YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM KUSUR KESTİRİMİ PROBLEMİNDE YAPAY BAĞIŞIKLIK SİSTEMLERİNİN UYGULANMASI

Bilgisayar Yük. Müh. Çağatay ÇATAL

FBE Bilgisayar Mühendisliği Anabilim Dalında Hazırlanan

DOKTORA TEZİ

Tez Savunma Tarihi : 21.11.2008
Tez Danışmanı : Yrd. Doç. Dr. Banu DİRİ (YTÜ)
Jüri Üyeleri : Prof. Dr. Bülent ÖRENCİK (İTÜ)
Prof. Dr. Coşkun SÖNMEZ (YTÜ)
Prof. Dr. Oğuz DİKENELLİ (EÜ)
Prof. Dr. Oya KALIPSIZ (YTÜ)

İSTANBUL, 2008

İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ

AAE	Average Absolute Error
AAR	Average Absolute Error
Ab	Antikor
ACC	Accuracy
AÇK	Alıcı Çalışma Karakteristikleri
Ag	Antijen
AIRS	Artificial Immune Recognition Systems
AIS	Artificial Immune Systems
ALT	Acquisition, Logistics & Technology
ANOVA	Analysis of Variance
APA	Average Prediction Age
APC	Antigen Presenting Cells
ARB	Artificial Recognition Ball
ARE	Average Relative Error
ARM	Automated Requirement Measurement
ASA	The Assistant Secretary of the U.S. Army
AUC	Area Under Curve
BBN	Bayesian Belief Networks
BCR	B Cell Receptor
BDF	Boolean Discriminant Functions
CAFE	From Concepts to Application in System-Family Engineering
CART	Classification and Regression Trees
CBO	Coupling Between Object Classes
CBR	Case Based Reasoning
CFS	Correlation Based Feature Selection
CGA	Clustering Genetic Algorithm
ChgSize	Size of Changed Classes
CK	Chidamber-Kemerer metrikleri
CLONALG	Clonal Selection Algorithm
COSMIC	The Common Software Measurement International Consortium
CSA	Consistency Based Feature Selection
CSCA	Clonal Selection Classifier Algorithm
DIT	Depth of Inheritance Tree
DNA	Deoksiribo Nükleik Asit
DR	Discrepancy Reports
EC	Export Coupling
EM	Expectation Maximization
ESAPS	Engineering Software Architectures, Processes and Platforms for System-Families
FALCON	Fuzzy Self-Adaptation Learning Control Network
FAMILIES	Fact-based Maturity through Institutionalization Lessons-learned and Involved Exploration of System-family Engineering
FDL	Formal Definition Language
FNR	Fuzzy Nonlinear Regression
GBDF	Generalized Boolean Discriminant Functions
GRNN	General Regression Neural Network
HR	High Risk
IADIS	International Association for Development of the Information Society
IASTED	International Association of Science and Technology for Development
IBL	Instance Based Learning

IEEE	Institute of Electrical and Electronics Engineers
IFPUG	International Function Point Users Group
IgE	Immünoglobün E
IJCAI	International Joint Conference on Artificial Intelligence
ITEA	Information Technology for European Advancement
IV&V	Independent Verification and Validation Facility
K/S	Kolmogorov-Smirnov İstatistiği
kd-tree	K-dimensional tree
KİP	Kalite İyileştirme Paradigması
Knn	K-nearest Neighbour
LAD	Least Absolute Deviations
LCOM	Lack of Cohesion of Methods
LMR	Low Module Risk
LRF	Logistics Regression Functions
LSR	Lineer Standart Regresyon
MAM	Marmara Araştırma Merkezi
MAR	Mean of Absolute Residual
MBLR	Multivariate Binary Logistic Regression
MBR	Memory Based Reasoning
MFM	Most Frequently Fixed
MHC	Major Histocompatibility Complex
MLP	Multi Layer Perceptron
MOOD	Metrics for Object Oriented Design
MRF	Most Recently Fixed
MRM	Most Frequently Modified
MS	Multiple Sclerosis
MSE	Mean Squared Error
NIPS	Neural Information Processing Systems
NIST	National Institute for Standards and Technology
NOC	Number of Children
NVPS	N-version Programming Scheme
OSR	Optimized Set Reduction
PAKDD	Pacific-Asia Conference on Knowledge Discovery and Data Mining
PCA	Principal Component Analysis
PD	Probability of Detection
PF	Probability of False Alarm
Prec	Precision
PRM	Poisson Regression Model
QA	Quality Assurance
QIP	Quality Improvement Paradigm
QMOOD	Quality Model for Object-Oriented Design
RAIN	Resource Limited Artificial Immune Network
RBF	Radial Basis Function
RBS	Recovery Block Scheme
RF	Random Forests
RFC	Response for a Class
RNA	Ribo Nükleik Asit
ROC	Receiver Operating Characteristic
RUP	Rational Unified Process
RvC	Regression via Classification
SEI	Software Engineering Institute

SIGSOFT	Special Interest Group on Software Engineering
SLOC	Source Lines of Code
SQL	Structured Query Language
SVM	Support Vector Machines
TCR	T Cell Receptor
TPR	True Positive Rate
TSVM	Transductive Support Vector Machine
TÜBİTAK	Türkiye Bilimsel ve Teknolojik Araştırma Kurumu
UBLR	Univariate Binary Logistic Regression
UCI	University of California Irvine
UML	Unified Modeling Language
WEKA	Waikato Environment for Knowledge Analysis
WMC	Weighted Method Count
WMC	Weighted Method Count
XOR	Exclusive-OR
YATSI	Yet Another Two Stage Idea
YBS	Yapay Bağışıklık Sistemleri
YSA	Yapay Sinir Ağları
YSA	Yapay Sinir Ağı
ZIP	Zero Inflated Poisson Regression Model

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Motivasyon, heyecan ve sabır...

Bilimsel ve teknolojik araştırmaların emek yoğun ve insan merkezli olduğu dikkate alınırsa, araştırmada başarıya giden yolun yukarıdaki üç faktörden geçtiğini söylemek mümkündür. Motivasyon, bu yolun başlangıç noktası iken heyecan ve sabır, kesinliği tartışma götürmeyecek bilgilere ulaşmadaki kritik unsurlardandır.

Lisans öğrenimimin ardından, TÜBİTAK'ta büyük ölçekli bir yazılım projesinin tüm yazılım yaşam çevriminde görev almakla birlikte profesyonel yazılım geliştirme süreçlerini, bu süreçlerde üretilen belgeleri ve diğer çıktıları, yazılım mühendislerinin uygulamada yaşadığı zorlukları ve araştırmaya açık alanları gözlemlemiş oldum. İş yaşamımda çalışma yaptığım Yazılım Mühendisliği alanında doktora tez çalışması yapıyor olmak, benim için en önemli motivasyon kaynağı olmuştur.

Yüksek Lisans öğrenimim sırasında, ağırlıklı olarak Yapay Zekâ dersleri almam ve bu yönde yüksek lisans tez çalışması gerçekleştirmem nedeniyle, Yapay Zekâ alanını da bu doktora tez çalışmasına katma yönünde bir isteğim oldu. Doktora programında almış olduğum Yazılım Mühendisliği ağırlıklı dersler sayesinde, doktora tezimi ağırlıklı olarak bu yönde gerçekleştirmeye ve yöntem olarak son yıllarda popülerlik kazanan Yapay Bağışıklık Sistem paradigmasından yararlanmaya karar verdim. Bu tez çalışması, bu nedenle Yazılım Mühendisliği ve Yapay Zekâ alanlarının kesişimi olarak değerlendirilebilir.

Yazılım Kusur Kestirimi konusunda çalışma yapmaya başladığım 2006 yılının başlarında; geçecek yıllarla birlikte konunun bu kadar popülerlik kazanabileceğini, önemli yazılım mühendisliği dergilerinde ve konferanslarında bu konuda yayınların sıkça çıkacağını, araştırmamız sırasında çok farklı alanlara açılımlar sağlayabileceğimizi ve son dönemde ele alacağımız problemleri içeren bir Araştırma Projemizin (TÜBİTAK-1001) kabul edileceğini hiç tahmin etmiyordum.

Bu tez çalışması süresince; değerli görüşleri, bilgileri ve tecrübesi ile yönlendirme sağlayan, her aşamada sabrı ve motivasyonu aşılayan doktora tez danışmanım Sn. Yrd. Doç. Dr. Banu DİRİ'ye teşekkür ederim. Hafta içi ve çoğu zaman hafta sonlarında, elektronik postalarla gönderdiğim sorularıma gelen hızlı yanıtları ve yorumları sayesinde vakit kaybetmeden yol alabildim ve araştırmanın bir yaşam biçimi olduğunu kendisinden öğrendim. Bu tez çalışmasının sonrasında da, Araştırma Projemizde kendisiyle çalışmalara devam edebiliyor olmak benim açımdan oldukça sevindiricidir.

Bu tez konusunun belirlenmesinde ve kapsamının şekillendirilmesinde değerli görüşlerini sunan ve Yazılım Mühendisliği alanında araştırma yapmaya beni teşvik eden Sn. Prof. Dr. Oya KALIPSIZ'a teşekkür ederim. Kendisinden doktora öğrenimim sırasında almış olduğum, Yazılım Kalitesi ve Test Teknikleri, Yazılım Proje Yönetimi gibi derslerde araştırma yapmaya bizleri yönlendirmeseydi bu kapsamda bir tez çalışması ortaya çıkamazdı.

2002 yılından bu yana çalışmakta olduğum TÜBİTAK-Marmara Araştırma Merkezi-Bilişim Teknolojileri Enstitüsü'nde kurum içi danışmanım ve Enstitü Müdürümüz Sn. Prof. Dr.

Bülent ÖRENCİK'e, bu tez çalışmasının son zamanlarında ele alınan konuları temel alan bir Araştırma Projesi oluşturmak üzere beni teşvik ettiği için teşekkür ederim. Tez çalışması sırasında üretilen bilimsel yayınların ulusal ve uluslararası konferanslarda paylaşılabilmesi için yönetimsel desteği sunması, kurum içinde Çalışma Alanı Gruplarını kurarak projelerin yanı sıra aynı konuda çalışan Enstitü araştırmacılarıyla ortak çalışma yapabilmemizi sağlaması ve tez izleme komitesi toplantılarında sunduğu görüşleri bu tezin bu noktalara ulaşmasında önemli katkılar sağlamıştır.

Ayrıca, doktora tez çalışmam sırasında gerekli bildirileri ve makaleleri kolaylıkla elektronik ortamda bulabildiğim, farklı üniversitelerden kütüphanesi sayesinde kitap temin edebildiğim ve teknolojik alt yapısından yararlandığım, çalıştığım kurum TÜBİTAK-Marmara Araştırma Merkezi'nin değerli yöneticilerine teşekkür ederim. Bu araştırma alt yapısı olmasaydı, böyle bir tez ve Araştırma Projesi bu süre zarfında ortaya çıkamayacaktı.

Yüksek Lisans tez danışmanım ve doktora tez izleme komitesi üyelerinden Sn. Prof. Dr. Coşkun SÖNMEZ'e Yapay Zekâ alanında sunmuş olduğu değerli bilgilerinden dolayı teşekkür ederim. Kendisinden almış olduğum Yapay Zekâ, Uzman Sistemler ve İleri Yapay Zekâ gibi dersler, bu konudaki bilgilerimi ve konuya olan ilgimi arttırmıştır. Ezberden uzak, yeterli çalışmayla her alanda başarılı olunabileceğini gösteren, sürekli pozitif düşünceyle öğrenmeyi öğreten yaklaşımlarından çok şeyler öğrendim.

Hayatımın her aşamasında sevgi ve desteklerini esirgemeyen, bana duydukları güveni boşa çıkarmamak için gereken çabayı gösterdiğim, doktorayı başarıyla tamamlamak için beni yüreklendiren ve motive eden sevgili anneme, babama ve tüm aileme teşekkürlerimi sunarım.

YAZILIM KUSUR KESTİRİMİ PROBLEMİNDE YAPAY BAĞIŞIKLIK SİSTEMLERİNİN UYGULANMASI

ÖZET

Yazılımların artan karmaşıklığı ile birlikte kullanıcı beklentilerinin de artması, bu beklentilerin ve yazılım kalitesinin Yazılım Kalite Mühendisliği adı verilen mühendislik disiplini ile ele alınmasını gerekli hale getirmiştir. Yazılım kusur kestirimi, Yazılım Kalite Mühendisliği disiplini içerisinde yer alan alt kalite güvence aktivitelerinden birisidir. Bu tez çalışmasında, Yazılım Kusur Kestirimi problemi Yapay Bağışıklık Sistem (YBS) paradigması tabanlı algoritmalarla çözülmeye çalışılmış ve bu kapsamda var olan algoritmalar analiz edilerek yeni modeller önerilmiştir. 5 adet NASA veri kümesinde YBS tabanlı farklı sınıflayıcıların performansları incelenmiş ve literatürde raporlanmış en iyi makine öğrenmesi algoritmaları ile kıyaslamalar gerçekleştirilmiştir. Bu çalışmalar sayesinde, literatürde ilk kez YBS tabanlı sınıflandırma algoritmalarının performansları kusur kestirimi için incelenmiştir. AIRS algoritmasının yüksek performans sunması sayesinde, sonraki çalışmada özellik azaltma teknikleri dikkate alınarak; "korelasyon tabanlı özellik seçimi" nin uygulandığı AIRS tabanlı bir model önerilmiştir. Bu model, özellikle çok büyük veri kümelerinde yüksek performans sunmuş ve birçok öğrenme algoritmasından daha iyi sonuçlar elde edilmiştir. Sınıf seviyesindeki metriklerle, benzer bir model oluşturulabilirse tasarım aşamasında kusur eğilimli modüller belirlenerek sınıfların yeniden düzenlenmesi sağlanabilecektir. Bu fikirden hareketle, sınıf seviyesindeki Chidamber-Kemerer (CK) metrikleri kullanılarak kusur kestirim modelleri üzerinde deneysel analizler gerçekleştirilmiştir. Kalıtım ağacının boyutuna ilişkin sınıf metriğinin kusur kestiriminde en düşük etkiye sahip olduğu ve nesneler arasındaki bağlılığa ilişkin metriğin en yüksek öneme sahip olduğu deneysel verilerle gösterilmiştir. CK metrikleri ve kod satır sayısının birlikte kullanıldığı durumda, AIRS tabanlı kestirim modelinin en yüksek performansı sunduğu raporlanmıştır.

Yarı eğiticili öğrenme konusu, makine öğrenmesinin son yıllardaki aktif araştırma konularından birisidir. Bu kapsamda, Yapay Bağışıklık Sistem paradigması tabanlı yarı-eğiticili bir öğrenme algoritması önerilmiş ve problemde uygulanmıştır. Önerilen algoritmanın sınırlı sayıda kusur verisi kullanıldığı durumda, AIRS algoritmasının performansını arttırdığı tespit edilmiştir. Ancak önerilen yaklaşımın bazı algoritmalar için performansı düşürdüğü de diğer tespit edilen noktalardandır. Ayrıca, yazılımın yeniden kullanılabilirliğini sağlamak üzere 2000'lerde popülerlik kazanmış olan "Yazılım Ürün Hatları" yaklaşımında, yazılım kusur kestiriminin gerçekleştirilebilmesi için çerçeve bir model önerilmiştir. Bu tez çalışması sırasında YBS tabanlı kusur kestirim modellerine ek olarak, "kestirim merkezli yazılım yaşam çevrimi" ve "kestirim merkezli yazılım süreçleri" isimli yeni bir geliştirme yaklaşımı ve yeni bir süreç yaklaşımı önerilmiştir. Aynı çalışmada, yazılım kusur kestiriminin sistem test senaryolarını önceliklendirme noktasında nasıl kullanılabileceği ve sağlayacağı yararlar vurgulanmıştır.

Anahtar kelimeler: Yazılım kusur kestirimi, yazılım mühendisliği, kalite mühendisliği, yapay bağışıklık sistemleri, kalite kestirimi

APPLICATION OF ARTIFICIAL IMMUNE SYSTEMS IN SOFTWARE FAULT PREDICTION PROBLEM

ABSTRACT

Because of rising complexity in software systems and increasing user expectations, it is necessary to manage the software quality as an engineering discipline called "Software Quality Engineering". Software fault prediction is one of the quality assurance activities which locates in Software Quality Engineering discipline. In this thesis study, the software fault prediction problem has been tried to be solved with Artificial Immune Systems (AIS) based algorithms and new models have been proposed by examining existing algorithms. The performance of different AIS based classifiers has been examined on five NASA datasets and compared with the best machine learning algorithms reported in literature. With the help of this study, AIS based classifiers' performance have been investigated for the first time in literature. Thanks to the high performance of AIRS algorithm, an AIRS based model using "correlation-based feature selection" technique has been proposed. This model specifically provided high performance for very large datasets and results were better than many learning algorithms. If a similar model can be built using class-level metrics, fault-prone modules can be identified and refactored in design phase. By using this idea, experimental analysis have been conducted using class-level Chidamber-Kemerer (CK) metrics on fault prediction models. It has been empirically proved that depth of inheritance tree (DIT) is the least significant metric and coupling between object classes (CBO) is the most significant one for the fault prediction. It has been reported that AIRS based prediction model using CK metrics and lines of code provides the highest performance.

Semi-supervised learning is one of the most active research topics in machine learning. An AIS based semi-supervised learning algorithm has been proposed and applied in this problem. It has been determined that AIRS algorithm's performance increases with the proposed algorithm when there is limited fault data. However, this approach decreases the performance of some algorithms for software fault prediction. In addition, a framework to use the software fault prediction in "Software Product Lines" approach, which is popular since the beginning of the 2000s for software reusability, has been proposed. In addition to AIS based fault prediction models, "prediction-centric software life cycle" and "prediction-centric software processes" have been proposed as a new development approach and a new process approach. Furthermore, that study showed how to use the fault prediction approaches to prioritize system test cases and emphasized the benefits of fault prediction approaches.

Keywords: Software fault prediction, software engineering, quality engineering, artificial immune systems, quality prediction

1. GİRİŞ

Yazılımların artan karmaşıklığı ile birlikte kullanıcı beklentilerinin de artması, bu beklentilerin ve yazılım kalitesinin Yazılım Kalite Mühendisliği adı verilen mühendislik disiplini ile ele alınmasını gerekli hale getirmiştir. Yazılım Kalite Mühendisliği disiplini altında; kusur dayanıklılığı (fault tolerance), biçimsel doğrulama (formal verification), inceleme (inspection), test, kusur kestirimi (fault prediction) gibi farklı alt kalite güvence (quality assurance) aktiviteleri bulunmaktadır. Mevcut yazılım geliştirme süreçleri dikkate alındığında, bu alt aktivitelerden en fazla uygulananı test aktivitesidir (Tian, 2005).

Görev-kritik sistemlerde yukarıda açıklanan kalite güvence aktivitelerinin uygulanmasının yanı sıra, yazılım kalitesinin sürekli şekilde ölçülmesine ihtiyaç duyulmaktadır. Yazılım geliştirme projelerinde yüksek riskli bileşenlerin kestiriminin yapılması için yazılım metrikleri yarar sağlamaktadır (Basili vd., 1996). Böylece proje yöneticisi ya da kalite güvence grubu, gerekli parasal kaynağı ve insan gücünü hata eğilimli modüllerin testine ayırarak test sürecini kısaltıp ürün kalitesini arttırabilmektedir.

Balistik füze kontrol sistemleri, insansız hava aracı ve uçuş kontrol yazılımları; görev-kritik gerçek zamanlı sistemlere örnek olarak verilebilir. Bu sistemler güvenilir şekilde çalışmak zorundadır ve kararlı (dependable) özellikte olması beklenir. Kararlı sistemler oluşturmak için; kusurdan sakınma (fault avoidance), kusurları ortadan kaldırma (fault removal), kusur dayanıklılığı (fault tolerance) ve kusur kestirimi (fault prediction) yaklaşımları kullanılmaktadır. Bu açıdan bakıldığında, yazılım kusur kestirim çalışmaları kusur eğilimli modülleri belirleyerek test sürecini iyileştiren bir teknik ve kararlı sistemler geliştirmek için kullanılabilecek önemli bir yaklaşımdır.

Kusur kestirimi modellerinde ihtiyaç duyulan bilgiler; araçlarla otomatik toplanan yazılım metrikleri ve geliştirilen yazılımın önceki sürümünün ya da benzer bir projenin kusur bilgileridir. Daha genel bir ifadeyle modellerde, müşteride ya da test aşamasında çıkan hata bilgileri ve yazılım metrikleri kullanılmaktadır. Hataların, hata takip sistemleri içerisinde saklanması gerekmektedir. Ülkemizde bazı firmalar ve kurumlar, Bugzilla adı verilen hata takip sistemini bu amaçla kullanmaktadır.

Literatürde bu tür modeller Kusur Kestirimi, Eksiklik Kestirimi (Defect Prediction) veya Yazılım Kalite Sınıflandırma (Software Quality Classification) başlıkları ile ele alınmaktadır. Çalışmalar incelendiğinde, eksiklik (defect) ve kusur terimlerinin birbiri yerine geçen terimler olarak kullanıldığı görülecektir. Eksiklik terimi; hata (error), kusur, arıza (failure) kelimelerini

kapsayan daha genel bir terim olarak literatürde kullanılmaktadır. Bu terimler arasındaki farklılıklar takip eden bölümlerde açıklanacaktır.

Çalışmalarda farklı terimler kullanılsa da, ortaya konulmak istenen model genellikle bir bağımlı (dependent) değişkenin yer aldığı ve n tane bağımsız (independent) değişkenin bulunduğu modelin kurulması üzerinedir. Kusur kestirimi çalışmalarında bağımlı değişken, modülün hata eğilimli olup olmadığını göstermektedir. Bu kestirimi yapmak üzere kullanılan bağımsız değişkenler ise, süreç veya ürün metrikleri olabilmektedir. Yapılan çalışmaların çok büyük bir kısmı, ürün metriklerini ele alarak modelleri kurmaya odaklanmıştır. Firmalar ya da kurumlar yeniden organize (reorganization) olduğu durumda, süreçlerde meydana gelen değişimler neticesinde süreç metriklerine dayalı kestirim modellerinin geçersiz hale geleceğini söylemek mümkündür. Bu nedenle; çoğu durumda araştırmacılar ürün odaklı modeller geliştirip, ürün metriklerini dikkate almaktadırlar.

Ürün metrikleri, metot ve sınıf seviyesindeki metrikler olarak 2 gruba ayrılabilir. Metot seviyesindeki metrikler; yordamsal (procedural) veya nesneye yönelik programlama paradigmasının kullanıldığı iki durumda da kestirim amaçlı kullanılabilmektedir. Bunun sebebi, iki paradigma içerisinde metotların mevcut olmasıdır. Halstead ve McCabe metrikleri, metot seviyesindeki metriklerden en fazla kullanılanlarıdır. Sınıf seviyesindeki metrikler ise, her türlü nesneye yönelik metrik olabilir. Dolayısıyla sınıf seviyesindeki metriklerin, sadece nesneye yönelik paradigma ile geliştirilen yazılımlar için kullanılabileceği açıktır. Chidamber-Kemerer (CK) sınıf seviyesindeki metrikler (Chidamber ve Kemerer, 1994), hata eğilimli modüllerin kestirimi için sıkça kullanılmaktadır.

Kestirim modelleri genellikle, istatistiksel yöntemleri ya da makine öğrenmesi algoritmalarını kullanmaktadır. Bu amaçla literatürde; Karar Ağaçları, Bulanık Mantık, Genetik Programlama, Durum-Tabanlı Usavurum (Case-Based Reasoning), Yapay Sinir Ağları, Lojistik Regresyon gibi birçok yöntem kullanılmıştır. Bu tez çalışmasında, literatürdeki diğer modellerle kıyaslama yapılabilmesi için NASA Metrik Veri Programı kapsamında oluşturulmuş ve PROMISE havuzunda (Sayyad ve Menzies, 2005) yer alan NASA'ya ait projelerin veri kümeleri tercih edilmiştir.

Test sırasında ya da müşteride hataya yol açan modüller, genellikle tüm modüllerin çok küçük bir kısmına karşı gelmektedir. Bu nedenle; bu tür hata veri kümelerinin "dengeli olmayan" (imbalanced) veri kümeleri olduğu söylenebilir. Dengeli olmayan veri kümeleri üzerinde yürütülen algoritmaların ya da kurulan modellerin performansını kıyaslamak için doğruluk

(accuracy) değerlendirme kriterini kullanmak mümkün değildir.

Makine öğrenmesi konusunda araştırma yapanlar, bu tür veri kümeleri için farklı değerlendirme kriterleri önermişlerdir. Örneğin; ROC (Receiver Operating Characteristic) eğrisi alanından, hata matrisinin (confusion matrix) sadece diyagonaldeki değerlerine bakarak, sadece F-ölçüm değerini kullanarak ya da G-ortalama1, G-ortalama2 ve F-ölçüm değerlerinin hepsini birden dikkate alarak değerlendirme yapanlar vardır. Örneğin; Ma ve arkadaşları (2006) bu tür modelleri kıyaslamak için G-ortalama1, G-ortalama2, F-ölçüm değerlerinin hepsini kullanarak Rastgele Orman (Random Forests) algoritmasının diğer makine öğrenmesi algoritmalarından daha üstün sonuç verdiğini raporlamışlardır.

Bu tez çalışmasının temel amacı yazılım kusur kestirimi probleminde Yapay Bağışıklık Sistem paradigmasından yararlanmak olup, diğer bir amacı da Yazılım Mühendisliği alanında çalışma yapan araştırmacılara Yapay Bağışıklık Sistem paradigmasının diğer Yazılım Mühendisliği problemlerinde uygulanmasını teşvik etmek ve daha olgun bir disipline ulaşmak için Makine Öğrenmesi algoritmalarından yararlanmalarını sağlamaktır.

Bu bölümde; temel kavramlar, teze ilişkin hipotez, amaçlar, literatüre sağlanan katkılar, tez çalışması sırasında gerçekleştirilen yayınlar ve tezin yapısı alt başlıklar halinde sunulacaktır.

1.1 Temel Kavramlar

Bu alt bölümde, tez çalışmasının ana probleminin ilişkili olduğu konu başlıkları irdelenerek çözüm yöntemi olarak seçilen yaklaşım kısaca açıklanacaktır.

1.1.1 Yazılım Kalite Mühendisliği

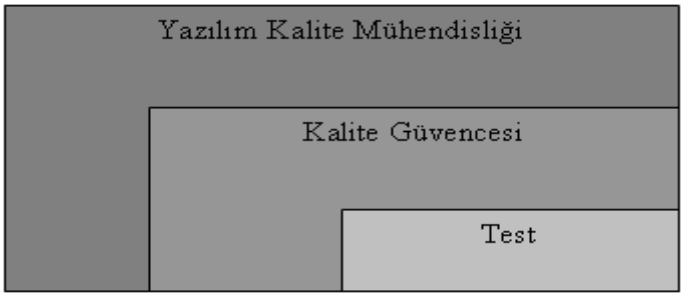
Bir yazılım sisteminin doğru işleri yaptığı geçerleme (validation) aktiviteleri ile bu işleri doğru şekilde gerçekleştirdiği ise doğrulama (verification) aktiviteleri ile kontrol edilmelidir. Yazılım kalitesinin tanımı, bakış açılarına göre değişiklikler göstermekte ve kaliteyi beş temel bakışla ele almak mümkündür (Tian, 2005).

1. Transandantal / deneyüstü bakış: “Kalitenin tanımlanması zordur veya soyut terimlerle tanımlanır, ancak mevcut ise farkına varılır. Kullanıcıları memnun eden bazı soyut özelliklerle ilişkilidir”.
2. Kullanıcı bakışı: Kalite, kullanıcı ihtiyaçlarına uygunluktur.
3. Üretim bakışı: Kalite, süreç standartlarına uyumdur.

4. Ürün bakışı: Ürünün iç özellikleri dikkate alınarak dış ürün davranışı ve kalite iyileştirilebilir.

5. Değer tabanlı bakış: Kalite, bir yazılım için kullanıcının para ödeme isteğidir.

Yazılımların artan karmaşıklığı ile birlikte kullanıcı beklentilerinin de artması, bu beklentilerin ve yazılım kalitesinin Yazılım Kalite Mühendisliği adı verilen mühendislik disiplini ile ele alınmasını gerekli hale getirmiştir. Yazılım Kalite Mühendisliği disiplini altında; kusur dayanıklılığı, biçimsel doğrulama, inceleme, test, kusur kestirimi gibi alt kalite güvence aktiviteleri bulunmaktadır. Bu alt aktivitelerin en önemlilerinden olan ve en fazla uygulananı test aktivitesidir. Şekil 1.1' de yazılım testinin, kalite güvencesi aktivitelerinden sadece birisi olduğu ve yazılım kalite mühendisliğinin farklı kalite güvencesi aktivitelerini içeren üst şemsiye olduğu gösterilmektedir.



Şekil 1.1 Yazılım kalite mühendisliği

Kalite mühendisliği bir süreç ile ele alınırsa, bu süreç içerisinde üç ana aktivite grubunun mevcut olduğunu söylemek mümkündür (Tian, 2005):

1. *Kalite Güvence Aktiviteleri Öncesi (Pre-QA activities) / Kalite Planlama:* Kalite güvence aktiviteleri gerçekleştirilmeden önce, kalite amaçlarının saptanması ve kalite stratejisinin belirlenmesi gerekmektedir. Kalite planlama içerisinde iki temel aktivite yer almaktadır:
    - Özel kalite amaçlarının saptanması
    - Kalite stratejisinin şekillendirilmesi
        - ✓ Uygun kalite güvence aktivitelerinin seçilmesi
        - ✓ Geri besleme, kalite değerlendirme ve iyileştirme için uygun kalite ölçümlerinin ve modellerinin seçilmesi
2. *Kalite Güvence Aktivitelerinin Yürütülmesi (In-QA activities) :* Planlanan kalite güvence

aktivitelerinin yürütülmesi ve belirlenen kusurların yönetilmesi gerekmektedir. Kaynakların en fazla tüketildiği gruptur.

3. *Kalite Güvence Aktiviteleri Sonrası (Post-QA activities):* Bu aktiviteler, isminin aksine kalite güvence aktiviteleri bittikten sonra gerçekleştirilen aktiviteler değildir. Kalite güvence aktiviteleri başladıktan sonra, bu aktivitelere paralel olarak farklı ölçüm ve analizler gerçekleştirilir. Süreç sırasında ya da "faaliyet sonu inceleme verileri" (post-mortem data) kullanılarak kalite değerlendirilmesi gerçekleştirilir ve kalite amaçlarının sağlanıp sağlanmadığı kontrol edilir.

Şekil 1.2, ana aktivite gruplarının yazılım kalite mühendisliği süreci içerisindeki etkileşimini yansıtmaktadır. Kalite planlama grubundan çıkan üç ok; kalite amaçlarını, kalite güvence aktivitelerini, seçilen ölçüm ve modelleri temsil etmektedir. Kalite amaçlarının sağlandığı durumda sürecin tamamlandığı, aksi durumda ise üç adet geri beslemenin söz konusu olduğu şekilde gösterilmektedir. Kalite mühendisliği sürecinde iki tür geri beslemeden söz edilebilir:

- *Kalite güvence aktivitelerine doğru olan kısa dönem doğrudan geri besleme:* Aktivitelerin zaman sırasının yeniden düzenlenmesi ya da gelişimin incelenmesi amaçlı geri besleme sunulabilir.

- *Uzun dönem geri besleme:* İki türde uzun dönem geri beslemesi gerçekleşebilir.

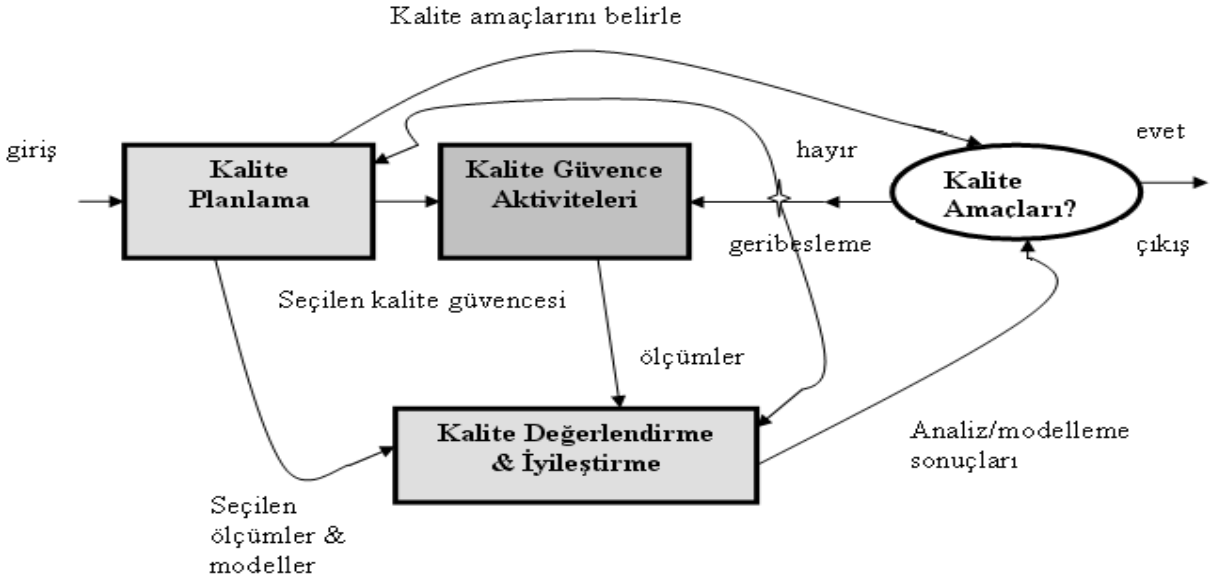
    - Kalite planlamaya olan geri besleme, başarılamaz türden olan amaçların ve uygun olmayan kalite güvence stratejisinin düzeltilmesini sağlar. Bu yaklaşım mevcut proje yerine, sonraki projelerde uygulanır.

    - Kalite değerlendirme ve iyileştirmeye olan geri besleme ise, model uygunsuzluğu olduğu durumda modelin değiştirilmesi ile çözülebilir.

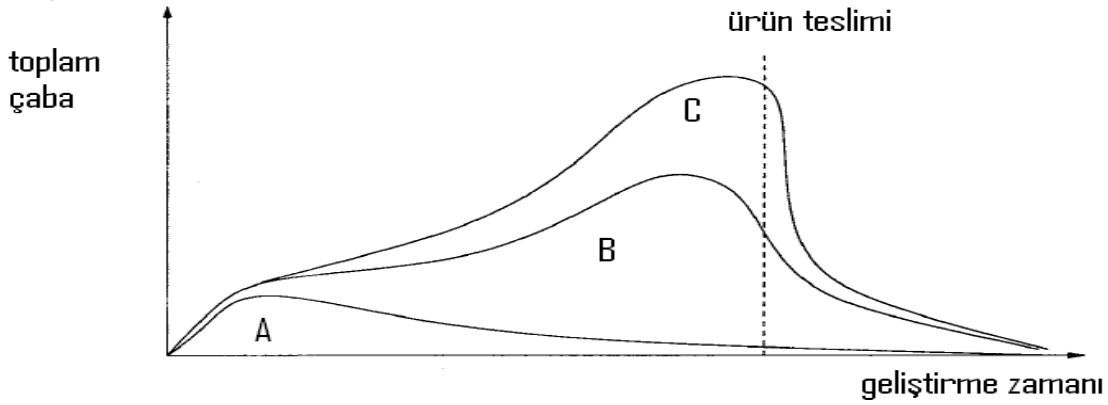
TAME projesi (Basili ve Rombach, 1988) ve devamındaki çalışmalarda; ölçüm, analiz, geri besleme, organizasyonel destek ile kalite iyileştirmenin başarıldığı raporlanmıştır. Bu adımların yürütüldüğü çerçeve, "Kalite İyileştirme Paradigması (KİP)" (Quality Improvement Paradigm-QIP) olarak adlandırılır ve anlama, değerlendirme, paketleme adımlarından oluşur. Mevcut durumun anlaşılmasının ardından iyileştirme fırsatları saptanarak deney ve pilot projelerle süreç değişikliklerinin etkileri değerlendirilir ve son adımda deneyim, güncellenmiş süreç ve sonuçlar paketlenerek organizasyonun iyileştirme programına yeni bilgiler olarak eklenir. Yazılım kalite mühendisliği süreci ve KİP arasında, aktiviteler açısından ilişki kurmak mümkündür. Kalite planlamayı, anlama adımıyla; kalite güvence aktivitelerinin

yürütülmesini değerlendirme adımında değişikliklerin gerçekleştirilmesiyle; kalite güvencesi aktiviteleri sonrasını paketleme adımıyla ilişkilendirebiliriz (Tian, 2005).

Şekil 1.3' de kalite mühendisliği süreci içerisindeki aktivitelerin, yazılım yaşam çevriminde tükettiği çabalar resmedilmektedir. A kalite planlamayı, B kalite güvence aktivitelerinin yürütülmesini, C ise kalite analizini göstermektedir. Ürün geliştirmenin başlangıç aşamasında kalite planlama (A), gereken çaba açısından kalite mühendisliği sürecinin baskın yönünü oluşturmaktadır. Test aktivitelerinin yoğunlaştığı noktada, kalite güvence aktivitelerinin yürütüldüğü bölümün (B) ve ürün teslimine yakın zamanda, Kalite Analizinin (C) en fazla çabayı tükettiği ifade edilebilir.



Şekil 1.2 Yazılım kalite mühendisliği süreci (Tian, 2005)



Şekil 1.3 Kalite mühendisliği çaba profili (Tian, 2005)

1.1.2 Yazılım Kararlılığı

Nükleer kontrol sistemleri, uçuş kontrol sistem yazılımları ve balistik füze sistemleri gibi gerçek zamanlı görev kritik sistemler, kararlılık (dependability) özelliğine sahip olmalıdır. Bir sistem; emniyetli (safe), güvenli (secure), güvenilir (reliable) ve erişilebilir (available) ise kararlı özellikte olduğu ifade edilmektedir (Laprie, 1992).

Yazılım geliştirme sürecinde; en iyi yöntemler, uygulamalar ve araçlar kullanıldığı durumda bile geliştirilen yazılımın hatasız olduğunu varsaymak oldukça risklidir ve bazı durumlarda insan yaralanmalarına, kitlesel ölümlere, işletme fırsatlarının kaybına yol açabilir. Yazılımdan kaynaklı tarihte birçok problem raporlanmıştır (Pullum, 2001).

- Endeavor (STS-49) uzay mekiği yazılımında, sıfıra yakın değerler sıfıra yuvarlandığı için Intelsat 6 ile iletişim sorunu yaşanmıştır.
- Körfez Savaşı sırasında; Scud füzelerini tespit edip yok etmede kullanılan Patriot sistemindeki yazılım hatası nedeniyle Dhahran'da Amerikan kışlası isabet almış, 29 Amerikalı asker ölmüş ve 97 asker yaralanmıştır. Bu yazılım, ilk olarak uçaklar için üretildiğinden ve uçağın uçuş süresi yerdeki kullanıma göre daha kısa olduğundan, hata önceden ortaya çıkmamıştır. Belirli bir süre sonra, yazılım yeniden başlatılsaydı Patriot sistemi Scud füzesinin rotasını kestirerek etkisiz hale getirebilecekti.
- Bazı Airbus A320 uçaklarındaki problemler, ilk başlarda pilotların anormal durumu yönetememesine bağlanmışken daha sonraları yazılımdaki hatalar gündeme gelmiştir.
- Therac–25 radyasyon tedavi makinesi, 1985-1987 yılları arasında, yazılımındaki hata nedeniyle hastalara aşırı radyasyon vererek, beş hastanın ölümüne neden olmuştur.
- 1996 yılında 500 milyon dolara mal olan Ariane 5 roketi, ilk kalkışında aritmetik taşma sebebiyle parçalanmıştır. Ariane 5, başarıyla fırlatılan Ariane 4'ün kaynak kodlarını kullanmasına rağmen, atış hızının daha yüksek değere sahip olması nedeniyle parçalanmıştır.

Yazılım sistemleri, buradaki örneklerde de görüldüğü gibi mantıksal kusurlar nedeniyle bazı durumlarda başarısız olmaktadırlar. Kusurlara karşı korunmak için, donanım kusurlarında olduğu gibi, fazlalık (redundancy) eklemek yazılım kusurlarını çözmez, sadece kusurun kopyalanmasını sağlar (Pullum, 2001).

IEEE, yazılım mühendisliği için hata (error), kusur (fault) ve arıza (failure) terimlerini

aşağıdaki şekilde tanımlamaktadır.

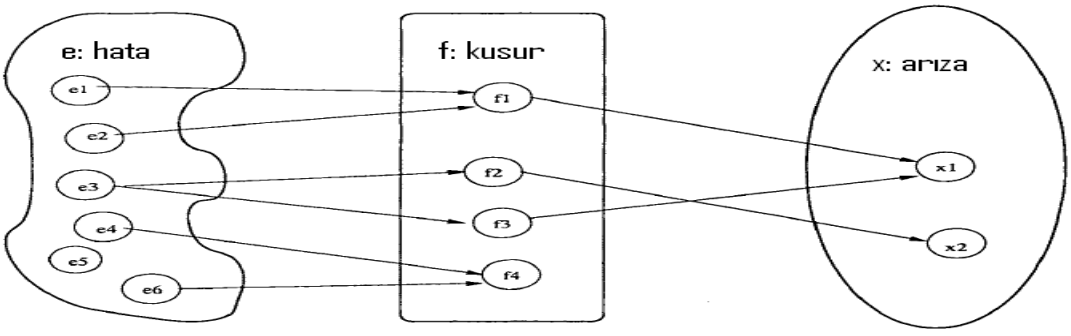
- *Hata:* "Hata, ürüne belirli kusurlar katan doğru olmayan ya da eksik insan eylemidir". Dolayısıyla, programcının kodu gerçekleme aşamasında, gözünden kaçırdığı noktaları hata olarak örneklemek mümkündür.

- *Arıza:* "Arıza, program davranışı kullanıcı beklentilerinden saptığı durumda ortaya çıkar." Yazılım isterler belirtimi (software requirement specification) belgesinde belirtilen isterlerden sapıldığı durumda, arızanın ortaya çıktığı söylenebilir.

- *Kusur:* "Kusur, bir yazılım programı içinde arızaya götüren temel nedendir." Programcı hatası nedeniyle, program girmemesi gereken bir duruma (state) girebilir ve bu durum yazılımın arıza ile sonlanmasına neden olabilir. Ancak her durumda arıza oluşmak durumunda değildir.

Eksiklik (defect) ise, yukarıda açıklanan üç terimin yerine de kullanılabilen ve daha genel bir anlam ifade eden bir terimdir.

"Hatalar, kusurlara sebebiyet verip yazılımın içine kusurların katılmasına neden olabilir ve yazılım yürütüldüğü zaman kusurlar arızalara neden olabilir. Bu ilişki 1'e 1 olmak zorunda değildir" (Tian, 2005). Şekil 1.4' deki resmin sol tarafında bir yazılım geliştiricinin yaptığı hatalar, ortasında yazılım sistemindeki kusurlar ve en sağda kullanım senaryolarına göre isterlerden sapmaların olduğu arızalar temsili olarak gösterilmektedir. Bu örneği inceleyecek olursak; e3 hatası f2 ve f3 kusurlarının oluşmasına neden olmuş; f1 kusuru e1 ve e2 hatalarının bir araya gelmesi nedeniyle doğmuş ve f4 arızaya neden olmamıştır (Tian, 2005).



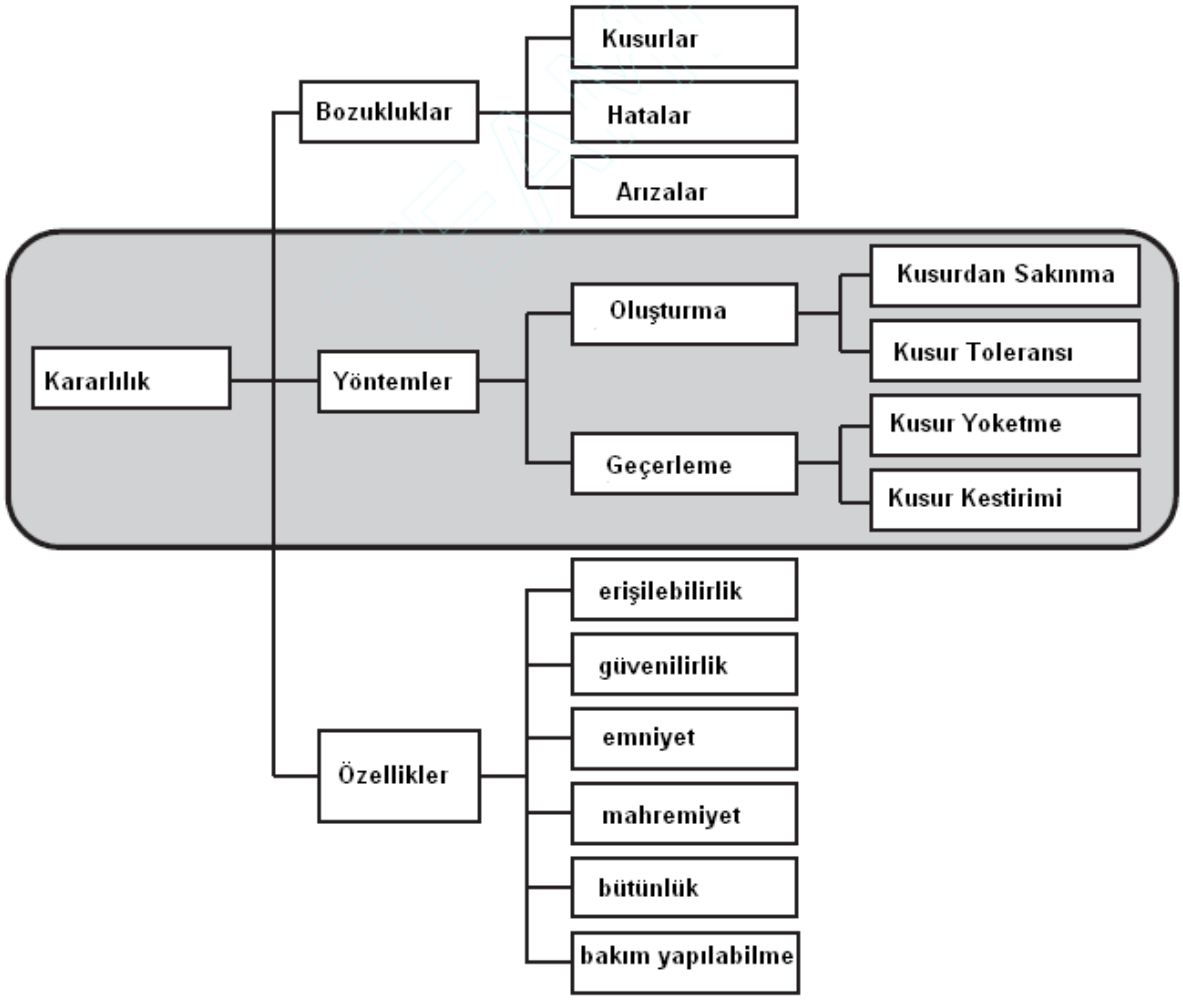
Şekil 1.4 Eksiklik ile ilişkili terimler ve ilişkiler (Tian, 2005)

Yazılımların kararlılık özelliği, birçok kavram ile ilişkilidir ve bu özelliği taşıyan sistemleri geliştirmek için bu kavramları ve yöntemleri biliyor olmak gerekir. Şekil 1.5' de verilen

ağaçta ilişkili kavramlar ve yöntemler sunulmaktadır.

Bozukluklar düğümü altında verilen; kusur, hata ve arıza terimleri önceden açıklandığı için bu kavramlara tekrar değinilmeyecektir. Özellikler altında verilen; erişilebilirlik (availability), güvenilirlik (reliability), emniyet (safety), mahremiyet (confidentiality), bütünlük (integrity), bakım yapılabilirlik (maintainability) gibi kalite özellikleri ise kararlı sistemlerin karakteristiklerini ortaya koymaktadır. Kararlı sistemlere ulaşmak için kullanılan yöntemleri; kusurdan sakınma (fault avoidance), kusur yok etme (fault removal), kusur toleransı (fault tolerance) ve kusur kestirimi (fault forecasting) olarak sıralayabiliriz. Bu yöntemler iki ana grupta değerlendirilebilir (Pullum, 2001):

- Yazılım oluşturma sürecinde uygulananlar (kusur toleransı ve kusurdan sakınma)
- Yazılım geliştirildikten sonra yazılımın geçerlenmesinde kullanılanlar (kusur ortadan kaldırma ve kusur kestirimi)



Şekil 1.5 Kararlılık kavramlarının sınıflandırılması (Pullum, 2001)

Bu tez kapsamında yürütülen çalışmalar; "Kusur Kestirimi" yöntemi altında açıklanacak olan "Güvenilirlik Kestirimi" (Reliability Prediction) çalışmaları olarak da değerlendirilebilir. Farklı terimler kullanılmış olsa da, güvenilirlik kestirimi ve kusur kestirimi çalışmaları aynı amaca hizmet etmektedir. Dolayısıyla, bu tez çalışmasının kararlı sistemler oluşturmak için kullanılabilecek bir yöntem üzerine olduğu sonucunu çıkarmak mümkündür. Kararlı sistemler oluşturmak için kullanılabilecek yöntemler aşağıda açıklanmaktadır (Pullum, 2001):

- *Kusurdan Sakınma:* "Yazılımın oluşturulması sırasında uygulanan kararlılık iyileştirme tekniklerindendir" (Pullum, 2001). Kodların gözden geçirilmesi (code inspection), biçimsel yöntemlerin kullanılması ve yazılımın yeniden kullanılması (reusability) kusurdan sakınma için kullanılabilecek tekniklerdendir.

- *Kusur Yok etme:* Yazılım doğrulama ve geçerleme (D&G) aktiviteleri süresince, kusur yok etme teknikleri uygulanmaktadır. Bu tekniklerle, kusurların tespit edilerek silinmesi sağlanır. Birim testi, tümleştirme testi ve biçimsel inceleme (formal inspection) kusur yok etme için kullanılabilecek tekniklerdendir.

- *Kusur Kestirimi:* Yazılım doğrulanması sırasında, kararlılığı iyileştirmek amacıyla kullanılabilen bir tekniktir. İki tip aktiviteden oluşmaktadır:

    - *Güvenilirlik ölçümü:* "Mevcut yazılım güvenilirliğini, sistemin çalışması veya sistem testi aşamasında elde edilen arıza bilgilerinden istatistiksel çıkarım teknikleriyle belirlemektir" (Pullum, 2001). Geleceğe yönelik bir kestirim yapmak yerine, mevcut bilgiler toplanarak güvenilirlik değerlendirilmesi yapılabilmektedir.

    - *Güvenilirlik kestirimi:* Yazılım metrikleri ve arıza bilgileri kullanılarak, bir sonraki yazılım sürümü için kusur eğilimli modüller belirlenmektedir.

- *Kusur Toleransı:* Yazılımın geliştirilmesi sırasında uygulanır ve geliştirme sonrasında yazılım içinde kalan kusurlara rağmen yazılımın çalışmasını kesintisiz sürdürebilmesini sağlamaktadır. N-Sürüm Programlama Şeması (N-version Programming Scheme-NVPS) ve Kurtarma Blok Şeması (Recovery Block Scheme-RBS) tasarım yaklaşımları, kusur dayanıklılığı için kullanılabilmektedir. NVPS'de, birden fazla algoritma aynı problem için geliştirilerek çoğunluk oylamasına (majority voting) göre karar verilmesi sağlanmaktadır. RBS'de ise, kabul kriteri sağlanana kadar farklı algoritmalar sırasıyla çalıştırılmaktadır.

Yakın gelecekte, gerçek zamanlı sistemlerin çalışma zamanı isterlerine (run-time requirements) göre kendisini uyarlaması ve bu sayede dinamik olarak oluşturulacak kodun gerçek zamanlı bir değerlendirme tekniği (real-time assessment technique) ile beklentileri karşılayıp karşılamadığının değerlendirilmesi gerekecektir. Kararlılığın düşük olduğu durumda, operatörler çalışma zamanı konfigürasyonunu değiştirecek ve yeniden kararlılık değerlendirilecektir. Bu amaçlarla, bir sonraki bölümde açıklanacak olan kusur kestirim modelleri kullanılarak kararlılığın değerlendirilmesi mümkün olabilecektir (Challagulla vd., 2005). Kusur kestirim modellerini ve yöntemlerini, yukarıda açıklandığı gibi, gelecekte gerçek zamanlı bir değerlendirme tekniği olarak görmek mümkün olabilir.

1.1.3 Yazılım Kusur Kestirimi

Yazılım sistemlerinde kusurların çok büyük bölümü, az sayıda modül üzerinde gerçekleşir (Fenton ve Ohlsson, 2000; Kaaniche ve Kanoun, 1996; Ohlsson ve Alberg, 1996) ve kusur taşıyan modüllerin yüzdesinin %10 - %20 arasında değiştiği deneysel çalışmalarda tespit edilmiştir. Bu deneysel bilgiden hareketle, kusur eğilimli modüllerin test aşamasından önce belirlenip test kaynaklarının bu modüllere odaklandırılması, ürünün sahada çıkacak kusurlarını en düşük seviyeye indirecektir sonucuna varılabilir. Yazılım kusur kestirim çalışmalarının çıkış noktasını ve temel amacını bu şekilde ifade etmek mümkündür.

Yüksek güvenceli ve kararlı sistemler oluşturmak için; kod gözden geçirme, kapsamlı yazılım testleri gerçekleştirme, otomatik test senaryosu oluşturma, kritik personelin projeye atanması ve düşük kalitedeki bileşenlerin yeniden mühendisliği (reengineering) gibi farklı kalite güvence aktiviteleri kullanılmaktadır (Khoshgoftaar vd., 2006). Çok büyük ölçekli yazılım sistemlerinde, bu aktiviteleri tüm modüller üzerinde gerçekleştirmek önemli mali ve zaman kayıplarına neden olmaktadır. Bu nedenle, yazılım kusur kestirim modelleri ile kusur eğilimli modüllerin sistem testleri öncesinde belirlenmesi ve sadece yüksek risk taşıdığı belirlenen modüller üzerinde kalite güvence aktivitelerinin uygulanması daha maliyet-etkin (cost-effective) bir yaklaşımdır.

1993 yılında Windows NT 3.1 sürümü 6 milyon kod satırından oluşurken, 2005 yılında Windows Vista Beta 2 sürümünün 50 milyon kod satırından oluşması; yazılım sistemlerindeki kod büyüklüğü açısından ulaşılan karmaşıklığı ortaya koymaktadır. Ayrıca; Mac OS X 10.4 işletim sistemi 86 milyon, Debian 3.1 Linux dağıtımı 215 milyon ve Eclipse Europa sürümüne ilişkin projeler 17 milyon kod satırından oluşmaktadır. Mac OS X, Debian ve Windows için verilen değerlerin sadece işletim sistemi ile ilişkili olduğu ve bu işletim

sistemleri üzerinde çalışan yazılım sistemlerinin ve uygulamaların da mevcut olacağı düşünülürse, kusur kestirim çalışmalarının sağlayacağı yarar açıkça görülecektir.

Yazılım Mühendisliği Enstitüsü, ASA ALT (The Assistant Secretary of the U.S. Army for Acquisition, Logistics, & Technology) tarafından finanse edilerek 12 aylık çalışmanın ardından Haziran 2006'da geleceğin sistemleri olarak "ultra-geniş-ölçekli sistemler" konusunda bir rapor yayımlamıştır. ASA ALT, geleceğin yazılım sistemleri için milyarlarca kod satırının söz konusu olacağını öngörmüş ve Yazılım Mühendisliği Enstitüsü'nden bu sistemleri tasarlama, kontrol etme, izleme ve değerlendirme noktasında uygulanabilecek prensiplerin belirlenmesini istemiştir. Hazırlanan raporda 7 araştırma alanı önerilmiş ve bu alanlardan birisi de "Şartlara Uyum Sağlayan ve Kestirilebilir Sistem Kalitesi" (Adaptable and Predictable System Quality) araştırma alanıdır (Northrop vd., 2006).

Kusur kestirim çalışmaları, düşük kaliteli modüllerin belirlenmesini sağladığı için bazı yayınlarda "yazılım kalite kestirimi" ismiyle kullanılmaktadır. Kusur kestirim çalışmaları kalitenin güvenilirlik özelliği ile doğrudan ilişkili olup, kalitenin kestirilebilmesi için esasında birçok özelliğin değerlendirilmesi gerekmektedir. Ancak mevcut durumuyla yazılım kusur kestirim modelleri, mevcut geniş ölçekli sistemlerde kullanılabilir ve geleceğin ultra-geniş-ölçekli yazılım sistemlerinde sistem kalite kestirimi için önemli teknolojilerdendir.

Yazılım kusur kestirim çalışmalarınının yararları aşağıdaki başlıklarda özetlenebilir:

- Test sürecinin iyileştirilip sistem kalitesinin arttırılması,
- Bakım aşamasında yeniden düzenleme (refactoring) gerektiren modüllerin saptanması,
- Sınıf seviyesindeki metriklerle tasarım aşamasında kestirimin gerçekleştirilmesi durumunda, tasarım alternatiflerinden en iyisinin seçilmesi,
- Kararlı ve yüksek güvenceli bir yazılım sistemine ulaşılması.

2002 yılındaki IEEE Metrics konferansının bir panelinde, yazılım gözden geçirme (software review) ile kusurların yaklaşık olarak %60'ının bulunabileceği sonucuna varılmıştır. Bu panelde; ne Fagan'ın iddia ettiği gibi (1986) incelemeler (inspection) ile test öncesi kusurların %95'inin bulunabileceği fikri desteklenmiş ne de Shull ve arkadaşlarının (2000) iddia ettiği gibi özelleşmiş incelemelerin diğer yöntemlerden %35 daha fazla kusur yakalayabileceği fikri desteklenmiştir (Menzies vd., 2007a).

Menzies ve arkadaşlarının yaptığı çalışmada (2007a), kusurların tespit yüzdesi (probability of

detection) %71 olarak elde edilmiştir. Bu değer, yazılım gözden geçirmelerinin kusurları tespit etme yüzdesi olan %60'dan daha yüksektir. Bu bilgilere göre, doğru kusur kestirim modellerinin yazılım gözden geçirmelerinden daha başarılı olabileceği ifade edilebilir.

Ayrıca, gözden geçirmelerde dakikada 8 ile 20 arasında kod satır sayısı incelenebildiği ve bu işlemin ekip üyesi sayısı 4-6 arasında değişen gözden geçirme ekibi tarafından tekrarlandığı dikkate alınırsa harcanan emeğin boyutu daha net anlaşılacaktır (Menzies vd., 2007a). Yazılım kusur kestiriminin bir araç yardımıyla uzman tarafından gerçekleştirildiği durumda ise, harcanacak zaman ve emek gözden geçirmelerdekine göre oldukça az olacaktır.

Yazılım kusur kestirim modellerinde çoğu zaman, n adet bağımsız değişken ve 1 adet bağımlı değişken kullanılmaktadır. Bağımsız değişkenler yazılım metrikleri olup, bağımlı değişken modüllerin kusurlu olup olmadığını göstermektedir. Örnek bir veri kümesi için 21 bağımsız ve 1 bağımlı değişkenden oluşan 3 adet modüle ilişkin veriler aşağıda sunulmaktadır:

11,3,1,1,49,215.22,0.07,14.67,14.67,3156.61,0.07,175.37,0,0,1,0,12,9,27,22,5, kusursuz

4,1,1,1,13,39,0.33,3,13,117,0.01,6.5,0,0,0,0,4,4,7,6,1, kusursuz

3,1,1,1,7,19.65,0.4,2.5,7.86,49.13,0.01,2.73,0,0,0,0,5,2,5,2,1, kusurlu

Kusur kestirim çalışmalarında çoğu zaman, bir önceki yazılım sürümünün veya benzer bir projenin, yazılım metrikleri ve kusur verileri toplanarak bir veri kümesi oluşturulur. Kusur verileri modüllerin, sistem testlerinde veya sahada kusura yol açıp açmadığını gösteren 1 veya 0 ile temsil edilebilecek verilerdir. Yazılım metrikleri ise metot, sınıf, bileşen seviyesinde olabilmektedir. Makine öğrenmesi algoritmaları ile metrik ve kusur verilerinden öğrenme süreci ilk aşamada gerçekleştirilerek, yeni metriklerin oluştuğu durumda modüllerin kusurlu olup olmadığı kestirilebilmektedir.

Metot seviyesindeki metriklere örnek olarak, yazılım mühendisliği içerisinde en fazla bilinen "çevrimsel karmaşıklık" (cyclomatic complexity) metriği verilebilir. "Kalıtım ağacının boyutu" (depth of inheritance tree), "çocuklarının sayısı" (number of children), "sınıf içerisindeki metot sayısı" gibi metrikler ise sınıf seviyesindeki metriklerdendir. Kusur eğilimli metotları belirlemek isteyen bir uzman, metot seviyesindeki metriklerle çalışması gerekirken kusur eğilimli sınıfları belirlemek isteyen uzman, sınıf seviyesindeki metriklerle çalışmalıdır. Programlama paradigmasının değiştirilmesi ve buna bağlı olarak soyutlama (abstraction) düzeyinin arttırılması, beraberinde farklı metriklerin gündeme gelmesini sağlamıştır. Ancak yapısal ve nesneye yönelik programlama paradigmasının ikisinde de metotlar mevcut olduğu

için, iki paradigmanın herhangi biriyle geliştirilen yazılımlarda metot seviyesindeki metrikler kullanılabilir. İlgiye yönelik (aspect-oriented) programlama yaklaşımının, her kavramı ilgi olarak değerlendirdiği simetrik durumlar için özel ilgi metriklerinin belirlenmesi ve bu kapsamda modellerin kurulması gelecekte gerçekleştirilebilecek çalışmalar olarak değerlendirilebilir. Literatürde ilgiye yönelik metriklerle, kusur eğilimli ilgilerin kestirildiği bir çalışmaya rastlanmamıştır. Kusur kestirim çalışmalarını dört başlık altında ele almak mümkündür:

- *İstatistiksel Yöntemler:* Tek ve çok değişkenli lineer regresyon, lojistik regresyon gibi farklı istatistiksel yöntemlerle yazılım kusur kestirimi gerçekleştirilebilmektedir. Ancak istatistiksel modeller, kara kutu (black-box) şeklinde çalıştığı ve veriye çok bağımlı olduğu ifade edilerek eleştirilmektedir (De Almeida ve Matwin, 1999).

- *Makine Öğrenmesi Tabanlı Yaklaşımlar:* Karar Ağaçları, Genetik Programlama, Bulanık Mantık, Yapay Sinir Ağları gibi farklı makine öğrenmesi teknikleri ile yazılım kusur kestirimi gerçekleştirilmiştir (Evett vd., 1998). Bölüm 2' de yazılım kusur kestirimi konusunda yapılan çalışmalar açıklanırken, bu yöntemlerin referansları ve detayları sunulacaktır.

- *Uzman Görüşü:* Yazılım maliyet kestirimi probleminde olduğu gibi kusur kestiriminde de uzman görüşü kullanılabilmektedir. Ancak binlerce modülün olduğu geniş ölçekli sistemlerde kusurlu modüllerin bir uzman tarafından kestirilmesi iyi sonuçlar üretmemektedir. İstatistiksel modellerin uzman görüşü ile kıyaslandığı çalışmada, istatiksel modellerin çok daha iyi sonuçlar verdiği raporlanmıştır (Tomaszewski vd., 2007).

- *Birleştirilmiş Modeller (Mixed):* Bazı modellerde, istatistiksel yöntemler ve yapay zekâ tabanlı yaklaşımlar birlikte kullanılmaktadır. Örneğin; özellik azaltma için "temel bileşen analizi" ya da "korelasyon tabanlı özellik azaltma" gibi istatistiksel yöntemlerden yararlanıp ardından belirlenen özellik kümesini yapay zekâ tabanlı yaklaşımda kullanmak çok uygulanan bir yaklaşımdır.

Kusur kestirim çalışmalarını, "kalite kestirim modelleri" ve "sınıflandırma modelleri" isimli iki grupta değerlendirmek mümkündür. Kalite kestirim modelleri, sistem testi aşamasında çıkacak olan kusurların sayısını kestirirken, sınıflandırma modellerinde kusur sayısı kestirilmesi yerine kusur eğilimli ya da değil şeklinde sınıflandırma yapmak söz konusudur. Bu tez sırasında gerçekleştirilen çalışmalar, sınıflandırılma kapsamında değerlendirilebilir.

1.1.4 Biyolojiden Esinlenen Hesaplama

İnsanoğlu yüz yıllardır doğayı gözlemleyerek, doğadaki karmaşık olayları anlamaya çalışmış ve bu amaçla teoriler üretmiştir. Ancak doğayı sadece gözlemlemek ondan elde edilebilecek kazanımları sınırlandırır. Doğanın milyonlarca yıldır başarıyla gerçekleştirdiği birçok olaydan esinlenip problemlerde çözüm yöntemi olarak kullanmak mümkündür. Son yıllarda hesaplama ve mühendislik disiplinleri, biyolojik süreçleri anlamak için modelleme ve simülasyondan yararlanmıştır. Benzer şekilde; hesaplama ve mühendislik, problemlere çözüm geliştirmek için biyolojideki fikirlerle zenginleşmiştir. Bu kapsamda, Yapay Sinir Ağları ve Evrimsel Algoritmalar gibi hesaplamalı zekâ (computational intelligence) yöntemleri geliştirilmiştir (De Castro ve Timmis, 2002).

Biyoloji ve hesaplama arasındaki çift yönlü etkileşimler nedeniyle, üç farklı yaklaşımdan söz edilebilir (De Castro ve Timmis, 2002):

1. *Biyolojiden Harekete Geçen Hesaplama (biologically motivated computing)*: Yapay sinir ağları, genetik algoritmalar, yapay bağışıklık sistemleri bu gruba girmektedir.

2. *Hesaplamadan Harekete Geçen Biyoloji (computationally motivated biology)*: Hesaplama bilimi, biyoloji için farklı modeller sunabilmektedir. Örneğin; hücresel otomatlar (cellular automata) biyolojide popülasyon büyüklüklerini modellemek için kullanılmaktadır.

3. *Biyolojik Mekanizmalarla Hesaplama (computing with biological mechanism):* Bu yaklaşımlar, biyolojik sistemlerin bilgi işlem yeteneklerinden yararlanmayı hedefler. Silikon tabanlı bilgisayarlar yerine malzeme olarak DNA (Deoksiribo Nükleik Asit) moleküllerinin kullanıldığı DNA bilgisayarlar, bu gruba örnek olarak verilebilir.

Biyolojiden harekete geçen ya da daha farklı bir ifadeyle biyolojiden esinlenen hesaplama (biologically inspired computing), şimdiye kadar Yapay Sinir Ağları, Evrimsel Algoritmalar, Genetik Programlama, Karınca Optimizasyonu, Sürü Zekâsı (Swarm Intelligence) gibi birçok yöntemin geliştirilmesini sağlamıştır. Bu noktada, model ve metafor kavramlarının farkını vurgulamak yararlı olacaktır.

Model; bir kavramı başka kavramlarla yeniden oluşturmak veya gösterimini gerçekleştirmek olarak tanımlanabilir. Örneğin; vücuda giren zararlı maddelerin, bağışıklık hücrelerini nasıl

etkilediğini gösterecek bir hareket denkleminin belirlenmesi bir model oluşturulması çalışmasıdır. Metafor ise bir bileşenin soyut ve üst seviye gösterimini oluşturmayı hedeflemektedir (De Castro ve Timmis, 2002).

Paton isimli araştırmacı (1992), metaforların geliştirilmesi için biyolojik sistemlerin dört özelliğini belirlemiştir: mimari, fonksiyonellik, mekanizma ve organizasyon. Mimari; sistemin yapısına işaret ederken, fonksiyonellik sistem davranışı ile ilgilidir. Mekanizma; parçaların birlikte çalışmasına işaret ederken, organizasyon tüm dinamik içinde sistem akvititelerinin gerçeklenme şekli ile ilgilidir (De Castro ve Timmis, 2002).

Örneğin; geçmişte bilim dünyasında "çelik yeleklerin" tasarlanması için "örümcek ağı" metafor olarak kullanılmıştır. Yeryüzündeki en sağlam malzemelerden birisinin örümcek ağı olduğu ve aynı kalınlıktaki telden beş kat daha sağlam olduğu dikkate alınırsa çelik yelek üretiminde esin kaynağı olarak kullanılma nedeni rahatlıkla anlaşılabilir. Kimyevi madde üreticisi DuPont firması, örümcek ağının moleküler yapısını belirleyerek "kevlar" isimli en sağlam malzemeyi bu sayede üretebilmiştir. Bu malzeme; Mars'a gönderilen Pathfinder'daki hava yastıklarında kullanılmıştır. Bu örnekte görüldüğü gibi, insanoğlu sürekli olarak doğadaki farklı yapılardan esinlenerek yeni teknolojiler ve yöntemler üretebilmektedir. Yapay Bağışıklık Sistemleri de, biyolojiden esinlenen yeni bir hesaplamalı zekâ yaklaşımı olup metafor olarak doğal bağışıklık sistemlerini kullanmaktadır.

1.1.5 Bağışıklık Sistemleri

Bu bölümde bağışıklık sistemleri, doğal ve yapay bağışıklık sistemleri olmak üzere iki bölümde ele alınacaktır.

1.1.5.1 Doğal Bağışıklık Sistemleri

Bağışıklık sisteminin temel sorumluluğu, vücuda dışarıdan giren patojenleri (zararlı canlılar) teşhis edip vücudu korumak için yanıt vermektir. Bağışıklık sistemleri (natural immune system) birbiri ile etkileşen iki mekanizmaya sahiptir.

İlki "doğal bağışıklık sistemidir" (innate immune system) ve doğuştan sahip olunan koruma mekanizmasıdır. İkinci mekanizma ise "adaptif bağışıklık sistemi" (adaptive immune system) olarak adlandırılır. Bu sistem sayesinde canlı, kendisine zararlı olabilecek patojene karşı ilk etkileşimin ardından tanıma yeteneğini arttırabilmektedir. Bu sayede, bir sonraki etkileşimde patojen bağışıklık sistemi tarafından tanınabilmekte ve bellek (memory) hücreleri sayesinde hızlıca yanıt verilebilmektedir (De Castro ve Timmis, 2002).

Lenfositler B hücreleri ve T hücreleri olmak üzere iki gruba ayrılır. Her lenfosit belirli bir istilacıyı tanıyabilmektedir ve bu istilacılar "antijen" olarak adlandırılmaktadır. B hücreleri antikor salgılayan, T hücreleri ise antijenleri öldüren hücrelerdir (Kim, 2002).

B hücresi ve antijen arasındaki "benzerlik derecesi" (similarity degree), "afinite" olarak adlandırılmaktadır. Eğer antijen B hücresi tarafından teşhis edilirse, B hücresi "klonal büyüme" (clonal expansion) isimli süreçle kopyalanır. Bu klonlar, afinite değerine göre "somatik hipermutasyon" işlemine tabi tutularak farklılaşma sağlanması için mutasyona uğratılır. Bu klonların bazıları B bellek hücrelerine dönüştürülür. Bu sayede, sonraki saldırılarda hızlıca yanıt verme imkanı ortaya çıkmaktadır. Bağışıklık sisteminin bu özelliğine "bellekli olma özelliği" adı verilir. Antijenle en iyi uyuşan klonlar hayatta kalır ve bu süreç "klonal seçim" (clonal selection) olarak adlandırılır. Bağışıklık sisteminin diğer temel özellikleri; dağıtık kontrol, deneyimle öğrenme, paralel işleme ve adaptasyon olarak sıralanabilir.

1.1.5.2 Yapay Bağışıklık Sistemleri

Bir önceki bölümde ele alınan doğal bağışıklık sistemlerindeki kavramları, karmaşık problemlerin çözümünde kullanmak mümkündür. Evrimsel hesaplama (evolutionary computation) açısından bağışıklık sistemlerindeki kavramları ele alırsak, bir problemin çözümünü sağlamak üzere basit olarak aşağıdaki şekilde kavramlar ve model elemanları arasındaki ilişkiyi kurabiliriz:

Problem → Antijen (dışarıdan vücuda giren patojenin bir parçası)

Aday çözüm → Antikor (antijeni tanıyacak olan vücuda özel bir molekül)

Aday çözümler kümesi → Repertuvar (antikor kümesi)

Çözümün probleme yakınlığı → Afinite (antikor ve antijenin uyuşma derecesi)

Antikorlar; "klonlanma", "hipermutasyon" ve "klonal seçim" aşamalarından geçirilerek uygun çözüme ulaşılmaya çalışılır. Evrimsel hesaplama açısından dönüşümü bu şekilde kolayca gerçekleştirmek ve Yapay Bağışıklık Sistemlerini (Artificial Immune Systems) Evrimsel Hesaplama'nın bir alt dalı olarak görmek mümkün gibi görünse de, Yapay Bağışıklık Sistemleri (YBS) yeni bir hesaplamalı zekâ yaklaşımıdır.

Yapay Sinir Ağları, Genetik Algoritmalar, Sürü Zekâsı, Karınca Optimizasyonu, Evrimsel Hesaplama biyolojiden esinlenen hesaplama paradigmalarındandır. Son yıllarda yapay zekâ

konusunda araştırma yapanlar, doğadaki karmaşık yapıları inceleyip hesaplama modeli olarak kullanmaya başlamışlardır. Yapay Bağışıklık Sistemleri, son dönemde karmaşık problemleri çözmedeki başarısı sebebiyle sıkça gündeme gelmektedir. Bu sistemler; örüntü tanıma (pattern recognition), zaman çizelgeleme (scheduling), bilgisayar güvenliği, uçak kontrolü (aircraft control), veri madenciliği, optimizasyon, robotik gibi konularda başarıyla uygulanmıştır (Watkins, 2001).

YBS'ler, bağışıklık sisteminden esinlenen makine öğrenmesi algoritmalarıdır. 15 yıllık geçmişi olmasına rağmen şimdiye kadar ilgilenilen alanlar kümeleme ve anomali tespiti üzerineydi. Son dönemde, optimizasyon ve sınıflandırma üzerine çok başarılı algoritmalar geliştirilmiştir. Çok farklı bağışıklık mekanizmaları modellenmiş ve uygulamalarda kullanılmıştır. Bu mekanizmalar; "bağışıklık ağ teorisi" (immune network theory), "klonal seçim" (clonal selection), "negatif seçim" (negative selection) olarak sıralanabilir.

YBS konusundaki çalışmalar 2001 yılına kadar eğiticisiz öğrenmeye odaklanmıştır. Watkins (2001) geliştirdiği bir algoritmayla sınıflandırma için, YBS'lerin oldukça başarılı sonuçlar üretebileceğini göstermiştir.

Watkins geliştirdiği "Yapay Bağışıklık Tanıma Sistemi" (Artificial Immune Recognition System-AIRS) algoritmasında; Timmis'in kaynak sınırlı (resource limited) yaklaşımını (Timmis, 2000a) ve De Castro'nun klonal seçim prensibini (De Castro ve Von Zuben, 2000a) kullanmıştır. Bu algoritmanın ve diğer YBS tabanlı algoritmaların detayları, takip eden bölümlerde açıklanacaktır.

Bu tez çalışması, yazılım kusur kestirimi probleminde YBS paradigmasından yararlanmayı hedeflediği için çok farklı YBS tabanlı algoritmalar üzerinde çalışılmış ve kusur kestirimi için yeni algoritma ve teknikler önerilmiştir.

1.2 Hipotez

Bu doktora tez çalışmasının temel hipotezini, aşağıdaki şekilde özetlemek mümkündür:

Yazılım kusur kestirimi problemi için, son yıllarda karmaşık problemlerin çözümünde tercih edilen ve doğadan esinlenen Yapay Bağışıklık Sistem paradigmasının uygulanması mümkündür ve etkin sonuçlar üretir.

Bu hipotezin geçerliliğini sağlamak üzere, farklı alt araştırma alanları belirlenerek tez çalışması süresince bu alanlarda farklı araştırmalar yürütülmüştür. Bir sonraki bölümde, tez

amaçları başlığı altında, bu alt alanlar açıklanarak temel hipotezin geçerliliğini sağlamak üzere sağladığı yararlar ortaya konulacaktır.

1.3 Amaçlar

Bu tez çalışmasının yedi amacı bulunmaktadır:

1. *Eğiticili sınıflandırma (supervised classification) kapsamında kullanılabilecek literatürde tanımlanmış Yapay Bağışıklık Sistem tabanlı algoritmaları belirleyerek yazılım kusur kestirimi problemi için açık veri kümeleri üzerinde performans analizlerini gerçekleştirmek.*

   Literatürde tanımlanmış algoritmaların problem üzerinde uygulanması, atılacak ileriki adımları saptamak için kritik öneme sahiptir. Performans açısından üstünlüğü saptanan algoritmalara odaklanılması, mevcut algoritmalar açısından problem uzayının daraltılması anlamına gelmektedir.

2. *Farklı özellik azaltma (feature selection) tekniklerinin YBS tabanlı kusur kestirim modellerinde performans etkisini incelemek.*

   Veri kümelerinde yer alan bağımsız değişkenler veya diğer bir ifadeyle metrikler arasında korelasyon bulunabilmekte ve bu korelasyon nedeniyle, sınıflayıcıların performansları olumsuz yönde etkilenmektedir. Korelasyon tabanlı özellik azaltma veya temel bileşen analizi gibi özellik azaltma teknikleri ile bu metrik alt kümeleri belirlenerek sınıflayıcılarda girdi olarak kullanılabilmektedir. En yüksek performansı sunan özellik azaltma yöntemi, YBS tabanlı sınıflayıcı ile tümleştirilerek gürbüz bir kusur kestirim modeli ortaya çıkmaktadır.

3. *Sınıf seviyesindeki metriklerle, tasarım aşamasında erken kestirimi sağlamak üzere YBS tabanlı modeller tasarlamak.*

   Sınıf seviyesindeki nesneye yönelik metrikleri kullanarak, yazılım kusur kestirimi gerçekleştirilebilirse tasarım aşamasında kusur eğilimli modüller belirlenir ve bu modüllerin yeniden düzenlenmesi yoluna gidilebilir. Kodlama aşamasında elde edilen metot seviyesindeki metriklerle kurgulanan YBS tabanlı sınıflayıcıların performansı, sınıf seviyesindeki metriklerle kurgulanan YBS tabanlı sınıflayıcıların performansı ile kıyaslanarak karşılaştırma gerçekleştirilebilir. Kusur kestirimi için en önemli ve en önemsiz sınıf seviyesindeki metrikler belirlenerek bu metriklerin ileriki çalışmalar için

toplanıp toplanmamasına karar verilebilir.

4. *Sınırlı sayıda kusur verisinin mevcut olduğu durumlar için, YBS tabanlı yarı-eğiticili sınıflandırma algoritması geliştirmek.*

   Bazı durumlarda modüllerin çok az bir kısmının önceki kusur bilgileri mevcut olabilir. Dağıtık geliştirilen yazılımlarda bazı firmalar bu kusur bilgilerini toplamayabilir ya da veri toplama araçlarının çalıştırılma maliyeti bu bilgilerin toplanmasını sınırlandırabilir. Çok sürümlü projelerde, bir sürümde modüllerin bir kısmının yazılım kalite verisi toplanabilir. Bu durumda; sadece az sayıda modülün kusur bilgisi mevcut olacaktır. Az sayıda etiketli modüller olduğu için; eğiticili sınıflandırma yaklaşımı iyi sonuçlar üretmeyecektir. Bunun yerine; etiketsiz verilerden de yararlanarak yarı-eğiticili öğrenme yaklaşımı daha iyi sonuçlar üretebilir.

5. *Yazılım Ürün Hatlarında yazılım kusur kestiriminin uygulanabilmesi için alan ve uygulama mühendisliği alt süreçlerini içerecek şekilde çerçeve bir model önermek.*

   Yazılım ürün hattı mühendisliği, mimari ve test senaryoları gibi yeniden kullanılabilir çekirdek varlıkları (reusable core assets) kullanarak ortak bir platformdan benzer ürünler geliştirmeyi hedefleyen ve gelişmekte olan yeni bir paradigmadır. Mevcut yazılım ürün hattı mühendisliği çerçevelerinde, çok az sayıda kalite güvence aktivitesi uygulanmaktadır ancak tekli sistem mühendisliği (single-system engineering) yaklaşımlarında birçok kalite güvence aktivitesi mevcuttur ve ürün hattı mühendisliğine uyarlanabilir. Yazılım kusur kestirimi çalışmalarında elde edilecek deneyimlerin, yazılım ürün hattı mühendisliğine de uygulanması bu tez çalışmasının amaçlarındandır.

6. *Yazılım Mühendisliği disiplini içerisinde kestirim uygulanabilecek problemleri belirleyerek, yeni bir yazılım yaşam çevrimi önermek ve test senaryolarını önceliklendirmek için kusur kestirim tabanlı yeni bir yöntem sunmak.*

   Bu tez çalışması kapsamında ele alınan kusur kestirim problemi, Yazılım Mühendisliği içerisinde yer alan kestirim problemlerinden sadece birisi olup yazılım kalite kestirimini sağlamak üzere yoğun şekilde üzerinde araştırma yürütülmektedir. Bu problem dışında, yeniden kullanılabilirliğinin kestirimi, büyüklük ve maliyet kestirimi gibi birçok kestirim problemi mevcuttur. Kestirimi merkeze taşıyarak daha kaliteli yazılım geliştirilmesi için, yeni bir yazılım yaşam çevriminin önerilmesi mümkündür. Ayrıca, test senaryolarının önceliklendirmesinde kusur kestirim yaklaşımları kullanılabilir.

1.4 Katkılar

Bu tez çalışması aşağıdaki katkıları sağlamıştır.

1. İlk kez YBS tabanlı sınıflandırma algoritmalarının (Immunos1, Immunos2, Immunos99, CSCA (Clonal Selection Classifier Algorithm), CLONALG, AIRS1 (Artificial Immune Recognition Systems), AIRS2, AIRS2 Paralel) performansları kusur kestirimi problemi için analiz edilmiştir (Bölüm 6.3).

2. YBS paradigması ilk kez eğiticili kusur kestirimi probleminde metot seviyesindeki metriklerle ve özellik azaltma tekniği ile birlikte kullanılmıştır. Korelasyon tabanlı özellik azaltma tekniği ve AIRS algoritmasının birlikte kullanıldığı modelin, büyük veri kümeleri için diğer öğrenme algoritmalarına göre daha yüksek performans sunduğu gözlemlenmiştir (Bölüm 6.1).

3. YBS paradigması ilk kez eğiticili kusur kestirimi probleminde sınıf seviyesindeki metriklerle birlikte kullanılmış ve tasarım aşamasında erken kestirimi sağlayabilecek bir model önerilmiştir. CK metrikleri ve kod satır sayısının birlikte kullanıldığı AIRS tabanlı modelin, en yüksek performansı sunduğu raporlanmıştır. Bu çalışma sayesinde, Kalıtım ağacının boyutuna (Depth of Inheritance Tree-DIT) ilişkin sınıf metriğinin kusur kestiriminde en düşük etkiye sahip olduğu ve nesneler arasındaki bağlılığa (Coupling between Object Classes-CBO) ilişkin metriğin en yüksek öneme sahip olduğu deneysel olarak gösterilmiştir (Bölüm 6.2).

4. YBS paradigması ilk kez yarı-eğiticili bir öğrenme algoritmasında yer almış ve yarı-eğiticili kusur kestirim probleminde kullanılmıştır. Bu kapsamda, YBS paradigması tabanlı yarı-eğiticili bir öğrenme algoritması önerilmiş ve problemde uygulanmıştır. Önerilen algoritmanın sınırlı sayıda kusur verisi kullanıldığı durumda, AIRS algoritmasının performansını arttırdığı tespit edilmiştir. Ancak önerilen yaklaşımın bazı algoritmalar için performansı düşürdüğü de diğer tespit edilen noktalardandır (Bölüm 7).

5. İlk kez, "yazılım ürün hatlarında" kusur kestirimin nasıl gerçekleştirilebileceği kavramsal bir çerçevede ortaya konulmuştur. Alan mühendisliği ve uygulama mühendisliği olmak üzere iki alt süreçte, yazılım kusur kestirimi yaklaşımı ele alınmıştır (Bölüm 8).

6. "Kestirim merkezli yazılım yaşam çevrimi" ve "kestirim merkezli yazılım süreçleri" isimli yeni bir geliştirme yaklaşımı ve yeni bir süreç yaklaşımı önerilmiştir. Ayrıca, kusur kestirim tabanlı test senaryosu önceliklendirme yaklaşımı ortaya konulmuştur (Bölüm 4).

1.5 Yayınlar

Bu tez çalışması sırasında aşağıdaki yayınlar gerçekleştirilmiştir.

Dergi Yayınları

1. Ç. Çatal ve B. Diri, "Unlabeled Extra Data does not Always Mean Extra Performance for Semi-Supervised Fault Prediction", *Expert Systems: Journal of Knowledge Engineering,* Blackwell Publishing, SCI, (Kabul Edildi, Baskı Sürecinde).

2. Ç. Çatal ve B. Diri, "A Systematic Review of Software Fault Prediction Studies", *Expert Systems with Applications,* Elsevier, SCI, (Kabul Edildi, Baskı Sürecinde).

3. Ç. Çatal ve B. Diri, "Investigating the Effect of Dataset Size, Metrics Sets, and Feature Selection Techniques on Software Fault Prediction Problem", *Information Sciences,* Elsevier, (Değerlendirilme Sürecinde).

4. Ç. Çatal ve B. Diri, "Systematic Integration of Fault-Prediction Sub-process into Software Product Line Engineering Framework", *Frontiers of Computer Sciences in China,* Springer ve Higher Education Press, (Özel sayı için gelen davet sonucu gönderildi).

Konferans Bildirileri

5. Ç. Çatal ve B. Diri, "Yazılım Metriklerini Kullanarak Düşük Kaliteli / Yüksek Riskli Modüllerin Otomatik Tespiti", Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu, 9-10 Ekim 2008, İstanbul, Türkiye.

6. Ç. Çatal ve B. Diri, "Cost-Effective Allocation of Testing Resources by Using Fault Prediction Methods", *South East European Software Testing Conference*, 2-3 Temmuz 2008, Sofya, Bulgaristan.

7. Ç. Çatal ve B. Diri, "A Fault Prediction Model with Limited Fault Data to Improve Test Process", *9th International Conference on Product Focused Software Process Improvement (PROFES 2008)*, Lecture Notes in Computer Science 5089, Springer-Verlag, 23-25 Haziran 2008, Roma, İtalya, sf. 244-257.

8. Ç. Çatal ve B. Diri, "A Conceptual Framework to Integrate Fault Prediction Sub-process for Software Product Lines", *2nd IEEE International Symposium on Theoretical Aspects of Software Engineering,* IEEE Computer Society, 17-19 Haziran 2008, Nanjing, Çin, sf. 99-106.

9. Ç. Çatal, “Predictable Software Quality: Complexity and Security Concerns”, Test Management Summit, European Test Center, 7-8 Nisan 2008, Krakow, Polonya, sf. 7-16.

10. Ç. Çatal ve B. Diri, “Yazılım Mühendisliğinde Başarılı Deneyleri Nasıl Gerçekleştirebiliriz?”, *Akademik Bilişim 2008*, 30 Ocak-2 Şubat 2008, Çanakkale 18 Mart Üniversitesi, Çanakkale, Türkiye.

11. Ç. Çatal ve B. Diri, “Yazılım Kalite Sınıflandırma Probleminde Yeni Yaklaşımlar: Yapay Bağışıklık Sistemleri”, *3. Ulusal Yazılım Mühendisliği Sempozyumu ve Sergisi*, Bilkent Üniversitesi, 27-28 Eylül 2007, Ankara, Türkiye, sf. 47-55.

12. Ç. Çatal ve B. Diri, “Software Fault Prediction with Object-Oriented Metrics Based Artificial Immune Recognition System”, *8th International Conference on Product Focused Software Process Improvement (PROFES 2007)*, Lecture Notes in Computer Science 4589, Springer-Verlag, 3-4 Temmuz 2007, Riga, Letonya, sf. 300-314.

13. Ç. Çatal, B. Diri, ve B. Özümüt, “An Artificial Immune System Approach for Fault Prediction in Object-Oriented Software”, *2nd International Conference on Dependability of Computer Systems (DepCos-Relcomex 2007)*, IEEE Computer Society, 14-16 Haziran 2007, Szklarska Poreba, Polonya, sf. 238-245.

14. Ç. Çatal ve B. Diri, "Application and Benchmarking of Artificial Immune Systems to Classify Fault-Prone Modules for Software Development Projects", *IADIS International Conference Applied Computing 200*7, 18-20 Şubat 2007, Salamanca, İspanya, sf. 347-354.

15. Ç. Çatal ve B. Diri, "Software Defect Prediction using Artificial Immune Recognition System", *25th IASTED International Multi-Conference Software Engineering*, 13-15 Şubat 2007, Innsbruck, Avusturya, sf. 285-290.

1.6 Tezin Yapılandırılması

Takip eden bölümlerde, bu tez çalışmasında hedeflenen yedi amacı karşılamak üzere gerçekleştirilen çalışmalar açıklanmaktadır.

Bölüm 2 yazılım kusur kestirim konusunda literatürde yapılmış çalışmaları açıklamaktadır. Bu çalışmalar, 74 uluslararası makale ve bildiriden oluşmaktadır.

Bölüm 3 yapay bağışıklık sistemleri paradigmasını detaylarıyla açıklamaktadır. Literatürde ortaya konulmuş YBS tabanlı algoritmalar tanıtılarak uygulama alanları açıklanmıştır.

Bölüm 4 yeni bir yazılım geliştirme yaklaşımı olarak “kestirim merkezli yazılım yaşam çevrimi” ’ni ve kusur kestirim tabanlı test senaryosu önceliklendirme yöntemini açıklamaktadır.

Bölüm 5 tez çalışması sırasında kullanılan veri kümelerini ve kullanılan metrikleri açıklamaktadır. Metot seviyesinde ve sınıf seviyesindeki metrikler ayrıntılı olarak sunulmaktadır. Ayrıca, performans değerlendirme kriterleri bu bölümde verilmektedir.

Bölüm 6 eğiticili kusur kestirimi amacıyla gerçekleştirilen çalışmaları ortaya koymaktadır. Metot seviyesindeki metriklerin ve özellik azaltma tekniklerinin kullanıldığı çalışmalar, sınıf seviyesindeki metriklerin kullanıldığı çalışmalardan ayrı olarak açıklanmıştır. Ayrıca, farklı algoritmaların eğiticili kusur kestirimi bağlamında karşılaştırmalı olarak incelendiği çalışma bu bölümde verilmektedir.

Bölüm 7 sınırlı sayıda kusur verisiyle yazılım kusur kestirimini gerçekleştirmek üzere önerilen YBS tabanlı yarı-eğiticili öğrenme algoritmasını tanıtmaktadır.

Bölüm 8 yazılım ürün hattı mühendisliğinde kusur kestiriminin gerçekleştirilebilmesi için önerilen çerçeve modeli açıklamaktadır. Bu kestirim yaklaşımının alan ve uygulama mühendisliği alt süreçleri açısından değerlendirilmesi yapılarak diğer alt süreçlerle ilişkisi ayrıntılandırılmıştır.

