden geçirilmesinde test veri setleri kullanılır.



Şekil 3.3. Eşik değeri türetim modelinin akış diyagramı

Bütün bu adımlar ve kullanılan algoritmalar sonraki kısımlarda detaylı olarak anlatılmıştır.

3.6.1. Model uyumu ve istatistiki analiz

Tek değişkenli Lojistik Regresyon, bu çalışmadaki öğrenme modelinin oluşturulmasında kullanılmıştır. Lojistik Regresyon, bağımlı değişkenin (dependent variable) iki durumdan sadece birisi olduğu (dichotomous output) istatistiki bir sınıflandırma tekniğidir. Yazılım hata veri setleri hatalı ve hatasız modüller içerdiğinden, bu metodun kullanımı uygundur. Birden fazla bağımsız değişken, modelin girdisi olabilmektedir. Ancak, bu çalışmada her bir metrik bağımsız olarak incelenmektedir; bu da modeli tek değişkenli (univariate) yapmaktadır. Lojistik Regresyon modelinin çıktısı, iki durumdan birinin olma olasılığıdır (hataya meyilli veya hatasızlığa meyilli). Lojistik Regresyon formülü denklem (3.3)'de verilmiştir.

$$P(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} \tag{3.3}$$

burada; X, metrik değerini; β_0, eğri başlangıç (intercept) katsayısını ve β_1 ise eğim (slope) katsayısını göstermektedir. Farklı katsayı değerlerinin kullanımı, doğrusal olmayan lojistik eğrilerin oluşmasını sağlar. Bağımsız değişken (örneğin LOC metrik

değeri) ve bağımlı değişken (hatalı olma olasılığı), sırasıyla, bu eğrinin X ve Y eksenlerinde yer almaktadır. Eğrinin kavis ve doğrusalsızlık seviyesi, başlangıç ve eğim katsayıları ile belirlenir. Bu katsayıların optimum değerleri, YAK algoritması ile taranır. YAK algoritması, bal arılarının yiyecek arama davranışını simüle eder. Bu yöntem, optimizasyon problemlerinin nümerik çözümünde kullanılmak üzere (Dervis Karaboga & Basturk, 2007) tarafından önerilmiştir. YAK algoritması bu çalışmanın odağı değildir; detaylarına çeşitli referanslardan ulaşılabilir (Karaboga ve ark., 2007; Ayan ve Kılıç, 2012; Ayan ve ark., 2015).

Katsayıların istatistiki olarak anlamlı olup olmadığının belirlenmesinde, %95 güven seviyeli (α = 0.05) tek kuyruklu *p*-değeri kullanılmıştır (Bruin, 2006). Eğer, hesaplanan *p*-değeri güven aralığının dışında (0.05'in altı) ise, ilgili metrik anlamlılık testini geçer ve sonraki adımda kullanım için kaydedilir. Aksi durumda; ilgili metrik, sonraki analizlere geçirilmeden elenir.

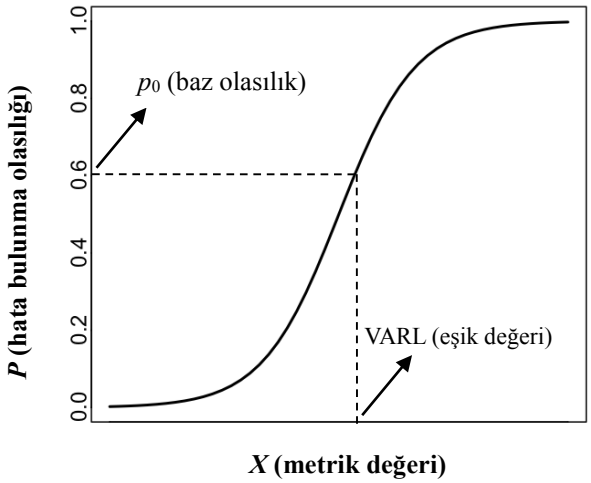
3.6.2. Eşik değerlerin türetimi

Tek kuyruklu testi geçen istatistiki anlamlı metrikler, eşik değerlerini bulmak üzere incelenirler. Eşik değerleri bulma işlemi için Bender Metodu kullanılır (Bender, 1999). Bender, Lojistik Regresyon denklemini dönüştürerek, Kabul Edilebilir Risk Değeri (Value of an Acceptable Risk Level (VARL)) adını verdiği bir yaklaşım önermiştir (Denklem (3.4)).

$$\mathrm{VARL} = \frac{1}{\beta_1}\left(\ln\left(\frac{p_0}{1-p_0}\right) - \beta_0\right) \tag{3.4}$$

burada; p_0, tanımlı baz olasılığı gösterir. Lojistik Regresyon eğrisinde yüksek bir olasılık değerine sahip bir modül hataya meyilli olarak kabul edilir; Lojistik Regresyon eğrisinde düşük bir olasılık değerine sahip bir modül ise hatasızlığa meyilli olarak kabul edilir. Diğer bir değişle, VARL'dan daha düşük bir değere sahip olan metriğin hatalı olma olasılığı, baz olasılık değerinin altındadır. Örneğin, eğer baz olasılık 0.6 (%60) olarak seçilir ve Lojistik Regresyon eğrisinde bu baz olasılığa karşılık gelen metrik değerinin 24 olduğu kabul edilirse; bu durumda, metrik değeri

24’ten küçük olan modüllerin hatalı olma olasılığı %60’ın altında olur. Diğer taraftan, metrik değeri 24’ten büyük olan modüllerin hatalı olma olasılığı ise %60’ın üstünde olur. Baz olasılık ile eşik değeri arasındaki ilişki Şekil 3.4.’de gösterilmiştir.



Şekil 3.4. Baz olasılık ile eşik değeri arasındaki ilişki

Bu çalışmadaki deneylerde belirlenen baz olasılık değerleri, ilgili kısımlarda anlatılacaktır.

3.6.3. Eşik değerlerin validasyonu

Her bir metriğe ait eşik değeri bulunduktan sonra, test veri setlerinin girdi olarak kullanıldığı bir validasyon işlemi gerçekleştirilir. Her bir test veri setindeki eşik değeri performansı, bağımsız olarak kaydedilir ve sonrasında bağımsız ve bütüncül bir şekilde analiz edilir. İlişkili eşik değerleri baz alınarak, her bir modülün hataya meyilli veya hatasızlığa meyilli olarak sınıflandırılmasında, Sınıflandırma Ağacı metodu kullanılır. Denklem (3.5), bu iki durumu göstermektedir; eğer *i* no’lu modülün *j* no’lu metrik değeri, X_{ij} , o metrik için türetilen eşik değerinden (THR_j) büyükse, o modülün sonucu hataya meyilli olarak belirlenir. Aksi durumda ise hatasızlığa meyilli olarak belirlenir (Denklem (3.5)).

$$y_i = \begin{cases} \text{hataya meyilli} & \text{eğer } X_{ij} \geq THR_j \\ \text{hatasızlağa meyilli} & \text{eğer } X_{ij} < THR_j \end{cases} \tag{3.5}$$

Sonuçları, tahmin edilen ve gerçek çıktıya göre gösteren hata matrisi Tablo 3.6.'da verilmiştir. Burada; EVET hataya meyillilik durumuna ve HAYIR ise hatasızlığa meyillilik durumuna karşılık gelmektedir.

Tablo 3.6. Hata matrisi

		Tahmin Edilen	
		EVET	**HAYIR**
Gerçek	**EVET**	TP (Doğru Pozitif)	FN (Yanlış Negatif)
	HAYIR	FP (Yanlış Pozitif)	TN (Doğru Negatif)

Bir veri setinde; bir sınıfa ait örnek sayısı, diğer sınıftaki örnek sayısından çok daha fazla ise, dengesiz sınıf dağılımlı veri olarak tanımlanmaktadır. Temel doğruluk (accuracy) ölçütünün, dengesiz sınıf dağılımlı veri setlerinde kullanımı uygun değildir (Nickerson ve ark., 2001). Örneğin; ant veri seti, %5.14 oranında 3+hatalı modül içermektedir (bkz. Tablo 3.4). Herhangi bir makine öğrenmesi tekniği kullanmaksızın bütün örnekleri 3+hm olarak tahmin eden bir tahmin edici, %94.86'lık bir doğruluk performansına sahip olacaktır. Bu çalışmada, Kubat ve Matwin (1997) tarafından sunulan geometrik ortalama (*g-mean*), performans ölçütü olarak kullanılmıştır. Verinin dengesiz dağılım sergilediği durumlar için *g-mean*'in diğer temel değerlendirme ölçütlerine göre daha güvenilir değerlendirme sonuçları verdiğini belirtmişlerdir. Çoğunluk (majority) ve azınlık (minority) sınıfının doğruluğu birlikte *g-mean* hesaplanmasında dikkate alınır (Malhotra ve Bansal, 2015). Shatnawi (2010) de çalışmasında *g-mean* kullanmıştır. Kaynak çalışma ve aynı alandaki diğer bazı çalışmalara (Seliya ve ark., 2010; Wang ve Yao, 2013) dayanarak bu çalışmada da *g-mean* kullanılmıştır. Birden fazla ölçüt kullanmak yerine tek bir ölçüt kullanılmasının, çalışmayı kolay karşılaştırılabilir yapması açısından da pratik bir faydası vardır.

g-mean formülü denklem (3.6)'da verilmiştir.

$$g\text{-}mean = \sqrt{\mathrm{TPR} \times \mathrm{TNR}} \tag{3.6}$$

g-mean denklemin geçen TPR ve TNR hesaplama yöntemi denklem (3.7) ve (3.8)'de verilmiştir.

$$\text{TPR} = \text{Duyarlılık} = \frac{\text{TP}}{\text{TP+FN}} \tag{3.7}$$

$$\text{TNR} = \frac{\text{TN}}{\text{TN+FP}} \tag{3.8}$$

burada; TPR, doğru olarak sınıflandırılan hataya meyilli modül oranını belirtmekte ve TNR ise doğru olarak sınıflandırılan hatasızlığa meyilli modül oranını belirtmektedir. *g-mean* ölçütü, bu iki oranının geometrik ortalamasıdır. Yüksek bir *g-mean* değeri elde edebilmek için TPR ve TNR'nin birlikte yüksek olması gerekmektedir. Bu iki değerden birinin düşük olması durumunda, performans da düşük olacaktır.

g-mean limit değeri olarak 0.6 kullanılmıştır. Bu limit değerinin üzerindeki validasyon sonuçları kabul edilebilir aralığı gösterir ve ilgili metrik için türetilen eşik değerinin geçerli olduğu önerilir. Gerçekte, hata kestiriminin kabul edilebilir olup olmadığını gösteren limit değeri konusunda araştırmacılar arasında bir fikir birliği yoktur. Bu çalışmada yukarıda belirtilen limit değerinin kullanılmasındaki gerekçeler aşağıda özetlenmiştir:

- Yazarın kendi araştırma, kodlama ve test tecrübesi, bu limit değerinin anlamlı olduğunu ve pratik olarak kullanışlı olduğunu göstermektedir.
- Menzies ve arkadaşları (2008, 2010) tarafından yapılan çalışmalarda, hata kestirim problemlerinde performans olarak bir üst sınıra dayandığını belirtmişlerdir; bunun sebebi olarak da statik kod özelliklerinin içerdiği bilginin sınırlı olduğunu vurgulamışlardır. Bu çalışmada da statik kod özellikleri kullanıldığından, deneylerde çok yüksek performans sonuçlarının alınması zordur. Bazı çalışmalar, kendi sonuçlarını, duyarlılık ve kesinlik gibi alternatif ölçütler ile değerlendirmişlerdir. Zimmermann ve arkadaşları (2009), sınıflandırıcı performansına ait duyarlılık, kesinlik ve doğruluk sonuçlarının 0.75 üzerinde olması durumunda iyi olarak nitelemişlerdir. Bir çok çalışma duyarlılık için 0.7 ve kesinlik için 0.5 baz değerlerini kullanmışlardır (He ve ark., 2011; Herbold, 2013). Bunlardan farklı olarak, bu çalışmada *g-mean*

ölçütü kullanılmıştır. Duyarlılık ve TPR farklı isimlendirmeler olsa da aynı ölçütlerdir. Ancak, kesinlik ve TNR farklıdır. Bu farka rağmen; önceden bahsedilen çalışmalardaki duyarlılık ve kesinliğin baz değerlerinin (0.7 ve 0.5) aritmetik ortalaması, *g-mean* limit değerinin önerilmesinde dikkate alınmıştır.

- Kaynak çalışmada, Shatnawi, üç metrik için 0.59, 0.64 ve 0.61 *g-mean* sonuçlarına ulaşmış ve bu sonuçların etkili olduğu şeklinde yorumlamıştır. Bu yoruma dayanarak, bu çalışmada önerilen limit değerinin mantıklı olduğu söylenebilir.

Bunların yanı sıra, önerilen limit değerinin pratik yönünü göstermesi açısından Tablo 3.7. yararlı olacaktır. Tabloda, 0.6 *g-mean* sonucunu veren üç TPR ve TNR örneği yer almaktadır. Üçüncü örnek, hatalı modüllerin %70'inin model tarafından yakalandığı anlamına gelmektedir. Eğer, bir yazılım 10 hatalı modül içeriyorsa, bunların 7'si yakalanmıştır. Bu modüllerin gözden geçirilmesi ve düzenlenmesi sonrasında yazılımın yayınlanması, test personelinin iş yükünü hafifletecektir. 10 modülün tamamına odaklanmaktansa, test personeli 3 modüle odaklanmış olur.

Tablo 3.7. 0.6 *g-mean* sonucu üreten TPR ve TNR örnekleri

TPR	TNR	*g-mean*
0.6	0.6	0.6
0.51	0.70	0.6
0.70	0.51	0.6

3.6.4. Kaynak çalışma ile farklar

Şekil 3.3.'de gösterimi yapılan eşik değeri türetim modeli, çoğunlukla kaynak çalışma ile aynıdır. Sadece aşağıda listelenen küçük değişiklikler yapılmıştır:

- VARL formülündeki baz olasılık seçimi, ilgili veri setindeki hatalı modül oranına göre yapılmıştır (şekilde gösterilmemiştir). Kaynak çalışmada, Shatnawi, deneylerini 0.050, 0.060, 0.065, 0.075, ve 0.100 gibi farklı baz olasılıklılarına göre kurgulamıştır. Bu çalışmadaki deneylerde ise aynı baz olasılık kullanımı ile iyi sonuçlar elde edilmemiştir.

- Lojistik Regresyon formülündeki optimum katsayıların bulunması için YAK algoritması kullanılmıştır (şekilde gösterilmemiştir). Kaynak çalışmada hangi optimizasyon algoritmasının kullanıldığından bahsedilmemiştir. YAK algoritmasının birçok avantajı vardır. Bazıları şunlardır: bütün mühendislik problemlerine yönelik kodlanması kolaydır; kapsamlı bir uygulanabilirliği vardır (aynı zamanda sürekli, ayrık veya karışık değişkenlerin olduğu kompleks fonksiyonlarda); lokal çözümleri bulma yeteneği vardır; erken yakınsama riskine sahip yüksek karmaşıklık altında dahi arama işleminde etkilidir; ilk çözüm popülasyonundaki dağılım ve fizibilitesinden etkilenmez, dirençlidir. YAK'ın diğer çok bilinen optimizasyon algoritmalarıyla yapılan bir karşılaştırma çalışması; bu algoritmanın sınıflandırma problemlerinde daha iyi sonuç verdiğini ortaya çıkarmıştır (Karaboga ve Ozturk, 2009).
- Metriklerin eşik değerlerinin sonuçları, 0.6 *g-mean* limit değerine göre yorumlanmıştır (şekilde gösterilmiştir). Kaynak çalışmada, Shatnawi, üç metrik eşik değeri için 0.59, 0.64, ve 0.61 *g-mean* sonuçlarına ulaşmış ve bu sonuçları etkili olarak yorumlamıştır; ancak *g-mean* limit değerinden bahsetmemiştir.

