Pamukkale Univ Muh Bilim Derg, 24(5), 906-914, 2018

Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi

Pamukkale University Journal of Engineering Sciences

Yazılım hata tahmininde kullanılan metriklerin karar ağaçlarındaki bilgi kazançlarının incelenmesi ve iyileştirilmesi

Analyzing and improving information gain of metrics used in software defect prediction in decision trees

İbrahim Berkan AYDİLEK[1*]

[1]Bilgisayar Mühendisliği Bölümü, Mühendislik Fakültesi, Harran Üniversitesi, Şanlıurfa, Türkiye.
berkanaydilek@harran.edu.tr

Geliş Tarihi/Received: 07.02.2018, Kabul Tarihi/Accepted: 06.04.2018 doi: 10.5505/pajes.2018.93584
* Yazışılan yazar/Corresponding author Araştırma Makalesi/Research Article

Öz

Yazılım kalitesinin somut bir şekilde ölçülebilmesi için kullanılan sayısal yazılım metrikleri içinde bilinen ve yaygın şekilde kullanılanlar arasında McCabe ve Halstead yöntem-seviye metrikleri bulunmaktadır. Yazılım hata tahmini, geliştirilecek olan yazılımda bulunan alt modüllerin hangisi veya hangilerinin daha çok hataya meyilli olabileceğini konusunda öngörüde bulunabilmektedir. Böylece işgücü ve zaman konusundaki kayıpların önüne geçilebilmektedir. Yazılım hata tahmini için kullanılan veri kümelerinde, hata var sınıflı kayıt sayısı, hata yok sınıflı kayıt sayısına göre daha az sayıda olabildiğinden bu veri kümeleri genellikle dengeli olmayan bir sınıf dağılımına sahip olmakta ve makine öğrenme yöntemlerinin sonuçlarını olumsuz etkilemektedir. Bilgi kazancı, karar ağaçları ve karar ağacı temeline dayanan kural sınıflayıcı, nitelik seçimi gibi algoritma ve yöntemlerde kullanılmaktadır. Bu çalışmada, yazılım hata tahmini için önemli bilgiler sunan yazılım metrikleri incelenmiş, NASA'nın PROMISE yazılım veri deposundan CM1, JM1, KC1 ve PC1 veri kümeleri sentetik veri artırım Smote algoritması ile daha dengeli hale getirilerek bilgi kazancı yönünden iyileştirilmiştir. Sonuçta karar ağaçlarında sınıflama başarı performansı daha yüksek yazılım hata tahmini veri kümeleri ve bilgi kazanç oranı yükseltilmiş yazılım metrik değerleri elde edilmiştir.

Anahtar kelimeler: Yazılım hata tahmini, Karar ağaçları, Bilgi kazanç oranı

Abstract

McCabe and Halstead method-level metrics are among the well-known and widely used quantitative software metrics are used to measure software quality in a concrete way. Software defect prediction can guess which or which of the sub-modules in the software to be developed may be more prone to defect. Thus, loss of labor and time can be avoided. The datasets which are used for software defect prediction, usually have an unbalanced class distribution, since the number of records with defective class can be fewer than the number of records with not defective class and this situation adversely affect the results of the machine learning methods. Information gain is employed in decision trees and decision tree based rule classifier and attribute selection methods. In this study, software metrics that provide important information for software defect prediction have been investigated and CM1, JM1, KC1 and PC1 datasets of NASA's PROMISE software repository have been balanced with the synthetic data over-sampling Smote algorithm and improved in terms of information gain. As a result, the software defect prediction datasets with higher classification success performance and the software metrics with increased information gain ratio are obtained in the decision trees.

Keywords: Software defect prediction, Decision trees, Information gain ratio

1 Giriş

Gün geçtikçe birçok kritik işlem ve süreçlerin yazılımlar sayesinde gerçekleştirilmesinden dolayı yazılım kalitesi, üzerinde önemle durulması gereken bir çalışma alanı haline gelmiştir. Yazılım kalitesini ifade eden ISO 25010 yazılım kalite güvence faaliyetleri ve çalışmalarında bir yazılım ürününün temelde kalitesi: fonksiyonel uygunluk, performans verimliliği, uygunluk, kullanılabilirlik, güvenilirlik, güvenlik, idame ve taşınabilirlik gibi kalite sınıf kategorileri açısından değerlendirilebilmektedir. Kaliteli bir yazılım geliştirme süreci gereksinimlerin analizi, tasarım, kodlama, uygulama, test ve entegrasyon, bakım ve destek gibi süreçlerden oluşmaktadır. Yazılım hataları proje, süreç ve ürün açısından sınıflandırılabilir, bu çalışmada yazılım ürününe yönelik hataların tespitine yönelik iyileştirmeler ve incelemeler yapılmaktadır. Yazılım kalite modelleri, kalite tahmini ve değerlendirme için bazı tanımlar sunar. Yazılım kalite güvence faaliyetleri ve çalışmaları, muhtemel hatalı yazılım modüllerini belirlemek için kullanılabilir. Yazılım kalite değerlendirme faaliyetleri, yazılım geliştirici ekibe potansiyel yazılım kusurlarını tespit ve takip imkânı tanımakla birlikte ayrıca gelecekte daha da kaliteli program geliştirebilmelerine yardımcı olmaktadır. Ayrıca yazılımın bir sonraki sürümündeki modüllerin kusurlu olup olmayacağı ile ilgili bazı öngörülerde bulunabilmektedir. Bugüne kadar, gerçek hayatta kullanılan yazılımların kalitesini ölçmek ve değerlendirmek amacıyla birçok yazılım kalite ölçüm tekniği önerilmiştir. Bunlardan biri kural tabanlı yazılım kalitesi ölçüm yaklaşımıdır. Kural tabanlı sistemler bir veya daha fazla kuraldan oluşmaktadır. Mevcut gerçek durumlara göre karar verme, şart oluşturma ve karar alma konularında oldukça etkili bir model sunmaktadır [1]. Yazılım geliştirme sürecinde hata tespiti ve hata düzeltme önemli bir yer tutmaktadır. Bir yazılımdaki hatalar, yazılım gerekliliğini, koşullarını ve son kullanıcıya ait bazı beklentilerin yerine getirilmemesine sebep olmaktadır. Genellikle yazılımlar aritmetik, mantık, söz dizimi veya takım çalışmasından kaynaklanan hataların oluşmasına meyillidirler. Yazılım hatalarını tanımlama ve düzeltme çoğu kez fazlasıyla zaman alan bir süreç olmaktadır. Yazılımlar günlük yaşamda özellikle bilgisayar sistemlerini kullanan topluluklarda oldukça önemli görevler üstlenebilmektedir. Bu sebeple, yazılım hataları, küçük veya ciddi boyutlarda büyük sorunlar meydana getirebilmektedir. Önem arz eden yazılım hata tahmin sürecinin uygun olarak ele alınması ve olası problemlerin en

