T.C.
TRAKYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ESNEK HESAPLAMA YAKLAŞIMI İLE YAZILIM HATA KESTİRİMİ

Ebru ARDIL
Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Yrd. Doç. Dr. Erdem UÇAR
2009
EDİRNE

TRAKYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ESNEK HESAPLAMA YAKLAŞIMI İLE YAZILIM HATA KESTİRİMİ

EBRU ARDIL

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN: YRD. DOÇ. DR. ERDEM UÇAR

2009
EDİRNE

T.C.
TRAKYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ESNEK HESAPLAMA YAKLAŞIMI İLE YAZILIM HATA KESTİRİMİ

EBRU ARDIL

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Bu tez 20/03/2009 tarihinde aşağıdaki jüri tarafından kabul edilmiştir.

Yrd. Doç. Dr. Erdem UÇAR
(Danışman)

Yrd. Doç. Dr. Rafet AKDENİZ

Yrd. Doç. Dr. Özlem UÇAR

Yüksek Lisans Tezi
Trakya Üniversitesi Fen Bilimleri Enstitüsü

ESNEK HESAPLAMA YAKLAŞIMI İLE YAZILIM HATA KESTİRİMİ

ÖZET

Bir yazılım sistemi geliştirildiğinde, hataların (fault) ekseriyeti modüllerin ufak bir kısmında bulunur. Birçok durumda, hataların %55'i kaynak kodunun yüzde 20'lik kısmında oluşur. Bu, hata-eğilimli yazılım modüllerinin projenin erken aşamasında bulunması yönünde ilgi oluşturmuştur. (Saida Benlarbi, Khaled El Emam, ve Nishith Geol, 1999). Yazılım karmaşıklık ölçütlerini kullanan teknikler, bileşenleri hata barındıran veya barındırmayan olarak sınıflandıran modeller geliştirirler. Bununla birlikte, algılanan hata sayısı arttıkça kalite de artmış olacaktır. Yazılım yaşam çevriminde hataların etkilerinin erken kestirimi, yazılım süreci kontrolünü geliştirme ve yüksek yazılım güvenilirliği elde etmede kullanılabilir. Yazılım modüllerinde hataların zamanında kestirimi, yüksek sayıda hata barındırıyor olması muhtemel modüllerin maliyet etkin (cost-effective) kalite arttırma çabalarını yönlendirmede kullanılabilir. Yazılım ölçütlerine dayanan kestirim modelleri, yazılım modüllerinde bir miktar hatayı kestirebilir. Hataların aciliyetinin kestirimi:

- Geliştirilmiş zamanlama (scheduling) ve proje kontrolü ile yazılım kalitesi mühendisliğini destekler.
- Yazılım testinin yönetimi ve bütün sürecin etkinliğini arttırma yönünde anahtar bir adım olabilir.
- Hataların etkin keşfine ve tanımlanmasına izin verir.
- Doğrulama ve kritik yazılım bileşenlerine odaklanan onaylama aktivitelerine izin verir.

- Yazılım süreci kontrolünün ve yüksek yazılım güvenilirliğinin geliştirmesinde kullanılabilir.
- Modüllerde maliyet etkin kalite arttırma çabalarını yönlendirmede kullanılabilir.

Bu çalışmada, farklı kestirim modellerini incelemek için Perl programlama dilinde kodlanan NASA'nın kamusal alan (public domain) hata veri kümesini kullandık. WEKA projesinin farklı öğrenici kategorilerine ait olan farklı makine öğrenmesi algoritmaları, Mamdani Tabanlı Bulanık Çıkarım Sistemi (Fuzzy Inference System) ve Uyarlamalı Nöral-Bulanık (Neuro-Fuzzy) Tabanlı Sistemler ile yazılım bakımında hata kestiriminin önemi değerlendirildi. Sonuçlar *Doğruluk* (Accuracy), *Ortalama Mutlak Hata* (Mean Absolute Error, MAE), ve *Kök Hata Kareler Ortalaması* (Root Mean Squared Error, RMSE) olarak kaydedildi. Sonuçlar Uyarlamalı Nöral-Bulanık (Neuro-Fuzzy) Tabanlı modelin diğer modellere nispeten daha doğru kestirim sağladığı ve bu nedenle yazılımın bakım aciliyeti kestiriminde kullanılabileceğini gösterdi.

Anahtar kelimeler: Yazılım hataları, Yazılım ölçütleri, Makine Öğrenmesi Algoritmaları, Bulanık, Nöral-Bulanık, Doğruluk, MAE, RMSE.

Yıl:2009
Sayfa: 83

Master Thesis
Trakya University Graduate School of
Natural and Applied Sciences
Department of Computer Engineering

SOFTWARE MAINTENANCE SEVERITY PREDICTION WITH SOFT COMPUTING APPROACH

ABSTRACT

When a software system is developed, the majority of faults are found in a few of its modules. In most of the cases, 55 % of faults exist within 20 % of source code. It is, therefore, much of interest is to find out fault-prone software modules at early stage of a project [1]. Using software complexity measures, the techniques build models, which classify components as likely to contain faults or not. Quality will be improved as more faults will be detected. Predicting the impact of the faults early in the software life cycle can be used to improve software process control and achieve high software reliability. Timely predictions of faults in software modules can be used to direct cost-effective quality enhancement efforts to modules that are likely to have a high number of faults. Prediction models based on software metrics, can estimate number of faults in software modules. Prediction of severity of faults:

- Supports software quality engineering through improved scheduling and project control.
- Can be a key step towards steering the software testing and improving the effectiveness of the whole process.
- Enables effective discovery and identification of defects.
- Enables the verification and validation activities focused on critical software components.

- Used to improve software process control and achieve high software reliability.
- Can be used to direct cost-effective quality enhancement efforts to modules.

In this research, we have explored the different predictor models to NASA's public domain defect dataset coded in Perl programming language. Different machine learning algorithms belonging to the different learner categories of the WEKA Project including Mamdani Based Fuzzy Inference System and Adaptive Neuro-fuzzy based system have been evaluated for the software maintenance severity prediction. The results are recorded in terms of *Accuracy, Mean Absolute Error* (MAE) and *Root Mean Squared Error* (RMSE). The results show that Adaptive Neuro-fuzzy based model provides relatively better prediction accuracy as compared to other models and hence, can be used for the maintenance severity prediction of the software.

Keywords: Software Faults, Software Metrics, Machine Learning Algorithms, Fuzzy, Neuro-Fuzzy, Accuracy, MAE, RMSE.

Year: 2009

Page: 83

ÖNSÖZ

Bu tez çalışmasının birinci bölümünde yazılım sistemlerinde görülen hatalara ilişkin bilgiler verilmiştir. İkinci bölümünde literatür incelemesi yapılmış, farklı durumlar için araştırmacılar tarafından kullanılan çeşitli makine öğrenmesi algoritmaları değerlendirilmiştir. Üçüncü bölümünde esnek hesaplama kavramından söz edilerek, makine öğrenmesi algortimaları ve bulanık çıkarım sistemleri irdelenmiştir. Dördüncü bölümde uygulama yapılırken kullanılan materyal ve metotlardan söz edilmiş, amaca ve sonuca ulaşmak için kullanılan adımlardan bahsedilmiştir. Beşinci bölümde çalışmanın sonuçları verilmiş ve değerlendirilmiş, önerilen çalışma üzerinde yapılan uygulama ve sonuçları gösterilmiştir. Altıncı bölümde sonuç olarak çıkartılacak olgulara yer yerilmiştir ve sunulan çalışmanın geleceğe yönelik amaçları anlatılmıştır.

Araştırmalarım sırasında yardımlarını esirgemeyen, önerileriyle araştırmanın gelişimine ışık tutan danışmanım Yrd. Doç. Dr. Erdem UÇAR'a, fikirleriyle bana yön veren Prof. Dr. Parvinder Singh SANDHU'ya, çalışmalarım sırasında manevi desteğini daima hissettiğim aileme en içten teşekkürlerimi sunmaktan onur duyarım.

Ebru ARDIL

İÇİNDEKİLER

ÖZET i

ABSTRACT iii

ÖNSÖZ v

İÇİNDEKİLER vi

TABLOLAR LİSTESİ viii

ŞEKİLLER LİSTESİ ix

1. GİRİŞ 1

2. LİTERATÜR ARAŞTIRMASI 4

2.1. Hata Kestirimi ile ilgili Literatür Atıfları 4

2.2. Makine Öğrenmesi Algoritmaları ile ilgili Literatür Atıfları 7

3. ESNEK HESAPLAMA YÖNTEM VE TEKNİKLERİ 9

3.1. Makine Öğrenmesi Teknikleri 9

3.1.1. Makine öğrenmesi algoritmalarının Bayes sınıfı 14

3.1.2. Makine öğrenmesi algoritmalarının Functions sınıfı 18

3.1.3. Makine öğrenmesi algoritmalarının Lazy sınıfı 19

3.1.4. Makine öğrenmesi algoritmalarının Meta sınıfı 20

3.1.5. Makine Öğrenmesi Algoritmalarının Misc Sınıfı 24

3.1.6. Makine öğrenmesi algoritmalarının Trees sınıfı 25

3.2. Mamdani Tabanlı Bulanık Çıkarım Sistemi 28

3.3. Uyarlamalı Nöral-Bulanık Sistemler 32

4. MATERYAL VE METOD 35

4.1. Amaçlar 36

4.2. Metodoloji 36

4.2.1. Yapısal kodun ve tasarım özelliklerinin bulunması 36

4.2.2. İfadenin sunumu olarak uygun ölçüt değerlerinin seçimi 36

4.2.3. Ölçütlerin analiz-tasfiyesi ve ölçüt değerlerinin standartlaştırılması 42

4.2.4. Yazılım sistemlerinde hata toleransı kestirimi için farklı bir algoritmanın keşfedilmesi 42

4.2.5. Algoritmaların karşılaştırılması 43
4.2.5.1. Doğruluk değeri 43
4.2.5.2. Ortalama Mutlak Hata (Mean Absolute Error) 44
4.2.5.3. Kök Hata Kareler Ortalaması (RMSE: Root Mean-Squared Error) 44
4.2.5.4. Kappa istatistiği (Cohen's kappa coefficient) 45
4.2.6. Sonuçlar 46
5. SONUÇLAR VE TARTIŞMA 47
5.1. Ölçütler 47
5.2. Veri Kümesi 50
5.3. Makina Öğrenmesi Algoritmalarından Elde Edilen Sonuçlar 51
5.4. Mamdani Tabanlı Bulanık Çıkarım Sistemi Sonuçları 58
5.5. Uyarlamalı Nöral-Bulanık Tabanlı Çıkarım Sistemi Sonuçları 60
6. SONUÇ VE GELECEK KAPSAM 63
KAYNAKLAR 65
ÖZGEÇMİŞ 74

TABLOLAR LİSTESİ

Tablo 1.1. Başarısızlığın Sınıflandırılması 2
Tablo 4.1. Makine Öğrenmesi Algoritmalarının Sınıfları 42
Tablo 4.2. 2-sınıflı problem için karmaşıklık matrisi 44
Tablo 5.1. Ölçütlerin Listesi 48
Tablo 5.2. Veri Kümesindeki modüllerin detayları 50
Tablo 5.3. Eğitim Fazı ve Bayes Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları 54
Tablo 5.4. Eğitim Fazı ve Lazy Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları 54
Tablo 5.5. Eğitim Fazı ve Misc Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları 55
Tablo 5.6. Eğitim Fazı ve Trees Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları 55
Tablo 5.7. Eğitim Fazı ve Meta Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları 56
Tablo 5.8. Eğitim Fazı ve Function Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları 57

ŞEKİLLER LİSTESİ

Şekil 1.1. Yazılım geliştirme modelinin şelale modelinde gösterimi 3
Şekil 3.1. Sınıflandırma 12
Şekil 3.2. Bir sınıflandırma modeli kurulumu için genel yaklaşım 12
Şekil 3.3. WEKA mimarisi 13
Şekil 3.4. Karar ağaçları (a) varsayımsal (hypothetical) derinlik- ilk karar ağacı (b) varsayımsal en iyi-ilk karar ağacı 26
Şekil 3.5. Bulanık kümenin üyelik fonksiyonu 28
Şekil 3.6. Bulanık Çıkarım Sistemi Şeması 29
Şekil 3.7. Mamdani Nöral-Bulanık Sistem Yapısı (Abraham, 2005) 31
Şekil 5.1. Veri Kümesindeki modül detaylarının grafiksel gösterimi 50
Şekil 5.2. Grafiksel Kullanıcı Arayüzünün ekran görüntüsü 52
Şekil 5.3. Elde edilen sonuçların ekran görüntüsü 53
Şekil 5.4. Otomatik sınıflandırma 58
Şekil 5.5. Mamdani Tabanlı Bulanık Çıkarım Sistemi 58
Şekil 5.6. Girdi ve çıktı özniteliklerinin üyelik fonksiyonu 59
Şekil 5.7. Mamdani tabanlı çıkarım sisteminin 15 kuralı 60
Şekil 5.8. Uyarlamalı nöral-bulanık çıkarım sisteminin yapısı 61
Şekil 5.9. Uyarlamalı nöral-bulanık sistemin eğitim verisi 61
Şekil 5.10. Test verisinin çizimi V/S FIS Çıktısı 62

1. GİRİŞ

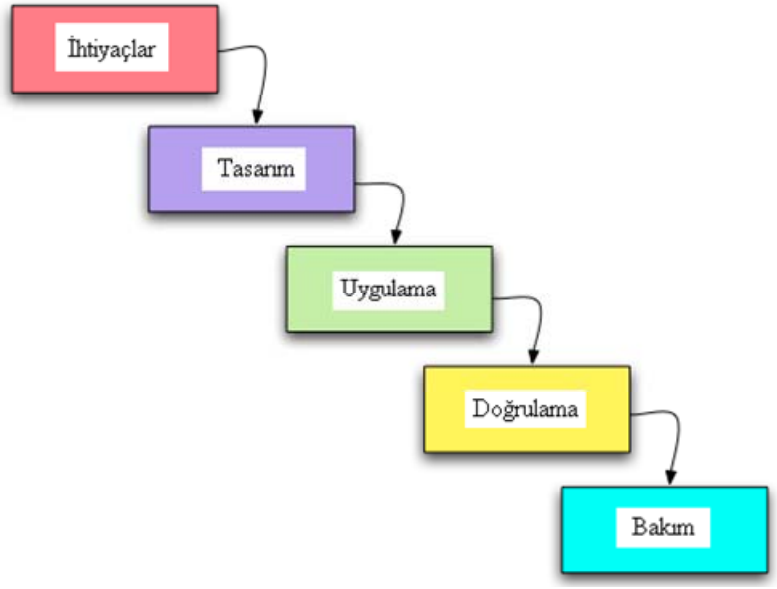
Yazılım sistemlerindeki hatalar temel problem olmaya devam ediyor. Yazılım bug-ı bilgisayar programının planlandığı gibi çalışmasını engelleyen (örneğin yanlış sonuçlar üreten) hata, aksaklık veya kusurdur. Yazılım hatası çalıştırılabilir üründe yazılım başarısızlığına sebep olan kusurdur. Yazılım mühendisliğinde, yazılımın ihtiyaçlarına uyumsuzluğu çoğunlukla bug olarak adlandırılır. Bugların çoğu insanlar tarafından programın kaynak kodunda veya tasarımında yapılan yanlışlar ve hatalar yüzünden, az bir kısmı da derleyicilerin yanlış kod üretmesinden oluşur. Olası kusurların nedenlerini bilmenin yanı sıra, projenin başlangıcından itibaren özene ihtiyaç duyan genel yazılım süreci alanlarını belirlemek, para, zaman ve iş kaybını önler. Yazılımdaki potansiyel bozukluğun erken kestirimi olasılığı, planlama, kontrol ve yazılım geliştirme aktivitelerini düzenlemede yardımcı olabilir.

Bir girdi (input) için çıktı (output) yanlış ise yazılım hata barındırıyor denir. Yazılım programının çıktısının yanlış olduğu her bir uygulama için başarısızlık (failure) gözlemleriz. Yazılım mühendisleri yazılım başarısızlıklarından yazılım hatalarını anlarlar. Başarısızlık durumunda yazılım kullanıcının umduğunu yapmaz. Öte yandan hata, başarısızlık olarak belirebilecek veya belirmeyecek gizli program kusurudur. Hata bilgisayar programının semantiğinin doğruluğundaki kusur olarak tanımlanabilir. Tam hesaplama koşulları sağlandığında hata, başarısızlık olarak sonuçlanır; koşullardan biri bilgisayar yazılımının hatalı kısmının CPU üzerinde çalıştırılmasıdır. Yazılım farklı bir donanım platformuna ya da farklı bir derleyiciye nakledildiğinde ya da geliştirildiğinde hata başarısızlığa dönüşebilir. Başarısızlığın sınıflandırılması Tablo 1.1.'de gösterilmiştir. Yazılım hataları tamamen yazılımı yaratan insan hatalarından kaynaklanmaktayken donanım hataları eskime, dış müdahaleler gibi rasgele olgulardan kaynaklanır.

Tablo 1.1. Başarısızlığın Sınıflandırılması

Başarısızlık Sınıfı	**Tanımlama**
Geçici (Transient)	Sadece belli girdilerle oluşur.
Daimi (Permanent)	Tüm girdilerle oluşur.
Telafisi mümkün (Recoverable)	Sistem kullanıcı müdahalesi olmadan düzeltebilir.
Telafisi mümkün olmayan (Unrecoverable)	Başarısızlıktan geri dönmek için kullanıcı müdahalesi gerekir.
Bozulmayan (Non-corrupting)	Başarısızlık sistem durumunu veya veriyi bozmaz.
Bozulan (Corrupting)	Başarısızlık sistem durumunu veya veriyi bozar.

Kalite kontrol ve testlere dayanarak hata azaltma gibi gelişme çabalarına rağmen birçok sistem kullanıcılara çok fazla hata ile teslim edilir. Uzun bir süredir anlaşıldı ki sistemin hata-eğilimli kısımlarını arayıp bulmak ve bu kısımları arttırılmış kalite kontrol ve test etme işlemleriyle hedef almak hata azaltmada etkili bir yoldur. Bu alanda günümüze kadar sınırlı sayıda çalışma yapıldı. Hata-eğilimli yazılım bileşenlerini tespit etmede güvenilir bir yaklaşım belirlemek zordur. Hata-eğilimli modüllerin kestirimi, geliştirilmiş planlama ve proje kontrolü aracılığıyla yazılım kalitesi mühendisliğini desteklemek için bir yol sağlar. Yazılımın kalitesi gittikçe önemli ve testlerle ilgili konular yazılım için kritik hale geliyor. Yazılım kalitesi tanımında çeşitlilik olmasına rağmen, birçok kusuru bulunan proje genişçe bir kesim tarafından düşük kaliteli olarak kabul görüyor. Test eforları, monitörleme işlemi maliyetleri ve ölçme sonuçlarının kestirimi için kullanılan metot ve teknikler yazılım testi verimliliğini arttırmaya yardımcı olabilir. Yazılımın hata-eğilimini (proneness) ölçebilme olasılığı yazılım testinin yönetimine ve tüm işlemin verimliliğini arttırmaya yönelik anahtar bir adım olabilir. Kestirimci modelleme modelin yaratıldığı veya modelin sonucun olasılığının en iyi kestirilmesi için seçildiği süreçtir. Hata-eğilimi modelinin amacı hata sınıflarını belirleme ve onların üzerine test çalışmaları yapmaktır.



Şekil 1.1. Yazılım geliştirme modelinin şelale modelinde gösterimi

İhtiyaçlar ve özellikler belirlendikten sonra, çözüm için plan tasarlanır.

Tasarım da, uyumluluk, genişletilebilirlik, hata toleransı, bakım kolaylığı, güvenilirlik, sağlamlık, güvenlik, kullanım kolaylığı göz önünde bulundurulur.

Uygulama program kodlarının yazıldığı aşamadır.

Doğrulama, sistemin kullanıcının ihtiyaçlarını doğru uygulayıp uygulamadığını, sistemin kullanılabilir olup olmadığının sorgulandığı, testlerin yapıldığı aşamadır.

Bakım aşaması programı güncel tutmak için gerekli görevleri içerir. Bakım kullanıcı tarafından istenebilecek genel iyileştirmeleri, eklemeleri ve diğer değişiklikleri de kapsar.

Yazılım üretimindeki hatalar yayılma özelliği gösterir. Bu nedenle, hata düzeltme maliyetleri ilerleyen aşamalarda giderek artar.

2. LİTERATÜR ARAŞTIRMASI

İstatistiksel metotlar, makine öğrenmesi ve birleşmiş teknikler yazılım hatalarının kestiriminde literatürde geniş çapta kullanılır. Birçok araştırmacı hata kestirimi alanında önemli çalışmalar yaptı. Literatür araştırması iki bölümde gerçekleştirildi: Birinci bölüm hata kestirimi ile ilgili literatür atıfları ile ilgiliyken, ikinci kısım makine öğrenmesi algoritmalarının araştırıldığı literatür atıfları ile ilgilidir.

2.1. Hata Kestirimi ile ilgili Literatür Atıfları

Lanubile ve diğerleri (1995) hata-eğilimli yazılım bileşenlerinin yazılım yaşam çevriminde erken belirlenmesi için modelleme tekniklerinin deneysel araştırmasını sundu. Teknikler, yazılım karmaşıklığı ölçülerini kullanarak, bileşenleri hata barındırma olasılığı olan veya olmayan şeklinde sınıflandıran modeller geliştirir. Çalışmalarında uygulanan modelleme teknikleri temel bileşen analizi, ayrık analiz, lojistik gerileme, mantıksal sınıflandırma modelleri ve katmanlı sinir ağlarını içeren ana sınıflandırma örneklerini kapsar. Lanubile hata barındıran bileşenler ile hata barındırmayan bileşenleri hiçbir modelin ayırt edemediğini öne sürdü.

Fenton ve diğerleri (1999) literatürün ciddi incelemesini sağladı ve konuya önemli katkılar yaptı (aksi halde deneysel çalışmalar eksik olurdu). Çalışmalarında, geniş kapsamlı kestirim modellerinin çoğu eksiklikleri kestirmek için büyüklük ve karmaşıklık ölçütü kullandılar. Diğerleri test verisine, gelişme sürecinin kalitesine dayanır veya çok değişkenli yaklaşımı ele alır. Eksiklikler ve başarısızlıklar arasındaki bilinmeyen ilişkinin üstesinden gelmenin modellerin yetersizliği yüzünden mümkün olmadığını söylediler.

Modelin geçerliliğini azaltan temel istatiksel ve veri kalitesi problemleri vardır. Daha önemlisi, birçok kestirim modeli temel problemin yalnızca parçasını modellemeye yönelir ve onu ciddi olarak yanlış belirler. Yazılım eksikliği kestirimi için tek konulu (single-issue) modellere alternatif yaklaşım olarak Bayesian İnanç Ağlarının (Bayesian

Belief Networks) kullanıldığı bütünsel modeller önerdiler. Eksiklik tanıtımı hakkında hipotezleri test etmek için yazılım ayrıştırma (decomposition) teorisi araştırması konusunda tartıştılar ve daha iyi yazılım mühendisliği bilimi düzenlenmesine yardım ettiler.

