T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

WEB UYGULAMALARINDA ALANA ÖZGÜ METRİKLER İLE HATAYA YATKINLIK TAHMİNİ

MEHMET SERDAR BİÇER

DOKTORA TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

DANIŞMAN
PROF. DR. BANU DİRİ

İSTANBUL, 2018

T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

WEB UYGULAMALARINDA ALANA ÖZGÜ METRİKLER İLE HATAYA YATKINLIK TAHMİNİ

Mehmet Serdar BİÇER tarafından hazırlanan tez çalışması 28.12.2017 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **DOKTORA TEZİ** olarak kabul edilmiştir.

Tez Danışmanı
Prof. Dr. Banu DİRİ
Yıldız Teknik Üniversitesi

Jüri Üyeleri

Prof. Dr. Banu DİRİ
Yıldız Teknik Üniversitesi _____

Prof. Dr. Oya KALIPSIZ
Yıldız Teknik Üniversitesi _____

Doç. Dr. Hasan SÖZER
Özyeğin Üniversitesi _____

Doç. Dr. Özgür Koray ŞAHİNGÖZ
İstanbul Kültür Üniversitesi _____

Doç. Dr. Mehmet Sıddık AKTAŞ
Yıldız Teknik Üniversitesi _____

ÖNSÖZ

Doktora tez danışmanım olmayı kabul eden ve tez sürecinde desteklerini ve yardımlarını esirgemeyen, tezin araştırma ve deneysel çalışmaları boyunca bana yol gösterici olan değerli hocam Prof. Dr. Banu DİRİ'ye sonsuz teşekkürlerimi sunarım. O olmadan bu çalışmayı bitiremezdim.

Ayrıca, çalışmam boyunca gösterdikleri destek için sevgili aileme,

Tez çalışmam konusunda beni motive eden, iyi ve kötü günümde yanımda olan, bundan sonra da olmaya devam edeceğini bildiğim sevgili nişanlıma teşekkür ediyorum.

Bu çalışma, yakın bir zaman önce aramızdan ayrılmış olan ağabeyim Doç. Dr. Suat BİÇER'e ithaf edilmiştir.

Ocak, 2018

Mehmet Serdar BİÇER

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

cr	Çaprazlama
g	Jenerasyon
k	Komşuluk sayısı
mc	Hafıza hücresi
M_S	S alt kümesinin faydası
NEC	Normalize beklenen masraf
pd	Hata yakalama oranı
pf	Yanlış alarm oranı
PC	Masraf olasılığı
$r_{\overline{cf}}$	Özellik-sınıflandırma korelasyonlarının ortalama değeri
$r_{\overline{ff}}$	Özellik-özellik korelasyonlarının ortalama değeri
R_i	i. kümedeki sıralama değerlerinin toplamı

KISALTMA LİSTESİ

AIRS	Artificial Immune Recognition Systems
ARB	Artificial Recognition Ball
AST	Abstract Syntax Tree
AUC	Area Under Curve
AVG	Average
CFS	Correlation-based Feature Selection
CMS	Content Management System
CSS	Cascading Style Sheets
CVS	Concurrent Version System
DOM	Document Object Model
HTML	HyperText Markup Language
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GA	Genetik Algoritmalar
GP	Genetik Programlama
GPL	GNU Public License
NB	Naive Bayes
PD	Probability of detection
PF	Probability of false alarms
PHP	PHP: Hypertext Preprocessor
RO	Rastgele Orman
RFC	Request For Comments
ROC	Receiver-Operator Characteristics
SQL	Structured Query Language
SVN	Subversion
TN	True Negative
TP	True Positive
TPR	True Positive Rate
YBS	Yapay Bağışıklık Sistemleri
YSA	Yapay Sinir Ağları
XML	eXtensible Markup Language
XSS	Cross-site Scripting

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

Sayfa

ÖZET

WEB UYGULAMALARINDA ALANA ÖZGÜ METRİKLER İLE HATAYA YATKINLIK TAHMİNİ

Mehmet Serdar BİÇER

Bilgisayar Mühendisliği Anabilim Dalı
Doktora Tezi

Tez Danışmanı: Prof. Dr. Banu DİRİ

Bilgisayarın ilk yapılışı bilimsel ve askeri amaçlar ile olmuştur. Teknolojinin gelişmesi ile birlikte bilgisayarların boyutları küçülmeye başladıkça kullanımları yaygınlaşmıştır ve başta ticari uygulamalar olmak üzere farklı alanlar için kullanılmaya başlanmıştır. Farklı programlama dilleri ortaya çıkmaya başladı. Yazılım projelerindeki karmaşıklığı yönetebilmek için yazılım mühendisliği kavramının ortaya atılması ile birlikte yazılımcıların hayatını kolaylaştırmak için farklı yöntemler üretilmeye başlandı. Yazılımın hataya yatkınlığının tahmini de bu yöntemlerden biridir. Her yazılım ürünü için kullanıma açılmadan önce test sürecinden geçmesi bir zorunluluktur. Ancak, yazılım geliştirme sürecinin her aşamasında olduğu gibi test aşamasında da kaynakların verimli bir şekilde kullanılabilmesi için planlama yapılması gerekmektedir. Test sürecinin düzgün bir şekilde planlanması için kaynakların aktarılacağı bölümlerin belirlenmesi gerekmektedir. Hataya yatkınlık tahmini yöntemleri proje yöneticileri tarafından, kısıtlı olan kaynakları test aşamasında verimli bir şekilde dağıtmak için kullanılmaktadır. Bu yöntemler yazılım testinde görev yapan kişilere test senaryolarının ne şekilde üretetileceğine ve organize edileceğine karar vermelerine yardımcı olmaktadır. Hatalı modüllerin doğru tahmin edilmesi yazılım testinin masrafını azaltmaktadır ve proje yöneticileri kısıtlı kaynaklarını işlere atama konusunda daha rahat hareket edebilmektedir.

Hataya yatkınlık tahmini, yazılım mühendisliğinde son 10 yılın en aktif araştırma konularından biri olmakla birlikte, son yıllardaki çalışmalarda tahmin modellerinin başarılarının doyum noktasına ulaştığı gösterilmiştir. Tahmin modellerinin performansını arttırmak için genel kanı kullanılan algoritmalardan ziyade veri kümelerinin iyileştirilmesi gerektiği yönündedir. Şimdiye kadar yapılan araştırmalarda veri kümelerinin iyileştirilmeye çalışılması için uygulama türüne özel veri çıkarımı gerçekleştirilmediğini fark ettik. Bu noktadan hareketle web uygulamaları alanını seçerek, bu türdeki açık kaynaklı uygulamalardan özel

metrik kümeleri çıkardık. Deneylerimiz sonucunda alana özel çıkarılan metriklerin hataya yatkınlık tahmini modellerinin performansını arttırmaya yardımcı olduğunu gözlemledik.

Anahtar Kelimeler: Yazılım hataya yatkınlık tahmini, yazılım mühendisliği, kalite kestirimi

ABSTRACT

PREDICTION OF DEFECT PRONENESS IN WEB APPLICATIONS BY DOMAIN-SPECIFIC METRICS

Mehmet Serdar BİÇER

Department of Computer Engineering
PhD. Thesis

Advisor: Prof. Dr. Banu DİRİ

Invention of the computer was for scientific and military purposes. As the size of computers began to shrink with the improvement in the technology, their use became widespread and they started to be used for different fields, mainly commercial applications. Different programming languages have begun to emerge. With the introduction of the concept of software engineering to manage complexity in software projects, different methods have begun to be developed to facilitate the software development process. Software defect prediction is one of these methods. Each software product has to pass through the testing process before the product is released. However, as in all stages of the software development process, it is necessary to plan for the resources to be used effectively in the test phase. In order to properly plan the testing process, it is necessary to determine the sections to which the resources are to be transferred. Software defect proneness prediction techniques are used by project managers to efficiently allocate their precious resources in testing phase. These techniques give software testers some insight to decide how to create and organise test cases. Accurate prediction of defect prone modules can reduce the cost of software testing and allow project managers to act more confidently in assigning their resources.

Predicting defect prone modules in software projects has been one of the most active research topics in software engineering in the last decade, and in recent years studies have shown that prediction models reached a ceiling effect. In order to improve the performance of the prediction models, the general consensus is that focus should be based on improving the datasets rather than algorithms used. We have noticed that, to this date, researchers have not benefited from domain-specific data. From this point, we choosed the web applications domain and extracted custom metric sets from some open source web applications. As a result of our experiments, we found that domain specific metrics

helped us to improve the performance of software defect proneness prediction models.

Keywords: Software defect proneness prediction, software engineering, quality prediction

BÖLÜM 1

GİRİŞ

Günümüzde yazılım uygulamaları yaşamımızın önemli bir parçasını oluşturmaktadır. Her yaştan ve meslekten insan günlük işlerinde farklı tipte yazılım ürünleriyle etkileşime geçmektedir. Bunlara örnek olarak kişisel bilgisayarlar, beyaz eşyalar, cep telefonları, uçaklar ve ATM makinalarında kullanılan yazılımları verebiliriz. Son 50 yılda, özellikle kişisel bilgisayarlar insanların hayatına girdikten sonra, yazılım tabanlı sistemlerin kullanımı gözle görülebilir derecede artmıştır. Yıllar geçtikçe yazılımların karmaşıklığı daha da artmış, yazılım projelerinin yönetimi ve geliştirilmesi daha zor bir iş haline gelmiştir. Yazılım mühendisliği disiplini, zaman içinde bu zorluklarla mücadele etme, geliştiricilere ve yöneticilere işlerini kolaylaştıracak yöntemler ve araçlar ortaya çıkarma gereksinimi üzerine doğmuştur. Bu alanda kullanılabilecek çok farklı teknolojiler bulunmaktadır ve bu teknolojileri birleştirmek çoğu zaman bir zorunluluktur.

Ne yazık ki, bütün bu karmaşıklıkların bir arada kullanılması bu büyük endüstride ciddi sorunlara neden olmaktadır. 2004 yılında Standish grubu, projelerin yalnızca %29'unun başarıyla tamamlandığını bildirmiştir. Projelerin %18'i başarısız olurken %53'ü bütçelerini ve / veya zaman kısıtlamalarını aşmıştır. 2012 yılında, raporlanmış başarılı projelerin yüzdesi [1] %39'a yükselirken, zorlanan projelerin oranı %43'e düşmüş, başarısız projeler ise %18'de kalmıştır. Başarılı projelerin oranında artış görülmesine rağmen, bu oran hala düşüktür. Bir projenin başarılı kabul edilebilmesi için, zaman ve bütçe dahilinde ve yüksek kalitede tamamlanmış olması gerekmektedir.

1.1 Literatür Özeti

Bir yazılım projesinin başarısını belirleyen ana faktör yazılımın kalitesidir [2]. Yazılım kalitesi için birden fazla tanım bulunmakla birlikte bunlar içinde öne çıkanı "yazılımın ne kadar iyi tasarlandığı ve çıkan ürünün bu tasarıma ne kadar uyduğu"dur [3]. Yazılımın kalitesi geliştirme sürecindeki test aşamasıyla çok yakından ilişkilidir. Bu ilişki projenin zaman ve bütçe kısıtlarını da çok yakından etkiler. Örneğin 2002'deki IEEE Metrik Paneli'nde [4] araştırmacılar harcanan eforun yarısının aslında önlenebileceğini, bunların %80'inin de

hataların küçük bir kısmından (yaklaşık %20) kaynaklandığını öne sürmüşlerdir. Bu tip önlenebilir eforlar daha önceden keşfedilip daha az maliyetle çözülebilecek veya tamamen önlenebilecek hatalardan kaynaklanmaktadır [5]. Dikkatli tasarlanmış test aktiviteleri başarılı ürünler doğururken kaotik, rastgele veya doğru yapılmayan test aktiviteleri kısıtları aşmış veya iptal edilmiş ürünlere yol açmaktadır. Yazılım testinde uygulanabilecek en basit yaklaşım verilen bir kod parçasındaki bütün olasılıkları test etmektir. Bu durum zaman ve bütçe kısıtları nedeniyle pratikte imkansızdır. Bu nedenle yazılım proje yöneticileri ürünlerindeki hataya yatkınlığı ölçmek için çoğunlukla öğrenme tabanlı tahmin yöntemleri kullanmaktadır.

Hataya yatkınlık tahmini yöntemleri proje yöneticileri tarafından, test aşamasında, kısıtlı olan kaynakları verimli bir şekilde dağıtmak için kullanılmaktadır. Bu yöntemler yazılım testinde görev yapan kişilere test senaryolarının ne şekilde üretetileceğine ve organize edileceğine karar vermelerine yardımcı olmaktadır. Hatalı modüllerin doğru tahmin edilmesi yazılım testinin maliyetini azaltır ve proje yöneticileri kısıtlı kaynaklarını işlere atama konusunda daha rahat hareket edebilirler [6]. İdealde bir hataya yatkınlık tahmini modeli bütün hataları doğru tahmin ederken hatasız modülleri hatalı olarak işaretlememelidir. Ama pratikte bu duruma çok az rastlanır [7]. En yeni tahmin modelleri bile bu noktaya erişmekten çok uzaktadır [8], [9]. Yüksek tahmin oranına sahip modeller yüksek yanlış alarm oranına sahiptir. Yüksek yanlış alarm oranları hatasız kodların boş yere test edilmesine yol açar. Bu durum yüksek güvenlik gerektiren uygulamalar için bir soruna yol açmaz çünkü bu tip uygulamalarda karşılaşılacak bir hatanın bedeli çok yüksektir. Ama bu durum kaynak açısından kritik projeler için ciddi bir problemdir [10], [11], [12]. Kodun gereksiz yere gözden geçirilmesi test aşamasını uzattığından bütçe ve zaman kısıtlarını aşma riskini arttırır. Bu nedenle mühendisler doğru ve yanlış tahmin oranlarını dengeleme yoluna gitmelidir [10].

Bu alanda çalışan araştırmacılar şimdiye kadar hataya yatkınlık tahmini modellerini kurarken statik kod metrikleri, kod değişim metrikleri, geliştirici ve modül ağları gibi farklı metrik kümelerinden yararlanmışlardır. Bunlar arasında statik kod metrikleri 1970'lerden beri kullanılmaktadır [13], [14], [15]. Otomatik araçlar yardımıyla da projelerden metrikleri çıkarmak çok daha kolay hale gelmiştir.

2012 yılı verilerine göre Kuzey Amerika'nın %78'i, Avrupa'nın %63'ü Internet kullanmaktadır [16]. Dünya çapında yapılan Internet tabanlı işlemlerin yıllık tutarı trilyon dolarlarla ölçülmektedir [17]. E-ticaret dışında her gün milyonlarca kullanıcının arama motorları (örn: Google), sosyal paylaşım platformları (örn: Facebook, Twitter), bilgi paylaşımı (örn: Wikipedia) gibi farklı amaçlarla farklı web sitelerini kullandıkları bilinmektedir. Internet kullanımının bu kadar büyük hacimde olması karşısında firmalar açısından erişilebilir olmak günümüzde büyük bir ihtiyaç halini almıştır. Son yıllarda kızışan tarayıcı savaşları ve buna paralel gelişen teknoloji ve performans artışı geliştiricilerin bu alanda ilerlemesine im-

kan sağlamıştır. Bu alanda geliştirme yaparken kullanılan teknolojilerin de ilerlemesiyle web geliştiricileri artık daha özgürce daha iyi uygulamalar çıkarabilmektedir. Mobil cihaz kullanımındaki artışla birlikte web uygulamalarına artık çok daha farklı tipte ekranlardan erişilebilmek gibi gereksinimler eklenmeye başlanmıştır.

Ancak, web uygulamalarındaki hatalar firmalara milyonlarca dolar kaybettirmeye devam etmektedir. Web uygulamalarının masaüstü uygulamalardan farklı olarak yüksek erişilebilirliğe sahip olması gerekmektedir. Uygulamada yaşanacak en ufak sıkıntıların firmalara faturası büyük olmaktadır. Örneğin, 2001 yılı şükran günü tatilinde Amazon'un yaşadığı sıkıntılar 20 dakikada 500 bin dolar kaybetmesine neden olmuştur [18]. Hataların görünmeyen faturası ise daha büyüktür, her hata kullanıcı sadakatinin bozulmasına ve müşteri kaybına neden olmaktadır [19].

Web uygulamalarını bu kadar özel kılan nedir? Aşağıdaki maddeler web uygulamalarını özel kılan, artan web sitesi kullanımın arkasındaki sebepler arasında gösterilebilir: [20]

- Web uygulamalarına her yerden erişilebilir,
- Değişiklikler son kullanıcılara kolay ve hızlı bir şekilde yansır,
- Daha zengin ve daha etkileşimli kullanıcı arayüzleri kolaylıkla oluşturulabilir,
- Diğer web uygulamalarındaki kaynaklara erişilmesi ve kullanılması (ör. sosyal platform entegrasyonu, video yerleştirme, vb.),
- Tek bir yerde farklı teknolojileri birleştirme ve kullanma becerisi. Bir web sayfası, biçimlendirme, CSS dosyaları, komut dosyaları, sunucu tarafı kodu, veritabanı erişimi gibi farklı bileşenlere sahip olabilir.

Web uygulamalarını geliştirmede kullanılan yaşam döngüsü, diğer uygulamalar için kullanılanlarla aynı olsa da teknik bakımdan ayrıştıkları bazı noktalar bulunmaktadır.

- Öncelikle web uygulamalarının geliştirilmesinde birden fazla programlama dili, tasarım özelliği, dışarıdan kullanılmakta olan kütüphane ve bileşenler bulunur. Bunlara örnek olarak geleneksel programlama dilleri, script dilleri, düz HTML sayfaları, XML tabanlı şablon dosyaları, veritabanları, resimler ve CSS kodları verilebilir.
- Geliştirilen uygulamalar tarayıcılara bağımlı halde çalışmaktadır. Aynı kod farklı tarayıcıda farklı şekilde çalışabilmektedir. Bunu önlemek için kodun tarayıcı bağımsız çalışacak şekilde yazılması ve uygulamanın farklı tarayıcılar için test edilmesi gerekmektedir.
- Güvenlik zafiyeti daha fazladır. Öncelikle kullanıcı tarafında çalışan kodlara erişip incelemek çok kolaydır. Ayrıca, Internet aracılığıyla daha geniş bir kullanıcı kitlesine hitap ettiğinden daha fazla tehdite maruz kalmaktadır.
- Dış dünya değişimlerinden daha çok etkilenmektedir. Internet bağlantısının kaybolması veya yavaşlaması durumları geliştirme sırasında hesap edilmezse istenmeyen

durumlarla karşılaşma şansı yüksektir.

- Uygulama bileşenleri gerçek ortamda ve hatta geliştirme sırasında farklı makinelere dağıtılmış halde bulunabilir ve bu halde birbirleriyle uyumlu ve bir bütün çalışmak durumundadırlar.

Bütün bunlar uygulamanın karmaşıklığını arttırıcı ve alana özel faktörlerdir [21].

1.2 Tezin Amacı

Araştırmacılar, şimdiye kadar önerdikleri modellerinde girdi olarak statik kod ölçütleri, kod geçmişi metrikleri, geliştirici ağları ve modül ağlarını kullanıyorlardı. Otomatik metrik çıkarma araçları, bu metrikleri kaynak kodu ya da sürüm kontrol sistemlerinden toplamayı ve kullanmayı kolaylaştırmaktadır. Son yıllarda araştırmacılar, bu metrik setlerin zaten bir performans tavanına [9] sahip olduğunu keşfetmişlerdir. Bu tavan etkisini ortadan kaldırmak için iki seçeneğimiz bulunmaktadır:

- Var olan metrik setleri üzerinde yeni veri madenciliği teknikleri uygulamak.
- Eğitim verisini var olan veri madenciliği tekniklerini uygulamak üzere değiştirmek / iyileştirmek.

Daha iyi algoritmaların araştırılmasının, kusur tahmin yöntemlerinin performansını iyileştirmek için çaba sarf etmeye değmediğine inanılmaktadır [9]. Lessmann vd. [8] de bu sorunu araştırmıştır. 22 sınıflandırma modeli ile bir karşılaştırma yapılmış ve aynı veri kümesindeki 19 öğrenicinin sonuçları arasında istatistiksel olarak bir fark görülmemiştir. Bu nedenle, verilerin eğitim kalitesini yükseltmek veya yeni metrikler kullanmanın, tahmin modellerinin performansını artırmak için daha verimli olacağı gözlemlenmiştir. Bu çalışmada tavan etkisini ortadan kaldırmak için sınıflandırıcıların eğitim verilerinin bilgi içeriğini iyileştirmek için motive olduk.

Yazılım hataya yatkınlık tahmini literatüründe en çok kullanılan metrik türleri statik kod metrikleri ve kod değişimi metrikleridir. Literatür taramasının detayları Bölüm 2'de verilecektir. Bu metrik türlerinde en çok kullanılan özellikler, ve literatürdeki diğer metrik setleri Ek A'da verilmiştir. Bu metriklerin kullanıldığı çalışmaların büyük bir bölümü masaüstü veya gömülü sistem uygulamalarından çıkarılan verilerle yapılmıştır. Yapılan literatür taramasında bir web uygulamasına özel metrikler kullanıldığı hiç görülmemiştir. Bu durum 2 seçenek getirmiştir:

- Web uygulamalarının ayrı olarak incelenmesi gerekmekte midir, yoksa diğer alanlardaki uygulamalarla yapılmış olan hatalılık tahmini teknikleri web uygulamalarına da uygulanabilir mi?
- Bu alanda bazı araştırmalar olabilir, ancak çıkmaza ulaştıklarından dolayı yayınlanmamıştır.

İlk seçenek için, Bölüm 2'den de görülebileceği üzere, web alanındaki uygulamalara özel yapılmış bazı çalışmalar bulunmaktadır. Bu çalışmalar uygulamaların farklı yönlerine odaklanmış olsalar da, hataya yatkınlık tahmini alanında bir çalışma bulunmamaktadır. Ancak, web uygulamalarına özel bazı karakteristik farklılıklar olduğunu görmemize yardımcı olmuşlardır. Dolayısıyla, web uygulamaları için özel metrik gereksiniminin güçlendirildiği sonucuna varılmıştır. İkinci seçenek göz ardı edilebilir çünkü buna dair bir veri bulunmamakla birlikte bu durum yalnızca araştırma sırasında fark edilebilir.

Bu tezin arkasındaki motivasyon, web uygulamalarının diğer uygulama türlerinde bulunmayan ve yazılım hataya yatkınlık tahmini alanında yeterince incelenmeyen bazı özelliklere sahip olmasıdır. Bu özellikler incelenerek hataya yatkınlık tahmininde yararlanmak üzere keşfedilecektir.

1.3 Orijinal Katkı

Yukarıda bahsedilen nedenlerden dolayı, web uygulamaları üzerinde yeni statik kod metrikleri seti sunmayı amaçladığımız veri merkezli bir yaklaşım öneriyoruz. Bu amaçla, web uygulamalarının hata taksonomileri incelenmiş ve hataya yatkın kısımları tespit etmeye yönelik olası metrikler tespit edilmiştir. Web uygulamalarında kullanılan teknolojiler iki kısma ayrılmıştır: sunucu tarafı ve istemci tarafı. Çalışmada öncelikle sunucu tarafı kodları analiz edilerek yeni metrikler sunulmuştur. Web uygulamalarının özellikleri ile yazılım ürünlerinin hatalı olma olasılığı arasındaki ilişki araştırılmıştır. İstemci taraflı kodlar ayrıca analiz edilerek bu bölüme özel ayrı metrikler çıkarılmıştır. Bulduğumuz sonuçların var olan statik kod metrikleriyle karşılaştırması yapılıp, farklı yöntemlerle iyileştirip iyileştirilemeyeceği ve pratikte ne kadar değerli oldukları araştırılmıştır.

Günümüzdeki çalışmalar, yazılım hataya yatkınlık tahmini modellerinde kullanılan algoritmaları iyileştirmeye çalışmanın, modellerin performanslarını artırmak için çaba harcamaya değer olmadığını göstermektedir. Algoritmaları iyileştirmeye çalışmak yerine tahmin modellerinin girdi verisinin kalitesini ve bilgi içeriğini zenginleştirmemiz gerekmektedir. Bildiğimiz kadarıyla web uygulamalarının özelliklerinin hatalılık tahmininde kullanılabilirliği bugüne kadar analiz edilmemiştir. Bu nedenlerden ötürü, web uygulamaları için yeni metrik kümesi sunduğumuz veri merkezli bir yaklaşım öneriyoruz.

Bu çalışmada web uygulamalarının hataya yatkınlığını tahmin etmek için yeni bir metrik kümesi çıkarmanın değerini araştırmak için iki araştırma sorumuz bulunmaktadır:

- Web uygulamalarının karakter özelliklerini kullanarak hangi metrik türleri çıkarılabilir?
- Kusurları tahmin etmek için web uygulamasına özgü metriklerden nasıl yararlanabiliriz?

Bu tez çalışması aşağıdaki katkıları sağlamıştır:

1. İlk kez web alanına özel hataya yatkınlık tahmini metrikleri ortaya atılmıştır.
2. İlk kez istemci tarafında kullanılan biçimlendirme dillerine özel hataya yatkınlık tahmini metrikleri tanımlanmış ve bu metriklerin hata yatkınlığı tahmininde kullanılabilirliği incelenmiştir.
3. İlk kez stil şablonlarına (CSS) özel hataya yatkınlık tahmini metrikleri tanımlanmış ve bu metriklerin hata yatkınlığı tahmininde kullanılabilirliği incelenmiştir.
4. Yazılımın hataya yatkınlığı tahmininde alana özgü metrikleri kullanmanın getirdiği fayda ilk defa araştırılmıştır.

BÖLÜM 2

İLGİLİ ÇALIŞMALAR

Yazılım ürünlerinin hataya yatkınlığının tahmin edilebilmesi 2005 yılından bu yana aktif olarak üzerinde çalışılan bir alan olmuştur [22]. Statik kod metrikleri bir çok araştırmacı tarafından üzerinde en çok çalışılan metrik tipidir [2], [8], [15], [13], [23], [24], [25], [26], [27]. Bu alanda literatürde bilinen ilk araştırma bir statik kod metriği olan kod satır sayısı metriği ile yapılmıştır [13]. Bundan sonra Halstead metrikleri [23] ve McCabe döngüsel karmaşıklığı [24] ortaya atılmıştır. Bu ölçütler elde edildikleri kodun boyutu ve karmaşıklığına dair bulgular edinmemizi sağlarlar ve günümüzde otomatik araçlar ile kaynak koddan kolayca çıkarılabilirler. Bu metrikler günümüzde hatalılık tahmini çalışmalarında yaygın olarak kullanılmaktadır. Fakat bu çalışmalarda, metrikler masaüstü veya gömülü uygulamalardan çıkarılmıştır ve bunların hiçbiri web uygulamalarından elde edilmiş metrik içermez.

Yaygın olarak kullanılan diğer bir metrik tipi de kod değişim (code churn) metrikleridir [28], [29], [30], [31]. Bu metrik seti SVN ve Git gibi versiyon kontrol sistemlerinden elde edilmektedir. Kod değişimi metrikleri ilk olarak Munson vd. [30] tarafından ortaya atılmıştır. Bu metrik tipinin raporlanmış sorunlarla yüksek korelasyon gösterdikleri gözlenmiştir. Eklenmiş/silinmiş kod satır sayısı, kodun yaşı, kodda yapılmış değişiklik sayısı, kod üzerinde çalışmış geliştirici sayısı gibi değerleri içermektedir.

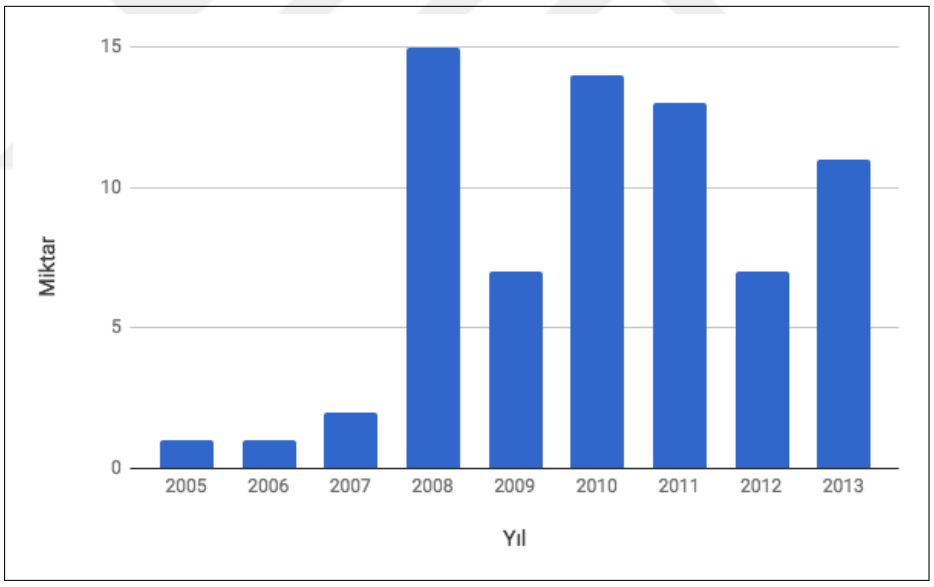
Bunlara ek olarak, nispeten yeni bir metrik türü olan sosyal ağ ölçümleri de yazılım hatası tahmininde kullanılmaktadır [32], [33], [34], [35], [36]. Bu tür metrikler, programlama dilinden veya kodun kendisinden bağımsızdır; üretilen sosyal ağlar aynı dosya üzerine çalışan geliştiricilerden veya birbiriyle bağlantılı dosyalardan oluşur. Bu alandaki araştırmacılar, bağımlı dosyaları veya aynı dosya üzerinde çalışan geliştiricileri birbiriyle bağlayarak sosyal ağlar kurar ve bu ağları ağ analiz teknikleri kullanarak incelerler.

Web uygulamaları üzerinde de benzer alanlarda çalışmalar bulunmaktadır. Bunların bazıları [21], [37] uygulamaların kalite özelliklerini tanımlamaya odaklanmıştır. Ancak, çoğu araştırma [38], [39], [40], [41], [42], [43], [44], [45], [46] test senaryolarını iyileştirme veya otomatize etmeye odaklanmıştır. Web uygulamalarında görülen hataların taksonomisini

yapmaya yönelik araştırmalar da bulunmaktadır [19], [44], [47], [48]. Bu araştırmalar web uygulamaların karakterlerindeki farklılıkları tanımlayabilmemizi sağlamaktadır.

Web uygulamalarına özel statik kod metrik seti çalışmaları da bulunmakla beraber [49], [50], [51], bu çalışmalar XSS veya SQL enjeksiyonu gibi saldırıların önlenmesi için kullanılmaktadır.

Tez gelişim sürecinde öncelikle literatürde var olan metriklerin web uygulamaları için yeterli olup olmadığı sorgulanmıştır. Bu metrikler web uygulamalarına özel ortaya atılmış olmasa bile programlama dillerinin genel yapısından dolayı uygunluk göstermeleri olasıdır. Bu amaçla yazılımın hataya yatkınlığı tahmini alanında yapılan çalışmalar incelenmiştir. Bu amaçla ağırlıklı olarak son 6 yılda yayınlanmış 71 farklı makale incelenip [6], [8], [9], [12], [15], [31], [32], [33], [34], [35], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61], [62], [63], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85], [86], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99], [100], [101], [102], [103], [104], [105], [106], [107], [108], [109], [110], [111] yayın yılları, kullanılan metrik tipleri, odak noktaları ve veri setlerinin çıkarıldığı uygulamaların programlama dilleri açısından sınıflandırılmıştır.

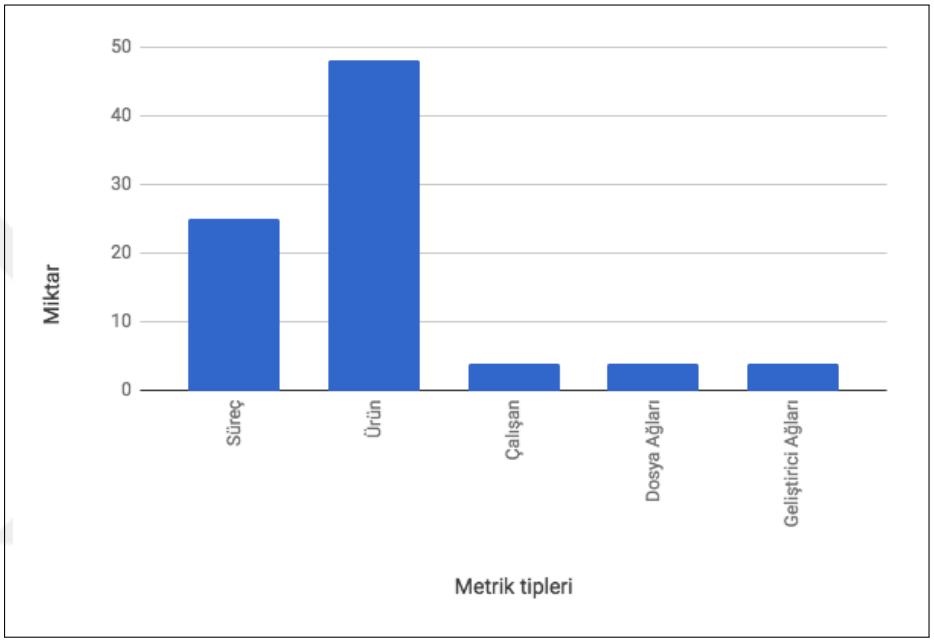


Şekil 2.1 Çalışmaların yıllara göre dağılımı.

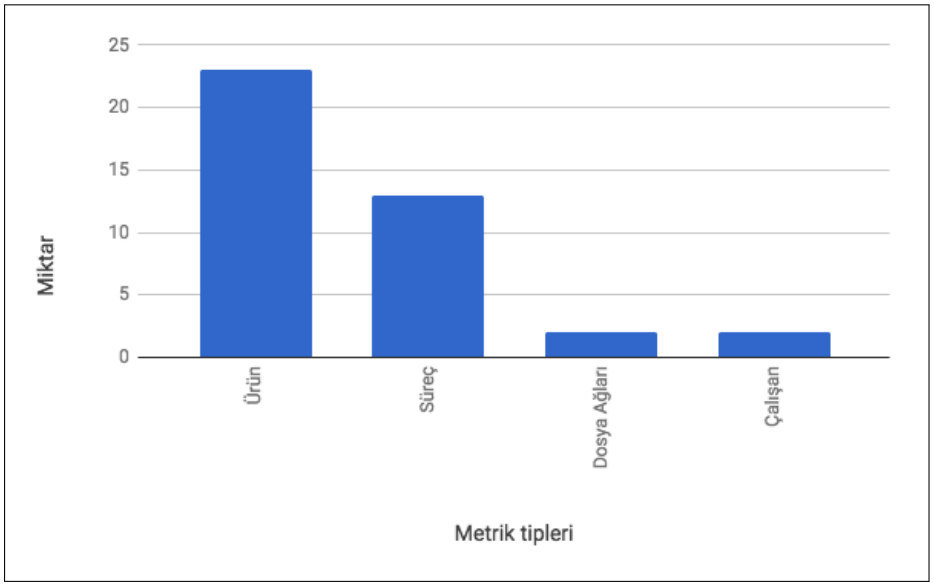
Şekil 2.2'de görüldüğü üzere çalışmalarda 5 farklı metrik tipi kullanılmıştır. İncelenen çalışmaların önemli bir kısmında birden fazla metrik tipi birden kullanılmakla beraber, en çok kullanılan metrik tipi 48 kullanım ile ürün tabalnlı metrikleridir. Bu metrikler statik kod metrikleri, nesneye yönelik metrikler ve işlev puanını içerir. Bu metrik tipini 25 kullanım ile süreç metrikleri izlemiştir. Bu metrik tipinde ağırlıklı olarak kod değişim metrikleri yer almakla birlikte nadiren efor ve test senaryosu sayısı gibi farklı metrikler de yer alır. Şekil 2.3'den de görüleceği üzere 2011-2013 yılları arasında yapılan çalışmalarda da genel dağılım değişmemektedir.

Bu metrik tiplerinden en çok kullanılmakta olanları Ek A'da listelenmiştir. Ancak bu çalışmaların ezici bir çoğunluğu masaüstü uygulamalarına ait veriler kullanılarak yapılmıştır. Bu durum aklımıza 2 olasılık getirmiştir:

- Web uygulamalarına özel bir çalışma yapmaya gerçekten gerek var mı, yoksa masaüstü uygulamalarında yapılan hataya yatkınlık tahmini çalışması web uygulamalarına bire bir uygulanabilir mi?
- Bu alanda bir çalışma yapılmış olabilir, ancak bir sonuca varılmadığı için yayınlanmamıştır.

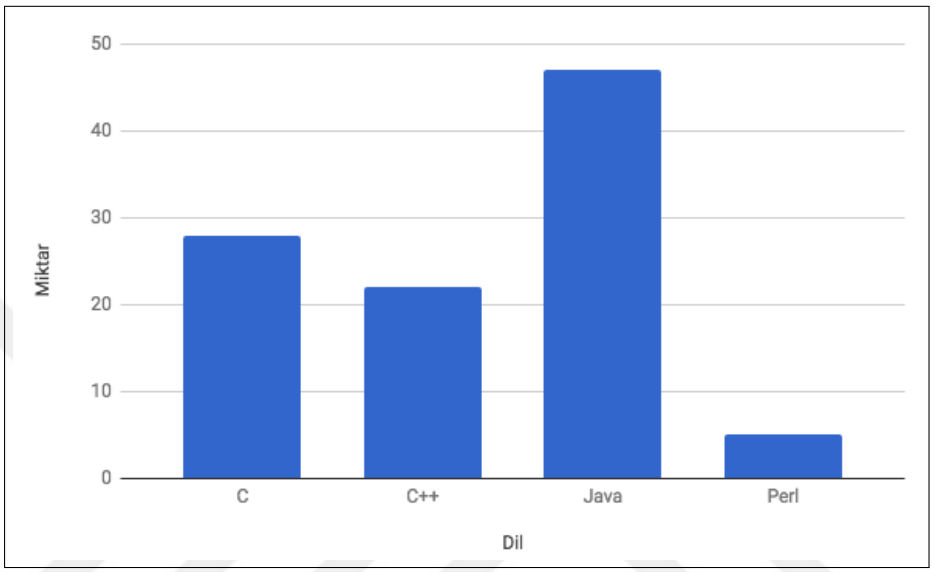


Şekil 2.2 Çalışmalarda kullanılan metrik tipleri



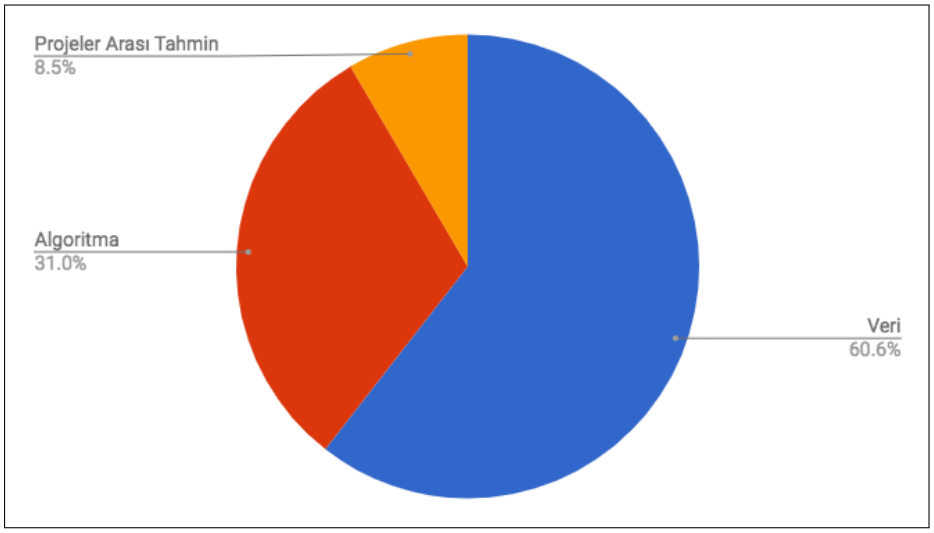
Şekil 2.3 2011-2013 arası çalışmalarda kullanılan metrik tipleri

Çalışmalarda kullanılan veri kümesi kaynaklarının sahip oldukları programlama dillerine bakıldığında C ailesinin (C ve C++) toplam 50 kullanımla en çok kullanılan diller olduklarını görmekteyiz. Java programlama dili tek başına 47 çalışmada kullanılarak en çok kullanılan dil olmuştur. Bu diller haricinde nadiren 5 çalışmada da Perl tabanlı projelerden çıkarılan metriklerin kullanıldığını görmekteyiz. Bu durum bize yazılımın hataya yatkınlığının tahmini araştırmalarının ağırlıklı olarak gömülü sistemler, masaüstü uygulamaları ve yazılım kütüphaneleri kullanılarak yapıldığını göstermiştir.