baştan çözülebilmesi için gün geçtikçe araştırmalar artarak devam etmektedir [4].

Yazılım hata tahmini araştırmaları yapılan veri kümeleri doğası gereği içeriğinde farklı sınıflara ait kayıt sayılarının eşit olmadığı bir yapıda yani dengeli olmayan bir sınıf dağılımı içerecek şekilde olabilmektedir. Bu türdeki veri kümelerine literatürde dengeli olmayan veri, aksine sınıf dağılımı eşit olan veri kümelerine ise dengeli veri adı verilmektedir. Dengeli olmayan bu sınıf dağılımı, makine öğrenmesinde ve veri madenciliğinde kullanılan algoritmaların peşin hükümlü ve yanlış sonuçlar vermesine neden olmakta ayrıca yapılan çalışmanın sonuçlarının da doğru bir şekilde elde edilmesini engelleyebilmektedir. Bu çalışmada yazılım tahmininde kullanılan veri kümelerinde görülen bu problemin giderilebilmesi için ilgili veri kümesi önce sentetik veri artırım Smote algoritması [14] ile daha dengeli hale getirilmiş daha sonra önceki çalışmalarda başarılı görülen kural tabanlı karar ağacı algoritmaları ile deneysel olarak değerlendirilmiştir. Smote algoritması azınlık (minority) sınıfına ait kayıtların sayısını yapay olarak artırmaktadır. Bu nedenle, dengeli olmayan veri küme problemine karşı bir çözüm getirebilmek için tercih edilerek kullanılmıştır.

Kural tabanlı sınıflayıcılar, bir veri kümesi içindeki kayıtları doğru şekilde sınıflayabilen eğer-ise şeklindeki kurallar ile oluşturulan bir yaklaşımdır. Kurallar birbirinden bağımsız kurallardan oluşabildiği gibi sıralı bir önceliğe sahip kural listesi şeklinde de üretilebilmektedir. Literatürde Part [34], NNge [35],[36] ve Prism [37] gibi farklı kural tabanlı sınıflayıcılar bulunmaktadır. Çalışmamızda C4.5 [16] ve Part algoritmasının kullanılmasının sebebi, bu algoritmaların entropi ve bilgi kazancı ile kural çıkarımı ve sınıflama yapan, karar ağacı tabanlı bir algoritma yapısında olmasından kaynaklanmaktadır. Bu sayede iyileştirilmiş bilgi kazancının deneysel sonuçları da elde edilebilecektir.

2 Literatür özeti ve ilgili çalışmalar

Koru ve diğ., [6] yazılım modüllerinin satır sayısı boyutuna göre hata tahmini yapmak için alt kümelere ayırdılar ve her alt kümedeki sınıf seviyesinde minimum, maksimum, toplam ve ortalama değerlere göre C4.5 karar ağacı kullanarak bir tahmin modeli önerdiler.

Pelayo ve diğ. [12] yazılım hata tahmininde kullanılan veri kümelerine Smote algoritmasını aynı anda farklı yüzde değerler ile hem azınlık sınıfı artıracak hem de çoğunluk sınıfı azaltacak şekilde uyguladılar. C4.5 karar ağacı algoritmasının, G-ortalama sınıflama başarısını ölçtüler. Bu şekilde sınıflama başarısının artırıldığını gösterdiler. Bu çalışmada çoğunluk sınıfı verileri veri kümesinden çıkarılmıştır. Bizim çalışmamızda var olan mevcut çoğunluk sınıfın azaltılması yerine sadece azınlık sınıfı çoğaltılmıştır. Bu şekilde elde edilen yeni veri kümesi yazılım metrikleri bilgi kazancı yönünden incelenerek C4.5 ve PART [18] gibi algoritmalardaki etkileri araştırılmıştır.

Menzies ve diğ. [9] yazılım hata tahmini için kartil grafiği kullandı. Veri madenciliği yöntemleri olan OneR [41], C4.5 ve NaiveBayes [42] kullanarak dinamik olmayan kod özellikleri ile bir hata tahmin yöntemi geliştirdiler.

Lessmann ve diğ. [10] yazılım hata tahmini için 22 adet sınıflayıcı ve 10 adet NASA tarafından oluşturulmuş halka açık yazılım hata tahmini veri kümesi kullanarak sınıflama başarı karşılaştırması yaptı. Elde edilen sonuçlarda metrik tabanlı sınıflayıcıların başarılı sonuçlar verdiği fakat sınıflayıcıların başarıları açısından kıyaslamada çok belirgin farkların oluşmadığını ifade ettiler. Bizim çalışmamızda veri kümelerine Smote algoritması ile veri önişleme süreci uygulanmıştır.

Catal ve diğ. [3] 2000 yılı öncesi ve sonrasında yazılım hata tahmini ile ilgili yapılan çalışmaları kapsayan sistematik bir inceleme makalesi yazdılar. Buna göre en fazla kullanılan yöntemlerin %59 ile makine öğrenme, %60 ile yöntem-seviye metrikleri olduğu ve konu ile ilgili araştırmaların 2005'li yıllardan sonra %52 oranında herkese açık veri kümeleri üzerinden ivmelenerek arttığını belirttiler. Bizim çalışmamız bu sistematik inceleme makalesindeki gelişmeler ile uyuşmaktadır.