Bölüm 9 sonuç ve gelecek çalışmaları ortaya koymaktadır. Tezin amaçları ve katkıları tekrar değerlendirilerek gelecekte yapılabilecek çalışmalar bu bölümde sunulmaktadır.

2. YAZILIM KUSUR KESTİRİMİ KONUSUNDAKİ ÇALIŞMALAR

Yazılım kusur kestirimi konusunda yapılan çalışmaları, dört bölümde ele almak mümkündür:

1. *İstatistiksel Yöntemler:* Lojistik regresyon, tek ve çok değişkenli lineer regresyon gibi istatistiksel yöntemlerle kusur kestirimi gerçekleştirilebilmektedir.

2. *Makine Öğrenmesi Tabanlı Yöntemler:* Karar ağaçları, genetik algoritmalar, yapay sinir ağları gibi makine öğrenmesi yöntemleriyle kusur kestirimi yapılabilir.

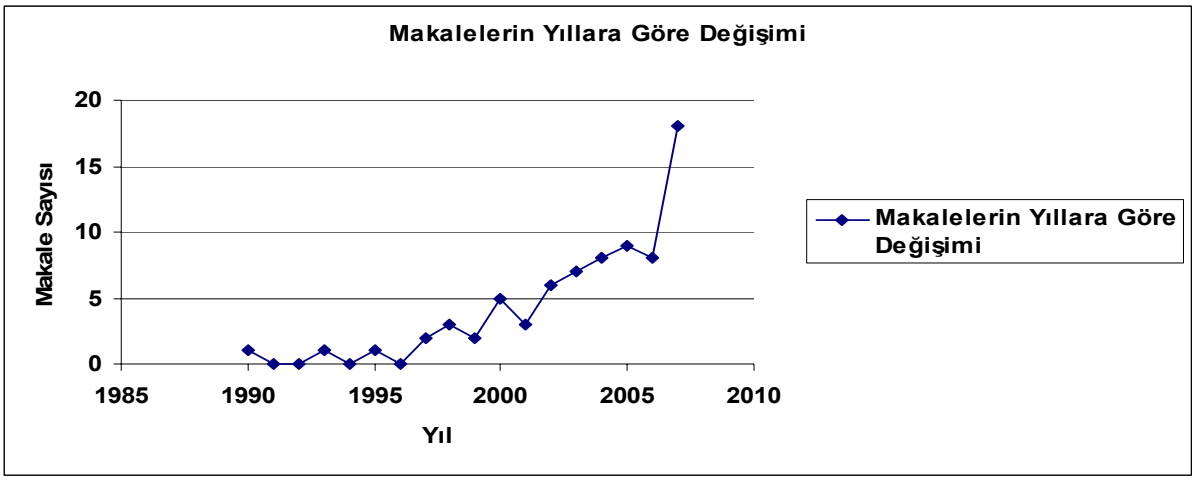
3. *Uzman Görüşü:* Kusur eğilimli modülleri belirlemek için projedeki uzmanın görüşü kullanılabilir ancak binlerce modülün olduğu projeler için etkin çalışmayacağı açıktır.

4. *Birleştirilmiş Modeller:* Genellikle istatistiksel ve makine öğrenmesi tabanlı yaklaşımlar bir arada kullanılabilmekte ve etkin modeller tasarlanmaktadır.

Tez çalışması süresince; farklı makaleler ve bildiriler incelenerek kusur kestiriminin farklı başlıklarında çalışmalar gerçekleştirilmiştir. Tez aşamasının son bölümünde literatürdeki çalışmaların, farklı açılardan değerlendirilmesini yapabilmek için yayınlar yeniden incelenerek 74 yayın tespit edilmiştir. Bu yayınların; yıl, veri kümesi, açık veri kümesi kullanıp kullanmaması, metrik, teknik, değerlendirme kriteri bilgilerine ilişkin notlar alınmıştır. Yayınların konu ile yakından ilgili olup olmamasına göre yayın seçimleri yapılmış olup, güncellik veya farklı kriterler belirleyici unsur olarak tercih edilmemiştir.

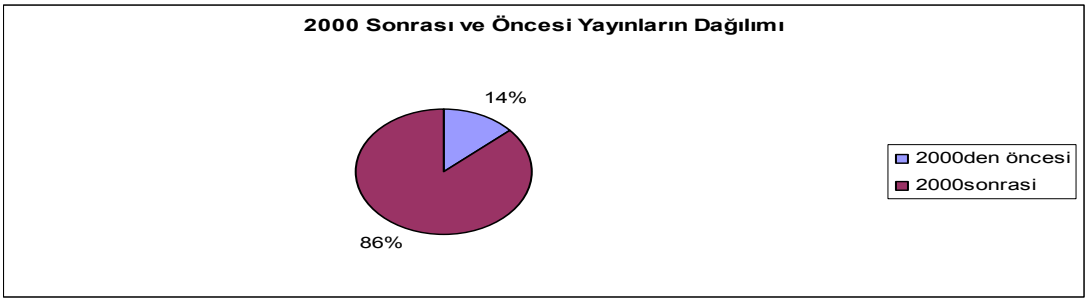
2.1 Yıllara Göre Yayınların Dağılımı

Şekil 2.1' de, 1990-2007 yılları arasındaki yayınların sayılarının değişimi verilmektedir.



Şekil 2.1 İncelenen yayınların yıllara göre değişimi (1990-2007)

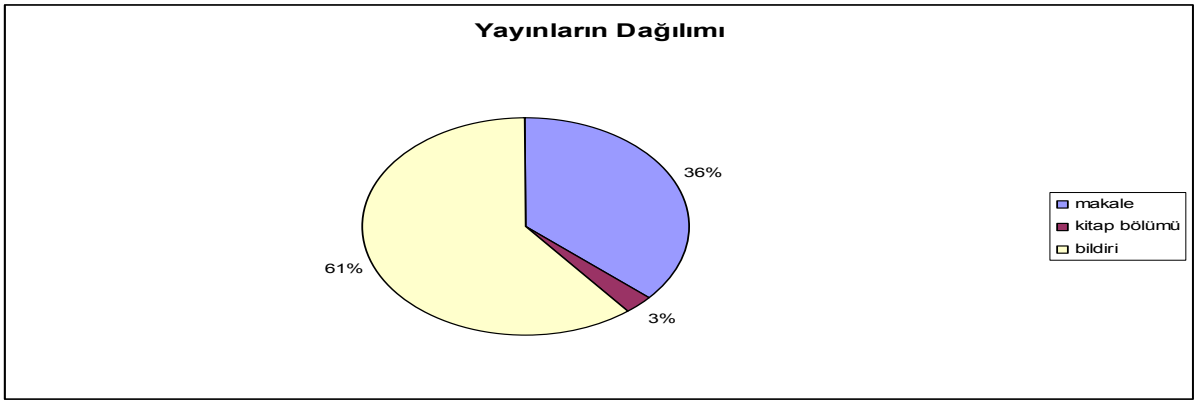
Grafikten görüldüğü gibi, 2000 yılı ve sonrasında bu konuda çıkan yayın sayısı oldukça fazladır. Bu analizde kullanılan yayınların seçimi, yıllar göz önüne alınmadan yapıldığı için geçen yıllarla birlikte konuya ilginin artmış olduğu çıkarımını yapabiliriz. Sayısal olarak ise; popülasyonda 2007 yılından 18, 2006 yılından 8, 2005 yılından 9, 2004 yılından 8, 2003 yılından 7, 2003 yılından 6, 2001 yılından 3, 2000 yılından 5, 1999 yılından 2, 1998 yılından 3, 1997 yılından 2, 1995 yılından 1, 1993 yılından 1, 1990 yılından 1 yayın vardır. Şekil 2.2' de, 2000 yılı ve sonrasında gerçekleştirilmiş olan yayınların grafiksel dağılımı verilmektedir. Bu dağılıma göre, yayınların %86' sı 2000 yılından sonra yapılmıştır. Sayısal olarak, popülasyonda 10 yayının 2000 yılından önce, 64 yayının 2000 yılından sonra gerçekleştirilmiş olduğunu söyleyebiliriz.



Şekil 2.2 2000 yılı öncesi ve sonrası yayınların dağılımı

2.2 Yayın Tiplerine Göre Dağılım

İncelenen yayınlar; kitap bölümü (book chapter), makale (journal) ve bildiri (proceedings) olarak 3 gruba ayrılmış ve bu yayınların dağılımı Şekil 2.3'de verilmiştir.



Şekil 2.3 Yayın tiplerine göre dağılım

Sayısal olarak; 2 yayın kitap bölümü olarak, 27 yayın dergide ve 45 yayın bildiriler kitabında

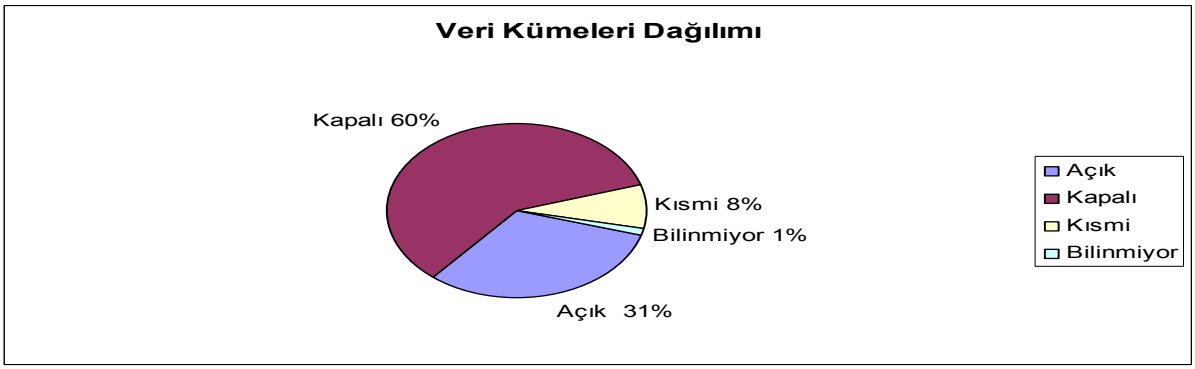
basılmıştır. Bu dergi yayınlarının; 9'u IEEE Transactions on Software Engineering, 3'ü Journal of Systems and Software, 1'i IEEE Transactions on Systems, Man, Cybernetics, 1'i IEEE Transactions for Reliability, 1'i IEEE Transactions on Neural Networks, 2'si IEEE Software, 3'ü Empirical Software Engineering, 1'i SIGSOFT Software Engineering Notes, 4'ü Software Quality Journal, 1'i International Journal of Software Engineering and Knowledge Engineering ve 1 tanesi Software-Practice and Experience dergisinde yayımlanmıştır.

2.3 Veri Kümelerinin Tipine Göre Yayınların Dağılımı

Yazılım kusur kestirimi çalışmalarında, en büyük sorunlardan birisi açık veri kümelerinin bazı çalışmalarda kullanılmıyor olmasıdır. Şirketler, projelerinde kestirim çalışmaları yürütüp dergi veya konferanslarda bu modellerini sundukları zaman veri kümelerini ticari gizlilik sebebiyle açmadıkları için bu modellerin eski ya da yeni önerilen modellerle kıyaslanması mümkün olmamaktadır. Makine öğrenmesi alanında çalışan araştırmacılar da benzer problemleri yaşadıkları için, 1990'lardan beri UCI (University of California Irvine) veri kümeleri üzerinde algoritmalarını test etmektedirler. Benzer yaklaşımla, 2005 yılında PROMISE havuzu (repository) yazılım mühendisliği konusunda araştırma yapacaklar için hazırlanmış ve NASA'nın birçok veri kümesi bu havuza eklenmiştir.

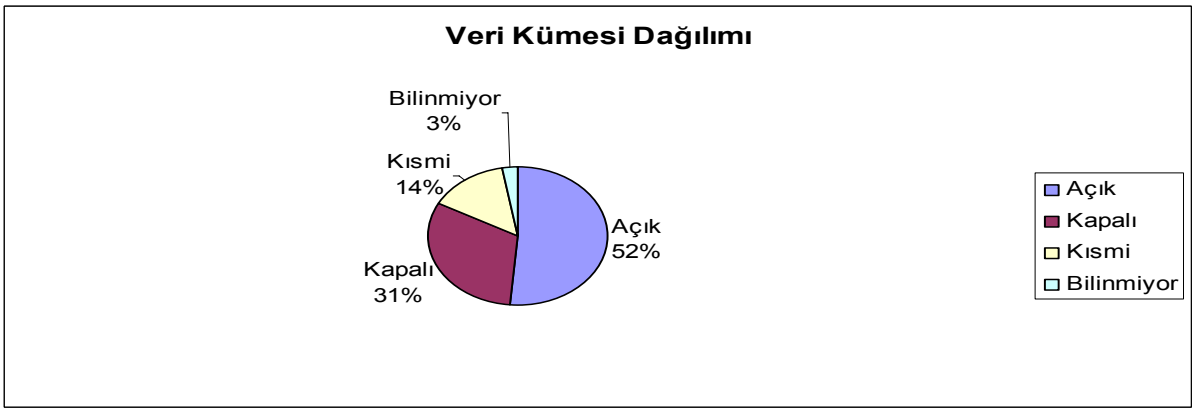
İncelenen 74 yayının; kullandığı veri kümeleri "açık", "kapalı", "kısmi" (partial), "bilinmiyor" şeklinde dört gruba ayrılmıştır. "Açık" veri kümeleri, PROMISE havuzundan kullanılan ya da ağırlıklı olarak NASA'nın açık veri kümelerinden elde edilen veri kümeleridir. "Kapalı" veri kümeleri, ticari değer taşıyan ve elde edilmesi mümkün olmayan veri kümeleridir. "Kısmi" olarak ifade edilen veri kümeleri, açık kaynaklı projeler üzerinden elde edilen ancak PROMISE havuzu gibi ortak bir alanda bulunmayan veri kümeleridir. Yayın içerisinde veri kümesine ilişkin bir bilgi bulunamadığı durumda ise, "bilinmiyor" şeklinde veri kümesi işaretlenmiştir.

74 yayın için Şekil 2.4' de, veri kümelerinin dağılımı gösterilmektedir. Bu yayınlardan, %60'ının kapalı veri kümesi kullandığı dolayısıyla ilgili çalışmalardaki deneylerin tekrarlanabilirliğinin olmadığını ve kıyaslamada kullanılamayacağını söyleyebiliriz. Açık veri kümeleri ise, tüm veri kümelerinin %31'ini oluşturuyor. Bu bilgiden hareketle, incelemekte olduğumuz üç yayından sadece birisindeki deneyi yeniden üreterek kendi modelimizle kıyaslayabileceğimizi söyleyebiliriz ki bu oran oldukça düşüktür. Sayısal olarak; 23 veri kümesi açık, 44 veri kümesi kapalı, 6 veri kümesi kısmi, 1 veri kümesi ise bilinmiyor tiptedir.



Şekil 2.4 Veri kümelerinin tiplerine göre yayınların dağılımı (1990-2007)

2005 yılından itibaren gerçekleştirilmiş yayınların dağılımı Şekil 2.5'de gösterilmektedir.



Şekil 2.5 2005 yılı sonrası veri kümesi tiplerine göre yayınların dağılımı (2005-2007)

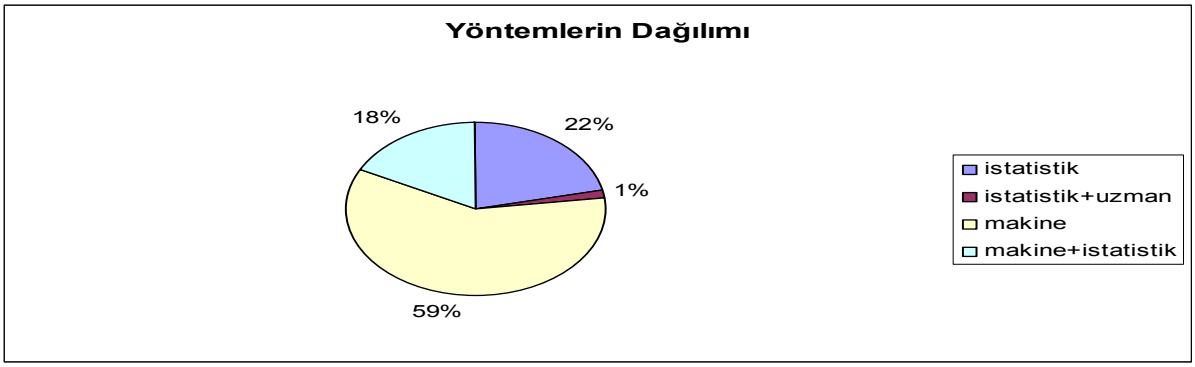
2005 yılından sonra, PROMISE havuzunun hazırlanması ve açık veri kümelerinin kullanılması konusunda bilincin oluşması sayesinde açık veri kümesi kullanma yüzdesi %31' den %52'ye çıkmıştır. Sayısal olarak; 2005 yılından sonra gerçekleştirilen yayınların, 18 tanesi açık, 11 tanesi kapalı, 5 tanesi kısmi, 1 tanesi bilinmiyor tipte veri kümesi içermektedir. Açık veri kümelerinin kullanılma yüzdesinin artması, daha olgun bir yazılım mühendisliği disiplinine doğru bir adım olarak değerlendirilebilir.

2.4 Yöntemlere Göre Yayınların Dağılımı

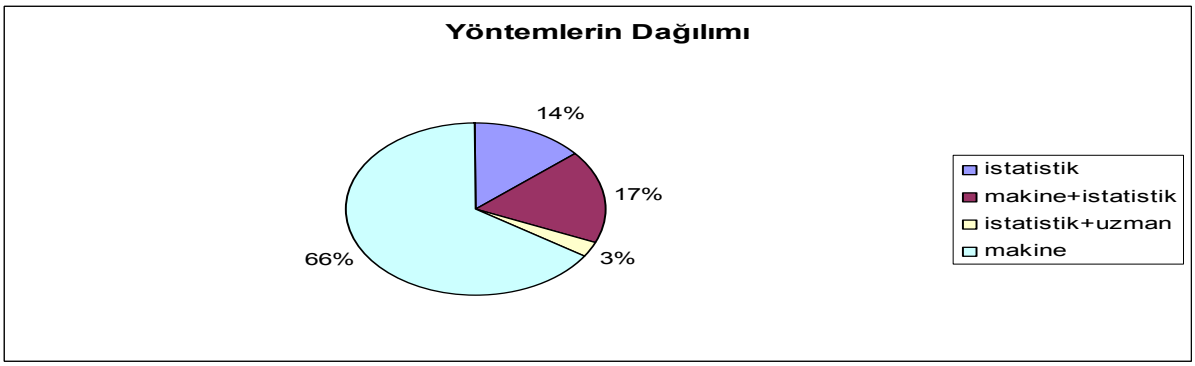
74 yayında kullanılan yöntemleri; istatistiksel yöntemler, makine öğrenmesi tabanlı yöntemler, uzman görüşü ve birleştirilmiş yöntemler olarak dört grupta değerlendirebiliriz. Şekil 2.6' da yöntemlere göre dağılımlar verilmektedir. Çalışmaların %59' u makine öğrenmesi tabanlı olup, istatistiksel yöntemler %22'lik bir dilime sahiptir. İstatistiksel

modellerin, kara kutu şeklinde çalıştığı ve veriye çok bağımlı olduğu dikkate alınırsa çalışmaların büyük çoğunluğunun makine öğrenmesi tabanlı olması umut vericidir. Uzman görüşü ise sadece 1 çalışmada kullanılmış ve elde edilen performans istatistiksel yöntemlerle kıyaslanmıştır. Sayısal olarak; 16 yayında istatistiksel yöntemler, 1 yayında istatistiksel ve uzman görüşü, 44 yayında makine öğrenmesi tabanlı yöntemler, 13 yayında istatistiksel yöntemler ve makine öğrenmesi yaklaşımları birlikte ya da kıyaslama amacıyla ilgili çalışmada kullanılmıştır.

2005 yılından sonraki yayınların yöntemlere göre dağılımı Şekil 2.7' de verilmektedir.



Şekil 2.6 Yöntemlere göre yayınların dağılımı (1990-2007)



Şekil 2.7 Yöntemlere göre yayınların dağılımı (2005-2007)

2005 yılından sonraki yayınlar incelendiğinde, makine öğrenmesi tabanlı yaklaşımların yüzdesinin %59'dan %66'ya çıkmış olacağı görülecektir. İstatistiksel yöntemler ise, %18'den %14'e düşmüştür. Bu sayısal verilere göre, son yıllarda makine öğrenmesi tabanlı yaklaşımların yazılım kusur kestiriminde daha fazla uygulanmaya başladığı ve istatistiksel yöntemlerden içerdiği problemler nedeniyle vazgeçilmeye başlandığı sonuçları çıkarılabilir. Sayısal olarak 2005 yılından sonra; 5 yayın istatistiksel yöntemleri, 6 yayın makine öğrenmesi

ve istatistiksel yöntemleri, 1 yayın uzman görüşü ve istatistiksel yöntemleri, 23 yayın makine öğrenmesi yaklaşımlarını kullanmıştır.

2.5 Metrik Tiplerine Göre Yayınların Dağılımı

İncelenen 74 yayındaki metrikler, altı grupta değerlendirilmiştir:

- *Metot seviyesinde:* Yapısal ve nesneye yönelik programlamanın her ikisinde de metotlar mevcuttur. Çevrimsel karmaşıklık (cyclomatic complexity), Halstead ve McCabe metrikleri metot bazında metriklere örnek olarak verilebilir.

- *Sınıf seviyesinde:* Nesneye yönelik programlamaya özel metriklerdir. Kalıtım ağacının derinliği, sınıfın çocuklarının sayısı gibi sınıf seviyesinde metrikler mevcuttur. Chidamber-Kemerer metrik kümesi, sınıf seviyesindeki metriklerden en fazla bilinenidir.

- *Süreç seviyesinde:* Metrikleri, ürün ve süreç metrikleri olarak iki grupta değerlendirmek mümkündür. Ürün metriklerini statik ve dinamik metrikler olarak iki alt gruba ayırabiliriz. Metot ve sınıf seviyesindeki metrikler, statik ürün metriklerindendir. Süreç seviyesindeki metrikler ise yazılım geliştirme sürecinin farklı aşamalarından elde edilen metriklerdir. İsterler analizi ya da tasarım aşamasında elde edilen süreç metrikleri, kusur kestirim modellerinde kullanılabilir.

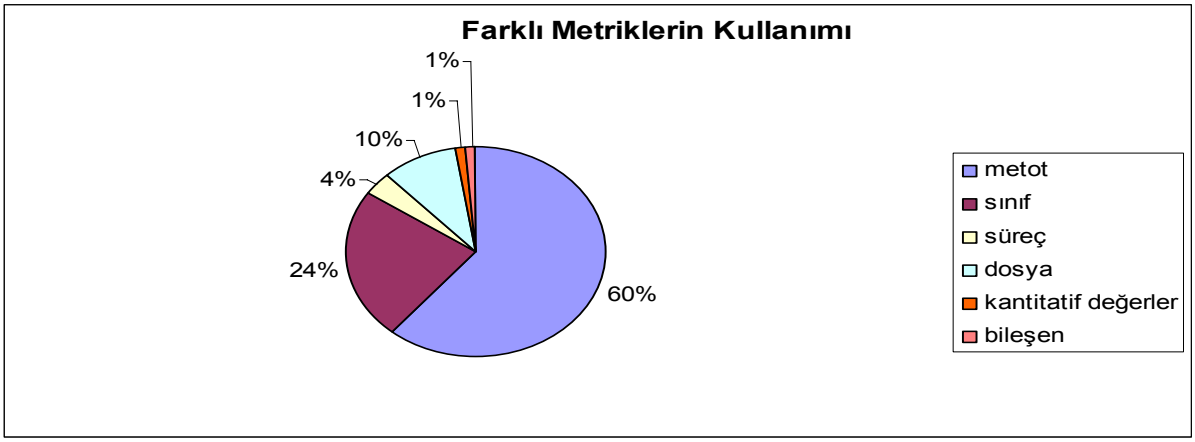
- *Bileşen seviyesinde:* Nesneye yönelik programlamadan sonra popüler olan bileşen tabanlı geliştirmeden elde edilen metrikler kullanılarak kestirim gerçekleştirilebilir.

- *Dosya seviyesinde:* Kaynak kodlar, dosya bazında ele alınarak, bu dosya üzerinde son zamanlarda değişiklik yapılıp yapılmadığı, dosyanın kod satır sayısı gibi dosyaya özel metrikler tanımlanarak modellerde kullanılabilir.

- *Farklı kantitatif değerler seviyesinde:* İşlemci kullanım oranı, disk kullanım oranı, belge kalitesi, kullanıcı sayısı gibi farklı kantitatif değerlerle kusur kestirimi gerçekleştirilebilir.

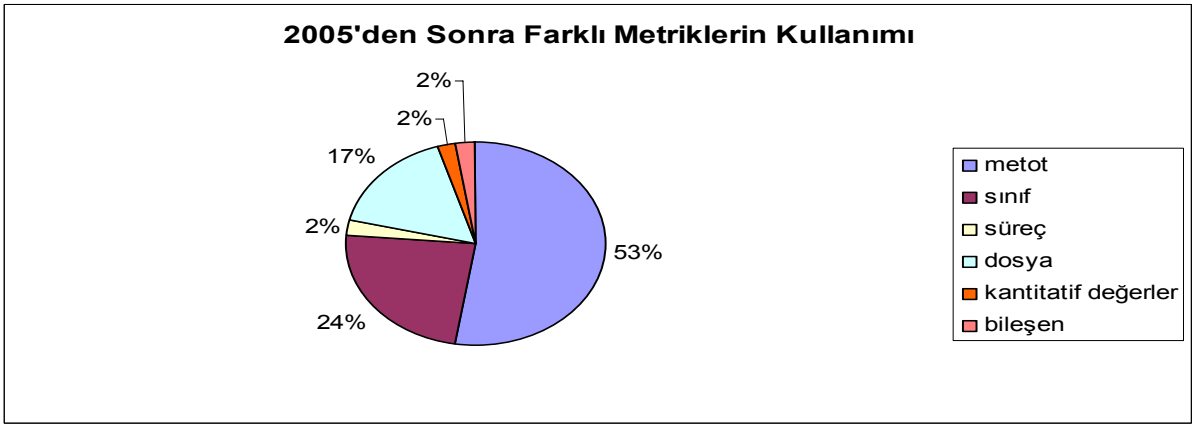
Bu altı metrik grubu dikkate alınarak, 74 yayının dağılımı Şekil 2.8' de verilmektedir. %60'lık yüzdeyle en büyük dilime sahip olan metrik tipi metot seviyesindeki metriklerdir. Sınıf seviyesindeki metrik tipi, %24'lük bir dilime sahiptir. Süreç bazında metriklerin yer aldığı çalışmalar ise %4'lük bir dilim tutmakta olup oldukça düşük düzeydedir. Metot seviyesindeki metriklerle kestirim yapmak, kusur eğilimli modülleri metot bazında

belirleyeceği için uygulayıcılara daha fazla yarar sağlamaktadır. Sınıf seviyesindeki metriklerle yapılan çalışmalarda ise, sadece sınıf düzeyinde kusurun olup olmadığı bilgisi mevcuttur. Sayısal olarak; 51 yayında metot, 20 yayında sınıf, 3 yayında süreç, 8 yayında dosya, 1 yayında bileşen ve 1 yayında kantitatif veri kümeleri kullanılmıştır.



Şekil 2.8 Farklı metrik tiplerine göre yayınların dağılımı (2000-2007)

2005 yılından sonraki çalışmalar için, farklı metriklerin kullanım yüzdeleri Şekil 2.9'da verilmektedir. Sınıf seviyesindeki metriklerle yapılan çalışmaların yüzdesi değişmezken, metot seviyesindeki metriklerle yapılan çalışmaların yüzdesi azalmış ve dosya seviyesindeki metriklerle yapılan çalışmaların yüzdesi artmıştır. Süreç ve bileşen seviyesindeki metriklerle yapılan çalışmaların yüzdesi yine çok düşüktür. Sayısal olarak; incelenen 74 yayın içerisinden 2005 yılından sonra yayımlananların 22'si metot, 10'u sınıf, 1'i süreç, 7'si dosya, 1'i bileşen ve 1'i kantitatif değerler seviyesinde metrikler kullanmıştır.



Şekil 2.9 Farklı metrik tiplerine göre yayınların dağılımı (2005-2007)

2.6 İncelenen Yayınların Açıklanması

2.6.1 1990-2000 Yılları Arasındaki Çalışmalar

Porter ve Selby (1990), NASA ve Hughes Aircraft şirketinin iki yazılımı üzerinde, metot seviyesinde metrikler ile otomatik sınıflandırma ağacı oluşturarak %79.3 doğruluk (accuracy) elde etmişlerdir. Yazılım kusur kestirimi çalışmalarındaki veri kümeleri, ayrık veri kümeleri olduğundan değerlendirme kriteri olarak doğruluk kullanılmamalıdır. Aksi halde, tüm veri kümesine göre çok düşük yüzdeye sahip olan kusurlu modüllerin hiçbirisinin doğru kestirilemediği durumda bile yüksek doğruluk elde edileceği için modelin yararlı olduğu gibi yanlış çıkarımlar yapılabilir. Bu çalışmanın eksikliği, doğruluğu değerlendirme kriteri olarak kullanmış olmasıdır.

Briand ve arkadaşları (1993); lojistik regresyon, sınıflandırma ağaçları ve Optimize Küme Azaltma (Optimized Set Reduction-OSR) yöntemlerini 260,000 kod satırından oluşan ADA sisteminin 146 bileşeni üzerinde analiz etmişlerdir. OSR; Maryland Üniversitesinde TAME projesi sırasında geliştirilmiş, birkaç yazılım mühendisliği probleminde uygulanmış ve ilk olarak proje maliyet tahminleme probleminde denenmiştir. Bu çalışmada, sınıf seviyesinde ve metot seviyesinde metrikler kullanmışlardır. Doğru şekilde kestirilen kusurlu modüllerin yüzdesi (hatasızlık / correctness) ve model tarafından kestirilen yüksek riskli modüllerin yüzdesi (tamlık / completeness) değerlendirme kriteri olarak seçilmiştir. OSR; 146 veri noktasından oluşan küçük bir veri kümesinde %90 hatasızlık ve tamlık değerlerine sahip olmuş, performans ölçüm değerlerine göre lojistik regresyondan daha başarılı sonuçlar üretmiş ve sınıflandırma ağaçlarından daha etkin sonuçlar vermiştir. Özetle, bu çalışmada en iyi performans OSR ile elde edilmiştir.

Lanubile ve arkadaşları (Lanubile vd., 1995), Bari Üniversitesinde (İtalya) geliştirilen 27 akademik proje üzerinde; temel bileşen analizi, diskriminant analizi, lojistik regresyon, mantıksal sınıflama modelleri, katmanlı sinir ağları ve holografik ağları kullanarak kusur eğilimli bileşenleri belirlemeye çalışmışlardır. Halstead, McCabe, Henry&Kafura bilgi akış metriklerinden oluşan 11 adet metot seviyesinde metrikden yararlanılmış, değerlendirme kriteri olarak yanlış sınıflama oranı (misclassification rate), başarılan kalite (achieved quality), doğrulama maliyeti (verification cost) kullanılmıştır. Bu çalışmaya göre; hiçbir yöntem iyi sonuç üretmemiş ve hiç bir yöntem kusurlu modüllerle kusurlu olmayan modüller arasındaki farkılığı belirleyememiştir.

Cohen ve Devanbau (1997), Maryland Üniversitesinde ortak ölçekli Bilgi Yönetim Sistemi

geliştirmeyi hedefleyen bir yazılım mühendisliği dersinin verilerini ve sınıf seviyesindeki bağlılık (coupling) metriklerini FOIL ve FLIPPER adı verilen iki Tümevarımsal Mantık Programlama (Inductive Logic Programming) sistemi içerisinde kullanmışlardır. Ayrıca, hata oranı ve süreyi değerlendirme kriteri olarak seçmişlerdir. FLIPPER, FOIL'den daha yüksek performans sunmuş ve daha hızlı çalışmıştır. Veri üzerinde gürültü olduğu durumlarda, FLIPPER daha iyi performans sunmuş ve yapay veri kümelerinde daha hızlı çalışmıştır. Bu çalışma sonunda, FLIPPER sistemi için iki değişiklik önerisinde bulunmuşlardır.

Khoshgoftaar ve arkadaşları (1997); 13 milyon kod satırından oluşan, PROTEL dilinin kullanıldığı çok büyük telekom yazılımı üzerinde bazı sınıf seviyesindeki metrikleri yapay sinir ağları ve diskriminant model dahilinde kullanarak kusur kestirimi üzerinde çalışmışlardır. Değerlendirme kriteri olarak; Tip-I hatası, Tip-II hatası ve toplam yanlış sınıflama oranlarını kullanmışlardır. Bu çalışmada bir modül, bir fonksiyonu yerine getirmek üzere birlikte çalışan kaynak kod dosyaları kümesi olarak ele alınmıştır. Çalışmaya göre, yapay sinir ağları parametrik olmayan diskriminant modelinden daha iyi sonuçlar vermiş, yapay sinir ağlarının Tip-II hatası Diskriminant'ın Tip-II hatasından daha küçük çıkmış ve bu sinir ağı modeli EMERALD yazılımına tümleştirilmiştir.

Evett ve arkadaşları (1998), genetik programlamayı kullanarak askeri iletişim sistemi ve bir telekom sistemi üzerinde 8 adet metot seviyesindeki metrikle (McCabe, Halstead ve kod satır sayısı) kalite kestirimini gerçekleştirmişlerdir. Değerlendirme kriteri olarak "sırasal değerlendirme prosedürü" (ordinal evaluation procedure) adı verilen bir yöntem kullanmışlardır. Bu çalışma sayesinde, ilk kez genetik programlama kalite modellemede değerlendirilmiş ve oldukça iyi performans elde edilmiştir.

Ohlsson ve arkadaşları (1998), özellik azaltmak için temel bileşen analizi yöntemini kullanmış ve ardından diskriminant analizi ile sınıflandırma işlemini gerçekleştirmişlerdir. Veri kümesi olarak, Ericsson telekom veri kümesi ve metrik olarak tasarım belgelerinden toplanan tasarım metrikleri kullanılmıştır. Tasarım fazında, Ericsson tarafından Formal Tanımlama Dili (Formal Definition Language-FDL) kullanıldığı için tasarım belgelerinden metrikler kolayca elde edilmiştir. Değerlendirme kriteri olarak, toplam yanlış sınıflama oranı kullanılmıştır. Khoshgoftaar'ın Diskriminant analiz yöntemi Ericsson veri kümesinde iyi sonuç vermemiştir. Bu yöntemdeki istatistiksel yöntembilimi incelendikten sonra, başarısızlığının nedeninin olasılık yoğunluk fonksiyonunun kestiriminden kaynaklandığı belirlenmiştir. Bu yöntembilimi iyileştirmişler ve çok değişkenli analiz teknikleriyle başarılı sonuçlar elde etmişlerdir. İki gruptan (kusurlu ve kusurlu değil) daha fazla grup kullandıkları

zaman performansın yükseldiğini gözlemlemişlerdir.

Binkley ve Schach (Binkley ve Schach, 1998), OASIS adı verilen ders kayıt sistemi üzerinde fan-in gibi modüller arası metrikleri ve çevrimsel karmaşıklık gibi modül içi metrikleri kullanarak "Spearman sıra (rank) korelasyon testi" ile çalışma anındaki arızaların kestirimini gerçekleştirmişlerdir. Bu çalışma sonucunda; statik modüller arası metriklerle çalışma anındaki davranışın kestirilmesinin mümkün olduğu ve modüller arası metriklerin çalışma anı hataları ile yüksek seviyede korele olduğu raporlanmıştır.

De Almeida ve Matwin (1999), yerel telefon trafiğinden sorumlu Bell Kanada'nın bazı programları üzerinde metot seviyesindeki metrikleri (Halstead, McCabe, kod satır sayısı) kullanarak yazılım kalite modeli kurma çalışması yürütmüşlerdir. Yöntem olarak; C4.5, NewID, CN2 ve FOIL yöntemlerinden yararlanmışlardır. Değerlendirme kriteri olarak; hatasızlık, tamlık ve doğruluk parametreleri kullanılmıştır. NewID, C4.5 ve CN2 benzer sonuçlar üretmiş ancak FOIL'in performansı diğer algoritmalardan daha kötü çıkmıştır. Anlaşılırlık açısından, C4.5'in kurallarının daha uygun olduğu belirtilmiştir. Bu çalışmada; istatistiksel modellerin kara kutu özellikte olduğu, bu tür modellerin yorumlanmasının tartışmalı olduğu ve veriye çok bağımlı oldukları ifade edilmiştir. Farklı organizasyonların verileriyle kestirim yapmanın uygun olmayacağı raporlanmıştır.

Kaszycki (1999), lojistik regresyon içerisinde süreç ve ürün metriklerinden yararlanmıştır. Süreç metrikleri olarak; programcı deneyimi gibi metrikleri dikkate almış ve 4 adet metriği kod üzerinden elde etmiştir. Değerlendirme kriteri olarak; Doğru Negatif (True Negative) ve Doğru Pozitif (True Positive) oranlarını kullanmıştır. Bu çalışmada; süreç metrikleri modele eklendiği zaman performansın iyileştiği ve EMERALD ürününde süreç metriklerinin dikkate alındığı raporlanmıştır. Kaszycki, Nortel Networks'de çalışan bir araştırmacı olup EMERALD Nortel Networks firmasında geliştirilmiş olan risk değerlendirme aracıdır. Bu araç, karmaşıklık metrikleri ve süreç metriklerini değerlendirerek bir model oluşturmaktadır. En basit süreç metriği olarak, modülün son zamanlarda değişip değişmediği bilgisi kullanılabilir. Bu çalışmanın önemi, süreç metriklerinin ürün metrikleri ile birlikte kullanıldığı durumda kestirim performansının olumlu yönde etkilendiğinin gösterilmesidir. Ancak şirketlerin yeniden yapılandığı durumda, süreçlerin de değiştiği dikkate alınırsa, süreç metriklerinin yeniden belirlenmesi ve modelin yeniden oluşturulması gerekir. Bu durum ise, şirketler için ek iş yükü doğurur.

2.6.2 2000-2003 Yılları Arasındaki Çalışmalar

Yuan ve arkadaşları (2000), öncelikle kusur sayısının kestirimi için "bulanık eksiltici kümeleme" (fuzzy subtractive clustering) yöntemini kullanmış, ardından "modül sıra modelleme" (module-order modeling) ile modüllerin kusur eğilimli olup olmadığının kestirimini yapmışlardır. Bu çalışmaya özel 10 adet metot seviyesindeki metrikle dört model geliştirmişler ve ek olarak, süreç ve ürün metriklerinin dikkate alındığı bir model oluşturmuşlardır. Değerlendirme kriterleri olarak; Tip-I hatası, Tip-II hatası, yanlış sınıflandırma oranı, etkinlik (effectiveness) yani doğru kestirilen kusurlu modüllerin oranı, verimlilik (efficiency) yani iyileştirilecek kusurlu modüllerin oranı kullanılmıştır. Bu çalışmanın sonunda; süreç metriklerinin sınıflandırma doğruluğunu iyileştirmediği ve yöntemin kabul edilebilir sonuçlar ürettiği raporlanmıştır. Çoklu lineer regresyonlar, bulanık eksiltici kümenin özel bir durumudur. Eğitim sürecinde sadece bir tane küme merkezi elde edilirse, bu yöntem çoklu lineer regresyon halini alır. Bu çalışmada bir modül, bir fonksiyonu gerçekleştirmek için bir arada olan birkaç kaynak dosyasından oluşmaktadır. Ayrıca, kusur olarak sadece müşterinin raporladığı kusurlar ele alınmıştır.

Denaro (Denaro, 2000), lojistik regresyon tekniği ile anten konfigürasyon yazılımında, ticari ve prototip araçlarla elde edilen 37 metrikle kusur eğilimli modülleri kestirmeye çalışmıştır. Değerlendirme kriteri olarak, R^2 kullanılmıştır. Bu değer, 1'e yaklaştıkça modelin önemi artmaktadır. Çalışmaya göre, statik metriklerle kusur eğilimliliği arasında korelasyon vardır.

Khoshgoftaar ve arkadaşları (2000a), Ada dilinde geliştirilmiş komuta-kontrol-iletişim yazılımı için senaryo tabanlı usavurum (case-based reasoning) tekniği ile 8 adet metot seviyesindeki metriği kullanarak (Halstead, McCabe, kod satır sayısı) yazılım kalitesi kestirimini gerçekleştirmişlerdir. Değerlendirme kriteri olarak; Tip-I ve Tip-II hataları kullanılmış ve sonuçta çok yararlı bir kestirici elde edilmiştir.

Xu ve arkadaşları (2000); Protel dili ile geliştirilmiş büyük bir telekom sistemi üzerinde, öncelikle özellik azaltma için temel bileşen analizi yöntemini kullanmışlar ve ardından "bulanık lineer olmayan regresyon" (fuzzy nonlinear regression-FNR) yöntemini uygulamışlardır. EMERALD aracından toplanan 24 metot seviyesinde ve 4 adet yürütme (execution) metriği kullanılmıştır. Değerlendirme kriteri olarak "ortalama mutlak hata" (average absolute error-AAR) seçilmiş ve erken fazdaki kestirim için FNR'nin umut veren bir teknoloji olduğu raporlanmıştır. FNR modelleme, klasik lineer regresyondan farklı olarak çıkışında bulanık değerler üretir. FNR modeli, belirli bir olasılık dahilinde her modülün sahip olacağı kusur sayısının denk düşeceği bir aralığın kestirimini yapar, kusur sayısının tam

kestirimini gerçekleştirmez. Bulanık kelimesi çıkışın bulanık küme olduğunu, lineer olmayan ifadesi yapay sinir ağlarının kullanıldığını, regresyon terimi miktarların (quantities) bir veri kümesinden elde edildiğini ortaya koyar. İlk yapay sinir ağı ile bağımlı değişkenin üst sınırları, ikinci yapay sinir ağı ile alt sınırları belirlenir ve bu eğitim birbirinden bağımsızdır. Bu çalışmada, bir modül, birden fazla kaynak kod dosyasından oluşan kümedir.

Guo ve Lyu (2000), EM (Expectation Maximization) algoritması ile "sonlu birleştirilmiş model analizi" (finite mixture model analysis) yöntemini kullanarak 40,000 kod satırından oluşan, Pascal ve Fortran dilleri ile geliştirilmiş medikal görüntüleme sistemi üzerinde kalite kestirim çalışmaları yürütmüşlerdir. Metot seviyesindeki metrikleri (Halstead, McCabe, Belady'nin bant genişliği, Jensen'in program uzunluğu) kullanıp Tip-I, Tip-II ve yanlış sınıflandırma oranı parametreleriyle algoritmanın değerlendirmesini yapmışlardır. Çalışma sonunda; bir önceki sürümün kusur verileri tam olmasa bile bu algoritmanın sonuç üretebildiği, Tip-II hatası için en iyi sonucun %13 olduğu ve diskriminant analizinde elde edilen hata değeriyle bu değerin aynı olduğu raporlanmıştır. Birleştirilmiş modellerde çoğu zaman Gauss dağılım modeli kullanılmaktadır.

Khoshgoftaar ve arkadaşları (2001), "sıfırla şişirilmiş Poisson regresyon modeli" (Zero-Inflated Poisson regression model-ZIP) ile iki büyük yazılım uygulamasında, dosya seviyesindeki ürün metriklerini kullanarak kusur kestirimini gerçekleştirmişlerdir. Bu metrikler; sistem testi öncesi kaynak kodun kaç kez incelendiği (inspection), kodlama öncesi mevcut olan kaynak kodun uzunluğu (otomatik oluşturulmuş kod), sistem testi öncesi kaynak kodun uzunluğu, kod öncesi mevcut olan yorum (comment) satırlarının uzunluğu (otomatik oluşturulmuş), sistem testi öncesi kaynak koddaki yorumların uzunluğu olarak seçilmiştir. Değerlendirme kriteri; "ortalama mutlak hata" ve "ortalama bağıl hata" (average relative error-ARE) olarak seçilmiştir. Bu çalışmada; Poisson regresyon modeli (PRM) ve sıfırla şişirilmiş Poisson regresyon modeli (ZIP) denenmiş ve ZIP, Vuong testine göre PRM'den daha iyi sonuçlar üretmiştir. Ayrıca; ZIP modelinin ortlama mutlak ve bağıl hatası daha küçük çıkmıştır. PRM ve ZIP modelleri biri diğerinin özel bir durumu olmadığından, hipotez testinde Vuang'in önerdiği test kullanılmıştır.

Schneidewind (2001), Uzay Mekiği yazılımına ilişkin veri kümesinde "İkili Diskriminant Fonksiyonlar" (Boolean Discriminant Functions-BDF) ve "Lojistik Regresyon Fonksiyonları" (LRF) yöntemleriyle 6 adet metot seviyesindeki metriği kullanarak yazılım kalite kestirimi üzerinde çalışmıştır. Değerlendirme için; Tip-I hatası, Tip-II hatası, yanlış sınıflandırma oranı, doğru şekilde sınıflanan kusurlu olmayan modüllerin oranı (LQC) kriterlerini kullanmıştır. Bu

çalışmada; BDF, LRF'den daha iyi performans sunmuştur. LRF, BDF ile birleştirildiği zaman inceleme maliyeti (inspection cost) artmıştır.

El-Emam ve arkadaşları (1999), ticari bir Java uygulamasında lojistik regresyon yöntemiyle sınıf seviyesindeki metrikleri kullanarak kusurlu sınıfların kestirimini sağlamışlardır. 2 metriği Chidamber-Kemerer metrik kümesinden, 8 metriği Briand'in metrik kümesinden kullanmışlar ve ek olarak büyüklük metriğinden yararlanmışlardır. Değerlendirme kriteri olarak; J katsayısını tercih etmişlerdir. J katsayısı; duyarlık (sensitivity) ve kesinlik (specificity) değerlerinin toplamından 1 çıkarılarak elde edilmektedir. Bu çalışmada; "kalıtım derinliği" ve "ihraç bağlılık" (Export Coupling-EC) metriklerinin sınıfların kusur eğilimliliğini belirlemede en yararlı metrikler olduğu raporlanmıştır. Önceki çalışmalarında, kalıtım derinliği önemsiz bir metrik olarak raporlandığından buradaki farklılık bu yazılımın kötü tasarlanmış olmasından kaynaklanabilir. EC metriği, kalıtım derinliğinden çok daha yararlıdır. Yüksek EC değerine sahip sınıflar; diğer sınıflar tarafından çok fazla kez çağrılır ve her sınıfta eşit sayıda kusur olsa bile, EC değerinin yüksek olduğu sınıflar çok kez çağrıldıkları için, bu sınıflarda daha fazla kusur ortaya çıkacaktır. Yüksek EC değerinin, kusur eğilimliliğine yansıması bu şekilde açıklanabilir.

Khoshgoftaar ve arkadaşları (2002a), kablosuz ürünlerin konfigürasyonunu yapan iki yazılım üzerinde PRM, ZIP ve modül-sıra modeli (module-order model) teknikleriyle 5 dosya seviyesindeki metriği kullanarak kusur kestirimini gerçekleştirmişlerdir. Ortalama mutlak hata ve ortalama bağıl hata, değerlendirme kriteri olarak seçilmiştir. Bu çalışmada, kestirim modelleri iyi sonuç vermemesine rağmen, modül-sıra modeli etkin çalışmıştır. Ayrıca; en iyi kestirimin, en iyi modül-sıra modeli anlamına gelmediği ve kötü kestirimin modül-sıra modeli için bir engel olmayacağı ifade edilmiştir.

Khoshgoftaar (2002b), askeri komuta-kontrol-iletişim sistemi üzerinde Genelleştirilmiş İkili Diskriminant Fonksiyonları (Generalized Boolean Discriminant Functions-GBDF) tekniği ile 8 adet metot seviyesindeki metriği (Halstead ve McCabe) kullanarak kalite sınıflandırma problemi üzerinde çalışmışladır. Değerlendirme için; Tip-I ve Tip-II hatası kullanılmıştır. Çalışmaya göre; GBDF BDF'den daha iyi sonuçlar üretmiştir. BDF, birçok modülü kusurlu olarak kestirdiğinden inceleme maliyeti yüksek çıkmıştır. Ayrıca, BDF'nin Tip-I hatası oldukça yüksektir.

Mahaweerawat ve arkadaşları (2002), veri kümesini açıklamadıkları bir projede ilk önce bulanık eksiltici kümeleme kullanmış ve ardından RBF (radial basis function) yöntemini

uygulamışlardır. Değerlendirme için; Tip-I hatası, Tip-II hatası, yanlış sınıflandırma oranı, inceleme, tamlık kriterlerini kullanmışlar ve metot seviyesindeki (Halstead, McCabe, Henry&Kafura) metriklerden yararlanmışlardır. RBF, çok katmanlı algılayıcıdan (Multi Layer Perceptron-MLP) daha iyi sonuçlar üretmiştir. MLP'nin tamlık değeri daha yüksek olmasına rağmen, inceleme maliyeti yüksek çıkmıştır. RBF, kusur eğilimli modülleri %83 doğrulukla kestirmesine rağmen MLP, %60 doğrulukla kestirmiştir.

Khoshgoftaar ve Seliya (2002c), büyük bir telekom sisteminde SPRINT ve CART yöntemleri ile 28 metot seviyesindeki metriği (24 ürün metriği ve 4 yürütme metriği) kullanarak kalite sınıflandırma konusunda çalışmışlardır. SPRINT, sınıflandırma ağacı algoritması iken CART karar ağacı algoritmasıdır. Değerlendirme kriteri için Tip-I hatası, Tip-II hatası ve yanlış sınıflandırma oranı seçilmiştir. Bu çalışmada; SPRINT algoritması daha düşük Tip-I hatasına sahip olmuş, kararlılığının daha iyi olduğu gösterilmiş, model oldukça gürbüz (robust) çalışmıştır. Ancak ağacın yapısı CART'a göre daha karmaşıktır.

Pizzi ve arkadaşları (2002), bir araştırma aracı üzerinde çok katmanlı algılayıcı ile metot ve sınıf seviyesindeki metriklerle kalite kestirimini gerçekleştirmişlerdir. Değerlendirme kriteri olarak doğruluk tercih edilmiştir. Çalışmada, MLP sınıflayıcısı öncesinde "medyan-düzeltilmiş sınıf etiketleri" (median-adjusted class labels) kullanmanın etkin bir ön işleme tekniği olduğu ifade edilmiştir.

Khoshgoftaar ve Seliya (2002d), büyük bir telekom sistemi üzerinde ağaç tabanlı yazılım kalite kestirim modelleri ile metot seviyesindeki tasarım metriklerini kullanarak kalite kestirimini gerçekleştirmişlerdir. İncelenen teknikler; CART-LS (least squares), S-PLUS, CART-LAD (least absolute deviations) olarak sıralanabilir. Değerlendirme için, ortalama mutlak hata ve ortalama bağlı hata kullanılmıştır. Çalışma sonucunda; CART-LAD ağaçlarının doğruluk değerinin iyi ve yorumlanmasının kolay, S-PLUS ağaçlarının kestirim doğruluğunun kötü ve çıkan ağacın oldukça karmaşık olduğu raporlanmıştır. Çalışma sonucunda, CART-LAD ağaçlarının kestirimde kullanılması önerilmiştir.

2.6.3 2003-2005 Yılları Arasındaki Çalışmalar

Reformat (2003), ticari bir medikal görüntüleme yazılımı üzerinde bulanık kural tabanlı modellerle 11 metot seviyesindeki metriği kullanarak kusur sayılarının kestirimi üzerinde çalışmıştır. Çalışmada; 9 modelin sınıflandırma oranının %62.82 ve % 79.49 arasında değiştiği, meta model kestirim sisteminin uygulandığı durumda sınıflandırma oranının %85.90'a ulaştığı raporlanmıştır. Reformat, kanıt teorisinin (evidence theory) elemanlarını

bulanık kural tabanlı modellerin kombinasyonuna uygulamıştır.

Koru ve Tian (2003), IBM ve Nortel Networks firmasının altı büyük ölçekli ürünü üzerinde, ağaç tabanlı modelleri kullanan basitleştirilmiş kümeleme algoritmasıyla, metot seviyesindeki metrikleri kullanarak (IBM ürünleri için 15 metot seviyesindeki metrik, Nortel ürünleri için 49 metot seviyesinde karmaşıklık metriği) yüksek kusurlu modüllerle yüksek riskli modüller arasındaki ilişkiyi incelemişlerdir. Değerlendirme kriteri olarak, Mann-Whitney U-testinden yararlanmışlardır. Çalışmada; yüksek kusur eğilimli modüllerin genellikle oldukça karmaşık olduğu ancak en karmaşık modüller olmadığı ve en karmaşık modüllerin kabul edilebilir bir kalitesinin olduğu saptanmıştır. Kusur eğilimliliği en fazla olan modüllerin, en karmaşık modüllerin altında kalan modüller olduğu raporlanmıştır. IBM ve Nortel Networks'deki bazı araştırmacılar, bazı modüllerin karmaşıklığının çözdükleri problemin doğasındaki karmaşıklıktan kaynaklandığını ifade etmiştir.

Denaro ve arkadaşları (2003a), endüstriyel bir telekom uygulaması üzerinde lojistik regresyonla sınıf seviyesindeki metrikleri kullanarak kusur kestirimi çalışmaları yürütmüşlerdir. Nesneler arasındaki bağlılık (coupling between object classes-CBO) ve metotların kohezyon eksikliği (lack of cohesion of methods-LCOM) metrikleri otomatik araçla hesaplanamadığından kullanılamamış, bu metriklerin dışında kalan CK metrikleri kullanılabilmiştir. Bu çalışmaya göre, hiçbir nesneye yönelik metrik kusur eğilimliliği ile korele değildir ve çok değişkenli modeller kod satır sayısına göre herhangi bir avantaj sağlamaz. Çalışmada kullanılan yazılım projesi, bir C uygulaması olarak başlamış ve zamanla nesneye yönelik hale gelmiştir. Metrikler ve kusur eğilimliliği arasında bir korelasyon çıkmaması, nesneye yönelik tasarımın iyi yapılmamış olmasından ve bu evrimsel gelişimden kaynaklanmış olabilir. Bu çalışmada, bir başlık (header) ve cpp dosyası modül olarak ele alınmıştır.

Thwin ve Quah (2003), insan makine arayüz yazılımı üzerinde Genel Regresyon Sinir Ağı (General Regression Neural Network-GRNN) ve Ward Sinir Ağıyla sınıf seviyesindeki metrikleri kullanarak yazılım kalite kestirimi üzerinde çalışmışlardır. Değerlendirme kriteri olarak; R^2 (çoklu determinasyon katsayısı), r (korelasyon katsayısı), ortalama karesel hata, ortalama mutlak hata, minimum mutlak hata, maksimum mutlak hata gibi kriterler kullanılmıştır. Bu çalışmaya göre; GRNN, Ward ağından daha yüksek doğrulukta sonuçlar üretmektedir.

Khoshgoftaar ve Seliya (2003), büyük bir telekom sistemi üzerinde CART-LS, CART-LAD,

S-PLUS, çoklu lineer regresyon, yapay sinir ağı, senaryo tabanlı usavurum yöntemleriyle 24 ürün ve 4 yürütme anı metriğini kullanarak kusur kestirimi üzerinde çalışmışlardır. Performans sıralamaları için, çoklu-eşli (multiple-pairwise) kıyaslamayı kullanmışlar ve deneysel tasarım yaklaşımı olarak "iki yollu ANOVA rastgele-tam blok tasarım modelini" (two-way ANOVA randomized-complete block design model) tercih etmişlerdir. Bu çalışmada performans sıralamaları en iyiden en kötüye doğru; CART-LAD, senaryo tabanlı usavurum, çoklu lineer regresyon, yapay sinir ağı, CART-LS, S-PLUS olarak verilmiştir. Ayrıca çalışmada, temel bileşen analizi yönteminin her zaman kusur kestirim doğruluğunu iyileştireceğinin beklenmemesi ifade edilmiştir. Ancak ham metrikler arasındaki korelasyonu yok ettiği ve sonucundaki modelin daha gürbüz olduğu açıklanmıştır.

Guo ve arkadaşları (2003), NASA'nın KC2 veri kümesi üzerinde Dempster-Shafer Belief ağlarıyla 21 adet metot seviyesindeki metriği kullanarak kusur eğilimli modüllerin kestirimini gerçekleştirmişlerdir. Değerlendirme kriteri olarak; tespit olasılığı (probability of detection-PD), çaba ve doğruluk kullanılmıştır. Çalışmada kullanılan ağın kestirim doğruluğu, lojistik regresyondan ve diskriminant analizinden daha yüksek çıkmış ve ROCKY'den çaba açısından daha ekonomik olduğu tespit edilmiştir. KC2 veri kümesinde 2 ve 4 özellik arasında metriğin kullanıldığı durumda optimal kestirimin sağlandığı ifade edilmiştir. 21 adet metriğin birbiri ile yüksek seviyede korele olduğundan en iyi küme lojistik regresyon prosedürü ile seçilmiştir. Ayrıca; Dempster-Shafer Belief ağları ayrık veri kümeleriyle çalışabildiğinden orjinal veri kümesi AWK programları ile ayrık değerlere dönüştürülmüştür. Ortalama değerinden küçük değere sahip olan özellikler 0, büyük olanlar 1 ile temsil edilmiştir.

Denaro ve arkadaşları (2003b), anten konfigürasyon sistemi ve Apache 1.3, 2.0 yazılımları üzerinde lojistik regresyonla çeşitli metot seviyesindeki metrikle modüllerin kusur eğilimliliğini belirlemeye çalışmışlardır. Değerlendirme kriteri olarak; R^2 , tamlık, kusurlu modül tamlığı, kusurlu modül hatasızlığı kullanılmıştır. Çalışma sonucunda; endüstriyel ihtiyaçları karşılamak üzere lojistik regresyon ve çapraz geçerleme (cross-validation) tabanlı bir yöntem önermişler ve lojistik regresyonun kusur kestirimindeki etkinliğini göstermişlerdir. Az sayıda verinin olduğu durum için, çapraz geçerlemenin uygulanmasını tavsiye etmişlerdir.

Menzies ve arkadaşları (2004a), PROMISE havuzundaki veri kümeleri üzerinde Naive Bayes yöntemiyle metot seviyesindeki metrikleri kullanarak kusur kestiriciler üzerinde çalışma gerçekleştirmişlerdir. Değerlendirme kriteri olarak; PD (probability of detection) veya başka ifadeyle kesinlik (specificity), PF (probability of false alarm) veya başka ifadeyle duyarlık (sensitivity) ve hassasiyet (precision) kriterlerini kullanmışlardır. Bu çalışmada; doğruluğun

uygun bir değerlendirme kriteri olmadığı, Naive Bayes'in J48'den daha iyi sonuç verdiği, öğrenme algoritmalarının 200-300 veri örneği ile kestiricileri oluşturabildiği raporlanmıştır. Ayrıca; KC1 veri kümesi için kusur tespit yüzdesinin (PD) %55 olduğu, bu değerin Fagan kod incelemeleri için %35-%65 arasında değiştiği, endüstriyel incelemelerde %13-30% arasında olduğu ifade edilmiştir.

Khoshgoftaar ve Seliya (2004), büyük bir telekom sisteminde lojistik regresyon, senaryo tabanlı usavurum, CART (classification and regression trees), S-PLUS ile ağaç tabanlı sınıflandırma, Sprint-Sliq, C4.5, Treedisc yöntemleriyle 24 ürün ve 4 yürütme anı metriği kullanarak, kusur kestirimi üzerinde çalışmışlardır. Değerlendirme kriteri olarak, yanlış sınıflamanın beklenen maliyeti (expected cost of misclassification) kriterini kullanmışlardır. Çalışmada; farklı sistem sürümleri için modellerin kestirim performansları oldukça farklı çıkmıştır. Bu bilgiye göre; yazılım mühendisliği alanında kestirim modelleri, verinin ve sistemin karakteristiklerinden etkinleniyor sonucu çıkarılabilir.

Wang ve arkadaşları (2004), C dili ile geliştirilmiş 480 yöntemden oluşan büyük bir telekom sisteminde, "Kümeleme Genetik Algoritması" (Clustering Genetic Algorithm-CGA) ile MATRIX analizöründen elde edilen 7 adet ürün metriğini kullanarak yapay sinir ağı tabanlı kalite kestirim modellerinin anlaşılırlığını arttırmak üzere bir çalışma yapmışlardır. Değerlendirme kriteri olarak doğruluk kullanılmış ve yapay sinir ağlarından kurallar çıkarabilmek için CGA algoritmasına başvurulmuştur. CGA'nın kural kümesi ile yapılan kestirimin doğruluğu, yapay sinir ağı ile elde edilen doğruluk değerinden daha düşük olmasına rağmen, sonuçlar daha anlaşılabilirdir.

Mahaweerawat ve arkadaşları (2004), farklı web sayfalarından elde ettiği 3,000 C++ sınıfı üzerinde öncelikle çok katmanlı algılayıcı ile modüllerin kusur eğilimli olup olmadığını, ardından RBF ile kusur tipini belirleme üzerinde çalışmışlardır. Sınıf seviyesindeki metrikler "Understand for C++" aracı ile toplanmış ve değerlendirme kriteri olarak; doğruluk, Tip-I hatası, Tip-II hatası, inceleme, başarılan kalite ve kayıp inceleme kriterleri kullanılmıştır. Çalışmada kullanılan sınıflar, farklı geliştiriciler tarafından geliştirildiği için genel bir kestirim modeli tasarlamak için yararlı olacağını düşünmüşlerdir. Ancak kestirim modellerinin bu kadar genel tasarlanamayacağı, proje farklılıklarına göre modellerin değişebileceği düşünülürse, çalışmanın veri kümesi açısından sorunlar taşıdığı söylenebilir. Çalışmada; bir sınıfın kusurluluğunu %90'dan daha fazla doğrulukla kestirebildiklerini, başarılan kalitenin %91.53, inceleme değerinin %59.55, kayıp incelemenin %3.51, Tip-I hatasının %5.32, Tip-II hatasının %2.09 olduğu raporlanmıştır. Ayrıca, kusurlu sınıfların sadece %2.09'unu tespit

edemediklerini ifade etmişlerdir.

Menzies ve Di Stefano (2004b), NASA açık veri kümeleri üzerinde lineer standart regresyon (LSR), model ağaçları, karar ağaçları, ROCKY ve Delphi detektörleriyle metot seviyesindeki metrikleri kullanarak kusur kestirimi üzerinde çalışmışlardır. Delphi detektörleri; çevrimsel karmaşıklığın 10'a eşit ya da daha büyük, McCabe tasarım karmaşıklığının (design complexity) 4'e eşit ya da daha büyük olduğu modüllerin kusurlu olduğunu öngörmektedir { V(g)>=10, iv(g)>=4, V(g)>=10 V iv(g)>=4}. Bu çalışmada; detektörlerin çoklu değerlendirme kriterleriyle belirlenmesi gerektiği, ROCKY algoritmasının McCabe metrikleriyle kullanılmasının önerildiği, ROCKY'den sonra Delphi detektörlerinin tercih edilebileceği ifade edilmiştir. Değerlendirme kriteri bu çalışmada; doğruluk, duyarlık, kesinlik, hassasiyet, çaba olarak seçilmiştir.

Kaminsky ve Boetticher (2004), NASA veri kümelerinden KC2 veri kümesi üzerinde genetik algoritmayla 21 adet metot seviyesindeki metrikle kusur kestirimi çalışması yürütmüşlerdir. Bu çalışmada, veri eşitlemenin (data equalization) etkilerini inceleyerek değerlendirme kriteri olarak T-testi kullanmışlardır. Veri açlığı çeken alanları (data starved domains), 2 gruba ayırmışlardır:

- *Belirgin (explicit):* Proje maliyet tahminleme alanı bu gruba girmektedir. Şirketler iyi bir model buldukları zaman pazar büyüklüklerini kaybetmemek için veri kümelerini paylaşmak istemezler.
- *Örtülü (Implicit):* Kusur kestirimi araştırma alanı bu gruba girmektedir. Yeterince veri noktası vardır ancak veri kümeleri yüksek seviyede dengesizdir (imbalanced).

Bu çalışmada, veri eşitlemenin genetik programlama uygulandığı zaman oldukça yararlı olduğu tespit edilmiştir. Veri kümesinin büyümesi nedeniyle, genetik programlamanın eğitim sürecinin yavaşladığı ise bir dezavantaj olarak ifade edilebilir. Eşitlenmemiş veri kümesinde ortalama uygunluk değeri (average fitness value) 12 iken eşitlenmiş veri kümesinde 81.4 olarak hesaplanmış ve T-test ile bu farkın oldukça önemli olduğu saptanmıştır.

Kanmani ve arkadaşları (2004), Hindistan'daki Pondicherry Mühendislik Kolejinde gerçekleştirilen öğrenci projeleri üzerinde GRNN yöntemiyle 64 adet sınıf seviyesindeki metrikle kalite kestirimi çalışması yürütmüşlerdir. Özellik seçiminde, temel bileşen analizi ve değerlendirme kriteri olarak korelasyon katsayısı (r), R^2, ortalama karesel hata, ortalama mutlak hata, maksimum mutlak hata, minimum mutlak hata kriterleri kullanılmıştır.

Çalışmaya göre; nesneye yönelik metriklerle GRNN iyi sonuçlar üretmiştir.

Zhong ve arkadaşları (2004), NASA'nın KC2 veri kümesi üzerinde K-ortalama kümeleme ve Neural-Gas kümeleme algoritmalarıyla uzman tabanlı yazılım kalite kestirimi çalışması gerçekleştirmişlerdir. Kümeleme kalitesini değerlendirmek için; ortalama karesel hata (mean squared error-MSE), ortalama saflık (average purity) ve zaman kullanılmıştır. Uzmanın etiketleme performansını değerlendirmek için; "fpr" (kusurlu olmayan modüllerin kusurlu olarak kestirilme yüzdesi), "fnr" (kusurlu modüllerin kusurlu değil şeklinde kestirilme yüzdesi) ve "yanlış sınıflandırma oranı" (yanlış sınıflanan modüllerin yüzdesi) parametreleri kullanılmıştır. Ortalama saflık ise ilgili küme içinde, baskın kategorinin yüzdesi olarak ifade edilebilmektedir. Bu çalışmaya göre; Neural-Gas, MSE parametresine göre çok daha iyi sonuçlar üretmişken ortalama saflık değerine göre k-ortalama kümelemeden biraz daha iyi sonuç vermiştir. K-ortalama kümeleme, Neural-Gas algoritmasından daha hızlı çalışmakta olup, Neural-Gas'ın yanlış sınıflandırma oranı K-ortalama kümeleme'den biraz daha iyidir. Uzmanların görüşlerine göre, Neural-Gas ile etiketleme yapmak daha kolaytır. Ayrıca; sınıflayıcı birlikteliği (ensemble of classifiers) ile elde edilen gürültülü modüllerin bu çalışmada belirlenen gürültülü modüllerle benzerlik taşıdığı ifade edilmiştir.

2.6.4 2005-2007 Yılları Arasındaki Çalışmalar

Xing ve arkadaşları (2005), medikal görüntüleme yazılımı üzerinde Destek Vektör Makineleriyle (Support Vector Machines-SVM) 11 metot seviyesinde karmaşıklık metriğini kullanarak (Halstead, McCabe, Jensen program uzunluğu ve Belady bant genişliği) erken kalite kestirimi üzerine çalışmışlardır. Tip-I ve Tip-II hatası, değerlendirme kriteri olarak kullanılmıştır. Bu çalışmaya göre; SVM kuadratik diskriminant analizinden ve sınıflama ağacından daha iyi sonuç üretmiştir. Ancak açık veri kümeleri üzerinde 2007 yılında yapılan çalışmalarda, SVM'nin performansının oldukça kötü olduğu raporlanmıştır. Sonraki bölümlerde bu kapsamda bilgi verilecektir.

Koru ve Liu (2005a), NASA açık veri kümeleri üzerinde J48 ve KStar algoritmalarıyla metot ve sınıf seviyesindeki metrikleri kullanarak, kusur kestiriminde modül büyüklüğünün etkisi üzerine araştırma yürütmüşler, F-ölçüm değerini değerlendirme kriteri olarak kullanılmışlardır. Bu çalışma sonucunda; kusur tespitinin büyük bileşenlerde gerçekleştirilmesi gerektiği ifade edilmiş ve küçük bileşenlerde metot seviyesi yerine sınıf seviyesindeki metriklerin kullanılması önerilmiştir. WEKA içerisindeki farklı makine öğrenmesi algoritmalarının analiz sonuçlarının ne umut vaat edici ne de cesaret kırıcı

olduğunu raporlamışlardır. Küçük modüllerde ölçüm değerlerinin 0'a yakın olması, bu değerlerle makine öğrenmesi algoritmalarının yüksek performans üretmesini engellemekte ve büyük modüllerin sayısının az olduğu durumda, kestirim modellerinin performansı iyi sonuçlar vermemektedir. Bu çalışmaya göre; Bayes Ağları, Yapay Sinir Ağları ve Destek Vektör Makineleri kötü performans sunmuştur. Küçük modüllerin yer aldığı veri kümelerindeki performansların oldukça düşük olduğu raporlanmıştır. Çalışma sonucunda kusur kestirimi için, J48 algoritmasının kullanılması önerilmiştir.

Khoshgoftaar ve arkadaşları (2005), kablosuz ürünlerin konfigürasyonunu sağlayan iki gömülü yazılım üzerinde, C4.5 karar ağacı, diskriminant analizi, senaryo tabanlı usavurum ve lojistik regresyon teknikleriyle beş adet dosya metriğini kullanarak üç gruplu yazılım kalite sınıflandırması konusunda çalışmışlardır. Değerlendirme kriteri olarak, yanlış sınıflandırmanın beklenen maliyeti kriterini kullanmışlardır. Çalışma sonunda; üç gruplu sınıf etiketlerinin (yüksek-orta-düşük / kırmızı-sarı-yeşil) umut verici sonuçlar ürettiği raporlanmıştır. Ayrıca; önerdikleri "üç gruplu sınıflandırma" tekniğiyle, normalde sadece iki gruba sınıflandırma yapabilen sınıflayıcıların, tasarımında değişiklik yapmadan üç gruba sınıflandırma yapabilecekleri ifade edilmiştir.

Koru ve Liu (2005b), NASA açık veri kümeleri üzerinde J48, K-Star, Rastgele Orman teknikleriyle metot ve sınıf seviyesindeki metrikleri kullanarak etkin kusur kestirimi modelleri inşa etme konusunda çalışmışlardır. Metot seviyesindeki 21 metriği sınıf seviyesindeki metriklere dönüştürebilmek için; her metriğin minimum, maksimum, ortalama, toplam değerleri hesaplanmış ve ilgili sınıfın metriği haline gelmiştir. Bu sayede, 21 metot seviyesindeki metrik 84 (21 * 4 = 84) sınıf seviyesindeki metriğe dönüştürülmüştür. Değerlendirme kriteri olarak, F-ölçüm kullanılmıştır. Çalışmada; sadece J48 algoritmasında değil, K-Star ve Rastgele Orman algoritmalarında da büyük modüllerin daha yüksek F-ölçüm değerine sahip olduğu tespit edilmiştir. Sınıf seviyesindeki metriklerle elde edilen F-ölçüm değeri 0.65 iken metot seviyesindeki metriklerle 0.40 değeri elde edildiğinden, sınıf seviyesindeki metriklerin performansı arttırdığı ifade edilmiştir.

Challagulla ve arkadaşları (2005), NASA açık veri kümeleri üzerinde lineer regresyon, pace regresyonu, destek vektör regresyonu, sürekli amaç alanı için sinir ağı (neural network for contionus goal field), destek vektör lojistik regresyonu, ayrık amaç alanı için sinir ağı, Naive Bayes, örnek tabanlı öğrenme (instance based learning-IBL), J48 ve 1-R teknikleriyle metot seviyesindeki metrikleri kullanarak kusur kestirimi çalışmaları yürütmüşlerdir. Değerlendirme kriteri olarak; ortalama mutlak hata kullanılmıştır. Çalışma sonucunda; tüm veri kümelerinde

en başarılı olacak bir yöntemin bulunamadığı, IBR ve 1R'nin doğruluk parametresine göre diğerlerinden daha iyi sonuçlar ürettiği, ön işlemede kullanılan temel bileşen analizinin ek bir avantaj sağlamadığı, büyüklük ve karmaşıklık metriklerinin kusur kestirimi için yeterli olmadığı, bu metriklerin arasındaki bağlılıkları ifade edecek metriklere ihtiyacın olduğu raporlanmıştır.

Gyimothy ve arkadaşları (2005), Mozilla açık kaynaklı projesi üzerinde lojistik ve lineer regresyon, karar ağaçları ve sinir ağları teknikleriyle sınıf seviyesindeki metrikleri kullanarak, kusur kestirimi için nesneye yönelik metriklerin geçerlenmesi üzerine çalışmışlardır. Değerlendirme kriteri olarak; tamlık, hatasızlık, hassasiyet kriterlerini seçmişlerdir. Çalışma sonucunda; dört yöntemin benzer sonuçlar ürettiği, kusur eğilimliliği için nesneler arasındaki bağlılık metriğinin (CBO) çok yararlı olduğu, kod satır sayısının yararlı olduğu ancak çok değişkenli modellerin çok daha fazla yarar sağladığı, metotların kohezyon eksikliği (LCOM) metriğinin hatasızlığının yüksek ancak tamlığının düşük olduğu, kalıtım ağacı derinliği (DIT) metriğinin güvenilir olmadığı, çocuklarının sayısı (number of children-NOC) metriğinin kullanılmaması gerektiği raporlanmıştır.

Ostrand ve arkadaşları (2005), iki endüstriyel sistem üzerinde "negatif binomlu regresyon modeli" 'yle (negative binomial regression model) programlama dili, dosya yaşı, dosya değişim durumu gibi metrikleri kullanarak kusurların sayısını ve yerini kestirmeye çalışmışlardır. Değerlendirme kriteri olarak, doğruluk seçilmiştir. Çalışma sonucunda; modelin yararlı sonuçlar verdiği, genel performansın %84, basitleştirilmiş modelin performansının %73 doğrulukta olduğu raporlanmıştır.

Tomaszewski ve arkadaşları (2005), büyük bir telekom sistemi üzerinde tek ve çok değişkenli lineer regresyon teknikleriyle metot ve sınıf seviyesindeki metrikleri kullanarak, erken kusur kestirimin doğruluğu üzerinde çalışmışlar, değerlendirme kriteri olarak "determinasyon katsayısı" nı (R^2) tercih etmişlerdir. Sistem gerçeklendikten sonra kurulan modellerin, gerçeklenmeden önce kurulanlara göre %34 daha doğru olduğu raporlanmıştır. Değişen sınıfların büyüklüğünü gösteren metrik (size of changed classes-ChgSize), sistem gerçekleştirildikten sonra hesaplandığından, performansı iyileştirdiği ifade edilmiştir. ChgSize metriğinin kullanılmadığı durumda, sisteme ilişkin elde edilen gerçekleme öncesi ve sonrası kestirim model performansları hemen hemen aynıdır. Tek değişkenli modellerin doğruluğu çok değişkenlilerden %5 daha düşük olarak hesaplanmış ancak bu farkın çok önemli olmadığı ifade edilmiştir. Tek değişkenli modellerin daha kararlı olması ve "çoklu bağlanım" (multicollinearity) özelliği taşımamaları nedeniyle, bu çalışma sonunda bu

modellerin kullanılması önerilmiştir. Çoklu bağlanım, değişkenler arasındaki korelasyon ilişkisinin olduğu durumda söz konusu olan bir özelliktir ve temel bileşen analizi gibi yöntemlerle yok edilebilmektedir.

Hassan ve Holt (2005), 6 açık kaynaklı proje üzerinde kusur eğilimliliği en yüksek olan ilk 10 bileşeni analiz etmeye çalışmışlardır. Metrikler olarak; değiştirilme sıklığı, büyüklük, son zamanlarda değiştirilip değiştirilmediği bilgilerini kullanmışlardır. Değerlendirme kriteri olarak; isabet oranı (hit rate) ve ortalama kestirim yaşı (average prediction age-APA) kriterlerini kullanmışlardır. Bu metrik, bu çalışmada önerilmiştir. En sık değişen (most frequently modified-MFM), en güncel değişen (most recently modified-MRM), en sık düzeltilen (most frequently fixed-MFF), en güncel düzeltilen (most recently fixed-MRF) teknikleri bu çalışma içerisinde önerilmiş ve kullanılmıştır. Tamponlama (caching) sistemlerinin performansını ölçmek üzere kullanılan isabet oranı, önerilen sezgisel tekniklerin performansını belirlemek için kullanılmıştır. MFM ve MFF, diğer yöntemlere göre daha başarılı olmuştur. APA değerinin daha büyük olması, daha iyi performans sunduğunu göstermektedir.

Ma ve arkadaşları (2006), NASA açık veri kümeleri üzerinde çok sayıda makine öğrenmesi algoritmasının performansını incelemiş ve bu kapsamda metot seviyesindeki metrikleri kullanmıştır. Değerlendirme kriteri olarak; G-ortalama1, G-ortalama2, F-ölçüm kriterlerini kullanmışlardır. Ayrıca; Rastgele Orman ve Dengeli Rastgele Orman teknikleri arasında kıyaslama yapmak için "Alıcı Çalışma Karakteristikleri" (Receiver Operating Characteristics) kısaca ROC eğrisi altında kalan alan kullanılmıştır. Çalışmada; Lojistik Regresyon, Diskriminant Analizi, Sınıflandırma Ağacı, Boosting, Çekirdek Yoğunluğu (KernelDensity), Naive Bayes, J48, IBk, Oylanmış Algılayıcı (Voted Perceptron), VF1, Hyperpipes, ROCKY, Rastgele Orman teknikleri üzerinde çalışılmıştır. G-ortalama1, G-ortalama2, F-ölçüm kriterlerine göre algoritmalar sıralandığında; üç kritere göre de ilk üçte yer alan algoritmanın en iyi kusur kestirimini gerçekleştirdiği değerlendirilmiştir. Rastgele Orman tekniğinin büyük veri kümelerinde (JM1 ve PC1) etkin çalıştığı ve en umut veren teknik olduğu raporlanmıştır. Literatürde birkaç kez kullanılmış olmasına rağmen; RuleSet, Boosting, tekli ağaç sınıflayıcılarının kalite kestirimi için uygun olmadığı açıklanmıştır. Çalışmada; PD ve doğruluk parametrelerinin düşük değere sahip olduğu sınıflayıcıların kesinlikle önerilmediği, dengeli rastgele ormanların geleneksel rastgele ormanlardan daha iyi performans verdiği, ancak bu tekniğin hazır araçlarla gelmediği, dengesiz veri kümelerinde değerlendirme kriteri olarak G-ortalama ve F-ölçüm'ün daha uygun olduğu ifade edilmiştir.

Challagulla ve arkadaşları (2006), NASA açık veri kümeleri ve Bellek Tabanlı Usavurum (Memory Based Reasoning-MBR) tekniğiyle 21 metot seviyesindeki metriği kullanarak kusur veri analizi üzerinde çalışmışlardır. Değerlendirme için; PD, PF ve doğruluk parametreleri kullanılmıştır. MBR, geçmiş deneyimlerden elde edilen bilgileri yeniden kullanarak yeni durumların çözülmesini sağlayan bir sınıflayıcı tekniğidir. Bu çalışmada sunulan çerçeve sayesinde, kullanıcı en yüksek kestirimi sağlayacak olan MBR konfigürasyonunu seçebilmektedir.

Khoshgoftaar ve arkadaşları (2006), büyük bir telekom sistemi üzerinde senaryo tabanlı usavurum tekniğiyle 24 ürün ve 4 yürütme anı metriğini kullanarak kusurların kestirimini sağlamaya çalışmışlardır. Değerlendirme kriteri olarak; ortalama mutlak hata ve ortalama bağıl hata kullanılmıştır. Çalışma sonucunda; CBR modellerinin çoklu lineer regresyondan daha iyi performans verdiği, Mahalanobis uzaklık fonksiyonu ile CBR'nin birlikte kullanıldığı durumda en iyi performansın elde edildiği, temel bileşen analizi ile metrik korelasyonları yok edildiği zaman Şehir Blok (City Blok) uzaklığının Mahalanobis uzaklığından doğruluğu daha yüksek sonuçlar ürettiği, korelasyon tabanlı özellik seçimi (correlation based feature selection) ve adımlı regresyon model seçimi (stepwise regression model selection) tekniklerinin kusur kestirim performansını iyileştirmediği raporlanmıştır.

Nikora ve Munson (2006), NASA'nın Jet İtici Güç Laboratuvarı'ndaki (Jet Propulsion Laboratory) Görev Veri Sistemi (Mission Data System) üzerinde metot seviyesindeki metriklerle yüksek kalitede kusur kestiricileri üzerinde çalışmışlardır. Şimdiye kadar yapılan çalışmalarda, bir yazılım kusurunun ne olduğunu tam olarak ortaya koyacak kantitatif bir tanım bulunmuyordu. Örneğin bazı araştırmacılar, modüllere karşı yazılmış "Farklılık Raporlarının" (Discrepancy Reports-DR) sayısını kusur sayısı olarak ele almıştır. Bu raporlar, ClearQuest gibi değişiklik yönetim araçlarında otomatik hazırlanabilmektedir. Ancak DR raporlarının isterlerden sapmaları gösterdiği ve bu sapmaların kusur yerine arıza (failure) anlamına geldiği düşünülürse doğru bir yöntemin uygulanmadığı söylenebilir. Bu çalışmada; kusur tanımı için kuralların belirli olduğu bir çerçeve inşa edilmiş ve iki sürüm arasındaki sembol (token) farklılıkları dikkate alınarak kusur sayılarının belirlenmesinin daha etkin kusur kestiriciler oluşturabileceği gösterilmiştir. Örneğin; a = b + c olan bir kod satırı için semboller kümesi B1={ <a>, <=>, <b>, <+>, <c>} şeklinde yazılabilir. Bu satırın bir sonraki sürümde, a = b - c haline geldiğini düşünürsek sembol farklarını gösteren küme B1-B2={<+>, <->}şeklinde ifade edilecektir. Bu durumda, iki sürüm arasında bir kusur olduğu, + sembolü yerine – sembolünün kullanıldığı söylenebilir.

Zhou ve Leung (2006), NASA'nın KC1 veri kümesinde lojistik regresyon, Naive Bayes, Rastgele Orman, genelleştirme ile en yakın komşu (nearest neighbor with generalization) teknikleriyle sınıf seviyesindeki metrikleri kullanarak yüksek ve düşük öncelikli kusurların kestirilmesi konusunda çalışma yürütmüşlerdir. Değerlendirme kriteri olarak; tamlık, hatasızlık, hassasiyet kriterlerini kullanmışlardır. Çalışma sonucunda; düşük öncelikli kusurların yüksek öncelikli kusurlara göre daha yüksek performansla kestirildiği, kalıtım ağacı derinliğinin kusur kestirimi için önemli bir metrik olmadığı, yüksek öncelikli kusur kestiriminde NOC metriğinin önemsiz olduğu ve diğer CK metriklerinin kusur kestiriminde yarar sağladığı ifade edilmiştir.

Mertik ve arkadaşları (2006), NASA veri kümeleri üzerinde Budanmış ve Budanmamış C4.5 (Prunned ve Unprunned C4.5), çoklu yöntem (multimethod), RBF çekirdeği kullanan SVM, lineer çekirdek kullanan SVM teknikleriyle metot seviyesindeki metrikleri kullanarak yazılım kalite kestirimi üzerinde çalışmışlardır. Çoklu yöntem veri madenciliği aracı Maribor Üniversitesinde geliştirilmiş bir araçtır ve sonuçları ağaçlar şeklinde göstermektedir. Araç içerisinde birçok yöntem birleştirilmiş haldedir ve doğruluk kriterine göre sunulan performanslar oldukça yüksektir. Çoklu yöntem ağacındaki düğümler farklı yöntem ve operatörleri içermekte olup bu düğüm popülasyonunun ortak hedefi sınıflandırma yeteneklerini arttırmaktır.

Boetticher (2006), NASA açık veri kümeleri üzerinde J48 ve Naive Bayes teknikleriyle metot seviyesindeki metrikleri kullanarak veri kümelerinin yazılım mühendisliğindeki etkilerini incelemiş ve değerlendirme kriteri olarak; PD, PF ve doğruluk kriterlerine başvurmuştur. Her veri kümesi üç bölüme ayrılmıştır: eğitim kümesi, iyi (nice) komşulardan oluşan test kümesi ve kötü (nasty) komşulardan oluşan test kümesidir. İyi komşular, aynı sınıfa en yakın olan komşular olup kötü komşular farklı sınıflarda yer alan komşulardır. Çalışma sonucunda; iyi komşuların bulunduğu test kümesi için doğruluk oranın %94, kötü komşuların bulunduğu test kümesi için doğruluk oranının %20 olduğu, 10 katlı (ten-fold) çapraz geçerlemenin yetersizliği ve veri kümelerinin zorluğunu ölçmek üzere bazı yeni metriklerin önerildiği raporlanmıştır.

Bibi ve arkadaşları (2006), Finlandiya'daki ticari bir bankaya ait Pekka isimli veri kümesinde, "Sınıflandırma Yoluyla Regresyon" (Regression via Classification-RvC) tekniğiyle farklı kantitatif değişkenleri kullanarak (disk kullanımı, işlemci kullanımı, kullanıcı sayısı, belge kalitesi) kusur kestirimi çalışması yürütmüşlerdir. Değerlendirme kriteri olarak; ortalama mutlak hata ve doğruluk kriterlerini kullanmışlardır. Eğiticili öğrenme; regresyon ve sınıflama

olarak iki gruba ayrılmaktadır. Bağımlı değişkenin gerçek (real) değerlere sahip olması durumunda, regresyon modellerinin kullanılması gerekmektedir. Tek başına sınıflandırma, çıkacak kusurların sayısının kestiriminde kullanılamayacağından ve regresyon modellerinin yorumlanmasındaki güçlük nedeniyle, açıklama gerektiren regresyon problemleri için RvC iyi bir çözümdür. RvC yönteminde; ilk önce sayısal değerler sınıflandırma için ayrık değerlere dönüştürülür ve ardından kestirilen sınıf etiketleri son olarak sayısal değerlere karşı düşürülür. Bu çalışma sonucunda, RvC'nin regresyon modellerinde anlaşılırlığı arttırmak için kullanılabileceği ifade edilmiştir.

2.6.5 2007 Yılındaki Çalışmalar

Gao ve Khoshgoftaar (2007), kablosuz telekom ürünlerinin konfigürasyonunu gerçekleştiren iki gömülü yazılım uygulaması üzerinde Poisson regresyon modeli, sıfırla şişirilmiş Poisson modeli, negatif binomlu regresyon modeli, sıfırla şişirilmiş negatif binomlu model, Hurdle regresyon (HP1, HP2, HNB1, HNB2) teknikleriyle 5 adet dosya seviyesindeki metriği kullanarak yazılım kusur kestirimi üzerine çalışmışlardır. Değerlendirme için; Pearson'ın chi square ölçümü, hipotez testi, bilgi kriteri (information criteria), ortalama mutlak hata (AAE), ortalama bağıl hata (ARE) kriterlerini kullanmışlardır. Çalışma sonucunda sıfırla şişirilmiş negatif binomlu modelin ve Hurdle modellerinin; hipotez testi, bilgi kriteri ve Pearson'ın chi square ölçümlerine göre daha iyi sonuç verdiği raporlanmıştır. AAE ve ARE ölçümlerine göre; kestirim doğruluğu açısından test verileri üzerinde HP2 diğerlerinden daha iyi performans vermiştir.

Li ve Reformat (2007), NASA açık veri kümesi olan JM1 üzerinde SimBoost isimli bir teknikle metot seviyesindeki metrikleri kullanarak ve doğruluk kriterine göre yazılım kusur kestirimi için uygulamada yararlı bir yöntem üzerine çalışmışlardır. SimBoost yöntemi bu çalışmada önerilmiş olup sınıflandırma için bulanık etiketler tavsiye edilmiştir. SimBoost'un performansı başlangıçta kötü çıkmış, dengeli veri kümeleri ile denenince yine kötü performans elde edilmiş ve en sonunda bulanık etiketlerle çalışılmıştır.

Mahaweerawat ve arkadaşları (2007), kaynağı açıklanmayan bir veri kümesi üzerinde önce "kendi organize olan harita kümelemesi" (self organizing map clustering) ve ardından RBF ağını kullanarak kusur kestirimi üzerine çalışmışlardır. Metot seviyesindeki metriklerle çalışılmış, değerlendirme kriteri olarak doğruluk ve "mutlak artanın ortalaması" (mean of absolute residual-MAR) kullanılmıştır. MAR değeri, gerçek kusur sayısından kestirilen kusur sayısının çıkarılıp mutlak değerinin alınmasıyla elde edilmektedir. Bu çalışmada, test

verisinden %93 doğruluk elde edildiği raporlanmıştır ancak bu tür çalışmalarda doğruluk kriteri doğru bir parametre değildir.

Menzies ve arkadaşları (2007a), NASA açık veri kümeleri üzerinde Naive Bayes algoritmasıyla metot seviyesindeki metrikleri kullanarak kusur kestiriciler üzerine çalışmışlardır. Değerlendirme kriteri olarak; PD, PF ve denge (balance) kriterlerini tercih etmişlerdir. Çalışma sonunda yöntem olarak; metriklerin ilk aşamada logaritmasının alınmasını ve ardından Naive Bayes uygulanmasını tavsiye etmişler, basit eşik seviyelerinin kusur kestiriminde kullanılmamasını önermişlerdir. Ayrıca; Naive Bayes'in J48 algoritmasından daha iyi performans verdiği, kusur kestiricilerin eldeki tüm özelliklerle oluşturulup ardından en uygun kümeyi bulmak için farklı tekniklerin uygulanması gerektiği, en iyi kestiricilerin veri kümesine göre değiştiği, sadece bir veri kümesinin ve sadece bir öğrenme algoritmasının kusur kestirimi için yeterli olmadığı, çok sayıda deneyin gerekli olduğunu ifade edilmiştir.

Zhang ve Zhang (2007), Menzies ve arkadaşlarının (2007a) çalışmasında kullandıkları pd ve pf değerlendirme kriterlerinin yetersiz olduğunu ifade etmişler ve hassasiyetin (precision) bu çalışma için oldukça düşük düzeylerde kaldığını ortaya koymuşlardır. Bu nedenle, Menzies ve arkadaşlarının önerdiği modelin uygulamada yarar sağlamayacağını belirtmişlerdir.

Menzies ve arkadaşları (2007b), Zhang ve Zhang'in (2007) eleştirilerine yanıt vermek üzere hassasiyetin yazılım mühendisliği problemlerinde yararlı bir kriter olmadığını, hassasiyetin düşük olmasına rağmen ortaya konulan modellerin birçok model bağlamında yararlı sonuçlar ürettiğini ifade etmişlerdir. İdealde hassasiyet kriterinin de çok yüksek değerlere sahip olması gerektiği ancak uygulamada bu durumun gerçekleşemediğini, bu tür düşük hassasiyetli modellerle konferansda en iyi bildiri ödülü kazanabildiği, bu tür modellerin önemli yararlar sağladığı örneklerle açıklanmıştır.

