# Birlikte Değiştirilen Dosyaları Tahmin Etme: Dış, Kavramsal Bir Çoğaltma

# Özet

Bir yazılım projesi, birbirine bağlı çok sayıda dosyadan oluşabilir. Bağlı olduğu dosyaları geliştirilen veya başka bir geliştirici tarafından değiştirilen bir yazılım geliştiricisi, tüm ilgili dosyalar üzerinde gerekli düzenlemeleri kaçırırken hataları tetikleyebilir. Kod değişikliği sırasında birlikte düzenlenmesi gereken dosyaları tanımlamak için bir tahmin modeli oluşturuyoruz ve modelimizin 10 yıldan uzun bir süredir iki Apache projesinin geliştirme geçmişindeki performansını değerlendiriyoruz. Wiese ve arkadaşlarının ortak değişmiş dosyaları öngörme konusundaki önceki çalışmasına dayanarak dışsal, kavramsal bir çoğaltma çalışması yürütüyoruz. Çalışmamız aynı hedefi paylaşıyor ancak deneysel tasarımı veri seti yapımı, dosya çifti seçimi, özellik seçimi ve model çıktısı açısından farklılaştırıyor. Tahmini modelimizin sonuçları, her ne kadar aynı performans ölçütleri kullanılsa da, esas olarak bu önlemlerin hesaplanmasındaki farklılıklar nedeniyle, Wiese ve arkadaşlarının çalışmasında bildirilenden çok daha düşük. Kararlılık derecesinde değerlendirilen modeller, tüm dosya çiftlerinde toplananlara göre sırasıyla% 20 ve% 45 daha düşük geri çağırma ve hassasiyet oranlarına ulaşabilir. Pratik olarak daha faydalı olsa da, bir işlem sırasında birlikte değiştirilecek tüm dosyaları tahmin etmek, belirli bir dosyanın bu işlemde değiştirilip değiştirilmeyeceğini tahmin etmekten daha zordur. Bir ortak değişimin içeriği, projedeki bir dosyanın merkeziyet derecesi veya proje özellikleri hakkında daha fazla bilgi, gelecekte bu tür tahmincilerin oluşturulmasında daha fazla fikir ortaya koyabilir.

# Giriş

Yazılım projeleri, içeriksel, işlevsel veya diğer bağımlılık türlerinin mevcut olduğu binlerce dosyadan oluşur. Kaynak dosyada yapılan bir değişiklik, bu tür bağımlılıklar nedeniyle diğer dosyalar üzerinde birkaç değişiklik gerektirebilir. Bir geliştiricinin kodu yanlış anlaması veya deneyim eksikliği, tüm ilgili dosyalar üzerinde yapılması gereken değişiklikleri engelleyebilir. Bu sonuçta yazılım projelerinde kalite sorunları ortaya çıkarır. Ortak değişim tahminine odaklanan ampirik çalışmalar, geliştiricilere belirli bir kod değişikliğine ilişkin denetlenmesi veya düzenlenmesi gereken gerekli kod parçalarına rehberlik ederek kod bütünlüğünü ve güvenilirliğini korumayı amaçlar.

Literatürde kavşak tahmin modellerini geliştirmek için başlıca iki yaklaşım vardır. Birincisi, herhangi bir varlık düzeyinde çıkarılan bir yazılım projesinin bağımlılık grafiklerini, yani işletme gereksinimleri arasındaki bağımlılıklardan, yöntemler arasındaki statik çağrı bağımlılıklarına [1], ikincisi ise yazılım kodu depolarını kullanır ve öğrenmeye dayalı modeller oluşturur [2]. İlk yaklaşım, genellikle artan bir şekilde ayarlanan değişiklik etkisini güncellemek için manuel çalışmayı gerektirir ve değişiklikler semantik olarak ilişkili ve analiz edilmemiştir [1]. İkinci yaklaşım, kaynak koddan otomatik veri toplanmasına ve çeşitli sürümlerde yayınlanmış depolara dayanarak ve tahmin modellerini buna göre güncelleyerek doğası gereği daha dinamiktir.

Bu yazıda, Wiese ve ark. [3] sürüm kontrol sistemlerinden bağlamsal bilgileri kullanarak ve kaynak depoları kullanarak kaynak dosyalar için ortak değişim tahmin modelleri önerir ve bunları temel deneysel çalışmamız olarak seçti. İki projenin 10 yıllık gelişim verilerini, Apache Derby ve Apache CXF'i alarak ve taahhüt ve geliştiriciyle ilgili özellikleri hesaplayarak, temeli kavramsal olarak [3] olarak kopyaladık. Daha sonra, komisyonlar üzerindeki sıralamalarına ve ilişkili ortak değiştirilen dosyaları temel alan bir alt dosya grubu için ortak değişim tahmin modelleri oluşturduk. Çalışmamız, ortak çalışmadan [3] birlikte değiştirilmiş dosya çiftleri, model yapımı ve performans değerlendirmesi seçiminde 2 deney protokolü olarak farklılık göstermektedir.

# İlgili Çalışmalar

İlgili Çalışma Madencilik kodu depoları aracılığıyla dosya ortak değişikliklerinin izlenmesi konusundaki ilk çalışmalardan biri [4] 'te önerilmiştir. [4] 'deki yazarlar, tarihsel gelişim verilerinin kod gelişimini izlemek için kullanılabileceğini göstermektedir. Gall ve ark. [5] ayrıca sürüm kontrol sistemlerinden gelen geçmiş sürüm verilerini kullandılar ve dosyalar arasında evrimsel eşleşmeyi tahmin edebildiler. Daha sonra Zimmermann ve ark. [6], madencilik yazılımı depoları aracılığıyla elde edilen bilgileri kullanarak, yöntem düzeyinde ortak değişimin meydana geldiğini tahmin etmek için bir Eclipse eklentisi geliştirdi. Tasarlanan araç bazı ortak değişiklikler önerdi, ancak kesinliği% 30 civarındaydı. Yanlış alarm oranları çok yüksekti ve önerilerin doğruluğunu arttırmak için daha ileri adımlar atılması gerekiyordu.