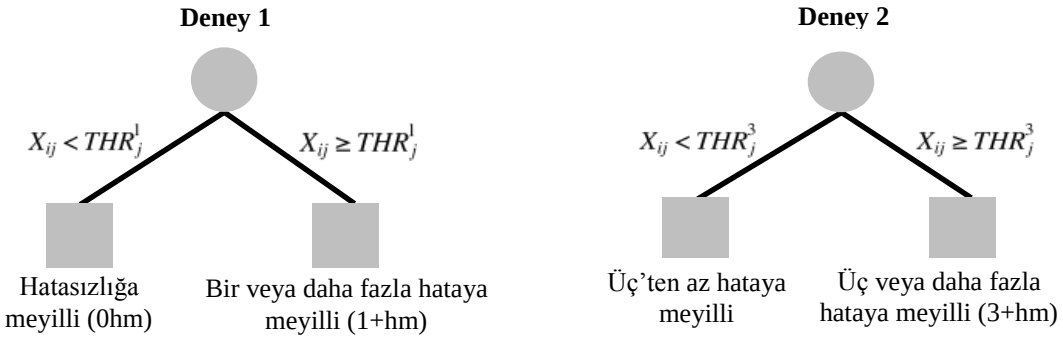
3.7. Deneyler ve Bulgular

Bu çalışmada iki farklı eşik değeri kümesi için iki farklı deney uygulanmıştır. Birincisi, modülleri 0hm veya 1+hm olarak kategorize etmek için türetilen eşik değerleridir ve ikincisi, modüllerin 3+hm olup olmadığının kategorizasyonu içindir. Bu iki eşik değerleri, sırasıyla, Eşik Değeri-1 (THR^1) ve Eşik Değeri-3 (THR^3) olarak adlandırılmıştır.

- Eşik Değeri-1 (THR^1): Bu eşik değerinin üzerinde metrik değerlerine sahip yazılım modülleri bir veya daha fazla hata bulundurmaya meyillidir (1+hm). Tersi durum; yani, bu eşik değerinin altında metrik değerine sahip olan modüller ise sıfır hataya meyilliliği gösterir (0hm).
- Eşik Değeri-3 (THR^3): Bu eşik değerinin üzerinde metrik değerlerine sahip yazılım modülleri üç veya daha fazla hata bulundurmaya meyillidir (3+hm). Tersi durum; yani, bu eşik değerinin altında metrik değerine sahip olan

modüller ise üç'ten az hataya meyilliliği gösterir (hiç hata bulunmaması durumu dahil).

Kolaylık ve anlaşılabilirlik açısından, önceki bölümlerde yer alan açıklamalar, sadece 0hm ve 1+hm'ye göre yapılmış, 3+hm'den çok bahsedilmemiştir. İki eşik değeri kümesinin gösterimi Şekil 3.5.'de yapılmıştır.



Şekil 3.5. Eşik Değeri-1 ve Eşik Değeri-3'un fonksiyonelliği (Sınıflandırma Ağacı gösterimi ile)

Öğrenme modelinin oluşturulmasında ve iki deneye ait kodlamaların gerçekleştirilmesinde MATLAB platformu kullanılmıştır.

3.7.1. Deney 1: Eşik Değeri-1 türetimi

Öncelikle, en az bir hataya sahip olan modüllerin çıktısının 1 olarak etiketlenmesi ile bu deney için birleştirilmiş eğitim verisi hazır hale getirilir. Sonrasında, eğitim verisi kullanılarak, YAK optimizasyonu destekli Lojistik Regresyon çalıştırılır. Aday başlangıç ve eğim katsayılarına göre her bir metrik için Lojistik Regresyon eğrileri elde edilmiş olunur. Bu eğrilerde, 0hm ve 1+hm modüllerini ayırt etmek için yatay karar doğruları kullanılır. Hata çıkma olasılığı, bu karar doğrusunun üzerinde olan modüller 1+hm olarak; hata çıkma olasılığı bu karar doğrusunun altında olanlar ise 0hm olarak kabul edilir. Olasılıksal karar doğrusu (p(kd)), her bir modülün çıktısı (y_i) ve minimize edilecek olan maliyet fonksiyonu (*Cost*) sırasıyla denklem (3.9), (3.10) ve (3.11)'de verilmiştir.

$$p(\text{kd}) = \frac{N(1+hatalı)}{N} \tag{3.9}$$

$$y_i = \begin{cases} 0\text{hm} & \text{eğer } P(X_{ij}) < p(\text{kd}) \\ 1 + \text{hm} & \text{eğer } P(X_{ij}) \geq p(\text{kd}) \end{cases} \tag{3.10}$$

$$Cost = 1 - g\text{-}mean \tag{3.11}$$

burada; *N*(1+hatalı), toplam *N* adet modül içerisindeki hatalı modülleri belirtir. Tablo 3.4. tekrar incelendiğinde; bütün modüllerin %74'ünün sıfır hataya sahip olduğu görülür. Bu sebeple, karar doğrusunun olasılık değeri, bu deney için 0.74 olarak belirlenmiştir. YAK'ın her bir iterasyonunda, tahmin edilen katsayılar ile Lojistik Regresyon eğrisi oluşturulur. *i* no'lu modülün çıktısı, y_i; incelenen metriğe (*j*) karşılık gelen hatalı çıkma olasılığı, $P(X_{ij})$, *p*(kd)'den büyükse 1+hm olarak tahmin edilir; tersi durumda ise 0hm olarak tahmin edilir. Hatalı çıkma olasılığı, $P(X_{ij})$, Lojistik Regresyon eşitliği kullanılarak hesaplanır (Denklem (3.1)). Bu karşılaştırmanın bütün modüllerde yapılması ile TP, TN, FP, FN sayıları elde edilmiş olur. Bu sayılarak kullanılarak *g-mean* hesaplaması yapılır. YAK'ın amacı, *Cost*'u minimize etmektir; diğer bir deyişle, g-*mean*'i maksimize etmektir. Her bir iterasyonda elde edilen *Cost* değerine göre Lojistik Regresyon katsayıları güncellenir. Ön tanımlı şartlar sağlandığında, başlangıç ve eğim katsayıları, Bender Metodunda kullanılmak üzere kaydedilir. Bender Metodu, Lojistik Regresyon katsayılarının yanında, baz olasılık değerine (p_0)'ye de ihtiyaç duyar. Bu çalışmada, karar doğrusuna ait olasılık değeri (*p*(kd)) baz olasılık değeri olarak kullanılır ve sonuç olarak buna karşılık gelen *X* değeri, metriğin eşik değeri olarak kabul edilir. Bu deneyde, her bir metrik için elde edilen optimal katsayılar, *p*-değeri ve türetilen eşik değerler Tablo 3.8.'de gösterilmiştir.

Tablo 3.8. Lojistik Regresyon ve Bender Metodu sonuçları (Deney 1)

Metrik	Başlangıç	Eğim	*p*-değeri	Eşik Değeri
WMC	-0,616	0,233	0,035	7,13
CBO	-4,000	0,666	0,031	7,58
RFC	-0,973	0,100	0,009	20,19
LCOM	-0,603	0,222	0,000	7,42
Ca	-2,289	2,741	0,000	1,22
Ce	-0,572	0,407	0,000	3,98
NPM	-0,183	0,251	0,000	4,90
AMC	-1,145	0,217	0,011	11,37
MAX_CC	-4,000	2,936	0,000	1,72
AVG_CC	-1,469	2,449	0,000	1,03
LOC	-3,779	0,042	0,000	114,11

p-değeri sütununda görüldüğü üzere; bütün metrikler istatistiki olarak anlamlıdır (0.05'den düşük değerler olduğundan). Beklendiği üzere, bütün eğim katsayıları pozitiftir; bu durum, hataya meyillilik olasılığı ile metrik değerinin aynı yönlü olduğunu göstermektedir (yani, daha yüksek metrik değeri, daha yüksek hataya meyilliliği ifade eder). Tablonun en son sütununda, türetilen eşik değerleri verilmiştir.

Bu eşik değerlerinin değerlendirilmesi, toplamda 12090 modülden oluşan, 27 test veri seti kullanılarak yapılmıştır. *g-mean* bazında sonuçlar Tablo 3.9.'da özetlenmiştir.

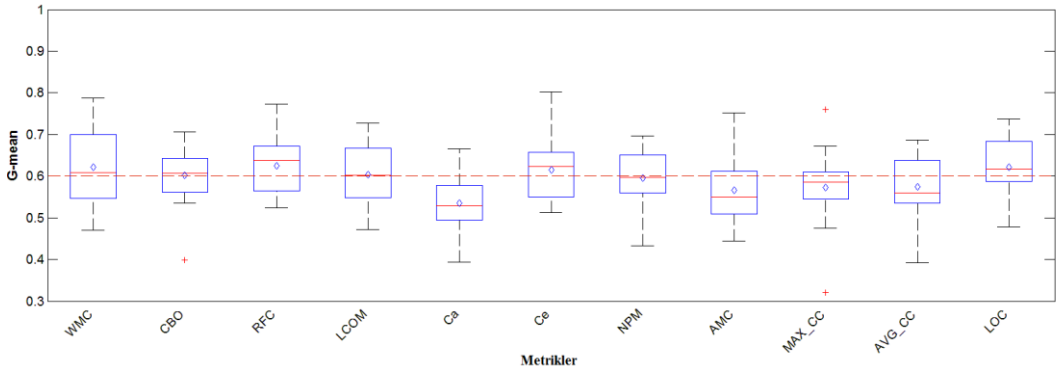
Tablo 3.9. Metrik bazında *g-mean* sonuçlarının detayları (Deney 1)

Metrik	Min *g-mean*	Max *g-mean*	St. Sap.	*N*(*g-mean*)>0.6	Ort. *g-mean*
WMC	0,47	0,79	0,08	15	0,62
CBO	0,40	0,69	0,15	16	0,60
RFC	0,52	0,77	0,07	17	0,63
LCOM	0,47	0,73	0,08	14	0,60
Ca	0,39	0,67	0,06	4	0,54
Ce	0,51	0,80	0,08	14	0,62
NPM	0,43	0,70	0,07	14	0,60
AMC	0,44	0,75	0,08	9	0,57
MAX_CC	0,32	0,76	0,08	11	0,57
AVG_CC	0,39	0,69	0,07	11	0,57
LOC	0,48	0,74	0,07	19	0,62

Tablodaki Min *g-mean* ve Max *g-mean* sütunları, test veri setleri arasında en iyi ve en kötü performans sonuçlarını göstermektedir. St. Sap. sütunu, her bir veri setindeki performansın ne kadar farklı olduğunu belirtir. *N*(*g-mean*)>0.6, ön tanımlı limit değerinin (0.6) üzerinde olan test veri setlerinin sayısı anlamına gelmektedir. Son sütun ise bütün *g-mean*'lerin ortalamasıdır. Yapılan bu deney sonucunda, yedi metriğin kabul edilebilir sonuçlar ürettiği ortaya çıkmıştır (*g-mean*'i 0.6'dan büyük olanlar). Bu durum, bu metriklere ait türetilen eşik değerlerinin geçerli olduğunu göstermektedir. Bölüm 3.3.2'de verilen sıfır hipotezin ret edildiği, yani alternatif hipotezin kabul edildiği metrikler şunlardır: WMC, CBO, RFC, LCOM, Ce ve LOC. Diğer taraftan, dört metrik tatminkar sonuçlar vermemiştir (*g-mean*'i 0.6'dan küçük olanlar). Bu durum, bu metriklere ait türetilen eşik değerlerinin geçerli olmadığını göstermektedir. Sıfır hipotezin kabul edildiği, yani alternatif hipotezin ret edildiği metrikler şunlardır: Ca, AMC, MAX_CC ve AVG_CC. Ancak, Ca haricindeki bütün metrikler en düşük 0.57 *g-mean*'e sahiptir. Bu durum, ilgili metriklerin eşik değerlerinin de modülün hataya meyilliliği konusunda fikir verebileceğini göstermektedir. 27 test setinden 17'sinde 0.6'dan büyük *g-mean* değerini üreten RFC,

metrikler arasında en iyi sonuca sahiptir. İlginç bir bulgu ise Ce metriğinin kabul edilebilir eşik değerine sahipken, Ca metriğinin olmamasıdır. Bu durum, şu anlama gelmektedir: Diğer modüllere bağımlılığı olan bir modül daha fazla hataya meyillidir. Ancak, daha sorumlu bir modülün daha fazla hataya meyilli olduğu yargısına varılamaz.

Daha detaylı bir istatistiki gösterim, kutu diyagramları ile Şekil 3.6.'da yapılmıştır. Kutu diyagramları, sonuçların dağılımını göstermek için uygun bir yöntemdir (Williamson ve ark., 1989).



Şekil 3.6. *g-mean* sonuçlarının kutu diyagramları ile gösterimi (Deney 1)

Kesik çizgili yatay doğru, bu çalışma kapsamında belirlenen limit değerini temsil etmektedir. Her bir kutu içerisindeki küçük sembol, ortalama değerini ve düz çizgiler ise medyan değerlerini göstermektedir. Kararlılık yönüyle en iyi sonucu LOC metriği vermiştir; hemen hemen kutunun alt bölümü (1. kartil) limit doğrusunun üstündedir. Diğer bir değişle, LOC metriğine ait türetilen eşik değeri üniversal kullanıma daha uygundur; yani, diğer benzer yazılım sistemlerinde daha az çekince ile uygulanabilir. Kararlılık yönüyle LOC'dan sonra gelen metrikler sırasıyla, RFC, CBO ve WMC'dir.

3.7.2. Deney 2: Eşik Değeri-3 türetimi

Öncelikle, en az üç hataya sahip olan modüllerin çıktısının 1 olarak etiketlenmesi ile bu deney için birleştirilmiş eğitim verisi hazır hale getirilir. Sonrasında, eğitim verisi kullanılarak, YAK optimizasyonu destekli Lojistik Regresyon çalıştırılır. Bu deneyde

kullanılan karar doğrusu, 3+hm ile 3'den az hataya meyilli modülleri ayırt etmek için kullanılır (Denklem (3.12)).

$$y_i = \begin{cases} 3'\text{den az hataya meyilli} & \text{eğer } P(X_{ij}) < p(\text{kd}) \\ 3 + \text{hm} & \text{eğer } P(X_{ij}) \geq p(\text{kd}) \end{cases} \quad (3.12)$$

Tablo 3.4 tekrar incelendiğinde; bütün modüllerin %94.86'sının üç'ten az hataya sahip olduğu görülür. Bu sebeple, karar doğrusunun olasılık değeri, bu deney için 0.9486 olarak belirlenmiştir. Eğer, bir modül için Lojistik Regresyon denkleminin çıktısı, bu karar doğrusunun üzerindeyse, bu modül 3+hm olarak; hata çıkma olasılığı bu karar doğrusunun altındaysa 3'den az hataya meyilli olarak kabul edilir.

Bu deneyde, her bir metrik için elde edilen optimal katsayılar, *p*-değeri ve türetilen eşik değerler Tablo 3.10.'da gösterilmiştir.

Tablo 3.10. Lojistik Regresyon ve Bender Metodu sonuçları (Deney 2)

Metrik	Başlangıç	Eğim	*p*-değeri	Eşik Değeri
WMC	-3,513	0,384	0.042	11,87
CBO	-4,000	0,624	0.011	8,09
RFC	-1,905	0,099	0.001	29,71
LCOM	0,266	0,037	0.000	20,83
Ca	-0,265	0,599	0.000	2,19
Ce	-0,751	0,268	0.017	6,71
NPM	-0,072	0,159	0.000	7,05
AMC	-2,922	0,230	0.021	17,22
MAX_CC	-1,751	0,755	0.000	3,70
AVG_CC	-0,732	1,733	0.000	1,03
LOC	-3,093	0,024	0.000	170,91

p-değeri sütununda görüldüğü üzere; bütün metrikler istatistiki olarak anlamlıdır. Bütün eğim katsayıları pozitiftir. Tablonun en son sütununda, türetilen eşik değerleri verilmiştir.

Bu eşik değerlerinin değerlendirilmesi, toplamda 10991 modülden (sınıf) oluşan, 24 test veri seti kullanılarak yapılmıştır. ant-1.5, jedit-4.3 ve poi-2.0 test setleri üç'ten fazla hataya sahip modül içermemektedir; bu sebeple, bunlar, bu deney çalışmasından çıkarılmıştır. *g-mean* bazında sonuçlar Tablo 3.11.'de özetlenmiştir.