Ostrand ve arkadaşları (2007), bakım destek yazılımı üzerinde dosya büyüklüğü, dosya durumu, dosya üzerinde kaç değişiklik olduğu, programlama dili gibi metrikleri kullanarak negatif binomlu regresyon modeli, kod satır sayısı modeli, verilerden katsayıların elde edildiği ön tanımlı model teknikleriyle kusur eğilimli modülleri elde etmeye çalışmışlardır. Bu veri kümesinde; doğruluk kriterine göre negatif binomlu regresyon tekniğinin yarar sağladığı raporlanmıştır.

Yang ve arkadaşları (2007), yapay bir veri kümesinde "Bulanık Kendi Kendine Adapte Olabilen Öğrenme Kontrol Ağı" (Fuzzy Self-Adaptation Learning Control Network-

FALCON) tekniğiyle bazı metrikleri (karmaşıklık, yeniden kullanım, kalıtım ağacının derinliği) girdi olarak kullanıp çıkışta güvenilirlik ve etkinliği ölçmeye çalışmışlardır. Bu çalışmada, FALCON yöntemi önerilmiş ancak deneysel olarak geçerliliğini sağlayacak gerçek bir veri kümesi kullanılmamış olup değerlendirme kriteri bilgisine yer verilmemiştir. FALCON, bulanık sinir ağlarının özel bir biçimidir. Önerdikleri modelin birçok kalite özelliğini (güvenilirlik, performans, bakım yapılabilirlik) ölçebileceğini ifade etmişler ancak gerçek veri kümeleri ile bu deneyleri gerçekleştirmemişlerdir. Bu açıdan, çalışmalarının çok erken aşamalarda olduğu söylenebilir.

Pai ve Dugan (2007), NASA veri kümeleri üzerinde lineer regresyon, Poisson regresyonu ve lojistik regresyon teknikleriyle Bayes Ağları'ndaki düğümlerin "koşullu olasılık yoğunluk" 'larını (conditional probability density) hesaplayarak modüllerin kusur eğilimliliğini belirlemeye çalışmışlardır. Sınıf seviyesindeki CK metrikleri ve kod satır sayısı metriği kullanılmış olup değerlendirme kriteri olarak duyarlık, kesinlik, hassasiyet, yanlış pozitif, yanlış negatif kriterleri seçilmiştir. Çalışma sonunda; CK metriklerinden sınıftaki metot sayısı (weighted method count-WMC), nesneler arasındaki bağlılık (coupling between object classes-CBO), bir sınıf için yanıt (response for a class-RFC), kod satır sayısı metriklerinin kusur eğilimliliği belirleme için çok önemli olduğu tespit edilmiştir. Poisson regresyon kullanıldığı zaman, CK metriklerinden kalıtım ağacının derinliği (DIT) ve çocuklarının sayısı (NOC) metriklerinin önemli olmadığı, metotlardaki kohezyon eksikliği (LCOM) metriğinin önemli olduğu saptanmıştır. LCOM değerleri içermeyen lineer model, LCOM içeren Poisson modelinden daha iyi sonuçlar üretmiştir. Ayrıca, Bayes modelinin süreç ve ürün metriklerini birleştirebileceği ifade edilerek bu yönde bir yaklaşım önerilmiştir.

Wang ve arkadaşları (2007), iki büyük telekom sistemi üzerinde genetik algoritma tabanlı bir modelle 18 metot seviyesinde metriği ve 14 dosya seviyesinde metriği kullanarak kalite kestirimi üzerine çalışmışlardır. Değerlendirme kriteri olarak; Tip-I hatası, Tip-II hatası, yanlış sınıflandırma oranı kullanılmıştır. Kümeleme ve özellik seçimi kriterleri genetik algoritmanın uygunluk (fitness) fonksiyonuna tümleştirilmiştir. Çalışma sonunda; önerilen modelin S-PLUS ve TreeDisc tabanlı yöntemlerden daha iyi olduğu, 5'den az sayıda üyesi olan kümelerin aykırı değerler (outlier) olduğu düşünülerek veri kümesinden silindiği, saflık değeri %80'den fazla olan kümelerin etiketlerinin baskın sınıfla kolayca etiketlendiği raporlanmıştır.

Seliya ve Khoshgoftaar (2007a), NASA açık veri kümelerinden JM1 kümesini eğitim kümesi, KC1, KC2, KC3 kümelerini test kümesi olarak kullanıp Beklenti Maksimizasyonu

(Expectation-Maximization-EM) tekniğiyle 13 metot seviyesindeki (Halstead, McCabe, kod satır sayısı) metriği kullanarak kusur kestirimi üzerine çalışmışlardır. Bu çalışmanın önemi, sınırlı (limited) sayıda kusur verisinin bulunduğu durumlar için yeni bir yaklaşım önermesidir. Tüm modüllerin bir önceki sürümlerindeki kusur verileri mevcut değilse, çok az sayıda modüle ilişkin kusur bilgisi varsa, etiketi (kusur bilgisi) olmayan modüllerden de yararlanarak bir kusur kestirim modeli ortaya koymak yarar sağlayacaktır. Aksi halde çok az sayıda etiketli veriyle eğiticili öğrenme modeli kurgulamak, performansın oldukça düşük olmasına neden olacaktır. Etiketsiz verilerden de yararlanarak sınıflandırma işlemini gerçekleştirmek literatürde "yarı-eğiticili sınıflandırma" (semi-supervised classification) olarak bilinmektedir. Bu çalışmada, değerlendirme kriteri olarak Tip-I hatası, Tip-II hatası ve yanlış sınıflandırma oranı kullanılmıştır. EM algoritması ilk olarak sınıf etiketi mevcut olmayan modülleri etiketlemek için kullanılır ve bu adımdan sonra tüm modüller dikkate alınarak EM algoritması ile kestirim modeli ortaya çıkarılır. Bu çalışmada, EM tabanlı kalite modeli, etiketli verilerle eğitilen karar ağacı modellerinden daha iyi performans sunmuştur.

Koru ve Liu (2007), KOffice ve Mozilla açık kaynaklı projeleri üzerinde ağaç tabanlı modellerle sınıf seviyesindeki metrikleri kullanarak değişim eğilimli (change-prone) sınıfları belirlemeye çalışmışlardır. Sınıfların %20'si değişmiştir ve ağaç tabanlı modeller bu sınıfları belirlemede çok yarar sağlamıştır.

Cukic ve Ma, (2007), NASA'nın JM1 veri kümesinde WEKA içerisindeki 16 öğrenme algoritmasıyla metot seviyesindeki metrikleri kullanarak kusur eğilimliliğinin belirlenmesi üzerine çalışmışlar, PD ve PF kriterlerini değerlendirmede kullanmışlardır. 15 algoritmadan sadece 4 tanesi, %50'den büyük PD değerine ve %50'den küçük PF değerine sahip olabilmiştir. Çalışmada; kusur kestirimi için açık veri kümelerinin kullanılması ve ortak bir değerlendirme kriterinin saptanmasının önemi vurgulanmıştır.

Tomaszewski ve arkadaşları (2007), Ericsson'da geliştirilmiş iki sistem üzerinde uzman görüşü tekniği ve tek değişkenli lineer regresyon analiziyle sınıf ve bileşen seviyesindeki metrikleri kullanarak kusur kestirimi üzerine çalışmışlardır. Değerlendirme kriteri olarak, doğruluk kullanılmıştır. 11 uzman bu araştırmaya davet edilerek sadece bileşen seviyesinde kestirimde bulunmaları istenmiştir. Çalışma sonunda; istatistiksel yöntemlerin uzman görüşünden çok daha başarılı olduğu, uzman görüşü ve istatistiksel modellerin kod birimlerini rastgele seçmekten daha etkin olduğu, uzmanların büyük veri kümelerinde kolaylıkla kestirim yapamadığı raporlanmıştır. Sınıf seviyesindeki kestirim modeli bileşen seviyesindeki modelden daha doğru sonuçlar vermiştir. İstatistiksel modeller kurmak uzman görüşleri ile

kestirim yapmaktan daha ucuz olmasına karşın, uzman görüşlerinin projeye özel hususları dikkate alabileceği ve daha esnek olduğu konusunda bilgiler çalışmada sunulmuştur. Ayrıca; çalışmada uzmanların ortak bir karara varamadığı, farklı bileşenleri kusurlu olarak seçtiği, aynı bileşenleri seçseler bile kusur yoğunluğunu farklı hesapladıkları gözlemlenmiş, uzmanların ürünler hakkındaki deneyiminin sonuçları etkilemediği ifade edilmiştir.

Seliya ve Khoshgoftaar (2007b), NASA veri kümeleri üzerinde "kısıt tabanlı yarı-eğiticili kümeleme yaklaşımı" 'yla (constraint-based semi-supervised clustering approach) 13 metot seviyesindeki metriği kullanarak yazılım kalite analizi üzerine çalışmışlardır. Yaklaşımlarında, k-ortalama algoritması kullanılmaktadır. Değerlendirme kriteri olarak; Tip-I hatası, Tip-II hatası, yanlış sınıflandırma oranı kullanılmıştır. Bu çalışma sayesinde ilk kez, yarı-eğiticili kümeleme (semi-supervised clustering) yazılım kusur kestirimi için uygulanmıştır. Yarı-eğiticili kümelemenin yarı-eğiticili sınıflamaya göre avantajı, bilinen kategoriler dışında yeni kategorilerin belirlenmesini sağlayabilmesidir. Yarı-eğiticili sınıflamada ise, çalışmanın en başında sınıfların biliniyor olması gerekmektedir. Yarı-eğiticili kümelemede, kısıtları tanımlamak, geri besleme sağlamak veya kitle merkezlerinin (centroids) belirlenmesi için kümeleme algoritmasında az sayıda etiketli veriden yararlanılır. Etiketli örnekler, kümelerin kitle merkezlerini (centroid) belirlemek veya diğer bir ifadeyle başlangıç tohumlamasını (initial seeding) sağlamak için kullanılır. Sınıf etiketlerinde gürültü varsa, sınıflayıcı birlikteliği yaklaşımı kullanılabilirken, metriklerde gürültü olması durumunda Seliya tarafından geliştirilen PANDA yaklaşımı kullanılabilir. Bu çalışma sonucunda; yarı eğiticili kümeleme şemasının eğiticisiz kümelemeden ve rastgele sınıflandırmadan daha yararlı olduğu raporlanmış, etiketlenemeyen modüllerinin yarısının gürültülü modüller olduğu ifade edilmiştir.

Olague ve arkadaşları (2007), Java dilinde geliştirilmiş açık kaynaklı JavaScript olan Rhino'nun altı sürümü üzerinde, tek değişkenli ikili lojistik regresyon (univariate binary logistic regression-UBLR) ve çok değişkenli ikili lojistik regresyon (multivariate binary logistic regression-MBLR) teknikleriyle, sınıf seviyesindeki Chidamber-Kemerer metrik kümesini (CK), Abreu'nun nesneye yönelik tasarım metriklerini (MOOD), Bansiya ve Davis'in kalite metriklerini (QMOOD) kullanarak kusur eğilimliliğin kestirimi üzerinde çalışmışlardır. Model geçerlemesi için doğruluk kriteri, metrik etkilerini incelemek için iki değişkenli Spearman korelasyonu kullanılmıştır. Çalışma sonunda; CK ve QMOOD metrik kümelerinin kusur eğilimli sınıfları belirmede yararlı, MOOD içindeki metriklerin yararsız olduğu ortaya konulmuştur. MBLR modellerinin iteratif ve çevik (agile) yazılım süreçlerinde

sınıfların kusur eğilimliliğini belirlemede yararlı olduğu, CK metrik kümesinden UBLR analizine göre WMC ve RFC metriklerinin, QMOOD içerisinden CIS ve NOM metriklerinin kusur kestiriminde fayda sağladığı ifade edilmiştir. MBLR analizine göre, CK metrikleriyle kurulan modelin en iyi performansı verdiği, QMOOD ile kurulan modelin sıralamada 2. sırada yer aldığı açıklanmış, CK ve QMOOD metriklerinin bir araya getirilip bir çalışma yapılmadığı, bunun sebebinin metrikler arasındaki korelasyonun varlığı olduğu ortaya konmuştur.

Binkley ve arkadaşları (2007); Mozilla projesi ve ticari bir ürün üzerinde "lineer birleştirilmiş etkilerin regresyon modeli" (linear mixed-effects regression model) ile QALP skoru, toplam kod uzunluğu, yorum ve boş satırlar dışında kalan kod uzunluğu metriklerini kullanarak kusur kestirimi üzerine çalışmışlardır. Değerlendirme kriteri olarak, determinasyon katsayısını seçmişlerdir. QALP skoru, kod kalitesini değerlendirecek bir aracın geliştirilmesi sırasında ortaya konulmuş olup bir modülün koduyla yorumları arasındaki benzerliği tanımlamaktadır. Bu benzerliği hesaplamak için, bir bilgi erişim aracından ilişkili belgeleri bulmak üzere geliştirilmiş olan "kosinüs benzerliği" tekniği kullanılmıştır. QALP'i hesaplamak için; bir modülden 2 farklı belge (modül yorumlarını ve kaynak kodu içeren) oluşturulur. Çalışma sonucunda; Mozilla için ne QALP skorunun ne de büyüklük ölçümünün iyi kusur kestirici olmadıkları ifade edilmiştir. Mozilla projesinde az sayıda yorum satırı ve raporlanmış kusur mevcuttur. Yorumlar çok az olduğu için, QALP skoru 0'a yakın olmuştur. Çok sayıda sıfırın olduğu durumlarda, istatistiksel gürültü olduğu söylenebilir.

Jiang ve arkadaşları (2007), NASA veri kümeleri üzerinde 1-R, Naive Bayes, Oylanmış Algılayıcı (voted perceptron), Lojistik Regresyon, J48, VFI, IBk, Rastgele Orman teknikleriyle metinsel isterlerden çıkardıkları metrikleri ve kod metriklerini kullanarak yaşam çevriminin erken fazlarında kusur kestirimi konusunda çalışmışlardır. Değerlendirme kriteri; PD ve PF olarak seçilmiştir. Çalışma sonunda; isterler metriklerinin kusur kestirimine yardımcı olamamasına rağmen kod metrikleriyle birleştirildiğinde performansı arttırdığı, Rastgele Orman tekniğinin tüm deneylerde en uyumlu performansı sunduğu, isterler ve kod metriklerinin birlikte kullanıldığı durumda Oylanmış Algılayıcı algoritması dışında tüm algoritmalarda performansın arttığı ifade edilmiştir. NASA'nın veri kümelerinin ait olduğu 13 projeden sadece üçünde isterler metriği mevcuttur. ARM aracı (Automated Requirement Measurement) NASA-Goddard Yazılım Güvence Araştırma Merkezi'nde geliştirilmiş olup, isterler belgelerinden isterler metriklerini çıkarabilmektedir. Araç içerisinde eşik seviyeleri olmasına rağmen model tüm metrikleri kullanmaktadır. İsterler belgesindeki belirtimler,

belirli gruplara ayrılarak metrikler üretilmektedir.

2.7 Değerlendirme

Yapılan literatür taramasına göre; makine öğrenmesi algoritmalarının kullanıldığı son yıllardaki çalışmalarda, Rastgele Orman (RF), Naive Bayes ve J48 algoritmalarının tercih edildiği ve yüksek performansın bu algoritmalarla sağlandığı tespit edilmiştir. Bu çalışmaların çok büyük kısmının, eğiticili öğrenme problemi olarak kusur kestirimine yaklaştığı, yarı-eğiticili sınıflama, yarı-eğiticili kümeleme, eğiticisiz öğrenme yaklaşımlarının çok az sayıda araştırmacı tarafından kusur kestirimi için ele alındığı belirlenmiştir. Naeem Seliya ve Taghi Khoshgoftaar isimli araştırmacıların, eğiticili öğrenme dışındaki yaklaşımları 2004 yılından itibaren ele almaya başladığı ve açık olmayan, telekom veri kümeleri üzerinde çalıştıkları saptanmıştır. Tim Menzies ve Güneş Koru isimli araştırmacıların yaptığı çalışmalar ise PROMISE havuzunda yer alan, açık veri kümelerini kullanmaktadır. Bu alanda aktif olarak çalışan araştırmacılar; Naeem Seliya, Taghi Khoshgoftaar, Tim Menzies, Güneş Koru, Bojan Cukic, Thomas Ostrand, Marek Reformat, John Munson, Ping Guo olarak sıralanabilir. Yakın zamanda; tüm araştırmacıların açık veri kümelerine daha fazla rağbet etmesi, ortak değerlendirme kriterlerinin belirlenmesi, bu kapsamda konferanslarda yarışmaların düzenlenmesi, kusur kestirimi için eğiticili sınıflandırma dışında kalan alanlarda çalışmalara başlanacağı beklenmektedir. Çalışmaların büyük kısmının metot seviyesindeki metriklere yoğunlaştığı ve yeni programlama paradigmaları için önerilecek metrik tipleriyle yeni kusur kestirimi çalışmalarının gerçekleştirilebileceği, makine öğrenmesi algoritmalarının daha fazla popülerlik kazanacağı değerlendirilmektedir.

3. DOĞAL VE YAPAY BAĞIŞIKLIK SİSTEMLERİ

Bu bölümde, öncelikle doğal bağışıklık sistemleri, daha sonra yapay bağışıklık sistem çerçevesi ve algoritmaları açıklanacaktır.

3.1 Doğal Bağışıklık Sistemleri

Bağışıklık Bilimi anlamına gelen "immunoloji" terimi, Eski Roma'da askerlikten muaf asillere verilen "İmmunitas" sözcüğünden gelmektedir. Eski çağlardan beri bağışıklık insanlığın önemli ilgi alanlarından birisi olmuş ve halen çok önemli araştırmaların gerçekleştirilmesini sağlamaktadır. Örneğin, XV. Yüzyılda Türkler ve Çinliler çiçek hastalığı geçirenlerin yaralarındaki kabukları toz şekline getirip burun yoluyla çekmiş ya da bıçakla oluşturdukları yaraya bulaştırmışlar, bu sayede bu hastalığa karşı bağışıklıklarını sağlamaya çalışmışlardır (Yücel, 2003). Bitkiler de dahil olmak üzere tüm canlıların farklı karmaşıklık düzeyinde bağışıklık sistemi mevcuttur. Örneğin bazı bitkilerde, zararlı canlılardan korunmak üzere dikenler mevcuttur. Bu sayede, bu canlılar korunmasız kalmamıştır (De Castro ve Timmis, 2003). Diğer bir örnek ise bakterilerdir. Bakteriler, bakteri virüsü adı verilen fajlara karşı farklı enzimler üretip faj DNA ve RNA'larını (Ribo Nükleik Asit) parçalayabilmektedir (Yücel, 2003).

Bu bölümde ele alınacak olan bağışıklık sistemleri, karmaşık yapıda olan insanoğlunun bağışıklık sistemidir.

Bağışıklık sistemi, zararlı canlılara (patojen) karşı organizmayı korumaktan ve normal şartların devamlılığını sağlamaktan sorumludur. Hastalıklara neden olan; virüs, bakteri, mantar ve parazit gibi mikroorganizmalar patojen olarak adlandırılmaktadır. Bağışıklık sisteminin temel elemanı olan hücreler, bu patojenleri doğrudan tanıyamazlar; patojenlerin antijen adı verilen bölümleri bağışıklık sisteminin temel hücrelerinin reseptör molekülleri tarafından tanınır. Bağışıklık sistemi; bu zararlıların tanınıp yok edilmesinden ve öz (self) antijenlerin yabancı (nonself) antijenlerden ayırt edilmesinden sorumludur. Öz antijenler, organizmanın kendisine aittir ve patojenlerin antijenlerinden ayırt edilmesi gerekir (De Castro ve Timmis, 2003). Bu ayrımın gerçekleştirilemediği durumda, oto bağışıklık hastalıkları (autoimmune diseases) ortaya çıkmaktadır.

Oto bağışıklık hastalıklarının sayısı günümüzde 50'nin üzerinde olup bu tür hastalıklara sahip olanların 2/3'ü kadındır. Merkezi sinir sistemi, sindirim sistemi, cilt, endokrin sistem ve kas-iskelet sistemini etkileyen farklı türde oto bağışıklık hastalıklar vardır. Örneğin; çoklu skleroz

(multiple sclerosis-MS) hastalığında bağışıklık sistemi merkezi sinir sistemine saldırmaktadır. Bağışıklık sistemindeki bazı hücreler, sinir hücrelerinin etrafında bulunan miyelin kılıfı vücuda zararlı olarak algılayıp yok etmeye çalışmakta ve myelin kılıf işlevsiz hale gelmektedir. Bu hastalıkların ortaya çıkmasına neden olan etkenler henüz anlaşılamamış ancak bazı çevresel faktörlerin hastalığın seyrini hızlandırdığı raporlanmıştır. Bağışıklık sisteminin fonksiyonelliğini yavaşlatan "bağışıklık baskılayıcı" ilaçlar bu kapsamda kullanılabilmekte ancak vücudun savunmasını enfeksiyonlara karşı zayıf bırakmaktadır (Çırakoğlu, 2003a).

Bağışıklık sistemleri gün geçtikçe daha fazla araştırmacının ilgisini çekmekte ve bu sistemlerin sahip olduğu yetenekler, bilgisayar bilimleri, mühendislik ve matematik gibi alanlardan farklı uzmanların bu kapsamda çalışma gerçekleştirmesini sağlamaktadır. Bağışıklık sistemlerini ilginç kılan temel özellikler aşağıda verilmektedir (Karaboğa, 2004):

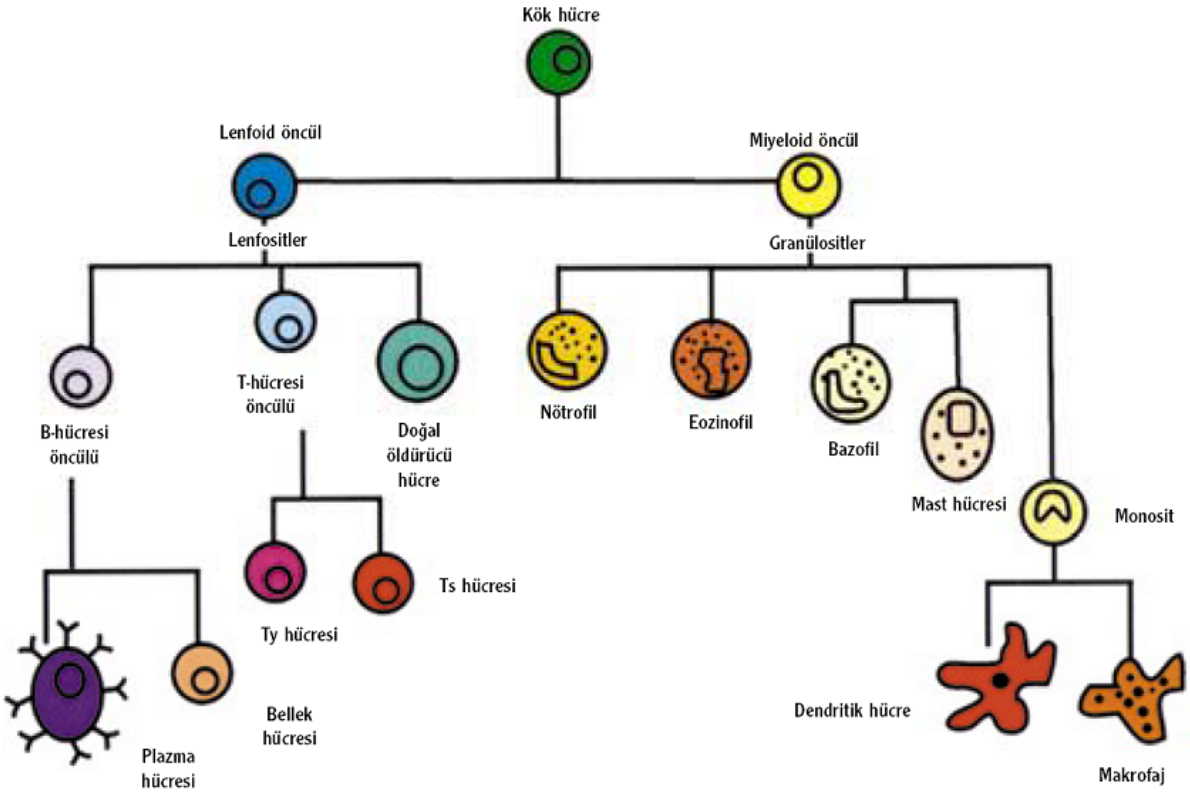
- *Teklik*: Bireylerin kendilerine özel bağışıklık sistemi vardır.
- *Anomali tespiti*: Vücuda giren patojenler bu sistem tarafından tespit edilebilmektedir.
- *Dağıtılmış tespit:* Merkezi bir sistem yoktur, hücreler tüm vücutta yer alır.
- *Gürültü toleransı:* Patojenin bire bir tanınma ihtiyacı yoktur, kısmi tanınma yeterlidir.
- *Takviyeli öğrenme:* Patojenlerin yapıları öğrenilebilmekte ve sonrasında daha hızlı yanıt verilebilmektedir.
- *Farklılaşma:* Sistem, farklı antijenlere yanıt vermek ya da aynı antijene karşı daha güçlü yanıt sağlamak üzere mutasyon gibi işlemleri kullanmaktadır.
- *Dış çevre uyumluluğu:* Sistem, dış çevreye karşı kendisini uyarlayabilmektedir.
- *Hafızaya sahip olma:* Bellek hücreleri, öğrenilen patojen yapılarına yanıtı hızlıca vermek için kullanılır. Bu nedenle, sistemin hafızalı olduğu ifade edilir.

3.1.1 Doğal Bağışıklık Sistemi Bileşenleri

Bağışıklık sistemi hücrelerinin tümü, Hematopoetik kök hücre adı verilen bir hücreden oluşmaktadır. Bu hücrenin değişiminden oluşan hücreler Şekil 3.1'de gösterilmektedir. Bu kök hücrenin değişimi ile lenfoid ve miyeloid hücreler ilk aşamada ortaya çıkmaktadır. Miyeloid öncüllerin değişmesi sayesinde; çok çekirdekli lökositler (nötrofil, eozinofil, bazofil), monositler ve mast hücreleri oluşur. Lenfoid öncüllerin değişmesi ile B hücreleri, T

hücreleri ve doğal öldürücü hücreler oluşmaktadır (Demiralp, 2003).

Patojenler solunum sistemi, deri gibi engelleri aşıp organizmaya girdiği zaman ilk olarak nötrofil ve monosit isimli bağışıklık hücreleri tarafından yok edilmeye çalışır, başarı sağlanamadığı durumda B ve T hücreleri adı verilen lenfositler sürece dahil olur. Bu aşamada, B hücreleri antikorları salgılarken, sitotoksik (öldürücü) T hücreleri antijenleri yok etmeye çalışır (Yücel, 2003).

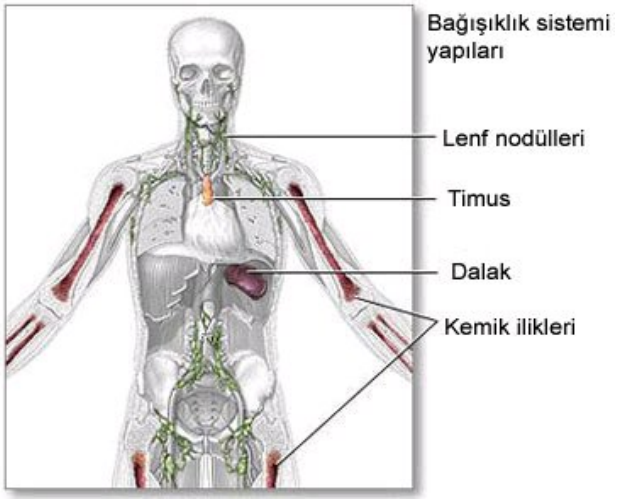


Şekil 3.1 Bağışıklık sisteminin hücreleri

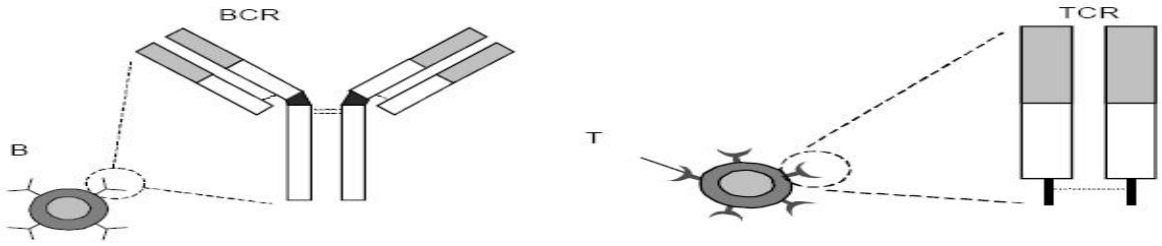
Kemik iliği ve timüs, bağışıklık hücrelerinin oluşmasını ve gelişmesini sağlayan iki önemli organdır. Kemik iliğinde tüm kan hücreleri oluşup bazıları gelişmekte iken timüste T hücreleri olgunlaşmaktadır. Şekil 3.2'de kemik iliği ve timüsün insan vücudunda yer aldığı bölümler gösterilmektedir. Çok farklı tiplerde bağışıklık hücreleri vardır ancak bu bölümde lenfositler açıklanacaktır. Lenfositler; B hücreleri ve T hücreleri adı verilen iki gruba ayrılır. Patojenleri tanımada lenfositler görev almaktadır (De Castro ve Timmis, 2003).

Kemik iliği içinde gelişen hücreler B hücreleri, timüste gelişenler de T hücreleridir. Bu nedenle, timüs kelimesinin ilk harfi bu hücrelerin isminde yer almaktadır. İki hücre de, yüzeyinde reseptör moleküllere sahiptir ve bu moleküller zararlı antijeni tanır. T hücresi reseptörü TCR, B hücresi reseptörü BCR ya da antikor (antibody-Ab) olarak

adlandırılmaktadır. Şekil 3.3'de B ve T hücreleri reseptör molekülleri ile birlikte gösterilmektedir (De Castro ve Timmis, 2003).



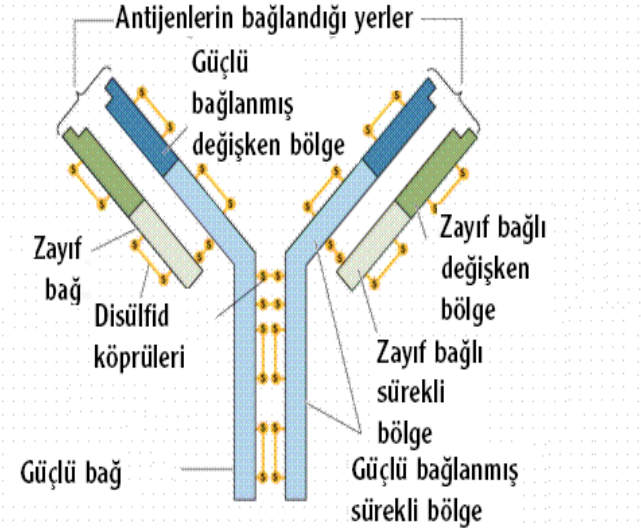
Şekil 3.2 Bağışıklık sistemi yapıları



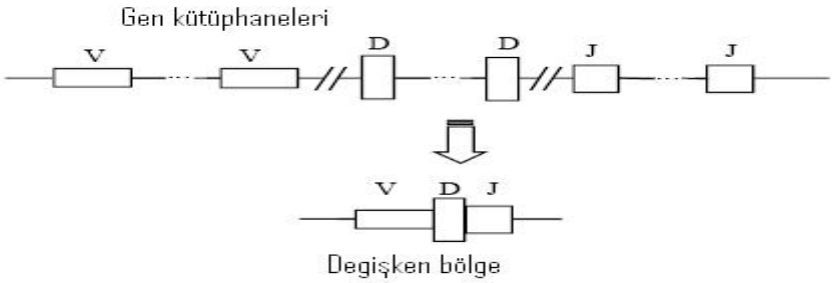
Şekil 3.3 B hücreleri, T hücreleri ve reseptör molekülleri (De Castro ve Timmis, 2003)

Şekil 3.2'de verilen kemik iliği, en uzun kemiklerin boşluklarında yer alan yumuşak bir dokudur. Bir kan hücresi B hücresine dönüştüğü zaman, yüzeyinde antikor molekülü oluşur ve bu molekül zararlı antijen ile bağlanma görevine sahiptir. Şekil 3.4'de antikor molekülünün Y biçimindeki yapısı gösterilmektedir. Antikorlar ağır ve hafif zincir adı verilen 2 protein zincirinden oluşmakta olup, ağır zincirler yaklaşık 440 aminoasit, hafif zincirler de yaklaşık 220 aminoasitten oluşur. Bu moleküller 100 derecelik açıyla dönebilir ve hareket edebilir (Yücel, 2003). "Değişken bölge" (variable region) antijeni tanımaktan, "sabit bölge" (constant region) antikoru hücre yüzeyine bağlamaktan sorumludur. Şekil 3.4'de verilen "sürekli bölge" ifadesi "sabit bölge" anlamında kullanılmıştır. Antikorlar normalde hücre yüzeyindedirler, ancak bağışıklık yanıtının ardından kan akışına geçebilirler. Antikor molekülleri, kemik iliğinde DNA'nın yeniden düzenlenmesiyle oluşur ve gen kütüphanesindeki genler bu molekülü oluşturmak üzere bitiştirilir (De Castro ve Timmis,

2003). Şekil 3.5'de bu durum resmedilmiştir.



Şekil 3.4 Antikorun yapısı (Yücel, 2003)



Şekil 3.5 Kemik iliğinde antikorun oluşması (De Castro ve Timmis, 2003)

3.1.2 Hastalıklarla Savaşta Antikorlar

Antikorlar son yıllarda viral ve bakteriyal enfeksiyonların tedavisi için sıkça kullanılmaktadır. Bu yaklaşımlar, ilk olarak serumla tedavi olarak uygulanmıştır. 1895 yılında Heriourt ve Richet isimli araştırmacılar, kanser hücreleriyle bağışıklık kazandırdıkları hayvan serumlarını hastalara uygulamışlar ve belirli ölçüde hastalığı azaltmayı başarmışlardır. 1975 yılında Köhler ve Milstein hibridoma yöntemini geliştirmiş ve 1984 yılında tıp dalında Nobel ödülü kazanmışlardır (Erdağ, 2003). Hibrodoma yönteminde, farelere antijen verilip antikorlar üretiliyor, fare B lenfositleri ile insan B lenfositleri birleştiriliyor, çeşitli kültür ortamlarına alınan hibrodomaların bölünerek çoğalması gerçekleştiriliyor, antijene bağlanabilen antikoru üreten hibridomalar fare vücudu ya da laboratuvar ortamında çoğaltılıyor ve antikorlar saflaştırılıyor (Erdağ, 2003). Fare kökenli bu yaklaşım, tanıda başarılı olsa da tedavi

noktasında sınırlı başarıya sahiptir. Hibrodoma teknolojisi yanında, tamamen insan antikorlarının elde edilmesine odaklanmış olan rekombinant DNA teknolojisi de gelişerek "antikor mühendisliği" adı verilen alan ortaya çıkmış, rekombinant antikor üretimi mümkün olmuştur (Erdağ, 2003).

3.1.3 Bağışıklık Sisteminin Hatası ve Aşılar

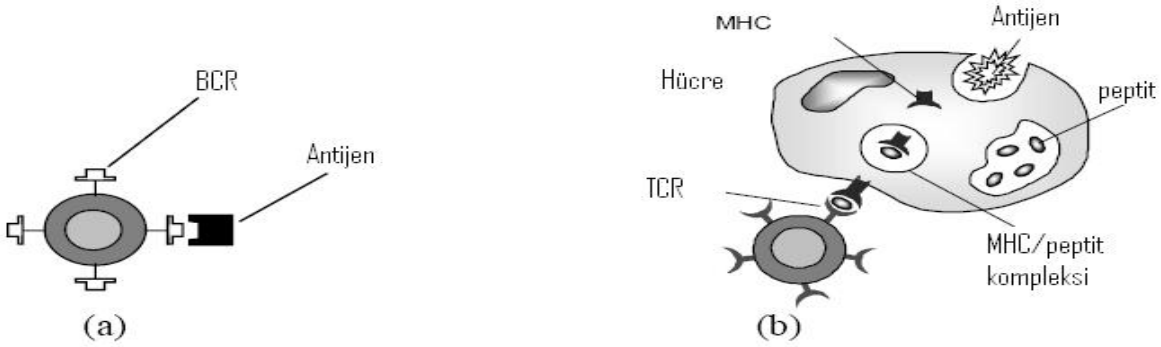
Oto bağışıklık hastalıkları dışında bağışıklık sistemi farklı düzeylerde hatalar yapabilmektedir. Alerji olarak bilinen ve fazla duyarlılık hali olarak açıklanabilecek bu durum, bireylerin bir molekülü zararlı olarak algılaması neticesinde şiddetli tepki oluşturmasıdır. Bu tür bireylerde, B hücreleri yanlış bilgi içerdiği için gereksiz şekilde immünoglobün E (IgE) molekülleri üretilmekte ve bu moleküller mast hücreleri ile bazofillere bağlanarak bu hücreleri aşırı duyarlı hale getirmektedir. Bu hücreler, histamin adı verilen vücudu enfeksiyonlara karşı koruyan yapıları içerdiğinden ve histaminin fazla salınması tahrip edici etkiye yol açtığından, kuvvetli alerjik tepkiler oluşabilmektedir (Çırakoğlu, 2003b).

Aşılar sayesinde bulaşıcı hastalıklar büyük oranda önlenebilmektedir. Aşılama, aktif ve pasif olarak 2 grupta değerlendirilebilir. Pasif aşılamada, başka canlıda üretilen antikorlar kişiye verilmektedir. Örneğin, kuduz bir hayvan tarafından ısırılan bir insana hem antikorlar hem de kuduz aşısı aynı anda verilmektedir. Aktif aşılamada ise hastalık yapıcı mikroplar önceden organizmaya tanıtılır ve bağışıklık hücreleri eğitilir. Aşılar, öldürülmüş mikroplardan veya etkileri azaltılmış halde hazırlanabilmektedir (Çırakoğlu, 2003b).

3.1.4 Bağışıklık Sisteminde Örüntü Tanıma

Vücuda dışarıdan giren zararlı canlıların tanınması moleküler düzeyde gerçekleşmektedir. B ve T hücrelerinin sahip olduğu yüzey reseptörlerinin şekliyle, yabancı antijenlerin şeklinin uyuşması incelenmektedir. Bu şekil tamamlayıcılığı söz konusu olduğu durumda, bir araya gelebileceklerinden bağışıklık yanıtı oluşacaktır. Antikorlar (BCR), doğrudan antijeni tanıyıp bağlanabilmektedir, ancak TCR'ler doğrudan bu işlemi gerçekleştiremez ve vücuda özel bazı moleküller tarafından sunulan antijenleri tanıyıp bağlanabilirler. TCR'ler peptit/MHC (major histocompatibility complex) kompleksleri adı verilen molekülleri tanırlar ve peptitler, antijenlerin parçalanmış hali olarak bilinmektedir (De Castro ve Timmis, 2003). Şekil 3.6 a'da BCR'nin doğrudan antijeni tanıyabildiği, Şekil 3.6 b'de TCR'nin MHC/peptit komplekslerini tanıyabildiği gösterilmektedir. MHC molekülü, antijeni kimyasal işleme tabi tuttuktan sonra, T hücrelerine sunmaktadır ancak bu noktada T hücreleri yardımcı hücre olan APC'ye (antigen

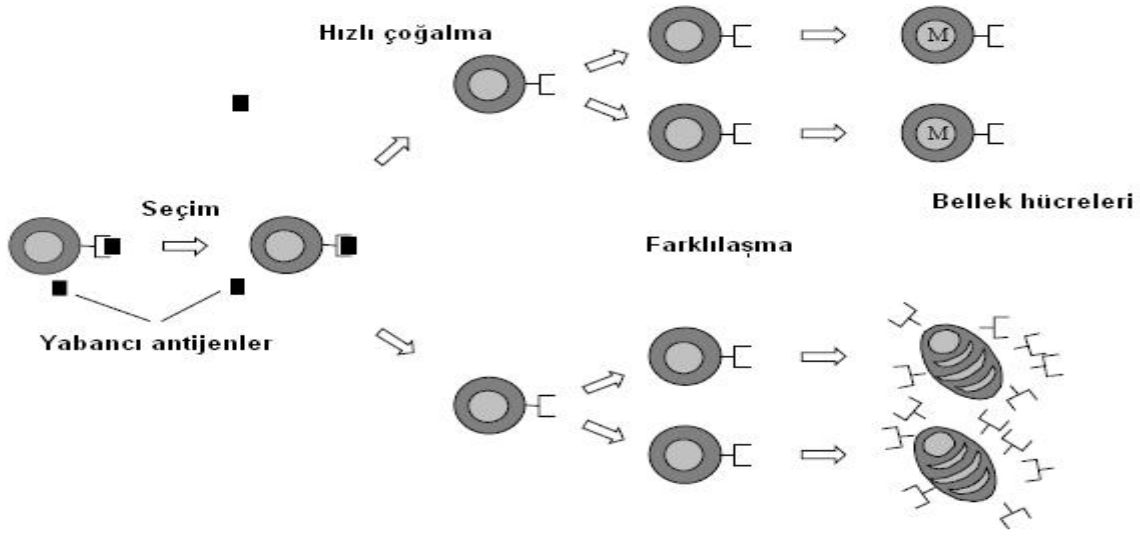
presenting cells) ihtiyaç duyar. APC'ler; antijenleri parçalara ayırır ve koparılan aminoasit dizilimini T hücresine bildirerek, T hücresinin görevine başlaması sağlar. Özetle; bağışıklık sisteminde T hücresinin tanıma işlemi, birçok hücrenin etkisi altında gerçekleşmektedir.



Şekil 3.6 BCR ve TCR'in antijeni tanıması (De Castro ve Timmis, 2003)

3.1.5 Klonal Seçim

Antijenler tanındıktan sonra, tanıma işlemini gerçekleştiren hücrelerin yeniden üretimi klonlama (mitoz) ile gerçekleştir. Klonlama işleminin ardından, mutasyon işlemi gerçekleşerek klon içinde ölçeği çok büyük olmayan farklılıklar oluşur. Seçici mekanizma sayesinde, antijeni daha iyi tanıyan hücreler bellek (memory) hücreleri haline dönüşür ve antijenler sonraki saldırılarında kolayca tanınır. Antijen tanıma, hücrelerin hızla çoğalması (proliferation) ve farklılaşma (diversification) adımları "klonal seçim" olarak adlandırılır (De Castro ve Timmis, 2003). Şekil 3.7'de klonal seçim temsili olarak gösterilmektedir.



Şekil 3.7 Klonal seçim (De Castro ve Timmis, 2003)

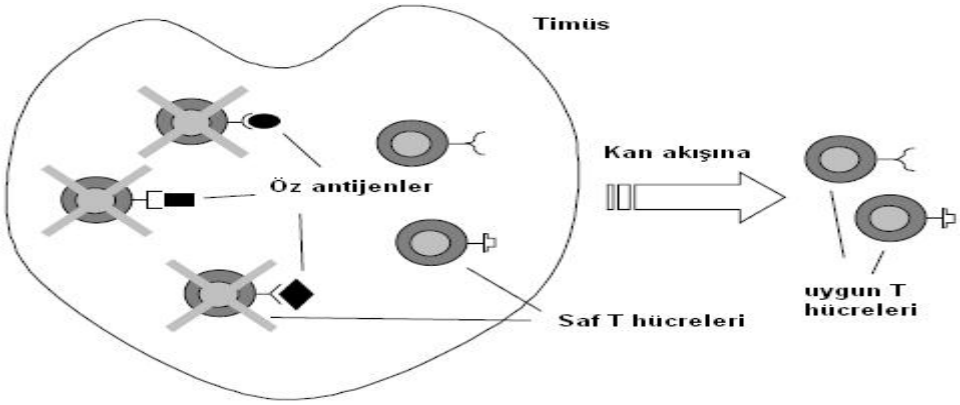
Hücre reseptörünün antijenlere bağlanma derecesi afinite (affinity) olarak tanımlanmaktadır ve bağlanma ne kadar güçlüyse afinite o kadar yüksektir. Mutasyon ve ardından antijenle en iyi uyuşan yavruların seçilmesini garanti eden sürece "afinite olgunlaşması" (affinity maturation) adı verilmektedir. Afinite olgunlaşması sırasında gerçekleşen mutasyona, "somatik mutasyon" adı verilmesinin sebebi lenfositlerin somatik hücreler olmasıdır (De Castro ve Timmis, 2003). Üreme hücreleri dışındaki hücrelere somatik hücre adı verilmektedir.

Klonlama sırasındaki mutasyon oranları çok yüksek değerlere sahip olduğundan, bağışıklık sistemindeki mutasyon kavramı, "somatik hipermutasyon" terimiyle kullanılmaktadır. Bir antijen ile hücre reseptörü arasındaki afinite ne kadar düşükse, mutasyon oranı o kadar yüksektir. Bu yaklaşım ile afinitesi yüksek olan hücrelerin korunması sağlanmıştır. Hücrenin çoğalma oranı (proliferation rate) ise afinite ile doğru orantılıdır. Dışarıdan organizmaya giren zararlı antijenleri bazı hücreler tanır ve bu hücreler daha sonra klonal seçim ve afinite olgunlaşması işlemine tabi tutulur. Klonal seçim, B ve T hücrelerini etkilerken afinite olgunlaşması B hücrelerinde gözlemlenir (De Castro ve Timmis, 2003).

3.1.6 Öz / Yabancı Ayrımı

Dışarıdan vücuda giren patojenlerin antijenlere sahip olması gibi, vücutta da bazı antijenler mevcuttur. Bağışıklık sisteminin, öz (self) antijenleri yabancı (nonself) antijenlerden ayırt etmesi oldukça karmaşıktır. Bu bölümde, T hücrelerinin timüs içerisinde negatif seçimi açıklanarak öz / yabancı ayrımının bir bölümü ortaya konulmuş olacaktır. Bağışıklık sistemindeki bu mekanizma, bilgisayar bilimlerindeki araştırmacılar tarafından bilgisayar güvenliği için 1990'larda kullanıldığı için bu bölümde kısaca açıklanacaktır.

Timüs, Şekil 3.2'de gösterildiği gibi göğsün üst bölgesinde yer alan bir organ olup T hücrelerinin kemik iliğinde üretildikten sonra göç ettiği yerdir. Olgunlaşmamış T hücreleri, timüs içerisinde negatif seçim sürecine tabi tutulur. Timüste yer alan tüm antijenler öz olduğundan, buradaki herhangi bir T hücresi antijenle birleşiyorsa bu T hücresi yok edilir. Birleşmeyen T hücreleri kana salınarak yabancı antijen aramaya başlar (De Castro ve Timmis, 2003). Şekil 3.8'de bu durum, basitleştirilmiş olarak temsil edilmektedir.



Şekil 3.8 Negatif seçimin basitleştirilmiş şekli (De Castro ve Timmis, 2003)

3.1.7 Bağışıklık Ağ Teorisi

Klonal seçim teorisi, yabancı antijenlere sistemin nasıl yanıt verdiğini açıklarken negatif seçim teorisi, öz antijenlerini tanıyan hücrelerin sistemde ele alınış şeklini ortaya koyar. Bunların dışında, bağışıklık sistem hücrelerinin diğer hücrelerle nasıl etkileşime girdiği de önemli bir sorudur (De Castro ve Timmis, 2003) .

Jerne (1974), bu etkileşimi açıklamak üzere "bağışıklık ağ teorisi" (immune network theory) isimli bir teori ortaya koymuş ve bu teori, bazı bağışıklık uzmanları tarafından (Langman ve Cohn, 1986) deneysel olarak ispatlanması zor olduğu için eleştirilmiş ve reddedilmiştir. Bu teoriye göre, antikor moleküllerinin diğer antikor molekülleri tarafından tanınması için reseptörlerinde bazı bölümler vardır. Bu bölümler yardımıyla, antikorlar sadece antijenleri değil öz antijenleri de tanıyabilmektedir. Antikorlar arasındaki bu etkileşimden dolayı bağışıklık ağı ortaya çıkmaktadır (De Castro ve Timmis, 2003).

3.2 Yapay Bağışıklık Sistemleri

"Esnek hesaplama (soft computing), hesaplamalı zekâ sistemlerinin kavramlaştırılması, tasarlanması ve geliştirilmesi için gerekli alt yapıyı sağlamak üzere Yapay Sinir Ağları, Bulanık Sistemler, Evrimsel Algoritmalar ve Olasılıksal Usavurum gibi hesaplamalı zekâ yaklaşımlarının bir füzyonudur" (De Castro ve Timmis, 2003).

Zadeh (1997), esnek hesaplamanın farklı yaklaşımlardan oluşan bir birliktelik olmadığını ve her yaklaşımın farklı bir yöntem bilimle katkıda bulunduğu bir ortaklık olduğunu ifade etmiştir. De Castro ve Timmis (2003), Yapay Bağışıklık Sistemleri'nin (YBS) de yeni bir esnek hesaplama paradigması olduğunu çalışmalarında açıklamaktadır. YBS'ler, karmaşık

problemleri çözmek üzere teorik bağışıklık biliminden, gözlemlenen bağışıklık fonksiyonlarından, prensiplerinden ve modellerinden esinlenen hesaplamalı sistemlerdir (De Castro ve Timmis, 2002). YBS'leri daha kolay tasarlamak için, bu yaklaşımları bir çerçeve (framework) içerisinde ele almak gerekir. Bir çerçevede aşağıdaki elemanlar yer alır (De Castro ve Timmis, 2003):

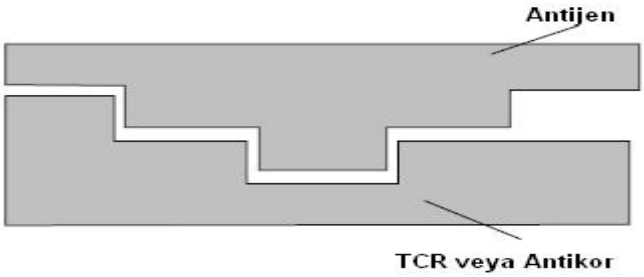
- Sistem bileşenlerinin gösterimi (representation),
- Çevreyle ve kendi aralarında bireylerin etkileşimini değerlendirecek mekanizmalar kümesi,
- Sistem dinamiğini yönetecek adaptasyon prosedürleri.

YBS'ler için açıklanacak olan çerçevede; bağışıklık hücrelerinin nasıl temsil edileceği (representation), bu hücrelerin zararlı antijenlerle arasındaki etkileşimin nasıl ölçüleceği (afinite ölçümleri) ve YBS dinamiğini yönetecek algoritmalar (bağışıklık algoritmaları) ortaya konulacaktır. Bu katmanlı yapı Şekil 3.9'de gösterilmektedir.

Şekil 3.9 YBS çerçevesi ve katmanlar (De Castro ve Timmis, 2003)

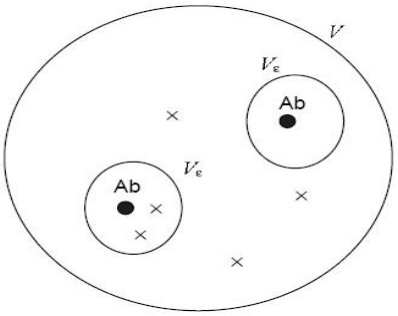
3.2.1 Gösterim

Antijen / antikor arasındaki bağlanmayı kavramsal olarak temsil etmek üzere "şekil-uzayı" (shape-space) (S) kavramı, Perelson ve Oster tarafından önerilmiştir (1979). S içinde, antijenik reseptör ve antijen arasındaki bağlanma derecesi "tamamlayıcılık bölgeleri" (regions of complementarity) ile ölçülür. Şekil 3.10'da tamamlayıcılık bölgeleri ile tanımayı temsil edecek şekilde antijen ve antikorlar arası bağlanma resmedilmiştir.



Şekil 3.10 Tamamlayıcılık bölgeleri ile tanıma (De Castro ve Timmis, 2003)

Bir molekülün ilişkili özelliklerinin kümesine "genelleştirilmiş şekil" adı verilir. N adet hücreden oluşan popülasyon, N nokta içeren V hacimli şekil-uzayına karşılık gelir. Antijenler ve antikorlar birbirini tam olarak tamamlamıyorsa, aralarındaki afinite düşük olacaktır. Bu durumu, bazı anahtarların kilite girebilmesine rağmen kiliti açamamasına benzetebiliriz. Her antikor, tanıma bölgesi (V_ε) içerisine düşen antijenlerle etkileşir ve bir antikor, birden fazla antijeni tanıyabilir (De Castro ve Timmis, 2003). Şekil 3.11'de 2 adet antikora (Ab) ait tanıma bölgesi gösterilerek bu bölgelerde yer alan antijenler (x) gösterilmiştir.



Şekil 3.11 Şekil uzayında tanıma bölgeleri (De Castro ve Timmis, 2003)

Bir molekülün genelleştirilmiş şekli, özellik katarı (string) ile temsil edilebilmektedir. L boyutlu şekil-uzayı olan S^L içerisindeki bir molekül için, sahip olduğu özellikler m = <m1, m2, ..., m_L>, $m \in S^L \subseteq \Re^L$ olarak gösterilebilir.

3.2.2 Afinite Ölçümleri

Antikor (Ab) - Antijen (Ag) arasındaki afinite; uzaklıkla ilişkili olduğundan, 2 vektör arasındaki uzaklığı ölçen Hamming, Manhattan ve Öklid uzaklık eşitliklerinden yararlanılabilir. Antikor koordinatları Ab = <Ab1, Ab2, ..., AbL> ve antijen koordinatları Ag = <Ag1, Ag2, ..., AgL> ise bu iki molekül arasındaki uzaklık aşağıdaki eşitliklerle hesaplanabilir (De Castro ve Timmis, 2003).

$$D = \sqrt{\sum_{i=1}^{L} (Ab_i - Ag_i)^2} \quad \text{Öklid Uzaklığı} \quad (3.1)$$

$$D = \sum_{i=1}^{L} \left| Ab_i - Ag_i \right| \quad \text{Manhattan Uzaklığı} \quad (3.2)$$

$$D = \sum_{i=1}^{L} \delta_i \quad \text{Hamming Uzaklığı} \quad (3.3)$$

Hamming uzaklığı hesaplanırken, $Ab_i \neq Ag_i$ ise $\delta_i = 1$ değerini aksi halde 0 değerini alır.

Çoğu YBS gerçeklemesinde, moleküllerin gösterimi ikili şekil-uzayı ile sağlanır. Bu durumda, antikor bit katarı ve antijen bit katarı arasındaki afinite farklı ölçümlerle belirlenir (De Castro ve Timmis, 2003):

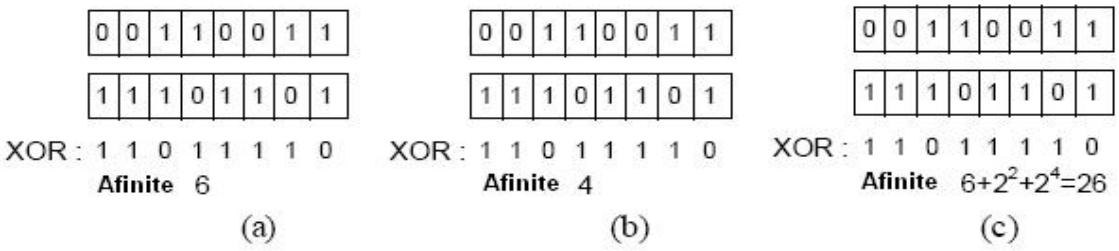
a. *Hamming uzaklığı:* İkili katarlara XOR (exclusive-or) operatörü uygulanarak hesaplanabilir. XOR operatörü uygulandıktan sonra, elde edilen katardaki 1'lerin sayısı iki bit katarı arasındaki afinite ölçüm değerini verecektir.

b. r-bitişik bit kuralı (r-contiguous bit rule): İkili katarlara XOR uygulandıktan sonra elde edilen katardaki peş peşe gelen 1'lerin sayısı, afinite ölçüm değerini verir.

c. Çoklu bitişik bit kuralı (multiple r-contiguos bit rule): İkili katarlara XOR uygulandıktan sonra, elde edilen katar üzerinde Eşitlik 3.4 uygulanarak afinite değeri elde edilir.

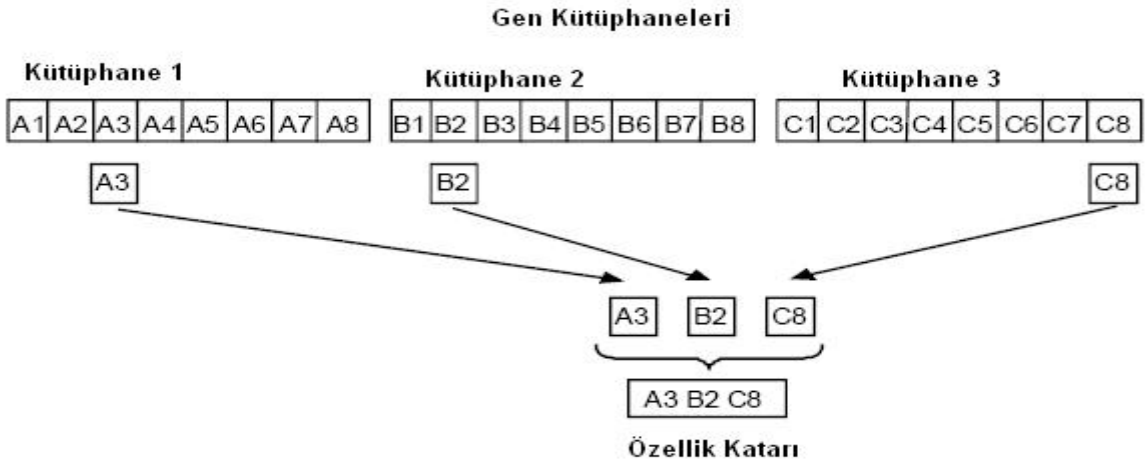
$$D = D_H + \sum_{i=1} 2^{l_i} \quad (3.4)$$

D_H, toplam Hamming uzaklığını göstermektedir. l_i, her tamamlayıcı bölge olan i'nin en azından 2 bitişik tamamlayıcı içeren bit uzunluğudur. l_i, XOR işleminden sonra oluşan katar içerisinde, peş peşe gelen ve 2'den az olmayan 1'lerden oluşan grupların bit uzunluğu olarak da ifade edilebilir. Şekil 3.12'de, 3 yönteme göre afinite hesaplanmaktadır. Şekil 3.12 a'da Hamming uzaklığı, b'de r-bitişik bit kuralı, c'de çoklu bitişik bit kuralı uygulanmıştır. Şekil 3.12 a'da XOR sonrası 6 adet 1 olduğundan afinite 6, b'de XOR sonrası 4 adet 1 peş peşe geldiğinden afinite 4 olarak hesaplanmıştır. Şekil 3.12 c'de ise afinite 6, peş peşe gelen 1 katarları (11 ve 1111) olduğundan toplam afinite Eşitlik 3.4'e göre 26 olarak elde edilmiştir.



Şekil 3.12 İkili Hamming şekil-uzayı için afinite ölçümleri (De Castro ve Timmis, 2003)

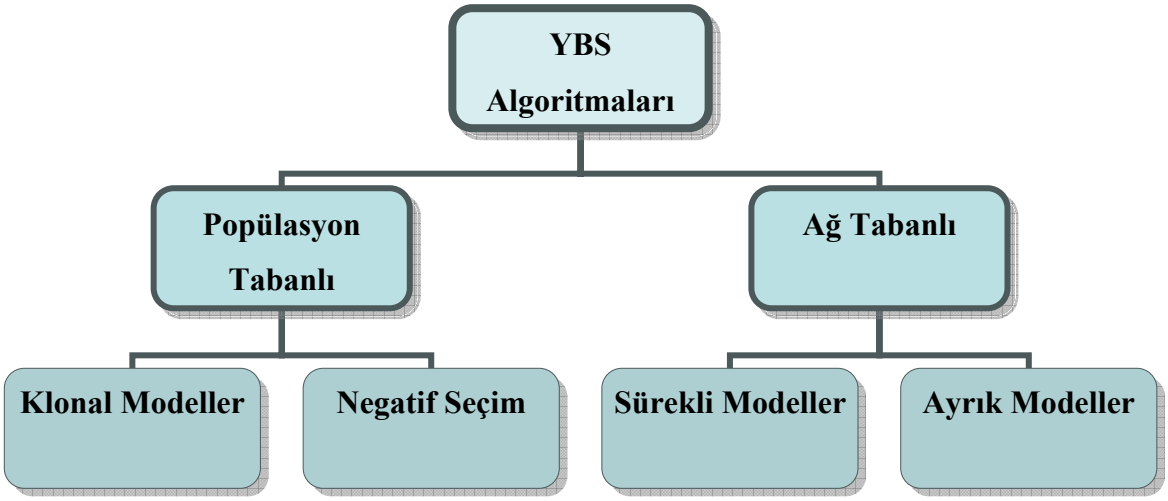
Bağışıklık hücreleri ve moleküller, kemik iliği içerisinde oluşturulduğundan en basit kemik iliği modelinde rastgele sayı jeneratörleri kullanılmaktadır. Daha karmaşık modellerde, antikor moleküllerinin sentezlenmesi için gen kütüphanelerinin kullanılması dikkate alınmıştır. Hightower ve arkadaşları (1995), genetik algoritmaları kullanarak antikor moleküllerinin genetik kodlaması için evrimin etkilerini çalışmıştır. Şekil 3.13'de gen kütüphanelerinden antikor sentezlenme süreci temsili olarak gösterilmektedir.



Şekil 3.13 Gen kütüphanelerinden antikor sentezlenme süreci (De Castro ve Timmis, 2003)

3.2.3 Bağışıklık Algoritmaları

YBS çerçevesinde en son katman, bağışıklık algoritmalarıdır. Şekil 3.14'de YBS algoritmalarının taksonomisi verilmektedir (De Castro ve Timmis, 2003).



Şekil 3.14 YBS algoritmaları taksonomisi

Şekil 3.14'de verilen taksonomi ağacını yeni algoritmalarla genişletmek mümkündür ancak bu bölümde sadece bu taksonomide verilen modeller açıklanacaktır. Aşağıda bu algoritmaların temel uygulama alanları verilmektedir (De Castro ve Timmis, 2003):

- *Klonal Seçim:* Örüntü tanıma, optimizasyon
- *Negatif Seçim:* Anomali ve kusur tespiti
- *Sürekli Bağışıklık Ağı:* Kontrol, robotik, optimizasyon ve örüntü tanıma
- *Ayrık Bağışıklık Ağı:* Örüntü tanıma, veri analizi, makine öğrenme ve optimizasyon

Bu bölüm, şimdiye kadar geliştirilmiş algoritmalar hakkında genel bilgiler sunmaktadır.

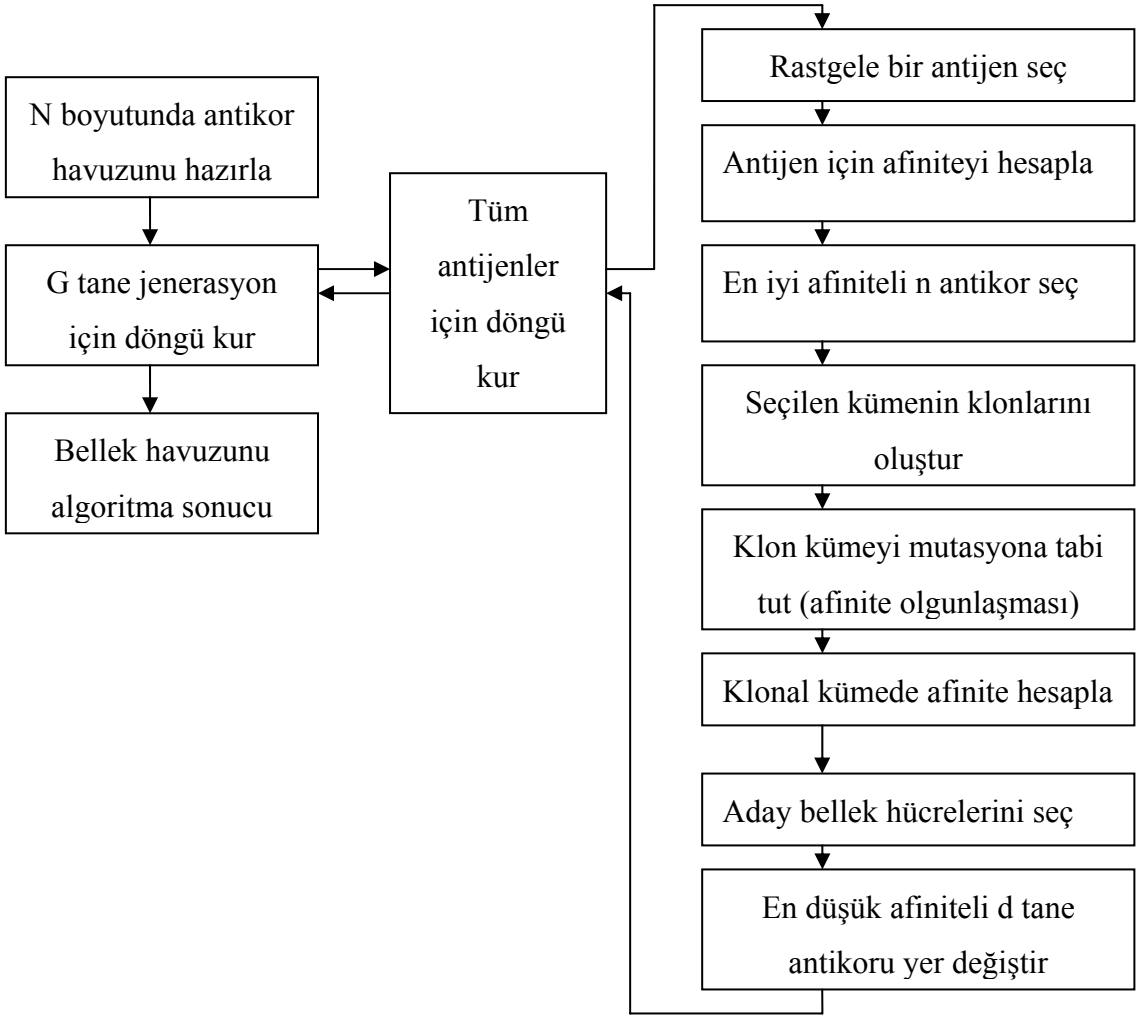
3.2.3.1 Klonal Seçim

Klonal seçim algoritması geliştirilirken aşağıdaki elemanlardan esinlenilmiştir (Brownlee, 2005a).

- Uyarılmamış hücrelerin ölümü,
- En fazla uyarılan bireylerin seçilmesi ve klonlanması,
- Bellek hücrelerinin idame ettirilmesi,
- Afinite olgunlaşması (mutasyon),
- Antijenle afinitesine bağlı olarak klonların yeniden seçimi,

- Çeşitliliğin oluşturulması ve idame ettirilmesi.

Algoritmanın amacı, çözümün bir elemanı olacak olan antikoru ve bu antikorlardan oluşan bellek havuzunu oluşturmaktır. Antijen ise problem uzayının bir elemanını gösterir. Algoritmada bellek antikorları havuzunu oluşturmak üzere 2 adet arama mekanizması mevcuttur. İlki yerel arama olup klonlanmış antikorların afinite olgunlaşması (hipermutasyon) sayesinde gerçekleşir. İkincisi global bir kapsama sahiptir ve rastgele oluşturulmuş antikorlar popülasyona eklenerek çeşitliliği arttırır, yerel optimuma takılmayı engeller (Brownlee, 2005a). Şekil 3.15'de klonal seçim (CLONALG) algoritmasının akışı gösterilmektedir.



Şekil 3.15 CLONALG algoritmasına genel bakış (Brownlee, 2005a)

CLONALG algoritmasının adımları aşağıda açıklanmaktadır (Brownlee, 2005a):

1) *İlklendirme:* Algoritmada ilk adım, N büyüklüğünde antikor havuzunun

oluşturulmasıdır. Bu havuz daha sonra, 2 bölüme ayrılır. İlki algoritma sonucunda çözümü temsil edecek olan bellek antikor bölümüdür (m). Diğer bölüm ise (r) sisteme ek çeşitlilik sağlamak amacıyla kullanılır.

2) *Döngü:* Sistem daha sonra G adet iterasyon ile tüm antijenlere maruz bırakılır. Sistemin yürüttüğü G jenerasyon sayısı kullanıcı tanımlı bir parametredir.

   a. *Antijen seç:* Antijen havuzundan rastgele bir antijen seçilir.

   b. *Maruz bırak:* Sistem, seçilen antijene maruz bırakılır. Tüm antikorların bu antijen ile arasındaki afinite değerleri belirlenir ve bu kapsamda Hamming uzaklığından yararlanılabilir.

   c. *Seçim:* Antijenle arasındaki afinite en yüksek olan n adet antikor, antikor havuzundan seçilir.

   d. *Klonlama:* Afiniteyle orantılı olarak seçilen antikorlar kümesi klonlanır.

   e. *Mutasyon (afinite olgunlaşması):* Klonlar daha sonra afinite olgunlaşma sürecine tabi tutulur ve mutasyon oranı ebeveyn afiniteleri ile ters orantılıdır.

   f. *Klonların maruz bırakılması:* Klonlar daha sonra antijene maruz bırakılır ve afiniteler hesaplanır.

   g. *Adaylık:* Klonlar içerisinde en yüksek afiniteli antikorlar m içerisine eklenmek üzere aday bellek antikorları olarak seçilir.

   h. *Yer değiştirme:* r içerisinde kalan en düşük afiniteli d adet birey rastgele yeni antikorlarla yer değiştirilir.

3) Bitiş: Eğitim tamamlandıktan sonra, antijen havuzunun m bileşeni çözüm olarak alınır. Probleme göre çözüm tek bir antijen veya antijenlerden oluşan bir havuz olabilir.

Afinitesi yüksek olan n adet antikor, azalan sırada dizilir ve her antikor için oluşacak klon sayısı Eşitlik 3.5'e göre hesaplanır. β klonlama faktörünü, N antikor havuzu büyüklüğünü, i ise antikorun sıralamadaki sırasını gösterir. Eşitlik 3.6'da ise antijen başına düşen yeni oluşturulan klonların toplam sayısının hesaplanma yöntemi gösterilmektedir (Brownlee, 2005a).

$$\text{Klon sayısı} = \left\lfloor \frac{\beta N}{i} + 0.5 \right\rfloor \tag{3.5}$$

$$N_c = \sum_{i=1}^{n} \left\lfloor \frac{\beta N}{i} + 0.5 \right\rfloor \quad (3.6)$$

CLONALG algoritması, genetik algoritmalara yapı olarak benzese de fazla sayıda farklılık vardır. Örneğin, çaprazlama (crossover) bu algoritmada yoktur. Ayrıca, evrimsel yaklaşımlar gerçek değerli kodlama yaparken CLONALG ikili gösterimle çalışır, mutasyon Gaussian dağılımı ile kontrol edilmez (De Castro ve Timmis, 2003).

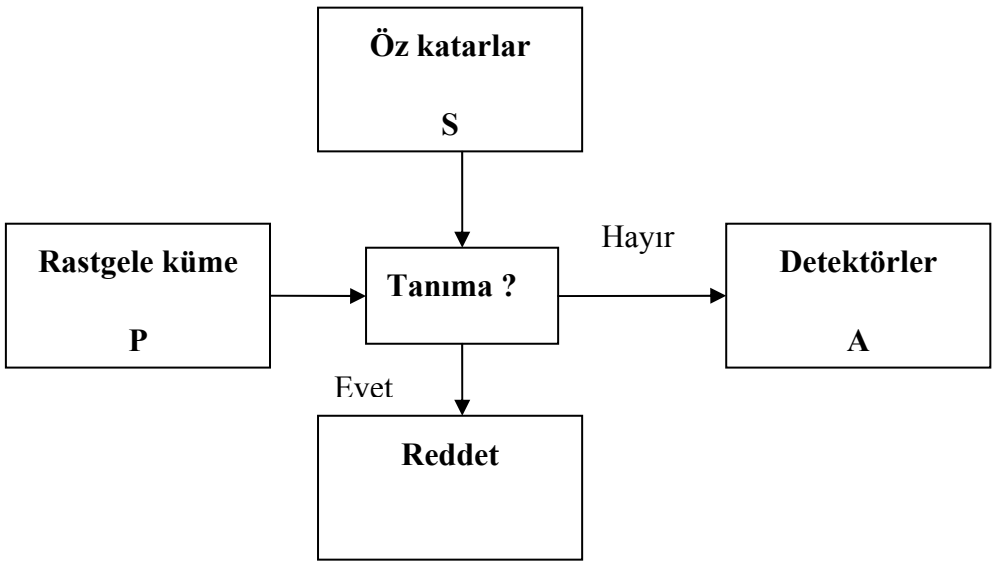
3.2.3.2 Negatif Seçim

T hücrelerinin negatif seçimi sayesinde, sadece yabancı antijenleri tanıyan T hücreleri bağışıklık sistemi içinde yer alır, öz antijenleri tanıyan T hücreleri yok edilir. Bu biyolojik olaydan esinlenerek Forrest ve arkadaşları (1994) "negatif seçim" adını verdikleri anomali tespit algoritması geliştirmişlerdir. Şekil 3.16'da bu algoritma temsili olarak gösterilmekte ve aşağıda açıklanmaktadır (De Castro ve Timmis, 2003):

1) Rastgele katarlar oluştur ve P kümesine yerleştir.

2) P içindeki tüm katarların afinitesini, öz kümesi olan S'deki katarlara göre belirle.

3) P içindeki bir katar ile S içinde en azından bir katar arasındaki afinite, eşik seviyesine eşit ya da ondan daha büyükse, P içindeki katar öz katarı tanımaktadır ve çıkarılması gerekir (negatif seçim); aksi durumda P içindeki katar yabancı kümeye aittir ve A kümesine eklenir.



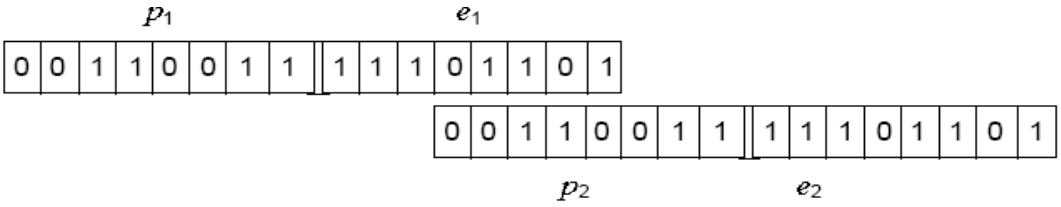
Şekil 3.16 Negatif seçim algoritması (De Castro ve Timmis, 2003)

3.2.3.3 Bağışıklık Ağ Modelleri

Jerne (1974), yabancı antijenlerin organizma içinde bulunmadığı durumda bile bağışıklık sisteminin dinamik bir yapı içerisinde olduğunu ve B hücrelerinin birbirini tanıyabildiğini ifade etmiştir. Bağışıklık biliminde çalışan uzmanlar bağışıklık sisteminin çalışmasını farklı ağ modelleriyle açıklamaya çalışmış, bu durum hesaplamalı zekâ alanında çalışanların yararlanabileceği modelleri doğurmuştur. İlk ağ modellerinde, popülasyon büyüklüğünü yönetmek üzere diferansiyel denklemler kullanılmış olup, bu modeller "sürekli bağışıklık ağı" (continuous immune network) olarak adlandırılmaktadır. Robotik, kontrol ve optimizasyonda bu tür ağlardan yararlanılmıştır (Ishiguro vd., 1997; Bersini ve Varela, 1994). Veri analizinde ağ yaklaşımları kullanılarak "ayrık bağışıklık ağı" (discrete immune network) ismi verilen ağlar geliştirilmiştir (De Castro ve Timmis, 2003). Bu tür ağlara örnek olarak, RAIN ve aiNET verilebilir.

3.2.3.4 Sürekli Bağışıklık Ağ Modeli

Farmer ve arkadaşları (1986), bağışıklık hücrelerini ve molekülleri Hamming şekil-uzayında bit katarları ile göstermiştir. Bir antikor molekülünün epitop (e) ve paratop (p) ismi verilen 2 ardışık bölümden oluştuğunu epitop bölümünün diğer antikorların paratop bölümü tarafından tanındığını ifade etmişlerdir (De Castro ve Timmis, 2003). Şekil 3.17'de örnek bir antikor molekülü için epitop ve paratop bölümleri gösterilmektedir.



Şekil 3.17 İki antikorda yer alan epitop ve paratop bölümleri (De Castro ve Timmis, 2003)

YBS içindeki her elemanın uyuşma derecesi m_{ij} olarak Eşitlik 3.7'ye göre hesaplanır. $e_i(n)$, i. epitopun n. bitini gösterirken $p_j(n)$, j. paratopun n. bitini gösterir. $\wedge$, $e_i(.)$, ve $p_j(.)$ arasındaki hamming uzaklığını ve ε, afinite eşiğini göstermektedir. k, bir paratop ve epitop arasında verilen hizalamaya (alignment) karşılık gelir, birden fazla hizalamada uyuşma olursa toplamlar alınmaktadır. G(.), epitop ve paratop arasında mümkün olan reaksiyon şiddetini ölçer. G(x), x>0 ise x değerini, aksi durumda 0 değerini alır.

$$m_{ij} = \sum_k G\left(\sum_n e_i(n+k) \wedge p_j(n) - \varepsilon + 1 \right) \tag{3.7}$$

Antikor konsantrasyonunun değişim oranı ise Eşitlik 3.8'e göre hesaplanmaktadır.

$$\frac{\partial c_i}{\partial t} = k_1\left(\sum_{j=1}^{N} m_{ji}c_i c_j - k_2\sum_{j=1}^{N} m_{ij}c_i c_j + \sum_{j=1}^{M} m_{ji}c_i y_j\right) - k_3 c_i \tag{3.8}$$

Eşitlik 3.8 için N adet antikorun $\{c_1, c_2, ...c_n\}$ konsantrasyonunda, M adet antijenin $\{y_1, y_2, ..., y_n\}$ konsantrasyonunda olduğu varsayılmıştır. Eşitliğe göre, antijenleri veya diğer antikorları tanıyan antikorlar çoğalır ve tanımayanlar elenir. Eşitlikteki ilk terim, i tipindeki antikorun paratopunun, j tipindeki antikorun epitopu tarafından uyarılma seviyesini gösterir. İkinci terim, i tipindeki antikorun epitopunun, j tipindeki antikorun paratopu tarafından tanındığı durumdaki baskılanma seviyesini gösterir. Üçüncü terim, antijenlerin konsantrasyonunu ve son terim hücrelerin ölme eğilimini modellemektedir. k_1, birim zamandaki çarpışma sayısına ve çarpışma neticesi uyarılan antikor oranına bağlı olan bir sabittir. k_2, uyarma ve baskılama arasındaki eşitsizliği temsil eden bir parametredir (De Castro ve Timmis, 2003).

3.2.3.5 Ayrık Bağışıklık Ağ Modelleri

Bu ağ modellerinde, sürekli ağ modellerinden farklı olarak diferansiyel denklemler yer almaz. RAIN ve aiNET algoritmaları ayrık bağışıklık ağ modellerinden olup B hücreleri ve antikorlar ağ modellerinin temel elemanlarıdır, antijenler ise girdi örüntüleridir (De Castro ve Timmis, 2003).

3.2.3.6 RAIN

RAIN (Resource Limited Artificial Immune Network) ağ modeli, Timmis tarafından geliştirilmiştir (Timmis ve Neal, 2000b). Ağın her elemanı; antikor (öklid şekil-uzayında özellik katarı), uyarılma seviyesi ve kaynaklardan oluşan bir B hücresine karşılık gelir. Başlangıçta; uyarılma seviyesi ve kaynak bilgileri sıfır olarak ilklendirilir ve antikorlar, antijenlerin (girdi örüntülerinin) bir alt kümesinin rastgele seçilmesi ile ilklendirilir. Sonrasında; her antijen ağ hücrelerine birer birer sunulur ve uyarılma seviyesi olan s_i, i hücresi için tüm antijenler sunulduktan sonra Eşitlik 3.9'a göre hesaplanır (De Castro ve Timmis, 2003).

M antijen sayısını, n bağlı B hücrelerinin sayısını göstermektedir. $D_{i,j}$ j isimli antijen ve i isimli B hücresi arasındaki öklid uzaklığını, $D_{i,k}$ i isimli hücre ve bağlı olduğu k isimli B hücresi arasındaki öklid uzaklığını temsil etmektedir. (1- $D_{i,j}$) bir B hücresinin antijenlerle veya ağ içerisindeki diğer B hücreleriyle arasındaki afinitesini göstermektedir. Bu hesaplamalarda, afinite uzaklıkla ters orantılıdır (De Castro ve Timmis, 2003).

$$s_i = \sum_{j=1}^{M}(1 - D_{i,j}) + \sum_{k=1}^{n}(1 - D_{i,k}) - \sum_{k=1}^{n} D_{i,k} \quad (3.9)$$

Uyarılma seviyesi olan s_i, klonal çoğalma için hangi hücrelerin seçileceğini ve hangilerinin ağdan silineceğini belirlemektedir. Uyarılma seviyesine göre hücrelere, sınırlı olan kaynaklardan atama yapılmaktadır ve ayrılan kaynak izin verilen sınırı aşıyorsa, sınır değere ulaşılana kadar en az kaynaklı hücreler ağdan çıkartılır. Uyarılma seviyelerine göre hücreler klonal çoğalma işlemine tabi tutulur. Klonal çoğalmadan sonra, klon antikorlara hipermutasyon uygulanarak antijene adapte olmaları sağlanır ve mutasyona uğramış klonların tüm ağ hücreleri ile afiniteleri hesaplanır. Afiniteler eşiğin altındaysa, hücreler arasında bağlantı kurulur. Bu işlem, sabit öz yineleme sayısına göre ya da ağ kararlı hale gelene kadar devam eder. Aşağıda RAIN, algoritmik olarak sunulmaktadır (De Castro ve Timmis, 2003):

1. *İlklendirme:* Antijenlerin bir alt kümesinden antikorları ilklendir.
2. *Antijenik sunum:* Her antijen için;
   - *2.1. Klonal seçim ve ağ etkileşimleri:* Her ağ hücresi için Eşitlik 3.9'a göre uyarılma seviyesini hesapla.
   - *2.2. Metadinamik:* Kaynak ayırma mekanizması ile düşük uyarılma seviyeli ağ hücrelerini yok et.
   - *2.3. Klonal çoğalma:* En fazla uyarılan ağ hücrelerini seç ve uyarılma seviyesi ile orantılı olarak yeniden üret.
   - *2.4. Somatik hipermutasyon:* Her klonu uyarılma seviyesi ile ters orantılı şekilde mutasyon işlemine tabi tut.
   - *2.5. Ağ oluşturma:* Mutasyona uğramış klonları ağın içine eklemek için seç.
3. *Çevrim:* 2. adımı durma kriteri sağlanana kadar devam ettir.

3.2.3.7 aiNET

aiNET algoritması De Castro ve Von Zuben tarafından geliştirilmiştir (2000b). Bu ağ modelinde, ağın her bir elemanı bir antikor molekülüne karşılık gelir (öklid şekil-uzayında temsil edilen bir özellik katarı). Sonrasında antijenler ağ hücrelerine birer birer sunularak Eşitlik 3.1'e göre afiniteler hesaplanır. Afinitesi yüksek olan antikorlar afinitesi oranında klonlanır. Klonal çoğalmanın ardından, somatik hipermutasyon işlemi gerçekleştirilir ve

yüksek afiniteli klonlar ağda kalarak klonal bellek oluşturulur. Geride kalan antikorlar arasındaki afiniteler belirlenir ve afinitesi eşik seviyesinin altında kalan antikorlar ağdan çıkartılır. Bu işlem klonal baskılama (clonal supression) olarak ifade edilir. Ek olarak rastgele oluşturulan yeni antikorlar, çıkarılan antikorlar yerine eklenir. Bu antikorların varolan antikorlar ile afinitesi belirlenir ve afinitesi eşikten küçük olan antikorlar çıkarılır. Aşağıda RAIN, algoritmik olarak sunulmaktadır (De Castro ve Timmis, 2003):

1. *İlklendirme:* Antikorlardan rastgele başlangıç populasyonu oluştur.
2. *Antijenik sunum:* Her antijen için;
   - *2.1. Klonal seçim ve çoğalma:* Her ağ elemanı için sunulan antijenle arasındaki afiniteyi belirle. Belirli sayıda yüksek afiniteli elemanı seçerek afinitesi ile orantılı olarak klonla.
   - *2.2. Afinite olgunlaşması:* Her klonu afinite ile ters orantılı olarak mutasyona uğrat. Belirli sayıda en yüksek afiniteli klonları yeniden seç ve klonal bellek kümesine yerleştir.
   - *2.3. Klonal etkileşimler:* Klonal bellek kümesinin tüm elemanları arasında afinite değerlerini belirle.
   - *2.4. Klonal baskılama:* Afinitesi eşik seviyesinin altında olan bellek klonlarını yok et.
   - *2.5. Metadinamik:* Antijen ile arasındaki afinite bir eşik seviyesinden küçük olan tüm bellek klonlarını yok et.
   - *2.6. Ağ oluşturma:* Klonal belleğin geride kalan klonlarını tüm ağ antikorları ile bir araya getir.
   - *2.7. Ağ etkileşimleri:* Ağ antikorlarının her çifti arasındaki benzerliği belirle.
   - *2.8. Ağ baskılama:* Afinitesi belirli bir eşik seviyesinin altında olan tüm ağ antikorlarını yok et.
3. *Çevrim:* 2. ve 3. adımları öz yineleme sayısına ulaşılana kadar tekrarla.

3.2.3.8 Ayrık Ağ Modellerinin Kıyaslanması

RAIN ve aiNET algoritmaları benzer görünse de aralarında aşağıdaki temel farkılıklar mevcuttur (De Castro ve Timmis, 2003):

- Temel ağ elemanı iki algoritmada farklı olarak ele alınmıştır. RAIN içinde; temel

eleman olan B hücresi antikor özellik katarı, uyarılma seviyesi, kaynak ayırma bilgisi gibi bilgiler taşımaktadır. aiNET içerisinde temel eleman sadece antikora ilişkin özellik katarıdır.

- Ağ içindeki her B hücresinin uyarılma seviyesi, RAIN'de diferansiyel denklemle hesaplanırken aiNET'de bu seviye tek denklemle hesaplanmaz, öğrenme sürecinde farklı zamanlarda gerçekleştirilir.

- RAIN, yüksek seviyede uyarılmış B hücrelerinin hayatta kalmasını teşvik eder. aiNET ise benzer antikorların yok edilmesini sağlayarak artıklığı azaltmaya çalışır. Bu işlem de diğer ağ antikorları ile afiniteler hesaplanarak saptanır. İki yöntem de popülasyon büyüklüğünü kontrol etmek için kullanılır.

3.3 Yapay Bağışıklık Sistem Tabanlı Sınıflandırma Algoritmaları

Bu bölümde, çalışmalarda kullanılan Yapay Bağışıklık Sistem tabanlı sınıflandırma algoritmaları açıklanmaktadır.

3.3.1.1 Yapay Bağışıklık Tanıma Sistemi (AIRS) Algoritması

Watkins (2001), sınıflandırma amaçlı Yapay Bağışıklık Tanıma Sistemi (Artificial Immune Recognition System-AIRS) algoritmasını geliştirdikten sonra, çok fazla bilim insanı algoritmayı iyileştirmek üzere araştırmalar gerçekleştirdi. Algoritmik karmaşıklığı azaltılmış ve doğruluğunda çok fazla azalma olmayan yeni bir algoritma, AIRS versiyon 2 olarak tanımlandı. Watkins, daha sonra AIRS algoritmasının paralel halini doktora çalışmaları kapsamında geliştirdi ancak bu algoritmanın üzerinde daha fazla çalışılması gerekiyor.

Algoritmadaki antijen kavramı, veri kümesindeki her bir satıra ya da daha farklı bir ifadeyle bir modülün metriklerine karşı gelmektedir. Yapay Tanıma Küreleri (ARB-Artificial Recognition Ball) ise birbiri ile benzer özellik gösteren modülleri içeren kümelerdir.

AIRS algoritması aşağıda sıralanan güçlü özelliklere sahiptir:

- *Genelleştirme:* Genelleştirme için tüm veri kümesi kullanılmaz ve algoritma, veri azaltma yeteneğine sahiptir.

- *Parametre Kararlılığı:* Kullanıcı tanımlı parametreler optimize edilmese bile, performansındaki değişim çok küçüktür.

- *Performans:* Birçok veri kümesinde, yüksek performans elde edilmiştir.

- Kendini-düzenleme (self-regulatory): Eğitimden önce topoloji belirlemeye gerek yoktur.

Brownlee (2005b), bu algoritmayı Java dili ile gerçeklemiş ve GPL (General Public Licence) lisansı ile kodlarını http://wekaclassalgos.sourceforge.net sitesinden erişilebilir kılmıştır. Algoritma 5 adımdan oluşmaktadır: İlklendirme, antijen eğitimi, sınırlı kaynak için yarış, bellek hücre seçimi ve sınıflandırma. 1. ve son adımlar bir kez uygulanırken; 2., 3., 4. adımlar eğitim kümesindeki her örnek için uygulanır. AIRS algoritmasında örneğe, antijen adı verilmektedir. Algoritmanın temel amacı, eğitim kümesini modellemek için tanıma hücreleri havuzu oluşturmak ve bilinmeyen veriyi bu havuza göre sınıflamaktır.

Algoritma 5 aşamadan oluşmaktadır (Brownlee, 2005b):

- *İlklendirme:* Veri kümesi [0,1] aralığına normalize edilir. Afinite eşik değişkeni hesaplanır. Ayrıca, iki vektör arasındaki maksimum uzaklık 0 ve 1 arasında olmalıdır. Bu nedenle; iki antijen arasındaki afinite değeri, distance'ın maxDistance'a bölünmesi ile elde edilir. distance, öklid uzaklığı ile hesaplanmaktadır.

- *Antijen Eğitimi:* Antijenler bellek havuzuna birer birer sunulur. Bellek havuzundaki tanıma hücreleri uyarılarak bu hücrelere uyarılma değeri atanır. En yüksek uyarılma değerine sahip olan tanıma hücresi, en iyi uyuşan bellek hücresi olarak seçilir ve afinite olgunluk sürecinde kullanılır. Bu hücre klonlanır ve mutasyona tabi tutulur. Klonlar, Yapay Tanıma Küreleri havuzuna eklenir. 3.11 nolu eşitlikle klonların sayısı hesaplanır. 3.10 nolu eşitlikle, stim değerinin nasıl hesaplanacağı gösterilmektedir. clonalRate ve hyperMutationRate; kullanıcı tanımlı parametrelerdir.

  $$\text{stim} = 1 - \text{affinity} \tag{3.10}$$

  $$\text{numClones} = \text{stim} * \text{clonalRate} * \text{hyperMutRate} \tag{3.11}$$

- *Sınırlı Kaynak için Yarış:* ARB havuzuna, mutasyona uğramış klonlar eklendiği zaman yarışma başlar. ARB havuzu antijen ile uyarılır ve uyarılma değerine göre sınırlı kaynak ataması yapılır. Yeterli kaynağı olmayan ARB'lar havuzdan atılır. Durma kriteri sağlanınca, işlem durdurulur. Aksi halde, ARB'ın mutasyona uğramış klonları oluşturulur ve bu döngü durma kriteri sağlanana kadar devam eder.