Ortak değişim tahmini bağlamında farklı yaklaşımları kullanan başka çalışmalar da olmuştur. Örneğin, Canfora ve diğ. [7], dosya değişimlerini öngörmek için geçmiş kod değişikliklerinden ilişkilendirme kurallarını çıkardılar ve % 30 F ölçümü aldılar. Bu kadar düşük F-ölçüsünün olmasının nedeni, ilişkilendirme kurallarının ürettiği yüksek yanlış alarmlardan kaynaklanıyordu. Kurallar ortak değişiklikler bulmakta başarılı olmakla birlikte, aynı zamanda yanlış pozitifleri de çok yüksektir. Wiese ve diğ. [3], çalışmalarındaki ilişkilendirme kurallarını bağlamsal özelliklerle karşılaştırarak bunu kanıtladılar. Ayrıca, Hassan ve ark. [8] değişim yayılımını araştırdılar ve beş büyük açık kaynaklı yazılım projesini analiz ederek işletme tanesellikteki ortak değişiklikleri öngörmek için farklı sezgisel tarama kullandılar. Sezgileri veri kaynaklarına ve budama tekniklerine göre tanımlarlar. Varlık tabanlı geçmiş ortak değişim verilerini hem sıklık hem de yineleme yöntemlerini birleştiren karma budama yöntemiyle kullanırken en iyi hatırlama performansını elde etmişlerdir. Hassas performansı iyileştirmek için Hassan ve ark. [8] melez sezgisel tarama kullandılar ve kod düzen sezgiselini kullanarak varlık temelli tarihsel ortak değişim ve varlık temelli kod yapısının sonuçlarını birleştirdiler. Bu yeni yaklaşımla hassasiyet% 49'a çıkarken geri çağırma% 51'e düştü. [8] 'in yazarları daha iyi performans elde etmek için daha karmaşık bir sezgisel buluşmanın geliştirilmesi gerektiğini belirtti.

Macho ve diğ. [9], kaynak kod değişiklikleri ve taahhüt kategorileriyle ilgili ayrıntılı bilgileri kullanarak, yapı değişikliklerinin tahminini geliştirmek için bir çalışma yaptılar. Önceki çalışmalara göre çok daha yüksek performans elde etmişler ve sorun takip sistemini tahmin modellerine dahil ederek performansı daha da arttırmayı planlamışlardır. Ayrıca, Kouroshfar [10], makalesinde ortak değişim dağılımı ile yazılım kalitesi arasında bir ilişki olup olmadığını araştırmıştır. Dört açık kaynaklı Apache projesi kullandı ve ortak değişikliklerin yerinin kod kusurlarını etkilediğini gösterdi.

# Temel Çalışma

Wiese ve diğ. [3], hem sorun izleme hem de sürüm kontrol sistemlerinden içerilen bilgiler kullanılarak değişim modellerinin tanımlanıp tanımlanamayacağını gözlemlemek için ampirik bir çalışma yürüttüler. Önceden dosya çiftlerinde yapılan değişikliklerle eğitilmiş Random Forest sınıflandırıcısını kullanarak öngörücü modeller oluşturdular ve çiftteki ikinci dosyanın değişip değişmemesini önerir. Her dosya çifti için ayrı bir model eğitilmiştir, ancak tarihsel kod değişikliklerinde bulunan tüm dosya çiftleri için bu yapılmamıştır. Bir sorun için yapılan tüm taahhütleri tek bir değişiklik işleminde gruplandırdılar ve bu değişiklik işlemi tarafından değiştirilen tüm dosyalar birlikte değiştirilmiş bir dosya çifti olarak kabul edildi. Her değişim dosyası çifti için destek ve güven değerlerine dayalı birleştirme kuralları oluşturulmuştur. Wiese ve diğ. [3] birlikte değiştirilen dosya çiftlerinin listesi olarak ilgili 25 ilk ilişkilendirme kuralını seçtiler. Daha sonra, sorunlardan, tarihten ve iletişim ağından özellikler çıkararak bu 25 çift için modeller geliştirdiler. Apache Derby ve Apache CXF projelerinde [3] 'te rapor edilen tahmin modelleri, yaklaşık% 90 hassasiyet ve hatırlama değerleri elde eder. Bu orijinal çalışma ortak değişim tahmini bağlamında dikkat çekici bir performans sunmasına rağmen, bu kadar yüksek hatırlama ve hassasiyet oranlarını gözlemlemenin birkaç nedeni olabilir.

Öncelikle ve en önemlisi, [3] 'te tasvir edilen sınıflandırıcı oluşturma ve değerlendirme süreci, gerçekte değişim yayılımını temsil etmeyebilir. Örneğin, bir geliştirici A dosyasında değişiklik yaptığında, o zaman B dosyasının da değiştirilip değiştirilmemesi gerektiğini önermek için (A, B) çiftlerinde eğitilmiş olan tahmin modeli kullanılır. Ancak, eğer bir başka dosyanın C'nin de değiştirilmesi gerekiyorsa, öneri için (A, C) çifti üzerinde eğitilmiş başka bir model kullanılır. Bununla birlikte, uygulamada bir geliştirici A dosyasını değiştirdiğinde, A ile birlikte değiştirilmesi gereken dosyaların listesini ister. Bu nedenle, birlikte değiştirilen dosya kümesinde A dosyasının bulunduğu tüm modeller (örneğin (A, B), ( A, C), (A, D)) A dahil her bir işte tavsiyede bulunmalıdır.

