Kaynak Kod Değişikliklerinin Denetimlerden Geçip Geçmeyeceğinin Tahmin Edilmesi

Zeki Mazan, Çağdaş Evren Gerede

TOBB Ekonomi ve Teknoloji Üniversitesi, Ankara, Türkiye
Web Page: http://cegerede.etu.edu.tr

Özet. Yazılım projelerinde, kaynak kodda yapılmak istenen değişiklik önerilerini bir denetim sürecinden geçirmek ortaya çıkan yazılımların hem kalitesine hem de ömrüne olumlu etkide bulunmaktadır. Kod denetim süreçleri zaman zaman uzun süreler almakta ve hem yazılım geliştirme maliyetine hem de çalışanların iş memnuniyetine olumsuz olarak yansımaktadır. Bir kod değişikliği önerisini denetleyen denetçilerin denetleme sürecinde ne tür yanıtlar vereceğini ve önerinin ne gibi revizyon taleplerine maruz kalacağını önceden tahmin edebilen mekanizmaları oluşturabilmek bu problemlerin ortaya çıkacağı durumları azaltacaktır. Zira öneriyi yapan geliştirici bu mekanizmalar sayesinde değişiklik önerisini denetlemeyi başlatmadan daha fazla olgunlaştırabilir ve sonucunda da öneriyi daha hızlı şekilde denetlemeden başarıyla geçebilecek hale getirebilir. Bu çalışmada bu amaca doğru ilk adım olarak geçmiş denetleme verilerini kullanarak bir değişiklik önerisinin denetlenmesi sonucunun ne olacağını ve herhangi bir revizyon talebine maruz kalıp kalmayacağını tahmin eden çözümler geliştirmeye çalıştık.

Anahtar Kelimeler: Kaynak kod denetimi, Gerrit, Denetim onayı tahmini

Automatically Determining Whether a Software Change Request Will Be Approved

Zeki Mazan, Çağdaş Evren Gerede

TOBB University of Economics and Technology, Ankara, Turkey
Web Sayfası: http://cegerede.etu.edu.tr

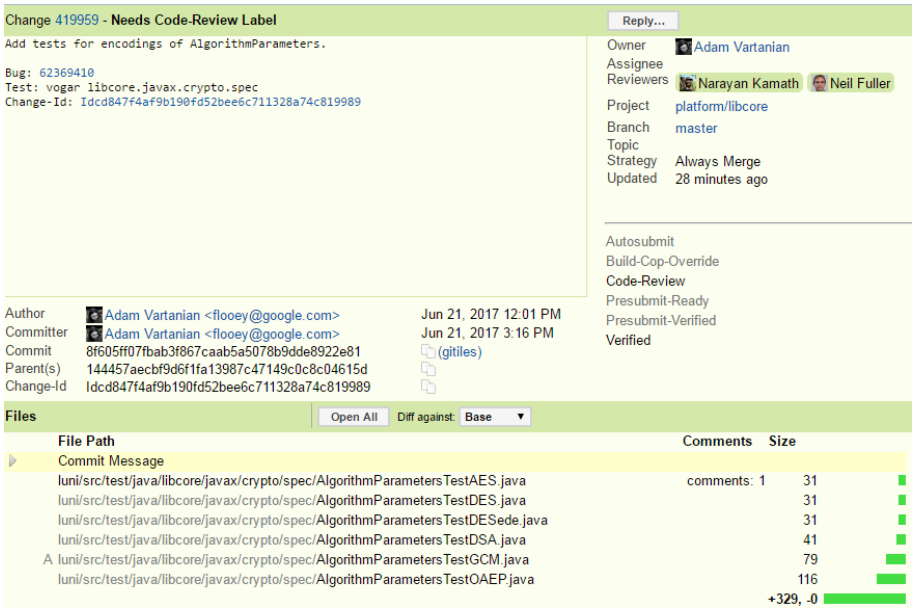
Abstract. In software projects, pushing software change request proposals through a code review process improves the quality and life time of the resulting software. Such processes sometimes take long periods of time and this reflects on the cost of the software development and the work satisfaction of developers negatively. The presence of mechanisms which can determine the types of revision requests the reviewers may request on a particular proposal can help a developer to prevent or reduce long code review cycles. This is because the developer can improve the revision further before starting the process to hear from the reviewers. In this study, as a first step towards this goal, we attempt to develop a mechanism that automatically determines whether a change proposal would be approved or undergo revisions.