- *Bellek hücre seçimi:* Bir önceki adımda, durma kriteri sağlandığı zaman, en yüksek uyarılma değerine sahip ARB, aday bellek hücre olarak seçilir. ARB'ın uyarılma

değeri orjinal en iyi uyuşan hücreden daha iyiyse, ARB bellek hücre havuzuna kopyalanır.

- *Sınıflandırma:* Bellek hücre havuzunun son hali çapraz geçerlemede kullanılır. K en yakın komşu yaklaşımına göre sınıflandırma gerçekleştirilir.

3.3.1.2 AIRS v1.0 ve AIRS v2.0 Algoritmaları Arasındaki Temel Farklılıklar

AIRS v1.0 ve AIRS v2.0 algoritmalarının temel adımları aynıdır ancak algoritmada küçük değişiklikler yapılarak karmaşıklığın azaltılması amaçlanmıştır. Bu bölümde, bu farklar açıklanmaktadır (Brownlee, 2005b).

AIRS v2.0 algoritması, AIRS v1.0'a göre daha basittir ve daha iyi veri azaltma yeteneğine sahiptir. AIRS v2.0'ın karmaşıklığı AIRS v1.0'a göre daha düşüktür, performanstaki azalma ise önemli ölçülerde değildir. Bu bilgilerden hareketle; karmaşıklık ve işlem gücünün önemli olmadığı durumlarda, AIRS v1.0 ile biraz daha yüksek performans alınacağı ifade edilebilir. Algoritmalar arasındaki farklar aşağıda sıralanmaktadır:

- AIRS v1.0 algoritmasında ARB havuzu, kalıcı bir kaynak olarak kullanılırken AIRS v2.0 algoritmasında, geçici bir kaynak olarak kullanılır. Kalıcı kaynak olarak kullanıldığı durumda bir önceki adımlardan kalan ARB'lar, sınırlı kaynak için yarışmaya dahil olarak, algoritmanın daha fazla zaman harcamasına neden olur. Bu nedenle, AIRS v2.0 algoritmasının karmaşıklığı daha azdır.

- AIRS v1.0 algoritmasında, mutasyon sürecinden sonra klonların sınıfları değişebilirken AIRS v2.0 algoritmasında sınıfların değişebilmesine izin verilmemektedir.

- AIRS v1.0, kullanıcı tanımlı mutasyon parametresini kullanırken AIRS v2.0, somatik hipermutasyon kavramını kullanır. Bir klonun uğradığı mutasyon oranı, afinite ile orantılıdır.

3.3.1.3 Paralel AIRS

Çok az sayıda algoritmada, doğal bağışıklık sisteminin paralel işleme özelliğinden esinlenilmiştir. Watkins (2005), doktora tez çalışmasında AIRS algoritmasının farklı işlemciler üzerinde dağıtılarak çalıştırılabileceği bir sürümünü geliştirmiştir.

AIRS algoritmasının paralelleştirilmesi, standart eğitim sürecine ek olarak aşağıdaki adımların da gerçekleştirilmesini gerektiriyordu ve yaklaşım oldukça basitti (Brownlee,

2005b): “

1. Eğitim veri kümesini, np adet parçaya böl (np, AIRS algoritmasını çalıştıracak proseslerin sayısıdır).
2. Eğitim veri kümesinin her bir parçasını bir prosese ata.
3. Her proses üzerinde, AIRS algoritmasının seri sürümünü çalıştır ve np adet bellek havuzunun oluşmasını bekle.
4. Ana (master) bellek havuzunu oluşturmak üzere, bir ‘birleştirme şeması’ (mergining schema) kullanarak bellek havuzlarını birleştir.

Veri kümesi çok küçük parçalara ayrılmadığı durumda, bu yaklaşım performansı korurken hızlı bir yaklaşımın sunulmasını sağlamaktadır. Birleştirme teknikleri, veri azaltma yüzdesi açısından daha kötü sonuçlar ortaya çıkmasına neden olmuştur ve bu problemin çözülmesi için ek araştırmalar gerçekleştirilmelidir (Brownlee, 2005b).

3.3.1.4 Immunos Algoritmaları

Immunos81 algoritması, doğal bağışıklık sisteminden esinlenen ilk sınıflandırma algoritmasıdır ve bir tıp doktoru olan Carter (2000) tarafından geliştirilmiştir.

Brownlee (2005c), Immunos81 algoritmasının sunduğu performans değerlerini elde edebilmek için Immunos1 ve Immunos2 isimlerini verdiği iki farklı programı Java ile geliştirmiştir. Immunos81 isimli algoritmada, bazı noktaların çok açık olmaması nedeniyle Brownlee, algoritmanın geliştiricisi olan Carter ile iletişime geçmeye çalışmış ancak başarılı olamamıştır.

Bu belirsizlikler yüzünden, Immunos81 algoritmasının iki farklı Java gerçeklemesi ortaya çıkmıştır. Brownlee (2005c), Carter’ın tıp doktoru olması sebebiyle algoritmadaki bazı noktaları, bilgisayar bilimleri dergilerinde yayımlanan makalelerdeki gibi açık şekilde ifade edemediğini belirtmiş ve bu yönde eleştirisini teknik raporunda sunmuştur.

“Immunos81 algoritması; antijenleri, T ve B hücrelerini, klonları ve bir amino asit kütüphanesini içermektedir. Bağışıklık sistemi içerisindeki temel kavramlar ve bu kavramların Immunos81 içerisindeki karşılıkları Çizelge 3.1’de verilmektedir.

Immunos81 algoritması iki standart veri kümesi ile test edilmiştir ve bu veri kümeleri tıp alanına aittir. Immunos81, B hücrelerinin üretimini kontrol etmek için T hücrelerini

kullanmaktadır. B hücreleri sırasıyla bilinmeyen örüntülerin tanınması için yarışmaktadır. Amino asit kütüphanesi, sistem içerisindeki epitop (veya değişkenler) kütüphanesi olarak davranır. Yeni bir antijen sisteme girdiği zaman, bu kütüphaneye değişkenleri girilir. T hücreleri daha sonra reseptörlerini oluşturmak için bu kütüphaneyi kullanır ve bu reseptörler yeni bir antijeni belirlemek için kullanılır.

Algoritmanın tanıma adımı sürecinde, T hücre paratopları antijenin epitopları ile eşleşir ve bir B hücresi daha sonra oluşturulur. B hücresinin sahip olduğu paratoplar da, antijenin epitopları ile eşleşebilmektedir. T hücreleri ve antijen bağlanma bölümleri, Hamming şekil uzayında ikili katarlar olarak gösterilir ve antijenle T hücresi arasındaki afinite, aralarındaki benzerlikle orantılıdır" (De Castro ve Timmis, 2002).

Çizelge 3.1 Bağışıklık Sistemi ve Immunos81 Arasındaki Dönüşüm

Bağışıklık Sistemi	**Immunos81**
Antijen	Herhangi bir tipte birden fazla değişken içerebilecek veri
T hücresi	Özel bir sınıfı temsil eden ve antijen içerisindeki değişken tiplerini ve sıralarını belirleyen kontrol ajanı
B hücresi	Öğrenme sürecinde özel bir sınıfın bir örneğini temsil eden varlık
Amino asit kütüphanesi	Karşılaşılan (bir isim ve bir tip atanan) ve yeni antijenler oluşturmak için kullanılabilen tüm değişkenler
Klon	Tanıma ajanı, B hücrelerinin bir repertuvarının matematiksel gösterimi

3.3.1.5 CLONALG

Bölüm 3.2.3.1'de açıklandığı için bu bölümde tekrar açıklanmayacaktır. Çalışmalar süresince, bu algoritmadan da yararlanılmıştır.

3.4 AIRS Algoritmasının Detaylı Açıklanması

Bu bölümde AIRS algoritması içerisinde kullanılan terim ve notasyonlar hakkında bilgi verilmiş ve algoritmanın adımları detaylı bir şekilde sunulmuştur.

3.4.1 AIRS Algoritması Temel Bilgiler

AIRS algoritmasının, doğal bağışıklık sistemlerindeki kavramlarla ilişkisi bu başlık altında üst seviyede ele alınıp, takip eden bölümlerde algoritmik ifadeler ile açıklanmıştır. Ayrıca, yazılım kusur kestirimi için gerekli veri kümelerinin AIRS içerisinde nasıl kullanıldığı bu bölüm içerisinde verilmektedir.

AIRS algoritması bağışıklık biliminin temel prensiplerinden esinlenilerek oluşturulmuş, ancak bire bir modelleme yerine çeşitli soyutlamalar sayesinde geliştirilmiş bir algoritmadır. Algoritma içerisinde kullanılan antijen terimi, veri kümesindeki her bir satırı temsil etmektedir. Yazılım kusur kestirim veri kümesi için şekil 3.18'deki gibi bir veri kümesi mevcut olması durumunda; veri kümesinin her bir satırı AIRS içerisinde antijen olarak değerlendirilmektedir. Örnek olarak şekil 3.18'de sunulan veri kümesi, WEKA içerisinde sıkça kullanılan ARFF formatında oluşturulmuştur. 5 adımlı olan AIRS algoritmasının, 2, 3 ve 4. adımları bu veri kümesindeki her bir satır için gerçekleştirilmektedir. Algoritmanın ilk adımı olan ilklendirme ve son adımı olan sınıflandırma, sadece bir kez gerçekleştirilmektedir.

```
@RELATION KusurVeriKumesi

@ATTRIBUTE CevrimselKarmasiklik  REAL

@ATTRIBUTE KodSatirSayisi        REAL

@ATTRIBUTE EşsizOperandSayisi    REAL

@ATTRIBUTE EşsizOperatorSayisi   REAL

@ATTRIBUTE sinif  {evet, hayir}

@DATA

40.0,220.0,140.0,240.0,evet

2.0,12.0,8.0,8.0,hayir

65.1,42.0,430.0,260.0,evet

3.0,10.0,13.0,8.0,hayir
```

Şekil 3.18 Örnek arff uzantılı veri kümesi

`@DATA` ifadesinden sonra yazılan her bir satır, bir önceki yazılım sürümündeki her bir modülün (metot veya sınıf) metriklerini ve sınıf etiketini göstermektedir. {evet} etiketi, ilgili modülde test aşamasında kusur çıktığını, {hayir} etiketi kusur çıkmadığını ifade etmektedir. Bu örnek veri kümesinde, 4 adet metot seviyesinde metrik mevcuttur (çevrimsel karmaşıklık, kod satır sayısı, eşsiz operand sayısı, eşsiz operatör sayısı) ve her modülün sahip olduğu metrik değeri, ilgili satırda verilmektedir.

Doğal bağışıklık sistemlerinde, vücuda dışarıdan giren zararlı antijenler, veri kümesinin her bir satırı ile temsil edilmiş olmaktadır. Onları tanıyan vücuttaki antikorlar ise algoritmanın ilk aşamasında rastgele seçilmiş bir vektör olarak tanımlanmaktadır. Rastgele seçilen bu vektör, AIRS içerisinde veri kümesindeki herhangi bir satırın seçilmesi ile belirlenir. Birden fazla vektör de seçilebilirken, ilklendirmede bir adet seçilmesi önerilmektedir (Brownlee, 2005b).

Algoritmanın başlangıcında, bellek hücre havuzu (memory cells pool) bu antikorları içermekte olup, eğitim aşamasının sonunda bellek hücre havuzu, sonraki aşamalarda (makine öğrenmesi açısından sonraki aşamalar test aşamasıdır ve doğal bağışıklık sistemleri açısından, antijenin vücuda tekrar girmesidir) kullanılacak olan bellek hücrelerini içerecektir. Bu nedenle, algoritmada bellek hücre havuzu dinamik ve evrimleşen şekilde gelişmektedir.

Algoritmada kullanılan bir diğer havuz da, Yapay Tanıma Küresi (Artificial Recognition Ball-ARB) havuzudur. ARB içerisinde evrimleşen ve ilgili antijen ile en iyi uyuşan ARB'ın belirli kontrollere göre, yerleşik bellek hücresi (established memory cell) olarak bellek hücre havuzuna aktarılması söz konusudur. Bu sayede, her bir antijen için bellek hücre havuzuna yerleşik bir bellek hücresi atanması gerçekleşmektedir, ancak AIRS algoritmasının veri azaltma özelliği sayesinde, bu durum bire bir şekilde gerçekleşmemektedir. Bu özellik AIRS'in veri azaltarak, öğrenme gerçekleştirdiğini ortaya koymaktadır.

Algoritmanın ilk 4 adımı tamamlandıktan sonra, eğitim süreci tamamlanmış olmaktadır ve eğitim sürecinin ardından bellek hücreleri, bellek hücre havuzunda oluşmuş olacaktır. AIRS algoritması aşağıdaki 5 temel adımdan oluşur ve 2–3–4 numaralı adımlar, her bir antijen için döngü şeklinde gerçekleştirilir. AIRS algoritmasının temel adımları aşağıda sunulmaktadır:

1. İlklendirme (Initialization)
2. Bellek hücresinin belirlenmesi ve ARB üretimi (Memory cell identification and ARB generation)
3. Kaynaklar için yarışma ve aday bellek hücrenin geliştirilmesi (Competition for

resources and development of a candidate memory cell)

4. Bellek hücresinin tanıtılması (Memory cell introduction)
5. Sınıflandırma

Farklı kaynaklarda; ikinci adım “antijen eğitimi”, üçüncü adım “sınırlı kaynak için yarışma”, dördüncü adım ise “bellek hücre seçimi” başlıklarıyla açıklanmaktadır. Başlıklar farklı olmasına rağmen, bu kaynaklardaki açıklamalar aynı işlemleri tanımlamaktadır.

Doğal bağışıklık sistemlerinde; vücuda dışarıdan giren antijenle uyuşan antikor, klonlama (mitoz) ile sayısını çoğaltır ve ardından mutasyon işlemi gerçekleştirilerek klon içinde ölçeği çok büyük olmayan farklılıklar oluşur.

Seçici mekanizma sayesinde, antijeni daha iyi tanıyan hücreler bellek hücreleri haline dönüşür ve antijenler sonraki saldırılarda kolayca tanınır. Antikorları salgılayan B hücrelerinin yüzeylerinde ise reseptör moleküller olarak BCR (antikor) bulunmaktadır. Bir B hücresinde, yüzlerce antikor molekülü bulunabilir ancak modellerde Yapay Tanıma Küresi (ARB) bir adet antikor içerecek şekilde modellenmiştir, aksi durumda aynı verinin tekrarlanması gibi bir problem ortaya çıkardı. ARB terimi, doğal bağışıklık sistemlerindeki B hücrelerinin temsilini sağlamaktadır ve bir ARB AIRS algoritması içerisinde; bir antikor, sahip olduğu kaynak sayısı, uyarılma değeri bilgilerine sahiptir.

Algoritmanın başlangıcında; bellek hücre havuzu (M) ve Yapay Tanıma Küresi (ARB) havuzu (P), rastgele değerlerle başlatılır. Bu rastgele vektör, veri kümesinde bulunan bir veri (satırdaki veriler) kullanılarak belirlenir. Bu nedenle, ilk aşamada bellek hücre havuzunda antikorlar bulunduğu ve veri kümesinde ise antijenlerin yer aldığı düşünülebilir. Bu tarz bir düşünüş tarzı aynen doğal bağışıklık sistemlerinde olan kavramların algoritmaya taşınmasını sağlamaktadır.

İlklendirme adımından sonra, her bir antijen bellek hücre havuzuna yani antikorlara sunularak aralarındaki uyuşma derecesi belirlenir. Bu ölçüm, vektörler arasındaki Öklid uzaklığı ile ters orantılıdır. Öklid uzaklığının 1 değerinden çıkarılması ile uyarılma derecesi belirlenmektedir. Öklid uzaklık değerinin düşük olması durumunda, uyarılma değeri daha yüksek olacaktır ve yüksek seviyede bir uyarılmanın olduğu ortaya konulmuş olacaktır. Bu sayede, veri kümesindeki ilgili veri (antijen) ile en iyi uyuşan antikor saptanır, bu antikor klonlama ve mutasyona maruz bırakılarak, oluşan çocuklar ARB havuzuna eklenir. Algoritmanın ikinci adımı bu noktada tamamlanmaktadır. Bu adımdan sonra, ARB havuzu içerisinde sınırlı

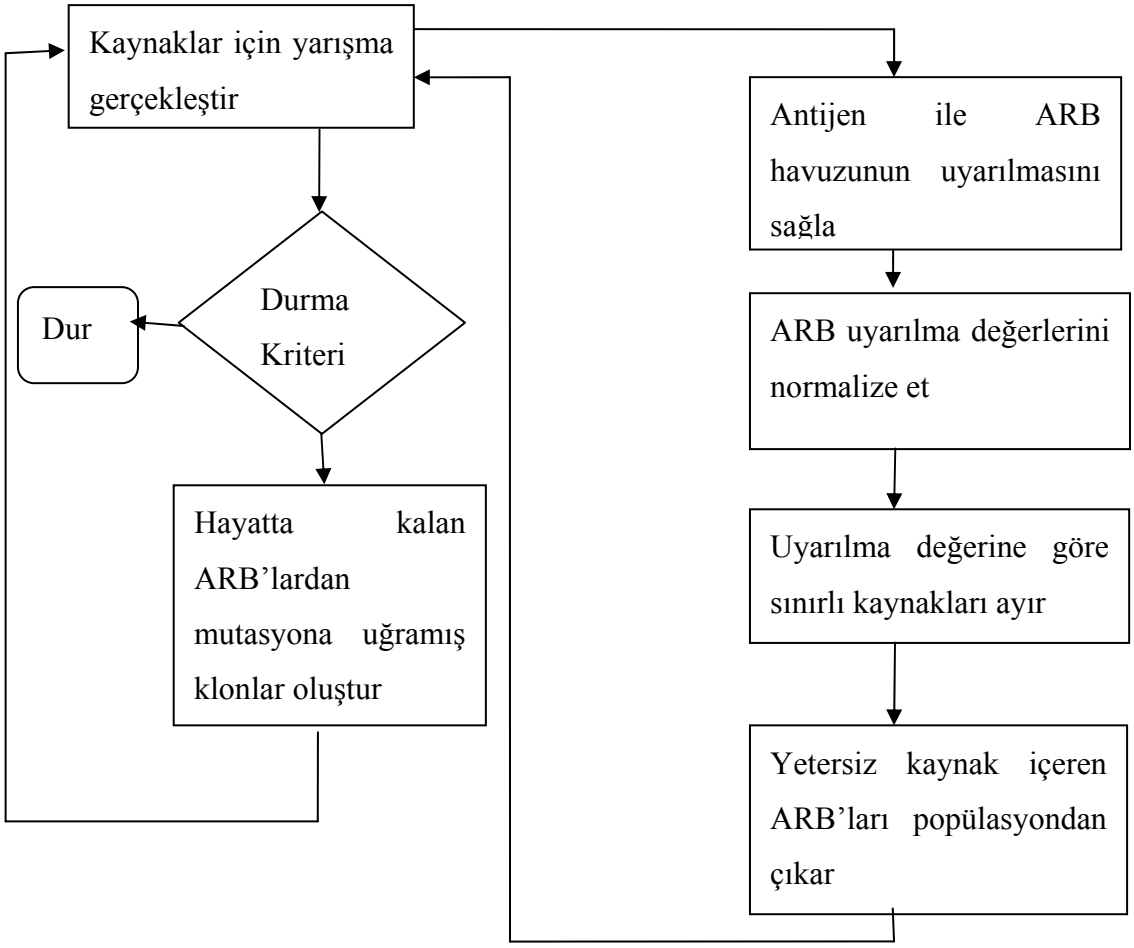
kaynak için yarışma başlar ve ARB'ların kullanabileceği tüm kaynak sayısı sistem üzerinde sınırlıdır. ARB havuzu içerisindeki ARB'lar da bu antijene maruz bırakılır ve uyuşma derecesine göre kaynak atamaları yapılır. Yeterli kaynağı olmayanlar ARB havuzundan çıkartılır. Durma kriteri sağlanmadıysa, hayatta kalmış olan her ARB'ın mutasyona uğramış çocuklar üretmesine izin verilir ve bu işlem durma kriteri sağlanana kadar devam eder.

Durma kriteri; ARB'ların ilgili antijen tarafından yeterince uyarılıp uyarılmadığını kontrol etmektedir. 4. adımda; en yüksek uyarılma değerine sahip olan ARB'ın (B hücresi) aday bellek hücresi (candidate memory cell) olarak belirlenmesi sağlanır. Bu "aday bellek hücresinin" antijenle uyuşma derecesi, algoritmanın 2. adımı sonunda belirlenmiş olan "en iyi uyuşan bellek hücresinin" aynı antijenle gösterdiği uyuşma derecesinden daha yüksek olması durumunda, bu aday bellek hücresinin, bellek hücre havuzuna alınması işlemi gerçekleştirilir. Algoritmanın gerçeklenmesi sırasında, bu aşamada bir kontrol daha yapılmaktadır, bu kontrol ve algoritmik yapı bölüm 3.4.2'de açıklanmaktadır.

Anlaşılırlığı bozmamak için bu bölümde tüm algoritmik ayrıntılar verilmemektedir. Aday bellek hücresi, bellek hücre havuzuna yerleşik hücre olarak alındıktan sonra, test aşamasında test verisi ile arasındaki uyuşma derecesi belirlenecektir.

Sunulan açıklamalara göre; bellek hücre havuzu, algoritmanın 2. aşamasında antikor havuzu gibi görünürken, 4. aşamanın sonunda yani eğitim sonunda, ismindeki gibi, bellek hücre havuzu gibi çalışmaktadır.

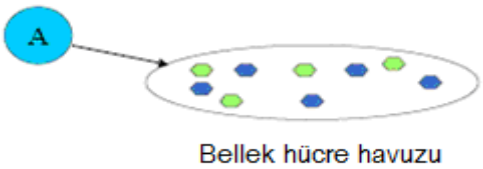
Bu açıklama; doğal bağışıklık sistemleri açısından yapılan bir değerlendirmedir. Algoritmanın son aşamasında ise test edilecek veri (antijen), bellek hücre havuzundaki hücrelere sunulur ve k tane en fazla uyarılan bellek hücresinin etiketlerinin çokluğuna göre sınıf etiketi belirlenir. Algoritmanın 3. adımı şekil 3.19'da resmedilmektedir.



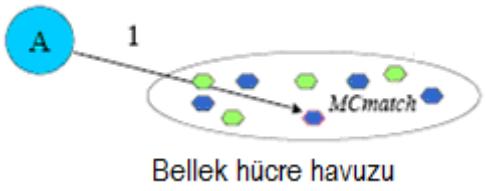
Şekil 3.19 ARB hücre havuzunda sınırlı kaynak ile iyileştirme (Brownlee, 2005b)

3.4.2 AIRS İçerisindeki Eğitim Sürecinin Şekiller ile Açıklanması

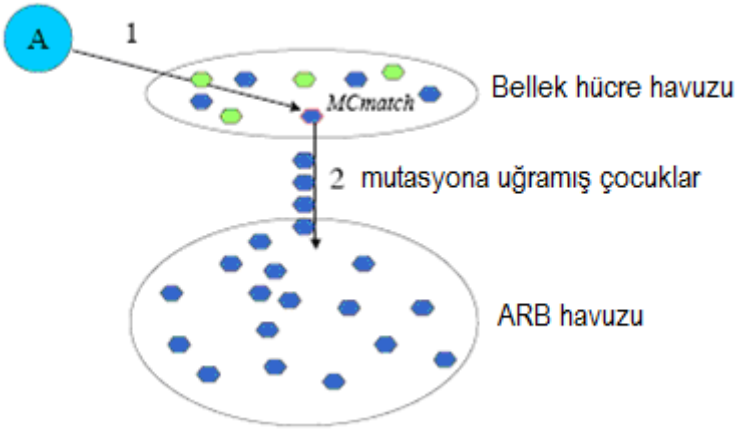
Algoritmanın ilk adımı olan ilklendirme şekillerde gösterilmemiştir. Şekil 3.20, 3.21, 3.22'de algoritmanın 2. adımı; Şekil 3.23, 3.24'te algoritmanın 3. adımı; Şekil 3.25 ve 3.26'da algoritmanın 4. adımı gösterilmektedir. Algoritmanın 5. adımında ise sınıflandırma yapılmaktadır. Dolayısı ile bu şekillerde eğitim süreci yansıtılmaktadır.



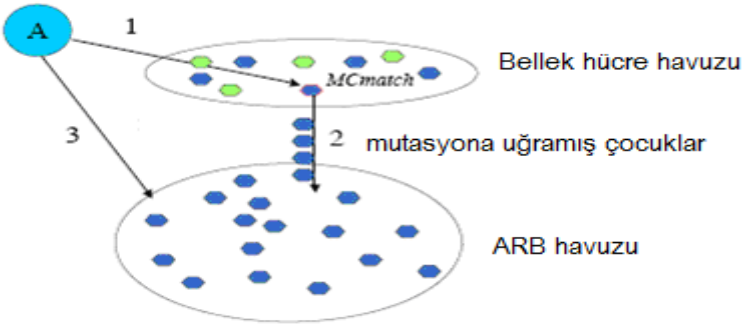
Şekil 3.20 ARB hücre havuzunda sınırlı kaynak ile iyileştirme (Timmis, 2008)



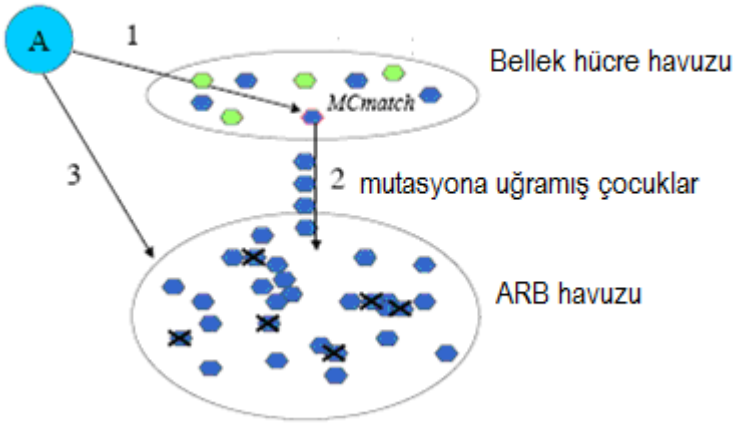
Şekil 3.21 Antijenle en iyi uyuşan bellek hücresinin (MCmatch) bulunması (Timmis, 2008)



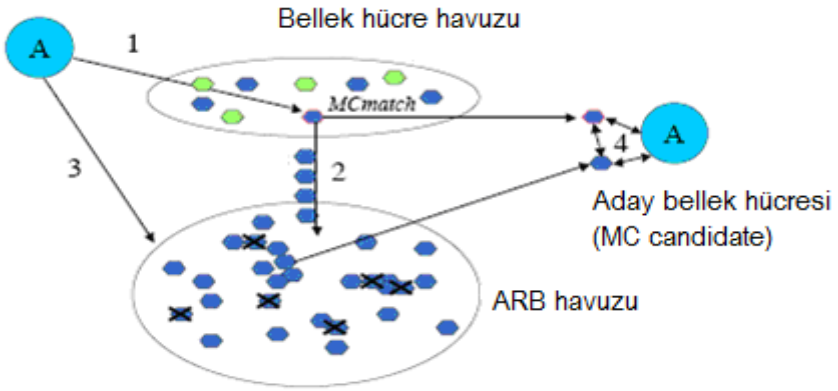
Şekil 3.22 ARB havuzuna mutasyona uğramış çocukların eklenmesi (Timmis, 2008)



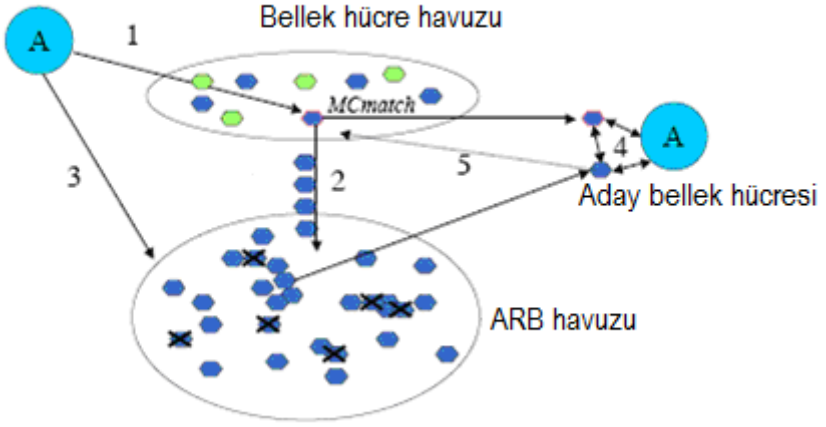
Şekil 3.23 ARB'ların antijene maruz bırakılması (Timmis, 2008)



Şekil 3.24 Aday bellek hücresinin geliştirilmesi (Timmis, 2008)



Şekil 3.25 En iyi uyuşan hücre ile aday bellek hücresinin karşılaştırılması (Timmis, 2008)



Şekil 3.26 Bellek hücre havuzuna aday bellek hücresinin girişi (Timmis, 2008)

3.4.3 AIRS Algoritması İçerisindeki Terimler

Bu bölümden itibaren sunulan bilgiler, AIRS algoritmasını geliştiren Andrew Watkins 'in (Watkins, 2001) yüksek lisans tezindeki bilgiler kullanılarak hazırlanmıştır. Bu sayede, daha sağlıklı bilginin ortaya konulması hedeflenmiştir.

Afinite (affinity): Antijen ve antikor arasındaki benzerliğin veya yakınlığın ölçüsüdür. Mevcut gerçeklemede, bu değerin 0–1 arasında olması garanti edilir ve 2 nesnenin özellik vektörleri (feature vectors) arasındaki Öklid uzaklığı hesaplanarak belirlenir. Bu nedenle, düşük afinite değeri güçlü bir afinite olduğunu ortaya koyar.

Afinite eşiği (affinity threshold-AT): Eğitim kümesi içerisindeki tüm antijenler arasındaki ortalama afinite değeridir (ya da seçilen bir alt küme içerisindeki).

Afinite eşik skaleri (affinity threshold scalar-ATS): 0–1 arasında bir değerdir ve AT değeri ile çarpılınca elde edilen bu değer (AT*ATS), AIRS algoritmasının eğitim aşamasında "bellek hücresi yer değişimi" (memory cell replacement) için "kesme değeri" (cut-off value) olarak kullanılır.

Antikor (antibody): Sınıf bilgisini de taşıyan özellik vektörüdür (feature vector).

Antijen (antigen): Antikordaki gibi bir temsile (representation) sahiptir ancak bu vektör, ARB'lara etkileşim için sunulduğu zaman bu adı alır.

ARB (artificial recognition ball - yapay tanıma küresi): B hücresi (B-cell) olarak da bilinir. Bir ARB; bir antikor, hücrenin sahip olduğu kaynak sayısını gösteren bir değer ve hücrenin mevcut uyarılma değeri (stimulation value) bilgisini içerir.

B hücresi: Daha yaygın kullanımı ARB terimidir (bkz. ARB).

Aday bellek hücre (candidate memory cell): Bir ARB'ın antikorudur. Eğitim amaçlı kullanılan antijenle aynı sınıfa sahip olan ve bu antijenden en fazla uyarılan hücredir.

Sınıf (class): Bir özellik vektörünün kategorisidir ve bir hücrenin çıktısı (output) da denilir.

Klonlama oranı (clonal rate): Bir ARB'ın üretmesine izin verilen, mutasyona uğramış klonlarının sayısını belirleyen tamsayı bir değerdir. Mevcut gerçeklemede; antijene yanıt oluştuktan sonra, bir ARB'ın (klonlama oranı * uyarılma değeri) sayısına kadar mutasyona uğramış klon üretmesine izin verilir. Bu çarpım; aynı zamanda bir ARB'a kaynakların atanmasında da kullanılmaktadır, bu nedenle çift role sahip olan bir oran olarak açıklanabilir. Kaynak ayırma faktörü ve klonlama mutasyon faktörü olarak kullanılır.

Yerleşik bellek hücresi (established memory cell): Sınırlı kaynak için olan yarışmada hayatta kalan ve ilgili antijenle en fazla uyarılan, evrimleşen "bellek hücreleri" (memory cells) kümesine eklenen ARB'ın antikorudur.

Özellik vektörü (feature vector): Değerler dizisi ile temsil edilen bir veridir. Dizilimin her bir pozisyonundaki eleman, farklı bir özelliği temsil eder.

Hiper mutasyon oranı (hiper-mutation rate): Bir bellek hücresinin hücre popülasyonuna katabileceği, mutasyona uğramış klonların sayısını belirleyen bir tamsayı değeridir. Mevcut gerçeklemede; seçilen bellek hücresi, en azından (hipermutasyon oranı * klonlama oranı * uyarılma değeri) adet mutasyona uğramış klonu hücre popülasyonuna ekler.

K-en yakın komşu (K-nearest neighbour-KNN): AIRS algoritması içerisinde; K-en yakın veri noktası, verilen bir test antijenine karşı , en fazla uyarılan k adet bellek hücresidir.

K değeri (k value): Bir test verisinin sınıfını belirlemede kullanılacak olan bellek hücre sayısını gösteren bir parametredir.

Bellek hücresi (memory cell - mc): Sunulan antijene karşı en fazla uyarılan ARB'ın antikorudur. Bir bellek hücresinin yerini başka bir bellek hücresi alabilir (replace). Bu durum sadece; aday bellek hücresinin antijenle uyarılma değeri, en fazla uyarılmış yerleşik bellek hücrenin (established memory cell) uyarılma değerinden yüksekse ve yerleşik bellek hücresi ile aday bellek hücresi arasındaki afinite değeri, kesme değerinden (cut-off value) daha küçükse gerçekleşir. Kesme değeri; (Afinite Eşiği*ATS) ile hesaplanmaktadır.

Mutasyon oranı (mutation rate): 0–1 arasında olan ve bir ARB'ın bir özelliğinin (feature) mutasyona uğrama olasılığını belirten parametredir.

Çıktı (output): Bir hücre ile ilişkili sınıf kategorisidir. Özellik vektörünün sınıfı ile aynı anlama gelir.

Kaynaklar (resources): Sistemde izin verilen ARB'ların sayısını sınırlayan bir parametredir. Her ARB'a; uyarılma değeri ve klonlama oranına bağlı olarak belirli sayıda kaynak atanır. Sistemin tüm kaynaklarının sayısı belirli bir değerle sınırlandırılır. Sistemde izin verilenden daha fazla sayıda kaynak tüketilirse, en düşük uyarılmaya sahip ARB'lardan kaynaklar çıkarılmaya başlanır ve sistemde izin verilen sayıya ulaşana kadar bu işlem gerçekleştirilir. Eğer bir ARB'ın kaynaklarının hepsi silinirse; ARB'da popülasyondan silinir.

Çekirdek hücre (seed cell): Eğitim başlangıcında, ARB popülasyonlarını ve bellek hücresini

ilklendirmek için kullanılan, eğitim kümesinden seçilen bir antikor.

Uyarılma fonksiyonu (stimulation function): Bir ARB'ın bir antijene verdiği yanıtı ölçmek için kullanılan bir fonksiyondur. Mevcut gerçeklemede, bu fonksiyon 0–1 arasında değerler döndürmektedir. Bu fonksiyon, antijen ve ARB'ın özellik vektörleri arasındaki Öklid uzaklığı ile ters uzantılıdır.

Uyarılma değeri (stimulation value): Uyarılma fonksiyonundan dönen değerdir.

Uyarılma eşiği (stimulation threshold): Bir antijen için, eğitimin durma kriteri olarak kullanılan 0–1 arasındaki bir parametredir. Mevcut gerçeklemede, sadece her bir sınıfa ilişkin ARB'ların ortalama uyarılma değeri, "uyarılma eşiği" 'nin üzerindeyse eğitim durur.

Test kümesi (test set): Eğitilmiş AIRS sınıflayıcısının performansını değerlendirmek için kullanılan antijenler kümesidir.

Eğitim kümesi (training test): AIRS'i eğitmek için kullanılan antijenlerin koleksiyonudur.

3.4.4 AIRS Algoritmasına İlişkin Notasyonlar

- MC, bellek hücreleri kümesini göstersin. mc ise bu kümenin bir üyesini temsil etsin.
- ag.c, bir antijenin sınıfını temsil etsin. ag.c $\in$ C = {1, 2, …nc} ve nc, veri kümesindeki sınıfların sayısıdır. Tez kapsamındaki yazılım kusur kestirim çalışmaları için ag.c $\in$ C = {1, 2} olarak tanımlanmaktadır.
- mc.c, bir bellek hücresinin sınıfını temsil etsin. mc.c $\in$ C = {1, 2, …nc} ve nc, veri kümesindeki sınıfların sayısıdır. Tez kapsamındaki yazılım kusur kestirim çalışmaları için ag.c $\in$ C = {1, 2} olarak tanımlanmaktadır.
- $MC_c \subseteq MC = \{MC1 \cup MC2 \cup \ldots MC_{nc}\}$ olarak tanımlansın ve mc.c$\equiv$c ise mc $\in$ MC_c olarak gösterilebilir.
- ag.f ve mc.f; antijen ve bellek hücresinin özellik vektörünü temsil etsin. $ag.f_i$, ag.f içerisindeki i. özelliğin değerini; $mc.f_i$, mc.f içerisindeki i. özelliğin değerini göstersin.
- ARB'lar kümesi, AB ile temsil edilsin. MU, ARB'ların mutasyona uğramış klonlarını temsil etsin. Ayrıca, ARB içerisindeki tek bir ARB ab ile gösterilsin ve bu durumda ab $\in$ AB şeklinde gösterilebilir.
- ab.c, bir ARB'ın sınıfını göstersin ve ab.c $\in$ C = {1, 2, …nc} şeklinde ifade edilsin.

Tez kapsamındaki yazılım kusur kestirim çalışmaları için ab.c $\in$ C = {1, 2} olarak tanımlanmaktadır.

- $AB_c \subseteq AB = \{AB_1 \cup AB_2 \cup \ldots AB_{nc}\}$ olarak tanımlansın ve ab.c $\equiv$ c ise ab $\in$ AB_c olarak gösterilebilir.
- ab.stim, ab'nin (bir adet ARB) uyarılma seviyesini temsil etsin.
- ab.resources, ARB (ab) tarafından tutulan kaynakların sayısını göstersin.
- TotalNumResources, sistemde izin verilen kaynakların toplam sayısını göstersin.

3.4.5 AIRS İlk Adımı: İlklendirme

- Veri kümesindeki tüm veriler normalize edilerek, her sayısal özelliğin (feature) [0, 1] arasında olması sağlanır. Ayrıca; 2 verinin özellik vektörü arasındaki Öklid uzaklığının [0, 1] aralığında olması için, bu normalize değerler $\frac{1}{\sqrt{n}}$ ile çarpılarak elde edilen değerler normalize değer olarak kullanılır. n, uzaklık değerlendirilmesinde kullanılan özelliklerin sayısıdır. Örneğin; 21 metriğin kullanıldığı kusur kestirim modelinde n=21 olarak kullanılmaktadır. Mevcut gerçeklemede; afinite ve uyarılma için Öklid uzaklığı seçilmiş olmasına rağmen, farklı fonksiyonlar da kullanılabilir.
- Afinite eşiği (affinity threshold) hesaplanır. Bu değer, tüm veri kümesindeki özellik vektörlerinin aralarındaki Öklid uzaklığının ortalamasıdır. Mevcut çalışmada, bu hesaplama için sabit bir sayı olarak 50 seçilmiş ve veri kümesindeki 50 verinin aralıklarındaki uzaklığın ortalamasını alacak bir gerçekleme yapılmıştır. Eşitlik 3.10'a göre bu hesaplama yapılmaktadır. agi ve agj, antijen kümesindeki i. ve j. antijenlerdir. Affinity(x, y), iki antijenin özellik vektörü arasındaki Öklid uzaklığını döndürür. n ise antijen sayısını gösterir.

$$AT = \frac{\sum_{i=1}^{n} \sum_{j=i+1}^{n} affinity(agi, agj)}{\frac{n(n-1)}{2}} \tag{3.10}$$

- Son adım, bellek hücrelerinin ve ARB popülasyonunun ilk değerlerle başlatılmasıdır. Bu ise eğitim kümesinden sıfır ya da daha fazla sayıda antijenin rastgele seçilerek bellek hücre kümesine ve ARB'lar kümesine eklenmesi ile yapılır.

3.4.6 AIRS İkinci Adımı: En İyi Uyuşan Hücrenin Belirlenmesi ve ARB Oluşumu

Veri kümesinden seçilen ve bellek hücre havuzuna sunulan bir antijen (*ag*) için, aşağıdaki özelliğe sahip olan bellek hücresi (mc_{match}), bellek hücre havuzundan bulunur.

$$mc_{match} = \text{argmax}_{mc \in MC_{ag.c}} \text{stimulation}(ag, mc) \quad (3.11)$$

$$\text{stimulation}(x, y) = 1 - \text{Euclidean\_distance}(x.f, y.f) \quad (3.12)$$

Eğer $MC_{ag.c} \equiv \emptyset$ ise, $mc_{match} \leftarrow ag$ ve $MC_{ag.c} \leftarrow MC_{ag.c} \cup ag$ olarak ifade edilmektedir. Bu gösterim; antijenle aynı sınıfta olan bellek hücreleri kümesinin boş olması durumunda, antijenin bellek hücreleri kümesine eklendiğini ve bu yeni eklenen bellek hücresinin mc_{match} olarak seçildiğini ortaya koymaktadır.

Mevcut gerçeklemede, uyarılma değeri Öklid uzaklığı ile belirlenirken her zaman bu şekilde yapılması gerekli değildir.

mc_{match} , belirlendikten sonra bu bellek hücresi mevcut ARB kümesine eklenecek yeni ARB'lar oluşturmak için kullanılır (Mevcut ARB'lar, önceki antijenlerin etkisiyle sistemde kalmış olan ARB'lar olabilir). Bu oluşturma ise Şekil 3.26'daki fonksiyonla gerçekleştirilir.

makeARB(x) fonksiyonu bir ARB döndürmektedir. *mutate(x, b)* ise Şekil 3.27'de gösterilmiştir. *drandom* [0, 1] arasında rastgele bir değer döndürürken (*lrandom( ) mod nc*) {0, nc} arasında rastgele bir değer döndürür.

```
MU ← Ø
MU ← MU ∪ makeARB (mcmatch)
stim ← stimulation (ag, mcmatch )
NumClones ← hyper_clonal_rate * clonal_rate * stim
while ( |MU| < NumClones )
do
      mut ← false
      mcclone ← mcmatch
      mcclone ← mutate (mcclone, mut)
      if(mut ≡ true )
            MU ← MU ∪ makeARB(mcclone)
done
AB ← AB ∪ MU
```

Şekil 3.26 ARB oluşumu için hipermutasyon (Watkins, 2001)

```
mutate(x, b)
{
      foreach(x.fi in x.f)
      do
                change ← drandom( )
                change_to ← drandom( )
                if(change < mutation_rate)
                      x.fi ← change_to * normalization_value
                      b ← true
    done
    if(b ≡ true)
              change ← drandom ( )
              change_to ← (lrandom( ) mod nc)
              if(change < mutation_rate)
                  x.c ← change_to
    return x;
}
```

Şekil 3.27 Mutasyon rutini (Watkins, 2001)

3.4.7 AIRS Üçüncü Adımı: Sınırlı Kaynaklar için Yarışma

Bu aşamada; mc_{match} isimli hücreyi içeren ARB'lardan oluşan bir küme (AB), mc_{match} hücresinden oluşan mutant hücreler ve (belki) önceki antijenlerden kalan ARB'lar bulunmaktadır. Algoritmanın bu bölümünde amaç; ilgili antijeni doğru şekilde sınıflayacak olan en başarılı aday bellek hücresinin geliştirilmesidir. Bu durum 3 mekanizma ile gerçekleştirilir:

- İlki, sistem kaynakları için olan yarıştır: ARB'lara kaynak atamaları, normalize edilmiş uyarılma (stimulation) değerlerine göre yapılır.
- İkinci mekanizma, çeşitlilik için mutasyon kullanılmasıdır.
- Üçüncü mekanizma; "ortalama uyarılma eşiğinin" ilgili antijenle başlatılmış olan (ag) eğitim adımının durdurulma kriterini belirlemek üzere kullanılmasıdır.

Genetik algoritmalardaki prensiplere benzer şekilde; AIRS algoritması, ARB popülasyonu içindeki bireylerin hayatta kalmasını ya da yok edilmesini sağlayacak bir mekanizma içermektedir. AB popülasyonu içindeki her ARB'a antijen (ag) sunulur ve ARB'ın uyarılma seviyesi belirlenir. Bu uyarım değeri, ARB popülasyonu üzerinde normalize edilir. Bu normalize değere göre, her $ab \in AB$'ye sonlu sayıda kaynak atanır. Bu kaynak ataması

sonucunda, sistemde izin verilen maksimum kaynak sayısını geçecek şekilde kaynak dağılımı olursa, en az uyarılan ARB'lardan başlayarak kaynaklar silinir. Kaynağı olmayan ARB'lar da ARB popülasyonundan çıkartılır. Bu süreç Şekil 3.28'de gösterilmektedir. Algoritmanın bu noktasında; AB içerisinde, sınırlı kaynak üzerine olan yarışmada hayatta kalabilmiş ARB'lar yer almaktadır. Daha sonra algoritma, AB içindeki ARB'ların bu antijen tarafından yeterince uyarılıp uyarılmadığını ve bu uyarılma derecesine bağlı olarak bu veri üzerinde eğitimin tamamlanıp tamamlanmayacağını belirler. Bu işlem; *nc* uzunluğunda s vektörünün tanımlanması ile yapılır. AB'nın her sınıfı için, ortalama uyarılma değeri bu vektörde tutulur.

$$s_i \leftarrow \frac{\sum_{j=1}^{|AB_i|} ab_j.stim}{|AB_i|} \quad , ab_j \in AB_i \tag{3.13}$$

s vektörü içerisindeki tüm elemanlar için $s_i \geq$ *stimulation_threshold* ise durma kriteri sağlanır. Durma kriterinin sağlanıp sağlanmadığına bakılmaksızın, algoritma AB içindeki her ARB'ın mutasyona uğramış çocuklar (offspring) üretmesi fırsatını sunar.

AIRS'in 2. adımında açıklanan; mutasyona uğramış çocukların ARB içerisine eklenmesine benzer şekilde bu adımda da işlemler gerçekleştirilir ancak bazı farklar vardır. Bu değiştirilmiş mutasyon oluşturma rutini Şekil 3.28'de verilmektedir.

Hayatta kalan her ARB'a mutasyona uğramış çocuklar üretmesine izin verildikten sonra, durma kriteri incelenir. Durma kriteri sağlanınca, bu antijen üzerinde eğitim durur. Durma kriteri sağlanmadıysa, tüm bu süreç, aşağıda verilen sözde kod, durma kriteri sağlanıncaya kadar devam eder.

Bu tekrarlı durumun tek istisnası, ilk geçiş dışında, uyarılma ve kaynak atama fazından sonra durma kriteri sağlanırsa, mutasyona uğramış çocukların üretimi yapılmaz. Durma kriteri sağlanınca, aday bellek hücresi seçilir. $mc_{candidate}$, antijen (ag) ile aynı sınıfta olan en fazla uyarılmış ARB'ın özellik vektörüdür. Aşağıdaki sözde (pseudo) kod, AIRS'in 3. adımındaki uyarılma, kaynak ayırma ve ARB silinmesi işlemlerini göstermektedir.

$\min Stim \leftarrow 2.0$
$\max Stim \leftarrow 0.0$
foreach($ab \in AB$)
do
$stim \leftarrow stimulation(ag, ab)$
$if\ (stim < \min Stim)\ \min Stim \leftarrow stim$
$if\ (stim > \max Stim)\ \max Stim \leftarrow stim$
$ab.stim \leftarrow stim$
done
foreach($ab \in AB$)
do
$if\ (ab.c \equiv ag.c)$
$ab.stim \leftarrow \frac{ab.stim - \min Stim}{\max Stim - \min Stim}$
else
$ab.stim \leftarrow 1 - \frac{ab.stim - \min Stim}{\max Stim - \min Stim}$
$ab.resources \leftarrow ab.stim * clonal\_rate$
done
$i \leftarrow 1$
while(i ≤ nc)
do
$resAlloc \leftarrow \sum_{j=1}^{|AB_i|} ab_j.resources, ab_j \in AB_i$
$if\ (i \equiv ag.c)$
$\text{NumResAllowed} \leftarrow \frac{TotalNum\ \text{Re}sources}{2}$
else
$\text{NumResAllowed} \leftarrow \frac{TotalNum\ \text{Re}sources}{2 * (nc - 1)}$
while (resAlloc > NumResAllowed)
do
$Num\ \text{Re}s\ \text{Re}move \leftarrow resAlloc - Num\ \text{Re}sAllowed$
$ab_{remove} \leftarrow \arg\min_{ab \in AB_i} (ab.stim)$
if ($ab_{remove}.resources \leq Num\ \text{Re}s\ \text{Re}smove$)
$AB_i \leftarrow AB_i - ab_{remove}$
$\text{resAlloc} \leftarrow resAlloc - ab_{remove}.resources$
else
$ab_{remove}.resources \leftarrow ab_{remove}.resources - Num\ \text{Re}s\ \text{Re}move$
$\text{resAlloc} \leftarrow resAlloc - Num\ \text{Re}s\ \text{Re}move$
done
$i \leftarrow i + 1$
done

Şekil 3.28 Uyarılma, kaynak ayırma ve ARB silinmesi (Watkins, 2001)

```
MU ← ∅
foreach( ab ∈ AB )
do
    rd ← drandom()
    if (ab.stim > rd)
        NumClones ← ab.stim * clonal _ rate
        i ← 1
        while( i ≤ NumClones )
        do
            mut ← false
            ab_clone ← ab
            ab_clone ← mutate(ab_clone , mut)
            if (mut ≡ true)
                MU ← MU ∪ ab_clone
            i ← i + 1
        done
done

AB ← AB ∪ MU
```

Şekil 3.29 Hayatta kalan ARB'ın mutasyonu (Watkins, 2001)

3.4.8 AIRS Dördüncü Adımı: Bellek Hücresi Havuzuna Yeni Hücre Eklenmesi

Eğitim aşamasında son adım, yeni gelişmiş olan aday bellek hücresinin (mccandidate) var olan bellek hücrelerine (MC) eklenme durumudur. Bu aşamada; ilklendirme adımında hesaplanan "afinite eşik" değeri, mccandidate hücresinin mcmatch ile yer değiştirip değiştirmemesinin kararının verilmesinde kullanılır.

Aday bellek hücrenin antijen ile uyarılma değeri, mcmatch'in antijen ile uyarılmasından daha güçlüyse, aday bellek hücre bellek hücreleri havuzuna eklenebilir ancak bir test daha yapılmalıdır. mccandidate ve mcmatch arasındaki afinite; (AT*ATC) çarpımından daha küçükse, mccandidate mcmatch ile değiştirilir. Şekil 3.30'da AIRS'in 4. adımı algoritmik olarak gösterilmektedir.

Aday bellek hücresinin, yerleşik bellek hücrelerine eklenmesinin değerlendirmesi bitince, bu antijen üzerinde eğitim tamamlanır. Daha sonra, veri kümesindeki diğer antijen seçilir ve eğitim süreci 2. adımdan devam eder. Bu süreç, tüm antijenler sisteme sunulana kadar devam eder.

```
CandStim  ← stimulation (ag, mc_candidate)
MatchStim ← stimulation (ag, mc_match )
CellAff   ← affinity(mc_candidate, mc_match)
if ( CandStim > MatchStim )
        if(CellAff > AT*ATS)
            MC ← MC - mc_match
        MC ← MC ∪ mc_candidate
```

Şekil 3.30 Bellek hücresinin tanıtılması

3.4.9 AIRS Beşinci Adımı: Sınıflandırma

Eğitim süreci bitince, evrimleşmiş bellek hücreleri sınıflandırmada kullanılabilir. Sınıflandırma, k en yakın komşu yaklaşımı ile yapılır. Her bellek hücresine, uyarılma için bir veri sunulur ve k tane en fazla uyarılan bellek hücresinin çıktı (output) değerinin çoğunluğuna göre verinin sınıf etiketi belirlenmiş olur.

3.4.10 Örnek: AIRS Algoritmasının Kusur Kestirim Veri Kümesinde Açıklanması

Bu bölüm, örnek bir kusur kestirim kümesinde, AIRS algoritmasının çalışma şeklini adım adım ve sayısal değerlerle ortaya koymaktadır. Bu örneğin hazırlanması için, ilgili algoritmanın gerçeklenmiş şeklini içeren ve bir sourceforge projesi olan wekaclassalgos (http://wekaclassalgos.sourceforge.net) projesinin kaynak kodları kullanılmıştır. Sayısal değerleri doğru ve daha kolay elde edebilmek için kod adım adım çalıştırılmış, o anda bellek hücre havuzu ve ARB havuzundaki veriler incelenmiştir. Bu sayede, AIRS algoritmasının ilk aşaması olan ilklendirme adımından başlanarak, son adımı olan sınıflandırma adımına kadar gerçekleşen tüm algoritmik işlemler, sayısal bir örnek üzerinden ortaya konulmuştur.

AIRS algoritmasının ilk gerçeklenmiş halini AIRS sürüm1 (AIRSv1) ile ifade edersek, bu örneğin AIRS sürüm1'i açıkladığını söyleyebiliriz. Sürüm 2'de ise temel adımlar korunmuş, bazı noktalarda küçük değişiklikler yapılmıştır. Bölüm 3.3.1.2 bu farklılıkları açıklamaktadır. AIRS sürüm1 algoritmik açıdan daha karmaşık olmasına rağmen, performans açısından AIRS sürüm2'den bir ölçüde daha başarılıdır. Bu nedenle, algoritmanın ilk çıkış noktası olan AIRS sürüm1, bu örneği açıklamak amacıyla tercih edilmiştir.

Yazılım kusur kestirim veri kümeleri üzerinde, yapay bağışıklık sistem paradigmasının ve algoritmalarının nasıl uygulandığını, AIRS algoritması ile açıklamamızın nedeni, bu algoritmanın diğer yapay bağışıklık sistem tabanlı algoritmalardaki bazı mekanizmaları da

içermesi (klonal seçim ve sınırlı kaynak), çalışmalarımızda yüksek performans sunmuş olması ve birçok karmaşık problemde halen kullanılıyor olmasıdır.

Şekil 3.31'de her bir modüle ilişkin yazılım metriklerini ve kusur bilgisini içeren örnek bir veri kümesi verilmektedir. Bu veri kümesi, bir önceki yazılım sürümüne ilişkin oluşturularak, sonraki yazılım sürümünde oluşacak yeni modüllerin veya metrikleri değişecek yazılım modüllerinin, kusur eğilimliliğini belirlemek amacıyla kullanılacaktır.

Makine öğrenmesi alanında, geliştirilen modellerin doğruluğu aynı veri kümesi üzerinde doğrulanmak durumundaysa, N-katlı çapraz geçerleme (N-fold cross-validation) tekniği kullanılmaktadır. Bu teknik, veri kümesini rastgele N adet eşit sayıda örnek içeren parçaya ayırır. Her bir parça için, (N-1) bölümle eğitim gerçekleştirilip, kalan bölümle test yapılır. Bazı araştırmacılar, N-katlı çapraz geçerlemenin M kez tekrarlanmasını önermiştir. Bu sayede, sıralamadan kaynaklı varyanslar oluşmayacak ve istatistiksel olarak daha anlamlı değerler elde edilecektir. Bu örnekte az sayıda veri olduğu için, 2 katlı çapraz geçerleme kullanıldığı düşünülerek (N=2), bu işlem 1 kez (M=1) tekrarlanmıştır.

```
@RELATION KusurVeriKumesi
@ATTRIBUTE CevrimselKarmasiklik  REAL
@ATTRIBUTE KodSatirSayisi        REAL
@ATTRIBUTE EşsizOperandSayisi    REAL
@ATTRIBUTE EşsizOperatorSayisi   REAL
@ATTRIBUTE sinif  {evet, hayir}
@DATA
40.0,220.0,140.0,240.0,evet
2.0,12.0,8.0,8.0,hayir
65.0,42.0,430.0,260.0,evet
3.0,10.0,13.0,8.0,hayir
5.0,14.0,7.0,6.0,hayir
100.0,240.0,160.0,234.0,evet
65.0,45.0,430.0,260.0,evet
8.0,9.0,13.0,8.0,hayir
9.0,24.0,7.0,6.0,hayir
150.0,640.0,160.0,234.0,evet
```

Şekil 3.31 Sayısal örnekte kullanılan arff uzantılı kusur kestirim veri kümesi

WEKA kaynak kodları içerisindeki çapraz geçerleme amaçlı fonksiyon çağrıldığında, Şekil 3.32'deki modüller eğitim kümesine seçilmiş, geri kalan modüller ise test amaçlı olarak kullanılmıştır. 2 katlı çapraz geçerlemede daha sonra, test verisi olarak az önce ifade edilen veriler eğitim amaçlı kullanılır ve az önce eğitim amaçlı seçilmiş veriler test verisi olarak değerlendirilir. Sunulan örneğin amacı, AIRS algoritmasını açıklamak olduğundan, bu bölümde eğitim kümesi olarak Şekil 3.32'deki veriler kullanılıp, geri kalanların test edildiği varsayılacaktır. Sunulan örneğin tekrarlı bilgiler içermemesi için, 2 katlı çapraz geçerlemenin ilk bölümü ortaya konulmuş olacaktır.

İlklendirme aşamasında gerçekleştirilen veri kümesinin normalizasyonu, WEKA kütüphanesi içindeki fonksiyonlar kullanılarak gerçekleştirilmiştir.

```
5.0,14.0,7.0,6.0,hayir
8.0,9.0,13.0,8.0,hayir
40.0,220.0,140.0,240.0,evet
150.0,640.0,160.0,234.0,evet
3.0,10.0,13.0,8.0,hayir
```

Şekil 3.32 Çapraz geçerlemede seçilen eğitim verileri

WEKA içerisinde normalizasyon eşitlik 3.14 ve eşitlik 3.15 kullanılarak, sütunlar seviyesinde yapılmaktadır.

$$\text{Aralık} = \text{SütunMaksimumDeğeri} - \text{SütunMinimumDeğeri} \tag{3.14}$$

$$\text{NormalizeDeğer} = (\text{NormalizeEdilecekDeğer} - \text{SütunMinimumDeğeri}) / \text{Aralık} \tag{3.15}$$

Örneğin; Şekil 3.32'deki verilerin, ilk satırının ilk elemanı olan 5.0 değeri normalize edilecek olsun. Bu durumda; Aralık = 150 – 3 = 147 olarak belirlenir ve (5 -3) / 147 = 0.013605 hesabı ile 5.0 değerinin o sütundaki normalize karşılığı olarak 0.013605 değeri saptanmış olur. Benzer işlemler aynı satırın diğer elemanları için de gerçekleştirilirse, ilgili satır tamamen normalize edilmiş olacaktır. Bu sayede o satır için [0.013605, 0.007924, 0, 0, hayir] vektörü algoritmada kullanılabilir. Benzer işlemler Şekil 3.32'deki tüm veriler için gerçekleştirilirse, elde edilen normalize küme Şekil 3.33'de verilen hale dönüşecektir.

```
0.013605,0.007924,0,0,hayir
0.034014,0,0.039216,0.008547,hayir
0.251701,0.33439,0.869281,1,evet
1,1,1,0.974359,evet
0,0.001585,0.039216,0.008547,hayir
```

Şekil 3.33 Normalize edilmiş eğitim kümesi

Takip eden bölümlerde, AIRS algoritmasının, Şekil 3.33'de verilen veri kümesini kullanarak eğitim sürecini ve sınıflandırmayı nasıl gerçekleştirdiği ortaya konulmaktadır. Örneği karmaşıklaştırmamak ve sayfa sayısını arttırmamak için, bazı noktalarda oluşan hücrelerin bir kısmı açıklanmakta, diğerleri ise benzer mantıkla oluşturulduğundan gösterilmemektedir. Bu nedenle, belirli soyutlamalar katılarak ortaya konulan bir örnek olduğu dikkate alınmalıdır.

3.4.10.1 Örnek Adım 1: İlklendirme

Bu adım; bölüm 3.4.5'de teorik olarak açıklanmıştı. İlk olarak veri kümesinin normalize edilme işlemi gerçekleştirilmektedir ve normalize küme Şekil 3.33'de verilmektedir. Ayrıca, afinite eşik değeri (affinity threshold-AT), normalize veri kümesinin tüm verileri arasındaki Öklid uzaklığının ortalaması hesaplanarak belirlenir. Bu veri kümesi için bu değer; AT=0.497 olarak hesaplanmıştır. Bu işlem için kullanılan veri noktası (data point) veya örnek sayısı, program çalıştırılmadan önce kullanıcıdan alınan *numInstancesAffinityThreshold* parametresine göre belirlenir. Kullanıcı bu parametreyi varsayılan (default) değeri olan -1 seçerse, tüm veriler kullanılarak AT hesaplanır. Program çalıştırılmadan önce kullanıcı tarafından bu parametre, -1 olarak belirlenmiş ve dolayısı ile tüm veriler işlemde kullanılmıştır.

Daha sonra; bellek hücre havuzu (MC) ve ARB havuzu (AB), veri kümesinden rastgele seçilen ilk değerlerle başlatılır. Bu havuzlarda, ilklendirme aşamasında bulunacak olan veri noktası sayıları da, kullanıcı tarafından seçilebilmektedir. Bu parametreler; *arbInitialPoolSize* ve *memInitialPoolSize* parametreleridir. Program çalıştırılmadan önce, 1 değeri (varsayılan değer), kullanıcı tarafından seçilmiştir. Bu işlem gerçekleştikten sonra; bellek hücre havuzunda, eğitim kümesinin 4. satırı olan [`1,1,1,0.974359,0`] vektörünün yer aldığı tespit edilmiştir. ARB havuzunda ise eğitim kümesinin ilk satırı olan [`0.013605,0.007924,0,0,1`] vektörünün bulunduğu gözlemlenmiştir. İlklendirme aşaması sonunda ARB havuzunun ve bellek hücre havuzunun içeriği, Şekil 3.34 ve Şekil 3.35'de gösterilmektedir. Bu şekillerde;

{evet} sınıf etiketi için 0, {hayir} sınıfı etiketi için 1 değerlerinin kullanıldığı görülmektedir. Bu değerler, dikkati çekmek için, diğer verilerden daha büyük yazı tipleriyle gösterilmiştir.

```
0.013605,0.007924,0,0,1
```

Şekil 3.34 ARB havuzunun ilklendirme sonrası durumu

```
1,1,1,0.974359,0
```

Şekil 3.35 Bellek hücre havuzunun ilklendirme sonrası durumu

Bu havuzların ilk atamaları yapıldıktan sonra, ilk aşama olan ilklendirme aşaması tamamlanmış olmaktadır.

3.4.10.2 Örnek Adım 2: En İyi Uyuşan Hücrenin Belirlenmesi ve ARB Oluşumu

Bu adım, Bölüm 3.4.6'da teorik olarak açıklanmıştı. Veri kümesinin ilk satırından başlanarak, her veri noktası antijen olarak değerlendirilip, algoritmanın ikinci, üçüncü ve dördüncü adımları gerçekleştirilmektedir. Bu nedenle, algoritmanın bu adımı için normalize veri kümesinin ilk satırı olan vektör [0.013605,0.007924,0,0,1] antijen olarak seçilir. Bu vektöre, *current* ismini verelim. Bu antijen ve bellek hücre havuzundaki antikor arasında bir afinite mevcut olacaktır ancak dikkat edilirse bu 2 vektörün sınıf etiketleri farklıdır. Bellek hücre havuzundaki antikorun sınıfı {evet} etiketli iken *current* antijenin sınıf etiketi {hayir} olduğundan, en iyi bellek hücresinin belirlenmesini sağlayan fonksiyon *null* değeri döndürecek ve *bestMatch* değişkeni null değer alacak, *current* antijen bellek havuzuna eklenecektir. Daha farklı bir ifadeyle; bellek hücre havuzu içerisinde, havuza tanıtılan antijenin sınıf etiketi ile aynı bir antikor bulunmadığından, mevcut antijen, bellek hücre havuzuna eklenmiş, sonraki aşamalar için böyle bir durumla karşılaşılmaması sağlanmıştır. Programda kullanılan fonksiyon, antijen ve antikor arasındaki uyarılma değerini hesaplamış olsa bile, sınıf etiketi farklılığından *null* değer döndürmüştür. Bellek hücre havuzunun yeni görüntüsü Şekil 3.36'daki gibi olacaktır.

```
1,1,1,0.974359,0
0.013605,0.007924,0,0,1
```

Şekil 3.36 Bellek hücre havuzunun yeni durumu

current antijen ile bellek hücre havuzundan, en iyi uyuşan bellek hücresi seçilemediği ve

mutant klonlar oluşturulamadığı için, normalize veri kümesinden sıradaki antijen ile aynı aşama (bellek hücre havuzuna antijenin tanıtılması) tekrar gerçekleştirilecektir. Sınıf etiketi farklılığından *null* değer döndüren fonksiyon, bu işlemi sıradaki antijenle yapacak olan fonksiyona dönmektedir. Bu nedenle; yeni *current* antijen vektörü, veri kümesinin ikinci vektörü olan [0.034014,0,0.039216,0.008547,1] vektörü olarak belirlenir. Bu antijen ve bellek hücre havuzundaki antikorlar arasındaki uyarılmalar hesaplanarak, en yüksek uyarılmaya sahip olan antikor belirlenir. Öklid uzaklığı ile afinite hesabı yapılıp, bu değer 1'den çıkarılarak uyarılma değerleri belirlenir. Bellek hücre havuzunun en üst sırasında gösterilen; [1,1,1,0.974359,0] antikoru ile [0.034014,0,0.039216,0.008547,1] antijeni arasındaki uyarılma değeri 0.1295 olarak belirlenir. Bellek hücre havuzunun ikinci sırasındaki antikoruyla [0.013605,0.007924,0,0,1] yine [0.034014,0,0.039216,0.008547,1] antijeni arasındaki uyarılma değeri 0.9795 olarak hesaplanır. İkinci hesaplanan uyarılma değeri, ilk hesaplanan değerden yüksek olduğundan ve bu antikorun sınıf etiketi (hayir) antijenin sınıf etiketi ile aynı olduğundan, bu en yüksek uyarılmaya sahip hücre, en iyi uyuşan hücre olarak seçilir. Bu sayede, mc_{match} = [0.013605,0.007924,0,0,1] şeklinde saptanmış oldu. Bu hücrenin uyarılma değeri 1'e eşit olsaydı, maksimum uyarılma söz konusu olacağından, ARB içine kopyalama amaçlı hiçbir işlem yapılmazdı. O durumda, doğrudan bu hücre, bellek hücre havuzuna eklenerek ilgili antijeni veya daha farklı bir ifadeyle sunulan eğitim verisini en iyi tanıyan hücre olarak belirlenmiş olurdu. Bu noktada hesaplanmış olan uyarılma değeri (0.975) 1'e eşit olmadığından, bu hücreden klonlar oluşturularak, mutasyona tabi tutulup ARB havuzuna bu hücreler eklenir.

Bu noktada, ilk olarak az önce hesaplanan en iyi uyuşan hücre (mc_{match}) ARB havuzuna eklenir. Bellek hücre havuzu ve ARB havuzu, gerçeklemede bağlantılı liste şeklinde tanımlanmıştır. En iyi uyuşan hücreden oluşturulacak klon sayısı, uyarılma değeri ile doğru orantılıdır ve *Math.round(uyarılma_değeri * klonlama_oranı * hipermutasyon_oranı)* şeklinde hesaplanır. Klonlama oranı için varsayılan değer 10, hipermutasyon oranı için varsayılan değer 2'dir ve bu program çalıştırılmadan önce, bu varsayılan değerler kullanılmıştır. Dolayısı ile en iyi uyuşan hücreden maksimum oluşturulabilecek klon sayısı (1 * 10 * 2 = 20) olarak ifade edilebilir. Bu örnekteki sayısal değerleri kullanırsak, *Math.round(0.975 * 10 * 2) = 20* adet klon oluşturulması gerektiğini söyleyebiliriz. Bu klonların mutasyona tabi tutulması için [0, 1] arasında rastgele bir değer oluşturularak, *mutationRate* parametresinden küçük olup olmama durumu kontrol edilmektedir. *mutationRate,* kullanıcı tarafından program başlatılmadan seçilen ve varsayılan değeri 0.1

olan bir değerdir. İlgili klon için; *rand* fonksiyonu ile oluşturulan [0, 1] arasındaki bu rastgele değer, *mutationRate* parametresinden küçükse, ilgili özelliğin (feature) değerine, yeniden *rand* ile oluşturulacak olan ve [0, 1] arasında olan bir değer atanır. *mutationRate* parametresinden küçük değilse, diğer özelliğe (feature) geçilerek o özellik üzerinde mutasyon işlemi gerçekleştirilmeye çalışılır. Dolayısı ile en iyi uyuşan hücrenin her klonu için, bu klonların özelliklerinin hangisinde mutasyon yapılacağı ve mutasyon sonucunda o özelliğe atanacak değerler, o anki rastgele belirlenen değerlere göre yapılmaktadır. Bu gerçeklemede, sınıf etiketi bile mutasyona tabi tutulabilmektedir ve sınıf sayısına bağlı olarak bu özelliğe de tamsayı bir değer rastgele atanabilmektedir. Örneğin; tesadüfi şekilde klonun sınıf etiketinde mutasyon oluşsun ancak bu sınıf etiketi için dönen rastgele değer yine sınıf etiketinin sahip olduğu değer olan 1 olsun. Bu oluşan klon, en iyi uyuşan hücrenin mutasyona uğramış hali olmasına rağmen, vektörlerinin tüm değerleri aynı olacaktır. AIRS gerçeklemesi adım adım çalıştırılırken, bu durumla karşılaşıldığından örnek olarak burada açıklanmıştır. Bu sayede, oluşan klon, en iyi uyuşan hücre ile bire bir aynı olacaktır. Bu mutanta mutant1 ismini verirsek, daha sonra mutant1, ARB havuzuna eklenecektir. ARB havuzunun yeni görüntüsü Şekil 3.37'deki gibi olacaktır.

```
0.013605,0.007924,0,0,1
0.013605,0.007924,0,0,1
0.013605,0.007924,0,0,1
```

Şekil 3.37 ARB havuzunun mutant1 eklendikten sonraki durumu

Şekil 3.37'deki en üstteki satır; ilklendirme aşamasında belirlenen hücreye ilişkindir. Bir alt satırındaki veri noktası, en iyi uyuşan hücreyi göstermektedir. Son sırada gösterilen veri noktası ise mutant1'dir. En iyi uyuşan hücreden, 20 adet klon oluşturulup mutasyona tabi tutulduktan sonra ARB havuzuna tüm mutant hücreler eklenmiş olacaktır. Bu sayede, AIRS algoritmasının ikinci adımı sonunda ARB havuzunda 22 adet ARB yer alacaktır. Şekil 3.37'de tesadüfi şekilde 3 adet aynı bilgi içeren veri noktası yer almaktadır. Farklı değerlerin oluştuğunu göstermek için, mutant2'nin de burada gösterilmesi faydalı olacaktır. Programın adım adım çalıştırılması sırasında; ikinci klonun mutasyon işleminin, ilk özellikte gerçekleştiği ve rastgele değer döndüren fonksiyondan 0.7617 değeri döndüğü gözlemlenmiştir. Bu nedenle, mutant2 vektörü mutant2 = [`0.7617,0.007924,0,0,1`] şeklinde temsil edilir. Bu oluşan mutant2 de, ARB havuzuna mutant1 gibi eklenecektir. Bu mutasyon işlemi, 20 klon için de gerçekleştirilerek AIRS'in bu adımı tamamlanır. Şekil

3.38'de; mutant2 eklendikten sonra ARB havuzunun içeriği gösterilmiştir.

```
0.013605,0.007924,0,0,1
0.013605,0.007924,0,0,1
0.013605,0.007924,0,0,1
0.7617,0.007924,0,0,1
...
```

Şekil 3.38 ARB havuzunun mutant2 eklendikten sonraki durumu

3.4.10.3 Örnek Adım 3: Sınırlı Kaynaklar İçin Yarışma

Bu adım, Bölüm 3.4.7'de teorik olarak açıklanmıştı. Bu bölümde, özetle, aday bellek hücresi (candidateMemoryCell) belirlenmeye çalışılmaktadır. En son kullanılan antijen, ARB havuzuna sunularak tüm ARB'ların uyarılma değerleri ve bu değerlere bağlı olarak kaynak atamaları yapılır.

Sınırlı kaynaklar için kaynak atamaları yapıldıktan sonra, durma kriteri sağlanıyorsa ARB iyileştirme süreci tamamlanır, durma kriteri sağlanmıyorsa, mutasyona uğramış klonlar oluşturularak ARB havuzuna kopyalanır. Ayrıca, durma kriteri ilk anda (first) sağlanmış olsa bile, mutasyona uğramış klonlar oluşturulmaktadır. Durma kriteri, ilgili antijenden kaynaklı oluşan uyarılma değeri ortalamalarına göre belirlenir. Problemde iki sınıf mevcut olduğundan, iki ayrı sınıf için ortalama uyarılma değerleri ayrı ayrı hesaplanır ve uyarılma eşiği (stimulationThreshold) parametresi ile karşılaştırılır. Bu parametre, program çalıştırılmadan önce kullanıcı tarafından tanımlanır ve varsayılan değeri 0.9'dur. Durma kriterinin sağlanması için; iki sınıf için ayrı ayrı hesaplanmış olan ortalama uyarılma değerlerinin, bu uyarılma eşiğinden yüksek olması gerekir. Bu durum gerçekleşirse, ilgili antijen ile yeterince çalışılmış olduğu sonucuna varılır ve AIRS'in dördüncü adımına geçilir.

AIRS temel adımları içerisinde en karmaşığı gibi görünmesine rağmen, sunulacak sayısal örnek bu adımı algılamayı kolaylaştırmaktadır.

ARB havuzunda, AIRS algoritmasının ikinci adımı tamamlandığında, 22 adet ARB oluşmuştu. Bu nedenle, antijen bu ARB'lara sunulduğunda, 22 adet uyarılma değeri hesaplanacaktır. Üzerinde çalışılan en son antijen, [0.034014,0,0.039216,0.008547,1] vektörüydü. Bu antijen, 22 adet ARB'a sunularak, uyarılma değerleri hesaplanır. Şekil 3.38'de gösterilen ARB havuzunun en üst sırasında yer alan vektör

[0.013605,0.007924,0,0,1] ile [0.034014,0,0.039216,0.008547,1] antijeni arasındaki uyarılma değeri hesaplanır. Bu uyarılma değeri, 0.9953 olarak belirlenmiştir. ARB havuzunun ikinci satırındaki ve en iyi uyuşan bellek hücresi de az önceki ARB ile aynı sayısal değerlere sahip olduğundan, uyarılma değeri 0.9953 olacaktır. Benzer şekilde, mutant1'in de uyarılma değeri 0.9953 olacaktır. ARB havuzunda dördüncü satırda yer alan veri [0.7617,0.007924,0,0,1] ile [0.034014,0,0.039216,0.008547,1] antijeni arasında hesaplanan uyarılma değeri, 0.234'dür. Bu uyarılma değerlerine göre, ARB'lara kaynak ataması yapılır. Bu uyarılma değerleri, klonlama oranı (*clonalRate*) olan 10.0 ile çarpılarak atanacak kaynak değerleri belirlenir. Antijen ile ARB'ın sınıf etiketleri farklı ise oluşan uyarılma değeri 1'den çıkarılarak o ARB'ın uyarılma değeri olarak belirlenmiştir. Antijen ile ARB aynı sınıf etiketine sahipse, uyarılma değeri değiştirilmemiştir. Bu sayede, farklı etikete sahip olan ARB'ın, uyarılması da düşük olacağından kaynak atamasını çok düşük değerlerde kalmamış olacaktır. AIRS sürüm1'de bu şekilde bir yaklaşım kullanılmışken, sürüm 2'de bu işlemden vazgeçilmiştir. Sadece antijen ile aynı etikete sahip olan ARB'ların uyarılma değerleri hesaplanmış ve bu sayede sadece bu ARB'ların kaynak atamaları yapılmıştır.

Kaynak atamaları yapıldıktan sonra, her sınıf için toplam kaç kaynak atanmış olduğu ayrı ayrı belirlenir. Program adım adım çalıştırılırken bu amaçla kullanılan; *resources[0]* ve *resources[1]* elemanlarının sırasıyla 0.0093 ve 112.92 değerlerine sahip olduğu görülmüştür. Program çalıştırılmadan önce, kullanıcı tarafından belirlenen *TotalResources* parametresinin varsayılan değeri olan 150, bu örnek için de aynı şekilde seçilmişti. Bu durumda, her sınıfın toplam sayısı 75'den küçük olmalıdır. ARB'lar, kaynak sayılarına göre ancak farklı iki küme oluşturacak şekilde ayrı ayrı sıralanır. Bir kümede, bir sınıf etiketine sahip olan ARB'lar yer alırken diğer kümede diğer sınıf etiketine sahip olan ARB'lar yer alacaktır. Sıfır ile temsil edilen sınıfın toplam kaynak sayısı (0.0093), izin verilen maksimum değer olan 75'den küçük olduğundan, bu tür ARB'lar (evet etiketliler) üzerine kaynak silme işlemine gerek yoktur.

Mevcut antijen {hayir} etiketli olduğundan, sıfır etiketli ARB'lardan en yüksek kaynaklısı *mostStimulatedSameClass* olarak seçilemiyor. Etiketi 1 ({evet}) olanların toplam kullandığı kaynak 112.92 olduğundan, bu değer 75'den büyüktür ve {evet} sınıf etiketini taşıyan en düşük kaynaklı ARB'lardan başlanarak kaynaklar azaltılır.

Kaynak değeri büyük olandan küçük olana doğru ARB'lar sıralanır. Silinmesi gereken kaynak sayısı (112.92 – 75 = 37.12) olduğundan kaynak sayısı bu değerin altında olan ARB'lar, tüm kaynaklarını kaybetmiş olacağından, ARB havuzundan silinirler. İzin verilen maksimum kaynak sayısı olan 75 değerine ulaşılana kadar, en düşük kaynağa sahip olan ARB'lar silinir.

Kaynak değeri olarak; 0.5570 değerine sahip olan ARB'ın en altta kaldığını düşünürsek bu ARB'lardan kaynaklar silinmeye başlanarak eşik seviyesi olan 75'e ulaşılmaya çalışılır. AIRS adım adım çalıştırılırken, bu değerler gözlemlendiği için bu sayısal örnek verilmiştir. Eşik seviyesi sağlandıktan sonra; antijen ile aynı sınıf etiketine sahip olan ARB'lardan en yüksek kaynağa sahip olanı, *mostStimulatedSameClass* olarak belirlenir. Bu örnek çalışmada, bu ARB, *mostStimulatedSameClass* = [0.04104,0.007924,0,0,1] olarak belirlenmiştir. Bu ARB'ın kaynak sayısı 10 olup uyarılma değeri ise 1'dir. Bu ARB, aynı zamanda aday bellek hücresidir ancak durma kriteri henüz kontrol edilmediği için bu konuda kesin bir bilgi vermek için erkendir.

Bu noktada, durma kriterinin sağlanıp sağlanmadığı kontrol edilmelidir. İki sınıfa ait ortalama uyarılma değerleri, uyarılma eşiği olan 0.9 ile kıyaslanacaktır. Bu nedenle; ARB havuzunda kaynak yarışında hayatta kalan ARB'lardan {evet} etiketli ve {hayir} etiketli veriler ayrı ayrı değerlendirilerek, kendi içlerinde ortalama uyarılma değerleri hesaplanır. Bu örnek adım adım çalıştırılırken; *meanStimulation[0]* = 0.0046 olarak belirlenmiş ve 0.9'dan küçük olduğundan durma kriterinin sağlanmadığı anlaşılmıştır. Bu nedenle, ARB havuzundak hayatta kalmış olan ARB'lardan klonlama yapılıp mutasyona tabi tutulacaktır. Bu kontrol ilk kez yapılırken, 0.9'un üzerinde bir ortalama uyarılma değeri elde edilse bile klonlar oluşturulmaktadır. Bir ARB'dan oluşturulabilecek klon sayısı (uyarılma_değeri * klonlama_oranı) ile hesaplanmaktadır. Dolayısı ile bir ARB'dan maksimum oluşturulabilecek klon sayısı 10 (1 * 10 =10) olarak ifade edilebilir. Bellek hücre havuzunda ise hipermutasyon parametresi nedeni ile bu değer 20'ydi. O havuzdaki fazla sayıda klon oluşturulmasının nedeni o havuzda çeşitliliğin daha fazla arttırılmasının istenmesidir.

ARB havuzunda hayatta kalan ARB'lardan birisi [0.013605,0.007924,0,0,1] vektörü olsun. Antijen ise [0.034014,0,0.039216,0.008547,1] vektörü ile gösteriliyordu. Bu iki vektör arasındaki, Öklid uzaklığı 0'a yakın olduğundan, uyarılma değeri 1'e yakın olacak ve üretilecek klon sayısı 10 olacaktır. Her bir klon mutasyona tabi tutulup ARB havuzuna eklenecektir. Bu işlem hayatta olan tüm ARB'lar için gerçekleştirilecektir. Durma kriteri sağlanmamış olduğundan, kaynaklar için yarışma yeniden yapılacaktır. Bu noktada, antijen tekrar ARB havuzundaki ARB'lara sunularak yetersiz kaynaklı olan ARB'lar çıkarılacaktır. İzin verilen maksimum kaynak sayısı bilgisi kullanılarak, çıkarılma durumu yukarıda açıklanan şekilde gerçekleştirilecektir. Şekil 3.19, bu döngüyü oldukça iyi göstermektedir.

Durma kriteri sağlandıktan sonra, en fazla kaynağa sahip ARB'ın [0.04104,0.007924,0.03916,0,1] olduğu gözlemlenmiştir. Bu ARB, aday bellek hücresi

olarak ifade edilir ve AIRS algoritmasının üçüncü aşaması tamamlanmış olur.

İlgili antijen, bellek hücre havuzuna AIRS algoritmasının ikinci adımında sunulduğunda en fazla uyuşan bellek hücresi mc_{match} olarak belirlenmişti ve bu hücre mutasyona uğramış klonlar oluşturarak ARB havuzuna kopyalanmıştı. AIRS algoritmasının bu bölümde açıklanan adımında ise aday bellek hücresi belirlenmiş oldu. Algoritmanın bir sonraki aşamasında, mc_{match} ve aday bellek hücresinden hangisinin antijenle daha fazla uyuşma gösterdiği belirlenecek, bellek hücre havuzuna yeni bir hücrenin (aday bellek hücresi) eklenip eklenmemesine karar verilecek ve gerekirse mc_{match} bellek hücre havuzundan çıkarılacaktır.

3.4.10.4 Örnek Adım 4: Bellek Hücre Havuzuna Yeni Hücre Eklenmesi

Bu adım, Bölüm 3.4.8'de teorik olarak açıklanmıştı. Bu bölümde, önceki bölümlerde adım adım örneklenmiş olan senaryo üzerinden sayısal olarak açıklanacaktır. Önceki adımlarda hesaplanan; mc_{match} , aday bellek hücresi ve antijen değerleri aşağıda verilmektedir.

$mc_{match} = [0.013605, 0.007924, 0, 0, 1]$

$aday\_bellek\_hücresi = [0.04104, 0.007924, 0.03916, 0, 1]$

$antijen = [0.034014, 0, 0.039216, 0.008547, 1]$

Bu *antijen* ve mc_{match}, bu *antijen* ve *aday_bellek_hücresi* arasındaki uyarılma değerleri ayrı ayrı hesaplanır. Bu örnekte; en iyi uyuşan bellek hücresinin uyarılma değeri 0.9795 olarak hesaplanmışken aday bellek hücresinin uyarılma değeri 0.99391 olarak belirlenmiştir. Aday bellek hücresinin uyarılma değeri, en iyi uyuşan bellek hücresinin uyarılma değerinden büyük olduğundan, aday bellek hücresi, bellek havuzuna kopyalanır. Kopyalama işleminin ardından, en iyi uyuşan bellek hücresinin (mc_{match}) bellek havuzundan silinip silinmemesi gerektiğine karar veren kontrol yapılır. Bu aşamada; mc_{match} ve *aday_bellek_hücresi* arasındaki afinite hesaplanır ve bu değer, bellek hücresinin yer değiştirme durumuna karar veren kesme değerinden (cut-off value) küçükse, en iyi uyuşan bellek hücresi silinir. Bu kesme değeri, afinite eşiği (AT) ve afinite eşiği skaleri çarpılarak elde edilir. Afinite eşiği skaleri, program çalıştırılmadan önce programcı tarafından belirlenir ve varsayılan değeri 0.2'dir. Afinite eşiği ise ilklendirme aşamasında, ilgili eğitim kümesi için hesaplanmıştı. Bu örnekte, kesme değeri 0.99391 olarak hesaplanmış olup, mc_{match} ve *aday_bellek_hücresi* arasındaki afinite 0.021138 olarak belirlenmiştir. $0.021138 < 0.99391$ olduğundan, mc_{match} bellek hücre havuzundan silinir. Bellek hücre havuzunun son hali Şekil 3.39'da verilmektedir.

```
1,1,1,0.974359,0
0.04104,0.007924,0.03916,0,1
```

Şekil 3.39 Bellek hücre havuzunun son durumu

Şekil 3.39'da da görüldüğü gibi mc_{match} havuzdan çıkarılmış ve bu havuza, aday bellek hücresi eklenmiştir. Bu adım sonunda, bellek hücre havuzunda iki adet bellek hücresi mevcuttur. Program çalıştırıldığında ARB havuzunda, 521 adet ARB'ın yer aldığı gözlemlenmiştir. Bu aşamada, ilgili antijen için AIRS algoritmasının 2., 3., 4. adımları tamamlanmış olmaktadır. Daha sonra, eğitim veri kümesinde sıradaki veri, *current* antijen olarak seçilecektir. Bu antijen kullanılarak; bu bölüme kadar anlatılan işlemler (AIRS algoritmasının 2., 3. ve 4. adımları) tekrarlanacaktır. Normalize eğitim kümesinden seçilen ve en son kullanılan antijenden sonraki veri; yeni *current* vektörüdür ve *current* = [0.251701,0.33439,0.869281,1,0] şeklinde gösterilir. Bu antijen de, AIRS algoritmasının 2., 3. ve 4. adımlarını geçtikten sonra bellek hücre havuzu Şekil 3.40'daki hale gelecektir.

```
1,1,1,0.974359,0
0.04104,0.007924,0.03916,0,1
0.27105,0.2215,0.84256,0.8412,0
```

Şekil 3.40 Bellek hücre havuzunun yeni antijen uygulandıktan sonraki durumu

Daha sonra, normalize eğitim kümesinde kalan diğer antijenler olan [1,1,1,0.974359,0] antijeni ve [0,0.001585,0.039216,0.008547,1] antijeni, AIRS algoritmasının 2., 3. ve 4. adımlarından geçecektir. Eğitim kümesinde başka antijen kalmayacağı için, çapraz geçerlemede kullanılacak bellek hücre havuzu oluşturulmuş olacaktır. Bu aşamanın sonunda, bellek hücre havuzunun sahip olduğu antikorlar Şekil 3.41'de gösterilmektedir.

```
0.04104,0.007924,0.03916,0,1
0.27105,0.2215,0.84256,0.8412,0
1,1,1,0.974359,0
```

Şekil 3.41 Bellek hücre havuzunun sınıflandırma öncesi durumu

3.4.10.5 Örnek Adım 5: Sınıflandırma

Bu adım, Bölüm 3.4.9'da teorik olarak açıklanmıştı. Eğitim kümesinin 5 adet antijenden

oluşan parçası kullanılarak eğitim tamamlanmış olup, geri kalan 5 adet antijenle test yapılacaktır. Çapraz geçerleme tekniğinde, bu aşamada test amaçlı kullanılan verilerin, bir sonraki aşamada eğitim verisi olarak kullanılması söz konusudur ancak amaç algoritmayı örneklemek olduğundan çapraz geçerlemenin devamı gösterilmeyecektir. Bu adımda, test edilecek veriler, bellek hücre havuzuna sokularak en fazla uyarılan K adet bellek hücresinin, çoğunluk etiket değerine göre, ilgili test verisine etiket ataması yapılmaktadır. Bu örnek için test veri kümesi, Şekil 3.42'de gösterilmektedir.

```
2,12,8,8,?
65,42,430,260,?
100,240,160,234,?
65,45,430,260,?
9,24,7,6,?
```

Şekil 3.42 Test veri kümesi

Bu veri kümesindeki veriler, önceki bölümlerde açıklandığı şekilde normalize edilecektir. Örneğin, test veri kümesinin ilk satırındaki veri normalize edilirse aşağıdaki vektör elde edilir.

```
test1 = [0,0,0.00236,0.00787,?]
```

Bu test verisi ile bellek hücre havuzundaki hücreler arasındaki uyarılma değerleri hesaplanır. Uyarılma değerleri hesaplanırken zaten sınıf etiketi kullanılmadığından, bu işlem kolaylıkla gerçekleştilir. Örnekte, bellek hücre havuzunda sadece 3 tane hücre yer aldığından ve kullanıcı tarafından program çalıştırılmadan önce girilen k-en yakın komşu parametresi (k-nn) 3 olduğundan, uyarılma seviyelerinin büyükten küçüğe doğru sıralanmasına gerek kalmadan, tüm hücrelerin sınıf etiketlerindeki çoğunluk dikkate alınarak sınıflandırma işlemi yapılacaktır. Bu bölümde sunulan örnekte çok az sayıda veri olduğundan, bellek hücre havuzunda sadece 3 adet hücre oluşmuştur ancak büyük veri kümelerinde, veri kümesindeki eleman sayısının yarısı kadar hücrenin oluşabileceğini ve bu noktada uyarılma seviyesinin ön plana çıkacağını söylemek mümkündür. Bu örnek için bellek hücre havuzunda 2 adet {evet} etiketli ve 1 adet {hayir} etiketli hücre mevcut olduğundan, test sonucunda test verilerine {evet} etiketi atanacaktır. Yazılım kusur kestirim veri kümesinin sahip olduğu özellikler ve veri kümesindeki veri sayısı oldukça basitleştirilmiş olarak algoritmada kullanılmıştır. NASA kümeleri üzerinde yapılan analizlerde ise gerçek metrik değerleri ve gerçek kusur bilgileri kullanılarak AIRS algoritması değerlendirilmiştir.