İkincisi, [3] 'deki performans değerlendirme yöntemi, sınıflandırma geliştirme stratejileri nedeniyle bu kadar yüksek oranlara yol açmış olabilir. Orijinal çalışma, hassasiyeti rapor eder ve iki yazılım ürününün çoklu sürümlerinde kullanılan tüm modelleri hatırlatır, ancak her sürümde birkaç dosya 3 çiftine karşılık gelen çoklu modeller vardır. Bu nedenle, geri çağırma ve hassasiyet değerleri, farklı kodlanmış çiftlere sahip ayrı modellerin performansını göstermez. Ayrıca, pratik senaryoya benzemese de performans değerlendirmesi, bir çift (A, B) [3] üzerinde eğitilmiş her model göz önüne alınarak yapılmıştır. Bir geliştirici bir şirketi (A) değiştirdiğinde, diğer varlıkların A ile birlikte değiştirilmesi gerekenleri öğrenmek ister. Önerilen varlıkların sayısı ideal olarak gerçek değiştirilen varlıklarla karşılaştırılmalı ve hatırlama ve hassasiyet hesaplanmalıdır. tek bir taahhüt için önerilen tüm varlıklar üzerinde. Bu şekilde, hangi varlıkların doğru bir şekilde tahmin edildiğini değerlendirmek veya yanlış alarmlar vermek ve birlikte değiştirilen çiftlerin nasıl daha iyi tahmin edilebileceğini tartışmak daha kolaydır.

# Yöntem ve Malzemeler

Araştırmamız, [3] 'te yayınlanan temel çalışmanın dışsal ve kavramsal bir kopyası olarak tanımlanmaktadır. [11] 'de, yazarlar tüm çoğaltma çalışma türlerini tartışır ve bunları iki kategoriye ayırır: “iç” ve “dış”. Hem orijinal hem de replikasyon çalışmalarını yürüten kişilerin aynı olması ve iki çalışmanın farklı araştırmacılar tarafından yapılması durumunda “harici” olması bir çalışmayı tanımlar. [11] 'deki yazarlar aynı zamanda temel deneysel sonuçları kontrol etmek için farklı deney protokolü kullanan kopyaların “kavramsal çoğaltma” olarak adlandırıldığını belirtmektedir. Shepperd ve diğ. [12], çoğaltma çalışmalarının önemini tartışıyor ve iç kopyaların, ampirik yazılım mühendisliği alanındaki dış etkilerden daha sık bildirildiğini iddia ediyorlar. Silva ve diğ. [13], [12] 'de herhangi bir ampirik bilimde replikasyon çalışmalarının gerekli olduğunu ve bu alandaki kaliteyi arttırmak için dış araştırmaların daha fazla teşvik edilmesi gerektiğini desteklemektedir.

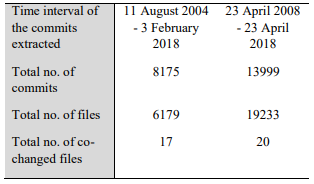
Aynı amacı, [3] ile sürüm kontrol sistemlerinden bağlamsal verileri kullanarak ortak varlıklar önermekle paylaşırken, çalışmamız, geliştiricinin daha iyi örneklendirdiğine inandığımız inanılan sınıflandırma oluşturma ve değerlendirme stratejileri ile ilgili temel çalışmadan farklıdır. uygulamada ihtiyaçlar. Wiese ve diğ. [3] birlikte değiştirilen dosyaların tahmin edilmesindeki her birinin performansını araştırmak için altı boyutu tanımlamaktadır. Bu boyutlar bir sorun izleme sisteminden (sorun, sorunlar üzerinden iletişim, geliştiricinin iletişimdeki rolü, iletişimdeki yapısal delik, iletişim özellikleri) ve sürüm kontrol sisteminden (içerik bağlamında) toplanabilir. Birlikte değişim öngörüsünün çok boyutlu bir fenomen olduğunu iddia etmelerine rağmen, bu modellerde en sık seçilen özellikler iki boyuttan kaynaklanmaktadır: bağlam bağlamı ve geliştiricinin iletişimdeki rolü [3]. Bununla birlikte, tüm bu boyutları toplamak, özellikle de çoklu sürüm kontrol ve sorun takip sistemlerine bağlı büyük ölçekli yazılım projeleri için çok maliyetlidir. Bu nedenle, [3] tarafından sonucuna vardığımız özelliklerimizi sadece taahhüt ve iletişim boyutlarından metriklere indirgendik.

Ayrıca, Wiese ve ark. [3], komisyonlara iliştirilen konulara dayanarak birlikte değiştirilen dosya çiftlerini buldu ve tüm komisyonlar üzerinden ilişkilendirme kurallarından türetilen destek ve güven değerlerine dayanarak ortak olarak değiştirilen ilk 25 dosya çiftini seçti. Bu yaklaşımı tüm taahhütlerde değiştirilen ilk 5 dosyayı göz önünde bulundurarak revize ettik, çünkü bir dosya için oluşturulmuş bir model ancak işleme geçmişinde dosyayla ilişkili yeterli sayıda değişiklik olması durumunda başarılı olabilir. Sonra bu ilk 5 dosyaya sahip çiftleri tanımladık, tüm kurallar üzerinden ilişkilendirme kurallarından elde edilen destek ve güven değerlerine dayanarak. Bu dernek kuralı hesaplamaları için detaylar sonraki alt bölümlerde verilecektir.

Son olarak, performans ölçülerinin hesaplanmasını the seçilen dosya çiftleri için farklı modellerde hatırlama ve hassasiyetten ’[3]“ commit bir bağlılığa birden fazla varlık öneren tek bir modelin hatırlatılması ve hassasiyetinden ”değiştirdik. Performans değerlendirmesinin detayları ilgili alt bölümde ayrıca açıklanmaktadır. Bulgularımızı özgün çalışma ile karşılaştırmak için [3] 'deki yaklaşım kullanılarak elde edilen performans ölçümlerini de rapor ediyoruz.