Keywords: Source code reviews, Gerrit, Review approval prediction

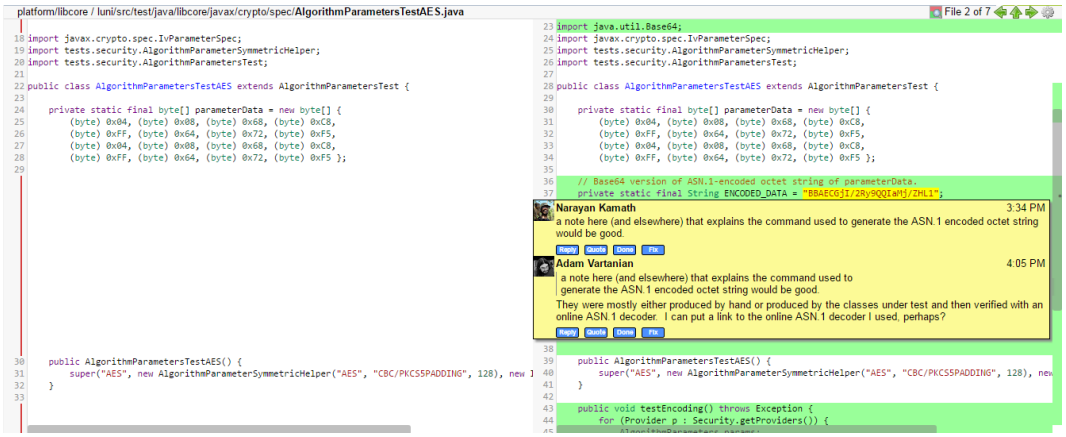
1 Giriş

Kaynak kod denetimleri (İng. code reviews) kaliteli ve güvenilir yazılım sistemleri inşa etmek için yapılması gereken önemli aktivitelerden bir tanesidir [10, 13, 18]. Günümüzde bir yazılım projesinde herhangi bir kod değişikliği önerisinin projenin kod havuzuna(İng. repository) entegre edilebilmesi için öncelikle kod denetiminden geçmesi istenir. Şekil 1'de Gerrit[4] adlı web tabanlı kod denetim aracından bir ekran görüntüsü yer almaktadır. Burada açık kaynak kodlu Android işletim sistemi projesinde yapılmak istenen bir değişiklik görülmektedir. Bu ekran vasıtasıyla değişikliği yapmak isteyen kişinin adı (İng. owner), değişikliği değerlendirecek kişilerin adları (İng. reviewers), değişikliğin yapıldığı dosyalar (ekranın alt tarafında), değişikliğe neden olan hata raporu (İng. bug), değişikliğin kısa özeti, tarihi, ait olduğu proje ve dal (ing. branch) isimleri gibi birçok bilgiye ulaşmak mümkündür. Şekil 2 ise değişiklik önerisinin bir parçası olan bir dosyadaki içerik değişikliğinin Gerrit tarafından nasıl görsellendiğini göstermektedir.

Kod denetlemesi yapan kişiler değişikliği inceleyip değişikliği olduğu gibi onaylayabilirler. Aksi halde değişiklikte beğenmedikleri yerleri, değişiklik önerisi yapan geliştiriciye, denetleme aracının arayüzü yardımıyla bildirebilirler. Şekil 2'de sarı arkafonlu kutularda geliştirici ile denetleyiciler arasında yapılan tartışmaya dair bir mesaj dizisi de görülmektedir. Değişikliği öneren geliştirici bu durumda geri bildirime dair yeni düzenlemeler yapar ve yeniden değerlendirilmek

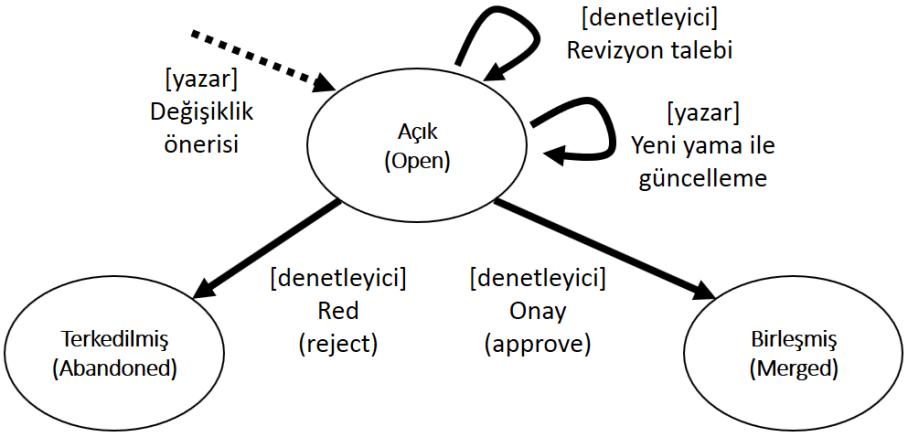


Şekil 1. Gerrit[4] kaynak kod denetim aracı ekran görüntüsü[7]



Şekil 2. Bir dosyada önerilen değişikliklerin dosyanın orijinal halinden farklılıkları şeklinde gösterilmesi[6]

üzere değişiklik önerisini günceller. Bu iterasyon değişiklikler kabul edilene kadar devam eder. Bazı durumlarda geri bildirimler sonucu değişiklik önerisinden vaz geçilir. Çoğunlukla ise iterasyonlar sonucu değişiklik önerisi olgun bir hal alır ve sonunda projenin kod havuzuna entegre edilir. Şekil 3'de bir değişiklik önerisinin içinde bulunabildiği durumlar ve bu durumlarda alınabilen eylemler görülebilir.



