SÜREKLİ ENTEGRASYON DESTEKLİ HATA ÖNGÖRÜ SİSTEMİ

Yüksek Lisans Tezi

İbrahim ESEN

Eskişehir 2019

SÜREKLİ ENTEGRASYON DESTEKLİ HATA ÖNGÖRÜ SİSTEMİ

İbrahim ESEN

YÜKSEK LİSANS TEZİ
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Doç. Dr. Özgür YILMAZEL

Eskişehir
Eskişehir Teknik Üniversitesi
Lisansüstü Eğitim Enstitüsü
Temmuz, 2019

JÜRİ VE ENSTİTÜ ONAYI

İbrahim ESEN'in "Sürekli Entegrasyon Destekli Hata Öngörü Sistemi" başlıklı tezi 08/07/2019 tarihinde aşağıdaki jüri tarafından değerlendirilerek "Eskişehir Teknik Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliği"nin ilgili maddeleri uyarınca Bilgisayar Mühendisliği Anabilim dalında Yüksek Lisans tezi olarak kabul edilmiştir.

Jüri Üyeleri	**Unvanı Adı Soyadı**	**İmza**
Üye (Tez Danışmanı)	: Doç. Dr. Özgür YILMAZEL	……
Üye	: Dr. Öğr. Üyesi Alper BİLGE	……
Üye	: Dr. Öğr. Üyesi Muammer AKÇAY	……

Prof. Dr. Murat TANIŞLI
Lisansüstü Eğitim Enstitüsü Müdürü

ÖZET

SÜREKLİ ENTEGRASYON DESTEKLİ HATA ÖNGÖRÜ SİSTEMİ

İbrahim ESEN

Bilgisayar Mühendisliği Anabilim Dalı
Eskişehir Teknik Üniversitesi, Lisansüstü Eğitim Enstitüsü, Temmuz, 2019

Danışman: Doç. Dr. Özgür YILMAZEL

Bu araştırmada yazılım geliştirme sürecinde sürekli entegrasyon sistemi verileri kullanılarak oluşabilecek hataların önceden tahmini üzerine çalışılmıştır. Yazılım geliştirme süreçlerinde kod değişikliklerinden, hata raporlarına, tasarım dokümanlarından, sürüm bilgilerine kadar uzanan bir dizi veri oluşur. Bu veriler kullanılarak yazılımdaki hata oluşabilecek noktalar önceden tahmin edilebilir.

Hata öngörü sistemleri için kullanılabilecek çok çeşitli veri setleri ve yöntemler bulunmaktadır. Bunların birçoğu kod değişiklikleri, statik kod analizi değerleri, geçmiş hataların analizi gibi verileri kullanır. Bu çalışmada ise kodun sürekli entegre edildiği ve derlendiği sistemin ürettiği veriler kullanılmıştır.

Çalışma için üç farklı açık kaynaklı yazılım projesi kullanılmıştır. Bu projelerin sürekli entegrasyon verileri, hata sistemlerindeki verileri ve sürüm verileri kullanılmıştır. Projeye uygun bir sürüm süresi belirlenerek gelecek sürümde çıkabilecek hata sayısı tahmini yapılmıştır. Bu sayede yazılım geliştirme sürecinde oluşan bu tahminler yazılım ekibine yol gösterici olur.

Anahtar Sözcükler: Hata öngörüsü, Yazılım geliştirme depoları, Veri madenciliği, Sürekli entegrasyon, Yazılım geliştirme süreci, Yazılım derleme.

ABSTRACT

BUG PREDICTION WITH CONTINUOUS INTEGRATION DATA

İbrahim ESEN

Department of Computer Engineering
Eskişehir Technical University, Institute of Graduate Programs, July, 2019

Supervisor: Assoc. Prof. Dr. Özgür YILMAZEL

In this thesis, software bug prediction using continuous integration system data is studied. Software development process produces various interesting data ranging from code changes, bug reports, design documents and release notes. By using this data software defects can be predicted before they occur.

There are a variety of data sets and methods that can be used in bug prediction systems. Most of the work in these systems use code changes, static code analysis results and analysis of past defects. In this work, data produced from continuous integration system, where code is integrated, compiled and tested, is used to predict future bugs.

Three open source project is used for his work. Continuous integration data, bug reports and release data of these projects' are used for bug prediction. An appropriate release period is determined for each project and the number of bugs that will occur in the next release is estimated. The results guide the software team for upcoming errors that will occur in the next release of the software.

Keywords: Bug prediction, Mining software repositories, Continuous integration, Software development, Software build

TEŞEKKÜR

Tez çalışmamın her aşamasında destek olan ve tez metninin hazırlanmasında yol gösteren danışmanım Doç. Dr. Özgür YILMAZEL'e teşekkürlerimi sunarım.

Desteğini hiçbir zaman esirgemeyen sevgili eşim ve varlığıyla güç veren oğluma teşekkür etmek isterim.

İbrahim ESEN

Temmuz, 2019

ETİK İLKE VE KURALLARA UYGUNLUK BEYANNAMESİ

Bu tezin bana ait, özgün bir çalışma olduğunu; çalışmamın hazırlık, veri toplama, analiz ve bilgilerin sunumu olmak üzere tüm aşamalarında bilimsel etik ilke ve kurallara uygun davrandığımı; bu çalışma kapsamında elde edilen tüm veri ve bilgiler için kaynak gösterdiğimi ve bu kaynaklara kaynakçada yer verdiğimi; bu çalışmanın Eskişehir Teknik Üniversitesi tarafından kullanılan "bilimsel intihal tespit programı"yla tarandığını ve hiçbir şekilde "intihal içermediğini" beyan ederim. Herhangi bir zamanda, çalışmamla ilgili yaptığım bu beyana aykırı bir durumun saptanması durumunda, ortaya çıkacak tüm ahlaki ve hukuki sonuçları kabul ettiğimi bildiririm.

.....

(İmza)

İbrahim ESEN

İÇİNDEKİLER

Sayfa

Sayfa

TABLOLAR DİZİNİ

Sayfa

ŞEKİLLER DİZİNİ

Sayfa

GÖRSELLER DİZİNİ

Sayfa

KISALTMALAR DİZİNİ

JSON : Javascript Object Notation

REST : Representational State Transfer

1. GİRİŞ

Yazılım geliştirme süreci, bir ekip veya organizasyon tarafından yazılım projeleri üzerinde uygulanan geliştirme yöntemi olarak tanımlanabilir. Bu sürecin amacı yazılımı üretmek ve yaşamasını sağlamaktır. Sürecin içerisinde amaca ulaşmak için gereken adımlar detaylı bir şekilde tanımlanır. Projelere ve yazılımı geliştiren ekiplere göre bu süreç detayları çeşitli farklılıklar gösterebilir. Ancak genellikle kullanılan temel adımlar planlama, gereksinim tanımlama, tasarlama, geliştirme, test etme, canlı üretime alma ve bakım yapma olarak listelenebilir. Bu süreç boyunca yazılımı geliştiren ekipler çeşitli tiplerde iş parçaları ile uğraşabilir. Bu işleri kabaca sınıflandıracak olursak; hata ayıklama ve düzeltme işleri, yeni özellik ile ilgili işler ile teknik ve altyapı işlemleri olarak sınıflandırılabilir. Burada en az olması istenen ve en çok zaman kaybettiren işler hatalardır. Ayrıca hatalar canlı ortamda çalışan yazılımda yüksek maliyete ve memnuniyetsizliğe sebep olan unsurların en önemlilerinden birisidir. Bu sebeple yazılım projelerinde hataların en az sayıda olması istenir.

Yazılım hatası, yazılımın beklenmedik veya hatalı bir sonuç üretmesi, geliştirme esnasındaki tasarıma göre istenmeyen bir biçimde davranması sonucu oluşan bir kusurdur [1]. Birçok hata yazılım kaynak kodundan dolayı oluşur, ancak yazılımın mimari tasarımı, bileşenleri çalıştığı işletim sistemi veya ortamdan da kaynaklı hatalar olabilmektedir.

Yazılım depoları üzerinde veri madenciliği alanı (Mining Software Repositories), yazılım geliştirme sürecinde birçok veri kaynağından gelen verileri analiz eden bir alandır [2]. Bu analizler sonucunda yazılım kalitesi, yazılımın gelişim süreci hakkında değerli sonuçlara ulaşılabilir. Bu sonuçlar yazılım hatalarını azaltmak için kullanılabilir. En sık kullanılan veri kaynağı sürüm kontrol sistemleridir [3]. Sürüm kontrol sistemlerinde yazılım değişiklikleri ve yazılım hataları ile ilgili değerli bilgiler bulunabilmektedir. Yapılan kod değişikliklerinin hangi dosyalarda olduğu, değişiklik ile ilgili yazılım geliştiricinin mesajı hata öngörüsü için önemlidir. Ayrıca hata düzelten kod değişiklikleri kritik yerlerin belirlenmesinde önemli bir bilgidir.

Bir yazılım projesinde hataların en az sayıya indirgenmesi, birçok yazılım ekibi ve organizasyon için önemli bir hedeftir. Yazılım geliştirme sürecinde kullanılan birçok araç, teknolojik altyapı ve süreç adımları genellikle bu hedefe göre tasarlanır. Bu geliştirme sürecindeki verileri kullanan hata öngörü sistemleri de hata sayısını azaltmak ve projenin gidişatını hakkında fikir vermek için kullanılan araçlardır.

Günümüzde veriye dayalı analiz ve karar destek sistemleri ile yapay zeka ve öğrenme temelli sistemler çeşitli alanlarda kritik çözümlerde yaygın bir biçimde kullanılmaktadır. Yazılım geliştirme süreçlerinin bütün adımlarında birçok veri ortaya çıkmaktadır. Bu verilerin süreç içinde faydalı bir şekilde analizinin yapılması yazılımdaki problemler hakkında fikir verebilir. Böylelikle hata oranları daha düşük seviyelerde olabilmektedir.

Bu çalışmada yazılım geliştirme sürecinde sık kullanılan bir araç olan sürekli entegrasyon ve derleme sistemlerinin oluşturduğu veriler kullanılarak hata öngörüsü üretmek amaçlanmıştır. Sürekli entegrasyon sistemleri farklı amaç ve yöntemler için kullanılabilir. Buradaki çalışmada sürekli entegrasyon sisteminin, yazılımdaki her bir değişiklik sonucunda veri ürettiği kullanım yöntemi temel alınarak çalışma yapılmıştır. Çalışma için seçilen üç açık kaynak kodlu yazılım projesinin iş ve sürüm takip sistemi verileri ile sürekli entegrasyon ve derleme sistemlerinin verileri toplanmıştır. Bu veriler bir ön işlemden geçirilerek öngörü yapılacak değer için bir altyapı modeli oluşturulmuştur. Daha sonra bu model kullanılarak bir sürekli entegrasyon derleme sonucu sonrasında sonraki ileriki zamanlarda çıkabilecek muhtemel hata sayısı tahmin edilmiştir. Değerlendirme esnasında veriler öğrenme ve test verisi olarak ayrılıp, test verisi üzerindeki ortalama mutlak hata, ortalama kare hata metrikleri gibi metrikler kullanılmıştır.

1.1. Hata Öngörü Sistemi

Yazılım geliştirme süreçlerinde kod değişikliklerinden, hata raporlarına, tasarım dokümanlarından, sürüm bilgilerine kadar uzanan bir dizi veri oluşur. Bu veriler kullanılarak yazılımdaki hata oluşabilecek noktalar önceden tahmin edilebilir. Bu tahmini yapan ve yazılım geliştirme sürecinde karar destek işlevini yerine getiren sistemlere hata öngörü sistemleri denir. Hata öngörüleri yazılım geliştirme ekipleri tarafından incelenip gerekli aksiyonlar alınabilir. Tüm öneri ve tahmin sistemlerinde olduğu gibi yüzde yüz kesinlikte sonuç verme yeteneğine sahip değildir. Hata öngörü sistemleri için çok çeşitli yöntemler uygulanmıştır. Bu yöntemlerin başarı oranı (accuracy), karmaşıklık ve işlenen veri tipine göre farklılıklar göstermektedir.

1.2. Sürekli Entegrasyon

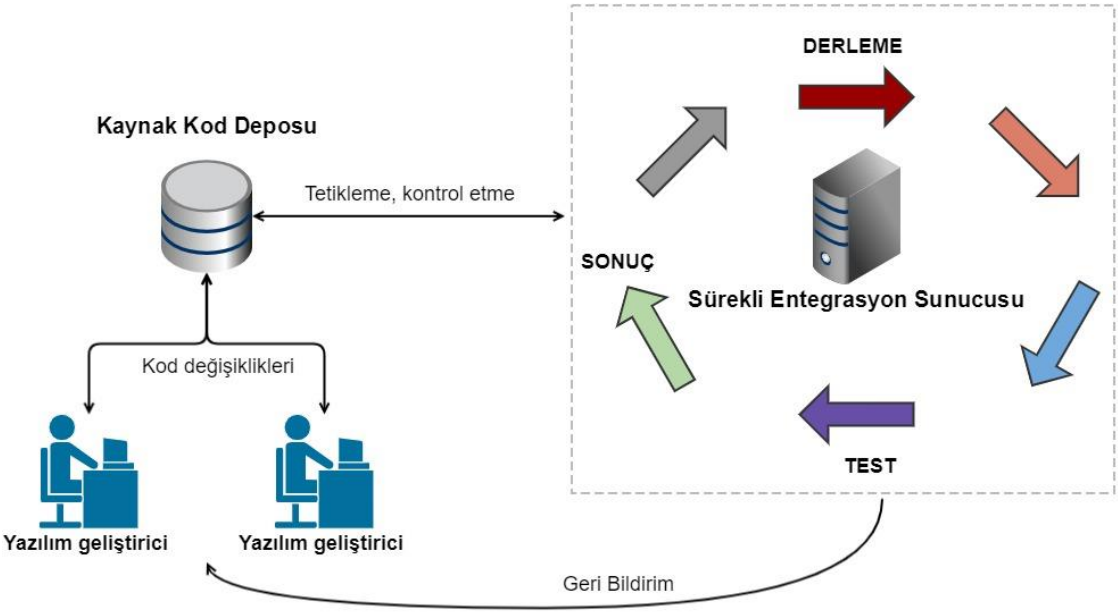
Sürekli entegrasyon bir yazılım projesindeki kod değişikliklerini, merkezi bir yerden entegre edip, derleyip, test etmeyi ve doğrulamayı hedefleyen bir pratiktir [4].

Genellikle yazılım geliştiriciler kod üzerinde yaptığı değişiklikler sonucunda kod deposuna atıp birleştirir. Sürekli entegrasyon araçları da bu kod deposuna bağlanıp değişiklerden haberdar olur. Bu değişiklikleri alarak önceden tanımlanmış yazılım derleme ve test etme gibi adımları kendi ortamında gerçekleştirir. Bu ortam önceden ayarlanmış ve yazılım mimarisine uygun bir ortamdır. Böylelikle yazılımın nasıl bir ortamda hazırlandığı ve yazılım kendisinin derlenebilir ve doğrulanmış olup olmadığı standart bir şekilde tanımlanmış olur.

Sürekli entegrasyon sistemleri her değişiklik sonrasında kontrol işlemi yaptıklarından yazılım geliştiriciler erken zamanda geri dönüş alırlar. Eğer bir problem var ise sürekli entegrasyon sistemi tarafından hemen haberdar olacaklarından problemin ertelenmeyip hızlı bir şekilde çözülmesi sağlanır. Böylelikle ekipler, hatalardan dolayı oluşacak zaman kaybını en aza indirgemiş olur. Ayrıca bu araçlar kodun kalitesi, statik kod metrikleri gibi verileri de sürekli olarak üretebilirler.

Sürekli entegrasyon araçları sadece derleme ve test etme gibi işleri değil, kendisine tanımlanan diğer işleri de yapabilirler. Örneğin bir yazılımı canlı ortama almak için bir dizi işlem yapılması gerekiyor olabilir. Bu işlemleri elle ve sürekli tekrarlayarak yapmak yerine bu araçlara yaptırmak hem hata oranını arttırır hem de zaman kaybı oluşturur.

Şekil 1.1'de sürekli entegrasyon sistemi genel iş akışı ve görünümü belirtilmiştir.



Şekil 1.1. *Sürekli entegrasyon sistemi*

Tüm bu sürekli entegrasyon sisteminin kullanımı süresince birçok veri açığa çıkar. Yazılım projeleri ile ilgili metrikler, derleme ve test işlemlerinin detaylı sonuçları bun verilere örnek olarak verilebilir. Eğer sürekli entegrasyon sistemi sürüm çıkarmak için de kullanılıyor ise sürümler ile ilgili bilgiler bu verilere örnek olarak verilebilir.

Bu çalışmada sürekli entegrasyon sisteminin kod değişiklikleri sonrası başlattığı derleme verileri temel alınmıştır. Bu derleme ile ilgili veriler kod değişikliği ve derleme içindeki işlerin durumları ile bir ön işlem aşaması ile zenginleştirilmiştir. Hata ve sürüm bilgileri de tarihleri ile beraber alınmış ve bir zaman akışında sürümler arasında kaç tane hata olduğu bilgisi alınmıştır. Daha sonra derleme verileri ile hata sayıları ilişkilendirilip makine öğrenme modelleri oluşturulmuştur. Çalışma için seçilen üç adet açık yanak kodlu proje için derlemeler sonrası oluşacak hata sayısı tahminleri yapılmış ve sonuçlar çeşitli istatistiksel yöntemler ile değerlendirilmiştir.