Saida Benlarbi ve diğerleri (1999) nesneye-yönelik ölçütlerin gelişiminin arkasındaki temel öncül görüş, onların hata barındıran veya bakımı maliyetli olan sınıfların erken kestirimcileri olarak çalışabilecekleriydi. Bugüne kadar ki ortak deneysel doğrulama metotlarının nesneye-yönelik ölçütün geçerliliğine göre yanıltıcı sonuçlar verebileceklerini gösterdiler. Nesneye-yönelik ölçütler kümesinin deneysel doğrulama çalışmasının sonuçlarını sundular. Makalelerinde büyüklüğün nesneye yönelik ölçütlerin doğruluğunda şaşırtıcı etkisi olduğunu gösterdiler.

Giovanni (2000) statik ölçütlerin makul (reasonable) kümesi ve yazılım hata eğilimi arasında korelâsyonun (ilgileşim) varlığını öne sürdü. Statik olarak kaynak kodunda hesaplanan statik ölçütler, örneğin, McCabe'nin cyclomatic numarası veya Halstead'in yazılım bilimi, yazılım karmaşıklığını nicelendirmeyi dener. Dinamik ölçütler, örneğin yapı ve veri akış kapsamı, testin mükemmelliğini test çalışmalarıyla elde edilen programın eleman miktarları ile ölçer. Bu ölçütler yazılım hata eğilimini etkileyen birçok beklentiyi sadece kısmen yansıtır ve bu yüzden test sürecini ayarlamak için limitli destek sağlar.

Runeson ve diğerleri (2001) yüksek kaliteli yazılım çabalarında, hata yönetiminin önemli rol oynadığını öne sürdü. Yazılım ürünlerinde bulunan hatalar yazılım modüllerinde eşit olarak dağıtılmazlar; bazı modüller diğerlerinden daha fazla hata eğilimlidirler. Onlar projelerini farklı modellerin yeterliliklerini karşılaştırmak ve uygun model seçmek için önerdiler.

Manasi Deodhar (2002) nesneye yönelik sistemlerde hata-eğilimi algılamasının ilginç bir araştırma alanı olduğu sonucuna vardı. Hata-eğilimi modelinin hedefi hata sınıflarını belirlemek ve onlar üzerlerinde test çabalarına odaklanmaktır. Her bir sınıf

için yapısal kalite hata eğilimi kestirimini sağlar. Sınıf büyüklük ölçüsü hata-eğilimi kestiriminde en önemli etmen bulundu.

Atchara Mahaweerawat (2004) yazılım dünyasının dinamik dünyasında rekabetçi kalmak için, organizasyonların kaliteli ürünleri zamanında ve maliyetinde teslim ederek limitli kaynakları en iyi şekilde kullanması gerektiğini analiz etti. Bu hata tanımının korunmasına, hızlı keşfine ve kalan hataların tamirine ihtiyaç duyar. Nesneye yönelik sistemlerde hataların önemli bölümünden sorumlu oldukları için özellikle, kalıtım ve polimorfizme (çok biçimlilik) dayalı hatalar ele alındı. Bu çalışmada önerilen hata kestirimi modeli Çok Katmanlı Perseptron (Multilayer Perceptron) sinir ağları kullanarak denetimli öğrenmeye dayanıyor. Sonuçlar sınıflandırma doğruluğu açısından analiz edildi ve sınıflandırma sonuçlarına göre hata barındıran sınıflar ayrıca incelenerek hatanın türüne göre sınıflandırıldı.

P. Bellini ve diğerleri (2005) hata eğilimi kestirim modellerini karşılaştırdı ve son yıllarda yazılım kalitesinin sistemlerin gelişmesinde en önemli ihtiyaçlardan biri haline geldiği sonucuna ulaştı. Hata eğilimi kestirimi yazılım ürünlerinin kalite kontrolünde anahtar rol oynayabilir. Ana amaçları, kestirim modelinin hata eğilimi kestirim oranı ve büyüklüğü arasında modelde kullanılan ölçütlerin sayısı açısından uyum bulmayı sağlamaktır. Çalışmalarında lojistik gerileme (regresyon) ve ayrık analiz metodolojilerini kullandılar.

Yan Ma ve diğerleri (2006) yazılım geliştirme sürecinde hata eğilimli modüllerin doğru kestiriminin eksikliklerin verimli keşfine ve belirlenmesine olanak sağladığını öne sürdü. Bu kestirim modelleri özellikle, doğrulama uzmanlarının ilgi ve kaynaklarını gelişmekte olan sistemlerdeki problem alanlarına odaklamaya ihtiyaç duyduğu, büyük ölçekli sistemler için değerlidir. Yan Ma makalesinde hata eğilimli modüllerin kestirimi için modifiye edilmiş rasgele orman algoritmalarını kullanan bir metodoloji sunuyor. Bu algoritma, sınıflandırma doğruluğunu ağaçlar topluluğu yetiştirerek ve onların sınıflandırma kararında seçim yapmalarına izin vererek geliştiriyor.

Sandhu ve diğerleri (2007), akıllı sistemlerin uygulanmasında yazılım bakımı aciliyet kestirimi için 35 makine öğrenmesi algoritması araştırdı ve ortalama mutlak hata (MAE), Kök Hata Kareler Ortalaması (RMSE) ve doğruluk yüzdesi esaslarına dayanarak lojistik model ağaçları (LMT) ve Complimentary Naïve Bayes (CNB) algoritmalarını önerdi.

2.2. Makine Öğrenmesi Algoritmaları ile ilgili Literatür Atıfları

S. Kanmani ve diğerleri (2003) yazılım kalitesi niteliği (örneğin hata oranı) kestirimi için Genel Gerileme Sinir Ağları (GRNN: General Regression Neural Network) uygulamalarını tartıştı. Çalışmaları bağımsız değişkenler olarak durgun nesneye-yönelik (OO) ölçütler (toplamda 64) ve bağımlı değişken olarak hata oranı kullanılarak gerçekleştirildi. Kalıtım, büyüklük, birleşme (cohesion) ve eşleşme (coupling) kavramlarını yazılım ölçütleri olarak kullanmışlardır. Kestirim modelleri ölçülerin 4 kategorisinin 15 olası kombinasyonu kullanılarak tasarlandı. Sinir ağları modelinin uygunluğunu standart parametrelerle test ettiler.

Quah ve Thwin (2003) yazılım kalitesi kestiriminde nesneye yönelik ölçütler kullanılarak gerçekleştirilen sinir ağları uygulamalarını sundu. Makalelerinde 2 çeşit araştırma gerçekleştirildi. İlki sınıflardaki eksikliklerin sayısının kestirimine ve ikincisi de her bir sınıf için değişen hatların sayısının kestirimine odaklanmışlardır. İki sinir ağları modeli kullanılmıştır: Ward Sinir ağları ve GRNN. Kalıtım ile ilgili ölçüler, karmaşıklık ölçüleri, birleşme (cohesion) ölçüleri, eşleşme ölçüleri ve bellek bölüştürme ölçüleri gibi nesneye yönelik tasarım ölçütleri bağımsız değişkenler olarak kullanıldı. GRNN ağ modelinin Ward ağ modeline göre daha kesin kestirimler yaptığı bulundu.

Challagulla ve diğerleri (2005) uzaktan kontrol (telecontrol)/uzaktan var olma (telepresence) sistemleri, robotik sistemler ve görev planlama sistemleri gibi gerçek zamanlı yazılım sistemlerinin büyük çoğunluğu göreve özel ihtiyaçlar ve işletim koşullarına dayanan dinamik kod birleşimi gerektirebilir. Bu, sistemlerin beklenen performansı gösterdiklerini ve başarısızlığa uğramadıklarını temin etmek için, dinamik

güvenilirlik değerlendirmesine olan ihtiyacını gösterir. Modülleri birleştirilmiş kodda, yazılım eksikliği kestirim tekniklerini kullanarak, dinamik değerlendirmek bunu başarmak için bir yaklaşımdır. Adımsal çoklu-doğrusal gerileme modeli ve çok değişkenli modeller gibi birçok istatiksel modeli ve yapay sinir ağları, örnek-tabanlı mantık, Bayesian inanç ağları, karar ağaçları ve kural indüklemeleri gibi makine öğrenme yaklaşımlarını yazılım kalitesi kestirimi için araştırdılar. Sonuç olarak, yazılım eksiklikleri için en iyi kestirim modeli hakkında ortak bir fikir birliğinin olmadığına karar verdiler. Makalelerinde, farklı kestirim modellerini, 4 birbirinden farklı gerçek-zamanlı yazılım eksikliği veri kümesinde değerlendirdiler. Sonuçları gösterdi ki, örnek-tabanlı öğrenme ile beraberindeki uyum-tabanlı altküme değerlendirme tekniği ve IR kombinasyonu doğruluk kestiriminde diğer modellere göre nispeten iyi tutarlılık sağlıyor.

Ceylan, Kutlubay ve Bener (2006) yazılım mühendisliğinin insanları, kısa teslim tarihlerini ve limitli bütçeleri kapsayan sıkıcı bir iş olduğunu öne sürdüler. Müşterinin isteğini teslim etmek programdaki eksikliklerin minimize edilmiş olmasını da kapsar. Bu nedenle kalite ölçülerini proje yaşam çevrimine erken yerleştirmek önemlidir. Araştırmalarının temel amacı, problemleri yazılım kodunda analiz etmek ve bu problemlerin proje yaşam çevriminde erken tespit edilmesine yardımcı olacak bir model sunmaktı. Önerdikleri model makine öğrenmesi metotlarını kullanıyordu. Temel bileşen analizi boyut azaltma için kullanıldı. Karar ağacı, Çok Katmanlı Perseptron (Multilayer Perceptron) ve radyal tabanlı fonksiyonlar eksiklik kestirimi için kullanıldı. Araştırmalarındaki deneyler Türkiye'deki 3 büyük yazılım şirketinin gerçek yaşam projelerinden elde edilen farklı yazılım ölçütü (metrics) veri kümeleri kullanılarak oluşturuldu.

3. ESNEK HESAPLAMA YÖNTEM VE TEKNİKLERİ

Esnek hesaplama geleneksel (conventional) hesaplamaya göre duyarsızlığa (imprecision), belirsizliğe (uncertainty), kısmi doğruya (partial truth) ve yaklaşıklığa (approximation) daha toleranslıdır. Esnek hesaplamanın rol modeli insan beynidir. Esnek hesaplamanın temel prensibi duyarsızlığa, belirsizliğe, kısmi doğruya ve yaklaşıklığa çözülebilirlik (tractability), dayanıklılık (robustness) ve düşük çözüm ücreti elde etmek için daha toleranslı olmasıdır. Esnek hesaplamanın şuan ki temel esasları birçok geçmiş etkilenim ile bağlantılıdır, bunlar arasında Zadeh'in bulanık kümeler (fuzzy sets), kompleks sistemlerin analizi & karar süreci, ve olasılık teorisi & esnek veri analizi ile ilgili makaleleri gösterilebilir. Esnek hesaplamaya nöral ve genetik hesaplama daha sonra dâhil edilmiştir.

Esnek hesaplama, Bulanık mantık (Fuzzy Logic), Nöral Hesaplama (Neural Computing), Evrimsel hesaplama (Evolutionary Computation), Makina Öğrenmesi (Machine Learning), Olasılıkçı Uslamlama (Probabilistic Reasoning) inanç ağları, kaos teorisi ve öğrenme teorisini kapsamaktadır. Burada, esnek hesaplamanın karışım olmadığı not edilmelidir. Aksine, her bir partner alanında bağımsız metodolojilerle katkıda bulunur. Bu perspektifte, esnek hesaplama birbirleri ile rakip değil, birbirlerini tamamlayan metodolojilerden oluşur. Esnek hesaplama kavramsal zekânın gelişen alanı için temel bileşen olarak görülebilir.

Çalışmamızda yukarıda bahsedilen yöntemlerden *Makine Öğrenmesi Algoritmaları, Mamdani Tabanlı Bulanık Çıkarım Sistemi* ve *Nöral-Bulanık Tabanlı Sistemler* kullanıldı.

3.1. Makine Öğrenmesi Teknikleri

Yapay zekânın geniş bir alt tabanı olarak makine öğrenmesi, bilgisayarların "öğrenmesine" izin verecek algoritma ve tekniklerin tasarımı ve gelişimi ile ilgilidir. Öğrenme, sistemde, sisteme aynı işi bir dahaki seferinde daha verimli yapmasına olanak veren değişiklikler yapmak anlamına gelir. Genel seviyede iki çeşit öğrenme vardır:

tümevarımsal (inductive) ve tümdengelimli (deductive). Tümevarımsal makine öğrenme metotları, kuralları ve örüntüleri (pattern) çok büyük veri kümelerinden dışarı çıkartırlar. Makine öğrenmesi araştırmasının başlıca odak noktası veriden bilgiyi hesaplanabilir ve istatiksel metotlarla otomatik olarak çıkarmaktır. Bundan dolayı, makine öğrenmesi veri madenciliği ve istatistikle yakından ilişkilidir.

Makine öğrenmesi makine zekâsının bir parçasıdır fakat daha uzmanlaştırılmış amaç ve kapsamı ele alır. Makine öğrenmesi algoritma ve uygulamaları, kendilerini sistemin davranışına genellikle verideki zaman-değişkenli örüntülerin keşfedilmesi yoluyla adapte ederler. Bu algoritmalar genellikle doğrusal ve doğrusal-olmayan gerilemeyi (regression), uyarlamalı (adaptive) kontrol teorisini, sinir ağlarını, istatiksel öğrenme teorisini, kural tüme varım ve karar ağacı üretimini kaynaştırır. Öğrenme ve zekâ arasındaki çok yakın ilişki yüzünden, neredeyse tüm makine zekâsı sistemleri öğrenmenin bazı biçimlerini kapsar. Öğrenme sistemde, sistemin aynı görevi bir dahaki sefere daha verimli yapmasına olanak verecek değişikliğe işaret eder. Öğrenme iki yolla olur:

- Denetimli (Supervised)
- Denetimsiz (Unsupervised)

Denetimli öğrenme, objektif fonksiyona sahiptir (bağımlı değişken) ve bağımsız değişkenler kümesini bağımlı değişkenler sınıfına ayıran kuralları öğrenmek için tarihi veri (eğitim verisi olarak adlandırılan) kullanır.

Denetimsiz öğrenme veri yığınları arasındaki dolaylı ilişkileri keşfeder ve diğer değişkenler üzerindeki en büyük nedensel etkiye sahip değişkenlerdeki davranış değişikliğini tanımlayan kuralları geliştirir. En çok kullanılan ve bizim de kullanacağımız yol denetimli öğrenmedir.

Öğrenme algoritmaları aşağıdaki gibi olabilir:

- Durgun: Durgun öğrenme algoritması örüntüleri keşfeder ve kesin kestirimler yapabilir. Durgun öğrenme algoritmalarında tanınmış örüntüler bununla birlikte

zamanla değişmemiş olarak kalır (örüntüler değişmemiş kaldığı sürece durgun öğrenme algoritmaları olarak adlandırılır).

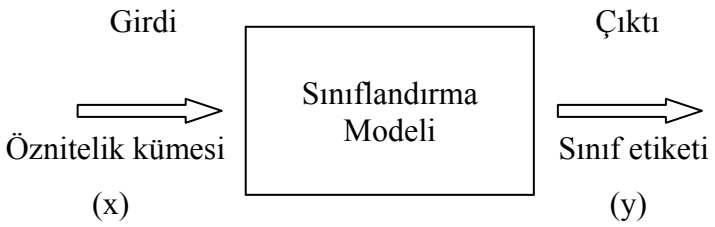
- İyileştirici (Regenerative): İyileştirici algoritma "geçmişini" unutur ve yeni veri kümesinden (sıklıkla üst üste biner) örüntüler kümesini tekrar öğrenir.
- Dinamik: Dinamik ve uyarlamalı algoritma sürekli eğitim mekanizması ve yanı sıra modelleme hata oranını ölçen ve gittikçe daya iyi bir model için dâhili kontrollerini gittikçe daha yakına ayarlayan geri besleme mekanizmasını kapsar.

Makine öğrenmesi yetenekleri dayanıklı, özuyarlamalı (selfadapting), bakımı kolay ve sıklıkla konvansiyonel (geleneksel) sistemlerden daha fazla hata toleranslı uygulamalar yaratır. Uyarlamalı geri besleme döngüsü sistemi işletme politikaları dâhilinde değişikliklere adapte edebilir ve onu daha esnek yapabilir. Öğrenme sistemleri ayrıca tecrübe kazandıkça yeteneklerine ince ayar yapan güçlü kestirimci ve sınıflandırma modelleri için çekirdek mekanizma sağlar. Makine öğrenmesi, bir görevde performanslarını tecrübeyle arttıran programları nasıl kurabileceği konusu ile ilgilenir.

Makine öğrenmesi algoritmalarının, çeşitli uygulama alanlarında büyük pratik değere sahip olduğu kanıtlanmıştır. Şaşırtıcı olmayan biçimde, yazılım mühendisliği alanı birçok yazılım geliştirme ve bakım görevinin öğrenme problemleri olarak formüle edilebildiği ve öğrenme algoritmaları olarak ele alındığı verimli bir sahaya dönüyor.

Makine öğrenmesi algoritmaları ile yapacağımız çalışma için sınıflandırma, eğitim kümesi ve test kümesi kavramlarını inceleyelim:

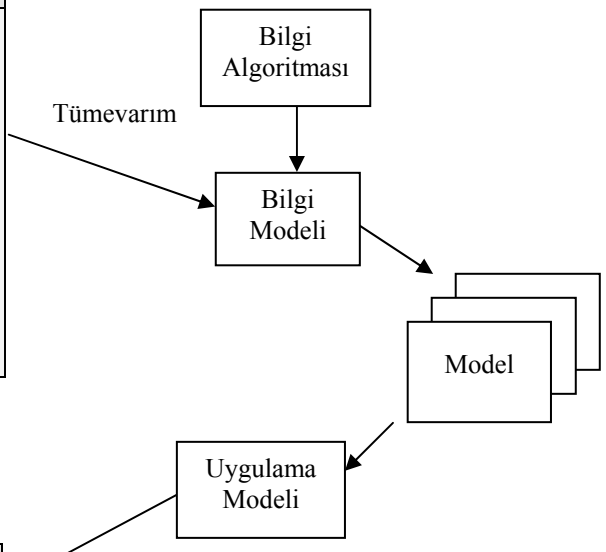
Sınıflandırma, nesneleri önceden tanımlanmış bilgilere ve bulunduğu duruma göre ilgili kategoriye atama anlamına gelir. Sınıflandırma; her bir öznitelik kümesi *x*'i, önceden tanımlanmış sınıf etiketlerinden biri olan *y* ile eşleştiren bir **hedef fonksiyon** olan *f*'yi öğrenme işidir.



Şekil 3.1. Sınıflandırma

Eğitim Kümesi

No	Öznitelik1	Öznitelik2	Öznitelik3	Sınıf
1	Evet	Büyük	125K	Hayır
2	Hayır	Orta	100K	Hayır
3	Hayır	Küçük	70K	Hayır
4	Evet	Orta	120K	Hayır
5	Hayır	Büyük	95K	Evet
6	Hayır	Orta	60K	Hayır
7	Evet	Büyük	220K	Hayır
8	Hayır	Küçük	85K	Evet
9	Hayır	Orta	75K	Hayır
10	Hayır	Küçük	90K	Evet



Test Kümesi

No	Öznitelik1	Öznitelik2	Öznitelik3	Sınıf
11	Hayır	Küçük	55K	?
12	Evet	Orta	80K	?
13	Evet	Büyük	110K	?
14	Hayır	Küçük	95K	?
15	Hayır	Büyük	67K	?

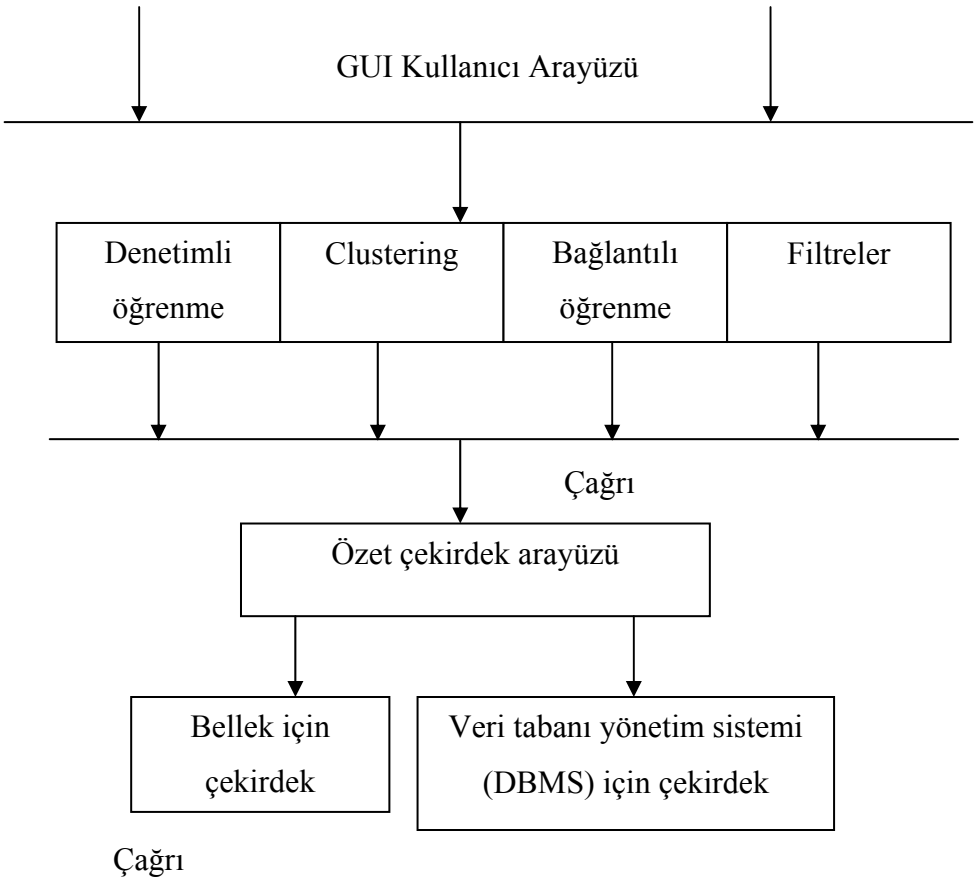
Şekil 3.2. Bir sınıflandırma modeli kurulumu için genel yaklaşım

Bir **sınıflandırma tekniği** (veya sınıflandırıcı), girilen veri kümesinden sınıflandırma modelleri geliştiren sistematik bir yaklaşımdır. Her teknik, giriş verisinin öznitelik kümesi ve sınıf etiketi arasındaki ilişkiyi en iyi yansıtan modeli tanımlamak için bir **öğrenme algoritması** kullanır.

Şekil 3.2. sınıflandırma problemi için kullanılan genel bir yaklaşımı göstermektedir. Öncelikle, sınıf etiketleri bilinen kayıtları içeren bir **eğitim kümesi**

sağlanmalıdır. Eğitim kümesi, bilinmeyen sınıf etiketli kayıtları içeren **test kümesi**ne sonradan uygulanacak olan bir sınıflandırma modelinin kurulumunda kullanılır (Aktaş).