Şekil 3. Değişiklik Önerisi Durum Diyagramı

Kod denetimi, yazılımlar içerisinde hataların erken bulunmasını sağlar; kodun okunaklılığını ve bakımını kolaylaştırır. Dolayısıyla kod denetimi yapmak, yazılımın hem kalitesine hem de ömrüne pozitif olarak etki etmektedir. Diğer taraftan denetim süreçlerinin uzaması yazılım geliştirme maliyetlerinin artmasına ve geliştiricilerin iş memnuniyetlerinin azalmasına yol açar. Özellikle fiziksel olarak birbirinden uzaktaki, hatta bazen farklı kıtalardaki takımların beraber geliştirdikleri yazılımlarda değişikliğin yazarı ile denetleyiciler farklı saat dilimlerinde bulunabilmektedir. Böyle durumlarda önerilen bir değişiklik için geri bildirim alınması ve değişikliğin onaylanarak sonuçlandırılması günler sürebilir. Dolayısıyla önerilen bir değişikliğin başka geliştiriciler tarafından nasıl bir reaksiyonla karşılaşacağını kestirmek ve buna göre denetim sürecini başlatmadan bir takım önlemler almak süreci kısaltacaktır. Örneğin, değişiklikleri denetimcilerin daha kolay anlayacağı ve daha emin şekilde onaylayabileceği ufak ardışık değişikliklere bölmek izlenebilecek bir yöntemdir. Fakat bu ve bunun gibi yöntemler ancak yeterli tecrübe kazanıldıktan sonra efektif olarak uygulanabilir. Beraber çalışılan takım üyelerini tanımak, bu kişilerin neye önem verdiğini anlamak ve kaynak kodda izlenen takım içi geleneklere aşina olmak, o projede uzun zaman çalışmış olmayı gerektirir.

Dolayısıyla önerilen bir kod değişikliğine denetimcilerin nasıl reaksiyon vereceğini tahmin edebilecek bir araç, geliştiricilerin değişikliklerini denetime açmadan önce daha fazla olgunlaştırabilmelerine ve onaylanması daha olası olacak

şekilde değişiklik önerilerini yapmalarına yardımcı olacaktır. Böylece denetim sürecinin süresi kısalacak ve yol açtığı maddi maliyet azalacaktır. Aynı zamanda geliştiricinin, yapmak istediği değişiklikleri daha hızlı bir şekilde yazılıma entegre edebilmesi, kişinin iş memnuniyetini arttıracaktır.

Bu çalışmada bu amaca doğru ilk adım olarak iki probleme cevap aramaya çalıştık:

1. Bir değişiklik önerisinin denetim süreci sonunda onaylanıp onaylanmayacağını tahmin edebilir miyiz?
2. Bir değişiklik önerisinin denetim süreci içerisinde herhangi bir revizyona uğramayıp doğrudan kabul mü olacağını, yoksa revizyona(lara) uğradıktan sonra mı kabul olacağını tahmin edebilir miyiz?

İlk soruyu başarıyla cevapladığımızda onaylanmayacak bir değişiklik önerisini denetim sürecine girmeden durdurabiliriz. İkinci soruyu cevapladığımızda ise revizyona uğrayarak kabul edilecek bir değişiklik önerisini yine denetim sürecine girmeden durdurabiliriz ve yazarın değişikliği olgunlaştırmasına hızlı bir şekilde olanak sağlayabiliriz. Tabii ki bir önceki paragrafta anlattığımız amaca ulaşılabilmesi için değişiklik önerisinin yazarına kabul/ret gibi ikili bir tahmin sonucu vermenin ötesinde, değişiklik önerisinin neden onaylanmayacağının veya neden revizyonlara uğraması gerektiğinin sebeplerinin bildirilmesi ve de ilgili kod bölgelerine işaret edilmesi gerekir. Daha önce de belirttiğimiz gibi bu iddialı amaca bir adım daha yaklaşmak üzere bu çalışmada yukarıda tanımladığımız iki alt probleme odaklandık.

2 İncelenen Veri Kümesi

Veri olarak açık kaynak kodlu Android işletim sisteminin geliştirilmesi sırasında ortaya çıkan kod denetimlerini incelemeye karar verdik. Android projesinde kod denetimi için bildirinin giriş bölümünde bahsettiğimiz web tabanlı Gerrit[4] aracı kullanılmıştır[2]. Tüm Gerrit verilerine Gerrit sunucusu üzerinden bir REST API vasıtasıyla ulaşılabilmektedir[5]. Sunucunun cevapları JSON formatındadır.

Hamasaki vd.[13] yaptıklari bir çalışmada REST API yoluyla 21 Ekim 2008 ile 27 Ocak 2012 tarihleri arasında yapılan $11,633$ değişiklik önerisinin kod denetimlerinde ortaya çıkan verileri indirmiş, bazı basit transformasyonlar uyguladıktan sonra[1] benzer çalışmalara yardımcı olmak amacıyla internet üzerinden yayımlamıştır[1]. Bu çalışmamızda bu veri kümesinden faydalandık.