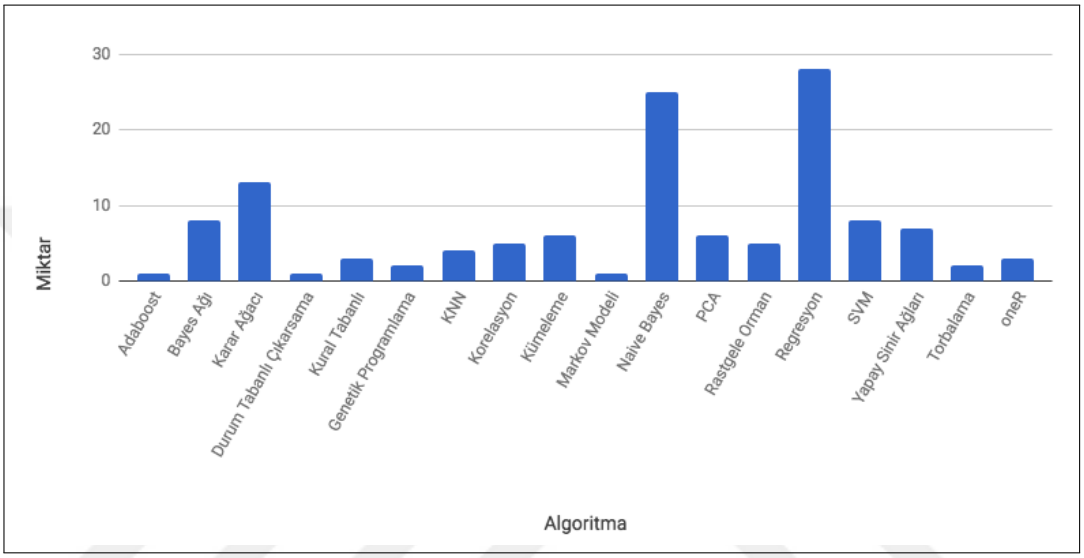
Şekil 2.4 Çalışmaların programlama dillerine göre dağılımı.

Bu alanda yapılan her çalışmanın bir odak noktası vardır. Bazı araştırmacılar istedikleri iyileştirmeyi farklı algoritmalar kullanarak, bazıları da veriyi iyileştirerek yapmaya çalışırlar. Bu alandaki bazı araştırmalarda da ayrı bir odak noktası olarak alabileceğimiz farklı veri kümeleri arası tahmin bulunmaktadır.



Şekil 2.5 Çalışmalardaki odak noktaları.

Yazılımın hataya yatkınlığının tahmini alanında farklı öğrenme yöntemleri kullanılmaktadır. Örneğin sadece tez çalışması kapsamında yapılan literatür araştırmasında 18 farklı algoritma kullanımına rastlanmıştır. Bu algoritmalar istatistiksel, sınıflandırma ve kümeleme tabanlı olarak 3 farklı kategoriye ayrılabilir. En çok kullanılan yöntemler sınıflandırma ve istatistiksel yöntemlerdir. Sınıflandırıcılarda Naive Bayes, istatistik yöntemlerinde Regresyon analizi kullanımı öne çıkmaktadır. Literatür araştırmasında en iyi sonucu verdiği bildirilmesine rağmen Rastgele Orman algoritmasının kullanımı düşük sayıda kalmıştır. Naive Bayes yeterince iyi sonuç veren basit bir algoritma olduğu için tercih edilmektedir.



Şekil 2.6 Çalışmalarda kullanılan algoritmalar.

2.1 Çalışmaların İncelenmesi

Koru ve Liu'nun çalışmasında [105] modül boyutunun statik kod metrikleri kullanıldığı durumda elde edilen hataya yatkınlık tahmini başarısına etkisi incelenmiştir. Çalışma sonucunda büyük modüllerde daha iyi sonuç alındığı görülmüş, küçük modüllerin de daha yüksek bir soyutlama seviyesi kullanılarak birleştirilmesi tavsiye edilmiştir. Bu sonuçla ilgili olarak, sınıf seviyesinde hataya yatkınlık tahmini yapmanın metot seviyesinden daha iyi sonuç verdiği gözlenmiştir.

Song vd. [6] hata düzeltme eforunu tahmin etmeye ve hata ilişkilerini ilişkilendirme kuralları aracılığı ile tahmin etmeye çalışmışlardır. Tahminlerinde %95-%93 civarında doğruluk değerlerine ulaşmalarına rağmen tek veri seti üzerinde çalışmaları nedeniyle çalışmanın doğrulanması gerekmektedir.

Weyuker vd. [58] farklı bir metrik kümesi kullanarak bir hataya yatkınlık tahmini çalışması yapmıştır. Dosyalar üzerinde çalışan geliştirici sayıları ve farklı varyasyonları kullanılarak hataya yatkınlık tahmin oranını %1 arttırmışlardır. Bu metrikler tek başlarına değil, daha önce kullandıkları kod değişimi metriklerine ek olarak kullanılmıştır.

Menzies vd. [15] statik kod metriklerinin hataya yatkınlık tahmini çalışmalarında kullanılabilirlğini araştırmışlardır. Ortalamada %71 doğru tahmin ve %25 yanlış alarm değerlerine ulaşmışlar ve bu değerin manuel yapılan incelemelerden çok daha iyi olduğunu, statik kod metriklerinin kullanımının faydalı olduğunu göstermişlerdir.

Menzies ayrıca başka bir ekiple yaptığı araştırmada [9] hataya yatkınlık tahmini yöntemlerinin karşılaştığı tavan etkisini incelemiştir. Promise havuzundan 12 veri kümesi Naive Bayes ve J48 kullandıkları bir modelle farklı örnekleme yöntemlerini denemiştir. Bu çalışma sınıflandırma algoritmalarının başarısının bir tavan değerine ulaştığını, başarıyı arttırmak için veri içeriğini arttırmak gerektiğini önermeleri açısından önemlidir.

Jiang vd. [12] NASA ve Promise kaynaklı veri kümeleri üzerinde yaptıkları deneylerle 5 farklı sınıflandırma algoritması kullanan hataya yatkınlık tahmini modellerinin başarısını masraf eğrileri kullanarak karşılaştırmışlardır. Hatalı sınıflandırmanın getirdiği masrafa dikkat çekmiş, masraf eğrilerinin modelleri karşılaştırmada standart hale getirilmesini önermişlerdir.

Kastro ve Bener [59] Linux kerneli veri setleri üzerinde, farklı versiyonlar arasındaki kusur değişimini tahmin etmek üzerine çalışmışlardır. Bu çalışma ürün gerçek ortama çıkarılmadan önce kalitesini tahmin edebilmek açısından önemlidir.

Zimmermann vd. [62] tarafından yapılan araştırmada kod değişim metriklerinin yazılım hatalılık tahmininde kullanılabilirliği incelenmiştir. Eclipse ve Windows Server 2003 veri kümeleri üzerinde yapılan çalışmalarda tahmin performansları manuel incelemeler ile karşılaştırılmış ve %37'ye karşı %69'luk performans ile iyileştirme yakalanmıştır.

Lessmann vd. [8] NASA kaynaklı 11 farklı veri seti üzerinde 22 makine öğrenmesi yöntemi uygulayarak yaptıkları sonuçlarda 2 önemli sonuca ulaşmışlardır. Birincisi, Rastgele Orman algoritması diğer algoritmalardan daha iyi sonuç vermektedir. İkincisi, algoritmaların büyük çoğunluğunun performansı zaten birbirinden çok da farklı değildir. Bu nedenle artık odak noktası algoritmaları iyileştirmek yerine verileri iyileştirmek olmalıdır.

Chang vd. [82] kusurları önlemek için yazılım hatalılık tahminini daha erken safhalarda kullanmaya mümkün kılabilecek ilişkilendirme kuralları tabanlı bir yöntem üretmiştir.

Meneely vd. [32] ortak dosya üzerinde çalışan geliştiricilerden sosyal ağlar kurup, bunlar üzerinde sosyal ağ analizi yaparak uygulamaların hataya yatkınlığını tahmin etmeye çalışmış, korelasyon analizi ile statik kod metriklerinden daha iyi sonuç verdiğini göstermişlerdir. Sonuçları kod değişim metriğinin arkasında olsa da alternatif veri kaynakları kullanma açısından umut verici bir çalışma olmuştur.

Moser vd. [83] statik kod metrikleri ile kod değişim metriklerini 2 veri kümesi üzerinde karşılaştırıp kod değişim metriklerinin daha iyi sonuç verdiklerini göstermişlerdir.

Pinzger vd. [33] geliştiricilerin çalıştıkları dosyalardan sosyal ağlar oluşturarak, sosyal ağ analizi ile hataya yatkınlık tahmini yapmaya çalışmışlardır.

Moser vd. [88] kod değişim metriklerinin veri zenginliği üzerine yaptıkları araştırmada 18 metrikten 3'ünün en fazla veriyi içerdiğini, ve 3 farklı veri kümesinde de aynı durumu

koruduklarını gözlemişlerdir. Bu 3 metrikle yapılan hataya yatkınlık tahmini modelinin başarısının tüm metriklerin kullanıldığı duruma göre fazla etkilenmediği de gözlenmiştir.
Tosun vd. [90] NASA veri kümeleri üzerinde yaptıkları çalışmada birden fazla makine öğrenmesi metodunu bir arada kullanarak geliştirdikleri modelde tek başına Naive Bayes kullanıldığı duruma göre tahmin performansında ve masrafta belirgin bir iyileşme sağlandığını bildirmişlerdir.
Ratzinger vd. [91] kod yeniden düzenlemelerinin hatalarla ilişkisi olup olmadığını araştırmış ve bunların arasında ters orantılı bir ilişki olduğunu gözlemlemiştir. Kodlarda düzenleme yapıldıkça ortaya çıkan hataların sayısında azalma gözlemlenmiştir.
Zimmermann ve Nagappan'ın çalışmasında [34] modül bağımlılık grafları üzerinde ağ analizi yaparak hataya yatkınlık tahmini yapılmaya çalışılmıştır. Kod karmaşıklığına dayalı ölçütlere göre %10 daha iyi tahmin performansı sağladığı gözlenmiştir.
Wahyuddin vd. [101] açık kaynaklı Apache projeleri üzerine yaptıkları çalışmalarda ürün ve süreç metriklerinin birlikte kullanılmasının hataya yatkınlık tahmini performansını arttırdığını göstermişlerdir. Değerlendirme için lineer regresyon kullanmışlar, bununla birlikte metriklerin değerlerini de ayrıca göstermişlerdir.
Tsakonas ve Dounias'ın çalışmasında [107] 4 NASA veri kümesi üzerinde genetik programlama kullanılarak hataya yatkınlık tahmini yapılmaya çalışılmıştır. Buldukları sonuçları Menzies'in çalışmaları ile [15], [25] ile karşılaştırmış ve bunlardan daha iyi sonuç aldıklarını göstermişlerdir.
Çağlayan vd. [31] Eclipse projesinden elde edilen bir metrik seti üzerinde kod değişim metriklerini kullanmışlar ve yanlış alarm oranlarını %32'den %23'e indirmişlerdir.
Tosun ve Bener [52] Naive Bayes tabanlı bir model üzerinde eşik değeri optimizasyonu yaparak yanlış alarm oranını ortalama %11 azaltmışlardır. Bu çalışmada Promise veri havuzundan alınan 12 veri kümesi kullanılmıştır.
Tosun vd. [64] özel bir telekom veri seti üzerinde statik kod analizi uygulayarak kusur kestirimi yapmışlardır. Yanlış alarm oranı %50'den %28 seviyesine inmiş, doğru tahmin oranında bir değişim olmamıştır.
Aynı araştırmacılar bir başka çalışmada [71] Eclipse veri kümeleri üzerinde kod metrikleri ile dosya ağlarının performansını karşılaştırmışlardır. Bu yaklaşımla doğru tahmin ve yanlış alarm arasında %76'lık denge değerine ulaşmışlardır. Büyük ve küçük çaplı projeler için ayrı ayrı değerlendirme yapılmış, küçük projelerde kod metriklerinin daha iyi sonuç verdiği görülmesine rağmen daha büyük projelerde kod metriklerinin dosya ağları ile birlikte kullanımının daha iyi sonuç verdiği sonucuna ulaşılmıştır.
Wolf vd. [35] IBM'in bir uygulamasından çıkardıkları veri kümesi üzerinde yaptıkları çalışmada, projede çalışan geliştiricilerin aralarındaki haberleşmenin sürümlerin başarısını tahmin etmede kullanılıp kullanılamayacağını incelemişlerdir. Tam olarak bir hataya yatkınlık tahmini çalışması sayılamasa da bu çalışmada kullanılan yöntemi izleyen başka bir çalışmada [36] geliştiiciler arasındaki iletişimin hataya yatkınlık tahmini için kullanılabile-

ceği gösterilmiştir.

Zimmermann vd. [99] 12 proje üzerinde yaptıkları analizle veri kümeleri arası hataya yatkınlık tahmininin sanılandan daha zorlu bir iş olduğunu ortaya koymuşlardır. Bunun yanında ileriki çalışmalara yardımcı olması bakımından veri kümelerinde bu farklılığa yol açabilecek bazı etmenleri tanımlamışlardır.

Çatal vd. [110] etiketlenmemiş veri kümeleri üzerinde yaptıkları çalışmada hiç etiketlenmemiş örnekler üzerinde farklı kümeleme yöntemlerini karşılaştırarak yanlış negatif sonuçlarına göre yaptıkları incelemede X-means yönteminin daha iyi sonuç verdiğini gözlemlemişlerdir.

Zheng'in çalışmasında [55] yapay sinir ağı tabanlı, hataya yatkın modüllerin yanlış sınıflandırılmasının getirdiği masrafı hesaba katan, eşik değeri değişimine dayalı bir yöntem kullanılmıştır. Hata içermeyen modüllerin yanlış sınıflandırılmasının getirdiği masraf diğer duruma göre daha düşüktür, ancak önceki çalışmaların bunu dikkate almadığını öne sürmüştür.

Jureczko ve Madeyski [63] 15 farklı açık kaynaklı proje üzerinde yaptıkları çalışmalarda dosya kümeleri oluşturarak aynı modelin farklı kümeler üzerinde kullanılabilirliğini ölçmeye çalışmışlardır. Dosyaların benzerliği içeriklerine göre değil, statik kod ve nesne metrikleri kullanılarak hataya yatkınlık tahmini yönünden gösterdikleri benzerliklere göre ölçülmüş ve kümeler k-means ve hiyerarşik kümeleme kullanılarak oluşturulmuştur.

Zhang vd. [65] hataların çoğunun kodun küçük bir kısmında bulunduğu veri kümeleri için örneklemeye dayalı bir yöntem kullanarak hataya yatkınlık tahmini performansını arttırmışlardır. Bu çalışmada da Promise havuzundan alınan veri kümeleri kullanılmıştır.

Shihab vd. [68] Eclipse veri kümeleri üzerine süreç ve kod metriklerinin birleşimi bir veri kümesi kullanıp, 34 metrikten sadece 4'ünü kullanarak başarılı bir hataya yatkınlık tahmini modeli oluşturulabileceğini göstermişlerdir. Ayrıca modellerinde istatistiksel bir yöntem olan regresyonu kullanmışlardır. Burada kullanılan süreç metrikleri dosyalar üzerinde yapılan değişiklik sayıları ile sürüm öncesi ve sonrası bulunan hata sayılarıdır.

Thilo Mende'nin Promise kullanan diğer çalışmalar üzerinde yaptığı incelemesi [69] neticesinde, diğer çalışmaların sonuçlarını replike etmenin aynı veri kullanılsa bile kullanılan araçlar elde olmayınca aynı sonuçlara ulaşmanın sanıldığı kadar kolay olmayacağı gösterilmiştir.

Çağlayan vd. [74] hata tiplerini kategorize ederek (fonksiyon testinde bulunan, sistem testinde bulunan, saha testinde bulunan) her kategori için statik kod ve kod değişimi metrikleri tabanlı farklı modeller kullanıldığı bir çalışma yapmış ve farklı modelleri birden kullanmanın tahmin performansını yaklaşık %10 arttırabileceğini göstermişlerdir.

Jureczko ve Spinellis [112] nesneye yönelik metrikleri statik kod metrikleri ile karşılaştırıp nesneye yönelik metrik kullanımının kusurları bulmadan daha uygun maliyetli olduğunu göstermişlerdir.

Menzies vd. [77] statik kod metriklerinin kusur kestirimindeki durumu hakkında genel

bir inceleme yapmıştır. WHICH adı verilen bir algoritmanın daha iyi sonuç verdiği gösterilmiştir. Bu çalışma Lessman'ın [8] çalışması ile karşılaştırılmış, tavan etkisinin varlığı kabul edilmekle birlikte masraf odaklı değerlendirme yapıldığı durumda tavan etkisinin henüz görülmediği öne sürülmüştür. Lessman ile karşılaştırılmasına rağmen bu çalışmada WHICH'in karşılaştırıldığı algoritmalar arasında en iyi sonuç veren Rastgele Orman bulunmamaktadır. Bu durum yapılan karşılaştırmanın ve öne sürülen performans iyileştirmesinin objektifliğini düşündürtmektedir.

Turhan vd. [78] başarılı bir kusur kestirim modeli oluşturmak için ne kadar az veri gerektiğini araştırmıştır. Sadece 30 hata raporu verisiyle de performanslı modeller oluşurulabileceğini, üstelik bu yöntemin projeler arası veri kullanımında da faydalı olduğunu göstermişlerdir.

Gray vd. [81] statik kod metrikleri kullanarak yaptıkları araştırmada kullanılan yöntemlerin dosyaların gerçekten kusurlu olup olmadığından ziyade kusura yatkın olup olmadığını bulduğuna dikkat çekmiştir.

Sami ve Fakhrahmad [86] NASA verileri üzerinde statik kod metrikleri tabanlı tasarım metriklerini öne sürüp bu metrikler ile hataya yatkınlık tahmini yapmış, %73 ve %91 arası doğruluk değeri elde etmişlerdir. Ancak, bu çalışmada kullanılan doğruluk değerinin hataya yatkınlık tahmini çalışmalarında kullanılmaması önerilmemektedir, bunla beraber aynı çalışmada farklı bir metrik seti ile karşılaştırma yapılmadığı için doğrulama zayıf kalmıştır.

Alhassan vd. [102] veri kümeleri üzerinde geliştirici ağları oluşturarak sosyal ağ analizi gerçekleştirmiştir. Bu çalışmada sosyal ağ metriklerinin tek başına kullanıldığı durumda yeterli performansı gösteremediği, ancak diğer metrik setleri ile birlikte kullanımının tavsiye edildiği belirtilmiştir. Ayrıca, sosyal ağların kompleksliği arttıkça hataya yatkınlığın arttığı gözlenmiştir.

Song vd. [103] hataya yatkınlık tahmini modelleri için genel bir çerçeve model oluşturmaya çalışmışlardır. Farklı veri kümeleri için farklı öğrenme düzenleri kullanılması gerektiğini öne sürmüşlerdir. Performans ölçümü için denge ve AUC ölçümleri kullanılmış, eğitim Naive Bayes, J48 ve OneR algoritmaları ile yapılmıştır.

Ostrand vd. [67] yaptıkları araştırmada dosyalar üzerinde değişiklik yapan geliştirici sayılarının hataya yatkınlığı nasıl etkilediği araştırılmıştır. Bir dosya üzerinde hangi geliştiricinin çalıştığı bilgisi tek başına bir anlam ifade etmeyebilirken kümülatif değişiklik yapan geliştirici sayısı bilgisinin faydalı olduğu bildirilmiştir.

Tan vd. [56] Promise havuzundan aldıkları veri kümelerini kullanarak, hataya yatkınlık tahmininin sınıflar yerine sınıfların kosinüs benzerliklerine göre kümelenmesi ile oluşturulan kümeler üzerinden yapılmasının daha iyi sonuç verdiğini göstermiştir. Bu yöntem özellikle logaritmik regresyon kullanımında doğru tahmin oranını %31.6'dan %99.2'ye çıkarmıştır. Ancak, deney sonucunda elde edilen yanlış alarm oranındaki değişim verilmemiştir. Hassasiyetin %73.8'den %91.6'ya artması yanlış alarmların ilk durumda da düşük olduğu veya yanlış alarm sayısında da doğru tahmin artışına yakın oranda artış görüldüğü anlamına ge-

lebilir. Bu çalışmada test eforunda da %6 iyileşme gözlemlenmiştir.

Wang vd. [57] yazılım hataya yatkınlık tahmini çalışmalarında kullanılması gereken vektör boyutunun kestirimini konusunda bir araştırma yapmıştır. Başlangıçta 208 metrik içeren kümeler kullanılan bu araştırma sonucunda sadece 3 metrik kullanılması durumunda bile yeterince verimli sonuçlar alınabildiği, metriklerin %98.5'inin atıldığı durumda tahmin modellerinin performansının arttığı gözlemlenmiştir.

Gao ve Khoshgoftaar [61] sınıf etiket dağılımlarının dengesiz olduğu veri kümelerinin kusur kestirimi performanslarını iyileştirmeye çalışmışlardır. Kusur kestirimi çalışmalarında genelde 2 sınıf bulunur: kusurlu ve kusursuz. Bu sınıfların yakın oranlarda bulunması beklenebilir ancak çoğu durumda sınıf oranları arasında denge bulunmamaktadır. Bu çalışmada örnekleme ve özellik seçimi yöntemlerinin farklı kombinasyonları kullanılmış, bu yöntemler arasında örneklenmiş veri üzerinde özellik seçimi yapılıp asıl veri üzerinde hataya yatkınlık tahmini yapılmasının daha iyi performans verdiği gözlenmiştir.

Paikari vd. [66] durum tabanlı çıkarsama kullanan hataya yatkınlık tahmini yöntemleri üzerinde benzerlik fonksiyonu, en yakın komşuluk sayısı, ağırlıklandırma ve çözüm algoritması seçimlerinin optimizasyonunu sağlamışlardır. Ayrıca, içeriğe özel kuralların farklı veri kümelerinde de kullanılması denenmiş ancak başarı gözlenmemiştir.

Li vd. [75] örnekleme ile birlikte yarı denetimli makine öğrenmesi modeli kullanımının, küçük örnek sayılarında da, yeterince iyi sonuç verdiğini göstermişlerdir. Java tabanlı 6 proje üzerinde ortalama %67 denge değerine ulaşmışlardır.

Çatal vd. [79] Eclipse verisi üzerinde Naive Bayes algoritması kullanarak, kusur kestirimi çalışmalarında kullanmak üzere RUBY adlı bir araç geliştirmişlerdir.

Kpodjedo vd. [80] CK metriklerinin özel bir kümesi olan, tasarım gelişim metrikleri adını verdikleri bir metrik seti öne sürmüş ve yaptıkları deneylerde farklı ölçüm birimleri kullanarak öne sürdükleri metrik setinin özellikle masraf iyileştirmesi sağladığını göstermişlerdir.

Shihab vd. bir başka çalışmasında [87] kusurları kategorilendirip yüksek öncelikli olanları bulmaya yönelik bir çalışma ortaya koymuştur. İki önemli kusur kategorisinde %42 ve %55 lik efor önlemesi sağlayabilmişlerdir.

Bird vd. [93] yazılım hata sayısı ve kod iten kişilerin değişimi ilişkisi üzerine yaptıkları araştırmada, bir kod parçası üzerinde yeterli yetkinliği olmayan kişilerin değişiklik yapmasının hata oranını arttırdığını göstermiş, bu gibi durumlarda kod gözden geçirmelere daha önem verilmesi gerektiğini bildirmişlerdir.

Mısırlı vd. [95] farklı hata kategorileri için farklı metriklerin kullanılabilirliğini incelemiş, sonuç olarak ağ ölçümlerinin, fonksiyonel test sırasında ve sahada bildirilen kusurlar için, kod metriklerinin sistem testi sırasında bildirilen kusurlar için daha iyi sonuç verdiğini göstermişlerdir.

Kim vd. [96] veri kümelerindeki gürültünün hataya yatkınlık tahmini performansına etkisini araştırmış, %20-%35 yanlış pozitif ve yanlış negatif içeren veri kümelerinde tahmin

performansının önemli ölçüde düştüğünü gözlemişlerdir. Ayrıca gürültülü verileri tanıyıp elimine edecek bir algoritmayı da tanıtmışlardır.

Jeet vd. [98] NASA veri kümeleri üzerinde Bayes ağı uygulayarak bir hataya yatkınlık tahmini modeli oluşturarak %80 doğruluk elde etmişlerdir. Modeli statik kod metriği veya kod değişimi gibi bu alanda kullanılan özellikler yerine, efor tahmini için kullanılan özellikler ile oluşturmuşlardır.

Jureczko'nun araştırmasında [100] 22 farklı projeden alınan 86 veri kümesi üzerinde statik kod metrikleri, nesneye yönelik metrikler ve kod değişim metriklerinin hataya yatkınlık tahminindeki performansları değerlendirilmiştir. Bu işlem metrik kümelerini birbiriyle karşılaştırmak şeklinde değil, her metriği ayrı ayrı değerlendirmek suretiyle yapılmıştır. Her metrik kümesinden bazı metrikler diğerlerine göre daha belirgin sonuç vermiştir. Ancak, değerlendirme için kullanılan regresyon yönteminin bu amaç için yeterli olmadığı Jureczko tarafından da kabul edilmiştir.

Rahman vd. [73] 9 Apache projesi üzerinde yaptıkları çalışmada sonuçların masraf odaklı değerlendirilmesi durumunda aslında projeler arası hataya yatkınlık tahmini performansının proje içi tahmin performansıyla pek farklılık göstermediğini ortaya koymuşlardır. Dikkat çekmek istedikleri nokta hataların çoğunun kodun küçük bir kısmında olduğu ve azaltılması gereken eforun bu küçük kısmın incelenmesinde olduğudur.

Wang ve Zhang [84] sorun bildirimleri üzerinde yaptıkları çalışmalarda, sorunların çözülme aşamasında geçtiği adımları modelleyerek hata sayısını tahmin etmeye çalışmışlardır.

Lu vd. [89] yarı denetimli makine öğrenmesi ve boyut azaltma tekniklerini kullanarak Rastgele Orman'dan daha iyi bir hataya yatkınlık tahmini oranı yakaladıklarını söylemişlerdir.

Wang vd. [97] COQUALMO adlı kalite modelindeki [113] nitel metrikler ile yazılım süreci ile ilgili test senaryo sayısı, takım boyutu, yazılım boyutu, efor gibi nicel metrikleri birleştirerek bir hataya yatkınlık tahmini modeli ortaya çıkarmışlardır. Deneyler 21 ticari projenin veri kümesi üzerinde gerçekleştirilmiştir.

Wang ve Yao [106] dengesiz sınıf dağılımları üzerine yaptıkları çalışmada örnekleme ve eşik değeri optimizasyonu kullanımının bu tip veri kümelerinde tahmin performansını arttırdığını göstermişlerdir.

Giger vd. 21 açık kaynaklı Java tabanlı projeden aldıkları veri kümeleri üzerinde yaptıkları çalışmada [108] statik kod metrikleri ile kod değişim metriklerinin performanslarını karşılaştırmışlardır. Çalışma sonucunda kod değişim metriklerinin aynı amaçlı diğer çalışmalara benzer şekilde daha iyi sonuç verdiği gözlenmiştir. Ancak, bu çalışmada dosya veya sınıf seviyesindeki metrik kullanımı ile bir karşılaştırma yapılmamıştır.

Malhotra açık kaynaklı projeler üzerinde yaptığı çalışmada [111] nesneye yönelik tasarım metriklerini kullanarak %83 AUC değerine ulaşmıştır. Ancak, bu çalışma başka metrik setleri kullanılmayıp sadece 1 veri kümesi kullanılmıştır.

He vd. [60] kaynak dosyalar arasındaki bağımlılıkları kullanarak dosya ağları oluşturmuş

ve bu sayede hataya yatkınlığı kestirebilmeye çalışmışlardır. Tomcat ve Ant uygulamalarından elde edilen veri kümeleri üzerinde yaptıkları deneylerde %54.7 ve %63.8 tahmin başarısı elde etmişlerdir. Ancak bu çalışmanın bir eksikliği, elde ettikleri sonuçları aynı uygulamalardan çıkarılmış başka bir metrik seti ile karşılaştırmamalarıdır.

Tassé'nin çalışmasında [70] daha kısa süre, özellikle geliştirme iterasyonları içinde, kullanılabilecek bir kusur kestirim modelinin geliştirilmesi amaçlanmıştır. Bunun için JEdit uygulamasının kod versiyonlama sisteminden kod değişimi ile ilgili metrikler çıkarılıp kullanılmıştır. Sonuçlar istenilen düzeyde olmamakla birlikte geliştirilmeye uygun görülmektedir.

Herbold'un araştırmasında [72] projeler arası tahmin denilen, bir projenin veri kümesi üzerinde oluşturulan model ile başka bir projenin veri kümesi kullanılarak test yapılması işleminin iyileştirilmesine yönelik bir çalışma yapılmıştır. Promise'den alınan 14 projedeki 44 veri seti ile yapılmış çalışma sonucunda en yakın komşu algoritmasına dayalı bir kümeleme yapıldığı durumda, komşuluk sayısı arttıkça hataya yatkınlık tahmini başarısının arttığı görülmüştür.

Canfora vd. [76] Promise kaynaklı 10 veri kümesi üzerinde logaritmik regresyon ve genetik algoritma tabanlı bir çalışma yaparak projeler arası tahmin performansını iyileştirmeye çalışmışlardır. Menzies'in sonuçları [53] ile aralarında bir karşılaştırma yapıp ortalama %22 daha uygun maliyetli bir model oluşturduklarını öne sürmüşlerdir.

Umar [85] yazılım test sürecinden alınan süreç metrikleriyle hataya yatkınlık tahmini çalışması yapmış ve 5 parametre ile %90 korelasyon yakalamıştır.

Nam vd. [92] projeler arası hataya yatkınlık tahmini problemi için çalışmış, TCA adını verdikleri bir yöntemle bir proje için geliştirilen modeli diğer projeye aktarmaya çalışmışlardır. Farklı veri kümeleri için ortalama %10 iyileştirme sağlamışlardır.

Peters vd. [54] projeler arası hataya yatkınlık tahmini problemini incelemiş, metriklere uygulanacak bir filtreleme yöntemi öne sürerek, bu sayede projeler arası tahmin performansının arttırılabileceğini göstermişlerdir.

Rahman ve Devanbu [94] yaptıkları araştırmada statik kod metrikleri ile kod değişimi metriklerini karşılaştırmış, farklı algoritmalar ve performans kriterleri kullaranak kod değişim metriklerinin daha iyi sonuç verdiğini göstermişlerdir.

Grbac vd. [104] dengesiz sınıf dağılımlarının hataya yatkınlık tahminine etkisini incelemişlerdir. NASA veri kümelerinde yaptıkları çalışmada sınıflar arası dengesizlik %80'i bulduğunda alınan sonuçların stabil olmadığını ortaya koymuşlardır.

Menzies vd. [53] hata ve efor tahmininde farklı kaynaklardan verilerin birlikte kullanılıp kullanılamayacağını araştırmışlardır. Farklı kaynaklara ait veri kümelerinin birlikte kullanılması amaçlanıyorsa en yakın komşuluklarına göre kümelenerek oluşturulan kümeler içinde yapılması gerektiğini öne sürmüşlerdir.

Çatal ve Diri [109] NASA kaynaklı veri setleri üzerinde yaptıkları çalışmada yarı denetimli bir model üzerinde çalışarak metot seviyesinde metrikler kullanmışlardır. Metrikler üze-

rinde eşik değeri tabanlı bir yöntemle filtreleme yapılmış, sınıf bilgisi eksik örneklerin bu sayede etiketlenmesi sağlanmıştır. Bundan sonra yapılan hataya yatkınlık tahmini işleminin benzer başka bir çalışma ile karşılaştırılması neticesinde daha iyi performans verdiği gözlenmiştir.

BÖLÜM 3

WEB METRİKLERİ

Bu bölümde, web uygulamalardan hangi metrik türlerinin çıkarılabildiğini açıklayacağız. Bu amaçla, öncelikle web uygulamalarında ortak bulunan hata kalıpları tanıtılacaktır. Daha sonra web uygulamalarına özel çıkarılabilecek metrikler ve bunların değerleri araştırılacaktır.

3.1 Web Uygulama Hataları

Araştırma sorularımıza cevap verebilmek için önce web uygulamalarının özelliklerini inceledik. Bu amaçla öncelikle web uygulamalarında karşılaşılan hataların analiz edilmesi gerekir, bunu yapmadan özel metrikler tanımlayamayız. Web uygulamalarındaki hatalar ile ilgili birçok çalışma vardır [19], [44], [47], [48]. Bu çalışmalarda küçük farklar olmasına rağmen, sonuçları yaklaşık olarak aynı doğrultudadır. Bu çalışmalar arasında Marchetto'nun çalışması [44], web uygulaması hatalarının en kapsamlı taksonomisini sağlamaktadır. Marchetto'nun taksonomisi Çizelge 3.1'de verilmiştir.

3.2 Önerilen Metrikler

Hatalılık tahmini çalışmaları için metrikleri çıkarmak için otomatik araçlar kullanılmalıdır. Aksi halde bu işlem proje üyelerinin değerli vakitlerini çalar ve hataya yatkınlık tahmini çalışmalarının kaynakları tasarruf etme amacına uymaz.
Bazı sorun türlerini yazılım metriklerine dönüştürmek zordur veya mümkün değildir. Örneğin, kullanıcı kimlik doğrulaması sırasında bir hata tespit edilebilir, ancak otomatik bir aracın böyle yüksek düzeyde bir senaryoyu tanımlaması mümkün değildir. Bir otomatik analiz aracı sadece bazı parametrelerin sunucuya gönderildiğini ve oturum erişimi yapıldığını anlayabilir ve veritabanı sorguları yapıldığını tanıyabilir. Anlaşılabilen bu noktalar da kaynak kodundan bulunabilir ve özel bir metrik kümesi oluşturulabilir.
Bu çalışmanın amacı, web uygulamalarının özelliklerinden yararlanarak uygulamaların hataya yatkınlığı tahmini performansını iyileştirmek için yeni bir metrik kümesi önermektir.

Çizelge 3.1 Web uygulamalarındaki hataların sınıflandırılması [44]

Karakteristikler	**Hata Türleri**
Çok katmanlı mimari	Tarayıcı uyumsuzluğuna bağlı hatalar Geri düğmesine bağlı hatalar Eklentilere bağlı hatalar Sayfaların istemci tarafında dinamik oluşturulmasındaki sorunlar Sunucu tarafındaki sayfaların girdilerindeki hatalar Dosya sistemine erişimdeki sorunlar Sunucu ortamıyla ilgili sorunlar Girdi verisinin karakter kodlamasına ait hatalar Form oluşturma hataları Veritabanına erişim veya veritabanı yönetimi hataları Bilgi arama işlemindeki hatalar Bilginin önbellekte hatalı tutulması
Arayüz	Tarayıcı tarafında HTML yorumlanmasıyla ilgili hatalar DOM nesnelerini değiştirirken oluşan hatalar Frame senkronizasyonu hataları Frame yükleme hataları Karakter kodlaması hataları Diller arasında istenmeyen sıçrama
Oturum tabanlı	Oturum senkronizasyonu hataları Oturum nesnelerinin kalıcılığındaki hatalar Çerezleri değiştirirken oluşan hatalar
Linkli yapı	Web sayfalarının entegrasyonuyla ilgili hatalar Dinamik URL oluştururken oluşan hatalar Ulaşılamayan kaynaklardan oluşan hatalar Bulunmayan kaynaklardan oluşan hatalar
Protokol tabanlı	Şifrelenmiş iletişim kullanımıyla ilgili hatalar Vekil sunucuların kullanılan belirli protokolleri desteklememesi
Kimlik doğrulama	Kullanıcı kimlik doğrulama hataları Hesap yönetimi hataları Kaynakların izinsiz erişimi/kullanımı Yönetim rolündeki hatalar

Statik kod ölçümleri, hatalılık tahmini çalışmalarında yaygın olarak kullanılır ve genel olarak her tür programlama dili için iyi performans gösterir. Ayrıca otomatik araçlar kullanılarak kolayca elde edilebilmektedirler. Bu çalışmada statik kod metriklerini web alanına özel olarak genişleterek daha yüksek seviyedeki hata durumlarını karşılayabilmek amaçlanmıştır.

Web uygulamalarının hata özelliklerini karşılayabilmek için Çizelge 3.2 ve 3.3'deki metrikleri önerdik. Önerilen metrikler farklı bölümlerde ele alınacak ve faydaları deneylerle ayrı ayrı incelenecektir.