WEKA (Waikato Bilgi Analiz Platformu) Waikato Üniversitesi'nde geliştirilmiş olan, Java'da yazılmış makine öğrenmesi yazılımı için popüler bir platformdur. WEKA GNU Genel Kamu Lisansı (General Public License) altında ücretsiz bir yazılımdır. WEKA makine öğrenmesi ve veri madenciliği için kapsamlıdır. WEKA'nın temel gücü bütün güncel makine öğrenmesi yaklaşımlarının sınıflandırma alanında yatar. Epeyce eski olanları temiz, nesne yönelimli Java sınıfı hiyerarşisin içinde uygulanmıştır. Gerileme, ilişki (Association) kuralları ve clustering algoritmaları da ayrıca uygulanmıştır. WEKA'da makine öğrenmesi algoritmaları sınıflandırıcı (Classifiers) olarak adlandırılır ve Şekil 3.3.'de gösterilmiştir.



Şekil 3.3. WEKA mimarisi

WEKA'daki sınıflandırıcılar, kestirim için hedef olan, tek sınıf özelliği kestirimi için eğitilmişlerdir. Bazı sınıflandırıcılar sadece nominal sınıfları öğrenebilirler, diğerleri ihmal ile sadece sayısal sınıfları öğrenebilirler ve sınıf verideki son öznitelik olarak alınmıştır (WEKA, 2007). Makine öğrenmesi algoritmalarının sınıflandırılmasıyla ilişkili literatür araştırması aşağıdaki gibi incelenmiştir:

Makine öğrenmesi algoritmalarının 6 temel sınıfı bulunur:

- BAYES
- FUNCTIONS
- LAZY
- META
- MISC
- TREES

Hazırladığımız kullanıcı arayüzü ile WEKA platformundaki bu 6 sınıfa ait aşağıdaki algortimaları kullandık:

3.1.1. Makine öğrenmesi algoritmalarının Bayes sınıfı

Bayes sınıfı algoritmalarını inceleyelim:

- **AODE:** AODE (Averaged One-Dependence Estimator-Ortalaması alınmış Tek-Bağımlı Kestirimci) Naïve-Bayes'e göre daha zayıf (ve bu yüzden daha az zararlı) bağımsız varsayımları olan Naïve-Bayes benzeri (like) alternatif modellerin tüm küçük alanlarının ortalamasını alarak yüksek kesinlikte sınıflandırma başarır. Sonuç algoritması birçok öğrenme görevinde yüksek kesinlikte sınıflandırmayı sağlarken hesaplama yönünden etkilidir (Webb and Boughton, 2002).

- **Bayes Net:** Bayes ağ öğrenmesi çeşitli araştırma algoritmalarını ve kalite ölçülerinin kullanımını kapsar. Bayesian ağ sıklıkla çok terimli (multinominal) verinin ayrık ve devamlı değişkenle modellenmesi için kullanılır. Bu algoritma, çözümün olasılık dağılımı kestirimi önermesini kullanır. Çok değişkenli veriyi modellemek için kullanılan dağılım teknikleri kestirimine yeni aday oluşturmak amacıyla Bayesian ağları

kullanılır. Bu algoritma daha zor problem sınıflarını daha verimli ve güvenilir çözmek amacıyla var olan metodu geliştirir. Bayesian ağları modellenmiş verideki değişkenler arasındaki ilişkiyi şifreler. Bayesian ağları problemin yapısını sunar (Pelikan, Goldberg and Paz, 1999). Matematiksel olarak, yönlendirilmiş kenarları olan çevrimsiz Bayesian ağı birleşik olasılık dağılımını şifreler:

$$p(X) = \prod_{i=0}^{n-1} p(X_i \mid \prod_{X_i}) \tag{3.1}$$

X = (X0… X_n - 1) değişkenlerin vektörüdür, $\prod_{Xi}$ ağdaki X_i'lerin atalarının (parents) kümesidir. $p(X_i / \prod_{X_i})$ X_i'nin şartlı olasılığıdır. Bu dağılım, modeldeki marjinal ve şartlı olasılıklardan faydalanarak yeni örnekler oluşturmada kullanılabilir. Bayesian Ağının avantajları aşağıdaki gibidir:

i. Yüksek düzeydeki etkileşimi koruma yeteneği.
ii. Umut vadeden çözümün ortak-dağılımının kestirimi amacıyla veri modellemeyi kullanır.
iii. Problem hakkında herhangi bir ön bilgi gerektirmez.
iv. Yapı bloklarını belirlenmiş düzene göre verimli bir şekilde yeniden oluşturur ve kaynaştırır.

Bayesian Ağının dezavantajları aşağıdaki gibidir:

i. Bayesian ağı karmaşık olasılık dağılımı gösterebilir.
ii. Verimlilik sonucu genişletmez.
iii. Çoğu zaman maliyetlidirler.
iv. Bu metodun uygulamaları zorluklarla mücadele ediyor.
v. Sonuç zamanı önceden kestirilemez.
vi. Yakınsama hakkında bilgi vermek zordur. Bazı durumlarda oluşmaz ve bazen yanlış çıktı verir.
vii. Bayesian ağı uygulamalarını limitler.

- **Complement Naive Bayes:** Tümleyen sınıf, Naive Bayes sınıflandırıcı yapılandıran ve kullanan sınıfla ilgilidir. Naive Bayes hızlı ve uygulanması kolay olduğu için metin (text) sınıflandırmada dayanak olarak sıklıkla kullanılır. Bize yalnızca daha iyi verimlilik vermez, aynı zamanda sonuç kalitesini aleyhte de etkileyebilir. Naive Bayes sınıflandırıcılar sistematik konularla ve tam olarak çokterimli modele göre üretilmediği için oluşan problemlerle de ilgilenir. Naive Bayes'in zayıf performansının sebebi, her problem için basit buluşsal çözümlerin olmasıdır (Rennie ve Shih, 2003).

Bir sınıfın diğer bir sınıftan daha fazla eğitim örnekleri varsa, Naive Bayes karar sınırı için düşük ağırlıkları seçer. Bu eğitim örneği az sınıflar için ağırlıkları küçülten dublör bias etkisinden dolayı gerçekleşir. Rennie ve diğerleri (2003) her kestirimde kullanılan eğitim örneklerinin miktarını dengelemek için, Naïve Bayes'in "Tümleyen sınıf" formülünü tanıttılar.

Naive Bayes ile ilgili diğer bir sistematik problem ise özniteliklerin münferit olarak varsayılmalarıdır. Sonuç olarak, kelimeler bağımlı dahi olsalar, her bir kelime başlı başına bulgu sağlar. Bu yüzden güçlü kelime bağımlılıklarına sahip sınıflar için ağırlığın önemi düşük kelime bağımlılığı olan sınıflardan büyüktür.

- **Naive Bayes:** Kestirim sınıflarını kullanan Naive Bayes sınıflandırıcısı için bir sınıftır. Sayısal kestirim kesinliği verileri eğitim verilerinin analizine dayanarak seçilirler. Bu yüzden sınıflandırıcı Güncellenebilir (Updateable) Sınıflandırıcı (tipik kullanımında sıfır eğitim örnekleriyle ilklendirilen) değildir. Güncellenebilir sınıflandırıcı özelliğine ihtiyaç duyulduğunda Naive Bayes Güncellenebilir Sınıflandırıcısı kullanılmalıdır. Naive Bayes güncellenebilir sınıflandırıcısı "yapı (build) sınıflandırıcısı" sıfır eğitim örnekleri tarafından çağırılınca sayısal öznitelikler için varsayılan duyarlılığı 0,1 olarak kullanacaktır (John ve Langley, 1995).

- **Naive Bayes Multinomial:** Bu sınıf çok terimli Naive Bayes sınıflandırıcısını yapmak ve kullanmak içindir. Hızlı, kolay uygulanabilir ve oldukça verimlidir. Multinomial Naive Bayes belgedeki kelimelerin dağılımını çokterimli olarak modeller. Belge kelime dizisi olarak ele alınır ve her kelimenin pozisyonunun diğerinden

bağımsız olarak oluşturulduğu varsayılır. Sadeleştirmeden sonra McCallum ve Nigam (1998) metin sınıflandırmak için çok terimli Naive Bayes modelini öne sürdü ve çok değişkenli Bernoulli modeline göre frekans bilgisi katkısından ötürü geliştirilmiş bir performans gösterdi. Naïve Bayes'in çok terimli versiyonu olan Çokterimli Naive Bayes (MNB) Rennie ve Shih (2003) tarafından tartışıldı, analiz edildi ve geliştirildi.

- **Naive Bayes Simple:** Mooney'e göre (1996), sayısal öznitelikler normal dağılımla modellenir. Ted Pedersen birkaç çok basit sınıflandırıcıyı çoğunluk oyu aracılığıyla belirsizliği gideren topluluğa birleştirerek yüksek doğrulukla sonuçlanan derlem (corpus) tabanlı bir yaklaşım sundu. Çoklu sınıflandırıcılar topluluğunun (ensemble) birleştirilmiş (combined) doğruluğu, topluluğu meydana getiren herhangi münferit (individual) sınıflandırıcıdan, çoğu kez, önemli ölçüde büyüktür. Bu çoklu sınıflandırma topluluğunun birleştirilmiş doğruluğunu gösterir. Naive Bayesian sınıflandırıcısı problemini temsil eden nitelik değişkenleri şartlı bağımsızdır, sınıflandırma değişkeni değeri verilmiştir. Çok anlamlı kelimenin oluştuğu şartlar nitelik değişkenleri (F_1, F_2 ... F_n) tarafından temsil edilir ve çok anlamlı kelime sınıflandırma değişkeni (S) tarafından temsil edilir. Tüm niteliklerin değişkenleri, F_i, ikilidir (binary) ve çok anlamlı kelimenin soluna veya sağına doğru olan kelimeler dâhilinde özel kelimenin oluşup oluşmadığını gösterirler (Mooney, 1996). Naive Bayesian sınıflandırıcıları için parametreler p(S) ve $p(F_i/S)$'dir. Özel anlamlı içeriksel (contextual) niteliklerin belirlenmiş kombinasyonunu gözlemlemenin ortak olasılığı

$$p(F_1, F_2, ...F_n, S) = p(s)\prod_{i=1}^{n} p(F_i \mid S) \qquad (3.2)$$

ile ifade edilir.

- **Naive Bayes Updateable:** Bayesian ağları için sürekli değişkenleri nasıl ele alacağı problemi vardır. Daha önceki çalışmaların çoğu problemi ihmal ederek veya verilerin tek (single) Gaussian ile oluşturulduğunu varsayarak çözdüler. Kernel kestiriminin Bayesian modellerini öğrenmede kullanışlı bir tool olduğunu öneren birçok doğal ve yapay veri kümelerindeki hatalarda büyük azalmalarla gözlendi.

Olasılıklı indüklemeye (probabilistic induction) veriler çok daha basit ve eski bir yaklaşım olan Naive Bayes sınıflandırıcıdan gelir. Güncellenebilir Naive Bayes, Naive bayes sınıflandırıcı için kestirim sınıfları kullanılarak oluşturulmuş bir sınıftır. Bu Naive Bayes'in güncellenebilir versiyonudur. Naive Bayes güncellenebilir sınıflandırıcısı "yapı (build) sınıflandırıcısı" sıfır eğitim örnekleri tarafından çağırılınca sayısal öznitelikler için varsayılan duyarlılığı 0,1 olarak kullanacaktır (John ve Langley, 1995).

3.1.2. Makine öğrenmesi algoritmalarının Functions sınıfı

Functions sınıfında sunulan çeşitli algoritmalar aşağıda tartışılacaktır.

- **Logistic:** Sınıf, WEKA (2007)'da sunulan ridge kestiricimsiyle çokterimli lojistik regresyon modelini kullanma ve yapılandırma da kullanıldı.

- **Multilayer Perceptron:** WEKA (2007) örnekleri sınıflandırmak için geri yayılım (Backpropagation) kullanan sınıflandırıcı içerir. Bu ağ elle yapılandırılabilir, algoritma ile yaratılabilir veya her ikisi de olabilir. Ağ ayrıca görüntülenebilir ve eğitim zamanı süresince modifiye edilebilir. Ağdaki düğümler sigmoidtir (bir şeyin haricinde; sınıf sayısal olduğu zaman çıkış düğümleri eşiksiz lineer birimler olur).

- **Simple Logistic:** Landwehr, Hall ve Frank (2005) linear logistic regresyon modeli için bir sınıflandırıcı araştırdı. Temel öğrenenler olarak basit regresyon fonksiyonları ile birlikte LogitBoost, lojistik modelleri ayarlamak (fitting) için kullanıldı. LogitBoost yinelemelerinin optimal sayısı çapraz-sağlandı (cross-validated). Çapraz sağlama otomatik öznitelik seçimine liderlik eder (Sumner, Frank ve Hall, 2005).

- **SMO:** Çok terimli (polynomial) kernel kullanarak destek vektör sınıflandırıcıyı eğitmek için John C. Platt'in (1998) Ardışık Minimal Optimizasyon Algoritmasını (Sequential Minimal Optimization Algorithm) uygular. Bu uygulama global olarak bütün kayıp değerleri yenisiyle değiştirir ve nominal öznitelikleri ikili olanlara dönüştürür. Ayrıca bütün öznitelikleri (attributes) önceden tanımlanmış değerlerle (default) normalize eder. Keerti, Shevade, Bhattacharya ve Murthy (2001 Platt'in algoritmasına iyileştirmeler tanıttılar. Çoklu-sınıf (Multi-Class) problemleri çiftli

sınıflandırma kullanarak çözüldü (Hastie ve Tibshirani, 1998'ye göre 1'e karşı 1 ve lojistik modeller çiftli eşleşme yapılırsa). Uygun olasılık kestirimleri elde etmek için, lojistik regresyon modellerinin destek *Vektör* Makinesinin (*Support Vector Machine* (SVM) çıktılarına uymasını sağlayan opsiyon kullanılmalıdır. Çoklu sınıf durumunda, kestirilen olasılıklar Hastie ve Tibshirani'nin (1998) çiftli eşleşme metodu kullanılarak eşleştirildi.

3.1.3. Makine öğrenmesi algoritmalarının Lazy sınıfı

Lazy sınıfına ait çeşitli algoritmaları inceleyelim:

- **IB1:** En yakın komşu sınıflandırıcısı kullanır. Verilen test örneğine en yakın eğitim örneğini bulmak için standartlaştırılmış öklit mesafesi kullanır ve bu eğitim örneğiyle aynı sınıfın kestirimini yapar. Eğer birden çok örnek test örneğine aynı (en küçük) mesafeye sahipse, ilk bulunan kullanılır (Aha ve Kibler, 1991).

- **IBk:** En yakın K-Komşu (K-Nearest Neighbors) algoritmasıdır. Bu algoritma sınıflandırma için kullanılır. K tabanlı komşuların uygun değerini çağraz doğrulama ile seçebilir. Ayrıca mesafe ağırlıklandırabilir (Aha ve Kibler, 1991).

- **KStar:** Cleary ve Trigg (1995) *K**'ın Örnek tabanlı (IB) sınıflandırıcı olduğunu göstermiştir. Bazı benzerlik fonksiyonlarıyla belirlendiği gibi, eğitim örnekleriyle aynı olan sınıfa istinaden, test örneğinin sınıfıdır. Diğer örnek tabanlı öğrenenlerden entropi tabanlı mesafe fonksiyonu kullanması yönüyle farklıdır.

- **LWL:** Atkenson, Moore ve Schaal (1996) Lokal olarak ağırlıklandırılmış öğrenmenin (Locally Weighted Learning) gerekliliklerini belirttiler; yani mesafe fonksiyonu, ayrılabilir kriter, yeterli veri, etiketli veri ve sunumları.

Frank, Hall ve Pfahringer (2003) Naive Bayes'in lokal modelleri, kestirim zamanında öğrenerek bağımsızlık varsayımını rahatlatan lokal olarak ağırlıklandırılmış versiyonunu sundular. Deney sonuçları, lokal olarak ağırlıklandırılmış Naive Bayes'in standart Naive Bayes ile kıyaslandığında doğruluğunun nadiren düşük olduğunu ve

birçok durumda doğruluğunu önemli derece arttırdığını gösterdi. Bu metodun Naive Bayes algoritmasını geliştirmeye çalışan diğer tekniklere göre temel avantajı kavramsal olması ve basit hesaplanabilmesidir. Bu lokal olarak ağırlıklandırılmış öğrenmenin diğer algoritmalara göre tercih edilmesinin temel nedendir.

3.1.4. Makine öğrenmesi algoritmalarının Meta sınıfı

Meta sınıfına ait çeşitli algoritmaları inceleyelim:

- **ADABoostM1:** Freund ve Schapire (1996) 2 aşamada deneysel araştırmalar yaptılar. İlk araştırma da Boosting ve Bagging metotları çeşitli sınıflandırıcılar (karar ağaçları ve tek öznitelikli-değer testleri dâhil olmak üzere) ile karşılaştırıldı ve makine öğrenmesi kriterleri ile 2 metodun performansını değerlendirildi. İkinci çalışmada Boosting'in optik karakter tanıma (OCR: Optical Character Recognition) problemi üzerinde en yakın komşu sınıflandırıcısı kullanarak performansı değerlendirildi. Sadece nominal sınıf problemleri ele alınabildi. Performansın çoğu kez iyi olduğu fakat bazen aşırı uymanın (overfit) olduğu gözlendi.

- **Attribute Selected Classifier:** Öznitelik seçilmiş sınıflandırıcıda eğitimin ve test verisinin boyutları sınıflandırıcıya aktarılmadan önce öznitelik seçimi ile düşürülmüştür **(**WEKA, 2007).

- **Bagging:** Breiman (1996)'a göre, Bagging kestirimleyicinin birden çok versiyonunu üretmek için bir metottur ve bunları kümelenmiş kestirimleyici elde etmek için kullanır. Kümeleme sayısal çıktıyı kestirirken bu versiyonların ortalamasını alır ve sınıfı kestirirken oy çokluğu (plurality vote) prensibini uygular. Öğretim kümesininin karışması yapılandırılan kestirimleyicide önemli değişikliklere yol açıyorsa, bagging doğruluğu arttırabilir.

- **Classification via Regression:** Model ağaç, yaprakları gibi, terminal sınıf değerleri yerine lineer regresyon fonksiyonlarını kullanarak, karar ağacının formunu alır. Sayısal değere sahip öznitelikler regresyon fonksiyonlarında doğal bir rol oynarlar bununla beraber ayrık değerliler de daha az doğal bir yolla ele alınabilirler. Frank, Wang, Inglis,

Holmes ve Witten (1998)'a göre model ağaçları, yapraklarında lineer regresyon fonksiyonlarıyla karar ağaçlarının bir çeşididir. Model ağaçları daimi sayısal değerleri kestirmek için son başarılı tekniğin ilkelerini biçimlendirir. Sınıflandırma problemini standart dönüştürme metodu ile fonksiyon yaklaştırma (approximation) problemine dönüştürmek suretiyle, sınıflandırma problemlerine uygulanabilirler.

- **Filtered Classifier:** Java sınıfına dayanan algoritmalar, isteğe bağlı filtreden (arbitrary filter) geçmiş veri üzerinde, isteğe bağlı sınıflandırıcıyı çalıştırmak için, sınıflar barındırır. Sınıflandırıcı gibi, filtrenin yapısı yalnızca eğitim verisine dayanır ve test örnekleri filtre tarafından yapıları değiştirilmeden işlenir (WEKA, 2007).

- **Grading:** Bu metotta temel sınıflar derecelendirilir (WEKA, 2007).

- **LogitBoost:** Boosting sınıflandırma metodolojileri arasında en önemli güncel gelişmelerden biridir. Boosting, eğitim verisinin yeniden ağırlıklandırılmış versiyonuna sınıflandırma algoritmasını sırayla uygulayarak ve sonra bu yolla üretilen sınıflandırma dizisinin ağırlıklandırılmış çoğunluk oyunu alarak çalışır (Friedman, Hastie ve Tibshirani (1998). Birçok sınıflandırma algoritması için, basit strateji, performansta ciddi bir artışla sonuçlanıyor. Bu, görünüşte gizemli fenomenin, bilinen istatiksel prensiplerle, yani toplamsal (additive) modelleme ve maksimum olasılık (likelihood), anlaşılabileceğini gösterir. 2-sınıf problemi için, Boosting, lojistik ölçüde Bernoulli olasılığını (likelihood) kriter alarak, toplamsal modele yaklaşım olarak görülebilir. Toplamsal lojistik regresyonun uygulaması için bir sınıftır. Bu sınıf temel öğrenen (base learner) gibi sınıflandırmayı regresyon şeması kullanarak gerçekleştiriyor ve çok sınıflı problemleri ele alabiliyor.

- **MultiBoostAB:** Webb (2000) sınıflandırıcıyı desteklemek için MultiBoosting metodunu kullanarak sınıf sundu. MultiBoosting karar komitelerini biçimlendiren oldukça başarılı AdaBoost tekniğine bir ilavedir. MultiBoosting, AdaBoost'un Wagging ile birleşimi olarak görülebilir. AdaBoost'un yüksek bias **(**Bir metodun sistematik hatası) ve varyans azaltma özelliklerinden Wagging'in üstün varyans azaltma özelliği ile faydalanabilir. Temel öğrenme algoritması olarak C4.5'i kullanarak,

AdaBoost veya Wagging'ten daha düşük hata ile karar komiteleri (decision committee) üretmek için Multi-boosting sunuldu. Paralel çalıştırmayı uygunlaştıran AdaBoost'a göre daha fazla avantaj sağlar. MultiBoosting AdaBoost'u Wagging ile birleştiren karar komitesi tekniğidir. Wagging, Bagging'in farklı bir biçimidir ve göreve direkt Bagging'ten daha uygundur. AdaBoost ve Bagging herhangi bir temel sınıflandırma tekniğiyle çalışabilecek genel tekniklerdir. (Webb, 2000).

- Multi Boosting AdaBoost veya Bagging'in herhangi birine göre ortalama hatayı azaltmada daha fazla başarılıdır.
- Multi Boosting Ada Boosted alt komiteleri (sub-committees) olan Bagging'i kullanır; en azından küçük ölçekli komitelerde.

- **MultiClassClassifier:** WEKA (2007) çok-sınıflı veri kümelerini 2 sınıflı (2 class) sınıflandırıcılar ile ele alabilmek için üst düzey sınıflandırıcı (metaclassifier) sunuyor. Bu sınıflandırıcı ayrıca, arttırılmış doğruluk için, hata düzeltme çıkış kodlarının uygulanmasına izin verir.

- **MultiScheme:** WEKA (2007) eğitim verisi üzerinde çapraz doğrulama kullanan çeşitli sınıflandırıcılardan veya eğitim verisi üzerindeki performanstan sınıflandırıcı seçmek için bir sınıfa sahiptir.

- **Ordinal Class Classifier:** Frank ve Hall (2001) sınıf değerlerinin çoğunlukla düzensiz olduğunu farz ederek sınıflandırma problemleri için Makine Öğrenmesi metotları tanımladı. Bununla beraber, birçok pratik uygulamada sınıf değerleri doğal sıralama E_g gösterir. Örneğin derecelendirmeyi öğrenmek için. Sırasal (ordinal) sınıflandırmaya standart yaklaşım, sınıf değerini sayısal niceliğe dönüştürür ve dönüştürülen veriye regresyon öğrenici uygular, son işleme (post-processing) adımında çıktıyı ayrık sınıf değerlerine çevirir. Bu metodun dezavantajı yalnızca regresyon planı (scheme) ile birlikte uygulanmasıdır. Ordinal Sınıf Sınıflandırıcısını karar ağacı öğrenicisi ile birlikte uygulayarak sınıf değerlerine düzensiz küme muamelesi yapan Naïve yaklaşımını daha iyi yapar.