Tablo 3.11. Metrik bazında *g-mean* sonuçlarının detayları (Deney 2)

Metrik	Min *g-mean*	Max *g-mean*	St. Sap.	*N(g-mean)*>0.6	Ort. *g-mean*
WMC	0,41	0,85	0,11	20	0,70
CBO	0,54	0,82	0,07	22	0,69
RFC	0,60	0,82	0,06	24	0,72
LCOM	0,47	0,84	0,11	14	0,65
Ca	0,35	0,72	0,08	14	0,61
Ce	0,51	0,83	0,08	20	0,68
NPM	0,46	0,83	0,11	13	0,66
AMC	0,43	0,80	0,08	14	0,62
MAX_CC	0,53	0,76	0,08	14	0,64
AVG_CC	0,50	0,79	0,07	17	0,64
LOC	0,61	0,77	0,05	24	0,71

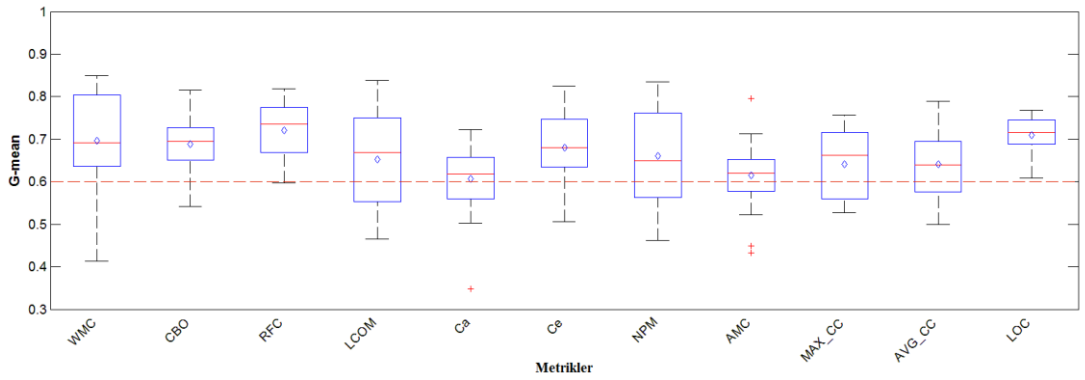
Tablonun son sütununda görüldüğü üzere; bütün metrikler makul sonuçlar üretmiştir. Bölüm 3.3.2'de verilen sıfır hipotez, bütün metrikler için ret edilmiş; yani alternatif hipotez bütün metrikler için kabul edilmiştir. Bu deneyde elde edilen sonuçlar, bir önceki deney ile karşılaştırıldığında, oldukça iyidir. Bu bulgular ışığında şu tespit yapılabilir: 3+hatalı modüllere ait metrik değerlerinin ayırt edici yönü daha fazladır. Tablo 3.12., 1+hatalı modüller ile 3+hatalı modüllerin metrik değerlerinin ortalamalarının farklarını göstermesi bakımımdan bilgilendiricidir. Bu tablonun

birleştirilmiş eğitim veri setinden oluşturulduğu dikkate alınmalıdır. En büyük fark, sırasıyla ortalama 431 ve 741 kod satır sayısı ile LOC metriğindedir. Tablo 3.11.'deki değerlendirmelere dönülecek olursa; RFC metriği için ortalama *g-mean* değeri 0.72 ve LOC metriği için ise 0.71'dir. En düşük performans Ca metriğine aittir; buna rağmen, bu metriğe ait ortalama *g-mean* değeri ön tanımlı limit değerinin üzerindedir.

Tablo 3.12. Eğitim verisinde yer alan 1+hatalı ve 3+hatalı modüllerin metrik değerlerinin ortalamaları

Ortalama / Metrik	1+hatalı	3+hatalı
WMC	14,84	21,81
CBO	13,52	20,23
RFC	40,10	66,29
LCOM	161,77	374,95
Ca	7,16	11,07
Ce	7,44	11,23
NPM	11,22	15,40
AMC	26,68	32,56
MAX_CC	5,78	8,66
AVG_CC	1,51	1,86
LOC	431,61	741,88

Sonuçların, kutu diyagramları şeklinde gösterimi Şekil 3.7.'de verilmiştir.



Şekil 3.7. *g-mean* sonuçlarının kutu diyagramları ile gösterimi (Deney 2)

RFC ve LOC metriklerinin eşik değerleri, tüm 24 test setinde 0.6'dan büyük *g-mean* sonucu üretmiştir. Diğer bir değişle, RFC ve LOC metriklerine ait türetilen eşik

değerleri üniversal kullanıma daha uygundur; yani, diğer benzer yazılım sistemlerinde daha az çekince ile uygulanabilir. Kararlılık yönüyle RFC ve LOC'dan sonra gelen metrikler sırasıyla, CBO, WMC ve Ce metrikleridir. Bu deneydeki kararlılık sıralaması, birinci deney ile nerdeyse aynıdır. Test setleri bazında değerlendirme yapılacak olursa; bazıları eşik değerlerine karşılık zayıf sonuçlar vermiş, bazıları ise çok iyi sonuçlar vermiştir. Örneğin; log4j-1.2 ortalama 0.57 *g-mean* performansına sahipken, log4j-1.1'nin performans ortalaması 0.76'dır. Bu bulgu dikkat çekicidir; çünkü, aynı eşik değerine karşılık, aynı yazılım sisteminin farklı versiyonları çok farklı bir şekilde reaksiyon göstermiştir. Diğer sonuçlar daha detaylı bir şekilde analiz edildiğinde; aynı yazılım sisteminin farklı versiyonlarının benzer sonuçlar verdiği görülmüştür. Bu durum, log4j-1.2'nin bir istisna olduğunu gösterir; bunun olası sebepleri sonraki kısımlarda anlatılacaktır.

3.7.3. Türetilen eşik değerlerine genel bakış

Deney 1 ve Deney 2 sonucunda geçerli kılınan eşik değeri kümeleri (Eşik Değeri-1 ve Eşik Değeri-3) Tablo 3.13.'de özetlenmiştir. Pratikte, AVG_CC haricinde önerilen eşik değerleri, tamsayı (integer) olmalıdır; gerçel (real) değer değil. Bu sebeple, türetilen eşik değerleri en yakın tamsayıya yuvarlanmıştır. CBO haricindeki tüm metriklerde, THR^1 ve THR^3 arasındaki fark makuldür. CBO için iki eşik değer de aynıdır (8). Ancak, THR^3, *g-mean* performansı bakımından THR^1'den daha başarılı sonuçlar vermiştir (sırasıyla 0.69 ve 0.60; bkz. Tablo 3.11. ve Tablo 3.9.). Bu nedenle, CBO metrik değeri 8'den büyük olan bir modülün en az üç hataya meyilli (3+hm) olduğu söylenebilir. Diğer taraftan, bir modülün LOC metrik değeri 114'den fazla ise, o modülün en az bir hataya meyilli olduğu (1+hm); LOC metrik değeri 171'den fazla ise en az üç hataya meyilli (3+hm) olduğu söylenebilir. Benzer çıkarımlar tablodaki diğer metrikler için de yapılabilir.

Tablo 3.13. Deney 1 ve Deney 2 sonucunda türetilen eşik değerleri

Eşik Değeri / Metrik	THR^1	THR^3
WMC	7	12
CBO	8	8
RFC	20	30
LCOM	7	21
Ca	-	2
Ce	4	7
NPM	5	7
AMC	-	17
MAX_CC	-	4
AVG_CC	-	1
LOC	114	171

3.7.4. Sonuçların kaynak çalışma ile karşılaştırması

Kaynak çalışma ve bu replikasyon çalışmasının genel özellikleri ve bulguları Tablo 3.14.'de sunulmuştur. Kaynak çalışmada, eşik değeri deneyinde sadece üç metrik yer almıştır. Bu sebeple, bu replikasyon çalışmasının birinci deneyindeki bu üç metriğe ait sonuçlar tabloya aktarılmıştır. Kaynak çalışmada veri seti olarak Eclipse kullanılmıştır. Eclipse, IBM şirketi tarafından geliştirilen, 4454 modül (sınıf) sayısı ile büyük ölçekli bir Entegre Geliştirme Ortamı (Integrated Development Environment (IDE))'dır. Bu özellikler, bu çalışma kapsamında belirlenen gruplama karakteristiklerinden farklıdır. Bunun haricinde, kaynak çalışmanın amacı, proje-içi eşik değerlerinin tespit edilmesidir; halbuki, bu çalışmanın amacı, özel bir grup açık kaynak kodlu yazılım sistemlerinde kullanılabilecek üniversal eşik değerlerine ulaşmaktır. Bu sebeplerden dolayı, bu iki çalışma kapsamında türetilen eşik değerlerinin birebir karşılaştırması yapılamaz. Sonuçlarla ilgili dikkat çeken bir bulgu şudur: Birçok geçmiş literatür çalışmalarında proje-içi öğrenme modellerinin performans olarak daha üstün olduğu belirtilmesine rağmen (Zimmermann ve ark., 2009; Briand ve ark., 2002), bu replikasyon çalışmasında elde edilen *g-mean* sonuçları kaynak çalışmaya göre daha iyidir.

Tablo 3.14. Kaynak çalışma ve replikasyon çalışmasının özellikleri ve bulguları

	Veri seti	Amaç	Eşik Değerleri			*g-mean*		
			RFC	CBO	WMC	RFC	CBO	WMC
Kaynak çalışma	Eclipse	Proje-içi	40	9	20	0,635	0,588	0,605
Replikasyon çalışması	Bkz. Tablo 3.3	Üniversal	20	8	7	0,635	0,600	0,630

3.8. Geçerliliği Tehdit Eden Faktörler

Bu çalışmayı ve deney sonuçlarını etkilemesi muhtemel tehditler bulunmaktadır. Bu tehditler aşağıda listelenmiştir:

- Yazılım modüllerine ait metrik değerleri, formülleri ve tanımları dikkate alınarak, çeşitli kalite araçları ile toplanabilmektedir. Ancak, bağımlı değişken olan hata verisi, test ekibi veya son kullanıcı tarafından raporlanır ve kaydedilir; bu durum, bu işlemi sübjektif yapar. Test personeli veya son kullanıcı, yazılımın kullanımı sırasında bütün hataları yakalayamayabilir veya hata ortaya çıkmasına rağmen kaydetmeyi unutabilir. Bazı yazılım sistemleri, doğası gereği, tüm özellikleri ile yoğun kullanıma maruz kalabilir; bazıları ise seyrek kullanılabilir. Bu durum, bağımlı ve bağımsız değişkenler arası ilişkinin kurulması önünde ciddi bir tehdit oluşturabilir. Bu sebeple, potansiyel olarak var olabilecek bir desen, veri setine yansımamış olur.
- Diğer bir tehdit ise ilgili bütün hataların raporlanmamasıdır. Özellikle küçük boyutlu projelerde; test personeli tarafından küçük bir hata tespit edilmesi durumunda, bunu kaydetmeksizin direk olarak yazılım geliştirici ile irtibata geçip, sözel bildirim yapabilir. Bu sebeple, güvenilir bir kestirim modeli oluşturulamaz.
- Deneysel sonuçlar şunu göstermiştir: sistemlerin farklı olmasından dolayı, bir sistem için elde edilen eşik değerleri, diğer sistemlere uygulanamaz (Briand ve ark., 2002). İyi bir eşik değer; organizasyonun yapısı, programlama dili, kullanılan araçlar ve yazılım geliştiricilerinin pratik ve tecrübesine göre değişebilmektedir (Herbold ve ark., 2011). Projeye özgü bağlam, model

geliştirme aşamasında dikkate alınması gereken bir durumdur (Dejaeger ve ark., 2013). Bu tehdidi aşmak için; bu çalışmadaki deneylerde, farklı açık kaynak kodlu yazılım sistemlerine ait veri setleri birleştirilerek, tek bir eğitim veri seti oluşturmuş ve sonuçta, belli bir yazılım grubu için makul sonuçların elde edildiği eşik değerlere ulaşılmaya çalışılmıştır. Ancak, bu çalışmadaki deneylerde, çok geniş geliştirme ortamlarından ve farklı programlama dillerinden yazılımlar kullanılmamıştır. Bu nedenle, türetilen eşik değerleri, önceki bölümlerde listelenen yazılım karakteristiklerini içermeyen farklı bir ortama uygulandığında, buradaki ile aynı performansı sergilemeyebilirler.

- Bu çalışmada, *g-mean* limit değeri için önerilen 0.6 değeri, önceki bölümlerde listelenen referanslara dayanılarak belirlenmiştir. Varsayımı yapılan bu limit değeri, eşik değerleri hakkında varılan sonuçların yorumlanmasını etkilemiştir. Diğer araştırmacılar, kendi çalışmalarında farklı limit değeri kullanabilirler; çünkü, araştırmacılar arasında bu konu hakkında ortak bir görüş oluşmamıştır.

3.9. Sonuç

Yazılım kalite metrikleri, yazılımın niteliğini niceliksel olarak yansıtır. Bu metrikler, projenin yönetilmesine, yazılımın geliştirilmesine ve özellikle yazılımın testine destek olurlar. Proje yöneticileri, yazılım müşteriye teslim edilmeden önce riskli taraflarını görebilirler. Yazılım geliştiricileri, gözden geçirilmeye ihtiyaç duyulan modülleri öğrenir ve yapacakları düzenleme ile yazılım kalitesini arttırmış ve dolayısıyla risk seviyesini azaltmış olurlar. Test personeli, daha fazla üzerine yoğunlaşılması gereken modülleri tespit eder ve kalite durumuna göre faaliyetlerini önceliklendirebilirler. Bu çalışmada, her bir metrik için modüllerin risk kategorisinin belirlenmesine yönelik etkili eşik değerleri bulunup bulunmadığının deneysel araştırması yapılmıştır. Öncelikle, üç farklı risk kategorisi tanımlanmıştır: hatasızlığa meyilli, hataya meyilli ve 3'ten fazla hataya meyilli. Bu risk kategorilerinin belirlenmesini destekleyen iki eşik değerinin türetilmesi için iki farklı deney gerçekleştirilmiştir. Bu çalışmanın ana hedeflerinden birisi, farklı yazılım sistemlerine de başarılı bir şekilde uygulanabilecek etkili üniversal eşik değerleri elde etmektir. Bu sebeple, toplamda 37 versiyon içeren 10 açık kaynak kodlu yazılım sistemi, veri seti olarak kullanılmıştır. 10 veri seti

birleştirilerek oluşturulan tek bir eğitim verisi ile öğrenme modeli kurulmuş ve geriye kalan 27 veri seti ise modelin validasyonu amacıyla kullanılmıştır. Sonuçta; bazı metrikler için geçerliliği tespit edilen etkili iki farklı eşik değeri önerilmiştir.

BÖLÜM 4. ÖZELLİK BAĞIMLI NAİVE BAYES YAKLAŞIMI VE YAZILIM HATA KESTİRİMİNDE UYGULANMASI

4.1. Giriş

Yazılım Yaşam Döngüsü (YYD) bir yazılım ürününün çeşitli aşamalarını temsil eder. YYD; planlama, analiz, tasarım, geliştirme ve test safhalarını ya doğrusal olarak (waterfall) ya da döngüsel olarak (incremental, iterative, agile) içermektedir. Test safhası, YYD'nin en önemli safhalarından biridir. Bu safhada yazılımın doğrulama ve validasyon kontrolleri yapılır, ürünün hatasız olması sağlanır ve gerçek çalışma ortamına kurulumuna onay verilmiş olunur. Yazılım geliştirme aşamasında tespit edilen bir hatanın etki maliyeti, müşteri tarafından gerçek ortamda bulunmasından çok daha düşüktür (Pelayo ve Dick, 2007). Müşteri tarafından bulunan bir hata, operasyonel işlemleri durdurabilir, hatta ürün görev-kritik ise daha ölümcül sonuçlara sebep olabilir. Bu durumu gösteren geçmişten iki örnek verilebilir: 1) 125 milyon dolarlık bütçe ile geliştirilen NASA uzay aracı görev uçuşu sırasında uzayda kayboldu. Bu durumun, küçük bir veri dönüştürme hatasından kaynaklandığı ortaya çıktı (Michaels, 2008). 2) Airbus A400M askeri kargo uçağı, test uçuşu sırasında düştü. Sonrasında yapılan araştırmalarda, düşüşün motor kontrol yazılımındaki bir hatadan kaynaklandığı tespit edildi (Flottau ve Osborne, 2015). Yazılım testi, YYD safhaları içinde en fazla zaman ve en fazla maliyet gerektiren safhadır. Bu nedenle, son yıllarda makine öğrenmesi algoritmalarına dayalı teste yardımcı olacak yaklaşımlar araştırmacılar tarafından önerilmiştir. Bu yaklaşımlarla hataya meyilli olan ve olmayan yazılım modülleri otomatik olarak sınıflandırılmaktadır. Böylelikle, yazılım ve test ekibinin kaynaklarını maliyet ve zaman etkin bir şekilde kullanmasına yardımcı olunmaktadır. Bu sınıflandırma yaklaşımları, genel olarak yazılım hata kestirimi olarak adlandırılmaktadır.