Bu tezin ana hatları sırası ile; yazılım geliştirme süreçleri, hata öngörüsü ve sürekli entegrasyon hakkında temel bilgi aktarımı, mevcut çalışmalar, öngörü oluşturmak için izlenen yöntem, altyapı bileşenleri, kullanılan sistemin mimari yapısı, temel veri modeli ve ön işlem açıklamaları, deneyler ve son olarak değerlendirme ile önerilerden oluşur.

2. MEVCUT ÇALIŞMALAR

Yazılım kalitesi, 1900'lü yılların ortasında, bilgisayar kullanımı yaygınlaşmaya başlamasından beri dikkate alınan bir konu olmuştur. Kalitenin yetersiz olduğu yazılımlar, hata oranı fazla, değişiklik yapılması zor ve paydaşlarına toplam maliyeti fazla bir durumda olurlar. Bu konuda yapılan birçok çalışma yazılım kalitesinin önemi sonucuna varmıştır [5-7]. Hatanın göreceli az olduğu yazılımın kalitesi daha fazla olur. Bu sebeple hata öngörü sistemleri de yazılım kalitesi açısından önemli sistemlerdir.

Hata öngörü sistemleri için çok çeşitli yöntemler uygulanmıştır. Kod metrikleri, süreç verileri ve geçmiş hata verileri çalışmalarda kullanılan işlenen veri tiplerine verilebilecek örneklerdir. Kod metrikleri kodun o anki hali üzerinden sonuç çıkartır iken, süreç verileri ise zamanla yapılan kod değişikliklerini ve bu değişikliklerin hata düzeltme ile olan ilişkilerini de inceler. Ayrıca yazılım geliştirme depolarındaki diğer verilerin kullanıldığı çalışmalar da mevcuttur. Bazı çalışmalarda da farklı yaklaşımların birleşimleri kullanılmış, karşılaştırmaları üzerine sonuçlar çıkartılmıştır. Tüm bu çalışmalarda yazılımın hata veri deposu önemli bir veri kaynağıdır. Bazı çalışmalarda bu veriler hata takip sistemleri üzerinden alınırken, bazı çalışmalarda da hata verileri kod değişikliği verilerindeki metinlerden çıkartılmıştır.

Kod metrikleri, yazılım projesinin mevcut durumunun analiz edilerek ortaya çıkan değerlerdir. Nesne tabanlı programlama metriklerine buna örnek gösterilebilir. Bu tip metriklere göre hataya en yatkın kaynak kod kısımlarını bulmayı hedefleyen çalışmalar yapılmıştır [8, 9]. En sık kullanılan kod kalite metriği olan kod karmaşıklığı [10] hata öngörü sistemleri çalışmalarında kullanılmıştır [11].

Yazılım projelerinde diğer bir kod metriği çeşidi bağımlılık üzerine olan metriklerdir. Yazılımlar kendi kodları dışında, başta kendi platformdakiler olmak üzere birçok farklı çözüm için hazırlanmış kütüphaneleri kullanılır. Geliştirilen yazılım derleme ve/veya çalışma zamanında kullanılan bu yazılım kütüphanelerine bağımlı olur. Bu kütüphane bağımlılıklarını bir grafik modeli çıkartarak hata öngörüsü yapan çalışmalar mevcuttur [12]. Diğer bir bağımlılık metriği çeşidi de sınıf seviyesinde olan bağımlılıkların grafiğidir. Geliştirilen yazılım içindeki sınıfların bağımlılık ağını model olarak kullanan bir hata öngörü çalışması da mevcuttur [13]. Sınıf bağımlılık ağı metriğinin diğer kod metrikleri ile karşılaştırması üzerine bir çalışma diğer kod metrikleri ile benzer sonuç verdiği sonucuna varmıştır [14].

Yazılım değişiklikleri verileri, yazılım geliştirme depoları üzerinde veri madenciliği ve anlamlandırma için sık kullanılan bir veri çeşididir. Yapılan çalışmalarda yazılım değişikliği farklı şekilde tanımlanabilmektedir. Yazılım geliştiricinin sürüm kontrol sistemine gönderdiği iki değişiklik arasındaki fark da bir değişiklik olarak tanımlanabilirken, belirli bir zaman aralığındaki değişikliklerin birleşimi de değişiklik olarak kabul edilebilmektedir.

Yazılım değişikliklerinde birbiri ile beraber değişen dosyaların analizi yapılarak hata öngörü çalışması mevcuttur. Bu çalışmada hatalı bir kod değişikliği ile beraber değişen dosyalardaki hataların tahmin edilmesi başarı incelenmiştir [15]. Diğer bir çalışmada ise birbiri ile değişen kod parçalarının mimari hata ve uyumsuzluklara yol açması inceleyen öngörü sistemi amaçlanmıştır [16]. Başka bir değişiklik verisi kullanan çalışmada ise en sık değişen, en son değişen, en çok hata düzeltmesi olan ve en son hata düzeltmesi olan dosyalar bir puanlama modeli ile sıralanmıştır [17]. Sıralamanın en üstündeki dosyalar en çok hataya sebep olabilecek dosyalar olarak değerlendirilip, yazılım geliştiriciler tarafından kod gözden geçirmelerinde daha dikkatli bir şekilde ele alınabilir.

Yazılım geliştirme depolarında veri madenciliği ve hata öngörü sistemleri alanında üzerinde çalışılmalar yapılmış önemli bir yöntem de *kod çalkantısı* [18] (code churn) değerini kullanarak hata öngörüsü yapma yöntemidir. Kod çalkantısı bir kod değişikliğinin karmaşıklığı belirlemek için kullanılan bir metriktir. Değişiklik içindeki yeni eklenen, değiştirilen ve silinen satır sayısı kullanılarak hesaplanan bir metriktir. Birçok çalışmada kod çalkantısı bir metrik olarak kullanılmıştır. Mutlak aralıktaki kod çalkantısı yerine göreceli aralıktaki kod çalkantı değeri kullanılarak yapılan öngörülerin daha başarılı olduğu bir çalışmada belirtilmiştir [19]. Diğer bir çalışmada ise satır sayısı tabanlı kod çalkantısı ile özgün olarak geliştirilmiş bir anlamlı kod değişikliği modeli karşılaştırılmıştır [20].

Bir hatanın oluştuğu dosyayı işaretleyerek, o dosya ile beraber değişen dosyaları bir modele göre saklayarak hata öngörüsü yapan bir çalışma yapılmıştır [21]. Bu çalışmada hata olan dosyaya yakınlık, dosyanın yeni olup olmadığı, hataya sebep olup olmadığı ve yakında değişikliğe uğrayıp uğramadığı da bütünleşik bir şekilde değerlendirilmiştir. Ödül almış bu yönteme FixCache ismi verilmiştir. Diğer bir çalışmada ise FixCache çalışmasının dosya saklama yöntemi üzerinde iyileştirmeler yapıp performansı arttırılmıştır [22].

Kod değişikliğinin karmaşıklığını, kodun statik değerleri yerine zaman içinde değişikliklerin karmaşıklığına göre tanımlayan, bu tanımladığı metriğe göre zaman içindeki entropiye göre hata öngörüsü yapan bir çalışma mevcuttur [23]. Çalışma sonunda karmaşıklığın fazla olduğu yerlerde hataların daha fazla çıkma olasılığı olduğu sonucuna varılmıştır.

Kaynak kod dosyaları yerine, kaynak koddaki fonksiyonları hedef alarak hata öngörüsü yapılan çalışmalar da mevcuttur [24, 25]. Başarılı sonuç veren bu çalışmalar yazılım geliştiricinin hata düzeltme işi için harcayacağı zamanı modül, paket, dosya veya sınıf yerine doğrudan fonksiyonu işaret ettiği için azaltabilir. Buna ek olarak kod değişikliklerinde beraber değişen kod kalıplarını, fonksiyonları tespit edip buna göre yeni bir değişiklik için unutulan kısımları ve potansiyel hataları öngören çalışmalar da yapılmıştır [26, 27].

Birden fazla yöntemi birleştirerek yapılan çalışmalar hata öngörü sistemlerinde kullanılmıştır [28]. Yakın zamanda çözülmüş hata bilgileri ile statik kod analizi metriklerini kullanarak hata öngören bir çalışma mevcuttur [29]. Kod karmaşıklığı, zaman içinde yapılan kod değişiklik geçmişi ve yazılım geliştirici aktivitesi bilgilerini kullanarak güvenlik açığı hatalarını öngören bir çalışma da yapılmıştır [30].

Yazılım geliştirme deposu dışında bir veri kaynağı kullanan bir çalışmada da hata öngörüsü amaçlanabilir [31]. Bu çalışmada kaynak kodu veya diğer değişiklik süreç verileri yerine programın gerçek ortamda çalışırken ürettiği veriler kullanılmıştır. Programın davranışı izlenerek aykırılık durumları tespit edilmiştir. Bu aykırılık durumu tespit edildikten sonra bir filtreden geçirilerek hatalı pozitif sonuçlar temizlenmiştir. Son olarak programın çalışma sonucuna göre hata öngörüsü doğrulanmıştır. Doğrulanan öngörü sonucunda hatalı çalışan kaynak kodu incelenebilmektedir.

Zamanla oluşan yazılım hata sayıları zaman serisi yöntemi ile analiz edip, hata sayısını tahmin eden bir çalışma mevcuttur [32]. Bu çalışma sonucunda kullanılan örneklerde hata sayılarının mevsimsel ve döngüsel bir kural içinde olduğu tespit edilmiştir. Hata sayısını öngören diğer bir çalışmada ise tahmin üretmek için farklı yöntemler denenmiş ve sonuç olarak birleşik bir yöntemin daha başarılı olduğu sonucuna varılmıştır [33].

Hata öngörü sistemlerinde çok çeşitli veri kaynaklarından metrikler kullanılmıştır. Bu veri özellikleri hata öngörü için girdi olarak kullanılabilmektedir. Kullanılan özellik sayısının azaltılarak hata öngörü performansının arttırılabileceği sonucuna varan bir

çalışma mevcuttur [34]. Özellik azaltmayı otomatik düzenleme yöntemi ile yapan başka bir çalışma da daha performanslı bir sonuca ulaşmıştır [35]. Makine öğrenmesi sistemlerinde kullanılan teknikler ile özellik azaltarak hata öngörüsü üreten bir çalışmada da daha iyi performans gözlenmiştir [36]. Otuz iki özellik seçme yöntemini hata öngörü sistemi üzerinde karşılaştırmalı olarak deneyen bir çalışma ile özellik seçme yöntemlerinin etkileri gözlemlenmiştir [37]. Tüm özelliklerin bir alt kümesini seçme dışında bazı özellikleri makine öğrenmesi teknikleri kullanarak birleştirme yöntemi ile de özellikler azaltılıp performans arttırılabilir [38]. Bu durumu gözlemleyen bir çalışma mevcuttur. Özgün bir makine öğrenmesi ve özellik seçme yöntemi kullanan bir çalışma ile geçmiş çalışmaları karşılaştıran bir çalışma da mevcuttur [39].

Yazılım depolarından çıkarılacak bilgi ile yapılabilecek farklı öngörü çeşitleri de mevcuttur. Örneğin 1990'lı ve 2000'li yıllardaki çoğu eski altyapılı projelerde entegrasyon derlemesi adında elle ve belirli aralıklar ile yapılan bir süreç kullanılıyordu. Özellikle birden fazla ekibin veya yazılım geliştiricinin çalıştığı büyük kod kaynaklarında yapılan değişikliklerin birbirini etkilediği ve nihai ürünün bütünleştirilip edilip derlendiği ve çıktısının alındığı uzun ve elle yapılan süreçler mevcuttu. Bu derleme zaman alan ve geri dönüşü yavaş bir işlem olduğundan yazılım geliştiriciler için zaman kaybı oluşturuyordu. Bu sebeple kod değişiklikleri ile bu tip uzun entegrasyon derlemelerinin sonuçlarını önceden tahmin edip, geliştiriciye erken geri bildirim veren bir çalışma yapılmıştır [40]. Diğer bir öngörü çeşidi ise yapılan kod değişikliğinin yazılım projesinin test setinin çalışma sonucu önceden tahmin etmeyi amaçlayan bir çalışmadır [41]. Bu öngörü, yazılımcı kod değişikliğini yaptığı anda yazılım geliştirme ortamında anında geri bildirim olarak geliştiriciye verilebilir.

Hataların çeşitli özelliklerini kullanarak bir sonraki sürümde düzeltilip, düzeltilmeyeceği öngören bir çalışma mevcuttur [42]. Ayrıca hata repolarında sıklıkla kullanılan bir özellik olan önem derecesi değerini hatanın diğer özelliklerine göre tahmin eden bir çalışma da mevcuttur [43]. Düzeltilmiş bir hatanın tekrar açılıp açılmayacağını makine öğrenmesi teknikleri ile tahmin eden bir çalışma mevcuttur [44]. Yazılımın zaman içinde eskimesi sebebiyle oluşan hataları öngören bir çalışma da mevcuttur [45]. Diğer bir çalışmada ise hata raporunun incelenerek hatanın bir ayarlama hata olup olmadığı sonucuna doğal dil işleme yöntemleri ile ulaşılmak istenmiştir [46].

Yapılan bazı çalışmalarda hata öngörü sistemlerinin yazılım geliştiriciler üzerinde etkisi araştırılmıştır [47]. Bu çalışmalarda yazılımı doğrudan geliştiren kişilerin hata

öngörü sistemine yaklaşımı gözlenmiş ve aksiyon alınabilir geri bildirimlerin daha faydalı olacağı belirtilmiştir. Ayrıca yazılımın kalitesi ile ilgilenen personelinin hata öneri geri bildirimlerine daha ilgili olabileceği sonucuna varılmıştır. Yazılım geliştiricinin kod değişikliğindeki dağınıklık davranışı ile bir metrik kullanan bir çalışma da mevcuttur [48]. Başka bir çalışmada ise bir hata öngörü sisteminin açık kaynak kodlu bir yazılım projesindeki performansından daha iyi bir performansı endüstrideki bir kapalı kaynak kodlu projede elde ettiği gözlemlenmiştir [49]. Hata öngörü sistemlerinde kullanılmak üzere açık bir hata öngörü veri seti yaratma çalışması da diğer çalışmalar için faydalı bir çalışmadır [50].

Hata öngörü sistemleri için proje özgü kurallar ile genel kurallar arasında bir karşılaştırma çalışması yapılmıştır [51]. Projeye özgü kural setinin daha başarılı olduğu gözlemlenmiştir. Ancak başka bir çalışmada birçok yazılım projesini belirli projeye özgü metrikler ile gruplayarak evrensel bir hata öngörü sistemi modeli yaratma çalışması da mevcuttur [52].

Çeşitli hata öngörü yöntemlerini karşılaştıran çalışmalar da hata öngörü alanına katkı yapmıştır. D'Ambros ve arkadaşlarının yaptığı çalışmada tek tek yöntemler ve bu yöntemlerin birleşimleri karşılaştırılmıştır [28]. Birden fazla yöntemin birleşiminin çok daha iyi sonuç verdiği gözlenmiştir. Ancak bazı çalışmalarda kullanılan veriler açık olmayan kurumsal veriler olduğundan bu çalışmalar, sadece bazı izole koşullarda belirtilen sonucu vermiştir. Bu sebeple aynı yöntem farklı çalışmalarda farklı sonuçlar doğurmuştur. Yazılım geliştirme depoları kullanarak veri madenciliği için geniş bir yöntem analizi çalışması da mevcuttur [2]. Yazılım deposu veri madenciliği alanını kısaca tanıtan, yapılan çalışma ve elde edilen başarıları özetleyen, karşılaşılan zorluklardan bahseden bir çalışma mevcuttur [53]. Hata öngörü sistemlerinin performansını ölçmek için özgün bir yöntem de başka bir çalışma ile geliştirilmiştir [54]. Değişiklik verilerinin statik kod verilerine göre daha iyi performans verdiği sonucuna ulaşan bir çalışma da yapılmıştır [55]. Dikkat çeken diğer bir karşılaştırma çalışmasında ise çeşitli hata öngörü sistemlerinin içyapıları ve ayarlamaları incelenmiştir [56]. Bu çalışmaya göre statik kod metrikleri ile beslenen değişiklik verileri kullanılarak, "random forest" [57] makine öğrenme tekniği kullanılması önerilmiştir. Ayrıca öngörü hedefi olarak da hata sayısının kullanılmasının en uygun olduğu sonucuna varılmıştır.