- **Random Sub Space:** Ho'ya (1998) göre, bu metot, eğitim verisinde yüksek doğruluğu sürdüren ve karmaşıklıkta geliştikçe doğruluğun genelleşmesini geliştiren, karar ağacı tabanlı sınıflandırıcı yapılandırır. Sınıflandırıcı sistematik olarak yapılandırılan sistematik ağaçlardan oluşur. Sınıflandırıcı, öznitelik (feature) vektörünün bileşenlerinin altkümelerini sözde-rastlantısal (pseudo-randomly) seçmesiyle oluşturulan çoklu ağaçlar içerir, diğer bir deyişle, rasgele seçilen alt uzaylarda yapılandırılan ağaçlar içerir.

- **Stacking:** Bu metot muhtelif sınıflandırıcıları birleştirir (WEKA, 2007).

- **StackingC:** Stacking versiyonundan daha etkilidir (WEKA, 2007).

- **Vote:** Kuncheva (2004) sınıflandırıcıları birleştirmek için bir sınıf tanıttı. Bu sınıflandırıcı da, sınıflandırma için olasılık kestirimlerinin farklı kombinasyonları mevcuttur. Öğrenme algoritmalarının ve parametrelerin ayarları çeşitliliği yüksek kaliteli model üretmede etkin görünüyorlar. Topluluk (ensemble) seçiminin güçlü performansı, birçok farklı öğrenme metotları ve parametre ayarlarını kullanmayı modellerin çeşitli koleksiyonlarını üretmenin etkili bir yolu olarak tavsiye eder. Topluluk, modellerin, kestirimleri ağırlıklı ortalama veya voting ile birleştirilmiş koleksiyonudur. Modeller topluluğa, toplulukta hâlihazırda bulunan modellerle birlikte, kestirimlerinin ortalamaları alınarak eklenirler (Kuncheva, 2004).

- **ClassBalancedND and DataNearBalancedND:** Dong, Frank ve Kermer (2005) çok sınıflı verikümelerini 2 sınıflı sınıflandırıcılarla rasgele sınıf - dengeli ağaç yapısı yapılandırarak ele almak için üst düzey sınıflandırıcı sundular (Frank ve Kermer, 2004).

- **ND:** Dong, Frank ve Kermer (2005) çok sınıflı verikümelerini 2 sınıflı sınıflandırıcılarla rasgele ağaç yapısı yapılandırarak ele almak için üst düzey sınıflandırıcı sundular.

- **Dagging:** Ting ve Witten (1997) veriden bir miktar ayrışık, katmanlı kıvrımlar (stratified folds) yaratan ve her bir veri yığınını tedarik (supplied) tabanlı

sınıflandırıcının bir kopyasına sağlayan meta sınıflandırıcı önerdi. Tüm üretilmiş taban sınıflandırıcılar Vote Meta sınıflandırıcının içinde olduğu için kestirimler oy çoğunluğu ile yapılır. Bunlar, eğitim verisindeki örneklerin sayısına göre zaman davranışı karesel (quadratic) veya kötü olan taban sınıflandırıcılar için faydalıdır.

- **Decorate:** DECORATE (Melville ve Mooney, 2003) özel yapılandırılmış yapay eğitim örnekleri kullanarak çeşitli sınıflandırıcı toplulukları üretmek için bir meta-öğrenicidir (learner). Karşılaştırmalı deneyler bu tekniğin daima taban sınıflandırıcı Bagging ve Random Forests'a göre, doğruluğunun daha fazla olduğunu göstermiştir. Küçük eğitim kümelerinde daha yüksek doğruluk sağladığı ve büyük eğitim kümelerinde kıyaslanabilir performans başardığı için Decorate Boosting'e tercih edilir.

- **END:** Dong, Frank ve Kermer (2005) çok sınıflı veri kümelerini 2 sınıflı sınıflandırıcılarla Nested (içiçe) Dichotomies (ikiye ayrılma) toplulukları yapılandırarak ele almak için üst düzey sınıflandırıcı sundular.

- **Ensemble Selection:** Caura, Niculescu, Crew ve Ksikes (2004) binlerce modelin kütüphanesinden topluluklar yapılandırmak için bir metot önerdiler. Model kütüphaneleri farklı öğrenme algoritmaları ve parametre ayarları kullanılarak üretildi; doğruluk, çapraz entropi ve ortalama kesinlik gibi, performans ölçütlerini maksimuma çıkaracak modelin topluluğa eklenmesi için ileri aşamalı seçim kullanıldı.

3.1.5. Makine Öğrenmesi Algoritmalarının Misc Sınıfı

MISC sınıfı aşağıdaki algoritmaları içerir:

- **FLR:** WEKA (2007) Bulanık (Fuzzy) Lattice Nedenleme (Reasoning) Sınıflandırıcısı (Classifier) (FLR) v5.0. sağlar. Bulanık Lattice Nedenleme Sınıflandırıcısı Nedenleme Çevresi yaratmak için Bulanık Lattice kavramını (notion) kullanır. Son versiyonu sayısal kestirimciler kullanarak sınıflandırıcılar için kullanılabilir.

- **VFI:** Demiroz ve Güvenir (1997) VFI (Voting Feature Intervals – Öznitelik Aralıklarını Oylama) adında yeni bir sınıflandırma algoritması sundu. Konsept, öznitelik aralıkları kümesiyle her bir öznitelik boyutunda ayrı ayrı sunuldu. Her öznitelik sınıflandırmaya sınıflar arasında gerçek değerli oylar dağıtarak katılır. Yüksek oyu alan sınıf kestirimi yapılan sınıf olarak ilan edilir. Aralıklar her bir sınıf etrafında her bir öznitelik için (temel olarak diskretizasyon) yapılandırıldı. Sınıf sayıları her bir özellik üzerindeki her bir aralık için kaydedilir. Sınıflandırma voting ile yapılır.

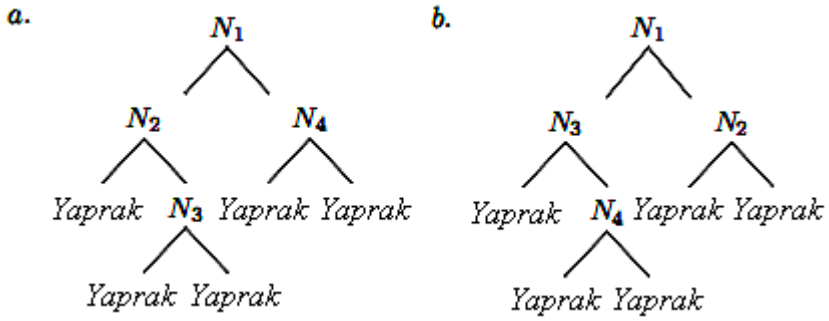
- **HyperPipes:** WEKA (2007) HyperPipe sınıflandırıcısı uygulayan sınıf içerir. Her bir kategori için o kategorinin tüm noktalarını içeren HyperPipe yapılandırılır (esasen herbir kategori için gözlenen özellik ölçülerini kaydeder). Test örnekleri "en çok örnek içerir" kategorisine göre sınıflandırılır. Test durumlarında sayısal sınıf veya kayıp değerleri ele almaz. Çok basit bir algoritma olmasına rağmen, çok hızlı olması ve çok fazla öznitelik olduğunda da çok iyi çalışması gibi avantajları vardır.

3.1.6. Makine öğrenmesi algoritmalarının Trees sınıfı

Trees sınıfında bulunan çeşitli algortimalar aşağıda tartışıldığı gibidir:

- **BFTree:** Shi (2007) en iyi-ilk karar ağaçlarının, standart derinlik-ilk karar ağaçlarına benzer, böl ve fethet (divide and conquer: büyük programları küçük parçalar halinde yazıp birleştirmek) modasıyla yapılandırıldığını tanımlar. En iyi–ilk ağaç şu şekilde oluşturulur: İlk önce, kök düğüme yerleştirmek için bir öznitelik seçilir ve bazı dallanmalar yapılır. Sonra kök düğümden uzanan her bir dal için bir tane olmak üzere eğitim örnekleri alt kümelere bölünür. Shi (2007) sadece ikili karar ağaçlarını ele aldı ve bu yüzden dalların sayısı tam olarak ikiydi. Sonra, bu adım seçilen dal için sadece gerçekten ona ulaşan örnekleri kullanarak tekrarlandı. Her bir adımda, genişlemek için uygun tüm alt kümeler içerisinden en iyi alt küme seçilebilir. Bu yapılandırma işlemi tüm düğümler saf (pure) kalana kadar veya belirli sayıda genişlemeye ulaşılıncaya kadar devam eder. Şekil 3.4. varsayımsal (hypothetical) ikili en iyi- ilk ağaç ve varsayımsal ikili derin- ilk ağaç arasındaki bölünme düzenindeki farkı gösteriyor (Shi, 2007). Sıra derin-ilk durumda daima aynı olsada en iyi-ilk ağaç için diğer sıralamaların seçilebileceğine dikkat edilmelidir. En iyi-ilk karar ağaçlarında gelişen problem ele

verilecek özelliğin nasıl belirleneceği ve verinin nasıl bölüneceğidir. Çünkü karar ağaçlarının en önemli amacı doğru ve küçük modeller aramaktır, saf düğümler en kısa sürede bulunmaya çalışılır.



Şekil 3.4. Karar ağaçları (a) varsayımsal (hypothetical) derinlik- ilk karar ağacı (b) varsayımsal en iyi-ilk karar ağacı

- **J48:** Quinlan (1993) budanmış veya budanmamış C4.5 karar ağacı üretmek için sınıf tanımladı, budama uygulayan ve ağaçları kurallara çeviren metotlar anlattı. Bu algoritma başlangıç ağacı yapılandırmak için kullanılır. Eksik öznitelik değerleri gibi çeşitli problemler ele alır. Karar ağacı algoritmaları durumlar veya örnekler kümesiyle başlar ve yeni durumları sınıflandırabilmek için kullanılan ağaç veri yapısı yaratır. Her durum sayısal veya sembolik değer alabilen öznitelikler kümesi ile tanımlanır. Her bir eğitim durumu sınıfın adıyla ifade edilen etikettir. Karar ağacının her bir iç düğümü test içerir, testin sonuçları o düğümden hangi dalın takip edileceğine karar vermek için kullanılır. Yaprak düğümleri testler yerine sınıf etiketleri içerir. Sınıflandırma modu test durumunda, etiket yoksa yaprak düğümüne ulaşır, C4.5 orada bulunan etiketi kullanarak sınıflandırır. Bu alanların çoğu için, C4.5 tarafından üretilen ağaçlar küçük ve doğrudurlar, hızlı ve güvenilir sınıflandırıcılar meydana getirirler. Bu özellikler karar ağaçlarını sınıflandırma için değerli ve popüler araçlar yapar (Quinlan, 1986). Böl ve fethet algoritması her yaprak tek sınıf durumu içerene kadar veya iki durumun farklı sınıflara ait olmalarına rağmen her bir öznitelik için aynı değerlere sahip olduğu daha fazla bölünmenin imkânsız olduğu duruma kadar veriyi parçalara ayırır. Sonuç olarak, çelişkili durumlar içermiyorsa, karar ağacı tüm eğitim durumlarını doğru sınıflandıracaktır. Aşırı uyma, bazı eğitim durumları kümelerini alt bölümlere ayrılmaktan koruyan durdurma kriterleri ile veya üretildikten sonra karar ağacının bir kısım yapısını kaldırarak önlenebilir (Kaariainen ve diğerleri, 2004).

- **LMT:** Landwehr, Hall ve Frank (2004) Lojistik Model Ağaçlar (Logistic Model Trees) yapılandırmak için sınıflandırıcı belirledi. Bu ağaçlar yapraklarında lojistik azalma fonksiyonları olan sınıflandırma ağaçlarıdır. Algoritma ikili ve çoklu-sınıf hedef değişkenleriyle, sayısal veya nominal özelliklerle ve eksik (missing) değerlerle ilgilenebilir.

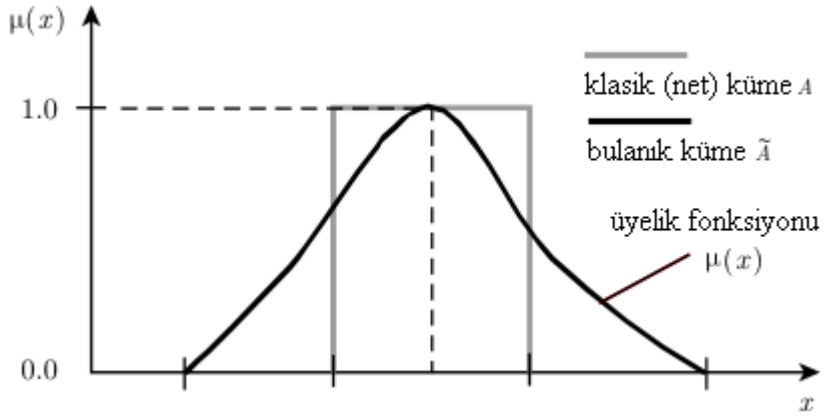
- **NBTree:** Kohavi (1996) daha önce tanıtılan Naive-Bayes indüksiyon algoritmalarının şaşırtıcı biçimde birçok sınıflandırma görevinde, hatta dayandıkları şartlı bağımsızlık varsayımları oylandığında bile doğru olduğu görüşünü tartıştı. Bununla beraber, çalışmaların çoğu küçük veritabanlarında gerçekleştiriliyor. Naive-Bayes'in doğruluğu karar ağaçlarında olduğu kadar yüksek değildir. Sonrasında hibrit karar ağacı ve Naive-Bayes sınıflandırıcılarına sebep olan NBTree adında yeni bir algoritma önerildi: karar ağacı düğümleri normal karar ağaçları gibi tek değişkenli ayrılma içerir, fakat yaprakları Naive-Bayes sınıflandırıcıları içerir.

- **RandomForest:** Breiman (2001) ağaç kestirimcilerin kombinasyonu olarak rasgele ormanlar tanımlar. Öyle ki, her bir ağaç bağımsız örneklenen ve ormandaki tüm ağaçlar için aynı dağılımlı rasgele vektörün değerlerine dayanır. Ormanlar için genelleme hatası ormandaki ağaç sayısı arttıkça limite yakınsar. Ağaç sınıflandırıcıların ormanın genelleme hatası ormandaki münferit (individual) ağaçların gücüne ve aralarındaki korelasyona (bağlılaşım) dayanır.

- **Simple CART**: Breiman, Friedman, Olshen ve Stone (1984) minimal maliyet-karmaşa budama uygulayan sınıf üzerine tartıştılar. Eksik değerlerle uğraşırken, vekil *bölünme metodu (surrogate split method) yerine "kesirli örnekler-fractional instances"* metodu kullanılır. İkili ağaçlar sınıflandırma veya regresyon problemlerinde veriye bakmanın ilginç ve sıklıkla açıklayıcı yolunu verir. Diğer metotların dışlanması için kullanılmamalıdırlar, daima daha iyi oldukları söylenemez.

3.2. Mamdani Tabanlı Bulanık Çıkarım Sistemi

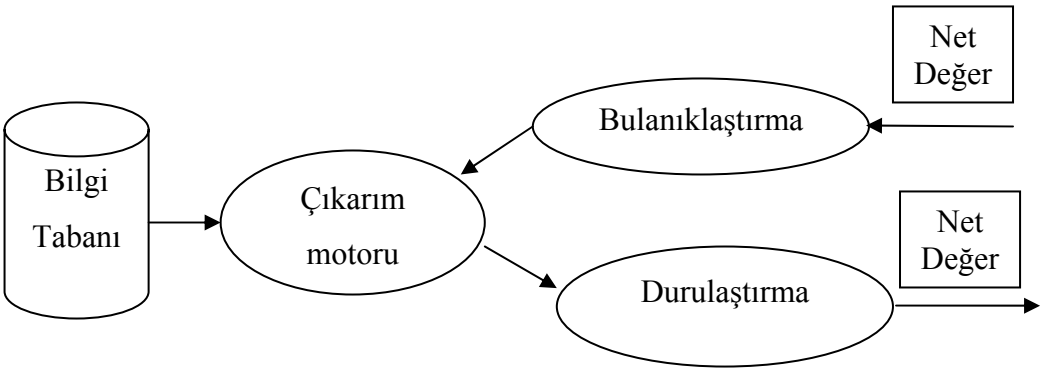
Bulanık mantık, bir bulanık küme mantığına dayanır ve ilk olarak Lofti A.Zadeh tarafından tanımlanmıştır. Bulanık küme, küme'ye aitlik derecesi *üyelik değeri* ile tanımlanmış olan kümeyi ifade eder. Klasik küme kavramında bir eleman bir kümenin üyesidir veya değildir. Bulanık mantıkta küme aitlik derecesi μ, 0 ile 1 arasında değişir. 0 kümeye ait olmamayı, 1 ise kesin olarak o kümenin üyesi olmayı gösterir. Küme aitlik derecesi üçgen, yamuk, Gaus eğrisi gibi standart fonksiyonlarlarla tanımlanabildiği gibi çok farklı fonksiyonlarda oluşturulabilir (Topuz, Akbaş, Tektaş, 2002).



Şekil 3.5. Bulanık kümenin üyelik fonksiyonu

Bulanık çıkarım eşlemeyi (mapping) verilen girdiden (input) çıktıya (output) bulanık mantık kullanarak formüle eden süreçtir. Eşleme kararların alınabileceği veya örüntülerin ayırt edilebileceği bir altyapı sağlar. Bulanık çıkarım süreci *Üyelik Fonksiyonları* (Membership Functions), *Mantıksal Operasyonlar* (Logical Operations) ve *If-Then kuralları* gibi birçok özelliği kapsar.

Bulanık çıkarım sistemleri *otomatik kontrol*, *veri sınıflandırma*, *karar analizi*, *uzmanlık* (expert) *sistemleri* ve *bilgisayar görüntüsü* (computer vision) gibi alanlarda başarıyla uygulanmıştır. Çoklu-disipliner doğasıyla bulanık çıkarım sistemleri *bulanık kural tabanlı sistemler*, *bulanık uzmanlık sistemleri*, *bulanık modelleme*, *bulanık çağrışımlı bellek* (associative memory), *bulanık mantık kontrolcüleri* ve *basit* (ve muğlak olarak) *bulanık sistemler* gibi birçok isimle ilişkilendirilmişlerdir.



Şekil 3.6. Bulanık Çıkarım Sistemi Şeması

Genel olarak bulanık çıkarım sistemi 4 modülden oluşur.

Bulanıklandırma modülü (Fuzzification module): Net değerler olan sistem girdilerini bulanık kümelere çevirirler. Bu bulanıklaştırma fonksiyonu uygulandığında gerçekleşir.

Bilgi tabanı (Knowledge base): Uzmanlar tarafından sağlanan if-then kurallarını saklar.

Çıkarım motoru (Inference engine): Girdiler ve if-then kuralları üzerinde bulanık çıkarım yaparak insan düşünce sürecini simüle eder.

Durulaştırma modülü (Defuzzification module): Çıkarım motorundan elde edilen bulanık kümeyi net değere dönüştürür.

En iyi bilinen bulanık kural tabanlı çıkarım sistemleri Mamdani bulanık metodu ve Tagaki-Sugeno (T-S) bulanık metodudur. Mamdani'nin metodu *bulanık küme teorisi* kullanılarak üretilen ilk kontrol sistemleri arasındadır. 1975 yılında Ebrahim Mamdani tarafından deneyimli (insan) operatörlerden elde edilen linguistik kontrol kuralları kümesini bireştirerek (synthesizing) buhar makinesi ve buhar kazanı kombinasyonunu kontrol etme teşebbüsüyle önerilmiştir. Mamdani'nin çalışmaları Lotfi Zadeh'in (1973) *karmaşık sistemler* ve *karar süreci* ile ilgili bulanık algoritmalarına dayanıyordu.

Mamdani tabanlı bulanık çıkarım sistemlerinin avantajları aşağıdaki gibidir:

- Sezgiseldir (intuitive).
- Geniş çapta kabul edilmiştir.
- İnsan idrakine uygundur.

Tagaki-Sugeno metodunun avantajları aşağıdaki gibidir:

- Hesaplama açısından verimlidir (efficient)
- Lineer tekniklerle iyi çalışır (T. Takagi ve M. Sugeno, 1983 ve 1985)
- Optimizasyon ve uyarlama teknikleriyle iyi çalışır.
- Matematiksel analizle iyi uyumludur.
- Çıktı yüzeyinin devamlılığını garantiler (T. Takagi ve M. Sugeno, 1983 ve 1985).

Abraham'a (2005) göre Mamdani Nöral-bulanık sistemi üyelik sistemi parametrelerini öğrenmek için denetimli öğrenme teknikleri (geriyayılım öğrenmesi) kullanır. (Yen ve Langri, 2003). Mamdani nöral-bulanık sistemi Şekil 3.7.'de gösterilmiştir. Her bir katmanın fonksiyonu aşağıdaki gibidir:

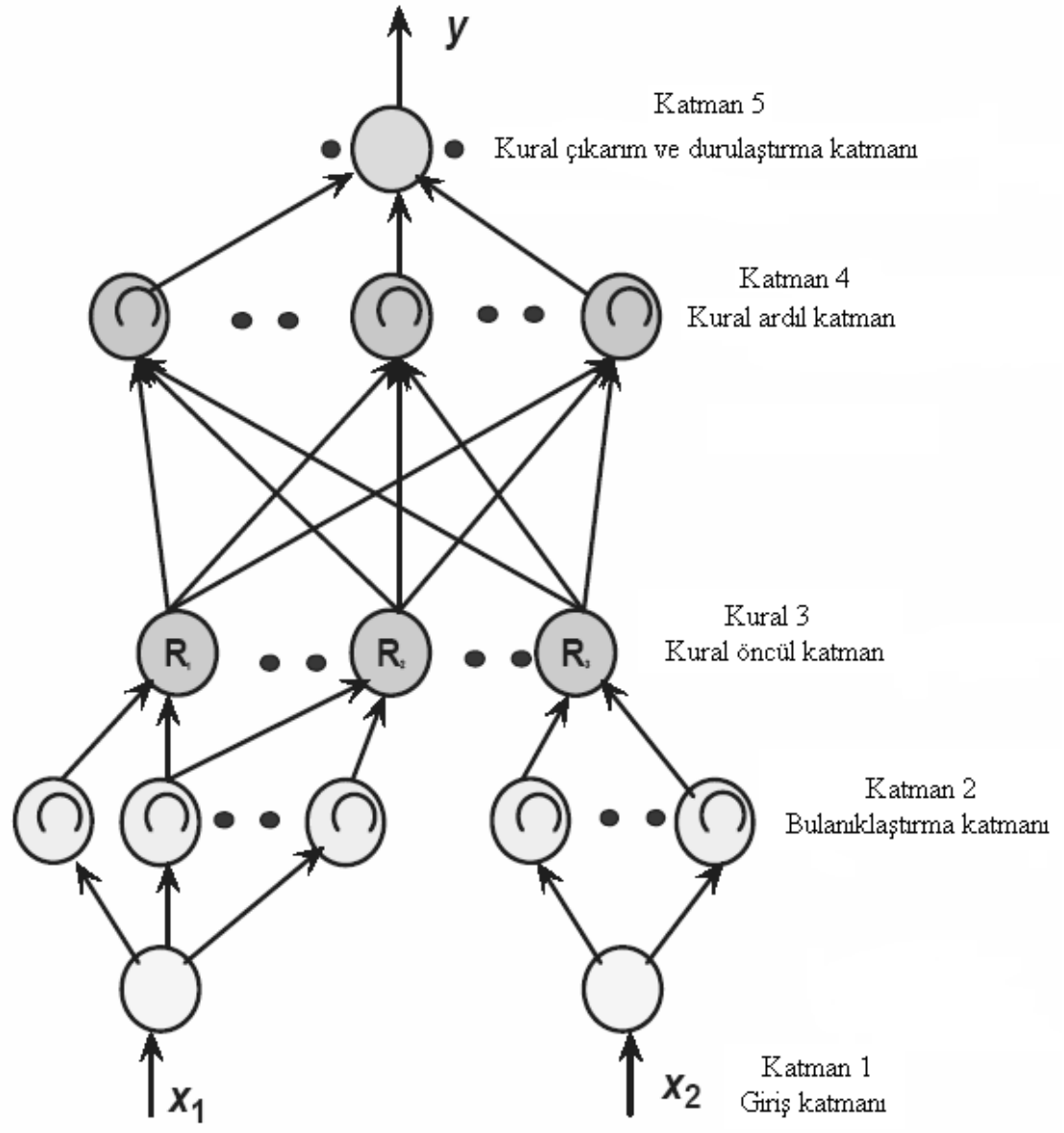
Katman-1 (Girdi Katmanı): Bu katmanda hiçbir hesaplama yapılmaz. Bu katmandaki bir girdi değişkenine karşılık gelen her düğüm sadece girdi verilerini bir sonraki katmana direk iletir. Katman-1'de bağlantı (link) ağırlığı tekliktir (unity).