Çizelge 3.2 Önerilen metrik kümesi (a)

Sunucu Metrikleri	Oturum Parametresi Kullanımı (Has_Session) Oturumdan Okuma Sayısı (Session_Reads) Oturuma Yazma Sayısı (Session_Writes) Çerez Kullanımı (Cookies) İstek Başlığı Kullanımı (Header_Access)	İstek Parametresi Kullanımı (Has_Param) İstek Parametrelerine Erişim Sayısı (Param_Reads) Bağlam Değişimi (Context_Switches) Veritabanı Sorgu Sayısı (DB_Query) Veritabanı Sorgu Kullanımı (DB_Query_Binary)
HTML metrikleri	Toplam / Tekil etiket sayısı (total_tags / unique_tags) Maksimum / Toplam / Ortalama derinlik (max_depth / total_depth / avg_depth) Toplam İçerik (total_text) Girdi sayısı (inputs) Script sayısı (scripts) Sayfada tanımlanan scriptler (in_page_scripts) Yüklenen stil etiketi sayısı (css_included) Link sayısı (anchors)	Toplam / Tekil / Ortalama özellik sayısı (total_attrs / unique_attrs / avg_attrs) Toplam Yorum Sayısı (total_comments) Form sayısı (forms) Çerçeve sayısı (frames) Dışarıdan yüklenen scriptler (external_scripts) Stil etiketi sayısı (style_tags) Sayfada tanımlanan stil etiketi sayısı (inline_css)

Çizelge 3.3 Önerilen metrik kümesi (b)

CSS metrikleri	Renk sayısı (colors)	Kompleks seçici sayısı (complexSelectors)
	Öznitelik bazında kompleks seçici sayısı (complexSelectorsByAttribute)	Tekrarlanan özellik sayısı (duplicatedProperties)
	CSS ifadeleri sayısı (expressions)	IE eski versiyonları için düzeltmeler (oldIEFixes)
	Zorlanan ifade sayısı (importants)	Birden fazla sınıflı seçici sayısı (multiClassesSelectors)
	Eski tarayıcıya özel ifade sayısı (oldPropertyPreFixes)	Nitelikli seçici sayısı (qualifiedSelectors)
	Kuralın özgüllüğü (specificity)	ID ortalama / toplam özgüllüğü (specificityIdAvg / specificityIdTotal)
	Sınıf ortalama / toplam özgüllüğü (specificityClassAvg / specificityClassTotal)	Etiket ortalama / toplam özgüllüğü (specificityTagAvg / specificityTagTotal)
	Özellik / sınıf / ID / etiketlere göre seçiciler (selectorsByAttribute / selectorsByClass / selectorsById / selectorsByTag)	Sahte seçicilerin sayısı (selectorsByPseudo)
	Kategorize edilmiş özgüllük (specificityCategory)	Genel seçici sayısı (universalSelectors)
	Seçici sayısı (selectors)	Tanım sayısı (declarations)
	Uzunluk (length)	

BÖLÜM 4

HATA TAHMİN MODELİ VE VERİ KÜMELERİ

Bu bölümde araştırma boyunca araştırma sorularına cevap vermek için kullanılan yöntemler açıklanmaktadır. Veri kümesi oluşturma süreci, algoritmalar, performans ölçütleri alt başlıklar halinde sunulmaktadır.

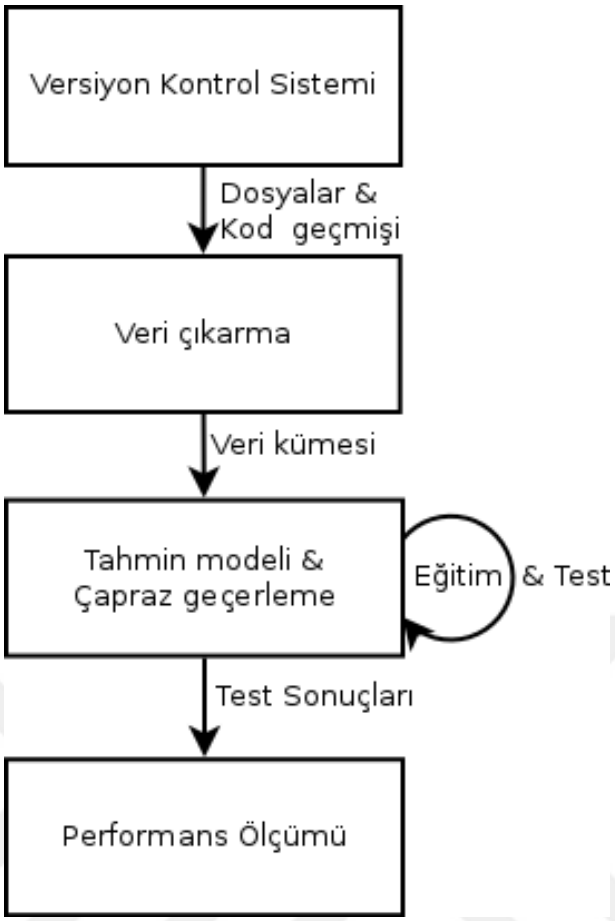
4.1 Mimari

Makine öğrenme algoritmaları, hatalara neden olduğu bilinen program özelliklerinin modellerini oluşturmak için kullanılmıştır. Hatalılık tahmini modelleri, öğrenme tabanlı modeller kullanılarak hazırlanmıştır. Ayrıca, yazılım modüllerinin kusurluluğunu öngören bir öğrenme sistemi tasarlanmış ve önerilmiştir. Bu sistemin grafik gösterimi Şekil 4.1'de verilmiştir. Kod değişim geçmişi, arızalı modülleri tanımlamak için kullanılmıştır.

Sistem üç farklı makine öğrenme algoritması kullanılarak veri kümesiyle eğitilmiştir, veri kümesindeki sıralama etkisinden kaynaklanacak önyargıyı ortadan kaldırmak için 10 katlı çapraz geçerleme kullanılarak test edilmiştir. Test işlemi yapıldıktan sonra, tahmin sonuçları elde edilmiştir. Çıktı hangi dosyaların kusurlu olduğu ve hangi dosyaların kusursuz olarak tahmin edildiğini göstermektedir. Ardından, sistemin performansı Bölüm 4.3'de verilecek olan hata yakalama (pd), yanlış alarm (pf), denge ölçütleri kullanılarak değerlendirilmektedir.

4.2 Algoritmalar

Bu kısımda hataya yatkınlık tahmini modellerinde kullanılan algoritmalar kısaca açıklanmaktadır. Algoritmaların uygulanması Weka kütüphanesi (v3.6.13) [114] kullanılarak yapılmıştır.



Şekil 4.1 Öğrenme tabanlı hataya yatkınlık tahmini sistemi mimarisi.

4.2.1 Rastgele Orman

Rastgele Orman, karar ağaçları oluşturma yöntemini kullanan tahmin edicilerin bir kombinasyonu olup, her ağaç bağımsız olarak örneklenmiş rasgele bir vektörün değerlerine ve ormandaki tüm ağaçlar için aynı dağılıma bağlıdır. Ormanlar için genelleme hatası ormandaki ağaçların sayısı arttıkça bir sınıra kadar yakınsamaktadır. Ağaç sınıflandırıcılarının bir ormanının genelleme hatası, ormandaki ağaçların mukavemetine ve aralarındaki korelasyona bağlıdır. Her bir düğümün bölünmesi için rastgele bir özellik seçimi, Adaboost [115] yöntemine göre daha iyi olan ancak gürültüye göre daha sağlam olan hata oranlarını verir. Dahili tahminler hata, güç ve korelasyonu izler ve bunlar bölme işleminde kullanılan özelliklerin sayısının artırılmasına tepki göstermek için kullanılır. Değişkenin önemini ölçmek için dahili tahminler de kullanılır. Bu fikirler, gerileme için de geçerlidir [116].

Rastgele Orman algoritması tahminde etkili bir araçtır. Büyük Sayılar Kanunu gereği aşırı uyum göstermezler. Doğru rasgele çeşitliliği enjekte etmek, onları sınıflandırma ve regresyon bakımından doğru araç haline getirir. Bunun yanında bireysel tahmin edicilerin gücü ve korelasyonları açısından, Rastgele Orman'ın tahmin edebilme yeteneği hakkında fikir verir. Torba dışı tahmini (OOB) kullanmak sağlamlık ve korelasyon yönünden teorik

olan değerleri harekete geçirir [116].

Rastgele Orman'ın uygulamasında Weka'da kullanılan varsayılan değerler Çizelge 4.1'deki gibidir.

Çizelge 4.1 Rastgele Orman için varsayılan değerler

Parametre	**Değer**
maxDepth	0
numFeatures	0
numTrees	100
seed	1

4.2.2 Naive Bayes

Naive Bayes Bayes Teorisine dayanan olasılıksal bir sınıflandırıcıdır [117]. Naive olarak adlandırılması çalışma mantığını basitleştiren iki önemli varsayımı kullanmasından gelir [118]. Öngörülen özelliklerin koşullu olarak birbirinden bağımsız olduğunu ve hiçbir gizli özniteliğin tahmin sürecini etkilemediğini varsaymaktadır. Bu varsayımlar sınıflandırıcıyı basitleştirir. Gerçek dünya verileri, birbiriyle ilişkili nitelikler içerecektir. Naive Bayes özniteliklerin bağımsız olması gerektiğini varsaydığı için bu ilişkiler Naive Bayes sınıflandırıcısı tarafından dikkate alınmaz ve yanlış sınıflandırmalara neden olabilir. Ancak, Domingos'un çalışmasında [119] gösterildiği üzere bu bağımsızlık varsayımı çoğu durumda bir soruna yol açmamaktadır. Bu da Naive Bayes yönteminin diğer karmaşık sınıflandırıcılar kadar iyi çalışmasını açıklamaktadır.

Bayes Teoremi'nde, bir sonraki değerin yeni kanıtların eski inançları nasıl etkilediği ile belirlendiğini belirtilmektedir [117]. Resmi gösterimi (4.1)'de verilmektedir:

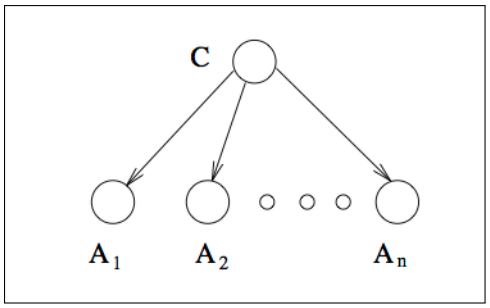
$$P(C|A_1, A_2...A_n) = \frac{P(C)}{P(A_1, A_2...A_n)} \times \prod_{i=1}^{n} P(A_i|C) \tag{4.1}$$

C sınıfının olasılığının şu anki koşullar A_i, C sınıfı için olan önceki olasılıklar $P(C)$, ve $\prod_{i=1}^{n} P(A_i|C)$ olasılığı ile belirlendiği gösterilmektedir.

Naive Bayes'in uygulamasında Weka'da kullanılan varsayılan değerler Çizelge 4.2'deki gibidir.

Çizelge 4.2 Naive Bayes için varsayılan değerler

Parametre	**Değer**
useKernelEstimator	false
useSupervisedDiscretization	false



Şekil 4.2 Naive Bayes ağının yapısı [121].

4.2.3 Bayes Ağı

Bayes ağları ile değişkenlerin birbirinden bağımsız olduğunu düşünen Naive Bayes'in eksikliğinin doldurulması önerilmiştir [120]. Bu ağlar, birleşik olasılık dağılımının rastgele bir değişken kümesi üzerinden verimli bir şekilde gösterilmesine izin veren yönlü dairesel olmayan graflardır. Grafın her kesişim noktası rasgele bir değişkeni temsil eder ve kenarlar değişkenler arasındaki korelasyonlardır.

Bir Naive Bayes sınıflandırıcısı, bir Bayes Ağı olarak temsil edildiğinde, Şekil 4.2'de gösterilen basit yapıya sahip olur. Bu ağ Naive Bayes sınıflandırıcısının arkasındaki ana varsayımı yakalar; diğer bir deyişle, sınıf değişkeninin (ağdaki kök) verildiği durumda, her öznitelik (ağdaki her yaprak) özniteliklerin geri kalanından bağımsızdır [121].

Bayes Ağı'nın uygulamasında Weka'da kullanılan varsayılan değerler Çizelge 4.3'deki gibidir.

Çizelge 4.3 Bayes Ağı için varsayılan değerler

Parametre	**Değer**
estimator	SimpleEstimator
searchAlgorithm	K2
useADTree	false

4.2.4 Çapraz Geçerleme

Çapraz geçerleme veriyi n sayıda birbiriyle yaklaşık olarak aynı boyutta sabit kümeye bölen istatistiksel bir yöntemdir [7]. Bir küme sınıflandırıcının test edilmesi için ayrılırken kalan $n-1$ küme sınıflandırıcının eğitiminde kullanılır. Bu işlem n kümenin her biri test için kullanılana kadar tekrarlanır. Bu işleme n-katlı çapraz geçerleme adı verilir. Öğrenici için n farklı sayıdaki doğruluk/hataya yatkınlık tahminlerinin ortalaması alınır, böylece genel bir doğruluk/hataya yatkınlık tahmini elde edilmiş olur.

Bu işlem katmanlaştırma ile birleştirilebilir, böylece her kümenin verinin geneli ile aynı

karakterde olduğu (sınıf oranlarının kaybolmadığı) garanti edilebilir [7]. Bu yönteme katmanlaştırılmış n-katlı çapraz geçerleme adı verilir.

Verileri sıralanmasının ya da sınıflandırıcıların varyasyonunun geçersiz ya da güvenilmez olmasına engel olmak için öğrenme işlemi 10 kat çapraz geçerlilikle sonuçlanır, 10 kat çapraz doğrulama işlemini 10 kez tekrar etmesi önerilir, böylece öğrenme algoritması bir veri kümesi üzerinde 100 kez çalıştırılmış olur. Bu adımdan sonra öğrenme algoritması için başarı oranını elde etmek için sonuçların ortalaması alınır.

4.2.5 Korelasyon Özellik Seçimi

Makine öğrenmesindeki temel problemlerden biri, belirli bir görev için bir sınıflandırma modeli oluşturacak temsili bir özellik vektörünü tanımlamaktır. Korelasyon Özellik Seçimi (CFS), korelasyon tabanlı bir yaklaşımla makine öğrenimi için özellik seçimi problemini ele alır. CFS algoritmasının ardındaki ana fikir şudur: "İyi bir özellik alt kümesi, sınıfı birbiriyle (öngörücü değil), korelasyonlu (prediktif) özelliklere sahip olan bir özelliktir." [122]

$$M_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}} \tag{4.2}$$

Eşitlik 4.2 CFS algoritmasının çekirdeğini oluşturur. Bu eşitlikte M_S, k adet özellik içeren S alt kümesinin "faydasıdır". $\overline{r_{cf}}$ ise bütün özellik-sınıflandırma korelasyonlarının ortalama değeridir. Son olarak $\overline{r_{ff}}$ ise bütün özellik-özellik korelasyonlarının ortalama değeridir. CFS bu eşitliği uygun bir korelasyon ölçütü ve sezgisel arama stratejisi ile birleştiren bir algoritmadır. CFS, yapay ve doğal veri setleri üzerindeki deneylerle değerlendirilmiştir. Denemeler, CFS'nin ilgisiz, gereksiz ve gürültülü özellikleri hızlıca tanımladığını ve elediğini; alakalı özellikleri de, alaka düzeyi diğer özelliklere kesinlikle bağlı olmadıkça, tanımladığını göstermiştir. Ayrıca, tüm özellik setini kullanarak sınıflandırma hassasiyetini eşit veya daha iyi tutarak özelliklerin yarısından fazlasını ortadan kaldırabilir.

4.2.6 Mann-Whitney U Testi

Mann-Whitney U Testi (Mann-Whitney-Wilcoxon olarak da bilinir) iki farklı örnek kümesinin aynı dağılıma ait olup olmadığını test etmek için kullanılan parametrik olmayan bir testtir. t-testine alternatif olarak kullanılmaktadır. t-testingde verinin normal dağılıma uyduğu kabul edilir, ancak hataya yatkınlık tahmini kümelerinde böyle bir kabule gitmemiz doğru olmaz. Bu gibi durumlar için parametrik olmayan testlerin kullanımı daha uygun olmaktadır. Mann-Whitney U Testini çalışmamız için iyi bir aday yapan özelliği, ayrık ya da sürekli sayısal testler için kullanılabilen parametrik olmayan bir test olmasıdır [123].

Bu test U adında bir istatistik hesaplar. U değerini hesaplayabilmek için veri önce artan sırada sıralanır. Her veriye sıralamada geldiği derece kadar bir puan verilir. İki küme için de kendi içlerinde verilerinin sıralama puanlarının toplamı alınır. i. küme için R_i puanların toplamı ve n_i popülasyon sayısı dersek, U değeri Eşitlik 4.3'deki gibi hesaplanır.

$$U_1 = n_1 \times n_2 + \frac{n_1 \times (n_1 + 1)}{2} - R_1 \tag{4.3}$$

$$U_2 = n_1 \times n_2 + \frac{n_2 \times (n_2 + 1)}{2} - R_2 \tag{4.4}$$

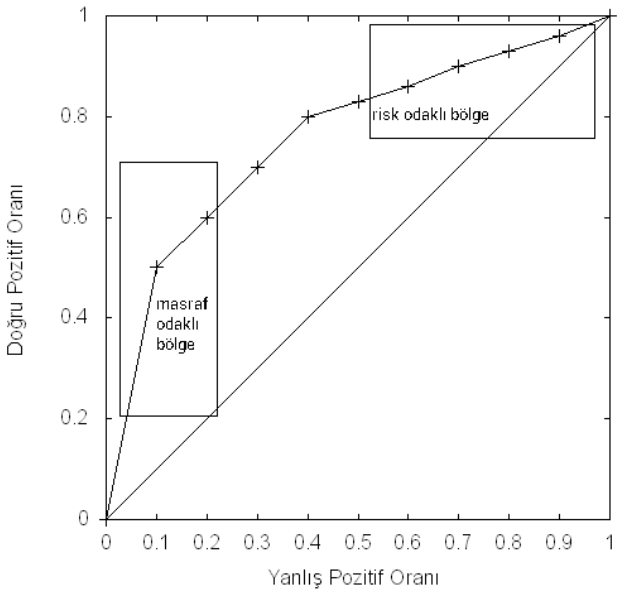
Daha sonra en küçük U değeri U dağılımı tablosundaki istenen güven aralığına karşılık gelen örnek boyutu değeri ile mutlak değer olarak karşılaştırılır. Eğer hesaplanan U değeri tablodakinden küçükse H_0 reddedilir.

Araştırma boyunca elde edilen bütün sonuçlar için, kullanılan yeni metrik setleri veya algoritmaların istatistiksel olarak bir fark yaratıp yaratmadığının anlaşılabilmesi için Mann-Whitney U testi kullanılmıştır. Bu sayede önerilen model veya algoritmaların faydalarının kanıtlanması amaçlanmıştır.

4.3 Performans Ölçümü

Bu araştırmada, yazılım hatalılık tahmini için kurulan modellerin performanslarının değerlendirilmesinde literatürdeki diğer hatalılık tahmini çalışmalarında da yaygın olarak kullanılan [8], [11], [15], [64] yakalama olasılığı (pd) ve yanlış alarm olasılığı (pf) ölçütleri kullanılmıştır. Pd (aynı zamanda hassasiyet olarak da bilinir) tahmin edicinin gerçekten hataya yatkın modülleri bulma ne kadar başarılı olduğunu gösterirken pf değeri de yanlış alamların, yani hataya yatkın olmayan modüllerin yanlış işaretlenmesinin oranını gösterir.

Tahmin edicilerin pd değerini yükseltirken pf değerini düşürmesi beklenir. Bu nedenle (pd, pf) çiftinin ideal noktası olan (1,0) değerine en çok yaklaşan tahmin model performansına ulaşmaya ihtiyacımız bulunmaktadır. Bu ideal duruma pratikte çok az rastlanır. Çünkü pf değeri pd ile birlikte artma eğilimi gösterir. Bu nedenle genelde pd-pf değerleri arasında bir uzlaşma aranır. Optimum (pd, pf) çiftlerini seçebilmek için "denge" adı verilen, üçüncü bir performans ölçütü kullanırız. Denge bir ROC eğrisi üzerindeki bir noktadan pd=0, pf=1 noktasına olan öklid uzaklığı olarak tanımlanabilir [15]. Denge ölçütü tahminlerimizin ideal duruma ne kadar yakın olduğunu görmemizi sağlar. Farklı (pd, pf) değerlerine sahip tahmin ediciler aynı denge değerine sahip olabilir. Bu durum aynı dengedeki bütün tahmin edicilerin pratikte aynı kullanım değerine sahip olduğu anlamına gelmez.



Şekil 4.3 ROC eğrisinin bölümleri [11]

Doğru tahmin bir modelin başarısını belirlemek için önemli bir etkendir ancak yanlış tahmin de oldukça önemlidir. Bu durum Şekil 4.3'de gösterilmiştir. Risk odaklı bölgedeki tahmin modelleri yüksek pd değerine sahip olmakla beraber pf değerleri de oldukça yüksektir. Bu durum hata içermeyen çok sayıda dosyanın hatalı olarak işaretlenmesi anlamına gelip, gereğinden fazla dosyanın incelenmesi sonucunu doğurur. Bu da test aşamasının masrafının artmasına neden olmaktadır. Hatasızlığın çok önemli olduğu projeler için bu kabul edilebilir bir durum olmakla beraber projelerin çoğu bu kategoride yer almamaktadır. Masraf odaklı bölge orta-düşük pd'ye ve çok düşük pf'e sahiptir. Bu bölgeye düşen tahmin modelleri sınırlı kaynaklara sahip projeler için daha kullanışlıdır [11]. Projelerin başarı kriterleri arasında bütçe ve kaynak kullanımı önemli etkenler sayılmaktadır. Yazılım hataya yatkınlık tahmini de bütçe ve kaynak kullanımının azaltılması amacıyla yapılan bir çalışmadır. Bu nedenle geliştirdiğimiz tahmin modellerinde sınırlı kaynaklara sahip projeler göz önüne alınarak ilerlenilmiştir. Aynı denge değerine sahip modelleri değerlendirirken (pd, pf) çiftleri arasında düşük pf değerine sahip olan modeller kabul edilmiştir.

$$pd = \frac{TP}{TP + FN} \tag{4.5}$$

$$pf = \frac{FP}{FP + TN} \tag{4.6}$$

$$denge = 1 - \frac{\sqrt{pf^2 + (1 - pd)^2}}{\sqrt{2}} \tag{4.7}$$

Bu değerler Çizelge 4.4'deki değerler ve Eşitlik (4.5), (4.6), ve (4.7) kullanılarak hesaplanır. Çalışmanın her adımında kontrollü deney yaparak modellerde değiştirilen etmenlerin

Çizelge 4.4 Hata matrisi

		Gerçek	
		Kusurlu	Kusursuz
Tahmin edilen	Kusurlu	TP	FP
	Kusursuz	FN	TN

(veri kümesi veya algoritma) denge değerleri kontrol kümesinde çıkan değerler ile karşılaştırılmıştır. Sonuçlardaki farkların istatistiksel olarak belirgin olup olmadığı %95 güven aralığında Mann-Whitney U Testi uygulayarak kontrol edilmiştir.

4.4 Veri Kümeleri

Deneylerimizde veri kümesi olarak 11 adet açık kaynaklı PHP projesinden ve 5 adet HTML & CSS projesinden çıkarılmış metrik kümeleri 2 farklı grupta kullanılmıştır. Kullanılacak projeleri belirlerken geniş bir kod tabanı olmasına ve uzun bir geliştirme geçmişine sahip olmasına dikkat edilmiştir. Bütün projelerin kod tabanları GitHub üzerinde bulunmaktadır. Bu projelerden çıkarılmış veri kümeleri de Github'da https://github.com/msbicer/phd-datasets adresinden görülebilir. Seçilen projelere ait bazı istatistikler Çizelge 4.5'den görülebilir. Bu istatistikler gitstats [124] uygulaması kullanılarak çıkarılmıştır.

Çizelge 4.5 Proje istatistikleri

Ad	Kategori	Versiyon	Geliştirici Sayısı	Satır Sayısı	Dosya Sayısı	Uygulama Değişim Sayısı
phpBB	Forum	3.0	23	228339	944	7497
punBB	Forum	1.4b	13	59668	237	893
FluxBB	Forum	1.5.3	32	43406	219	1360
phpMyAdmin	Veritabanı Yönetimi	3.5	330	1140741	1142	70113
phpPgAdmin	Veritabanı Yönetimi	5.0	26	149136	496	1957
Drupal	İçerik Yönetimi	6.16	9	88399	464	8996
Wordpress	İçerik Yönetimi	3.0	53	382600	1246	25712
Joomla	İçerik Yönetimi	3.1	239	581606	5573	15726
ZenPhoto	İçerik Yönetimi	1.4.4	18	536863	1666	3038
PyroCMS	İçerik Yönetimi	2.2.2	289	332318	3941	9963
Typecho	Blog Oluşturma	0.9-13.12.1	29	87856	316	839
2048	Oyun	-	3	3840 / 658	26	3
adarkroom	Oyun	1.2	Bilinmiyor	9861 / 915	38	Bilinmiyor
hextris	Oyun	-	8	6097 / 1726	67	340
effeckt.css	Kütüphane	-	34	64737 / 50126	142	350
bootstrap	Kütüphane	3.2	621	52514 / 9621	341	9496

phpBB ücretsiz ve açık kaynak kodlu bir forum yazılımıdır. İlk versiyonu 2000'de piyasaya sürülmüş ve halen büyük bir topluluk tarafından kullanılmakta ve desteklenmektedir.

punBB, phpBB'ye alternatif olarak geliştirilmiş bir forum yazılımıdır. İlk versiyonu 2003 yılında yayınlanmış ve halen aktif olarak geliştirilmektedir.

FluxBB, punBB'nin yaratıcıları tarafından punBB bir firmaya satıldıktan sonra onun kodundan türetilmiş bir forum yazılımıdır. 2008 yılından beri geliştirilmektedir.

phpMyAdmin MySQL veritabanlarını yönetmek için geliştirilmiş web tabanlı bir araçtır.

Veritabanı yönetimi için çeşitli özellikleri desteklemektedir. 1998'den beri geliştirilmektedir.

phpPgAdmin yine veritabanı yönetimi için geliştirilmiş web tabanlı bir araçtır. PostgreSQL veritabanlarında kullanılmak üzere geliştirilmiştir. Başlangıçta kodu phpMyAdmin'in kod tabanından alınmış, ancak 2002 yılında baştan yazılmıştır.

Drupal açık kaynak kodlu bir içerik yönetim sistemidir. Geçmişi 2001 yılına dayanmaktadır ve en ünlü CMS yazılımları arasında yer almaktadır.

Wordpress de bir CMS yazılımı ve belki de en ünlü ve en yaygın kullanılan CMS yazılımıdır. 2003 yılından beri geliştirilmekte olup, çeşitli eklentiler tarafından desteklenmektedir.

Joomla da ünlü bir CMS aracıdır. 2005 yılından bu yana geliştirilmektedir.

ZenPhoto multimedya odaklı web siteleri için geliştirilmiş bağımsız çalışan bir CMS ürünüdür. 2005 yılından bu yana geliştirilmektedir.

PyroCMS da yine bir içerik yönetim sistemidir. Laravel üzerine yapılmış ve MVC desteği ile öne çıkmaktadır. 2009 yılından beri geliştirilmektedir.

Typecho 2013 yılında geliştirilmeye başlanmış bir blog oluşturma aracıdır.

2048 mobil uygulama olarak ortaya çıkmış oyunun web'e uyarlanmış halidir.

adarkroom metin tabanlı, web üzerinden oynanabilen bir oyundur.

hextris HTML5 tabanlı grafiksel bir oyundur.

effeckt.css Web geliştiricileri için oluşturulmuş bir kütüphanedir.

bootstrap Ünlü web kütüphensinin tema kısmıdır.

BÖLÜM 5

SUNUCU TARAFLI KOD METRİKLERİ

Web uygulamalarının kodlanmasında kullanılan diller ve teknolojiler sunucu taraflı ve istemci tabanlı olarak 2'ye ayrılmaktadır. İki taraf için de kullanılan kavramlar ve yöntemler birbirinden çok farklı olabilmektedir. Öyle ki sunucu tarafında kullanılan programlama dilleri klasik prosedürel dillerdir (PHP, Java, C#, Python vb). İstemci tarafında ise prosedürel bir dil olan JavaScript'e ek olarak markup veya şablon dilleri (HTML, CSS ve türevleri) kullanılmaktadır. Markup ve şablon dillerinde herhangi bir karar yapısı, döngü, değişken tanımı gibi alışık olduğumuz kontrol yapıları ve ifadeler bulunmamaktadır. Bu gibi teknik farklılıklardan ötürü web uygulamalarında kullanılan teknlojileri birbirinden ayırarak incelemenin zorunlu olduğu düşünülmüştür.

5.1 Kullanılan Metrikler

Çalışmada sunucu taraflı uygulama kodlarının analizi için Çizelge 5.2'de görülen metrik kümesi kullanılmıştır. Bu çalışmadaki amacımız, statik kod ölçütlerini değiştirmek değil, bilgi içeriğini zenginleştirerek onları desteklemektir. Çizelgede görülen statik kod metrikleri kısmı, literatürde şimdiye kadar kullanılmakta olan, bütün prosedürel programlama dillerinden otomatik araçlar yardımıyla çıkarılabilecek olan metriklerden oluşmaktadır ve bizim kontrol grubumuzu oluşturmaktadır. Önerilen metrik seti ise Bölüm 3'te anlatılmış olan web uygulamalarındaki hataların uygulama kodlarına adreslenmesi ile oluşturulmuş metrik kümesidir. Bu kümenin oluşturulmasında Bölüm 3.1'de verilen hata çizelgesi göz önüne alınmıştır. Parantez içinde her özellik için kullanılan kısa kod verilmiştir. Veri kümesinde ve çalışmanın geri kalanında da bu kısa kodlar kullanılmaktadır.

- Oturum parametresi kullanımı: Bir oturum parametresine dosya içinde okuma veya yazma için erişilip, erişilmediği.
- Oturumdan okuma sayısı: Oturum parametrelerine dosya içinde okuma için yapılmış erişim sayısı.
- Oturuma yazma sayısı: Oturum parametrelerine dosya içinde yazma için yapılmış

erişim sayısı.

- Çerez kullanımı: Dosya içinde çerezlere okuma veya yazma için yapılmış erişim sayısı.
- İstek başlığı kullanımı: Dosya içinde istek/yanıt başlıklarına okuma veya yazma için yapılmış erişim sayısı.
- İstek parametresi kullanımı: Dosya içinde bir istek parametresine erişim yapılıp yapılmadığı.
- İstek parametrelerine erişim sayısı: Dosya içinde istek parametrelerinden okuma sayısı.
- Bağlam değişimi: dosya içinde sunucu tarafı ile istemci tarafı kodu arasında yapılmış değişim sayısı.
- Veritabanı sorgu sayısı: Dosya içindeki veri çekme sorgularının kullanım sayısı.
- Veritabanı sorgu kullanımı: Dosya içinde veritabanına erişim yapılıp yapılmadığı.

5.2 Veri Kümesi Oluşumu

Bu bölümde, Bölüm 4.4'de açıklanan projelerden elde edilen veriler kullanılmıştır. Öncelikle Çizelge 4.5 içindeki projelerin belirtilen sürümlerinin kesitleri Github sayfalarından indirilmiştir. İndirilen modüller kesit olarak alınmış ve modülleri hatalı-hatasız olarak etiketlemek için aldığımız tarihten geriye dönük 1 yıl içinde her dosya için hata düzeltmesi olup olmadığı kontrol edilmiştir. Bir kod değişiminin hata düzeltmesi olarak etiketlenebilmesi için, değişim mesajlarında (bug, error, fix, fail gibi) bazı anahtar kelimeleri aradık. Sonunda, bu mesajların geçtiği düzeltmelerden etkilenen dosyalar kusurlu olarak etiketlendi.

Statik kod ölçütlerini çıkartmak için Understand [125] uygulamasının 4.0 versiyonu kullanılmıştır. Bu tür metrikler yalnızca programlama dilleri için kullanılabilir olduğundan, veri kümelerine yalnızca PHP dosyaları eklenmiştir. Bu adımın sonucu, statik kod ölçütlerini içeren özellik vektörlerinden oluşan bir veri kümesidir.

Her proje için deneylerde 4 farklı veri kümesi oluşturulmuş ve kullanılmıştır:

- **Tam**: Mevcut statik kod metrikleri ve önerilen metrikler birlikte kullanılmıştır. Bu kurulumu kontrollü bir deney olarak düşünürsek, bu grup veri kümesi deney grubu rolünde olmaktadır.
- **Eski**: Sadece mevcut statik kod metrikleri kullanılmıştır. Kontrollü deneyimizde, bu grup veri kümesi deney grubu rolünde olmaktadır.
- **Tam-Filtreli**: Tüm metriklerden oluşan veri kümesine CFS algoritması uygulanarak

Çizelge 5.1 Sunucu taraflı metrik kümesi (a)

Statik Kod Metrikleri	
Ortalama Döngüsel Karmaşıklık (AvgCyclomatic)	Yol Sayısı (CountPath)
Ortalama Değiştirilmiş Döngüsel Karmaşıklık (AvgCyclomaticModified)	İfade Sayısı (CountStmt)
Ortalama Katı Döngüsel Karmaşıklık (AvgCyclomaticStrict)	Tanımlama İfadesi Sayısı (CountStmtDecl)
Ortalama Gerekli Döngüsel Karmaşıklık (AvgEssential)	Uygulama İfadesi Sayısı (CountStmtExe)
Ortalama Satır Sayısı (AvgLine)	Döngüsel Karmaşıklık (Cyclomatic)
Ortalama Boş Satır Sayısı (AvgLineBlank)	Değiştirilmiş Döngüsel Karmaşıklık (CyclomaticModified)
Ortalama Kod Satır Sayısı (AvgLineCode)	Katı Döngüsel Karmaşıklık (CyclomaticStrict)
Ortalama Yorum Satır Sayısı (AvgLineComment)	Gerekli Karmaşıklık (Essential)
Sınıf Sayısı (CountDeclClass)	Maksimum Döngüsel Karmaşıklık (MaxCyclomatic)
Dosya Sayısı (CountDeclFile)	Maksimum Değiştirilmiş Döngüsel Karmaşıklık (MaxCyclomaticModified)
Fonksiyon Sayısı (CountDeclFunction)	İç içe Geçme (MaxNesting)
Satır Sayısı (CountLine)	Yorum-Kod Oranı (RatioCommentToCode)
Boş Satır Sayısı (CountLineBlank)	Toplam Döngüsel Karmaşıklık (SumCyclomatic)
Kod Satır Sayısı (CountLineCode)	Toplam Değiştirilmiş Döngüsel Karmaşıklık (SumCyclomaticModified)
Yorum Satır Sayısı (CountLineComment)	Toplam Katı Döngüsel Karmaşıklık (SumCyclomaticStrict)
	Toplam Gerekli Karmaşıklık (SumEssential)

filtrelenmiş halidir. Sadece filtreden geçebilen özellikler kullanılmıştır. Her veri kümesi için seçilen özellikler değişiklik gösterebilir.

- **Eski-Filtreli**: Eski metriklerden oluşan veri kümesine CFS algoritması uygulanarak filtrelenmiş halidir. Sadece filtreden geçebilen özellikler kullanılmıştır. Her veri kümesi için seçilen özellikler değişiklik gösterebilir.

Çizelge 5.2 Sunucu taraflı metrik kümesi (b)

Önerilen Metrik Seti	
Oturum Parametresi Kullanımı (Has_Session)	İstek Parametresi Kullanımı (Has_Param)
Oturumdan Okuma Sayısı (Session_Reads)	İstek Parametrelerine Erişim Sayısı (Param_Reads)
Oturuma Yazma Sayısı (Session_Writes)	Bağlam Değişimi (Context_Switches)
Çerez Kullanımı (Cookies)	Veritabanı Sorgu Sayısı (DB_Query)
İstek Başlığı Kullanımı (Header_Access)	Veritabanı Sorgu Kullanımı (DB_Query_Binary)
HTML Kodu Bulundurması (Has_HTML)	

5.3 Sonuçlar

Özellik seçimi filtresinin her proje için sonuçları Çizelge 5.3'den görülebilir. Sonuçlar, ilk araştırma sorusu için umut verici işaretler göstermektedir. Öncelikle en az 1 metrik, 11 projenin 10'unda özellik filtresini geçmiştir. Kalın yazıyla gösterilen değerler, önerilen metrik kümemizden alınmıştır. Önerilen metriklerin mevcut kod metriklerine oranı projeler arasında değişmektedir. En çok seçilen metrik türü DB sorguları ile ilgili olanlardır. Bu durum bu metrik türünün hatalılık tahmini için daha yararlı olabileceğini göstermektedir. Üstelik, toplam 11 metrik önermemize rağmen bunlardan toplam 7 tanesi CFS filtresini geçebilmiştir. Kalan ölçütlerin tamamen yararsız olduğunu söyleyemeyiz, ancak hatalılığı göstermede daha düşük korelasyona sahiplerdir.

11 küme için elimizde 44 veri kümesi oluşmuştur. Veri kümelerinin her biri 3 farklı makine öğrenmesi algoritması ile değerlendirilmiştir. Deneyler sonucunda elimizde bulunan pd, pf ve denge değerlerinin her biri Çizelge 5.4, 5.5 ve 5.6'ten görülebilir. Sonuçların her biri çift halinde karşılaştırılmış; yani filtrelenmiş metrik kümelerini birbiriyle, filtrelenmemiş metrik kümeleri de birbiriyle karşılaştırılmıştır. Değerlerdeki * işareti, işaretlenmiş değerin aynı çiftteki diğer değerden istatistiksel olarak daha yüksek olduğunu gösterir. Farkların istatistiksel olarak anlamlı olup olmadığı Mann-Whitney U Testi kullanılarak test edilmiştir.