Gupta ve diğ. [1] yazılım kalitesini değerlendiren bir model oluşturabilmek için karşılaştırmalı olarak var olan mevcut yöntemleri kullanarak performanslarını kıyasladılar. Test edilen yöntemler: yapay sinir ağları, durum tabanlı kurallar, regresyon ağacı [43], kural tabanlı sistemler, çoklu lineer regresyon ve bulanık sistemlerdir. Bu karşılaştırmalar sonucunda bulanık ve kural tabanlı sistemlerin yazılım hata tahmini için iyi sonuçlar verdiğini belirttiler. Bu çalışmanın avantajı yazılım hata tahmini için kural tabanlı sistemlerin iyi model olduğunun gösterilmesidir. Bizim çalışmamız bu doğrultuda karar ağaçları temelli kural çıkarım sürecini iyileştirmektedir.

Hall ve diğ. [29] 2000 ile 2010 yılları arasında yazılım mühendisliği alanında program kaynak kodlarında hata tahmini çalışmaları ile ilgili derleme bir makale yayınladılar. Daha önce konu ilgili yapılan inceleme makalelerde görülen eksiklikleri gidermek için sistematik bir yaklaşım önerdiler. Bu anlamda 208 adet makale incelendi. İçerik olarak büyük kod boyutları içeren sistemlerin özellikle Eclipse üzerine yapılan çalışmaların daha başarılı olduğu, işlemlerin daha çok dinamik olmayan kod verileri üzerinden yani metrikler üzerinden yapıldığı ve dengesiz sınıf üzerinde C4.5'in iyi olmadığı NaiveBayes ve lojistik regresyon yöntemlerinin iyi sonuçlar verdiği söylenmiştir. Bu çalışmada karar ağacı C4.5 algoritması ile dengeli veri ile kullanımın iyi sonuçlar vermediği belirtilmiştir. Bizim çalışmamız C4.5 gibi bilgi kazancı kullanan algoritmaların dengeli veri ile kullanılması üzerine yapılmış ve elde edilen sonuçların daha iyi olduğu gösterilmektedir.

Wang ve diğ. [30] dengesiz olan yazılım hata tahmini veri kümeleri için dengesiz öğrenme yöntemlerinden; eşik değer taşıma ve birlik algoritmalarını incelediler. Çalışmada verileri dengeli hale getirmek için uygun parametre değer ve yüzdelerini belirleyecek bir birlik öğrenme modeli önerdiler. Buna göre AdaBoost.NC [44] yönteminin iyi olduğu gösterildi. Bu çalışmada Smote algoritması yükseltici öğrenme yöntemiyle (SmoteBoost) hibrit bir şekilde kullanılmış ve sonuçlar NaiveBayes ve Random Forest [45] algoritmaları ile karşılaştırılmıştır.

Paramshetti ve diğ. [28] yazılım hata tahmini üzerine çalışılan makine öğrenme teknikleri ile ilgili bir araştırma makalesi yayımladılar. Sınıflama, kümeleme, birliktelik analizi ve hibrit çalışmalar ile ilgili özet bilgiler verilerek yapay sinir ağları, destek vektör makineleri, karar ağaçları, birliktelik kuralları ve kümeleme ile ilgili avantaj ve sınırlamaları liste halinde sundular. Buna göre karar ağaçlarının diğerlerine göre daha doğru sonuç verdiğini fakat tasarımının kompleks olduğunu ifade ettiler.

Aleem ve diğ. [31] denetimli ve denetimsiz makine öğrenmesi öğrenme algoritmalarını NASA veri kümesi üzerinde test etmiştir. 15 veri kümesi üzerinde 11 yöntem sınanarak

F-ölçütü, mutlak hata ve doğruluk kıyaslamaları yapılmıştır. Buna göre destek vektör makineleri, çok katmanlı yapay sinir ağları ve yerine koyarak örnekleme (Bagging) öğrenme yöntemlerinin iyi olduğu gösterilmiştir.

Pal ve diğ. [4] yazılım hata tahmini veri kümeleri üzerinde birleştirilmiş kümeleme yaklaşımı ile özellik seçimi yaptılar. Yapılan bu özellik seçiminden sonra Smote uygulanmış ve uygulanmamış veri kümeleri ile olasılık tabanlı sinir ağları (PNN) [46] sınıflayıcısı üzerinde testler yaptılar. Sonuçlarda özellik seçimi yapılmış ve Smote algoritmasının uygulandığı PNN uygulamalarının, özellik seçimi yapılmamış çalışmalardan daha iyi olduğu gösterildi. Bu çalışmanın avantajı Smote algoritması ile birlikte kümeleme tabanlı özellik seçme işlemlerinin yapılmasıdır.

Magal ve diğ. [13] yazılım hata tahmini veri kümeleri üzerinde korelasyon ve bilgi kazancı temelli CFS, Info Gain ve Gain Ratio özellik seçme yöntemleri uygulanarak geliştirilmiş Random Forest algoritması ile sınıflama yaptı. Önerilen CFS özellik seçme yöntemli sınıflama algoritmasının normal Random Forest algoritmasına göre yazılım hata tahmininde daha iyi olduğunu gösterildi. Bu çalışmanın eksikliği veri kümelerinin dengeli sınıf dağılımlı hale getirilmemesi olarak görülmektedir.

Prasad ve diğ. [32] literatürde bulunan farklı yazılım metrikleri ile yapılan veri madenciliği ve makine öğrenme yöntemleri hakkında araştırma sundular. Çalışmanın amacı yazılım geliştiricilere var olan mevcut değişik yazılım metriklerinin yazılım hatasına olan etkilerinin ortaya çıkarılmasına ve daha kaliteli yazılımların geliştirilmesine yardımcı olmaktır. Bu çalışmada bahsedilen metrikler ve makine öğrenmesi yöntemlerinin başarılı olması bizim bu makalede yaptığımız çalışmanın amacını ortaya çıkarması bakımından olumlu etki sağlamaktadır.