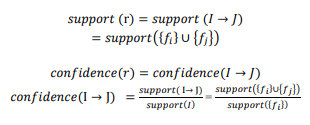
# Kullanılan Veriler

Wiese ve diğ. [3] ortak değişim tahmin modellerini oluşturmak için Apache Derby ve Apache CXF adlı iki depo analiz ettiler. Bulgularımızı özgün çalışma ile karşılaştırmak ve karşılaştırmak için aynı iki projenin GitHub havuzlarını incelemekteyiz. Taahhüt tarihini çıkarırken GitHub API kullandık. Seçtiğimiz özellikler kararlı ve geliştirici iletişim bağlamlarından olduğu için, çalışmamızda yalnızca GitHub depolarını benimsemek yeterliydi. Apache Derby ve CXF için tanımlayıcı istatistikler Tablo 1'de rapor edilmiştir. Projelerde 10-12 yıllık geliştirme verileri ve yıllar içinde 17 ila 20 geliştirici tarafından düzenlenen binlerce dosya bulunur. GitHub API’nin ana dalda geçmişe dair değişiklik verileri sağladığını ve bu nedenle araştırmamızdaki ana daldan oluşturulan dalları dikkate almadığımızı unutmayın.



# Güven Değerlerinin Hesaplanması

Wiese ve diğ. [3], bir sorunla ilgili aynı taahhüt grubunda değiştirilen her bir dosya çifti için tren ve test veri setleri hazırladılar ve güven değerlerine göre ilk 25'i seçtiler. Güven değerlerini hesaplamak için ilk önce, daha önce belirtildiği gibi bir dosya çifti olan her ilişkilendirme kuralının (r) destek değerlerini hesapladılar. Destek ve güven hesaplamaları aşağıdaki gibidir:



Destek değerleri, fi'deki diğer fj dosyalarının da değiştiği işlemlerin toplam sayısını ifade eder. Başka bir deyişle, ortak işlemlerin sayısıdır. Destek hesaplaması yalnızca dosya çiftlerini dikkate alırken ve birlik kurallarının sırasını göz önünde bulundurmazken, güven hesaplaması dosyaların sıralarının önemli olduğu ilişkilendirme kurallarını göz önünde bulundurur. Daha spesifik olarak, iki ilişkilendirme kuralı yapılabilir

bir dosya çiftinden. Örneğin, dosya çifti “a” ve “b” dosyalarından oluşuyorsa, olası ilişkilendirme kuralları a-> b ve b-> a'dır. Destek değerleri toplam ortak işlemlere eşit olduğu için aynı olsa da, güven değerleri farklıdır, çünkü toplam ortak işlemlerin toplam işlem sayısına göre bölünmüş olan toplam işlemlerin toplam sayısından elde edilen bir kesirdir. dernek kuralı. Bu nedenle, ilk dosyanın seçimi önemlidir.

Bu çalışmada, birlikte değiştirilen dosya çiftlerini elde ettik ve yukarıdaki formüllere dayanarak güven değerlerini hesapladık. Ardından, birlikte değişiklik dosya çiftlerindeki ilk dosya (a, b) en fazla değiştirilen ilk 5 dosyayı seçerek belirlenir, yani toplam dosya sayısını temel alarak, ikinci dosya bir çift olarak (b) a ve b dosyaları arasındaki güven değerine göre bulunur. A ve b arasındaki güvenirlik değeri en az 0,15 ise, bu iki dosyayı birlikte değiştirilen çiftlerden biri olarak seçtik. Bu güven değeri, antrenman süresince değişimlerin tüm değişikliklere oranını temsil edecektir.

# Özellik çıkarma

Hangi özelliklerin kullanılabileceğini belirlemek için, API'nin sağladığı bilgiler analiz edilir. Buna dayanarak, Wiese ve diğerlerinde taahhüt ve geliştirici iletişim bağlamıyla ilgili özelliklerin belirlendiğine karar verdik. [3] çalışmamız için hesaplanabilir. Yukarıda da belirtildiği gibi bu özellikler, orijinal çalışmada eş değişimleri öngörmek için en sık seçilen özellikler olarak bulunur. Özelliklerin listesi ve açıklamaları aşağıda verilmiştir:

Commit İçeriği :

* Tarih: İşlemin yapıldığı tarih. Bu sayısal ve hesaplanarak belirlenir bir taahhüt arasındaki gün içindeki zaman farkı ve ilk işlem tarihi
* Geliştirici: Adının soyadı commit yapan geliştirici.
* Değiştirilen kod satırı sayısı: değiştirilen kod satırlarının toplam sayısı taahhütte bulunan dosyaya aittir.
* Eklenen kod satırlarının sayısı: eklenen kod satırlarının toplam sayısı taahhütte bulunan dosyaya aittir.
* Silinen kod satırlarının sayısı: ait silinmiş satırların toplam sayısı taahhütte bulunduğu dosyaya.

Geliştirici İletişim İçeriği:

Bu özellikler, geliştiricilerin satır, dosyaların sütun olduğu ve matristeki her bir dizinin (d, f) geliştiricinin d dosyasında değişiklik yapıp yapmadığını gösterir. Geliştiricinin bir dosyaya katkısını gösteren ikili değerler matrisidir (0,1). Bu matris daha sonra, bir geliştiricinin başkalarıyla iletişim düzeyini temsil etmek için UCINET yazılımı kullanarak yakınlık ve kazanç özelliklerine dönüştürülür.

* Yakınlık: Bir geliştirici ile diğer geliştiriciler arasındaki geliştiricinin düzenlediği dosya sayısına göre toplam mesafenin (kenarların) sayısıdır. Bu özellik, geliştiricilerin bitişiklik matrisi, yani geliştirici-geliştirici matrisi, iletişim matrisinden çıkarılarak elde edilir.
* Arasında: Bir kaç kez geliştirici, arasındaki en kısa yollarda görünür. tüm geliştiriciler. Bu özellik ayrıca bitişik matrisin çıkarılması iletişim matrisinden geliştiriciler.