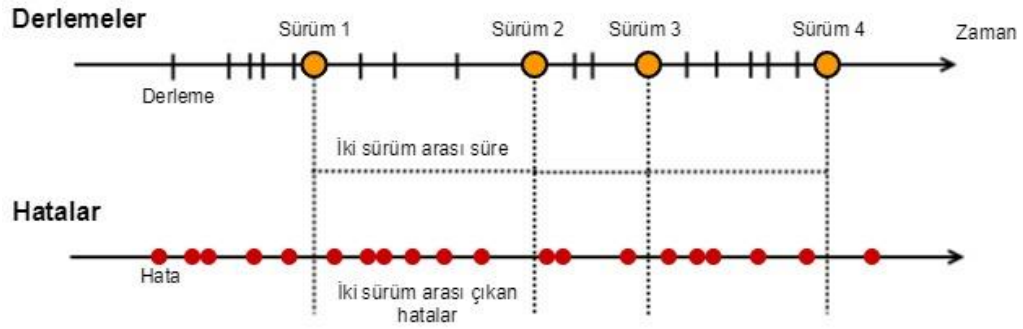
Bu çalışmada özellik seçimi, hata sayısı öngörü hedefinin daha uygun bir öngörü hedefi olduğu, test sonucunun bir metrik olarak kullanılabileceği önceki çalışmalardan

faydalanarak çıkartılmıştır. Sürekli entegrasyon sistemi verilerinin kullanılabileceği ise eskiden elle ve daha seyrek yapılan birleşik derlemeyi tahmin eden sistemden esinlenerek yapıldığı söylenebilir. Birçok çalışmada ise makine öğrenmesi teknikleri kullanılmıştır. Bu sebeple makine öğrenmesinin başarısının bu çalışmada da kullanılması hedeflenmiştir.

3. YÖNTEM VE ALTYAPI

Bu çalışmadaki geliştirilen hata öngörü sisteminin amacı, sürekli entegrasyon derleme sonuçları kullanılarak, yazılım projesinde oluşacak hata sayısı arasındaki ilişkiyi incelemektir. Oluşturulan model ile bir sonraki derleme sonucunda ileride oluşabilecek hata sayısı için öngörü üretilmiştir.

Bir yazılım projesi yaşam döngüsü süresince hata raporları oluşabilmektedir. Bu hata raporları yazılımın herhangi bir sürümünü etkilemiş olabilir. Bu çalışmadaki amaç oluşan hata sayısını öngörmenin yanı sıra, bu hataların ne zaman oluşacağı hakkında bir sonuç belirtmektir. Hedeflenen öngörü sonucuna şu cümle örnek olabilir: "Bu sürekli entegrasyon derlemesi ile bir sonraki sürümde (veya 5 gün içinde) dört adet hata çıkabilir.". Bu çalışmada hata sayısının tahmin edildiği aralık için iki yöntem ayrı ayrı denenmiştir. İlk yöntemde projenin iki sürümü arasındaki geçen sürenin ortalaması alınmıştır. Öngörü hedefi olarak derleme zamanı sonrasında bu geçen süre içinde çıkan hata sayısı kullanılmıştır. İkinci yöntemde ise derleme zamanından itibaren bir sonraki gelen sürüm içinde çıkan hata sayısı kullanılmıştır.

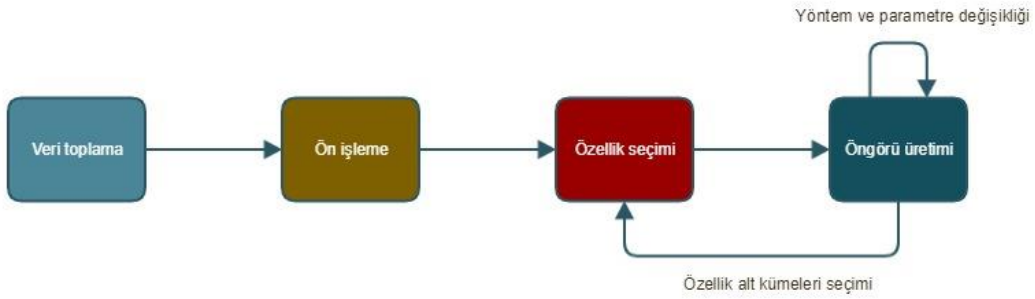


Şekil 3.1. *Derleme, hata ve sürümler*

Şekil 3.1'de öngörü hedefi, sürekli entegrasyon derlemeleri ve hatalar arasındaki zamansal ilişki gösterilmiştir.

Bu çalışmadaki iş akışı adımları sırası ile veri toplama, ön işleme, hazırlama, özellik seçimi ve öngörü üretimidir. Veri toplama adımında seçilen bir projenin sürekli entegrasyon derleme sonucu ile ilgili bilgiler, hata kayıt sistemindeki hata verileri ve sürüm kayıt sistemindeki veriler toplanır. Veri hazırlama adımında önemsiz alanlar ve veri tipi uyumsuz alanlar düzenlenir. Özellik seçimi işlemleri ve istatistiksel analiz

yöntemleri ile tüm özelliklerin alt kümesi oluşturulur. Öngörü hedef verisi ile öngörü sisteminde özellik olarak kullanılacak verileri boş olanlar kayıtlar, öngörü öncesinde temizlenir. Öngörü aşamasında çeşitli makine öğrenme algoritması ile modeller oluşturulup öngörü tahmin performansı birden fazla metrik ile değerlendirilir.



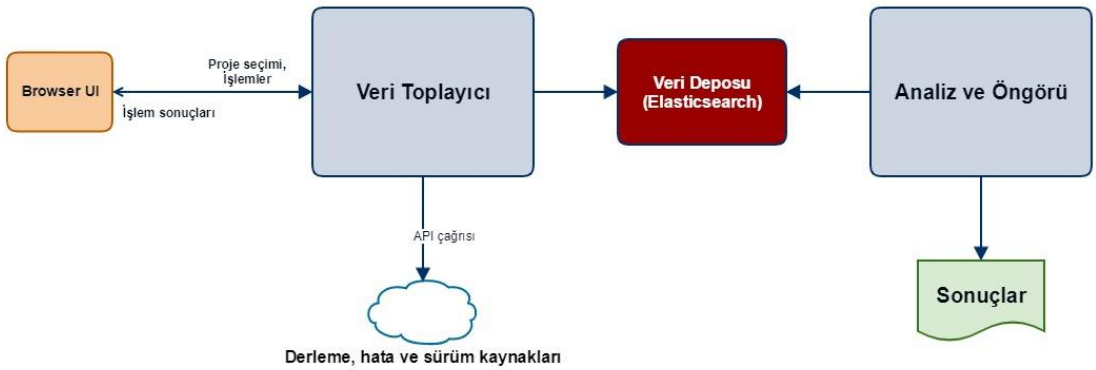
Şekil 3.2. *Genel iş akışı*

Şekil 3.2'deki iş akışında da görüldüğü gibi hata öngörüsünden önce özellik seçme adımı bulunmaktadır. Bu adım genel olarak tüm makine öğrenme çalışmalarında da olduğu gibi önemli bir adımdır. Özellik seçilirken üç yöntem denenmiştir: sezgisel seçim, literatürdeki özellik seçimi algoritmaları destekli bir yöntem ile seçim ve tüm özelliklerin seçimidir. Yazılımda yapılan değişiklik ile olan derlemelerde, yazılımda yapılan kod değişikliği ile ilgili bir özellik kullanılmıştır. Bu özellik kod değişiklik satır sayısı olarak veride bulunur. Bu durum daha önceki araştırmalardaki kod çalkantısı metriğinin yarattığı sonuçların kullanılıp üzerine sürekli entegrasyon derlemesi verilerinin de eklenip bir hata öngörüsü yapılmasına sebep olmuştur [18].

3.1. Altyapı

Bu çalışmada geliştirilen altyapıda çeşitli iş parçaları için modüller oluşturulmuştur. Veri toplama ve ön işleme işleri bir web uygulaması aracılığıyla yapılır. Bu uygulama seçilen yazılım projesi için tanımlanmış derleme sistemine, iş takip sistemine ve sürüm sistemine bağlanarak verileri alır. Alınan veriler bir ön işlem sürecinden geçirilir. Bütün işlemler adım adım yapılır. Böylelikle herhangi bir hata durumunda işlemler kalınan yerden devam eder. Veriler ortak bir veri deposuna kaydedilir. Bu veri deposu belirli bir formatta yapısı olan veriyi değil, alınan veriye göre dinamik bir yapıda veriyi saklar. Daha sonra özellik seçimi ve öneri üretme işleri veri analiz betikleri ile gerçekleştirilir. Bu

betikler ortak veri deposuna bağlanarak analiz işlemlerini gerçekleştir. Verinin istatistiksel analizi de bu betikler ile yapılır. Algoritma parametreleri gibi değişken değerlere göre birçok deney bu betikler ile yapılır. Deneylerin sonuçları konsol çıktısı olarak görülür. Ayrıca birçok performans metriğinin sonucu da üretilir. Bu çeşitli metriklere göre kullanılan algoritmanın, seçilen özelliklerin ve algoritma parametrelerinin performansı değerlendirilir.



Şekil 3.3. *Tüm sistem altyapısı genel görünümü*

3.1.1. Veri modeli

Öngörü sisteminde kullanılmak üzere temelde üç veri modeli bulunmaktadır. Bunlar derleme, hata ve sürüm bilgisidir.

Derleme bilgisi sürekli entegrasyon sistemlerinden toplanır. Bu çalışmada seçilen yazılım projeleri Travis [58] sürekli entegrasyon sistemini kullanmaktadır. Derleme bilgileri bu sistemdeki veri yapısını temel almaktadır. Derleme bilgisi temelde numara, durum, önceki durum, süre, olay tipi, başlangıç zamanı, bitiş zamanı, kod değişikliği, yapılan derleme işi detayları, tetikleyen kullanıcı bilgilerin içerir. Bu verilere ek olarak bazı yeni özellikler de türetilmiştir. Bunlar başarılı derleme işi sayısı, başarısız derleme işi sayısı ve kod değişiklik sayısıdır. Başarılı ve başarısız derleme sayıları derleme işi detaylarından türetilirken, kod değişiklik sayısı da değişikliğin detay bilgilerinden oluşturulur. Bu bilgi değişiklik yapılan satır sayısı olarak ek bir özellik olarak derleme verisine eklenir.

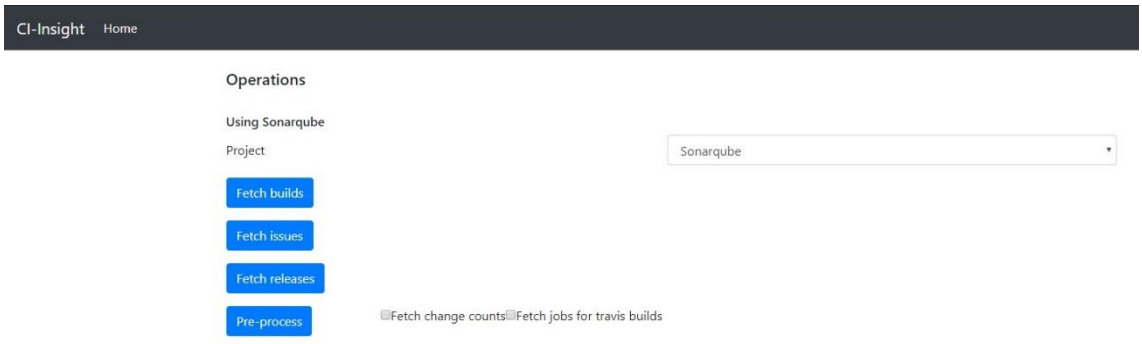
Hata verisi yazılım projesinin kullandığı hata takip sisteminden alınır. Bu çalışmada kullanılan bazı yazılım projeleri için JIRA [59] iş takip sistemi verisi, bazı projeler için ise GitHub [60] iş takip sistemi verisi kullanılmıştır. Bu kaynaklar doğrultusunda hata

bilgisi numara, adres, başlık, oluşturan kullanıcı, etiket, durum, üzerine atanan kullanıcı, yorum sayısı, oluşturulma zamanı, son güncellenme zamanı, kapanma zamanı ve açıklama özelliklerine sahiptir.

Sürüm verisi yazılımın kullandığı sürüm bilgisinin kaydedildiği sistemden alınır. Bu sistem genellikle hata verisinin olduğu sistem ile aynı sistem olur. Bu çalışmada seçilen yazılımım projeleri için GitHub ve JIRA sürüm yönetim sistemi verileri kullanılmıştır. Sürüm verisi numara, isim, sürüm çıkış tarihi özelliklerine sahiptir. Sürüm verisinin en önemli özelliği çıkı tarihidir. Bu tarihe göre belirli aralıklardaki hata sayısı verisi kullanılmaktadır.

3.1.2. Veri toplayıcı

Veri toplayıcı uygulamasının amacı kullanıcının seçtiği yazılım projesi için derleme, hata ve sürüm bilgilerini toplayıp, ön işlemden geçirerek veri deposuna kaydetmektir. Uygulama ara yüz kısmı ve iş mantığı kısmı olarak iki temel parçadan oluşmaktadır. Uygulamanın ara yüz kısmı "tek sayfalı web uygulaması" (single page web application) [61] olarak geliştirilmiştir. Angular [62] çatısı ara yüz kısmında kullanılmıştır. Ara yüzün işlemler için kullandığı iş mantığı kısmında ise Spring Boot [63] çatısı kullanılmıştır. Bu uygulamada öncelikle daha önceden ayarlanmış yazılım projeleri listelenir. Kullanıcı bu listeden işlem yapacağı projeyi seçer. Seçtiği proje üzerinde derleme toplama, hata toplama, sürüm toplama ve ön işleme işlemlerini gerçekleştirebilir.



Görsel 3.1. *Veri toplayıcı işlemleri*

Veri toplayıcı için seçilen yazılım projeleri, Travis sürekli entegrasyon sistemini kullanmaktadır. Derleme verisi toplama işlemi için Travis sisteminin uygulama programlama ara yüzü kapsamındaki web servis uç noktaları kullanılır. Derleme verisi JSON formatında alınır ve ön işleme öncesi olduğu gibi veri deposuna aktarılır. Veri deposunda dinamik yapılı depolama sistemlerinden biri olan Elasticsearch [64] tercih edildiğinden doğrudan aktarmada uyumsuzluk olmamaktadır. Elasticsearch deposuna veriler aktarılırken yüksek seviye istemci kütüphanesi kullanılmıştır. Bu uygulama kütüphanesi alt seviye ağ bağlantıları yerine uygulama seviyesi ağ protokolleri bağlantı sağlamaktadır.

Hata verileri için iki tip kaynak kullanılmıştır. Bunlardan biri JIRA tabanlı bir iş takip sistemi, diğeri de GitHub sistemidir. Her iki kaynaktan da veriler web servis uç noktaları kullanılarak toplanmıştır. Hata verileri de JSON formatındadır ve kolaylıkla dinamik yapılı depolama sistemine aktarılır.

Seçilen yazılım projeleri sürüm bilgileri için de GitHub sistemini kullanır. Burada yazılımın çıkartılmış tüm sürümleri ve tarih bilgisi bulunmaktadır. GitHub bünyesindeki web servis uç noktasından alınan JSON formatındaki veriler depolama sistemine aktarılır. GitHub sisteminde iki tip sürüm verisi olabilir. Bazı projeler doğrudan sürüm model tipini kullanırken, bazı projeler etiket (tag) kullanarak sürümlerini belirtmiştir. Toplayıcı her iki tip sürüm bilgisini de toplayabilecek şekilde geliştirilmiştir.

Toplama işlemlerinden sonra en önemli işlem olan ön işlem adımına sıra gelir. Bu adımda eksik özellikler mümkünse toplanır ve öngörü hedef bilgisi tüm derlemeler için ayrı ayrı hesaplanıp derleme verisine eklenir. Derleme ham verisinin içinde değişiklik sayısı bulunmadığından derlemeye sebep olan kod değişikliğinin verisi toplanarak buradaki satır sayısı değişikliği özelliği derleme verisine eklenir. Ayrıca bir derleme içindeki başarılı ve başarısız iş parçacığı sayısı verisi de ek olarak toplanıp derleme verisine eklenir. Tüm bu eklenen özellikler öngörü oluşturma da yardımcı olmaktadır. Daha önceki çalışmalardaki kod çalkantısı metriği, bu çalışmadaki derlemeye sebep olan kod değişikliğinin çalkantısı şeklinde kullanılmış ve doğru öngörüye ulaşmak için katkısından faydalanılmıştır. Ayrıca derleme içindeki iş parçalarının ayrı ayrı durumlarının öngörüye katkısı kullanılmak istenmiştir.