Japkowicz ve diğ. [38] sınıf dengesizliği problemi ile başa çıkmak için daha önce önerilen rastgele, veri artırımı, azalımı, maliyet düzenleme vb. yöntemler ile ilgili araştırmalar yaparak, eğitim kümesi büyüklüğü ve sınıf dengesizliği seviyesine bağlı olarak sınıflama başarısının değişkenliğini incelemişlerdir. Elde edilen genel sonuçlarda dengesiz sınıf dağılımının karar ağaçları sonuçlarının etkilediği gibi diğer sinir ağları ve destek vektör makinesi yöntemler üzerinde de etkilerinin olduğu ortaya koyulmuştur.

Batista ve diğ. [39] Smote + Tomek ve Smote + ENN adını verdikleri yöntemler ile sentetik veri artırımı ve sentetik veri azalımı yaparak 13 veri kümesi üzerinde öğrenme performansını ölçen testler yaptılar. Elde edilen genel sonuçlarda sentetik veri artırım yöntemlerinin sentetik veri azalımı yapan yöntemlere göre daha üstün olduğu gösterilmiştir.

He ve diğ. [40] dengesiz öğrenme problemi için ADASYN: adaptif (uyabilen) sentetik veri örnekleme yaklaşımı geliştirdiler. Farklı azınlık sınıfı yüzde seviyelerine göre öğrenme sürecinde yaşanan zorluklar için ağırlıklı bir dağılım tercih edilmiştir. Böylece sınıf dengesizliği sonucu oluşan eğilim azaltılmış ve zor örnekleri de kapsayacak şekilde sınıflayıcının sınırlarının artırıldığı gösterilmiştir.

Catal ve diğ. [47] veri kümesi boyutu, yazılım metrikleri ve özellik seçim tekniğinin yazılım hata tahminine olan etkilerini inceleyen bir çalışma yaptı. NASA veri kümesi ile 7 farklı şekilde deneyler yapıldı. Küçük boyutlu veri kümeleri için NaiveBayes, büyük boyutlu veri kümelerinde ise Random Forest algoritmasının başarılı olduğu, yöntem-seviye metrikleri ile yapay bağışıklık tanıma (AIRS2Parallel) yönteminin başarılı olduğu gösterilmiştir.

Catal ve diğ. [48] yazılım test süreçlerini iyileştirmek yapay bağışıklık tanıma sistemlerini temel alan bir hata tahmin modeli önerdi. Bu model ile Halstead, McCabe yöntem-seviye ve diğer bazı sınıf-seviye metriklerini kullandılar elde edilen sonuçlarda yapılan çalışmanın C4.5 algoritmasına göre daha yüksek performanslı olduğu ifade edildi.

Tomar ve diğ. [26] dengeli olmayan yazılım hata tahmini veri kümelerinde, hatalı sınıfın sınıflama maliyetinin, hatalı olmayan sınıfa göre daha yüksek olduğunu belirtti. Bu yüzden dengesiz sınıf problemini göz önüne alan ağırlıklı en küçük kareler ikiz destek vektör makinesini (WLSTSVM) önerdiler. Araştırmada performans ölçütü olarak G-ortalama ve F-ölçütünü kullanıldı. Sınıflama açısından maliyeti yüksek olan hatalı olma ile maliyeti az olan hatalı olmama sınıf etiketleri arasındaki dengeyi sağlayacak bir sınıflama yaklaşımı geliştirildi. Bu çalışmada yazılım hata tahmini çalışmalarında dengeli sınıf dağılımına sahip veri kümelerinin daha başarılı sonuçlar verdiği gösterilmiştir. Bizim çalışmamızın farkı veri kümesi Smote algoritması ile dengeli hale getirilmiş ve bilgi kazancı, kazanç oranı yönünden incelenmiş ve iyileştirilmiştir.

Genel olarak literatür çalışmalarına bakıldığında, yazılım hata tahmini çalışmalarının popüler ve üzerinde oldukça çalışılan bir konu olduğu, çalışmalarda genellikle Halstead, McCabe yöntem-seviye yazılım metriklerinin ve herkese açık olan NASA yazılım veri kümelerinin kullanıldığı görülmektedir. Yapılan bu çalışmalarda kural tabanlı sınıflayıcı ve karar ağaçlarının performanslarının ön plana çıktığı görülmekte ve bu yaklaşımların eksiklik ve problemlerinin giderilmesi için bir çalışma yapılması gerekliliğinin ortaya çıktığı görülmektedir. Bizim çalışmamızda Smote algoritması ile dengeli veri haline getirilen yazılım veri kümeleri karar ağacı ve kural tabanlı sınıflayıcıların sınıflama üzerindeki etkileri incelenmiş ve başarısı artırılmıştır.

3 Yazılım hata tahmini veri kümeleri ve yazılım metrikleri

Yazılım hata tahmini çalışmaları; programcıların, geliştirme sürecinde hataya meyilli olabilecek kodlara odaklanmalarını sağlayarak, yazılım kalitesinin yükseltilmesine ve kaynakların daha etkin kullanılmasına yardımcı olmaktadır [2]. Yazılım hata tahmini araştırmaları genel olarak metrikler, veri kümeleri ve yöntemler üzerinde olarak 3 farklı kategori başlığı altında toplanmaktadır. Bu çalışmada yöntem-seviye yazılım metriklerini içeren yazılım hata tahmini veri kümeleri üzerinde makine öğrenme yöntemleri uygulanarak deneysel sonuçlar elde edilmektedir. Metrikler: yöntem-seviyesi, sınıf-seviyesi, bileşen-seviyesi, dosya-seviyesi, işlem-seviyesi, nicel-seviye gibi farklı kategorilerde olmaktadır. Yazılım metrikleri kısaca yazılım kalitesinin somut şekilde ifade edilmesini sağlayan, ölçülebilen değer veya değer kümeleridir. Yazılım hata tahmini araştırma alanında, yöntem-seviye yazılım metrikleri, en yaygın olarak kullanılan metriklerdir. Bu metrikler makine öğrenme algoritmaları ile popüler olarak kullanılmaktadır [3]. Veri kümesi, bir konu ile ilgili ham verilerin tutulduğu yapıdadır. Herkese açık yazılım hata tahmini veri kümesi depoları gün geçtikçe artmaktadır. Bunlardan birisi uzay araştırmaları yapan NASA kurumuna ait PROMISE veri kümesi deposudur [5]. Bu yazılım veri kümeleri, hata tahmin modellerinin oluşturulmasına ve deneysel sonuçların elde edilebilmesini sağlamaktadır. Ayrıca bu veri kümeleri