Çizelge 5.3 Projelerde filtrelenmiş metrikler

Veri Kümesi	Seçilen Özellikler			
WordPress	AvgLineCode **Param_Reads**	CountLine **Context_Switches**	Cyclomatic **DB_Query**	MaxCyclomatic **DB_Query_Binary**
Joomla	CountLine **DB_Query_Binary**	CountLineCode	RatioCommentToCode	**DB_Query**
Drupal	AvgLine CountLineCode **DB_Query**	AvgLineCode CountStmt	CountLine MaxNesting	CountLineBlank RatioCommentToCode
phpMyAdmin	AvgCyclomaticStrict Cyclomatic	CountLineBlank MaxCyclomatic	CountLineComment SumEssential	CountStmt
phpPgAdmin	AvgLine MaxCyclomaticModified	AvgLineBlank **Session_Reads**	CountLineBlank **Param_Reads**	CyclomaticModified
phpBB	CountLine SumCyclomaticModified	CountLineBlank **DB_Query**	CountLineComment	SumCyclomatic
ZenPhoto	CountPath	MaxCyclomaticModified	RatioCommentToCode	**Context_Switches**
PunBB	CountLine Essential	CountLineBlank RatioCommentToCode	CountLineComment **Cookies**	Cyclomatic **Param_Reads**
Typecho	CountLineCode CountStmtExe **Header_Access**	CountLineComment CyclomaticModified	CountPath Essential	CountStmtDecl RatioCommentToCode
FluxBB	Cyclomatic	**Param_Reads**		
PyroCMS	CountLine MaxCyclomatic	CountLineBlank MaxNesting	CountLineCode **Context_Switches**	CountStmt

Elimizde olan 11 veri kümesi için toplamda 66 karşılaştırma yapılmıştır. Denge değerleri için 66 değerin 19'unda (yaklaşık %30) istatistiksel olarak belirgin bir artış bulunmuştur. Filtrelenmiş ve filtrelenmemiş sonuçlar için yaklaşık olarak artışlar yaklaşık olarak aynı sayıdadır (9 - 10). Bunla birlikte 66 değerin 7'sinde (yaklaşık %10) eski değerlere göre azalma olmuştur. Filtrelenmiş metriklerdeki azalma oranı ise diğerlerine göre daha fazladır (5 - 2). Ancak genel sonuç olarak baktığımızda denge değerlerinde eski metrik kümelerine göre %2'lik bir artış elde edilmiştir. Artış veri kümesine göre %2 ile %30 arasında değişmektedir. Ortalamada bulunan değişiklik küçük olmasına rağmen az da olsa, bize yeni metriklerin bilgi sağladıklarını ve veri kümelerinin bilgi içeriğinin daha da zenginleştirilebileceğini göstermektedir. Bu açıdan baktığımızda önerilen metriklerin hataya yatkınlık tahmini çalışmalarında olumlu etkilerinin olduğunu söyleyebiliriz.

5.4 Değerlendirme ve Yorumlar

Çalışmanın bu bölümünde araştırma sorularımıza cevap bulmak için web uygulamalarına ait sunucu taraflı kodlar incelenmeye çalışılmıştır. Web uygulamalarının karakterlerinden bazı metrikler çıkarıp bunları hataya yatkınlık tahmini modelimize sokarak performansları incelenmiştir. Bu amaçla 11 metrik tanımladık. Tanımladığımız metriklerin web uygulamalarının hatalılığını tahmin etmedeki kullanışlılığını görmek için bir öğrenme tabanlı hatalılık tahmini modeli oluşturulmuştur. Bulgularımızı mevcut ölçümlerin performansıyla karşılaştırdık. Sınıflandırıcıların performansı, pd, pf ve denge metriklerini kullanarak 10×10'luk çapraz doğrulama işlemi ile değerlendirilmiştir. Sonuç olarak, performans açısından küçük fakat istatistiksel olarak önemli farklar elde edilmiştir. Genel tahmin performansındaki artış, veri kümesine bağlı olarak %2 ila %30 aralığında değişmektedir. Bu sonuç, önerdiğimiz metriklerin hataya yatkınlık tahmini edicilerinde kullanışlı olduğunu göstermektedir. İleriki kısımlarda hataya yatkınlık tahmini performansını arttırmak için, tanımladığımız metrikleri ek metriklerle destekleyebilmeyi inceleyeceğiz.

Çizelge 5.4 Hata yakalama oranları

Veri Kümesi	**Algoritma**	Tam	Eski	Tam-Filtreli	Eski-Filtreli
WordPress	**NaiveBayes**	0.51	0.44	0.52	0.32
	Rastgele Orman	0.60	0.59	0.60	0.57
	Bayes Ağı	0.71	0.65	0.72	0.72
Joomla	**NaiveBayes**	0.88	0.88	0.85	0.85
	Rastgele Orman	0.79	0.81	0.72	0.78
	Bayes Ağı	0.70	0.69	0.74	0.74
Drupal	**NaiveBayes**	0.56	0.54	0.55	0.56
	Rastgele Orman	0.87	0.86	0.88	0.87
	Bayes Ağı	0.86	0.86	0.86	0.86
phpMyAdmin	**NaiveBayes**	0.41	0.42	0.35	0.35
	Rastgele Orman	0.35	0.36	0.37	0.37
	Bayes Ağı	0.49	0.48	0.38	0.38
phpPgAdmin	**NaiveBayes**	0.66	0.63	0.60	0.43
	Rastgele Orman	0.80	0.76	0.76	0.75
	Bayes Ağı	0.96	0.96	0.86	0.87
phpBB	**NaiveBayes**	0.32	0.37	0.28	0.29
	Rastgele Orman	0.23	0.20	0.21	0.18
	Bayes Ağı	0.71	0.71	0.23	0.25
ZenPhoto	**NaiveBayes**	0.24	0.24	0.19	0.19
	Rastgele Orman	0.88	0.87	0.82	0.84
	Bayes Ağı	0.77	0.80	0.89	0.91
PunBB	**NaiveBayes**	0.49	0.45	0.48	0.48
	Rastgele Orman	0.45	0.42	0.49	0.51
	Bayes Ağı	0.60	0.60	0.57	0.56
Typecho	**NaiveBayes**	0.48	0.44	0.34	0.36
	Rastgele Orman	0.30	0.33	0.27	0.29
	Bayes Ağı	0.61	0.61	0.54	0.54
FluxBB	**NaiveBayes**	0.43	0.43	0.55	0.47
	Rastgele Orman	0.64	0.64	0.64	0.58
	Bayes Ağı	0.70	0.23	0.71	0.23
PyroCMS	**NaiveBayes**	0.27	0.26	0.31	0.28
	Rastgele Orman	0.83	0.83	0.83	0.83
	Bayes Ağı	0.52	0.53	0.67	0.64
Ortalama		0.59	0.57	0.56	0.54

Çizelge 5.5 Yanlış alarm oranları

Veri Kümesi	**Algoritma**	Tam	Eski	Tam-Filtreli	Eski-Filtreli
WordPress	**NaiveBayes**	0.13	0.12	0.10	0.06
	Rastgele Orman	0.13	0.14	0.14	0.15
	Bayes Ağı	0.20	0.21	0.21	0.22
Joomla	**NaiveBayes**	0.78	0.81	0.70	0.77
	Rastgele Orman	0.32	0.34	0.35	0.37
	Bayes Ağı	0.34	0.35	0.35	0.37
Drupal	**NaiveBayes**	0.12	0.11	0.10	0.10
	Rastgele Orman	0.39	0.39	0.40	0.39
	Bayes Ağı	0.34	0.34	0.33	0.33
phpMyAdmin	**NaiveBayes**	0.11	0.10	0.08	0.08
	Rastgele Orman	0.09	0.10	0.10	0.10
	Bayes Ağı	0.20	0.18	0.12	0.12
phpPgAdmin	**NaiveBayes**	0.13	0.13	0.10	0.10
	Rastgele Orman	0.09	0.11	0.08	0.10
	Bayes Ağı	0.20	0.20	0.14	0.12
phpBB	**NaiveBayes**	0.15	0.16	0.10	0.10
	Rastgele Orman	0.03	0.03	0.04	0.04
	Bayes Ağı	0.37	0.37	0.10	0.11
ZenPhoto	**NaiveBayes**	0.10	0.09	0.06	0.04
	Rastgele Orman	0.53	0.51	0.54	0.48
	Bayes Ağı	0.66	0.74	0.84	0.84
PunBB	**NaiveBayes**	0.09	0.11	0.08	0.06
	Rastgele Orman	0.09	0.09	0.10	0.10
	Bayes Ağı	0.13	0.13	0.11	0.09
Typecho	**NaiveBayes**	0.07	0.06	0.03	0.04
	Rastgele Orman	0.03	0.04	0.04	0.04
	Bayes Ağı	0.14	0.15	0.09	0.09
FluxBB	**NaiveBayes**	0.17	0.18	0.12	0.14
	Rastgele Orman	0.28	0.28	0.29	0.29
	Bayes Ağı	0.21	0.17	0.22	0.17
PyroCMS	**NaiveBayes**	0.05	0.05	0.05	0.04
	Rastgele Orman	0.21	0.21	0.21	0.21
	Bayes Ağı	0.15	0.16	0.21	0.21
Ortalama		0.21	0.21	0.19	0.19

Çizelge 5.6 Denge değerleri

Veri Kümesi	**Algoritma**	Tam	Eski	Tam-Filtreli	Eski-Filtreli
WordPress	**NaiveBayes**	0.64*	0.59	0.65*	0.52
	Rastgele Orman	0.70	0.70	0.70*	0.68
	Bayes Ağı	0.75*	0.71	0.75	0.75
Joomla	**NaiveBayes**	0.44*	0.42	0.49*	0.45
	Rastgele Orman	0.73	0.72	0.68	0.70*
	Bayes Ağı	0.68	0.67	0.69	0.68
Drupal	**NaiveBayes**	0.68	0.67	0.68	0.68
	Rastgele Orman	0.71	0.71	0.71	0.71
	Bayes Ağı	0.74	0.74	0.75	0.75
phpMyAdmin	**NaiveBayes**	0.57	0.59	0.54	0.54
	Rastgele Orman	0.53	0.54	0.55	0.55
	Bayes Ağı	0.61	0.61	0.55	0.55
phpPgAdmin	**NaiveBayes**	0.74	0.73	0.71*	0.59
	Rastgele Orman	0.84*	0.81	0.82	0.81
	Bayes Ağı	0.85	0.85	0.86	0.87
phpBB	**NaiveBayes**	0.51	0.54*	0.49	0.49
	Rastgele Orman	0.46*	0.43	0.44*	0.42
	Bayes Ağı	0.66	0.66	0.45	0.46
ZenPhoto	**NaiveBayes**	0.45	0.46	0.43	0.43
	Rastgele Orman	0.62	0.63	0.60	0.64*
	Bayes Ağı	0.50*	0.46	0.40	0.40
PunBB	**NaiveBayes**	0.63*	0.60	0.63	0.63
	Rastgele Orman	0.60*	0.58	0.63	0.65*
	Bayes Ağı	0.70	0.71	0.69	0.68
Typecho	**NaiveBayes**	0.63*	0.60	0.53	0.55*
	Rastgele Orman	0.50	0.52*	0.48	0.50*
	Bayes Ağı	0.70	0.71	0.67	0.67
FluxBB	**NaiveBayes**	0.58	0.58	0.67*	0.61
	Rastgele Orman	0.68	0.68	0.67*	0.63
	Bayes Ağı	0.74*	0.44	0.74*	0.44
PyroCMS	**NaiveBayes**	0.48	0.48	0.51*	0.49
	Rastgele Orman	0.81	0.81	0.81	0.81
	Bayes Ağı	0.64	0.65	0.72	0.71
Ortalama		0.64	0.62	0.62	0.60

BÖLÜM 6

İSTEMCİ TARAFLI KOD METRİKLERİ

Bir önceki bölümde sadece sunucu taraflı kod incelemesi yapılmıştı. Bu bölümde ise kullandığımız projelerden daha önce hiçbir hataya yatkınlık tahmini çalışmasında kullanılmamış olan HTML ve CSS kodları çıkarılıp analiz edilmiştir. Amacımız tahmin yöntemlerinin performansını yeni metrikler üretmek yoluyla mümkün olduğunca arttırmaktır.

Web uygulamalarındaki fonksiyonaliteler günümüzde ağırlıklı olarak PHP, Java, Python vb. diller kullanılarak sunucu tarafında yapılan geliştirmeler ile sağlanmaktadır. Bu nedenle hataların çoğunun sunucu taraflı kodlarda yapılacak analizlerle anlaşılması doğal bir sonuçtur. Ancak yapılan web sayfalarının kullanıcıya gösterilebilmesi, kullanıcıdan aksiyon alınabilmesi, dahası sayfalara stil özellikleri verilebilmesi için istemci tarafında çalışacak bazı kodların yazılması kaçınılmazdır. Bu nedenle esas geliştirme bahsettiğimiz gibi sunucu tarafında olurken bu geliştirmeleri destekleyecek yeni geliştirmeler halen istemci taraflı bazı diller (HTML ve CSS) kullanılarak yapılmaktadır.

6.1 HTML Metrikleri

Bir web sayfasının tarayıcıda oluşturulup gösterilmesi için HTML üretilmesi kaçınılmazdır, ancak günümüzde web uygulamalarına dinamiklik getiren Web 2.0'ın yaygınlaşmasıyla sadece HTML kodu kullanımı giderek azalmıştır. HTML JavaScript veya sunucu tarafında dinamik olarak oluşturulmaya başlanmıştır, ancak yine de kaynak dosyalarında HTML kullanımı tamamen terkedilmemiştir.

Bu çalışmanın amacı, web uygulamalarının özelliklerinden yararlanarak hatalılık tahmini performansını iyileştirmek için yeni bir metrik küme önermektir. Amacımız yine statik kod ölçütlerini değiştirmek yerine, bilgi içeriğini zenginleştirerek onları desteklemektir. Dolayısıyla, önerilen metrik setimiz statik kod ölçütlerine ek olarak kullanılacaktır. Önerilen metriklerimizin tamamı Çizelge 6.1'de görülebilir.

Çizelge 6.1 HTML metrikleri

Toplam / Tekil etiket sayısı (total_tags / unique_tags)	Toplam / Tekil / Ortalama özellik sayısı (total_attrs / unique_attrs / avg_attrs)
Maksimum / Toplam / Ortalama derinlik (max_depth / total_depth / avg_depth)	Toplam Yorum Sayısı (total_comments)
Toplam İçerik (total_text)	Form sayısı (forms)
Girdi sayısı (inputs)	Çerçeve sayısı (frames)
Script sayısı (scripts)	Dışarıdan yüklenen scriptler (external_scripts)
Sayfada tanımlanan scriptler (in_page_scripts)	Stil etiketi sayısı (style_tags)
Yüklenen stil etiketi sayısı (css_included)	Sayfada tanımlanan stil etiketi sayısı (inline_css)
Link sayısı (anchors)	

6.1.1 HTML Nedir?

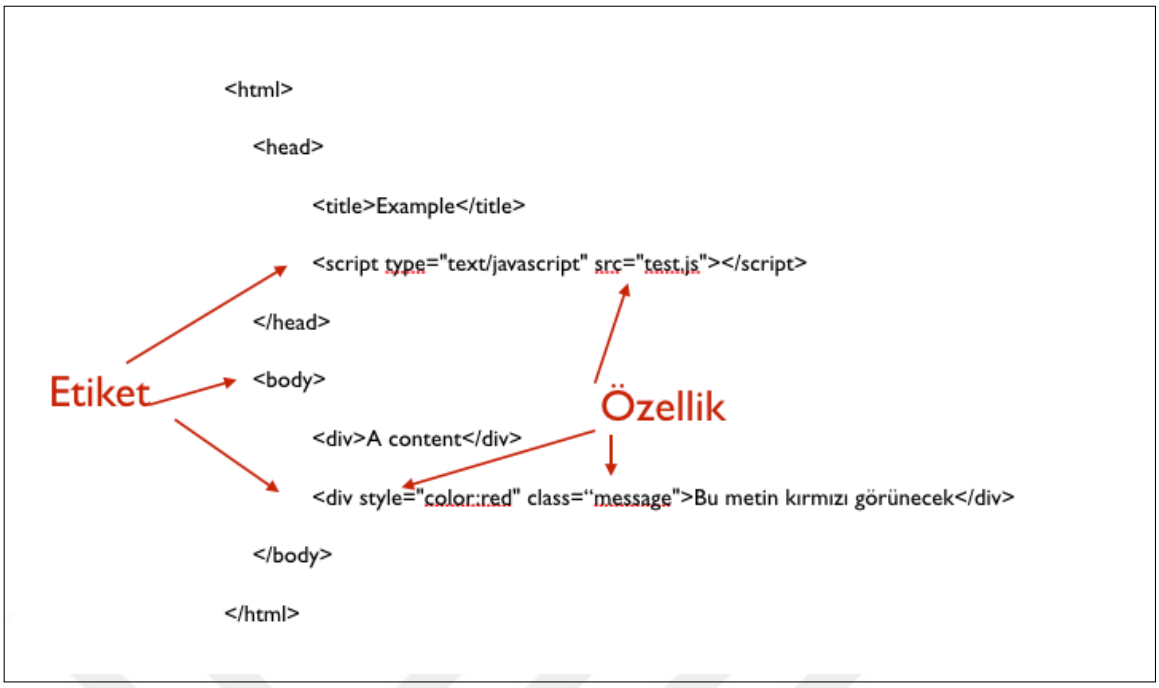
HTML, web sayfalarını oluşturmak ve yapısını tanımlamak için kullanılan standart bir dildir [126]. CSS ve Javascript ile birlikte WWW'in temel yapıtaşlarından biridir. Tarayıcıların web sayfalarını görüntüleyebilmek için sunucudan HTML kodlarını almaları gerekmektedir. HTML ile bir web sayfasının nasıl görüneceği tanımlanır.

HTML 1989 yılında CERN'de bir araştırmacı olan Tim Berners-Lee tarafından ortaya atılmış ve W3C'in [127] de kurulmasıyla birlikte dünyanın dört bir yanından araştırmacıların katkılarıyla gelişmeye devam etmekte olan bir biçimlendirme dilidir [128]. HTML başlıklar, formlar, listeler, bağlantılar gibi bileşenler kullanılarak yapılandırılmış belgeler yaratmak için bir araç sağlar. HTML elemanları, < ve > kullanılarak yazılmış etiketlerle çizilir. <a /> ve <input /> gibi etiketler ile içeriğin ne şekilde kullanılacağı belirtilir. Tarayıcılar HTML etiketlerini görüntülemez ancak bunları sayfanın içeriğini yorumlamak için kullanırlar. Örnek bir HTML kodu Şekil 6.1'de görülebilir.

HTML, web sayfası davranışını ve içeriğini etkileyen JavaScript gibi bir betik dili ile yazılmış programları içerebilir. CSS'nin dahil edilmesi içeriğin görünümünü ve düzenini tanımlar.

6.1.2 Veri Analizi

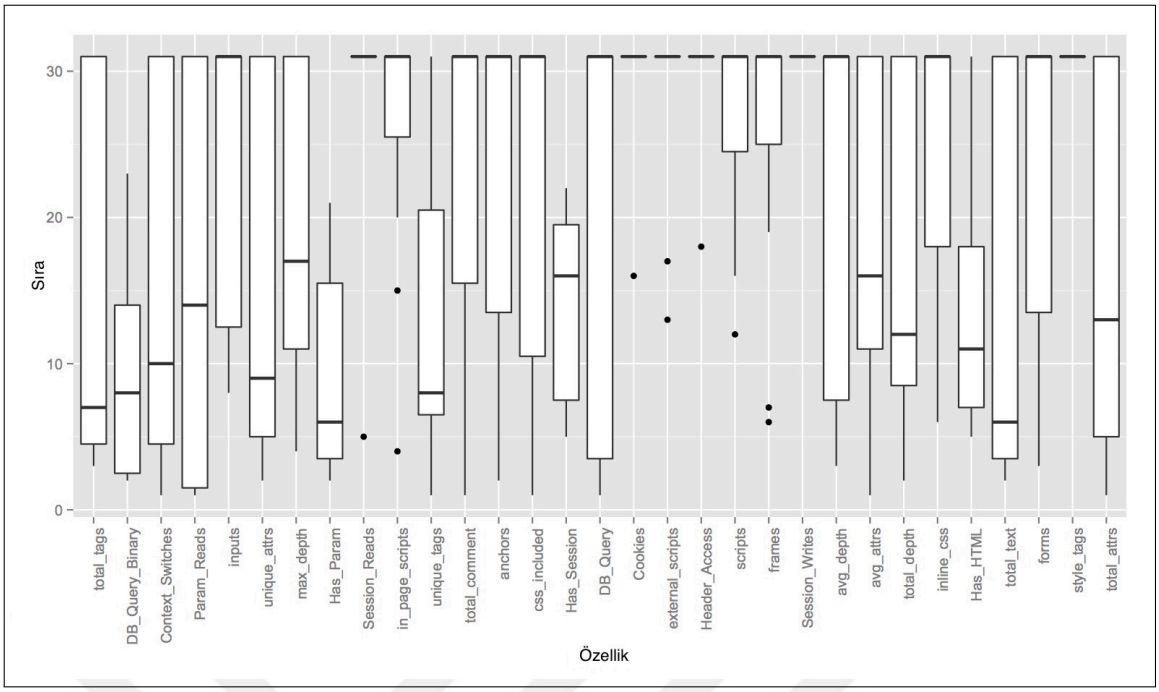
Önceki bölümde aldığımız sonuçlar umut verici olmasına rağmen, veri kalitesini artırmanın yollarını aramaya devam etmeliyiz. Bu amaçla önerilen niteliklerin değerini belirlememiz gerekir. Veri kümelerindeki her özniteliğin önemini ölçmek için Kullback-Leibler ayrımı [129] (Bilgi Kazancı) algoritmasını kullandık. Aynı özellik, farklı veri kümelerinde



Şekil 6.1 HTML kod örneği

farklı kazançlar gösterebilir, bu nedenle bu değeri normalleştirmeliyiz. Bu normalizasyon nitelikleri sıralarına göre sıralamak ve bilgi kazancı değerleri yerine sıralama değerleri kullanarak yapılır. Bu analizin oluşturulması sırasında bilgi kazancı değeri sıfır olan özellikler de sıralamada son sıraya alınmışlardır. Bu nedenle sıralamada sonuncu gelmek, birden fazla özellik için görülen en kötü değer olabilir. Sıralar Şekil 6.2'te incelenmek üzere görselleştirilmiştir. Her özellik, her veri kümesinin içinde, aldıkları bilgi kazancı değerine göre sıralanmıştır. Elimizde 11 proje olduğu için, her özellik için 11 farklı değer bulunmaktadır. Bu değerlerin minimum, maksimum, 1. çeyrek, medyan ve 3. çeyrek değerleri kutu grafiği şeklinde gösterilmektedir. Değerler yerine sıralama derecelerini kullandığımız için, az olan değer daha iyidir. Sıralamadan bazı gözlemlerimiz bulunmaktadır:

- Özniteliklerin önemi, veri kümeleri arasında dengesiz dağılmıştır. Değerlerin çoğu sıralamada en düşük değerde görülebilmiştir. 14 metriğin medyan değeri minimum dereceden yüksektir.
- Metriklerden bazılarının veri setlerinin çoğunda neredeyse hiçbir değeri yoktur, ancak bunların nadiren kullanışlı olabileceği görülmektedir. Bu durum aykırı noktalarla gösterilir. Bununla birlikte, regresyon temelli bir sınıflandırma algoritması kullanmadığımız için, aykırı değerleri gözardı etmemeliyiz.
- İki niteliğin (oturuma yazmalar ve stil etiketleri) her zaman son olarak sıralandıkları için tamamen yararsız olduğu görülmüştür. Dolayısıyla bu metrikler, çalışmanın devamında sunulacak olan sonuçlardan çıkarılmıştır.



Şekil 6.2 Özelliklerin bilgi kazanımına göre sıralanması (HTML ve sunucu)

6.1.3 Veri Kümesi Oluşumu

Bu bölüm için izlenen dosya etiketleme süreci sunucu taraflı metrik analizi kısmında izlenen süreçle aynıdır. Metrik çıkarma süreci ise hedeflenen teknoloji farkından dolayı farklılaşmıştır.

HTML etiketleri PHP dosyaları üzerinde soyut sözdizimi ağacı (AST) analizi yapılarak çıkarılmıştır. Her dosya ayrı olarak analiz edilmiş, bu nedenle veri kümesindeki bir özellik vektörü farklı bir dosyaya karşılık gelmiştir. Dinamik olarak oluşturulmuş HTML kodları, HTML metrik çıkarımında değerlendirilmez, çünkü kapsam dışındadır, bu işlem sunucu tarafında yapıldığı için zaten bir önceki bölümde dolaylı olarak incelenmiştir. Ayrıca kaynak kodlarında sadece statik analiz yapılmaktadır. Bu yöntem, istemci tarafı biçimlendirme için statik kod ölçütlerini çıkarmamıza ve bunları daha önce kullanılan statik kod ölçütleriyle aynı veri kümelerinde temsil etmemize izin vermiştir. Bu adımın sonucu, sunucu tarafı ve istemci tarafı için statik kod ölçümlerine sahip bir veri kümesidir.

Her proje için deneylerde 4 farklı veri kümesi oluşturulmuş ve kullanılmıştır:

- **Tam**: Mevcut statik kod metrikleri ve önerilen metrikler birlikte kullanılmıştır. Bu kurulumu kontrollü bir deney olarak düşünürsek, bu grup veri kümesi deney grubu rolünde olmaktadır.
- **Eski**: Sadece mevcut statik kod metrikleri kullanılmıştır. Kontrollü deneyimizde, bu grup veri kümesi deney grubu rolünde olmaktadır.
- **Tam-Filtreli**: Tüm metriklerden oluşan veri kümesine CFS algoritması uygulanarak

filtrelenmiş halidir. Sadece filtreden geçebilen özellikler kullanılmıştır. Her veri kümesi için seçilen özellikler değişiklik gösterebilir.

- **Eski-Filtreli**: Eski metriklerden oluşan veri kümesine CFS algoritması uygulanarak filtrelenmiş halidir. Sadece filtreden geçebilen özellikler kullanılmıştır. Her veri kümesi için seçilen özellikler değişiklik gösterebilir.

6.1.4 Sonuçlar

Bu bölümde, deneyler sunucu taraflı kodlardan elde edilen metrik setine ek olarak HTML kodlarından elde edilen genişletilmiş metrik seti kullanılarak tekrar edilmiştir.

Özellik seçiminin her proje için sonuçları Çizelge 6.2'de verilmiştir. Bu bölümde eklenen metriklerle birlikte filtreden geçebilen metrik sayısı bir önceki bölüme göre artmıştır. Özellikle phpMyadmin projesinde de önerilen metrik kümesinden seçilebilen özellikler olmuştur. Yeni eklenen 20 HTML metriğinden toplam 12 tanesi en az bir proje için özellik seçiminden geçebilmiştir. Yine seçilen metrikler ve sayıları projeler arasında farklılık göstermektedir.

11 proje için elimizde 22 veri kümesi bulunmaktadır. Bunların her biri 3 farklı makine öğrenme algoritması ile eğitilip test edilmiştir. Sonuçlar bir önceki bölümde olduğu gibi yine çiftler arasında karşılaştırılmıştır. Yıldız ile işaretlenen sonuçlar çiftinden istatistiksel olarak belirgin fark olan değerleri göstermektedir. Belirginlik Mann-Whitney U Testi kullanılarak ölçülmüştür.

Toplamda 66 karşılaştırma yapılmıştır. 66 sonucun 29'unda (yaklaşık %44) metrik performanslarında ilerleme gözlenmiştir. 6 karşılaştırmada da (yaklaşık %9) başlangıç durumuna göre daha kötü sonuçlar elde edilmiştir. Bu sayılar bir önceki bölümde elde edilen değerlere göre daha iyi durumdadır, onbir karşılaştırmada daha iyi istatistiksel belirgin değişim elde edilmiştir. Sonuçların ortalamasına bakıldığında filtrelenmiş ve filtresiz sonuçların ikisinde de %4'lük bir ilerleme gözlenmiştir.

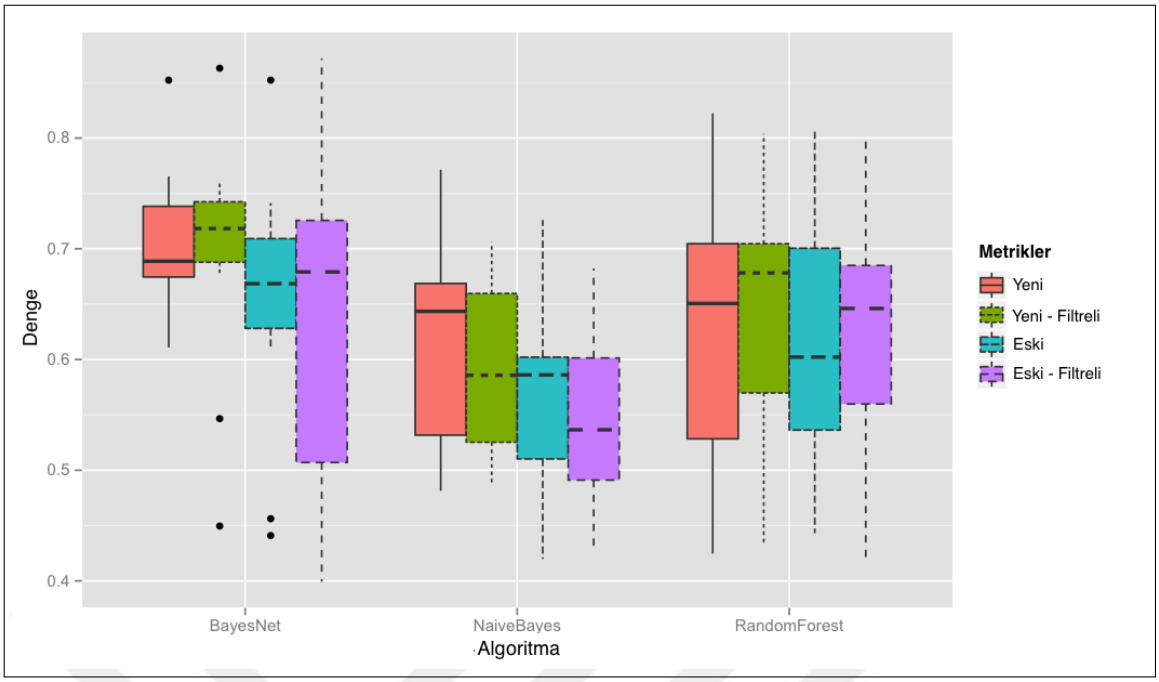
Sonuçlarda HTML ve sunucu metrik kullanımında sadece sunucu metrikleri kullanılarak yapılan deneylere göre görülen değişimler her değer için parantez içinde verilmiştir. Bazı değerlerde küçük azalmalar görülmekle birlikte genelde %1-%31 arası iyileşme görülmüştür. Bu oranlar kullanılan algoritmaya ve veri kümesine göre değişmektedir. Ortalama değerlere baktığımızda sunucu metriklerini kullanmaya göre %2 lik bir iyileşme gözlenmektedir.

Çizelge 6.2 Projelerde filtrelenmiş metrikler

Veri Kümesi	Seçilen Özellikler			
WordPress	AvgLineCode **Param_Reads** **Total_Text**	CountLine **DB_Query** **Forms**	Cyclomatic **DB_Query_Binary** **Anchors**	MaxCyclomatic **Total_Tags**
Joomla	CountLine **DB_Query_Binary**	CountLineCode	RatioCommentToCode	**DB_Query**
Drupal	CountLine RatioCommentToCode **Unique_attrs**	CountLineCode CountStmt **Avg_Attrs**	CountStmtExe **DB_Query** **In_Page_Scripts**	MaxCyclomaticModified **DB_Query_Binary**
phpMyAdmin	AvgCyclomaticStrict Cyclomatic **CSS_Included**	CountLineBlank MaxCyclomatic	CountLineComment SumEssential	CountStmt **Frames**
phpPgAdmin	AvgCyclomatic CyclomaticModified **Unique_Tags**	AvgLine MaxCyclomaticModified	AvgLineBlank **Session_Reads**	CountLineBlank **Param_Reads**
phpBB	CountLineBlank **Unique_Attrs**	CountLineComment	SumCyclomatic	**DB_Query**
ZenPhoto	CountPath **Total_Comments**	MaxCyclomaticModified **Anchors**	RatioCommentToCode **Frames**	**Context_Switches**
PunBB	CountLine MaxCyclomatic **Unique_Attrs**	CountLineBlank RatioCommentToCode	CountLineComment **Cookies**	Essential **Param_Reads**
Typecho	CountLineCode CountStmtExe **Header_Access** **Inline_CSS**	CountLineComment CyclomaticModified **Param_Reads**	CountPath Essential **Avg_Attrs**	CountStmtDecl RatioCommentToCode **External_Scripts**
FluxBB	**Has_Params**	**Param_Reads**	**Unique_Tags**	**Forms**
PyroCMS	CountLine MaxCyclomatic	CountLineBlank MaxNesting	CountLineCode **Context_Switches**	CountStmt **Total_Attrs**

Çizelge 6.3 Denge (sunucu ve HTML)

Veri Kümesi	Algoritma	Tam	Eski	Tam-Filtreli	Eski-Filtreli
WordPress	**NaiveBayes**	0.64*	0.59	0.63* (-0.02)	0.52
	Rastgele Orman	0.71 (+0.01)	0.70	0.73* (+0.03)	0.68
	Bayes Ağı	0.77* (+0.02)	0.71	0.76 (+0.01)	0.75
Joomla	**NaiveBayes**	0.67* (+0.23)	0.42	0.51* (+0.02)	0.45
	Rastgele Orman	0.74* (+0.01)	0.72	0.70 (+0.02)	0.70
	Bayes Ağı	0.69* (+0.01)	0.67	0.70* (+0.01)	0.68
Drupal	**NaiveBayes**	0.70* (+0.02)	0.67	0.68	0.68
	Rastgele Orman	0.71	0.71	0.70 (-0.01)	0.71
	Bayes Ağı	0.74	0.74	0.75	0.75
phpMyAdmin	**NaiveBayes**	0.57	0.59	0.55 (+0.01)	0.54
	Rastgele Orman	0.54 (+0.01)	0.54	0.55	0.55
	Bayes Ağı	0.61	0.61	0.55	0.55
phpPgAdmin	**NaiveBayes**	0.77* (+0.03)	0.73	0.71*	0.59
	Rastgele Orman	0.85* (+0.01)	0.81	0.82	0.81
	Bayes Ağı	0.85	0.85	0.86	0.87
phpBB	**NaiveBayes**	0.48 (-0.03)	0.54*	0.49	0.49
	Rastgele Orman	0.45* (-0.01)	0.43	0.44*	0.42
	Bayes Ağı	0.66	0.66	0.45	0.46
ZenPhoto	**NaiveBayes**	0.51* (+0.06)	0.46	0.54* (+0.11)	0.43
	Rastgele Orman	0.61 (-0.01)	0.63*	0.63 (+0.03)	0.64
	Bayes Ağı	0.69* (+0.19)	0.46	0.71* (+0.31)	0.40
PunBB	**NaiveBayes**	0.66* (+0.03)	0.60	0.64 (+0.01)	0.63
	Rastgele Orman	0.58 (-0.02)	0.58	0.64 (+0.01)	0.65
	Bayes Ağı	0.69 (-0.01)	0.71*	0.68 (-0.01)	0.68
Typecho	**NaiveBayes**	0.66* (+0.03)	0.60	0.59* (+0.06)	0.55
	Rastgele Orman	0.48 (-0.02)	0.52*	0.50 (+0.02)	0.50
	Bayes Ağı	0.74* (+0.04)	0.71	0.72* (+0.05)	0.67
FluxBB	**NaiveBayes**	0.56 (-0.02)	0.58*	0.68* (+0.01)	0.61
	Rastgele Orman	0.67 (-0.01)	0.68	0.73* (+0.06)	0.63
	Bayes Ağı	0.73* (-0.01)	0.44	0.74*	0.44
PyroCMS	**NaiveBayes**	0.49 (+0.01)	0.48	0.49 (-0.02)	0.49
	Rastgele Orman	0.81	0.81	0.81	0.81
	Bayes Ağı	0.61 (+0.04)	0.65*	0.73* (+0.01)	0.71
Ortalama		0.66 (+0.02)	0.62	0.64 (+0.02)	0.60



Şekil 6.3 Sınıflandırıcılar arası performans karşılaştırması (HTML ve sunucu)

Gözlemlere göre değişimlere en iyi tepki Bayes yöntemlerinden, özellikle Naive Bayes'ten alınmıştır, 22 karşılaştırmanın 13'ünde (%59) artış, ortalama olarak da +%4.5 iyileşme görülmüştür. Rastgele Orman algoritması bu değişime en düşük tepki veren algoritma olmuştur, 22 karşılaştırmanın sadece 6'sında (%27) iyileşme gözlenmiştir. Algoritmalar arasındaki karşılaştırma Şekil 6.3'de verilmiştir. Yeni metrik kümesinin en verimli olduğu algoritmanın NaiveBayes olduğunu görmekteyiz.

6.2 CSS Metrikleri

Web 2.0 kavramının ortaya çıkmasından önce web sayfaları çoğunlukla HTML ve az sayıda CSS kodunun birleşimi ile oluşturulmaktaydı. Günümüzde HTML kodlarının üretilmesi işi çeşitli kütüphaneler kullanılarak JavaScript veya sunucu tarafında yapılan geliştirmelere kaydırılmaya başlanmıştır, kütüphane olmasa bile HTML kodları yine dinamik olarak üretilmektedir. CSS kodları da bir miktar HTML ile birlikte üretiliyor olsa da halen büyük çoğunluğu sadece CSS kodu içeren dosyaların yazılıp sayfalara eklenmesi ile yapılmaktadır. CSS sayfaların görsel açıdan biçimlendirilmesini sağladığı için, özellikle CSS 3'ün çıkışı ve tarayıcıların Flash desteğini sonlandırması ile, yıllar geçtikçe daha çok önem kazanmaya devam etmektedir.