Katman-2 (Bulanıklaştırma Katmanı): Bu katmandaki her bir düğüm bir linguistik etikete (mükemmel, iyi, vs..), katman-1'deki girdi değişkenine karşılık gelir. Diğer bir deyişle, çıktı bağlantısı üyelik değerini sunar. Bu değer, katman-2'de hesaplanan, bulanık kümeye ait girdi verisinin derecesini nitelendirir. Kümeleme (clustering) algoritması her bir giriş değerine tahsis edilecek başlangıç (initial) sayısına ve üyelik fonksiyonu çeşidine karar verir. Üyelik fonksiyonlarının son şekilleri ağ öğrenmesi sırasında ayarlanır.

Katman-3 (Kural Öncül Katman): Bu katmandaki düğüm kuralın öncül bölümünü temsil eder. Bu düğümde genellikle T-norm operatörü kullanılır. Katman-3 düğümünün çıktısı, karşılık gelen bulanık kuralın, halka (ring) gücünü temsil eder.

Katman-4 (Kural Ardıl Katman): Bu düğümün temel olarak 2 görevi vardır. Gelen kural öncüllerini birleştirmek ve bulundukları çıktı linguistik etiketine (yüksek, orta, düşük vs…) derece atamaktır. Bu katmandaki düğümlerin sayısı kuralların sayısına eşit olacaktır.

Katman-5 (Kombinasyon ve Durulaştırma Katmanı): Bu düğüm T-conorm operatorü kullanılarak ardıl olarak gelen tüm kuralların kombinasyonunu yapar ve son olarak net (crisp) değeri hesaplar.



Şekil 3.7. Mamdani Nöral-Bulanık Sistem Yapısı (Abraham, 2005)

J-S. R. Jang ve C.T. Sun, 1995'a göre bulanık sistem *f* olarak adlandırılan parametrelerle ifade edilmiş nonlineer harita olarak ele alınabilir.

$$f(x) = \frac{\sum_{l=1}^{m} y^{l} \left(\prod_{i=1}^{n} \mu_{A_i^l}(x_i) \right)}{\sum_{l=1}^{m} \left(\prod_{i=1}^{n} \mu_{A_i^l}(x_i) \right)} \tag{3.3.}$$

y^l çıktı hanesidir. Mamdani çıkarımı uygulandığında tektir, Sugeno çıkarımı uygulandığında sabittir. Üyelik fonksiyonu $\mu_{Ai}^{l}(x_i)$, Kural 1'in x=[x_1, x_2, x_3,… x_m] girdisine karşılık gelir. Öncüldeki "ve-and" bağlacı ürün tarafından ve durulaştırma yerçekimi metodunun merkezi tarafından gerçekleştirilmiştir. Aşağıdaki kurallara sahip Sugeno tipi bulanık sistemin kurallarını ele alalım:

Kural 1: If x is A1 and y is B1, then f1= $p_1x + q_1y + 1$

Kural 2: If x is A2 and y is B2, then f2= $p_2x + q_2y + r2$

Bulanık kümelerin üyelik fonksiyonları (Ai, Bi, i=1,2), μ_{Ai}, μ_{Bi} olsun.

-Yukarıdaki kurallar için kural öncül sonuçları i = 1,2 iken $w_i = \mu_{Ai}(x) * \mu_{Bi}(y)$'de değerlendirme.

- önermelerin değerlendirilmesi ve kural ardılları (3.4.)'ü verir.

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} \tag{3.4.}$$

$$\overline{w_i} = \frac{w_i}{w_1 + w_2} \tag{3.5.}$$

Olursa, bu durumda f, (3.6.)'daki gibi yazılabilir.

$$f = \overline{w_1} f_1 + \overline{w_2} f_2 \tag{3.6.}$$

3.3. Uyarlamalı Nöral-Bulanık Sistemler

Nöral-Bulanık sistem veri örneklerini işleyerek parametrelerini (bulanık kümeler ve bulanık kurallar) belirlemek için nöral ağ teorisinden çıkartılmış veya esinlenilmiş öğrenme algoritmalarını kullanan bulanık sistemdir.

Öğrenme (sezgisel) prosedürü yerel bilgi üzerinde çalışır ve temel bulanık sistemde sadece yerel modifikasyonlara sebep olur.

Nöral-bulanık sistem ileribildirim sinir ağında 3 katman olarak görülebilir. İlk katman girdi değişkenlerini, orta (saklı) katman bulanık kuralları üçüncü katman çıktı değişkenlerini temsil eder. Bulanık kümeler (bulanık) bağlantı ağırlıkları olarak şifrelenmişlerdir. Üzerine öğrenme algortiması uygulamak için böyle bir bulanık sistem sunmak gerekli değildir. Bununla birlikte elverişli olabilir, çünkü veri akışını ve öğrenmeyi model içinde sunar.

Not: Bazen bulanık kümelerin ikinci veya dördüncü katman birimlerinde kullanıldığı 5 katmanlı mimari kullanılır.

Nöral bulanık sistem daima (yani önce, esnasında ve öğrenme sonrasında) bulanık kurallar sistemi olarak yorumlanmıştır. Bulanık kurallar biçimindeki öncül bilgiden ilklendirme mümkün olduğu için, sistemi, eğitim verisinden ayrıca en başından itibaren yaratmak mümkündür.

Not: Tüm nöral-bulanık modeller öğrenme modellerini bulanık kural yaratmak için nitelendirmezler.

Nöral-bulanık sistemin öğrenme prosedürü temel bulanık sistemin semantik özelliklerini dikkate alır. Sistem parametrelerine uygulanabilecek olası modifikasyonlarda kısıtlılıklarla sonuçlanır.

Not: Tüm nöral-bulanık yaklaşımlar bu özelliğe sahip değildirler.

Uyarlamalı (Adaptive) Nöral-Bulanık Çıkarım Sistemi (ANFIS) üyelik fonksiyonu parametreleri sadece *geriyayılma algortiması* veya *geriyayılma ve küçük kareler metodu* ile birlikte ayarlanmış (adjusted) bulanık çıkarım sistemi yapılandırır. Bu ayarlama, bulanık sisteminizin modellenen veriden öğrenmesini sağlar.

ANFIS, komplex bir çıkarım sistemidir ve sadece Sugeno modelleme sistemlerini destekler. Bu model şu şartları sağlamalıdır:

- İlk veya sıfırıncı sıra Sugeno-sistem olmalı.
- Ağırlıklandırılmış ortalama durulaştırma ile elde edilmiş tek çıktıya sahip olmalıdır.
- Tüm çıktı üyelik fonksiyonları aynı ve lineer ya da constant olmalıdır.
- Kural paylaşımı yoktur. Farklı kurallar aynı çıktı üyelik fonksiyonunu paylaşamazlar. Yani çıktı üyelik fonksiyonu kural sayısına eşit olmalıdır.

4. MATERYAL VE METOD

Yazılım modülünün hata-eğilimi, modülün hata barındırma olasılığıdır. Yazılımın hata eğilimi ve kodun ölçülebilir özellikleri (başka bir ifadeyle statik ölçütler) ve test etme (başka bir ifadeyle dinamik ölçütler) arasında ilişki oluşur. Hata-eğilimli yazılım bileşenlerinin erken algılanması, doğrulama uzmanlarının gelişme altındaki yazılım sistemlerinin problemli alanlarında kendi zamanlarına ve olanaklarına konsantre olmalarına izin verecektir. Yazılım kalitesi modellerinin, kritik bileşenleri kesin olarak tanımlamak için yeteneği, manuel denetlemeden otomatik biçimsel analiz metotlarına kadar odaklanmış doğrulama aktivitelerinin uygulamaları için izin verir (Bellini, 2005). Yazılım kalitesi modelleri teslim edilen ürünlerin güvenilirliğini garantiler. Özellikle büyük ölçekli gelişme çabaları için yazılım gelişimi yaşam çevriminin erken evresinde iyi yazılım kalitesi modelleri geliştirmek ve uygulamak önemli hale gelmiştir. Yazılım kalitesi kestirim modelleri bileşenin hata toleranslı olup olmadığını kestirmeye çalışır. Hata-eğilimli yazılım modüllerini belirlemeye çalışan metotlar, verimli doğrulamayla maliyetten kurtulmaya olanak tanınması, zamanlama ve olanak planlama için yardım sağlarlar. Bu modeller bir modulün sınıfı (örneğin hata eğilimli veya hata-eğilimli değil) veya modül için kalite faktörü (örneğin hataların sayısı) olabilen cevap değişkenini kestirmek için kullanılabilirler. İlki genellikle sınıflandırma modeline ikincisi ise kestirim modeline tekabül eder. Yazılım kalitesi modelleri ürün ve süreç özelliklerine dayanan yazılım bileşenlerinin kalite faktörlerini kestirmeye çalışır.

Yazılım kalitesi kestirimi temel hipotezi şu şekildedir: Eğer aynı ortamda geliştirilmiş önceki projede, aynı ürün veya süreç ölçütlü modül, hata eğilimli ise, gelişme altındaki modül hata toleranslıdır. Bu nedenle, yürürlükteki projenin erken aşamasında bulunan bilgi veya önceki projeden geçerli bilgi kestirim üretmek için kullanılabilir. Bu metodoloji geniş ölçekli projelerde çok yararlıdır. Hata toleranslı modüllerin kesin kestirimi kritik yazılım bileşenleri üzerinde doğrulama ve onaylama aktivitelerine olanak verir. Bu nedenle, yazılım geliştiricilerinin yazılım kalitesi modellerine özel ilgisi vardır. Hata toleransı kestirim modellerinin verimli ve kesin olması istenir. Hata toleranslı modeller kod ve kodun hataları hakkındaki bilgiden ve kod-hata ilişkisinden gelen bilgiden geliştirilen modellerdir.

Yazılımda sınıfların varlığı geçmiş veriden hata toleransı modellerini türetmeye izin verir ve sonra bu modelleri, özellikle güvenlik-kritik uygulamalarda, aynı sınıfın yeni yazılım uygulamalarının hata toleransını kestirmek için kullanır. Test aktivitelerini planlarken, yazılım hata toleransı hakkındaki bilgiler, maliyetleri önceden bilme ve aktiviteleri paylaştırma açısından avantajlı olabilir. Aynı zamanda, test uygulaması, bu bilgiyi sonuçların kalitesini değerlendirmek için kullanabilir.

4.1. Amaçlar

Bu tezin amacı yazılım bileşenlerinde hatalı modülleri bulmak ve bu modüllerin bakım aciliyet derecesini araştırmaktır. Temel hedefler aşağıdaki gibidir:

- Yazılım sistemlerinin yapısal kodunu ve tasarım özniteliklerini bulmak
- Yazılım bakımı aciliyeti için en iyi algortimaları/yöntemleri bulmak

4.2. Metodoloji

Metodoloji aşağıdaki adımları içerir:

4.2.1. Yapısal kodun ve tasarım özelliklerinin bulunması

İlk adım, yazılım sistemlerinde yapısal kodun ve tasarım özelliklerinin başka bir ifadeyle yazılım ölçütlerinin bulunmasıdır. Gerçek zamanlı eksiklik veri kümeleri NASA'nın MDP (ölçüt veri programı - Metric Data Program) veri havuzundan alınmıştır. Veri kümesi NASA tarafından geliştirilen güvenlik-kritik yazılım sistemleri ile ilişkilidir.

4.2.2. İfadenin sunumu olarak uygun ölçüt değerlerinin seçimi

Bu veri kümelerinin arasından ürün (product) modül ölçütü gibi uygun ölçütler ele alındı. Ürün terimi modül düzey verisine istinaden kullanıldı. Ölçütler verisi terimi ürünün ölçülmüş özelliklerini ve karakteristiğini tanımlayan herhangi bir sonlu sayısal değerle ilgilidir. Ürün, eksiklik verisi ve ölçüt verisinin birleştirilebileceği bir birimdir.

Birçok durumda ürünler fonksiyon ve sistemler/alt-sistemler gibi kodla ilişkili öğelerle eşanlamlı olurlar.

Ölçütler aşağıdaki gibidir:

- MODULE – Bu ölçüt ürünün tek sayısal belirleyicisidir.
- LOC_BLANK – Bu ölçüt modüldeki boş satırları tanımlar başka bir ifadeyle üzerlerinde hiç bir şey bulunmayan satırları.
- BRANCH_COUNT – Bu ölçüt dallanma hesap ölçütünü tanımlar başka bir ifadeyle her bir modüldeki dallanma sayısını tanımlar. Dallanmalar karar düğümünden çıkan kenarlar olarak tanımlanmıştır. Program modülündeki dallanmaların sayısı ne kadar fazlaysa, test olanaklarına ihtiyaç da o kadar fazladır.
- CALL_PAIRS – Bu ölçüt modüldeki diğer fonksiyonlara çağrıların sayısını tanımlar.
- LOC_CODE_AND_COMMENT –Bu ölçüt modülde aynı anda kod ve yorum bulunduran satırların sayısını tanımlar.
- LOC_COMMENTS –Bu ölçüt modüldeki yorum içeren satırların sayısını tanımlar. Kodun tamamen yorumlanmış ve bu yüzden derleyici tarafından tamamen ihmal edilmiş tüm satırlarını içerir.
- CONDITION_COUNT – Bu ölçüt verilen modüldeki şartların sayısını tanımlar.
- CYCLOMATIC_COMPLEXITY – Bu ölçüt modülün kod karmaşıklığını tanımlar. Lineer bağımsız yolların sayısıdır. Programın verilmiş kontrol akış çizgesi (G) ve kodun karmaşıklığı V(G) iken V(G)=E-N+2 şeklinde hesaplanabilir.

- CYCLOMATIC_DENSITY – Bu ölçüt modülün kod karmaşıklığının uzunluğuna oranını tanımlıyor. Amaç karmaşıklığın büyüklük bileşenini çarpanlara ayırmaktır. Modülün karmaşıklığını standartlaştırma etkisi vardır ve bu yüzden bakımı zordur.
- DECISION_COUNT – Verilen modüldeki karar noktalarının sayısını tanımlar.
- DECISION_DENSITY – Bu ölçüt CC/LLOC şeklinde hesaplanır. Burada CC kodun karmaşıklığı ve LLOC kodun mantıksal satırıdır (logical line of the code). Bu ölçüt projenizin prosedürleri çerçevesinde satırların ortalama kod yoğunluğunu gösterir.
- DESIGN_COMPLEXITY – Bu ölçüt modülün tasarım karmaşıklığını tanımlar. Tasarım karmaşıklığı programdaki modüller arasındaki etkileşimi ölçer.
- DESIGN_DENSITY – Modülün tasarım yoğunluğunu tanımlar ve tasarım karmaşıklığı bölü kod karmaşıklığı olarak hesaplanır.
- EDGE_COUNT – Bu ölçüt verilen modüldeki kenarların sayısını tanımlar. Bir modülden diğerine kontrol transferini sunar.
- ERROR_COUNT – Bu ölçüt modülle ilişkili hataların sayısını tanımlar.
- ERROR_DENSITY – Bu ölçüt, modül için, kodun 1000 satır başına eksiklik sayısını tanımlar. ERROR_DENSITY = 1000*(ERROR_COUNT/LOC_TOTAL) şeklinde hesaplanır.
- ESSENTIAL_COMPLEXITY – Modülün başlıca karmaşıklığını hesaplar. Hesaplamak için ilk önce, yapısal programın temel öğeleri modülün kontrol akış çizgesinden, çizge daha fazla indirgenemeyinceye kadar kaldırılır, daha sonra indirgenen çizgenin kod karmaşıklığı hesaplanır.
- ESSENTIAL_DENSITY – Zaruri yoğunluk (ev (G)-1)/ (v (G)-1) şeklinde hesaplanır. Burada ev (G) zaruri yoğunluğu ve v (G) kod karmaşıklığını temsil eder.

- LOC_EXECUTABLE – Modüldeki yürütülebilir kodun satır sayısıdır. Bu kodun tam olarak yorumlanmamış ve boş satırları da dâhil olmak üzere tüm satırlarını içerir.
- PARAMETER_COUNT – Verilen modüldeki parametrelerin sayısını tanımlar.
- GLOBAL_DATA_COMPLEXITY – Global veri karmaşıklığı modülün yapısının kod karmaşıklığını ölçülendirir (global/parametre verisiyle ilişkili olduğu için). Global veri karmaşıklığı modülle olan global ve parametre verisinin karmaşıklığını ölçer. Programda çoklu modüller tarafından erişilebilen veriye global veri denir. Global verinin kullanılışı, modüle potansiyel bakım problemlerine sebep olabilecek dış veri eşlemeyi getirir.
- GLOBAL_DATA_DENSITY – Global veri yoğunluğu gdv(G)/v(G) şeklinde, yani global veri karmaşıklığını kod karmaşıklığına bölerek hesaplanır.
- HALSTEAD_CONTENT – Bu ölçüt modülün Halstead uzunluk içeriğini tanımlar. Halstead ölçüleri programın direkt kaynak kodundan türetilmiş 4 basamaklı sayılara dayanır.

  n1 ayrı işleçlerin (operator) sayısıdır,

  n2 ayrı işlenenlerin (operand) sayısıdır,

  N1 tüm işleçlerin sayısıdır ve

  N2 tüm işlenenlerin sayısıdır.

  Bu nedenle, Halstead uzunluk içeriği n = n1 + n2'dir.
- HALSTEAD_DIFFICULTY – Programın zorluk derecesi veya hata eğilimi (D) programdaki tek operatörlerle orantılıdır. D ayrıca tüm operatörlerin sayısının tek operatörlerin sayısına oranına orantılıdır (yani aynı operatörler programda çok kullanılmışlarsa, hatalara daha fazla eğilimlidir). D = (n1 / 2) * (N2 / n2) şeklinde hesaplanır.

- HALSTEAD_EFFORT – Bu ölçüt modülün Halstead çaba ölçütünü tanımlar. Çaba programı uygulamak için gerekli olan mental ayrımların (discriminations) sayısıdır ve program okumak ve anlamak için çaba gereklidir. Uygulamak veya programı anlamak için gerekli olan çaba (E) programın hacim (V) ve zorluk seviyesiyle orantılıdır. E= D * V'dir.
- HALSTEAD_ERROR_EST - Bu ölçüt modülün Halstead hata kestirim ölçütünü tanımlar. Uygulamadaki hata sayılarının kestirimidir.
- HALSTEAD_LENGTH - Bu ölçüt operatör ve operand oluşumlarının toplam sayısını içeren modülün Halstead uzunluk ölçümünü tanımlar.
- HALSTEAD_LEVEL - Bu ölçüt modülün Halstead seviye ölçümünü yani programın anlaşılabileceği seviyeyi tanımlar. Program seviyesi (L) programın hata eğiliminin aksidir. Düşük seviyeli program hatalara yüksek seviyeli programdan daha eğilimlidir. L=1/D şeklinde verilir.
- HALSTEAD_PROG_TIME – Bu ölçüt modülün Halstead programlama zaman ölçümünü tanımlar. Bu algoritmanın uygulanması için gerekli olan tahmini bir süredir. Programın uygulanması veya anlaşılması için gerekli olan zaman (T) çabayla orantılıdır. Halstead eforun 18'e bölünmesinin yaklaşık zamanı saniye cinsinden verdiğini bulmuştur. T = E / 18
- HALSTEAD_VOLUME - Bu ölçüt, programın kodlanması için gerekli olan minimum sayıda biti içeren modülün Halstead hacim ölçütünü tanımlar. Program hacmi (V) programın matematiksel bitlerle ölçülen bilgi içeriğidir. Program uzunluğunun logaritma 2 tabanında sözcük haznesi (benzersiz operatör ve operandların sayısının toplamı) ölçüsü (n) ile çarpılmasıyla hesaplanır. V = N * log2 (n)

- MAINTENANCE_SEVERITY – Yürütülebilirlik güçlüğü ev (G)/v (G) olarak hesaplanır yani zaruri karmaşıklığın kod karmaşıklığına bölümüdür.
- MODIFIED_CONDITION_COUNT – Bu ölçüt modülde var olan modifiye edilmiş koşulların sayısını tanımlar. Karar sonucunu bağımsız olarak etkilemek sadece o koşulun değiştirilmesiyle mümkündür.
- MULTIPLE_CONDITION_COUNT – Bu ölçüt modülde var olan çoklu koşulların sayısını tanımlar.
- NODE_COUNT – Bu ölçüt verilen modülde bulunan düğümlerin sayısını tanımlar. Düğümler, daha fazla ilişkiye sahip karmaşıklık ölçütlerinin çoğunu hesaplamak için kullanılan taban ölçütlerdir.
- NORMALIZED_CYLOMATIC_COMPLEXITY – Bu ölçüt standartlaştırılmış kod karmaşıklığını tanımlar. Standartlaştırılmış karmaşıklık, basit bir ifadeyle, modülün kod karmaşıklığının modüldeki kodun satır sayısına bölümüdür. Yoğun karar mantığına sahip bir modül daha az yoğun karar mantığına sahip diğer bir modülden bakım için daha fazla çaba gerektirecektir.
- NUM_OPERANDS - Bu ölçüt modülün içerdiği operand sayısını tanımlar.
- NUM_OPERATORS - Bu ölçüt modülün içerdiği operatör sayısını tanımlar.
- NUM_UNIQUE_OPERANDS - Bu ölçüt modülün içerdiği benzersiz operandların sayısını tanımlar. Benzersiz değişkenlerin sayısıdır ve modül içinde sabittir.
- NUM_UNIQUE_OPERATORS - Bu ölçüt modülün içerdiği benzersiz operatörlerin sayısını, yani modüldeki ayrı operatörlerin sayısını tanımlar.
- NUMBER_OF_LINES – Bu ölçüt modüldeki satır sayısını tanımlar. Açık parantezden kapalı paranteze sade bir sayımdır ve karakter içeriğini dikkate almadan aradaki tüm satırları kapsar.