Şimdiye kadar yapılan çalışmalar yazılım kalitesi ile hata kestirimi arasında bir ilişki olduğunu göstermiştir (Basili ve ark., 1996; Gyimothy ve ark., 2005; Singh ve ark., 2010; Malhotra ve Bansal, 2015). Yazılım kalitesi, Chidamber & Kemerer (Chidamber ve Kemerer, 1994), Halstead (Halstead, 1977) ve Kod Satır Sayısı gibi önerilmiş metrik kümeleri ile değerlendirilmektedir. Yazılım hata kestiriminde bu metrikler bağımsız girdi; yazılımın hata eğilimli olma durumu (hataya meyilli (hm), hatasızlığa meyilli (hsm)) ise ikili bağımlı çıktı olarak ele alınmaktadır. Yazılım hata kestiriminde en çok kullanılan yöntemlerden biri Naïve Bayes (NB) yöntemidir. NB, özelliklerin bağımsızlığı ve eşit ağırlığa sahip olması varsayımlarına sahiptir.

Bu çalışmada önişleme adımları sonrasında, özelliklerin bağımlılığına dayanan Naïve Bayes sınıflandırması (Feature Dependent Naive Bayes (FDNB)) yöntemi önerilmiştir. Bu yöntem, NASA PROMISE veri seti kullanılarak yazılım hata kestiriminde uygulanmıştır. Elde edilen sonuçlar, bu yöntemin standart Naive Bayes'e göre daha iyi ve dirençli olduğunu göstermiştir. Bazı çalışmalarda yanlış sınıflandırma altyapısı kullanılması sonucunda olması gerekenden daha iyi sonuçlar alınabilmekte ve diğer araştırmacılar yanlış yönlendirilebilmektedir. Bu durumun önüne geçmek için önerdiğimiz sınıflandırma altyapısı kurulurken, Song ve arkadaşları (2011) tarafından önerilen standart çerçeve (framework) temel alınmıştır.

4.2. Kaynak Araştırması

Yazılım hata kestiriminde NB birçok çalışmada uygulanmıştır. Bunların en önemlilerinden birinde, Menzies ve arkadaşları (2007) yaptıkları çalışma sonucunda log-filtre ve özellik seçimi sonrasında kullanılan NB'nin en iyi sonucu verdiğini belirtmişlerdir. Turhan ve Bener (2009) yaptıkları çalışmada, Menzies ve arkadaşlarının (2007) çalışmasını karşılaştırma amaçlı dayanak noktası olarak kullanmışlar ve NB'nin özellik bağımsızlığı ve özelliklerin eşit ağırlığa sahip olması varsayımlarını sorgulamışlardır. Birinci varsayımı kaldırmak için veriyi Temel Bileşenler Analizi (Principal Component Analysis (PCA)) önişleme sonrasında sınıflandırmaya tabi tutmuşlar. İkinci varsayımı kaldırmak için ise özellik ağırlıklandırma şeması kullanılmışlardır. İki yöntem ile elde edilen sonuçların standart

NB'ye göre daha başarılı olduğu görülmüştür. Jiang ve arkadaşları (2016) derin özellik ağırlıklandırma (Deep Feature Weighting (DFW)) adında, NB algoritmasının şartlı olasılık hesaplamalarında kullanılmak üzere yeni bir yöntem önermişler ve bunu metin sınıflandırma amaçlı kullanmışlardır. Standart NB ile yaptıkları performans karşılaştırmasına göre daha iyi sonuçlar elde etmişlerdir. NB algoritmasının iki varsayımını sorgulayan başka çalışmalar da yapılmış ve başarılı sonuçlar elde edilmiştir (Zheng ve Webb, 2000; Zhang ve Sheng, 2004; Hall, 2007; Zhang ve ark., 2016). Yapılan bir çalışmada, eğitim verisinde ağırlıklandırılmış özellik frekansı hesaplanarak NB formülündeki koşullu ihtimal tahmininde kullanılmıştır. Bu yöntem, Deep Feature Weighting (DFW) olarak adlandırılmıştır (Jiang ve ark., 2016). Yapılan bu çalışmada önerilen yöntem ile NB'nin özellik bağımsızlığını sorgulanmış ve özelliklerin birbiriyle birlikte ikili olarak ele alınması durumundaki performansı değerlendirilmiştir. Song ve arkadaşları (2011), Menzies ve arkadaşlarının (2007) çalışmasında $M \times N$ çapraz validasyon kullanılmasına rağmen, özellik seçimi yaklaşımlarının problemli olduğunu ve değerlendirme sonuçlarında bir sapmaya sebep olduğunu belirtmişlerdir. Bütün veri seti üzerinde yapılan bir özellik sıralamanın yanlış olduğunu, çünkü gerçekte elde sadece eğitim verilerinin olduğu ve önişleme adımlarının bu eğitim verileri üzerinde yapılmasının doğru olduğunu vurgulamışlardır. Bu yanlış yaklaşımın potansiyel sonucu; sınıflandırıcının olması gerekenden daha iyi sonuç vermesi ve dolayısıyla çalışmayı temel alan araştırmacıların yanıltılmasına sebebiyet vermesidir. Önerilen yöntem ile elde edilen sonuçlar, Song ve arkadaşları (2011) tarafından uygulanan log-filtreleme ve özellik seçim önişlemlerinden geçirilmiş NB sınıflandırma sonuçları ile karşılaştırılmıştır.

4.3. Veri Setleri ve Metrikler

Bu çalışmada, araştırmacılar tarafından yaygın olarak kullanılan NASA MDP (Metrics Data Program) havuzundan sekiz veri seti kullanılmıştır (Chapman ve ark., 2004). Bu veri setleri NASA tarafından geliştirilen uzay uçuş kontrol, yer kontrolü için veri depolama yönetimi ve uzay aracı enstrümantasyon yazılım projelerinden türetilmiştir. Her bir veri seti yazılım modüllerinden oluşmaktadır. Her bir yazılım modülü ise, sınıflandırıcıda girdi olarak kullanılacak olan çeşitli statik yazılım kod metrikleri ve

çıktı olarak ise, ilgili modülde bulunan hata sayısını içermektedir. Hata sayısı ikili yapıya dönüştürülmüştür. Yani, birden fazla hata varsa, hatalı modül; hiç hata yoksa hatasız modül olarak etiketlenmiştir. Tablo 4.1., kullanılan sekiz veri setine ait genel özellikleri içermektedir. Tablonun en son sütununda, hatalı modüllerin oranının %0.41'den %48.80'e değişkenlik gösterdiği görülmektedir. Bu sekiz veri setinin ortalama hata oranı ise %13.14'dür. Veri setlerindeki hata oranları incelendiğinde yazılım hata verilerinin dengeli dağılmadığı (unbalanced) görülmüştür. Sınıflandırıcı altyapısı kurulurken bu durum dikkate alınmalıdır.

Tablo 4.1. Çalışmada kullanılan veri setlerinin genel özellikleri

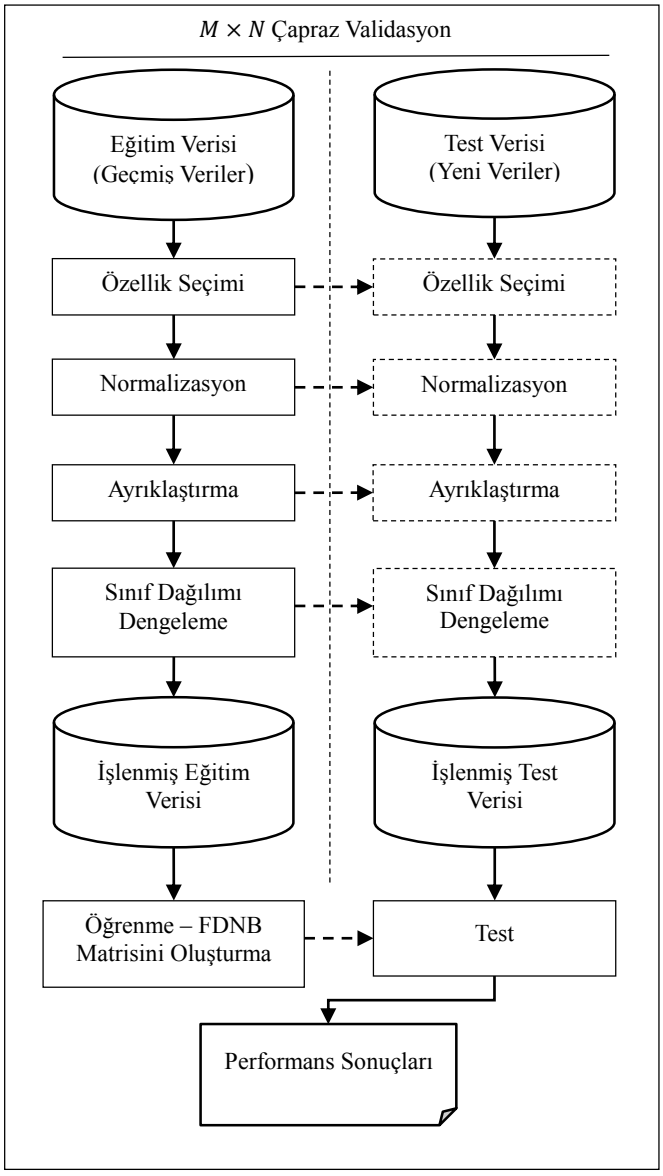
Adı	Özellik (#)	Modül (#)	Hatalı (#)	Hatalı (%)
CM1	38	505	48	9,5
PC1	38	1107	76	6,87
PC2	38	5589	23	0,41
PC3	38	1563	160	10,24
PC4	38	1458	178	12,21
KC3	38	458	43	9,39
KC4	38	125	61	48,80
MW1	38	403	31	7,69

Kullanılan bu veri setleri, 38 adet McCabe, Halstead, Kod Satır Sayısı (LoC) ve diğer muhtelif türetilmiş kalite metriklerinden oluşur. Bu metrikler kaynak kodun boyutu ve karmaşıklığı hakkında fikir verirler. Her bir metrik kümesi, farklı bir yönden kodun kalitesini gösterir. LOC, basit olarak kaynak kodun satır sayısını baz alarak boyutunu belirtir. McCabe ve Halstead ise sırasıyla kodun dallanmasını baz alarak karmaşıklığını ve okunabilirliğini değerlendirir. KC3 ve KC4 haricindeki yazılımların geliştirilmesinde C dili kullanılmıştır. KC3, Java ve KC4 ise Perl dili ile kodlanmıştır. Bu metriklerin detaylı açıklamalarına orijinal MDP veri seti kaynağından (Chapman ve ark., 2004) ve Menzies ve arkadaşlarının (2007) çalışmasından ulaşılabilir.

4.4. Öğrenme Modeli Çerçevesi

Öğrenme modeli altyapısı oluşturulurken, daha önce de bahsedildiği gibi Song ve ark. tarafından uygulanan çerçeve (framework) (Song ve ark., 2011) örnek alınmıştır. Bu

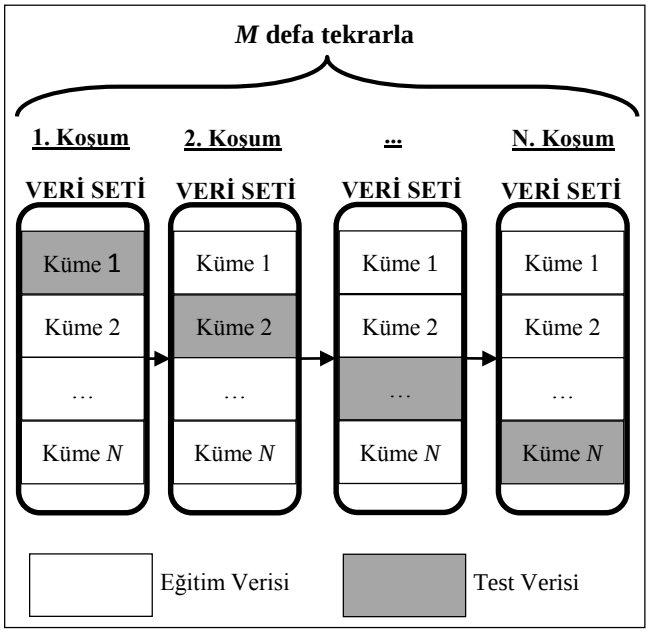
çerçevenin esaslarından en önemlisi, öğrenme modelinin test verisinden bağımsız olarak, tamamen eğitim verisinden oluşturulmasıdır. Sınıflandırıcı performansının yeni veriler ile değerlendirilmesi için bu durum gerekli bir ön şarttır. Bu çalışma kapsamında oluşturulan öğrenme modelinin gösterimi Şekil 4.1.'de verilmiştir.



Şekil 4.1. Öğrenme modeli altyapısı

Eğitim seti üzerinde veri önişleme adımları uygulanmış ve bu uygulama sonucunda elde edilen parametrelere göre test seti de aynı önişlemlerden geçirilmiştir. Ön işlenmiş eğitim verisi baz alınarak sınıflandırıcı formülünde kullanılacak değişkenler belirlenmiştir. Test verisindeki her bir örneğin sınıf tahmini, sınıflandırıcı formülüne

göre yapılmakta ve sonuçta bir performans raporu ortaya çıkmaktadır. Bu adımlar, *M*x*N* çapraz validasyon yöntemine göre tekrarlanmaktadır. Bu yönteme göre tüm veri seti *N* alt kümeye bölünür ve bir küme test verisi, geriye kalan (*N*-1) alt küme ise eğitim verisi olarak kullanılır. Bu işlem her bir alt kümenin test verisi olarak kullanılmasını sağlamak için *N* defa tekrarlanır. Böylelikle, örnek seçiminden kaynaklanabilecek yanılgılı sonuç (sampling bias) minimize edilmiş olur. Diğer taraftan, verideki örneklerin sıralamasının etkisini azaltmak ve istatistiki olarak güvenilir sonuçlar elde etmek için bu işlem de *M* defa tekrarlanır ve her seferinde veri seti sıralaması rasgele olarak karıştırılır. Bu *M*x*N* çapraz validasyon yöntemi Şekil 4.2.'de gösterilmiştir. Bu çalışmada yapılan deneylerde 20x10 çapraz validasyon kullanılmıştır (*M*=20 ve *N*=10). Deney performansı olarak öğrenme modelinin 200 koşumdan elde edilen sonuçları değerlendirilmiş ve diğer çalışmalarla karşılaştırılmıştır.



Şekil 4.2. *M*×*N* çapraz validasyon yöntemi

Öğrenme modeli altyapısı, Java dili ile kodlanmıştır. Çapraz validasyon, özellik seçimi ve veri önişleme (normalizasyon, ayrıklaştırma, sınıf dağılımı dengeleme) adımlarının uygulanmasında WEKA[5] yazılımından (Hall ve ark., 2009) yararlanılmıştır. Bu çalışma kapsamında önerilen yöntemin geliştirilmesi veya araştırmacıların kendi

[5] http://www.cs.waikato.ac.nz/~ml/weka

modellerini oluşturabilmeleri için kaynak kodları açık hale getirilmiştir. Bu kaynak kodlarının içerisinde kullanılan veri setleri de bulunmaktadır, yani indirilip derlenmeye ve çalıştırılmaya hazır Eclipse projesi şeklinde internete yüklenmiştir[6].

Bu bölümün diğer kısımlarında öğrenme modelinde yer alan özellik seçimi, normalizasyon, ayrıklaştırma, sınıf dağılımı dengeleme ve modelin performansını değerlendirmek üzere kullanılan ölçümler ele alınacaktır.

4.4.1. Özellik seçimi

Veri setleri, sadece yazılım hata kestiriminde kullanılması amacıyla oluşturulmamışlardır. Her bir metrik farklı görevler için yararlı olsa da hata kestirim modeline katkısı olmayabilir. Bu nedenle eğitim verisi üzerinde özellik seçim önişlemi uygulanmalıdır. Bu işlem sonucunda modelin başarı performansına katkısı olmayan özellikler elenmektedir. Özellik seçim yöntemleri, filtre (filter) ve sarmalama (wrapper) olarak iki kategoride toplanır (Chandrashekar ve Sahin, 2014). Filtre yöntemleri, kullanılan sınıflandırma algoritmasından bağımsız olarak eğitim verisinin genel karakteristiklerine göre özellikleri değerlendirir. Sarmalama yöntemlerinde ise sınıflandırıcı ile beraber kullanılarak en iyi alt özellik kümesinin seçimi yapılır.