Ön işlem adımının en önemli ve gerekli adımında öngörü hedef verisinin hesaplanması gelmektedir. Bu çalışmada iki adet öngörü hedef verisi birbirinden bağımsız olarak kullanmıştır. Bunlardan ilki bir sonraki sürümde çıkan hata sayısı,

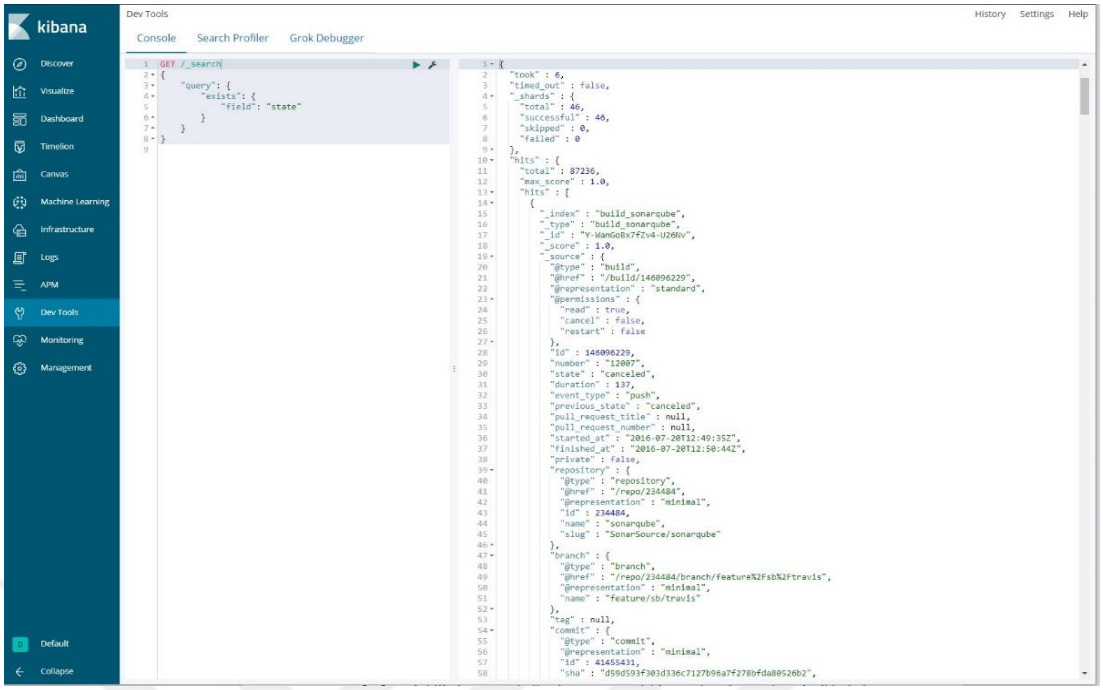
ikincisi ise ortalama sürüm süresi sonrası olan aralıkta çıkan hata sayısıdır. Toplayıcı uygulamasının ön işlem aşamasında tüm sürümlerin ortalama süresi hesaplanır. Her bir derleme için derlemenin olduğu zamandan ortalama sürüm süresi sonrası olan hata sayısı hesaplanır bir öngörü hedefi olarak kaydedilir. Diğer öngörü hedefi için ise tüm sürümler alınıp kronolojik olarak bir veri yapısında sıralanır. Tüm derlemeler için derlemenin olduğu zamandan bir sonraki sürüm bu veri yapısı vasıtası ile bulunur. Bu sürümdeki hata sayısı da diğer bir öngörü hedefi olarak kaydedilir.

3.1.3. Veri deposu

Bu çalışmadaki veriler birçok kaynaktan gelebilmektedir. Bu sebeple veri yapısı sabit olan depolama çözümleri yerine dinamik veri yapıları ile uyumlu bir sistem tercih edilmiştir. Elasticsearch [64] depolama ve arama işlemleri için kullanılan, dinamik veri yapıları ile uyumlu, açık kaynaklı bir yazılımdır. Bu çalışmadaki veriler Elasticsearch üzerindeki "index" adı verilen alanlara kaydedilmiştir. Kaynaklardan JSON formatında gelen ham veri doğrudan veri deposuna kolaylık konulabilmektedir. Çünkü Elasticsearch de veriyi kendi içinde JSON dokümanları olarak kaydetmektedir.

Çalışmada kullanılan her bir yazılım projesi için bir kısa anahtar değeri bulunmaktadır. Örneğin "Sonarqube" yazılım projesinin anahtar değeri "sonarqube" değeridir. Derleme, hata ve sürüm verileri kaydedilirken projelerin anahtar değerleri Elasticsearch index isimlerinde kullanılmıştır. Örneğin Sonarqube projesi için derlemeler "build_sonarqube" index bölgesinde, hatalar "issue_sonarqube" index bölgesinde, sürümler ise "release_sonarqube" index bölgesinde kaydedilmiştir.

Veri deposu ve altyapısının kurulumu için kap (container) yöntemi kullanılmıştır. Kap altyapısı olarak Docker [65] kullanılmıştır. Böylelikle çok hızlı bir kurulum gerçekleşmiş ve çalışmaya en kısa başlanılmıştır. Ayrıca Elasticsearch veri depolama yazılımı için önerilen bir arama ve veri gözleme aracı olan Kibana [66] da kap yöntemi ile kurulup, çalıştırılıp, geliştirme esnasında kullanılmıştır.



Görsel 3.2. *Kibana ve örnek elasticsearch sorgu sonuç verisi*

3.1.4. Analizci ve özellik seçici

Analiz, özellik seçimi ve öngörü üretimi modülleri için Python [67] ve Anaconda [68] veri bilimi ortamı kullanılmıştır. Buradaki uygulamalar Python betiklerinden oluşmaktadır ve yapılacak işler için Jetbrains PyCharm [69] geliştirme ortamında tetiklenip çalıştırılmaktadır.

İlk önce veri deposu olan Elasticsearch istemcisi kullanılarak proje verileri alınır. Alınan veriler Python ortamında makine öğrenmesi işlerinde çok sık kullanılan Pandas ve NumPy [70] veri seti tiplerine dönüştürülür.

Öncelikle derleme verisinin ön analizi bu kısımda yapılır. Derlemedeki tüm özellikler ve veri tipleri listelenir. Uyumsuz veri tipleri uyumlu tiplere çevrilir. Metin tabanlı olan ancak kategorik değer kümesine ait olan özelliklerin tipi de kategorik özel bir tipe dönüştürülür.

Özellik seçmeden önce derleme verisindeki eşsiz olan özellikler silinir. Öngörü hedeflerinin boş olduğu derleme verileri de silinir. Daha sonra projenin derleme verisi için bir dizi istatistiksel grafik oluşturulur. Özelliklerin hedef değerleri dahil karşılıklı ilişkilerini gösteren ısı grafiği (heatmap) üretilir. Ayrıca hedef değerlerin diğer özellikler ile olan ilişkisini gösteren ikiz grafik (pairplot) üretilir. Son olarak hedef değerlerin kendi içindeki dağılımını gösteren çeşitli grafikler üretilir. Tüm bu grafiklerin örnekleri

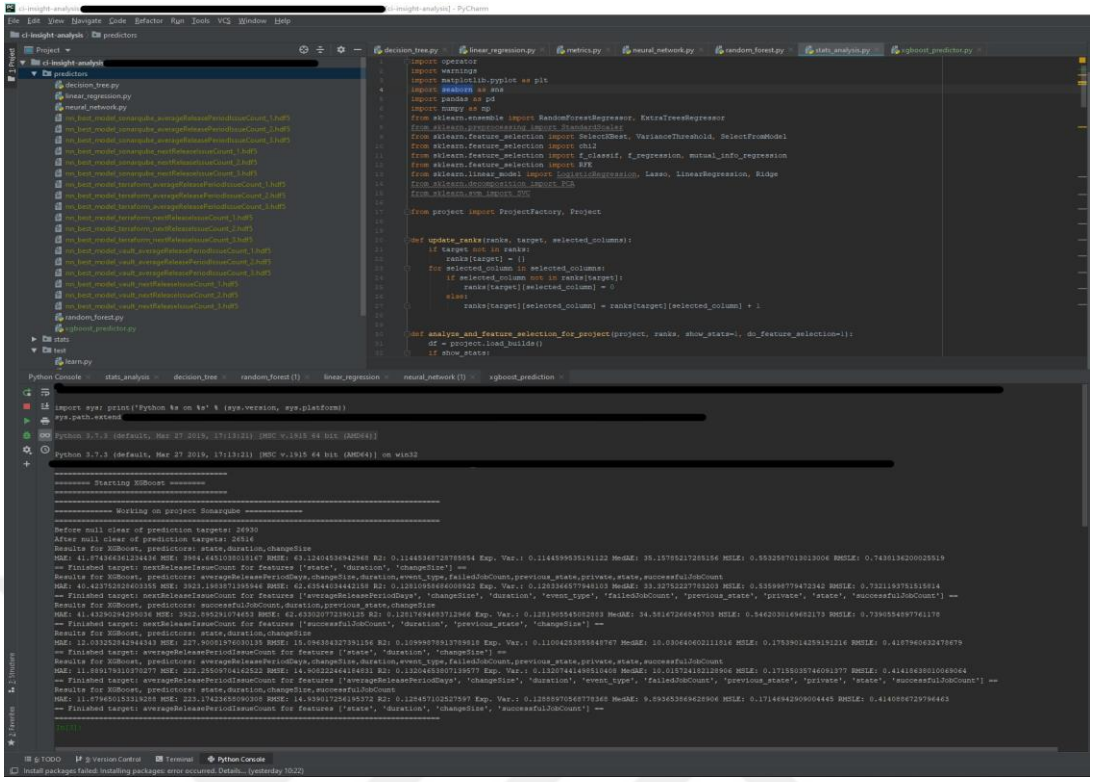
deneyler kısmında bulunmaktadır. Grafikler Seaborn ve Matplotlib [71] kütüphaneleri kullanılarak oluşturulmuştur.

Özellik seçme, makine öğrenmesi iş akışında önemli yere sahiptir. Gerek algoritmanın performansını, gerekse de hızını arttırmak için özellik seçme işlemi birçok makine öğrenme çalışmasında uygulanmaktadır [72]. Özellik seçme işlemleri için Python makine öğrenme kütüphanesi olan Scikit-learn [73] kullanılmıştır.

3.1.5. Öngörü üretici

Öngörü üretici de özellik seçici gibi Python, Anaconda ve Jetbrains PyCharm araçları kullanılarak geliştirilmiştir.

Öngörü üretici öncelikle hangi öngörü hedefi için çalışacağını (iki tane var), hangi özellikleri kullanacağını seçer. Daha sonra öncelik hedef verisinin ve seçilen özellik verisinin boş olduğu derleme verileri silinir ve algoritmalar için kullanılmaz. Ayrıca kaynakta metin tipinde olan ancak kategorik olarak sabit değerleri olan, "state", "previous_state" özellikleri kategorik ve sayısal veri tipine dönüştürülür. Çünkü bu çalışmadaki makine öğrenme algoritmaları sayısal veriler ile çalışırlar. Veri öğrenme ve test olarak rastgele bir biçimde ikiye ayrılır. Verinin %25'i test veri olarak, geri kalanı da öğrenme verisi olarak kullanılır. Tüm bu işlemler ve makine öğrenme işlemleri için Scikit-learn ve Keras [74] kütüphanesi kullanılmıştır. Öngörü sistemleri, hedef değerlerinin tipine göre ikiye ayrılabilir. Bunlar kategorik hedef verisi olan ve sayısal hedef verisi olan sistemler olabilir. Bu modülde sayısal hedef öngörüsü üretmek için kullanılabilecek bir dizi makine öğrenme yöntemi denenmiştir. Öncelikle ikiye ayrılan verinin öğrenme kısmı ile makine öğrenme modeli oluşturulur ve diğer test kısmı ile de oluşturulan model üzerinden öngörü oluşturulur. Öngörünün üretiminin performansını ölçen çok çeşitli yöntemler bulunmaktadır. Bir dizi makine öğrenme performans metriği kullanılmıştır. Bu metrikler sayısal hedef değerine uygun metriklerdir.



Görsel 3.3. *Özellik seçme ve öngörü betikleri, konsoldaki sonuçlar*

3.2. Özellik Seçme ve Öneri Üretme

Özellik seçme, makine öğrenmesi işlemlerinden önce verinin hangi alanlarının kullanılacağının belirlendiği adımdır. Özellik seçme yöntemleri kabaca üç kategoriye ayrılmaktadır. Bunlar filtreleme tipi, kapsama tipi (wrapper) ve gömülü tipidir. [72]. Filtreleme tipinde olan istatistiksel olarak verinin en önemli özelliklerini bulmaya çalışır. Bu süreçte hiçbir makine öğrenmesi yöntemi kullanmaz. Bu sebeple herhangi bir makine öğrenmesi algoritmasından bağımsız olarak çalışır. Kapsama tipli olan özellik seçme yöntemleri bir makine öğrenme algoritması kullanarak özelliklerin performans değerini belirler. Bu değere göre de özellikler seçilir. Gömülü yöntemlerde ise özellik seçimi algoritmanın öğrenme aşamasında tekrarlı bir şekilde yapılır.

Bu çalışmada üç şekilde özellikler seçilmiştir. İlkinde sezgisel olarak bazı özellikler seçilmiştir. İkinci yöntemde tüm özellikler makine öğrenmesi için kullanılmıştır. Son yöntemde ise çeşitli özellik seçme yöntemleri veri üzerinde uygulanıp, çıkan sonuçlardaki özellikler puanlanarak en yüksek puanı alan dört özellik seçilmiştir. Bu yöntem bir çeşit topluluk tabanlı (ensemble) bir yöntem olarak tanımlanabilir. Bütün öngörü üretme işlemlerinde tüm bu özellik seçme yöntemleri ayrı ayrı denenmiş ve sonuçları oluşturulmuştur.

Sezgisel özellik seçiminde derlemelerin "durum", "süre" ve "kod değişikliği" özellikleri seçilmiştir. Durum bir derlemenin sonuç durumu belirtir. Bu sonuç başarılı, başarısız, iptal gibi değerler olabilir. Durum özelliğinin seçilmesindeki sebep başarısız derlemelerin olduğu zamanlarda hata durumunun bundan etkilenmesinin düşünülmesidir. Süre özelliğinin seçilmesinin sebebi ise uzun süren derlemede projedeki boyutunun büyüdüğü ve hata sayısının bundan etkileneceğinin düşünülmesidir. Kod değişikliği özelliği ise birçok hata öngörü sisteminde kullanılan başarılı bir metriktir. Hata sayısı öngörüsünü üretirken bu özellikten de yararlanmak düşünülmüştür.

Bu çalışmada kullanılan diğer bir özellik seçme yönteminde literatürdeki çeşitli özellik seçme algoritmaları kullanılmıştır. "Variance Threshold" kullanılan basit algoritmalardan birisidir [75]. Belirli bir değerin altında değişkenliğe sahip olmayan özellikler elenir. Kullanılan diğer bir özellik seçme algoritması için "chi-square" olarak adlandırılan bir algoritmadır [76]. ANOVA isimli algoritma da sayısal öngörü hedefi olan veriler için uygun olan bir özellik seçme yöntemidir. Bu yöntem ile de özellik seçme uygulaması yapılmıştır [77]. "F-Regression" [78], "mutual info regression" [79] ve "RFE" [80] diğer kullanılan kapsama tipli özellik seçme algoritmalarıdır. Gömülü yöntemlerden olan "Lasso" [81] ve "Ridge" [82] algoritmaları da özellik seçme için kullanılmıştır. Bu çalışmada kullanılan, makine öğrenme algoritması içeren diğer özellik seçme algoritmaları "Linear Regression" [83], "Random Forest Regressor" [57] ve "Extra Trees Regressor" [84] olarak listelenebilir. Bu özellik seçme algoritmalarından bazıları parametreler için önerilen varsayılan ayarlar kullanılmıştır.

Tüm bu özellik seçme yöntemleri sonucunda seçilen özellikler derecelendirilmiştir. Örneğin bir özellik bir yöntem sonucu seçilmiş ise 1 puan alır. Tüm özellik seçme işlemleri sonucunda özellikler puanlarına göre büyükten küçüğe sıralanır. En yüksek puanı alan dört özellik makine öğrenme işlemleri öncesinde seçilir. Böylelikle topluluk tabanlı bir özellik seçme sonucu oluşmuştur.

Seçilen özelliklerden sonra deneyler iki ana parametre ile yapılır. Bunlardan ilki öngörü hedef özelliğidir. Daha önce de belirtildiği gibi bu çalışmada ortalama sürüm süresi sonrası çıkan hata sayısı ve bir sonraki sürümde çıkan hata sayısı olmak üzere iki adet öngörü hedefi kullanılarak çalışılmıştır. Öngörü hedefinden sonraki parametre, makine öğrenme algoritması için yukarıda anlatılan yöntemlerden biri kullanılarak seçilen özelliklerdir. Deneylerde, öngörü hedefinin ve özellik seçimi yöntemlerinin tüm kombinasyonları kullanılmıştır. Kullanılan makine öğrenme algoritmasının çeşidine göre

bazı model hiperparametreleri [85] de değişik kombinasyonlarda kullanılmıştır. Bazı durumda hiperparametreler için varsayılan veya sabit değerler kullanılmıştır.