- PATHOLOGICAL_COMPLEXITY – Bu ölçüt modülün patolojik karmaşıklığını tanımlar.
- PERCENT_COMMENTS – Bu ölçüt kodun yorum olan yüzdesini tanımlar. Bu kodun tam olarak yorumlanmış satırlarının koddaki tüm satırlara bölünmesiyle elde edilen sayıdır ve okunurluğun bir göstergesi olarak kullanılır.
- LOC_TOTAL – Bu ölçüt verilen modül için tüm satırların sayısını tanımlar. Yürütülebilir ve yorumlanmış satırların toplamıdır. Boş satırlar sayılmamıştır.

4.2.3. Ölçütlerin analiz-tasfiyesi ve ölçüt değerlerinin standartlaştırılması

Bir sonraki adımda ölçütler analiz edildi, tasfiye edildi ve standartlaştırıldı ve sonra yazılım sistemlerinde hata toleransının modellenmesi için kullanıldı.

4.2.4. Yazılım sistemlerinde hata toleransı kestirimi için farklı bir algoritmanın keşfedilmesi

Tablo 4.1. Makine Öğrenmesi Algoritmalarının Sınıfları

Sr. No.	Makine Öğrenmesi Algoritmalarının Sınıfları
1.	Bayes
2.	Functions
3.	Lazy
4.	Meta
5.	Misc
6.	Trees

Yazılım bileşenlerinin hata tolerans sistemlerinin farklı düzeylerine sınıflandırılması için uygun algoritmanın bulunması çok önemlidir. Yazılım sistemlerinin hata toleransını kestirmede kullandığı makine öğrenmesi algoritmalarının çeşitli sınıfları Tablo 4.1.'de gösterildi. Tablo 4.1. makine öğrenmesi algoritmalarının 6

sınıfını gösteriyor. Uygulamada ilk önce eğitim evresi vardı, daha sonra üretilen kestirim modelini test etmek için 10 katlı çapraz onaylama kullanıldı.

***v*-kat çapraz onaylama** eldeki verinin daha iyi kullanılmasına olanak sağlamaktadır. *v* kat çapraz onaylamada orijinal örnek *v* alt örneğe ayrılır. Bu altörneklerden biri modelin test edilmesi için doğrulama verisi olarak ayrılır, diğer *v-1* örnek ise eğitim kümesi olarak kullanılır. Çapraz doğrulama süreci her bir altörneğin test verisi olarak kullanılması için *v* kez tekrarlanır. Bu tekrarlardan elde edilen *v* sonucun tek bir kestirim yapılmak üzere ortalaması alınır.

4.2.5. Algoritmaların karşılaştırılması

Karşılaştırmalar MAE ve RMSE değerlerinin en küçük değer ilkesine göre yapıldı. Kestirim modelinin *doğruluk değeri* (Accuracy value) de karşılaştırma için kullanıldı. Ortalama mutlak hata standart hata olarak seçildi. Ortalama mutlak hata değeri küçük olan teknik en iyi hata kestirimi tekniği seçildi.

4.2.5.1. Doğruluk değeri

Bir sınıflandırma modelinin performans değerlendirmesi, model tarafından doğru veya yanlış olarak kestirimi yapılmış test kayıtlarının sayısına dayanır. Bu sayılar **karmaşıklık matrisi** olarak bilinen bir tablo üzerinde çizelgelendirilirler. Tablo 4.2. bir ikili sınıflandırma problemi için oluşturulan karmaşıklık matrisini göstermektedir. Tablodaki her f_{ij} girişi, j sınıfında olması beklenilip, i sınıfında olan kayıtların sayısını gösterir. Örneğin f_{01}, hatalı bir şekilde 1 sınıfında olduğu kestirilip, gerçekte 0 sınıfında olan kayıtların sayısıdır. Karmaşıklık matrisinin girişlerine dayanarak, model tarafından doğru kestirimi yapılanların sayısı ($f_{11} + f_{00}$) ve yanlış kestirmi yapılanların sayısı ise ($f_{10} + f_{01}$) diyebiliriz (F. Özden Aktaş).

Tablo 4.2. 2-sınıflı problem için karmaşıklık matrisi

		Kestirim sınıfı	
		Sınıf = 1	Sınıf = 0
Gerçek Sınıf	Sınıf = 1	f_{11}	f_{10}
	Sınıf = 0	f_{01}	f_{00}

Karmaşıklık matrisi bir sınıflandırma modelinin performansının ne kadar iyi olduğunu belirlemek için gereken bilgiyi sağlamasına rağmen, bu bilgiyi tek bir sayı ile özetlemek farklı modellerin performanslarını karşılaştırabilmek için daha elverişlidir. Bu, aşağıda da belirtildiği gibi **doğruluk** gibi bir **performans ölçütü** kullanılarak yapılabilir:

$$\text{Doğruluk} = \frac{\text{Doğru kestirimlerin sayısı}}{\text{Toplam kestirim sayısı}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}} \quad (4.1)$$

4.2.5.2. Ortalama Mutlak Hata (Mean Absolute Error)

Ortalama Mutlak Hata, (MAE) tüm test durumlarında kestirimi yapılan ve gerçek değerin arasındaki farkın ortalamasıdır; ortalama kestirim hatasıdır (Challagulla, 2005). MAE'nin hesaplanması için gerekli formül eşitlik (3.1) de gösterildiği gibidir:

$$\frac{|a_1 - c_1| + |a_2 - c_2| + \ldots + |a_n - c_n|}{n} \quad (4.2)$$

Gerçek çıktının *a*, beklenen çıktının ise *c* olduğunu kabul edelim.

4.2.5.3. Kök Hata Kareler Ortalaması (RMSE: Root Mean-Squared Error)

RMSE model veya kestirimci tarafından kestirimi yapılan değerler ve modellenen veya kestirimi yapılandan elde edilen gerçek değerler arasındaki farkın ölçüsü olarak sıklıkla kullanılır (Challagulla, 2005).

Hata Kareleri Ortalamasının (MSE: Mean Square Error) karekökü alınarak hesaplanır:

$$\sqrt{\frac{(a_1-c_1)^2+(a_2-c_2)^2+\ldots+(a_n-c_n)^2}{n}} \tag{4.3}$$

Hata kareleri ortalaması sayısal kestirimler için en çok kullanılan başarı ölçülerinden biridir. Bu değer, her hesaplanan değer ve onun karşılık gelen doğru değer arasındaki farkın karelerinin ortalaması alınarak hesaplanır. RMSE basitçe MSE'nin kareköküdür. RMSE hata değerini gerçek ve kestirilen değerdeki gibi aynı boyutta verir. MAE ve RMSE her bir makine öğrenmesi algoritması için hesaplanır.

- MAE ve RMSE kestirimler kümesindeki hatalardaki varyasyonu tanımlamak için beraber kullanılırlar.
- RMSE her zaman MAE'den büyüktür veya ona eşit olacaktır. Aralarındaki en büyük fark örnekteki özgün hatalardaki varyasyon en büyük olunca olur.
- Eğer RMSE, MAE'ye eşitse, tüm hatalar aynı önemdedir.
- RMSE ve MAE 0'dan ∞'a kadar değer alabilir.
- MAE ve RMSE negatif yönelimlidirler ve düşük değerler daha iyidir. Bu yüzden düşük MAE değerine sahip algoritma en iyi algoritma olarak değerlendirilir.

4.2.5.4. Kappa istatistiği (Cohen's kappa coefficient)

Değerlendiriciler arasındaki anlaşmanın istatistiksel ölçüsüdür.

Kappa = (Gözlenen anlaşma-Tesadüf anlaşma)/(1 – Tesadüf anlaşma)

Tam anlaşma durumunda kappa istatistiği değeri 1 olur. Kappa istatistiği değeri 1'e ne kadar yakınsa anlaşma o derece iyidir. Hiç bir anlaşmanın olmaması durumunda kappa istatistiği değeri sıfıra eşittir veya sıfırdan küçüktür.

4.2.6. Sonuçlar

a) Tablo 4.1.'de makine öğrenmesi algoritmaları için verilen her kategorinin tüm algoritmaları incelenir ve MAE, RMSE, doğruluk ve kappa istatistiği değerleri bulunur.
b) (a) da bulunan değerler karşılaştırılır ve her kategori içinde en iyi algoritma bulunur.
c) (b) de bulunan en iyi algoritmalar karşılaştırılır ve tüm bu sınıflar arasında en iyi performansı veren algoritma bulunur.
d) Mamdani tabanlı bulanık çıkarım sistemi için MAE, RMSE ve doğruluk değerleri bulunur.
e) Uyarlamalı Nöral-bulanık çıkarım sistemi için MAE, RMSE ve doğruluk değerleri bulunur.
f) (c),(d) ve (e) aşamalarında elde edilen sonuçlar karşılaştırılarak en iyi model önerilir.

5. SONUÇLAR VE TARTIŞMA

5.1. Ölçütler

Ölçüt, "yazılım geliştirme sürecine" ve "anlamlı ve vaktinde yönetim bilgisi sağlayan ürünlerine" ölçüme dayalı tekniklerin sürekli uygulamasıdır. Bahsedilen tekniklerin kullanımıyla bu süreç ve ürünler geliştirilir. Yazılım ölçütü tamamen ölçme ile ilişkindir ve ilklenmesinden bakımına kadar yazılım geliştirme yaşam çevriminin tüm evrelerine uygulanabilir. IEEE Standart sözlükçesi ölçütü "sistem, bileşen veya bir niteliğe sahip sürecin nicel miktar derecesi" olarak tanımlar.

Verimli Yazılım Ölçütlerinin Öznitelikleri:

- Basit ve hesaplanabilir
- Deneysel ve sezgisel olarak ikna edici
- Tutarlı sonuçlar sağlar
- Birimlerin ve boyutların kullanımında tutarlıdır
- Programlama dili bağımsızlığı

Ölçütler yazılım kalitesi hakkında bilgi sağlamak ve olası hata ve kötü tasarımın kolayca algılanması için tasarlanmıştır. Statik ve dinamik olabilirler. Statik ölçütler yazılım kod analizi esnasında toplanırken dinamik ölçütler çalışma esnasında toplanır. Yazılım karmaşıklığı ölçütleri yazılım gelişimi yaşam çevriminde nispeten erken elde edildikleri için sıklıkla güvenilirliğin endirekt ölçütü olarak kullanılırlar. Hata içeriyor gibi görünen bileşenleri belirlemek için karmaşıklık ölçütünün kullanılması yazılım mühendislerine onlar üzerinde doğrulama (verification) çabalarına odaklanmalarına, böylece düşük maliyette güvenilir bir ürün kazanmalarına izin veriyor.

Ölçütler 3 ana kategoride gruplandırılır:

- Ürün ölçütleri, gelişim aşamalarını dikkate almaksızın, yazılım ürünün dokümantasyon, tasarım ve performans gibi özelliklerini ölçer. Ürün ölçütleri sınanabilirlik için kod karmaşıklığı (cyclomatic complexity) ve bakım için eşleşme faktörü (coupling factor) gibi dış yazılım özelliklerinin göstergesidir.

- Süreç ölçütleri gelişme zamanı, kullanılan metodoloji ve kalite güvence teknikleri gibi yazılım gelişme sürecini vurgular.
- Kaynak ölçütleri geliştirici yetenek seviyesi, donanım güvenilirliği, yazılım bileşen kalitesi gibi insan, donanım ve yazılım kaynaklarına vurgu yapar.

Ölçütün ana amacı yazılım ürününün kalitesini kestirmektir. Yazılımın kalitesini bakım, hata yoğunluğu (defect density), hata eğilimi, standartlaştırılmış yeniden işleme (normalized rework), anlaşılabilirlik, yeniden kullanılırlık vs. gibi çeşitli nicelikler belirler.

Yazılım mühendisliği uygulamalarında tüm değişimi veya yeniden kullanılabilirlik ölçütlerini kestirebilen güvenilir büyüme modelleri geliştirmek için sinir ağları hâlihazırda kullanılmıştır. Sinir ağları formül veya kurallar üretmek yerine verilen doğru sınıflandırma örnekleri kümesini tekrar üretmek için eğitilmiştir (Eric Rotenberg, 1999). Sinir ağları karmaşık fonksiyonları modellemeye yarayan doğrusal-olmayan gelişmiş modelleme teknikleridir. Sinir ağları teknikleri girdi ve çıktıların tam yapılarının bilinmediği durumlarda kullanılır. Anahtar özellik girdi ve çıktı arasındaki ilişkiyi eğitimle öğrenmeleridir.

İlk adım yazılım sistemlerinin yapısal kod ve tasarım özelliklerinin başka bir ifadeyle yazılım ölçütlerinin bulunmasıdır. Son ölçütler Tablo 5.1.'de gösterilmiştir.

Tablo 5.1. Ölçütlerin Listesi

Sr. No.	**Ölçütlerin Listesi**
1.	LOC_BLANK
2.	BRANCH_COUNT
3.	CALL_PAIRS
4.	LOC_CODE_AND_COMMENT
5.	LOC_COMMENTS
6.	CONDITION_COUNT
7.	CYCLOMATIC_COMPLEXITY

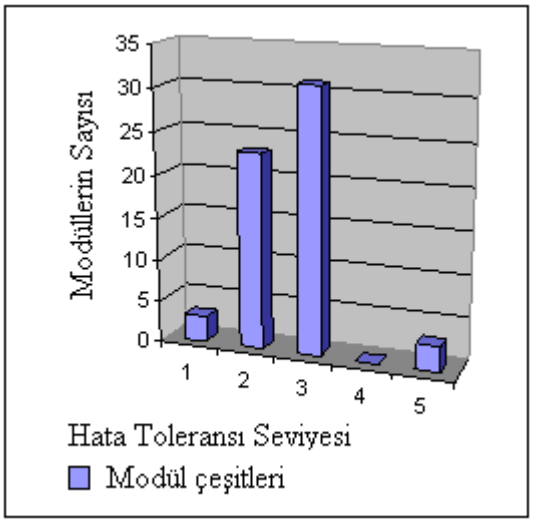
8.	CYCLOMATIC_DENSITY
9.	DECISION_COUNT
10.	DESIGN_COMPLEXITY
11.	EDGE_COUNT
12.	ESSENTIAL_COMPLEXITY
13.	ESSENTIAL_DENSITY
14.	PARAMETER_COUNT
15.	LOC_EXECUTABLE
16.	HALSTEAD_CONTENT
17.	HALSTEAD_DIFFICULTY
18.	HALSTEAD_EFFORT
19.	HALSTEAD_ERROR_EST
20.	HALSTEAD_LENGTH
21.	HALSTEAD_LEVEL
22.	HALSTEAD_PROG_TIME
23.	HALSTEAD_VOLUME
24.	MAINTENANCE_SEVERITY
25.	MODIFIED_CONDITION_COUNT
26.	MULTIPLE_CONDITION_COUNT
27.	NODE_COUNT
28.	NORMALIZED_CYLOMATIC_COMPLEXIY
29.	NUM_OPERANDS
30.	NUM_OPERATORS
31.	NUM_UNIQUE_OPERANDS
32.	NUM_UNIQUE_OPERATORS
33.	NUMBER_OF_LINES
34.	PATHOLOGICAL_COMPLEXITY
35.	PERCENT_COMMENTS
36.	LOC_TOTAL

5.2. Veri Kümesi

Kullanılan gerçek zamanlı hatalı veri kümeleri NASA'nın MDP veri havuzundan alındı. KC4 projesi olarak adlandırılan 125 modülden oluşmuş bu küme, 1,2,3,4 ve 5 olarak etiketlendirilmiş, farklı hata toleransı değerlerine sahip olan 60 modül içerir. Veri kümesindeki modüllerin çeşitleri tablo biçiminde Tablo 5.2.'de, grafiksel biçimde Şekil 5.1.'de gösterilmiştir.

Tablo 5.2. Veri Kümesindeki modüllerin detayları

Aciliyet Derecesi / Etiket	**Adet**
1	3
2	23
3	31
4	0
5	3



Şekil 5.1. Veri Kümesindeki modül detaylarının grafiksel gösterimi

Hata toleransı seviyesi (ya da bakım aciliyeti/önemi) hatanın çevresine etkisini belirtir. 1 ile etiketlenmiş olanlar en acil/önemli hataları gösteriyorken 5. etiket derecesine doğru hataların aciliyeti/önemi azalır. Örneğin aciliyeti 1 ile etiketlenmiş hata, geçici bir çözüm uygulanamamaksızın fonksiyondaki düşüş anlamına gelirken, 5. aciliyet derecesi hatanın yüzeysel bir etkiye sebep olduğu ve sistemde herhangi bir aksamaya sebep olmadığı anlamına gelir. Bundan sonraki bölümlerde 125 modülden oluşan KC4 projesi içindeki bu 60 hatalı modülün, hatalarının aciliyet derecelerinin ne kadar doğru kestirilebildiğinin deneysel sonuçlarını göreceğiz.

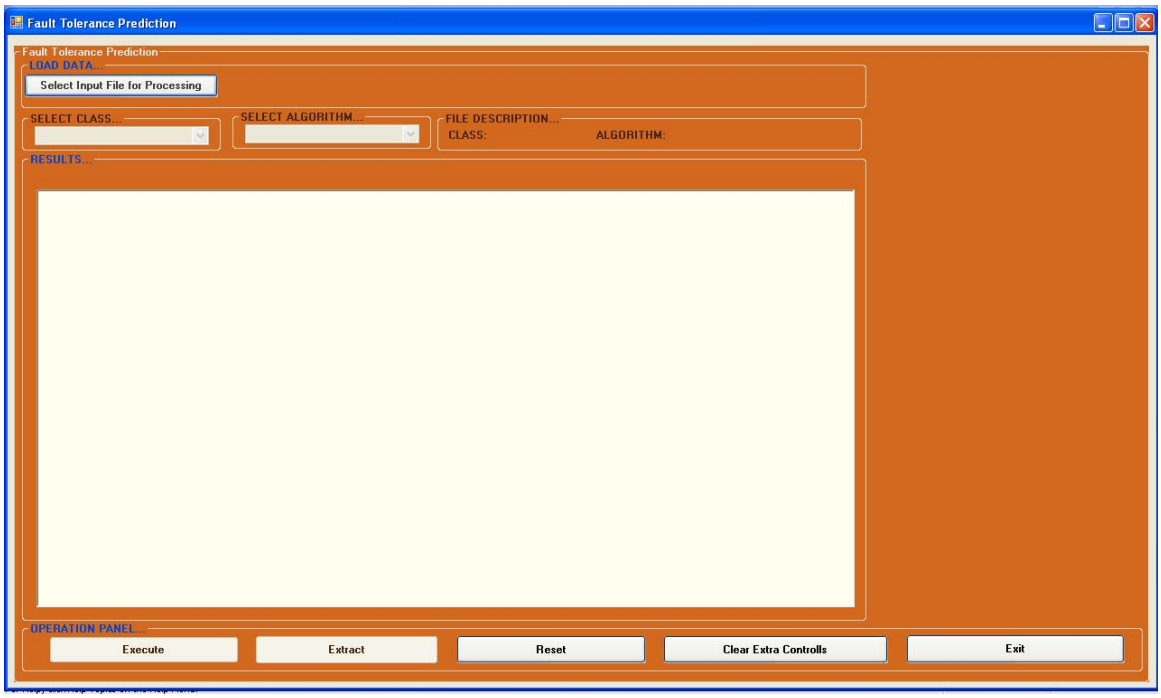
5.3. Makina Öğrenmesi Algoritmalarından Elde Edilen Sonuçlar

Araştırılan algoritmalar WEKA projesinde JAVA sınıfları yapılandırdı. Bunun için çeşitli makine öğrenmesi algoritmaları ve sinir ağları teknikleri analiz edildi. Naïve Bayes Algorithm, Multinomial Naïve Bayes Algorithm, Updateable Naïve Bayes Algorithm, Logistic Regression Modeling, Multilayer Perceptron Algorithm, RBF Network Algorithm, Linear Logistic Algorithm, Sequential Minimal Optimization Algorithm (SMO), Nearest-neibhour Classifier, k- Nearest-neibhour Classifier, k* instance-based Classifier, Locally Weighted Learning Algorithm, Additive Regression Algorithm, Attribute Selected Classifier, Bagging Algorithm, Classification Via Regression, Cross-validation Parameter Selection, Decorate Algorithm, Grading Algorithm, Additive Logistic Regression, MetaCost Algorithm, MultiBoosting Algorithm, Metaclassifier For Multi-Class Datasets, Ordinal Class Classifier, Raced Incremental Logit Boost, Random Committee Algorithm, Stacking Algorithm, StackingC Algorithm, Vote Algorithm, HyperPipe Algorithm, Voting Feature Intervals Algorithm, Decision Stump Algorithm, Pruned C 4.5 Decision Tree Algorithm, Logistic Model Tree Algorithm, Random Forest Algorithm, Random Tree Algorithm and REP Tree Algorithm gibi çeşitli makine öğrenmesi algoritmaları analiz edildi. Model uygulandı ve en iyi algoritma seçildi. Algoritmalar aşağıdaki kriterlere göre analiz edildi.

WEKA yazılımı ortalama mutlak hata, kök hata kareler ortalaması, bağıl mutlak hata (relative absolute error) ve kök bağıl hata kareleri (root relative squared error)

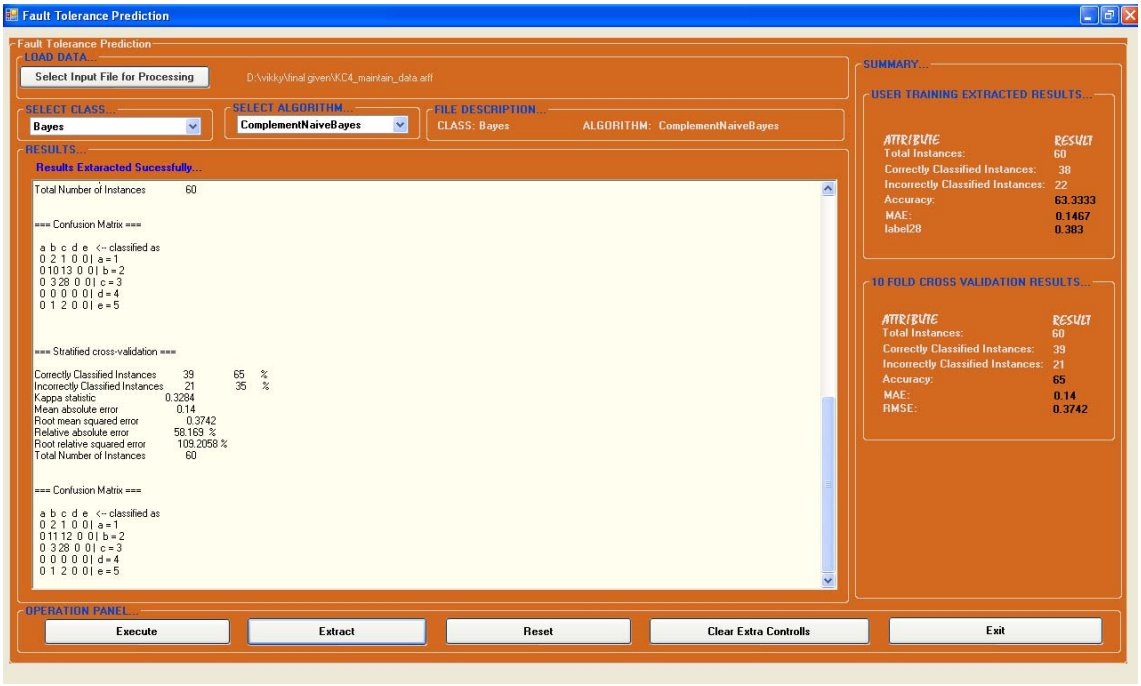
hesaplayabilir. Bununla beraber, en çok bildirilen hata ortalama mutlak hata ve kök hata kareler ortalamasıdır. RMSE verideki uçdeğerlere (outliers) ortalama mutlak hatadan daha duyarlıdır. Uçdeğerlerin etkisini azaltmak için ortalama mutlak hata standart hata olarak seçildi. En düşük ortalama mutlak hata değerine sahip kestirim tekniği en iyi kestirim tekniği olarak seçildi.