Bu çalışmada ise yaygın olarak kullanılan bir filtre yöntemi olan Korelasyon-tabanlı Özellik Seçimi (Correlation-based Feature Selection (CFS)) kullanılmıştır (Hall ve Smith, 1999). Bu yöntemde özellik alt kümelerin yararlılığı sezgisel bir fonksiyona göre yapılır. Bu sezgisel değerlendirme fonksiyonu, seçilen alt kümenin sınıflandırma tahmin (kestirim) gücüne ve özelliklerin birbirleriyle olan korelasyon (intercorrelation) seviyesine bakar. Bu fonksiyon denklem (4.1)'deki gibi tanımlanır (Hall ve Holmes, 2003).

$$Merit_s = \frac{n\overline{r_{cf}}}{\sqrt{k+(k-1)\overline{r_{ff}}}} \tag{4.1}$$

[6] https://github.com/ofarar/fdnb-classifier

burada;

- $Merit_s$, n özellik içerisindeki s alt özellik kümesinin yararlılığını gösteren sezgiseldir.
- Ortalama sınıf-özellik korelasyonu $\overline{r_{cf}}$ ile gösterilir, ($f \in s$).
- Ortalama özellik-özellik interkorelasyonu ise $\overline{r_{ff}}$ ile gösterilir.

$Merit_s$'in uygulanabilmesi için korelasyon matrisi hesaplanmalı ve iyi bir özellik altkümesi bulunması için bu matris üzerinde sezgisel arama algoritması çalıştırılmalıdır. Bu çalışmada, en iyi alt kümeyi bulmak için Açgözlü İleriye Doğru Arama (Greedy Stepwise Forward) kullanılmıştır.

Özellik seçimi ile ilgili kullanılan WEKA paketi ve parametreleri:

- weka.filters.supervised.attribute.AttributeSelection
- evaluator: CfsSubsetEval
- search: GreedyStepwise
- search direction: forward

4.4.2. Normalizasyon

Yazılım metrik değerlerinin aralığı çok farklılıklar gösterebilmektedir. Standart sapma, minimum ve maksimum değerler her metrik için çok farklı olabilmektedir. Bu durum sınıflandırma performansını genellikle olumsuz etkilemektedir. NB'nin log-filtre'den geçirilmiş verilerle daha etkili sonuçlar ürettiği bazı çalışmalarda belirtilmiştir (Menzies ve ark., 2007; Turhan ve Bener, 2009).

Bu çalışmada oluşturulan öğrenme modelinde, veriye min-max normalizasyon işleminin uygulanması ile daha iyi sonuçlar elde edildiği görülmüştür. Bu yöntem ile veriler belli bir sabit aralığa çekilir. Bu durumda özellik içerisindeki ve özellikler arası standart sapma azaltılmış olur; dolayısıyla, uçdeğerlerin (outlier) sınıflandırıcı üzerindeki negatif etkisi de giderilmiş olur. Veri setindeki her bir v özellik değeri, denklem (4.2)'deki formül ile v' değerine dönüştürülür:

$$v' = \frac{v - min_f}{max_f - min_f}\left(yeni\_max_f - yeni\_min_f\right) + yeni\_min_f \qquad (4.2)$$

burada;

- f özelliğine ait veri setindeki minimum değer min_f
- f özelliğine ait veri setindeki maksimum değer max_f
- Dönüştürülmek istenen aralığın minimum değeri $yeni\_min_f$
- Dönüştürülmek istenen aralığın maksimum değeri $yeni\_max_f$

olarak ifade edilir.

Yapılan deneylerde en iyi sonucun [0-10] dönüşümü ile elde edildiği görülmüştür.

Normalizasyon ile ilgili kullanılan WEKA paketi ve parametreleri aşağıda verilmiştir:

- weka.filters.unsupervised.attribute.Normalize
- translation: 0
- scale: 10

4.4.3. Ayrıklaştırma

Yazılım metrik özellikleri, nümerik değerlere sahip olduğundan, Fayyad ve Irani'nin önerdiği ayrıklaştırma şeması yöntemi (Fayyad ve Irani, 1992) ile kategorik değerlere dönüştürülür. Ayrıklaştırma yönteminin yazılım hata kestirim performansına olumlu katkısı olduğunu gösteren çeşitli çalışmalar yapılmıştır (Singh ve Verma, 2009; Hewett, 2011). Normal bir ayrıklaştırma işlemi dört adımdan oluşur (Liu ve Hussain, 2002):

- Ayrıklaştırılacak özelliğe ait bütün sürekli (continuous) değerler sıralanır.
- Veriyi ayırmak için bir kesme noktası (cut point) seçilir.
- Sürekli değerleri içeren aralıklar bölünür veya birleştirilir.
- Ayrıklaştırma işlemi için bir durma kriteri seçilir.

Bu çalışmada ayrıklaştırma yöntemi olarak, entropi tabanlı bölme ile beraber durma kriteri olarak minimum betimleme uzunluğu (minimum description length) kullanılmıştır.

Ayrıklaştırma ile ilgili kullanılan WEKA paketi ve parametreleri aşağıda verilmiştir:

- weka.filters.supervised.attribute.Discretize
- Tüm parametreler: varsayılan

Verilerin, ayrıklaştırma önişlemi sonrasında NB sınıflandırıcıda kullanımı, ilgili bölümde anlatılmaktadır.

4.4.4. Sınıf dağılımı dengeleme

Yazılım hata veri setleri dengesiz (unbalanced) dağılım gösterirler. Hatalı modül sayısı (minority class), hatasız modül (majority) sayısından azdır. Bu durum veri önişleme ile giderilmediği veya sınıflandırıcı altyapısı kurulurken dikkate alınmadığı takdirde başarısız sonuçlar alınabilmektedir. Veri önişleme aşamasında bu dağılım dengesizliğini gidermek üzere eğitim verisinde, azınlık sınıfına ait örnek arttırımı (oversampling), çoğunluk sınıfına ait örnek azaltımı (undersampling) ve SMOTE gibi çeşitli tekrar örnekleme (resampling) yöntemleri uygulanmaktadır (Wang ve Yao, 2013).

Oluşturulan öğrenme modelinde, WEKA'nın ilgili paketi kullanılmıştır. Bu paket ile hatalı modül örnekleri çoğaltılarak ve hatasız modül örnekleri azaltılarak sınıf dağılımı dengelenmiş olur.

Tekrar örnekleme ile ilgili kullanılan WEKA paketi ve parametreleri aşağıda verilmiştir:

- weka.filters.supervised.instance.Resample
- biasToUniformClass: 1.0
- sampleSizePercent: 100
- noReplacement: false

4.4.5. Performans ölçütleri

Bu çalışmada elde edilen sonuçların, NB metodunu kullanan diğer çalışmalarla (Menzies ve ark., 2007) karşılaştırabilmek için aynı performans ölçütleri

kullanılmıştır. Hata matrisi Tablo 4.2.'de gösterilmiştir. Bu matriste test veri setindeki örneklerin gerçek sınıfı ve sınıflandırıcı tarafından yapılan kestirim (tahmin) sınıfının gösterimi yapılmıştır.

Tablodaki kısaltmalar şu anlamlara gelmektedir:

- TP, gerçekte hatalı olan modüllerden kaç tanesinin doğru tahmin edildiği
- FP, gerçekte hatasız olan modüllerden kaç tanesinin yanlış olarak hatalı tahmin edildiği
- TN, gerçekte hatasız olan modüllerden kaç tanesinin doğru tahmin edildiği
- FN, gerçekte hatalı olan modüllerden kaç tanesinin yanlış olarak hatasız tahmin edildiği

Tablo 4.2. Hata matrisi

		Gerçek Sınıf	
		Hatalı	**Hatasız**
Tahmin Edilen Sınıf	**Hatalı**	TP	FP
	Hatasız	FN	TN

Hata matrisinden türetilen hatalı modül tespit oranı (pd, probability of detection) ve yanlış alarm oranı (pf, probability of false alarm) formülleri denklem (4.3) ve (4.4)'de gösterilmiştir.

$$pd = \frac{\text{TP}}{\text{TP+FN}} \tag{4.3}$$

$$pf = \frac{\text{FP}}{\text{FP+TN}} \tag{4.4}$$

Araştırmacıların öğrenme modeli sonuçlarını tek ölçüt ile karşılaştırabilmeleri için pd ve pf'den balans (*bal*) ölçütü üretilmiştir (Denklem (4.5)). Bu değer ideal (pd, pf) noktası olan (1,0)'a olan normalize edilmiş öklid uzaklığıdır:

$$bal = 1 - \frac{\sqrt{(1-pd)^2+(0-pf)^2}}{\sqrt{2}} \quad (4.5)$$

4.5. Önerilen Öğrenme Modeli

Bu bölümde çalışma kapsamında önerilen Özellik Bağımlı Naive Bayes (Feature Dependent Naive Bayes (FDNB)) yöntemi anlatılacaktır. FDNB yöntemi, ordinal (sıralı) değerlere sahip özelliklerin nominal (kategorik) değere dönüştürülmüş olduğunu varsaymaktadır. Bu dönüşüm için ayrıklaştırma (discretization) ön işlemi yapılmaktadır. Burada FDNB'yi daha etkin anlatabilmek adına öncelikle standart NB ve veri ayrıklaştırma işlemlerinden bahsedilecektir.

4.5.1. Naive bayes

En etkili sınıflandırıcı yöntemlerinden bir tanesi NB yöntemidir. Bu yöntem, diğer makine öğrenmesi algoritmaları ile kıyaslandığında anlaşılması kolay ve bunun yanında şaşırtıcı derecede iyi sonuçlar üreten bir sınıflandırıcıdır.

Bayes teoremine göre bir örneğin sonraki (posterior) dağılımı, önceki (prior) dağılımı ve olabilirliği (likelihood) ile orantılıdır. NB formülü denklem (4.6) ile verilmiştir.

$$P(C_k|X) = \frac{P(C_k)P(X|C_k)}{P(X)} \quad (4.6)$$

burada;

- C_k belli bir sınıfı ifade etmektedir. k=0,1 (hataya meyilli (hm), hatasızlığa meyilli (hsm))
- $X = (x_1, x_2, \dots, x_n)$ ise bir yazılım modülüne ait metrik değerlerini ifade etmektedir.

Sınıflandırıcı; test veri seti içerisinde X metrik değerlerine sahip bir yazılım modülünü, eğitim veri setindeki dağılımları dikkate alarak hm veya hsm şeklinde etiketler. Olasılık değeri daha yüksek çıkan sınıfa göre etiketleme yapılır: $argmax_k(P(C_k|X))$.

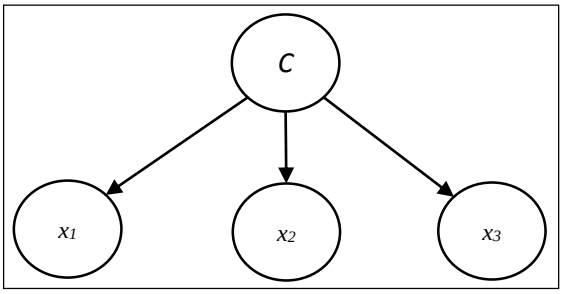
Formülde geçen payda bütün k sınıfları için aynı olacağından en son etiketleme kararını etkilemeyecektir, dolayısıyla formülden çıkartılabilir ve sonunda denklem (4.7)'deki ifadeye dönüştürülür:

$$P(C_k|X) = P(C_k)P(X|C_k) \tag{4.7}$$

$P(C_k)$ ve $P(X|C_k)$ olasılıkları, eğitim veri setinden türetilmelidir. Ancak, eğitim veri setinde $X = (x_1, x_2, \dots, x_n)$ metrik değerlerine sahip bir yazılım modülü büyük olasılıkla olmaz. Bu problemden dolayı NB'de özelliklerin bağımsızlığı varsayımı yapılmıştır. Yani, her bir metrik değerinin olasılığı ayrı hesaplanmaktadır. Bu durumda NB formülü denklem (4.8)'deki gibi olur.

$$P(C_k|X) = P(C_k) \prod_{i=1}^{n} P(x_i|C_k) \tag{4.8}$$

Üç özelliğe sahip bir örnek için NB gösterimi Şekil 4.3.'deki gibi olur.



Şekil 4.3. Naive Bayes'de özellikler ile sınıf arasındaki ilişki

Nominal (kategorik) değerlere sahip özellikler için olasılık değeri, basitçe ilgili kategorilerin toplanması ile elde edilir. Ordinal (sıralı) özelliklerin olasılık değeri ise genel varsayım olarak normal dağılıma sahip olduğu kabul edilerek hesaplanır (Witten ve ark., 2011). Ancak, pratikte veriler bu dağılımı göstermeyebilir. Bu durum sınıflandırma performansında kayıplara sebep olabilmektedir (Taheri ve ark., 2014). Veri ayrıklaştırma önişlemi ile dağılıma bakılmaksızın veriler anlamlı aralıklara bölünürler.

4.5.2. Ayrıklaştırma

Birçok canlı sistem uygulamasına ait veri setleri numerik değerler içerirler. Yazılım hata kestirimi için kullanılan metrikler de aynı şekilde numerik değere sahiptirler. Bu veri setlerinde NB performansını arttırmak için en bilinen yöntem, bu değerlerin aralıklar şeklinde ayrıklaştırılmasıdır. NB sınıflandırıcı için birçok ayrıklaştırma yöntemi uygulanmıştır (Lu ve ark., 2006; Kayah, 2008; Wu ve ark., 2008; Yang ve Webb, 2009). Özellikler, entropi tabanlı yöntem ile ayrıklaştırıldığında, NB performansı önemli derecede artmaktadır (Dougherty ve ark., 1995). Bu nedenle bu çalışmada Fayyad ve Irani (Fayyad ve Irani, 1992) tarafından önerilen minimal entropi sezgiseline (minimal entropy heuristic) dayalı ayrıklaştırma yöntemi kullanılmıştır. Bu yöntemin ayrıntılı açıklamalarına çeşitli kaynaklardan ulaşılabilir (Dougherty ve ark., 1995; Kayah, 2008). Ayrıklaştırma işlemi sonunda $(x_1, x_2, \dots, x_n)$ numerik özellik değerleri, $(x_1^*, x_2^*, \dots, x_n^*)$ nominal aralık değerlerine dönüştürülmüş olur. Her bir x_i^* değeri, $(a_j, b_j]$ aralığını temsil etmektedir. Orijinal numerik değer olan $x_i \in (a_j, b_j]$, x_i^* aralık değeri ile değiştirilmiş olur. Kullandığımız ayrıklaştırma yöntemine göre aralık sayısı (interval number) her bir özellik için farklı olabilmektedir. NB formülünde geçen olasılık hesaplamaları, metrik değerleri bu aralıklara denk gelen yazılım modül sayısı dikkate alınarak yapılır. Bir örnek üzerinde anlatmak gerekirse; aşağıda dört eğitim verisi ve en alt satırda bir test verisi bulunmaktadır. Bu verilerde üç özellik (LOC_CODE_AND_COMMENT, HALSTEAD_CONTENT, MAINTENANCE_SEVERITY) ve sınıf değeri bulunmaktadır:

0.71, 0.27, 1.53, hsm

0.24, 0.08, 4.90, hsm

7.38, 1.21, 0.10, hm

0.24, 0.16, 3.16, hsm

1.19, 0.24, 4.55, ?

Eğitim veri seti, Fayyad ve Irani'nin yöntemine göre ayrıklaştırma işleminden geçirilince her bir özellik için Tablo 4.3.'deki aralıklar üretilmiş olur. Bu tabloda

gösterilen değerler PC-3 veri seti bütün olarak kullanılarak elde edilmiştir. Tabloda görüldüğü üzere LOC_CODE_AND_COMMENT (x_1) metriği üç aralığa bölünmüştür: 0.12'ye kadar olan değerler 1. aralığı (A1); 0.12 ile 0.83 arasındakiler 2. aralığı (A2) ve 0.83'ün üzerindeki değerler ise 3. aralığı (A3) temsil etmektedirler. Dikkat edilirse aralık sayısı farklı olabilmektedir. Örneğin, x_1 ve x_2 özellikleri üç aralığa bölünmüşken, x_3 özelliği ise iki aralığa bölünmüştür.