Öngörü üretirken 5 farklı makine öğrenme algoritması kullanılmıştır. Bunlar: "Linear Regression" [83], "Decision Tree" [86], "Random Forest" [57], "XGBoost" [87] ve "Sequential Neural Network" [88, 89] algoritmalarıdır. Bu algoritmalar sayısal veri tahminleri için daha uygun olması sebebiyle seçilmiştir. "Linear Regression" algoritmasın herhangi bir ek parametresi yoktur ve veriler normalleştirilmeden öngörüler üretilmiştir. "Decision Tree" algoritmasında maksimum dallanma sayısı isimli önemli bir parametre mevcuttur. Bu parametre arttırıldığında belirli bir noktaya kadar performansta bir iyileşme gözlenir ancak daha fazla arttırıldığında performans ya düşer ya da daha fazla yükselmez ve çalışma süresi de artar. Bu çalışmada maksimum dallanma parametreleri 50, 500, 5000 ve 50000 olarak seçilip öngörülerde tüm kombinasyonlar denenmiştir. "Random Forest" yönteminde ise tahminleyici sayısı ve hata ölçme kriteri olmak üzere iki adet parametre mevcuttur. Tahminleyici sayısı için 10 ve 50 değerleri, hata ölçme kriteri için ise ortalama hata değeri ile ortalama hata değerinin karesi metrikleri kullanılmıştır. Tüm bu parametrelerin kombinasyonları ile diğer yöntemlerde olduğu gibi öngörüler oluşturulmuştur. "XGBoost" algoritmasının birçok parametresi mevcuttur. Özellik seçimi gibi, XGBoost için parametre optimizasyonu ve seçimi adında bir ön işlem adımı diğer makine öğrenme çalışmalarında mevcuttur [90]. Bu çalışmada da XGBoost parametreleri rasgele seçilip model doğrulama yöntemi [91] ile en performanslı hiperparametre değerleri otomatik olarak seçilmiştir. Son olarak "Sequential Neural Network" yapay sinir ağı algoritması kullanılmıştır. Üç gizli katmanlı bir yapay sinir ağı oluşturulmuştur. Veri giriş katmanının boyutu seçilen özellik parametresinin sayısı ile aynıdır. Veri giriş katmanı çıkışı 128 nöron dan oluşur. Daha sonra gelen gizli katmanların ise 256 çıkış boyutu vardır. Giriş ve gizli katmanlarda aktivasyon fonksiyonu olarak ReLU [92] fonksiyonu kullanılmıştır. Son çıkış katmanı hata sayısının öngörüsünün oluştuğu katmandır, aktivasyon fonksiyonu lineer fonksiyondur ve çıktı boyutu tek boyutludur. Yapay sinir ağı derlenirken hata fonksiyonu değeri olarak ortalama mutlak hata metriği kullanılmış ve optimizasyon için "adam" [93] yöntemi kullanılmıştır. Derlenen yapay sinir ağı ile model oluşturmak için öğrenme aşaması gerçekleştirilir ve en iyi model dosya olarak kaydedilir.

Tüm makine öğrenme algoritmaları parametreleri belirlendikten sonra yazılım proje verisi öğrenme ve test olmak üzere iki parçaya ayırılır. Test parçası üzerinde yapılan

öngörülerin performansı çeşitli metrikler ile değerlendirilir. Bu metrikler “ortalama mutlak hata”, “ortalama mutlak hata karesi”, “ortalama mutlak hata karesi karekökü”, “R2 değeri”, “explained variance score”, “Median mutlak hata”, “ortalama logaritmik hata karesi” ve “ortalama logaritmik hata karesi karekökü” olarak listelenebilir. Deney sonuçlarında tüm bu metriklerin tüm kombinasyonlarda değerleri görülüp performans değerlendirilmiştir.

4. DENEYLER

Bu çalışmada daha altyapı ve mimari bölümünde detayları anlatılan uygulamalar ile deneyler yapılmıştır. Deneyler için üç adet popüler ve geliştirilmesi aktif olan açık kaynaklı yazılım projesi seçilmiştir. Bu projelerin sürekli entegrasyon derleme sistemi verileri, hata takip sistemi verileri ve sürüm verileri de aynı zamanda açık ve erişilebilir durumdadır. Seçilen üç proje şunlardır: Sonarqube [94], Terraform [95] ve Vault [96]. Sürüm, derleme ve hata verileri projelerin başlangıcından 9 Mayıs 2019 tarihine kadar olan verilerdir.

Tüm projeler sürekli entegrasyon sistemi olarak Travis CI sistemini kullanmaktadır. Bu sebeple derleme verileri Travis CI REST servis hizmeti üzerinden alınmıştır. Her bir derleme için JSON tipinde bir sonuç alınmıştır. EK-1'de örnek bir derleme verisi gösterilmiştir. Bu veri içinde tek bir derlemeye ait olan eşsiz özellikler mevcuttur. Numara, tarih, kullanıcı adı gibi bu tip bilgiler öngörü üretimi için ve özellik seçimi için kullanılmaz. Ayrıca derlemenin ilişkili olduğu kod değişikliği, kullanıcı, kod deposu gibi diğer veri tipleri ile ilgili bilgiler de doğrudan kullanılmaz. Derlemenin bitiş zamanı özelliği o derlemenin zamanı olarak kabul edilir. Öngörü hedef verileri hazırlanırken (bir sonraki sürümdeki hata sayısı, ortalama sürüm süresi içindeki hata sayısı) derlemenin bitiş zamanı (finished_at) özelliği dikkate alınır.

Terraform ve Vault projeleri hata takip sistemi olarak GitHub altyapısını kullanırlar. Hata takip verileri GitHub REST servisleri üzerinden alınır. EK-2'de örnek bir GitHub hata verisi gösterilmiştir. Sonarqube projesinde ise JIRA hata takip sistemi kullanılmıştır. Hata verileri JIRA REST servisi üzerinden alınmıştır. EK-3'te örnek bir JIRA hata verisi gösterilmiştir. Hata verisindeki en önemli özellik oluşturulma tarihidir (created ve created_at).

Sürüm verileri için Terraform ve Vault projeleri GitHub sistemini kullanmaktadır. Veriler diğer GitHub veriler gibi REST servisi üzerinden alınmaktadır. EK-4'te örnek bir sürüm verisi gösterilmiştir. Sonarqube projesi de sürüm bilgiler için JIRA sistemini kullanmaktadır. JIRA Rest servisleri üzerinden Sonarqube projesi için sürüm bilgileri alınır. Sürüm verisindeki yayınlama zamanı özelliği (published_at, releaseDate), sürüm zaman akış yapısını oluşturulmak için kullanılır. Sürüm zamanı ve hata verisindeki oluşturulma zamanı verileri ile bir sürüm çıktıktan sonra oluşan hata sayısı verisi elde edilir. Ayrıca sürümler arası geçen sürelerin gün cinsinden ortalaması alınır. Bu ortalama,

“ortalama sürüm süresi içinde oluşan hata sayısı” öngörüsü için kullanılır. Yine bu öngörü hedefinde hataların oluşturulma zamanı özelliği dikkate alınmıştır.

Derlemenin hangi kod değişikliği ile ilgili olduğu bilgisinin numarası derleme verisi içinde mevcuttur ve bu numara ile kod değişikliği bilgisi GitHub REST servisleri üzerinden veri ön işleme adımında alınır. Buradan dönen sonuçtaki değişiklik sayısı alanı kullanılır. Bu alan değişen kod satır sayısıdır ve öngörü üretme ile özellik seçme işlemlerinde kullanılır. Derleme verisinin içindeki diğer bir bilgi ise derlemede yapılan iş parçacıkları ile ilgili verilerdir. Bu iş parçalarının da detay verisi veri ön işleme adımında REST servis üzerinden alınır. Bu iş parçalarının detayındaki durum değeri dikkate alınarak, toplan başarılı iş sayısı ve toplan başarısız iş sayısı özellikleri türetilir. Bu özellikler de öngörü üretimi ve özellik seçme adımlarında kullanılır. Bu özelliklerin türetilmesinin sebebi derlemenin genel durumuna ek olarak alt işlerin durumunun kullanılıp, genel öngörü üretimine katkı sağlamaktır.

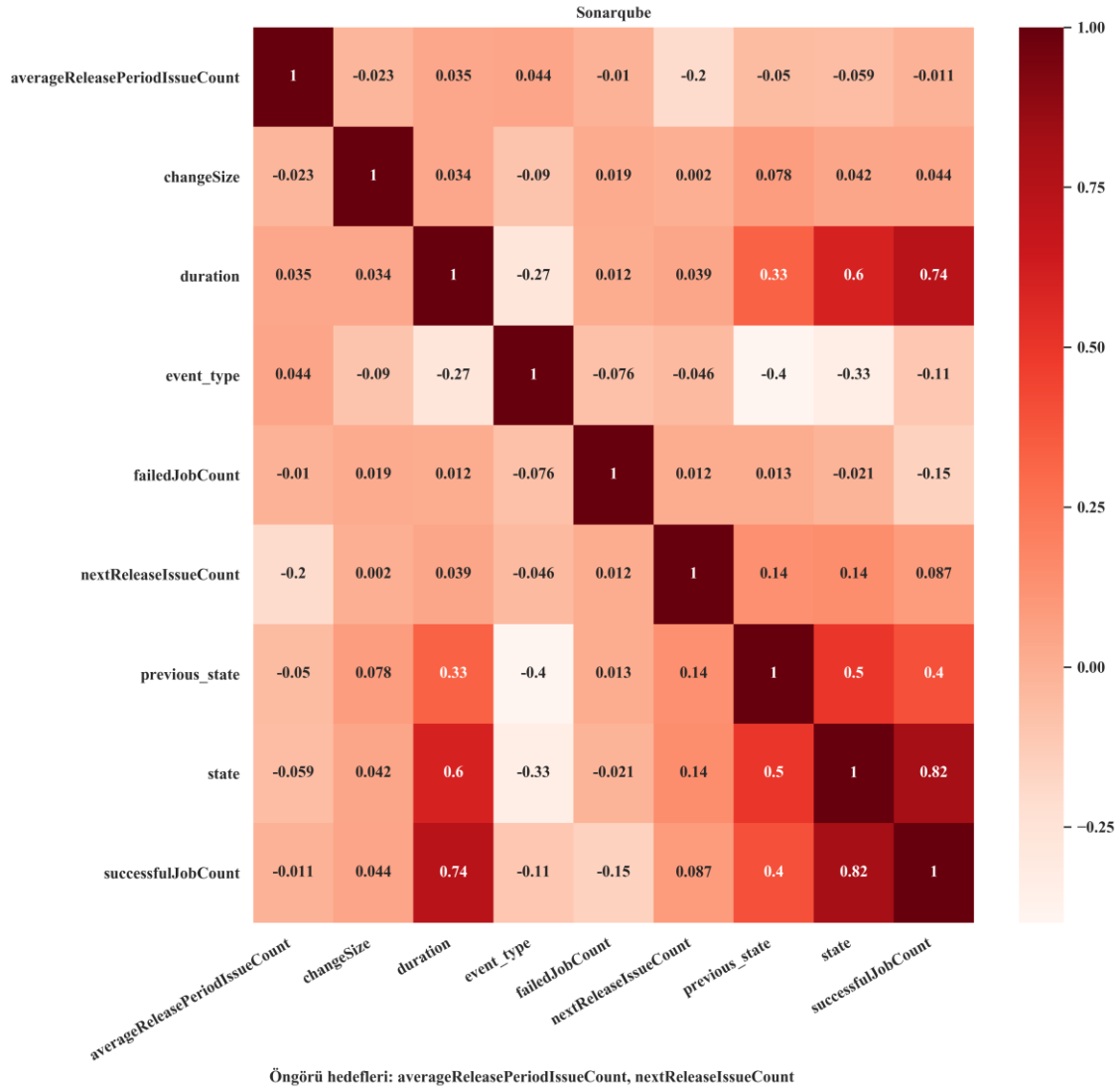
Ön işleme adımında, öngörü hedef verileri hesaplanır ve derleme verisine eklenir (nextReleaseIssueCount, averageReleasePeriodIssueCount). Bu esnada eğer derleme zamanından sonra sürüm yok ise (güncel derlemelerden sonra daha sürüm çıkmamış olabilir) bu derleme verileri dikkate alınmaz ve kullanılmaz. Ayrıca kod değişiklik sayısı, başarılı ve başarısız alt iş sayısı verileri oluşturularak derlemeye eklenir.

İstatistiksel analiz kısmında öngörü hedef verilerinin dağılımı, özellikler ile ilişkisi ve grafiksel gösterimleri üretilir. Öngörü hedeflerinin dağılımı tablosu Tablo 4.1’de gösterilmiştir.

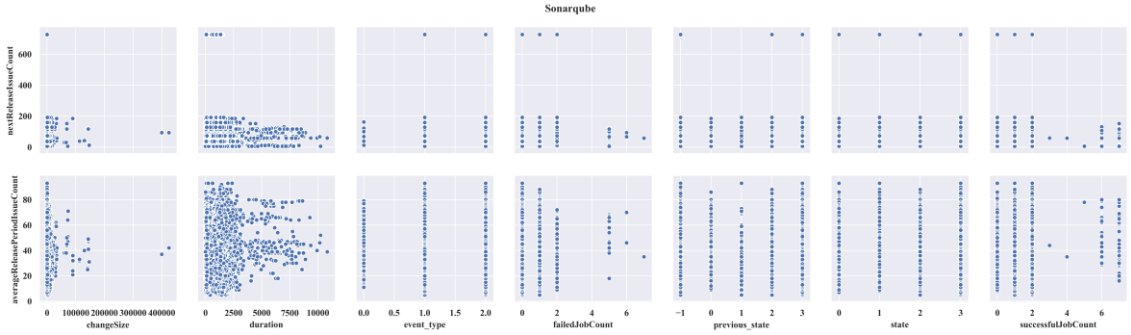
Tablo 4.1. *Öngörü hedeflerinin istatistiksel dağılımı*

Proje	**Sonarqube**		**Terraform**		**Vault**	
Öngörü Hedefi	**Sonraki sürüm hata**	**Ortalama sürüm süresi sonrası hata**	**Sonraki sürüm hata**	**Ortalama sürüm süresi sonrası hata**	**Sonraki sürüm hata**	**Ortalama sürüm süresi sonrası hata**
Toplam	26516	26516	20180	20180	12216	12216
Ortalama	89,05	37,6	215,37	142,83	93,11	17,57
Standart Sapma	64,62	15,91	141,96	54,16	65,37	9,28
Minimum	6	5	10	20	0	0
25%	42	25	79	99	34	11
50%	73	35	184	148	85	16
75%	117	48	339	186	136	24
Maximum	727	93	858	251	261	53

Öngörü hedef değerlerinin ortalama değeri, öngörü oluşturma yöntemlerinin performansında kullanılan ortalama mutlak hata metriğini değerlendirirken bir ölçüt olarak kullanılır. Öngörü hedeflerinin, diğer özellikler ile olan ilişkisi ısı haritası ve dağılım Şekil 4.1 ve Şekil 4.2 ile gözlemlenebilir. Bu grafiklerde sürekli sabit değerde olan iki özellik gösterilmemiştir.



Şekil 4.1. *Sonarqube projesi özellik ilişkileri*



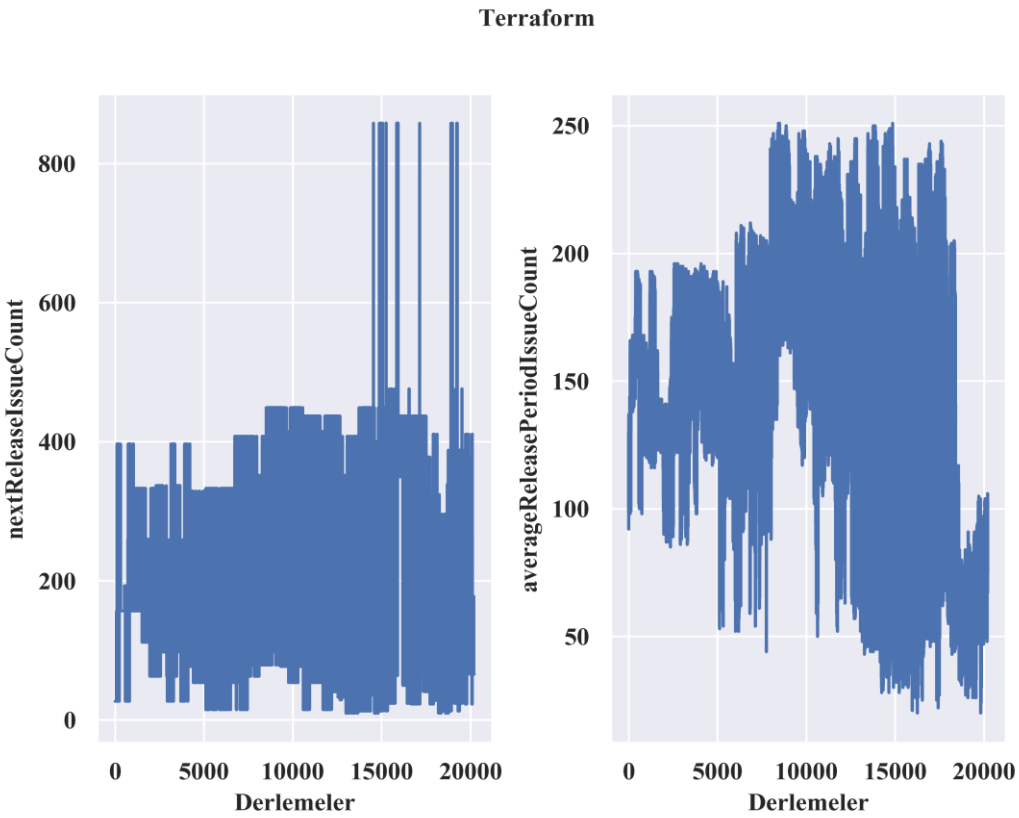
Şekil 4.2. *Sonarqube projesi özellik ilişkileri çoklu grafiği*

Derleme verisindeki özellik ilişkilerine tüm projeler için bakılınca sezgisel olarak "state" (durum), "duration" (süre) ve "changeSize" (kod değişiklik sayısı) özellikleri seçilmiştir. İkinci özellik seçme yönteminde ise tüm özellikler seçilmiştir. Daha önce bahsedilen çeşitli özellik seçme algoritmalarının tümleşik olarak uygulandığı üçüncü yöntemde ise Tablo 4.2'deki sonuçlar oluşmuştur. Bu sonuçlara göre en iyi puanlı dört özellik, üçüncü yöntem için seçilmiştir.