C#.NET'te geliştirilen grafiksel kullanıcı arayüzü (GUI) Şekil 5.2.'de gösterilmiştir. MAE ve RMSE makine öğrenmesi algoritmalarının tüm sınıfları için hesaplanmıştır. Visual C#.NET programı grafiksel kullanıcı arayüzü (GUI) yapmak, makine algoritmalarının Java sınıflarından yararlanmak ve Şekil 5.3.'te gösterildiği gibi bunlardan istenilen çıktıyı almak için kullanılmıştır. Arayüz, yüklenebilir.dll dosyası olarak yapılmıştır, bu yüzden her yere yüklenebilir fakat aynı şekilde çalışması için .NET frameworküne ihtiyacı vardır.



Şekil 5.2. Grafiksel Kullanıcı Arayüzünün ekran görüntüsü

Şekil 5.2.'den görülebileceği gibi "Select Class-Sınıf Seç" butonu ile makine öğrenmesinin 6 sınıfından her biri sırayla seçilir. "Select Algorithm-Algortima Seç" butonu ile seçilmiş olan sınıfın içinde bulunan algoritmaların her biri sırayla seçilir. Veri kümesinin (.arff uzantılı) yüklenmesinin ardından yapılan bu seçimlerin sonucu elde edilen sonuçların ekran görüntüsü Şekil 5.3.'teki gibidir:



Şekil 5.3. Elde edilen sonuçların ekran görüntüsü

Tablo 5.3. verikümesi için kestirim tekniklerinin çeşitli Bayes kategorilerinin değerlendirme sonuçlarını gösteriyor. Makine öğrenmesi algoritmalarının Bayes sınıfı için, en iyi algoritma doğruluk, MAE, RMSE ve KAPPA değerleri sırasıyla 65, 0.14, 0.3742, 0.3284 ve 65, 0.1802, 0.3719, 0.3354 olan Complement Naive Bayes ve Naive Bayes Multinomial algoritmalarıdır.

Tablo 5.3. Eğitim Fazı ve Bayes Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları

Algoritma	Eğitim İstatistikleri				10 Katlı Çapraz Onaylama İstatikleri			
	Doğruluk (%)	MAE	RMSE	Kappa	Doğruluk (%)	MAE	RMSE	Kappa
Bayes Net	51.66667	0.2359	0.341	0	51.6667	0.2372	0.3423	0
Complement Naive Bayes	63.33333	0.1467	0.383	0.2934	65	0.14	0.3742	0.3284
Naive Bayes	21.66667	0.2727	0.4641	0.0025	23.3333	0.281	0.4813	0.026
Naive Bayes Multinomial	63.33333	0.1718	0.3658	0.2934	65	0.1802	0.3719	0.3354
Naive Bayes Updateable	21.66667	0.2727	0.4641	0.0025	23.3333	0.281	0.4813	0.026

Tablo 5.4. verikümesi için kestirim tekniklerinin çeşitli Lazy kategorilerinin değerlendirme sonuçlarını gösteriyor. Makine öğrenmesi algoritmalarının Lazy sınıfı için, en iyi algoritma doğruluk, MAE, RMSE ve KAPPA değerleri sırasıyla 55, 0.2209, 0.3558 ve 0.1474 olan LWL'dir.

Tablo 5.4. Eğitim Fazı ve Lazy Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları

Algoritma	Eğitim İstatistikleri				10 Katlı Çapraz Onaylama İstatikleri			
	Doğruluk (%)	MAE	RMSE	Kappa	Doğruluk (%)	MAE	RMSE	Kappa
IB1	98.33333	0.0067	0.0816	0.9717	53.3333	0.1867	0.432	0.1876
IBk	98.33333	0.0286	0.0644	0.9709	53.3333	0.1995	0.4151	0.1876
KStar	96.66667	0.05	0.1234	0.9411	51.6667	0.2105	0.3954	0.1247
LWL	70	0.1858	0.2989	0.4567	55	0.2209	0.3558	0.1474

Tablo 5.5. verikümesi için kestirim tekniklerinin çeşitli Misc kategorilerinin değerlendirme sonuçlarını gösteriyor. Makine öğrenmesi algoritmalarının Misc sınıfı için, en iyi algoritma doğruluk, MAE, RMSE ve KAPPA değerleri sırasıyla 38.3333, 0.2843, 0.3769 ve 0.1232 olan VFI'dir.

Tablo 5.5. Eğitim Fazı ve Misc Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları

Algoritma	Eğitim İstatistikleri				10 Katlı Çapraz Onaylama İstatikleri			
	Doğruluk (%)	MAE	RMSE	Kappa	Doğruluk (%)	MAE	RMSE	Kappa
HyperPipes	33.33333	0.296	0.3822	0.0909	28.3333	0.2971	0.3838	0.0047
VFI	53.33333	0.2948	0.3809	0.3058	38.3333	0.2843	0.3769	0.1232

Tablo 5.6. verikümesi için kestirim tekniklerinin çeşitli Trees kategorilerinin değerlendirme sonuçlarını gösteriyor. Makine öğrenmesi algoritmalarının Trees sınıfı için, en iyi algoritma doğruluk, MAE, RMSE ve KAPPA değerleri sırasıyla 65, 0.2145, 0.3285 ve 0.3269 olan LMT'dir.

Tablo 5.6. Eğitim Fazı ve Trees Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları

Algoritma	Eğitim İstatistikleri				10 Katlı Çapraz Onaylama İstatikleri			
	Doğruluk (%)	MAE	RMSE	Kappa	Doğruluk (%)	MAE	RMSE	Kappa
J48	78.33333	0.1378	0.2625	0.5929	55	0.219	0.3885	0.1649
Decision Stump	65	0.2033	0.3188	0.334	55	0.2229	0.3557	0.1474
LMT	70	0.1965	0.3062	0.4168	65	0.2145	0.3285	0.3269
Random Forest	96.66667	0.0522	0.1148	0.9413	55	0.2058	0.3626	0.1932
Random Tree	98.33333	0.0067	0.0577	0.9709	46.6667	0.2167	0.4637	0.0533
REPTree	51.66667	0.2369	0.3413	0	60	0.2236	0.3512	0.2484
Simple Cart	81.66667	0.1148	0.2396	0.6584	53.3333	0.2168	0.3555	0.1213
BFTree	70	0.1807	0.3006	0.4433	48.3333	0.2277	0.3843	0.0882

Tablo 5.7. verikümesi için kestirim tekniklerinin çeşitli Meta kategorilerinin değerlendirme sonuçlarını gösteriyor. Makine öğrenmesi algoritmalarının Meta sınıfı için, en iyi algoritma doğruluk, MAE, RMSE ve KAPPA değerleri sırasıyla 63.3333, 0.216, 0.3358, 0.3053 olan Bagging'dir.

Tablo 5.7. Eğitim Fazı ve Meta Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları

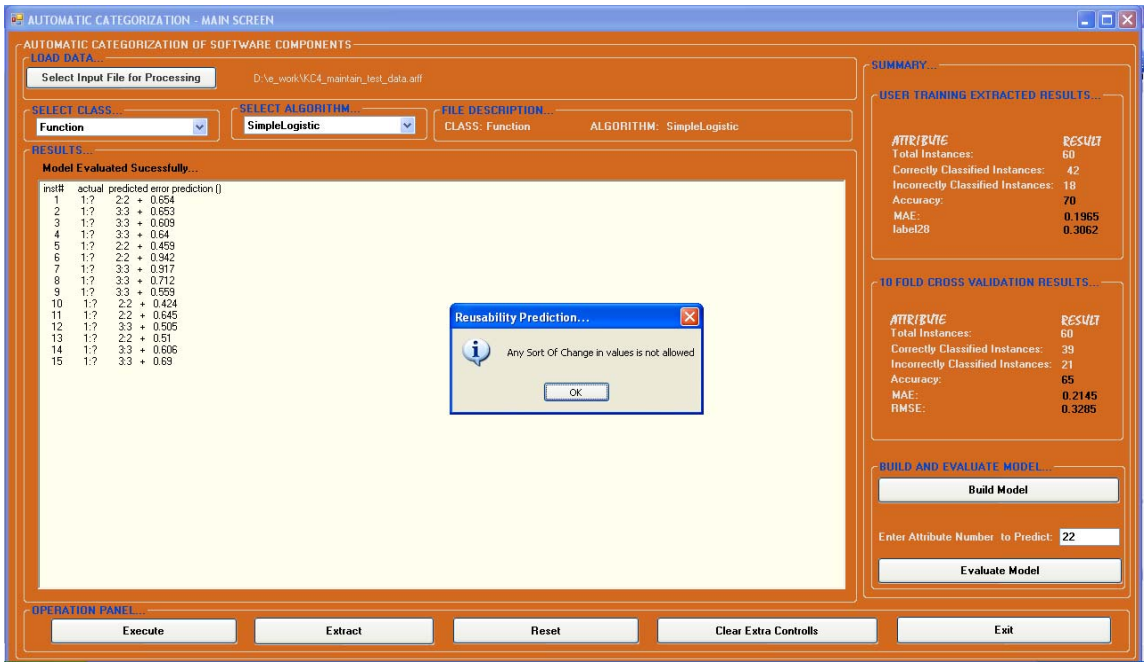
Algoritma	**Eğitim İstatistikleri**				**10 Katlı Çapraz Onaylama İstatikleri**			
	Doğruluk (%)	**MAE**	**RMSE**	**Kappa**	**Doğruluk (%)**	**MAE**	**RMSE**	**Kappa**
AdaBoostM1	65	0.203	0.3188	0.334	55	0.252	0.3583	0.1474
Attribute Selected Classifier	51.66667	0.232	0.3409	0	48.3333	0.239	0.3559	0.0544
Bagging	68.33333	0.195	0.2952	0.4025	63.3333	0.216	0.3358	0.3053
Classifivation ViaRegression	65	0.211	0.3157	0.3197	58.3333	0.225	0.3396	0.183
Filtered Classifier	51.66667	0.232	0.3409	0	51.6667	0.233	0.3423	0
Grading	51.66667	0.193	0.4397	0	51.6667	0.193	0.4397	0
LogitBoost	86.66667	0.129	0.2248	0.7624	55	0.212	0.3629	0.1752
MultiBoostAB	65	0.203	0.3188	0.334	55	0.252	0.3583	0.1474
Multi Class Classifier	68.33333	0.301	0.3779	0.4025	63.3333	0.304	0.3815	0.3279
Multi Scheme	51.66667	0.239	0.3413	0	51.6667	0.240	0.3426	0
Ordinal Class Classifier	73.33333	0.164	0.2905	0.5072	55	0.221	0.3635	0.1253
Raced Incremental Logit Boost	51.66667	0.239	0.3413	0	51.6667	0.240	0.3426	0
Random Committee	98.33333	0.006	0.0577	0.9709	50	0.188	0.3822	0.1228
Stacking	51.66667	0.232	0.3413	0	51.6667	0.240	0.3426	0
StackingC	51.66667	0.236	0.3411	0	51.6667	0.237	0.3425	0
Vote	51.66667	0.239	0.3413	0	51.6667	0.240	0.3426	0

Tablo 5.8. verikümesi için kestirim tekniklerinin çeşitli Function kategorilerinin değerlendirme sonuçlarını gösteriyor. Makine öğrenmesi algoritmalarının Function sınıfı için, en iyi algoritma doğruluk, MAE, RMSE ve KAPPA değerleri sırasıyla 60, 0.1934, 0.3369, 0.2771 ve 60, 0.2138, 0.3307, 0.2453 olan Logistic ve Multilayer Perceptron'dır.

Tablo 5.8. Eğitim Fazı ve Function Tekniklerinin 10 Katlı Çapraz Onaylama Sonuçları

Algoritma	**Eğitim İstatistikleri**				**10 Katlı Çapraz Onaylama İstatikleri**			
	Doğruluk (%)	**MAE**	**RMSE**	**Kappa**	**Doğruluk (%)**	**MAE**	**RMSE**	**Kappa**
Decorate	81.66667	0.1547	0.2556	0.6633	50	0.2264	0.3756	0.0741
Dagging	56.66667	0.2265	0.3373	0.1156	55	0.2383	0.3419	0.1438
END	68.33333	0.1928	0.3036	0.4289	56.6667	0.2118	0.3421	0.1925
ClassBalanced-ND	73.33333	0.1697	0.2902	0.5072	56.6667	0.2111	0.3558	0.1649
DataNear-BalancedND	73.33333	0.1697	0.2902	0.5072	51.6667	0.2239	0.3711	0.0938
Nested-Dichotomies	68.33333	0.1852	0.3031	0.4289	60	0.2151	0.3561	0.2547
Logistic	70	0.1618	0.2842	0.4363	60	0.1934	0.3369	0.2771
RBF Network	65	0.1747	0.2965	0.3532	51.6667	0.218	0.3717	0.1487
Simple Logistic	70	0.1965	0.3062	0.4168	65	0.2145	0.3285	0.3269
SMO	55	0.2633	0.3512	0.0774	53.3333	0.2667	0.3559	0.0476
Multilayer Perceptron	71.66667	0.1959	0.2997	0.4516	60	0.2138	0.3307	0.2453

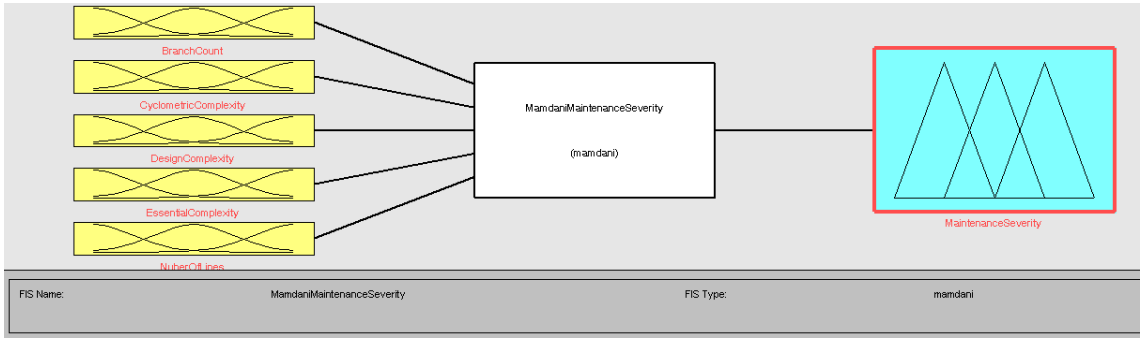
Yukarıdaki tabloda ND Nested-Dichotomies anlamına gelmektedir. Tüm algoritmaların performanslarını analiz ederken, LMT ve Simple Logistic algoritmaları karşılaştırılmalı çalışmada kullanılan diğer algoritmalardan daha üstün performans gösterdiler.



Şekil 5.4. Otomatik sınıflandırma

5.4. Mamdani Tabanlı Bulanık Çıkarım Sistemi Sonuçları

Mamdani tabanlı bulanık çıkarım sisteminin tasarlanmasında diğer ölçütlerin çoğunun değerleri sıfır olduğu için veya aynı özelliğe işaret ettikleri için girdi özniteliği olarak sadece 5 ölçüt ele alınır ve yazılım bakımı aciliyet düzeyi (severity level) Şekil 5.5.'de gösterildiği gibi çıktı özelliği olarak kullanılır.

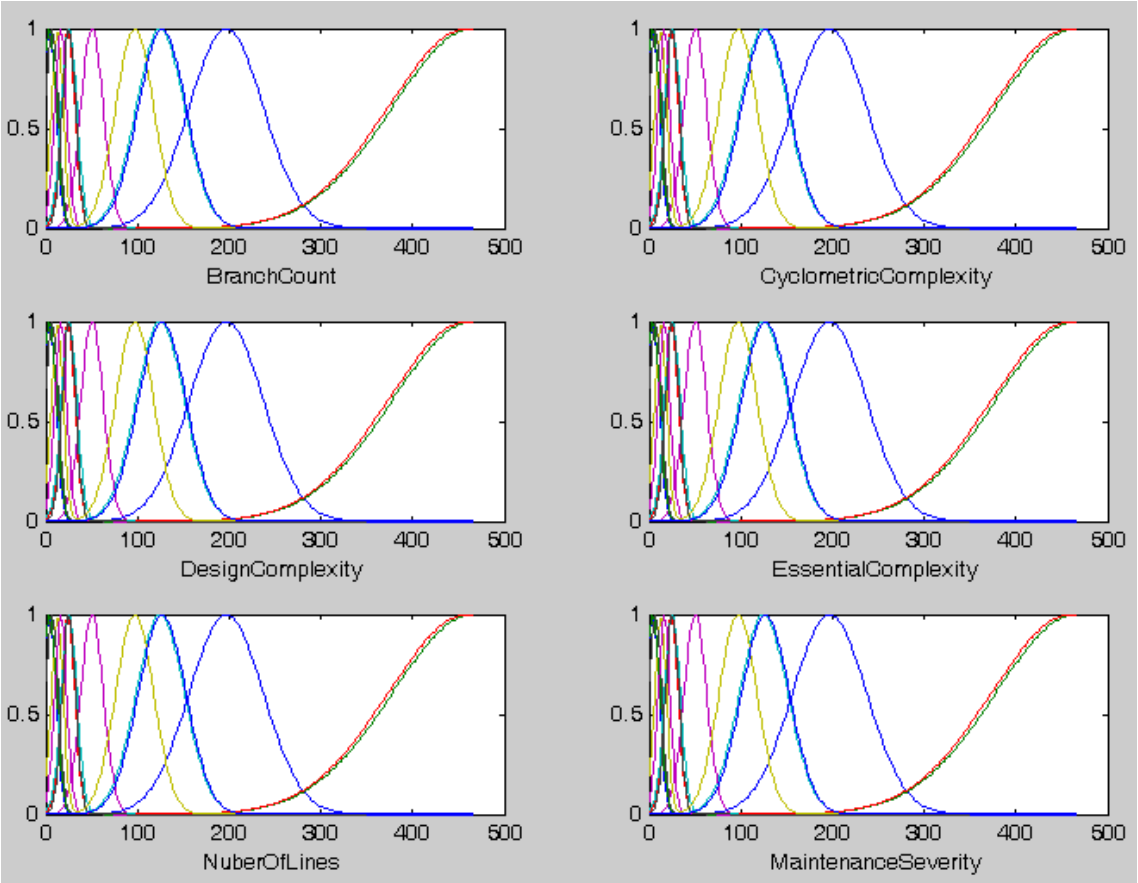


Şekil 5.5. Mamdani Tabanlı Bulanık Çıkarım Sistemi

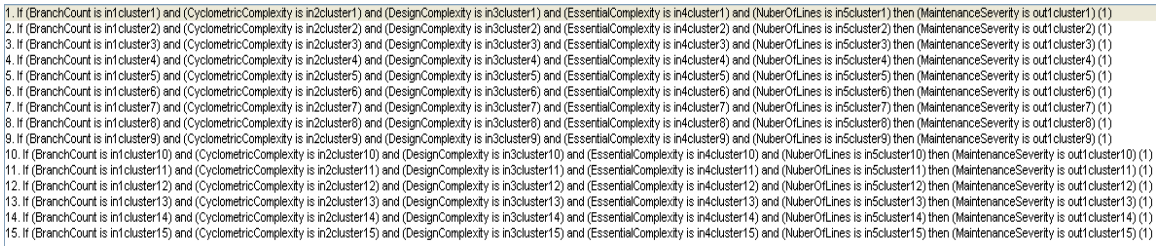
Ele alınan beş girdi özelliği aşağıdaki gibidir:

- Branch_Count (Dallanma_Sayısı)
- Cyclometric_Complexity (Karmaşıklık Katsayısı)
- Design_Complexity (Tasarım_Karmaşıklığı)
- Essential_Complexity (Elzem_Karmaşıklık)
- Number_Of_Lines (Hat_Sayısı)

Her girdi ve çıktı öznitelikleri onbeş bulanık set ile gösterilmiştir ve her özelliğin üyelik fonksiyonu değerleri Şekil 5.6.'da gösterilmiştir. Şekil 5.7.'de Mamdani tabanlı bulanık çıkarım sistemi için 15 kural kullanılmıştır.



Şekil 5.6. Girdi ve çıktı özniteliklerinin üyelik fonksiyonu



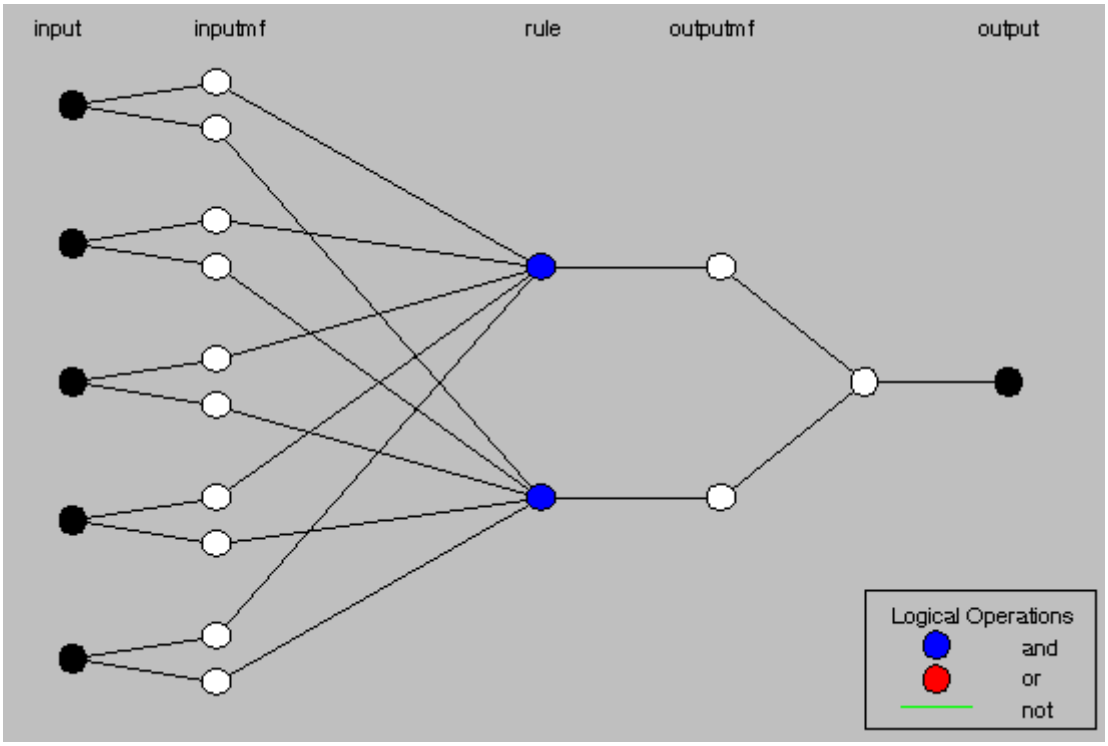
Şekil 5.7. Mamdani tabanlı çıkarım sisteminin 15 kuralı

Mamdani tabanlı bulanık çıkarım sisteminin test evresinde 15 girdi kullanıldı ve MAE için 0.2183, RMSE için 0.3066 ve doğruluk için %80 değerleri elde edildi.