Tablo 4.3. PC-3 veri seti için aralık değerleri

Özellik	A1	A2	A3
x_1	(-inf-0,12]	(0,12-0,83]	(0,83-inf)
x_2	(-inf-0,11]	(0,11-0,22]	(0,22-inf)
x_3	(-inf-3,11]	(3,11-inf)	-

Özelliklere ait bu aralık değerleri dikkate alınarak, ordinal değerler nominal değerlere dönüştürülecektir. Yukarıda verilen örnek eğitim ve test verisi ayrıklaştırma işlemi sonrasında Tablo 4.4.'deki gibi aralık değerlerine sahip olacaktır.

Tablo 4.4. Ayrıklaştırma işlemi sonrası örnek veri seti

Veri No	x_1	x_2	x_3	C
1	A2	A3	A1	hsm
2	A3	A1	A2	hsm
3	A3	A3	A1	hm
4	A2	A2	A2	hsm
5 (test)	A3	A3	A2	?

NB formülünde yer alan olasılık hesaplamaları bu nominal değerler dikkate alınarak yapılacaktır. Verilen sette her bir aralığa denk gelen hm ve hsm örnek sayıları sırasıyla Tablo 4.5. ve Tablo 4.6.'da gösterilmiştir.

Tablo 4.5. Örnek veri setinde her bir aralıktaki hm modül sayısı

Özellik	A1	A2	A3
x_1	0	0	1
x_2	0	0	1
x_3	1	0	-

Tablo 4.6. Örnek veri setinde her bir aralıktaki hsm modül sayısı

Özellik	A1	A2	A3
x_1	0	2	1
x_2	1	1	1
x_3	1	2	-

Bu tablolarda bazı özelliklerin bazı aralıklarda 0 değerine sahip olduğu (yani o aralıkta hiç örnek olmadığı) görülür. NB formülündeki çarpımlarda bu sıfır olasılık değerlerinin önüne geçmek için Laplace düzeltme (Laplace smoothing) yapılır. En bilinen Laplace düzeltme, bir ekle (add one) yöntemi ile yapılır (Manning ve Schütze, 1999). Yani, bütün özelliklerin bütün aralık değerleri hm ve hsm örnekleri için 1 arttırılır. Bu durumda Tablo 4.5. ve Tablo 4.6.'daki değerleri bir arttırarak NB formülünde kullanmak gerekir.

NB formülündeki $P\left(x_i^{Aj}\middle|C_{hm}\right)$ hesaplamasında, payda kısmında küçük bir değişiklik yapılarak formül denklem (4.9)'daki hale dönüştürülür.

$$P\left(x_i^{Aj}\middle|C_{hm}\right) = \frac{hmN(x_i^{Aj})}{hmN(x_i^{Aj})+hsmN(x_i^{Aj})} \tag{4.9}$$

burada; i no'lu özelliğin j no'lu aralığının (Aj), hm sınıfına ait olma olasılığı hesaplaması verilmektedir. $hmN(x_i^{Aj})$, eğitim verisinde i özelliğinin j aralığında bulunan hm sınıfındaki örnek sayısını temsil etmektedir. $hsmN(x_i^{Aj})$ ise eğitim verisinde i özelliğinin j aralığında bulunan hsm sınıfındaki örnek sayısını temsil etmektedir. $P\left(x_i^{Aj}\middle|C_{hsm}\right)$ hesaplaması da aynı şekilde denklem (4.10)'daki gibi olur.

$$P\left(x_i^{Aj}\middle|C_{hsm}\right) = \frac{hsmN(x_i^{Aj})}{hmN(x_i^{Aj})+hsmN(x_i^{Aj})} \tag{4.10}$$

Tüm bunlardan sonra, test verisinin sınıfının belirlenmesi için NB hesaplaması sonucunda hm olma olasılığı aşağıdaki gibi olur (Laplace düzeltme ile tablodaki her değer 1 arttırılmış hali ile formülde kullanılmıştır):

$$P(hm|x_1^{A3}, x_2^{A3}, x_3^{A2}) = P(hm) \times P(x_1^{A3}|hm) \times P(x_2^{A3}|hm) \times P(x_3^{A2}|hm)$$

$$P(hm|x_1^{A3}, x_2^{A3}, x_3^{A2}) = \frac{11}{28} \times \left(\frac{2}{4} \times \frac{2}{4} \times \frac{1}{4}\right) = 0.025$$

Test verisinin hsm olma olasılığı ise aşağıdaki gibi olur:

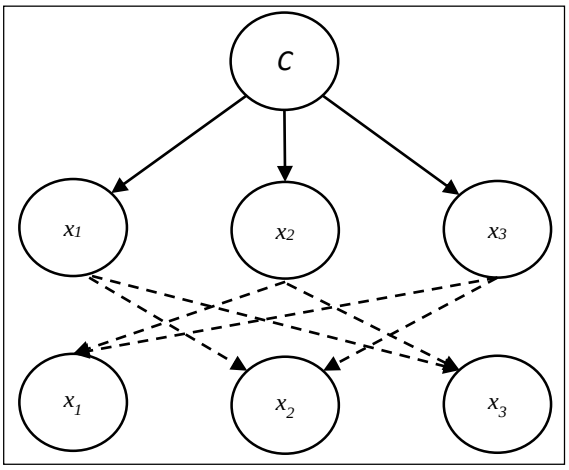
$$P(hsm|x_1^{A3}, x_2^{A3}, x_3^{A1}) = P(hsm) \times P(x_1^{A3}|hsm) \times P(x_2^{A3}|hsm) \times P(x_3^{A1}|hsm)$$

$$P(hsm|x_1^{A3}, x_2^{A3}, x_3^{A1}) = \frac{17}{28} \times \left(\frac{2}{4} \times \frac{2}{4} \times \frac{3}{4}\right) = 0.114$$

$P(C_{hsm}|X) > P(C_{hm}|X)$ olduğundan, NB sınıflandırıcısı, test verisini hatasızlığa meyilli (hsm) bir modül olarak tahmin etmiş olur.

4.5.3. Özellik bağımlı naive bayes

Özelliklerin bağımsız ele alınması varsayımı, ilişkili özelliklere sahip olan durumlarda NB'nin performansına negatif olarak yansımaktadır (Jiang ve ark., 2007). Bu varsayımı gevşetmek için araştırmacılar tarafından eşitli yöntemler önerilmiştir. Bunlardan, Tree Augmented Naive Bayes (TANB) yöntemi ağaç yapısını kullanarak özellikler arasındaki ilişkiyi belirlemeye çalışır (Friedman ve ark., 1997). Improved Naive Bayes (INB) yöntemi ise özellikler arasındaki ilişkiyi belirlemede koşullu olasılıkları kullanır (Taheri ve ark., 2011). Bu çalışmada önerilen Özellik Bağımlı Naive Bayes (Feature Depedent Naive Bayes (FDNB)) yöntemi ile özellikle ikili olarak hesaplamaya katıldı. Üç özelliğe sahip bir örnek için FDNB gösterimi Şekil 4.4.'deki gibi olur. Burada her bir özellik kendisi haricindeki diğer bütün özellikler ile beraber ele alınır.



Şekil 4.4. FDNB'de özellikler ile sınıf arasındaki ilişki

Buna göre FDNB sınıflandırıcı formülü denklem (4.11)'deki gibi verilir:

$$argmax_k P(C_k|X) = P(C_k) \prod_{i=1}^{n} \prod_{s=1,s\neq i}^{n} P(x_i, x_s|C_k) \quad (4.11)$$

burada; eğitim verisinde her bir özelliğin diğerleri ile birlikte bulunma olasılığı hesaba katılır. Özellikler ve aralıklar ikili olarak düşünüldüğünde dört boyutlu bir matris karşımıza çıkmaktadır. Bu dört boyut şunlardır:

1. Boyut: Özellikler (birincil olarak kullanılacak), *i*
2. Boyut: Özelliklere ait tüm aralık değerleri
3. Boyut: Özellikler (ikincil olarak kullanılacak), *s*
4. Boyut: Özelliklere ait tüm aralık değerleri

Bir önceki bölümde verilen ayrıklaştırılmış örnek veriler (Tablo 4.3.) üzerinden FDNB sınıflandırıcı hesaplaması örneklendirilebilir. Dört boyutlu matris gösteriminin karmaşıklığından dolayı, veriler iki boyutlu matrise dönüştürülmüştür. Tablo 4.3.'de verilen sette her bir aralığa denk gelen hm ve hsm örnek sayıları sırasıyla Tablo 4.7. ve Tablo 4.8.'de gösterilmiştir. Her bir veri örneğinden özellik sayısının faktöryeli ($n!$) kadar ikili ilişki ortaya çıkar. İlk örnekteki (Tablo 4.4.) özelliklerin ikili ilişkileri denklem (4.12)'deki gibi olur:

$$(x_1^{A2}, x_2^{A3}, x_3^{A1}) = (x_1^{A2},x_2^{A3}),(x_1^{A2},x_3^{A1}),(x_2^{A3},x_1^{A2}),(x_2^{A3}, x_3^{A1}),(x_3^{A1}, x_1^{A2}), (x_3^{A1}, x_2^{A3}) \quad (4.12)$$

Tablo 4.7.'de koyu renkle işaretli olan hücreyi şu şekilde okumak gerekir: ikinci özelliğin birinci aralığı (x_2^{A1}) ve üçüncü özelliğin ikinci aralığının (x_3^{A2}) birlikte bulunduğu ve sınıf değeri hm olan eğitim verisi sayısı. Tablo 4.8.'de koyu renkle işaretli olan hücreyi ise şu şekilde okumak gerekir: ikinci özelliğin birinci aralığı (x_2^{A1}) ve üçüncü özelliğin ikinci aralığının (x_3^{A2}) birlikte bulunduğu ve sınıf değeri hsm olan eğitim verisi sayısı.

Tablo 4.7. Örnek veri setinde her bir aralıktaki hm modül sayısı

		x_1			x_2			x_3	
Özellik	Aralık	A1	A2	A3	A1	A2	A3	A1	A2
x_1	A1	-	-	-	0	0	0	0	0
	A2	-	-	-	0	0	0	0	0
	A3	-	-	-	0	0	1	1	0
x_2	A1	0	0	0	-	-	-	0	0
	A2	0	0	0	-	-	-	0	0
	A3	0	0	1	-	-	-	1	0
x_3	A1	0	0	1	0	0	1	-	-
	A2	0	0	0	0	0	0	-	-

Tablo 4.8. Örnek veri setinde her bir aralıktaki hsm modül sayısı

		x_1			x_2			x_3	
Özellik	Aralık	A1	A2	A3	A1	A2	A3	A1	A2
x_1	A1	-	-	-	0	0	0	0	0
	A2	-	-	-	1	1	0	1	1
	A3	-	-	-	1	0	0	0	1
x_2	A1	0	0	1	-	-	-	0	1
	A2	0	1	0	-	-	-	0	1
	A3	0	1	0	-	-	-	1	0
x_3	A1	0	1	0	0	0	1	-	-
	A2	0	1	1	1	1	0	-	-

FDNB formülündeki ikili olasılık değerleri, hm sınıf tahmini için denklem (4.13)'de verildiği şekilde hesaplanır:

$$P\left(x_i^{Aj}, x_s^{Ar} \middle| C_{hm}\right) = \frac{hmN(x_i^{Aj}, x_s^{Ar})}{hmN(x_i^{Aj}, x_s^{Ar}) + hsmN(x_i^{Aj}, x_s^{Ar})} \quad (4.13)$$

burada; *i* no'lu özelliğin *j* no'lu aralığının, *s* no'lu özelliğin *r* no'lu aralığı ile birlikte bulunma durumlarındaki hm sınıfına ait olma olasılığı verilmiştir. $hmN(x_i^{Aj}, x_s^{Ar})$, eğitim verisinde *i* özelliğinin *j* aralığında ve *s* özelliğinin *r* aralığında bulunan hm sınıfındaki örnek sayısını temsil etmektedir. $hsmN(x_i^{Aj}, x_s^{Ar})$ ise eğitim verisinde *i* özelliğinin *j* aralığında ve *s* özelliğinin *r* aralığında bulunan hsm sınıfındaki örnek sayısını temsil etmektedir. $P\left(x_i^{Aj}, x_s^{Ar} \middle| C_{hsm}\right)$ hesaplaması da aynı şekilde denklem (4.14)'deki gibi olur:

$$P\left(x_i^{Aj}, x_s^{Ar} \middle| C_{hsm}\right) = \frac{hsmN(x_i^{Aj}, x_s^{Ar})}{hmN(x_i^{Aj}, x_s^{Ar}) + hsmN(x_i^{Aj}, x_s^{Ar})} \quad (4.14)$$

Tüm bunlardan sonra, test verisinin sınıfının belirlenmesi için NB hesaplaması sonucunda hm olma olasılığı denklem (4.15)'deki gibi olur (Laplace düzeltme ile tablodaki her değer 1 arttırılmış hali ile formülde kullanılmıştır):

$$P(hm|x_1^{A3}, x_2^{A3}, x_3^{A2}) = P(hm) \times P(x_1^{A3}, x_2^{A3}|hm) \times P(x_1^{A3}, x_3^{A2}|hm) \times P(x_2^{A3}, x_1^{A3}|hm) \times P(x_2^{A3}, x_3^{A2}|hm) \times P(x_3^{A2}, x_1^{A3}|dp) \times P(x_3^{A2}, x_2^{A3}|hm) \quad (4.15)$$

$$P(hm|x_1^{A3}, x_2^{A3}, x_3^{A2}) = \frac{11}{28} \times \left(\frac{2}{3} \times \frac{1}{3} \times \frac{2}{3} \times \frac{1}{2} \times \frac{1}{3} \times \frac{1}{2}\right) = 0.005$$

Test verisinin hsm olma olasılığı ise aşağıdaki gibi olur (Denklem (4.16)):

$$P(hsm|x_1^{A3}, x_2^{A3}, x_3^{A2}) = P(hsm) \times P(x_1^{A3}, x_2^{A3}|hsm) \times P(x_1^{A3}, x_3^{A2}|hsm) \times P(x_2^{A3}, x_1^{A3}|hsm) \times P(x_2^{A3}, x_3^{A2}|hsm) \times P(x_3^{A2}, x_1^{A3}|hsm) \times P(x_3^{A2}, x_2^{A3}|hsm) \quad (4.16)$$

$$P(hsm|x_1^{A3}, x_2^{A3}, x_3^{A2}) = \frac{17}{28} \times \left(\frac{1}{3} \times \frac{2}{3} \times \frac{1}{3} \times \frac{1}{2} \times \frac{2}{3} \times \frac{1}{2}\right) = 0.008$$

$P(C_{hsm}|X) > P(C_{hm}|X)$ olduğundan, FDNB sınıflandırıcısı, test verisini hatasızlığa meyilli (hsm) bir modül olarak tahmin etmiş olur.

4.6. Araştırma Bulguları

Bu çalışmada oluşturulan öğrenme modeli ile sekiz veri seti için Tablo 4.9.'daki sonuçlar elde edilmiştir. Bu tabloda her bir veri seti için 200 koşum sonucundaki ortalama pd, pf ve bal değerleri verilmektedir. Ortalama başarı değerleri tablonun en alt satırında yer almaktadır.

Tablo 4.9. FDNB öğrenme modeli performans sonuçları

Adı	pd (%)	pf (%)	bal (%)
CM1	71	31	70
PC1	69	25	72
PC2	56	6	69
PC3	78	28	75
PC4	96	28	80
KC3	71	27	72
KC4	86	27	78
MW1	49	12	63
Ort.	72	23	72,38

Elde edilen bu sonuçlar, Menzies ve arkadaşları (2007) tarafından yapılan ve özellik seçimi ve log fitrenin kullanıldığı çalışma ile karşılaştırılmıştır. Bu makaleyi karşılaştırma amaçlı kullanmamızdaki sebepler şunlardır:

- Araştırmacılar tarafından çok fazla atıf yapılması (şu ana kadar 703 atıf)
- Referans model olarak kabul edilmesi (state of the art) ve karşılaştırmaların buradaki sonuçlara göre yapılması
- Birden fazla sınıflandırıcı kullanımı sonrasında en başarılı performansı, özellik seçimi+log filtre+NB ile elde etmiş olmaları

Ancak, Song ve arkadaşları (2011), daha önceki bölümlerde değinilen gerekçelerden ötürü, bu çalışmanın mantık olarak yanlış kurgulandığını belirtmişler ve önerilen modeli doğru kurgulayarak tekrarlamışlardır. Bu nedenle bu çalışmada karşılaştırma yapılırken Song ve arkadaşlarının tekrarladıkları deney sonuçlarına yer verilmiştir. Bu çalışmada sadece bal sonuçları verildiğinden, karşılaştırma bu ölçüte göre yapılmıştır. Önerilen FDNB sınıflandırıcı ile bu çalışmadaki sonuçların karşılaştırması bal esas alınarak Tablo 4.10.'da verilmiştir.