2.1 Veride Bulunan Kolonlar

Bu veri kümesinde her bir değişiklik önerisine ait 35 tane öznitelik kolonu bulunmaktadır. Bu kolonlardan öncelikle her iki problemin de çözümü sırasında kullanımı mantıklı olmayacak olanları veriden çıkardık. Bu kolonların bazıları her

[1] Örneğin, bir değişikliğe atanmış denetleyicilerin adları kaldırılıp, yerine toplam atanan denetleyici sayısı bilgisi yerleştirilmiştir

değişiklik için eşsiz bir değere sahip, bir kısmı da denetim süreci başında henüz bilinmeyen değerleri göstermekte: Değişikliğik önerisinin belirteci (Change ID), önerinin web adresi (Link), denetim sürecinde yapılmış toplam mesajlaşma sayısı (#Comments), denetim sürecinde dosya içlerinde yapılmış toplam mesajlaşma sayısı (InlineComments), yamalar içerisinde bulunan toplam dosya sayısı (#FilesInPatch).

Geriye kalan kolonları dört temel guruba ayırabiliriz:

1. Değişiklik önerisi sahibine ait denetleme süreçleri sonucu oluşmuş istatikler: Yazara ait açık / terkedilmiş / birleştirilmiş / onaylanmış / doğrulanmış durumdaki değişiklik önerisi sayısı; yazarın yaptığı kod denetimi sayısı;yazarın kod denetimlerinde toplam mesajlaşma miktarı vb.
2. Değişiklik önerisinin denetlenmesi sırasında değişik rollerde kaç kişinin yer alacağı ile ilgili rakamlar: Değişiklik için kaç denetleyici atanmış; kaç kişinin denetimi onaylaması gerekiyor; kaç kişinin denetimi doğrulaması gerekiyor vb.
3. Değişiklik ile ilgili üstveri sayılabilecek öznitelikler: Değişikliğin hangi proje için yapıldığı; değişikliğin versiyon sistemi açısından projenin hangi dalında (İng. branch) yapıldığı vb.
4. Değişikliğin içeriği ile ilgili istatistikler: Değişiklik sonucu eklenen / silinen / değişen satır sayısı; eklenen / silinen / değişen / ismi değişen dosya sayısı vb.

Bu kolonların dışında veride iki kolon daha mevcuttur:

1. Status: Değişiklik önerisinin denetim sonucunun birleştirilmiş mi (İng. merged) yoksa terk edilmiş mi (İng. abandoned) olduğu[2];
2. PatchSets: Önerinin maruz kaldığı toplam yama sayısı.

Buraya kadar anlattığımız terminoloji ışığında üzerinde çalışacağımız iki problemi şu şekilde yeniden tanımlayabiliriz:

Bir değişiklik önerisi verildiğinde bu değişiklik önerisinin yukarıda verilen dört kategorideki değerlerini göz önünde bulundurarak şunları ne kadar başarılı bir şekilde tahmin edebiliriz?:

- (1. Problem) Status kolonunun son değeri
- (2. Problem) PatchSets kolonundaki son değerin bir mi, birden büyük mü olacağı

2.2 Veri Kümesinin Temizlenmesi

Çalışmalara başlamadan önce veriden gürültü sayılabilecek aykırı değerleri çıkardık. Bu ayıklama sırasında kullandığımız kısıtlar şu şekildeydi:

[2] 1 seneden fazladır durumu "açık" olanları da terkedilmiş var saydık

– Bir alt projeye ait değişiklik önerisi sayısı 100'den az ise bu alt projenin kısa/deneysel olduğunu ve projeye özel bir denetleme kültürünün oluşması için projenin yeterli bir ömrünün olamamış olacağını varsaydık.
– Bir değişiklik önerisinde toplam satır değişikliği sayısı 500'den fazla ise bu değişikliğin bir otomasyon vasıtasıyla yapılmış kod göçünün bir parçası olduğunu, dolayısıyla yaşanan denetleme sürecinin ortalama bir değişiklik önerisini değerlendirmekte temel alınamayacağını varsaydık.

Bu ayıklama sonucu 11 bin değişiklik önerisinden geriye yaklaşık 9 bin öneri kaldı. Son olarak ikinci problemde kullanmak üzere, terkedilmiş (abandoned) olarak işaretlenmiş değişiklik önerilerinin PatchSets kolonu değerini sonsuz olarak değiştirdik.

3 Denetim Sonucu Tahmini

Bu bölümde, bir değişiklik önerisi verildiğinde bu değişiklik önerisinin "Status" kolonunun son değerini etkili bir şekilde tahmin edip edemeyeceğimizi inceledik. Çözümümüzde Weka[9] isimli içerisinde makine öğrenme algoritmaları içeren veri madenciliği yazılımını kullandık.

3.1 Baskın Özniteliklerin Seçimi

İlk adım olarak verimizdeki kolonlar üzerinden Status kolon değerini belirlemede en baskın olanları tespit etmeye çalıştık (İng. feature selection). Bunun için üç farklı yöntem uyguladık: 1) karşılıklı ilişkilere bakma (CorrelationAttributeEval), 2) bilgi kazanımına bakma (InfoGainAttributeEval), ve 3) öğrenici kullanma (Learner olarak J48 ağacı ve arama metodu olarak BestFirst arama metodu ile). Bu yöntemler sonucu ortaya Tablo 1'de görülen sonuçlar çıktı. Bu sonuçlara göre bir sonraki adımda kullanmak üzere her üç yöntemin önemli olduğunu düşündüğü veya bir yöntem içerisinde önem değeri (karşılıklı ilişki değeri veya bilgi kazanımı değeri) oldukça yüksek olan öznitelikleri seçtik.