4. KESTİRİM MERKEZLİ YAZILIM GELİŞTİRME SÜRECİ

Bu bölümde, kestirim merkezli yazılım geliştirme süreci ve test senaryosu önceliklendirme amaçlı kusur kestirim modellerinin nasıl kullanılabileceği açıklanmaktadır.

4.1 Giriş

Programlama paradigmalarında yaşanan değişimle birlikte, bu paradigmaların kullanılacağı yöntembilimlerin de (methodology) değişmesi gerekmektedir. Yapısal, nesneye yönelik, bileşen tabanlı, ilgiye yönelik programlama gibi farklı paradigmaların farklı yöntembilimlerle ele alınması, daha etkin yazılım geliştirmeye olanak vermektedir. Örneğin; yapısal programlamada kullanılan analiz ve tasarım yöntemleri veri akışına odaklanmıştır. Yapısal analiz ve tasarımda kullanılan modelleme diyagramlarının (veri akış diyagramı, kontrol akış diyagramı, vb.) nesneye yönelik analiz ve tasarımda uygulanabilmesi ve etkin bir geliştirmenin sağlanabilmesi mümkün değildir. Bu nedenle, nesneye yönelik analiz ve tasarım için Birleştirilmiş Modelleme Dili (Unified Modeling Language) geliştirilmiş ve bu amaçla halen kullanılmaktadır. Çoğu zaman, yöntembilim ve geliştirme süreci kavramları karıştırılmaktadır. Yöntembilim; geliştirme sürecini, süreç içindeki belgelendirmeleri, farklı aşamalarda kullanılabilecek araçları, belirtimleri, paradigmaya özel analiz ve tasarım yöntemlerini kapsamaktadır. Dolayısıyla, her geliştirilen programlama paradigması ile birlikte bir yöntembilimin oluşturulması söz konusudur. Ancak geliştirme sürecinin değiştirilmesine gerek olmayabilir. Mevcut durumda en fazla bilinen yöntembilimler; Yapılandırılmış Yöntembilim (Structured) ve Nesneye Yönelik Yöntembilim olarak sıralanabilir. Özetle; "Yöntembilim; yazılım yaşam çevrimi boyunca kullanılacak süreç, belirtim, belgelendirme gibi yöntemler bütününü içeren bir disiplindir" (Sarıdoğan, 2004). Yazılım geliştirme süreci; "yazılım yaşam çevrimi" ve "yazılım süreci" terimlerinin eş anlamlısıdır. Şimdiye kadar literatürde birçok geliştirme süreci önerilmiş ve farklı dönemlerde bazıları daha fazla ön plana çıkmıştır. Örnek olarak; klasik çevrim, V modeli, prototipleme, spiral model (evrimsel geliştirme, evrimsel prototipleme, artımlı geliştirme, araştırmaya dayalı geliştirme), hızlı uygulama geliştirme, Rational birleştirilmiş süreç, çevik (agile) geliştirme (Uç programlama (extreme programming), Scrum, özellik güdümlü geliştirme) verilebilir. Standish Group'un 2003 yılındaki raporunda, bilişim projelerindeki başarı oranının artmasının sebebi geliştirme süreçlerinde klasik çevrim olan şelale (waterfall) modelinden iteratif geliştirmeye geçiş olarak ifade edilmiştir. Bu örnekte de görüldüğü gibi, geliştirilme süreci bilişim projelerini başarıya taşımada kritik öneme sahiptir.

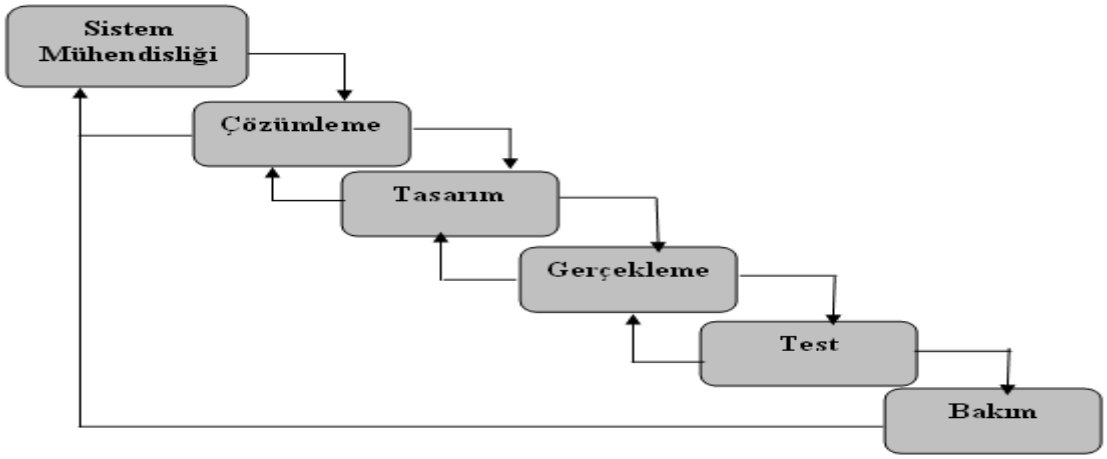
Bu tez çalışmasında, kestirimi merkez alan yeni bir geliştirme süreci önerilmiş ve yazılım mühendisliği içerisindeki kestirim yaklaşımlarına vurgu yapılmıştır.

4.2 Mevcut Geliştirme Süreçleri

Bu bölümde, şimdiye kadar önerilmiş bazı geliştirme süreçleri açıklanacaktır.

4.2.1 Çağlayan Modeli

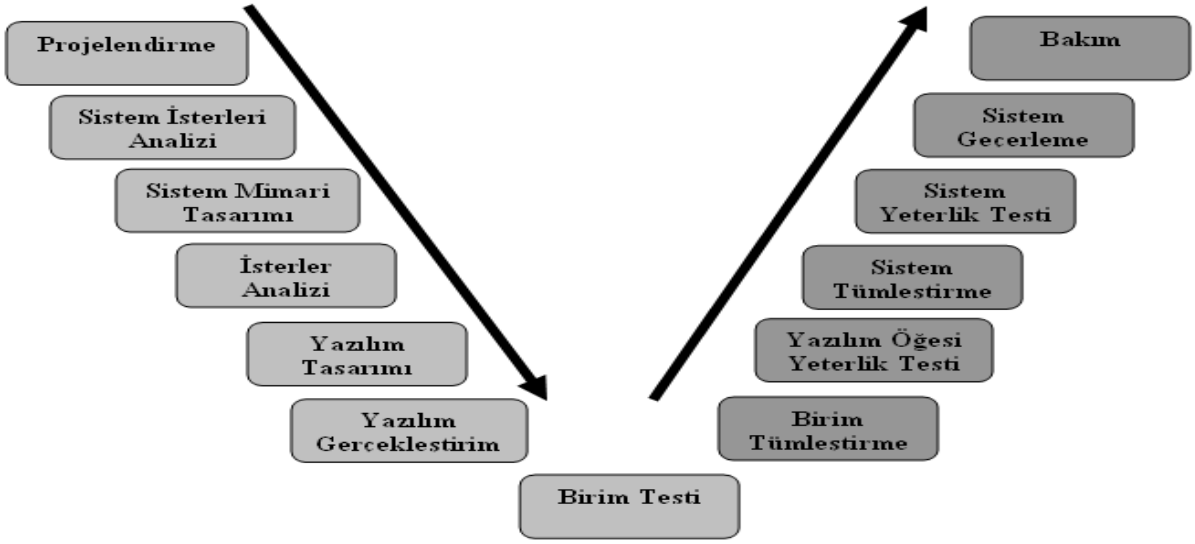
Çağlayan modelinde, geliştirme süreci doğrusaldır ve isterler projenin en başında net olarak tanımlanmak zorundadır. Sistem mühendisliği, çözümleme, tasarım, gerçekleme, test ve bakım aşamaları sırasıyla gerçekleştirilerek geliştirme süreci tamamlanır. Ancak müşteri gereksinimlerinin projenin en başında belirlenmesi mümkün olmadığından, değişen ihtiyaçlara karşılık verme noktasında bu modelin sorunlar içerdiği bilinmektedir. "Özellikle iyi tanımlı, isterleri kesinleşmiş, fazla zaman alması beklenmeyen projeler için uygun bir yöntemdir" (Sarıdoğan, 2004). Şekil 4.1' de bu model gösterilmektedir.



Şekil 4.1 Çağlayan modeli (Sarıdoğan, 2004)

4.2.2 V Modeli

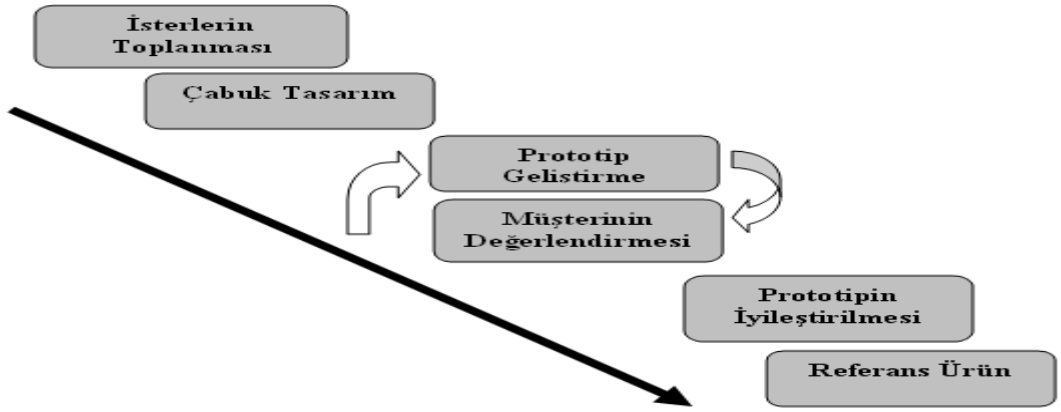
"V modeli, geliştirme sürecine sistem kavramıyla yaklaşarak klasik modeldeki test işlemlerinin ne zaman yapılacağını ön plana çıkarır. Sol kanat üretim etkinlikleri, sağ kanat da test etkinlikleri ile ilgilidir" (Sarıdoğan, 2004). Örneğin; birim testlerinde sorun çıkması durumunda yazılım gerçekleştirimine, yazılım öğesi yeterlik testinde problem yaşanması durumunda yazılım tasarımına dönülmesi gerektiği bu V modeli şeklinden açıkça görülmektedir. V modeli, çağlayan modelinde olduğu gibi isterleri kesinleşmiş ve az belirsizlik olan projeler için uygundur. Şekil 4.2' de V modeli gösterilmektedir.



Şekil 4.2 Yazılım geliştirmede V modeli (Sarıdoğan, 2004)

4.2.3 Prototipleme

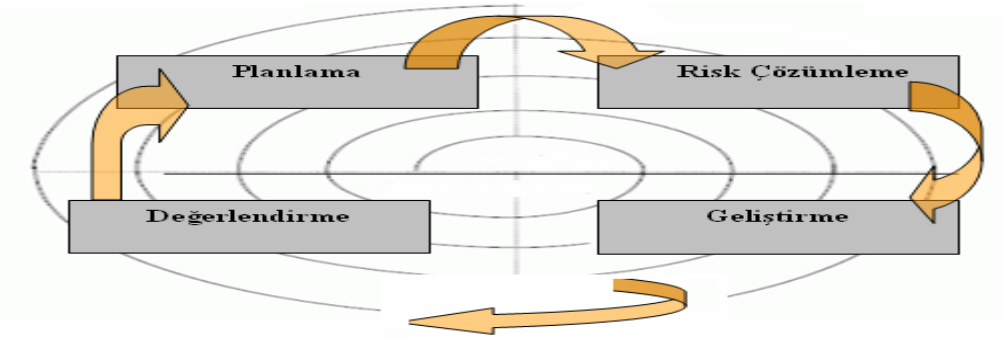
Müşterinin üründen beklentilerinin çok fazla belirgin olmadığı veya geliştiricinin mimari ya da işletim sistemi gibi noktalarda tereddütlerinin bulunduğu projeler için prototipleme iyi bir yaklaşımdır. Müşterinin temel gereksinimleri dikkate alınarak bir tasarım ve ardından prototip oluşturulur ve bu prototip sayesinde kullanıcı grafiksel arayüzleri konusunda fikir birliğine ilk aşamalarda varılabilir. Bazı durumlarda ilk prototip, sistemin devamı için kullanılabilirken bazı durumlarda yok edilebilmektedir. "Prototipleme yöntemi kullanıldığında müşteri yazılım ürününün çok çabuk bittiği kanısına kapılabilir. Oysa, çabuk tasarlanıp geliştirilen bu üründe ne nitelik vardır, ne de uzun dönemde bakım yapılabilmesi mümkündür" (Sarıdoğan, 2004). Şekil 4.3' de prototipleme ile geliştirme süreci gösterilmektedir.



Şekil 4.3 Protipleme ile geliştirme (Sarıdoğan, 2004)

4.2.4 Spiral Model

Spiral model; planlama, risk çözümleme, geliştirme ve değerlendirme aşamalarından oluşmaktadır. İlk isterler planlama aşamasında toplanarak risk çözümlemede bu isterlere göre bir çözümleme gerçekleştirilir. Geliştirme aşamasında prototipleme ile örnek bir ürün ortaya çıkarılarak değerlendirme aşamasında müşteriye sunulmaktadır. Bu şekilde, 4 aşamadan oluşan döngüler şeklinde yazılım geliştirme aktiviteleri yürütülmektedir. Evrimsel geliştirme, evrimsel prototipleme, artımlı geliştirme spiral model örneklerindendir. "Artımlı geliştirme modeli, isterlerinin tamamı belirli olan bir ürünün sürümler halinde geliştirilmesi ilkesine dayanmaktadır" (Sarıdoğan, 2004). Şekil 4.4'de spiral model resmedilmektedir.



Şekil 4.4 Spiral model

4.2.5 Uç Programlama

Uç programlama (extreme programming), son yıllarda popülerlik kazanmış çevik (agile) geliştirme yaklaşımlarındandır. Yöneticiler, geliştiriciler, müşteri bir ekip bilinci ile çalışarak müşterinin memnun kalacağı yazılım ürününü ortaya koymaya çalışır. İletişim, basitlik, geri besleme, cesaret ilkeleri, uç programlama yaklaşımının temelini oluşturmaktadır. Gerektiğinde bir geliştirici, başka bir geliştiricinin kodu üzerinde değişiklik yapabilmeli ve bu bilgiye sahip olmalıdır. İsterlerin çok fazla değiştiği projeler için bu yaklaşım etkin olarak çalışmaktadır. Genellikle 2-10 kişi arasında değişen geliştirici grupları için önerilmiş olup çok büyük projeler için uygulanması problemler doğurmaktadır. "Uç programlama, düzen gerektirmeyen bir yöntem olarak algılanmamalıdır. Belgelendirme, düzenleşim ve nitelik güvence etkinlikleri yine yürütülmelidir" (Sarıdoğan, 2004).

4.2.6 Rational Birleştirilmiş Süreç

Rational firması tarafından geliştirilen bu yöntem (Rational Unified Process-RUP), organizasyon içindeki sorumlulukları detaylı olarak belirleyerek disiplinli bir yazılım

geliştirme süreci sunar. Bu yöntemde aktivite gruplarına, disiplin adı verilmektedir. Gereksinimler, analiz, tasarım, gerçekleme ve test temel disiplinlere örnek olarak verilebilir. RUP içinde yazılım yaşam çevrimi 4 aşamadan oluşmaktadır: Başlangıç (Inception), Düzenleme (Elaboration), Oluşturma (construction) ve Geçiş (transition). Her aşama bir durak noktası ile sonlanmakta ve her aşama sonunda bir değerlendirme yapılarak amaçlara ne oranda ulaşıldığı incelenmektedir. Bu durak noktalarını sırası ile; amaç, mimari, operasyonel yetenek ve ürün teslimi olarak sıralayabiliriz. Başlangıç aşamasında; yaklaşık tahminler yapılarak temel gereksinimler ortaya konulmaya çalışılır. 2. aşamada daha gerçekçi tahminler yapılarak mimari belirlenir ve yüksek riskler için çözümler sunulur. 3. aşamada, ürün hazır hale getirilerek hatalar belirlenir. 4. aşama ise müşteriye ürünün teslim edildiği ve müşteriden gelen geri beslemeye göre iyileştirmelerin yapıldığı aşamadır.

Dikkat edilmesi gereken önemli bir nokta; RUP içindeki her aşamada yazılım mühendisliğinin tüm aktivitelerinin farklı oranlarda uygulanıyor olmasıdır. Klasik yazılım geliştirme süreci olan çağlayan modelinde ise, her aktivite bir önceki aktivitenin bitmesinin ardından gerçekleştirilmektedir. Çağlayan modelinde kodlama aktivitesi, tasarımın ardından gerçekleştirilirken; RUP içinde her aşamada kodlama yapılabilmektedir.

4.3 Kestirim-Merkezli Yazılım Geliştirme

Kestirim, belirli bir olayın gelecekte olacağının bilimsel yöntemlerle iddiasıdır. Yazılım mühendisleri, çeşitli kusur problemleri üzerinde 20 yılı aşkın süredir çalışmaktadır ancak hala yazılım mühendisliği problemleri için gürbüz tahminleme (estimation) ve kestirim (prediction) modelleri oluşturmada ciddi zorluklar mevcuttur. Tahminleme ve kestirim terimleri çoğu zaman birbiri yerine kullanılıyor olsa da, anlamları tamamen farklıdır. Örneğin; yazılımın mevcut güvenilirliği tahminlenebilir fakat bu son amaç değildir. Yazılım ürün yöneticisi yazılımın gelecek davranışının kestirimini yapmalı ve kusur eğilimli modülleri belirlemelidir (Xie, 1991). Tahminleme ile mevcut durumun analizi yapılırken, kestirim ile geleceğe yönelik iddialarda bulunulur.

Yazılım mühendisliği içerisindeki tahminleme ve kestirim problemlerinde hesaplamalı zekâ teknikleri kullanılmaktadır. Özellikle makine öğrenmesi algoritmaları bu problemlerde etkin sonuçlar üretebilmektedir. Yazılım mühendisliğinde makine öğrenmesi algoritmalarının uygulandığı alanlar ortaya konulurken, tahminleme ve kestirim araştırmaları "Kestirim Aktivitesi" altında gösterilmiştir (Zhang ve Tsai, 2003). Makine öğrenmesinin kullanılmış olduğu bazı tahminleme/kestirim problemleri aşağıda verilmektedir:

- *Yazılım kalite kestirimi* (Evett vd., 1998; El-Emam vd., 2001a; Lanubile ve Visaggio, 1997)*:* Kestirim modeli inşa etmek üzere bir önceki yazılım sürümünün metrikleri ve kusur verileri kullanılır. Daha sonra, yeni yazılım metrikleri girdileriyle her modülün kusur sayısı ya da kusur eğilimlilik etiketleri elde edilir. Esasında yazılım kalitesi çok boyutlu bir kavramdır ve kalite kestirim modellerine diğer kalite özellikleri de dahil edilmelidir. Güvenilirlik dışında, kullanılabilirlik, erişilebilirlik, güvenlik gibi birçok kalite özelliği söz konusudur. Ancak, araştırmacılar kalite kestirimi ve kusur kestirimi terimlerini literatürde birbiri yerine geçecek şekillerde kullanmışlardır.

- *Yazılım büyüklük tahminleme:* Yapay sinir ağları ve genetik programlama ile bazı araştırmacılar yazılım büyüklük tahminini gerçekleştirmeye çalışmıştır (Dolado, 2000). Makine öğrenmesi algoritmaları dışında; Fonksiyon Nokta yöntemleri ile (IFPUG, Mark-II, COSMIC) büyüklük tahminleme yapılmaktadır. Elde edilen fonksiyon nokta değerlerinden, bazı eşitliklerle kod satır sayısı da hesaplanabilmektedir.

- *Yazılım geliştirme maliyet kestirimi:* Yazılım büyüklüğü belirlendikten sonra; bakım, işletme maliyetleri, insan emeği gibi faktörler dikkate alınarak geliştirme maliyeti kestirilebilir. Bazı araştırmacılar Bayes analizi ve farklı yöntemlerle maliyet kestirimi konusunda çalışmalar yürütmüşlerdir (Briand vd., 1999a; Chulani vd., 1999).

- *Proje geliştirme çabası kestirimi:* Senaryo tabanlı usavurum (case based reasoning-CBR), yapay sinir ağları ve genetik programlama farklı araştırmacılar tarafından bu problem için kullanılmıştır (Shepperd ve Schofield, 1997; Srinivasan ve Fisher, 1995).

- *Bakım çabasının kestirimi:* Yapay sinir ağları ve karar ağacı modelleme, farklı araştırmacılar tarafından bu problemde kullanılmıştır (Jorgensen, 1995).

- *Düzeltme maliyeti tahminlemesi (correction cost estimation):* Her kusurun düzeltilmesi gerekmektedir ve kusur düzeltme maliyeti tahminlenmelidir. Karar ağacı modelleme ve "tümevarımsal mantık programlama" (inductive logic programming), farklı araştırmacılar tarafından bu problem için kullanılmıştır (De Almeida vd., 1998).

- *Yazılım güvenilirlik kestirimi:* Güvenilirlik modelleri zamanla birlikte güvenilirliğin nasıl değiştiğini karakterize eder. Güvenilirliğin gelecekteki değerleri güvenilirlik modeli kullanılarak kestirilir. Bu amaçla, yapay sinir ağları kullanılmıştır (Karunanithi vd., 1992).

- *Yazılım kusur kestirimi:* Yazılım kalite kestirimi problemi içinde açıklanmıştır.
- *Yeniden kullanılabilirlik kestirimi:* Kalıtım, karmaşıklık gibi özelliklerle yeniden kullanılabilirliğin kestirimi yapılabilmektedir. Karar ağacı modelleme tekniği bu amaçla kullanılmıştır (Mao vd., 1998).
- *Yazılım yayım zamanlama kestirimi (software release timing prediction):* Maliyeti en düşük değerlere indirme kriteri, zaman serisi kestirim problemine dönüştürülmektedir. Kullanıcı, geliştirici ve pazar bu kestirimden etkilenmektedir çünkü erken veya geç yayımlama pazar dengelerini alt üst edecektir. Yapay sinir ağları bu kapsamda kullanılmış olan tekniklerdendir (Dohi vd., 1999).
- *Program modüllerinin test edilebilirliğinin kestirimi:* Kaynak kodundan bazı metrikleri elde ederek gerçekleştirilebilir. Yapay sinir ağaçları bu amaçla kullanılmıştır (Khoshgoftaar vd., 2000b).

Bu araştırma konularından görüldüğü gibi, yazılım mühendisliğinde birçok görevin kestirimini yapmak mümkündür ve bu tür çabalar, yeterli zamanda istenen güvence seviyesine yazılım geliştiricileri ulaştırabilir. Yazılım geliştirme süresi boyunca kestirim daima ön planda tutulmalı ve yazılım yaşam çevriminin her süreci içinde bu tür kestirim yöntemlerine başvurulmalıdır.

Bu yaklaşım dahilinde ele alınan süreçlere, bu tez çalışmasında *"kestirim merkezli yazılım süreçleri" (prediction-centric software processes)* ve tüm yazılım yaşam çevrimine *"kestirim merkezli yazılım yaşam çevrimi"* adı verilmiştir. Bu kavramsal yaklaşımın, mevcut uygulama pratiklerini ve süreçlerini iyileştireceği öngörülmektedir. Örneğin; sistem testleri öncesinde kusur kestirim modelleri ile sistem test senaryolarını önceliklendirerek testleri kusur eğilimli modüllere odaklamak test sürecini iyileştirecek bir yaklaşımdır. Ayrıca, gerçekleme süreci içerisinde program modüllerin test edilebilirliliği kestirilebilir ve mevcut test edilebilirlik seviyesine bağlı olarak uygun eylemler gerçekleştirilebilir.

Yazılım yaşam çevriminde kullanılabilecek, tahminleme ve kestirim yaklaşımları uygun süreçlere göre aşağıda gruplandırılmıştır:

- İsterler Analizi: Büyüklük tahminleme, maliyet kestirimi, çaba kestirimi
- Tasarım: Kalite kestirimi, kusur kestirimi, yeniden kullanılabilirlik kestirimi
- Gerçekleme: Güvenilirlik kestirimi, kalite kestirimi, kusur kestirimi, yeniden

kullanılabilirlik kestirimi, yayım zamanlaması kestirimi, program modüllerinin test edilebilirliliğinin kestirimi.

- Test: Test çabası kestirimi, düzeltme maliyeti kestirimi.
- Bakım: Düzeltme maliyeti kestirimi, bakım yapılabilirliğin tahminlenmesi.

Yazılım yaşam çevrimi süresince çeşitli kestirim yöntemlerine başvurmak mevcut yazılım geliştirme anlayışımızı değiştirerek yazılım kalitesini arttıracaktır. Yapılan incelemelere göre, kestirimi temel alan ilk yaşam çevrimi modeli bu çalışmada açıklanan "kestirim merkezli yazılım yaşam çevrimi" modelidir. Bu model içerisindeki kestirim alanlarının genişlemesi, yazılım mühendisliği içerisinde deneysel çalışmaların artmasıyla mümkün olacaktır. Kestirimin çok az uygulandığı günümüz pratikleriyle ortaya konan bu modelin, ultra geniş ölçekli sistemler için çok farklı kestirim alanlarını kendisine katacağı öngörülmektedir.

4.4 Kusur Kestirim Tabanlı Test Senaryosu Önceliklendirme

Tüm test senaryolarını (test cases) çalıştırmak için yeterli zaman ve kaynağın olmadığı durumlarda veya öncelikli olarak çalıştırılması gereken test senaryolarını belirlemek için bir ihtiyacın olduğu durumda "Test Senaryosu Önceliklendirme" (Test Case Prioritization) yöntemlerine ihtiyaç duyulur. Test senaryolarından oluşan test kümesi (test suite) çok büyük değil ise bu önceliklendirme, test ekibi tarafından kolaylıkla gerçekleştirilebilir ancak test senaryolarının sayısı 100 ya da 1000 gibi değerlerle ifade ediliyorsa bu işlemin farklı yöntemlerle otomatik olarak gerçekleştirilmesi kaçınılmazdır. Test senaryosu önceliklendirme yaklaşımları sayesinde, kritik hatalar test aşamasının başlangıcında hızlıca belirlenebilmekte, kusur tespit oranı ve kapsama (coverage) oranı artmaktadır. Microsoft'ta yapılan araştırmalarda, büyük ölçekli sistemler için bu tür önceliklendirme yaklaşımlarının oldukça etkin çalışabildiği raporlanmıştır. Ayrıca; literatürde test senaryosu önceliklendirme yaklaşımlarının, rastgele önceliklendirme yapıldığı veya hiç önceliklendirme yapılmadığı durumdan daha etkin şekilde kusurları saptayabildiği ifade edilmiştir. Literatürde, çok farklı test senaryosu önceliklendirme yöntemi mevcuttur. Bu yöntemlerden birisi, "toplam fonksiyon kapsama önceliklendirmesi" (total function coverage prioritization) yöntemidir. Bu yöntem, test senaryolarının işlettiği (execute) fonksiyon sayılarına göre test senaryolarını sıralar ve en fazla fonksiyonu işleten test senaryosu, en yüksek önceliğe sahip olur. Birden fazla test senaryosu aynı sayıda fonksiyonu işletiyorsa, bu test senaryoları kendi aralarında rastgele sıralanmaktadır.

Bu önceliklendirme yönteminin temelinde, büyük bir problem bulunmaktadır. Fonksiyonların basit ya da karmaşık olup olmadığı, bu yöntemde dikkate alınmamıştır. Ekrana basit mesajlar yazan bir fonksiyonla, 40-50 adet bağımsız yola sahip olan karmaşık bir fonksiyon kesinlikle aynı şekilde ele alınmamalıdır. Aksi halde bu basit fonksiyonlar hata çıkarmayacak olmasına rağmen, ilgili test senaryosunun yüksek öncelik değerine sahip olmasına neden olabilir. Hatalara neden olan fonksiyonlar çoğu durumda; çok fazla sayıda bağımsız yola sahip olan, karmaşık ve anlaşılması güç olan fonksiyonlardır.

Bu nedenle, bu tez kapsamında kusur kestirim tabanlı yeni bir test senaryosu önceliklendirme yöntemi önerilmiştir. "Toplam kusur eğilimli fonksiyon kapsama önceliklendirmesi" (total fault-prone function coverage prioritization) adı verilen bu yöntemde, her fonksiyon aynı değere sahip değildir. Bu yöntemde, test senaryolarının işlettiği (execute) kusur eğilimli fonksiyonların sayılarına göre test senaryoları sıralanır ve en fazla kusur eğilimli fonksiyonu işleten test senaryosu, en yüksek önceliğe sahip olur. Birden fazla test senaryosu aynı sayıda kusur eğilimli fonksiyonu işletiyorsa, bu test senaryoları kendi aralarında rastgele sıralanmaktadır. Önerilen bu yaklaşım, literatürde mevcut olan diğer yaklaşımlar gibi kod tabanlıdır. Bu test senaryosu önceliklendirme yaklaşımında, kusur eğilimli fonksiyonlar test aşamasından önce kusur kestirim yöntemleri ile belirlenmektedir. Bölüm 7 ve 8'de, kusur kestirim yöntemleri detayları ile sunulmaktadır.

Aşağıdaki örnekte 3 test senaryosu ve işlettiği fonksiyonlar gösterilmektedir. Literatürdeki yönteme göre, işlettikleri fonksiyon sayıları ile orantılı olarak test senaryosu önceliklendirme sırası, TS3 > TS1 > TS2 şeklinde belirlenecektir.

Test Senaryosu 1: *f1, f3, f2*

Test Senaryosu 2: *f2*

Test Senaryosu 3: *f3, f4, f5, f6*

Önerdiğimiz yöntemde ise, kusur eğilimli fonksiyonlar veya karmaşık olarak ifade edilebilecek fonksiyonlar kusur kestirim yöntemleri ile test öncesinde belirlenmektedir. Aşağıdaki şekilde test öncesinde bu fonksiyonların iki gruba ayrıldığını varsayalım.

f1, f2 → kusur eğilimli fonksiyonlar

f3, f4, f5, f6 → kusur eğilimli olmayan fonksiyonlar

Bu durumda; önerdiğimiz yöntemde önceliklendirme sırası TS1 > TS2 > TS3 olacaktır.

5. METRİKLER, DEĞERLENDİRME KRİTERLERİ VE VERİ KÜMELERİ

Bu bölüm, gerçekleştirilen deneysel çalışmalara ilişkin veri kümelerini, değerlendirme kriterlerini ve metrikleri sunmaktadır. Bölüm 5.1 yazılım metriklerini, Bölüm 5.2 değerlendirme kriterlerini ve Bölüm 5.3 veri kümelerini açıklamaktadır.

5.1 Yazılım Ölçümü ve Yazılım Metrikleri

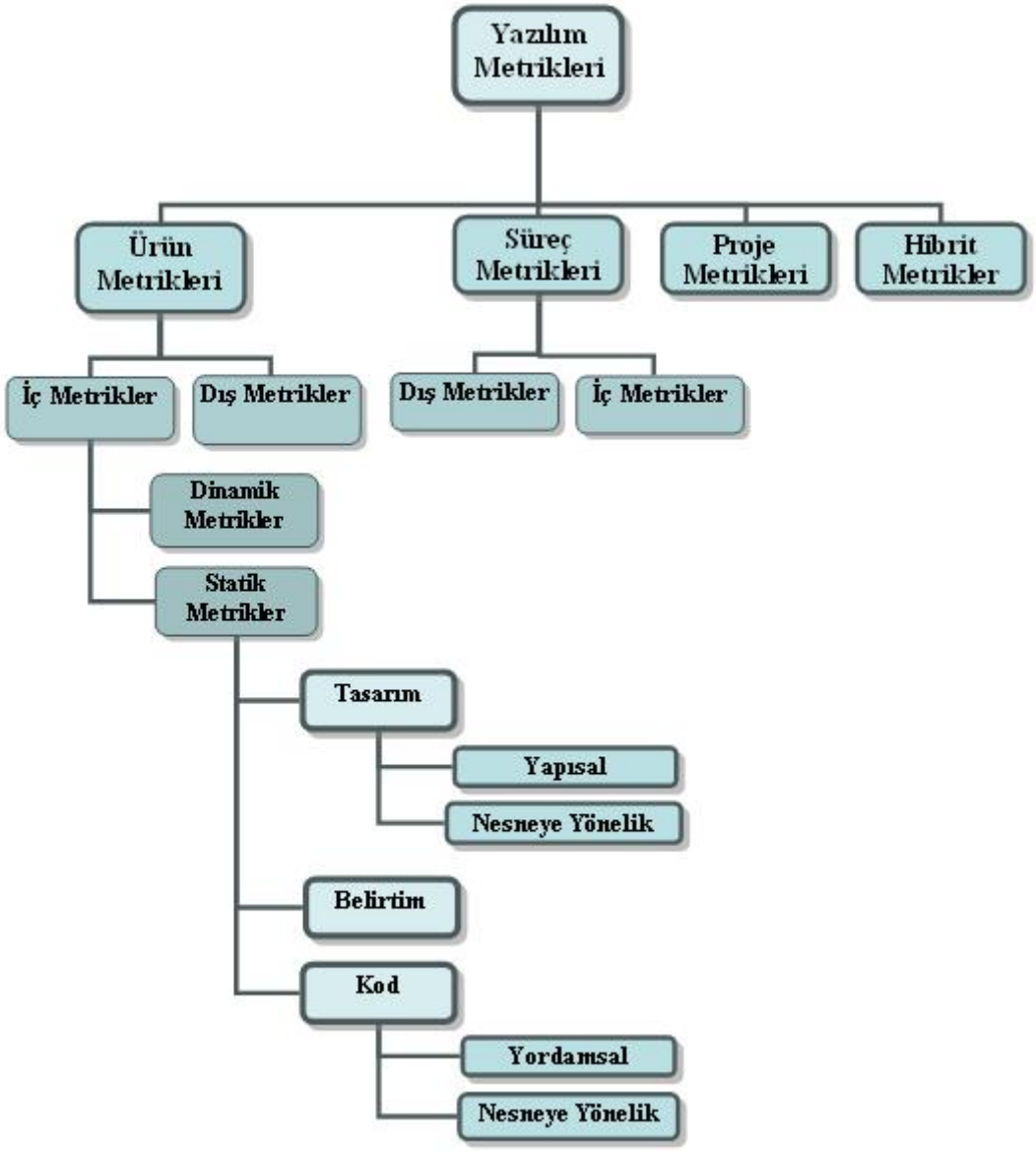
Yazılım Mühendisliği disiplini, Bilgisayar Bilimleri disiplininden evrimleşerek ortaya çıkmıştır. Fizik ve makine mühendisliği disiplinleri arasındaki ilişkiyi, bilgisayar bilimleri ve yazılım mühendisliği disiplinleri arasındaki ilişkiye benzetmek mümkündür. Fizik, doğa kanunları için teorik alt yapıyı geliştirmeye çalışırken, makine mühendisliği fiziksel dünyadaki problemleri çözmek için fiziğin temel kurallarını kullanır. Benzer şekilde; bilgisayar bilimleri bilgisayar ve hesaplama için teorik alt yapıyı geliştirirken yazılım mühendisliği alanı bilgisayar bilimlerinin uygulamalı bir disiplinidir (Munson, 2003).

Bilimsel deneylerin çıktıları, bilimsel topluluğun kabul ettiği birimlerle sayısal olarak ifade edilir ve bu birimler standartlar halinde ortaya konulur. Örneğin; Amerika Birleşik Devletleri'nde (ABD), Ulusal Standartlar ve Teknoloji Enstitüsü (National Institute for Standards and Technology - NIST) bu ölçüm standartlarının ortaya konulması ile görevlidir. Ancak ne NIST ne de farklı bir ulusal standart organizasyonu yazılım ölçümleri için herhangi bir standardı henüz sunmamakta ve bu durum, yazılım mühendisliği alanında gerçekleştirilen deneylerin paylaşılması noktasında çeşitli sorunlara neden olmaktadır (Munson, 2003).

Yazılım ölçümünde temel olarak 2 kural vardır. İlki ölçümlerin yeniden üretilebilir olması ve ikincisi ölçülen özelliklerin o bağlamda geçerli olmasıdır. Ayrıca, türetilmiş metrikler oluştururken farklı kavramsal metrikler toplanmamalıdır. Örneğin; düzeltilmemiş fonksiyon büyüklüğü (unadjusted function point) hesaplanırken girdi sayısı, çıktı sayısı, sorgulama (inquiry) sayısı, dış dosya sayısı ve iç dosya sayısı gibi metriklerin belirli ağırlıklandırmalardan sonra toplanması son yıllarda büyük eleştiriler almaktadır.

Yazılımlardan çeşitli ölçümler gerçekleştirilerek sayısal değerler elde edilebilmektedir ancak bu sayısal değerleri anlamlı kılan, bilgiye dönüşmüş halleri olan yazılım metrikleridir. Yazılım Mühendisliği disiplinini, sanat boyutundan bilimsel düzeye taşıyacak yaklaşımlardan birisi de bu metriklerin geliştirme sürecinin her aşamasında ölçülmesi ve izlenmesidir. Yazılım metriklerini daha iyi anlayabilmek için, bu metrikleri sistematik olarak sınıflandırmak gerekir. Şekil 5.1'de verilen yazılım metrikleri taksonomisi, El-Wakil ve

arkadaşları (2004) tarafından ortaya konulmuştur ve Fenton ve Pfleeger'in (1997) 3 temel metrik sınıfı (süreç, ürün, proje) yaklaşımını genişletmektedir.



Şekil 5.1 Yazılım metrik taksonomisi (El-Wakil vd., 2004)

Taksonomi içerisindeki alt sınıflar aşağıda açıklanmaktadır (El-Wakil vd., 2004):

- *Süreç Metrikleri (process metrics):* Yazılım yaşam çevrimi ile ilişkili özellikler ölçülmektedir. Örneğin; "yürütülen test senaryosu başına düşen kusur sayısı" metriği test sürecinde toplanabilecek bir süreç metriğidir. Takım liderleri çoğunlukla bu metriklerle ilgilenmektedir ve bu gruptaki metriklerden en önemlileri; zaman, çaba ve maliyet ile ilişkili olanlarıdır. İç süreç metrikleri geliştirmeyi yapan firmanın

süreçleriyle ilişkili iken, dış süreç metrikleri müşteri memnuniyeti gibi geliştirme dışında kalan süreçleri içermektedir. Yazılım geliştiren firmalar ya da kurumlar yeniden organize olduğunda, süreçleri değişebileceğinden süreç metrikleri de etkilenebilmektedir. Ayrıca, farklı yazılım yaşam çevrimleri için uygulanan süreçler farklı olabileceğinden, farklı süreç metrikleri kullanılması gerekmektedir. Bu sebeplerle, yazılım ölçüm alanında her firmanın uygulayabileceği ortak süreç metrikleri oluşmamıştır.

- *Proje Metrikleri (project metrics):* Bu metrikler, kaynak (resource) metrikleri olarak da bilinmektedir. Projedeki kaynakları ortaya koyan bu metriklere örnek olarak; geliştirici sayısı, geliştirici yeteneği (skill), güvenilirlik ve performans örnek olarak verilebilir.

- *Hibrit Metrikler (hybrid metrics):* Süreç ve ürün metriklerinin bir arada kullanıldığı metriklerdir. Örnek olarak; fonksiyon nokta başına düşen maliyet metriği verilebilir.

- *Ürün Metrikleri (product metrics):* Yazılım yaşam çevriminin çıktılarını tanımlayan metriklerdir. Yazılım mühendisleri ve yazılım kusur kestirimi konusunda çalışan araştırmacılar ağırlıklı olarak bu gruptaki metriklerle ilgilenmektedir.

    - Dış Ürün Metrikleri (external product metrics): Ürün kullanıcıları tarafından görülebilen metriklerdir. Örnek olarak; ürünün güvenilirliği, fonksiyonelliği, performansı ve kullanılabilirliği (usability) verilebilir. Bu metrikleri ölçmek daha zordur ve iç ürün metriklerinden yararlanarak erken safhada ölçülmeye çalışılırlar.

    - İç Ürün Metrikleri (internal product metrics): Sadece geliştirme takımı tarafından görülebilen metriklerdir, son kullanıcı ürüne kapalı kutu olarak baktığı durumda bu metrikleri göremez. Örnek olarak; kod satır sayısı, çevrimsel karmaşıklık, modül sayısı verilebilir.

        - Dinamik Metrikler: Yazılımın çalıştırılmasının ardından elde edilen metriklerdir. Yazılım metrikleri alanında çalışma yapan araştırmacılar, bu alanda çok az sayıda çalışma gerçekleştirdiğinden henüz ortak kabul edilen metrikler mevcut değildir. İleriki dönemlerde bu alanda da çalışmalar sürdürülmesi, kusur kestirim modellerinin performansını arttırmak üzere bu metriklerin kullanılmasını sağlayabilir.

- Statik Metrikler: Yazılım çalıştırılmadan elde edilen metriklerdir.
    - Belirtim Metrikleri (specification metrics): Yazılım resmi belirtimlerinden (formal specification) metrik türetmek mümkündür. De Marco'nun Bang metriği resmi belirtim notasyonlarından türetilmiştir (El-Wakil vd., 2004). Bu metrik dışında; zayıf sözcelerin (phrase) sayısı ve metinlerin sayısı gibi metrikler yazılım belirtim belgelerinden elde edilebilir.
    - Tasarım Metrikleri: Tasarım aşamasında elde edilebilen metrikler bu gruba girmektedir. Kullanılan tasarım yaklaşımına göre, bu metrikleri El-Wakil ve arkadaşları (2004) 2 alt grupta göstermiştir. Ancak, yeni paradigmalarla birlikte bu grubun sayısını arttırmak mümkündür. Örneğin; bileşen tabanlı ve ilgiye yönelik metrikleri bu gruba dahil edebiliriz. Henüz bu paradigmalara ilişkin kabul edilmiş bir metrik kümesi oluşmadığından, taksonomide bunlara yer verilmemiştir.
        - Yapısal: Yapısal tasarıma ilişkin metrikler bu gruba girmektedir. Henry&Kafura'nın bilgi akış metriği örnek olarak verilebilir.
        - Nesneye Yönelik: Nesneye yönelik programlarda kalıtım, sarma (encapsulation), bağlılık (coupling), kohezyon ölçülerek elde edilen metriklerdir.
    - Kod Metrikleri: Doğrudan kod üzerinden yapılan ölçümlere ilişkin metriklerdir.
        - Yordamsal: Kod satır sayısı, McCabe ve Halstead metrikleri bu alanda kullanılan metriklerdir. Yordamsal programlama dili ile geliştirilmiş programların ölçümleri bu metriklerle gerçekleştirilir.
        - Nesneye Yönelik: CK metrik kümesi, QMOOD ve MOOD metrik kümeleri örnek olarak verilebilir.

5.2 Yordamsal Kod Metrikleri

1968 yılında NATO'nun organize ettiği ilk Yazılım Mühendisliği konferansında, yazılım sistemlerinin gittikçe karmaşıklaştığı ve Yazılım Krizi'nin mevcut olduğu ifade edilmiştir. Test edilebilir ve bakım yapılabilir yazılımların geliştirilebilmesi için yazılımların modüler geliştirilmesi gerektiği o yıllardan günümüze kadar gelmiş ve halen geçerliliğini korumaktadır.

Programlama paradigmalarındaki değişimler, daha üst soyutlama düzeyinde yazılım geliştirilmesini sağlamıştır. Bu değişimi; makine kodunun kullanılması, assembly programlama dili, 3. jenerasyon olarak ifade edilebilen C gibi yordamsal (procedural) diller, SQL (structured query language) gibi 4. jenerasyon diller, C++ gibi nesneye yönelik programlama dilleri, bileşen tabanlı geliştirme, servis yönelimli yaklaşımlar, yazılım ürün hatları ve yazılım fabrikaları şeklinde kronolojik olarak sıralayabiliriz.

Bu bölümde C gibi yordamsal programlama dillerinde geliştirilmiş kodlar için kullanılabilen metrikler açıklanmaktadır. Nesneye yönelik programlamada da benzer şekilde, metotlar yer aldığı için buradaki metot seviyesindeki metrikleri o yaklaşımlarda kullanmak mümkündür. En çok bilinen yordamsal kod metrikleri, Halstead ve McCabe metrikleridir.

5.2.1 Halstead Metrikleri

Maurice Halstead, yazılım ölçümü konusunda Thomas McCabe ile birlikte öncü isimlerdendir. Halstead, yazılım bilimi metriklerini (software science metrics), primitif ve türetilmiş metrikler olarak 2 gruba ayırmıştır (Halstead, 1977). Programların operatör ve operandlardan oluştuğunu; operatör ve operand sayılarının program karmaşıklığı ile ilişkili olduğunu ifade etmiştir. Operatörler, derleyicinin derleme ya da çalışma zamanında bir eylem gerçekleştirmesine neden olur (Munson, 2003). Diğer program sembolleri (token) ise operatörlerin ihtiyaç duyduğu operandlardır. Yorum (comment) satırları ne operand ne de operatördür.

Halstead aşağıdaki metrikleri, primitif metrikler olarak açıklamıştır:

- μ_1: eşsiz (unique) operatör sayısı
- μ_2 : eşsiz (unique) operand sayısı
- N_1: toplam operatör sayısı
- N_2: toplam operand sayısı

Halstead, primitif metrikleri kullanarak yeni metrikler türetmiştir ancak bu türetilmiş metrikler birçok açıdan eleştirilmiştir.

İlk olarak; programın gerçekleme uzunluğu (implementation length of program) ve sözcük (vocabulary) metriklerine bakalım. Eşitlik 5.1 ve 5.2'ye göre, primitif metriklerin toplamlarının alındığı görülmektedir ve bu yeni metriklerin bu şekilde yeni bilgiler ortaya koyması mümkün değildir.

$N = N_1 + N_2$ (gerçekleme uzunluğu) (5.1)

$\mu = \mu_1 + \mu_2$ (sözcük) (5.2)

Bilgisayar bilimlerinde bir değerin 2 tabanına göre logaritmasının alınması gelenek haline geldiğinden, programın hesaplanmış uzunluğu (calculated length of program) ve programın hacmi (volume) metriklerinde 2 tabanına göre logaritma uygulanmıştır (Munson, 2003). Eşitlik 5.3 ve 5.4'de bu türetilmiş metriklerin hesaplanma yöntemi gösterilmektedir.

$\tilde{N} = \mu_1 \log_2 \mu_1 + \mu_2 \log_2 \mu_2$ (programın hesaplanmış uzunluğu) (5.3)

$V = N \log_2 \mu$ (programın hacmi) (5.4)

Temel bileşen analizi (principal component analysis) gerçekleştirilirse, bu iki yeni metriğin primitif metriklerle aynı şekilde değiştiği, bu nedenle yeni bilgiler sunmadığı görülecektir.

Bu noktada yeni metrikler türetmek üzere, 2 yeni kavram ortaya konulmuştur:

- μ_1*, gerçeklenen algoritma için farklı operatörlerin minimum sayısı
- μ_2*, gerçeklenen algoritma için farklı operandların minimum sayısı

Bu yeni kavramlardan yeni metrikler türetilmek istenmesine rağmen, bu özelliklerin gerçek değerlerini ölçecek kurallar mevcut değildir. Bu nedenle, bu önerilen özellikler ölçümlere gürültü katmaktadırlar. Bu metrikler, yazılım metrik uzmanları tarafından yıllarca kullanılmış ancak bu metriklerin geçerlenmesi yapılmamıştır (Munson, 2003). Munson'a göre, 4 primitif metrik yeterlidir.

Yazılım kusur kestirim çalışmalarında bütün metrikleri kullanıp, daha sonra korelasyonun olduğu metrikleri özellik azaltma teknikleriyle yok etmek ve uygun alt metrik kümesini belirlemek mümkündür. Bu sayede, birbirleriyle korelasyon taşıyan metrikler veri kümesinden çıkartılarak model performansı iyileşecektir.

Aşağıda bu özellikleri kullanan türetilmiş metrikler verilmektedir:

$V^* = (\mu) \ (\log_2 \mu^*)$ (programın potansiyel hacmi) (5.5)

$V^{**} = (2+ \mu_2^*) \ (\log_2 \ (2+ \mu_2^*))$ (programın sınır hacmi / minimal gerçekleme) (5.6)

$L = V^* / V$ (program seviyesi) (5.7)

$\acute{L} = (\mu_1^*) \ (\mu_2) \ / \ (\mu_1) \ N_2$ (program seviyesinin alternatif gösterimi) (5.8)

$D = 1 / L$ (bir programı gerçeklemenin zorluğu) (5.9)

$I = \acute{L} * V$ (programın zekâ içeriği) (5.10)

$E = V / L$ (program çabası) (5.11)

$T = E / 18$ (programı yazmak için gerekli zaman) (5.12)

$B = E^{2/3} / 3000$ (programdaki tahmini hata sayısı) (5.13)

5.2.2 NASA Açık Veri Kümelerinde Yer Alan Halstead Metrikleri

Tez çalışması süresince PROMISE havuzunda yer alan NASA açık veri kümeleri kullanılmıştır. Bu veri kümelerinde, 21 metrik özellik (feature) olarak yer almaktadır ve bu metrikler, kusur kestirim modellerinin bağımsız değişkenlerini oluşturur. Yazılım modülünün kusurlu olup olmadığını gösteren özellik ise en son sütunda yer almaktadır. 21 özelliğin 5 tanesi kod uzunluğu ile ilgili olup 4 tanesi primitif Halstead metriği, 8 tanesi türetilmiş Halstead metriği, 1 tanesi dallanma sayısını gösteren metrik (branch count), 3 tanesi McCabe metriğidir. Çizelge 5.1' de NASA açık veri kümelerinde yer alan primitif Halstead metrikleri, Çizelge 5.2'de türetilmiş Halstead metrikleri ve hesaplanmaları için gerekli eşitliklerin numaraları verilmektedir.

Çizelge 5.1 NASA açık veri kümeleri içerisinde yer alan primitif Halstead metrikleri

Sembol	Halstead Primitif Metriği	Gösterim
uniq_Op	Eşsiz (unique) operatörler	μ_1
uniq_Opnd	Eşsiz (unique) operandlar	μ_2
total_Op	Toplam operatörler	N_1
total_Opnd	Toplam operandlar	N_2

Çizelge 5.2 NASA veri kümeleri içerisinde yer alan türetilmiş Halstead metrikleri

Sembol	Halstead Türetilmiş Metriği	Eşitlik No
n	Halstead gerçekleme uzunluğu (length)	6.1
v	Halstead hacmi (volume)	6.4
l	Halstead program seviyesi (level)	6.7
d	Halstead zorluğu (difficulty)	6.9
i	Halstead zekâsı (intelligence)	6.10
e	Halstead çabası (effort)	6.11
b	Halstead hatası (bug)	6.13
t	Halstead zaman kestirimi (time estimator)	6.12

Çizelge 5.2'de geçen türetilmiş metrikler aşağıda açıklanmaktadır:

- *Halstead uzunluğu:* Bir modülü tamamen tanımlamak için gerekli olan sözcüklerin (words) toplam sayısıdır. Toplam operatör sayısı ve toplam operand sayısı toplanarak hesaplanır. 200 civarı oldukça standarttır ve 300'e kadar olması kabul edilebilirdir. 300'den fazla olanlar uzun modül olup, 500'den fazla olanlar ise kötü tasarlanmış ve kötü gerçeklenmiş modüllerdir.

- *Halstead hacmi:* Bir algoritmanın gerçekleme boyutunu tanımlar. 30 ve 1000 arasındaki değerler oldukça standarttır. 1000'den büyük değerlere pek rastlanmaz. 1200'den büyük modüllerin yeniden tasarlanması veya parçalara ayrılması gerekir.

- *Halstead program seviyesi:* Zorluğun tersidir. 1 ve 0,02 arasındaki değerler oldukça tipiktir. 0,02'den küçük değerler hata eğilimli modülleri gösterir ve 0,01'den küçük değerler problemli modülleri işaret eder.

- *Halstead zorluğu:* Bir modülün zorluğunu veya hata eğilimini gösterir. Aynı operandların aşırı kullanılmasına bağlı olarak hatayı ortaya koyan bir metriktir. Tek bir operand fazla sayıda kullanılmışsa ve hatalı atama yapıldıysa, birçok yerde hataya neden olacaktır. 35'e kadar olan değerler normaldir. 50'yi geçen değerler zorluğu ve hata eğilimini ortaya koymaktadır. 100'den büyük değerler karmaşıklığı ve hata

eğiliminin şiddetini gösterir.

- *Halstead zekâsı:* Algoritmanın karmaşıklığının gösterimini sağlar ve teorik olarak programlama dili değişse de bu metrik aynı değere sahip olmalıdır. Hacim ve seviyenin çarpılması ile elde edilir. 50'den küçük değerler kabul edilir düzeydedir. 50 ve 100 arasındaki değerler modülün karmaşık olduğunu gösterir. 100'den büyük değere sahip metotların incelenmesi gerekir.

- *Halstead programlama çabası:* Bir modülü gerçekleme veya anlama için gerekli çabadır. 100,000'den küçük değerler oldukça standarttır. 100,000 ve 500,000 arasındaki değerler düşük kaliteli kodu işaret eder. 500,000'den büyük değerler düşük kalitedeki kodu gösterir ancak tek başına bu metrik kullanılmamalıdır.

- *Halstead hatası:* Kod içinde yer alan hataların sayısına ilişkin tahmini gösterir. Yazılımın karmaşıklığına bağlı olarak artar. Bir dosyadaki Halstead hata sayısı, 2'den küçük olmalıdır.

- *Halstead zaman kestirimi:* Modülü anlamak veya gerçeklemek için gerekli zamanı gösterir. 3600 değeri programcının 1 saat zaman harcaması gerektiğini gösterir. 5000'e kadar olan değerler kabul edilebilirdir. 5000'den büyük değerler modülün düşük kalitede olduğunu tek başına göstermez, diğer metriklerle birlikte değerlendirilmelidir.

NASA açık veri kümelerinde yer alan ve kod satır sayısı ile ilişkili olan Halstead metrikleri Çizelge 5.3'de verilmektedir.

Çizelge 5.3 NASA veri kümeleri içerisinde yer alan Halstead kod uzunluk metrikleri

Sembol	**Halstead Kod Uzunluk Metriği**
lOCode	Sadece kod ve boşluk içeren satır sayısı
lOComment	Sadece yorum içeren satır sayısı
lOBlank	Sadece boş satırların sayısı
lOCodAndComment	Yorum ve kod içeren satırların sayısı

5.2.3 McCabe Metrikleri

McCabe (1976), yazılımın kontrol akışını ölçen ve çevrimsel karmaşıklık (cyclomatic complexity) adını verdiği bir yöntem geliştirmiştir. Bu metrik karmaşıklığı ölçerek bakım yapılabilirliği ve testi güç olan modülleri belirlemeyi sağlamaktadır. Halstead'in geliştirdiği yöntemler, modüllerin okunurluğuna odaklanmışken McCabe, modüller için karmaşık yollara (path) odaklanmıştır. McCabe metrikleri içerisinde, en fazla popülerlik kazanan metrik çevrimsel karmaşıklık olup daha birçok McCabe metriği mevcuttur. Farklı programlama dilleri için bu metriklerin ölçümlerini McCabe&Associates, Inc. firmasının McCabe IQ aracı yapabiliyor olmasına rağmen lisanslama fiyatının yüksekliği tercih edilmesini sınırlandırmaktadır. NASA açık veri kümelerinde, 4 tane McCabe metriği kullanıldığı için bu tez çalışmasında bu metriklerden yararlanılmıştır. Bu metrikler; McCabe kod satır sayısı, McCabe çevrimsel karmaşıklık, McCabe temel (essential) karmaşıklık ve McCabe tasarım karmaşıklığıdır. McCabe kod satır sayısı, McCabe kod sayma kurallarına göre hesaplanır. Bu bölümde NASA kümelerinde yer alan McCabe metrikleri açıklanacak olup diğer McCabe metriklerine yer verilmeyecektir. Bu metrikler dışında; tümleştirme karmaşıklığı, normalize edilmiş karmaşıklık, global veri karmaşıklığı, belirtilmiş (specified) veri karmaşıklığı, patolojik karmaşıklık gibi birçok McCabe metriği mevcuttur.

5.2.4 NASA Açık Veri Kümelerinde Yer Alan McCabe Metrikleri

Bu bölümde sırasıyla McCabe çevrimsel karmaşıklık, temel karmaşıklık ve tasarım karmaşıklık metrikleri açıklanacaktır.

5.2.4.1 Çevrimsel Karmaşıklık

Bir modül içerisinde yer alan lineer bağımsız yolların sayısı, çevrimsel karmaşıklık metriğinin sayısal değerini vermektedir. Modülde yer alan karar yapılarına göre, modül karmaşıklığı değişmekte ve test edilebilirlik bu metriğin sayısal değerinin artmasıyla birlikte düşmektedir. Modül için bir akış grafı çizilerek bu akış grafında düğümler program deyimlerini (statement) gösterirken yönlü oklar bir deyimden diğerine geçen akışı temsil etmektedir. Çevrimsel karmaşıklık, eşitlik 5.14'e göre hesaplanmakta, e ile graftaki ayrıtların (edge) sayısı, n ile düğümlerin (node) sayısı gösterilmektedir.

$$V(G) = e - n + 2 \tag{5.14}$$

Graf teorisinin yanı sıra pratik bir yöntem ile de bu metrik elde edilebilir. Her modül için bu metrik değeri 1 değeri ile başlatılır ve her *if, while, for, repeat, and, or, switch*'deki her *case*

için 1 arttırılarak değer elde edilir. Pratik yöntem sayesinde, geliştiriciler kodlama aşamasında hızlıca karmaşıklıkları hesaplayabilmekte ve dikkat etmesi gereken noktaları görebilmektedir. Yazılım Mühendisliği Enstitüsü, çevrimsel karmaşıklık ve programların risk değerlendirmesi arasında ilişki kurmuştur. Çizelge 5.4'de çevrimsel karmaşıklık değerlerine göre programların risk değerlendirilmesinin nasıl yapılabileceği gösterilmektedir.

Çizelge 5.4 Çevrimsel karmaşıklık ve risk değerlendirmesi

Çevrimsel Karmaşıklık	**Risk Değerlendirmesi**
1-10	Basit program, risksiz
11-20	Daha karmaşık, orta risk
21-50	Karmaşık, yüksek riskli
> 50	Test edilemez, çok yüksek riskli

Çizelge 5.4'de verilen değerler fonksiyon başına hesaplanmakta olup, yüksek riskli olduğu belirlenen fonksiyonların yeniden düzenlenmesi (refactoring) sayesinde sayısal değerleri azaltılabilir. NASA yaptığı deneysel çalışmalarda, çevrimsel karmaşıklığın eşik değerinin 20 olduğunu göstermiştir (Laird ve Brennan, 2006). Bu değerin 50'den büyük olması, kod bloğunun alarm verdiği ve çok yüksek riskli olduğu anlamına gelir. Yazılım üzerinde testler gerçekleştirilirken, akış grafındaki bağımsız lineer yolların her birinin en azından bir kez işletilmesi gerekmektedir. Bu metriğin ölçülmesi sayesinde; kodun karmaşık olmayan bölümleri belirlenerek kod gözden geçirmelerinin yoğunlaşması gereken modüller saptanabilir ve daha fazla test gerektiren modüller ortaya çıkarılır.

Şekil 5.2'de çevrimsel karmaşıklık hesabını örneklemek için bir kod bloğu verilmektedir.

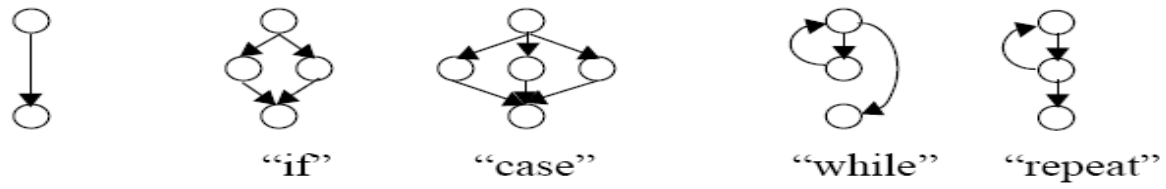
```
157 /**
158  * strncat - Append a length-limited, %NUL-terminated string to another
159  * @dest: The string to be appended to
160  * @src: The string to append to it
161  * @count: The maximum numbers of bytes to copy
162  *
163  * Note that in contrast to strncpy, strncat ensures the result is
164  * terminated.
165  */
166 char *strncat(char *dest, const char *src, size_t count)
167 {
168         char *tmp = dest;
169
170         if (count) {
171                 while (*dest)
172                         dest++;
173                 while ((*dest++ = *src++) != 0) {
174                         if (--count == 0) {
175                                 *dest = '\0';
176                                 break;
177                         }
178                 }
179         }
180         return tmp;
181 }
```

Şekil 5.2 Linux çekirdeğinden örnek bir kod parçası

Şekil 5.2'de verilen kod bloğu, "/linux/lib/string.c" dosyası içerisinden alınmıştır. Bu fonksiyon içerisinde; 2 tane if, 2 tane while olduğundan bu fonksiyonun çevrimsel karmaşıklığının 5 (1 + 4 = 5) olduğu kolaylıkla söylenebilir.

5.2.4.2 Temel Karmaşıklık

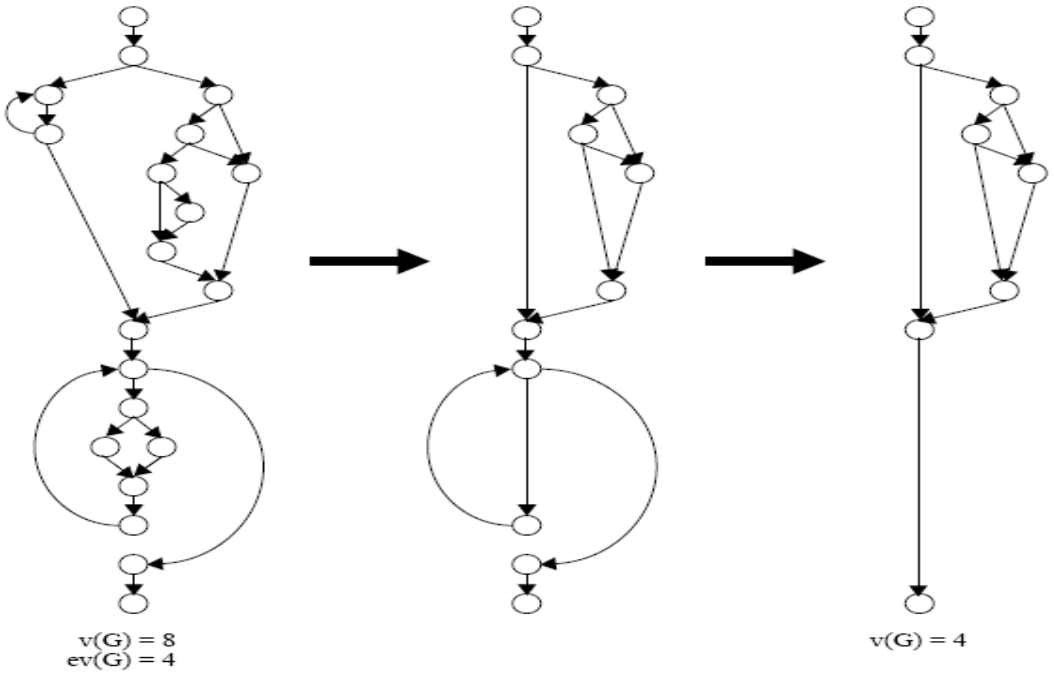
Yapısal programlamanın (structured programming) temel operasyonları olan sıralama (sequence), seçim (selection) ve iterasyon Şekil 5.3'de gösterilmektedir (Watson ve McCabe, 1996). Seçim operasyonu *if* ve *case* anahtar sözcüklerini, iterasyon operasyonu *while* ve *repeat* anahtar sözcüklerini temsil ederken Şekil 5.3 en solda verilen graf sıralamayı göstermektedir. Dikkat edilirse her primitif yapı, bir girişli ve bir çıkışlıdır.



Şekil 5.3 Yapısal programlamanın temel operasyonları (Watson ve McCabe, 1996)

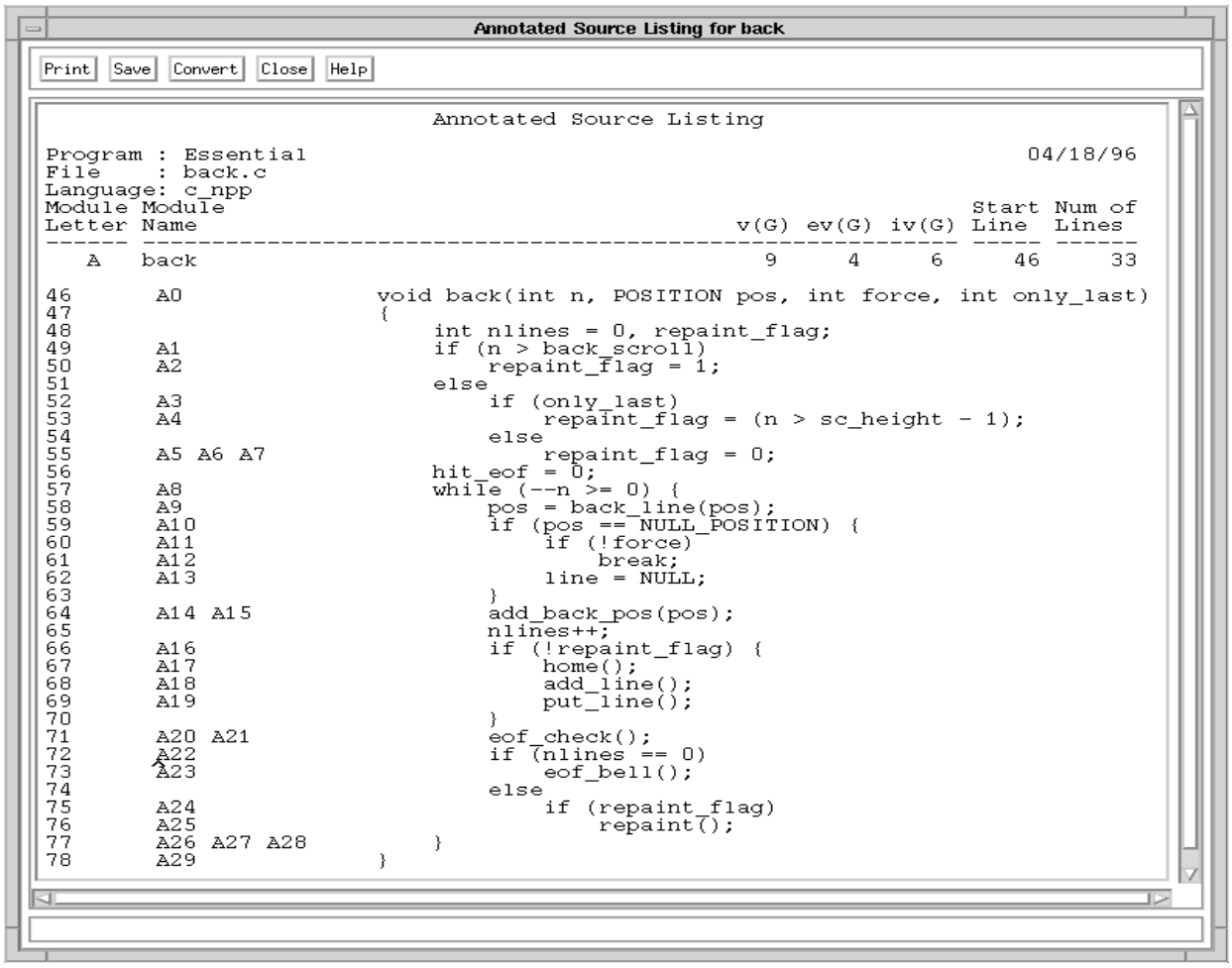
Temel karmaşıklık (ev(G)) değeri için öncelikle modülün akış grafından Şekil 5.3'de verilen yapılandırılmış programlama primitifleri çıkarılır ve kalan azaltılmış grafta çevrimsel

karmaşıklık hesaplanır. Yapısal programlama dilleri için "*goto*" ifadesinden sakınılması önerilmekte ve bu ifadenin kullanımı temel karmaşıklığı arttırmaktadır. Bu metriğin değeri en az 1 olabilmekte ve en fazla çevrimsel karmaşıklığa eşit olmaktadır. NASA bu metriğin eşik seviyesini 4 olarak belirlemiştir. Bir girişli-bir çıkışlı alt akış grafları "D-prime graflar" olarak da bilinmektedir ve bu grafların akış grafından çıkarılması ile ortaya çıkan grafın çevrimsel karmaşıklık değeri, temel karmaşıklık değeri olarak belirlenir.



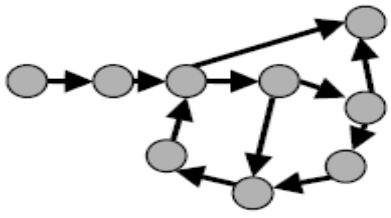
Şekil 5.4 Temel karmaşıklık hesaplama örneği (Watson ve McCabe, 1996)

Şekil 5.4'de temel karmaşıklığın hesaplanması bir örnek üzerinde gösterilmektedir. Akış grafının çevrimsel karmaşıklığı 8 iken temel karmaşıklık 4 olarak bulunmuştur. Şekil 5.5'de bu grafın ait olduğu C fonksiyonu gösterilmektedir. Temel karmaşıklığın nedeni, döngüden şarta bağlı olarak "break" ile çıkılmış olmasıdır.

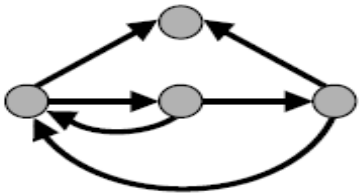


Şekil 5.5 Temel karmaşıklık örneğine ilişkin kaynak kod (Watson ve McCabe, 1996)

Şekil 5.6'da farklı bir örnek verilmektedir. Bu akış grafının azaltılmış hali Şekil 5.7'de gösterilmektedir. Azaltılmış grafa göre temel karmaşıklığın 4 olduğu kolaylıkla söylenebilir. Ayrıca çevrimsel karmaşıklık, graftaki kapalı alanların sayısına 1 eklenerek de hesaplanabilir.



Şekil 5.6 Örnek akış grafı (Laird ve Brennan, 2006)



Şekil 5.7 Örnek azaltılmış akış grafı (Laird ve Brennan, 2006)

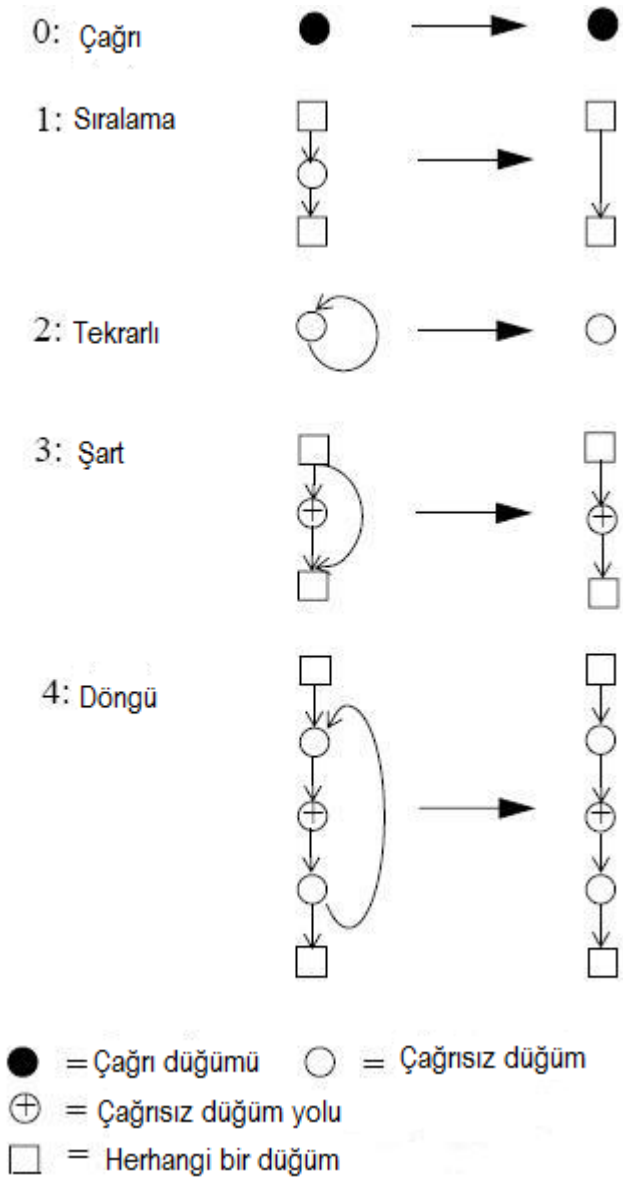
Bu metrik, modülün yapılandırılmamış yapılar (unstructured constructs) içerme derecesini ortaya koymaktadır. Bu tür yapılar, kod kalitesini azaltır ve bakım yapmak üzere gereken çabayı arttırır. Bu değerin yüksek olması diğer bir ifadeyle çevrimsel karmaşıklık değerine yakın olması, modülerliğin ve bakım yapılabilirliğin düşük olduğunu gösterir. İyi yapılandırılmış kod bloklarının, temel karmaşıklık değerinin 1 olması beklenir.

5.2.4.3 Tasarım Karmaşıklığı

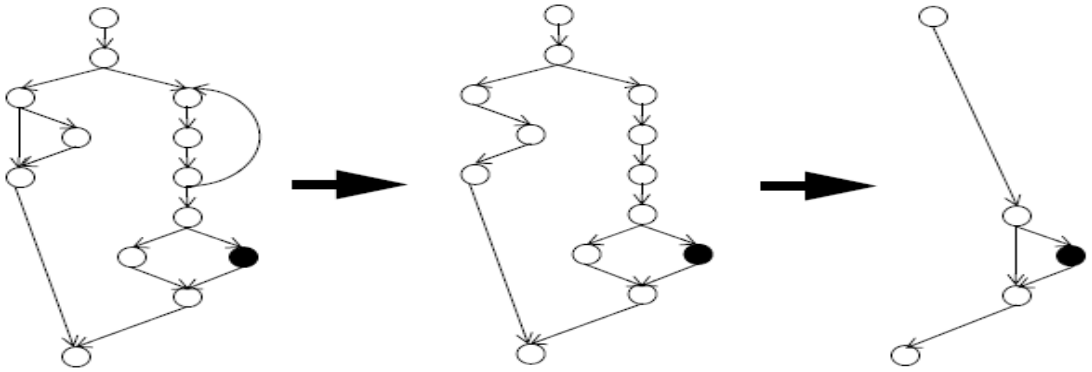
Bu metrik, bir modülün daha alt derecede (subordinate) modüllerle tümleştirilmesi sırasında gerekecek olan test çabasını ölçmektedir. Yüksek tasarım karmaşıklığı değeri, bakım yapılabilirliğin ve yeniden kullanımın düşük olacağını ortaya koymaktadır. Daha alt derecedeki modüllere çağrısı olmayan tüm kararlar ve döngüler akış grafından çıkartılarak, azaltılmış grafın çevrimsel karmaşıklığı hesaplanır ve tasarım karmaşıklığı belirlenmiş olur.

Şekill 5.8'de tasarım azaltma kuralları verilmektedir. Bu kuralların hepsinin uygulanmasının ardından ortaya azaltılmış akış grafı çıkmaktadır. İçi siyah düğümler, fonksiyon çağrılarını göstermektedir ve şekilde "çağrı düğümü" olarak ifade edilmiştir. Kural 0'e göre bu tür düğümler azaltılamaz. İçi boş düğümler ise çağrısı olmayan düğümleri (çağrısız düğüm) temsil etmektedir. Kural 1, çağrısı olmayan düğümlerin sıralamasını yok etmektedir. Kural 2'ye göre tekrarlı (repetative) olan ve çağrı içermeyen modüller yok edilmektedir. Kural 3'e göre, şartlı (conditional) olan ve çağrı içermeyen modüller yok edilmektedir. Kural 4 ise modül çağrısı içermeyen döngüleri yok etmektedir (Watson ve McCabe, 1996).

Şekil 5.9'da tasarım azaltma için örnek bir graf verilmektedir. Orjinal grafın çevrimsel karmaşıklığı 5 iken, son grafın çevrimsel karmaşıklığı diğer bir ifadeyle tasarım karmaşıklığı 2'dir. Kural 3 ve 4 uygulanarak, ara graf elde edilmiştir. Daha sonra Kural 1, sol şartlı dallanmaya 3 kez uygulanmış, ardından Kural 3 uygulanmış ve sonrasında Kural 1, 5 kez uygulanarak grafın son hali elde edilmiştir (Watson ve McCabe, 1996).



Şekil 5.8 Tasarım azaltma kuralları (Watson ve McCabe, 1996)



Şekil 5.9 Tasarım azaltma örneği (Watson ve McCabe, 1996)

5.3 Nesneye Yönelik Metrikler

Metot seviyesindeki metriklerle, kodlama aşaması tamamlanmış bir yazılımın modülleri üzerinde kusur kestirimi yapılabilmektedir. Daha erken aşamalarda bu kestirim yapılabilseydi, yeniden tasarım ya da yeniden yapılandırma ile hataya eğilimli modüllerin metriklere bağlı olarak hataları düzeltilebilir ve test aşamasında ya da müşteride hataları düzeltme ile zaman kaybedilmemiş olurdu. Kusur kestirimi çalışmalarında temel varsayım, yazılımlardaki hataların metriklerle ilişkili olmasıdır. Programcının bilinçli olarak, hata eğilimli olmayacak bir modüle hatalı bir kod eklemesi bu tür kusur kestirim çalışmalarının kapsamı dışında kalmaktadır.

Nesneye yönelik sistemler için 1990'larda birçok metrik önerilmiş, bunlardan en fazla kullanılanları Chidamber-Kemerer (CK) metrik grubu olmuştur. Birçok araştırmacı ve araç geliştiricisi bu metrikleri kullanmıştır. Bu tez çalışmasında CK metrik kümesinden yararlanıldığı için bu bölümde bu metrik kümesi açıklanacaktır. PROMISE içerisinde bu metrikler sadece KC1 veri kümesi için mevcuttur.

- *WMC (Weighted Method per Class):* WMC, bir sınıftaki metot sayısıdır.
- *DIT (Depth of Inheritance Tree):* DIT, bir sınıftan kalıtım ağacındaki kök elemana olan en uzak yolun mesafesidir.
- *RFC (Response for a Class):* Bir sınıf içerisinde yer alan metot sayısı ile kalıtımdan gelen metot sayısı toplanarak hesaplanır.
- *NOC (Number of Children):* Sınıfın çocuklarının sayısı ya da bir sınıftan türeyen sınıfların sayısı olarak tanımlanabilir.
- *CBO (Coupling between Object Classes):* A sınıfı B sınıfının özelliklerini (attribute) veya operasyonlarını kullanıyorsa, A sınıfının B sınıfına bağlı (coupled) olduğu söylenir. CBO, bir sınıfın kalıtım dışında bağlı olduğu farklı sınıfların sayısıdır.
- *LCOM (Lack of Cohesion in Methods):* Her özellik (attribute) için o özelliği kullanan metotların yüzdesi belirlenip ortalaması alınır ve bu değer 100'den çıkarılır. Elde edilen değer, LCOM değeridir. Sınıf içi kohezyon yüksek olması gerekirken, sınıflar arası bağlılık (coupling) düşük olmalıdır.

Bu 6 metriği karmaşıklık açısından ele alırsak 3 gruba ayırmak mümkündür. Bu gruplar aşağıda verilmektedir:

- *Kalıtım Karmaşıklığı (inheritance complexity):* DIT ve NOC metrikleri.
- *Bağlılık Karmaşıklığı (coupling complexity):* RFC ve CBO metrikleri.
- *Sınıf İç Karmaşıklığı (class internal complexity):* WMC ve LCOM metrikleri.

Bu CK metriklerinin yanı sıra, PERCENT_PUB_DATA, ACCESS_TO_PUB_DATA, DEP_ON_CHILD, FAN_IN metrikleri sınıf seviyesinde metrikler olarak çalışmalarda kullanılmıştır. PERCENT_PUB_DATA, bir sınıf için public veya protected özelliklerin yüzdesi olarak hesaplanır. ACCESS_TO_PUB_DATA bir sınıf için, public veya protected özelliklere erişim sayısı olarak belirlenmektedir. DEP_ON_CHILD, bir sınıfın çocuğuna bağlı olup olmadığını gösterir. FAN_IN, daha üst modüllerden gelen çağrı sayısıdır.

Rosenberg ve arkadaşları (1999); 6 CK metriği, çevrimsel karmaşıklık, satır sayısı ve yorum yüzdesi bilgileri ile güvenilirlik değerlendirilmesi yapılabileceğini önermiştir. Briand ve arkadaşları (1999b); lojistik regresyon yöntemini kullanarak CBO, RFC ve LCOM metriklerinin sınıfların hata eğilimini bulmada kullanabileceğini göstermişlerdir. Zhou ve Leung (2006), DIT dışındaki CK metriklerinin ve kod satır sayısının hata eğilimliliği belirlemede önemli olduğunu tespit etmişlerdir. Yu ve arkadaşları (2002), DIT dışındaki metriklerin önemini vurgulamışlardır. Gyimoth ve arkadaşları (Gyimoth vd., 2005), NOC metriğinin kusur kestirimi için önemsiz olduğu sonucuna varmışlardır. Tang ve arkadaşları (1999), DIT, NOC, CBO metriklerinin önemsiz olduğunu raporlamışlardır. Yapılan çalışmaların çoğunda lojistik regresyon kullanılmış ve WMC, RFC, CBO metriklerinin önemli olduğu saptanmıştır.

Bu tez çalışması sırasında; metot ve sınıf seviyesindeki metriklerden yararlanılmıştır. Ayrıca, bu metriklerin farklı kombinasyonları denenerek en optimum metrik kümesinin belirlenebilmesi için çalışmalar gerçekleştirilmiştir.

5.4 Değerlendirme Kriterleri

Hatalı modüller, tüm yazılım modüllerinin çok küçük bir bölümünü oluşturmaktadır. Bu nedenle, hata kestirim modelleri için kullanılan veri kümelerinin çoğu durumda ayrık ya da dengesiz veri kümeleri olduğu söylenebilir. Bu tür veri kümelerinde, modelleri kıyaslarken doğruluk (accuracy) değerini kullanmak mümkün değildir. Örneğin; veri kümesinde %10 kusurlu ve %90 kusursuz verilerin olduğu durumu ele alalım. Bu durumda, tüm modülleri kusurlu değil şeklinde kestiren bir modelin doğruluk oranı %90 olacak ve bu bilgi uygulayıcıları yanıltabilecektir. Son dönemlerde araştırmacılar bu tür veri kümeleri için farklı

değerlendirme kriterleri önermişlerdir. Örneğin; Alıcı Çalışma Karakteristikleri (AÇK) (Receiver Operating Characteristics-ROC) eğrisi altında kalan alan 1 değerine ne kadar yakınsa, model o kadar doğru çalışmaktadır. Bir diğer parametre, F-ölçüm değeridir ve araştırmacılar bunu problemlerde sıkça kullanmaya başlamıştır.

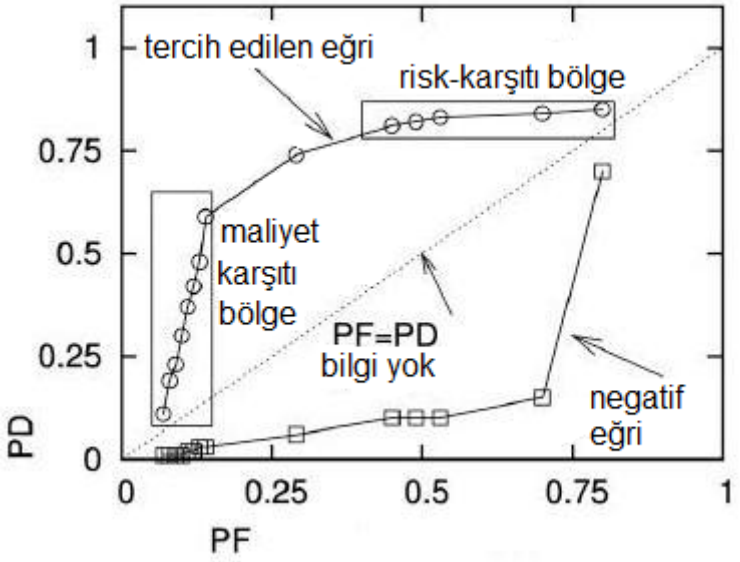
Bu bölümde kusur kestirimi problemlerinde sıkça kullanılan değerlendirme kriterleri açıklanarak bu tez çalışmasında tercih edilen değerlendirme kriterleri ortaya konulacaktır.

5.4.1 ROC Eğrisi Kullanılarak Gerçekleştirilen Değerlendirmeler

"Yazılım eksiklik kestiricileri (defect predictors), bir yazılım modülünün eksiklik eğilimli olduğunu gösteren bir işaret (signal) arayışı peşindedir" (Menzies vd., 2007a).

Farklı kestiricilerin performans değerlendirilmesi için ROC eğrileri kullanılabilmekte ve bu analiz yöntemi "işaret tespit teorisi" (signal detection theory) (Heeger, 1998) tarafından açıklanmaktadır. ROC eğrisi, 2 boyutlu düzlemde x ekseninde yanlış alarm olasılığını (probability of false alarm-PF) gösterirken, y ekseninde tespit olasılığını (probability of detection) gösterir. Bu eğri, tanım olarak (0,0) ve (1,1) noktalarından geçmektedir (Menzies vd., 2007a). Teorik olarak kusurlar için hiç bir zaman işaret üretmeyen kestirici yanlış alarm hiç vermeyecek ve (0, 0) noktasında bulunacaktır. Benzer şekilde daima alarm veren bir kestirici daima yanlış alarm verecek ve (1,1) noktasından geçecektir.

Şekil 5.10'da ROC eğrisinin bazı önemli bölgeleri gösterilmektedir.



Şekil 5.10 ROC eğrisinin bölgeleri (Menzies vd., 2007a)

Şekil 5.10 üzerinde 3 farklı yörünge (trajectory) yer almaktadır (Menzies vd., 2007a):

- (0,0) ve (1,1) noktalarından geçen düz çizgi kestirim konusunda ek bir bilgi sunmamaktadır. Bu eğri altında kalan alan 0.5 olacağı için, rastgele tahminden (random guess) bir farkının olmadığı ifade edilir. Şekil 6.10'da, PF=PD "bilgi yok" şeklinde gösterilmektedir.
- Negatif eğri altında kalan alan 0.5'den küçük olacağı için böyle bir kestiricinin performansı kabul edilebilir değildir. Şekil 6.10'da böyle bir eğri örnek olarak verilmiştir.
- ROC eğrisinde ideal konum, (pf=0, pd=1) noktasıdır. Bu nokta, tüm hataların saptanabildiği ve hiç yanlış kestirim yapılmayan nokta olup "tercih edilen eğri" (preferred curve) bu noktaya doğru kıvrılır.

ROC eğrilerinin bir faydası da maliyet-fayda ödünleşimlerini (cost-benefit trade-offs) yapmaya izin vermeleridir (Menzies vd., 2003). Maliyet-karşıtı bölgeye (cost-adverse region) düşen kestiriciler düşük yanlış alarm değerine sahiptir ve doğrulama&geçerleme bütçesi sınırlı olduğu zaman en uygun kestiricilerin olduğu alandır. Risk-karşıtı bölgede ise bir işaretin tespit olasılığı (probability of detection) yüksek olmasına rağmen yüksek yanlış alarmdan dolayı maliyet söz konusudur (Menzies vd., 2003). Ancak görev kritik uygulamalar için yanlış alarmın maliyeti, sistem arızası oluşması durumunda oluşacak maliyetten düşük olacağı için bu tür uygulamalar dikkate alındığında bu bölgedeki kestiricilerin daha uygun olacağı ifade edilebilir. Kusur kestirimi açısından konuya bakacak olursak, risk-karşıtı bölgede çalışmak daha fazla test yapılmasına neden olacak ancak daha fazla kusur doğru olarak kestirilmiş olacaktır.

5.4.1.1 PD, PF ve Denge Parametreleri

Makine öğrenmesi algoritmaları için model geçerlenmesi, verinin modele dönüştürüldüğünden ve modelin gözlemlenen sistemi yeterli doğrulukta ortaya koyduğundan emin olunabilmesi demektir. Bu amaçla, farklı yöntemler kullanılmakta olup bu yöntemlerden en fazla uygulanılanı "N-katlı çapraz geçerleme" (N-fold cross-validation) tekniğidir. Bu teknik veri kümesini rastgele N adet eşit sayıda örnek içeren parçaya ayırır. Her bir parça için, (N-1) bölümle eğitim gerçekleştirilip, kalan bölümle test yapılır. Bazı araştırmacılar, N-katlı çapraz geçerlemenin M kez tekrarlanmasını önermiştir. Bu sayede, sıralamadan kaynaklı varyanslar oluşmayacak ve istatistiksel olarak daha anlamlı değerler elde edilecektir. Örneğin;

"araya ekleyerek sıralama" (insertion sort) algoritmasında girdiler ters sırada diziliyse bu algoritma en yavaş performansı sunmaktadır (Menzies vd., 2007a).

Bu testlerin ardından, Çizelge 5.5'de verilen hata matrisi (confusion matrix) elde edilmektedir. Bu matrisin, sütunlarında kestirim bilgileri verilirken satırlarında gerçekteki bilgiler sunulmaktadır. Hatalı modüller, EVET ile gösterilirken hatalı olmayan modüller HAYIR ile ifade edilmiştir. Dolayısıyla, matrisin diyagonalindeki elemanlar (TN ve TP) doğru yapılan kestirimleri gösterirken, diğer elemanlar (FN ve FP) kestirimlerdeki hataları ortaya koyar. Örneğin; bir test sonucunda gerçekte hatalı olmayan bir modüle hatalı deniyorsa (FP), bu test sonucu B alanına eklenmektedir. Gerçekleştirilen M*N adet testin ardından, Çizelge 5.5'deki A, B, C, D değerleri elde edilmektedir.

Çizelge 5.5 Hata matrisi

	HAYIR (Kestirim)	**EVET (Kestirim)**
HAYIR (Gerçek)	Doğru Negatif (True Negative-TN) A	Yanlış Pozitif (False Positive-FP) B
EVET (Gerçek)	Yanlış Negatif (False Negative-FN) C	Doğru Pozitif (True Positive-TP) D

PD, PF ve doğruluk (accuracy) parametrelerinin hesaplanmasını sağlayan eşitlikler aşağıda verilmektedir.

$$\text{tespit olasılığı} = \text{pd} = \text{çağrı (recall)} = \frac{D}{B+D} \tag{5.15}$$

$$\text{yanlış alarm olasılığı} = \text{pf} = \frac{C}{A+C} \tag{5.16}$$

$$\text{doğruluk} = \frac{\text{A} + \text{D}}{(\text{A} + \text{B} + \text{C} + \text{D})} \tag{5.16}$$

Denge parametresi ise (0, 1) ve (pf, pd) noktaları arasındaki Öklid uzaklığı olarak tanımlanmaktadır. Kusur kestiricilerde; pd, doğruluk ve denge değerleri maksimize edilmeye çalışılırken pf değeri minimize edilmeye çalışılmaktadır. Menzies ve arkadaşları (2007a), en iyi kusur kestiricilerinin ortalama %71 pd değerine ve %25 pf değerine sahip olduğunu

raporlamışlardır. Bu çalışmalarında değerlendirme kriteri olarak; pd, pf ve denge kriterlerini kullanmışlardır.