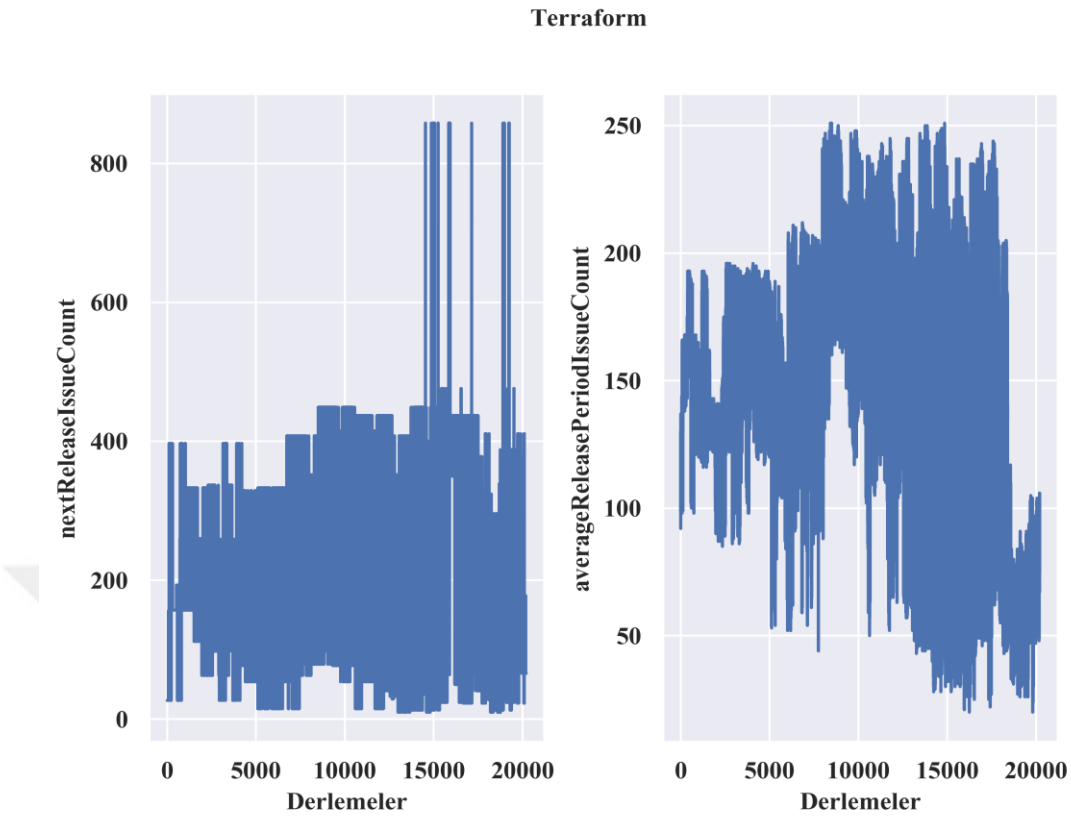
Tablo 4.2. *Özellik seçme algoritmalarının tümleşik sonucu*

nextReleaseIssueCount		**averageReleasePeriodIssueCount**	
Özellik	**Puan**	**Özellik**	**Puan**
successfulJobCount	20	state	22
duration	20	duration	21
previous_state	18	changeSize	19
changeSize	18	successfulJobCount	14
state	16	previous_state	14
event_type	15	event_type	12
failedJobCount	11	failedJobCount	10

Deneylerden önce derleme verisi 25% oranı ile test ve öğrenme verisi olmak üzere rastgele bir biçimde ikiye ayrılmıştır.



Şekil ***4.3***'te "Terraform" projesi için her iki öngörü hedefinin de dağılımı gözükmektedir.



Şekil 4.3. *Terraform projesi için öngörü hedeflerinin dağılımı*

Deneylerdeki kullanılan makine öğrenme yöntemleri ve öngörü performansları Tablo 4.3'te gösterilmiştir. Bu tablolardaki "F1" sütunu birinci özellik seçme yöntemini (sezgisel seçim), "F2" sütünü ikinci özellik seçme yöntemini (tüm özellikler) ve "F3" sütünü üçüncü özellik seçme yöntemini (özellik seçme algoritmaları bütünleşik puanlaması ile) temsil etmektedir. Makine öğrenme algoritmaları çeşitli hiperparametreler ile de denenmiştir.

Tablo 4.3. *"Sonraki sürüm içindeki hata sayısı" hedefi için sonuçlar*

Algoritma	**Decision Tree (50 leaf)**					
Proje	**Sonarqube**			**Terraform**		
Seçilen özellikler	**F1**	**F2**	**F3**	**F1**	**F2**	**F3**
MAE	43,15	43,29	42,97	115,03	112,85	112,52
MSE	4465,96	4705,39	4426,96	17917,46	17469,28	17393,19
RMSE	66,76	68,59	66,53	133,85	132,17	131,88
R2	0,009	-0,04	0,01	0,11	0,13	0,13
MSLE	0,5683	0,5691	0,5649	0,8315	0,7973	0,7933
RMSLE	0,7539	0,7544	**0,7516**	0,9118	0,8929	**0,8927**

Tablo 4.3. (Devam) *"Sonraki sürüm içindeki hata sayısı" hedefi için sonuçlar*

Algoritma	**Linear Regression**					
Proje	**Sonarqube**			**Terraform**		
Seçilen özellikler	**F1**	**F2**	**F3**	**F1**	**F2**	**F3**
MAE	46,42	46,09	46,2	127,75	125,98	127,83
MSE	4388,7	4361,55	4390,73	20169,29	19867,76	20144,53
RMSE	66,24	66,04	66,26	142,01	140,95	141,93
R2	0,02	0,03	0,02	0,001	0,016	0,002
MSLE	0,6147	0,6108	0,6162	0,9031	0,9025	0,9019
RMSLE	0,784	**0,7815**	0,7849	0,9503	**0,95**	0,9497

Algoritma	**Random Forest (Error = mae, estimator = 50)**					
Proje	**Sonarqube**			**Terraform**		
Seçilen özellikler	**F1**	**F2**	**F3**	**F1**	**F2**	**F3**
MAE	42,23	41,49	42,05	113,89	111,5	110,78
MSE	4381,85	4295,46	4437,7	19738,02	19089,71	19020,67
RMSE	66,19	65,54	66,61	140,49	138,16	137,91
R2	0,02	0,04	0,01	0,02	0,05	0,05
MSLE	0,5649	0,5581	0,5674	0,8499	0,8109	0,8045
RMSLE	0,7516	**0,7471**	0,7532	0,9219	0,9	**0,8915**

Algoritma	**XGBoost**					
Proje	**Sonarqube**			**Terraform**		
Seçilen özellikler	**F1**	**F2**	**F3**	**F1**	**F2**	**F3**
MAE	41,9	40,06	41,13	114,513	112,72	112,5
MSE	4009,7	4013,93	3970,64	17766,94	17336,53	17257,31
RMSE	63,32	63,35	63,01	133,29	131,66	131,36
R2	0,1	0,1	0,11	0,12	0,14	0,14
MSLE	0,5538	0,5192	0,5453	0,81	0,7966	0,7948
RMSLE	0,7442	**0,7205**	0,7384	0,9	0,8925	**0,8907**

Algoritma	**Sequential Neural Network**					
Proje	**Sonarqube**			**Terraform**		
Seçilen özellikler	**F1**	**F2**	**F3**	**F1**	**F2**	**F3**
MAE	41,01	39,53	40,46	114,78	110,63	112,11
MSE	4366,47	4346,17	4398,82	20475,78	19965,03	20794,66
RMSE	66,07	65,92	66,32	143,09	141,29	144,2
R2	0,02	0,03	0,02	-0,01	0,01	-0,02
MSLE	0,5513	0,5421	0,5535	0,8762	0,8131	0,8389
RMSLE	0,7424	**0,7362**	0,744	0,936	**0,9017**	0,9159

Tablo 4.3. (Devam) *"Sonraki sürüm içindeki hata sayısı" hedefi için sonuçlar*

Algoritma	**Decision Tree (50 leaf)**		
Proje	**Vault**		
Seçilen özellikler	**F1**	**F2**	**F3**
MAE	45,3	42,53	43,7
MSE	3309,06	3053,54	3189,69
RMSE	57,52	55,25	56,47
R2	0,15	0,22	0,18
MSLE	0,7279	0,6621	0,6847
RMSLE	0,8531	**0,8149**	0,8274

Algoritma	**Linear Regression**		
Proje	**Vault**		
Seçilen özellikler	**F1**	**F2**	**F3**
MAE	50,32	49,86	49,9
MSE	3906,26	3692,78	3797,9
RMSE	62,5	60,76	61,57
R2	0,006	0,06	0,03
MSLE	0,8505	0,7911	0,8162
RMSLE	0,9222	**0,8894**	0,9034

Algoritma	**Random Forest (Error = mae, estimator = 50)**		
Proje	**Vault**		
Seçilen özellikler	**F1**	**F2**	**F3**
MAE	45,12	41,25	42,21
MSE	3521,14	3065,25	3211,14
RMSE	59,33	55,36	56,66
R2	0,1	0,22	0,18
MSLE	0,7009	0,6046	0,6248
RMSLE	0,8372	**0,7815**	0,7904

Algoritma	**XGBoost**		
Proje	**Vault**		
Seçilen özellikler	**F1**	**F2**	**F3**
MAE	44,85	41,97	43,01
MSE	3179,84	2841,04	2983,99
RMSE	56,39	53,3	54,62
R2	0,19	0,27	0,24
MSLE	0,7017	0,6221	0,6498
RMSLE	0,8377	**0,7787**	0,8061

Tablo 4.3. (Devam) *"Sonraki sürüm içindeki hata sayısı" hedefi için sonuçlar*

Algoritma	**Sequential Neural Network**		
Proje	**Vault**		
Seçilen özellikler	**F1**	**F2**	**F3**
MAE	45,16	43,24	42,83
MSE	3812,06	3549,31	3563,59
RMSE	61,74	59,57	59,69
R2	0,03	0,09	0,09
MSLE	0,7441	0,6399	0,6785
RMSLE	0,8626	**0,7999**	0,8237

Tablo 4.4. *"Ortalama sürüm süresi içindeki hata sayısı" hedefi için sonuçlar*

Algoritma	**Decision Tree (50 leaf)**					
Proje	**Sonarqube**			**Terraform**		
Seçilen özellikler	**F1**	**F2**	**F3**	**F1**	**F2**	**F3**
MAE	12,25	12,08	12,1	41,94	40,28	40,87
MSE	234,8	229,13	229,52	2681,01	2592,25	2610,63
RMSE	15,32	15,13	15,15	51,77	50,91	51,09
R2	0,08	0,1	0,1	0,11	0,14	0,14
MSLE	0,1816	0,1764	0,177	0,2015	0,1943	0,1961
RMSLE	0,4261	**0,42**	0,4207	0,4489	**0,4408**	0,4428

Algoritma	**Linear Regression**					
Proje	**Sonarqube**			**Terraform**		
Seçilen özellikler	**F1**	**F2**	**F3**	**F1**	**F2**	**F3**
MAE	13,05	13,04	13,06	46,56	46,43	46,45
MSE	253,6	253,04	253,54	3001,15	2978,79	2991,22
RMSE	15,92	15,9	15,92	54,78	54,57	54,69
R2	0,009	0,01	0,009	0,013	0,02	0,016
MSLE	0,1958	0,1953	0,1957	0,2325	0,2295	0,2315
RMSLE	0,4425	**0,4419**	0,4424	0,4822	**0,4791**	0,4812

Algoritma	**Random Forest (Error = mae, estimator = 50)**					
Proje	**Sonarqube**			**Terraform**		
Seçilen özellikler	**F1**	**F2**	**F3**	**F1**	**F2**	**F3**
MAE	12,45	12,16	12,29	42,43	39,8	40,6
MSE	254,79	245,53	250,23	3113,18	2826,37	3968,48
RMSE	15,96	15,66	15,81	55,79	53,16	54,48
R2	0,004	0,04	0,022	-0,02	0,07	0,02
MSLE	0,1917	0,1877	0,1874	0,2234	0,2043	0,2125
RMSLE	0,4378	**0,4332**	0,4329	0,4726	**0,4502**	0,461

Tablo 4.4. (Devam) *"Ortalama sürüm süresi içindeki hata sayısı" hedefi için sonuçlar*

Algoritma	**XGBoost**					
Proje	**Sonarqube**			**Terraform**		
Seçilen özellikler	**F1**	**F2**	**F3**	**F1**	**F2**	**F3**
MAE	12,07	11,75	11,79	42,12	39,6	40,01
MSE	228,14	222,12	222,78	2702,14	2522,99	2542,25
RMSE	15,1	14,9	14,92	51,98	50,22	50,42
R2	0,1	0,13	0,12	0,11	0,17	0,16
MSLE	0,1756	0,1666	0,1684	0,1964	0,1864	0,1906
RMSLE	0,4191	**0,4082**	0,4103	0,4432	**0,4317**	0,4366

Algoritma	**Sequential Neural Network**					
Proje	**Sonarqube**			**Terraform**		
Seçilen özellikler	**F1**	**F2**	**F3**	**F1**	**F2**	**F3**
MAE	12,18	12,02	12,1	42,49	38,78	39,63
MSE	246,23	238,86	245,68	3252,33	2918,99	3084,22
RMSE	15,69	15,45	15,67	57,02	54,02	55,53
R2	0,03	0,06	0,04	-0,06	0,04	-0,01
MSLE	0,1795	0,1758	0,1774	0,2442	0,2061	0,2144
RMSLE	0,4237	**0,4193**	0,4212	0,4941	**0,454**	0,4631

Algoritma	**Decision Tree (50 leaf)**		
Proje	**Vault**		
Seçilen özellikler	**F1**	**F2**	**F3**
MAE	7,19	7,11	7,18
MSE	82,21	82,22	82,22
RMSE	9,06	9,06	9,06
R2	0,09	0,09	0,09
MSLE	0,3031	0,3019	0,3032
RMSLE	0,5506	**0,5494**	0,5506

Algoritma	**Linear Regression**		
Proje	**Vault**		
Seçilen özellikler	**F1**	**F2**	**F3**
MAE	7,495	7,497	7,489
MSE	87,87	87,77	87,66
RMSE	9,374	9,369	9,362
R2	0,02	0,03	0,031
MSLE	0,33	0,3292	0,329
RMSLE	0,5744	0,5737	**0,5736**

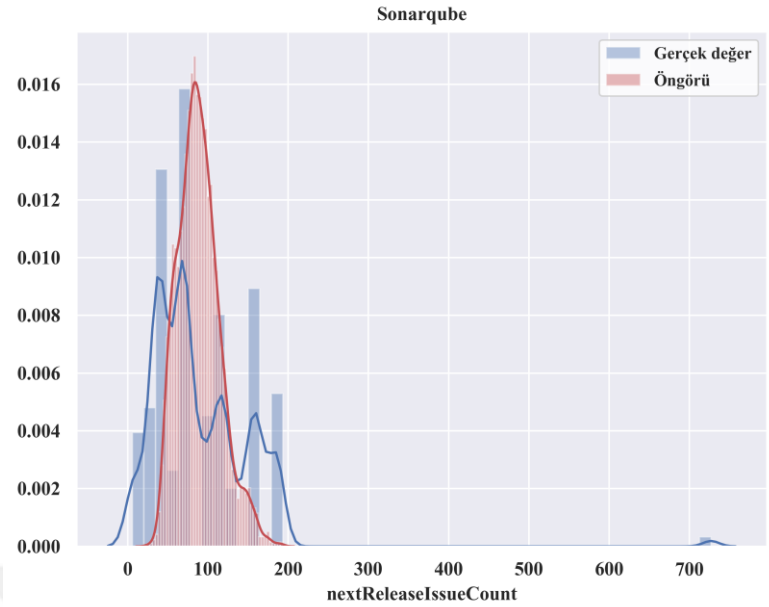
Tablo 4.4. (Devam) *"Ortalama sürüm süresi içindeki hata sayısı" hedefi için sonuçlar*

Algoritma	**Random Forest (Error = mae, estimator = 50)**		
Proje	**Vault**		
Seçilen özellikler	**F1**	**F2**	**F3**
MAE	7,29	6,94	7,27
MSE	87,23	78,78	86,29
RMSE	9,34	8,87	9,28
R2	0,03	0,12	0,04
MSLE	0,3124	0,2897	0,3084
RMSLE	0,559	**0,5382**	0,5553

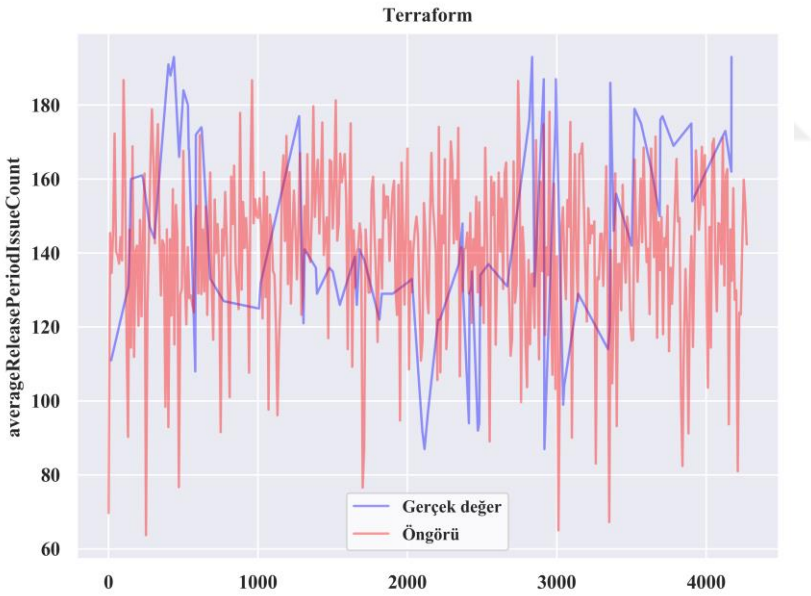
Algoritma	**XGBoost**		
Proje	**Vault**		
Seçilen özellikler	**F1**	**F2**	**F3**
MAE	7,01	6,93	7,01
MSE	77,57	75,75	78,11
RMSE	8,8	8,7	8,83
R2	0,14	0,16	0,13
MSLE	0,2914	0,2862	0,2919
RMSLE	0,5398	**0,535**	0,5403

Algoritma	**Sequential Neural Network**		
Proje	**Vault**		
Seçilen özellikler	**F1**	**F2**	**F3**
MAE	7,45	7,34	7,31
MSE	87,41	85,9	85,52
RMSE	9,34	9,26	9,24
R2	0,03	0,05	0,05
MSLE	0,3298	0,3205	0,3126
RMSLE	0,5743	0,5661	**0,5591**

Deney sonuçlarına göre en yüksek performans ile öngörü üreten algoritma XGBoost olmuştur. Ancak Neural Network algoritması da XGBoost'a yakın bir sonuç üretmiştir. Şekil 4.4 ve Şekil 4.5'teki örnek grafikler ile öngörü hedef verisi ile tahmin edilen değer arasındaki ilişki görülebilir.