3.2 Öğrenme Yöntemlerinin Uygulanması

Seçtiğimiz öznitelikleri kullanarak veri modelini eğitmek için üç farklı öğrenme yöntemi uyguladık: Bayesian Network, SVM ve Random Forest. Bu yöntemler ile eğitilen modellerin tahmin konusunda başarısını 10 katlı çapraz doğrulama (İng. 10-fold cross validation) tekniği kullanarak ölçtük. Başarı ölçümü için şu metrikleri kullandık:

– Doğruluk (İng. accuracy): Değişim önerilerinden ne kadarının denetim sonucu doğru tahmin edilmiştir
– Tutturma (İng. precision): Birleşmiş (İng. merged) olacağı tahmin edilen değişim önerilerinin kaçta kaçı gerçekten birleşmiştir. Benzer şekilde terkedilmiş (İng. abandoned) olacağı tahmin edilen değişim önerilerinin kaçta kaçı gerçekten terkedilmiştir.

Tablo 1. Denetim Sonucunu Belirleyen En Baskın öznitelikler

Yöntem	**Özellikler**	**Açıklama**
CorrelationAttributeEval	Approvers, CodeReviewers, Dev_#approved, Dev_#assigned, Dev_#authors, Dev_#merges, Dev_#submitted, Dev_RealReviewer, Dev_#verified, Verifiers	Karşılıklı ilişki değeri 0.3'ten büyük olanlar
InfoGainAttributeEval	Approvers, CodeReviewers, Dev_#authors, Dev_#committed, Dev_#merges, Dev_#owners, Dev_#PostComments,, Verifiers	Bilgi kazanımı değeri 0.25'ten büyük olanlar
Learner Tabanlı	Branch, Approvers, AssignedReviewers, CodeReviewers, DelFiles, Dev_#merges, Dev_#submitted, Dev_#verified, Dev_RealReviewer, Verifiers	

Doğru Pozitif (DP): Gözlem pozitif, tahmin edilen pozitif

Yanlış Pozitif (YP): Gözlem negatif, tahmin edilen pozitif

Doğru Negatif (DN): Gözlem negatif, tahmin edilen negatif

Yanlış Negatif (YN): Gözlem poizitif, tahmin edilen pozitif

$$Tutturma = \frac{DP}{DP + YP} = \frac{\text{doğru olarak pozitif tahmin edilenler}}{\texttt{tüm pozitif tahmin edilenler}}$$

– Bulma (İng. recall): Birleşmiş olan değişim önerilerinin kaçta kaçının birleşeceği tahmin edilmiştir. Benzer şekilde terkedilmiş olan değişim önerilerinin kaçta kaçının terkedileceği tahmin edilmiştir.

$$Bulma = \frac{DP}{DP + YN} = \frac{\texttt{doğru olarak pozitif tahmin edilenler}}{\texttt{tüm pozitif gözlemlenenler}}$$

Tablo 2 her bir yöntem için elde edilen başarım değerlerini göstermektedir. Tüm yöntemlerde başarım oranının oldukça yüksek olduğunu görüyoruz. Sonuçları detaylı incelediğimizde öğrenme yöntemlerinin aslında denetim sonucunu iki kategorideki özniteliklere göre belirlediğini anlıyoruz:

1. Değişiklik önerisini yapan yazarın geçmiş istatistikleri (ne kadar değişiklik yaptığı, kaç denetimden geçtiği vb.) yani tecrübesi: Tecrübe ile ilişkili özniteliklerin değerleri arttıkça denetim sürecinin başarıyla sonuçlanacağı tahmin ediliyor.
2. Denetim sürecine katılacak kişilerin sayısı (denetleyiciler, doğrulayıcılar, onaylayıcılar vb.): Denetime katılan kişi sayısı azaldıkça denetim sürecinin başarıyla sonuçlanacağı tahmin ediliyor.

Tablo 2. Denetim Sonucunu Tahmin Başarısı

Yöntem	Doğruluk	Tutturma	Bulma	Sınıf
Bayesian Network	%96.43	0.96	0.98	Birleşmiş
		0.96	0.93	Terkedilmiş
SVM	%97.26	0.97	0.98	Birleşmiş
		0.97	0.94	Terkedilmiş
Random Forest	%97.89	0.97	0.99	Birleşmiş
		0.98	0.95	Terkedilmiş

Buna göre eğer deneyimsiz bir yazar çok sayıda denetleyicinin denetim yapacağı bir değişiklik önerisinde bulunursa, yüksek oranda bu denetimin başarısızlıkla sonuçlanacağı tahmin edilecektir.

4 Değişiklik Önerisinin Revizyona Uğrayıp Uğramayacağının Tahmin Edilmesi

Bu bölümde, bir değişiklik önerisi verildiğinde bu değişiklik önerisinin birden fazla revizyona uğrayıp uğramayacağını tahmin etmeye çalışacağız. Bir değişiklik önerisinin "PatchSets" adlı kolonuna bakarak denetim süreci sırasında kaç tane yama (İng. patch) güncellemesine maruz kaldığını görebilmekteyiz. Her değişiklik önerisi bir yama ile başlamaktadır. Eğer denetleyiciler tarafından revizyon talepleri olursa, o zaman önerinin yazarı bunları gerçekleştirip yeni bir yama ile öneriyi güncellemektedir.