5.4.1.2 ROC Eğrisi Altındaki Alan (area under curve-AUC)

Ayrık veri kümelerinde, makine öğrenmesi algoritmalarının performansının değerlendirilmesi için sıkça kullanılan bir diğer parametre de "ROC eğrisi altında kalan alan" (Area Under Curve-AUC) değeridir. Bradley (1997), çeşitli makine öğrenmesi algoritmalarının performansını kıyaslamak için AUC değerini kullanmış ve AUC'un doğruluk parametresine göre daha iyi özelliklere sahip olduğunu ifade etmiştir. Ling ve arkadaşları (2003), sınıflandırma sistemlerini değerlendirmede AUC parametresini önermiş ve AUC'un, dengeli ve dengesiz veri kümelerinde, doğruluk parametresinden istatistiksel olarak daha uygun olduğunu göstermişlerdir.

Van Hulse ve arkadaşları (2007), 11 öğrenme algoritmasını ve 35 veri kümesini kullanarak ayrık veri kümeleri üzerinde sınıflayıcılarla çalışacak olan uygulayıcılara rehber sağlamayı hedeflemişlerdir. Bu çalışmalarında, sınıflandırma algoritmalarının performansını ölçmek için AUC değerini, Kolmogorov-Smirnov istatistiğini (K/S) (Hand, 2005), geometrik ortalamayı, F-ölçümü, doğruluğu ve doğru pozitif oranı (true positive rate-TPR) parametrelerini kullanmışlardır. Çalışmalarında ilk 2 parametrenin (AUC ve K/S), sınıflayıcının pozitif ve negatif sınıflar arasında ayrım yapmadaki genel yeteneğini ölçtüğünü ifade etmişler ve tablolarda algoritmaların AUC değerlerini göstermişlerdir.

Li ve arkadaşları (2007), Chawla ve Karakoulas (2005) ayrık veri kümeleri üzerinde gerçekleştirdikleri çalışmada değerlendirme kriteri olarak AUC değerini kullanmıştır. 11. Pasifik-Asya Bilgi Keşfi ve Veri Madenciliği konferansında (11th Pacific-Asia Conference on Knowledge Discovery and Data Mining-PAKDD2007) düzenlenen yarışmada, bir ayrık veri kümesi için AUC değerine göre değerlendirmeler yapılmış, %70.01 AUC değerini sağlayan model birincilik ödülünü kazanmıştır. Bu tez çalışmasında, AUC değeri kullanılarak performans değerlendirmeleri gerçekleştirilmiştir.

5.4.1.3 G-ortalama1, G-ortalama2 ve F-ölçüm Parametreleri

Bazı araştırmacılar, ayrık veri kümelerinde performans değerlendirmesi için G-ortalama1, G-ortalama2 ve F-ölçüm değerlerini kullanmaktadır. Bu değerleri hesaplamak için aşağıdaki eşitlikler ve Çizelge 5.5'deki değerler kullanılmaktadır.

$$\text{Hassasiyet (precision)} = \frac{TP}{TP + FP} \tag{5.17}$$

$$\text{Tespit olasılığı (PD)} = \frac{TP}{TP + FN} \tag{5.18}$$

$$\text{Doğru negatif oran (TNR)} = \frac{TN}{TN + FP} \tag{5.19}$$

$$\text{G-ortalama1 (G-mean1)} = \sqrt{hassasiyet * PD} \tag{5.20}$$

$$\text{G-ortalama2 (G-mean2)} = \sqrt{PD * TNR} \tag{5.21}$$

$$\text{F-ölçüm (F-measure)} = \frac{2\ (\text{PD} * \text{Hassasiyet})}{\text{PD} + \text{Hassasiyet}} \tag{5.22}$$

Ma ve arkadaşları (2006), farklı modelleri kıyaslarken G-ortalama1, G-ortalama2 ve F-ölçüm değerlerini kullanmışlardır. Bu parametrelerin her birisi için en yüksek performansı veren 3 sınıflayıcı işaretlenmiş ve her 3 parametrede de ilk 3'e giren sınıflayıcı belirlenmiştir. Yaptıkları bu çalışmaya göre, "Dengeli Rastgele Orman" (balanced random forests) algoritması diğer algoritmalardan çok daha yüksek performans vermektedir. Ayrıca; Boosting, Rule Set, Single Tree sınıflayıcıları literatürde bu tür problemlerde kullanılmış olsa da, diğer algoritmalara kıyasla iyi sonuç vermediği sonucuna varılmıştır.

Koru ve Liu (2005b) yaptıkları çalışmada, F-ölçüm parametresine göre kusur kestirimi için sınıflayıcıların performanslarını değerlendirmişlerdir. Bu tez çalışması sırasında, G-ortalama1, G-ortalama2, F-ölçüm değerleri performans değerlendirme amaçlı farklı aşamalarda kullanılmıştır.

5.4.1.4 Duyarlık, Kesinlik ve J katsayısı

El-Emam ve arkadaşları (1999), yazılım mühendisliği içerisinde ikili sınıflayıcıların doğruluğunu ölçmek üzere J parametresinin kullanımını önermişlerdir. J parametresi Youden tarafından (1950) ilk kez tıp alanında kullanılmak üzere ortaya konulmuştur.

Duyarlık (sensitivity) ve kesinlik (specificity) parametreleri kullanılarak J katsayısı hesaplanmaktadır ve El-Emam ve arkadaşları (2001b) tarafından kıyaslama için kullanılmıştır. Bu değerleri hesaplamak için aşağıdaki eşitlikler ve Çizelge 5.5'deki değerler kullanılmaktadır.

$$\text{Duyarlık} = \frac{D}{C+D} \tag{5.23}$$

$$\text{Kesinlik} = \frac{A}{A+B} \tag{5.24}$$

$$\text{J} = \text{duyarlık} + \text{kesinlik} - 1 \tag{5.25}$$

Duyarlık, gerçekte hatalı olan modüllerin doğru olarak kestirilme oranını ortaya koyarken kesinlik, gerçekte hatalı olmayan modüllerin doğru olarak kestirilme oranını ifade etmektedir.

5.4.1.5 Tip-I hatası, Tip-II Hatası ve Yanlış Sınıflandırma Oranı Parametreleri

Bazı araştırmacılar tarafından Tip-I ve Tip-II hatası oranları, kusur kestirim modellerinin performansını değerlendirmek üzerine kullanılmıştır (Yuan vd., 2000; Guo ve Lyu, 2000; Seliya ve Khoshgoftaar, 2007a). Bu iki oranı dikkate alan diğer bir parametre de "tam yanlış sınıflandırma oranı" (overall misclassification rate) parametresidir. Bu değerleri hesaplamak için aşağıdaki eşitlikler ve Çizelge 5.5'deki değerler kullanılmaktadır.

$$\text{Tip-I hatası} = \frac{C}{A+B+C+D} \tag{5.26}$$

$$\text{Tip-II hatası} = \frac{B}{A+B+C+D} \tag{5.27}$$

$$\text{Tam yanlış sınıflandırma oranı} = \frac{C+B}{A+B+C+D} \tag{5.28}$$

5.4.1.6 Hatasızlık ve Tamlık Parametreleri

Bazı araştırmacılar, hatasızlık (correctness) ve tamlık (completeness) parametrelerini değerlendirme kriteri olarak kullanmaktadır. Doğru şekilde kestirilen kusurlu modüllerin yüzdesi hatasızlık ve model tarafından kestirilen yüksek riskli modüllerin yüzdesi tamlık olarak ifade edilmektedir. Bu değerleri hesaplamak için aşağıdaki eşitlikler ve Çizelge 5.5'deki değerler kullanılmaktadır.

$$\text{Hatasızlık} = \frac{D}{B+D} \tag{5.29}$$

$$\text{Tamlık} = \frac{D}{C+D} \tag{5.30}$$

Bu değerlendirme parametreleri literatürdeki kusur kestirim çalışmalarında kullanılmıştır

(Briand vd., 1993; Zhou ve Leung, 2006; Denaro vd., 2003b; Gyimothy vd., 2005; Mahaweerawat vd., 2002).

5.4.2 Kusur Sayısını Kestirmede Kullanılan Değerlendirme Kriterleri

Modülün kusurlu olup olmadığını kestirmek yerine kusur sayısını kestirerek bu sınıflandırmayı gerçekleştirmek mümkündür. Mevcut test kaynakları dikkate alınarak, kusur sayısına göre modüller büyükten küçüğe sıralandıktan sonra daha fazla test edilmesi gereken modüller saptanmaktadır.

Kusur sayısının kestirildiği çalışmalar, "yazılım kalite kestirimi" olarak literatürde açıklanırken sınıflandırmanın yapıldığı çalışmalar "yazılım kalite sınıflandırma" şeklinde ifade edilmektedir.

5.4.2.1 Ortalama Mutlak Hata ve Ortalama Bağıl Hata Parametreleri

Ortalama mutlak hata (average absolute error-AAE) ve ortalama bağıl hata (average relative error-ARE) parametreleri birçok araştırmacı tarafından kusur kestirimi çalışmalarında değerlendirilme kriteri olarak kullanılmıştır (Khoshgoftaar vd., 2006; Khoshgoftaar ve Seliya, 2002d; Khoshgoftaar vd., 2002a; Khoshgoftaar vd., 2006; Gao ve Khoshgoftaar, 2007). Bu değerleri hesaplamak için aşağıdaki eşitlikler kullanılmaktadır. Eşitliklerde yer alan y_i, gerçek kusur sayısı değerini, y_j kestirilen kusur sayısı değerini ve n veri kümesindeki toplam modül sayısını göstermektedir. Tez çalışmasında, kusur kestirimi problemi sınıflandırma açısından ele alındığı için bu parametrelerden yararlanılmamıştır.

$$\text{AAA} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - y_j\right| \tag{5.31}$$

$$\text{ARE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - y_j}{y_i + 1}\right| \tag{5.32}$$

ARE denkleminde kullanılan 1 değeri, sıfıra bölmeyi engellemek için eklenmiştir.

5.4.2.2 R^2 Parametresi

"R^2, kestirilen ve gerçek kusur sayıları arasındaki korelasyon gücünü ölçer" (Tomaszewski vd., 2005). R^2 parametresi yerine, çoklu determinasyon katsayısı (coefficient of multiple determination) ifadesi de kullanılabilmektedir. Kusur sayılarının kestirildiği çalışmalarda bu parametre sıkça kullanılmaktadır. Tez çalışmasında, kusur kestirimi problemi sınıflandırma açısından ele alındığı için bu parametreden yararlanılmamıştır.

Birçok araştırmacı bu parametreyi kusur kestirim çalışmalarında kullanmıştır (Denaro vd., 2003b; Thwin ve Quah, 2003; Denaro, 2000; Tomaszewski vd., 2005; Nikora ve Munson, 2006; Binkley vd., 2007).

R^2 değeri 1'e yaklaştıkça modelin önemi artmaktadır. R^2 değerini hesaplamak için aşağıdaki eşitlik kullanılmaktadır. Eşitliklerde yer alan y_i gerçek kusur sayısı değerini, $\hat{y}_i$ kestirilen kusur sayısı değerini ve $\bar{y}$ ortalama kusur sayısını göstermektedir.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{\mathbf{y}})^2} \tag{5.33}$$

5.5 Veri Kümeleri

Araştırma kapsamında yürüttüğümüz tüm çalışmalarda, NASA'da geliştirilmiş olan projelerin metrikleri ve hata bilgileri kullanılmıştır. Bu projelerdeki hata bilgilerini ve metrikleri kullanmamızın sebebi; literatürdeki diğer modellerle kıyaslama imkanı oluşturması, erişiminin tüm araştırmacılara ücretsiz sunulması ve oldukça detaylı şekilde bilgilerin oluşturulmuş olmasıdır.

NASA, yazılım eksiklik kestirimi konusunun önemini belirlemiş ve 1993 yılında "Bağımsız Doğrulama ve Geçerleme Kolaylığı" (Independent Verification and Validation Facility-IV&V) bölümünü kurmuştur. NASA'nın bu bölümü, görev kritik uygulamalarda yüksek güvenliği ve maliyet etkinliğini sağlamak üzere Metrik Veri Programı Veri Havuzu'nu (Metrics Data Program Data Repository) oluşturmuştur. Bu programda oluşturulan metrikler ve diğer bilgiler, daha kolay erişim için PROMISE havuzuna aktarılmıştır. Tez çalışması süresince ağırlıklı olarak aşağıda verilen veri kümeleri kullanılmıştır:

- JM1: C dili ile geliştirilmiş, gerçek zamanlı bir projedir. 315,000 kod satırına ve 10,885 fonksiyona sahiptir. 8 yıllık hata bilgisi içermekte ve modüllerde hata raporlandıkça düzeltmeler yapılmıştır. Kullanılan en kapsamlı veri kümelerinden birisidir. Veri kümesindeki modüllerin %19'u test ya da sahada hataya neden olmuştur.

- KC1: C++ dili ile geliştirilmiştir ve 2109 modülden oluşmaktadır. Veri kümesindeki modüllerin %15'i test ya da sahada hataya neden olmuştur. 750,000 satırdan oluşmakta ve 5 yıllık hata verisi içermektedir.

- PC1: C dili ile geliştirilmiş, uçuş kontrol yazılımıdır. 1109 modülü vardır ve modüllerin %7'si test ya da sahada hataya neden olmuştur. 40,000 satırdan oluşmakta ve 3 yıllık hata bilgisi içermektedir.
- KC2: C++ ile geliştirilmiş veri işleme projesidir. 523 modül yer almakta ve modüllerin %21'i test ya da sahada hataya neden olmuştur.
- CM1: C dili ile geliştirilmiş, 20,000 kod satırından oluşan, 2 yıllık hata bilgisi mevcut olan, 523 modülden oluşan ve modüllerinin %10'u hatalı olan bir projedir.

Bu veri kümeleri, PROMISE havuzu içerisinde yer almaktadır. Ancak içerdiği metrikler metot seviyesindeki metriklerdir. Sadece KC1 veri kümesi için sınıf seviyesinde metriklere erişilebilmektedir. Bu nedenle, nesneye yönelik metrikleri kullanarak araştırma yapmak isteyenler sadece KC1 veri kümesini kullanabilmekte ve modellerini bu veri kümesi üzerinden analiz edebilmektedirler.

6. EĞİTİCİLİ YAZILIM KUSUR KESTİRİMİ

Bu bölümde, eğiticili öğrenme kapsamında gerçekleşirilen kusur kestirim çalışmaları ve oluşturulan modeller deneysel çalışmaları ile birlikte açıklanmaktadır. İlk bölümde metot seviyesinde metrikler kullanılarak geliştirilen model, ikinci bölümde nesneye yönelik metriklerle oluşturulan kestirim modeli ve son bölümde, birçok algoritmanın farklı metrik grupları ile birlikte kullanıldığı durumda elde edilen performanslar, önerilen yöntemler açıklanmaktadır.

6.1 Metot Seviyesinde Metriklerle Yazılım Kusur Kestirimi

Bu bölümde, metot seviyesindeki metriklerle yapılan çalışmalar, geliştirilen modeller ve deneysel verilere dayalı elde edilen önemli çıkarımlar sunulmaktadır.

6.1.1 Özet

Bu çalışmada, metot seviyesindeki 21 adet metrik ve AIRS algoritması kullanılarak farklı modeller üzerinde çalışılmıştır. Özellik azaltma tekniği olarak Korelasyon Tabanlı Özellik Seçimi (correlation based feature selection) ve arama metotu (search method) olarak Greedy Stepwise seçildiği zaman ortaya çıkan metrikler, AIRS algoritmasında kullanıldığı durumda oldukça başarılı kusur kestirim sonuçlarına ulaşılmıştır. Bu özellik azaltma tekniğinin yanı sıra, Temel Bileşen Analizi (Principal Component Analysis) ve Uyumluluk Tabanlı Özellik Seçimi (consistency based feature selection) teknikleri de incelenmiştir. Ancak en iyi performans, korelasyon tabanlı özellik seçimi ile elde edilmiştir. JM1 veri kümesindeki analizlere göre, G-ortalama1 ve F-ölçüm değerleri açısından önerilen model, diğer algoritmalardan çok daha yüksek performanslı sonuçlar vermiştir. Ma ve arkadaşları (2006), yaptıkları çalışmada birçok algoritmayı kıyaslamış ve Dengeli Rastgele Orman algoritmasının en iyi sonucu sunduğunu raporlamıştır. Yukarıda açıklanan modeli kullandığımız durumda, Dengeli Rastgele Orman algoritması ile elde edilen G-ortalama1, F-ölçüm değerlerinden daha yüksek değerler elde edilmiştir. Tüm değerlendirme kriterlerine göre diğer algoritmalardan daha iyi performans sağlanmıştır. G-ortalama2 değerinde, Dengeli Rastgele Orman daha yüksek sonuç vermiştir. Bu parametreye göre önerdiğimiz model sonucunda kaybımız biraz daha fazla test yapmak olacaktır. Ancak hatalı modüller, önerdiğimiz modelde daha yüksek doğrulukla yakalanmaktadır.

6.1.2 Kullanılan Metrikler, Veri Kümeleri, Değerlendirme Kriterleri

Bu çalışmada kullanılan metrikler Çizelge 6.1'de verilmektedir.

Çizelge 6.1 Metot seviyesindeki metrikler

Özellik	Bilgi
loc	McCabe kod satır sayısı
v(g)	McCabe çevrimsel karmaşıklık
ev(g)	McCabe temel karmaşıklık
iv(g)	McCabe tasarım karmaşıklığı
n	Halstead toplam operatör + toplam operand
v	Halstead hacim
l	Halstead program uzunluğu
d	Halstead zorluk
i	Halstead zekâ
e	Halstead çaba
b	Halstead teslim edilen hata (bug)
t	Halstead zaman kestiricisi
lOCode	Halstead kod satırı sayısı
lOComment	Halstead yorum satırı sayısı
lOBlank	Halstead boş satır sayısı
lOCodeAndComment	Yorum ve kod satır sayısı
uniq_Op	Eşsiz (unique) operatör sayısı
uniq_Opnd	Eşsiz (unique) operand sayısı
total_Op	Toplam operatör
total_Opnd	Toplam operand
branchCount	Akış grafının dal sayısı

Veri kümesi olarak; JM1, KC1, PC1, KC2 ve CM1 veri kümeleri kullanılmıştır. Veri kümesi içindeki metrikleri ve kusur bilgilerini kullanarak modelimizi oluşturduktan sonra, 10 katlı çapraz-geçerleme tekniği ile test ettik. Çapraz geçerlemeden sonra elde edilen hata matrisini kullanarak, performans değerlendirme kriterlerini hesapladık. Değerlendirme kriteri olarak; G-ortalama1, G-ortalama2 ve F-ölçüm değerleri kullanılmıştır.

6.1.3 Deneysel Sonuçlar

Yapay Bağışıklık Sisteminden esinlenen 4 farklı model, 5 veri kümesinde yazılım kusur kestirimi için test edilmiştir. 4 farklı modele ilişkin bilgiler aşağıda sunulmaktadır:

- İlk model: AIRS1 algoritması kullanılmıştır.
- İkinci model: Korelasyon tabanlı özellik seçim tekniği (cfs) ve Greedy Stepwise arama metodu kullanılarak uygun özellikler seçilmiş, seçilen özellikler üzerinde AIRS1 algoritması çalıştırılmıştır.
- Üçüncü model: Özellik seçimi için temel bileşen analizi (pca) metodu kullanılmış ve seçilen özellikler AIRS1 algoritmasında kullanılmıştır.
- Dördüncü model: Uyumluluk tabanlı özellik seçimi (cse) metodu ve Greedy Stepwise arama algoritması ile uygun özellikler seçilmiş ve seçilen özellikler AIRS1 algoritmasında kullanılmıştır.

Eğitim ve test aşamalarından sonra, G-ortalama1, G-ortalama2, F-ölçüm değerleri hesaplanmıştır. Elde edilen değerler, Ma ve arkadaşlarının (2006) elde ettiği sonuçlarla kıyaslanmıştır. Çizelge 6.2, 6.3, 6.4, 6.5, 6.6'da Ma ve arkadaşlarının (2006) deneylerine ilişkin sonuçlar verilmektedir. Bu sonuçlar ile önerdiğimiz modellerle elde edilen sonuçlar, ileriki bölümlerde karşılaştırılacaktır.

JM1 Projesi: En büyük veri kümesi olan JM1 üzerinde, AIRS1 algoritması ile birlikte korelasyon tabanlı özellik seçim (cfs) algoritmasının kullanıldığı model (cfs-AIRS1), oldukça başarılı sonuçlar üretmiştir. Öncelikle, cfs ile özellikler seçilmiş ve seçilen özellikler üzerinde AIRS1 algoritması çalıştırılmıştır. Ma ve arkadaşlarının (2006) çalışmasına göre, Rastgele Orman algoritması en yüksek G-ortalama ve F-ölçüm değerlerine sahiptir. cfs-AIRS1 modelinin G-ortalama1 ve F-ölçüm değerleri, bu çalışmada ve Ma ve arkadaşlarının çalışmasında incelenen tüm algoritmalardan daha yüksektir. Bu çalışmaya göre, cfs tekniği AIRS1 algoritmasının performansını arttırmaktadır sonucu çıkarılabilir. Çizelge 6.7'de AIRS1 algoritmasının tek başına kullanılması durumunda elde edilen, G-ortalama ve F-ölçüm değerleri verilmektedir. Bu değerlerden, cfs tekniği sayesinde AIRS1'in tüm performans indikatörlerinin arttığı kolayca görülmektedir. Yukarıda verilen üçüncü model, G-ortalama1 ve F-ölçüm değerlerinin artmasını sağlamaktadır ancak G-ortalama2 değerinin azalmasına neden olmaktadır. Bu nedenle, ikinci modelin bu veri kümesi için daha uygun ve diğer yüksek performanslı modellerle karşılaştırılabilir ölçüde olduğu söylenebilir. İkinci modelimiz, geniş ölçekli yazılım projelerinde kusur kestirimini sağlamak üzere etkin şekilde kullanılabilir ve performans indikatörlerini arttırmak için diğer makine öğrenmesi algoritmalarıyla da birlikte kullanılarak yeni modeller kurgulanabilir.

KC1 Projesi: cfs-AIRS1 modelinin performans sonuçları, RF algoritması ile karşılaştırılabilir düzeydedir. KStar algoritması, cfs-AIRS1 algoritmasından daha iyi sonuç üretmektedir ancak modelimiz diğer algoritmalardan daha başarılıdır. Elde ettiğimiz performans sonuçları ile RF performans sonuçları arasında çok büyük fark yoktur. Modelimizin G-ortalama1 değeri, RF algoritmasının G-ortalama1 değerinden daha yüksektir. F-ölçüm değerimiz RF algoritmasının F-ölçüm değerinden daha yüksektir. RF ile önerdiğimiz model arasında performans indikatörleri açısından çok büyük farklar olmadığı için modelimizin bu veri kümesinde de başarılı çalıştığını söyleyebiliriz ancak en iyi performans bu veri kümesinde RF ile sağlanmaktadır. Temel bileşen analizi (pca) tekniği kullanmak, G-ortalama2 değerinin azalmasına ve diğer indikatörlerde küçük artışlara neden olmuştur. Bu veri kümesinde G-ortalama2 değerini arttırmak istediğimizden, pca tekniği istenen özellikte değildir. Uyumluluk tabanlı özellik seçim (cse) tekniği ise performansta çok küçük artış sağlamıştır. Çizelge 6.8'de bu veri kümesinde elde edilen performans indikatörleri gösterilmektedir. Bu veri kümesinde elde edilen sonuçlar, AIRS algoritmasının gücünün kusur kestiriminde incelenmeye devam edilmesini teşvik etmektedir.

PC1 Projesi: cfs-AIRS1 modelinin performans sonuçları birçok makine öğrenmesi algoritmasından daha yüksektir ancak RF algoritması en yüksek indikatörlere sahiptir. cfs-AIRS1 modeli AIRS1 algoritmasından ve pca-AIRS1 modelinden daha başarılıdır. cse-AIRS1'in F-ölçüm ve G-ortalama1 değerleri, cfs-AIRS1'den daha yüksektir ancak G-ortalama2 değeri daha düşüktür. Çizelge 6.9'da bu veri kümesine ilişkin gerçekleştirilen deneylerin sonuçları verilmektedir. cfs-AIRS1'in G-ortalama1 değeri, RF(0.9, 0.1) modelinin G-ortalama1 değerinden daha yüksektir ve cfs-AIRS1'in G-ortalama2 değeri RF(0.7, 0.3) modelinin G-ortalama2 değerinden daha yüksektir.

KC2 Projesi: AIRS1 algoritmasının performans sonuçları oldukça yüksektir ve G-ortalama1, F-ölçüm değerleri şimdiye kadar elde edilen en yüksek değerlerdir. AIRS1 modelinin G-ortalama2 değeri RF(0.7, 0.3) modelinden daha yüksektir ancak diğer RF modellerinden biraz daha düşüktür. Bu veri kümesi için en iyi kestirici model AIRS1'dir ve RF algoritmasından daha iyi sonuçlar vermiştir. Çizelge 6.10, bu veri kümesine ilişkin deney sonuçlarını vermektedir. Bu deney, yazılım kusur kestirimi için AIRS1 algoritmasının gücünü bir kez daha ortaya koymaktadır.

CM1 Projesi: cfs-AIRS1'in sonuçları, RF algoritmasından daha iyidir. Modelimizin G-ortalama1 ve F-ölçüm değerleri diğer RF modellerinden daha yüksektir. G-ortalama2 değeri iki RF modelinden daha düşük olmasına rağmen bir RF modelinin G-ortalama2 değerinden

daha yüksektir. Diskriminant analizi bu veri kümesi için en iyi kestirici modeldir. Önerdiğimiz model, birçok kusur kestirim algoritmasından daha iyi sonuç üretmiştir. NASA kalite mühendislerine göre, bu veri kümesi en basit olanıdır. Çizelge 6.11, bu veri kümesinde elde edilen sonuçları sunmaktadır.

Çizelge 6.2 Ma ve arkadaşlarının (2006) JM1 üzerinde elde ettiği performanslar

Girdiler	**PD**	**ACC**	**Prec**	**G-1**	**G-2**	**F**
Logistic	0.654	0.658	0.315	0.454	**0.656**	0.425
Discriminant	0.509	0.736	0.368	0.433	0.634	**0.427**
Tree	0.131	0.811	0.546	0.268	0.357	0.211
RuleSet	0.115	0.810	0.540	0.249	0.335	0.190
Boosting	0.118	0.808	0.515	0.246	0.339	0.192
KernelDensity	0.194	0.810	0.523	0.319	0.431	0.283
NaiveBayes	0.197	0.803	0.477	0.306	0.432	0.279
J48	0.247	0.802	0.476	0.343	0.481	0.325
IBk	0.369	0.761	0.379	0.374	0.562	0.374
IB1	0.376	0.756	0.371	0.373	0.564	0.373
VotedPerceptron	0.604	0.560	0.243	0.383	0.576	0.347
VF1	0.868	0.418	0.232	0.448	0.519	0.366
HyperPipes	1.000	0.195	0.194	0.440	0.046	0.324
ROCKY	0.338	0.752	0.352	0.345	0.536	0.345
RF (0.9,0.1)	0.827	0.557	0.281	**0.482**	0.638	0.419
RF (0.8,0.2)	0.645	0.696	0.346	**0.472**	**0.676**	**0.450**
RF (0.7,0.3)	0.483	0.784	0.445	**0.463**	**0.643**	**0.463**

Çizelge 6.3 Ma ve arkadaşlarının (2006) PC1 üzerinde elde ettiği performanslar

Girdiler	PD	ACC	Prec	G-1	G-2	F
Logistic	0.234	0.885	0.208	0.221	0.467	0.220
Discriminant	0.429	0.849	0.211	0.301	0.615	0.283
Tree	0.221	0.929	0.476	0.324	0.466	0.302
RuleSet	0.177	0.932	0.531	0.307	0.418	0.265
Boosting	0.130	0.940	1.000	0.361	0.361	0.230
J48	0.247	0.934	0.556	0.370	0.493	0.342
IB1	0.416	0.914	0.389	0.402	0.629	**0.402**
KStar	0.273	0.921	0.399	0.330	0.514	0.324
NaiveBayes	0.299	0.887	0.244	0.270	0.528	0.269
VF1	0.883	0.193	0.071	0.251	0.353	0.132
RF (0.9,0.1)	0.740	0.823	0.245	**0.426**	**0.784**	0.368
RF (0.8,0.2)	0.506	0.908	0.379	**0.438**	**0.689**	**0.433**
RF (0.7,0.3)	0.442	0.934	0.531	**0.484**	**0.655**	**0.482**

Çizelge 6.4 Ma ve arkadaşlarının (2006) KC1 üzerinde elde ettiği performanslar

Girdiler	PD	ACC	Prec	G-1	G-2	F
Logistic	0.752	0.711	0.317	0.488	0.727	0.446
Discriminant	0.638	0.790	0.390	0.499	0.722	0.484
Tree	0.193	0.848	0.523	0.318	0.432	0.282
RuleSet	0.187	0.852	0.564	0.325	0.427	0.281
Boosting	0.169	0.862	0.732	0.352	0.409	0.275
KStar	0.509	0.855	0.532	**0.521**	0.684	**0.520**
VF1	0.957	0.195	0.156	0.387	0.231	0.269
RF (0.9,0.1)	0.844	0.711	0.330	**0.528**	**0.761**	0.475
RF (0.8,0.2)	0.700	0.782	0.387	**0.520**	**0.747**	**0.498**
RF (0.7,0.3)	0.561	0.825	0.447	0.501	0.700	**0.498**

Çizelge 6.5 Ma ve arkadaşlarının (2006) KC2 üzerinde elde ettiği performanslar

Girdiler	PD	ACC	Prec	G-1	G-2	F
Logistic	0.849	0.685	0.382	0.569	0.738	0.527
Discriminant	0.632	0.821	0.559	**0.594**	**0.742**	**0.593**
Tree	0.547	0.819	0.564	0.555	0.698	0.555
RuleSet	0.500	0.836	0.630	0.561	0.680	0.557
Boosting	0.434	0.835	0.651	0.531	0.638	0.521
IBk	0.509	0.812	0.548	0.528	0.673	0.528
DecisionStump	0.642	0.808	0.529	0.583	0.739	**0.580**
VotedPerceptron	0.849	0.376	0.228	0.440	0.463	0.360
VF1	0.887	0.586	0.319	0.532	0.671	0.469
ROCKY	0.722	0.727	0.409	0.543	0.725	0.522
RF (0.9,0.1)	0.880	0.723	0.419	**0.607**	**0.774**	0.567
RF (0.8,0.2)	0.806	0.769	0.465	**0.612**	**0.782**	**0.590**
RF (0.7,0.3)	0.620	0.784	0.482	0.547	0.716	0.543

Çizelge 6.6 Ma ve arkadaşlarının (2006) CM1 üzerinde elde ettiği performanslar

Girdiler	PD	ACC	Prec	G-1	G-2	F
Logistic	0.776	0.657	0.192	0.386	0.707	0.308
Discriminant	0.694	0.841	0.346	0.490	0.771	0.462
Tree	0.204	0.906	0.561	0.338	0.448	0.299
RuleSet	0.102	0.882	0.253	0.161	0.314	0.145
Boosting	0.020	0.892	0.145	0.054	0.141	0.035
KStar	0.204	0.859	0.243	0.222	0.436	0.222
NaiveBayes	0.306	0.849	0.267	0.286	0.527	0.285
VF1	0.898	0.339	0.120	0.328	0.450	0.211
RF (0.9,0.1)	0.714	0.651	0.179	0.358	0.678	0.287
RF (0.8,0.2)	0.490	0.805	0.250	0.350	0.641	0.331
RF (0.7,0.3)	0.224	0.855	0.244	0.234	0.456	0.234

Çizelge 6.7 JM1 analizi

Girdiler	PD	ACC	Prec	G-1	G-2	F
RF (0.9,0.1)	0.827	0.557	0.281	0.482	0.638	0.419
RF (0.8,0.2)	0.645	0.696	0.346	0.472	0.676	0.450
AIRS1 (k=3)	0.612	0.717	0.282	0.415	0.571	0.386
cfs-AIRS1 (k=3)	0.599	0.668	0.402	**0.491**	**0.574**	**0.481**
cfs-AIRS1 (k=1)	0.562	0.610	0.450	0.503	0.551	0.500
pca-AIRS1 (k=3)	0.561	0.636	0.391	0.468	0.547	0.461
pca-AIRS1 (k=1)	0.549	0.603	0.434	0.488	0.540	0.484
cse-AIRS1(k=3)	0.602	0.702	0.305	0.428	0.567	0.405
cse-AIRS1(k=1)	0.566	0.643	0.385	0.467	0.550	0.458

Çizelge 6.8 KC1 analizi

Girdiler	PD	ACC	Prec	G-1	G-2	F
AIRS1 (k=3)	0.634	0.746	0.298	0.435	0.586	0.405
cfs-AIRS1 (k=3)	0.692	0.763	0.368	0.504	0.628	0.480
cfs-AIRS1 (k=1)	0.589	0.660	0.426	0.501	0.569	0.494
pca-AIRS1(k=3)	0.563	0.653	0.383	0.464	0.548	0.456
pca-AIRS1(k=1)	0.574	0.634	0.448	0.507	0.561	0.503
cse-AIRS1(k=3)	0.708	0.773	0.374	**0.515**	**0.638**	**0.489**
cse-AIRS1(k=1)	0.647	0.715	0.426	0.525	0.609	0.514

Çizelge 6.9 PC1 analizi

Girdiler	PD	ACC	Prec	G-1	G-2	F
AIRS1 (k=3)	0.739	0.873	0.221	0.404	0.633	0.340
cfs-AIRS1 (k=3)	0.796	0.891	0.234	**0.432**	**0.662**	**0.362**
cfs-AIRS1 (k=1)	0.596	0.784	0.260	0.394	0.560	0.362
pca-AIRS1(k=3)	0.440	0.829	0.091	0.200	0.466	0.151
pca-AIRS1(k=1)	0.596	0.806	0.221	0.363	0.557	0.322
cse-AIRS1(k=3)	0.781	0.888	0.221	0.415	0.653	0.344
cse-AIRS1(k=1)	0.675	0.817	0.299	0.450	0.610	0.414

Çizelge 6.10 KC2 analizi

Girdiler	PD	ACC	Prec	G-1	G-2	F
AIRS1 (k=3)	0.835	0.822	0.570	**0.690**	**0.750**	**0.677**
cfs-AIRS1 (k=3)	0.725	0.747	0.495	0.599	0.668	0.588
cfs-AIRS1 (k=1)	0.597	0.619	0.533	0.564	0.588	0.563
pca-AIRS1(k=3)	0.656	0.672	0.579	0.616	0.639	0.615
pca-AIRS1(k=1)	0.641	0.663	0.551	0.594	0.624	0.593
cse-AIRS1(k=3)	0.751	0.759	0.617	0.680	0.712	0.677
cse-AIRS1(k=1)	0.768	0.774	0.589	0.672	0.715	0.667

Çizelge 6.11 CM1 analizi

Girdiler	PD	ACC	Prec	G-1	G-2	F
AIRS1 (k=3)	0.809	0.809	0.224	0.378	0.581	0.332
cfs-AIRS1 (k=3)	0.720	0.845	0.224	**0.402**	**0.624**	**0.342**
cfs-AIRS1 (k=1)	0.556	0.749	0.245	0.369	0.535	0.340
pca-AIRS1(k=3)	0.526	0.785	0.163	0.293	0.515	0.249
pca-AIRS1(k=1)	0.438	0.729	0.163	0.267	0.461	0.238

6.1.4 Sonuçlar ve Gelecek Çalışmalar

Bu çalışmada, AIRS algoritması istatistiksel tekniklerle birleştirilmiş ve korelasyon tabanlı özellik azaltma tekniğinin AIRS ile birlikte kullanıldığı durumda, JM1 veri kümesi için en iyi performansı sunan RF algoritmasından daha iyi performans elde edilebildiği gözlemlenmiştir.

Geliştirilen model açık veri kümeleri üzerinde sınandığı için, deneylerin yeniden üretilebilirliği ve geçerliliğinin sağlanması mümkündür. Yapay Bağışıklık Sistem paradigması bu problemde olduğu gibi birçok problemde de yazılım mühendislerine yardımcı olabilecek güçtedir. Bu çalışmada elde edilen modelin, PROMISE havuzuna yeni eklenecek veri kümeleri üzerinde denenmesi planlanmaktadır. Ayrıca, diğer yüksek performans sunduğu belirlenen algoritmalarla birlikte kullanılarak daha etkin modellerin geliştirilmesi hedeflenmektedir.

6.2 Nesneye Yönelik Metriklerle Yazılım Kusur Kestirimi

Bu bölümde, nesneye yönelik metriklerle yapılan çalışmalar, geliştirilen modeller ve deneysel verilere dayalı elde edilen önemli çıkarımlar sunulmaktadır.

6.2.1 Özet

Nesneye yönelik analiz ve tasarım için 1990'larda birçok metrik önerilmiştir. Bu metriklerden en fazla kabul göreni ve en fazla uygulananı, Chidamber-Kemerer (CK) metrik grubudur. Kusur kestirimini gerçekleştiren Predictive isimli araç içerisinde sınıf seviyesindeki metrikler, CK metrik grubundaki 6 adet metriktir.

Bu çalışmada; CK metrik kümesi içerisinde yer alan metrikler ve bazı metot seviyesindeki metrikler, Yapay Bağışıklık Tanıma Sistemi (AIRS) algoritması ile birlikte kullanılarak oluşturulan modelin yazılım kusur kestirimindeki performansının saptanması ve incelenen CK metrik kümesinde kusur eğilimliliğini belirlemede en önemli/en önemsiz metriklerin belirlenmesi hedeflenmiştir. Deneylerde kullanılan veri kümesi (KC1) NASA Metrik Veri Programının bir parçası olup sınıf seviyesindeki metrikler PROMISE havuzundan elde edilmiştir. Metrikleri tek tek geçerlemek yerine, çalışmadaki temel motivasyon modelin performansını iyileştirmektir. Çalışma sonucunda, kusur kestirimi ile test sürecini iyileştirebilecek yeni bir yazılım kusur kestirim modeli önerilmiştir. AIRS algoritması tabanlı geliştirilen modelin, bağımsız değişkenleri olarak Halstead, McCabe, CK metrikleri üzerinde çalışılmış ve deneysel çalışmalar neticesinde, CK metrik kümesindeki metrikler ve kod satır sayısı metriğinin AIRS ile birlikte kullanılmasına karar verilmiştir. CK metrikleri ve kod satır sayısı metriğinin bağımsız değişkenler olarak kullanıldığı AIRS tabanlı modelin performansı, literatürde önerilmiş ve sınıf seviyesindeki metrikleri kullanan J48 tabanlı yaklaşımdan daha iyi sonuç vermiştir. Ayrıca, sınıf seviyesindeki metriklerle elde edilen performansların metot seviyesindeki metriklerle elde edilen performanslardan daha yüksek olduğu saptanmıştır. Metot seviyesindeki metriklerle çalışıldığı durumda, kusurlar metot bazında kullanıcıya sunulabilirken sınıf seviyesindeki metrikler kullanıldığı zaman, kusurlu sınıflar model kullanıcısına sunulur. Bu açıdan bakıldığında, sınıf seviyesindeki metriklerin kestirim performansının daha yüksek olması, bu inceleme derinliği ile açıklanabilmektedir. Yakın gelecekte, gerçek zamanlı sistemlerin kendilerini özel ortam koşullarına ve isterlere adapte etmesi gerekebileceğinden, modülleri kusur eğilimli ya da değil şeklinde sınıflandırabilecek gerçek zamanlı kararlılık değerlendirme tekniği (real-time dependability assessment technique) ihtiyacı göz önüne alındığında, geliştirilen modelin bu tekniğin bir parçası olabileceği değerlendirilmektedir.

6.2.2 Giriş

Uçuş kontrol yazılımı gibi gerçek zamanlı görev kritik sistemlerin kararlı olması beklenir ve kararlılık; emniyet (safety), güvenlik (security), güvenilirlik (reliability), erişilebilirlik (availability) özellikleri sağlandığı durumda elde edilir (Laprie, 1992). Kararlı sistemlere ulaşabilmek için; kusurdan sakınma, kusurları ortadan kaldırma, kusur toleransı ve kusur kestirimi yaklaşımları uygulanabilmektedir. Yakın gelecekte, gerçek zamanlı sistemler kendilerini çalışma zamanı isterlerine (run-time requirements) adapte etmesi gerekecek ve dinamik olarak oluşturulacak kodun beklentileri karşılayıp karşılamadığı gerçek zamanlı değerlendirme tekniği ile incelenebilecektir (Challagulla vd., 2005). Kararlılık düşük ise operatör çalışma zamanı konfigürasyonunu değiştirebilecektir. Bir yaklaşım ise kusur kestirim modelini kullanarak kararlılığı değerlendirmektir (Challagulla vd., 2005).

Bu çalışmada, aşağıdaki araştırma soruları (research questions) deneyler öncesi belirlenmiş ve deneyler bu sorulara yanıt vermek üzere tasarlanmıştır.

- ***Araştırma sorusu 1:*** *AIRS tabanlı kusur kestirim modeli kusur eğilimli sınıfları ne tür bir doğrulukla kestirir?*
- ***Araştırma sorusu 2:*** *Nesneye yönelik sistemler için hangi CK metrikleri kusurların oluşması ile ilişkilidir?*
- ***Araştırma sorusu 3:*** *Kusur kestirimi performansı açısından sınıf seviyesindeki verilerle oluşturacak model mi yoksa metot seviyesindeki metriklerle oluşturulacak model mi daha başarılı olacaktır?*

Bu çalışma sayesinde, CK metrikleri Yapay Bağışıklık Sistem tabanlı bir algoritmada ilk kez kullanılarak AIRS tabanlı kusur kestirim modelinin performansı iyileştirilmiştir. Ayrıca, korelasyon tabanlı özellik azaltma tekniği gibi farklı istatistiksel tekniklerle metrikleri seçmek ve modelin yeni performansını değerlendirmek mümkün olmuştur.

6.2.3 Metrikler ve Veri Kümesi

CK metrik kümesi 1994 yılında önerilmiş ve o yıldan bu yana, birçok yazılım araç üreticisi, araştırmacı çalışmalarında bu metrikleri kullanmıştır. Bu çalışma, 6 adet CK metriğini kullanmaktadır. Bu metriklere ek olarak, 4 adet sınıf seviyesindeki metrikten daha faydalanmak istedik. Koru ve Liu (2005a), bu metrikleri KC1 projesi için hesaplamış ve PROMISE havuzuna oluşturduğu verileri sunmuştur. Bu 4 metrik; *Percent_Pub_Data, Access_To_Pub_Data, Dep_On_Child, Fan_In* şeklinde kısaltılmıştır. Çalışmada ayrıca

Halstead ve McCabe metriklerinden oluşan metot seviyesindeki metriklerin kullanılması da değerlendirilmiştir.

KC1 veri kümesi normalde metot seviyesindeki metrikleri içermektedir. Koru ve Liu (2005a), minimum, maksimum, ortalama ve toplam operatörlerini kullanarak bu metrikleri sınıf seviyesindeki metriklere dönüştürmüştür. Bu sayede, 21 adet metot seviyesindeki metrik 84 adet metriğe dönüşmüştür. 6 adet CK metriği, 4 adet yukarıda ifade edilen metrikler ve dönüşümden elde edilen 84 adet metrik bir araya getirilerek, 94 adet metrik elde edilmiştir.

KC1 veri kümesinde yer alan ve dönüşüme uğratılan 21 adet metrik Çizelge 7.1'de verilmektedir. Araştırmada kullanılan nesneye yönelik diğer metrikler ise *Percent_Pub_Data, Access_To_Pub_Data, Dep_On_Child, Fan_In, Coupling_Between_Obj, Depth_Of_Inheritance_Tree, Lack_Of_Cohes_Of_Methods, Num_Of_Children, Response_For_Class, Weighted_Methods_Per_Class* metrikleridir. Bu çalışmada kullanılan veri kümesi, yerdeki veriyi alıp / göndermek için kullanılan depolama yönetimi projesindeki veriler kullanılarak NASA'da oluşturulmuştur. Programlama dili olarak C++ kullanılmıştır. 2107 metot, 145 sınıf ve yaklaşık olarak 40,000 kod satırı bu proje için gerçeklenmiştir.

Çalışmada kullanılan nesneye yönelik metrikler aşağıda açıklanmaktadır:

- *WMC (Weighted Method per Class):* WMC, bir sınıftaki toplam metot sayısıdır.
- *DIT (Depth of Inheritance Tree):* DIT, bir sınıftan kalıtım ağacındaki kök elemana olan en uzak yolun mesafesidir.
- *RFC (Response for a Class):* Bir sınıf içerisinde yer alan metot sayısı ile kalıtımdan gelen metot sayısı toplanarak hesaplanır.
- *NOC (Number of Children):* Sınıfın çocuklarının sayısı ya da bir sınıftan türeyen sınıfların sayısı olarak tanımlanabilir.
- *CBO (Coupling between Object Classes):* A sınıfı B sınıfının özelliklerini (attribute) veya operasyonlarını kullanıyorsa, A sınıfının B sınıfına bağlı (coupled) olduğu söylenir. CBO, bir sınıfın kalıtım dışında bağlı olduğu farklı sınıfların sayısıdır.
- *LCOM (Lack of Cohesion in Methods):* Her özellik (attribute) için o özelliği kullanan metotların yüzdesi belirlenip ortalaması alınır ve bu değer 100'den çıkarılır. Elde edilen değer, LCOM değeridir. Sınıf içi kohezyon yüksek olması gerekirken, sınıflar arası bağlılık (coupling) düşük olmalıdır.

- *PERCENT_PUB_DATA*: Bir sınıf için public veya protected özelliklerin (attribute) yüzdesi olarak hesaplanır.
- *ACCESS_TO_PUB_DATA*: Bir sınıf için, public veya protected özelliklere erişim sayısı olarak belirlenmektedir.
- *DEP_ON_CHILD*: Bir sınıfın çocuğuna bağlı olup olmadığını gösterir.
- *FAN_IN*: Daha üst modüllerden gelen çağrı sayısıdır.

WMC, DIT, RFC, NOC, CBO ve LCOM metrikleri, CK metrik kümesine ait olan metriklerdir. Basili ve arkadaşları (1996), bu metrikleri çalışmalarında kullanmıştır. Rosenberg ve arkadaşları (1999), güvenilirlik değerlendirilmesi (reliability assessment) için 6 adet CK metriğini, çevrimsel karmaşıklığı, kod satır sayısını ve yorum satırı yüzdesini önermiştir. Briand ve arkadaşları (Briand vd., 1999b), lojistik regresyon yöntemini kullanarak CBO, RFC ve LCOM metriklerinin sınıfların kusur eğilimliliği ile önemli oranda korele olduğunu gözlemlemişlerdir. Briand ve arkadaşları (2000), kod satır sayısının etkisini incelemiş ve CBO, RFC, DIT, kod satır sayısı, NOC metriklerinin kusur kestirimi için önemli olduğunu raporlamışlardır. Briand ve arkadaşları (2001), NOC metriğinin kusur eğilimliliğini belirlemede önemli bir metrik olmadığını belirlemişlerdir ve bu çalışmaları 2000 yılındaki çalışmalarındaki sonuçlarıyla çelişmektedir. Yu ve arkadaşları (Yu vd., 2002), WMC, kod satır sayısı, CBO, RFC, LCOM, NOC metriklerinin kusur eğilimliliği ile ilişkili olduğunu ancak DIT metriğinin kusur kestirimi için yararlı olmadığını göstermişlerdir. Gyimothy ve arkadaşları (2005), NOC metriğinin kusur kestiriminde önemli olmadığını, kod satır sayısı ve diğer CK metriklerinin önemli olduğunu ifade etmişlerdir. Zhou ve Leung (2006), DIT metriği dışında diğer CK metriklerinin ve kod satır sayısı metriğinin kusur eğilimliliğini belirlemede önemli olduğunu göstermişlerdir. Gerçekleştirilen bu çalışmalara göre, çoğu araştırmacı lojistik regresyon yöntemini kullanmış ve WMC, RFC, CBO metrikleri hemen hemen tüm çalışmalarda kusur kestirimi için önemli olduğu saptanmıştır.

6.2.4 Performans Ölçüm Kriteri

Bu çalışmada, G-ortalama1, G-ortalama2, F-ölçüm parametreleri performans değerlendirilmesi için kullanılmıştır. KC1 veri kümesi içindeki metrikleri ve kusur bilgilerini kullanarak modelimizi oluşturduktan sonra, 10 katlı çapraz-geçerleme tekniği ile test ettik. Çapraz geçerlemeden sonra elde edilen hata matrisini kullanarak, performans değerlendirme kriterlerini hesapladık. Kullanılan parametrelere ilişkili ayrıntılı bilgiler 6. bölümde

açıklanmaktadır.

6.2.5 Deneysel Çalışmalar

Metot seviyesindeki metriklerle yaptığımız çalışmada, AIRS algoritmasının korelasyon tabanlı özellik seçimi (cfs) ile birlikte kullanıldığı modelin büyük ölçekli projelerde yüksek performans sunduğunu belirlemiştik. Ancak yapılan o çalışmada, sınıf seviyesindeki metrikler kullanılmamıştı. Bu çalışmada ise nesneye yönelik metrikleri AIRS tabanlı kestirim modelinde kullanmak hedeflendi. Kusur kestirimi için önemli nesneye yönelik metrikler ve üzerinde çalışılan metriklerin en iyi kombinasyonu belirlenmeye çalışıldı. Nesneye yönelik her CK metriğinin önem seviyesi, o metrikle sağlanan AIRS tabanlı modelin performans seviyesine bağlı olarak saptanmıştır. Sonuçlar Çizelge 6.12'de gösterilmektedir. Çizelge 6.12'ye göre, CK ve kod satır sayısı metriklerinin AIRS ile birlikte kullanılması sonucu oluşan model performansı, diğer metrikleri kullanan AIRS tabanlı yaklaşımlardan daha başarılıdır.

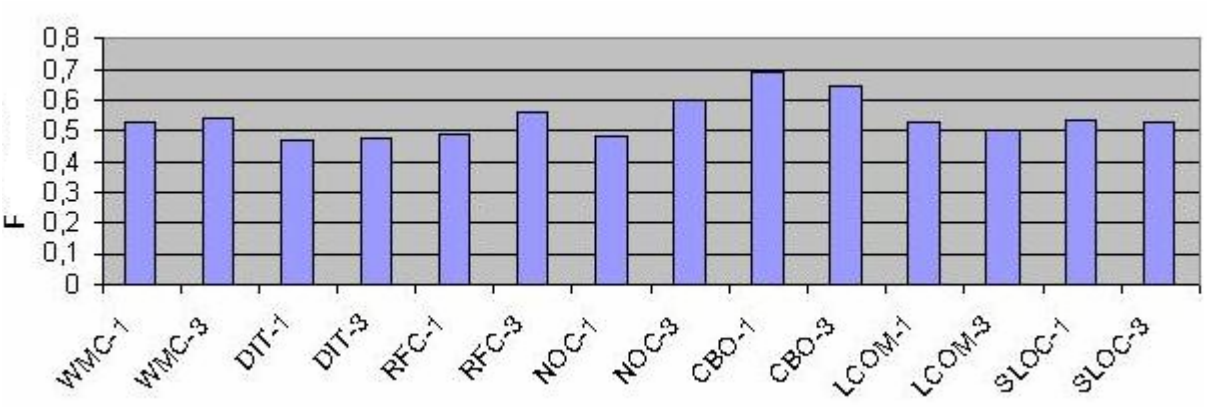
Çizelge 6.12'de verilen, cfs tekniği olarak ifade edilen yöntem, korelasyon tabanlı özellik azaltma tekniğidir. cse tekniği olarak ifade edilen yöntem ise uyumluluk tabanlı özellik seçim (consistency-based feature selection) yöntemidir.

Çizelge 6.12'ye göre, performans indikatörleri için eşik seviyesi 0.5 olarak belirlendiği zaman, kusur kestirimi için önemli olmayan metrik sadece DIT metriğidir. AIRS algoritmasındaki k değeri 1 ve 3 olarak seçildiği zaman, performansların daha yüksek olduğu saptandığı için çizelgede sadece bu sonuçlar verilmiştir. AIRS algoritmasındaki k değeri 1 veya 3 olarak seçildiği zaman, DIT dışındaki diğer CK metriklerinin performans indikatör değerleri 0.5'den daha yüksektir. Bu nedenle, diğer CK metriklerinin kusur kestirimi için önemli olduğu tespit edilmiştir. Bu empirik sonuç, Zhou ve Leung (2006) isimli araştırmacıların sonuçlarıyla uyumludur. Bu araştırmacılar, tek değişkenli lojistik regresyon tekniğini aynı veri kümesinde kullanarak sadece DIT metriğinin aşamasız kusur kestiriminde önemli olmadığını saptamışlardır. "Aşamasız önem kusur analizi" (ungraded severity faults analysis) başlığı altında sundukları bilgiler, kusur seviyelerine öncelik seviyesinin verilmediği ve içerik açısından gerçekleştirdiğimiz deneylerle doğrudan ilişkili olan bilgilerdir. Bu çalışmalarında, DIT metriğinin kusur analizinde önemsizliğini vurgulamışlardır. Kullandığımız model ve performans kriterlerimiz farklı olmasına rağmen, elde ettiğimiz sonuçlar Zhou ve Leung'un çalışmasıyla aynıdır. Tang ve arkadaşları (1999), Yu ve arkadaşları (Yu vd., 2002), El-Emam ve arkadaşları (El-Emam vd., 2001c) sırasıyla lojistik

regresyon, olağan en küçük kare lineer regresyon (ordinary square linear regression), lojistik regresyon teknikleriyle DIT metriğinin kusur eğilimliliği belirlemede önemli olmadığını göstermişlerdir.

Çizelge 6.12'ye göre, en yüksek performans indikatörlerine sahip olan CBO metriğinin kusur kestirimi için en önemli CK metriği olduğu sonucunu çıkarabiliriz. Bu sonucumuz yine Zhou ve Leung (2006) isimli araştırmacıların sonucuyla uyumludur. Çalışmalarında, R^2 parametresine göre en önemli CK metriği olarak CBO metriğini bulduklarını ifade etmişlerdir. R^2, tek değişkenli lojistik regresyon analizlerinde bağımsız değişkenlerin etkisini göstermektedir. Ancak Zhou ve Leung, kendi yaptıkları çalışmada kod satır sayısı metriğine ilişkin R^2 değerinin CBO'ya ilişkin R^2 değerinden daha yüksek olduğunu açıklamışlardır. Bu nedenle, yaptıkları çalışmaya göre kod satır sayısı metriği kusur eğilimliliği için CBO metriğinden daha önemli bir metriktir fakat bizim çalışmalarımıza göre bu bilgi geçerli değildir.

Gerçekleştirdiğimiz çalışmalarda, CBO ile elde edilen performans indikatörleri kod satır sayısına ilişkin elde edilen indikatörlerden daha yüksektir. Bu noktada farklılık olmasına rağmen, iki çalışma da CBO metriğinin en önemli CK metriği olduğunu ortaya koymaktadır. Şekil 6.1'de AIRS tabanlı model için CK metriklerinin ve kod satır sayısı metriğinin performansları, F-ölçüm değerine göre gösterilmektedir. Bu şekle göre, en yüksek performansı CBO, en düşük performansı da DIT metriği sunmaktadır.



Şekil 6.1 CK metrikleri ve kod satır sayısı metriğinin AIRS modelindeki performansları

Çizelge 6.12 KC1-sınıf-seviyesindeki dosya için AIRS ile elde edilen sonuçlar

Girdiler	k	PD	ACC	Prec	G-1	G-2	F
WMC	1	0.548	0.538	0.506	0.527	0.545	0.526
WMC	3	0.504	0.517	0.576	0.539	0.505	0.538
DIT	1	0.470	0.469	0.459	0.465	0.471	0.465
DIT	3	0.508	0.496	0.447	0.476	0.507	0.475
RFC	1	0.472	0.476	0.506	0.488	0.469	0.488
RFC	3	0.570	0.565	0.553	0.561	0.568	0.561
NOC	1	0.517	0.503	0.447	0.481	0.515	0.480
NOC	3	0.510	0.552	0.729	0.610	0.518	0.600
CBO	1	0.639	0.683	0.765	0.690	0.672	0.696
CBO	3	0.594	0.627	0.706	0.647	0.615	0.645
LCOM	1	0.520	0.524	0.541	0.530	0.520	0.530
LCOM	3	0.541	0.524	0.471	0.505	0.536	0.503
SLOC (kod satır sayısı)	1	0.558	0.545	0.506	0.532	0.553	0.531
SLOC (kod satır sayısı)	3	0.579	0.552	0.482	0.528	0.568	0.526
6 CK metriği	1	0.633	0.662	0.718	0.674	0.653	0.673
6 CK metriği	3	0.646	0.676	0.729	0.686	0.667	0.685
SLOC + 6 CK metriği	1	0.653	0.655	0.659	0.656	0.654	0.656
SLOC + 6 CK metriği	**3**	**0.712**	**0.724**	**0.741**	**0.726**	**0.721**	**0.726**
10 Nesneye yönelik metrik	1	0.590	0.614	0.671	0.629	0.604	0.628
10 Nesneye yönelik metrik	3	0.621	0.641	0.682	0.651	0.634	0.650
SLOC + 10 Nesneye yönelik metrik	1	0.650	0.662	0.682	0.666	0.658	0.666
SLOC + 10 Nesneye yönelik metrik	3	0.676	0.710	0.765	0.719	0.702	0.718
cfs tekniği ile azaltılmış metrikler	1	0.690	0.710	0.741	0.715	0.705	0.715
cfs tekniği ile azaltılmış metrikler	3	0.669	0.696	0.741	0.704	0.689	0.703
cse tekniği ile azaltılmış metrikler	1	0.647	0.690	0.765	0.704	0.679	0.701
cse tekniği ile azaltılmış metrikler	3	0.640	0.676	0.741	0.689	0.666	0.687
SLOC, WMC, RFC, NOC, CBO, LCOM	1	0.633	0.662	0.718	0.674	0.653	0.673
SLOC, WMC, RFC, NOC, CBO, LCOM	3	0.659	0.690	0.741	0.699	0.681	0.698

Koru ve Liu'ya göre (2005a), sınıf seviyesindeki veriler metot seviyesindeki verilerden daha iyi kestirim olanağı sunmaktadır. KC1 veri kümesi üzerinde, metot seviyesindeki metrikleri kullanarak farklı algoritmaların performans indikatörlerini elde ettik ve bu değerleri bu çalışmada elde ettiğimiz değerlerle kıyasladık. Çizelge 6.13'de metot seviyesindeki metriklerle elde edilen sonuçlar verilmektedir. Çizelge 6.13'de pca-AIRS1 ile ifade edilen model, özellik azaltma tekniği olarak temel bileşen analizinin kullanıldığı ve sonrasında elde edilen metrikler üzerinde AIRS1 algoritmasının uygulandığı modeldir. Benzer şekilde; cfs korelasyon tabanlı, cse ise uyumluluk tabanlı özellik seçimi tekniklerinin uygulandığı modelleri temsil etmektedir. Ma ve arkadaşlarının (2006), Rastgele Orman algoritmasını kusur kestirimi için önerdiği çalışmada elde edilen değerler de Çizelge 6.13'de gösterilmektedir. Ma ve arkadaşlarının (2006) çalışmasında metot seviyesinde metriklerle elde edilen en yüksek performans değerleri, sınıf seviyesindeki metrikleri kullandığımız AIRS tabanlı modelimizin performansından daha düşüktür. Bu nedenle, sınıf seviyesindeki metrikleri kullanmanın model performansını arttırdığını söyleyebiliriz. Koru ve Liu (2005a), *kc1-sınıf-seviyesi* veri kümesinde J48 ve KStar algoritmalarıyla daha iyi performans aldıklarını ifade etmiştir. Ancak bu yaptıkları çalışmaların amacının, çeşitli algoritmaları kıyaslayarak en iyisini bulmak olmadığını da vurgulamışlardır. Bu noktada, AIRS tabanlı modelimizin performansını Koru ve Liu'nun (2005a) elde ettiği performans sonuçlarıyla kıyaslamak istedik. İki çalışmada da aynı veri kümelerinin kullanılmış olması bu isteği mümkün kılmıştır.

Koru ve Liu (2005a), performans indikatörü olarak F-ölçüm, hassasiyet (precision), çağrı (recall) değerlerini kullandığı için deneylerimizde bu değerleri de hesapladık. Karşılaştırmalı sonuçlar Çizelge 6.14'de verilmektedir. AIRS tabanlı kestirim modelimiz HR ve LMR sınıflarının ikisi için de J48 tabanlı modele göre daha yüksek F-ölçüm değerleri (HR sınıfı için 0,68, LMR için 0,76) sunmaktadır. HR, kusur eğilimliliği yüksek sınıfları temsil ederken, LMR kusur eğilimi düşük sınıfları göstermektedir.

Koru ve Liu çalışmalarında, 6 CK metriği ve kod satır sayısı metriğini kullanmadığı için bu çalışmamızda onların önerdiği algoritmaları, bu metrikler ile kullanarak gerekli performans indikatörlerini de hesapladık. Çizelge 6.14'deki son iki satır bu sonuçları göstermektedir. Bu çizelgeye göre, 6 adet CK metriği ve kod satır sayısı metriğinin kullanıldığı AIRS tabanlı kusur kestirim modelinin Koru ve Liu tarafından önerilen J48 tabanlı yaklaşımdan daha yüksek performanslı olduğu sonucunu çıkarabiliriz.

Çizelge 6.13 KC1 için metot seviyesi metriklerle elde edilen sonuçlar

Girdiler	PD	ACC	Prec	G-1	G-2	F
Logistic	0.752	0.711	0.317	0.488	0.727	0.446
Discriminant	0.638	0.790	0.390	0.499	0.722	0.484
Tree	0.193	0.848	0.523	0.318	0.432	0.282
RuleSet	0.187	0.852	0.564	0.325	0.427	0.281
Boosting	0.169	0.862	0.732	0.352	0.409	0.275
KStar	0.509	0.855	0.532	0.521	0.684	0.520
VF1	0.957	0.195	0.156	0.387	0.231	0.269
RF (0.9,0.1)	0.844	0.711	0.330	0.528	0.761	0.475
RF (0.8,0.2)	0.700	0.782	0.387	0.520	0.747	0.498
RF (0.7,0.3)	0.561	0.825	0.447	0.501	0.700	0.498
AIRS1 (k=3)	0.634	0.746	0.298	0.435	0.586	0.405
cfs-AIRS1 (k=3)	0.692	0.763	0.368	0.504	0.628	0.480
cfs-AIRS1 (k=1)	0.589	0.660	0.426	0.501	0.569	0.494
pca-AIRS1(k=3)	0.563	0.653	0.383	0.464	0.548	0.456
pca-AIRS1(k=1)	0.574	0.634	0.448	0.507	0.561	0.503
cse-AIRS1(k=3)	0.708	0.773	0.374	0.515	**0.638**	**0.489**
cse-AIRS1(k=1)	0.647	0.715	0.426	0.525	0.609	0.514

Çizelge 6.14 Önerilen modelin Koru ve Liu'nun çalışmasıyla karşılaştırılması

Algoritmalar ve Girdiler	Sınıf	Hass.	Çağrı	F-ölçüm
J48, 94 metrik	HR	0.62	0.68	0.65
J48, 94 metrik	LMR	0.76	0.71	0.73
k=3, AIRS ve 6 CK + SLOC	HR	0.66	0.70	**0.68**
k=3, AIRS ve 6 CK + SLOC	LMR	0.78	0.74	**0.76**
J48 ve 6 CK + SLOC	HR	0.63	0.63	0.63
J48 ve 6 CK + SLOC	LMR	0.74	0.74	0.74

6.2.6 Sonuçlar ve Gelecek Çalışmalar

Bu çalışmada, Yapay Bağışıklık Tanıma Sistemi (AIRS) algoritmasını kullanan bir model geliştirilmiştir. İlk aşamada, CK metrikleri tek tek AIRS algoritmasında kullanılmış ve sonrasında farklı metrik kombinasyonları daha iyi kusur kestirim modelleri elde edilmesi için incelenmiştir. CK metrikleri ve kod satır sayısı metriğinin birlikte kullanıldığı AIRS tabanlı modelin diğer modellere göre daha iyi performans sunduğu gözlemlenmiştir. Elde edilen sonuçlar kusur kestirimi açısından oldukça memnun edici düzeydedir.

Bu çalışma literatüre çeşitli katkılar sunmaktadır: İlk olarak, kusur eğilimliliği ve 6 adet CK metriği arasındaki ilişkiyi ortaya koyan yeni bir kanıt sunduk. Çalışmamızda KC1 isimli açık bir veri kümesini PROMISE havuzundan kullanarak sonuçlarımızın yeniden üretilebilir, doğrulanabilir veya reddedilebilir olmasını sağladık. İkinci olarak, nesneye yönelik metrikleri kullanan AIRS tabanlı kusur kestirim modelini önerdik ve geçerlenmesini KC1 kümesi üzerinde sağladık. Yazılım kusur kestirimi için, Yapay Bağışıklık Sistem paradigmasının nesneye yönelik metriklerle birlikte kullanıldığı literatürdeki bu ilk çalışmanın sonuçları oldukça ümit vericidir. Üçüncü olarak, bu çalışma sınıf seviyesindeki metriklerin kusur kestiriminde metot seviyesindeki metriklerden daha önemli olduğunu ortaya koymaktadır. Son olarak deneysel çalışmalar, AIRS tabanlı önerdiğimiz modelin literatürde önceden önerilmiş J48 tabanlı modelden daha iyi sonuç verdiğini göstermiştir.

Zhou ve Leung'un çalışmasında (2006) olduğu gibi, bu çalışmadaki eksiklik sadece tek bir projeye ait veri kümesinin kullanılmasıdır. PROMISE havuzunda bu veri kümesi dışında nesneye yönelik metriklerin sunulduğu daha farklı bir veri kümesi olmadığından, sadece bu veri kümesi ile çalışılabilmiştir. Nesneye yönelik metriklerin bulunduğu daha farklı veri kümelerine ileriki dönemlerde erişilebilirse, geliştirilen modellerin test edilmesi sağlanabilir. Ayrıca, önerilen modeldeki AIRS algoritması literatürde performansı yüksek olduğu raporlanan diğer algoritmalarla birlikte kullanılarak daha yüksek performanslı modeller kurgulanabilir. Sınıflayıcı birlikteliği (ensemble of classifiers) yaklaşımları bu noktada değerlendirilebilecek yaklaşımlardandır.

6.3 Kusur Kestirimi Modellerinin Karşılaştırmalı İncelenmesi

Bu bölümde, literatürde kusur kestirimi için en yüksek performans sunduğu raporlanan makine öğrenmesi tabanlı sınıflayıcılar ve Yapay Bağışıklık Sistem tabanlı sınıflandırma algoritmaları ile gerçekleştirilen deneyler sunulmaktadır. NASA açık veri kümeleri üzerinde farklı yazılım metrik kümeleri ve farklı özellik azaltma teknikleri kullanıldığı durumda elde edilen performanslar karşılaştırılmalı olarak açıklanmaktadır. Deneyler, araştırma öncesi belirlenen çeşitli araştırma soruları ışığında gerçekleştirilmiştir. Literatürde ilk kez Yapay Bağışıklık Sistem tabanlı sınıflandırma algoritmaları ve yüksek performanslı kusur kestiriciler bu detayda kusur kestirimi amaçlı incelenmektedir.

6.3.1 Özet

Birçok araştırmacı, makine öğrenmesi tabanlı ve istatistiksel yöntemlerle kusur kestirimi için modeller geliştirmiş ve farklı veri kümelerinde modellerinin geçerlenmesini sağlamışlardır. Kullanılan yazılım metrikleri, veri kümeleri ve özellik azaltma teknikleri modellerin performansını etkileyen faktörlerdir. Bu çalışmada, makine öğrenmesi tabanlı yüksek performans sunan sınıflayıcılara ve YBS tabanlı sınıflandırma algoritmalarına odaklanılmıştır. Kestirim modellerine ilişkin deneyleri, tekrarlanabilir, doğrulanabilir ya da reddedilebilir kılmak için NASA açık veri kümeleri PROMISE havuzu içerisinden kullanılmıştır. Araştırma sorularına yanıt bulabilmek için 7 test grubu belirlenmiş ve 5 veri kümesi üzerinde 9 sınıflayıcı incelenmiştir. Bu çalışmaya göre, Rastgele Orman algoritması büyük veri kümelerinde en iyi performansı sunmaktadır ve daha küçük veri kümeleri için Naive Bayes algoritması en iyi kestirimi sağlamaktadır. AIRS algoritmasının paralel gerçeklenmesi (AIRS2Paralel), metot seviyesinde metrikler kullanıldığı durumda YBS tabanlı algoritmalardan en iyisi olarak karşımıza çıkmaktadır. Sınıf seviyesindeki metrikler kullanıldığı durumda ise Immunos2 algoritmasının performansı oldukça yüksektir.

6.3.2 Giriş

Birçok araştırmacı, farklı algoritmaların kusur kestiriminde en iyi performansı sunduğunu ifade etmiştir. Örneğin; farklı araştırmacılar tarafından Rastgele Orman (Ma vd., 2006), J48 (Koru ve Liu, 2005a), Naive Bayes (Menzies vd., 2007a) algoritmalarının kusur kestiriminde en iyi performansı sunduğu raporlanmıştır. Kullanılan metrikler, uygulanan ön işleme yöntemleri ve özellik azaltma teknikleri bu noktada modellerde farklılıklar yaratabilmektedir. Temelinde aynı algoritma kullanılmış olsa da, bu farklılıklar nedeniyle performanslar değişebilmektedir. Bu şartlar dikkate alınarak, en yüksek performansı sunduğu raporlanan bu

sınıflayıcılar ile birlikte YBS tabanlı sınıflayıcıların kapsamlı olarak deneysel çalışmalarla incelenmesi hedeflenmiştir. Rastgele Orman (Random Forests-RF) algoritması, yüzlerce veya binlerce ağacı kullanmakta, sınıflandırma için bu ağaçların sonuçlarını değerlendirmektedir. J48 algoritması, Quinlan'ın C4.5 algoritması tabanlı olup bir karar ağacı gerçeklemesidir. Naive Bayes algoritması, olasılıksal sınıflandırma algoritmasıdır ve güçlü bağımsızlık varsayımları içerir.

Bu algoritmalara ek olarak; Immunos1, Immunos2, CLONALG, AIRS1, AIRS2 ve AIRS2Paralel algoritmaları deneylerde kullanılmıştır. AIRS algoritması bağışıklık sisteminden esinlenen ve birçok sınıflandırma probleminde yüksek performans sunduğu bilinen bir algoritmadır. 5 adımdan oluşmaktadır: ilklendirme, antijen eğitimi, sınırlı kaynak için yarışma, bellek hücresi seçimi ve sınıflandırmadır. Immunos81 algoritması bağışıklık sisteminden esinlenmiş ilk sınıflandırma algoritmasıdır. Brownlee, Carter (2000) tarafından açıklanan sonuçları elde edebilmek için genelleştirilmiş Immunos-81 sisteminin Immunos1 ve Immunos2 gerçeklemelerini geliştirmiştir. CLONALG, bağışıklık alanında yer alan klonal seçim teorisini kullanmaktadır. CLONALG algoritması AIRS algoritmasına kıyasla daha az sayıda kullanıcı tanımlı parametre içermektedir ve karmaşıklığı daha düşüktür (Brownlee, 2005a). CLONALG; örüntü tanıma, fonksiyon optimizasyonu ve kombinasyonal (combinatorial) optimizasyon gibi alanlarda kullanılmıştır.

Çizelge 6.1'de NASA veri kümelerinde mevcut olan 21 adet metrik gösterilmektedir. Ancak bazı araştırmacılar (Seliya ve Khoshgoftaar, 2007a), bu 21 metrik yerine sadece 13 tanesini çalışmalarında kullanmaktadır. Bunun sebebi ise türetilmiş Halstead metriklerinin (1977) yazılım kusur kestirimi için ek bir bilgi sunmadığını düşünüyor olmalarıdır. Munson (2003), "Yazılım Mühendisliği Ölçümü" (Software Engineering Measurement) kitabında Halstead metriklerinden sadece 4 temel metriğin yeterli olduğunu ve türetilmiş Halstead metriklerinden yeni bir bilgi elde etmenin mümkün olmadığını açıklamıştır. Bu nedenle, bu deneylerde ilk olarak 21 adet metrikle çalışılmış ve ardından metrik sayısı 13'e indirilerek deneyler tekrarlanmıştır. Bu sayede, metrik değişimi ile birlikte en iyi sınıflandırma algoritmasının değişip değişmediği de gözlemlenebilmiştir. Gerçekleştirilen bu iki deney türü, ilk ve ikinci araştırma sorularına yanıt vermek üzere kurgulanmıştır.

Bazı araştırmacılar, temel bileşen analizi gibi özellik azaltma teknikleri ile model performanslarını iyileştirmek ve çoklu bağlanımı (multicollinearity) azaltmak istemektedirler. Algoritmaları uygulamadan önce, korelasyon tabanlı özellik seçimi (cfs) ile uygun metrikleri elde ederek bazı testleri gerçekleştirdik. Bölüm 6.1'de, cfs algoritması ile elde edilen

performansların diğer özellik azaltma yöntemlerine göre daha yüksek olması nedeniyle bu deneylerde kullanılmasına karar verilmiştir. Bu sayede, üçüncü araştırma sorusuna yanıt verme imkanı doğmuş oldu.

Metot seviyesindeki metriklere ek olarak, sınıf seviyesindeki metriklerle de yazılım kusur kestirimi yapılabilmektedir. Metot seviyesindeki metrikler, yordamsal ve nesneye yönelik programlama paradigması için uygundur. Ancak sınıf seviyesindeki metrikler, sadece nesneye yönelik programlama dilleri ile geliştirilmiş programlar için hesaplanabilir. Açık veri kümelerinden sadece KC1 veri kümesi sınıf seviyesindeki metrikleri içermektedir ve bu nedenle modellerin performansını sadece bu veri kümesinde inceleyebildik.

Üç deney türünde sınıf seviyesindeki metrikler kullanılmış ve bu sayede; araştırma sorularından dördüncü, beşinci, ve altıncısına yanıt verilmiştir. Son deney türü olan yedinci deneyde, çapraz geçerleme metodunun kullanılmadığı, veri kümesinin %20'sinin eğitim kümesi geri kalanının test kümesi olarak kullanıldığı durumda elde edilen performanslar incelenmiştir. Bu son deney, yedinci araştırma sorusuna yanıt vermek üzere gerçekleştirilmiştir.

Araştırma soruları aşağıda açıklanmaktadır:

- ***Araştırma Sorusu 1:*** *Metot seviyesindeki metrikler kullanıldığı zaman (21 metrik) büyük veri kümelerinde en iyi performansı hangi makine öğrenme algoritması verir?*
- ***Araştırma Sorusu 2:*** *Türetilmiş Halstead metrikleri kullanılmadığı zaman büyük veri kümelerinde en iyi performansı hangi makine öğrenme algoritması verir?*
- ***Araştırma Sorusu 3:*** *Tüm metot seviyesindeki metrikler üzerinde (21 metrik) korelasyon tabanlı özellik seçim tekniği çalıştırıldıktan sonra elde edilen metrikler kusur kestiriminde kullanıldığı zaman, büyük veri kümelerinde en iyi performansı hangi makine öğrenme algoritması sunar?*
- ***Araştırma Sorusu 4:*** *Altı adet CK metriği ve kod satır sayısı metriği kullanıldığı durumda açık veri kümelerinde (sadece KC1) en iyi performansı hangi makine öğrenme algoritması sunar?*
- ***Araştırma Sorusu 5:*** *Korelasyon tabanlı özellik seçim tekniği 94 adet sınıf seviyesindeki metrik üzerinde çalıştırıldıktan sonra elde edilen metriklerin kullanıldığı durumda açık veri kümelerinde (sadece KC1) en iyi performansı hangi makine*

öğrenme algoritması sunar?

- ***Araştırma Sorusu 6:*** *94 adet sınıf seviyesindeki metrik kullanıldığı zaman açık veri kümelerinde (sadece KC1) en iyi performansı hangi öğrenme algoritması sunar?*
- ***Araştırma Sorusu 7:*** *10 katlı çapraz geçerleme yerine, veri kümesinin %20'sinin eğitim kümesi ve geri kalanın test kümesi olarak kullanıldığı durumda, büyük veri kümelerinde, en iyi performansı hangi makine öğrenme algoritması sunar?*

Yaptığımız literatür taramasına göre bu çalışma, yazılım kusur kestirimi için bu düzeyde farklı metrik kombinasyonlarının kullanıldığı, yüksek performanslı kusur kestirim modellerinin ve YBS tabanlı sınıflayıcıların kusur kestirim amaçlı incelendiği ilk çalışmadır.

6.3.3 Deneyin Tanımlanması

Bu bölümde gerçekleştirilen deneyin tanımı, bağlam seçimi, değişken seçimi, hipotez formülasyonu, öznelerin seçimi, deneyin tasarımı, enstrümantasyon, geçerlilik değerlendirmesi ve operasyon açıklanmaktadır.

6.3.3.1 Deney Tanımı

Gerçekleştirdiğimiz deneyler aşağıdaki özelliklerle tanımlanabilmektedir:

- ***Çalışmanın nesnesi:*** Bir önceki yazılım metrikleri ve kusur verisidir.
- ***Amaç:*** Amaç, farklı metrik gruplarının kullanıldığı durumda en iyi kusur kestirim algoritmasını belirlemektir.
- ***Kalite odağı:*** Kalite odağı güvenilirlik ve kusur kestirim modellerinin etkinliğidir.
- ***Perspektif:*** Perspektif, araştırmacının bakış açısıdır.
- ***Bağlam:*** Deney NASA projelerinde elde edilen veri kümeleri kullanılarak gerçekleştirilmiştir.

6.3.3.2 Bağlam Seçimi

Deneyin bağlamı NASA veri kümeleridir. NASA veri kümelerinden JM1, KC1, PC1, KC2 ve CM1 veri kümeleri deneylerde kullanılmıştır.

6.3.3.3 Değişkenlerin Seçimi

Araştırma sorularına yanıt verebilmek için deneylerde farklı bağımsız değişkenler

kullanılmıştır. Ancak tüm deneylerde bağımlı değişken, kusur eğilimliliğidir. Bu değişken, kusur eğilimli veya kusur eğilimli değil şeklinde iki değer alabilmektedir. Aşağıda gerçekleştirilen deney tipleri ve bu deneylerde kullanılan bağımsız değişkenler açıklanmaktadır. Her deney türü bir araştırma sorusuna yanıt vermek üzere gerçekleştirilmiştir.

- **Deney 1:** Veri kümelerinde yer alan 21 adet metot seviyesindeki metrik kullanılmıştır.
- **Deney 2:** Türetilmiş Halstead metrikleri dışında kalan metot seviyesindeki metrikler kullanılmıştır.
- **Deney 3:** 21 adet metot seviyesindeki metrik üzerinde korelasyon tabanlı özellik seçim tekniği uygulanmış ve elde edilen metrikler bu deneylerde kullanılmıştır. Her veri kümesi için bu sayede farklı bağımsız değişkenler kullanılmıştır.
- **Deney 4:** Altı adet CK metriği ve kod satır sayısı metriği kullanılmıştır.
- **Deney 5:** 94 adet sınıf seviyesindeki metrik üzerinde korelasyon tabanlı özellik seçim tekniği uygulanmış ve elde edilen yedi adet metrik KC1 veri kümesi için kullanılmıştır. 94 metriğin elde edilme şekli, bölüm 7.2'de açıklanmıştır.
- **Deney 6:** Bu deneyde 94 adet sınıf seviyesindeki metrik kullanılmıştır.
- **Deney 7:** Sadece 13 adet metot seviyesindeki metrik kullanılmış ve türetilmiş Halstead metrikleri kullanılmamıştır. Bu deneyde, çapraz geçerleme tekniği kullanılmamıştır. Veri kümesinin %20'si eğitim amaçlı ve geri kalanlar test amaçlı kullanılmıştır.

6.3.3.4 Hipotez Formülasyonu

Yazılım mühendisliği deneyleri için hipotez formülasyonu kritik bir bileşendir. Deneylere ilişkin hipotezler aşağıda verilmektedir:

- ***Hipotez 1:*** *"21 metot seviyesindeki metrikle kestirim" hipotezi.* 21 metot seviyesindeki metrik kullanıldığı zaman büyük veri kümeleri için en iyi kusur kestirim algoritması RF'dir. (Sıfır hipotezi: 21 metot seviyesindeki metrik kullanıldığı zaman büyük veri kümeleri için en iyi kusur kestirim algoritması RF değildir.)
- ***Hipotez 2:*** *"13 metot seviyesindeki metrikle kestirim" hipotezi.* 13 metot seviyesindeki metrik kullanıldığı zaman büyük veri kümeleri için en iyi kusur kestirim algoritması

RF'dir. (Sıfır hipotezi: 13 metot seviyesindeki metrik kullanıldığı zaman büyük veri kümeleri için en iyi kusur kestirim algoritması RF değildir.)

- ***Hipotez 3:*** *"Metot seviyesindeki metrikler üzerinde korelasyon tabanlı özellik seçimi tekniği ile kestirim" hipotezi.* 21 metot seviyesindeki metrik üzerinde korelasyon tabanlı özellik seçim tekniği uygulandığı zaman, elde edilen metriklerle kusur kestirimi gerçekleştirildiğinde, en iyi kestirimi RF algoritması verir. (Sıfır hipotezi: 21 metot seviyesindeki metrik üzerinde korelasyon tabanlı özellik seçim tekniği uygulandığı zaman, elde edilen metriklerle kusur kestirimi gerçekleştirildiğinde, en iyi kestirimi veren algoritma, RF algoritması değildir.)

- ***Hipotez 4:*** *"CK metrik kümesiyle kestirim" hipotezi.* CK metrik kümesi ve kod satır sayısı metriği kullanıldığı durumda en iyi kestirimi veren algoritma RF'dir. (Sıfır hipotezi: CK metrik kümesi ve kod satır sayısı metriği kullanıldığı durumda en iyi kestirimi veren algoritma RF değildir.)

- ***Hipotez 5:*** *"94 sınıf seviyesindeki metrik üzerinde korelasyon tabanlı özellik seçim tekniğiyle (cfs) kestirim" hipotezi.* cfs tekniği 94 sınıf seviyesindeki metrik üzerinde kullanıldığı zaman, elde edilen metriklerle kusur kestirimi gerçekleştirildiğinde, en iyi kusur kestirimini sağlayan algoritma RF'dir. (Sıfır hipotezi: cfs tekniği 94 sınıf seviyesindeki metrik üzerinde kullanıldığı zaman, elde edilen metriklerle kusur kestirimi gerçekleştirildiğinde, en iyi kusur kestirimini sağlayan algoritma RF algoritması değildir.)

- ***Hipotez 6:*** *"94 metot seviyesindeki metrikle kestirim" hipotezi.* 94 adet metot seviyesindeki metrik kullanıldığı zaman en iyi kestirimi veren algoritma RF'dir. (Sıfır hipotezi: 94 adet metot seviyesindeki metrik kullanıldığı zaman en iyi kestirimi veren algoritma RF değildir.)

- ***Hipotez 7:*** *"%20 eğitim kümesiyle kestirim" hipotezi.* Veri kümesinin %20'si eğitim ve geri kalanı test kümesi olarak kullanıldığı zaman en iyi kestirimi sunan algoritma RF'dir. (Sıfır hipotezi: Veri kümesinin %20'si eğitim ve geri kalanı test kümesi olarak kullanıldığı zaman en iyi kestirimi sunan algoritma RF değildir.)

6.3.3.5 Öznelerin Seçimi

Özneler, NASA projelerinde çalışan deneyimli geliştiricilerdir.

6.3.3.6 Deneyin Tasarımı

Yedi farklı deney, yedi araştırma sorusuna ve yedi hipoteze bağlı olarak gerçekleştirildi. Tasarım bir faktörlü (makine öğrenme algoritması) ve 9 işlemlidir (treatment). Bu işlemler; RF, J48, Naive Bayes, Immunos1, Immunos2, CLONALG, AIRS1, AIRS2, AIRS2Paralel algoritmalarıdır.

6.3.3.7 Enstrümantasyon ve Ölçüm

Metrikler ve veri kümeleri, NASA veri kümesinden kullanıldığı için deneyin öznelerine herhangi bir materyal ve enstrüman sunmadık. Veri kümeleri PROMISE havuzundan aynen alınarak kullanılmıştır.

6.3.3.8 Geçerlilik Değerlendirmesi

Bu bölümde, deneylerin geçerliliğini tehdit edebilecek hususlara ilişkin bilgiler sunulacaktır.

- **Dış Geçerlilik:** Dış geçerlilik için tehlikeler, sonuçların deney şartları dışında genelleştirilebilirliğini sınırlandırmaktadır. Empirik çalışmalarda kullanılabilecek sistemlerin aşağıdaki özellikleri içermesi gerekir: "(1) Bir birey tarafından değil, bir grup tarafından geliştirilmiş olması; (2) öğrenciler tarafından değil, profesyoneller tarafından geliştirilmiş olması; (3) yapay bir ortamda değil, endüstriyel bir ortamda geliştirilmiş olması; (4) gerçek endüstriyel projelere göre kıyaslanabilmesi için yeterince büyük olması" (Khoshgoftaar vd., 2006). Tüm deneylerimiz bu özellikleri sağlayan sistemleri kullanmaktadır ve sonuçlar bu sayede deney şartları dışında genelleştirilebilir durumdadır. Ancak verinin karakteristikleri modellerin performanslarını etkileyebilmektedir. Bu nedenle farklı sistemler üzerinde en iyi algoritmanın ismi değişebilir. İncelenen algoritmalar içerisinde, RF ya da Naive Bayes'in yine iyi performans sunması beklenmektedir.

- **Sonuç Geçerliliği:** "Sonuç geçerliliği işlem ve çıktı (outcome) arasındaki istatistiksel ilişki ile ilgilenmektedir. Sonuç geçerliliği için tehlikeler, doğru çıkarımların çıkarılabilmesini etkileyen konulardır" (Khoshgoftaar vd., 2006). Bu çalışmada, testler 10-katlı çapraz geçerleme kullanılarak gerçekleştirilmiştir. İstatistiksel olarak anlamlı sonuçlar üretebilmek için testler 10 kez tekrarlanmıştır. Sınıflayıcı performansları, ROC eğrisi altında kalan (AUC) değerleri kullanılarak karşılaştırılmıştır.

- **İç Geçerlilik:** "İç geçerlilik; sonuçların, deneysel işlem değişkenlerinin (experimental treatment variables) manipulasyonuna bağlı olarak elde edildiğini gösterebilmektir"

(Khoshgoftaar vd., 2006). Kötü kusur kestirimi, gürültülü modüllerden kaynaklanmış olabilir. Gürültü metriklerde veya bağımlı değişkende yer alabilir. Önceki bölümlerde açıklandığı gibi JM1 veri kümesinde bu tür modüllerin varlığı bilinmektedir. Bu tür modüllerin gerçekten gürültülü olup olmadığı NASA kalite güvence uzmanları tarafından raporlanmadığı için çalışmalarımızda veri kümelerinden çıkartmadık. Eğer bu modüller çıkarılmış olsaydı, en iyi kusur kestirim algoritmalarının isimleri değişebilirdi.

- **Yapı Geçerliliği:** "Bu, çalışmada bağımlı ve bağımsız değişkenlerin ölçmeyi iddia ettiği konu ile ilgilidir" (Pai ve Dugan, 2007). Bu çalışmada iyi bilinen ve önceden geçerlenmiş yazılım metriklerini kullandığımız için, bu beğişkenlerin yapı geçerliliği için tehlike oluşturmadığını ifade edebiliriz. Kullanılan metot seviyesindeki metrikler ve CK metrik kümesi önceki çalışmalarda geçerlenmiştir ve birçok araştırmacı tarafından başarı ile kusur kestirim amaçlı kullanılmıştır.

6.3.3.9 Deney Operasyonu

NASA veri kümeleri aynen kullanılmış ve ek işlemler yapılmamıştır. Eğer metrikleri ve kusur verisini, çevrim içi bir projeden elde edecek olsaydık bu işlemi hazırlık ve yürütme aşamalarına bölebilirdik.

6.3.4 Deneyin Analizi

Bu bölümde, araştırma sonuçlarını adresleyen deneyler açıklanmaktadır. Her alt bölüm veya deney, bir araştırma sorusu ve bir hipotezle ilgilidir.

6.3.4.1 Deney #1

Tüm deneylerde kullanılan algoritmalar; J48, RF, Naive Bayes, Immunos1, Immunos2, CLONALG, AIRS1, AIRS2, AIRS2Paralel olarak sıralanabilir. Tüm deneyler 10 katlı çapraz geçerleme ile gerçekleştirilmiştir. Testler 10 kez tekrarlanmıştır. Performanslar, AUC değerlerine göre kıyaslanmıştır.

Bu deneyde 21 metrik kullanılmıştır ve bu metrikler Çizelge 6.1'de gösterilmiştir. Çizelge 6.15, bu deney için algoritmaların sunduğu performansları ortaya koymaktadır. AUC değerlerinin yanı sıra, doğruluk değerlerini de hesaplayarak kabul edilebilir doğruluğu sağlamayan algoritmaları belirleyebildik. Çizelge 6.16'da bu deney için algoritmalarla elde edilen doğruluk değerleri verilmektedir.