Birlikte değişikliklere sınıf etiketleri atamak için, en sık değiştirilen dosyaları (bir çiftin ilk dosyası olarak) ve seçilen dosyalarla (çiftin ikinci dosya olarak) birlikte değiştirilen dosyaları seçtik. Bu nedenle, birlikte değiştirilen dosya çiftleri, çiftin her iki öğesi de değiştirildiğinde 1 olarak etiketlenirken, çiftin yalnızca birincisi değiştirildiğinde 0 olarak etiketlenir. Sonuç olarak, veri kümeleri, taahhüt ve geliştiricilerin iletişim ağına ilişkin özelliklerden ve eş değişiklik olup olmadığını belirten sayısal etiketlerden (0 veya 1) oluşur. Lütfen farklı modellerin daha sonra her dosya çifti için ortak bir değişiklik olup olmayacağını tahmin etmek için kullanılacağını unutmayın.

# Model Yapımı

Daha önce belirtildiği gibi, tahmin modelleri dosya çiftleri için oluşturulmuştur. Çiftin ilk maddesi ilk 5 dosya listesinden biridir, ikinci madde ilk ile değişen ve en az 0,15 güvene sahip olan dosyadır. Bir dosya çifti ile ilgili her modelin eğitim ve test veri setleri aşağıdaki şekilde oluşturulmuştur:

Her ortak değişiklik dosya çifti için (a, b),

* Her iki dosyanın da (a, b) değiştirildiği komisyonların %80'i ve sadece ilk dosyanın (a) değiştirildiği komisyonların %80'i eğitim setine eklenmiştir. Taahhütlerin %20'si kalan kısmı test setine eklenmiştir.
* Eğitim seti, etiketler olarak sadece (a, b) çiftinin birlikte değiştirilmiş durumunu içerir. Değiştirilen örneklerin sayısı (1) birlikte değiştirilemeyenlerden (0) daha fazla olduğundan, eğitim setinde çoğunluk sınıfına örneklemenin altında örnekleme uyguladık. Başka bir deyişle, birlikte değiştirilmeyen örneklerin sayısı, her iki sınıftan (0 ve 1) eşit sayıda örnek bulunana kadar orijinal veri setinden rastgele seçilir.
* Test seti, etiket olarak a (örneğin, ((a, b), .. (a, x)) dosyasına karşılık gelen tüm çiftlerin birlikte değiştirilmiş durumunu içerir. Bu, tüm test çiftleri için art arda tüm çiftler için oluşturulan modelleri kullanarak öngörülerde bulunmak için yapılır.
* Rastgele Orman algoritması, modeli tüm dosya çiftleri için ayrı ayrı eğitim setinde oluşturmak için kullanılır. Analizimizi orijinal çalışmayla karşılaştırılabilir tutmak için [3] 'te kullanılan algoritmayı seçtik.
* Her model, test örnekleri üzerinde, bir dosya için, modellerin karşılık gelen dosyayı (b, .., x) a ile değiştirebileceğini belirten 0 ya da 1 sağladığı gibi tahminlerde bulunur.
* Tahminler test örneği başına değerlendirilir, yani geri çağırma ve hassasiyet, tüm çiftler boyunca doğru tahmin edilen ortak değişikliklerin sayısına dayanarak hesaplanır.

Bu işlem her dosya çifti için beş kez tekrarlanır ve model, her bir yinelemede farklı bir eğitim ve test seti kullanılarak oluşturulur. İki kümedeki komisyonların farklı olması gerektiğine ve test kümesinin, ilk çiftin birden fazla modelin test kümelerinden gelen bir çiftteki çift komisyonlarını içermemesi gerektiği not edilmelidir. Bundan kaçınmak için, her bir dosya çifti için setler üretildiğinde, eğitimin test setinden herhangi bir örnek içerip içermediğini kontrol ederiz ve öyleyse, bunları veri setinden rastgele örnekleme ile yeni örneklerle değiştirdik.

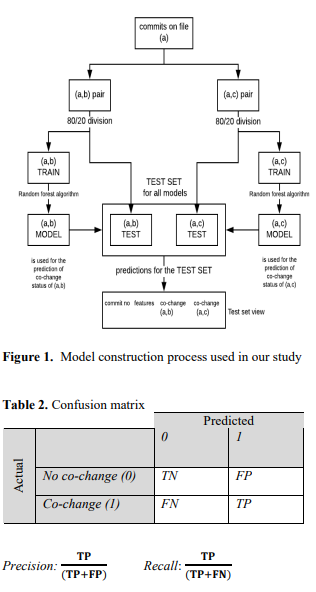
Model oluşturma işlemi Şekil 1'de gösterilmektedir. Şekil 1'e göre, “a” dosyası olduğunu ve veri kümesindeki çoğu komisyonda değiştirilen ilk 5 dosyadan biri olduğunu varsayalım. Önce “a” ile değiştirilen tüm dosyalar bulunur. Ardından, dosya çiftlerinin güven değerlerine dayanarak, içeriğimize uygun dosya çiftleri belirlenir. Şekil 1'de, bu dosyalar “b” ve “c” olarak adlandırılmıştır. İlk önce, eğitim ve test setleri% 80/20 kuralını izleyen (a, b) ve (a, c) dosya çiftleri için üretilir. İkinci olarak, eğitim setleri farklı modeller oluşturmak için ayrı tutulurken, (a, b) ve (a, c) test setleri bir test setine sahip olmak için birleştirilir. Bu aşamada, her iki kümeye de iki kez herhangi bir taahhütte bulunmamaya dikkat ettik, çünkü (a, b) test seti, c dosyasının da değiştirilebileceği bir taahhüt içerebilir. Böyle bir durumda, bu örnek (a, c) eğitim setinde olabilir. Model yapımına geçmeden önce bu davaları iki kez kontrol etmek zorunda kaldık. Son test setimiz “a” dosyasının değiştiği taahhütlere tekabül ediyor ve tüm modeller b veya c'nin de karşılık gelen taahhütte değişip değişmeyeceği konusunda bir öngörüde bulunacak.

Dahası, eğitim setlerinin (a, b) ve (a, c) örnekleri, b veya c'nin a ile birlikte değişip değişmediğini, bir dosya için test örnekleri içeriyorsa gösterir. b ve c için etiketler.