araştırmacılar arasında, yöntem ve sonuçlar üzerinde tartışma ve gelişmelerin yapılmasını imkan sağlamaktadır. Tablo 1'de NASA tarafından geliştirilmiş bazı yazılımlara ait ve yaygın olarak kullanılan PROMISE veri deposuna ait veri kümelerinden olan CM1, JM1, KC1, PC1 görülmektedir. Kullanılan veri kümelerinde bir takım metrik ölçüm değerleri ve değişkenler vardır. Veri kümesinin her kaydında ilgili yazılım modülünde bilenen veya geri bildirilmiş hata olma veya hata olmama durumunu ifade eden bir sınıf etiketi bulunmaktadır. Hatanın var olma durumu, ilgili yazılım modülünde veya diğer ilişkili modüllerde değişiklik veya güncellemelerin yapılması gerektiğini göstermektedir [6]. Yöntem-seviye metriklerinden bazıları Halstead ve McCabe tarafından 1970'li yıllara önerilmesine rağmen günümüzde hala popüler bir şekilde kullanılmaya devam edilmektedir. Yöntem-seviye metrikleri yapısal veya aslında kendine özgü metrikleri olan nesneye yönelik programlama ile geliştirilmiş yazılımlar için de kullanılabilir. Yöntem-seviye metrikleri ile hataya eğilimli modül önceden tahmin edildiğinde muhtemelen sistem veya alan testlerinde ilgili modülde hata olabileceği anlamına gelmektedir. Tablo 1'deki NASA tarafından geliştirilmiş bazı yazılımlara ait CM1, JM1, KC1 ve PC1 yazılım hata tahmini veri kümeleri üzerinde değerlendirmeler yapılmıştır [3],[4],[7],[8],[10],[33].

Veri kümesindeki öznitelikler yani yöntem-seviye metrikleri Tablo 2'de görülmektedir [3],[4],[7],[8],[10]. Veri kümeleri ve metrikler ile ilgili ayrıntılı bilgi [5],[9]'da bulunabilir. Yazılımda; çevrimsel (v(g)), tasarım (iv(g)) ve esas (ev(g)) gibi karmaşıklık metrikleri önemli bilgiler vermektedir. Karmaşıklık, örneğin v(g), kısaca bir yazılım modülündeki karar mantık ölçüsünü vermektedir. Program kontrol akış diyagramlarında bulunan kenar ve düğüm sayısına göre hesaplanmakta ve kodun karmaşıklığı hakkında bilgiler sunmaktadır [25]. Kenarlar, düğümlerdeki kontrol ifadelerini, düğümler ise işlem yapılan durum veya ifadeleri göstermektedir. v(g)=kenar-düğüm+2 olarak tespit edilmektedir. Tekil operatör (uniq_Op) ve tekil işlenen (uniq_Opnd) metrikleri de bir yazılım modülünün hataya meyilli olup olmasına dair önemli bilgiler sunmaktadır. Örneğin, "i>0", i değişkenin sıfırdan büyük olduğunu ifade eden bir kod satırında ">" operatör, "i" ve "0" ise işlenen kısımları oluşturmaktadır.

Tablo 1: Yazılım hata tahmini veri kümeleri.

Adı	Kayıt (Modül)	Özellik (Metrik)	Dil	Boyut	Yazılım	Hatalı (evet/hayır)
CM1	498	21	C	14k	Uzay aracı aletleri	49/449
JM1	10885	21	C	447k	Yer sistemi	2106/8779
KC1	2109	21	C++	42k	Yer depo yönetimi	326/1783
PC1	1109	21	C	25k	Dünya yörüngesi uçuş	77/1032

Tablo 2: Yöntem-seviye metrikleri.

Metrik	Açıklama
McCabe	
loc	Toplam satır sayısı
v(g)	Çevrimsel karmaşıklığı
ev(g)	Esas karmaşıklığı
iv(g)	Tasarım karmaşıklığı
Halstead	N1 = operatör sayısı N2 = işlenen sayısı u1 = tekil operatör sayısı u2 = tekil işlenen sayısı
N	Uzunluk = N1+N2
V	Hacim = $N*\log_2(u1+u2)$
L	Seviye = V^*/V, $V^*= (2+u2^*)\log_2(2+u2^*)$
D	Zorluk = 1/L
I	Kabiliyet = L^**V, $L^*=(2/u1)*(u2/N2)$
E	Efor = V/L^*
B	Hata sayısı
T	Zaman = E/18 saniye
lOCode	Kod satırı sayısı
lOComment	Yorum satırı sayısı
lOBlank	Boş satır sayısı
lOCodeAndComment	Yorumlu kod satır sayısı
uniq_Op	Tekil operatör
uniq_Opnd	Tekil işlenen
total_Op	Toplam operatör sayısı
total_Opnd	Toplam işlenen sayısı
branchCount	Akış grafiğindeki dal sayısı
Hata (Sınıf)	Evet / Hayır

Makine öğrenmesinde sınıflama yaklaşımı, modülleri bir takım yazılım metrik veya kod özelliklerine göre hataya meyilli veya hataya meyilli değil şeklinde sınıflara ayırır [10]. Yazılımlardaki hatalar genellikle kaynak kodlar sebebiyle meydana gelmektedir. Belli bir hata, yanlış çıktılara ve düşük kaliteli yazılım ürünlerine sebebiyet vermektedir. Nihai sonuç olarak, hatalı yazılım modülleri, yüksek geliştirme, bakım maliyetlerinden ve son kullanıcı memnuniyetsizliğinden sorumlu olmaktadır [13].