Çizelge 6.15 Deney #1 için algoritmaların AUC değerleri

Algoritmalar	**KC1**	**KC2**	**PC1**	**CM1**	**JM1**
J48	0.70	0.69	0.64	0.56	0.66
RandomForests	**0.79**	0.80	**0.81**	0.70	**0.72**
NaiveBayes	**0.79**	**0.84**	0.71	**0.74**	0.69
Immunos1	0.68	0.68	0.53	0.61	0.61
Immunos2	0.51	0.72	0.50	0.50	0.50
CLONALG	0.52	0.58	0.51	0.50	0.51
AIRS1	0.60	0.69	0.56	0.55	0.55
AIRS2	0.57	0.67	0.57	0.53	0.54
AIRS2 Paralel	0.61	0.69	0.57	0.54	0.56

Çizelge 6.16 Deney #1 için algoritmaların doğruluk değerleri

Algoritmalar	**KC1**	**KC2**	**PC1**	**CM1**	**JM1**
J48	84.04	81.19	93.63	88.05	79.81
RandomForests	85.37	82.66	93.48	88.68	80.95
NaiveBayes	82.46	83.62	89.00	84.84	80.42
Immunos1	50.57	53.49	15.84	32.74	56.23
Immunos2	75.18	65.90	93.06	88.92	80.67
CLONALG	82.46	77.94	90.61	87.72	74.21
AIRS1	77.11	78.38	89.40	82.67	69.76
AIRS2	73.10	79.60	91.37	85.92	69.56
AIRS2 Paralel	81.94	81.98	91.86	86.00	71.48

JM1 ve PC1 veri kümeleri, deneylerde kullanılan en büyük veri kümeleridir. Çizelge 6.15'e göre, JM1 ve PC1 veri kümesinde en iyi performansı RF algoritması vermektedir. Bu nedenle ilk hipotezimiz geçerlidir ve ilk araştırma sorusuna yanıtımız RF algoritmasıdır. RF algoritmasına ek olarak, Naive Bayes algoritmasının performansı da oldukça yüksektir. KC2 ve CM1 gibi küçük veri kümelerinde, Naive Bayes'in performansı RF'den biraz daha iyidir. RF algoritması veri kümeleri büyüdükçe daha iyi sonuçlar üretebilmektedir, bu da makine öğrenmesi algoritmalarının karakteristik özelliklerindendir. AIRS algoritmasının paralel gerçeklemesi, büyük veri kümeleri için YBS tabanlı en iyi algoritma özelliğindedir. PC1 ve JM1 veri kümelerinde Immunos1, Immunos2 ve CLONALG algoritmalarının performansı

çok düşüktür, bu nedenle büyük veri kümelerinde bu algoritmalar kullanılamaz. PC1 ve JM1 için Immunos1'in performansı sırasıyla %15.84 ve %56.23'tür. AIRS1 algoritmasının performansı AIRS2'den daha iyidir ancak AIRS2 Paralel algoritmasından daha düşüktür. Bu deneylere göre, RF algoritmasını büyük veri kümeleri için, Naive Bayes algoritmasını da küçük veri kümeleri için öneriyoruz.

6.3.4.2 Deney #2

Bu deney türünde, 13 metot seviyesindeki metrik kullanılmış ve türetilmiş Halstead metrikleri kullanılmamıştır. Çizelge 6.17, bu deney için algoritmaların sunduğu performansları göstermektedir. Çizelge 6.18'de ise algoritmaların doğruluk değerleri verilmektedir. Çizelge 6.17'ye göre, JM1 ve PC1 için en iyi performansın yeniden RF algoritması ile sağlandığı kolayca söylenebilir. RF ve Naive Bayes algoritmalarının AUC değerleri çok fazla değişmemiştir. AIRS2Paralel algoritmasının performansı düşmüş ancak performans değişimi çok önemli düzeyde değildir. İkinci hipotezimiz geçerlidir ve ikinci araştırma sorusuna yanıtımız RF algoritmasıdır. AIRS2Paralel algoritması, yine büyük veri kümeleri için, YBS tabanlı en iyi kusur kestirimi algoritmasıdır. PC1 ve JM1 veri kümesi için Immunos1, Immunos2, CLONALG algoritmalarının performansı yine çok düşüktür. Deney 1 ve deney 2'nin sonuçları, türetilmiş Halstead metriklerinin performansı önemli ölçüde etkilemediğini göstermiş ve kusur kestirimi için toplanmalarının gerekli olmadığı sonucunu çıkarmamızı sağlamışlardır. Bu deney sonuçlarına göre, büyük veri kümeleri için RF algoritmasını ve küçük veri kümeleri için Naive Bayes algoritmasını öneriyoruz.

Çizelge 6.17 Deney #2 için algoritmaların AUC değerleri

Algoritmalar	**KC1**	**KC2**	**PC1**	**CM1**	**JM1**
J48	0.70	0.70	0.65	0.53	0.69
RandomForests	**0.80**	0.81	**0.79**	0.72	**0.72**
NaiveBayes	0.79	**0.84**	0.70	**0.74**	0.66
Immunos1	0.71	0.73	0.64	0.63	0.63
Immunos2	0.50	0.51	0.50	0.50	0.50
CLONALG	0.53	0.61	0.50	0.50	0.52
AIRS1	0.59	0.69	0.56	0.55	0.53
AIRS2	0.59	0.65	0.56	0.54	0.56
AIRS2 Paralel	0.60	0.67	0.58	0.53	0.56

Çizelge 6.18 Deney #2 için algoritmaların doğruluk değerleri

Algoritmalar	**KC1**	**KC2**	**PC1**	**CM1**	**JM1**
J48	84.63	81.36	93.48	88.60	80.22
RandomForests	85.77	83.55	93.84	88.45	80.66
NaiveBayes	82.54	83.68	88.80	86.22	80.43
Immunos1	56.16	61.37	37.00	40.25	62.58
Immunos2	84.54	79.47	93.06	90.16	80.67
CLONALG	81.55	78.08	92.19	88.38	77.30
AIRS1	75.69	76.88	87.16	82.07	63.10
AIRS2	78.11	76.01	88.83	84.91	72.49
AIRS2 Paralel	80.73	80.88	91.31	85.59	71.48

6.3.4.3 Deney #3

Bu deneyde, korelasyon tabanlı özellik azaltma (cfs) tekniğini 21 metrik üzerinde kullanarak çoklu bağlanımı azaltmayı hedefledik. cfs uygulandıktan sonra, aşağıdaki metrikler elde edilmiştir:

- CM1: loc, iv(g), i, lOComment, lOBlank, uniq_Op, uniq_Opnd
- JM1: loc, v(g), ev(g), iv(g),i, lOComment, lOBlank, locCodeAndComment
- KC1: v, d, i, lOCode, lOComment, lOBlank, uniq_Opnd, branchCount
- KC2: ev(g), b, uniq_Opnd
- PC1: v(g), I, lOComment, locCodeAndComment, lOBlank, uniq_Opnd

Çizelge 6.19 algoritmaların AUC değerlerini göstermektedir. Çizelge 6.20'de algoritmaların doğruluk değerleri verilmektedir. Çizelge 6.19'a göre, JM1 ve PC1 için en iyi performansı RF algoritması sunmaktadır. RF ve Naive Bayes algoritmalarının AUC değerleri, önemli ölçüde değişmemiştir. AIRS2Paralel algoritmasının performansı azalmıştır ancak performans değişimi çok önemli değildir. Üçüncü hipotezimiz geçerlidir ve üçüncü araştırma sorusuna yanıtımız RF algoritmasıdır. AIRS2Paralel algoritması büyük veri kümeleri için YBS tabanlı en iyi kusur kestirim algoritmasıdır. AIRS2 algoritması, AIRS2Paralel algoritması ile aynı performansı sunmuştur. PC1 ve JM1 için Immunos1, Immunos2 ve CLONALG algoritmalarının performansı yine çok düşüktür ve büyük veri kümelerinde kullanılması

mümkün değildir. İlk üç deney tipine göre seçilen metriklerin kusur kestirimi için çok önemli olmadığı sonucu çıkarılabilir. Daha kritik olan nokta seçilen sınıflandırma algoritması veya modeldir. Bu deney tipine göre, büyük veri kümeleri için yine RF algoritmasını küçük veri kümeleri için ise Naive Bayes algoritmasını öneriyoruz.

6.3.4.4 Deney #4

Bu deneyde, sınıf seviyesindeki metrikler ve kod satır sayısı metriği kullanılmıştır. Sadece KC1 veri kümesinde sınıf seviyesindeki metrikler mevcut olduğu için, dokuz algoritma KC1 veri kümesi üzerinde, altı CK metriği ve kod satır sayısı metriği kullanılarak analiz edilmiştir. KC1 veri kümesi, çok büyük bir veri kümesi değildir ve JM1'e kıyasla daha az veri noktası ve daha az metrik içermektedir. Çizelge 6.21 algoritmaların AUC değerlerini, Çizelge 6.22 doğruluk değerlerini göstermektedir. Çizelge 6.21'e göre, sınıf seviyesindeki metrikler ve kod satır sayısı metriği kullanıldığı durumda en iyi performansı RF algoritması vermektedir.

Ayrıca; Naive Bayes, J48, Immunos2 algoritmaları bu veri kümesi için iyi sonuçlar sunmuştur. Bu deneylerde, Immunos2 algoritmasının performansı AIRS2 algoritmasından daha yüksektir. Çizelge 6.15'e göre Immunos2 algoritması bu veri kümesinde 0.51 AUC değerine sahip iken, Çizelge 6.21'e göre Immunos2 algoritması farklı metriklerle bu veri kümesinde 0.72 AUC değerine sahip olmuştur. Sınıf seviyesindeki metriklerin, Immunos2 algoritmasının performansını arttırdığını söyleyebiliriz. Benzer şekilde AIRS2 Paralel algoritmasının performansını da artmıştır. Ancak RF ve Naive Bayes algoritmalarının performansı sınıf seviyesindeki metriklerle artmamıştır. Dördüncü hipotezimiz geçerlidir ve dördüncü araştırma sorusuna yanıtımız RF algoritmasıdır. Immunos2 algoritması sınıf seviyesindeki metrikler kullanıldığı zaman YBS tabanlı en iyi kusur kestirim algoritması özelliğindedir. Bu sonucu genelleştirebilmek için farklı veri kümelerinde de test etmek gerekir ancak henüz bu kapsamda kullanılabilecek KC1 dışında açık bir veri kümesi yoktur. Ayrıca, YBS tabanlı algoritmaların sınıf seviyesindeki metrikler kullanıldığında, daha iyi performans verdiği sonucunu çıkarabiliriz. Metot seviyesindeki metriklerle yapılan kestirimin metot bazında kusurları kestirdiğini düşünürsek, performansların artmasını kolaylıkla açıklayabiliriz.

Çizelge 6.19 Deney #3 için algoritmaların AUC değerleri

Algoritmalar	**KC1**	**KC2**	**PC1**	**CM1**	**JM1**
J48	0.70	0.80	0.69	0.51	0.66
RandomForests	0.79	0.78	**0.81**	0.65	**0.71**
NaiveBayes	**0.80**	**0.84**	0.75	**0.76**	0.67
Immunos1	0.68	0.67	0.70	0.70	0.60
Immunos2	0.49	0.52	0.50	0.50	0.50
CLONALG	0.52	0.62	0.50	0.50	0.53
AIRS1	0.60	0.65	0.58	0.54	0.54
AIRS2	0.60	0.68	0.58	0.52	0.56
AIRS2 Paralel	0.60	0.66	0.58	0.52	0.56

Çizelge 6.20 Deney #3 için algoritmaların doğruluk değerleri

Algoritmalar	**KC1**	**KC2**	**PC1**	**CM1**	**JM1**
J48	84.50	84.60	93.42	89.22	80.94
RandomForests	85.04	80.83	93.72	87.97	80.02
NaiveBayes	82.40	84.25	89.17	86.32	80.43
Immunos1	50.05	50.58	59.17	69.61	60.16
Immunos2	80.24	69.05	93.06	90.16	80.67
CLONALG	82.42	80.48	91.61	88.56	76.62
AIRS1	75.85	74.83	89.03	82.85	64.32
AIRS2	77.16	77.12	90.49	84.70	72.03
AIRS2 Paralel	78.67	75.22	89.84	85.32	70.29

Çizelge 6.21 Deney #4 için algoritmaların AUC değerleri

Algoritmalar	**KC1-sınıf-seviyesi**
J48	0.75
RandomForests	0.79
NaiveBayes	0.76
Immunos1	0.69
Immunos2	0.72
CLONALG	0.67
AIRS1	0.70
AIRS2	0.71
AIRS2 Paralel	0.68

Çizelge 6.22 Deney #4 için algoritmaların doğruluk değerleri

Algoritmalar	**KC1-sınıf-seviyesi**
J48	68.78
RandomForests	71.08
NaiveBayes	69.55
Immunos1	63.50
Immunos2	70.49
CLONALG	68.65
AIRS1	71.10
AIRS2	71.90
AIRS2 Paralel	70.15

6.3.4.5 Deney #5

Bu deneyde, 94 sınıf seviyesindeki metrik üzerinde cfs tekniği kullanılmış ve ilişkili metrikler elde edilmiştir. 94 metrik üzerinde cfs uygulandıktan sonra; *CBO, maxLOC_COMMENTS, maxHALSTEAD_CONTENT, maxHALSTEAD_VOLUME, maxLOC_TOTAL, avgHALSTEAD_LEVEL, avgLOC_TOTAL* metrikleri elde edilmiştir ve deneylerde sadece bu metrikler kullanılmıştır. Çizelge 6.23 algoritmaların AUC değerlerini gösterirken Çizelge 6.24 doğruluk değerlerini göstermektedir. Çizelge 6.23'e göre, Naive Bayes algoritması en iyi performansı sunmuştur. Ayrıca, J48, RF ve Immunos2 algoritmaları yüksek performans

sağlamıştır. Immunos2 algoritmasının performansı AIRS2 algoritmasından daha yüksektir. Beşinci hipotezimiz geçerli değildir ve beşinci araştırma sorusuna yanıtımız Naive Bayes algoritmasıdır. Immunos2 algoritması, YBS tabanlı en iyi kusur kestirim algoritması özelliğindedir ancak bu sonucu genelleştirmek için farklı veri kümelerinde de test etmek gerekir. Bu deneyde Naive Bayes algoritmasının AUC değeri 0.82'dir ve tüm deneyler için en yüksek değer olma özelliği taşır.

Çizelge 6.23 Deney #5 için algoritmaların AUC değerleri

Algoritmalar	**KC1-sınıf-seviyesi**
J48	0.74
RandomForests	0.79
NaiveBayes	0.82
Immunos1	0.69
Immunos2	0.74
CLONALG	0.69
AIRS1	0.71
AIRS2	0.72
AIRS2 Paralel	0.71

Çizelge 6.24 Deney #5 için algoritmaların doğruluk değerleri

Algoritmalar	**KC1-sınıf-seviyesi**
J48	69.99
RandomForests	70.54
NaiveBayes	71.27
Immunos1	63.79
Immunos2	70.91
CLONALG	69.25
AIRS1	70.84
AIRS2	72.83
AIRS2 Paralel	71.98

6.3.4.6 Deney #6

Bu deneyde, 94 adet metrik kullanılmıştır. Çizelge 6.25 algoritmaların AUC değerlerini, Çizelge 6.26 doğruluk değerlerini göstermektedir. Çizelge 7.25'e göre, 94 adet sınıf seviyesindeki metrikle en iyi sonucu Naive Bayes algoritması sunmuştur. Altıncı hipotezimiz geçerli değildir ve altıncı araştırma sorusuna yanıtımız Naive Bayes algoritmasıdır. YBS tabanlı algoritmaların performansının, cfs uygulandığı zaman arttığı saptanmıştır. 94 metot seviyesindeki metrik kullanıldığı zaman en iyi performansı sunan YBS tabanlı algoritma Immunos2'dir.

Çizelge 6.25 Deney #6 için algoritmaların AUC değerleri

Algoritmalar	**KC1-sınıf-seviyesi**
J48	0.70
RandomForests	0.80
NaiveBayes	0.81
Immunos1	0.67
Immunos2	0.69
CLONALG	0.68
AIRS1	0.64
AIRS2	0.64
AIRS2 Paralel	0.64

Çizelge 6.26 Deney #6 için algoritmaların doğruluk değerleri

Algoritmalar	**KC1-sınıf-seviyesi**
J48	67.70
RandomForests	71.58
NaiveBayes	68.59
Immunos1	61.17
Immunos2	66.59
CLONALG	69.42
AIRS1	65.15
AIRS2	65.15
AIRS2 Paralel	65.66

6.3.4.7 Deney #7

Bu deneyde, çapraz geçerleme kullanılmamıştır. Veri kümesinin %20'si eğitim, geri kalanı test kümesi olarak kullanılmıştır. 13 adet metot seviyesindeki metrik bu çalışmada kullanılmış, türetilmiş metrikler yer almamıştır.

Çizelge 6.27 algoritmaların AUC değerlerini, Çizelge 6.28 doğruluk değerlerini göstermektedir. Çizelge 6.27'ye göre, çapraz geçerleme uygulanmadığı zaman yine en iyi performansın RF ile alındığı tespit edilmiştir. RF algoritması PC1 ve JM1 veri kümelerinde en iyi performansı sunar.

Yedinci hipotezimiz geçerlidir ve yedinci araştırma sorusuna yanıtımız RF algoritmasıdır. AIRS1 algoritması çapraz geçerleme uygulanmadığı zaman en iyi YBS tabanlı kusur kestirim algoritmasıdır. RF algoritmasının büyük veri kümelerinde, Naive Bayes algoritmasının daha küçük veri kümelerinde en iyi performansı sunduğu sonucunu çıkarabiliriz. Ayrıca, algoritmaların performansı önemli ölçülerde değişmemiştir.

Çizelge 6.27 Deney #7 için algoritmaların AUC değerleri

Algoritmalar	**KC1**	**KC2**	**PC1**	**CM1**	**JM1**
J48	0.64	0.70	0.57	0.55	0.63
RandomForests	0.74	0.78	0.72	0.66	0.68
NaiveBayes	0.79	0.84	0.68	0.73	0.67
Immunos1	0.70	0.72	0.60	0.63	0.62
Immunos2	0.50	0.55	0.50	0.50	0.50
CLONALG	0.53	0.61	0.51	0.51	0.51
AIRS1	0.59	0.67	0.55	0.56	0.56
AIRS2	0.58	0.65	0.54	0.53	0.55
AIRS2 Paralel	0.58	0.67	0.53	0.56	0.55

Çizelge 6.28 Deney #7 için algoritmaların doğruluk değerleri

Algoritmalar	**KC1**	**KC2**	**PC1**	**CM1**	**JM1**
J48	83.96	83.31	92.50	87.24	80.22
RandomForests	84.20	82.56	93.04	88.52	79.76
NaiveBayes	82.69	83.98	86.96	**83.22**	80.39
Immunos1	55.27	61.29	39.63	40.01	64.05
Immunos2	84.55	80.19	93.04	90.18	80.67
CLONALG	79.89	78.28	92.31	87.42	67.80
AIRS1	78.05	77.80	86.49	79.88	72.33
AIRS2	79.36	80.07	90.93	80.94	71.49
AIRS2 Paralel	80.15	80.84	90.59	85.68	74.35

6.3.5 Özet ve Gelecek Çalışmalar

Bu çalışmada, yüksek performanslı kusur kestirim algoritmaları ve YBS tabanlı algoritmalar farklı metrik grupları ve teknikleriyle NASA veri kümelerinde incelenmiştir. Yedi araştırma sorusuna yanıt verebilmek için yedi farklı deney grubu saptanmıştır ve dokuz algoritma beş veri kümesinde analiz edilmiştir. Üç test grubu sadece bir veri kümesinde test edilebilmiştir. Bunun sebebi de sadece bir veri kümesinde sınıf seviyesinde metriklerin yer almasıdır.

Yaptığımız literatür taramasına göre; bu çalışma, yüksek performanslı kusur kestirim algoritmalarının ve YBS tabanlı algoritmaların farklı metrik kombinasyonlarıyla incelendiği ilk çalışmadır.

Gerçekleştirdiğimiz deneylere göre, hipotez 1, 2, 3, 4, 7 geçerli iken 5 ve 6 geçerli değildir. RF algoritması çoğu durumda büyük veri kümelerinde en iyi performansı sunarken, Naive Bayes algoritması küçük veri kümelerinde RF algoritmasından daha iyi sonuç vermiştir. Metot seviyesindeki metrikler kullanıldığı durumda AIRS2Paralel algoritması en iyi YBS tabanlı kusur kestirim algoritmasıdır. Sınıf seviyesindeki metrikler kullanıldığı zaman, YBS tabanlı en iyi kusur kestirim algoritması Immunos2'dir. Bu çalışma, yazılım kusur kestiriminde en önemli bileşenin kullanılan algoritmalar olduğunu, yazılım metrikleri olmadığını göstermiştir.

7. YARI-EĞİTİCİLİ YAZILIM KUSUR KESTİRİMİ

Bu bölümde, sınırlı sayıda kusur verisiyle yüksek performanslı yazılım kusur kestirimini sağlamak üzere gerçekleştirilen çalışmalar ve geliştirilen yöntemler açıklanmaktadır.

7.1 Özet

Bu araştırma; eğiticili kusur kestiriminde yüksek performans sunduğu raporlanmış olan J48, Rastgele Orman, Naive Bayes, KStar, ve Yapay Bağışıklık Tanıma Sistemleri (AIRS) algoritmaları ile sınırlı sayıda kusur verisiyle yazılım kusur kestirimi gerçekleştirilmesi üzerine odaklanmış, karşılaştırmalar ve incelemeler ağırlıklı olarak bu temel algoritmalar bağlamında gerçekleştirilmiştir. Aynı zamanda, yeni bir yarı-eğiticili sınıflandırma algoritması olan YATSI (Yet Another Two Stage Idea) incelenerek bu algoritmanın ilk aşamasında yukarıda ifade edilen yüksek performanslı sınıflayıcılar çalışılmıştır. YATSI algoritması, ilk aşamasında farklı algoritmaların uygulanmasına izin veren meta bir algoritmadır. Bu çalışmalar kapsamında, literatürde ilk kez Yapay Bağışıklık Sistem paradigmasını kullanan yarı-eğiticili sınıflandırma algoritması önerilmiş ve performansı NASA açık veri kümelerinde değerlendirilmiştir. Deneysel çalışmalarımız, etiketsiz verilerin etiketli verilere yardımcı olmak üzere kullanıldığı durumda, YATSI algoritmasının Naive Bayes algoritmasının performansını her zaman iyileştiremediğini göstermiştir. Gerçekleştirdiğimiz deneylere göre, Naive Bayes algoritması küçük veri kümelerinde yarı-eğiticili kusur kestirim modeli oluşturmak için en iyi seçimdir ve YATSI algoritması büyük veri kümelerinde Naive Bayes algoritmasının performansını iyileştirmektedir. Ayrıca, YATSI algoritması tüm veri kümelerinde Naive Bayes dışında tüm sınıflayıcıların performansını arttırmıştır. Yapay Bağışıklık Sistem paradigmasının kullanıldığı YATSI tabanlı AIRS algoritmamızın, etiketsiz verilerle performansının iyileşmesi sebebiyle ileriki dönemlerde algoritmada yapılabilecek iyileştirmelerin yarı-eğiticili kusur kestirimi kapsamında yarar sağlayacağı değerlendirilmektedir.

7.2 Giriş

Makine öğrenmesi içerisinde eğiticili sınıflandırma yaklaşımları, sınıflandırma modeli oluşturmak için verilere ilişkin özellikleri (features) ve sınıf etiketlerini kullanmaktadır. Ancak sınıf etiketlerinin toplanması; konuşma tanıma, istenmeyen e-posta tespiti, medikal teşhis gibi gerçek dünya problemleri için zor, pahalı ve zaman alıcı bir süreçtir. Bu problemi çözmenin bir yolu, öğrenme süreci sırasında etiketsiz veriler ve etiketli verilerden

yararlanabilecek sınıflandırma algoritmalarını geliştirmektir. Bu tür algoritmalar, makine öğrenmesi içerisinde "yarı-eğiticili sınıflandırma yaklaşımları" olarak adlandırılmaktadır.

Araştırmacılar, modelleme için yeterince etiketli veri olmadığı durumda etiketli verilerle birlikte etiketsiz verilerden de yararlanmak için oldukça iyimserdir. Problem alanı için etkin bir yarı-eğiticili sınıflandırma modeli oluşturmak, düşünüldüğünden çok daha zordur. Birçok araştırmacı (Miller ve Uyar, 1996; Goldman ve Zhou, 2000) etiketsiz veriler kullanıldığı durumda; sınıflayıcıların performansında iyileşmeler sağlandığını raporladığından, makine öğrenmesi topluluğunda mevcut eğilim, etiketsiz verilerden sınıflayıcılar için mümkün olan her zaman yararlanma şeklindedir. Etiketsiz veriler kullanıldığı zaman, sınıflayıcı performansında azalmaları gösteren empirik kanıtlar mevcut olduğu için, etiketsiz verileri kullanma konusunda bu iyimser yaklaşımın değişmesi gerekmektedir. Nigam ve arkadaşları (2000b), verinin model varsayımlarına uymaması durumunda etiketsiz veriler kullanmanın sınıflayıcı performansını azaltabileceğini raporlamışlardır. Ayrıca; Elworthy (1994), Saklı Markov Model (Hidden Markov Model) performansının etiketsiz verilerle bazı durumlarda kötüleşebileceğini göstermiştir.

Bu çalışmada, YATSI (Yet Another Two Stage Classifier) yarı-eğiticili sınıflandırma algoritmasının yüksek performanslı sınıflayıcılara bağlı olarak kusur kestirimi açısından performans değişimi incelenmiş ve dengesiz veri kümelerinde, etiketsiz veri yüzdelerinin farklı değerleriyle analizler gerçekleştirilmiştir. Driessens ve arkadaşları (Driessens vd., 2006) YATSI algoritmasını önermiş ve doğruluk parametresini kullanarak çeşitli sınıflayıcıların performanslarını YATSI ile birlikte kullanılması durumunda değerlendirmişlerdir. Çalışmalarında, YATSI algoritmasının Rastgele Orman algoritması dışında tüm sınıflayıcıların performansını iyileştirebileceklerini ortaya koymuşlardır. Ancak bu çalışmaları dengesiz veri kümelerine odaklanmamış ve buna bağlı olarak dengesiz veri kümeleri özelinde kullanılması gereken değerlendirme parametrelerinden yararlanmamışlardır. Yazılım kusur kestirim çalışmaları, dengesiz veri kümelerinden oluştuğu için bu çalışmalarımızda doğruluk parametresi değerlendirme krititeri olarak kullanılmamıştır.

Bu çalışmalarımız sınırlı sayıda kusur verisiyle yazılım kusur kestirimi problemine odaklanmıştır. Bu problemde, etiketsiz verilerin herhangi bir kusur etiketine sahip olmadığı düşünülmektedir. Çevrimsel karmaşıklık veya kod satır sayısı gibi yazılım metrikleri eğitim kümesinin özellikleridir ve kusur etiketi (kusur-eğilimli ya da kusur-eğilimli değil) bu problem için sınıf etiketidir. Eğitim kümesi içerisinde, Çizelge 7. 2'de sunulan 21 adet özellik (metrik) her bir yazılım modülü (metot ya da prosedür) için yer almaktadır.

Makine öğrenmesi içerisindeki eğiticili sınıflandırma algoritmaları, bir önceki yazılım metrikleri ve kusur verisiyle kestirim modelleri oluşturmak üzere kullanılabilmektedir. Ancak bazı durumlarda gürbüz modeller oluşturmak için yeterli kusur etiketi mevcut olmayabilir. Örneğin, bazı proje ortakları bazı proje bileşenleri ya da tamamı için kusur verisini toplamayabilir ya da tüm sistem üzerinde yazılım metrik toplama araçlarının çalıştırılma maliyeti oldukça pahalı olabilir (Seliya vd., 2004). Bu durumlarda, sınırlı sayıda kusur verisiyle gürbüz kusur kestirimi sağlayabilecek sınıflandırma algoritmalarına ihtiyaç duyarız veya etiketsiz verileri kullanabilen güçlü yarı-eğiticili sınıflandırma algoritmalarından yararlanabiliriz. Seliya ve arkadaşları (2004) Beklenti-Maksimizasyonu (Expectation-Maximization) algoritmasını sınırlı sayıda kusur verisiyle kusur kestirimi için kullanmıştır ve bu çalışma, bu kapsamda literatürde sunulmuş ilk yaklaşımdır.

Bu araştırma 5 sınıflayıcıyı ve bu sınıflayıcıların YATSI algoritmasının ilk adımında kullanıldığı sınıflayıcıları karşılaştırmaktadır. Bu sayede, 10 farklı algoritma ROC eğrisi altında kalan alan (AUC-Area Under ROC Curve) değerleri kullanılarak karşılaştırılmıştır. ROC eğrisi, x ekseninde yanlış alarm olasılığını (probability of false alarm-PF) gösterirken y ekseninde tespit olasılığını (probability of detection-PD) göstermektedir. Doğruluk ve hassasiyet parametreleri değerlendirme için kullanılmamıştır. Bunun sebebi de dengesiz veri kümelerinde bu parametrelerin kullanılmasına şiddetle karşı çıkılmasıdır (Menzies vd., 2007a). Bradley (1997), AUC parametresini çeşitli makine öğrenmesi algoritmalarını kıyaslamak için kullanmış ve AUC parametresinin doğruluk parametresinden daha iyi özelliklerinin olduğunu göstermiştir. Ling ve arkadaşları (2003) sınıflama sistemlerini kıyaslarken AUC değerinin kullanılmasını önermiş ve AUC parametresinin dengesiz veya dengeli veri kümelerinde doğruluktan istatistiksel olarak daha uygun olduğunu göstermişlerdir.

AUC değeri 0 ve 1 arasında değer alabilmektedir. AUC değeri 1 olan sınıflayıcı mükemmel sınıflayıcı olarak adlandırılır ve AUC değeri 0.50 olan sınıflayıcının performansının rastgele tahmin (random guessing) ile aynı olduğu ifade edilir (Guo vd., 2004). Genellikle yazılım kusur veri kümeleri dengesizdir ve kusur eğilimli modüller tüm veri kümesinin genellikle %20'sini oluşturur. Bu çalışmada kullanılan JM1, KC2, PC1 ve CM1 veri kümeleri PROMISE havuzundan alınarak kullanılmıştır (Sayyad ve Menzies, 2005). Çalışmada, Rastgele Orman (Ma vd., 2006), J48 (Koru ve Liu, 2005a), Naive Bayes (Menzies vd., 2007a), KStar (Koru ve Liu, 2005a) ve AIRS (Çatal ve Diri, 2007) algoritmaları eğiticili kusur kestiriminde sundukları yüksek performans sebebiyle tercih edilmiştir.

KStar, örnek-tabanlı (instance-based) öğrenicidir ve entropi tabanlı uzaklık fonksiyonu kullanır (Cleary ve Trigg, 1995). Rastgele Orman algoritması onlarca ya da yüzlerce ağaca başvurur ve sınıflama için bu ağaçların sonuçlarını kullanır. Bu çalışmada, Breiman'in (Breiman, 2001) Rastgele Orman gerçeklemesi, Weka açık kaynaklı makine öğrenme aracı içerisinden kullanılmıştır. J48, Quinlan'ın (Quinlan, 1993) C4.5 algoritması tabanlı olup, bir karar ağacı gerçeklemesidir. Naive Bayes, olasılıksal sınıflandırma algoritmasıdır ve güçlü bağımsızlık (independence) varsayımları vardır (John ve Langley, 1995). Watkins (2001) tarafından geliştirilen AIRS algoritması, bağışıklık sisteminden esinlenen yüksek performanslı bir sınıflandırma algoritmasıdır ve 5 adımdan oluşur: ilklendirme, antijen eğitimi, sınırlı kaynak için yarışma, bellek hücresi seçimi, sınıflandırma. YATSI, 2 adımlı yarı-eğiticili sınıflandırma algoritmasıdır (Driessens vd., 2006). Bu algoritmanın detayları takip eden bölümlerde verilmektedir.

Yaptığımız araştırmalara göre, bu çalışma çeşitli yüksek performanslı sınıflayıcıların yarı-eğiticili kusur kestirimi bağlamında incelendiği ilk çalışmadır.

7.3 Yarı-Eğiticili Öğrenme

Makine öğrenmesi içerisindeki eğiticili ve eğiticisiz öğrenme yöntemleri, sadece geliştirilmiş ileri yöntemler bağlamında değil, aynı zamanda sağladıkları açık yararlar açısından da iyi çalışılmış konulardır. Eğiticili öğrenme yöntemleri, girdi-çıktı ilişkisini öğrenmek üzere girdileri ve ilişkili etiketleri içeren eğitim kümesini kullanır. Eğiticisiz öğrenme yöntemleri ise sınıf etiketlerini kullanmaz. Eğiticisiz öğrenme yaklaşımlarını, kümeleme ve bileşen analizi şeklinde iki gruba ayırabiliriz (Huang ve Kecman, 2005). Sınıf etiketlerinin toplanması zaman alıcı ve pahalı bir süreçtir. Bunun sebebi de bu sürecin, kullanılacak her veri noktasının bir insan tarafından etiketlenmesini gerektirmesidir (Zhu, 2005). Konuşma tanımada, konuşmanın kayıt edilmesi çok ucuzdur ancak çok sayıda konuşma verisinin etiketlenmesi aşaması, bir insanın tüm kayıt edilmiş veriyi dinlemesine ihiyaç duyar (Chapelle vd., 2006). Sınıflandırma ve kümeleme için etiketli verilerin yanı sıra etiketsiz verilerden de yararlanabilseydik, etiketleme aşamasında gereksiz zaman harcamayı önleyebilirdik. Yarı-eğiticili öğrenme alanı, etiketli ve etiketsiz verilerden yararlanarak öğrenme sağlayabilen bu tür algoritmaların tasarlanması, incelenmesi ve kıyaslanması ile ilgilenmektedir. Genetik araştırmalar, medikal teşhis, istenmeyen elektronik posta tespiti, biyoinformatik, bilgisayarlı görü (computer vision), yarı-eğiticili öğrenme algoritmalarına ihtiyaç duyulan bazı alanlardır.

Yarı-eğiticili öğrenme alanı iki gruba ayrılabilmektedir: yarı-eğiticili sınıflandırma ve yarı-

eğiticili kümeleme. İlk grupta amaç veri noktalarının sınıflandırılması iken ikinci grupta amaç uyumlu (coherent) kümelerin oluşturulmasıdır. Çalışmalarımızda ilk grup konusunda çalışıldığı için ikinci grup konusunda detaylı bilgi sunulmayacaktır. Geleneksel makine öğrenmesi tabanlı sınıflayıcılar etiketsiz verilerden yararlanamaz. Bunun sebebi de bu amaçla tasarlanmamış olmalarıdır. Bu nedenle etiketsiz verilerden de yararlanabilecek algoritmalara ihtiyaç duymaktayız. Aslında yarı-eğiticili öğrenme kavramı çok yeni bir fikir değildir. 1960'larda öz-öğrenme (self-training) yaklaşımları önerilmişti (Scudder, 1965; Fralick, 1967; Agrawala, 1970) ve bu yaklaşımlar etiketsiz veriden yararlanan ilk yöntemler olarak kabul edilmektedir.

Bazı araştırmacılar yeterli etiketli veri olmadığı durumda yarı-eğiticili sınıflandırma algoritmalarını kullanma noktasında oldukça iyimserdirler (Cozman vd., 2003) ancak güçlü bir yarı-eğiticili model tasarlamak ve oluşturmak çok fazla zaman almaktadır. Ayrıca, sınıflandırma için etiketli verilerle birlikte etiketsiz verilerden de yararlanmak sınıflayıcı performansını arttırmayı garanti etmez. IJCAI 2001, NIPS 1998, NIPS 1999, NIPS 2000 çalıştaylarında birçok yayın (Baluja, 1998; Nigam vd., 2000b; Miller ve Uyar, 1996; Goldman ve Zhou, 2000; Bennett ve Demiriz, 1999), etiketsiz veriler kullanıldığı durumda sınıflayıcıların performansının arttığını raporlamıştır (Cozman vd., 2003).

Bu yayınlara rağmen, etiketsiz veriler kullanıldığı durumda sınıflayıcıların performansının da düşebildiğini gösteren bazı yayınlar mevcuttur (Cozman ve Cohen, 2002). Nigam ve arkadaşları (2000b), verinin model varsayımlarına uymaması durumunda etiketsiz veri kullanmanın sınıflayıcı performansında azalmaya yol açabileceğini raporlamışlardır. Baluja (1998) ve Shahshahani ve arkadaşları (1994) görüntü anlamada benzer şekilde performans azalmasını ortaya koymuşlardır. Bruce (2001), Bayes ağ sınıflayıcılarında (Bayesian network classifiers) performans azalmasını raporlamıştır. Elworthy (1994) etiketsiz verilerle birlikte Saklı Markov Modelinin bazı durumlarda performansının azalabileceğini saptamıştır. Bazı araştırmacılar, etiketsiz verilerin sadece sınıflayıcı performanslarının memnun edici düzeyde olmadığı durumda kullanılmasını önermektedir (Cozman vd., 2003). Vapnik ve Chervonenkis (1974) tarafından önerilen "transdüktif çıkarım" (transductice inference) yarı-eğiticili öğrenmeye oldukça benzerdir. Transdüktif çıkarım için genel karar kuralı oluşturulmaz ve sadece etiketsiz noktaların etiketleri kestirilir (Chapelle vd., 2006).

Yarı-eğiticili öğrenme algoritmalarını 5 gruba ayırabiliriz:

- *Öz-eğitim (self-training):* Bir sınıflayıcı eğitim için etiketli veri kümesinin küçük bir

bölümünü kullanır ve oluşturulan model etiketsiz verileri etiketlemek için kullanılır. Yeni etiketlenen bu verilerden en fazla güven duyulan veri noktaları, eğitim kümesine eklenir ve bu işlem yakınsama (convergence) sağlanana kadar devam eder (Zhu, 2005). Sözlük anlamı belirsizliği (word sense disambiguation) problemi için Yarowsky (1995) öz-eğitim yaklaşımını kullanmıştır. Bu problemin çözümünde, birden fazla anlamı olan sözcüklerin bağlamdaki anlamının saptanması hedeflenmektedir.

- *Birlikte-öğrenme (co-training):* Birlikte öğrenme Blum ve Mitchell tarafından (1998) önerilmiştir. Özellikler iki kümeye ayrılarak her sınıflandırma algoritması kümelerden birisiyle eğitilir. Her sınıflayıcı etiketsiz verinin etiketlerini kestirir ve en güvenilir veri noktaları diğer sınıflayıcıya öğretilir. Bu adımdan sonra, sınıflayıcılar yeniden eğitilir ve bu süreç tekrarlanır. Nigam ve Ghani (2000a), birlikte-öğrenme yaklaşımlarını "üretimsel karma modeller" (generative mixture models) ve EM algoritması ile kıyaslayarak birlikte-öğrenmenin uygulanabilirliğini ve etkinliğini incelemişlerdir. Birlikte-öğrenme yaklaşımlarından kazanç sağlanması isteniyorsa, özelliklerin şartlı bağımsız (conditional independent) olması gerekir.

- *Transdüktif Destek Vektör Makineleri (Transductive Support Vector Machines-TSVM):* Bennett ve Demiriz (1999) TSVM'leri ilk kez tam sayı programlama yöntemi (integer programming method) ile gerçeklemiştir. Joachims (1999), SVM-light TSVM adı verilen kombinasyonal transdüktif yaklaşımı gerçeklemiş ve bu ürün, bu alanda ilk kez profesyonelce kullanılabilecek yazılım haline gelmiştir (Zhu, 2005). Amaç; etiketsiz verileri etiketleyerek maksimum sınıra (maximum margin), mevcut olan etiketli ve yeni etiketlenmiş verilerle ulaşabilmektir (Zhu, 2005).

- *Graf tabanlı yöntemler (Graph-based methods):* Graf içerisinde, etiketsiz ve etiketli veri noktaları düğümler (nodes) olup ayrıtlar, örneklerin benzerliğini temsil eder (Zhu, 2005). Blum ve Chawla (2001), Zhu ve arkadaşları (2003) farklı graf tabanlı yöntemler önermişlerdir. Küme çekirdekleri (cluster kernels) ve Markov rastgele yürüyüşler (random walks) bazı graf tabanlı yöntemlerdendir. Bu yöntemlerin performansı, graf yapısına ve ayrıt ağırlıklarına bağlı olarak değişir.

- *Üretimsel modeller (Generative models):* Bu yaklaşımlar, diskriminant modellerle çelişmektedirler. Diskriminant modellerde tüm değişkenler doğrudan ölçülebilmektedir. Daha fazla bilgi, Zhu'nun (2005) çalışmasından elde edilebilir.

7.4 İlişkili Çalışmalar

Yazılım kusur kestirim çalışmaları çoğu durumda eğiticili öğrenme yaklaşımlarını kullanmaktadır ve bu yaklaşımlar Genetik Programlama, Yapay Sinir Ağları, Naive Bayes, Dempster-Shafer Ağları, Karar Ağaçları gibi yöntemlerden yararlanmaktadır.

Seliya ve arkadaşları (2004), sınırlı sayıda veriyle yazılım kusur kestirimi için EM algoritmasını kullanmışlardır. EM tabanlı yaklaşımlarının, sınıflandırma algoritmaları ile karşılaştırılabilir ölçüde iyi performans sunduğunu raporlamışlardır. Literatür taramamız sırasında ve sonrasında, Seliya ve arkadaşlarının yaptığı bu çalışma dışında yarı-eğiticili kusur kestirimi için herhangi bir çalışmaya rastlamadık.

Driessens ve arkadaşları (Driessens vd., 2006), yarı-eğiticili sınıflandırma algoritması olan YATSI algoritmasını geliştirmiş ve farklı veri kümelerinde incelemişlerdir. YATSI'nın etiketsiz verilerden yararlandığı durumda Rastgele Orman algoritması dışında diğer algoritmaların performansını arttırdığını göstermişlerdir. Çalışmalarında değerlendirme kriteri olarak doğruluk parametresini kullanmışlardır. Kusur kestirimindeki veri kümelerinin dengesiz olması nedeniyle, doğruluk yerine AUC parametresi kullanılabilmektedir. Driessens ve arkadaşları, çalışmalarında YATSI algoritmasının en yakın komşu parametre değerini 10'a sabitlemişlerdir. %1 etiketli veri kullanıldığı durumda, eğitim ve test örneklerinin ağırlıkları arasında büyük farklar olması sebebiyle en yakın komşu parametre değerini 20 ve 50 olarak da denemişlerdir. Ancak performans değişimi 10 değeri ile elde edilene göre çok az olmuştur (az iyileşme veya bazı durumlarda küçük kayıplar).

Ma ve arkadaşları (2006) ve Guo ve arkadaşları (2004), yaptıkları çalışmalarda Rastgele Orman algoritmasının daha iyi performans sunduğunu ortaya koymuşlardır. Koru ve arkadaşları (Koru ve Liu, 2005a), J48 ve KStar algoritmalarıyla KC2 veri kümesinde sınıf seviyesindeki metriklerle daha iyi performans aldıklarını raporlamışlardır. Menzies ve arkadaşları (2007a), yazılım kusur kestirimi problemi için Naive Bayes algoritmasının logNums filtresiyle birlikte kullanılmasını önermişlerdir.

Menzies ve arkadaşları (2008), 50 tane rastgele seçilmiş modülün (25 kusurlu ve 25 kusur içermeyen) kusur kestirimi için daha büyük veri kümeleri kadar bilgiyi verebildiğini ortaya koymuşlardır. 2007 yılında önerdikleri Naive Bayes algoritmasının yine en iyi performansı sunduğunu ifade etmişlerdir. Bu nedenle, karmaşık makine öğrenmesi algoritmalarına yoğunlaşmak yerine veri kümelerinin bilgi içeriğine yoğunlaşılmasının gerekliliğini vurgulamışlardır. Çalışmalarında, veri kümelerinin çok küçük örneklemleri etkin kusur

kestiricileri için yeterli olmuş ve büyük veri kümelerini yok saymak zararlı bir sonuç doğurmamıştır. Bu çalışmalarında, yarı-eğiticili kusur kestirimine odaklanmamalarına rağmen elde ettikleri sonuçlar bu problemle ilişkilidir. Eğer basit Naive Bayes ve çok küçük veri kümesi etkin kusur kestiriciler oluşturabiliyorsa, etiketli verilerle birlikte etiketsiz verilerin kullanıldığı durumda çok daha iyi kusur kestiriciler elde edilebilir. Bu nedenle yaptığımız çalışmalar, Menzies ve arkadaşlarının (2008) çalışmasından farklıdır çünkü bu çalışmalarımızda amacımız etiketsiz modüllerle kestirim modellerinin performansını iyileştirmektir.

Son yıllarda gerçekleştirilmiş bu çalışmalar nedeniyle, deneylerimizde J48, KStar, Naive Bayes, AIRS ve Rastgele Orman algoritmalarını sınırlı kusur verisiyle yazılım kusurlarını kestirmek üzere kullanmaya karar verdik. YATSI algoritmasının ilk adımında, bu algoritmaların her biri kullanılarak, algoritmaların performans değişimleri incelenmiştir.

7.5 Yapay Bağışıklık Sistem Tabanlı YATSI Algoritması

Bu çalışmada, yapay bağışıklık sistem tabanlı yarı-eğiticili sınıflandırma algoritması önerilmektedir. Bu algoritma, YATSI algoritmasının özelleştirilmiş hali olup, Yapay Bağışıklık Sistem paradigmasından yararlanmaktadır. Yazılım kusur kestirim probleminde modülleri kusur eğilimli ya da değil şeklinde 2 gruba ayırdığımız için, bu algoritma sadece 2 sınıf üzerinde çalışmaktadır. Önerdiğimiz algoritma için, ağırlıkların düzeltme faktörü (K) 1 olarak seçilmiştir. Algoritmada, en yakın komşu arama işlemi için KD-ağaçları (k-dimensional tree) kullanılmaktadır.

Geliştirilen algoritmanın ilk adımında, AIRS isimli yapay bağışıklık sistem paradigmasından esinlenen algoritma kullanılmaktadır. Bu algoritmanın detayları, önceki bölümlerde açıklandığı için tekrar açıklanmayacaktır. AIRS algoritması ve etiketli veriler kullanılarak model oluşturulduktan sonra, etiketsiz verilere bu model yardımıyla "ön-etiketler" (pre-labels) verilir. Bu aşamadan sonra, yeni etiketlenmiş verilere (ön-etiketlerin verildiği veriler) ve önceden etiketli bulunan verilere ağırlıklar atanır. Önceden etiketli bulunan verilere güven daha yüksek olduğu için, önceden etiketli verilerin ağırlıkları yeni etiketlilerden daha yüksek olmaktadır. Önceden etiketli verilerin ağırlıkları 1 olarak seçilirken, yeni etiketlenmiş noktaların ağırlıkları 1'den küçüktür. 1'den küçük olan bu değer, etiketli veri noktalarının sayısının etiketsiz veri noktaları sayısına bölünmesiyle elde edilir ve tüm yeni etiketlenmiş noktalar için bu değer kullanılır.

Algoritmanın 2. adımında, kestirime ihtiyaç duyan her veri noktasının k adet en yakın

komşusu belirlenir ve bu komşuların ağırlıkları her bir sınıf (kusur eğilimli ya da değil) için ayrı ayrı toplanır. En büyük ağırlığa sahip olan sınıf, kestirimi yapılmakta olan veri noktasının etiketini göstermektedir. Örneğin; 3 komşunun ağırlıkları sırasıyla 0.2 (kusur eğilimli modül), 0.2 (kusur eğilimli modül) ve 1.0 (kusur eğilimli olmayan modül) ise kestirime ihtiyaç duyan modülün etiketi "kusur eğilimli değil" olacaktır. Bunun sebebi de, 1 değerinin 0.4'den (0.2 + 0.2 = 0.4) büyük olması ve 1.0 ağırlıklı modülün kusur eğilimli olmamasıdır.

Gerçekleştirilen algoritmanın sözde kodu aşağıda verilmektedir:

YBS tabanlı YATSI algoritmasının sözde kodu (pseudo code)

Girdiler: *Etiketli veri D_l, etiketsiz veri D_u, en yakın komşu sayısı K, $N=|D_l|$, $M=|D_u|$, etiketsiz veri noktası d_u*

Aşama 1:

1.1 *D_l veri kümesini ve AIRS algoritmasını kullanarak M_l modelini oluştur*

1.2 *D_u içindeki veri noktalarına ön-etiket vermek için M_l modelini kullan*

1.3 *D_l içindeki veri noktalarına 1.0, D_u içindekilere N/M ağırlık değerini ata*

1.4 *D'yi oluşturmak üzere D_l ve D_u'yu birleştir*

Aşama 2: *D_u içerisindeki her veri noktası d_u için*

2.1 *d_u'ya en yakın K adet komşuyu ara*

2.2 *Kusur eğilimli K adet komşunun ağırlıklarını topla (W_{fp})*

2.3 *Kusur eğilimli olmayan K adet komşunun ağırlıklarını topla (W_{nfp})*

2.4 *d_u'nun etiketi olarak, en büyük ağırlığa (W_{fp} veya W_{nfp}) sahip olan sınıfın etiketini (kusur eğilimli ya da değil) ata*

7.6 Deneysel Çalışmalar

Bölüm 7.6.1'de, deneysel çalışmalarda kullanılan açık kaynaklı projeler ve açık veri kümeleri açıklanmaktadır. Bölüm 7.6.2, ilk vaka çalışmasını (case study) sunarken Bölüm 7.6.3'de ikinci vaka çalışması ortaya konulmaktadır. İki vaka çalışmasındaki temel fark, YATSI algoritmasının ilk adımında kullanılan Knn parametre değerinin farklılığıdır. İlk vaka çalışmasında Knn değeri 10 iken ikinci çalışmada 60 olarak belirlenmiştir. Bölüm 7.6.2'de

sadece iki farklı Knn değerine bağlı olarak deneylerin sunulmasının nedeni açıklanmaktadır.

7.6.1 Deney Ortamının Ayarlanması

MARSDEN projesi (www.cs.waikato.ac.nz/~fracpete/marsden) Java ile gerçeklenmiş yarı-eğiticili öğrenme algoritmalarını içermektedir ve YATSI bunlardan birisidir. WEKA projesi (www.cs.waikato.ac.nz/~ml/weka) Java dilinde gerçeklenmiş çeşitli sınıflayıcıları içermektedir ve J48, Naive Bayes, Rastgele Orman, KStar algoritmaları bu proje içerisinden kullanılmıştır. AIRS sınıflayıcısı, bir sourceforge projesi olan wekaclassalgos (http://wekaclassalgos.sourceforge.net) projesindeki gerçeklemesi ile deneylerde ele alınmıştır. Bu projelerde yer alan tüm kaynak kodlar, hazırlanan bir Eclipse projesine aktarılmış ve bu projeden bir jar dosyası oluşturulmuştur. AIRS algoritmasını, YATSI'nin ilk aşamasında kullanılabilmesi için kaynak kodlarda bazı değişiklikler yapılmıştır. AIRS 3. parti bir sınıflayıcı olduğundan, Capabilities çerçevesini içermemektedir. AIRS1 içerisindeki *getCapabilities* yöntemi gerçeklenerek eskisi ezilmiştir (override). NASA veri kümelerinden olan JM1, KC2, CM1, PC1 PROMISE havuzundan elde edilmiş ve deneylerde kullanılmıştır. JM1 veri kümesi 315000 (315K) kod satırı içerirken 10885 modüle sahiptir. Kusurlu metotların yüzdesi %19 olarak verilmekte olup, metotlar C programlama dili ile gerçeklenmiştir. Diğer veri kümelerinin detayları Çizelge 7.1'de verilmektedir. Veri kümelerinde yer alan modüllere ilişkin metrikler Çizelge 7.2'de sunulmaktadır.

Çizelge 7.1 Kullanılan veri kümeleri ve özellikleri

Veri Kümesi	Dil	Kod satır sayısı	Proje	% Kusur	Metot sayısı
JM1	C	315 K	Gerçek zaman	% 19	10885
PC1	C	40 K	Uçuş yazılımı	% 7	1109
KC2	C++	43 K	Veri işleme	% 21	523
CM1	C	20 K	Enstrüman	% 21	498

Çizelge 7.2 Veri kümelerindeki metrikler

Özellik	Bilgi
loc	McCabe kod satır sayısı
v(g)	McCabe çevrimsel karmaşıklık
ev(g)	McCabe temel karmaşıklık
iv(g)	McCabe tasarım karmaşıklığı
n	Halstead toplam operatör + toplam operand
v	Halstead hacim
l	Halstead program uzunluğu
d	Halstead zorluk
i	Halstead zekâ
e	Halstead çaba
b	Halstead teslim edilen hata (bug)
t	Halstead zaman kestiricisi
lOCode	Halstead kod satırı sayısı
lOComment	Halstead yorum satırı sayısı
lOBlank	Halstead boş satır sayısı
lOCodeAndComment	Yorum ve kod satır sayısı
uniq_Op	Eşsiz (unique) operator sayısı
uniq_Opnd	Eşsiz (unique) operand sayısı
total_Op	Toplam operatör
total_Opnd	Toplam operand
branchCount	Akış grafının dal sayısı

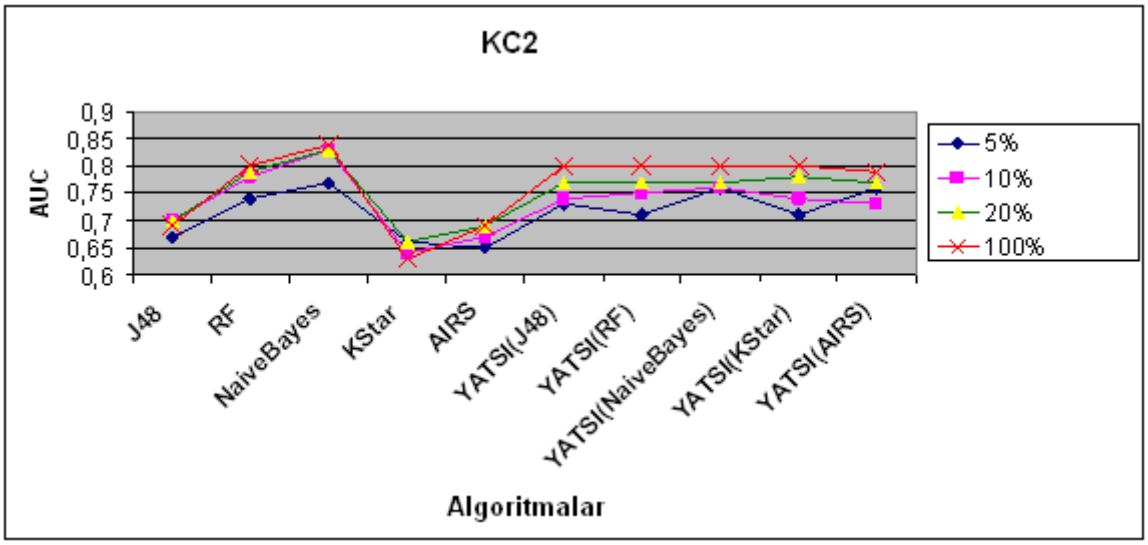
Deneyler sırasında, WEKA içerisinde yer alan Experimenter aracı kullanılmıştır. Datasets panelinden 4 veri kümesi, Algorithms panelinden 10 algoritma seçilmiştir. Deneylerde istatistiksel olarak doğru sonuçlar elde etmek için, deneyler 20 kez tekrarlanmıştır. Deney tipi, "Train/Test Percentage Split (data normalized)" olarak seçilmiş ve eğitim yüzdesi (Train percentage) olarak 5, 10, 20 değerleri kullanılmıştır. Bu değerler, veri kümesinin yüzde kaçının etiketli olarak algoritmada kullanılacağını göstermektedir.

7.6.2 Deneysel Sonuçlar (Vaka Çalışması #1)

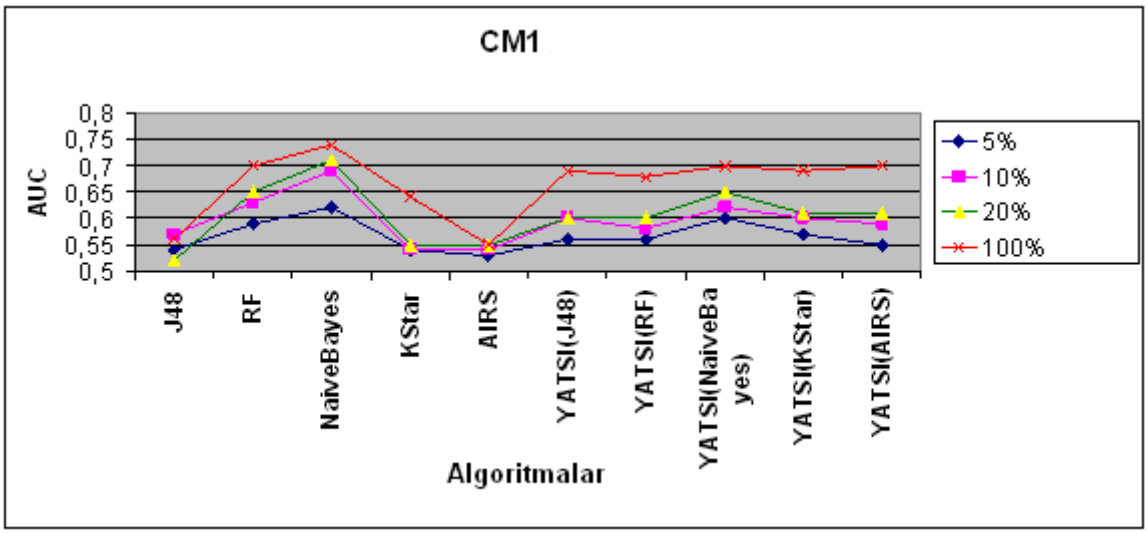
Az etiketli-çok etiketsiz veri probleminin benzetimini, %5, %10 ve %20 etiketli veri oranlarıyla gerçekleştirdik. Her sınıflayıcının performansını bu koşullar altında ayrı ayrı değerlendirdik. Bu oranlara göre; J48, RF, KStar, AIRS, Naive Bayes gibi eğiticili öğrenme algoritmaları için veri kümesinin sırasıyla %5, %10, ve %20'sinin eğitimde, geri kalanların testte kullanıldığı söylenebilir. Ancak bu oranlar, YATSI tabanlı yarı-eğiticili sınıflayıcılar için farklı bir anlam taşımaktadır. Bu tür sınıflayıcılar için veri kümesinin sırasıyla %5, %10 ve %20'sinin etiketli veri olarak kullanıldığı, geri kalanın eğitim sürecinde etiketsiz veri olarak kullanıldığı ifade edilebilir. Bu açıdan, eğiticili ve yarı-eğiticili sınıflandırma algoritmaları için oranlar farklı anlamlarda kullanılmaktadır.

Veri kümeleri üzerinde, Menzies ve arkadaşlarının (2007a) yaptığı logNum filtreleme gibi ön işleme yapılmamıştır. Çizelge 7.3, 7.4, 7.5, 7.6 her algoritmanın NASA veri kümeleri üzerinde farklı oranlara bağlı olarak sunduğu performansları göstermektedir. Bu deneylerde, YATSI algoritmasının Knn parametresi 10 olarak sabitlenmiştir. %100 sütunu, veri kümesinin tamamının eğitimde kullanıldığı durumu göstermektedir. %100 oran için testler 10-katlı çapraz geçerleme tekniğini kullanmıştır. Gerçekleştirilen testler 20 kez tekrarlanmış ve sınıflayıcı performansları AUC değerlerine göre kıyaslanmıştır.

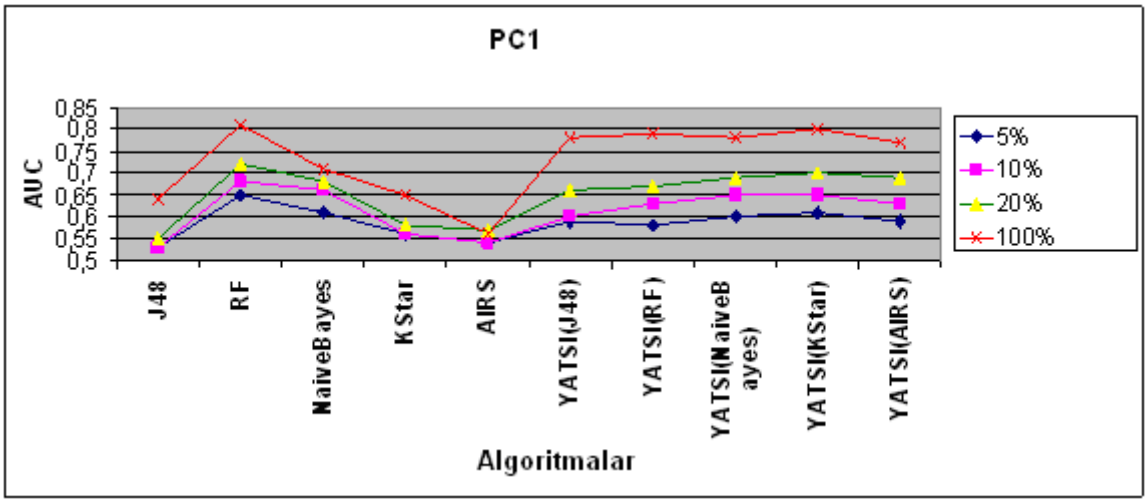
Şekil 8.1, 8.2, 8.3, 8.4; KC2, CM1, PC1, JM1 veri kümeleri üzerinde çeşitli algoritmaların farklı etiketli veri yüzdesine göre sunduğu performansları göstermektedir. Bu şekillerin temsil ettiği deneylerde, Knn parametresi 10 olarak seçilmiştir. Knn parametresi 10 ve 100 arasında 10'ar arttırılarak deneyler (10, 20, 30, 40, 50, 60, 70, 80, 90, 100) gerçekleştirilmiştir. 60 değerinden sonra, algoritmalarının performans değişiminin çok küçük olduğu saptanmıştır. Bu nedenle Vaka Çalışması #2'de Knn=60 ile elde edilen değerler sunulurken bu bölümde Knn=10 iken saptanan performanslar verilecektir. Knn değerinin arttırılması YATSI tabanlı yarı-eğiticili sınıflayıcıların performansını arttırmıştır.



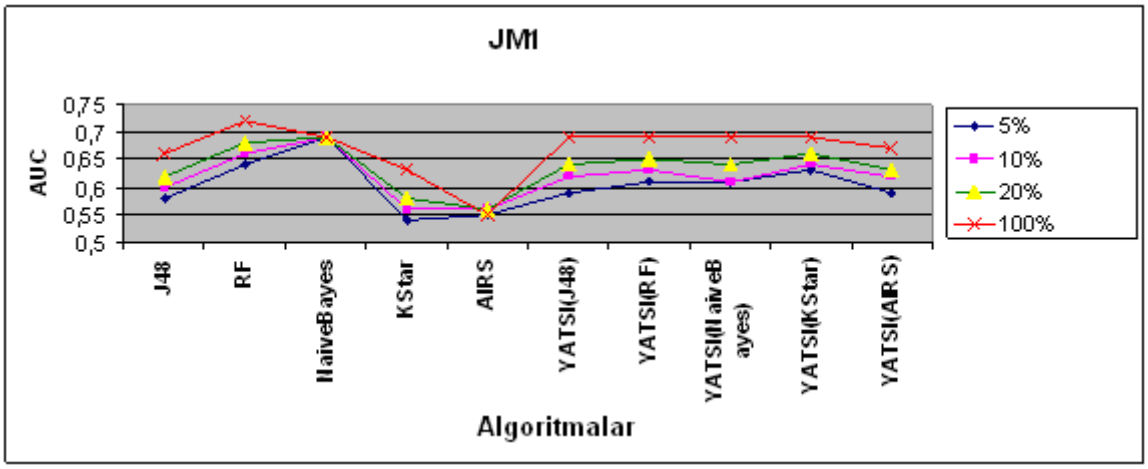
Şekil 7.1 KC2 veri kümesi üzerinde algoritmaların performansları



Şekil 7.2 CM1 veri kümesi üzerinde algoritmaların performansları



Şekil 7.3 PC1 veri kümesi üzerinde algoritmaların performansları



Şekil 7.4 JM1 veri kümesi üzerinde algoritmaların performansları

Vaka çalışması #1'deki deneysel çalışmalara göre, KC2, JM1 ve CM1 veri kümeleri için en iyi sınıflayıcı Naive Bayes algoritmasıdır çünkü Naive Bayes bu veri kümelerinde en yüksek AUC değerine sahip olmuştur. Bu empirik sonuç, Şekil 7.1, 7.2, 7.4'den kolaylıkla görülebilir. Bu şekillerde sunulan grafiklerde en yüksek noktalar, Naive Bayes algoritması ile elde edilmiştir. Rastgele Orman algoritması, PC1 veri kümesi için en iyi sınıflayıcı olarak çalışmaktadır. Bu empirik sonuç, Şekil 7.3'den görülebilir ve 7.3'deki en yüksek noktalar Rastgele Orman algoritmasının kullanıldığı deneylerde elde edilmiştir. Naive Bayes algoritmasının performansı, PC1 üzerinde de yüksektir ancak Rastgele Orman

algoritmasından daha iyi değildir. Vaka çalışması #1'e göre, yeterli kusur verisi olmadığı durumlarda Naive Bayes algoritmasını uygulamanın, ideal bir çözüm olabileceğini söyleyebiliriz.

JM1 veri kümesi deneylerde kullanılan en büyük veri kümesidir ve ikinci en büyük veri kümesi olan PC1 veri kümesinden yaklaşık olarak 10 kat büyüktür. Bu nedenle, yeterli kusur verisinin olmadığı çok geniş ölçekli yazılım geliştirme projeleri için yazılım kusur kestirimini Naive Bayes ile gerçekleştirmeyi savunuyoruz. Seliya ve arkadaşları (2004) JM1 veri kümesinin bazı uyumsuz (inconsistent) veriler içerdiğini raporlamıştır. YATSI tabanlı sınıflayıcıların performansının bu uyumsuz modüllerden etkilenmiş olabileceğinden şüphelenmekteyiz.

YATSI uygulandığı zaman Rastgele Orman ve Naive Bayes algoritmalarının performansının azaldığını gözlemledik. Örneğin; Rastgele Orman (RF) algoritmasının KC2 veri kümesi üzerinde %5 etiketli verilerle sunduğu AUC değeri 0.74 iken YATSI(RF) algoritmasının KC2 veri kümesi üzerinde %5 etiketli verilerle sunduğu AUC değeri 0.71'dir. AUC değerinin azalması sınıflayıcı performansındaki azalmayı ortaya koymaktadır. Deneysel çalışmalara göre, YATSI algoritması J48, AIRS, KStar algoritmalarının performansını tüm veri kümeleri üzerinde arttırmıştır. Yeterli kusur verisinin olmadığı durumlarda, bu algoritmaları YATSI algoritması dahilinde kullanarak bu algoritmaların performanslarını arttırabiliriz. Deneyler sırasında, tüm algoritmalar varsayılan (default) değerleri ile kullanılmıştır ve sınıflayıcıların performanslarını parametrelerine bağlı olarak optimize etmek mümkündür.

Tüm algoritmalar için daha fazla etiketli modül kullanmak, daha iyi sonuçlar elde etmeyi sağlamıştır. Örneğin; KC2 veri kümesi üzerinde %5 etiketli modüllerle RF algoritması çalıştırıldığında elde edilen AUC değeri 0.74 iken KC2 veri kümesi üzerinde %10 etiketli modüllerle RF algoritması çalıştırıldığında elde edilen AUC değeri 0.78 olmuştur. Şekil 7.1, 7.2, 7.3, 7.4 bu deneysel sonucu grafiksel olarak göstermektedir. %20 etiketli veri ile çizilen grafikler, tüm veri kümeleri için, %5 ve %10 etiketli veri ile çizilen grafiklerden daha üst konumlarda yer almaktadır. Bu nedenle, Naive Bayes algoritmasının %20 eğitim kümesi ve %80 test kümesi ile kullanılarak yarı-eğiticili kusur kestiriminin gerçekleştirilmesi en etkin çözümdür. Bunun yanı sıra, YATSI tabanlı sınıflayıcıların performansı daha fazla etiketli veri kullanıldığı durumda artmıştır. Ancak performans değişimi, gerçekleştirilen deneyler için çok fazla değildir ve bu durum Menzies ve arkadaşları (2008) tarafından gerçekleştirilen deneylerde de ifade edilmiştir. Menzies ve arkadaşları (2008), gerçekleştirdikleri mikro-örnekleme deneyinde, etiketli veri sayısını arttırmanın daima denge (balance) parametresini

arttırmadığını ortaya koymuşlardır. Çalışmalarında; algoritmaların performanslarını kıyaslamak için denge, tespit olasılığı (pd), yanlış alarm olasılığı (pf) parametrelerini kullanmışlardır. Çizelge 7.6'da, JM1 veri kümesinde daha fazla etiketli verinin Naive Bayes algoritması ile birlikte kullanılmasına rağmen performansın artmadığı görülmektedir. Bu nedenle, bu yaptığımız çalışma Menzies ve arkadaşlarının (2008) yaptığı çalışmayı desteklemektedir ve etiketli veri yüzdesini arttırmak daima sınıflayıcı performansını arttırmaz.

Tüm sınıflayıcılar en yüksek AUC değerlerine KC2 veri kümesinde ulaşmıştır. Bu empirik sonuç, KC2 veri kümesinin çok az gürültülü veri içerdiğini ya da hiç içermediğini ortaya koyabilir. Seliya ve arkadaşları (2004), JM1 veri kümesinin uyumsuz modüller (eş metriklere sahip ancak farklı etiketlerin mevcut olduğu modüller) içerdiğini raporlamış ve bu modülleri deneylerinden önce kümelerinden çıkarmışlardır ancak bu modüllerin gerçekten gürültülü olduğundan emin olamıyoruz. Bunun sebebi de, NASA kalite güvence grubunun bu kusurları ve metrikleri toplamış olması, potansiyel gürültülerle ilgili herhangi bir bilgiyi şimdiye kadar sunmamış olmasıdır. Bu açıdan, literatürdeki birçok çalışma gibi bu tür modülleri kümeden çıkarmayarak deneyleri gerçekleştirdik. Bu uyumsuz ve gürültülü modüller sebebiyle sınıflayıcıların performansları olumsuz yönde etkilenebilmektedir. Deneylerimizde en büyük veri kümesi JM1 olmasına rağmen, en iyi performanslar bu kümede elde edilmemiştir. Makine öğrenmesi algoritmalarının veri kümesi büyüdükçe performanslarında iyileşmeler olduğu bilgisi dikkate alınırsa, en büyük veri kümesinde en iyi performansların alınmaması bu veri kümesinde yer alan uyumsuz veya gürültülü modüllerle açıklanabilir. Seliya ve arkadaşları (2004), kestirim modelini JM1 veri kümesini eğitim kümesi olarak kullanarak oluşturmuş ve modeli KC2 veri kümesi ile test etmiştir.

CM1 ve KC2 veri kümeleri kabaca aynı sayıda modül içermesine rağmen, algoritmaların performansı KC2 üzerinde daha yüksektir. KC2 veri kümesi üzerinde, Naive Bayes algoritmasının performansı, %5, %10 ve %20 etiketli veri kullanıldığı durumlarda, CM1 veri kümesindeki performanslara göre %24, %20 ve %17 daha yüksektir. Ayrıca, KC2 projesinin kod satır sayısı CM1'in kod satır sayısının 2 katından daha fazladır. Bu gözlemler ışığında, Naive Bayes algoritmasının KC2 ve CM1 veri kümelerindeki performans farklılıklarını, metot başına düşen kod satır sayısının farklılığı ile açıklayabiliriz.

Rastgele Orman algoritması sadece PC1 veri kümesi üzerinde en iyi sınıflayıcı olarak gözlemlenmiştir ancak PC1 veri kümesinde bu algoritmanın performansı, KC2 üzerindeki performansından %10 daha düşüktür. KC2 kümesindeki kusurlu modüllerin yüzdesi (%21), PC1 kümesindeki kusurlu modüllerin yüzdesinden (%7) 3 kat daha fazladır ve yazılım

mühendisliği topluluğunda kusurlu modül yüzdeleri genellikle %20 olduğu bilinmektedir. PC1 veri kümesinde kusurlu modüllerin oranının az olmasının nedenini, gürültülü modüllerin mevcut olabileceği ihtimali veya bu projeyi gerçekleyen personelin daha fazla deneyimli olma olasılığı ile açıklayabiliriz. Doğrusu, bu düşük kusur yüzdesine yazılım projelerinde ulaşabilmek pek kolay görünmemektedir. Bu nedenle bu vaka çalışmamızda, diğer veri kümelerinde en iyi performansı elde ettiğimiz Naive Bayes algoritmasını yarı-eğiticili kusur kestiriminde kullanmak üzere öneriyoruz.

Ayrıca, YATSI(KStar) sınıflayıcı performansı, %20 etiketli verinin kullanıldığı durumda, CM1 veri kümesi dışında tüm veri kümelerinde diğer YATSI tabanlı sınıflayıcılardan daha yüksek performans sunmuştur. CM1 veri kümesinde ise YATSI(Naive Bayes) algoritması YATSI(KStar) algoritmasının performansından daha iyidir. Bu nedenle, yarı-eğiticili kusur kestiriminde YATSI tabanlı algoritmaları kullanmak isteyen bir araştırmacı ya da uygulayıcı için KStar ve Naive Bayes algoritmaları incelemesini öneriyoruz.

Çizelge 7.3 KC2 üzerinde AUC değerleri

Sınıflayıcı	**%5**	**%10**	**%20**	**%100**
J48	0,67	0,70	0,70	0,69
RF	0,74	0,78	0,79	0,80
Naive Bayes	**0,77**	**0,83**	**0,83**	**0,84**
KStar	0,66	0,64	0,66	0,63
AIRS	0,65	0,67	0,69	0,69
YATSI(J48)	0,73	0,74	0,77	0,80
YATSI(RF)	0,71	0,75	0,77	0,80
YATSI(NaiveBayes)	0,76	0,76	0,77	0,80
YATSI(KStar)	0,71	0,74	0,78	0,80
YATSI(AIRS)	0,76	0,73	0,77	0,79

Çizelge 7.4 CM1 üzerinde AUC değerleri

Sınıflayıcı	**%5**	**%10**	**%20**	**%100**
J48	0,54	0,57	0,52	0,56
RF	0,59	0,63	0,65	0,70
Naive Bayes	**0,62**	**0,69**	**0,71**	**0,74**
KStar	0,54	0,54	0,55	0,64
AIRS	0,53	0,54	0,55	0,55
YATSI(J48)	0,56	0,60	0,60	0,69
YATSI(RF)	0,56	0,58	0,60	0,68
YATSI(NaiveBayes)	0,60	0,62	0,65	0,70
YATSI(KStar)	0,57	0,60	0,61	0,69
YATSI(AIRS)	0,55	0,59	0,61	0,70

Çizelge 7.5 PC1 üzerinde AUC değerleri

Sınıflayıcı	**%5**	**%10**	**%20**	**%100**
J48	0,53	0,53	0,55	0,64
RF	**0,65**	**0,68**	**0,72**	**0,81**
Naive Bayes	0,61	0,66	0,68	0,71
KStar	0,56	0,56	0,58	0,65
AIRS	0,54	0,54	0,57	0,56
YATSI(J48)	0,59	0,60	0,66	0,78
YATSI(RF)	0,58	0,63	0,67	0,79
YATSI(NaiveBayes)	0,60	0,65	0,69	0,78
YATSI(KStar)	0,61	0,65	0,70	0,80
YATSI(AIRS)	0,59	0,63	0,69	0,77

Çizelge 7.6 JM1 üzerinde AUC değerleri

Sınıflayıcı	%5	%10	%20	%100
J48	0,58	0,60	0,62	0,66
RF	0,64	0,66	0,68	**0,72**
Naive Bayes	**0,69**	**0,69**	**0,69**	0,69
KStar	0,54	0,56	0,58	0,63
AIRS	0,55	0,56	0,56	0,55
YATSI(J48)	0,59	0,62	0,64	0,69
YATSI(RF)	0,61	0,63	0,65	0,69
YATSI(NaiveBayes)	0,61	0,61	0,64	0,69
YATSI(KStar)	0,63	0,64	0,66	0,69
YATSI(AIRS)	0,59	0,62	0,63	0,67

7.6.3 Deneysel Sonuçlar (Vaka Çalışması #2)

İlk vaka çalışmasının ardından, YATSI tabanlı algoritmaların Naive Bayes ve J48 gibi algoritmalara göre performansının kötü olduğu gözlemlenmiştir. Bu nedenle, YATSI algoritmasının iç parametrelerinin ve Knn parametresinin incelenmesine karar verilmiştir. İlk vaka çalışmasında, Driessens ve arkadaşlarının (Driessens vd., 2006) çalışmasında olduğu gibi Knn parametresi 10 olarak belirlenmişti. İkinci vaka çalışmamızda ise aynı deneysel ayarları kullanarak sadece Knn parametresinin değiştirilmesi yoluna gidildi. 10 ve 100 arasında, Knn parametresini 10'ar arttırarak algoritma performansının değişimi gözlemlendi. 60 değerinden sonra, performans iyileşmesi çok küçüktü ve bazı durumlarda bir önceki performans seviyesinin altına düşüldü. Knn parametresi YATSI algoritması için 60 olarak seçildiği durumda, 4 veri kümesinde her bir algoritma ile elde edilen AUC değerleri Çizelge 7.7, 7.8, 7.9 ve 7.10'da gösterilmektedir.

Vaka çalışması #2'deki deneylere göre, KC2, JM1 ve CM1 veri kümeleri için en iyi sınıflayıcı yeniden Naive Bayes olarak belirlenmiştir. YATSI uygulandığı zaman, KC2, JM1 ve CM1 veri kümeleri üzerinde Naive Bayes performansı azalmıştır. YATSI algoritması Naive Bayes algoritması dışında tüm algoritmaların performanslarını tüm veri kümelerinde arttırmıştır. İlk ve ikinci vaka çalışması arasındaki fark, RF algoritmasının performansının ikinci vaka çalışmasında iyileşmiş olmasıdır. Bu nedenle, Knn=60 değerinin RF algoritması için daha uygun olduğunu ve YATSI'nın Naive Bayes algoritması dışında tüm sınıflayıcıların

performansını arttırdığını söyleyebiliriz.

Büyük veri kümelerinde, etiketli veri yüzdesi arttıkça (%5, %10, %20) daha fazla etiketli veri yer alacağı için yarı-eğiticili sınıflayıcıların performansında iyileşme görüleceğini teorik olarak bekliyorduk. Bu teorik fikrimiz, PC1 veri kümesi için empirik olarak doğrulandı. PC1 veri kümesi için Knn=60 ve %20 etiketli veri kullanıldığı durumda, en iyi performansı YATSI tabanlı bir algoritma sunmuştur. YATSI(KStar) algoritması ile bu koşullarda, AUC değeri olarak 0,76 değeri elde edilmiş ve Naive Bayes algoritması da dahil olmak üzere tüm algoritmalardan daha yüksek bir değere ulaşılmıştır. PC1 veri kümesi üzerinde diğer YATSI tabanlı algoritmaların performansı da oldukça yüksektir. Naive Bayes algoritmasının performansı, bu veri kümesinde YATSI(KStar) algoritmasından iyi olmamasına rağmen, yine başarılı sonuç verdiği ifade edilebilir.

En büyük veri kümesi olan JM1'de YATSI'nın Naive Bayes algoritmasının performansını iyileştirmemesinin nedeni, JM1 içinde yer alan uyumsuz (eş metrikler fakat farklı etiketler) metriklerden kaynaklanıyor olabilir. Seliya ve arkadaşları (2004), bu modülleri deneylerden önce silmiş ve 10885 modül yerine sadece 8850 modül kullanmışlardır. NASA'nın bu metrikleri sunmuş olması nedeniyle, bu metriklerin gerçekten gürültülü ya da uyumsuz olduğundan emin olamıyoruz. Bu nedenle, çalışmalarımızda bu modülleri çıkartmadık. Eğer eğitim sırasında bu uyumsuz modüller seçildiyse, YATSI'nın Naive Bayes sınıflayıcısının performansını arttırmaması mümkün olabilir.

Ayrıca, ikinci vaka çalışmasındaki YATSI tabanlı sınıflayıcıların performansı ilk vaka çalışmasındaki YATSI tabanlı sınıflayıcılardan daha iyidir. Bu nedenle, yarı-eğiticili yazılım kusur kestirim çalışmalarında YATSI algoritmasının kullanılması düşünülürse Knn parametresinin farklı değerlerinin veri kümelerinde test edilmesini öneriyoruz.

Yapılan deneysel çalışmalara göre Naive Bayes algoritmasının yarı-eğiticili kusur kestiriminde en iyi performansı sunduğunu ancak büyük veri kümelerinde YATSI tabanlı bir yaklaşım ile performansının artabileceğini söyleyebiliriz. Ayrıca, ileriki dönemlerde Naive Bayes algoritması dışında daha farklı bir algoritmanın performansı Naive Bayes'den yüksek olursa, bu algoritmanın performansı YATSI ile iyileştirilir. Bu kapsamda, Knn parametresinin farklı değerlerini test etmek de yarar sağlayacaktır.

Çizelge 7.7 Vaka çalışması #2 için KC2 üzerinde AUC değerleri

Sınıflayıcı	Knn	%5	%10	%20
YATSI(J48)	60	0,78	0,80	0,82
YATSI(RF)	60	0,76	0,78	0,82
YATSI(NaiveBayes)	60	0,79	0,80	0,82
YATSI(KStar)	60	0,71	0,74	0,78
YATSI(AIRS)	60	0,71	0,73	0,77

Çizelge 7.8 Vaka çalışması #2 için CM1 üzerinde AUC değerleri

Sınıflayıcı	Knn	%5	%10	%20
YATSI(J48)	60	0,60	0,64	0,67
YATSI(RF)	60	0,60	0,63	0,65
YATSI(NaiveBayes)	60	0,63	0,65	0,68
YATSI(KStar)	60	0,62	0,63	0,66
YATSI(AIRS)	60	0,60	0,64	0,65

Çizelge 7.9 Vaka çalışması #2 için PC1 üzerinde AUC değerleri

Sınıflayıcı	Knn	%5	%10	%20
YATSI(J48)	60	0,64	0,69	0,74
YATSI(RF)	60	0,64	0,70	0,75
YATSI(NaiveBayes)	60	0,63	0,69	0,73
YATSI(KStar)	60	0,65	0,72	0,76
YATSI(AIRS)	60	0,64	0,69	0,74

Çizelge 7.10 Vaka çalışması #2 için JM1 üzerinde AUC değerleri

Sınıflayıcı	Knn	%5	%10	%20
YATSI(J48)	60	0,63	0,67	0,69
YATSI(RF)	60	0,63	0,66	0,69
YATSI(NaiveBayes)	60	0,64	0,66	0,68
YATSI(KStar)	60	0,65	0,67	0,69
YATSI(AIRS)	60	0,62	0,65	0,66

7.6.4 Sonuçlar ve Gelecek Çalışmalar

Araştırmacılar, yeterli etiketli veri olmadığı zaman sınıflandırma için etiketsiz verilerden yararlanma konusunda oldukça iyimserdir. Ancak bu çalışmada bu iyimser fikrin tüm sınıflayıcılar ve tüm veri kümeleri için geçerli olmadığını gösteren yeni bir kanıt ortaya koyduk. YATSI algoritmasının ilk adımında Naive Bayes kullanıldığı zaman, oluşan algoritmanın performansı KC2, CM1 ve JM1 veri kümelerinde etiketsiz veriler kullanılmasıyla birlikte azalmıştır. YATSI algoritması, J48, RF, AIRS, KStar algoritmalarının performansını arttırmaktadır ancak Naive Bayes algoritmasının performansını KC2, CM1 ve JM1 veri kümeleri üzerinde azaltmıştır. Algoritmaların performansı daha fazla etiketli veri kullanılmasıyla çoğu durumda artmıştır. PC1 veri kümesi üzerinde tüm sınıflayıcıların performansı YATSI ile artmıştır. Naive Bayes algoritmasının yarı-eğiticili yazılım kusur kestirimi problemi için en iyi performansı sunduğunu, büyük veri kümeleri için YATSI algoritmasının ilk adımında kullanılması ile birlikte performansının artacağını ifade edebiliriz. En büyük veri kümesi olan JM1'de YATSI'nın Naive Bayes algoritmasının performansını arttıramamasının nedeni, JM1 içindeki mevcut olabilecek uyumsuz (eş metrik fakat farklı etiketler) modüllerdir.

Naive Bayes algoritması sınırlı sayıda kusur verisiyle yazılım kusur kestirimi yapmak için en ideal algoritmadır. Naive Bayes algoritmasından yararlanan yarı-eğiticili algoritmalar geliştirebilirsek daha iyi sonuçlar elde edebiliriz. Naive Bayes algoritması temel algoritma olarak seçilerek öz-eğitim (self-training) ve birlikte-öğrenme (co-training) yaklaşımları kullanılarak performansı daha iyi algoritmalar geliştirilebilir. Bu problemde, Rastgele Orman algoritmasının performansı da oldukça yüksek olduğundan oluşturulacak modellerde bu algoritmanın kullanımı yoluna gidilebilir. Birlikte-öğrenme yaklaşımı incelenirken, Naive Bayes ve Rastgele Orman algoritmalarının birlikte kullanılması ele alınabilir.

Bu çalışmada literatüre çeşitli katkılar sunulmuştur: İlk olarak, YATSI algoritmasının Naive Bayes gibi bazı algoritmaların performansını azaltabileceine ilişkin bazı kanıtlar sunduk. İkinci olarak, Naive Bayes algoritmasının sınırlı kusur verisiyle yazılım kusur kestirimi yaparken kullanılabilecek en iyi sınıflandırma algoritması olduğunu empirik deneylerle gösterdik. Üçüncü olarak, yeni bir model bağlamında, ilk kez Yapay Bağışıklık Sistem paradigmasının kullanıldığı yarı-eğiticili sınıflandırma algoritması olan YATSI(AIRS) algoritmasını önerdik. Dördüncü olarak, daha fazla etiketli veri kullanıldığı zaman algoritmaların performansının arttığını gösteren yeni kanıtlar sunduk. Son olarak bu çalışma, yarı-eğiticili kusur kestirimi için çeşitli sınıflayıcıların performanslarını karşılaştırmakta ve

sonraki araştırmalar için bir sınıflayıcı önermektedir.

Gelecekte Naive Bayes, Rastgele Orman ve YATSI algoritmalarının kullanıldığı yarı-eğiticili algoritmalar geliştirilebilir. Performansı arttırmak üzere YATSI algoritmasında farklı uzaklık fonksiyonlarından yararlanılabilir. Ek olarak gürültü tespit algoritmaları JM1 veri kümesinde uygulanarak, gürültülü modüller saptandıktan sonra deneyler yeniden gerçekleştirilebilir.

8. YAZILIM ÜRÜN HATLARINDA KUSUR KESTİRİMİ

Bu bölümde, yazılım ürün hattı mühendisliğinde kusur kestiriminin nasıl uygulanabileceği açıklanmaktadır.

8.1 Motivasyon

Yazılım ürün hattı mühendisliği (software product line engineering); mimari belgeleri, tasarım dosyaları, kod bileşenleri, test senaryoları gibi yeniden kullanılabilir çekirdek varlıkları (reusable core assets) kullanarak benzer ürünleri geliştirmede yararlanılan yeni bir paradigmadır. Genel amaç, geliştirme maliyetlerini ve süresini kısaltarak pazara yüksek kaliteli bir ürün sunmaktır. Bu nedenle, geleneksel tekli sistem mühendisliği (single-system engineering) yaklaşımlarında kullanılan etkin kalite güvencesi aktivitelerinin daha fazlasının yazılım ürün hattı mühendisliğinde kullanılması gerekmektedir. Mevcut yazılım ürün hattı mühendisliği çerçevelerinde, çok az sayıda kalite güvence aktivitesine başvurulmaktadır. Ancak günümüz tekli sistem mühendisliğinde fazla sayıda kalite güvencesi aktivitesi mevcuttur ve bu yaklaşımların ürün hattı mühendisliğine uyarlanması mümkündür. Bu bölümde, yazılım kusur kestirimi alt süreci (sub-process) yazılım ürün hattı mühendisliğine sistematik olarak tümleştirilecek ve bu alt süreçlerin bir parçası olması gereken anahtar aktiviteler tanımlanacaktır. Önerilen kavramsal çerçeve modelin, yazılım ürün hattı mühendisliği için kaliteyi arttıracağı, test alt sürecini iyileştireceği ve geliştirme sürelerini azaltacağı öngörülmektedir.

8.2 Giriş

Yazılımın yeniden kullanılabilirliği ilk kez NATO tarafından gerçekleştirilen Yazılım Mühendisliği toplantısında ortaya konulmuştur (McIlroy, 1968) ve daha sonra farklı yazılım geliştirme paradigmalarının temel ilgisi haline gelmiştir. 1960'larda alt rutinler, 1970'lerde modüller, 1980'lerde nesneler, 1990'larda bileşenler yeniden kullanım için anahtar teknoloji elemanları olarak değerlendirilmiştir. 2000'lerden itibaren ise yazılım ürün hatları, yeniden kullanılabilirliğinin en ümit verici yaklaşımı olarak görülmektedir. Yazılım ürün hattı mühendisliği yaklaşımı, yeniden kullanım aktivitelerinin sistematik ve planlanmış olarak yazılım yaşam çevrimi süresince uygulanmasını beklemekte ve buna bağlı olarak üretkenlik ve kalitenin artacağı konusunda umut vermektedir. Yeniden kullanım aktiviteleri, yazılım kaynak kodunun yeniden kullanımı ile sınırlı olmayıp isterler mühendisliği, tasarım, gerçekleme ve test alt süreçlerinin çıktılarının yeniden kullanılabilirliğini içermektedir.

Yazılım Mühendisliği Enstitüsü'nün tanımına göre (Carnegie Mellon Üniversitesi, 2007), "yazılım ürün hattı, pazarın özel bir segmentinin veya özel bir amacın spesifik ihtiyaçlarını karşılamak üzere ortak ve yönetilen özellikler kümesini paylaşan, ortak çekirdek varlıklar kümesinden geliştirilen yazılım-yoğun sistemler kümesidir". Bu tanım, ortak çekirdek varlıklardan geliştirme yapmayı ve kapsam ekonomilerini (economies of scope) vurgulamaktadır. Microsoft veya Oracle gibi firmalarda olduğu şekliyle, yazılımı bir kez oluşturup aynı ürünü değişiklik yapmaksızın milyonlarca kullanıcıya satabiliyorsanız, ölçek ekonomileri (economies of scale) yaklaşımını rahatlıkla kullanabilirsiniz, ancak çoğu zaman bu yöntem mümkün değildir ve farklı kullanıcı gruplarının beklentilerini karşılayabilmek için ürün üzerinde bazı varyasyonlar gerçekleştirmeye ihtiyaç duyarsınız. Bu durumda, kapsam ekonomileri yaklaşımı gündeme gelir, maliyet ve zaman bu sayede hızlı şekilde azalır.

Greenfield ve arkadaşları (2004), ölçek ve kapsam ekonomisini aşağıdaki şekilde tanımlamaktadır:

"Ölçek ekonomileri, tek bir tasarımdan çok sayıda eş örnek tek tek üretilmek yerine birlikte üretildiği zaman ortaya çıkar. Kapsam ekonomileri ise, çok sayıda benzer ancak farklı tasarımlar ve prototipler tek tek üretilmek yerine birlikte üretildiği zaman ortaya çıkar". Yazılım ürün hatları, "platform" adı verilen yeniden kullanılabilir çekirdek varlıkları sunarak pazarın bir segmenti için benzer ürünler geliştirmeyi sağlayıp kapsam ekonomilerinin gerçekleştirilmesini kolaylaştırır. Clements ve Northrop (2002), yazılım ürün hatlarının hangi anlamlarda kullanılmadığını açıklamıştır. Yazılım ürün hatları; tekli sistem mühendisliğine yeniden kullanımın eklenmesi, bileşen tabanlı geliştirmenin aynısı, yeniden konfigüre edilebilir bir mimari, bir ürünün farklı sürümleri ve teknik standartlar kümesi kesinlikle değildir.