Çalışmanın bu kısmındaki amacımız CSS kodlarının hataya yatkınlığını tahmin edebilmektir. CSS üzerine literatürde daha önce yapılmış bir metrik çıkarımı veya hataya yatkınlık tahmini yapılması üzerine bir çalışma göremedik. Bu nedenle oluşturacağımız CSS metriklerinin diğer metrik setleriyle kombine edilerek kullanılması mümkün olmayacaktır. Önerdiğimiz metrikler ve anlamları aşağıda verilmiştir:

- colors: Kullanılan tekil renk sayısı
- complexSelectors: Kompleks seçici sayısı (üçten daha fazla ifade içeren, örnek: header ul li .foo)

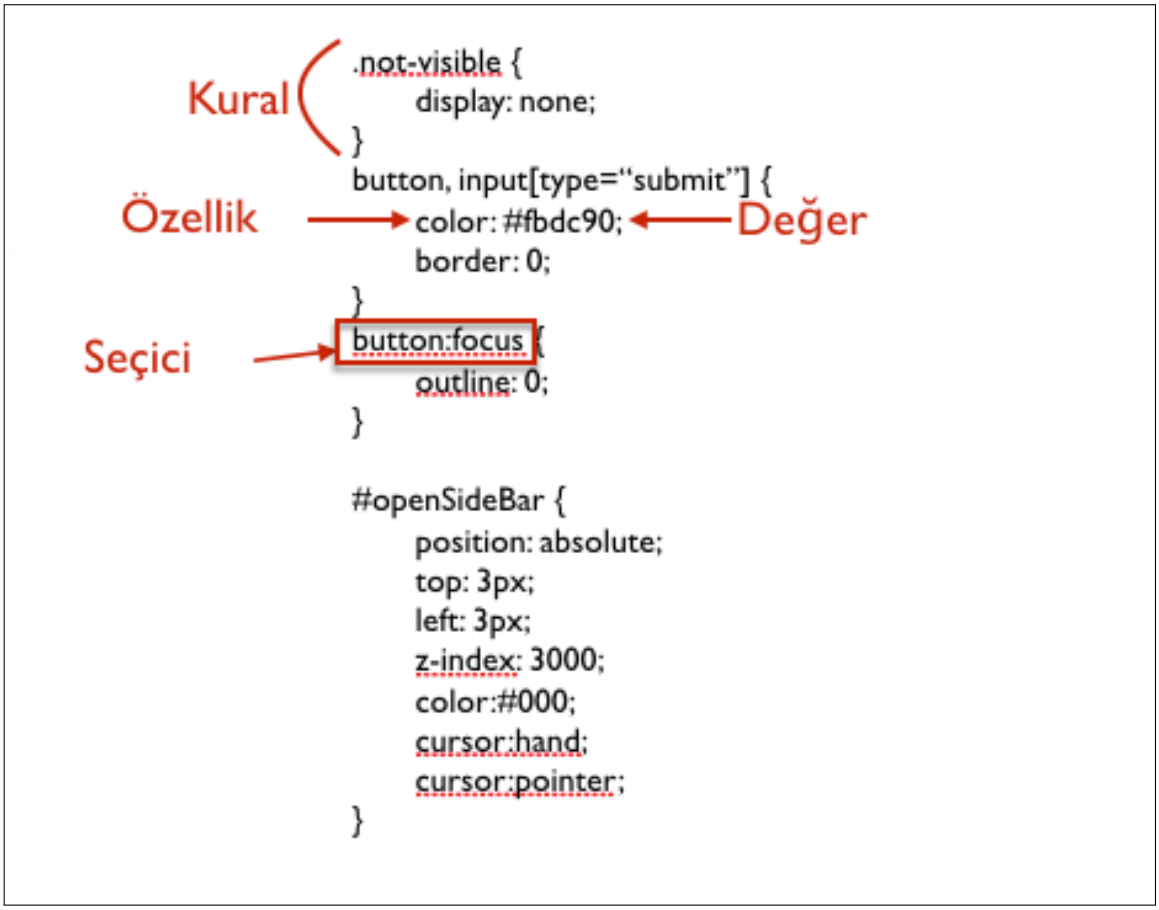
- complexSelectorsByAttribute: Öznitelik bazında karmaşık eşlemeye sahip seçicilerin sayısı (örnek: [class$=”foo”])
- duplicatedProperties: Seçici içerisinde tekrarlanan CSS özellik tanımlarının sayısı
- expressions: CSS ifadeleri içindeki dinamik kuralların sayısı
- oldIEFixes: Internet Explorer’ın eski versiyonları için kullanılan düzeltmelerin sayısı (örnek: * html .foo {} ve .foo { *zoom: 1 })
- importants: !important ifadesi ile zorla önceliklendirilen özelliklerin sayısı
- multiClassesSelectors: Birden fazla sınıf barındıran seçicilerin sayısı (örnek: span.foo.bar)
- oldPropertyPrefixes: Artık gerek duyulmayan, istemciye özel özelliklerin sayısı (örnek: –moz-border-radius)
- qualifiedSelectors: Nitelikli seçici sayısı (örnek: header#nav, .foo#bar, h1.title)
- specificity: Kuralın özgüllüğü. Aşağıdaki gibi hesaplanır: Seçicideki kimlik özelliklerinin sayısı sayılır (= a). Seçicideki diğer niteliklerin ve sahte sınıfların sayısı sayılır (= b). Seçicideki öğe adlarının ve sözde öğelerinin sayısı sayılır (= c). A-b-c’nin üç sayısının birleştirilmesi, özgüllük değerini [130] verir. Aynı kuralın içinde birden fazla seçicinin olması durumunda maksimum özgüllük değeri alınır.
- specificityIdAvg: ID için ortalama özgüllük
- specificityIdTotal: ID için toplam özgüllük
- specificityClassAvg: Sınıf, sahte sınıf veya özellik için ortalama özgüllük
- specificityClassTotal: Sınıf, sahte sınıf veya özellik için toplam özgüllük
- specificityTagAvg: Etiket için ortalama özgüllük
- specificityTagTotal: Etiket için toplam özgüllük
- selectorsByAttribute: Özelliklere göre seçicilerin sayısı (örnek: .foo[value=bar])
- selectorsByClass: Sınıflara göre seçici sayısı
- selectorsById: ID’ye göre seçici sayısı
- selectorsByPseudo: Sahte seçicilerin sayısı (örnek: :hover)
- selectorsByTag: Etiket adına göre seçicilerin sayısı
- specificityCategory: Özgüllük için kategorize edilmiş değer ($\geq 100 = yksek, \geq 10 = orta, < 10 = dk$)
- universalSelectors: Her öğeyi eşleştirmeye çalışan seçicilerin sayısı (örnek: .foo >*)
- length: CSS kuralının uzunluğu (byte cinsinden)
- selectors: Seçici sayısı (örnek: .foo, .bar { color: red } iki olarak sayılır - .foo ve .bar)
- declarations: Tanım sayısı (örnek: .foo, .bar { color: red } tek tanım olarak sayılır - color: red)

6.2.1 CSS Nedir?

CSS, bir işaretleme dili kullanılarak yazılmış web sayfalarının kullanıcıya ne şekilde sunulacağını belirleyen dildir [126]. Çoğunlukla HTML sayfaları için kullanılmakla birlikte XML ta-

banlı herhangi bir işaretleme dili ile de kullanılabilir. Aynı sayfanın farklı ekran veya baskı tiplerine göre farklı şekilde gösterilmesini sağlar. CSS ile içeriğin ve sunumun ayrılması sağlanmaktadır. Bu ayrım geliştiricilere esneklik sağlamakla birlikte aynı stilin farklı sayfalar için de kullanılması açısından kolaylık sağlamaktadır. 1998 yılında RFC 2318 [131] ile birlikte text/css tipinin kullanımı bir standarda bağlanmıştır.

CSS stil sayfaları halinde yazılır. Bir stil sayfası, bir kurallar listesinden oluşur. Her kural veya kural kümesi bir veya daha fazla seçici ile bir bildirim bloğundan oluşur. Örnek bir stil sayfası Şekil 6.4'de görülebilir.



Şekil 6.4 CSS kod örneği

6.2.2 CSS Seçicileri

CSS'de, seçiciler bir biçimlendirme kuralının, biçimlendirmenin hangi bölümünde geçerli olacağını çeşitli kuraların birleşimi ile ifade ederler. Seçiciler aşağıdaki kurallar ile tanımlanabilir:

- Etiketler: h2, a, div gibi element isimleri.
- Sınıflar: Etiketlere verilen sınıf isimleri. Bir etiket birden fazla sınıfa sahip olabilir.
- ID: Doküman içinde tekil bir tanımlayıcı değeri.
- Sahte sınıflar: Doküman içinde yer almayan, ancak dokümanların konumları, durumları vb. özellikleri kullanılarak ifade edilen bilgileri.

Bir dokümanda aynı elemente uyan birden fazla seçici bulunabilir. Burada hangi kuralın önceliğinin olduğuna seçicinin özgüllüğüne göre karar verilir. Çizelge 6.4'de farklı kurallar için hesaplanan özgüllük değeri verilmektedir. Aynı özgüllük değerine sahip kurallar için son (söz dizimi açısından daha aşağıda) tanımlanan geçerli olmaktadır.

Çizelge 6.4 Özgüllük örnekleri

Seçici	**Özgüllük**
a	0, 0, 0, 1
div a	0, 0, 0, 2
.green	0, 0, 1, 0
a.blue	0, 0, 1, 1
div.green a.blue	0, 0, 2, 2
#id1	0, 1, 0, 0
style="color:red"	1, 0, 0, 0

6.2.3 Veri Kümesi

Çalışmanın bu kısmında farklı veri kümeleri kullanmak zorunda kaldık, çünkü daha önce kullanılan projeler CSS kullanımı açısından zengin değildi. Bu durum da metriklerimizi deneyeceğimiz veri kümelerinin yeterince zengin olmamasını getirmektedir. Denemelerimizde kod tabanları GitHub'da bulunmakta olan 5 açık kaynak projenin verilerini kullandık. Projeler CSS kullanımı temel alınarak seçilmiştir.

Çizelge 6.5 Uygulama istatistikleri

Proje	**Kategori**	**Versiyon**	**Geliştirici Sayısı**	**Satır Sayısı (Toplam / CSS)**	**Dosya Sayısı**	**Kod Değişim Sayısı**
2048	Oyun	-	3	3840 / 658	26	3
adarkroom	Oyun	1.2	Bilinmiyor	9861 / 915	38	Bilinmiyor
hextris	Oyun	-	8	6097 / 1726	67	340
effeckt.css	Kütüphane	-	34	64737 / 50126	142	350
bootstrap	Kütüphane	3.2	621	52514 / 9621	341	9496

Seçilen projelerle ilgili bazı istatistikler Çizelge 6.5'den görülebilir. Bu istatistikler, tüm kod tabanlarında belirtilen sürümler için gitstats [124] aracı kullanılarak oluşturulmuştur. Kullandığımız bazı projeler için düzgün bir versiyonlama bilgisi bulunmamaktadır, bu nedenle çizelgede sürüm bilgileri verilememiştir.

6.2.4 Veri Çıkarımı

Bu bölümde, 6.2.3 bölümünde açıklanan 5 projeden elde edilen verileri kullandık. CSS dosyalarını işlemede karşılaşılan genel sorun, projelerde dosya sayılarının diğer dosya tiplerine göre daha az olmasıdır. Genellikle normal bir web uygulaması onlarca veya yüzlerce kaynak dosyasından oluşabilir, ancak CSS dosyası sayısı 10'a bile ulaşmaz. Her dosyayı veri kümemizde farklı bir özellik vektörü olarak düşünürsek, analizin dosya bazında yapılması veri kümelerinde çok az sayıda satır meydana getirebilir ve böylece doğru sonuç alınabilecek bir tahmin modeli kurulamaz. Bu nedenle CSS dosyalarındaki her kurala ilişkin metrikleri ayrı ayrı çıkarıldı ve hatalılık tahmini dosya düzeyinde değil CSS kuralı düzeyinde

gerçekleştirildi. CSS tanımlarının ayrıştırılması ve öznitelik çıkarımı iki açık kaynaklı modül kullanılarak yapılmıştır: analyze-css [132] ve cssselect [133].

Öncelikle projelerin anlık kesitleri belirtilen sürümler için Github sayfalarından indirilmiştir. Her proje için CSS dosyalarından çıkarılan kurallar, anlık görüntü tarihinden sonra değiştirildiyse kusurlu olarak etiketlenmiştir. Bu işlem sırasında her kod değişikliği bir kusur olarak kabul edilmiştir.

Her proje için deneylerde 2 farklı veri kümesi oluşturuldu ve kullanıldı:

- **Tam**: Önerilen metrik seti filtrelenmeden kullanılmıştır.
- **Filtreli**: Tüm CSS metriklerinden oluşan veri kümesine CFS algoritması uygulanarak filtrelenmiş halidir. Filtrelenmiş metrikler Çizelge 6.6'da verilmiştir.

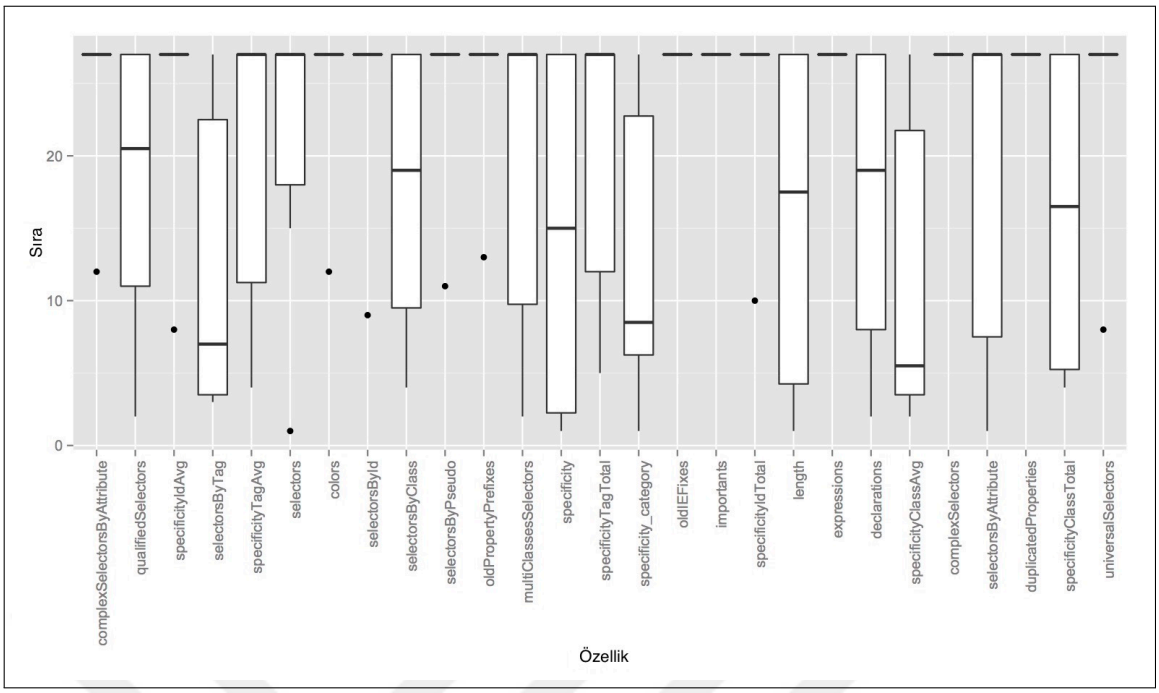
Çizelge 6.6 CSS projelerinde filtrelenmiş metrikler

2048	specificity selectorsByTag multiClassesSelectors length
adarkroom	selectorsByClass qualifiedSelectors selectors
effeckt.css	specificity selectorsByAttribute specificityClassAvg qualifiedSelectors declarations length multiClassesSelectors
hextris	specificityTagTotal declarations length
bootstrap	specificity selectorsByAttribute declarations

6.2.5 Veri Analizi

Bölüm 6.1.2'de HTML kodlarından çıkarılan özellikler için yapılan analiz CSS kodlarından çıkardığımız özellikler için de uygulanmıştır. CSS özellikleri her projede gösterdikleri bilgi kazançlarına göre sıralanmış ve puanlanmıştır. Sıralamalar Şekil 6.5'de incelenmek üzere görselleştirilmiştir. Değerler yerine sıralama derecelerini kullandığımız için, yine az olan değer daha iyidir. Bilgi kazancı değeri sıfır olan özellikler de sıralamada son sıraya alındıkları için sıralamada sonuncu gelmek, birden fazla özellik için görülen en kötü değer olabilir.

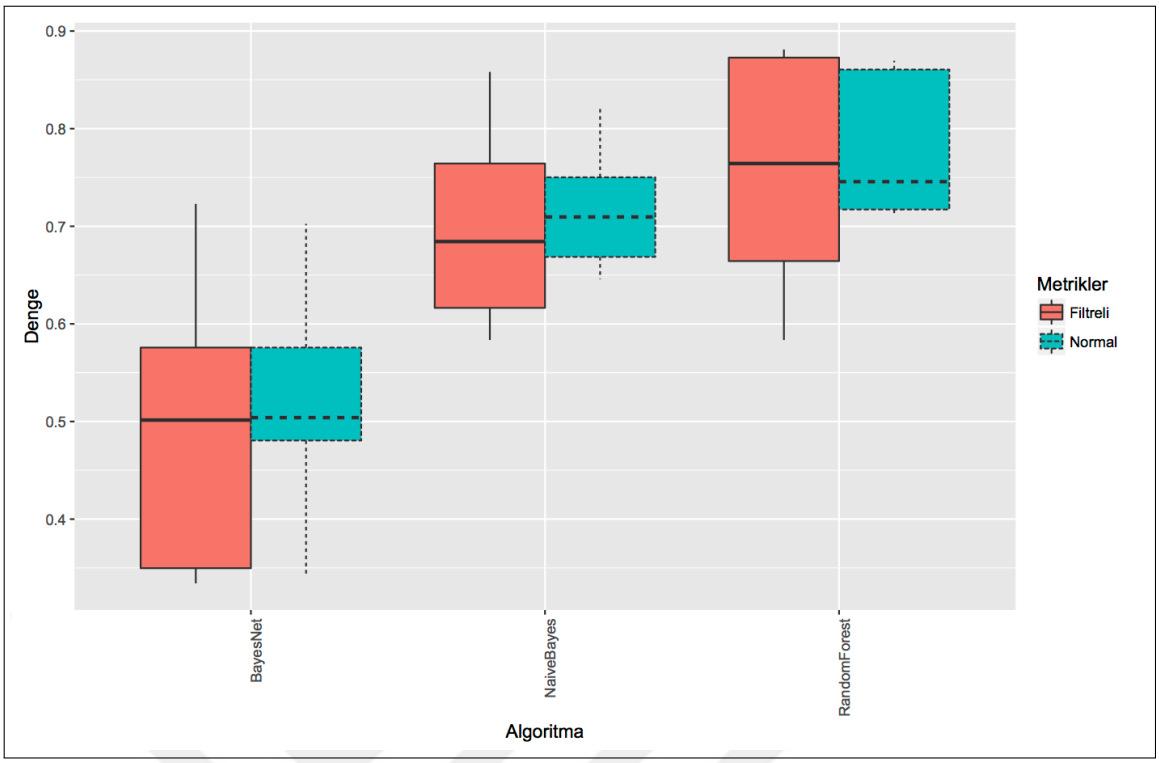
Başlangıç olarak CSS metrikleri için toplam ürettiğimiz 28 metrik bulunmaktaydı, hepsi en az bir kereliğine asgari derecede yer almıştır. Metriklerin 9'u medyan değerlerin son



Şekil 6.5 Özelliklerin bilgi kazanımına göre sıralanması (CSS)

öznitelik indeksinden daha büyük bir medyan değerine sahip olduğunu gösterir. Ölçümlerin 5'i, her zaman en sonuncusu olarak sıralananlara göre gereksiz bulundu. Dolayısıyla, 5 gereksiz nitelik ve 9 öznitelikten CSS sapması öngörüsü için daha fazla olan noktaları kaldırdık ve aynı tahmin modelini veri kümelerimize uyguladık. Kalan özellikler aşağıda bulunmaktadır, vereceğimiz sonuçlar sadece bu metrikler kullanılarak elde edilmiş olacaktır:

- multiClassesSelectors
- qualifiedSelectors
- specificity
- specificityClassAvg
- specificityClassTotal
- specificityTagAvg
- specificityTagTotal
- selectorsByAttribute
- selectorsByClass
- selectorsByTag
- specificityCategory
- length
- selectors
- declarations



Şekil 6.6 Sınıflandırıcılar arası performans karşılaştırması (CSS)

6.2.6 Sonuçlar

5 proje için elimizde 10 veri seti bulunmaktadır. Şimdiye kadar yaptığımız diğer deneylere benzer olarak 3 farklı makine öğrenmesi kullanılarak modelleme yapılmıştır. Deneylerin sonucunda performans karşılaştırması denge ölçütüne göre yapılmıştır, sonuçlar Çizelge 6.7'den görülebilir. Bildiğimiz kadarıyla CSS metrikleri için sonuçlarımızı karşılaştırabileceğimiz başka bir hataya yatkınlık tahmini çalışması bulunmamaktadır. Bu nedenle metriklerimizin performansını insan emeği ile yapılan kod gözden geçirme işlem performansı ile karşılaştırmamız gerekmektedir. Önceki çalışmalar [2] bu işlemin %35 ile %60 arası doğru sonuç verdiğini göstermiştir. Çizelgede %60'tan yüksek sonuç veren değerler koyu olarak işaretlenmiştir.

30 gözlemin 20'sinde denge değerleri %60'tan yüksek bulunmuştur, bu değerler %62 ile %87 arasında değişmektedir. Tahmin performansı ortalama olarak %67 olarak bulunmuştur, özellik seçimi kullanıldığı zaman bu değer %65'tür. Sınıflandırıcılar arası performans karşılaştırması Şekil 6.6'de verilmiştir. Bu şekilde sınıflandırma algoritmalarının performansındaki fark açıktır. Bayes Ağı algoritmasının, CSS metrikleri için Naive Bayes ve Rastgele Orman'dan önemli ölçüde daha az performans gösterdiğini görebilmekteyiz. Bu durum, Bayes Ağı'nın özellikle arası bağlantı araması nedeniyle, özellikle arasındaki şartlı bağımlılıkların düşük olduğu veya var olmadığı ile açıklanabilir. Bayes Ağı algoritmasının sonuçlarını görmezden gelirsek, ortalama performans (0.75, 0.73) değerlerine çıkmaktadır.

Çizelge 6.7 CSS için denge değerleri

Proje	Algoritma	Tam	Filtreli
2048	**NaiveBayes**	**0.73**	**0.86**
	Rastgele Orman	**0.87**	**0.86**
	Bayes Ağı	**0.69**	**0.74**
adarkroom	**NaiveBayes**	**0.82**	0.58
	Rastgele Orman	**0.73**	0.58
	Bayes Ağı	0.44	0.34
effeckt.css	**NaiveBayes**	**0.65**	**0.62**
	Rastgele Orman	**0.85**	**0.84**
	Bayes Ağı	0.51	0.51
hextris	**NaiveBayes**	**0.72**	**0.68**
	Rastgele Orman	**0.67**	**0.66**
	Bayes Ağı	0.31	0.31
bootstrap	**NaiveBayes**	**0.65**	**0.75**
	Rastgele Orman	**0.74**	**0.76**
	Bayes Ağı	0.58	0.58
Ortalama		**0.67**	**0.65**

6.3 Değerlendirme ve Yorumlar

Tez çalışmasının bu kısmında, önerilen metrik setlerinin performansını artırmaya ve farklı veri setleri üzerinde test ederek ve ek performans ölçümleri kullanarak hatalılık tahmininde alana özgü ölçümleri kullanmanın değerini araştırmaya çalıştık.

Sunucu tarafı ve HTML metrik kümesi için, metriklerin değeri, bilgi kazançları kullanılarak denetlenmiş ve metrikler buna göre değerlendirilmiştir. Sonuç olarak, kullanılması planlanan iki metriğin, tahmin katsayısına sahip olmadıkları için, veri kümelerinden kaldırılabileceği bulunmuştur. Sonuçta bazı veri setleri için %23'e kadar performans artışı sağlanmıştır.

CSS metrikleri için 5 proje kullanılmış, başta 28 metrik tanımlanmış, yapılan veri analizleriyle bunların 14 tanesinin kullanımının faydalı olacağı görülmüştür. Literatürde CSS için hatalılık tahmini yapılmadığı için sonuçlarımızı karşılaştırmak için başka bir metrik setimiz bulunmamaktadır. Sonuçlar, CSS dosyalarındaki hatalılık tahmini genel tahmin performansının, manuel kod incelemesi için maksimum seviyede olan $\geq \%60$ olduğu için otomatik araçlarla yapılabileceğini göstermektedir. Bu sonuç, literatürde CSS hataya yatkınlık tahminin ilk defa yapılması açısından önem taşımaktadır. Alana özgü metrik çıkarımının faydalı ve kullanılabilir olduğunun gösterilmesi açısından önemlidir.

Bu bölümde ayrıca bütün metrik tipleri için önerilen metriklerin faydalılığını incelemek üzere veri analizi yapılmıştır. Yapılan analiz özelliklerin bilgi kazancına göre incelenmesi şeklinde olmuştur. İşlem sonucunda her metrik tipi için en anlamlı olarak bulunan metrikler aşağıdaki gibidir:

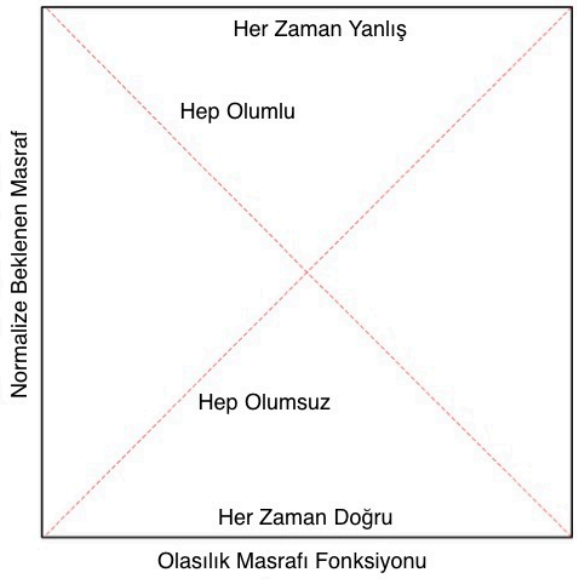
- İstek parametresi kullanımı
- Veritabanı sorgu kullanımı
- Bağlam değişimi

- HTML kodu bulundurması
- İstek parametrelerine erişim sayısı
- Toplam içerik sayısı
- Toplam etiket sayısı
- Tekil etiket sayısı
- Tekil özellik sayısı
- Toplam derinlik
- Kuralın özgüllüğü
- Etiketlere göre seçiciler
- Sınıf ortalama özgüllüğü
- Sınıf toplam özgüllüğü
- Uzunluk

BÖLÜM 7

FAYDA MALİYET ANALİZİ

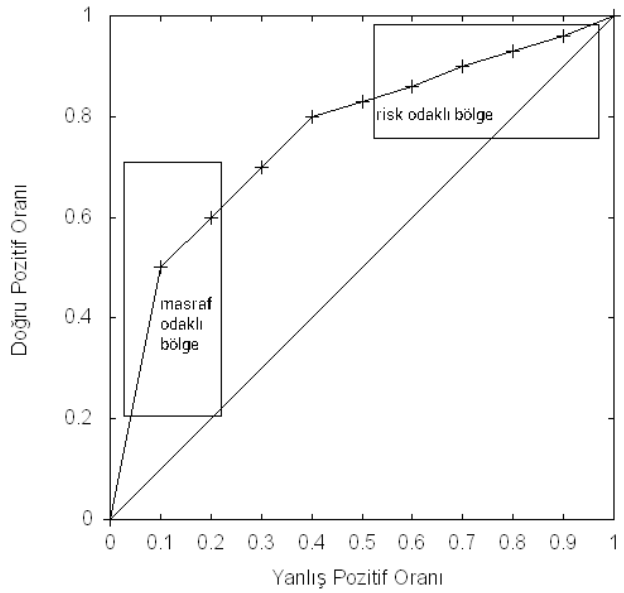
Fayda Maliyet Analizi farklı alternatifleri faydalarının masraflarından daha fazla olup olmadığını anlamak için farklı disiplinlerde uygulamaları olan genel bir terimdir. Makine öğrenmesi alanında sınıflandırıcıların performanslarını değerlendirmek için kullanılmaktadır. Maliyet eğrisi, Drummond ve Holte tarafından ROC eğrilerinin eksikliklerinin sağlanması için önerilmiştir [134]. Örneğin, karşılaştırılan ROC eğrileri kesişiyorsa, biri daha iyi sınıflandırma performansı gösterirken diğerinin görünüşte daha büyük AUC değerine sahip olması muhtemeldir. Birçok pratik uygulamada, karşılaştırılan ROC eğrilerinin kesişiyor olması muhtemeldir. Bunun bir nedeni de karşılaştırmaların genelde benzer performansa sahip sınıflandırıcılar arasında yapılmasıdır. Ancak AUC değerinin daha önemli bir eksikliği ise, farklı sınıflandırıcılar için farklı sınıflandırma maliyeti dağılımı kullanmasıdır. Bu durum ise farklı sınıflandırıcıları değerlendirmek için farklı metrikleri kullanmakla eşdeğerdir. Ayrıca sahte-pozitif ve sahte-negatifler için farklı maliyetler olmasını hesaba katmaz [135], [136].



Şekil 7.1 Masraf eğrisinin bölümleri.

Fayda maliyet analizi sınıflandırıcının performansını yanlış sınıflandırmanın maliyetine göre gösteren bir görselleştirme tekniğidir. X ekseni pozitif sınıf olasılığını, iki yanlış sınıflandırma maliyetinin kombinasyonunu ve sınıf dağılımını tek bir değere işaret eden PC'yi (+)

gösterir. Y ekseni hata oranını belirten Normalleştirilmiş Beklenen Maliyeti (NEC) gösterir. Bu değerler (7.3) ve (7.4) eşitlikleri kullanılarak hesaplanır.

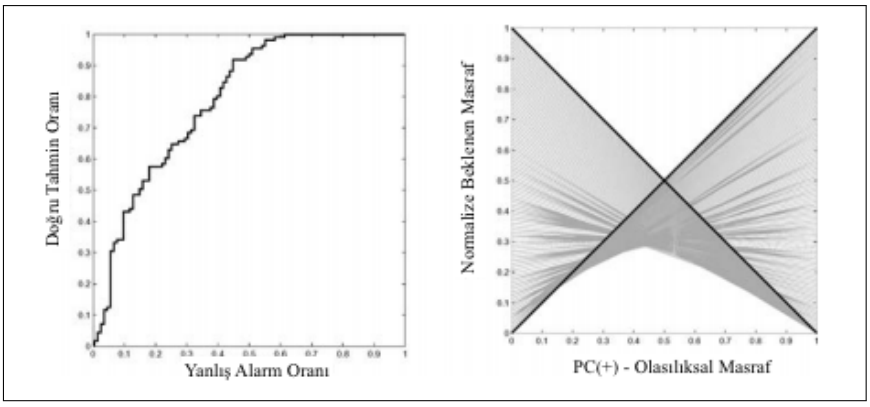


Şekil 7.2 ROC eğrisinin bölümleri [11]

Bir masraf eğrisinin bölümleri Şekil 7.1'de verilmiştir. (0,0) noktasından (1,1) noktasına giden çizgi bütün örnekleri hatasız etiketiyle sınıflandıran bir sınıflandırıcıyı temsil eder. (0,1) noktasından (1,0) noktasına giden çizgi ise bütün örnekleri hataya eğimli etiketiyle sınıflandıran bir sınıflandırıcıyı temsil eder. (1,0) ve (1,1) noktaları arasındaki çizgi 0 doğruluğa sahip bir sınıflandırıcıyı temsil eder. Son olarak (0,0) dan (1,0) a giden çizgi %100 doğruluğa sahip bir sınıflandırıcıyı temsil eder. Bu göstermektedir ki kullanılabilir durumda olan sınıflandırıcılar (0.5,0.5) koordinatındaki kesişim noktasının altında kalmalıdır.

Bu çalışmada Arisholm ve Briand'ın [137] çalışmasına dayanarak maliyet fayda analizi yapılmıştır. Bir tahmin yönteminin kalitesini değerlendirmede pd oranı önemli bir faktördür ancak pf de önemlidir. Bu durum Şekil 7.2'de gösterilmiştir. Şekil 7.2'deki maliyetsiz bölge orta-düşük pd ve düşük pf değerlerine sahiptir. Bir sınıflandırıcı bu bölgeye düşerse, sınırlı doğrulama bütçelerine sahip projeler için yararlı olacaktır. Risksiz bölge, yüksek pd ve yüksek pf değerlerine sahiptir. Yüksek seviye, kusurlu olma eğilimi gösteren modüllerin önemli bir kısmının aslında hatasız olması demektir. Bu durum, tüm bu modüllerin gereksiz yere muayene edilmesine ve çabalar açısından ekip geliştirme ve ekip maliyetinin artmasına neden olur. Güvenlik-kritik sistemlerde bu tercih edilebilir ancak projelerin çoğu bu kategoride değildir. Dolayısıyla, inceleme süresindeki tasarrufların harcanmış çabayla belirlenmesi için (7.1) ve (7.2) eşitliklerini kullandık. Denge oranları ne kadar yüksekse o kadar yüksek kazanç olur.

Masraf eğrileri ROC eğrilerinden türetilebilir. ROC eğrisi üzerindeki her (PD,PF) noktası, (0,PF) ve (1,1-PD) noktaları arasında düz bir çizgi ile temsil edilir. Bütün noktaları çizgilere çevirdikten sonra, en alt kesişim noktaları maliyet eğrisinin alt formunu oluşturur. Bu işlem Şekil 7.3'den görülebilir. Bu form verilen her x değeri için masraf eğrilerinin herhangi birindeki en düşük y değeri olarak da tanımlanabilir. İki sınıflandırıcının alt formlarını karşılaştırırken, aynı x değeri için bulunan y değerinin daha düşük değerde olup olmadığına bakılır. Düşük y değerine sahip sınıflandırıcı daha iyi performansa sahip demektir.



Şekil 7.3 Maliyet eğrisinin oluşturulması [134]

$$Fayda\% = pd - \frac{A+B}{A+B+C+D} \tag{7.1}$$

$$Fayda(dosya) = (pd - \frac{A+B}{A+B+C+D}) x DosyaSayisi \tag{7.2}$$

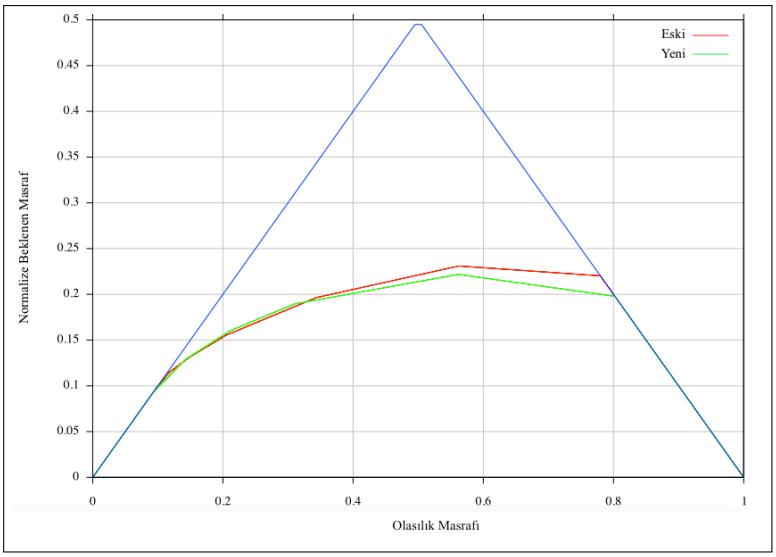
$$PC(+) = \frac{p(+) \times C(-|+)}{p(+) \times C(-|+) + p(-) \times C(+|-)} \tag{7.3}$$

$$NEC = FN \times PC(+) + FP \times (1 - PC(+)) \tag{7.4}$$

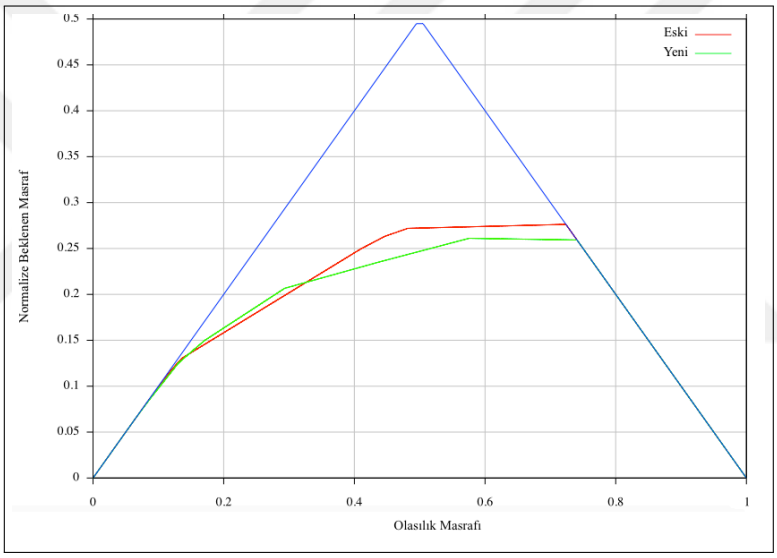
7.1 Sonuçlar

Modelimizdeki tahmin sonuçlarının doğrulama eforunda azalmaya neden olup olmadığına bakmak için sonuçlarımıza maliyet fayda analizi uygulandı. İstemci ve sunucu tarafı ölçümlerimiz için mevcut ölçümlerin performansıyla geleneksel statik kod metriklerinin performansını karşılaştırabiliriz. Ancak CSS metrikleri için karşılaştırılacak mevcut bir metrik seti yoktur, bu nedenle sadece rastgele seçim durumuyla karşılaştırma yapabiliriz. Teorik olarak, rastgele olarak kusurları bulmak için dosyaları seçersek, arızalı dosyaların maksimum %X'ini bulmak için dosyaların %X'ini incelemeliyiz. Hatalılık tahmini çalışmalarının amacı, kaynakları gereksiz yere test etmekten kurtarmak olduğundan, iyi bir hatalılık tahmini modelinden bu çabayı azaltması beklenir.

Modellerin maliyet etkinliğini kıyaslarken belirli bir olasılık eşiğinden bağımsız bir değerlendirmenin olması uygun olur [137]. Bu kıyaslama eğri altındaki yüzey alanını hesaplayarak gerçekleştirilmiştir. Maliyet fayda analizinin metrik kategorilerdeki sonuçları Çizelge 7.1, 7.2, 7.3 ve 7.4'de görülebilir. Çizelge 7.1 ve 7.3'de maliyet eğrilerinin alt formları verilmiştir. Maliyet eğrisi altındaki alan AUC sütununda bulunabilir. Daha düşük AUC değeri maliyet eğrilerinde [134] daha iyi olduğundan, 66 karşılaştırmanın 53'ünde (%80) önerilen metriklerin mevcut statik kod ölçümlerine kıyasla inceleme maliyetini düşürdüğünü görmekteyiz. 66 karşılaştırmanın sadece 10'unda (%15) eski metrik setler maliyet fayda değerlendirmesinde daha iyi performans göstermiştir. Çizelge 7.2 ve 7.4'de doğrulama maliyetindeki kazanç yüzdesi verilmektedir. AUC'nin karşılaştırılmasından farklı olarak,



Şekil 7.4 Filtrelenmemiş Typecho veri kümesi için maliyet eğrisi



Şekil 7.5 Filtrelenmiş Typecho veri kümesi için maliyet eğrisi

varsayılan eşik değeri (0.5) kullanılarak hesaplanır. Bu karşılaştırmadan, PHP tabanlı veri kümeleri için karşılaştırmaların %50'sinde iyileşme görüyoruz. Bununla birlikte, yüzdede fayda azalması AUC'den fazla, %28'dir. Örnek maliyet eğrileri Typecho veri kümesi için, Şekil 7.4 ve 7.5'de verilmiştir. Şekillerden görüleceği üzere farklı olasılık değerleri için farklı masraflar bulunmaktadır. Bu eğrilerin altında kalan alanlar her model için beklenen toplam masraf değerini vermektedir.