4 Dengeli olmayan verinin Smote algoritması ile bilgi kazancının yükseltilmesi

Bir veri kümesindeki hatasız olan sınıf oranı baskın yani çoğunluk (majority) sınıf durumunda ise sınıflayıcı tahminlerinde büyük oranda hatasız yazılım modülü tahmininde bulunacaktır. Yani hata var/hata yok oranının 1/99 olduğu bir veri kümesinde sınıflayıcı model tüm tahminlerinde hata yok diye tahminde bulunur ise sınıflayıcı başarısı %99 olmaktadır. Bu durum makine öğrenmesinde dengesiz sınıf veri problemini oluşturmaktadır. Bir veri kümesinde bir sınıf etiketi sayısı diğerine göre çoğunluk (majority) ise sınıflayıcı bazen azınlık (minority) sınıf etiketine sahip kayıtları tahmin edemez. Sınıflayıcı model eğitilmiş verideki çok olan sınıfa doğru eğilim gösterir. Diğer bir ifade ile az olan sınıf değeri tahmini yapılırken büyük oranda hatalı tahminler yapılmış olunur. Sınıflayıcı her zaman yüksek sayıda olan sınıfı tahmin etse bile yüksek derecede başarılı doğruluk oranına sahip olmakta fakat gerçekte kullanışsız olmaktadır [12].

Smote algoritması, dengeli olmayan veri kümelerinde azınlıkta olan sınıf değerine sahip kayıt sayısını sentetik olarak artırılmasını sağlayarak dengeli veri kümesi oluşturabilen bir yaklaşımdır [14]. Smote algoritması genel parametreleri ile aşağıdaki çalışma adımlarına göre işlem yapar.

1. Azınlık sınıfına ait bir örneğe, azınlık sınıfı içinde Öklid mesafesine göre en yakın 5 adet örnek veri bulunur,
2. Yakın örnekler içinden rastgele 1 komşu seçilir,
3. Rastgele seçilmiş komşu 0 ile 1 arasındaki rastgele bir sayı ile çarpılır,
4. Çarpım sonucu elde edilen değer ile ilk örnek toplanarak yeni sentetik örnek oluşturulur.

Smote algoritması bu şekilde sentetik olarak oluşturduğu örnekler ile azınlık sınıf sayısını çoğunluk sınıf değerine eşitleyerek sınıf örneği sayısı bakımında dengeli bir veri kümesi elde etmektedir.

Tablo 3'te sınıflama başarısı ölçümü için kullanılan karışıklık matrisi görülmektedir [19]. Tablodaki,

TP (True Positive) : Sınıflayıcı model tarafından başarılı şekilde tahmin edilen pozitif örneklerin sayısını,

FP (False Positive): Sınıflayıcı model tarafından başarısız şekilde tahmin edilen negatif örneklerin sayısını,

FN (False Negative): Sınıflayıcı model tarafından başarısız şekilde tahmin edilen pozitif örneklerin sayısını,

TN (True Negative): Sınıflayıcı model tarafından başarılı şekilde tahmin edilen negatif örneklerin sayısını göstermektedir [50].

Tablo 3: Karışıklık matrisi.

		Tahmin edilen sınıf	
		Sınıf 1	Sınıf 2
Gerçek Sınıf	Sınıf 1	TP	FN
	Sınıf 2	FP	TN

Tablodaki değerlere göre hassasiyet (precision)= P/(TP+FP) ve anma (recall)=TP/(TP+FN) şeklinde hesaplanmaktadır. F-ölçütü değeri, Denklem 1'deki gibi hesaplanır. *B* parametresi genellikle 1 olarak alındığından bu çalışmada da 1 olarak kullanılmıştır. F-ölçütü, hassasiyet ve anma gibi sınıflayıcı başarı ölçütlerinin kombinasyonudur. Eğer bu iki ölçüt aynı anda iyi olursa F-ölçütü değeri iyi olmaktadır. Böylece dengesiz bir veri kümesi için sınıflayıcı değerlendirme kıstası olarak kullanılabilmektedir. G-ortalama başarı ölçütü ise Denklem 2'deki gibi hesaplanmaktadır. Bu ölçüt, sınıflama algoritma başarısının iki sınıf arasındaki dengeli olma durumunu göstermektedir. Her iki ölçütte daha büyük bir değer daha başarılı bir sınıflayıcıyı ifade etmektedir [20]. F-ölçütü değeri, dengesiz veri kümesinin azınlık sınıfına ait sınıflama başarısını değerlendirmek için kullanılırken, G-ortalama ise tüm sınıfların geneli için değerlendirme yapmaktadır [21],[22].

$$F-\text{ölçütü} = \frac{(1+B^2)x\ anma\ x\ hassasiyet}{(B^2\ x\ anma) + hassasiyet}$$
$$anma, duyarlılık\ (recall, sensitivity) = TP/TP + FN \quad (1)$$
$$hassasiyet\ (precision) = TP/TP + FP$$

$$G-ortalama = \sqrt{anma\ x\ özgüllük}$$
$$özgüllük\ (specificity) = TN/TN + FP \quad (2)$$

Sınıflama çalışmalarında karar ağaçları sıklıkla kullanılmaktadır. Bir karar ağacı sebeplerin haritasını sunarak bir sonucun nasıl elde edildiğini açıklayabilmektedir. Özellik değerlerinden sınıf değerine kadar giden yolu gösteren bir ters ağaç yapısı şeklindedir [11]. Fakat bu ağacı oluşturmak zahmetli ve zor bir süreç olabilmektedir. Ağacın en iyi yapısı bilgi kazancı ve/veya kazanç oranı hesaplamaları ile oluşturabilmektedir.