Tablo 4.10. FDNB ile NB karşılaştırması

Adı	FDNB	NB
CM1	**70**	69,5
PC1	**72**	62,7
PC2	69	**76,2**
PC3	**75**	71,0
PC4	80	**82,2**
KC3	**72**	69,7
KC4	**78**	68,1
MW1	63	**66,5**
Ort.	72,38	70,74

Bu çalışmada önerilen yöntem 8 veri setin 5'inde daha iyi sonuç üretmiştir. Daha iyi sonuç değerleri tabloda koyu renkli olarak işaretlenmiştir. Ortalama değerde ise %1.64'lük bir iyileşme olmuştur. Bu durumda, NB'de özellik bağımlılık ve diğer önişleme yöntemleri kullanıldığında, performansının daha yukarıya çekilebileceği görülmüştür.

Tablo 4.11.'e bakıldığında, Song ve arkadaşları (2011) tarafından vurgulanan, veri ön işlemenin bütün veri setinde uygulanmasının yanıltıcı sonuçlar doğuracağı iddiasının desteklendiği görülmüş olur. Bu tablodaki sonuçlar; özellik seçimi, normalizasyon ve ayrıklaştırma işlemleri bütün veri setinde uygulanarak ve sonrasında N çapraz validasyon ile eğitim ve test bölümlendirmesi yapılarak elde edilmiştir. Bu yanlış kullanım sonucunda FDNB sınıflandırıcının çok yüksek performans sonuçları üretmiş olduğu görülmektedir. En sondaki sütun ise performans değişimini oransal olarak göstermektedir. Bütün veri setlerinde pozitif yönlü bir performans artışı olduğu

görülmektedir. Ortalama bal değeri %72.38'den %76'ya yükselmiştir, yani yaklaşık %5'lik performans artışı gerçekleşmiştir. Tüm bunlar, öğrenme modelinin yanlış tasarlanmasının yanıltıcı sonuçlar doğurabileceğini açık bir şekilde göstermektedir.

Tablo 4.11. Tüm veri setinde veri önişleme yapılması durumunda FDNB performans sonuçları

Adı	pd (%)	pf (%)	bal (%)	Değişim (%)
CM1	82	29	76	8,57
PC1	78	26	76	5,56
PC2	69	8	78	13,04
PC3	82	27	77	2,67
PC4	97	27	81	1,25
KC3	67	18	74	2,78
KC4	87	27	79	1,28
MW1	54	10	67	6,35
Ort.	77	21,5	76	5,00

4.7. Sonuç

Bu çalışmada NB'de varsayım olarak kabul edilen özelliklerin bağımsızlığı sorgulanmış ve FDNB yöntemi önerilmiştir. Pratikte özelliklerin birbiriyle ilişkili olduğu gerçeğinden hareketle, bunun NB'ye aktarılması durumundaki sınıflandırma performansı değerlendirilmiştir. Olasılık değerleri hesaplanırken, özellikler ikili olarak ele alınmıştır. Yani, her bir özelliğin diğerleri ile birlikte bulunma sayısı hesaplamaya katılmıştır. Uygulama alanı olarak ise yazılım hata kestirimi problemi seçilmiştir. NASA tarafından genel erişime açılan ve araştırmacıların sıklıkla kullandığı sekiz veri seti deneylerde kullanılmıştır. Elde edilen sonuçlar, veri önişleme adımları ile birlikte kullanılan FDNB yönteminin standart NB'ye göre daha başarılı sonuçlar verdiğini göstermiştir. Ayrıca, öğrenme modelinin yanlış kurgulanmasının yanıltıcı yüksek performans sonuçları doğurabileceği de bu çalışma kapsamında gösterilmiştir. İdeal öğrenme modelinde, veri ön işleme parametreleri sadece pratikte elde bulunan eğitim verisi ile belirlenebilir. Tüm veri setinin ön işlemede kullanılması öğrenme modelinin beklenenden daha iyi sonuçlar vermesine sebep olur.

BÖLÜM 5. SONUÇLAR VE ÖNERİLER

Bu tez kapsamında üç ayrı çalışma yapılmıştır. Bu üç çalışmadaki ortak amaç; makine öğrenme algoritmaları ve çeşitli veri madenciliği işlemleri kullanılarak yazılım hata kestiriminin yapılmasıdır. Birinci çalışmada; YSA ve YAK algoritmaları hibrit bir şekilde kullanılarak maliyet-duyarlı bir öğrenme modeli sunulmuş ve yazılım hata kestirimi problemine uygulanmıştır. Bu yöntem ile riskten-kaçınan ve maliyetten-kaçınan işletmeler, kendi yazılım projelerinin ihtiyaçlarına göre en uygun katsayıları belirleyebilmeleri sağlanmıştır. Elde edilen sonuçlar, hem maliyet-duyarlılık yönüyle hem de maliyet-duyarlılık yönü dikkate alınmaksızın, aynı amaç için yayınlanmış diğer çalışmalar ile karşılaştırılmış ve performansının diğerlerinden daha iyi olduğu görülmüştür. İkinci çalışmada; Lojistik Regresyon destekli Bender Metodu ile yazılım kalite metriklerinin, yazılım hata kestiriminde kullanılabilecek eşik değerlere sahip olup olmadığı deneysel olarak incelenmiştir. Sonuçta, tanımlanan iki risk kategorisi için bazı metriklerin etkili eşik değerlerine sahip olduğu görülmüştür. Ayrıca, bu çalışmadaki deney kurgusu oluşturulurken, türetilen eşik değerlerinin benzer özelliklere sahip diğer açık kaynak kodlu yazılım sistemlerinde de uygulanabilmesi amaçlanmıştır. Son olarak üçüncü çalışmada ise yazılım hata kestiriminde yaygın olarak kullanılan NB algoritmasının etkinliği arttırılmaya çalışılmıştır. Bunun için, NB'nin varsayımlarından biri olan özelliklerin bağımsızlığı sorgulanmıştır. Özelliklerin bağımsız olarak ele alınmasındansa, ikili olarak NB formülünde kullanılması durumundaki performansı analiz edilmiştir. Elde edilen sonuçlar, özelliklerin ikili bağımlı yapılması ile yazılım hata kestirim performansında iyileşmelerin olduğunu göstermiştir.

Bu üç çalışma ile çeşitli makine öğrenmesi algoritmaları ve veri ön işleme adımlarının yer aldığı yazılım hata kestirim modelleri oluşturulmuştur. Bu modeller, projenin yönetilmesine, yazılımın geliştirilmesine ve özellikle yazılımın testine destek olurlar.

Proje yöneticileri, yazılım müşteriye teslim edilmeden önce riskli taraflarını görebilirler. Yazılım geliştiricileri, gözden geçirilmeye ihtiyaç duyulan modülleri öğrenir ve yapacakları düzenleme ile yazılım kalitesini arttırmış ve dolayısıyla risk seviyesini azaltmış olurlar. Test personeli, daha fazla üzerine yoğunlaşılması gereken modülleri tespit eder ve kalite durumuna göre faaliyetlerini önceliklendirebilirler.

Tezin çalışmalarının tamamlanması ile beraber gelecek çalışmalar olarak aşağıdakilerin yapılması planlanmaktadır:

- Türetilen eşik değerleri, farklı diller (C++, C# gibi) ile kodlanmış yazılım sistemlerini içeren daha fazla veri seti kullanılarak test ve analiz edilecektir. Ayrıca, bu eşik değerlerinin etkinliği büyük boyutlu endüstriyel projelerde de değerlendirilecektir.
- Tez kapsamında geliştirilen eşik değeri türetim modeli, metot-seviyeli metriklere de uygulanacak ve yapısal diller için de eşik değerleri önerilecektir.
- Eşik değeri türetim modeli, metriklerin bağımsız ele alınmasının yanı sıra, metrikler arası korelasyon da dikkate alınacak şekilde geliştirilecektir
- Veri ön işleme adımları üzerinde yoğunlaşarak, farklı ön işleme yöntemlerinin kullanılmasının FDNB sınıflandırıcı performansına etkisi değerlendirilecektir.
- Bu çalışmada ikili özelliklerin ağırlıkları eşit olarak değerlendirmeye katılmıştır. İkili özelliklerin ağırlıklarının farklı olması durumundaki FDNB sınıflandırıcı performansı değerlendirilecektir. Özellikler ikili olarak ele alınırken, iki özelliğin de eşit ağırlığa sahip olduğu varsayılmıştır. PCA benzeri bir yaklaşımla ikili özellikler kendi içerisinde, her bir özelliğe farklı ağırlık değerleri atanması durumundaki FDNB sınıflandırıcı performansı değerlendirilecektir.

KAYNAKLAR

Alves, T.L., Ypma, C., Visser, J. 2010. Deriving metric thresholds from benchmark data, 2010 IEEE International Conference on Software Maintenance. IEEE, 1–10.

Arar, Ö.F., Ayan, K. 2015. Software defect prediction using cost-sensitive neural network. Applied Soft Computing 33: 263–277.

Arisholm, E., Briand, L.C. 2006. Predicting fault-prone components in a java legacy system, Proceedings of the 2006 ACM/IEEE International Symposium on International Symposium on Empirical Software Engineering - ISESE '06. ACM Press, New York, USA, 8–17.

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

Ayan, K., Kılıç, U. 2012. Artificial bee colony algorithm solution for optimal reactive power flow. Applied Soft Computing 12(5): 1477–1482.

Ayan, K., Kılıç, U., Baraklı, B. 2015. Chaotic artificial bee colony algorithm based solution of security and transient stability constrained optimal power flow. International Journal of Electrical Power & Energy Systems 64: 136–147.

Bansiya, J., Davis, C.G. 2002. A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering 28(1): 4–17.

Basili, V.R., Briand, L.C., Melo, W.L. 1996. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering 22(10): 751–761.

Baxter, G., Frean, M., Noble, J., Rickerby, M., Smith, H., Al., E. 2006. Understanding the shape of Java software. ACM SIGPLAN Notices 41(10): 397–412.

Bender, R. 1999. Quantitative Risk Assessment in Epidemiological Studies Investigating Threshold Effects. Biometrical Journal 41(3): 305–319.

Boehm, B. 1987. Industrial software metrics top 10 list. IEEE Software 4(5): 84–85.

Boehm, B.W., Papaccio, P.N. 1988. Understanding and controlling software costs. IEEE Transactions on Software Engineering 14(10): 1462–1477.

Briand, L.C., Melo, W.L., Wust, J. 2002. Assessing the applicability of fault-proneness models across object-oriented software projects. IEEE Transactions on Software Engineering 28(7): 706–720.

Bruin, J. 2006. Introduction to SAS. UCLA: Statistical Consulting Group. http://www.ats.ucla.edu/stat/sas/notes2/, Erişim Tarihi: 2.11.15.

Bullinaria, J.A., AlYahya, K. 2014. Artificial Bee Colony Training of Neural Networks, Nature Inspired Cooperative Strategies for Optimization (NICSO 2013). Springer International Publishing, 191–201.

Canfora, G., De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., Panichella, S. 2013. Multi-objective Cross-Project Defect Prediction, 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation. IEEE, 252–261.

CapGemini. 2016. World Quality Report 2015–16. https://www.capgemini.com/resources/world-quality-report-2015-16, Erişim Tarihi: 18.5.16.

Carvalho, A. De, Pozo, A., Vergilio, S. 2010. A symbolic fault-prediction model based on multiobjective particle swarm optimization. Journal of Systems and Software 83(5): 868–882.

Carver, J. 2010. Towards reporting guidelines for experimental replications: a proposal, Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research. 2–5.

Catal, C., Alan, O., Balkan, K. 2011. Class noise detection based on software metrics and ROC curves. Information Sciences 181(21): 4867–4877.

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

Catal, C., Diri, B. 2009b. A systematic review of software fault prediction studies. Expert Systems with Applications 36(4): 7346–7354.

Chandrashekar, G., Sahin, F. 2014. A survey on feature selection methods. Computers & Electrical Engineering 40(1): 16–28.

Chapman, M., Callis, P., Jackson, W. 2004. Metrics Data Program, Technical Report, NASA IV and V Facility. NASA.

Chawla, N., Bowyer, K., Hall, L. 2002. SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research 16: 321–357.

Chidamber, S.R., Kemerer, C.F. 1994. A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20(6): 476–493.

Coallier, F. 2001. Software engineering–Product quality–Part 1: Quality model. Geneva, İsviçre.

Conover, W. 1999. Practical nonparametric statistics. (3rd edition)John Wiley & Sons, New York, USA, 367–373.

Crosby, P.B. 1979. Quality is free : the art of making quality certain. McGraw-Hill, New York, USA.

Dejaeger, K., Verbraken, T., Baesens, B. 2013. Toward comprehensible software fault prediction models using bayesian network classifiers. IEEE Transactions on Software Engineering 39(2): 237–257.

Dick, S., Meeks, A., Last, M., Bunke, H., Kandel, A. 2004. Data mining in software metrics databases. Fuzzy Sets and Systems 145(1): 81–110.

Dougherty, J., Kohavi, R., Sahami, M. 1995. Supervised and unsupervised discretization of continuous features. Machine learning: proceedings of the twelfth international conference 12: 194–202.

El Emam, K., Benlarbi, S., Goel, N., Rai, S.N. 2001. Comparing case-based reasoning classifiers for predicting high risk software components. Journal of Systems and Software 55(3): 301–320.

Elish, K.O., Elish, M.O. 2008. Predicting defect-prone software modules using support vector machines. Journal of Systems and Software 81(5): 649–660.

Erdemir, U. Tekin, U., Buzluca F. 2008. Nesneye Dayalı Yazılım Metrikleri ve Yazılım Kalitesi, Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu (YKGS08), İstanbul, 249–258.

Erni, K., Lewerentz, C. 1996. Applying design-metrics to object-oriented frameworks, Proceedings of the 3rd International Software Metrics Symposium. IEEE Comput. Soc. Press, 64–74.

Estabrooks, A., Jo, T., Japkowicz, N. 2004. A multiple resampling method for learning from imbalanced data sets. Computational intelligence 20(1): 18–36.

Evett, M., Khoshgoftar, T., Chien, P. 1998. GP-based software quality prediction. Proceedings of the Third Annual Conference Genetic Programming, 60–65.

Erturk, E., Sezer, E.A. 2015. A comparison of some soft computing methods for software fault prediction. Expert Systems with Applications 42(4), 1872–1879.

Fawcett, T. 2006. An introduction to ROC analysis. Pattern Recognition Letters 27(8): 861–874.

Fayyad, U.M., Irani, K.B. 1992. On the handling of continuous-valued attributes in decision tree generation. Machine Learning 8(1): 87–102.

Fenton, N.E., Pfleeger, S.L. 1998. Software Metrics: A Rigorous and Practical Approach. PWS Publishing Co., London.

Ferreira, K.A.M., Bigonha, M.A.S., Bigonha, R.S., Mendes, L.F.O., Almeida, H.C. 2012. Identifying thresholds for object-oriented software metrics. Journal of Systems and Software 85(2): 244–257.

Flottau, J., Osborne, T. 2015. Software Cut Off Fuel Supply In Stricken A400M. Aerospace Daily & Defense Report. http://aviationweek.com/defense/software-cut-fuel-supply-stricken-a400m, Erişim Tarihi: 27.11.15.

Friedman, N., Geiger, D., Goldszmidt, M. 1997. Bayesian network classifiers. Machine learning.

Gao, K., Khoshgoftaar, T.M., Wang, H. Seliya, N. 2011. Choosing software metrics for defect prediction: an investigation on feature selection techniques. Software: Practice and Experience 41(5): 579–606.

Guo, L., Ma, Y., Cukic, B., Singh, H. 2004. Robust prediction of fault-proneness by random forests, Proceedings - International Symposium on Software Reliability Engineering, ISSRE. IEEE, 417–428.

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

Hall, M. 2007. A decision tree-based attribute weighting filter for naive Bayes. Knowledge-Based Systems 20(2): 120–126.

Hall, M. 1999. Correlation-based feature selection for machine learning. The University of Waikato, Doktora Tezi.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H. 2009. The WEKA data mining software. ACM SIGKDD Explorations Newsletter 11(1): 10–18.