Bu problemin çözümü için ilk probleme benzer bir yaklaşım izleyerek öncelikle PatchSets kolonu değerlerini belirleyen en etkili öznitelikleri seçtik. Bu öznitelikleri kullanarak yine üç farklı yöntemle veri üzerinde eğitim yaptık ve daha sonra çapraz doğrulama ile tahmin başarısını ölçtük. Tablo 3, uyguladığımız her öğrenme yöntemi için elde ettiğimiz başarım değerlerini göstermektedir.

Tablo 3. Revizyona Maruz Kalma Tahmin Başarısı

Yöntem	Doğruluk	Tutturma	Bulma	Sınıf
Bayesian Network	%80.54	0.71	0.86	Uğramaz
		0.89	0.76	Uğrar
SVM	%91.47	0.84	0.96	Uğramaz
		0.97	0.88	Uğrar
Random Forest	%94.59	0.94	0.92	Uğramaz
		0.94	0.96	Uğrar

Sonuç olarak ilk problemdeki sonuca benzer bir şekilde başarı oranının yüksek olduğunu ve yazarın deneyimi ile denetim sürecine dahil olan kişilerin sayısının tahmin sonucuna etki ettiğini görüyoruz.

5 Yazar ve Denetimci Sayısından Bağımsız Tahmin Yapılması

Her iki problemde de geliştirdiğimiz tahmin mekanizmaları yazarın deneyimini ve denetim sürecinde yer alacak kişi sayısını öne çıkarmaktadır. Dolayısıyla bu mekanizmalar deneyimsiz bir geliştiricinin yazarı olduğu ve çok sayıda denetimcinin yer aldığı bir değişiklik önerisinin reddolacağı veya revizyonlara maruz kalacağı yönünde tahminde bulunmaktadır. Bu bölümde yazar ve denetimci sayısı verilerini çıkarıp, yalnızca değişikliğin içeriği ile ilgili verilere bakarak, başarılı tahminlerde bulunup bulunamayacağımızı inceledik.

Veri kümesinde değişiklik önerilerinin içeriği ile ilgili iki tip öznitelik bulunmakta:

1. Eklenen/silinen/değişen toplam satır sayısı;
2. Eklenen/silinen/değişen/ismi değişen dosya sayısı.

Yalnızca bu öznitelikler kullanılarak RandomForest öğrenme yöntemi uygulandığında ikinci problem ile ilgili tahmin başarısı Tablo 4'de görülmektedir.

Tablo 4. Yazar ve Denetimci Sayısı Bağımsız Revizyona Maruz Kalma Tahmin Başarısı

Yöntem	**Doğruluk**	**Tutturma**	**Bulma**	**Sınıf**
RandomForest	%83	0.78	0.79	Uğramaz
		0.86	0.85	Uğrar

Başarım daha önceki yaklaşıma göre bir miktar düşmüş olsa da halen oldukça yüksek bir düzeyde. Bununla beraber bu defa tahmin mekanizmasının değişiklik büyüklüğü üzerinden tahmin yaptığını gözlemlemekteyiz. Diğer bir değişle bir değişiklik önerisinde toplam değişen satır sayısı ve dosya sayısı büyük olduğunda önerinin revizyona maruz kalacağı tahmin edilir.

6 Değişim İçeriğinin Tahmin İçin Kullanılması

Bir önceki bölümün sonucuna göre, yalnızca bir satır değişiklik içeren bir öneri yapıldığında bu önerinin herhangi bir revizyona uğramadan kabul olacağının tahmin edilmesi muhtemeldir. Fakat değişiklik aslında kodun kritik bir yerinde ise revizyonlara maruz kalabilir. Bu tip durumların üstesinden daha iyi gelebilmek için bu bölümde alternatif bir yaklaşım denedik. Buna göre kod dosyalarında yapılan değişikliklerin ne tür değişiklik oldukları bilgisini üretip bu bilgileri tahmin için kullanmaya çalıştık. Kod dosyalarındaki değişikliklerin türünü anlamak için ChangeDistiller[3] adı verilen Fluri vd. [12] tarafından üretilen aracı kullandık. Bu araç bir Java dosyasının iki versiyonu arasındaki farkı hesap edip bu farkı 48 ayrı değişiklik sınıfına sınıflandırmaktadır. Bu sınıflardan bazıları şöyle:

yorum silinmiş, yorum eklenmiş, şart ifadesi değişmiş, ifade silinmiş, ifade eklenmiş, ifade sırası değişmiş. Tüm sınıf listesine `https://goo.gl/KeCxce` adresinden ulaşılabilir.