Başarılı yazılım ürün hatlarına ulaşabilmek için, yazılım ürün hattı mühendisliği adı verilen yeni bir yaklaşıma ihtiyaç vardır. "Yazılım ürün hattı mühendisliği, yazılım uygulamaları (yazılım-yoğun sistemler ve yazılım ürünleri) geliştirmek için platformları ve kitlesel uyarlamayı (mass customization) kullanan bir paradigmadır" (Pohl vd., 2005). Platform, ortak çekirdek varlıklar olup benzer ürünler geliştirmeyi kolaylaştırır ve sadece yazılım kodundan oluşmakla kalmayıp isterler, mimari, bileşenler ve test senaryoları gibi çıktıları da içerir. 1980'lerde geniş ölçekli üretim için kullanıcı gruplarının ihtiyaçlarına göre ürünleri uyarlamada kullanılan kitlesel uyarlama yaklaşımı, kitlesel üretim (mass production) için alternatif bir yöntemdir. Kapsam ekonomileri, kitlesel uyarlamayı gerektirmektedir.

Yazılım ürün hattı mühendisliği çerçevesi, alan (domain) ve uygulama (application) mühendisliği adı verilen iki farklı süreç tanımlamaktadır (Pohl vd., 2005) ve bu ayrım Linden'in (2002) çalışmasında vurgulanmaktadır. ITEA (Information Technology for European Advancement) projeleri içerisinde yer alan; CAFE (From Concepts to Application in System-Family Engineering), ESAPS (Engineering Software Architectures, Processes and Platforms for System-Families) ve FAMILIES (Fact-based Maturity through Institutionalization Lessons-learned and Involved Exploration of System-family Engineering) projeleri, bu çerçevenin oluşturulmasına yardımcı olmuştur (Pohl vd., 2005).

Alan mühendisliği; çekirdek varlıkları kullanarak benzer ürünler geliştirmek için gerekli olan platformu inşa eder ve ürün hattının benzerliklerini (commonality) ve değişkenliklerini (variability) tanımlar. Uygulama mühendisliği; ürün hattının değişkenliğine odaklanır ve ürünleri inşa eder. Bu ayrım, müşteriye özel uygulamaların hızlıca inşa edilmesini sağlar ve çerçeveyi daha yönetilebilir kılar.

Şimdiye kadar; yazılım ürün hatlarının analiz, tasarım ve gerçeklemesi için çok sayıda çalışma yürütülmüştür ancak test gibi kalite güvence aktiviteleri konusunda yapılan çalışmaların sayısı oldukça az ve yetersizdir. 2004 yılındaki Yazılım Ürün Hattı Konferansı'nda, yazılım ürün hatları için yapılan çalışmalarda kalite güvencesi aktivitelerinin yeterince irdelenmediği ve bu alanın fırsatlar ve zorluklarla dolu (challenging) olduğu ifade edilmiştir (Kolb ve Muthig, 2004).

Tekli-sistem mühendisliğinde kullanılan farklı kalite güvence aktivitelerini inceleyerek yazılım ürün hatları için uyarlamak önemli ihtiyaçlardan birisidir. İnceleme (inspection) ve gözden geçirme (review) aktiviteleri gibi diğer kalite güvence aktivitelerinin de yazılım ürün hattı mühendisliği çerçevesine tümleştirilmesi gerekmektedir (Pohl vd., 2005).

Bu tez çalışmasında, yazılım kusur kestirimi alt sürecini içeren değiştirilmiş bir yazılım ürün hattı mühendisliği çerçevesi önerilmektedir. İleriki bölümlerde alan ve uygulama kusur kestirimi olarak ifade edilecek alt süreçler, yazılım ürün hattı mühendisliğine tümleştirilecek ve anahtar aktiviteler tartışılacaktır. Bu tümleştirmeden beklenen kazançlar; daha kararlı yazılım ürün hattına ulaşabilme, yeniden yapılandırmaya (refactoring) aday olan alan ve uygulama modüllerini belirleyebilme, alan ve uygulama modüllerini sistem testleri için önceliklendirebilme olarak sıralanabilir.

Kolb ve Muthig (2006), yazılım ürün hatları için test süresini azaltma çalışmalarındaki yetersizliği vurgulamışlar, kalite güvencesi ve testin ürün hattı mühendisliğinde dar boğaz

oluşturduğunu ifade etmişlerdir. Ayrıca; ürün hattı mühendisliği için gerekli test maliyetlerinin tekli sistem mühendisliğindeki test maliyetlerinden daha yüksek değerlere ulaşmakta olduğunu ifade etmişlerdir. Bu nedenle; ürün hatları için inceleme, kusur kestirimi, kusur toleransı, kusur önleme ve formal doğrulama gibi farklı kalite güvence faaliyetleri konusunda daha fazla çalışma yürütmek gerekmektedir.

Bu tez çalışması, yazılım ürün hattı mühendisliğinde kusur kestirim yöntemlerinin kullanılmasının alan ve uygulama test süreçlerini iyileştireceğini ve üretkenliği arttıracağını savunmaktadır. Önerilen kusur kestirim alt süreci ve anahtar aktiviteler; evrimsel model veya spiral model gibi geliştirme yaklaşımları ile birleştirilebilecek esnekliktedir.

Bu çalışmanın yazılım kusur kestirim uygulayıcılarını ve yazılım ürün hattı uzmanlarını, ürün hattı kusur kestirim modelleri geliştirmek üzere motive edeceğine inanıyoruz. Yaptığımız incelemelere göre, bu çalışma yazılım ürün hattı mühendisliğinde kusur kestiriminin ele alındığı ilk çalışma ve ilk çerçeve modeldir. Yazılım ürün hatları için mevcut durumda maalesef, açık bir veri kümesi bulunmamaktadır ve bu durum önerilen çerçevedeki bazı aktivitelerin deneysel olarak geçerlenmesini zorlaştırmaktadır.

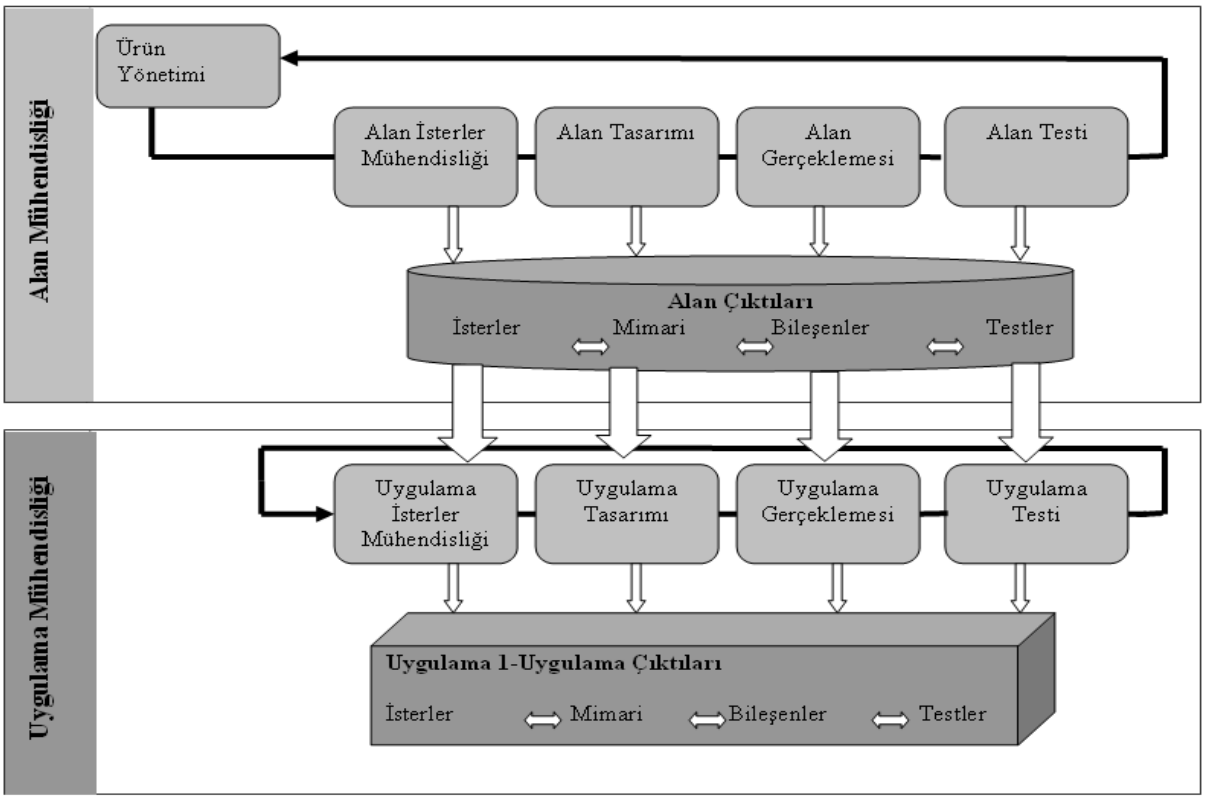
8.3 Yazılım Ürün Hattı Mühendisliği Çerçevesi

Yazılım ürün hattı mühendisliği paradigması birbirini tamamlayan iki alt süreç içermektedir. Bu ayrım ilk kez Weiss ve Lai (1999) tarafından önerilmiş ve genel pratikler bu yaklaşım altında şekillenmiştir.

Alan mühendisliği, yeni ürünler için uygun değişkenlik (variation) noktalarını dikkate alarak platformu oluşturur ve uygulama mühendisliği farklı uygulamalar için bu noktalara başvurarak platformu kullanır. ESAPS ve CAFE gibi ITEA projeleri, bu bölümde açıklanacak olan çerçevenin temellerini oluşturmuştur. Şekil 8.1' de yazılım ürün hattı mühendisliği çerçevesi gösterilmektedir.

Alan mühendisliği, beş alt süreç içerir: ürün yönetimi, alan isterler mühendisliği, alan tasarımı, alan gerçeklemesi ve alan testi. Uygulama mühendisliği, dört alt süreç içermektedir: uygulama isterler mühendisliği, uygulama tasarımı, uygulama gerçekleme ve uygulama testi (Pohl vd., 2005).

Bu çerçeve içerisinde, mimari belgeler ve test senaryoları gibi geliştirme çıktıları iki farklı çıktı (artifact) grubu içinde değerlendirilir: alan ve uygulama çıktıları.



Şekil 8.1 Yazılım ürün hattı mühendisliği çerçevesi (Pohl vd., 2005)

Alan mühendisliğindeki süreçler aşağıda açıklanmaktadır (Pohl vd., 2005):

- *Ürün Yönetimi:* Bu alt süreç, yazılım ürün hattının gelecek ürünlerinin kapsamını tanımlar ve bu kapsam, şirket stratejileri ve üst yönetimin beklentileri ile uyumlu olmalıdır. Bu alt sürece olan girdi şirket amaçlarıdır ve çıktılar; ürün özellikleri, sürüm tarihleri ve mevcut ürünlerin listesidir.

- *Alan İsterler Mühendisliği:* Bu alt sürecin amacı, alan isterlerini belirlemek ve uygulama-isterler matrisi veya öncelik tabanlı analiz (priority-based analysis) gibi yöntemlerle ortak ve değişken isterler şeklinde kategorize etmektir. Bu alt sürece olan girdi, ürün yol haritası (sonraki ürünlerin özellikleri) olup, çıktılar yeniden kullanılabilir isterler ve değişkenlik (variability) modelidir.

- *Alan Tasarımı:* Bu alt süreç, yeterince esnek olmak zorunda olan referans mimariyi tanımlamaktan sorumludur. Girdiler, alan isterleri ve alan isterleri değişkenlik modelidir. Çıktılar, referans mimari ve teknik değişkenlikleri içeren değişkenlik modelidir.

- *Alan Gerçeklemesi:* Alan bileşenlerinin detaylı tasarım ve gerçeklemesi bu alt süreç

içinde gerçekleştirilir. Bu alt sürece olan girdi, referans mimaridir ve çıktılar; detaylı tasarım ve gerçeklenmiş bileşenlerdir.

- *Alan Testi:* Gerçeklenmiş alan bileşenleri, bu alt süreç içinde geçerlenir ve doğrulanır. Doğrulama ve geçerleme adımları, bileşenlerin ilişkili alan çıktıları (isterler veya tasarım varlıkları) ile uyumluluğunu kontrol eder. Girdiler; isterler, referans mimari, alan bileşen tasarımı ve bileşenlerdir. Çıktılar, yeniden kullanılabilir test çıktıları ve test sonuçlarıdır.

Uygulama mühendisliği alt süreçleri aşağıda açıklanmaktadır (Pohl vd., 2005):

- *Uygulama İsterler Mühendisliği:* Bu alt süreç, uygulama için isterler çıktılarını belgelendirmekten sorumludur. Alan isterlerinin mümkün olduğu kadar kullanılması gerekir ve uygulamaya özel isterler de eklenebilir. Girdiler; ürün yol haritası ve alan isterleridir. Çıktı ise uygulamaya özel ister belirtimidir.

- *Uygulama Tasarımı:* Bu alt süreç; uygulamaya özel mimariyi, referans mimari ve gerekli uyarlamaları kullanarak oluşturur. Alan tasarımında, oluşturulan yeniden kullanılabilir bileşenler ve arayüzler bir araya getirilir ve uygulamaya özel bileşenler ve arayüzler tasarlanır.

- *Uygulama Gerçeklemesi:* Uygulamaya özel bileşenlerin detaylı tasarım ve gerçeklemesi bu alt süreçte gerçekleştirilir. Yeniden kullanılabilir alan bileşenleri ve uygulamaya özel bileşenler kullanılarak uygulama inşa edilir. Girdiler; uygulama mimarisi ve platform bileşenleridir. Çıktı, çalışan bir uygulamadır.

- *Uygulama Testi:* Uygulama, belirtime göre doğrulanır ve geçerlenir. Girdiler; uygulama ister belirtimleri, uygulama mimarisi, bileşenler ve alan/uygulama test çıktılarıdır. Çıktılar; test raporları ve raporlanmış kusurlardır.

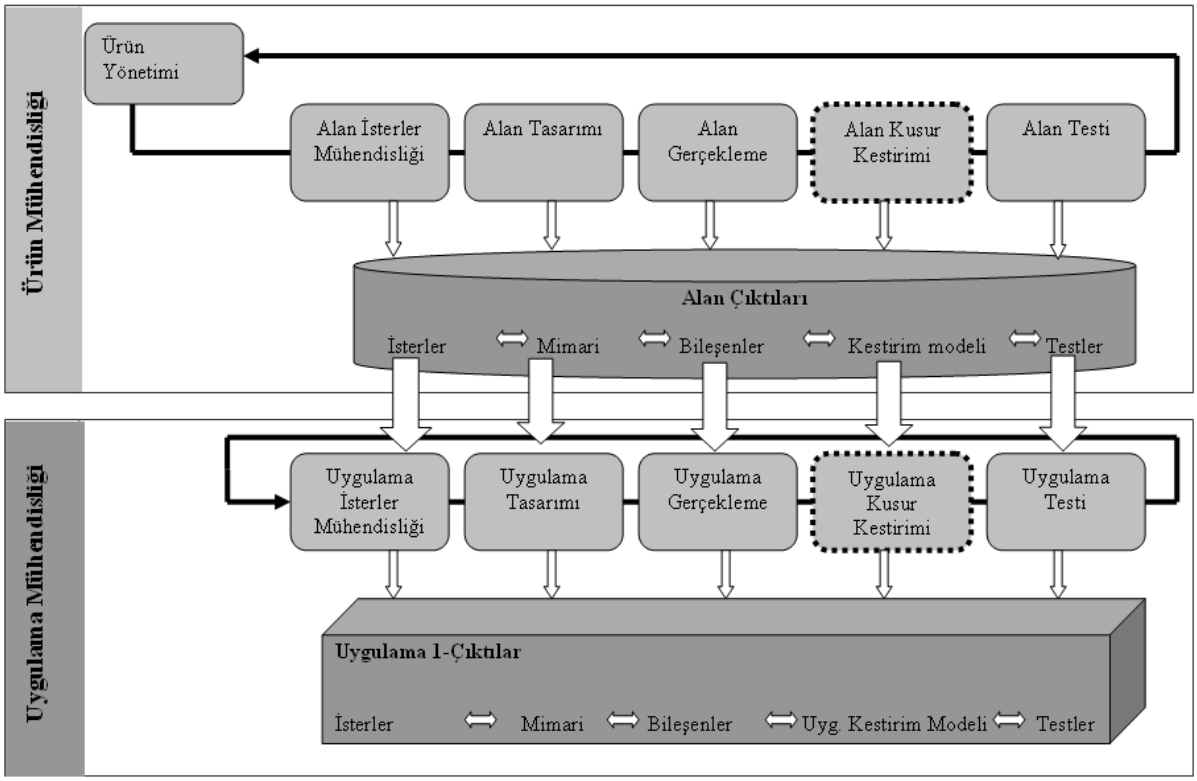
8.4 Yaklaşım

Bu bölümde, yazılım kusur kestirimi alt sürecinin yazılım ürün hattı mühendisliği çerçevesine nasıl tümleştirildiği açıklanacaktır.

8.4.1 Değiştirilmiş Yazılım Ürün Hattı Mühendisliği Çerçevesi

Bir önceki bölümde açıklanan çerçeve, yazılım ürün hattı oluşturmak için oldukça güçlü bir yöntemdir ancak eksik olan, kusur kestirimi gibi etkin kalite güvence aktiviteleridir.

Şekil 8.2' de verilen değiştirilmiş ürün hattı mühendisliği çerçevesinde; alan kusur kestirimi ve uygulama kusur kestirimi adı verilen iki yeni alt süreç kesikli çizgilerle gösterilmektedir. Ayrıca, alan kusur kestirim model çıktıları ve uygulama kestirim model çıktıları gösterilmiştir. Alan bileşenlerinin kusur eğilimliliği, alan kusur kestirim modeli ile belirlenmektedir. Alan kestirim modeli, uygulama bileşenlerinin kestirimini yapmak üzere de kullanılacakır ancak ek olarak uygulama kusur veri kümesi adı verilen ek bir veri kümesine daha ihtiyaç vardır. Alan kusur kestirim modellerinde ise, uygulama kusur verisi ve metriklerinden yararlanılmamaktadır. Buradaki farklılığın nedeni, ürün değişkenliklerinin metriklerde ve kusurlarda farkılılıklara neden olacağı gerçeğidir.



Şekil 8.2 Değiştirilmiş yazılım ürün hattı mühendisliği çerçevesi

8.4.2 Alan Kusur Kestirimi

Alan kusur kestirimi alt süreci, gevşek bağlı alan bileşenleri arasından kusur eğilimli modüllerin belirlenmesi ile ilgilenir. Bu sürecin girdileri; alan bileşenlerinin yazılım metrikleri ve bir önceki alan testi alt sürecinden gelen kusur verisidir. Metrikler, ürün veya süreç seviyesinde olabilir ancak ürün seviyesindeki metrikler kusur kestirim çalışmaları için kullanılmaktadır. Kusur kestirim alt süreci alan gerçeklemesinden sonra yapıldığı için, detaylı tasarımın sınıf metrikleri ya da kaynak koddan metot seviyesindeki metrikler kullanılabilir.

Alan kusur kestiriminin çıktısı, alan bileşen modüllerinin kusur eğilimlilik etiketleri ve yeniden kullanılabilir alan kusur kestirim modelidir. Alan kusur kestirimi, tekli sistem mühendisliğindeki kusur kestiriminden aşağıdaki yönleriyle farklıdır:

- Platformlarda çok sayıda alan bileşenleri olduğu için, yüksek performanslı kestirim modelleri tasarlamak oldukça zordur.
- Platformun oluşturulma maliyeti oldukça pahalıdır ve alan testi süresinin tekli sistem mühendisliğindeki teste göre çok daha fazla kısaltılabilmesi gerekir.
- Alan kusur kestirimi sadece alan bileşenlerini dikkate alır. Uygulama ürünlerinin modülleri, bu alt süreç için kullanılmaz.

Alan kusur kestirim çıktıları; alan metrik veri kümesini, alan kusur veri kümesini, alan kestirim modelini, alan modüllerinin kusur eğilimlilik etiketlerini ve kestirim sonrası alan eylem (action) planını içerir. Alan metrikleri veri kümesi, her alan bileşeninin metriklerini gösteren basit bir metin belgesi olabilir. Alan kusur veri kümesi, her alan bileşeninin bir önceki sürümde test aşamasında kusur çıkarıp çıkarmadığını gösterir. Bu iki veri kümesi, alan kestirim veri kümesi adı verilen tek bir veri kümesinde birleştirilebilir. Alan kusur verisi, Bugzilla veya ClearQuest gibi değişiklik yönetim sistemlerinde saklanabilir. Alan metrikleri ise, Subversion veya Clearcase gibi sürüm kontrol sistemlerinden elde edilebilir. Kestirim sonrası alan eylem planı, alan bileşenleri etiketlendikten sonra gerçekleştirilecek eylemleri tanımlamaktadır. Örneğin; bu bileşenler yeniden düzenleme işlemine tabi tutulabilir ya da test aşamasında önceliklendirme için kusur eğilimlilik bilgileri kullanılabilir. Alan kestirim modeli, yazılım ürün hattından elde edilecek ilk ürünün kusur kestirimini gerçekleştirmek üzere kullanılabilir çünkü o aşamada, uygulama kusur kestirim modelleri için kusur bilgileri oluşmamış olacaktır.

8.4.3 Uygulama Kusur Kestirimi

Alan kusur kestirim modeli, uygulama modüllerinin kusur eğilimliliğini değerlendirmek üzere yeniden kullanılabilir. Ancak, uygulamayı merkez alan yeni bir kusur kestirim modeli oluşturmak daha yararlı olacaktır. Alan kusur veri kümesi ile uygulama kusur veri kümesini birleştirerek bir model ortaya koymak, platform modüllerinde yer alan fazla sayıda metrik ve kusur bilgisini dikkate alacağı için, kestirim performansının daha yüksek olmasını sağlayacaktır. Yazılım ürün hatları için açık veri kümeleri mevcut olmadığından, bu tümleştirme yaklaşımını henüz deneysel olarak geçerleyemedik ancak açık veri kümelerinin

PROMISE havuzuna aktarılması ile birlikte bu aşamanın geçerlenmesi sağlanacaktır. Uygulama kusur kestirim alt süreci, kusur eğilimli uygulama modüllerini belirleyecek ve uygulama testi için modüllerin önceliklendirilmesini sağlayacaktır. Girdiler; alan kusur kestirim modeli, uygulama bileşenlerinin metrikleri ve kusur verisidir. Çıktılar; uygulama modüllerinin kusur eğilimlilik etiketleri, uygulama kusur kestirim modeli ve kestirim sonrası uygulama eylem planıdır. Uygulama kusur kestirimi aşağıdaki yönleriyle tekli sistem mühendisliğinin kusur kestiriminden farklıdır:

- Uygulama kusur kestirim alt süreci, kestirim modelini baştan oluşturmaz, alan kestirim modeli ve uygulamaya özel verileri kullanır.
- Uygulama kusur kestirim modeli, alan kestirim modeli veya uygulama kusur veri kümeleri değiştiği zaman güncellenir.

Uygulama kusur kestirim çıktıları; yeniden kullanılabilir kusur kestirim modeli, uygulama metrik veri kümesi, uygulama kusur veri kümesi, uygulama modüllerinin kusur eğilimlilik bilgileri, kestirim sonrası uygulama eylem planıdır.

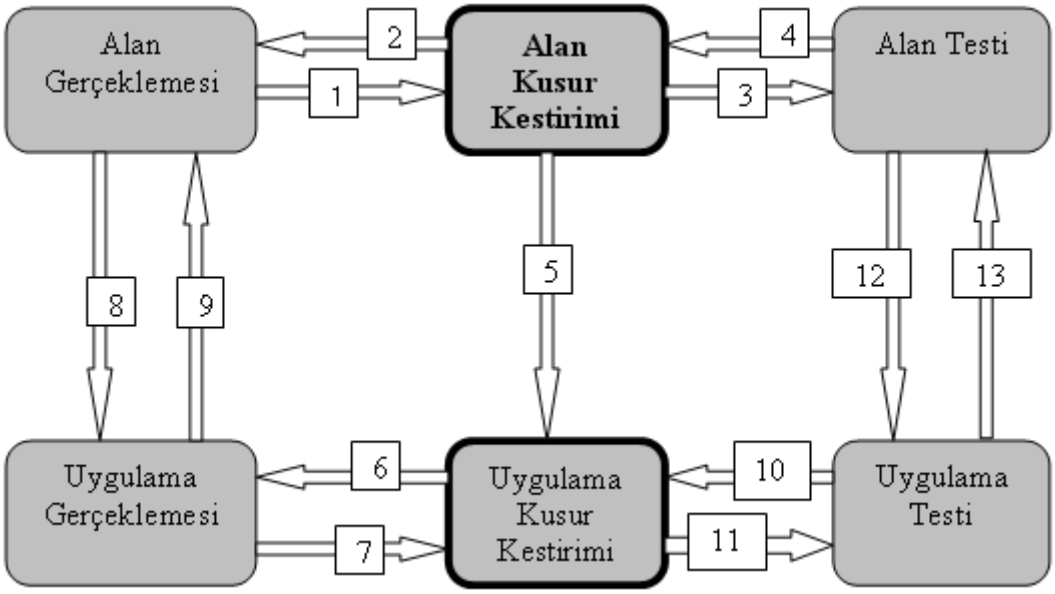
8.4.4 Kusur Kestirimi ve Diğer Alt Süreçler Arasında Bilgi Akışı

Alan kusur kestirim alt süreci ile temel ilişkide bulunan alt süreçler; alan gerçeklemesi, alan testi ve uygulama kusur kestirimidir. Uygulama kusur kestirim alt süreci ile temel ilişkide bulunan alt süreçler; uygulama gerçeklemesi, uygulama testi, alan kusur kestirimidir. Ancak, alan kusur kestirimi ve uygulama kusur kestirimi arasındaki ilişki tek yönlüdür. Şekil 8.3’ de bu alt süreçler arasındaki ilişkiler gösterilmektedir. 8, 9, 12 ve 13 nolu ilişkiler bu bölümde önerilen yaklaşımlarla ilgili olmadığı için detayları Pohl ve arkadaşlarının (2005) kitabından incelenebilir. Şekilde gösterilmesinin nedeni, söz konusu olabilecek tüm ilişkilerin ilgili resme yansıtılmasıdır.

8.4.4.1 Alan Kusur Kestirimi ve Alan Gerçeklemesi Arasındaki İlişki

Alan gerçeklemesi tamamlandıktan sonra, detaylı tasarım ve gerçekleme çıktıları hazır hale gelecektir. Alan gerçeklemesi, yeniden kullanılabilir bileşenleri ve arayüz tanımlarını (#1) alan kusur kestirimi alt sürecine sunmaktadır. Alan kusur kestirimi, alan bileşenlerinin ve arayüzlerinin yazılım metriklerini Alan Metrik Toplama Aracı (Domain Metrics Gathering Tool) ile hesaplamaktadır. Detaylı tasarım, alan gerçeklemesinden sonra hazır olduğu için kusur kestirim uzmanı metot seviyesindeki metriklerin yanı sıra sınıf seviyesindeki metriklerden de bu aşamada yararlanabilmektedir. Önerilen çerçeve, herhangi bir

programlama paradigmasına bağlı olmadığı için araştırmacılar farklı metrikler kullanabilir. Örneğin; ilgiye yönelik programlamanın kullanılmaya başlayacağı yıllarda ilgiye yönelik metriklerin bu çerçeve dahilinde kullanılması mümkündür. Alan kusur kestirimi, yeniden yapılandırmaya aday alan bileşenleri (refactoring domain candidates), alan gerçeklemesi bileşenine (#2) sunmaktadır. Alan gerçeklemesi, metriklerinde anormallik bulunan modüllerin iç yapısını bu sayede değiştirebilmektedir.



Şekil 8.3 Kusur kestirimi ve diğer alt süreçler arasındaki bilgi akışları

8.4.4.2 Alan Kusur Kestirimi ve Alan Testi Arasındaki İlişki

Alan testi, alan kusur kestirimi alt sürecine alan kusur verisini (#4) sunmaktadır. Alan Kusur Veri Toplama Aracı (Domain Fault Data Gathering Tool) bu verileri modelde kullanılabilecek bir formata dönüştürmekten sorumludur. Alan kusur kestirimi, bu verilerle yazılım metriklerini kullanarak alan kusur kestirim modelini oluşturur. Alan kusur kestirimi; test önceliklendirme verisini ve kestirim sonrası alan eylem planını (#3) alan testine sunmaktadır. Alan testi bu önceliklendirme bilgisine göre alan modüllerinin testini gerçekleştirir. Yüksek öncelikli modüller, kusur eğilimli olduğundan öncelikle test edilirler. Kusur kestirim sonrası alan eylem planı, kusur eğilimli modüllerin nasıl kullanılacağını tanımlar. Test önceliklendirme veya yeniden yapılandırma gibi tercihler bu planda belirtilmektedir.

8.4.4.3 Alan Kusur Kestirimi ve Uygulama Kusur Kestirimi Arasındaki İlişki

Alan kusur kestirimi, alan kusur kestirim modelini (#5) uygulama kusur kestirimine sunar.

Platform tüm uygulamalar için kullanıldığından ve uygulamalar arasında değişikliklerin boyutu yüksek olabileceğinden, veri akışı tek yönlü tanımlanmıştır.

8.4.4.4 Uygulama Kusur Kestirimi ve Uygulama Gerçeklemesi Arasındaki İlişki

Uygulama kusur kestirimi, yeniden yapılandırma gerektiren uygulama bileşenlerini (#6) uygulama gerçeklemesine sunmaktadır. Uygulama gerçeklemesi, yeniden kullanılabilir bileşenleri ve uygulama arayüz tanımlamalarını (#7) uygulama kusur kestirimine sunar. Uygulama kusur kestirimi alt sürecinde, Uygulama Metrik Toplama Aracı ile uygulama metrikleri hesaplanır. Bu kapsamda, metot ya da sınıf seviyesindeki metrikler kullanılabilir.

8.4.4.5 Uygulama Kusur Kestirimi ve Uygulama Testi Arasındaki İlişki

Uygulama testi, uygulama kusur verisini (#10) uygulama kusur kestirimi alt sürecine sunar. Uygulama Kusur Toplama Aracı bu verileri modelde kullanılabilecek bir formata dönüştürmekten sorumludur. Uygulama kusur kestirimi bu verileri metriklerle birlikte kullanarak uygulama kusur kestirim modelini oluşturur. Uygulama kusur kestirimi test önceliklendirme verisini ve kestirim sonrası alan eylem planını (#11) uygulama testine sunar. Bu önceliklendirme verisine göre, uygulama testi gerçekleştirilir. Kestirim sonrası eylem planında ise, modüllerin yeniden yapılandırılıp yapılandırılmayacağı gibi kararlar oluşturulur.

8.5 İlişkili Çalışmalar

Zhang ve arkadaşları (2003), yazılım ürün hatları için Bayes Ağı (Bayesian Belief Network-BBN) tabanlı kalite kestirim yaklaşımını önermişler ancak bu çalışmalarında, yazılım kusur kestirimi konusuna değinmemişlerdir. Ürün hatları için, özelliklerin ağaç yapılarıyla gösterildiği özellik diyagramlarında (feature diagrams), kalite unsurlarını dikkate alarak en iyi konfigürasyonu elde edecek bir model üzerinde çalışmışlardır. Örneğin; kullanıcı beklentisi yüksek performanslı bir sistem iken konfigürasyon düşük performansa neden oluyor ise bu konfigürasyonun silinmesi gerekmektedir. Tasarım kararları ağacın üst bölümünde düğümler olarak gösterilmiş ve kalite faktörleri alt bölümde düğümler olarak temsil edilmiştir. Yönlü kenarlar (edge) ilişkiyi tanımlamaktadır. Her düğüme koşullu olasılık atanarak diğer düğümlerin olasılık dağılımları farklı yöntemlerle (jonksiyon ağacı vb.) hesaplanmıştır. Bu çalışmada, her düğüme olasılık atanması oldukça zor bir süreçtir ve uzman bilgisi veya yeterince objektif veri yoksa uyumsuz sonuçlar üretebilir. Zhang ve arkadaşlarına göre, çalışmaları erken aşamadadır ve olasılıkların alan uzmanları tarafından elde edilebilmesi için bazı yöntemlerin geliştirilmesi gerekmektedir. Bu çalışmalarında, kusur kestirimi konusunda

herhangi bir bilgi sunulmamıştır.

Trendowicz ve Punter (2003), yazılım ürün hatlarında kalite modelleme için Prometheus yaklaşımını önermişlerdir. Bu yaklaşım 3 fazdan oluşmaktadır: belirtim, uygulama ve paketleme. Bu çalışmada da, yazılım kalite modelleme için BBN (Bayesian Belief Networks) ağları kullanılmıştır. Ancak bu çalışmada da kusur kestirimi üzerine herhangi bir bilgi sunulmamıştır. Aldekoa ve arkadaşları (2006), yazılım ürün hatlarında bakım yapılabilirliğin ölçülebilmesi için bir yöntem önermişlerdir ancak diğer kalite faktörlerini ve yazılım kusur kestirimini incelememişlerdir.

8.6 Sonuçlar ve Öneriler

Bu çalışmada yazılım ürün hattı mühendisliğinde kusur kestiriminin uygulanabilmesi için çerçeve bir model tanımlanarak alt süreçler arasındaki bilgi akışları açıklanmıştır. Bu çerçeve modelin, geleneksel kusur kestirim modellerini yazılım ürün hatlarına taşıması nedeniyle alan ve uygulama test sürecini iyileştirerek ürün hattı kalitesini arttıracağı öngörülmektedir. Yazılım ürün hatları için açık veri kümeleri oluştuğu zaman, bu çerçeve modeldeki bazı kararların deneysel olarak geçerliliğinin sağlanması gerekmektedir. Bu kapsamda, bir ihtiyacın olduğu açıktır. Bu çalışmanın, yazılım kalite mühendislerini yazılım ürün hatlarında farklı kalite güvence aktivitelerini gerçeklemek üzere çalışmalara yönlendirmesi ve bu bağlamda Yapay Bağışıklık Sistem paradigması gibi farklı yöntemlerden yararlanmaları beklenmektedir. Açık veri kümeleri oluştuğu zaman, tez çalışmasında önerilen farklı kusur kestirim yöntemlerinin bu çerçeve dahilinde uygulanması mümkün olacaktır.

9. SONUÇ ve GELECEK ÇALIŞMALAR

Bu bölümde tez çalışmasının sonuç ve gelecek çalışmalarına ilişkin bilgiler sunulmaktadır.

9.1 Giriş

Bu tez çalışması, yazılım kusur kestirim probleminde yapay bağışıklık sistem paradigmasının uygulanmasına odaklanmıştır. Bu amaçla, ilk aşamalarda yapay bağışıklık sistemleri ve yazılım kusur kestirimi konusunda detaylı literatür taraması gerçekleştirilmiştir. Araştırma hipotezini destekleyen araştırma amaçları listesi, tezin giriş bölümünde verilmiştir. Bu bölümde, bu amaçlar yeniden değerlendirilerek tezin temel katkıları ortaya konulacaktır.

9.2 Tez Amaçlarının Yeniden Değerlendirilmesi

Tez çalışması öncesinde belirlenen amaçlar ve bu amaçlar için gerçekleştirilen çalışmalar aşağıda açıklanmaktadır:

1. *Eğiticili sınıflandırma kapsamında kullanılabilecek literatürde tanımlanmış Yapay Bağışıklık Sistem tabanlı algoritmaları belirleyerek yazılım kusur kestirimi problemi için açık veri kümeleri üzerinde performans analizlerini gerçekleştirmek.*

Bölüm 6.3'de bu kapsamda gerçekleştirilen deneyler ayrıntıları ile birlikte verilmektedir. Literatürde kusur kestirimi için yüksek performans sunduğu ifade edilen birçok algoritma ve model mevcuttur ancak bu çalışmaların çoğu açık veri kümelerinde gerçekleştirilmemiştir. Bu nedenle, NASA açık veri kümeleri üzerinde birçok algoritmayı kıyaslayan çalışmaların sonuçları elde edilerek en iyi performansı verdiği ifade edilen algoritmalar saptanmıştır. Çalışmalarda performans değerlendirme için farklı değerlendirme kriterleri kullanılabildiğinden ve tüm çalışmalar aynı algoritma kümesi üzerinde çalışmadığından, karşılaştırmalı analizleri sunan bu tür çalışmalarda önerilen algoritmalar farklı olmuştur. Örneğin; Rastgele Orman (Ma vd., 2006), J48 (Koru ve Liu, 2005a), Naive Bayes (2007a) algoritmalarının farklı araştırmacılar tarafından kusur kestiriminde en iyi performansı sunduğu raporlanmıştır. Aynı veri kümeleri kullanılsa bile kullanılan metrik kümesinin farklılığından dolayı algoritmaların performansları değişebilmekte ve bazı algoritmalar ön plana çıkabilmektedir.

Bu algoritmaları ve diğer YBS tabanlı sınıflandırma algoritmaları farklı metrik kümeleriyle NASA açık veri kümelerinde kullanarak eğiticili kusur kestirimini gerçekleştirdik. YBS tabanlı sınıflandırıcılar olarak; Immunos1, Immunos2, CLONALG, AIRS1, AIRS2 ve

AIRS2Paralel algoritmalarından yararlandık. Literatürde ilk kez YBS tabanlı algoritmalar bu inceleme seviyesinde kusur kestiriminde kullanılmıştır. Araştırma sorularına yanıt bulabilmek için 7 test grubu belirlenmiş ve 5 veri kümesi üzerinde 9 sınıflayıcı incelenmiştir. Yapılan deneysel çalışmalara göre, Rastgele Orman algoritması büyük veri kümelerinde en iyi performansı sunmaktadır ve daha küçük veri kümeleri için Naive Bayes algoritması en iyi kestirimi sağlamaktadır. AIRS algoritmasının paralel gerçeklenmesi (AIRS2Parallel), metot seviyesinde metrikler kullanıldığı durumda YBS tabanlı algoritmalardan en iyisi olarak karşımıza çıkmaktadır. Sınıf seviyesindeki metrikler kullanıldığı durumda ise Immunos2 algoritmasının performansı oldukça yüksektir. Bu çalışmalarda, YBS algoritmalarının diğer birçok makine öğrenmesi tabanlı algoritmadan daha iyi performans verdiği ancak en iyi performansın çoğunlukla Rastgele Orman ve Naive Bayes algoritmaları ile alınabildiği belirlenmiştir. Bu nedenle, sınıflayıcı birlikteliği yaklaşımlarının kullanılacağı durumlarda YBS tabanlı algoritmalar önemli bir aday olarak karşımıza çıkmaktadır.

2. *Farklı özellik azaltma tekniklerinin YBS tabanlı kusur kestirim modellerinde performans etkisini incelemek.*

Bölüm 6.1'de bu amaçla gerçekleştirilen deneyler ayrıntıları ile birlikte sunulmaktadır. AIRS algoritması istatistiksel tekniklerle birleştirilmiş ve korelasyon tabanlı özellik azaltma tekniğinin AIRS ile birlikte kullanıldığı durumda, JM1 veri kümesi için en iyi performansı sunan RF algoritmasından daha iyi performans elde edilebildiği saptanmıştır. Korelasyon tabanlı özellik azaltma tekniğinin yanı sıra, temel bileşen analizi ve uyumluluk tabanlı özellik azaltma teknikleri de incelenmiştir. Geliştirilen model açık veri kümeleri üzerinde sınandığı için, deneylerin yeniden üretilebilirliği ve geçerliliğinin sağlanması mümkündür. Bu çalışma, AIRS algoritması üzerinde gerçekleştirilmiştir.

Bölüm 6.3'de verilen deney 3'de ise korelasyon tabanlı özellik azaltma tekniği ile 21 adet metrik azaltılarak farklı modellerde kullanılmış ve performans değişimleri incelenmiştir. Naive Bayes ve Rastgele Orman algoritmalarının performansının çok fazla değişmediği gözlemlenmiştir. Bu deneylere göre, kusur kestiriminde kullanılan metriklerden ziyade uygulanan algoritmanın daha kritik olduğu belirlenmiştir.

3. *Sınıf seviyesindeki metriklerle, tasarım aşamasında erken kestirimi sağlamak üzere YBS tabanlı modeller tasarlamak.*

Bölüm 6.2'de bu amaçla gerçekleştirilen çalışmalar ayrıntıları ile birlikte sunulmaktadır. CK metrik kümesi içerisinde yer alan metrikler ve bazı metot seviyesindeki metrikler, Yapay

Bağışıklık Tanıma Sistemi algoritması ile birlikte kullanılarak oluşturulan modelin yazılım kusur kestirimindeki performansının saptanması ve incelenen CK metrik kümesinde kusur eğilimliliği belirlemede en önemli/en önemsiz metriklerin saptanması hedeflenmiştir. Deneylerde kullanılan veri kümesi (KC1) NASA Metrik Veri Programının bir parçası olup sınıf seviyesindeki metrikler PROMISE havuzundan elde edilmiştir. AIRS algoritması tabanlı geliştirilen modelin, bağımsız değişkenleri olarak Halstead, McCabe, CK metrikleri üzerinde çalışılmış ve deneysel çalışmalar neticesinde, CK metrik kümesindeki metrikler ve kod satır sayısı metriğinin AIRS ile birlikte kullanılmasına karar verilmiştir. CK metrikleri ve kod satır sayısı metriğinin bağımsız değişkenler olarak kullanıldığı AIRS tabanlı modelin performansı, literatürde önerilmiş ve sınıf seviyesindeki metrikleri kullanan J48 tabanlı yaklaşımdan daha iyi sonuç vermiştir. Ayrıca, sınıf seviyesindeki metriklerle elde edilen performansların metot seviyesindeki metriklerle elde edilen performanslardan daha yüksek olduğu saptanmıştır.

Bu çalışma literatüre çeşitli katkılar sunmaktadır: İlk olarak, kusur eğilimliliği ve 6 adet CK metriği arasındaki ilişkiyi ortaya koyan yeni bir kanıt sunduk. Çalışmamızda KC1 isimli açık bir veri kümesini PROMISE havuzundan kullanarak sonuçlarımızın yeniden üretilebilir, doğrulanabilir veya reddedilebilir olmasını sağladık. İkinci olarak, nesneye yönelik metrikleri kullanan AIRS tabanlı kusur kestirim modelini önerdik ve geçerlenmesini KC1 kümesi üzerinde sağladık. Yazılım kusur kestirimi için, Yapay Bağışıklık Sistem paradigmasının nesneye yönelik metriklerle birlikte kullanıldığı literatürdeki bu ilk çalışmanın sonuçları oldukça ümit vericidir. Üçüncü olarak, bu çalışma sınıf seviyesindeki metriklerin kusur kestiriminde metot seviyesindeki metriklerden daha önemli olduğunu ortaya koymaktadır. Son olarak deneysel çalışmalar, AIRS tabanlı önerdiğimiz modelin literatürde önceden önerilmiş J48 tabanlı modelden daha iyi sonuç verdiğini göstermiştir.

Tasarım aşamasında önerilen modeli kullanabilmek için, sadece 6 adet CK metriği ile çalışılması yeterli olacaktır. Kod satır sayısı metriğinin bu aşamada belirlenmesi mümkün olmadığından, modelde bağımsız değişken olarak yer alamayacaktır.

4. *Sınırlı sayıda kusur verisinin mevcut olduğu durumlar için, YBS tabanlı yarı-eğiticili sınıflandırma algoritması geliştirmek.*

Bölüm 7'de bu amaçla gerçekleştirilen çalışmalar ve geliştirilen modeller ayrıntıları ile birlikte sunulmaktadır. Araştırmacılar, yeterli etiketli veri olmadığı zaman sınıflandırma için etiketsiz verilerden yararlanma konusunda oldukça iyimserdir. Ancak bu çalışmada bu iyimser fikrin tüm sınıflayıcılar ve tüm veri kümeleri için geçerli olmadığını gösteren yeni bir

kanıt ortaya koyduk. YATSI algoritmasının ilk adımında Naive Bayes kullanıldığı zaman, oluşan algoritmanın performansı KC2, CM1 ve JM1 veri kümelerinde etiketsiz veriler kullanılmasıyla birlikte azalmıştır. YATSI algoritması, J48, RF, AIRS, KStar algoritmalarının performansını arttırmaktadır ancak Naive Bayes algoritmasının performansını KC2, CM1 ve JM1 veri kümeleri üzerinde azaltmıştır. Algoritmaların performansı daha fazla etiketli veri kullanılmasıyla çoğu durumda artmıştır. PC1 veri kümesi üzerinde tüm sınıflayıcıların performansı YATSI ile artmıştır. Naive Bayes algoritmasının yarı-eğiticili yazılım kusur kestirimi problemi için en iyi performansı sunduğunu, büyük veri kümeleri için YATSI algoritmasının ilk adımında kullanılması ile birlikte performansının artacağını ifade edebiliriz. En büyük veri kümesi olan JM1'de, YATSI'nın Naive Bayes algoritmasının performansını arttıramamasının nedeni, JM1 içinde mevcut olabilecek uyumsuz (eş metrik fakat farklı etiketler) modüllerdir.

Naive Bayes algoritması sınırlı sayıda kusur verisiyle yazılım kusur kestirimi yapmak için en ideal algoritmadır. Bu çalışmada literatüre çeşitli katkılar sunulmuştur: İlk olarak, YATSI algoritmasının Naive Bayes gibi bazı algoritmaların performansını azaltabileceine ilişkin bazı kanıtlar sunduk. İkinci olarak, Naive Bayes algoritmasının sınırlı kusur verisiyle yazılım kusur kestirimi yaparken kullanılabilecek en iyi sınıflandırma algoritması olduğunu empirik deneylerle gösterdik. Üçüncü olarak, yeni bir model bağlamında, ilk kez Yapay Bağışıklık Sistem paradigmasının kullanıldığı yarı-eğiticili sınıflandırma algoritması olan YATSI(AIRS) algoritmasını önerdik. Dördüncü olarak, daha fazla etiketli veri kullanıldığı zaman algoritmaların performansının arttığını gösteren yeni kanıtlar sunduk.

5. *Yazılım Ürün Hatlarında yazılım kusur kestiriminin uygulanabilmesi için alan ve uygulama mühendisliği alt süreçlerini içerecek şekilde çerçeve bir model önermek.*

Bölüm 8'de bu amaçla gerçekleştirilen çalışmalar ayrıntıları ile birlikte açıklanmaktadır.

Mevcut yazılım ürün hattı mühendisliği çerçevelerinde, çok az sayıda kalite güvence aktivitesine başvurulmaktadır ancak günümüz tekli sistem mühendisliğinde fazla sayıda kalite güvencesi aktivitesi mevcuttur ve bu yaklaşımların ürün hattı mühendisliğine uyarlanması mümkündür. 8. bölümdeki çalışmalarda, yazılım kusur kestirimi alt süreci yazılım ürün hattı mühendisliğine sistematik olarak tümleştirilmiş ve bu alt süreçlerin bir parçası olması gereken anahtar aktiviteler tanımlanmıştır. Ayrıca, kusur kestirimi ve diğer alt süreçler arasındaki bilgi akışları şekiller üzerinden açıklanmıştır. Bu çalışma, literatürde ilk kez kusur kestiriminin yazılım ürün hatlarında nasıl kullanılabileceğini ortaya koymuştur.

6. *Yazılım Mühendisliği disiplini içerisinde kestirim uygulanabilecek problemleri belirleyerek, yeni bir yazılım yaşam çevrimi önermek ve test senaryolarını önceliklendirmek için kusur kestirim tabanlı yeni bir yöntem sunmak.*

Bölüm 4'de kestirim problemleri açıklanarak önerilen "kestirim merkezli yazılım yaşam çevrimi" ortaya konulmuştur. Kestirim problemlerinde henüz istenen olgunluğa erişilmediğinden, bu aşamada bu problemlere odaklanılması ve etkin çözümler üretilmesi gerekmektedir. Test senaryolarını önceliklendirmek için "toplam kusur eğilimli fonksiyon kapsama önceliklendirmesi" adı verilen yöntem önerilmiştir.

9.3 Gelecek Çalışmalar

Tez çalışması sırasında elde edilen ümit verici sonuçlar nedeniyle, yapay bağışıklık sistem paradigmasının ileriki dönemlerde birçok yazılım mühendisliği probleminde uygulanması beklenmektedir. Gerçekleştirilebilecek gelecek çalışmalar takip eden bölümlerde açıklanmaktadır.

9.3.1 Yarı-Eğiticili Kusur Kestirimi Kapsamında Yapılabilecek Çalışmalar

Naive Bayes algoritması sınırlı sayıda kusur verisiyle yazılım kusur kestirimi yapmak için en ideal algoritmadır. Naive Bayes algoritmasından yararlanan yarı-eğiticili algoritmalar geliştirebilirsek daha iyi sonuçlar elde edebiliriz. Naive Bayes algoritması temel algoritma olarak seçilerek öz-eğitim (self-training) ve birlikte-öğrenme (co-training) yaklaşımları kullanılarak performansı daha iyi algoritmalar geliştirilebilir. Bu problemde, Rastgele Orman algoritmasının performansı da oldukça yüksek olduğundan oluşturulacak modellerde kullanımı yoluna gidilebilir. Birlikte-öğrenme yaklaşımı incelenirken, Naive Bayes ve Rastgele Orman algoritmalarının birlikte kullanılması ele elınabilir. Gelecekte Naive Bayes, Rastgele Orman ve YATSI algoritmalarının kullanıldığı yarı-eğiticili algoritmalar geliştirmeyi planlıyoruz. Performansı arttırmak üzere YATSI algoritmasında farklı uzaklık fonksiyonları kullanmak istiyoruz. Ek olarak, gürültü tespit algoritmalarını JM1 veri kümesinde uygulayarak gürültülü modülleri saptayacağız.

9.3.2 Sınıf Seviyesindeki Metriklerle Yapılabilecek Çalışmalar

Zhou ve Leung'un çalışmasında (2006) olduğu gibi, sınıf seviyesindeki metriklerle yapılan çalışmadaki eksiklik sadece tek bir projeye ait veri kümesinin kullanılmasıdır. PROMISE havuzunda bu veri kümesi dışında nesneye yönelik metriklerin sunulduğu daha farklı bir veri kümesi olmadığından, sadece bu veri kümesi ile çalışılabilmiştir. Nesneye yönelik metriklerin

bulunduğu daha farklı veri kümelerine ileriki dönemlerde erişilebilirse, geliştirilen modellerin test edilmesi sağlanabilir. Ayrıca, önerilen modeldeki AIRS algoritması literatürde performansı yüksek olduğu söylenen diğer algoritmalarla birlikte kullanılarak daha yüksek performanslı modeller kurgulanabilir. Sınıflayıcı birlikteliği yaklaşımları bu noktada değerlendirilebilecek yaklaşımlardır.

9.3.3 Metot Seviyesindeki Metriklerle Yapılabilecek Çalışmalar

Yapay Bağışıklık Sistem paradigması bu problemde olduğu gibi birçok problemde de yazılım mühendislerine yardımcı olabilecek güçte olduğu gözlemlenmiştir. Bu çalışmada elde edilen modelin, PROMISE havuzuna yeni eklenecek veri kümeleri üzerinde denenmesi planlanmaktadır. Ayrıca, diğer yüksek performans sunduğu belirlenen algoritmalarla birlikte kullanılarak daha etkin modellerin geliştirilmesi hedeflenmektedir.

9.3.4 Yazılım Ürün Hatları için Önerilen Model Konusundaki Çalışmalar

Yazılım ürün hatları için açık veri kümeleri oluştuğu zaman, bu çerçeve modeldeki bazı kararların deneysel olarak geçerliliğinin sağlanması gerekmektedir. Bu kapsamda, bir ihtiyacın olduğu açıktır. Bu çalışmanın, yazılım kalite mühendislerini yazılım ürün hatlarında farklı kalite güvence aktivitelerini gerçeklemek üzere çalışmalara yönlendirmesi ve bu bağlamda Yapay Bağışıklık Sistem paradigması gibi farklı yöntemlerden yararlanmaları beklenmektedir. Açık veri kümeleri oluştuğu zaman, tez çalışmasında önerilen farklı kusur kestirim yöntemlerinin bu çerçeve dahilinde uygulanması mümkün olacaktır.

KAYNAKLAR

Aldekoa, G., Trujillo, S., Mendieta, G. S. ve Díaz, O., (2006), "Experience Measuring Maintainability in Software Product Lines", JISBD2006 (Jornadas de Ingenieria del Software y Bases de Datos), 3-6 Ekim 2006, Sitges, İspanya, 173-182.

Baluja, S., (1998), "Probabilistic Modeling for Face Orientation Discrimination: Learning from Labeled and Unlabeled Data", Neural Information Processing Systems, 30 Kasım-5 Aralık 1998, Denver, Colorado, ABD, 854-860.

Basili, V. R. ve Rombach, H. D., (1988), "The TAME project: Towards Improvement-Oriented Software Environments", IEEE Transactions on Software Engineering, 14(6):758-773.

Basili, V. R., Briand, L. C. ve Melo, W. L., (1996), "A Validation of Object-Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, 22(10): 751-761.

Bennett, K. P. ve Demiriz, A., (1998), "Semi-Supervised Support Vector Machines", Advances in Neural information Processing Systems, 30 Kasım-5 Aralık 1998, Denver, Colorado, ABD, 368-374.

Bersini, H. ve Varela, F. J., (1994), "The Immune Learning Mechanisms: Reinforcement, Recruitment and Their Applications", Computing with Biological Metaphors, Chapman & Hall, 1(2):166-192.

Bibi, S., Tsoumakas, G., Stamelos, I. ve Vlahvas, I., (2006), "Software Defect Prediction Using Regression via Classification", IEEE International Conference on Computer Systems and Applications, 8-11 Mart 2006, Dubai, UAE, 330-336.

Binkley, A. B. ve Schach, S. R., (1998), "Prediction of Run-Time Failures Using Static Product Quality Metrics", Software Quality Journal, 7(2):141-147.

Binkley, D., Feild, H. ve Lawrie, D., (2007), "Software Fault Prediction using Language Processing", Testing: Industrial and Academic Conference, Practice and Research Techniques, 12-14 Eylül 2007, Windsor, UK, 99-108.

Blum, A. ve Mitchell, T., (1998), "Combining Labeled and Unlabeled Data with Co-Training", 11th Annual Conference on Computational Learning Theory, 24-26 Temmuz 1998, Madison, Wisconsin, ABD, 92-100.

Blum, A. ve Chawla, S., (2001), "Learning from Labeled and Unlabeled Data using Graph Mincuts", 18th International Conference on Machine Learning, 28 Haziran-1 Temmuz 2001, Massachusetts, ABD, 19-26.

Boetticher, G., (2006), "Improving Credibility of Machine Learner Models in Software Engineering", Advanced Machine Learner Applications in Software Engineering, Series on Software Engineering and Knowledge Engineering, Idea Group Publishing, Hershey, PA, ABD.

Bradley, A. P., (1997), "The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms", Pattern Recognition, 30: 1145-1159.

Briand, L. C., Basili, V. ve Hetmanski, C., (1993), "Developing Interpretable Models with Optimized Set Reduction for Identifying High Risk Software Components", IEEE

Transactions on Software Engineering, 19(11):1028-1044.

Briand, L. C., El-Emam, K., Maxwell, K., Surmann, D. ve Wieczorek, I., (1999a), "An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques", 21st International Conference on Software Engineering, 16-22 Mayıs 1999, Los Angeles, CA, ABD, 313-322.

Briand, L. C., Wust, J., Ikonomovski, S.V. ve Lounis, H., (1999b), "Investigating Quality Factors in Object-Oriented Designs: An Industrial Case Study", 21st International Conference on Software Engineering, 16-22 Mayıs 1999, Los Angeles, CA, ABD, 345-354.

Briand, L. C., Wust, J., Daly, J. W. ve Porter, D. V., (2000), "Exploring the Relationships between Design Measures and Software Quality in OO Systems", Journal of Systems and Software, 51(3):245-273.

Briand, L. C., Wust, J. ve Lounis, H., (2001), "Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs", Empirical Software Engineering, 6(1):11-58.

Brownlee, J., (2005a), "Clonal Selection Theory and CLONALG - The Clonal Selection Classification Algorithm", Technical Report, Centre for Intelligent Systems and Complex Processes, Faculty of Information and Communication Technologies, Swinburne University of Technology, Victoria, Australia.

Brownlee, J., (2005b), "Artificial Immune Recognition System (AIRS) A Review and Analysis", Technical Report, Centre for Intelligent Systems and Complex Processes, Faculty of Information and Communication Technologies, Swinburne University of Technology, Victoria, Australia.

Brownlee, J., (2005c), "Immunos-81 The Misunderstood Artificial Immune System", Technical Report, Centre for Intelligent Systems and Complex Processes, Faculty of Information and Communication Technologies, Swinburne University of Technology, Victoria, Australia.

Bruce, R., (2001), "Semi-supervised Learning using Prior Probabilities and EM", IJCAI Workshop on Text Learning, 17-22.

Carter, J. H., (2000), "The Immune System as a Model for Pattern Recognition and Classification" Journal of the American Medical Informatics Association, 7(1):28-41.

Challagulla, V. U. B., Bastani, F. B., Yen, I. ve Paul, R. A., (2005), "Empirical Assessment of Machine Learning based Software Defect Prediction Techniques", 10th International Workshop on Object-Oriented Real-Time Dependable Systems, 2-4 Şubat 2005, Sedona, AZ, ABD, 263-270.

Challagulla, V. U. B., Bastani, F. B. ve Yen, I. L., (2006), "A Unified Framework for Defect Data Analysis Using the MBR Technique", 18th IEEE International Conference on Tools with Artificial Intelligence, 13-15 Kasım 2006, Washington, ABD, 39-46.

Chapelle, O., Schölkopf, B. ve Zien, A., (2006), Semi-Supervised Learning, MIT Press, Cambridge, MA, ABD.

Chawla, N. V. ve Karakoulas, G. J., (2005), "Learning From Labeled And Unlabeled Data: An Empirical Study Across Techniques And Domains", Journal of Artificial Intelligence Research (JAIR), 23: 331-366.

Chidamber, S. R. ve Kemerer, C. F., (1994), "A Metrics Suite for Object-Oriented Design",

IEEE Transactions On Software Engineering, 20(6): 476-493.

Chulani, S., Boehm, B. ve Steece, B., (1999), "Bayesian Analysis of Empirical Software Engineering Cost Models", IEEE Transactions on Software Engineering, 25(4):573-583.

Clements, P. ve Northrop, L., (2002), Software Product Lines: Practices and Patterns, Addison-Wesley, Boston, ABD.

Cohen, W. W. ve Devanbu, P. T., (1997), "A Comparative Study of Inductive Logic Programming Methods for Software Fault Prediction", 14th International Conference on Machine Learning, 8-12 Temmuz 1997, Nashville, Tennessee, ABD, 66-74.

Cozman, F. G. ve Cohen, I., (2002), "Unlabeled Data can Degrade Classification Performance of Generative Classifiers", 15th International Florida Artificial Intelligence Research Society Conference, 14-16 Mayıs 2002, Florida, ABD, 327–331.

Cozman, F. G., Cohen, I. ve Cirelo, M. C., (2003), "Semi-supervised Learning of Mixture Models", International Conference on Machine Learning, 21-24 Ağustos 2003, Washington, ABD, 99–106.

Cukic, B. ve Ma, Y., (2007), "Predicting Fault-Proneness: Do We Finally Know How?", Reliability Analysis of System Failure Data, Cambridge, UK.

Çatal, Ç. ve Diri, B., (2007), "Software Fault Prediction With Object-Oriented Metrics Based Artificial Immune Recognition System", Product Focused Software Process Improvement (PROFES 2007), 2-4 Temmuz 2007, Riga, Letonya, 300-314.

Çırakoğlu, B., (2003a), "İmmünoloji ve Gelecek", TÜBİTAK Bilim ve Teknik Dergisi, Mart Sayısı Eki, 6-7.

Çırakoğlu, B., (2003b), "İmmünoloji ve Gelecek", TÜBİTAK Bilim ve Teknik Dergisi, Mart Sayısı Eki, 17-19.

De Almeida, M. A., Lounis, H. ve Melo, W., (1998), "An Investigation on the Use of Machine Learned Models for Estimating Correction Costs", International Conference on Software Engineering, 19-25 Nisan 1998, Kyoto, Japonya, 473-476.

De Almeida, M. A. ve Matwin, S., (1999), "Machine Learning Method for Software Quality Model Building", 11th International Symposium on Foundations of Intelligent Systems, 8-11 Haziran 1999, Varşova, Polonya, 565-573.

De Castro, L. N. ve Von Zubben, F. J., (2000a), "The Clonal Selection Algorithm with Engineering Applications", Artificial Immune Workshop, Genetic and Evolutionary Computation Conference (GECCO), 8-12 Temmuz 2000, Las Vegas, Nevada, ABD, 36-37.

De Castro, L. N. ve Von Zuben, F. J., (2000b), "An Evolutionary Immune Network for Data Clustering", 6th Brazilian Symposium on Neural Networks, 22-25 Kasım 2000, Rio de Janiero, Brezilya, 84-89.

De Castro, L. N. ve Timmis, J., (2002), Artificial Immune Systems: A New Computational Intelligence Approach, Springer-Verlag, Londra.

De Castro, L. N. and Timmis, J., (2003), "Artificial Immune Systems as a Novel Soft Computing Paradigm", Soft Computing, 7(8): 526-544.

Demiralp, E., (2003), "Bağışıklık Sisteminin Hücreleri", TÜBİTAK Bilim ve Teknik Dergisi,

Mart Sayısı Eki, 10-13.

Denaro, G., (2000), “Estimating Software Fault-proneness For Tuning Testing Activities”, 22nd International Conference on Software Engineering, 4-11 Haziran 2000, Limerick, İrlanda,704-706.

Denaro, G., Lavazza, L. ve Pezzè, M., (2003a), “An Empirical Evaluation of Object Oriented Metrics in Industrial Setting”, The 5th CaberNet Plenary Workshop, 5-7 Kasım 2003, Porto Santo, Portekiz.

Denaro, G., Pezzè, M. ve Morasca, S., (2003b), “Towards Industrially Relevant Fault-Proneness Models”, International Journal of Software Engineering and Knowledge Engineering, 13(4):395-417.

Dohi, T., Nishio, Y. ve Osaki, S., (1999), “Optimal Software Release Scheduling based on Artificial Neural Networks”, Annals of Software Engineering, 8(1):167-185.

Dolado, J., (2000), “A Validation of Component-based Method for Software Size Estimation”, IEEE Transactions on Software Engineering, 26(10):1006-1021.

El-Emam, K., Benlarbi, S. ve Goel, N., (1999), “Comparing Case-Based Reasoning Classifiers for Predicting High Risk Software Components”, Technical Report, National Research Council of Canada, NRC/ERB-1058, Kanada.

El-Emam, K., Benlarbi, S., Goel, N. ve Rai, S., (2001a), “Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components”, Journal of Systems and Software, 55(3):301-320.

El-Emam, K., Melo, W. ve Machado, J. C., (2001b), “The Prediction of Faulty Classes Using Object-Oriented Design Metrics”, Journal of Systems and Software, 56(1):63-75.

El-Emam, K., Benlarbi, S., Goel, N. ve Rai, S. N., (2001c), “The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics”, IEEE Transactions on Software Engineering, 27(7):630-650.

El-Wakil, M., El-Bastawisi, A., Boshra, M. ve Fahmy, A., (2004), “Software Metrics-A Taxonomoy”, 3rd International Conference on Computer Science, Software Engineering, Information Technology, e-Business,and Applications (CSITeA'04), 27-29 Aralık 2004, Kahire, Mısır.

Elworthy, D., (1994), “Does Baum-Welch Re-estimation Help Taggers?”, 4th Conference on Applied Natural Language Processing, 13-15 Ekim 1994, Stuttgart, Almanya, 53-58.

Erdağ, B., (2003), “Bağışıklık Sisteminin Hücreleri”, TÜBİTAK Bilim ve Teknik Dergisi, Mart Sayısı Eki, 14-16.

Evett, M., Khoshgoftaar, T., Chien, P. ve Allen, E., (1998), “GP-based software quality prediction”, 3rd Annual Genetic Programming Conference, 22-25 Temmuz 1998, Madison, Wisconsin, ABD, 60-65.

Fagan, M. E., (1986), "Advances in Software Inspections," IEEE Transactions on Software Engineering, 12(7), 744-751.

Farmer, J. D., Packard, N. H. ve Perelson, A. S., (1986), “The Immune System, Adaptation, and Machine Learning”, Physica, 22:187-204.

Fenton, N. E. ve Pfleeger, S., (1997), Software Metrics, PWS Publishing Company, Boston, ABD.

Fenton, N. E. ve Ohlsson, N., (2000), "Quantitative Analysis of Faults and Failures in a Complex Software System", IEEE Transactions on Software Engineering, 26(8): 797-814.

Forrest, S., Perelson, A., Allen, L. ve Cherukuri, R., (1994), "Self-Nonself Discrimination in a Computer", IEEE Symposium on Research in Security and Privacy, 16-18 Mayıs 1994, Los Alamitos, CA, ABD, 202-212.

Gao, K. ve Khoshgoftaar, T. M., (2007), "A Comprehensive Empirical Study of Count Models for Software Fault Prediction", IEEE Transactions for Reliability, 56(2):223-236.

Goldman, S. ve Zhou, Y., (2000), "Enhancing Supervised Learning with Unlabeled Data", 17th International Joint Conference on Machine Learning, Stanford, CA, ABD, 327-334.

Greenfield, J., Short, K., Cook, S. ve Kent, S., (2004), Software Factories, Assembling Applications with Patterns, Models, Framework, and Tools, Wiley Publishing, Indianapolis, ABD.

Guo, L., Cukic, B. ve Singh, H., (2003), "Predicting Fault Prone Modules by the Dempster-Shafer Belief Networks", 18th IEEE International Conference on Automated Software Engineering, 6-10 Ekim 2003, Montreal, Kanada, 249-252.

Guo, L., Ma, Y., Cukic, B. ve Singh, H., (2004), "Robust Prediction Of Fault-Proneness By Random Forests", 15th International Symposium on Software Reliability Engineering, 2-5 Kasım 2004, Brittany, Fransa, 417-428.

Guo, P. ve Lyu, M. R., (2000), "Software Quality Prediction Using Mixture Models with EM Algorithm", 1st Asia-Pacific Conference on Quality Software, 30-31 Ekim 2000, Hong Kong, Çin, 69-80.

Gyimothy, T., Ferenc, R. ve 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.

Halstead, M., (1977), Elements of Software Science, Elsevier North-Holland, New York.

Hand, D. J., (2005), "Good Practice in Retail Credit Scorecard Assessment", Journal of the Operational Research Society, 56: 1109-1117.

Hassan, A. E. ve Holt, R. C., (2005), "The Top Ten List: Dynamic Fault Prediction", 21st IEEE International Conference on Software Maintenance, 25-30 Eylül 2005, Budapeşte, Macaristan, 263-272.

Heeger, D., (1998), Signal Detection Theory, http://white.standford.edu/~heeger/sdt/sdt.html.

Hightower, R. R., Forrest, S. ve Perelson, A. S., (1995), "The Evolution of Emergent Organization in Immune System Gene Libraries", Sixth Annual Conference on Genetic Algorithms , 15-19 Temmuz 1995, Pittsburgh, PA, ABD, 344 -350.

Huang, T. M. ve Kecman, V., (2005), "Performance Comparisons of Semi-Supervised Learning Algorithms", Workshop on Learning with Partially Classified Training Data, International Conference on Machine Learning, 7-11 Ağustos 2005, Bonn, Almanya, 45-49.

Ishiguro, A., Watanabe, Y. ve Kondo, T., (1997), "A Robot with a Decentralized Consensus-

Making Mechanism Based on the Immune System", 3rd International Symposium on Autonomous Decentralized Systems, 9-11 Nisan 1997, Berlin, Almanya, 231-237.

Jerne, N. K., (1974), "Towards a Network Theory of the Immune System", Ann. Immunol. (Inst. Pasteur), 125C: 373–389.

Jiang, Y., Cukic, B. ve Menzies, T., (2007), "Fault Prediction Using Early Lifecycle Data", 18th IEEE International Symposium on Software Reliability, 5-9 Kasım 2007, Trollhättan, İsveç, 237-246.

Joachims, T., (1999), "Transductive Inference for Text Classification using Support Vector Machines", International Conference on Machine Learning, 27-30 Haziran 1999, Bled, Slovenya, 200–209.

Jorgensen, M., (1995), "Experience with the Accuracy of Software Maintenance Task Effort Prediction Models", IEEE Transactions on Software Engineering, 21(8):674-681.

Kaaniche, M. ve Kanoun, K., (1996), "Reliability of a Commercial Telecommunications System", 7th International Symposium on Software Reliability Engineering, 30 Ekim-2 Kasım 1996, White Plains, New York, ABD, 207-212.

Kaminsky, K. ve Boetticher, G., (2004), "How to Predict More with Less, Defect Prediction Using Machine Learners in an Implicitly Data Starved Domain", The 8th World Multi-Conference on Systemics, Cybernetics and Informatics, 18-21 Temmuz 2004, Orlando, FL, ABD.

Kanmani, S., Uthariaraj, V. R., Sankaranarayanan, V., ve Thambidurai, P., (2004), "Object Oriented Software Quality Prediction Using General Regression Neural Networks", SIGSOFT Software Engineering Notes, 29(5):1-6.

Karaboğa, D., (2004), Yapay Zekâ Optimizasyon Algoritmaları, Atlas Yayınevi, Yayın No:38, İstanbul.

Karunanithi, N., Whitely, D. ve Malaiya, Y., (1992), "Prediction of Software Reliability using Connectionist Models", IEEE Transactions on Software Engineering, 18(7):563-574.

Kaszycki, G., (1999), "Using Process Metrics to Enhance Software Fault Prediction Models", 10th International Symposium on Software Reliability Engineering, 1-4 Kasım 1999, Boca Raton, Florida, ABD.

Khoshgoftaar, T. M., Allen, E. B., Hudepohl, J. P. ve 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. ve Busboom, J. C., (2000a), "Modeling Software Quality: the Software Measurement Analysis and Reliability Toolkit", 12th IEEE International Conference on Tools with Artificial Intelligence, 13-15 Kasım 2000, Vancouver, BC, Kanada, 54-61.

Khoshgoftaar, T., Allen, E. ve Xu, Z., (2000b), "Predicting Testability of Program Modules using a Neural Network", IEEE Symposium on Application-Specific Systems and Software Engineering Technology, Washington, ABD, 57-62.

Khoshgoftaar, T., Gao, K. ve Szabo, R. M., (2001), "An Application of Zero-Inflated Poisson Regression for Software Fault Prediction", 12th International Symposium on Software Reliability Engineering, Washington, ABD, 66-73.

Khoshgoftaar, T. M., Geleyn, E. ve Gao, K., (2002a), "An Empirical Study of the Impact of Count Models Predictions on Module-Order Models", 8th International Symposium on Software Metrics, 4-7 Haziran 2002, Ottawa, Kanada, 161-172.

Khoshgoftaar, T. M., (2002b), "Improving Usefulness of Software Quality Classification Models Based on Boolean Discriminant Functions", 13th International Symposium on Software Reliability Engineering, 12-15 Kasım 2002, Annapolis, MD, ABD, 221-230.

Khoshgoftaar, T. M. ve Seliya, N., (2002c), "Software Quality Classification Modeling Using The SPRINT Decision Tree Algorithm", 4th IEEE International Conference on Tools with Artificial Intelligence, 4-6 Kasım 2002, Washington, ABD, 365-374.

Khoshgoftaar, T. M. ve Seliya, N., (2002d), "Tree-Based Software Quality Estimation Models for Fault Prediction", 8th IEEE Symposium on Software Metrics, 4-7 Haziran 2002, Ottawa, Kanada, 203-215.

Khoshgoftaar, T. M. ve Seliya, N., (2003), "Fault Prediction Modeling for Software Quality Estimation: Comparing Commonly Used Techniques", Empirical Software Engineering, 8(3): 255-283.

Khoshgoftaar, T. M. ve 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. ve Gao, K., (2005), "Assessment of a New Three-Group Software Quality Classification Technique: An Empirical Case Study", Empirical Software Engineering, 10(2):183-218.

Khoshgoftaar, T. M., Seliya, N. ve Sundaresh, N., (2006), "An Empirical Study of Predicting Software Faults with Case-Based Reasoning", Software Quality Journal, 14(2):85-111.

Kim, J. W., (2002), "Integration Artificial Immune Algorithms for Intrusion Detection", Doktora Tezi, University College London, Bilgisayar Bilimleri Bölümü.

Kolb, R. ve Muthig, D., (2004), "Quality Assurance for Software Product Lines", 3rd Software Product Line Conference, Boston, MA, ABD, 312.

Kolb, R. ve Muthig D., (2006), Making Testing Product Lines More Efficient by Improving the Testability of Product Line Architectures, "2nd International Workshop on the Role of Software Architecture for Testing and Analysis", Portland, Maine, ABD, 22-27.

Koru, A. G. ve Tian, J., (2003), "An Empirical Comparison and Characterization of High Defect and High Complexity Modules", Journal of Systems and Software, 67(3):153-163.

Koru, A. G. ve Liu, H., (2005a), "An Investigation of the Effect of Module Size on Defect Prediction Using Static Measures", Workshop on Predictor Models in Software Engineering, St. Louis, Missouri, ABD, 1-5.

Koru, A. G. ve Liu, H., (2005b), "Building Effective Defect Prediction Models in Practice" IEEE Software, 22(6):23-29.

Koru, A. G. ve Liu, H., (2007), "Identifying and Characterizing Change-Prone Classes in Two Large-Scale Open-Source Products", Journal of Systems and Software, 80(1):63-73.

Laird, L. M. ve Brennan, M. C., (2006), Software Measurement and Estimation: A Practical Approach, John Wiley & Sons, Inc., Hoboken, New Jersey, ABD.

Langman, R. E. ve Cohn, M., (1986), "The Complete Idiotype Network is an Absurd Immune System", Imm. Today, 7(4): 100-101.

Lanubile, F., Lonigro, A. ve Visaggio, G., (1995), "Comparing Models for Identifying Fault-Prone Software Components", Seventh International Conference on Software Engineering and Knowledge Engineering, 22-24 Haziran 1995, Rockville, Maryland, ABD, 312–319.

Lanubile, F. ve Visaggio, G., (1997), "Evaluating Predictive Quality Models Derived from Software Measures: Lessons Learned", Journal of Systems and Software, 38: 225-234.

Laprie, J. C., (1992), Dependability: Basic Concepts and Terminology, Springer-Verlag, Viyana.

Li, X., Wang, L. ve Sung, E., (2007), "AdaBoost with SVM-based Component Classifiers", Engineering Applications of Artificial Intelligence, 14 Eylül 2007 Çevrim İçi Erişilebilir, (baskıda).

Li, Z. ve Reformat M., (2007), "A Practical Method for the Software Fault-Prediction", IEEE International Conference on Information Reuse and Integration, 13-15 Ağustos 2007, Las Vegas, Nevada, ABD, 659-666.

Linden, F., (2002), "Software Product Families in Europe: The ESAPS and CAFE projects", IEEE Software, 19(4):41-49.

Ling, C. X., Huang, J. ve Zhang, H., (2003), "AUC: A Better Measure Than Accuracy in Comparing Learning Algorithms", Canadian Conference on Artificial Intelligence, 11-13 Haziran 2003, Halifax, Kanada, 329-341.

Ma, Y., Guo, L. ve Cukic, B., (2006), "A Statistical Framework for the Prediction of Fault-Proneness", Advances in Machine Learning Application in Software Engineering, Idea Group Inc., 237-265.

Mahaweerawat, A., Sophasathit, P. ve Lursinsap, C., (2002), "Software Fault Prediction Using Fuzzy Clustering and Radial Basis Function Network", International Conference on Intelligent Technologies, 3-5 Aralık 2002, Vietnam, 304-313.

Mahaweerawat, A., Sophatsathit, P., Lursinsap, C. ve Musilek, P., (2004), "Fault Prediction in Object-Oriented Software Using Neural Network Techniques", InTech Conference, 2-4 Aralık 2004, Houston, TX, ABD, 27-34.

Mahaweerawat, A., Sophatsathit, P. ve Lursinsap, C., (2007), "Adaptive Self-Organizing Map Clustering for Software Fault Prediction", 4th International Joint Conference on Computer Science and Software Engineering, 2-4 Mayıs 2007, Khon Kaen, Tayland, 35-41.

Mao, Y., Sahraoui, H. ve Lounis, H., (1998), "Reusability Hypothesis Verification using Machine Learning Techniques: A Case Study", 13th IEEE International Conference on Automated Software Engineering, 13-16 Ekim 1998, Honolulu, Hawaii, ABD, 84-93.

McCabe, T., (1976), "A Complexity Measure", IEEE Transactions on Software Engineering, 2(4): 308-320.

McIlroy, M. D., (1968), "Mass-Produced Software Components", 1st International NATO Conference on Software Engineering, 7-11 Ekim 1968, Garmisch, Almanya, 88-98.

Menzies, T., Di Stefano, J., Ammar, K., McGill, K., Callis, P., Chapman, R. ve Davis, J., (2003), "When Can We Test Less?", 9th International Symposium on Software Metrics, 3-5

Eylül 2003, Sidney, Avustralya, 98.

Menzies, T., DiStefano, J., Orrego, A. ve Chapman, R., (2004a), "Assessing Predictors of Software Defects", Predictive Software Models Workshop, 12th International Workshop on Software Technology and Engineering Practice (STEP 2004), 17-19 Eylül 2004, Chicago, Illinois, ABD.

Menzies, T. ve Di Stefano, J. S., (2004b), "How Good Is Your Blind Spot Sampling Policy?", 8th IEEE International Symposium on High-Assurance Systems Engineering, 25-26 Mart 2004, Tampa, FL, ABD, 129-138.

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

Menzies, T., Dekhtyar, A., Distefano, J. ve Greenwald, J., (2007b), "Problems with Precision: A Response to Comments on Data Mining Static Code Attributes to Learn Defect Predictors", IEEE Transactions on Software Engineering, 33(9):637-640.

Menzies, T., Turhan, B., Bener, A., Gay, G., Cukic, B. ve Jiang, Y., (2008), "Implications of Ceiling Effects in Defect Predictors", 4th International Workshop on Predictor Models in SE (PROMISE 2008), 12-13 Mayıs 2008, Leipzig, Almanya.

Mertik, M., Lenic, M., Stiglic, G. ve Kokol, P., (2006), "Estimating Software Quality with Advanced Data Mining Techniques", International Conference on Software Engineering Advances, 28 Ekim-2 Kasım, Papeete, Tahiti, French Polynesia, 19.

Miller, D. J. ve Uyar, H. S., (1996), "A Mixture of Experts Classifier with Learning Based on Both Labeled and Unlabelled Data", Neural Information Processing Systems, 2-5 Aralık 1996, Colorado, ABD, 571–577.

Munson, J. C., (2003), Software Engineering Measurement, CRC Press Company, Boca Raton London, New York.

Nigam, K. ve Ghani, R., (2000a), "Analyzing the Effectiveness and Applicability of Co-training", Ninth International Conference on Information and Knowledge Management, 6-11 Kasım 2000, Washington, ABD, 86–93.

Nigam, K., McCallum, A. K., Thrun, S. ve Mitchell, T., (2000b), "Text Classification from Labeled and Unlabeled Documents using EM", Machine Learning, 39:103–144.

Nikora, A. P. ve Munson, J. C., (2006), "Building High-Quality Software Fault Predictors", Software-Practice and Experience, 36(9): 949-969.

Northrop, L., Feiler, P., Gabriel, R. P., Goodenough, J., Linger, R., Longstaff, T., Kazman, R., Klein, M., Schmidt, D., Sullian, K. ve Wallnau, K., (2006), Ultra-Large-Scale Systems: The Software Challange of the Future, Carnegie Mellon University, Pittsburgh, ABD.

Ohlsson, N. ve Alberg, H., (1996), "Predicting Fault-prone Software Modules in Telephone Switches", IEEE Transactions on Software Engineering, 22(12):886-894.

Ohlsson, N., Zhao, M. ve Helander, M., (1998), "Application of Multivariate Analysis for Software Fault Prediction", Software Quality Journal, 7(1): 51-66.

Olague, H. M., Gholston, S. ve 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.

Ostrand, T. J., Weyuker, E. J. ve Bell, R. M., (2005), "Predicting the Location and Number of Faults in Large Software Systems", IEEE Transactions on Software Engineering, 31(4): 340-355.

Ostrand, T. J., Weyuker, E. J. ve Bell, R. M., (2007), "Automating Algorithms for the Identification of Fault-Prone Files", International Symposium on Software Testing and Analysis, 9-12 Temmuz 2007, Londra, UK, 219-227.

Pai, G. J. ve Dugan, J. B., (2007), "Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods", IEEE Transactions on Software Engineering, 33(10):675-686.

Paton, R., (1992), "Towards a Metaphorical Biology", Biology and Philosophy, 7:279-294.

Perelson, A. S. ve Oster, G. F., (1979), "Theoretical Studies of Clonal Selection: Minimal Antibody Repertoire Size and Reliability of Self-Nonself Discrimination", Journal of Theoretical Biology, 81: 645-670.

Pizzi, N. J., Summers, R. ve Pedrycz, W., (2002), "Software Quality Prediction Using Median-Adjusted Class Labels", International Joint Conference on Neural Networks, 12-17 Mayıs 2002, Honolulu, HI, ABD, 2405- 2409.

Pohl, K., Bockle, G. ve Linden, F., (2005), Software Product Line Engineering, Springer-Verlag, New York, ABD.

Porter, A. A. ve Selby, R. W., (1990), "Empirically Guided Software Development Using Metric-Based Classification Trees", IEEE Software, 7(2): 46-54.

Pullum, L. L., (2001), Software Fault Tolerance: Techniques and Implementation, Artech House, Norwood, MA, ABD.

Reformat, M., (2003), "A Fuzzy-Based Meta-model for Reasoning about Number of Software Defects", 10th International Fuzzy Systems Association World Congress, 29 Haziran-2 Temmuz 2003, İstanbul, 644-651.

Rosenberg, L., Stapko, R. ve Gallo, A., (1999), "Object-Oriented Metrics for Reliability", IEEE International Symposium on Software Metrics, 4-6 Kasım 1999, Boca Raton, FL, ABD.

Sarıdoğan, M. E., (2004), Yazılım Mühendisliği, Papatya Yayıncılık, İstanbul.

Sayyad, S. J. ve Menzies, T. J., (2005), The PROMISE Repository of Software Engineering Databases, University of Ottowa, Kanada.

Schneidewind, N. F., (2001), "Investigation of Logistic Regression as a Discriminant of Software Quality", 7th International Symposium on Software Metrics, 4-6 Nisan 2001, Londra, UK, 328-337.

Seliya, N., Khoshgoftaar, T. M. ve Zhong, S., (2004), "Semi-Supervised Learning for Software Quality Estimation", 16th IEEE International Conference on Tools with Artificial Intelligence, 15-17 Kasım 2004, Boca Raton, FL, ABD, 183-190.

Seliya, N. ve Khoshgoftaar T. M., (2007a), "Software Quality Estimation with Limited Fault Data: A Semi-Supervised Learning Perspective", Software Quality Journal, 15(3):327-344.

Seliya, N. ve Khoshgoftaar T. M., (2007b), "Software Quality Analysis of Unlabeled Program

Modules With Semisupervised Clustering", IEEE Transactions on Systems, Man, and Cybernetics, 37(2):201-211.

Shahshahani, B. M. ve Landgrebe, D. A., (1994), "The Effect of Unlabeled Samples in Reducing the Small Sample Size Problem and Mitigating the Hughes Phenomenon", IEEE Transactions on Geoscience and Remote Sensing, 32:1087–1095.

Shepperd, M. ve Schofield, C., (1997), "Estimating Software Project Effort using Analogies, IEEE Transactions on Software Engineering", 23(12):736-743.

Shull, F., Rus, I. ve Basili, V., (2000), "How Perspective-Based Reading Can Improve Requirements Inspections," IEEE Computer, 33(7):73-79.

Srinivasan, K. ve Fisher, D., (1995), "Machine Learning Approaches to Estimating Software Development Effort", IEEE Transactions on Software Engineering, 21(2):126-137.

Tang, M. H., Kao, M. H. ve Chen, M. H., (1999), "An Empirical Study on Object-Oriented Metrics", Sixth International Software Metrics Symposium, 4-6 Kasım 1999, Boca Raton, FL, ABD, 242-249.

Thwin, M. M. ve Quah, T., (2003), Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics, 19th International Conference on Software Maintenance, 22-26 Eylül 2003, Amsterdam,Hollanda, 113-122.

Tian, J., (2005), Software Quality Engineering: Testing, Quality Assurance and Quantifiable Improvement, John Wiley and Sons Inc., Hokoben.

Timmis, J., (2000a), "Artificial Immune Systems: A Novel Data Analysis Technique Inspired by the Immune Network Theory", Doktora Tezi, Department of Computer Science, University of Wales, Galler.

Timmis, J. ve Neal, M., (2000), "Investigating the Evolution and Stability of a Resource Limited Artificial Immune Systems", Genetic and Evolutionary Computation Conference, Workshop Program, 8-12 Temmuz 2000, Las Vegas, Nevada, ABD, 40-41.

Timmis, J., (2008), www.artificial-immune-systems.org/courses/Lectures/lecture6.pdf.

Tomaszewski, P., Lundberg, L. ve Grahn, H., (2005), "The Accuracy of Early Fault Prediction in Modified Code", 5th Conference on Software Engineering Research and Practice in Sweden, 20-21 Ekim 2005, Västerås, İsveç, 57-63.

Tomaszewski, P., Håkansson, J., Grahn, H. ve Lundberg, L., (2007), "Statistical Models vs. Expert Estimation for Fault Prediction in Modified Code - an Industrial Case Study", Journal of Systems and Software, 80(8):1227-1238.

Trendowicz, A. ve Punter, T., (2003), "Quality Modeling for Software Product Lines", 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, 21-25 Temmuz 2003, Darmstadt, Almanya, 7.

Van Hulse, J., Khoshgoftaar, T. M. ve Napolitano, A., (2007), "Experimental Perspectives on Learning from Imbalanced Data", 24th International Conference on Machine Learning (ICML), 20-24 Haziran 2007, Corvalis, Oregon, ABD, 935-942.

Vapnik, V. ve Chervonenkis, A., (1974), Theory of Pattern Recognition, Nauka, Moskova, Rusya.

Wang, Q., Yu, B. ve Zhu, J., (2004), "Extract Rules from Software Quality Prediction Model Based on Neural Network", 16th IEEE International Conference on Tools with Artificial Intelligence, 15-17 Kasım 2004, Boca Raton, FL, ABD, 191-195.

Wang, Q., Zhu, J. ve Yu, B., (2007), "Feature Selection and Clustering in Software Quality Prediction", 11th International Conference on Evaluation and Assessment in Software Engineering, 2-3 Nisan 2007, Keele, UK.

Watkins, A., (2001), AIRS: A Resource Limited Artificial Immune Classifier, Yüksek Lisans Tezi, Mississippi State University, Mississippi, ABD.

Watkins, A., (2005), Exploiting Immunological Metaphors in the Development of Serial, Paralel, and Distributed Learning Algorithms, Doktora Tezi, The University of Kent, UK.

Watson, A. H. ve McCabe T. J., (1996), "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric", Technical Report 500-235, NIST Computer Systems Laboratory.

Weiss, D. M. ve Lai, C. T. R., (1999), Software Product-line Engineering-A Family-based Software Development Process, Addison-Wesley, Reading, MA.

Xie, M., (1991), Software Reliability Modeling, World Scientific, Singapore.

Xing, F., Guo, P. ve Lyu, M. R., (2005), "A Novel Method for Early Software Quality Prediction Based on Support Vector Machine", 16th IEEE international Symposium on Software Reliability Engineering, 8-11 Kasım 2005, Chicago, IL, ABD, 213-222.

Xu, Z., Khoshgoftaar, T. M. ve Allen, E. B., (2000), "Prediction of Software Faults Using Fuzzy Nonlinear Regression Modeling", 5th IEEE International Symposium on High Assurance Systems Engineering, 15-17 Kasım 2000, Albuquerque, New Mexico, 281-290.

Yang, B., Yao, L. ve Huang, H. Z., "Early Software Quality Prediction Based on a Fuzzy Neural Network Model", Third International Conference on Natural Computation, 24-27 Ağustos 2007, Haikou, Çin, 760-764.

Yarowsky, D., (1995), "Unsupervised Word Sense Disambiguation Rivaling Supervised Methods", 33rd Ann. Meeting of the Assoc. for Compt. Linguistics, 26-30 Haziran 1995, Cambridge, Mass, 189–196.

Youden, W., (1950), "Index for Rating Diagnostic Tests", Cancer, 3(1): 32-35.

Yu, P., Systa, T. ve Muller, H., (2002), "Predicting Fault-Proneness Using OO Metrics", Sixth European Conference on Software Maintenance and Reengineering, 11-13 Mart 2002, Budapeşte, Macaristan, 99-197.

Yuan, X., Khoshgoftaar, T. M., Allen, E. B. ve Ganesan, K., (2000), "An Application of Fuzzy Clustering to Software Quality Prediction", 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, 24-25 Mart 2000, Richardson, Texas, ABD, 85.

Yücel, F., (2003), "Bağışıklığın Akıllı Molekülleri: Antikorlar", TÜBİTAK Bilim ve Teknik Dergisi, Mart Sayısı Eki, 2-5.

Zadeh, L. A., (1997), "What is Soft Computing", Soft Computing, 1(1): 1.

Zhang, D. ve Tsai, J. J. P., (2003), "Machine Learning and Software Engineering", Software

Quality Journal, 11(2):87-119.

Zhang, H., Jarzabek, S. ve Yang, B., (2003), "Quality Prediction and Assessment for Product Lines", Conference on Advanced Information Systems Engineering, 16-18 Haziran 2003, Klagenfurt, Avusturya, 681-695.

Zhang, H. ve Zhang, X., (2007), "Comments on Data Mining Static Code Attributes to Learn Defect Predictors", IEEE Transactions on Software Engineering, 33(9):635-637.

Zhong, S., Khoshgoftaar, T. M. ve Seliya, N., (2004), "Unsupervised Learning for Expert-Based Software Quality Estimation", 8th IEEE International Symposium on High Assurance Systems Engineering, 25-26 Mart 2004, Tampa, FL, ABD, 149-155.

Zhou, Y. ve 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.

Zhu, X., (2005), "Semi-supervised Learning Literature Survey", Technical Report 1530, University of Wisconsin-Madison, ABD.

Zhu, X., Ghahramani, Z. ve Lafferty, J., (2003), "Semi-supervised Learning using Gaussian Fields and Harmonic Functions", 20th International Conference on Machine Learning, 21-24 Ağustos 2003, Washington, ABD, 912-919.

ÖZGEÇMİŞ

Doğum tarihi	20.01.1981	
Doğum yeri	Ceyhan	
Lise	1994–1998	Pertevniyal Lisesi, İstanbul
Lisans	1998–2002	İTÜ Elektrik-Elektronik Fakültesi Bilgisayar Mühendisliği
Yüksek Lisans	2002–2004	İTÜ Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği
Doktora	2004–2008	YTÜ Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği

Çalıştığı kurumlar

	2002–Devam	TÜBİTAK-Marmara Araştırma Merkezi Bilişim Teknolojileri Enstitüsü, Proje Yöneticisi, Uzman Araştırmacı