Hall, M., Holmes, G. 2003. Benchmarking attribute selection techniques for discrete class data mining. Knowledge and Data Engineering, IEEE 15(6): 1437–1447.

Hall, M., Smith, L. 1999. Feature Selection for Machine Learning: Comparing a Correlation-Based Filter Approach to the Wrapper. FLAIRS conference 235–239.

Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S. 2011. A Systematic Review of Fault Prediction Performance in Software Engineering. IEEE Transactions on Software Engineering 38(6): 1276–1304.

Halstead, M.H. 1977. Elements of Software Science. Elsevier Science Inc., New York.

He, P., Li, B., Liu, X., Chen, J., Ma, Y. 2015. An empirical study on software defect prediction with a simplified metric set. Information and Software Technology 59, 170–190.

He, Z., Shu, F., Yang, Y., Li, M., Wang, Q. 2011. An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering 19(2): 167–199.

Henderson-Sellers, B. 1996. Object-Oriented Metrics: Measures of Complexity. Prentice Hall.

Herbold, S. 2013. Training data selection for cross-project defect prediction, Proceedings of the 9th International Conference on Predictive Models in Software Engineering - PROMISE '13. ACM Press, New York, New York, USA, 1–10.

Herbold, S., Grabowski, J., Waack, S. 2011. Calculation and optimization of thresholds for sets of software metrics. Empirical Software Engineering 16(6), 812–841.

Hewett, R. 2011. Mining software defect data to support software testing management. Applied Intelligence 34(2): 245–257.

IEEE, 1990. Standard glossary of software engineering terminology.

Japkowicz, N., Myers, C., Gluck, M. 1995. A novelty detection approach to classification, Proceedings of the Fourteenth Joint Conference on Artificial Intelligence. Montreal, Canada, 518–523.

Jiang, L., Li, C., Wang, S., Zhang, L. 2016. Deep feature weighting for naive Bayes and its application to text classification. Engineering Applications of Artificial Intelligence. 52: 26–39.

Jiang, L., Wang, D., Cai, Z., Yan, X. 2007. Survey of Improving Naive Bayes for Classification. Springer Berlin Heidelberg, 134–145.

Jiang, Y., Cukic, B., Ma, Y. 2008. Techniques for evaluating fault prediction models. Empirical Software Engineering 13(5): 561–595.

Jin Huang, Ling, C.X. 2005. Using AUC and accuracy in evaluating learning algorithms. IEEE Transactions on Knowledge and Data Engineering 17(3): 299–310.

Johnson, R., Wichern, D. 1992. Applied multivariate statistical analysis. Prentice hall, Englewood Cliffs, NJ.

Jureczko, M., Madeyski, L. 2010. Towards identifying software project clusters with regard to defect prediction, Proceedings of the 6th International Conference on Predictive Models in Software Engineering - PROMISE '10. ACM Press, New York, New York, USA, 1–10.

Jureczko, M., Spinellis, D.D. 2010. Using Object-Oriented Design Metrics to Predict Software Defects. In Models and Methods of System Dependability. 69–81.

Kanmani, S., Uthariaraj, V.R., Sankaranarayanan, V., Thambidurai, P. 2007. Object-oriented software fault prediction using neural networks. Information and Software Technology 49(5): 483–492.

Karaboga, D., Akay, B. 2009. A comparative study of Artificial Bee Colony algorithm. Applied Mathematics and Computation 214(1): 108–132.

Karaboga, D., Akay, B., Ozturk, C. 2007. Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg.

Karaboga, D., Basturk, B. 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization 39(3): 459–471.

Karaboga, D., Ozturk, C. 2011. A novel clustering approach: Artificial Bee Colony (ABC) algorithm. Applied Soft Computing 11(1): 652–657.

Karaboga, D., Ozturk, C. 2009. Neural networks training by artificial bee colony algorithm on pattern classification. Neural Network World 19(3): 279–292.

Kayah, F. 2008. Discretizing Continuous Features for Naive Bayes and C4. 5 Classifiers. University of Maryland publications: College Park, MD.

Khoshgoftaar, T.M., Allen, E.B., Hudepohl, J.P., Aud, S.J. 1997. Application of neural networks to software quality modeling of a very large telecommunications system. IEEE Transactions on Neural Networks 8(4): 902–909.

Khoshgoftaar, T.M., Allen, E.B., Jones, W.D., Hudepohl, J.I. 1999. Classification tree models of software quality over multiple releases, Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No.PR00443). IEEE Comput. Soc, 116–125.

Khoshgoftaar, T.M., Seliya, N. 2004. Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study. Empirical Software Engineering 9(3): 229–257.

Khoshgoftaar, T.M., Seliya, N. 2003. Analogy-Based Practical Classification Rules for Software Quality Estimation. Empirical Software Engineering 8(4): 325–350.

Kohavi, R. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection, Roceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95). Montreal, Canada, 1137–1145.

Koru, A.G. 2005. Building Defect Prediction Models in Practice. IEEE Software 22(6): 23–29.

Koru, A.G. Liu, H., 2005. An investigation of the effect of module size on defect prediction using static measures. ACM SIGSOFT Software Engineering Notes 30(4): 1–5.

Kubat, M., Matwin, S. 1997. Addressing the Curse of Imbalanced Training Sets: One-Sided Selection, In Proceedings of the Fourteenth International Conference on Machine Learning. Nashville, 179–186.

Liu, H., Hussain, F. 2002. Discretization: An Enabling Technique. Data Mining and Knowledge Discovery 6: 393–423.

Liu, Y., Khoshgoftaar, T.M., Seliya, N. 2010. Evolutionary Optimization of Software Quality Modeling with Multiple Repositories. IEEE Transactions on Software Engineering 36(6): 852–864.

Lu, J., Yang, Y., Webb, G.I. 2006. Incremental Discretization for Naïve-Bayes Classifier. Springer Berlin Heidelberg, 223–238.

Malhotra, R. 2015. A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing 27, 504–518.

Malhotra, R., Bansal, A.J. 2015. Fault prediction considering threshold effects of object-oriented metrics. Expert Systems 32(2): 203–219.

Manning, C.D., Schütze, H. 1999. Foundations of statistical natural language processing. MIT Press, Cambridge, Massachusetts.

Martin, R. 1994. OO design quality metrics - an analysis of dependencies, Proceedings of Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics. 151–170.

McCabe, T. 1976. A complexity measure. IEEE Transactions on Software Engineering 2(4): 308–320.

Menzies, T., Caglayan, B., Kocaguneli, E., Krall, J., Turhan, B. 2012. The promise repository of empirical software engineering data. West Virginia University: Department of Computer Engineering.

Menzies, T., Greenwald, J., Frank, A. 2007. Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Transactions on Software Engineering 33(1): 2–13.

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

Menzies, T., Turhan, B., Bener, A., Gay, G., Cukic, B., Jiang, Y. 2008. Implications of ceiling effects in defect predictors, Proceedings of the 4th International Workshop on Predictor Models in Software Engineering - PROMISE '08. ACM Press, New York, New York, USA, 47–54.

Michaels, P. 2008. Faulty software can lead to astronomic costs. http://www.computerweekly.com/opinion/Faulty-software-can-lead-to-astronomic-costs, Erişim Tarihi: 27.11.15.

Morasca, S., Lavazza, L. 2016. Slope-based fault-proneness thresholds for software engineering measures, in: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering - EASE '16. ACM Press, New York, New York, USA, pp. 1–10.

Moser, R., Pedrycz, W., Succi, G. 2008. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction, Proceedings of the 13th International Conference on Software Engineering - ICSE '08. ACM Press, New York, USA, 181–190.

Nagappan, N., Ball, T., Zeller, A. 2006. Mining metrics to predict component failures, Proceeding of the 28th International Conference on Software Engineering - ICSE '06. ACM Press, New York, New York, USA, 452–461.

Neumann, D. 2002. An enhanced neural network technique for software risk analysis. Software Engineering, IEEE Transactions on 28(9): 904–912.

Nickerson, A.S., Japkowicz, N., Milios, E. 2001. Using unsupervised learning to guide resampling in imbalanced data sets. Proceedings of the Eighth International Workshop on AI and Statitsics. 261–265.

Olague, H.M., Etzkorn, L.H., Gholston, S., Quattlebaum, S. 2007. Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes. IEEE Transactions on Software Engineering 33(6): 402–419.

Oliveira, P., Valente, M.T., Lima, F.P. 2014. Extracting relative thresholds for source code metrics, 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). IEEE, 254–263.

Padberg, F., Ragg, T., Schoknecht, R. 2004. Using machine learning for estimating the defect content after an inspection. IEEE Transactions on Software Engineering 30(1): 17–28.

Pelayo, L., Dick, S. 2007. Applying Novel Resampling Strategies To Software Defect Prediction. NAFIPS 2007 - 2007 Annual Meeting of the North American Fuzzy Information Processing Society. 69–72.

Pérez-Miñana, E., Gras, J.-J. 2006. Improving fault prediction using Bayesian networks for the development of embedded software applications. Software Testing, Verification and Reliability 16(3): 157–174.

Provost, F., Fawcett, T. 2001. Robust Classification for Imprecise Environments. Machine Learning 42(3), 203–231.

Radjenović, D., Heričko, M., Torkar, R., Živkovič, A. 2013. Software fault prediction metrics: A systematic literature review. Information and Software Technology 55(8): 1397–1418.

Rosenberg, L., Stapko, R., Gallo, A. 1999. Object-oriented metrics for reliability, in: Presentation at IEEE International Symposium on Software Metrics.

Ryu, D., Baik, J. 2016. Effective multi-objective naïve Bayes learning for cross-project defect prediction. Applied Soft Computing.

Ryu, D., Choi, O., Baik, J. 2016. Value-cognitive boosting with a support vector machine for cross-project defect prediction. Empirical Software Engineering 21(1), 43–71.

Sánchez-González, L., García, F., Ruiz, F., Mendling, J. 2012. A study of the effectiveness of two threshold definition techniques, 16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012). IET, 197–205.

Selby, R.W., Porter, A.A. 1988. Learning from examples: generation and evaluation of decision trees for software resource analysis. IEEE Transactions on Software Engineering 14(12): 1743–1757.

Seliya, N., Khoshgoftaar, T.M., Hulse, J. 2010. Predicting Faults in High Assurance Software, 2010 IEEE 12th International Symposium on High Assurance Systems Engineering. IEEE, 26–34.

Shatnawi, R. 2015. Deriving metrics thresholds using log transformation. Journal of Software: Evolution and Process 27(2): 95–113.

Shatnawi, R. 2010. A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems. IEEE Transactions on Software Engineering 36(2): 216–225.

Shatnawi, R., Althebyan, Q. 2013. An Empirical Study of the Effect of Power Law Distribution on the Interpretation of OO Metrics. ISRN Software Engineering 2013, 1–18.

Shatnawi, R., Li, W., Swain, J., Newman, T. 2010. Finding software metrics threshold values using ROC curves. Journal of Software Maintenance and Evolution: Research and Practice 22(1): 1–16.

Singh, P., Verma, S. 2009. An Investigation of the Effect of Discretization on Defect Prediction Using Static Measures, 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies. IEEE, 837–839.

Singh, Y., Kaur, A., Malhotra, R. 2010. Empirical validation of object-oriented metrics for predicting fault proneness models. Software Quality Journal 18(1): 3–35.

Song, Q., Jia, Z., Shepperd, M., Ying, S., Liu, J. 2011. A General Software Defect-Proneness Prediction Framework. IEEE Transactions on Software Engineering 37(3): 356–370.

Spinellis, D. 2005. Tool writing: A forgotten art? IEEE Software 22(4): 9–11.

Sun, Z., Song, Q., Zhu, X. 2012. Using Coding-Based Ensemble Learning to Improve Software Defect Prediction. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42(6): 1806–1817.

Taheri, S., Mammadov, M., Bagirov, A. 2011. Improving naive Bayes classifier using conditional probabilities. Proceedings of the Ninth Australasian Data Mining Conference 121: 63–68.

Taheri, S., Yearwood, J., Mammadov, M., Seifollahi, S. 2014. Attribute weighted Naive Bayes classifier using a local optimization. Neural Computing and Applications 24(5): 995–1002.

Tang, M.-H., Kao, M.-H., Chen, M.-H. 1999. An empirical study on object-oriented metrics, Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403). IEEE Comput. Soc, 242–249.

Thwin, M.M.T., Quah, T.-S. 2005. Application of neural networks for software quality prediction using object-oriented metrics. Journal of Systems and Software 76(2): 147–156.

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

Turhan, B., Tosun Mısırlı, A., Bener, A. 2013. Empirical evaluation of the effects of mixed project data on learning defect predictors. Information and Software Technology 55(6): 1101–1118.

Turney, P.D. 1995. Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm. Journal of Artificial Intelligence Research 2, 369–409.

Veado, L., Vale, G., Fernandes, E., Figueiredo, E. 2016. TDTool, in: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering - EASE '16. ACM Press, New York, New York, USA, pp. 1–5.

Vandecruys, O., Martens, D., Baesens, B., Mues, C. 2008. Mining software repositories for comprehensible software fault prediction models. Journal of Systems and 81(5): 823–839.

Wang, S., Yao, X. 2013. Using Class Imbalance Learning for Software Defect Prediction. IEEE Transactions on Reliability 62(2), 434–443.

Williamson, D.F., Parker, R.A., Kendrick, J.S. 1989. The box plot: a simple visual method to interpret data. Annals of internal medicine 110(11), 916–21.

Witten, I.H., Frank, E., Hall, M.A., 2011. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers Inc.

Woods, V., Meulen, R. van der, 2016. Gartner Says Worldwide IT Spending Is Forecast to Decline 0.5 Percent in 2016. Gartner Inc.

Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., Zhou, Z.-H., Steinbach, M., Hand, D.J., Steinberg, D. 2008. Top 10 algorithms in data mining. Knowledge and Information Systems 14(1), 1–37.

Fisher, D.H., Xu, L., Zard N. 1992. Ordering Effects in Clustering, Machine Learning Proceedings 1992: Proceedings of. Morgan Kaufmann, Aberdeen, Scotland, 162–168.

Yang, Y., Webb, G.I. 2009. Discretization for naive-Bayes learning: managing discretization bias and variance. Machine Learning 74(1), 39–74.

Yao, X. 1993. Evolutionary Artificial Neural Networks. International Journal of Neural Systems 4(3), 203–222.

Zhang, H., Sheng, S. 2004. Learning Weighted Naive Bayes with Accurate Ranking, Fourth IEEE International Conference on Data Mining (ICDM'04). IEEE, 567–570.

Zhang, L., Jiang, L., Li, C., Kong, G. 2016. Two feature weighting approaches for naive Bayes text classifiers. Knowledge-Based Systems 100, 137–144.

Zheng, J. 2010. Cost-sensitive boosting neural networks for software defect prediction. Expert Systems with Applications 37(6), 4537–4543.

Zheng, Z., Webb, G.I. 2000. Lazy Learning of Bayesian Rules. Machine Learning 41(1), 53–84.

Zhou, Y., Leung, H. 2006. Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. IEEE Transactions on Software Engineering 32(10), 771–789.

Zhou, Z.-H., Liu, X.-Y. 2006. Training cost-sensitive neural networks with methods addressing the class imbalance problem. IEEE Transactions on Knowledge and Data Engineering 18(1), 63–77.

Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B. 2009. Cross-project defect prediction, Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering on European Software Engineering Conference and Foundations of Software Engineering Symposium - E. ACM Press, New York, New York, USA, 91–100.

Zimmermann, T., Premraj, R., Zeller, A. 2007. Predicting Defects for Eclipse, Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007). IEEE, 9–9.

ÖZGEÇMİŞ

Ömer Faruk ARAR, 01.03.1982'de Şanlıurfa'da doğdu. İlk, orta ve lise eğitimini Şanlıurfa'da tamamladı. 2000 yılında başladığı Marmara Üniversitesi Bilgisayar Mühendisliği Bölümü'nü 2005 yılında bitirdi. 2005 yılında TÜBİTAK BİLGEM'de Araştırmacı olarak işe başladı. Halen, TÜBİTAK BİLGEM'de Başuzman Araştırmacı olarak görev yapmaktadır. Yüksek lisans eğitimi için 2007 yılında başladığı Sakarya Üniversitesi Bilgisayar ve Bilişim Mühendisliği Bölümü'nden 2009 yılında mezun oldu. 2009 yılında yine Sakarya Üniversitesi Bilgisayar ve Bilişim Mühendisliği Bölümü'nde doktora eğitimine başladı.