# Performans değerlendirmesi

Tahmin modellerimizin performansını ölçmek için, karışıklık matrislerini hesapladık ve [3] 'te de kullanılan hassasiyet ve hatırlamayı rapor ettik. Tablo 2, 6 ortak değişim öngörüsü için bir karışıklık matrisinin gösterimini göstermektedir. Hassasiyet, modelin ortak değişiklik tahminlerinin yüzde kaçının doğru olduğunu verirken, hatırlama ne

Aslında birlikte değiştirilen dosya çiftlerinin yüzdesi, gerçekten birlikte değişim olarak tahmin edilmektedir. Çalışmamız ve Wiese ve ark. Arasındaki farklardan biri. [3] bu önlemlerin hesaplanmasında yatar.



[3] 'te hatırlama, tüm modellerdeki doğru sınıflandırılmış örneklerin toplam sayısına göre hesaplanır. Öyleyse, (a, b) ve (a, c) için iki model, 2/3 ve 2/2 birlikte değişikliklerini doğru bir şekilde sınıflandırırsa, hatırlama [3] 'te 4/5,% 80'dir. Çalışmamızda, a. Bu beş komisyonun toplamında, bir taahhüdün üç dosyanın da değiştiğini varsayalım (a, b, c) ve modeller b ve c'yi doğru şekilde sınıflandırdı. Yani bu taahhüt için, hatırlama% 100'dür. Beş komisyondan ikisi yalnızca (a, b) değişmişse ve yalnızca bir tanesi doğru şekilde sınıflandırılmışsa, hatırlama sırasıyla% 0 ve% 100'dür. Son olarak, beş komisyondan ikisinin yalnızca (a, c) değişmesi ve modelin bir tanesini doğru şekilde sınıflandırması durumunda, geri çağırma% 0 ve% 100'dür. Bu senaryoda, modelimizin hatırlanması (0 + 100 + 100 + 100 + 0) / 5 =% 60'tır. Modelin bir taahhütte ne kadar doğru şekilde önerildiğini göreceğinden, bu değerlendirmenin geliştiriciler için pratik olarak daha faydalı olduğuna inanıyoruz. Diğer taraftan, yaklaşımımız [3] 'den daha düşük geri çağırma ve hassasiyet oranları üretmektedir. Bu nedenle, [3] den daha kötü performans görmeyi bekliyoruz. Hem yaklaşımımızı hem de Wiese ve arkadaşlarının Sonuçlar Bölümündeki yaklaşımını kullanarak geri çağırma ve hassasiyet oranlarını rapor ediyoruz.

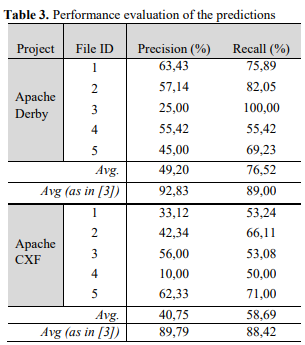
# SONUÇ

Apache Derby ve CXF projelerindeki sonuçlarımız Tablo 3'te sunulmuştur. Her proje için, ilk 5 dosya için oluşturulan modeller ve bu beş dosyanın test setlerindeki performansları hassasiyet ve hatırlama açısından rapor edilmiştir. Ortalama performans Tablo 3'te iki farklı şekilde sunulmaktadır: İlk ortalama, performans hesaplamaya dayanırken, ikinci ortalama [3] 'te kullanılan yaklaşıma dayanmaktadır. Tabloda gösterildiği gibi, hatırlama değerleri dosyadan dosyaya değişir; Bazı durumlarda, hedef dosyayla birlikte değiştirilen dosyaların% 100'ünü tahmin edebiliriz, örn. Apache Derby'deki dosya # 3, diğer durumlarda ise hedef dosya ile değiştirilen dosyaların yarısını doğru bir şekilde tahmin etmek mümkündür; dosya # 4 Derby'de ve dosya # 1'de CXF'de. Ortalama olarak, önerdiğimiz modelle birlikte% 59 ila% 76 ortak değişiklik tahmin edilebilir. Öte yandan, kesin değerler tahminlerimizin yaklaşık yarısının doğru olduğunu, yani yanlış alarmların yüksek olduğunu göstermektedir. Performans ölçüleri Wiese ve ark. [3], hem kesinlik hem de hatırlama değerlerinin sırasıyla% 92 ve% 89'a yükseldiği görülmektedir. Bu oranlar, Wiese ve diğ. [3].

Ayrıca, beş dosya ve ilişkili çiftleri için hassasiyet ve hatırlama değerlerini ayrı ayrı hesapladık, ayrıca bu çiftlerin bu beş dosyadan birinin toplam işlem sayısına göre değiştiği toplam işlem sayısının oranı. Tablo 4 ve 5, bu istatistikleri iki proje için rapor etmektedir. Apache Derby'de, bir modelin eşleştirilmiş çiftleri tahmin etmesinin daha kolay olduğunu gözlemleyebiliriz (ortalama% 93 hatırlama), ancak model aynı zamanda performansı hassasiyet açısından azaltan (ortalama% 62) yanlış pozitifler üretmektedir. Aksine, Apache CXF'de hem hassasiyet hem de hatırlama değerleri% 50'nin altındadır.