Entropi, 1872'de Boltzmann tarafından önerilmiş ve bir sistemin belirsizlik ölçütünün tanımı olarak kullanılmaktadır. Karar ağaçlarında entropi, Denklem 3'teki gibi hesaplanmakta ve bilgi kazancının hesaplamasının temelini oluşturmaktadır. Bilgi kazancı yüksek olan öznitelik karar ağacında değerlidir ve ters ağacın kökünü oluşturmak için kullanılmaktadır. Denklemdeki *I(p,n)*, veri kümesindeki bir örneği sınıflayabilmek için gerekli ortalama bilgi miktarını [50], *p* pozitif sınıf, *n* negatif sınıf sayısını, *v* ise *A* özelliğinde bulunan farklı değer sayısını göstermektedir.

$$I(p,n) = -\frac{p}{p+n}log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n}log_2\left(\frac{n}{p+n}\right)$$
$$Entropi(A) = \sum_{i=1}^{v}\frac{pi+ni}{p+n}I(pi,ni) \quad (3)$$

Bilgi kazancı, veri kümesindeki nitelikler ile sınıf etiketi arasındaki matematiksel ilişkiler hakkında bilgiler sunan ve dengeli sınıf etiketi dağılımında daha kararlı sonuç verebilen olasılık temelli bir hesaplamadır. Bilgi kazancı, Denklem 4'deki

hesaplanmaktadır. Yazılım hata tahmini veri kümelerinde bulunan metrik değerinin ilgili modülün hatalı olma veya hatalı olmama durumuna olan katkısını göstermektedir [15]. Katkısı yani sınıf etiketi üzerindeki etkisi en fazla olan metrik için en iyi bilgi kazancına sahip metrik denmektedir. Karar ağaçlarında sınıflama yapabilmek veya sınıflama kuralı oluşturabilmek için ilk olarak en yüksek bilgi kazançlı metriğin tespit edilmesi gerekmektedir.

$$Bilgi\ Kazancı(A) = I(p,n) - Entropi(A) \quad (4)$$

C4.5 algoritması çalışmalarda sıklıkla kullanılan, başarılı ve güvenilir bir karar ağacı algoritmasıdır. C4.5 algoritmasında bilgi kazancı yerine bilgi kazanç oranı kullanılmaktadır. Çok sayıda farklı değer içeren özelliklerde oluşan kısıtlamaların giderilmesi için bilgi kazancı yerine önerilmiştir. Bilgi kazanç oranı, bilgi kazancı değerinin ilgili özelliğe ait entropi değerine bölünmesi ile elde edilmektedir [16],[17]. PART algoritması, C4.5 algoritmasının en iyi yaprak değerlerini kural haline dönüştürerek sınıflama yapmaktadır. Böl ve fethet mantığı ile çalışmaktadır. Bu algoritmada kısmi bir C4.5 karar ağacı oluşturularak her döngüsel işlemde en iyi yapraklar kural haline dönüştürülmektedir. C4.5 karar ağacı ile kıyaslandığı zaman daha az sayıda fakat daha doğru ve tam olan kurallar oluşturulduğu görülmektedir [18].

Hem C4.5 hem de PART algoritması, entropi ve bilgi kazancı ile kural çıkarımı ve sınıflama yapan, karar ağacı temelli algoritmalardır. Bu sebeple Smote algoritması ile elde edilen sonuçlar bu algoritmalar üzerinden değerlendirilmiş ve deneysel sonuçlar bu şekilde elde edilmiştir.

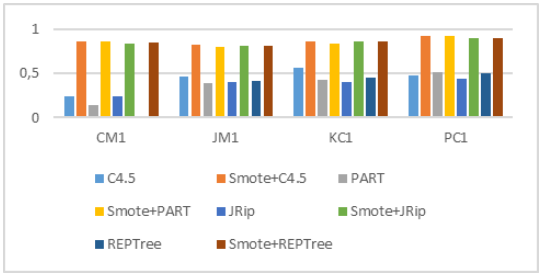
5 Deneysel sonuçlar ve tartışma

Sonuçlar, Weka [23] programı ve 10-katlı çapraz geçerleme [27] kullanılarak bulunmuştur. 10-katlı çapraz geçerleme, 10 parçaya bölünmüş veri kümesinin 9 parçasının eğitim geri kalan 1 parçasının ise test verisi olarak kullanılması sonucu elde edilen sınıflama başarısı değerlerinin aritmetik ortalamasının alınması sonucu hesaplanmaktadır. Tüm veri kümeleri azınlık (minorty) ile çoğunluk (majority) verileri arasındaki yüzde oranı farkına göre Smote algoritması ile dengeli sınıf dağılımına sahip durumuna getirilmiştir. Örneğin CM1 veri kümesinde 49 azınlık sınıfına ait veri örneği ve 449 çoğunluk sınıfına ait veri örneği bulunduğu için bu veri kümesine %816 oranında Weka programında Smote algoritması uygulanmıştır. Bu şekilde CM1, JM1, KC1, PC1 veri kümelerine sırasıyla %816, %317, %447, %1240 oranlarında Smote algoritması uygulanmıştır. Tablo 4'e bakıldığında, Smote algoritması ile birlikte veri kümelerindeki kayıt sayıları artığından dolayı kullanılan yöntemler açısından veri kümeleri daha büyük boyutlu hale gelmiştir. Bu sebeple elde edilen kural sayısı, JRip [49], REPTree, C4.5 ve PART algoritmalarında tüm veri kümeleri düzeyinde artmıştır. Tablodaki kural sayıları, Weka programı ile oluşturulan sınıflayıcı modelden alınan bilgiler ve oluşturulan karar ağaçlarındaki yaprak sayılarına göre belirlenmiştir. Yazılım hatalı olma veya hatalı olmama sınıfları için sınıflama modeli başarısını ölçen G-ortalama değeri tüm veri kümelerinde oldukça yükseldiği görülmektedir. Hatalı olma sınıfı doğal olarak tüm veri kümelerinde azınlık durumundadır. F-ölçütü hatalı olma durumunun başarısını veri kümesi dengesiz iken ölçmekte ve fikir vermektedir. Buna göre tüm veri kümelerinde hata olma sınıfı için F-ölçütü, Smote algoritması ile birlikte yükselmiştir. PART algoritması sadece NASA tarafından C programlama dili ile geliştirilen uzay aracı aletleri yazılım olan CM1 veri kümesinde C4.5 algoritmasını geçebilmiştir. Diğer tüm veri kümelerinde C4.5 karar ağacı algoritması, kural sınıflayıcı PART algoritmasına göre daha başarılı olmuştur. Bunun sebebi PART algoritmasının sınıflama kural sayısının C4.5 algoritmasına göre daha az ve özet şeklinde olmasından kaynaklanmaktadır.