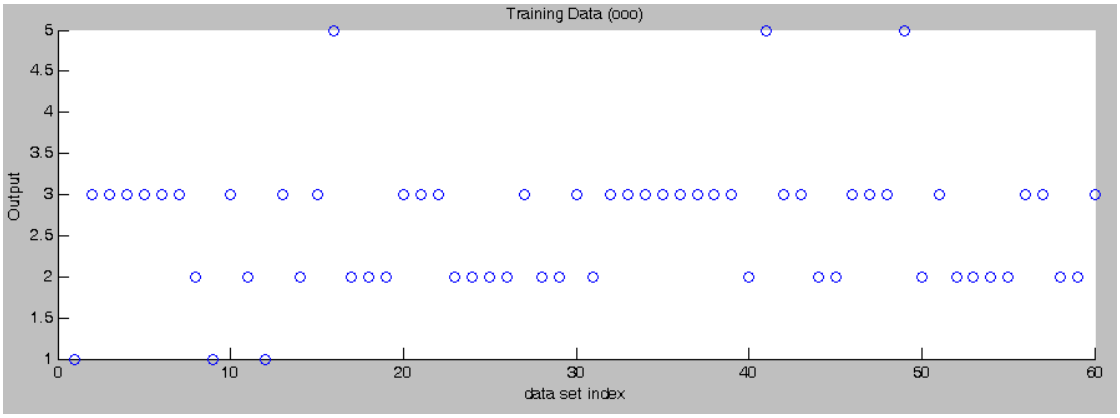
5.5. Uyarlamalı Nöral-Bulanık Tabanlı Çıkarım Sistemi Sonuçları

Uyarlamalı (adaptive) nöral-bulanık çıkarım sistemi hibrit nöral-bulanık sistemler arasında en iyi olarak bulunmuştur [56] ve max-min bileşimli (composition) Mamdani tabanlı uyarlamalı nöral-bulanık çıkarım sisteminin yapısında ve hesaplanmasındaki ekstra karmaşıklık daha iyi öğrenme yeteneği veya yaklaştırma gücü (approximation power) anlamına gelmek zorunda değildir [57]. Bu nedenle MATLAB 7.4, Sugeno tabanlı Uyarlamalı Nöral-bulanık çıkarım sistemi yazılım bakımı öneminin modellenmesinde kullanılır. Yazılım bileşenlerinin değerlendirilmesi için ideal çıkarım sisteminde karmaşıklık daha az, kesinlik daha yüksek olmalıdır. Eğitilmiş olan çıkarım sistemi, ölçüt değerlerini daha önceki aşamalardan alacaktır ve yazılım bileşenleri veya modüllerinin yazılım bakımı aciliyet değerinin kestirimini yapacaktır. Uyarlamalı nöral-bulanık çıkarım sisteminin yapısıyla ile ilgili bilgi aşağıdaki gibidir (ayrıca bakınız: Şekil 5.8.):

Düğümlerin sayısı: 32
Lineer parametrelerin sayısı: 12
Non-Lineer parametrelerin sayısı: 20
Toplam parametrelerin sayısı: 32
Eğitim veri eşlerinin sayısı: 60
Kontrol veri eşlerinin sayısı: 0
Bulanık kuralların sayısı: 2



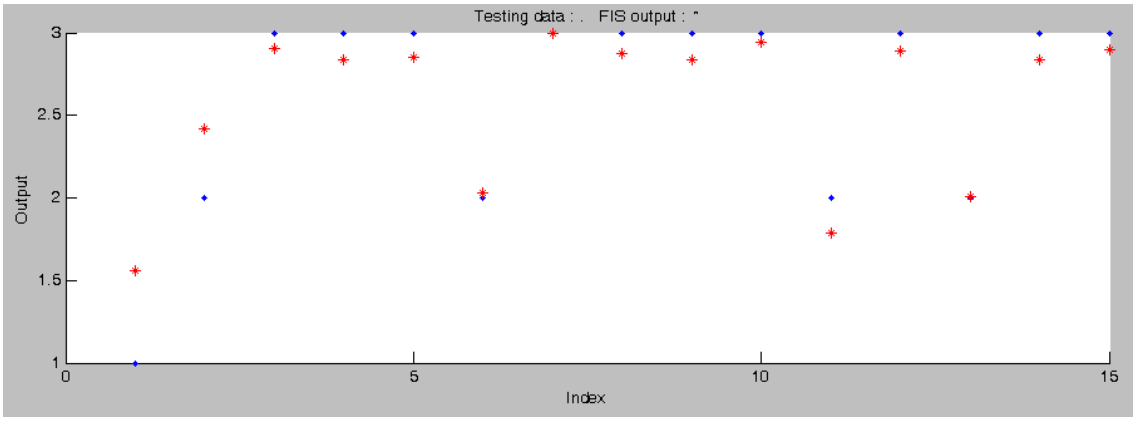
Şekil 5.8. Uyarlamalı nöral-bulanık çıkarım sisteminin yapısı



Şekil 5.9. Uyarlamalı nöral-bulanık sistemin eğitim verisi

Uyarlamalı nöral-bulanık (NB) sistem için girdi örneklerinin grafiksel sunumu Şekil 5.9.'da gösterilmiştir. Uyarlamalı nöral-bulanık sistem hibrit öğrenme algoritmaları, en küçük kareler yöntemi ve geri yayılım kullanılarak eğitilmiştir. İleri geçişte, ardıl (consequent) parametreler en küçük kareler yöntemi kullanılarak tanımlanmıştır ve geri geçişte öncül (premise) parametreler geri yayılım kullanılarak tanımlanmıştır. Eğitilen NB sistem daha sonra 15 girdi için test edildi ve MAE için

0.1571, RMSE için 0.2140 ve doğruluk için %93.3333 değerleri elde edildi. Farklı girdiler için beklenen değerlerin ve NB sisteminin çıktılarının çizimi Şekil 5.10.'da gösterilmiştir.



Şekil 5.10. Test verisinin çizimi V/S FIS Çıktısı

6. SONUÇ VE GELECEK KAPSAM

Müşteri kurulumlarında (installation) yürütülen sistemler için yazılım hata toleransı başarısızlıklara karşı son savunma hattını sunar. Hata toleransı, donanım fazlalığı veya yazılım istisna işlemesiyle (exception handling) ile eşanlamlı hale gelmiştir. Yazılım modüllerindeki yazılım bakımı aciliyetinin kestirimi, geliştirilmiş zamanlama ve proje kontrolüyle, yazılım kalitesi mühendisliğini destekler. Bu, yazılım testini yönetmek ve tüm sürecin etkinliğini iyileştirmek için anahtar adımdır. Hata kestirimi yazılım süreci kontrolünü geliştirmek ve yüksek yazılım güvenilirliğini başarmak için kullanılır. Makine öğrenmesi algoritmalarının bütün sınıflarının karşılaştırılması üzerine Lojistik model ağaçları ve Basit lojistik algortimaların diğer makine öğrenmesi algoritmalarına göre hata toleransı kestiriminde %65 doğrulukla daha iyi teknikler oldukları gözlenmektedir. WEKA projesinin her iki algoritmasında, sınıflandırma algoritması yani lojistik sınıflandırıcı aynıdır. Her iki algoritmada en düşük *ortalama mutlak hata (*Mean Absolute Error, MAE: 0.2145) ve *kök hata kareler ortalaması* (Root Mean Square Error, RMSE: 0.3285) değerlerine sahiptir. Her iki algoritmanın gösterdiği *Kappa istatistiği* 0.3269 sonucu ile oldukça gelecek vadediyor.

Test verisi için Mamdani tabanlı bulanık çıkarım sisteminin sonuçları, Lojistik model ağaçları ve Basit lojistik algortimaların MAE=0.2183, RMSE=0.3066, doğruluk=80 değerleriyle karşılaştırmalı olarak eşittir.

Uyarlamalı nöral-bulanık tabanlı modelleme tekniği test verisine dayalı diğer tekniklere göre MAE=0.1571, RMSE=0.2140, doğruluk=93.3333 değerleriyle üstün performans göstermiştir.

Bununla beraber, model uygulanmıştır ve yazılım bileşenlerinin hata etkisinin farklı önem/aciliyet seviyelerine sınıflandırılmasında en iyi algoritmanın uyarlamalı nöral bulanık tabanlı teknik olduğu bulunmuştur. Algortima, hatalardan ağır etkilenen ve hataları ayıklanabilinen (debugged) modülleri belirlemek için model geliştirme amacıyla kullanılabilir.

Bu çalışma gelecekte aşağıdaki yönlerde genişletilebilir:

- Bu çalışma diğer programlama dillerine uygulanabilir.
- Daha fazla algoritma değerlendirilip, en iyi olanı bulunabilir.
- İleri araştırma yapılabilir ve hata toleransında özniteliklerin etkisi bulunabilir.
- Özniteliklerin ilişkileri ve hata toleransını eşleştirmek (mapping) için yazılım kalitesinin diğer boyutları ele alınabilir.

KAYNAKLAR

Abraham, Ajith, "Neuro-Fuzzy Systems: State-of-the-Art Modeling Techniques, Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence", Mira Jose, Prieto Alberto ed., *Lecture Notes in Computer Science,* vol. 2084. Germany: Springer-Verlag; 2001, sayfa. 269-276.

Abraham A., "*Hybrid Intelligent Systems: Evolving Intelligence in Hierarchical Layers*", Studies in Fuzziness and Soft Computing, vol. 173, 2005, sayfa 159–179.

Aha, D. ve Kibler, D. (1991), "*Instance-based learning algorithms*", Machine Learning, vol. 6, Issue no. 1, January 1991, sayfa 37-66.

Atkeson, C., Moore, A. ve Schaal, S. (1996). "*Locally weighted learning*", AI Review, vol. 11, Feb 1996, sayfa 11-73.

Bellini, P. (2005), "*Comparing Fault-Proneness Estimation Models*", 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), vol. 0, 2005, sayfa 205-214.

Breiman L. (2001), "*Random Forests*", Machine Learning, vol. 45, Issue no. 1, 2001, sayfa 5-32.

Breiman, L. (1996), "*Bagging predictors*". Technical Report, 1996.

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984), "*Classification and Regression Trees*", Wadsworth International Group, Belmont, California, 1984.

Buntine, W.L. (1996), "*Operations for Learning with Graphical Models*", Journal of Artificial Intelligence Research, vol. 2, 1994, sayfa 159-225.

Caruana, R., Niculescu, A., Crew, G., ve Ksikes, A. (2004), "*Ensemble Selection from Libraries of Models*", Proceedings International Conference on Machine Learning (ICML'04), 2004.

Challagulla, V.U.B. , Bastani, F.B., I-Ling Yen, Paul, (2005) "*Empirical assessment of machine learning based software defect prediction techniques",* 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS 2005, 2-4 Feb 2005, sayfa 263-270.

Cleary, J.G. ve Trigg, L.E. (1995), "*K*: An Instance-based Learner Using an Entropic Distance Measure*", Proceedings Tweleveth International Conference on Machine Learning, 1995, sayfa 108-114.

Demiroz, G., ve Guvenir, A. (1997), "*Classification by voting feature intervals*", Proceedings Ninth European Conference on Machine Learning, 1997, sayfa 85-92.

Dong, L., Frank, E. ve Kramer, S. (2005), "*Ensembles of Balanced Nested Dichotomies for Multi-class Problems*", Jorge, A., et al. (eds), Proceedings Ninth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2005), Springer-Verlag, Berlin, 2005, sayfa 84-95.

E. H. Mamdani, Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis, *IEEE Trans. Computers*, 26(12):1182-1191, 1977.

Eisenstein, J. ve Davis, R. **(**2006), "*Visual and Linguistic l* Techniques ACM, Vol. 5, sayfa 130-150.

Eric Rotenberg (1999), "*AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors*", Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, June 15-18, sayfa 84-90.

Evren Ceylan, F. Onur Kutlubay, Ayse B. Bener (2006), "*Software Defect Identification Using Machine Learning Techniques*" Proceeding of 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06), Bogaziçi University, Turkey, sayfa 240-247.

F. Esragh ve E.H. Mamdani, *A general approach to linguistic approximation, Fuzzy Reasoning and its Applications*, Academic Press,1981.

Fenton, N. E. ve Neil, M. (1999), "*A Critique of Software Defect Prediction Models*", Bellini, I. Bruno, P. Nesi, D. Rogai, University of Florence, *IEEE Trans. Softw. Engineering,* vol. 25, Issue no. 5, sayfa 675-689.

Frank, E. ve Witten, I. H. (1998), "*Generating Accurate Rule Sets without Global Optimization*", Proceedings of fifteenth International Conference on Machine Learning, Morgan Kaufmann Publishers, 1998, sayfa 144-151.

Frank, E. ve Hall, M. (2001), "*A Simple Approach to Ordinal Classification*", Proceedings of Twelfth European Conference on Machine Learning, 2001, sayfa 145-156.

Frank, E. ve Kramer S. (2004), "*Ensembles of nested dichotomies for multi-class problems*", Proceedings Twenty-first International Conference on Machine Learning, 2004.

Frank, E., Hall, M. ve Pfahringer, B. (2003), "*Locally Weighted Naive Bayes*", Proceedings Nineteenth Conference in Uncertainty in Artificial Intelligence, 2003, sayfa 249-256.

Frank, E., Wang, Y., Inglis, S., Holmes, G. ve Witten, I.H. (1998), "*Using model trees for classification*", Machine Learning, vol. 32, Issue no.1, 1996, sayfa 63-76.

Freund, Y. ve Schapire, R.E. (1996), "*Experiments with a new boosting algorithm*", Proceedings Thirteenth International Conference on Machine Learning, San Francisco, 1996, sayfa 148-156.

Friedman, J., Hastie, T. ve Tibshirani, R. (1998), "*Additive Logistic Regression: a Statistical View of Boosting*", Stanford University, 1998.

Giovanni Denaro (2000), "*Estimating Software Fault-Proneness for Tuning Testing Activities*" Proceedings of the 22nd International Conference on Software Engineering (ICSE2000), Limerick, Ireland, June 2000.

Hastie, T., ve Tibshirani, R. (1998), "*Classification by Pairwise Coupling*", Advances in Neural Information Processing Systems, Annals of Statistics, vol. 26, Issue no. 2, 1998, sayfa 451-471.

Ho, T.K. (1998), "*The Random Subspace Method for Constructing Decision Forests*", Proceedings of IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, Issue no. 8, sayfa 832-844.

Jang, J.-S. R., Sun, C.-T. ve Mizutani, E., Neuro-Fuzzy and Soft Computing- A Computational Approach to Learning and Machine Intelligence, Pearson Education (Singapore) Pvt. Ltd., 1st Edition, 2004.

John, G.H. ve Langley, P. (1995), "*Estimating Continuous Distributions in Bayesian Classifiers*", Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo, 1995, sayfa 338-345.

J-S. R. Jang ve C.T. Sun, "Neuro-fuzzy Modeling and Control," Proceeding of the IEEE, March 1995.

Kaariainen, M. ve Malinen, T. (2004), " *Selective Rademacher Penalization and Reduced Error Pruning of Decision Trees*", Journal of Machine Learning, Vol. 5, sayfa 1107-1126.

Kanmani, S., Sankaranarayanan, V., Thambidurai, P. ve Rhymend Uthariaraj, V. (2004), "*Object oriented software quality prediction using general regression neural networks*", Acm Sigsoft Software Engineering Notes, vol. 29, Issue 5, Sep 2004, sayfa 1-6.

Keerthi, S.S., Shevade, S.K., Bhattacharyya, C. ve Murthy, K.R.K. (2001), "*Improvements to Platt's SMO Algorithm for SVM Classifier Design*", Neural Computation, vol. 13, Issue no. 3, 2001, sayfa 637-649.

Kohavi, R. (1996), "*Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid*", Proceedings of Second International Conference on Knowledge Discovery and Data Mining, 1996, sayfa 202-207.

Kohavi, R. (1995), "*The Power of Decision Tables*", Proceedings of Eighth European Conference on Machine Learning, vol. 912, 1995, sayfa 174-189.

Landwehr, N., Hall, M. ve Frank, E. (2005), "*Logistic Model Trees*", Kluwer Academic Publishers, Hingham, MA, United States of America, vol. 59, Issue no.1-2, May 2005, sayfa 161-205.

Lanubile F., Lonigro A., ve Visaggio G. (1995) "*Comparing Models for Identifying Fault-Prone Software Components*", Proceedings of Seventh International Conference on Software Engineering and Knowledge Engineering, June 1995, sayfa 12-19.

Ma, Y., ve Guo, L. (2006), "A Statistical Framework for the Prediction of Fault-Proneness", West Virginia University, Morgantown.

Mahaweerawat, A. (2004), "*Fault-Prediction in object oriented software's using neural network techniques*", Advanced Virtual and Intelligent Computing Center (AVIC), Department of Mathematics, Faculty of Science, Chulalongkorn University, Bangkok, Thailand, sayfa 1-8.

Manasi Deodhar (2002), "*Prediction Model and the Size Factor for Fault-proneness of Object Oriented Systems*", Ott.

Mamdani, E.H. ve S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, Vol. 7, No. 1, sayfa 1-13, 1975.

Mccallum, A. ve Nigam, K. (1998), "*A Comparison of Event Models for Naive Bayes Text Classification*", AAAI/ICML-98 Workshop on Learning for Text Categorization, Technical Report WS-98-05, AAAI Press, 1998, sayfa 41-48.

Melville, P. ve Mooney, R. J. (2003), "*Constructing Diverse Classifier Ensembles Using Artificial Training Examples*", Proceedings Eighteenth International Joint Conference on Artificial Intelligence, 2003, sayfa 505-510.

Mooney, R. (1996), "*Comparative experiments on disambiguating word senses: An illustration of the role of bias in machine learning*", Proceedings of Empirical Methods in Natural Language Processing (EMNLP), 1996, sayfa 82-91.

Parvinder Singh Sandhu, Sunil Kumar ve Hardeep Singh, "Intelligence System for Software Maintenance Severity Prediction", Journal of Computer Science, Vol. 3 (5), sayfa 281-288, 2007

Pelikan M., Goldberg, D. E. ve Cantu-Paz, Erick (1999), "*BOA: The Bayesian Optimization Algorithm*", Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, 1999, sayfa 525-532.

Platt J. (1998), "*Fast Training of Support Vector Machines using Sequential Minimal Optimization*", Advances in Kernel Methods - Support Vector Learning, Schölkopf, B., Burges, C., and Smola, A., eds., MIT Press, 1998.

Quinlan R. (1993), "*C4.5: Programs for Machine Learning*", Morgan Kaufmann Publishers, San Mateo, CA, 1993, sayfa 302.

Quinlan, J.R. (1986), "*Induction of Decision Trees*", Machine Learning, vol. 1, 1996, sayfa 81-106.

Rennie, J.D.M., Shih, L. (2003), "*Tackling the Poor Assumptions of Naive Bayes Text Classifiers*", Proceedings of Twentieth International Conference on Machine Learning (ICML), 2003, sayfa 616-623.

Runeson, Claes Wohlin ve Magnus C. Ohlsson (2001), "*A Proposal for Comparison of Models for Identification of Fault-Proneness*", Dept. of Communication Systems, Lund University, Profes 2001, LNLS 2188, sayfa 341-355.

Saida Benlarbi, Khaled El Emam, Nishith Geol (1999), "*Issues in Validating Object-Oriented Metrics for Early Risk Prediction*", by Cistel Technology 210 Colonnade Road Suite 204 Nepean, Ontario Canada K2E 7L5.

Shi, H. (2007), "*Best-first decision tree learning"*, University of Waikato, Hamilton, Newzland.

Sumner, M., Frank, E. ve Hall, M. (2005): "*Speeding up Logistic Model Tree Induction*". Proceedings of Ninth European Conference on Principles and Practice of Knowledge Discovery in Databases, 2005, sayfa 675-683.

T. Takagi ve M. Sugeno, Derivation of fuzzy control rules from human operators control actions, *Proc. IFAC Symp. on Fuzzy lnformation, Knowledge Representation and Decision Analysis*, 55–60, July 1983.

T. Takagi ve M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. Syst., Man, Cybern*, 15:116–132, 1985.

Ted Pedersen, A simple approach to building ensembles of Naive Bayesian classifiers for word sense disambiguation, , Proceedings of the first conference on North American chapter of the Association for Computational Linguistics, 2000, Seattle, Washington, sayfa 63 - 69

Ting, K. M. ve Witten, I. H. (1997), "*Stacking Bagged and Dagged Models*", Proceedings of Fourteenth international Conference on Machine Learning, San Francisco, CA, 1997, sayfa 367-375.

Topuz, V., Akbaş, A., Tektaş, M.: "Boğaz Köprüsü Yolunda Katılım Noktalarında Trafik Akımlarının Bulanık Mantık Yaklaşımı ile Kontrolü ve Bir Uygulama Örneği", Uluslararası Trafik ve Yol Güvenliği Kongresi ve Fuarı , Ankara, Ağustos -2002.

Quah, T. ve Thwin, M. (2003), "*Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics*" Proceedings of the International Conference on Software Maintenance, sayfa 116.

Quah, T. ve Thwin, M. (2002), "*Application of Neural Networks for Predicting Software Development Faults Using Object-Oriented Metrics*" Proceedings of Ninth International Conference on Neural Information Processing (ICONIP'02), vol. 5, sayfa 2312-2316.

Veri Madenciliği Dönem Projesi, F. Özden Aktaş.

Webb, G., Boughton, J. ve Wang, Z. (2002), "*Averaged One-Dependence Estimators: Preliminary Results*", In Simoff, S.J. ve Williams, G.J. ve Hegland, M. (eds.), Proceedings of the First Australasian Data Mining Workshop (AusDM02), Canberra, Australia, Sydney, University of Technology, 2002, sayfa 65-73.

Webb, G.I. (2000), "*MultiBoosting: A Technique for Combining Boosting and Wagging*", Machine Learning, vol. 40, Issue no. 2, 2000, sayfa 159-196.

WEKA (2007), www.cs.waikato.ac.nz/~ml/weka/.

What are Neuro-Fuzzy Systems?
http://fuzzy.cs.uni-magdeburg.de/nfdef.html

Zadeh, L.A., "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 3, No. 1, sayfa 28-44, Jan. 1973.

ÖZGEÇMİŞ

Ebru ARDIL, 5 Kasım 1982 yılında Edirne ili Uzunköprü ilçesinde doğdu. İlköğretimini Edirne Kurtuluş İlkokulu, Ortaöğretimini Edirne Anadolu Lisesi ve Lise öğretimini Edirne Süleyman Demirel Fen Lisesinde tamamladıktan sonra 2000 yılında Ondokuz Mayıs Üniversitesi, Mühendislik Fakültesi, Elektrik-Elektronik Mühendisliği bölümünü kazandı. Bu bölümden 2005 yılında mezun oldu. Aynı yılın Eylül ayında Yüksek Lisans çalışmalarına başladı.