# Tartışma

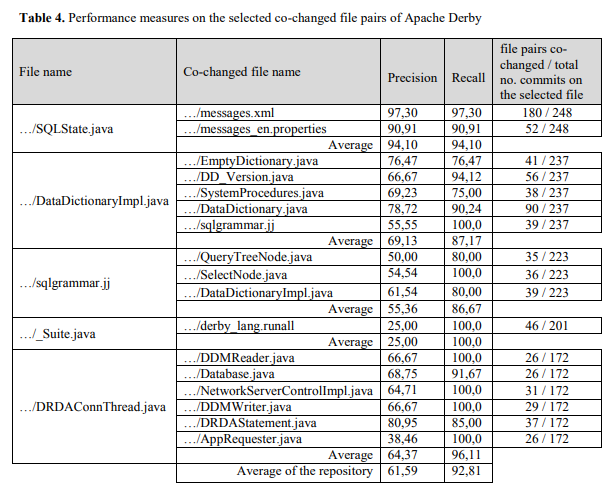
İki Apache projesinde elde edilen bulgular, 1) performans değerlendirme stratejisinin ortak değişim tahminindeki çağrıları büyük ölçüde etkilediğini göstermektedir. ve 3) bir birlikte değişiklik tahmin modelinin performansı, dosya çiftlerinde yapılan önceki değişikliklerin miktarıyla ilgili görünmüyor, ancak birlikte değiştirilen dosyaların özellikleri / özellikleri ile ilgili olabilir. Bu bölümde aşağıdaki gözlemlerin her birini tartışıyoruz.



Bölüm 4.5'te belirtildiği gibi, tahmincilerin kesinliğini ve hatırlama değerlerini hesaplamak için iki farklı yaklaşım kullandık. Tablo 3, örneğin # 1'in değiştirildiği her bir işlem için (Derby'deki SQLState.java) her bir işlem için istatistiki düzeydeki istatistikleri bildirir, çiftleri için doğru tahminlerin sayısı (messages.xml ve messages\_en.properties) hesaplamak için toplanır hassasiyet. Tablo 4'teki istatistiklere dayanarak, # 1 dosyası için, bunların 180'i messages.xml ile, bunlardan 52'i messages\_en.properties dosyaları ile olan 248 işlem yapılmıştır. Tablo 3, bu 248 taahhüt üzerindeki dosya # 1'in hassasiyetini% 63 olarak bildirmektedir. Bu, tahminlerin% 63'ünün bu iki dosyanın her birinin # 1 dosyasıyla birlikte değiştirilip değiştirilmediğini doğru bir şekilde vurguladığını gösterir. Öte yandan, Wiese ve ark. [3] 'in yaklaşımında, # 1 (SQLState.java) dosyasıyla ilişkili iki dosya çiftini tahmin etmek için kullanılan modellerin kendi başlarına çok başarılı olduğunu gözlemledik. 180'den fazla taahhüt, tahminlerin% 97'si ilk çiftin birlikte değiştiğini doğru bir şekilde vurguladı.

Genel olarak, [3] 'deki bulguları doğrulayabildiğimizi söyleyebiliriz. Öte yandan, bu oranların ayrıntı derecesine, yani taahhüt seviyesine veya dosya çifti seviyesine doğru saptığını gözlemliyoruz. Bu fark, belirli bir varlığın hedef dosya ile tek bir taahhütte değiştirilip değiştirilmeyeceğini öngörmekten ziyade, tek bir taahhütte ortaklaşa değiştirilecek olan tüm varlıkları tahmin etmenin daha zor olduğunu göstermektedir. Bizim bakış açımıza göre, birincisini başarmak ikincisinden daha başarılı. Bir taahhüt sırasında birden fazla öngörüde bulunmak için daha fazla analiz ve farklı modelleme yaklaşımları gerekebilir.

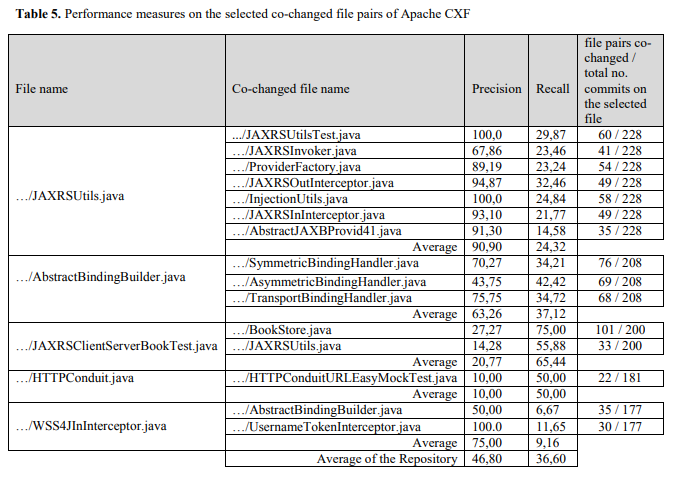
Bununla birlikte, bir modelleme ve analiz yaklaşımının her koşulda en iyi sonuçları vereceğini iddia etmiyoruz. Apache Derby'de, hem Wiese, hem de [3] 'ler ve dosya çifti seviyesindeki (Tablo 4 ve 5) bulgularımız çok yüksek hassasiyete ve hatırlama değerlerine ulaşır. Bununla birlikte, aynı yaklaşım CXF'de belirgin şekilde kötüleşti. Neredeyse tüm CXF dosya çiftleri için taahhüt ve geliştirici iletişim özelliklerini kullanan öngörücüler, rastgele bir sınıflandırıcı kadar iyi olamadı. Hassas değerler, CXF'deki geri çağırma değerlerinden nispeten daha iyidir. Ancak özellikle # 3 (JAXRSClientServerBookTest.java) ve # 4 (HTTPConduit.java) dosyalarında% 20 ile% 10 arasında kesinlik ve yanlış pozitif olduğu kesinliği görülmüştür. Öncelikle bu tür farklılıkların eğitim örneklem büyüklüğünden kaynaklanabileceğini iddia ediyoruz, çünkü bu yukarıda belirtilen çiftlerde, sırasıyla ilgili modellerin eğitim setlerinde 22 ve 33 vakanın yalnızca% 80'i mevcuttu ve rastgele orman sınıflandırıcısının daha büyük bir örneklem alması gerekebilir. daha doğru tahminler. Öte yandan, benzer bir senaryo # 5 (WSS4JInInterceptor.java) dosyasında ve çiftinde (UsernameTokenInterceptor.java) ve çiftinde (UsernameTokenInterceptor.java) benzer bir senaryo olmasına rağmen, bu çift üzerinde eğitilen model yaklaşık 30 örnekten oluşuyor,% 100 kesinliğe ulaştı. Yanlış pozitiflerin nedenleri ve bu nedenle düşük hassasiyetli oranların sebepleri daha fazla araştırılmalıdır. Öte yandan, CXF projesinde düşük geri çağırma oranlarının nedenleri, birlikte değiştirilen varlıkların özellikleri analiz edilerek açıklanabilir.