Çizelge 7.1 Sunucu ve istemci metrikleri için fayda maliyet analizi

Veri kümesi	Algoritma	Tam	Eski	Tam-Filtreli	Eski-Filtreli
WordPress	**NaiveBayes**	0.183*	0.189	0.166*	0.179
	Rastgele Orman	0.174*	0.180	0.166*	0.182
	Bayes Ağı	0.156*	0.165	0.151*	0.157
Joomla	**NaiveBayes**	0.213*	0.232	0.202*	0.227
	Rastgele Orman	0.171*	0.180	0.185*	0.196
	Bayes Ağı	0.190*	0.210	0.186*	0.205
Drupal	**NaiveBayes**	0.166	0.165*	0.165*	0.167
	Rastgele Orman	0.165*	0.170	0.174*	0.175
	Bayes Ağı	0.160*	0.164	0.152*	0.156
phpMyAdmin	**NaiveBayes**	0.213	0.210*	0.208*	0.213
	Rastgele Orman	0.208	0.208	0.210*	0.215
	Bayes Ağı	0.206	0.206	0.205	0.205
phpPgAdmin	**NaiveBayes**	0.115	0.111*	0.112*	0.113
	Rastgele Orman	0.098*	0.111	0.106*	0.110
	Bayes Ağı	0.083*	0.089	0.077*	0.082
phpBB	**NaiveBayes**	0.233	0.224*	0.206*	0.209
	Rastgele Orman	0.202*	0.205	0.211*	0.215
	Bayes Ağı	0.187	0.184*	0.178	0.177*
ZenPhoto	**NaiveBayes**	0.219*	0.231	0.197*	0.224
	Rastgele Orman	0.181*	0.199	0.189*	0.205
	Bayes Ağı	0.189*	0.235	0.178*	0.233
punBB	**NaiveBayes**	0.174*	0.176	0.164	0.162*
	Rastgele Orman	0.202*	0.204	0.196	0.195*
	Bayes Ağı	0.189	0.175*	0.172	0.168*
Typecho	**NaiveBayes**	0.144*	0.158	0.135*	0.156
	Rastgele Orman	0.159*	0.163	0.178*	0.184
	Bayes Ağı	0.154*	0.157	0.138*	0.146
FluxBB	**NaiveBayes**	0.216*	0.218	0.153*	0.177
	Rastgele Orman	0.219*	0.230	0.157*	0.216
	Bayes Ağı	0.193*	0.249	0.176*	0.249
PyroCMS	**NaiveBayes**	0.216*	0.218	0.212*	0.217
	Rastgele Orman	0.143*	0.144	0.146*	0.148
	Bayes Ağı	0.187*	0.191	0.188*	0.191
Ortalama		**0.184**	**0.186**	**0.171**	**0.183**

Çizelge 7.2 Sunucu ve istemci metrikleri için faydaların yüzdeli karşılaştırması

Veri kümesi	Algoritma	Tam	Eski	Tam-Filtreli	Eski-Filtreli
WordPress	**NaiveBayes**	26*	22	27*	18
	Rastgele Orman	28	28	32*	27
	Bayes Ağı	38*	31	37*	35
Joomla	**NaiveBayes**	16*	3	8*	4
	Rastgele Orman	21*	20	18	18
	Bayes Ağı	17*	16	18*	17
Drupal	**NaiveBayes**	16*	15	15	16*
	Rastgele Orman	16	17*	15	16*
	Bayes Ağı	18	18	18	18
phpMyAdmin	**NaiveBayes**	22	24*	22*	20
	Rastgele Orman	15	16*	19*	16
	Bayes Ağı	22	22	19	20*
phpPgAdmin	**NaiveBayes**	39*	35	34*	23
	Rastgele Orman	46*	44	44	44
	Bayes Ağı	51	51	50	51*
phpBB	**NaiveBayes**	13	19*	15	16*
	Rastgele Orman	14	16*	14*	12
	Bayes Ağı	29	29	11	12*
ZenPhoto	**NaiveBayes**	6*	5	9*	5
	Rastgele Orman	10	10	10	10
	Bayes Ağı	12*	2	13*	2
punBB	**NaiveBayes**	34*	26	34*	33
	Rastgele Orman	24*	23	24	30*
	Bayes Ağı	35	37*	35*	34
Typecho	**NaiveBayes**	40*	35	34*	29
	Rastgele Orman	21	22*	20	22*
	Bayes Ağı	47*	42	46*	41
FluxBB	**NaiveBayes**	14	14	22*	18
	Rastgele Orman	18*	14	28*	18
	Bayes Ağı	26*	3	27*	3
PyroCMS	**NaiveBayes**	11	11	12	12
	Rastgele Orman	31*	30	30	30
	Bayes Ağı	17	18*	23*	21
Ortalama		**24**	**21**	**24**	**21**

Çizelge 7.3 CSS metrikleri için fayda maliyet analizi

Veri kümesi	Algoritma	Tam	Filtreli
2048	**NaiveBayes**	0.039	0.049
	Rastgele Orman	0.060	0.044
	Bayes Ağı	0.144	0.113
adarkroom	**NaiveBayes**	0.136	0.121
	Rastgele Orman	0.099	0.101
	Bayes Ağı	0.208	0.216
effeckt.css	**NaiveBayes**	0.182	0.191
	Rastgele Orman	0.067	0.072
	Bayes Ağı	0.183	0.181
hextris	**NaiveBayes**	0.132	0.121
	Rastgele Orman	0.159	0.160
	Bayes Ağı	0.226	0.248
bootstrap	**NaiveBayes**	0.183	0.122
	Rastgele Orman	0.156	0.123
	Bayes Ağı	0.178	0.178
Ortalama		**0.129**	**0.134**

Çizelge 7.4 CSS metrikleri için faydaların yüzdeli karşılaştırması

Veri kümesi	Algoritma	Tam	Filtreli
2048	**NaiveBayes**	97	93
	Rastgele Orman	95	96
	Bayes Ağı	82	85
adarkroom	**NaiveBayes**	79	83
	Rastgele Orman	90	86
	Bayes Ağı	69	63
effeckt.css	**NaiveBayes**	77	74
	Rastgele Orman	94	94
	Bayes Ağı	78	78
hextris	**NaiveBayes**	81	83
	Rastgele Orman	80	65
	Bayes Ağı	43	44
bootstrap	**NaiveBayes**	69	81
	Rastgele Orman	72	80
	Bayes Ağı	71	71
Ortalama		**78.8**	**78.9**

7.2 Değerlendirme ve Yorumlar

Çalışmanın bu bölümünde önerilen metrik kümesinin değerini daha da incelemek için maliyet-fayda analizi uygulanmıştır. Bu analiz tipi elde edilen değerlerin gerçek dünyada ne kadar fayda sağlayacağını ölçmek için kullanılmaktadır. Karşılaştırmaların %24'ünde sunucu ve HTML metrikleri için önerilen metrik kümesi kullanılarak, mevcut metriklere

kıyasla, inceleme maliyetinin düşürüldüğü görülmüştür. Bu değer, önerilen metrik setinin de eklenmesiyle birlikte rastgele teste kıyasla pratikte ortalama %24 daha az dosyanın incelenmesi gerekeceği anlamına gelmektedir, diğer metrik setine kıyasla da test eforunda %15 daha az dosyanın incelenmesi gerekecektir. Bu sonuç, önerilen metriklerin değerini gösterir, çünkü hataya yatkınlık tahmini yöntemlerinin kullanımındaki, gereksiz incelemeden kaçınarak, test maliyetini düşürme amacına tamamen uygundur. CSS metrikleri için bulunan masraf iyileşmesi, HTML metriklerinin kullanıldığı duruma göre daha fazladır. Hatalılık tahmini kullanmanın faydası rasgele teste kıyasla yaklaşık %79 daha az dosya incelemesine eşittir.

BÖLÜM 8

OPTİMİZASYON ALGORİTMALARININ HATA TAHMİNİNE UYGULANMASI

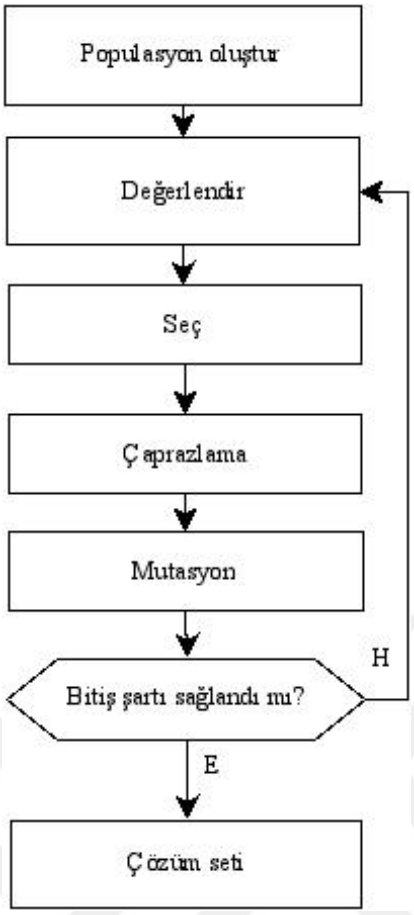
Bu bölümde, makine öğrenmesi alanında kullanılan optimizasyon algoritmaları, elde edilen veri setine uygulanmış, şimdiye kadar kullandığımız makine öğrenmesi algoritmalarının performansları ile karşılaştırmalar yapılmıştır.

8.1 Genetik Programlama

Genetik algoritmalar; ekonomik teorideki uyarlanabilir ajanlardan uçak türbinleri ve entegre devreler gibi karmaşık cihazların tasarımındaki makine öğrenme tekniklerine kadar değişen karmaşık adaptif sistemler üzerinde yapılan araştırmalarda giderek daha önemli bir rol oynamaktadır [138]. Genetik algoritmalar (GA), doğal seleksiyon ve genetiğin evrimsel fikirlerine dayalı, adaptif buluşsal arama algoritmasıdır. Bunlar, optimizasyon problemlerini çözmek için kullanılan rasgele bir aramanın akıllı bir kullanımını temsil eder. Yapılarında her ne kadar şans faktörü olsa da, genetik algoritmalar hiçbir şekilde tesadüf değildir; aksine aramayı, arama alanındaki daha iyi performans gösterebilecekler bölge içine yönlendirmek için geçmiş bilgileri kullanırlar. Genetik algoritmaların temel teknikleri, Charles Darwin'in "en sağlıklı olanın hayatta kalması" ile ilklendirdiği ilkelere dayanmaktadır. Doğada, yetersiz kaynaklar için bireyler arasındaki rekabet, en zayıf olanları ortadan kaldırırken, en iyi bireylerin hayatta kalmasına neden olur.

GA sürecinin yapısı oldukça basittir. Şekil 8.1'de verilmiştir. Genetik Programlama (GP) da GA'nın bir uzantısıdır. Algoritma rastgele başlatılmış bir kromozom popülasyonu oluşturmayla başlar. Bizim çalışmamızda kullanılan yöntem Genetik Programlama olduğu için, kromozomlar rasgele oluşturulmuş tahmin setleri olup, temel matematiksel fonksiyonlar (toplam, çarpma, min, maks, logaritma vb.) kullanılır ve eğitim setinin genel uygunluğunu hesaplanır. Eğitim değeri her eğitim setinde hesaplanır. Uygunluk genellikle bir soruna özgü bir işlevle hesaplanır, ancak makine öğrenmesi problemlerinde bu mümkün değildir çünkü asıl problem verilen örneğin sınıfını bulmaktır. Çocuk kromozomları, iki ebeveyn kromozomunun uygunluğuyla doğru orantılı bir olasılıkla seçilerek birleştirilmesiyle üretilir. Bu işleme çaprazlama denir. İki ana kromozomun parçaları tek bir kromozomda birleştirilir. Bu parçalar rasgele seçilir. Düşük uygunluk değerine sahip kromozomlar eğitim setinden çıkarılır. Olgular, olasılıklarla değiştirilebilir (genellikle 0.01 kadar düşük olarak belirlenmiştir). Algoritma maksimum yineleme sayısı veya en iyi uygunluk değerine ulaşılana kadar devam eder.

Sınıflandırma, verilen örneğin uygunluk değerini en iyi kromozomları kullanarak ve her sı-



Şekil 8.1 Genetik algoritma

nıf için ortalama hesaplayarak yapılır. Yüksek puanı kazanmış olan sınıf kazanır ve örnek o sınıfla sınıflandırılır. Genetik algoritmalar, nüfus üretme ve çaprazlamada rasgele seçimle çalıştığı için, her bir çalışmada farklı puanlara neden olurlar. Bu etkiyi ortadan kaldırmak için algoritmalar 10 kere çalıştırılmış ve ortalamaları alınmıştır. Genetik Programlama algoritmasının uygulamasında Weka'da kullanılan varsayılan değerler Çizelge 8.1'deki gibidir.

Çizelge 8.1 GP için varsayılan değerler

Parametre	**Değer**
eliteSize	5
maxDepth	5
fitnessEvaluatorCode	OneClass
programSelectorCode	FitnessProportional

8.2 Yapay Bağışıklık Sistemleri

Yapay Bağışıklık Sistemleri (YBS) doğadan ilham alınarak ortaya çıkan, hesaplamalı olarak akıllı sistemlerin bir başka ailesidir. Bu algoritma ailesi, memelilerin bağışıklık sisteminin ilkelerinden ve süreçlerinden esinlenmiştir. YBS hakkındaki erken araştırmalar, 80'li yıl-

larda Farmer, Packard ve Perelson'un Immune Networks [139] üzerine yazdığı makalelerle başlamıştır.

Bağışıklık sistemi, yeni durumları öğrenmek ve uyarlamak yeteneğine sahiptir [139]. Karmaşık bilgi işleme görevlerini gerçekleştirebilir. En üst düzeyde sorumluluğu karşılaştığı yabancı molekülleri (antijenler) tanımak ve onları organizmadan temizlemektir. Bu süreçte antijenin yapısını öğrenir ve gelecekteki karşılaşmalar için kendisini aynı antijeni tanımaya adapte eder [140].

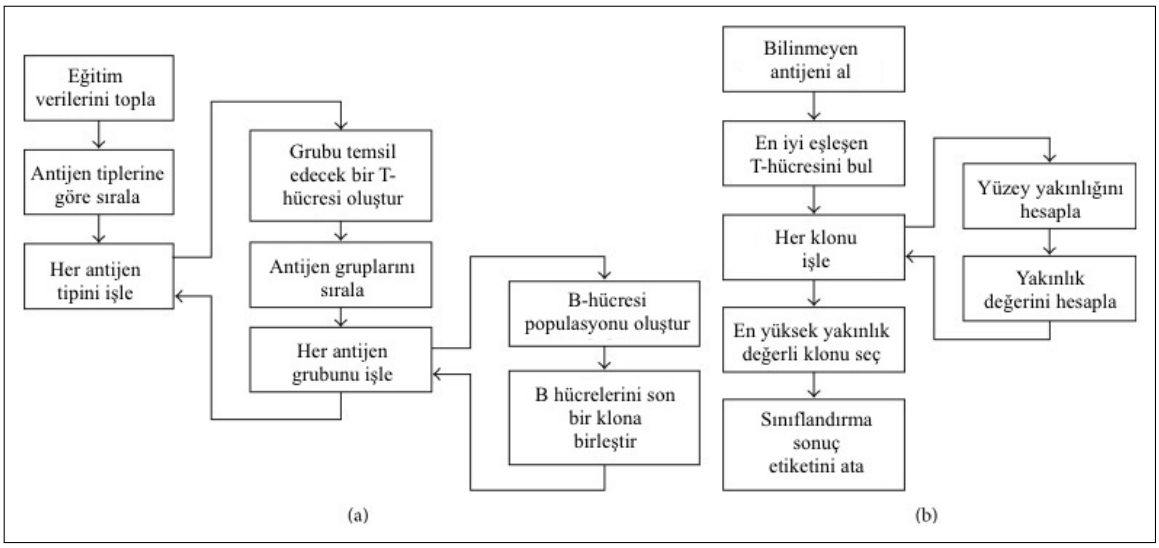
Bağışıklık sisteminin ana bilgi işleme faaliyetleri, esas itibariyle iki tür hücrenin antijen sunum hücreleri (APC'ler) ve lenfositlerin etkileşimlerinin sonucu olarak ortaya çıkar. Lenfositler iki şekilde bulunur: B hücreleri ve T hücreleri. T hücreleri timusta üretilir ve yanıtı protein antijenlerine yönlendirirler. Her T hücresi, antijeni tanıma yeteneğine sahip, eşsiz bir dizi yüzey reseptörü taşımaktadır [141]. B hücreleri kemik iliğinde üretilir ve yüzeylerinde immünoglobülin (IgM ve IgD) taşırlar. Belirli bir hücredeki tüm immünoglobülinler özdeştir ve antijenle [142] bağlanma yeneteğine sahiptir.

Bir B hücresi veya T hücresi, yüzey reseptörleri için yeterli yakınlığa sahip bir antijenle karşılaştığında, hücreler harekete geçirilir. Bu gerçekleştiğinde B ve T hücreleri bölünür. Bu tepki kışkırtıcı antijen için orijinal reseptörün yakınlık derecesi ile orantılı görünmektedir. Bu nedenle, antijenle en kuvvetli şekilde bağlanan B ve T hücre çiftleri, zayıf bağlanma yakınlıklarına sahip olanlardakinden daha fazla hayatta kalan nesiller üretir. B hücreleri, bu şekilde "yakınlık olgunlaşması" olarak adlandırılan olağanüstü bir dönüşüme uğrarlar. Yakınlık olgunlaşması, aktive olmuş hücrelerin bazı nesillerinin (antijen bağlama sahalarını kodlayan DNA üzerinde nokta mutasyonları yoluyla) kışkırtıcı antijen için ana hücrede mevcut olana kıyasla daha da yüksek bir yakınlıkla antikor üretme sürecini tanımlar. Böylece, bağışıklık yanıtı ilerledikçe, sistem işgalci proteinleri daha iyi tanır ve böylece nasıl temizlediğini öğrenir. Yanıtın son kısmı hafıza hücreleri oluşumu ile belirtilir. Bunlar sistemde aylarca ya da yıllarca kalmış olan B hücreleri ve T hücreleridir ve daha önce tanınmış antijenlere, aktif hale getirilmiş hücrelerden çok daha kısa sürede ve daha fazla aktif olarak tepki verme yeteneğine sahiptirler [140].

8.2.1 Tanımlar

Bu bölümde YBS'de kullanılan bazı kavramlar makine öğrenmesi açısından açıklanmaktadır.

- *Artificial Recognition Ball (ARB)*: aynı zamanda B hücresi olarak bilinir. Bir antikor, hücre tarafından tutulan kaynak sayısının bir sayısı ve hücrenin mevcut uyarılma değerinden oluşur.
- *Antikor*: Özellik vektörüdür.
- *Antijen*: Eğitim verisidir.
- *Yakınlık*: iki antikor veya antijen arasındaki "yakınlık" veya benzerlik ölçüsü. Mevcut uygulamada bu değerin 0-1 aralığında olduğu garanti edilir ve basitçe iki nesnenin özellik vektörlerinin Öklid uzaklığı olarak hesaplanır. Böylece, küçük yakınlık değerleri, güçlü yakınlığı belirtir.
- *Memory cell (mc)*: Verilen bir eğitim antijeni ile etkileşim sonucu o antijenden en fazla uyarılan ARB'nin antikordur. Gelen eğitim antijenlerine cevap olarak hipermutasyon için kullanılır. Bununla birlikte, bir mc değiştirilebilir. Bu yalnızca, bir aday



Şekil 8.2 Immunos-81 [144]
(a) Eğitim (b) Test

mc'nin, en çok uyarılmış belirlenen mc'den belirli bir antreman antijenine göre daha fazla uyarıldığında ve belirlenen mc ile aday mc arasındaki yakınlık, Yakınlık Eşiği ve Yakınlık Eşik Sayısı ürününden daha az olduğunda oluşur.

8.2.2 Immunos-81

Immunos-81 bir tıp doktoru olan Carter tarafından tasarlanmıştır [140]. Bağışıklık sisteminden etkilenen ilk sınıflandırma sistemi olduğu iddia edilmektedir. Bu çalışmada Immunos-81 algoritmasının ilk versiyonunu ele aldık. Sistem antijenleri sınıflarına göre gruplara bölerek eğitilir. Daha sonra bilinmeyen bir antijen giriş yaptığında sistem her grup için ilgi ve istek değerlerini hesaplar ve antijeni en iyi yakınlık değerini gösteren gruba ait olarak sınıflandırır [143]. Algoritmanın akış diagramı Şekil 8.2'de verilmiştir. Yakınlık ve istek değerlerinin hesaplanması her grup için Eşitlik (8.1) ve (8.2) de verilmiştir.

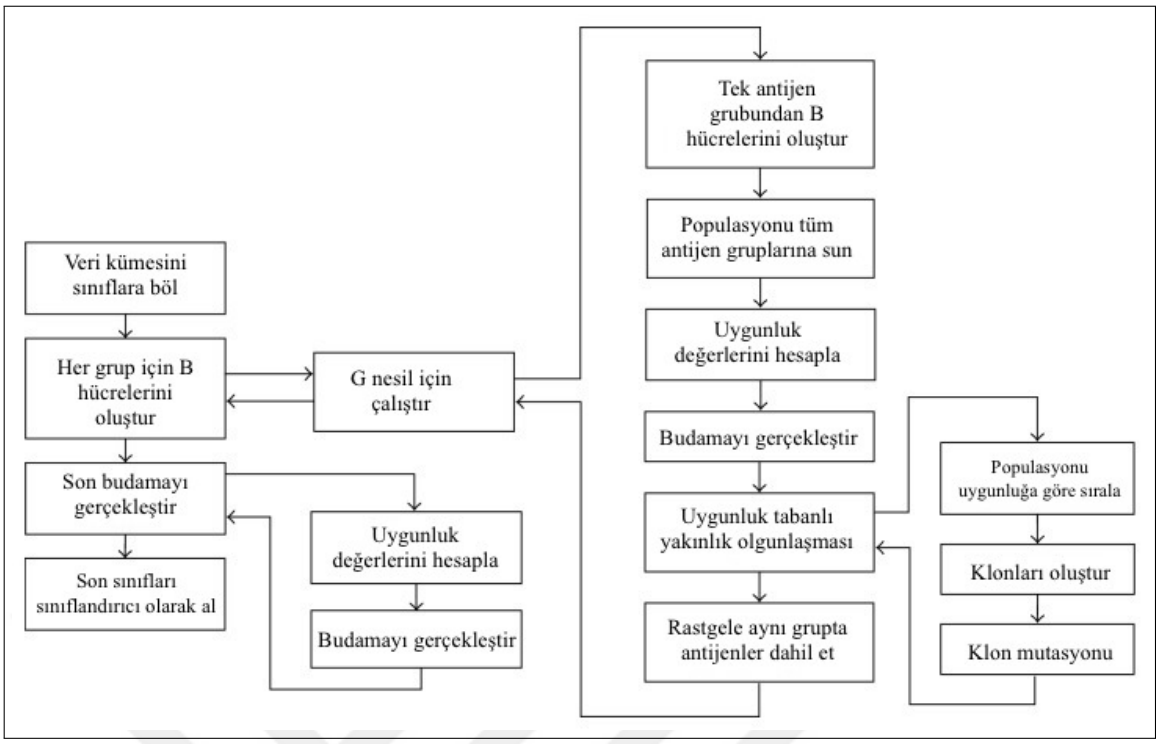
$$yakinlik = \sum_{i=1}^{n} \sqrt{\sum_{j=1}^{k} mesafe(a_i j, b_i j)^2} \tag{8.1}$$

$$istek = n/yakinlik \tag{8.2}$$

Bu eşitlikte n değeri gruptaki örnek sayısını, k değeri a ve b örneklerindeki özellik sayısını temsil eder.

8.2.3 Immunos-99

Brownlee [143] tarafından ortaya atılan Immunos-99, Immunos-81 algoritmasının gelişmiş bir halidir. Immunos-81'deki faydalı ve sistem için tekil görünen elementlerden fay-



Şekil 8.3 Immunos-99 [144]

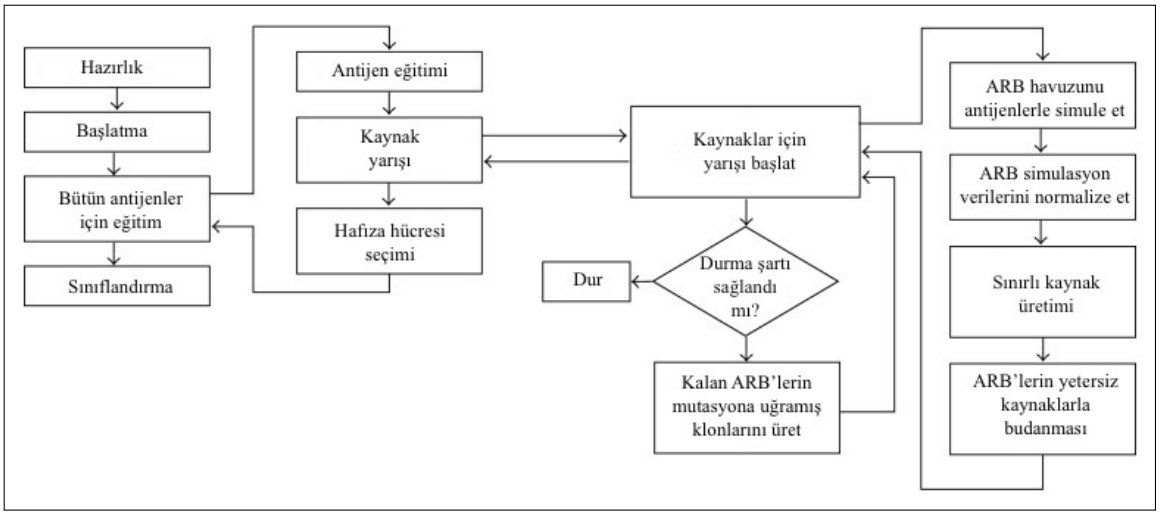
dalanarak hücre üretimi ile mutasyon tekniklerini diğer bağışıklık sistemi tekniklerinden entegre eder.

Immonos-81'de olduğu gibi, eğitim verisi sınıflandırma etiketleri ile gruplara bölünür. Bundan sonra her grup için bir başlangıç B hücresi popülasyonu oluşturulur. Kullanıcı tarafından belirlenmiş jenerasyon sayısı için B hücreleri bütün gruplardan bütün bilinen antijenlere maruz bırakılır. Bu demektir ki her B hücresi bütün antijenlerle karşılaştırılır ve yakınlık ayrı ayrı hesaplanarak sınıf sayıları belirlenir. Uygunluk değeri hesaplanmış sınıf sayıları kullanılarak hesaplanır. Budama işlemi eşik değerinden daha düşük uygunluğa sahip B hücrelerini çıkararak yapılır. Eğer eşik değeri tanımlanmamışsa grup ortalaması alınır. Gruba yeni üye eklemeleri en iyi üyelerin klonlanması ile yapılır ve mutasyon azalan olasılıkla gerçekleştiriliyor olabilir. Üretilmiş klonların sayısı budama aşamasında çıkarılmış B hücrelerinin sayısına eşit olmalıdır. Sınıflandırma işlemi Immunos-81'e benzer şekilde yapılmaktadır. Algoritmanın tamamı Şekil 8.3'de görülebilir.

Immunos-99 algoritmasının implementasyonunda kullanılan varsayılan değerler Çizelge 8.2'de görülebilir.

Çizelge 8.2 Immunos-99 için varsayılan değerler

Parametre	**Değer**
seedPopulationPercentage	0.2
minimumFitnessThreshold	-1
seed	1



Şekil 8.4 AIRS algoritması [144]

8.2.4 AIRS

Yapay bağışıklık tanıma sistemi (AIRS), Watkins [145] tarafından uygulanmıştır. Bu algoritma aslında sınıflandırıcının k en yakın komşusunu kullanarak kümeye dayalı bir uygulamasıdır. AIRS veri azaltımı ile genellemeler yapar. Bununla birlikte k en yakın komşu sınıflandırma için tüm eğitim verilerini kullanmaktadır. Kısacası AIRS algoritması eğitim verisi setini temsil etmek üzere yapay tanıma toplarının (ARB'ler) (bellek hücreleri) bir popülasyonunu oluşturmakla ilgilidir [144]. ARB eşleşen veya spesifik tanıma hücreleridir. Yakınlık hesaplamasının ardından mutasyona uğramış klonlardan toplanırlar. Bir antijen simülasyonu oluşturmak amacıyla en iyi çalışanlar sınırlı kaynaklar için yarıştırıldıktan sonra seçilirler. AIRS algoritması [145], ARB havuzundaki bir antijen için en iyi eşleşme belleğinin mutasyona uğramış klonlarını kullanarak bir sınıflandırma yapar. Algoritma süreci Şekil 8.4'de verilmiştir.

Çizelge 8.3 AIRS için varsayılan değerler

Parametre	**Değer**
affinityThresholdScalar	0.2
totalResources	150
mutationRate	0.1
stimulationValue	0.9
clonalRate	10
hypermutationRate	2.0
numInstancesAffinityThreshold	-1
arbInitialPoolSize	1
memInitialPoolSize	1
seed	1

Watkins ayrıca [146]'te kendi orijinal algoritmasının geliştirilmiş bir versiyonunu önermişti. Genel yapıyı değiştirmeden aşağıdaki değişiklikler yapılmıştır:

- Başlatma sırasında hem MC havuzu (M) hem de ARB havuzu (P) değil, yalnızca bellek hücresi havuzu eklenmiştir.

- Kaynaklar sadece aynı sınıfın ARB'leri için ayrılmıştır.
- Mutasyon sırasında, bir hücrenin uyarılma seviyesi dikkate alınır. Her bireysel genin sadece bir sınırda değişmesine izin verilir. Uyarlama düzeyi düşük olduğunda sınır daha yüksektir.
- Eğitim durdurma ölçütü artık tüm sınıflardaki ARB'lerin uyarılma değerini dikkate almamaktadır.

AIRS algoritmasının implementasyonunda kullanılan varsayılan değerler Çizelge 8.3'deki gibidir.

8.3 Sonuçlar

Bu bölümde, önerilen metrik kümeleri üzerinde optimizasyon algoritmaları (Genetic Programlama, Immunos81, Immunos99, AIRS1, AIRS2) kullanılarak oluşturulan deneylerin sonuçları ile daha önce kullanılan algoritma sonuçlarının (Naive Bayes, Rastgele Orman, Bayes Ağı) karşılaştırmalarını içermektedir.

Her proje için, 4 farklı özellik vektörü tipi kullanılmıştır:

- **Yeni**: Statik kod metrikleri ile beraber önerilen metrik seti kullanılmıştır. Kontrollü deneyimizde deney grubu rolündedir.
- **Eski**: Sadece statik kod metrikleri kullanılmıştır. Kontrollü deneyde kontrol grubu rolündedir.
- **Yeni-Filtreli**: Yeni özellik vektörünü içeren veri kümelerinin CFS algoritması ile filtrelenmiş haline uygulanmış ve sadece filtreden geçen vektör kümesi kullanılmıştır.
- **Eski-Filtreli**: Eski özellik vektörünü içeren veri kümelerinin CFS algoritması ile filtrelenmiş haline uygulanmış ve sadece filtreden geçen vektör kümesi kullanılmıştır.

Çizelge 8.4'da Bayes Ağı ve Rastgele Orman algoritmalarının verilen metrik setler için halen en başarılı algoritmalar olduklarını görebiliriz. Bunları Naive Bayes ve Genetik Programlama izlemektedir. En düşük algoritmalar YBS ailesi olarak bulunmuştur. Deneyler farklı parametreler kullanarak tekrarlanmış, ancak bu da performanslarını iyileştirmede yardımcı olamamıştır. Denemeler hem varsayılan hem de özelleştirilmiş parametrelerle yapılmıştır. Örneğin GP için, denemeler yineleme başına 2 ve 10 genetik değişimle gerçekleştirilmiştir. Immunos-99 için, 1 ila 10 kuşak için deneyler tekrarlanmıştır. AIRS için, kNN için k değeri değiştirilerek kullanıldı ve k= 3 ve k= 5 ile denenmiştir. Performans karşılaştırması Şekil 8.5, 8.6 ve 8.7'den de görülebilir.

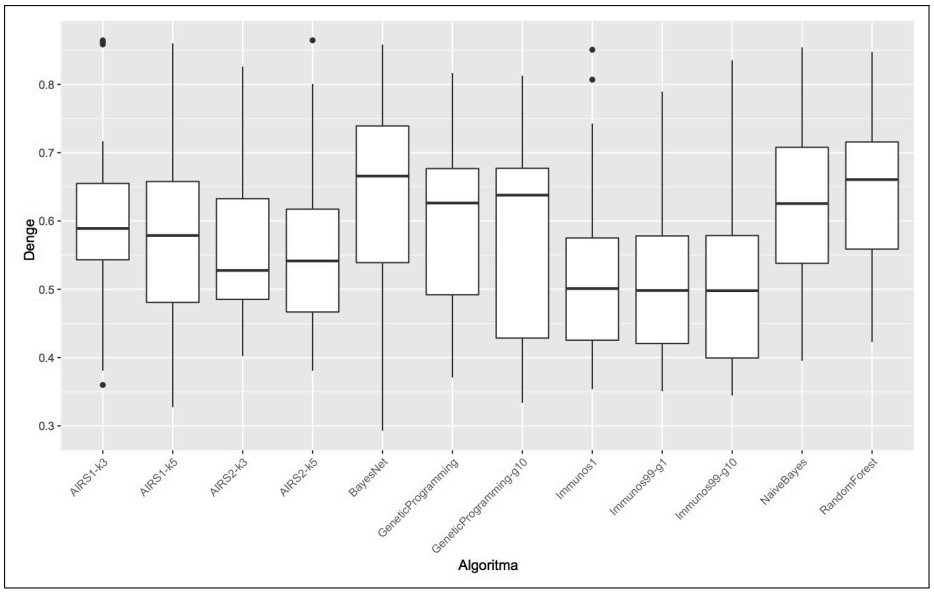
CSS metrikleri için de benzer sonuçlar gözlemlenmiştir. Ortalama olarak, Rastgele Orman'ın en iyi sonuçları verdiği ve Naive Bayes ile GP'nin ikinci sırada yer aldığı görülebilir. YBS ailesi CSS metrikleri için daha yüksek değerlerle sonuçlanmıştır, ancak halen GP'nın arkasında yer almaktadır. Sonuçlar Çizelge 8.5 'de verilmiştir. Karşılaştırmalar Şekil 8.8, 8.9 ve 8.10'de de görülebilir.

Çizelge 8.4 Tüm metrik kümesi için denge değerleri (sunucu ve HTML)

Veri Kümesi	Naive Bayes	Rastgele Orman	Bayes Ağı	Immunos-81	Immunos-99 g=1	Immunos-99 g=10	AIRS-1 k=3	AIRS-1 k=5	AIRS-2 k=3	AIRS-2 k=5	GP cr=2	GP cr=10
WordPress	0.64	0.66	0.77	0.57	0.53	0.57	0.63	0.59	0.52	0.56	0.67	0.63
Drupal	0.70	0.70	0.74	0.73	0.70	0.71	0.71	0.70	0.67	0.69	0.68	0.66
PyroCMS	0.49	0.81	0.61	0.36	0.39	0.37	0.57	0.56	0.55	0.56	0.66	0.66
Joomla	0.67	0.71	0.69	0.48	0.50	0.49	0.62	0.62	0.63	0.61	0.68	0.68
PhpBB	0.48	0.42	0.66	0.47	0.46	0.44	0.36	0.33	0.45	0.39	0.37	0.33
PunBB	0.66	0.56	0.69	0.60	0.50	0.58	0.55	0.48	0.47	0.47	0.57	0.54
FluxBB	0.56	0.65	0.73	0.54	0.63	0.54	0.61	0.63	0.50	0.50	0.60	0.64
PhpPgAdmin	0.77	0.82	0.85	0.85	0.79	0.84	0.87	0.81	0.83	0.86	0.81	0.81
PhpMyAdmin	0.57	0.50	0.61	0.40	0.38	0.38	0.52	0.50	0.53	0.51	0.49	0.42
ZenPhoto	0.51	0.58	0.69	0.60	0.54	0.39	0.56	0.55	0.53	0.55	0.51	0.39
Typecho	0.66	0.47	0.74	0.43	0.47	0.43	0.38	0.40	0.44	0.40	0.44	0.39
Ortalama	0.61	0.63	0.71	0.55	0.54	0.52	0.58	0.56	0.56	0.55	0.58	0.56

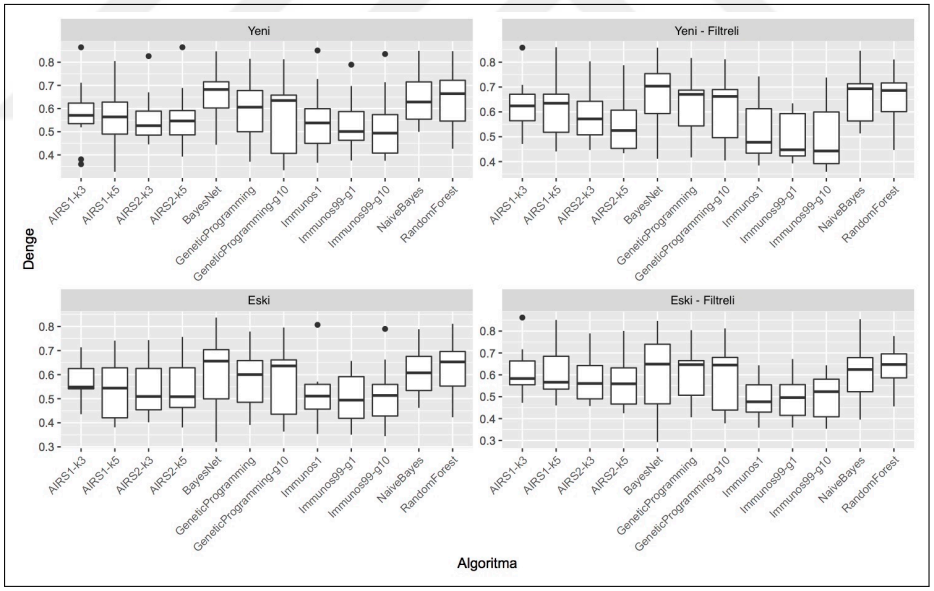
Çizelge 8.5 Tüm metrik kümesi için denge değerleri(CSS)

Veri Kümesi	Naive Bayes	Rastgele Orman	Bayes Ağı	Immunos-81	Immunos-99 g=1	Immunos-99 g=10	AIRS-1 k=3	AIRS-1 k=5	AIRS-2 k=3	AIRS-2 k=5	GP cr=2	GP cr=10
2048	0.73	0.87	0.69	0.64	0.64	0.64	0.85	0.77	0.72	0.70	0.89	0.86
adarkroom	0.82	0.73	0.44	0.80	0.68	0.77	0.78	0.72	0.63	0.62	0.73	0.55
hextris	0.72	0.71	0.31	0.69	0.59	0.64	0.65	0.56	0.61	0.58	0.71	0.74
bootstrap	0.65	0.74	0.58	0.74	0.61	0.73	0.67	0.58	0.68	0.65	0.70	0.62
effeckt	0.65	0.85	0.51	0.74	0.67	0.73	0.62	0.59	0.60	0.59	0.56	0.34
Ortalama	0.72	0.78	0.51	0.72	0.63	0.71	0.71	0.65	0.65	0.63	0.71	0.63



Şekil 8.5 Performans karşılaştırması (HTML ve sunucu)

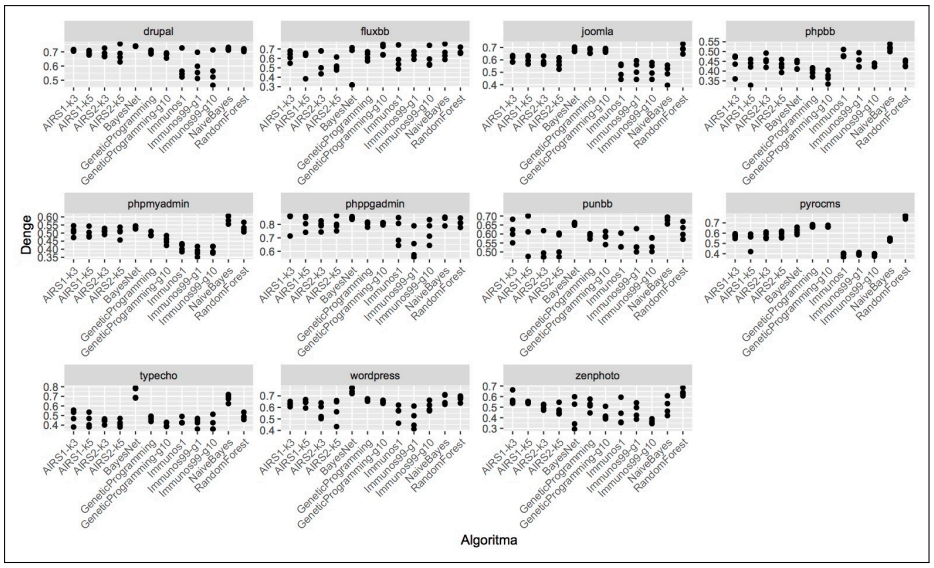
Şekil 8.5'de HTML ve sunucu metrikleri için algoritma bazında performans karşılaştırması kutu grafiği olarak gösterilmiştir. Buradan da görülebileceği üzere medyan değerlerine bakıldığı zaman Bayes Ağı ve Rastgele Orman algoritmaları diğerlerine göre daha iyi sonuç vermektedir.