Bu aracı kullanmak için öncelikle, veri kümesinde bulunan her bir değişiklik önerisi için, veri önerisi içerisinde bulunan kaynak dosyalarının orijinal hallerini ve değişim önerisi ile yapılmak istenen değişikliği Gerrit üzerinden indirdik. Her bir dosyanın orijinal ve yeni halini ChangeDistiller'a verdik ve ChangeDistiller'dan dönen değişiklik ile ilgili tespit edilen sınıfları kaydettik. Böylece her bir değişiklik önerisi için 48 değişiklik sınıfından kaçar tane olduğunu bulduk. Daha sonra bu veriyi öğrenme yöntemlerinde tahmin modelimizi eğitmek için kullandık ve yalnızca bu veri kullanılarak her iki problemi de çözmeye çalıştık.

Maalesef deney sonuçları bu yaklaşımın henüz etkin olmadığını gösterdi. Buna sebep olarak ChangeDistiller değişiklik sınıflarının değişiklikleri karakterize etmekte yetersiz kalmış olabileceğini düşünüyoruz. Yaklaşımımızın mantıklı olduğuna inanıyoruz; bu yüzden, gelecekteki çalışmalarımızda değişiklik sınıflarının arttırılması ve değişiklikleri tanımlama açısından sınıfların derinleştirilmesi üzerinde çalışmayı düşünüyoruz.

7 İlgili Çalışmalar

Giderek kod denetimi verilerinin açık kaynak kodlu projeler vasıtasıyla halka açık hale gelmeye başlaması ile beraber bu veriler üzerinde faydalı veri madenciliği faaliyetleri yapılabileceğini işaret eden öncü çalışmalar yapılmıştır. Mukadam vd. [17] çalışmasında Android projesinin Gerrit üzerindeki kod denetim verisinin genel yapısını incelenmiştir. Hamasaki vd. [13] ise bu gibi verileri indirmek ve işlemek üzerine bir yaklaşım önermiş; toplam 3 projeden topladıkları verileri yayınlamıştır[8].

Bu bildiride çalıştığımıza benzer, kod denetimi sürecinde kod değişikliğini öneren yazara yardımcı olmaya yönelik bazı geri bildirimlerin yapılabileceği fikri Jeong vd. [15] ile Hellendoorn vd. [14] tarafından da incelenmiştir. Jeong vd. [15] Firefox ve Mozilla projelerinde Bugzilla sistemi üzerinden kod denetim verilerini inceleyerek hem bir kod değişim önerisinin denetimden geçip geçmeyeceğini tahmin etmeye çalışmış hem de bir değişiklik için en iyi denetimi yapabilecek kişileri tahmin etmeye çalışmıştır. Hellendoorn vd. [14] dil modelleri kullanarak bir değişiklik önerisinin projede daha önce kabul edilen değişikliklere ne kadar benzediğini hesaplayarak değişikliğin onaylanıp onaylanmayacağını değerlendirmeye çalışmıştır.

Kod denetimi yaparken denetimcinin işini kolaylaştırmaya yönelik çalışmalar da yapılmıştır. Zhang vd. [19] kod denetimi sürecinde denetim yapanların değişikliklerin etkilerini daha kolay anlamalarını sağlamak için bir yama inceleme aracı geliştirmiştir. Bosu vd. [11] kod denetimlerinde denetimcilerin verdiği geri bildirimlerden faydalı olanların ne tür karakteristikleri olduğunu incelemiş ve buna dayanarak bir geri bildirimin sürece faydalı olup olmayacağını ayırt edebilen bir sınıflandırma modeli geliştirmiştir.

Genel olarak kod denetimi süreçlerinin incelenmesi ve iyileştirilmesine yönelik çabalar da bulunmaktadır. Rigby vd. [18] 10'dan fazla projenin kod denetim verilerini inceleyerek denetimlerin ne kadar sürdüğü, kaç kişinin denetimlerde görev aldığı, denetim süreçlerinin ne kadar etkin olduğu gibi sorulara cevaplar aramaya çalışmıştır. Kitagawa vd. [16] kod denetim süreçlerinde denetimcilerin davranışlarını daha iyi anlamak için denetim sürecini bir oyun teorisi modeli kullanarak tanımlamış ve bu modelin geçerliliğini Gerrit veri kümeleri üzerinde doğrulamıştır. Bird vd. [10] Microsoft firması içerisinde gerçekleşen kod denetimi bilgilerini toplayan ve bu bilgiler üzerinden bir takım metrikler üreterek bunları geliştirme takımlarına sunan bir analiz sistemi geliştirmiştir. Bu sistem yardımıyla takımların denetim süreçlerini daha iyi yönettikleri bildirilmiştir.