Örneğin, # 1 (JAXRSUtils.java) dosyası, birlikte değiştirilen dosya grubunda toplam yedi dosyaya sahiptir. Bu yedi çiftin her birinde eğitilen modeller oldukça yüksek hassasiyet oranları elde edebilmelerine rağmen, ilişkili test setlerinde gerçek eş değişiklik vakalarının yalnızca% 24'ünü yakalayabildiler. Bu, bazı projelerdeki ve dosya çiftlerindeki ortak değişimleri öngörmek için kullanılan özelliklerin açıklayıcı olmadığını gösterebilir. Wiese ve ark. [3] vurgulanan taahhüt ve geliştirici iletişim ağının iyi göstergeler olduğu bağlamında, her proje ve dosya çiftinin farklı bir bağımlılık göstermesi mümkün olabilir ve bu bağımlılıklar sadece konu ve kod havuzlarından farklı bağlamsal bilgilerle yakalanabilir.

# Geçerliliğe Yönelik Tehditler

Kopyalamalar hem iç hem de dış geçerlilik konularının ele alınmasına yardımcı olmaktadır [14]. Orijinal çalışmayı [3] dış ortamda kopyaladık ve aynı projeleri kullanmanın ancak farklı bir deney protokolü izleyerek takip ettiğimizi; bulgular birlikte değiştirilen dosyaların öngörülmesi bağlamında önemli ölçüde değişebilir. İç geçerlilik açısından, hem orijinal çalışmada hem de replikasyonumuzdaki bulguların sadece belirtilen koşullarda olduğunu gözlemliyoruz. Ampirik bir çalışmada kullanılan özellikler, model yapım metodolojisi ve değerlendirme kriterleri tutarsız bulgulara yol açabilir. Örneğin, birlikte değiştirilen dosya kümelerine sahip olanlar üzerinde ayrı yordayıcılar oluşturmak için ilk beş dosyayı seçtik. Modellerin eğitim için makul miktarda veriye sahip olması için geliştirme tarihi boyunca en sık düzenlenen dosyaları seçtik. Bu seçim, özellikle çok yüksek hassasiyet ve hatırlama değerleri söz konusu olduğunda elde edilen bulguları tehlikeye atabilir. Ancak, tüm dosyalar için modeller oluşturduysak, başka hiçbir dosyanın birlikte değişmediği dosyaları da bulabilirdik. Bu nedenle, bir ortak değişim tahmincisi oluşturmak için yazılım projelerindeki en aktif dosyaları seçmek zorunda kaldık.



İlk 5 dosyayla birlikte değiştirilen dosyaları filtrelemek için, güven değerlerini göz önünde bulundurduk. 0,15'lik güven oranı, bir model için belirlenen eğitimin (a, b) a dosyasındaki 100 değişiklikten oluştuğunu ve bu 100 değişikliğin 15'inin de b dosyasını içerdiğini gösterir. Bu oranı seçerken hata tahmin modellerine atıfta bulunduk, çünkü hata tahmininde, kusurlu ve hatasız örneklerin oranı kamu veri setlerinde% 15-25 civarındadır ve bir makine öğrenme sınıflandırıcısı hatalı örneklerin kabaca% 70'ini tespit edebilir.

Bu çalışmada seçtiğimiz algoritma [3] ile aynı olan rastgele ormandır. Birden fazla faktörün bulgularımız üzerindeki etkisini ortadan kaldırmak için aynı sınıflandırıcıyı kasten seçtik. Destek vektör makinesi ve lojistik regresyon ile analizler yaptık ve daha iyi performans gösterip göstermediğini kontrol ettik. Apache Derby'de, bu ek sınıflandırıcılarla elde edilen en iyi performans, hassasiyet ve hatırlama açısından% 43 ve% 55'tir. Bu nedenle rastgele orman sınıflandırıcısına bu çoğaltma çalışmasında devam ettik.

# Sonuç

Bu çalışmada, orijinal çalışmanın kavramsal bir şekilde çoğaltılmasını Wiese ve ark. [3] özellikler listesi, eğitim verileri oluşturma ve değerlendirme kriterleri ile ilgili deney protokolünü değiştirerek. Sonuçlar Apache Derby ve CXF projelerinde taahhüt ve geliştirici iletişim bilgilerini kullanarak yaklaşık% 90 hassasiyet ve geri çağırma oranlarına ulaşabileceğimizi doğrulamaktadır. Bununla birlikte, bu oranlar, performans ölçütlerinin hesaplandığı ayrıntı düzeyine büyük ölçüde bağlıdır. Bir geliştirici bir taahhüt sırasında görmek istediği ve hangi dosyaların o anda düzenlemekte olduğu dosya ile değiştirileceğini görmek istiyorsa, yordayıcı modeller dosyaların yalnızca% 59-76'sını önerebilir.

Ayrıca, bu bulguların proje özelliklerine, dosya özelliklerine ve / veya diğer yönlere bağlı olabileceğini gözlemledik. Eş değişiklikleri öngören modelleri iyileştirmek için, proje bağlamında ve dosya düzeyinde, örneğin, bağlamsal olarak diğer bağlamsal bilgilere bakmanızı öneririz. sistemdeki birçok dosyayla değiştirilen çekirdek dosyaları, genellikle bir dosya alt kümesiyle değiştirilen dosyalardan ayırt etmek. Geliştirici ilişkilerini göstermek için yakınlık ve netlik önlemleri kullanılmasına rağmen, bunları tahmin modellerine dahil etmek için dosya düzeyinde böyle bir bağımlılık oluşturmak da yararlı olabilir.