Şekil 4.4. *"Sonraki sürüm içindeki hata sayısı" için XGBoost ve F2 sonuçları*



Şekil 4.5. *"Sonraki sürüm içindeki hata sayısı" için XGBoost ve F1 sonuçları*

Bağımlı örneklem t-testi [97] sonuçları Tablo 4.5 ve Tablo 4.6'da gösterilmiştir. Testler için her bir yöntem, proje ve öngörü hedefi için en iyi performans ile en kötü performans gösteren sonuçlar karşılaştırılmıştır. T-test sonuçlarındaki "t" değerinin büyük olması (aynı zamanda "p" değerinin 0.05'ten küçük olması) karşılaştırma arasındaki anlamlı farkın büyük olduğunu gösterir. Buna göre bazı durumlarda sonuçlar

anlamlı performans artışını işaret etse de genelde performans artışı ağırlığı çok yüksek çıkmamıştır. Buna göre kullanılan 5 adet makine öğrenme algoritmasının hepsinin de hata öngörüsü üretmek için yeterli olabileceği sonucuna varılabilir. Algoritmalar arasında öngörü performansı olarak büyük farklar yoktur.

Tablo 4.5. *"Sonraki sürüm içindeki hata sayısı" için t-test sonuçları*

Öngörü Hedefi	**T1**		
Proje	**Sonarqube**		
Özellik seçimi	**F1**	**F2**	**F3**
T-Test	t = 40.7443 p = 0.0	t=26.8527 p=1.2228e-150	t=-0.2061 p= 0.8367
Proje	**Terraform**		
Özellik seçimi	**F1**	**F2**	**F3**
T-Test	t = -0.3886 p = 0.6975	t = -0.7388 p = 0.4600	t = -0.5068 p = 0.6122
Proje	**Vault**		
Özellik seçimi	**F1**	**F2**	**F3**
T-Test	t = 0.9368 p = 0.3489	t = -0.0210 p = 0.9832	t = 0.2958 p = 0.7673

Tablo 4.6. *"Ortalama sürüm süresi hata sayısı" için t-test sonuçları*

Öngörü Hedefi	**T2**		
Proje	**Sonarqube**		
Özellik seçimi	**F1**	**F2**	**F3**
T-Test	t = -0.3786 p = 0.7049	t = 0.0888 p = 0.9292	t = 5.5321 p = 3.2872e-08
Proje	**Terraform**		
Özellik seçimi	**F1**	**F2**	**F3**
T-Test	t = 2.0547 p = 0.0399	t = 1.2435 p = 0.2137	t: 5.1800 p = 2.3206e-07
Proje	**Vault**		
Özellik seçimi	**F1**	**F2**	**F3**
T-Test	t = 2.0173 p = 0.0437	t = 1.7829 p = 0.0747	t = 0.5932 p = 0.553058930209652

Bu deney sonuçlarına göre hata öngörü sistemlerinde sürekli entegrasyon verileri makine öğrenme teknikleri ile hata tahmini üretmek için kullanılabilir. Kullanılacak makine öğrenme algoritması ve özellik seçimi yöntemi, projenin verisinin büyüklüğüne, işlem kaynaklarının kapasitesine göre seçilebilir. Hataların fazla oluşabileceği durumlar

için önceden faydalı bir biçimde tahmin edilebilir. Böylelikle yazılım projesinin kalitesi için bir iyileştirme sağlanır.

5. SONUÇ VE ÖNERİLER

Yazılım projelerinde oluşan hatalar, istenmeyen sonuçlar ve maliyete sebep olmaktadır. Yazılım hatalarının önceden tahmin edilip gerekli önlemlerin alınması hataların etkileri azaltır. Bu sebeple hata öngörü sistemleri geliştirilip ve yazılım geliştirme depoları üzerindeki veri madenciliği çalışmaları yapılmaktadır. Bu alanda yapılan daha önceki çalışmalar başarılı ve etkili olmuştur. Bu çalışmada yazılım geliştirme yaşam döngüsü esnasında kullanılan bir araç olan sürekli entegrasyon derleme sistemlerinin oluşturduğu verilerin hata öngörüsü üretmedeki faydası üzerine çalışılmıştır.

Yöntem kısaca şu şekilde özetlenebilir; bir yazılım projesinin sürekli entegrasyon derleme, hata ve sürüm verileri toplanarak geçmişteki hataları kullanarak makine öğrenme modeli oluşturulup, bir sonraki derlemeden sonra oluşacak hata sayısı tahmin edilir. Çalışmada iki adet öngörü hedefi seçilmiştir. Bir tanesi sürekli entegrasyon derlemesinden sonraki çıkacak olan sürümde oluşacak hata sayısı, diğer hedef ise tüm sürümlerin ortalama süresi hesaplanıp, derlemeden bu ortalama süre kadar geçecek zaman içerisinde oluşan hata sayısıdır. Veriler toplandıktan sonra boş özellikler temizlenir ve öngörü hedef değerleri hesaplanır. Daha sonra özellik seçme adımı gelir ve bu adımda üç farklı yöntem izlenir. İlk yöntemde verinin bir ön analizi yapılır ve sezgisel olarak birkaç özellik seçilir. İkinci yöntemde ise tüm özellikler seçilir. Son özellik seçme yönteminde ise çeşitli özellik seçme algoritmalarının sonuçları tümleşik bir biçimde kullanılır. Öngörü üretici modelini oluşturmak için beş farklı makine öğrenme algoritması kullanılmıştır. Tüm öngörü hedefleri ve özellik seçme yöntemleri için üç adet açık kaynaklı yazılım projesi üzerinde deneyler yapılmıştır. Deney sonuçları çeşitli makine öğrenme değerlendirme metrikleri ile değerlendirilmiştir. T-test ve grafikler ile sonuçlar daha detaylı analiz edilmiştir.

Çalışma sonunda sürekli entegrasyon verisinin, hata ve sürüm bilgileri ile birlikte hata öngörüsü üretmek için kullanılabileceği, yazılım kalitesi üzerine faydalı olacağı sonucuna ulaşılmıştır. Hataların daha fazla olabileceği önceden fark edilip, yazılım ekipleri tarafından gerekli görülen önleyici adımlar atılabilir.

Bu çalışma sürekli entegrasyon verilerinin yazılım deposu veri madenciliği için kullanılabileceğini göstermiştir. Çalışmada oluşturulan modeller, kendi projesi dışındaki diğer projeler için kullanılabilir. Böylelikle farklı projelerdeki karakteristik özellikler oraya çıkabilir. Bir çok mevcut hata öngörü sistemi çalışmasında kategorik öngörü hedefi

kullanılmıştır (Hata üretebilir, sorunsuz, vs. gibi). Bu çalışmada kullanılan hata sayısı sayısal hedefinin yanı sıra, derlemenin hatalı olup olmadığı gelecekte çıkan hata durumlarına göre sınıflandırılabilir. Bu sınıflandırmaya göre bir sonraki derlemenin sınıfı tahmin edilebilir. Sınıflandırma hedefi ile ayrıca daha farklı makine öğrenme ve özellik seçme algoritmaları kullanılabilir. Başka bir yöntem olarak ise anomali tespiti olabilir. Sürekli entegrasyon derleme verisindeki anomaliler ve normal olmayan değişim durumları tespit edilerek hata öngörüsü yapılabilir. Sürekli entegrasyon derlemelerinin büyük çoğunluğu derleme işleri esnasında büyük oranda metin formatında olay kaydı üretebilmektedir. Bu kayıtlar metin analizi yöntemleri kullanılarak hatalı bir duruma sebep olup olmayacağının analizi yapılabilir. Sürekli entegrasyon derleme verilerinde özellikler bütüncül olarak tasarlanmış geçmiş hata öngörü çalışmalarına metrik olarak eklenebilir ve genel öngörü sonucuna fayda sağlayabilir.

KAYNAKÇA

[1] McDonald, M., R. Musson, and R. Smith. (2007). *The practical guide to defect prevention.*). Redmond, WA, USA: Microsoft Press.

[2] Kagdi, H., M.L. Collard, and J.I. Maletic. (2007). A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of software maintenance and evolution: Research and practice*. 19 (2), 77-131.

[3] Spinellis, D. (2005). Version Control Systems. *IEEE Software*. 22 (5), 108-109.

[4] Fowler, M. and M. Foemmel. (2006). Continuous integration. *(Thought-Works) http:// www. thoughtworks. com/Continuous Integration. pdf*. 122, 14.

[5] Dromey, R.G. (1996). Cornering the Chimera [software quality]. *IEEE Software*. 13 (1), 33-43.

[6] Boehm, B.W., J.R. Brown, and M. Lipow. (1976). Quantitative evaluation of software quality. *Proceedings of the 2nd international conference on Software engineering*, San Francisco, California, USA:IEEE Computer Society Press, s. 592-605.

[7] Kan, S.H. (2002). *Metrics and models in software quality engineering.* Addison-Wesley Longman Publishing Co., Inc.

[8] Chidamber, S.R. and C.F. Kemerer. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*. 20 (6), 476-493.

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

[10] McCabe, T.J. (1976). A Complexity Measure. *IEEE Transactions on Software Engineering*. SE-2 (4), 308-320.

[11] Zimmermann, T., N. Nagappan, and A. Zeller. (2008). Predicting Bugs from History. T. Mens and S. Demeyer, (Editörler), *Software Evolution* içinde (s. 69-88). Berlin, Heidelberg: Springer Berlin Heidelberg.

[12] Zimmermann, T. and N. Nagappan. (2008). Predicting defects using network analysis on dependency graphs. *2008 ACM/IEEE 30th International Conference on Software Engineering*, Leipzig, Germany:ACM Press, s. 531-540.

[13] He, P., et al. (2013). Using Software Dependency to Bug Prediction. *Mathematical Problems in Engineering*. 2013 (10), 1-12.

[14] Prateek, S., A. Pasala, and L.M. Aracena. (2013). Evaluating Performance of Network Metrics for Bug Prediction in Software. *2013 20th Asia-Pacific Software Engineering Conference (APSEC),* Bangkok, Thailand:IEEE, s. 124-131.

[15] Ying, A.T.T., et al. (2004). Predicting source code changes by mining change history. *IEEE transactions on Software Engineering*. 30 (9), 574-586.

[16] Schwanke, R., L. Xiao, and Y. Cai. (2013). Measuring architecture quality by structure plus history analysis. *2013 35th International Conference on Software Engineering (ICSE),* San Francisco, CA, USA:IEEE, s. 891-900.

[17] Hassan, A.E. and R.C. Holt. (2005). The top ten list: dynamic fault prediction. *21st IEEE International Conference on Software Maintenance (ICSM'05),* Budapest, Hungary, Hungary:IEEE, s. 263-272.

[18] Munson, J.C. and S.G. Elbaum. (1998). Code churn: a measure for estimating the impact of code change. *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272),* Bethesda, MD, USA, USA:IEEE Comput. Soc, s. 24-31.

[19] Nagappan, N. and T. Ball. (2005). Use of relative code churn measures to predict system defect density. *Proceedings of the 27th international conference on Software engineering,* Saint Louis, MO, USA, USA:ACM, s. 284-292.

[20] Giger, E., M. Pinzger, and H.C. Gall. (2011). Comparing fine-grained source code changes and code churn for bug prediction. *Proceedings of the 8th Working Conference on Mining Software Repositories,* Waikiki, Honolulu, HI, USA:ACM Press, s. 83-92.

[21] Kim, S., et al. (2007). Predicting Faults from Cached History. *29th International Conference on Software Engineering (ICSE'07),* Minneapolis, MN, USA:IEEE, s. 489-498.

[22] Rahman, F., et al. (2011). BugCache for inspections: hit or miss? *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering,* ACM, s. 322-331.

[23] Hassan, A.E. (2009). Predicting faults using the complexity of code changes. *Proceedings of the 31st International Conference on Software Engineering,* Vancouver, BC, Canada:IEEE Computer Society, s. 78-88.

[24] Giger, E., et al. (2012). Method-level bug prediction. *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, Lund, Sweden:ACM Press, s. 171-180.

[25] Hata, H., O. Mizuno, and T. Kikuno. (2012). Bug prediction based on fine-grained module histories. *2012 34th International Conference on Software Engineering (ICSE)*, Zurich, Switzerland:IEEE, s. 200-210.

[26] Livshits, B. and T. Zimmermann. (2005). DynaMine: finding common error patterns by mining software revision histories. *ACM SIGSOFT Software Engineering Notes*, ACM Press, s. 296-305.

[27] Zimmermann, T., et al. (2005). Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*. 31 (6), 429-445.

[28] D'Ambros, M., M. Lanza, and R. Robbes. (2010). An extensive comparison of bug prediction approaches. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, Cape Town, South Africa:IEEE, s. 31-41.

[29] Williams, C.C. and J.K. Hollingsworth. (2005). Automatic mining of source code repositories to improve bug finding techniques. *IEEE Transactions on Software Engineering*. 31 (6), 466-480.

[30] Shin, Y., et al. (2011). Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE Transactions on Software Engineering*. 37 (6), 772-787.

[31] Dimitrov, M. and H. Zhou. (2009). Anomaly-based Bug Prediction, Isolation, and Validation: An Automated Approach for Software Debugging. *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, Washington, DC, USA:ACM Press, s. 61-72.

[32] Wu, W., et al. (2010). Time series analysis for bug number prediction. *The 2nd International Conference on Software Engineering and Data Mining*, Chengdu, China, s. 589-596.

[33] Pati, J. and K.K. Shukla. (2014). A comparison of ARIMA, neural network and a hybrid technique for Debian bug number prediction. *2014 International Conference on Computer and Communication Technology (ICCCT)*, Allahabad, India:IEEE, s. 47-53.

[34] Shivaji, S., et al. (2009). Reducing Features to Improve Bug Prediction. *2009 IEEE/ACM International Conference on Automated Software Engineering*, Auckland, New Zealand:IEEE, s. 600-604.

[35] Osman, H., M. Ghafari, and O. Nierstrasz. (2017). Automatic feature selection by regularization to improve bug prediction accuracy. *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, Klagenfurt, Austria:IEEE, s. 27-32.

[36] Shivaji, S., et al. (2013). Reducing Features to Improve Code Change-Based Bug Prediction. *IEEE Transactions on Software Engineering*. 39 (4), 552-569.

[37] Xu, Z., et al. (2016). The Impact of Feature Selection on Defect Prediction Performance: An Empirical Comparison. *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, Ottawa, ON, Canada:IEEE, s. 309-320.

[38] Kondo, M., et al. (2019). The impact of feature reduction techniques on defect prediction models. *Empirical Software Engineering*, 1-39.

[39] Puranik, S., P. Deshpande, and K. Chandrasekaran. (2016). A Novel Machine Learning Approach for Bug Prediction. *Procedia Computer Science*. 93, 924-930.

[40] Hassan, A. and K. Zhang. (2006). Using Decision Trees to Predict the Certification Result of a Build. *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, Tokyo, Japan:IEEE, s. 189-198.