Şekil 8.6 Metrik seti bazında performans karşılaştırması (HTML ve sunucu)

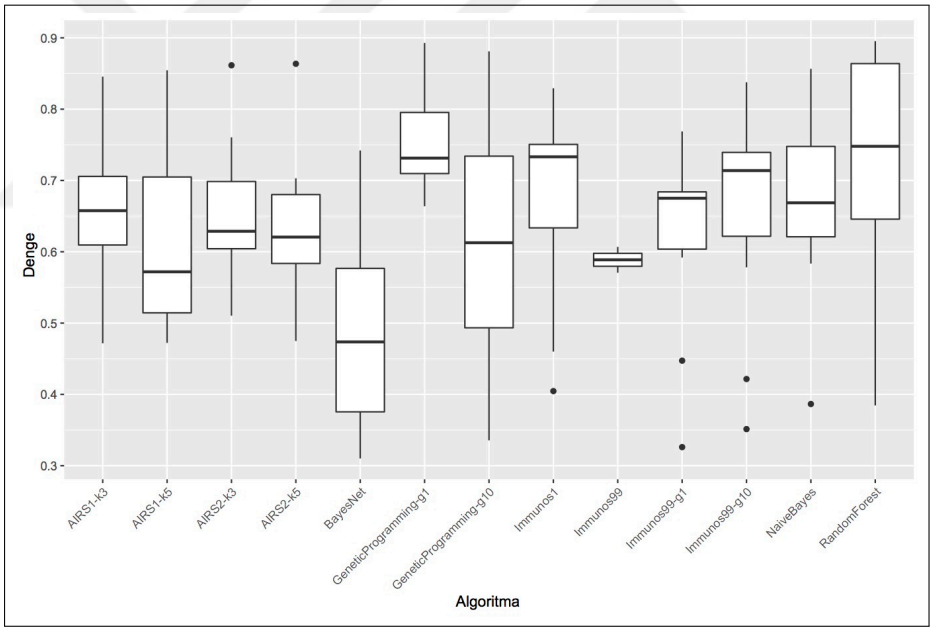
Şekil 8.6'da HTML ve sunucu metrikleri için algoritma ve metrik seti bazında performans karşılaştırması kutu grafiği olarak gösterilmiştir. Algoritmaların performansları kullanılan metrik setine göre değişmekle birlikte sıralamanın genel olarak aynı dağılımı izlediği görülebilir.

Şekil 8.7'de HTML ve sunucu metrikleri için algoritma ve proje bazında performans karşılaştırması kutu grafiği olarak gösterilmiştir. Burada performansların ve proje içindeki sıralamaların projeye göre değiştiği görülebilir. Örneğin PhpBB veri kümesi için Naive Bayes en iyi sonucu vermekteyken Joomla veri kümesi için en kötü sonucu veren algorit-



Şekil 8.7 Proje bazında performans karşılaştırması (HTML ve sunucu)

malardan biri olmaktadır. Optimizasyon yöntemleri de phpPgAdmin ve FluxBB için en iyi sonuçları vermekteyken PyroCMS ve Typecho projelerinde çok düşük kalmaktadırlar.

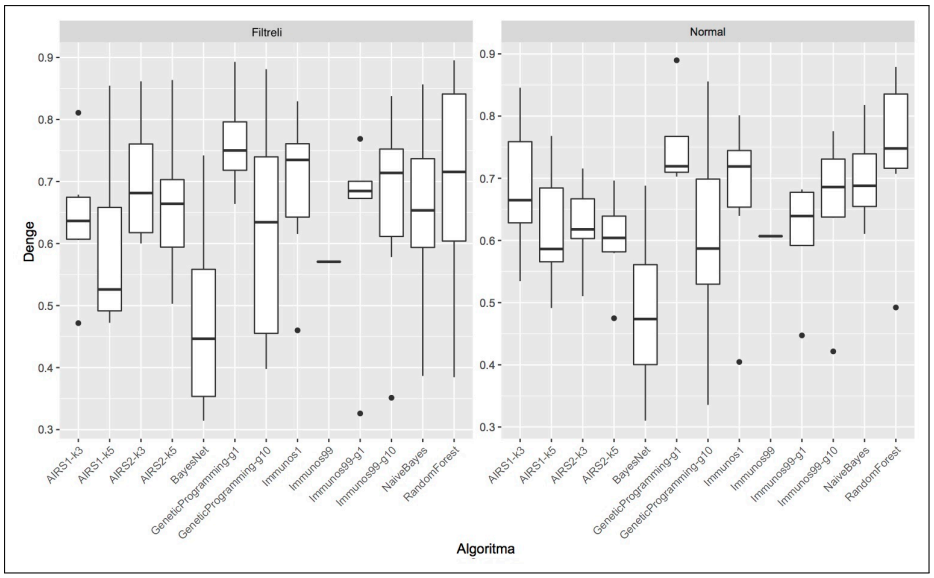


Şekil 8.8 Performans karşılaştırması (CSS metrikleri)

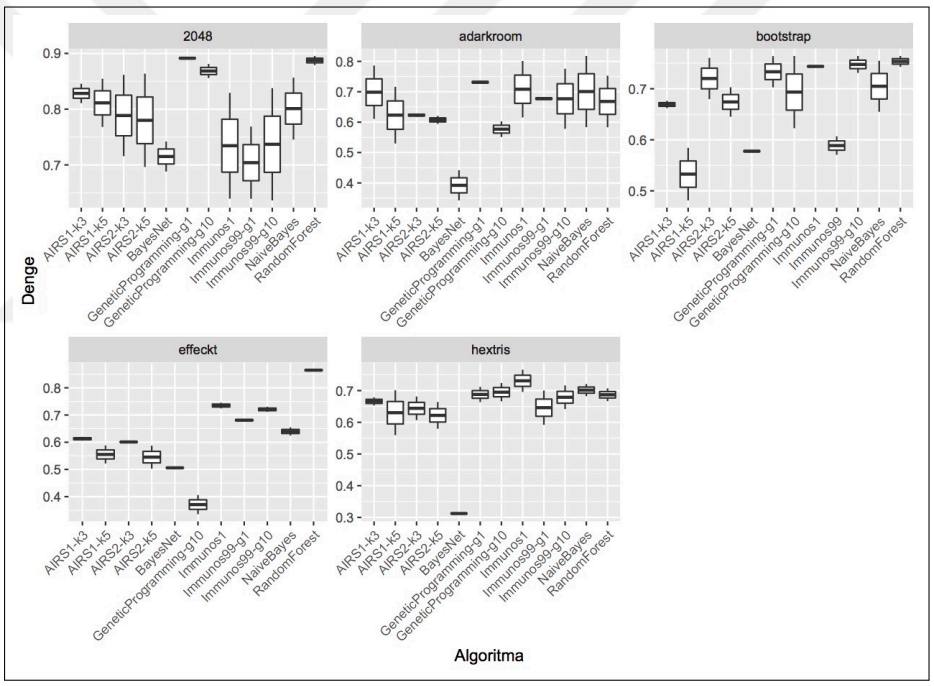
Şekil 8.8'de CSS metrikleri için algoritma bazında performans karşılaştırması kutu grafiği olarak gösterilmiştir. Buradan medyan değerlerine göre bakıldığı zaman Rastgele Orman, Genetik Programlama (g=2) ve Immunos-1 algoritmalarının başa baş sonuç verdiği görülmektedir. Ortalama değerde en iyi sonuç veren 2. algoritma olan Naive Bayes ise medyan değerine göre daha geride görülmektedir.

Şekil 8.9'de CSS metrikleri için metrik seti ve algoritma bazında performans karşılaştırması gösterilmiştir. Filtre uygulanmış ve uygulanmamış veri kümeleri için performans değerleri görülebilir. Arada farklılıklar olmakla birlikte dağılımlar benzer olmuştur.

Şekil 8.10'da CSS metrikleri için proje ve algoritma bazında performans karşılaştırması



Şekil 8.9 Metrik seti bazında performans karşılaştırması (CSS metrikleri)



Şekil 8.10 Proje bazında performans karşılaştırması (CSS metrikleri)

gösterilmiştir. HTML metriklerine benzer olarak, performans dağılımlarının veri kümeleri arasında farklılıklar gösterdiği görülebilir.

8.4 Değerlendirme ve Yorumlar

Çalışmanın bu bölümünde optimizasyon algoritmaları kullanılarak önerilen metrik kümelerin performansını artırmaya çalıştık. Bu amaçla deneylerde Genetik Programlama ve Yapay Bağışıklık Sistemi algoritma ailesi kullanılmıştır.

Sunucu tarafı ve HTML metrik kümesi için sonuçlarda iyileşme gözlenmemiştir. Gelenek-

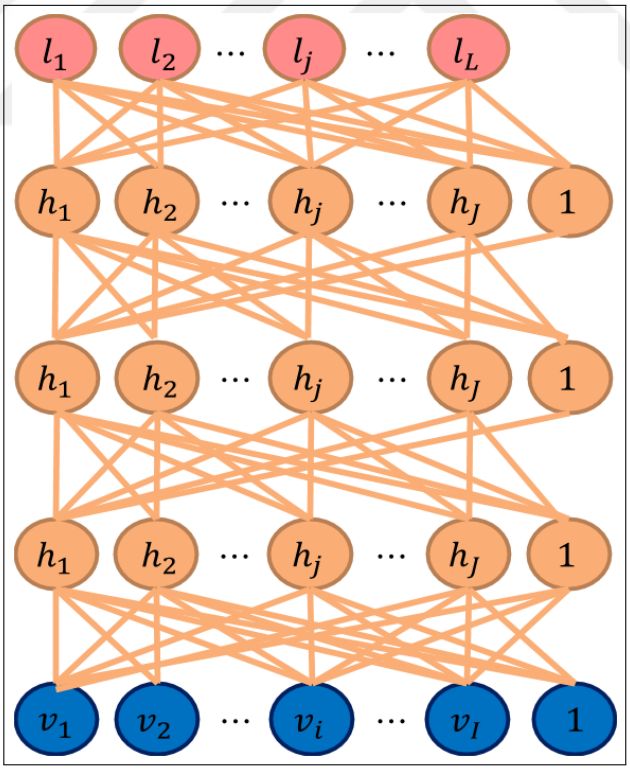
sel makine öğrenme algoritmalarının hem GP hem de YBS algoritmalarından daha iyi performans gösterdiği gözlenmiştir. CSS metrik seti için yine yeni algoritmalarla geliştirmeleri göremedik. Bu metrik tipinde de RO algoritması yine sonuçlarda daha yüksek başarı göstermektedir.

Farklı sınıflandırma algoritmalarının kusur tahmin verilerindeki performans farklılığı ile ilgili araştırmalara göre bu sonuç şaşırtıcı değildir [8]. Benzer incelemeler GP veya YBS aileleri gibi optimizasyon algoritmaları kullanılarak gerçekleştirilmemesine rağmen, sınıflandırma algoritmalarının çoğunda önemli bir fark olmadığını belirtmişlerdir. Tez çalışmasının bu kısmındaki sonuçlar ile bir kez daha görülmüştür ki hatalılık tahmini araştırmalarında odak algoritmaları iyileştirmek yerine veri kümelerini iyileştirmek olmalıdır.

BÖLÜM 9

DERİN ÖĞRENME

Teorik çalışmalar, yüksek seviyede soyutluk içeren karmaşık fonksiyonların öğrenilmesi için birden fazla non-lineer seviyede işlemden oluşan derin yapıların kullanılmasına ihtiyaç olduğunu göstermiştir [147]. Derin öğrenme, yapay sinir ağlarının özel bir formu olan bir makine öğrenme algoritması sınıfıdır [148]. YSA'ya ek olarak özellik çıkarma ve dönüştürme için bir çok nonlineer katmandan oluşan kademeli bir yapı kullanır. Her ardışık katman, önceki katmandaki çıktıyı girdi olarak kullanır. Algoritmalar supervised veya unsupervised olabilir ve uygulamalar desen analizi ve sınıflandırma içerir. Örnek bir derin öğrenme ağı Şekil 9.1 de görülebilir.



Şekil 9.1 Derin öğrenme ağı [148]

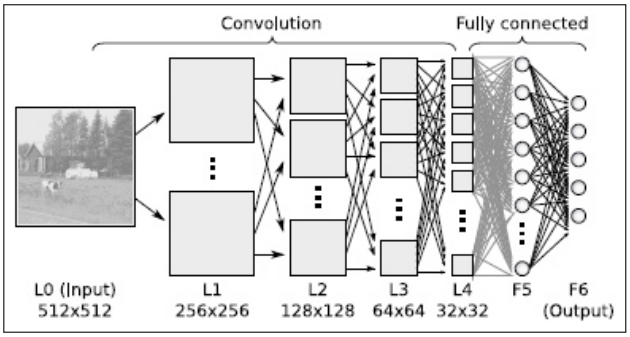
Verilerin özellikleri veya görünümlerinin birden fazla seviyesinin öğrenilmesine dayanmaktadır. Farklı soyutlama seviyelerine karşılık gelen birden çok temsil seviyesi öğrenilir. Üst düzey özelliklerin alt düzey özelliklerden türetilmesiyle hiyerarşik bir temsil oluşturulur [148]. Derin bir öğrenme algoritmasında kullanılan doğrusal olmayan işlem birimle-

rinin bir bileşiminin bileşimi çözülecek soruna bağlıdır. Derin öğrenmede kullanılan katmanlar yapay sinir ağı gizli katmanlarını ve karmaşık önerme eşitliklerini içerir. [147]

İlk genel çoklu seviye perseptron içeren, denetlenen ileri beslemeli derin öğrenme algoritması 1965 yılında Ivakhnenko and Lapa tarafından ortaya atılmıştır [149]. Derin öğrenme ilk olarak Londra Dünya Okullar Konseyi tarafından uygulanmıştır. [150]. Giriş değerlerinin sığ öğrenme algoritmalarından daha fazla katman kullanarak dönüştürmek için kullanılmıştır. Her katmandaki sinyal, parametreleri yinelemeli olarak eğitim yoluyla ayarlanan yapay bir nöron gibi bir işlem ünitesi tarafından dönüştürülür [147]. Günümüze kadar artarak gelen derin öğrenme çalışmalarının dönüm noktası 2009 yılında NVIDIA'nın GPU teknolojisi ile entegre çalıştırılması olmuştur [151]. Bu sayede haftalar alan hesaplamalar günler seviyesine inmeye başlamıştır.

9.1 Konvolüsyonel Sinir Ağları

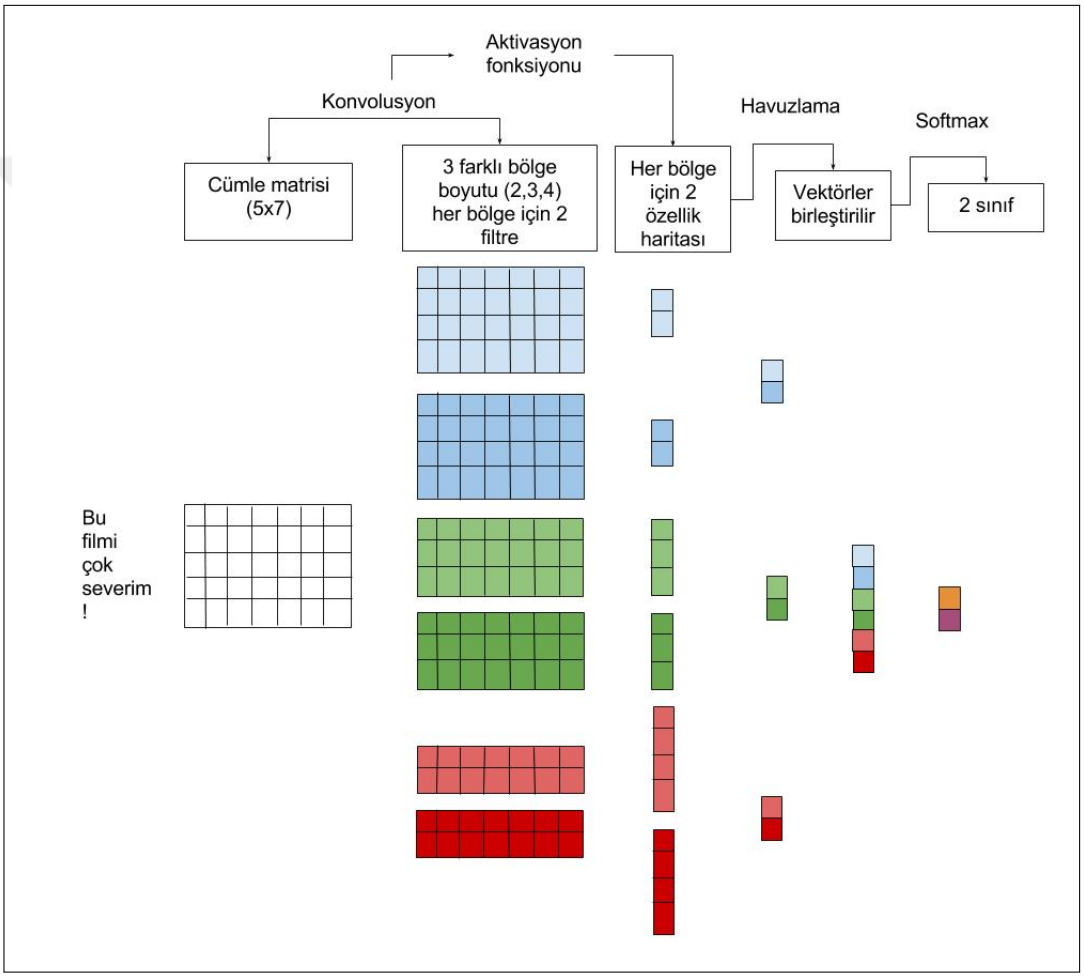
Konvolüsyonel Sinir Ağları (CNN), yapay sinir ağlarının gelişmiş bir modelidir. Genelde resim tanıma işlemleri için kullanılır. Nöronlar arasındaki sinyal akışına göre normal sinir ağlarından farklıdır. Tipik sinir ağları, sinyalleri ağa geri dönmesine izin vermeden, giriş-çıkış kanalı boyunca sinyalleri tek bir yönde iletir. Buna ileri besleme denir. İleri besleme ağları görüntü ve metin tanıma için başarılı bir şekilde kullanıldığı halde, tüm nöronların bağlanması gerekir ve bu da aşırı karmaşık bir ağ yapısına neden olur. Ağın, bilgisayar işlem hızlarının sınırlamaları ile birleşince büyük eğitim setleri üzerinde eğitilmesi gerektiğinde, karmaşıklığın maliyeti büyür ve eğitim süresi çok uzun olur. Dolayısıyla ileri besleme ağları, günümüzün yüksek çözünürlüklü, yüksek bant genişliği yoğun medya çağında ana makine öğreniminden yoksun duruma düşmüştü. 1986'da araştırmacılar Hubel ve Wiesel, alıcı alanının tüm görsel alanı kapsayacak şekilde birbiri üzerine katmanlı alt bölgelerden oluştuğunu keşfettikleri bir kedi görsel korteksini inceliyorlardı. Bu katmanlar, girdi görüntülerini işleyen, daha sonra sonraki katmanlara geçirilen filtreler gibi davranır. Bu, sinyal taşımak için daha basit ve etkili bir yol olduğu kanıtlandı. 1998 yılında Yann LeCun ve Yoshua Bengio, kedinin görsel korteksindeki nöronları yapay sinir ağının bir formu olarak yakalayarak ilk CNN'nin temelini oluşturdu. Görüntü tanıma için üretilmiş olsa da doğal dil işleme alanında da kullanılmaktadır. Şekil 9.2 ve 9.3 de görüntü işleme ve doğal dil işleme için kullanılan örnek sinir ağları yer almaktadır.



Şekil 9.2 Konvolüsyonel sinir ağı [152]

9.1.1 Konvolüsyon

Bir giriş sinyali alan ilk katmanlara konvolüsyon filtreleri denir. Konvolüsyon, ağın geçmişte öğrendiklerini referans alarak girdi sinyalini etiketlemeye çalıştığı bir süreçtir [148]. Giriş sinyali, daha önce görmüş olduğu önceki görüntülerden birine benziyorsa, referans sinyali giriş sinyaline karıştırılacak veya konvolüsyona tabi tutulacaktır. Elde edilen çıkış sinyali daha sonra bir sonraki katmana geçirilir. Bu, her konvolüsyon filtresinin bir ilgi özelliğini temsil ettiğini ve CNN algoritmasının hangi özelliklerin sonuçtaki referansı içerdiğini öğrendiği anlamına gelir. Çıkış sinyal gücü, özelliklerin nerede bulunduğu üzerine değil, yalnızca özelliklerin bulunup bulunmadığına bağlıdır. Dolayısıyla, tanınmak istenen nesne giriş verisinde farklı konumlarda olabilir ve CNN algoritması bunu halen tanıyabilir.



Şekil 9.3 Doğal dil işlemede kullanılan konvolüsyonel sinir ağı [153]

9.1.2 Aktivasyon

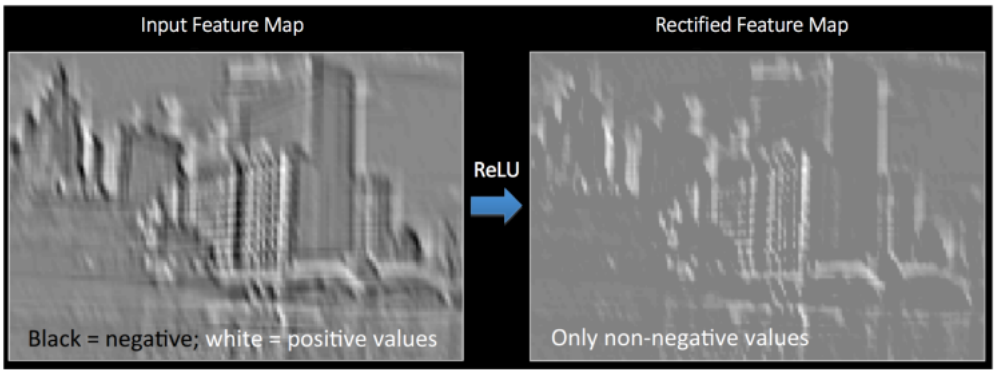
Aktivasyon katmanı sinyalin bir katmandan diğerine ne şekilde aktığını kontrol eder ve beyindeki nöronların nasıl tetiklendiğini taklit eder. Geçmiş referanslarla güçlü bir şekilde ilişkilendirilmiş çıkış sinyalleri daha fazla nöron aktive ederek, sinyallerin tanımlama için daha verimli bir şekilde yayılmasını sağlar.

CNN, sinyal yayılımını modellemek için çok çeşitli karmaşık aktivasyon fonksiyonlarıyla

uyumludur; en yaygın fonksiyon, daha hızlı eğitim hızı için tercih edilen Düzeltilmiş Doğrusal Birimdir (ReLU) [154].

9.1.2.1 ReLU

ReLU, element başına uygulanan bir işlemdir (örneğin görüntü işlemede piksel başına uygulanır) ve özellik matrisindeki tüm negatif piksel değerlerini sıfır ile değiştirir. ReLU'nun amacı, CNN'de nonlineerliği tanıtmaktır çünkü gerçek dünyadaki verilerin çoğu doğrusal olmayacaktır. Konvolüsyon lineer bir işlemdir (matris çarpımı ve toplaması), bu nedenle nonlineerliği katmak için ReLU gibi non-lineer fonksiyonlara ihtiyaç duyulmuştur.



Şekil 9.4 ReLU [155]

ReLU işlemi Şekil 9.4'dan anlaşılabilir. Konvolüsyon aşamasında elde edilen özellik haritalarından birine uygulanan ReLU işlemini göstermektedir. Buradaki çıkış özellik haritasına 'rektifiye' özellik haritası da denir.

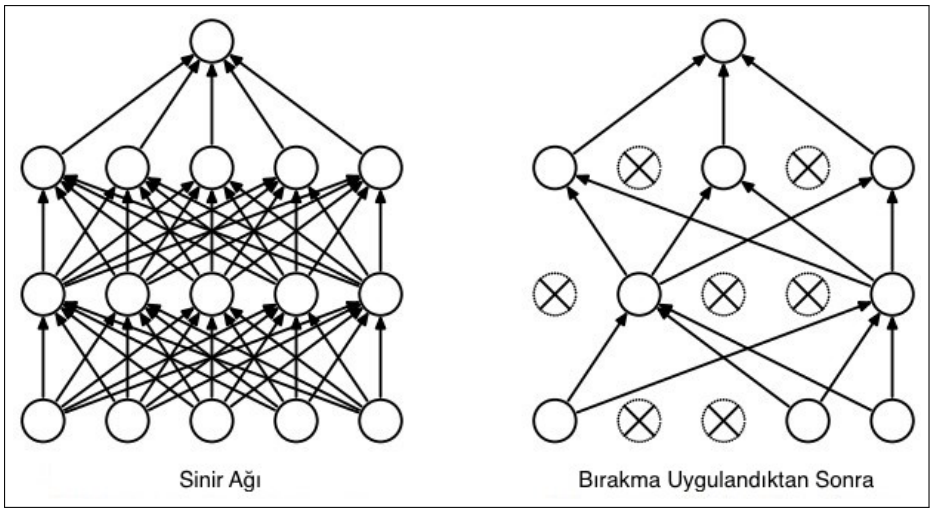
9.1.3 Havuzlama

Havuz katmanları genellikle konvolusyonel tabakalardan hemen sonra kullanılır. Bir havuz tabakası, konvolüsyon katmanından çıkan bilgiyi basitleştirerek yoğunlaştırılmış bir özellik haritası hazırlar. Havuzlama için yaygın olarak kullanılan yöntem, maksimum havuz yöntemidir. Maksimum havuzda, bir havuz birimi, belli bir boyuttaki giriş için maksimum aktivasyon üretir.

Konvolüsyon katmanının çıktısı birden fazla özellik haritası içerir. Maksimum havuzlama her özellik haritasına ayrı ayrı uygulanır. Bu yöntem bir bilgisayar ağına verinin herhangi bir yerde olup olmadığını sormaya benzer. Bulunduğu konumun bir önemi yoktur, asıl fayda, daha az biriken özelliklerin olması ve böylece daha sonraki katmanlarda gerekli olan parametre sayısının azaltılmasına yardımcı olmasıdır.

9.1.4 Tam Bağlanma

Tam bağlı katman, çıktı katmanında softmax aktivasyon işlevini kullanan bir çok katmanlı algılayıcı'dır (MLP). Ağdaki son katmanlar ardışık olarak tamamen bağlıdır, yani önceki katmandaki her nöron sonraki katmandaki her nörona bağlanır. Bu durum girdiden çıktıya kadar olan tüm olası yolların göz önüne alındığı yüksek düzey çıkarımı taklit eder.



Şekil 9.5 Bırakma [156]

Havuz katmanının çıkışı, girdi verisinin üst düzey özelliklerini temsil eder. Tam bağlı katmanın amacı, bu özelliklerin giriş verisini sınıflandırmak için kullanılmasıdır. Tam bağlı katmanın çıktı olasılıklarının toplamı 1'dir. Bu değer tam bağlı katmanın çıktı katmanındaki etkinleştirme işlevi olarak Softmax kullanılarak sağlanmaktadır. Softmax işlevi, rastgele reel sayılardan oluşan bir vektör alır ve 0-1 arası değerlerden oluşan, toplamı 1 olan bir değer vektörüne atar. Çıktıdaki değerler girişin her sınıf için olasılığına denk gelmektedir.

9.1.4.1 Bırakma

Aşırı uyum (Overfit), derin öğrenme'de görülen bir sorundur: model, kendini yalnızca eğitim setine uyarlamaya o kadar alışır ki kendini zorluklara adapte edip karar sınırlarını öğrenmeyi bırakır. Bu sorun için birçok çözüm önerilmekle birlikte, Bırakma yöntemi diğerlerinden daha öne çıkmıştır [156].

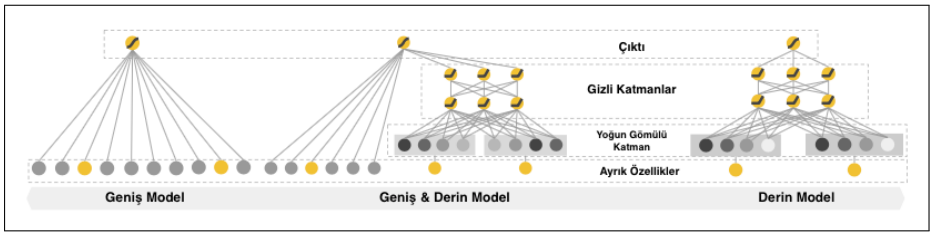
Bırakma'nın arkasındaki fikir, bir yapay sinir ağı toplamı oluşturmak ve tek bir ağ eğitmek yerine bütün toplulukların sonuçlarını ortalama olarak almaktır. Sinir ağları nöronların her birini belli bir olasılığa göre atmak veya korumak şeklinde oluşturulur. Bir nöron bırakıldığında, girdi veya ilişkili öğrenilen parametre ne olursa olsun, çıktısı sıfır olarak ayarlanır. Atılan nöronlar, geri yayılımın hem ileri hem de geri adımlarında eğitim aşamasına katkıda bulunmaz; bu nedenle bir nöron atıldığında eğitim fazı yeni bir ağ üzerinde yapılır.

9.2 Deney Tasarımı

Deneyde derin öğrenme için 2 farklı deney hazırlanmıştır. İlk deneyde derin öğrenmenin yazılım hataya yatkınlık tahminindeki performansını öğrenmek için doğal dil işlemeye benzer bir yaklaşım kullanılmıştır. Girdi projelerdeki dosyaların her biri önce tek satıra düşürülerek bir karakter vektörüne dönüştürülüp, hatalı/hatasız olma durumlarına göre 2 etikete bölünmüştür. Derin öğrenme uygulaması için Google tarafından üretilen Tensorflow [157] kütüphanesi kullanılmıştır. Tensorflow'a bu karakter kümeleri verilerek metinlerin sınıflandırma performansı ölçülmüştür. Yani derin öğrenmede kaynak kodundan çıkarılan özellikler değil, kodun tamamı kullanılmıştır. Görüntü işlemede kullanılan bu yön-

Çizelge 9.1 Parametreler

Parametre	Değer
Filtre boyutları	3,4,5
Filtre sayısı	128
Bırakma olasılığı	0.5
Yığın boyutu	64
Iterasyon sayısı	200



Şekil 9.6 Geniş & derin öğrenme [158]

temin yazılım hataya yatkınlık tahmininde vereceği sonuçlar gözlenmeye çalışılmıştır. Bu yaklaşımda tek satıra indirilen dosyalar bir matrise oturtularak işlenir. Böylece işleme görüntü tanıma işlemi gibi yaklaşılıp derin öğrenme uygulanabilir. Bu deneyde kullanılan parametreler Çizelge 9.1'deki gibidir:

9.3 Sonuçlar

Doğal dil işleme yaklaşımının kullanıldığı deney, işlem süresi çok uzun (12 saat - 1 gün) olduğu için 3 dataset üzerinde uygulanabilmiştir. Sonuçlar Çizelge 9.2'de görülebilir. Verilen sonuçlar 20 adımda bir üretilen ara değerlendirmelerde alınan en iyi değerlerden seçilmiştir. İşlem süresi çok uzun olduğu için değerlendirmeler çapraz geçerleme yerine veri %70 eğitim - %30 test şeklinde ayrılarak elde edilmiştir.

Sonuçlardan görüleceği üzere doğruluk yüksek çıksa da Denge ve AUC değerleri istenenin çok altındadır. Bu sonuçlardan hareketle kaynak kodu olduğu gibi verip derin öğrenmenin hatalı-hatasız şeklinde sınıflandırmasını beklemektense tezin önceki aşamalarında olduğu gibi özellik vektörleri vererek çalışmasının gözleneceği ikinci denemeye geçilmiştir. Bu aşamada Google tarafından ortaya atılmış Wide and Derin öğrenme modeli [158] kullanılmıştır. Bu model Şekil 9.6'de görülebileceği üzere temelde derin öğrenme yöntemi ile bazı özellikler derin ağa katılmadan, doğrudan bir çıkış birimine atanmaktadır. Bu deneyde kullanılan geniş model için diskrit özellikler, derin model için sürekli değerler kullanılmıştır. Çalışmanın bu kısmında kullanılan kütüphane, önceki bölümlerde kullandığımız TPR ve FPR ölçütlerini verememektedir. Bu nedenle karşılaştırmalar sadece alabildiğimiz doğruluk ve AUC değerleri ile yapılabilmiştir.

Çizelge 9.3'deki sonuçlar girdi olarak özellik vektörü kullanmanın bu problem için kaynak kodu doğrudan tanıtmaya göre çok daha iyi sonuçlar vermekte olduğunu göstermektedir. Üstelik bu deney çok daha kısa sürede tamamlandığı için daha fazla dataset üzerinde çalışma imkanı bulunmuştur.

Başarı bir öncekine göre daha iyi seviyede olsa da, sonuçlar aynı veriler üzerine kullanılan

Çizelge 9.2 NLP yaklaşımı sonuçları

Veri Kümesi	Doğruluk	TPR	FPR	Denge	AUC
Wordpress	0.71	0.52	0.20	0.63	0.65
PunBB	0.80	0.125	0	0.38	0.56
phpMyAdmin	0.74	0.2	0.028	0.43	0.585

Çizelge 9.3 Geniş & derin öğrenme sonuçları

	Derin öğrenme		Rastgele Orman
Veri Kümesi	**Doğruluk**	**AUC**	**AUC**
Wordpress	0.76	0.75	**0.84***
PunBB	0.73	0.74	**0.76***
phpMyAdmin	0.69	0.65	**0.76***
phpBB	0.80	0.70	**0.77***
Joomla	0.65	0.73	**0.85***
FluxBB	0.46	**0.67***	0.58

Rastgele Orman algoritmasının başarısına göre genel olarak daha düşük seviyede kalmaktadır.

9.4 Değerlendirme ve Yorumlar

Bu dönemde derin öğrenme metotlarının yazılımın hataya yatkınlığı tahmininde kullanılabilirliğini ve performansı incelenmeye çalışıldı. Bu amaçla 2 farklı yöntem izleyerek deneyler gerçekleştirildi.

Probleme metin sınıflandırma problemi şeklinde yaklaşıldığı durumda başarının geleneksel yöntemlere göre çok fazla düştüğü gözlemlendi. Özellik vektörü tabanlı derin öğrenme yöntemi kullanıldığı zaman başarının arttığı, ancak halen klasik sınıflandırma yöntemine göre düşük seviyede kaldığı gözlemlendi.

Kullanıldığı alanlarda çok iyi sonuçlar veren derin öğrenme yönteminin bu alanda neden yeteri başarıyı gösteremediği bir soru işareti olmakla birlikte, bu durum tezin çıkış noktası olan ”algoritmaların değil, veri setlerinin iyileştirimesi gerektiği” argümanını güçlendirir niteliktedir. Daha önceden incelenen genetik programlama ve yapay bağışıklık sistemlerinde de benzer sonuçla karşılaşıldığı hatırlanmalıdır. Öte yandan derin öğrenme, eğitim aşamasında geniş bir eğitim setine ihtiyaç duyan bir yöntemdir, performanstaki başarısızlık bu durumun bir sonucu da olabilir. Veri kümesinin daha da genişletilmesi durumunda öğrenme performansının incelenmesi diğer araştırmacılara bırakılmıştır.

BÖLÜM 10

SONUÇLAR

Tez çalışmasının hedefinde web alanındaki projelerde yazılım hatalılık tahmini performansını iyileştirebilmek bulunmaktadır. Bu amaçla tez kapsamında Kısım 1.3'de belirtilmiş olan 2 ana araştırma sorusu bulunmaktadır.

Birinci araştırma sorusunu cevaplayabilmek için web uygulamalarında hatalılığı tahmin etmeyi uygulamaların karakteristik özelliklerinden faydalanarak iyileştirebilecek ek metriklerin ne olabileceği incelenmiştir. Bu amaçla öncelikle web uygulama hatalarının taksonomisi incelenmiş, bulunan hata tipleri için otomatize araçlar kullanılarak çıkarılabilecek metrik kümeleri araştırılmıştır. Bulunan metrikler sunucu tabanlı ve istemci tabanlı olarak 2 ana kümede incelenmiştir. İstemci tabanlı metrikler de HTML ve CSS metrikleri olarak 2 alt kümede incelenmiştir. Öne sürülen metrikler veri kümelerindeki bilgi kazancı değerlerine göre incelenerek bazı metrikler kapsam dışı bırakılmıştır. 3 metrik kategorisi içinde bilgi kazançlarına göre en anlamlı bulunan ve tahmin performansını daha iyi etkilediği belirlenen en iyi beşer metrik Çizelge 10.1'de verilmiştir. Sonuçta elimizde bulunan, ve deneyler sonucu performansı iyileştirdiği gözlenen metrikler de Çizelge 10.2'de verilmiştir. Buna göre 10 sunucu metriği, 19 HTML metriği ve 14 CSS metriği anlamlı bulunup hatalılık tahmini performansını arttırabilecek metrik kümeleri olarak belirlenmiştir.