8 Sonuçlar ve Gelecek Çalışmalar

Kod denetimleri yapmak yazılım sistemlerinin kalitesine ve ömrüne olumlu etki etmektedir. Zaman zaman kod denetimleri uzun süreler alır. Bu durum çalışanların memnuniyetini olumsuz etkiler ve kod denetiminin yazılım geliştirmeye olan maliyetini arttırır. Kod değişikliğine denetimcilerin ne tür yanıt vereceğini önceden tahmin edebilmek ve buna göre denetime girecek bir değişikliği öncesinde daha fazla olgunlaştırmak bu problemin ortaya çıkacağı durumları azaltacaktır. Bu amaca doğru ilk adım olarak denetleme verilerini kullanarak denetleme süreci sonucunu ve değişiklik önerisinin revizyona maruz kalıp kalmayacağını tahmin eden çözümler geliştirmeye çalıştık. Geliştirdiğimiz tahmin modellerinin, büyük oranda, değişikliği yapan yazarın ne kadar tecrübeli olduğuna ve denetimde kaç kişinin yer alacağına bakarak tahmin yaptığını gördük. Değişikliğin içeriğini kullanarak tahmin yaptığımızda ise, değişikliğin büyüklüğünün tahmini etkilediğini gözlemledik. Değişiklik içeriklerini ChangeDistiller adlı araç vasıtasıyla değişiklik sınıfları ile etiketleyip buna göre tahmin yapmayı denediğimizde ise maalesef beklediğimiz sonucu alamadık. Bunun ana sebebinin değişiklik sınıflarının değişiklikleri yeterli oranda karakterize edecek derinliğe sahip olmaması olduğunu düşünüyoruz. İleriki çalışmalarımızda ChangeDistiller sınıflarına benzer, fakat kod değişimlerini daha derinlemesine temsil edebilecek sınıflar geliştirmeyi ve denetim sonucunu bu şekilde tahmin etmeyi düşünüyoruz. Ayrıca kabul/ret gibi bir tahminin yanında, değişikliğin yazarına, revizyona neden olacak kod bölgelerinin neler olduğunu bildirebilecek mekanizmalar üzerinde çalışmayı planlıyoruz.

Kaynakça

1. Android gerrit veri kümesi (jun 2017), `http://sdlab.naist.jp/reviewmining/`
2. Android projesi gerrit web arayüzü (jun 2017), `https://android-review.googlesource.com`
3. Changedistiller aracı kaynak kodu (jun 2017), `https://bitbucket.org/sealuzh/tools-changedistiller/wiki/Home`
4. Gerrit (jun 2017), `https://www.gerritcodereview.com/`
5. Gerrit rest api (jun 2017), `https://gerrit-review.googlesource.com/Documentation/rest-api.html`

6. Gerrit Örnek dosya farkı ekranı (jun 2017), https://goo.gl/3MGPKs
7. Gerrit Örnek kod denetimi (jun 2017), https://android-review.googlesource.com/\#/c/419959/
8. Review mining (jun 2017), http://sdlab.naist.jp/reviewmining/
9. Weka (jun 2017), http://www.cs.waikato.ac.nz/ml/weka/
10. Bird, C., Carnahan, T., Greiler, M.: Lessons learned from building and deploying a code review analytics platform. In: 12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015. pp. 191–201 (2015), https://doi.org/10.1109/MSR.2015.25
11. Bosu, A., Greiler, M., Bird, C.: Characteristics of useful code reviews: An empirical study at microsoft. In: Proceedings of the 12th Working Conference on Mining Software Repositories. pp. 146–156. MSR '15, IEEE Press, Piscataway, NJ, USA (2015), http://dl.acm.org/citation.cfm?id=2820518.2820538
12. Fluri, B., Wuersch, M., PInzger, M., Gall, H.: Change distilling: Tree differencing for fine-grained source code change extraction. IEEE Trans. Softw. Eng. 33(11), 725–743 (Nov 2007), http://dx.doi.org/10.1109/TSE.2007.70731
13. Hamasaki, K., Kula, R.G., Yoshida, N., Cruz, A.E.C., Fujiwara, K., Iida, H.: Who does what during a code review? datasets of oss peer review repositories. In: Proceedings of the 10th Working Conference on Mining Software Repositories. pp. 49–52. MSR '13, IEEE Press, Piscataway, NJ, USA (2013), http://dl.acm.org/citation.cfm?id=2487085.2487096
14. Hellendoorn, V., Devanbu, P.T., Bacchelli, A.: Will they like this? evaluating code contributions with language models. In: 12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015. pp. 157–167 (2015), https://doi.org/10.1109/MSR.2015.22
15. Jeong, G., Kim, S., Zimmermann, T., Yi, K.: Improving code review by predicting reviewers and acceptance of patches. Research on Software Analysis for Error-free Computing Center Tech-Memo (ROSAEC MEMO 2009-006) pp. 1–18 (2009)
16. Kitagawa, N., Hata, H., Ihara, A., Kogiso, K., Matsumoto, K.: Code review participation: Game theoretical modeling of reviewers in gerrit datasets. In: Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering. pp. 64–67. CHASE '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2897586.2897605
17. Mukadam, M., Bird, C., Rigby, P.C.: Gerrit software code review data from android. In: Proceedings of the 10th Working Conference on Mining Software Repositories. pp. 45–48. MSR '13, IEEE Press, Piscataway, NJ, USA (2013), http://dl.acm.org/citation.cfm?id=2487085.2487095
18. Rigby, P.C., Bird, C.: Convergent contemporary software peer review practices. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. pp. 202–212. ESEC/FSE 2013, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2491411.2491444
19. Zhang, T., Song, M., Pinedo, J., Kim, M.: Interactive code review for systematic changes. In: Proceedings of the 37th International Conference on Software Engineering - Volume 1. pp. 111–122. ICSE '15, IEEE Press, Piscataway, NJ, USA (2015), http://dl.acm.org/citation.cfm?id=2818754.2818771