[41] Stoerzer, M., et al. (2006). Finding failure-inducing changes in java programs using change classification. *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering - SIGSOFT '06/FSE-14*, ACM Press, s. 57-68.

[42] Ihara, A., et al. (2012). An Investigation on Software Bug-Fix Prediction for Open Source Software Projects -- A Case Study on the Eclipse Project. *2012 19th Asia-Pacific Software Engineering Conference*, Hong Kong, China:IEEE, s. 112-119.

[43] Yang, C.-Z., et al. (2014). Improving severity prediction on software bug reports using quality indicators. *2014 IEEE 5th International Conference on Software Engineering and Service Science*, Beijing, China:IEEE, s. 216-219.

[44] Xin, X., et al. (2013). A Comparative Study of Supervised Learning Algorithms for Re-opened Bug Prediction. *2013 17th European Conference on Software Maintenance and Reengineering* Genova, Italy:IEEE, s. 331-334.

[45] Qin, F., et al. (2015). Cross-Project Aging Related Bug Prediction. *2015 IEEE International Conference on Software Quality, Reliability and Security* Vancouver, BC, Canada:IEEE, s. 43-48.

[46] Xia, X., et al. (2014). Automated Configuration Bug Report Prediction Using Text Mining. *2014 IEEE 38th Annual Computer Software and Applications Conference*, Vasteras, Sweden:IEEE, s. 107-116.

[47] Lewis, C., et al. (2013). Does bug prediction support human developers? Findings from a Google case study. *2013 35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, USA:IEEE, s. 372-381.

[48] Di Nucci, D., et al. (2018). A Developer Centered Bug Prediction Model. *IEEE Transactions on Software Engineering*. 44 (1), 5-24.

[49] Jú, M.C., M. Mendonça, and F. Rodrigues. (2009). Mining Software Change History in an Industrial Environment. *2009 XXIII Brazilian Symposium on Software Engineering*, Fortaleza, Ceara, Brazil:IEEE, s. 54-61.

[50] Tóth, Z., P. Gyimesi, and R. Ferenc. (2016). A Public Bug Database of GitHub Projects and Its Application in Bug Prediction. *Computational Science and Its Applications -- ICCSA 2016*, Cham:Springer International Publishing, s. 625-638.

[51] Menzies, T., et al. (2012). Local versus Global Lessons for Defect Prediction and Effort Estimation. *IEEE Transactions on software engineering*. 39 (6), 822-834.

[52] Zhang, F., et al. (2014). Towards building a universal defect prediction model. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM Press, s. 182-191.

[53] Hassan, A.E. (2008). The road ahead for Mining Software Repositories. *2008 Frontiers of Software Maintenance*, Beijing, China:IEEE, s. 48-57.

[54] Lessmann, S., et al. (2008). Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Transactions on Software Engineering*. 34 (4), 485-496.

[55] Moser, R., W. Pedrycz, and G. Succi. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. *2008 ACM/IEEE 30th International Conference on Software Engineering*, Leipzig, Germany:ACM Press, s. 181-190.

[56] Osman, H., et al. (2017). An Extensive Analysis of Efficient Bug Prediction Configurations. *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, ACM Press, s. 107-116.

[57] Breiman, L. (2001). Random Forests. *Machine Learning*. 45 (1), 5-32.

[58] http-1: https://travis-ci.org/ (Erişim tarihi: 21.05.2019)

[59] http-2: https://www.atlassian.com/software/jira (Erişim tarihi: 21.05.2019)

[60] http-3: https://github.com/ (Erişim tarihi: 21.05.2019)

[61] Mikowski, M. and J. Powell. (2013). *Single page web applications: JavaScript end-to-end.* Manning Publications Co.

[62] Fain, Y. and A. Moiseev. (2016). *Angular 2 Development with TypeScript.*). Greenwich, CT, USA: Manning Publications Co.

[63] Walls, C. (2016). *Spring Boot in action.* Manning Publications.

[64] Gormley, C. and Z. Tong. (2015). *Elasticsearch: The definitive guide: A distributed real-time search and analytics engine.* " O'Reilly Media, Inc.".

[65] Anderson, C. (2015). Docker [software engineering]. *IEEE Software*. 32 (3), 102-c3.

[66] Gupta, Y. (2015). *Kibana Essentials.* Packt Publishing Ltd.

[67] Oliphant, T.E. (2007). Python for scientific computing. *Computing in Science & Engineering*. 9 (3), 10-20.

[68] http-4: https://www.anaconda.com/ (Erişim tarihi: 21.05.2019)

[69] http-5: https://www.jetbrains.com/pycharm/ (Erişim tarihi: 21.05.2019)

[70] McKinney, W. (2012). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython.* " O'Reilly Media, Inc.".

[71] http-6: https://www.kevinsheppard.com/images/b/b3/Python_introduction-2016.pdf (Erişim tarihi: 20.05.2019)

[72] Guyon, I. and A. Elisseeff. (2003). An introduction to variable and feature selection. *Journal of machine learning research*. 3 (Mar), 1157-1182.

[73] Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*. 12 (Oct), 2825-2830.

[74] Gulli, A. and S. Pal. (2017). *Deep Learning with Keras.* Packt Publishing Ltd.

[75] http-7: https://scikit-learn.org/stable/modules/feature_selection.html#variance-threshold (Erişim tarihi: 21.05.2019)

[76] Lancaster, H.O. and E. Seneta. (2005). Chi-square distribution. *Encyclopedia of biostatistics*. 2.

[77] Girden, E.R. (1992). *ANOVA: Repeated measures.* Sage.

[78] Myers, R.H. and R.H. Myers. (1990). *Classical and modern regression with applications.* Duxbury press Belmont, CA.

[79] Van Dijck, G. and M.M. Van Hulle. (2006). Speeding up the wrapper feature subset selection in regression by mutual information relevance and redundancy analysis. *International Conference on Artificial Neural Networks*, Springer, s. 31-40.

[80] Granitto, P.M., et al. (2006). Recursive feature elimination with random forest for PTR-MS analysis of agroindustrial products. *Chemometrics and Intelligent Laboratory Systems*. 83 (2), 83-90.

[81] Hans, C. (2009). Bayesian lasso regression. *Biometrika*. 96 (4), 835-845.

[82] Saunders, C., A. Gammerman, and V. Vovk. (1998). Ridge Regression Learning Algorithm in Dual Variables. *Proceedings of the Fifteenth International Conference on Machine Learning*, San Francisco, CA, USA, s. 515-521.

[83] Seber, G.A. and A.J. Lee. (2012). *Linear regression analysis.* John Wiley & Sons.

[84] http-8: https://scikitlearn.org/stable/modules/generated/sklearn.ensemble-.ExtraTreesRegressor.html (Erişim tarihi: 21.05.2019)

[85] MacKay, D.J. (1999). Comparison of approximate methods for handling hyperparameters. *Neural computation*. 11 (5), 1035-1068.

[86] Xu, M., et al. (2005). Decision tree regression for soft classification of remote sensing data. *Remote Sensing of Environment*. 97 (3), 322-336.

[87] Chen, T. and C. Guestrin. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California, USA:ACM, s. 785-794.

[88] http-9: https://keras.io/getting-started/sequential-model-guide/ (Erişim tarihi: 21.05.2019)

[89] Park, J. and D.W. Edington. (2001). A sequential neural network model for diabetes prediction. *Artificial intelligence in medicine*. 23 (3), 277-293.

[90] Xia, Y., et al. (2017). A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring. *Expert Systems with Applications*. 78, 225-241.

[91] Bergstra, J. and Y. Bengio. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*. 13 (Feb), 281-305.

[92] http-10: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (Erişim tarihi: 21.05.2019)

[93] Kingma, D.P. and J. Ba. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*. 2014 (12).

[94] http-11: https://github.com/SonarSource/sonarqube (Erişim tarihi: 21.05.2019)

[95] http-12: https://github.com/hashicorp/terraform (Erişim tarihi: 21.05.2019)

[96] http-13: https://github.com/hashicorp/vault (Erişim tarihi: 21.05.2019)

[97] Keselman, H., et al. (2004). The new and improved two-sample t test. *Psychological Science*. 15 (1), 47-51.

EK-1 ÖRNEK TRAVIS DERLEME VERİSİ


```
{
  "@type": "build",
  "@href": "/build/145511826",
  "@representation": "standard",
  "@permissions": {
    "read": true,
    "cancel": false,
    "restart": false
  },
  "id": 145511826,
  "number": "15582",
  "state": "passed",
  "duration": 400,
  "event_type": "pull_request",
  "previous_state": "passed",
  "pull_request_title": "Increased lambda event mapping creation timeout",
  "pull_request_number": 7657,
  "started_at": "2016-07-18T11:26:18Z",
  "finished_at": "2016-07-18T11:32:58Z",
  "private": false,
  "repository": {
    "@href": "/repo/2741330"
  },
  "branch": {
    "@href": "/repo/2741330/branch/master",
  },
  "tag": null,
  "commit": {
    "id": 41281681,
    "sha": "c2f2ab5bc57cb6ec73d183da0e40fb78f397de99",
    "ref": "refs/pull/7657/merge",
```

"message": "Increased lambda event mapping creation timeout\n\nIncreased the retry timeout from 1 to 5 minutes due to the time for IAM permission propagation",
"compare_url": "https://github.com/hashicorp/terraform/pull/7657",
"committed_at": "2016-07-18T11:25:15Z"
},
"jobs": [
{
"@href": "/job/145511827"
}
],
"stages": [],
"created_by": {
"@href": "/user/281647"
},
"updated_at": "2019-04-11T10:25:34.192Z"
}

EK-2 ÖRNEK GITHUB HATA VERİSİ


```
{
    "url": "https://api.github.com/repos/hashicorp/vault/issues/6776",
    "id": 447249201,
    "node_id": "MDExOlB1bGxSZXF1ZXN0MjgxMjg0NTkz",
    "number": 6776,
    "title": "Audit listing with format json returns json, not a string",
    "user": {
      "id": 28627
    },
    "labels": [],
    "state": "open",
    "locked": false,
    "assignee": null,
    "assignees": [],
    "milestone": {
      "url": "https://api.github.com/repos/hashicorp/vault/milestones/52",
      "id": 4269383
    },
    "comments": 0,
    "created_at": "2019-05-22T17:07:12Z",
    "updated_at": "2019-05-23T05:22:48Z",
    "closed_at": null,
    "author_association": "MEMBER",
    "pull_request": {
      "url": "https://api.github.com/repos/hashicorp/vault/pulls/6776"
    },
    "body": "Fixes #6775"
  }
```

EK-3 ÖRNEK JIRA HATA VERİSİ


```
{
    "expand":
"operations,versionedRepresentations,editmeta,changelog,renderedFields",
    "id": "84899",
    "self": "https://jira.sonarsource.com/rest/api/2/issue/84899",
    "key": "SONAR-12139",
    "fields": {
      "issuetype": {
        "self": "https://jira.sonarsource.com/rest/api/2/issuetype/1",
        "id": "1",
        "description": "A problem which impairs or prevents the functions of
the product.",
        "name": "Bug",
        "subtask": false,
        "avatarId": 10383
      },
      "project": {
        "self": "https://jira.sonarsource.com/rest/api/2/project/10930",
        "id": "10930",
        "key": "SONAR",
        "name": "SonarQube"
      },
      "fixVersions": [
        {
          "self": "https://jira.sonarsource.com/rest/api/2/version/14939",
          "id": "14939",
          "description": "",
          "name": "7.8",
          "archived": false,
          "released": false,
          "releaseDate": "2019-06-03"
        }
```

```
],
"resolution": null,
"created": "2019-05-23T09:48:47.000+0200",
"priority": {
  "name": "Major",
  "id": "3"
},
"versions": [],
"issuelinks": [],
"assignee": null,
"updated": "2019-05-23T09:48:47.000+0200",
"status": {
  "self": "https://jira.sonarsource.com/rest/api/2/status/1",
  "description": "The issue is open and ready for the assignee to start work on it.",
  "name": "Open",
  "id": "1"
},
"components": [
  {
    "self": "https://jira.sonarsource.com/rest/api/2/component/11366",
    "id": "11366",
    "name": "Web"
  }
],
"description": ”https://aws1.discourse-cdn.com/sonarsource/uploads/default/original/2X/a/a1373a48c26d815b2c2fdae7f0a9e2d3443a36ce.jpeg|width=262,height=215!",
"summary": "glitch on Administration > Projects Management",
"creator": {
  "name": "jeremy.davis",
  "key": "jeremy.davis",
  "emailAddress": "jeremy.davis@sonarsource.com",
```

```
            "displayName": "Jeremy Davis",
            "active": true,
            "timeZone": "Europe/Paris"
        },
        "subtasks": [],
        "reporter": {
            "name": "jeremy.davis",
            "key": "jeremy.davis",
            "emailAddress": "jeremy.davis@sonarsource.com",
            "displayName": "Jeremy Davis",
            "active": true,
            "timeZone": "Europe/Paris"
        },
        "duedate": null
    }
}
```

EK-4 ÖRNEK GITHUB SÜRÜM VERİSİ


```
{
  "id": 17407736,
  "node_id": "MDc6UmVsZWFzZTE3NDA3NzM2",
  "tag_name": "v0.11.14",
  "target_commitish": "master",
  "name": "v0.11.14",
  "draft": false,
  "author": {
    "login": "apparentlymart",
    "id": 20180,
  },
  "prerelease": false,
  "created_at": "2019-05-16T20:41:32Z",
  "published_at": "2019-05-16T20:49:01Z",
  "assets": [],
  "tarball_url":
"https://api.github.com/repos/hashicorp/terraform/tarball/v0.11.14",
  "zipball_url":
"https://api.github.com/repos/hashicorp/terraform/zipball/v0.11.14",
  "body": "NEW FEATURES:\r\n\r\n* `terraform 0.12checklist` command detects and reports on some preparation steps that will make a subsequent Terraform 0.12 upgrade smoother. ([#21241](https://github.com/hashicorp/terraform/issues/21241))\r\n\r\nIMPROVEMENTS:\r\n\r\n* provider/terraform: The `terraform_remote_state` data source is now able to read outputs from a state snapshot created by Terraform 0.12, to provide more flexibility when upgrading individual configurations to Terraform 0.12 in a decomposed environment. ([#21226](https://github.com/hashicorp/terraform/issues/21226))\r\n* backend/s3: Support DynamoDB, IAM, and STS endpoint configurations ([#20659](https://github.com/hashicorp/terraform/issues/20659))\r\n* backend/s3: Support for AWS regions `eu-north-1` and `us-gov-east-1` ([#20659](https://github.com/hashicorp/terraform/issues/20659))\r\n* backend/s3: Enhance retry logic and provide `max_retries` configuration for retry attempts
```

([#20659](https://github.com/hashicorp/terraform/issues/20659))\r\n* backend/s3: Enhance S3 `NoSuchBucket` error to include additional information ([#20659](https://github.com/hashicorp/terraform/issues/20659))\r\n* backend/s3: Remove unused EC2 platform and AWS Account ID lookup, and deprecate equivalent `skip_get_ec2_platforms` and `skip_requesting_account_id` arguments ([#20659](https://github.com/hashicorp/terraform/issues/20659))\r\n* backend/remote: Do not unlock a workspace after a failed state upload ([#21148](https://github.com/hashicorp/terraform/issues/21148))\r\n\r\nBUG FIXES:\r\n\r\n* backend/remote: Ensure variables are loaded correctly when using `terraform console` ([#20858](https://github.com/hashicorp/terraform/issues/20858))\r\n* backend/remote: Make sure workspaces are correctly uploaded ([#20953](https://github.com/hashicorp/terraform/issues/20953))\r\n* backend/remote: Fix panic when loading cached configuration with 0.12 backend schema ([#21199](https://github.com/hashicorp/terraform/issues/21199))\r\n* core: Fix a potential crash when loading module manifests with Windows paths ([#20812](https://github.com/hashicorp/terraform/issues/20812))\r\n* core: Make sure UIInput keeps working after being canceled ([#21140](https://github.com/hashicorp/terraform/issues/21140))\r\n* core: Always try to select a workspace after initialization ([#21230](https://github.com/hashicorp/terraform/issues/21230))"
}

EK-5 ÖRNEK JIRA SÜRÜM VERİSİ


```
{
  "self": "https://jira.sonarsource.com/rest/api/2/version/14467",
  "id": "14467",
  "description": "LTS Bug Fixes",
  "name": "6.7.5",
  "archived": false,
  "released": true,
  "releaseDate": "2018-08-06",
  "userReleaseDate": "06/Aug/18",
  "projectId": 10930
}
```

ÖZGEÇMİŞ

Adı Soyadı : İbrahim ESEN

Yabancı Dil : İngilizce

Doğum Yeri ve Yılı : Eskişehir / 1987

E-Posta : ibrahmesen@gmail.com

Eğitim ve Mesleki Geçmişi:

- 2014 - … , Yazılım Mimarı, Anadolu Üniversitesi BAUM
- 2011 - 2014, Yazılım Mühendisi, Anadolu Üniversitesi BAUM
- 2008 - 2009, Yazılım Mühendisi, Karakullukçu Danışmanlık
- 2005 - 2010, Bilgisayar Mühendisliği, Anadolu Üniversitesi
- 2001 - 2005, Eskişehir Anadolu Lisesi