İkinci araştırma sonucu için, belirlenen metrik kümeleri kullanılarak, deneyler ve performans ölçümleri her metrik kümesi için ayrı ayrı yapılmıştır. Deneylerin uygulanmasında veri veya kullanılan yöntemden ötürü herhangi bir eğilim ve yanılma olmaması için veri kümeleri 3 farklı makine öğrenme algoritması kullanılarak eğitilmiş, test için 10 kümeli 10 katlı çapraz geçerleme kullanılmıştır. Performans ölçümü için de yazılım hatalılık tahmini literatüründe yaygın olarak kullanılan denge ölçütü kullanmıştır. Ayrıca, elde edilen ölçümlerdeki farklılıkların istatistiki olarak belirgin olup olmadığı kontrol edilmiş, bunun için de verideki dağılımdan etkilenmeyen Mann-Whitney U Testi kullanılmıştır. 11 adet farklı PHP tabanlı projeden çıkarılan metrik kümeleri kullanılarak yapılan deneylerde sunucu metriklerinin, şimdiye kadar var olan diğer metriklerle birlikte kullanılması durumunda, tahmin performansını ortalama olarak %2 oranında arttırdığı gözlenmiştir. Yine aynı projelerde HTML metriklerinin de kullanılması durumunda ortalama performansın %2 daha artış gösterdiği gözlenmiştir. Bu sonuçlarla alana özgü metrik setlerinin literatürde var olan statik kod metrik kümelerine ek olarak kullanımının yazılım modüllerinin hatalılığının tahmininde faydalı olduğu gözlenmektedir.

CSS metriklerinin kullanımı 5 farklı projeden alınmış veri kümeleri kullanılarak gözlenmiştir. Bu metrik seti ile yapılan çalışmalarda da aynı metodoloji kullanılmıştır. Ancak CSS metrikleri için, yazılım hatalılık tahmini literatüründe, sonuçlardaki iyileştirmenin karşı-

Çizelge 10.1 En iyi performans gösteren metrik kümesi

Sunucu Metrikleri	İstek Parametresi Kullanımı (Has_Param) Veritabanı sorgu kullanımı (DB_Query_Binary) Bağlam değişimi (Context_Switches) HTML kodu bulundurması (Has_HTML) İstek parametrelerine erişim sayısı (Param_Reads)
HTML metrikleri	Toplam içerik sayısı (total_text) Toplam etiket sayısı (total_tags) Tekil etiket sayısı (unique_tags) Tekil özellik sayısı (unique_attrs) Toplam derinlik (total_depth)
CSS metrikleri	Kuralun özgüllüğü (specificity) Etiketlere göre seçiciler (selectorsByTag) Sınıf ortalama özgüllüğü (specificityClassAvg) Sınıf toplam özgüllüğü (specificityClassTotal) Uzunluk (length)

laştırılabileceği bir metrik kümesi veya benzer çalışma bulunmamaktadır. Bu nedenle bu alandaki performans kazanımı ortalama %60 olarak belirlenen manuel kod inceleme performansı ile karşılaştırılmıştır. Ortalama performansın bu sınır değerden yüksek bulunması nedeniyle hatalılık tahmininde CSS metriklerinin kullanımı önerilmektedir.

Araştırma sorularının yanında farklı makine öğrenme algoritmaları kullanmanın hatalılık tahmini performansını arttırıp arttıramadığı da incelenmiştir. Bu amaçla ilk olarak sezgisel yöntemler olan Genetik Programlama ve Yapay Bağışıklık Sistemi algoritmaları kullanılarak Naive Bayes, Rastgele Orman ve Bayes Ağlarının performansları ile karşılaştırılmıştır. Yapılan deneyler sonucunda Naive Bayes ve Rastgele Orman algoritmalarının bazı veri kümeleri için geride kalmış olsalar da, genel olarak daha iyi performans gösterdiği gözlenmiştir.

Çizelge 10.2 Kabul edilen metrik kümesi

Sunucu Metrikleri	
Oturum Parametresi Kullanımı (Has_Session)	İstek Parametresi Kullanımı (Has_Param)
Oturumdan Okuma Sayısı (Session_Reads)	İstek Parametrelerine Erişim Sayısı (Param_Reads)
Veritabanı Sorgu Kullanımı (DB_Query_Binary)	Bağlam Değişimi (Context_Switches)
Çerez Kullanımı (Cookies)	Veritabanı Sorgu Sayısı (DB_Query)
İstek Başlığı Kullanımı (Header_Access)	
HTML Metrikleri	
Toplam / Tekil etiket sayısı (total_tags / unique_tags)	Toplam / Tekil / Ortalama özellik sayısı (total_attrs / unique_attrs / avg_attrs)
Maksimum / Toplam / Ortalama derinlik (max_depth / total_depth / avg_depth)	Toplam Yorum Sayısı (total_comments)
Toplam İçerik (total_text)	Form sayısı (forms)
Girdi sayısı (inputs)	Çerçeve sayısı (frames)
Script sayısı (scripts)	Dışarıdan yüklenen scriptler (external_scripts)
Sayfada tanımlanan scriptler (in_page_scripts)	Link sayısı (anchors)
Yüklenen stil etiketi sayısı (css_included)	Sayfada tanımlanan stil etiketi sayısı (inline_css)
CSS Metrikleri	
Uzunluk (length)	Birden fazla sınıflı seçici sayısı (multiClassesSelectors)
Seçici sayısı (selectors)	Nitelikli seçici sayısı (qualifiedSelectors)
Kuralın özgüllüğü (specificity)	Tanım sayısı (declarations)
Sınıf ortalama / toplam özgüllüğü (specificityClassAvg / specificityClassTotal)	Etiket ortalama / toplam özgüllüğü (specificityTagAvg / specificityTagTotal)
Özellik / sınıf / etiketlere göre seçiciler (selectorsByAttribute / selectorsByClass / selectorsByTag)	Kategorize edilmiş özgüllük (specificityCategory)

Bundan sonra son olarak derin öğrenme yönteminin uygulanması denenmiştir. Derin öğrenme resim tanıma gibi makine öğrenmesi alanlarında çok iyi sonuçlar verdiği gösterilmiş olsa da, yazılım hatalılık tahmininde kullanımı şimdiye kadar incelenmemiş bir konudur. Derin öğrenme yöntemi için öncelikle probleme doğal dil işleme problemi gibi yaklaşılıp bütün olarak verilen kodun hataya yatkın olup olmadığının tahmin edilmesi amaçlanmıştır. İşlemlerin çok uzun sürmesi nedeniyle bu yöntem öncelikle 3 veri kümesinde uygulanmış, performansın çok düşük kalması nedeniyle devam ettirilmemiştir. İkinci yaklaşım olarak Derin-Geniş Öğrenme yöntemi uygulanmıştır. Bu işlem 6 veri kümesi üzerinde denenmiş, performans değerleri kabul edilebilir seviyelerde olmasına rağmen başarı Rastgele Orman algoritmasına göre geride kalmaktadır. Bu nedenle bu yöntemin de başarıyı arttıramadığı kabul edilmiştir. Ancak derin öğrenme, eğitim aşamasında geniş bir eğitim setine ihtiyaç duyan bir yöntemdir, performanstaki başarısızlık bu durumun bir sonucu da olabilir. Veri kümesinin daha da genişletilmesi durumunda öğrenme performansının incelenmesi diğer araştırmacılara bırakılmıştır.

KAYNAKLAR

[1] Standish Group, (2013). CHAOS Manifesto 2013.

[2] Shull, F., Basili, V., Boehm, B., Brown, A. W., Costa, P., Lindvall, M., Port, D., Rus, I., Tesoriero, R., ve Zelkowitz, M., (2002). "What We Have Learned About Fighting Defects", 8th International Symposium on Software Metrics, METRICS '02, Washington, DC, USA.

[3] Pressman, R. S., (2005). Software Engineering: A Practitioner's Approach, 6. baskı, McGraw-Hill Higher Education.

[4] Basili, V., McGarry, F., Pajerski, R., ve Zelkowitz, M., (2002). "Lessons learned from 25 years of process improvement: the rise and fall of the NASA software engineering laboratory", ICSE 2002.

[5] Boehm, B. ve Basili, V. R., (2001). "Software Defect Reduction Top 10 List", Computer, 34(1):135–137.

[6] Song, Q., Shepperd, M., Cartwright, M., ve Mair, C., (2006). "Software Defect Association Mining and Defect Correction Effort Prediction", IEEE Trans. Softw. Eng., 32(2):69–82.

[7] Alpaydın, E., (2010). Introduction to Machine Learning, 2. baskı, The MIT Press.

[8] Lessmann, S., Baesens, B., Mues, C., ve Pietsch, S., (2008). "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings", IEEE Trans. Softw. Eng., 34(4):485–496.

[9] Menzies, T., Turhan, B., Bener, A., Gay, G., Cukic, B., ve Jiang, Y., (2008). "Implications of Ceiling Effects in Defect Predictors", 4th International Workshop on Predictor Models in Software Engineering, PROMISE '08, New York, NY, USA.

[10] Menzies, T., Stefano, J., Ammar, K., McGill, K., Callis, P., Davis, J., ve Chapman, R., (2003). "When can we test less?", Software Metrics Symposium.

[11] Jiang, Y., Cukic, B., ve Menzies, T., (2007). "Fault Prediction using Early Lifecycle Data", ISSRE '07.

[12] Jiang, Y., Cukic, B., ve Menzies, T., (2008). "Cost Curve Evaluation of Fault Prediction Models", ISSRE '08.

[13] Akiyama, F., (1971). "An Example of Software System Debugging", IFIP Congress.

[14] Basili, V. R. ve Perricone, B. T., (1984). "Software Errors and Complexity: An Empirical Investigation", Communications of ACM, 27(1):42–52.

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

[16] Internet World Stats, World Internet Users Statistics Usage and Population Stats, http://www.internetworldstats.com/stats.htm, 20 Ekim 2014.

[17] Sprenkle, S. E., (2007). Strategies for Automatically Exposing Faults in Web Applications. Doktora tezi, University of Delaware, Newark, DE, USA.

[18] CNet, California power outages suspended–for now, http://news.cnet.com/2100-1017-251167.html, 12 Nisan 2014.

[19] Soila Pertet and Priya Narasimhan, (2005). Causes of Failure in Web Applications, Parallel Data Laboratory, Carnegie Mellon University.

[20] Dholakia, U. ve Rego, L. L., (1998). "What makes commercial Web pages popular?: An empirical investigation of Web page effectiveness", European Journal of Marketing, 32(7):724–736.

[21] Offutt, J., (2002). "Quality Attributes of Web Software Applications", IEEE Softw., 19(2):25–32.

[22] Çatal, C. ve Diri, B., (2009). "Review: A Systematic Review of Software Fault Prediction Studies", Expert Syst. Appl., 36(4):7346–7354.

[23] Halstead, M. H., (1977). Elements of Software Science (Operating and Programming Systems Series), Elsevier Science Inc., New York, NY, USA.

[24] McCabe, T., (1976). "A Complexity Measure", Software Engineering, IEEE Transactions on, SE-2(4):308–320.

[25] Menzies, T., Distefano, J., S, A. O., ve (mike Chapman, R., (2004). "Assessing predictors of software defects", Workshop on Predictive Software Models.

[26] Menzies, T., Di Stefano, J., Chapman, M., ve McGill, K., (2002). "Metrics that matter", Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE.

[27] Nagappan, N. ve Ball, T., (2005). "Static Analysis Tools As Early Indicators of Pre-release Defect Density", 27th International Conference on Software Engineering, ICSE '05, New York, NY, USA.

[28] Nagappan, N. ve Ball, T., (2005). "Use of Relative Code Churn Measures to Predict System Defect Density", 27th International Conference on Software Engineering, ICSE '05, New York, NY, USA.

[29] Graves, T. L., Karr, A. F., Marron, J. S., ve Siy, H., (2000). "Predicting Fault Incidence Using Software Change History", IEEE Trans. Softw. Eng., 26(7):653–661.

[30] Munson, J. C. ve Elbaum, S. G., (1998). "Code Churn: A Measure for Estimating the Impact of Code Change", International Conference on Software Maintenance, ICSM '98, Washington, DC, USA.

[31] Çağlayan, B., Bener, A., ve Koch, S., (2009). "Merits of using repository metrics in defect prediction for open source projects", FLOSS '09.

[32] Meneely, A., Williams, L., Snipes, W., ve Osborne, J., (2008). "Predicting Failures with Developer Networks and Social Network Analysis", 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, SIGSOFT '08/FSE-16, New York, NY, USA.

[33] Pinzger, M., Nagappan, N., ve Murphy, B., (2008). "Can Developer-module Networks Predict Failures?", 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, SIGSOFT '08/FSE-16, New York, NY, USA.

[34] Zimmermann, T. ve Nagappan, N., (2008). "Predicting Defects Using Network Analysis on Dependency Graphs", 30th International Conference on Software Engineering, ICSE '08, New York, NY, USA.

[35] Wolf, T., Schroter, A., Damian, D., ve Nguyen, T., (2009). "Predicting Build Failures Using Social Network Analysis on Developer Communication", 31st International Conference on Software Engineering, ICSE '09, Washington, DC, USA.

[36] Biçer, S., Bener, A. B., ve Çağlayan, B., (2011). "Defect Prediction Using Social Network Analysis on Issue Repositories", 2011 International Conference on Software and Systems Process, ICSSP '11, New York, NY, USA.

[37] Misra, S. ve Cafer, F., (2012). "Estimating Quality of JavaScript.", Int. Arab J. Inf. Technol., 9(6):535–543.

[38] Andrews, A. A., Offutt, J., ve Alexander, R. T., (2005). "Testing web applications by modeling with fsms", Software and Systems Modeling, 4:326–345.

[39] Artail, H. ve Abi-Aad, M., (2009). "An Enhanced Web Page Change Detection Approach Based on Limiting Similarity Computations to Elements of Same Type", J. Intell. Inf. Syst., 32(1):1–21.

[40] Artzi, S., Kiezun, A., Dolby, J., Tip, F., Dig, D., Paradkar, A., ve Ernst, M. D., (2008). "Finding Bugs in Dynamic Web Applications", 2008 International Symposium on Software Testing and Analysis, ISSTA '08, New York, NY, USA.

[41] Benedikt, M., Freire, J., ve Godefroid, P., (2002). "VeriWeb: Automatically Testing Dynamic Web Sites", 11th International World Wide Web Conference WWW'2002.

[42] Hieatt, E. ve Mee, R., (2002). "Going faster: testing the Web application", Software, IEEE, 19(2):60–65.

[43] Lucca, G. D., Fasolino, A., ve Faralli, F., (2002). "Testing Web Applications", International Conference on Software Maintenance (ICSM'02), ICSM '02, Washington, DC, USA.

[44] Marchetto, A., Ricca, F., ve Tonella, P., (2007). "Empirical Validation of a Web Fault Taxonomy and its usage for Fault Seeding", WSE.

[45] Ricca, F. ve Tonella, P., (2005). "Anomaly detection in Web applications: a review of already conducted case studies", Software Maintenance and Reengineering.

[46] Sneed, H. M., (2004). "Testing a Web Application", Web Site Evolution, Sixth IEEE International Workshop, WSE '04, Washington, DC, USA.

[47] Guo, Y. ve Sampath, S., (2008). "Web Application Fault Classification - an Exploratory Study", Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '08, New York, NY, USA.

[48] Vijayaraghavan, G. V., (2003). A Taxonomy of E-Commerce Risks and Failures. Doktora tezi, Florida Institute of Technology.

[49] Huang, Y.-W., Yu, F., Hang, C., Tsai, C.-H., Lee, D.-T., ve Kuo, S.-Y., (2004). "Securing Web Application Code by Static Analysis and Runtime Protection", 13th International Conference on World Wide Web, WWW '04, New York, NY, USA.

[50] Livshits, V. B. ve Lam, M. S., (2005). "Finding Security Vulnerabilities in Java Applications with Static Analysis", 14th Conference on USENIX Security Symposium, SSYM'05, Berkeley, CA, USA.

[51] Wassermann, G. ve Su, Z., (2008). "Static Detection of Cross-site Scripting Vulnerabilities", 30th International Conference on Software Engineering, ICSE '08, New York, NY, USA.

[52] Tosun, A. ve Bener, A., (2009). "Reducing false alarms in software defect prediction by decision threshold optimization", ESEM 2009.

[53] Menzies, T., Butcher, A., Cok, D., Marcus, A., Layman, L., Shull, F., Turhan, B., ve Zimmermann, T., (2013). "Local versus Global Lessons for Defect Prediction and Effort Estimation", Software Engineering, IEEE Transactions on, 39(6):822–834.

[54] Peters, F., Menzies, T., ve Marcus, A., (2013). "Better cross company defect prediction", MSR 2013.

[55] Zheng, J., (2010). "Cost-sensitive Boosting Neural Networks for Software Defect Prediction", Expert Syst. Appl., 37(6):4537–4543.

[56] Tan, X., Peng, X., Pan, S., ve Zhao, W., (2011). "Assessing Software Quality by Program Clustering and Defect Prediction", WCRE 2011.

[57] Wang, H., Khoshgoftaar, T. M., ve Seliya, N., (2011). "How Many Software Metrics Should be Selected for Defect Prediction?", FLAIRS Conference.

[58] Weyuker, E., Ostrand, T., ve Bell, R., (2007). "Using Developer Information as a Factor for Fault Prediction", PROMISE'07.

[59] Kastro, Y. ve Bener, A. B., (2008). "A Defect Prediction Method for Software Versioning", Software Quality Control, 16(4):543–562.

[60] Peng, H., Li, B., Ma, Y., ve He, L., (2013). "Using Software Dependency to Bug Prediction", Mathematical Problems in Engineering, 2013:12.

[61] Gao, K. ve Khoshgoftaar, T. M., (2011). "Software Defect Prediction for High-Dimensional and Class-Imbalanced Data", SEKE.

[62] Thomas Zimmermann, Nachiappan Nagappan, A. Z., (2008). Predicting Bugs from History, chapter Predicting Bugs from History, 69–88, Springer.

[63] Jureczko, M. ve Spinellis, D. D., (2010). "Using Object-Oriented Design Metrics to Predict Software Defects", RELCOMEX.

[64] Tosun, A., Turhan, B., ve Bener, A., (2009). "Practical Considerations in Deploying AI for Defect Prediction: A Case Study Within the Turkish Telecommunication Industry", 5th International Conference on Predictor Models in Software Engineering, PROMISE '09, New York, NY, USA.

[65] Zhang, H., Nelson, A., ve Menzies, T., (2010). "On the Value of Learning from Defect Dense Components for Software Defect Prediction", 6th International Conference on Predictive Models in Software Engineering, PROMISE '10, New York, NY, USA.

[66] Paikari, E., Sun, B., Ruhe, G., ve Livani, E., (2011). "Customization Support for CBR-based Defect Prediction", 7th International Conference on Predictive Models in Software Engineering, Promise '11, New York, NY, USA.

[67] Ostrand, T. J., Weyuker, E. J., ve Bell, R. M., (2010). "Programmer-based Fault Prediction", 6th International Conference on Predictive Models in Software Engineering, PROMISE '10, New York, NY, USA.

[68] Shihab, E., Jiang, Z. M., Ibrahim, W. M., Adams, B., ve Hassan, A. E., (2010). "Understanding the Impact of Code and Process Metrics on Post-release Defects: A Case Study on the Eclipse Project", 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '10, New York, NY, USA.

[69] Mende, T., (2010). "Replication of Defect Prediction Studies: Problems, Pitfalls and Recommendations", 6th International Conference on Predictive Models in Software Engineering, PROMISE '10, New York, NY, USA.

[70] Tassé, J., (2013). "Using Code Change Types in an Analogy-based Classifier for Short-term Defect Prediction", 9th International Conference on Predictive Models in Software Engineering, PROMISE '13, New York, NY, USA.

[71] Tosun, A., Turhan, B., ve Bener, A., (2009). "Validation of Network Measures As Indicators of Defective Modules in Software Systems", 5th International Conference on Predictor Models in Software Engineering, PROMISE '09, New York, NY, USA.

[72] Herbold, S., (2013). "Training Data Selection for Cross-project Defect Prediction", 9th International Conference on Predictive Models in Software Engineering, PROMISE '13, New York, NY, USA.

[73] Rahman, F., Posnett, D., ve Devanbu, P., (2012). "Recalling the "imprecision" of cross-project defect prediction", In the 20th ACM SIGSOFT FSE.

[74] Çağlayan, B., Tosun, A., Miranskyy, A., Bener, A., ve Ruffolo, N., (2010). "Usage of Multiple Prediction Models Based on Defect Categories", 6th International Conference on Predictive Models in Software Engineering, PROMISE '10, New York, NY, USA.

[75] Li, M., Zhang, H., Wu, R., ve Zhou, Z.-H., (2012). "Sample-based Software Defect Prediction with Active and Semi-supervised Learning", Automated Software Engg., 19(2):201–230.

[76] Canfora, G., De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., ve Panichella, S., (2013). "Multi-objective Cross-Project Defect Prediction", ICST 2013.

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

[78] Turhan, B., Bener, A., ve Menzies, T., (2010). Regularities in learning defect predictors, Product-Focused Software Process Improvement, volume 6156 of Lecture Notes in Computer Science, 116–130. Springer Berlin Heidelberg.

[79] Çatal, C., Sevim, U., ve Diri, B., (2011). "Practical Development of an Eclipse-based Software Fault Prediction Tool Using Naive Bayes Algorithm", Expert Syst. Appl., 38(3):2347–2353.

[80] Kpodjedo, S., Ricca, F., Galinier, P., Guéhéneuc, Y.-G., ve Antoniol, G., (2011). "Design Evolution Metrics for Defect Prediction in Object Oriented Systems", Empirical Softw. Engg., 16(1):141–175.

[81] Gray, D., Bowes, D., Davey, N., Sun, Y., ve Christianson, B., (2010). "Software defect prediction using static code metrics underestimates defect-proneness", IJCNN 2010.

[82] Chang, C.-P., Chu, C.-P., ve Yeh, Y.-F., (2009). "Integrating in-process software defect prediction with association mining to discover defect pattern", Information & Software Technology, 51(2):375–384.

[83] Moser, R., Pedrycz, W., ve Succi, G., (2008). "A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction", 30th International Conference on Software Engineering, ICSE '08, New York, NY, USA.

[84] Wang, J. ve Zhang, H., (2012). "Predicting Defect Numbers Based on Defect State Transition Models", ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '12, New York, NY, USA.

[85] Umar, S. N., (2013). "Software Testing Defect Prediction Model-A Practical Approach", International Journal of Research in Engineering and Technology, 2(5):741–745.

[86] Sami, A. ve Fakhrahmad, S. M., (2010). "Design-level Metrics Estimation Based on Code Metrics", 2010 ACM Symposium on Applied Computing, SAC '10, New York, NY, USA.

[87] Shihab, E., Mockus, A., Kamei, Y., Adams, B., ve Hassan, A. E., (2011). "High-impact Defects: A Study of Breakage and Surprise Defects", 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11, New York, NY, USA.

[88] Moser, R., Pedrycz, W., ve Succi, G., (2008). "Analysis of the Reliability of a Subset of Change Metrics for Defect Prediction", Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '08, New York, NY, USA.

[89] Lu, H., Cukic, B., ve Culp, M., (2012). "Software Defect Prediction Using Semi-supervised Learning with Dimension Reduction", 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012, New York, NY, USA.

[90] Tosun, A., Turhan, B., ve Bener, A., (2008). "Ensemble of Software Defect Predictors: A Case Study", Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '08, New York, NY, USA.

[91] Ratzinger, J., Sigmund, T., ve Gall, H. C., (2008). "On the Relation of Refactorings and Software Defect Prediction", 2008 International Working Conference on Mining Software Repositories, MSR '08, New York, NY, USA.

[92] Nam, J., Pan, S. J., ve Kim, S., (2013). "Transfer Defect Learning", 2013 International Conference on Software Engineering, ICSE '13, Piscataway, NJ, USA.

[93] Bird, C., Nagappan, N., Murphy, B., Gall, H., ve Devanbu, P., (2011). "Don'T Touch My Code!: Examining the Effects of Ownership on Software Quality", 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11, New York, NY, USA.

[94] Rahman, F. ve Devanbu, P., (2013). “How, and Why, Process Metrics Are Better”, 2013 International Conference on Software Engineering, ICSE ’13, Piscataway, NJ, USA.

[95] Mısırlı, A. T., Çağlayan, B., Miranskyy, A. V., Bener, A., ve Ruffolo, N., (2011). “Different Strokes for Different Folks: A Case Study on Software Metrics for Different Defect Categories”, 2nd International Workshop on Emerging Trends in Software Metrics, WETSoM ’11, New York, NY, USA.

[96] Kim, S., Zhang, H., Wu, R., ve Gong, L., (2011). “Dealing with Noise in Defect Prediction”, 33rd International Conference on Software Engineering, ICSE ’11, New York, NY, USA.

[97] Wang, D., Wang, Q., Hong, Z., Chen, X., Zhang, L., ve Yang, Y., (2012). “Incorporating Qualitative and Quantitative Factors for Software Defect Prediction”, 2nd International Workshop on Evidential Assessment of Software Technologies, EAST ’12, New York, NY, USA.

[98] Jeet, K., Bhatia, N., ve Minhas, R. S., (2011). “A Bayesian Network Based Approach for Software Defects Prediction”, SIGSOFT Softw. Eng. Notes, 36(4):1–5.

[99] Zimmermann, T., Nagappan, N., Gall, H., Giger, E., ve Murphy, B., (2009). “Cross-project Defect Prediction: A Large Scale Experiment on Data vs. Domain vs. Process”, the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ESEC/FSE ’09, New York, NY, USA.

[100] Jureczko, M., (2011). “Significance of Different Software Metrics in Defect Prediction”, Software Engineering: An International Journal, 1(1):86–95.

[101] Wahyudin, D., Schatten, A., Winkler, D., Tjoa, A., ve Biffl, S., (2008). “Defect Prediction using Combined Product and Project Metrics - A Case Study from the Open Source ”Apache” MyFaces Project Family”, Software Engineering and Advanced Applications, 2008. SEAA ’08. 34th Euromicro Conference.

[102] Alhassan, S., Çağlayan, B., ve Bener, A. B., (2010). “Do More People Make the Code More Defect Prone?: Social Network Analysis in OSS Projects”, SEKE.

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

[104] Grbac, T. G., Mausa, G., ve Basic, B. D., (2013). “Stability of Software Defect Prediction in Relation to Levels of Data Imbalance”, SQAMIA.

[105] Koru, A. G. ve Liu, H., (2005). “An Investigation of the Effect of Module Size on Defect Prediction Using Static Measures”, SIGSOFT Softw. Eng. Notes, 30(4):1–5.

[106] Wang, S. ve Yao, X., (2013). “Using Class Imbalance Learning for Software Defect Prediction”, Reliability, IEEE Transactions on, 62(2):434–443.

[107] Tsakonas, A. ve Dounias, G., (2008). Predicting defects in software using grammar-guided genetic programming, Artificial Intelligence: Theories, Models and Applications, volume 5138 of Lecture Notes in Computer Science, 413–418. Springer Berlin Heidelberg.

[108] Giger, E., D'Ambros, M., Pinzger, M., ve Gall, H. C., (2012). "Method-level Bug Prediction", ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '12, New York, NY, USA.

[109] Çatal, C. ve Diri, B., (2013). "A Fault Detection Strategy for Software Projects", Technical Gazzette, 20(1):1–7.

[110] Çatal, C., Sevim, U., ve Diri, B., (2009). "Clustering and Metrics Thresholds Based Software Fault Prediction of Unlabeled Program Modules.", ITNG.

[111] Malhotra, R., (2012). "A Defect Prediction Model for Open Source Software", World Congress on Engineering.

[112] Jureczko, M. ve Madeyski, L., (2010). "Towards Identifying Software Project Clusters with Regard to Defect Prediction", 6th International Conference on Predictive Models in Software Engineering, PROMISE '10, New York, NY, USA.

[113] Sunita Devnani-chulani, (1997). Incorporating Bayesian Analysis to Improve the Accuracy of COCOMO II and Its Quality Model Extension, University of Southern California.

[114] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., ve Witten, I. H., (2009). "The WEKA Data Mining Software: An Update", SIGKDD Explor. Newsl., 11(1):10–18.

[115] Freund, Y. ve Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting, (1997).

[116] Breiman, L., (2001). "Random Forests", Mach. Learn., 45(1):5–32.

[117] Bayes, T., (1763). "An Essay towards solving a Problem in the Doctrine of Chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, M. A. and F. R. S.", Philosophical Transactions of the Royal Society, 53:370–418.

[118] John, G. ve Langley, P., (1995). "Estimating Continuous Distributions in Bayesian Classifiers", In Eleventh Conference on Uncertainty in Artificial Intelligence.

[119] Domingos, P. ve Pazzani, M., (1997). "On the Optimality of the Simple Bayesian Classifier Under Zero-One Loss", Mach. Learn., 29(2-3):103–130.

[120] Pearl, J., (1988). Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[121] Friedman, N., Geiger, D., ve Goldszmidt, M., (1997). "Bayesian Network Classifiers", Mach. Learn., 29(2-3):131–163.

[122] Hall, M., (1999). Correlation-based Feature Selection for Machine Learning. Doktora tezi, University of Waikato.

[123] Mann, H. B. ve Whitney, D. R., (1947). "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other", The Annals of Mathematical Statistics, 18(1):50–60.

[124] Sourceforge, GitStats - git history statistics generator, http://gitstats.sourceforge.net, 5 Temmuz 2015.

[125] SciTools, Understand - Source Code Analysis & Metrics, http://scitools.com, 3 Mayıs 2014.

[126] World Wide Web, HTML & CSS, https://www.w3.org/standards/webdesign/htmlcss, 15 Ekim 2015.

[127] World Wide Web, World Wide Web Consortium (W3C), https://www.w3.org, 15 Ekim 2015.

[128] Raggett, D., Lam, J., Alexander, I., ve Kmiec, M., (1998). Raggett on HTML 4, 2. baskı, Addison-Wesley Professional.

[129] Kullback, S. ve Leibler, R. A., (1951). "On Information and Sufficiency", Ann. Math. Statist., 22(1):79–86.

[130] World Wide Web, Selectors Level 3, http://www.w3.org/TR/selectors/#specificity, 15 Mayıs 2015.

[131] IETF, RFC 2318: The text/css Media Type, https://tools.ietf.org/html/rfc2318, 15 Ekim 2015.

[132] Github, CSS selectors complexity and performance analyzer, https://github.com/macbre/analyze-css, 15 Ekim 2015.

[133] Python, cssselect 0.9.1 : Python Package Index, https://pypi.python.org/pypi/cssselect, 15 Ekim 2015.

[134] Drummond, C. ve Holte, R. C., (2006). "Cost curves: An improved method for visualizing classifier performance.", Mach. Learn., 65(1):95–130.

[135] Hand, D. J., (2009). "Measuring classifier performance: a coherent alternative to the area under the ROC curve", Machine Learning, 77(1):103–123.

[136] Halligan, S., Altman, D. G., ve Mallett, S., (2015). "Disadvantages of using the area under the receiver operating characteristic curve to assess imaging tests: A discussion and proposal for an alternative approach", European Radiology, 25(4):932–939.

[137] Arisholm, E. ve Briand, L. C., (2006). "Predicting fault-prone components in a java legacy system", ISESE '06, New York, NY, USA.

[138] Holland, J. H., (1992). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence, MIT Press, Cambridge, MA, USA.

[139] Farmer, J., Packard, N. H., ve Perelson, A. S., (1986). "Fifth Annual International Conference The immune system, adaptation, and machine learning", Physica D: Nonlinear Phenomena, 22(1):187 – 204.

[140] Carter, J. H., (2000). "Research Paper: The Immune System as a Model for Pattern Recognition and Classification", JAMIA, 7(1):28–41.

[141] Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., ve Walter, P., (2002). Molecular Biology of the Cell, 4. baskı, Garland Science.

[142] Cooper, M. D., (2015). "The early history of B cells", Nat Rev Immunol, 15(3):191–197.

[143] Brownlee, Jason, (2005). Immunos-81: The Misunderstood Artificial Immune System, Technical Report 3-01, Swinburne University.

[144] Çelik, U., Yurtay, N., Koç, E. R., Tepe, N., Güllüoglu, H., ve Ertas, M., (2015). "Diagnostic Accuracy Comparison of Artificial Immune Algorithms for Primary Headaches.", Comp. Math. Methods in Medicine, 2015:465192:1–465192:8.

[145] Watkins, A. ve Boggess, L., (2002). "A new classifier based on resource limited artificial immune systems", Evolutionary Computation, 2002. CEC '02. 2002 Congress on.

[146] Watkins, A., Timmis, J., ve Boggess, L., (2004). "Artificial Immune Recognition System (AIRS): An Immune-Inspired Supervised Learning Algorithm", Genetic Programming and Evolvable Machines, 5(3):291–317.

[147] Bengio, Y., (2009). "Learning Deep Architectures for AI", Found. Trends Mach. Learn., 2(1):1–127.

[148] Deng, L. ve Yu, D., (2014). "Deep Learning: Methods and Applications", Foundations and Trends in Signal Processing, 7(3–4):197–387.

[149] Ivakhnenko, A., Lapa, V., ve ENGINEERING., P. U. L. I. S. O. E., (1965). Cybernetic Predicting Devices, JPRS 37, 803, Purdue University School of Electrical Engineering.

[150] Schmidhuber, J., (2014). "Deep Learning in Neural Networks: An Overview", CoRR, abs/1404.7828.

[151] NVIDIA Blog, AI Drives the Rise of Accelerated Computing in Data Centers, https://blogs.nvidia.com/blog/2017/04/10/ai-drives-rise-accelerated-computing-datacenter/, 17 Haziran 2017.

[152] Bonn University, University of Bonn, Computer Science VI, Autonomous Intelligent Systems, http://www.ais.uni-bonn.de/deep_learning/, 16 Haziran 2017.

[153] Zhang, X., Zhao, J. J., ve LeCun, Y., (2015). "Character-level Convolutional Networks for Text Classification", CoRR, abs/1509.01626.

[154] Dahl, G. E., Sainath, T. N., ve Hinton, G. E., (2013). "Improving deep neural networks for LVCSR using rectified linear units and dropout", 2013 IEEE International Conference on Acoustics, Speech and Signal Processing.

[155] Rob Fergus, (2015). Neural Networks, http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf, 17 Haziran 2017.

[156] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., ve Salakhutdinov, R., (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Journal of Machine Learning Research, 15:1929–1958.

[157] Google, Tensorflow, https://www.tensorflow.org/, 17 Haziran 2017.

[158] Cheng, H., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., ve Shah, H., (2016). "Wide & Deep Learning for Recommender Systems", CoRR.

EK A

LİTERATÜRDEKİ YAZILIM METRİKLERİ

Çizelge A.1 Statik kod metrikleri

McCabe	Kod satır sayısı Döngüsel karmaşıklık Tasarım karmaşıklığı Zorunlu karmaşıklık
Halstead	Toplam işlem & işlenen Hacim Program uzunluğu Zorluluk Zeka Efor Zaman tahmini Satır sayısı Yorum satır sayısı Boş satır sayısı
Tekil işlem & işlenen	
Toplam işlem & işlenen	
Dallanma sayısı	

Çizelge A.2 Kod değişim metrikleri

Değişim sayısı
Geliştirici sayısı
Eklenen satır sayısı
Silinen satır sayısı
Son sürümdeki değişimler

Çizelge A.2 – Kod değişim metrikleri (devamı)

Son sürümdeki geliştiriciler
Son sürümde eklenen/silinen satır sayısı
Dosyalarda üst geliştiricilerin yüzdesi

Çizelge A.3 C-K metrikleri

Bağlama faktörü
Methodlarda yapışkanlık azlığı
Döngüsel karmaşıklık
Özellik gizleme faktörü
Method gizleme faktörü
Kalıtım derinliği
Çocuk sayısı
Sınıf başına ağırlıklanmış methodlar
Sınıf sayısı
Kod satır sayısı

Çizelge A.4 Sosyal ağ metrikleri

Arasındalık merkeziyeti
Yakınlık merkeziyeti
Barycenter merkeziyeti
Derece merkeziyeti
Grup derece merkezlik indeksi
Yoğunluk
Çağ
Kümeleme katsayısı
Köprü oranı
Karakteristik yol uzunluğu

Çizelge A.5 Diğer metrikler

Fonksiyon noktası
Aynı sayfada çalışan geliştirici sayısı
Hata düzeltme sayısı
Bildirilmiş hata sayısı

Çizelge A.5 – Diğer metrikler (devamı)

Eklenen özellik sayısı
İyileştirme sayısı
Sınıf yapısında değişiklik sayısı
Takım büyüklüğü
Test senaryosu büyüklüğü
Proje eforu

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Mehmet Serdar BİÇER

Doğum Tarihi ve Yeri : 26.08.1986 - İstanbul

Yabancı Dili : İngilizce

E-posta : serdar.bicer@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/üniversite	Mezuniyet Yılı
Y. Lisans	Bilgisayar Müh.	Boğaziçi Üniversitesi	2010
Lisans	Bilgisayar Müh.	Yıldız Teknik Üniversitesi	2008
Lise	Fen Bilimleri	Adnan Menderes Anadolu Lisesi	2004

İŞ TECRÜBESİ

Yıl	Firma/Kurum	Görevi
2013 - devam ediyor	Valven	CTO
2011 - 2013	Invio	Yazılım Mühendisi
2008 - 2011	Gerger Consulting	Yazılım Mühendisi

YAYINLARI

Makale

1. Biçer, M. S., Diri, B., (2016). ”Defect prediction for Cascading Style Sheets”, Applied Soft Computing, 49:1078-1084.

Bildiri

1. Biçer, M. S., Diri, B., (2015). ”Predicting Defect Prone Modules in Web Applications”, ICIST 2015, 15-16 Kasım 2015, Druskininkai, Lithuania.

2. Biçer, S., Diri, B., (2014). ”Yazılım Hata Tahmininin Web Uygulamalarında Kullanılabilirliği”, UYMS 2014, 8-10 Eylül 2014, Güzelyurt, KKTC.

3. Biçer, S., Bener, A. B., Çağlayan, B., (2011). ”Defect prediction using social network analysis on issue repositories”, ICSSP 2011, 21-22 Mayıs 2011, Honolulu, HI, USA.

ÖDÜLLERİ

1. En İyi Bildiri Ödülü (2014). Biçer, S., Diri, B., (2014). ”Yazılım Hata Tahmininin Web Uygulamalarında Kullanılabilirliği”, UYMS 2014, 8-10 Eylül 2014, Güzelyurt, KKTC.