Tablo 4: Veri kümelerine ait farklı yöntemlerin sonuçları.

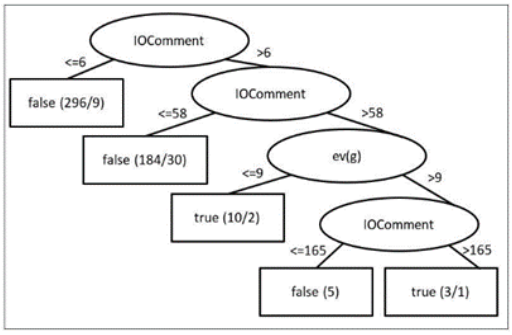
	Yöntem	Kural Sayısı	G-ortalama	F-ölçütü
CM1	JRip	3	0.245	0.102
	Smote+ JRip	10	0.842	0.848
	REPTree	1	0.000	0.000
	Smote+ REPTree	28	0.846	0.853
	C4.5	5	0.243	0.091
	Smote+C4.5	51	**0.867**	0.869
	PART	5	0.140	0.034
	Smote+PART	16	0.864	**0.870**
JM1	JRip	5	0.399	0.252
	Smote+ JRip	30	0.809	0.798
	REPTree	98	0.412	0.262
	Smote+ REPTree	405	0.817	0.809
	C4.5	340	0.464	0.305
	Smote+C4.5	673	**0.829**	**0.823**
	PART	34	0.385	0.237
	Smote+PART	102	0.802	0.785
KC1	JRip	3	0.406	0.306
	Smote+ JRip	20	0.865	0.864
	REPTree	32	0.450	0.302
	Smote+ REPTree	81	0.857	0.854
	C4.5	56	0.557	0.399
	Smote+C4.5	163	**0.868**	**0.867**
	PART	25	0.429	0.279
	Smote+PART	30	0.841	0.837
PC1	JRip	3	0.439	0.288
	Smote+ JRip	14	0.905	0.905
	REPTree	4	0.506	0.360
	Smote+ REPTree	38	0.905	0.906
	C4.5	24	0.480	0.327
	Smote+C4.5	69	**0.922**	**0.923**
	PART	5	0.518	0.375
	Smote+PART	33	0.921	0.922

Şekil 1'de tüm veri kümelerinin genel olarak Smote + C4.5 uygulamasında daha başarılı görülmesine rağmen genel anlamda PART algoritmasının orijinal veri kümeleri üzerinde daha iyi gelişim gösterdiği ortaya çıkmaktadır. Bunun nedeni Smote algoritması ile yükseltilen bilgi kazancına karşı PART algoritmasının daha duyarlı olması ve orijinal dengesiz durumda olan veri kümesinin kural sayısının ve sınıflama başarı değerlerinin ilk başta C4.5'e göre daha düşük olmasından kaynaklanmaktadır.



Şekil 1: G-ortalama değerlerine göre başarı değişimi.

Bilgi kazancı ve kazanç oranı, daha önce anlatıldığı gibi Şekil 2'deki CM1 veri kümesine ait C4.5 karar ağacını oluşturmak için önemlidir. Bu ağaçta kök ve diğer dallar bölme, ayırma hesaplarına göre belirlenmektedir. Bu şekilde oluşturulmuş bir ağaçtan elde edilen sınıflama kurallarına göre bilinmeyen veya yeni geliştirilmekte olan bir yazılım modülü için hatalı (true) veya hatasız (false) gibi öngörülerde bulunulabilmektedir.



Şekil 2: CM1 veri kümesine ait C4.5 karar ağacı.

Tablo 5'te PART algoritması ile CM1 veri kümesinden elde edilmiş sınıflama kural listesi gösterilmiştir. Bu kuralların elde edilmesinde karar ağacında olduğu gibi bilgi kazancı değerleri önem arz etmektedir. Buradaki parantez içindeki değerlerin anlamı, elde edilen kural ile kaç modülün kapsandığı ve sınıflama sonucunda kaçında başarısız tahmin yapıldığını göstermektedir.

Tablo 5: CM1 yazılım hata tahmini PART kural listesi.

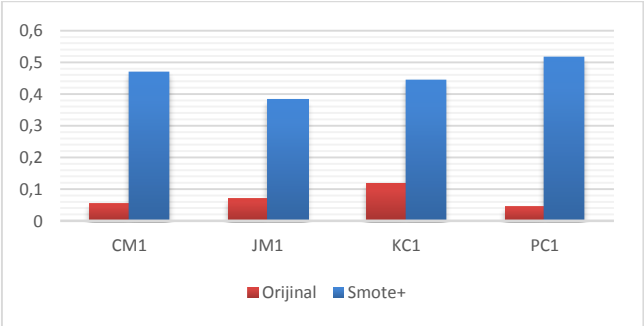
Kural	Kural içeriği
1	EĞER lOComment <= 6 ve lOBlank <= 6 ve iv(g) <= 2 İSE false (221/2)
2	EĞER lOComment <= 58 ve l > 0.02 ve uniq_Op <= 18 ve branchCount > 5 İSE false (54/2)
3	EĞER lOComment <= 58 İSE false (205/35)
4	EĞER ev(g) <= 9 ve l <= 0.02 ve iv(g) > 9 İSE true (5)
5	DEĞİLSE false (13/5)

Karar ağacı oluşturma ve kural çıkarım işlemlerinde, sınıf etiket ve diğer metrik oranları etkili olduğundan Smote algoritması ile elde edilmiş yeni kayıtlarla birlikte genel bilgi kazancı ve kazanç oranları değişmiş ve yükselmiştir. Tablo 6'de Smote algoritması ile dengeli sınıf haline getirilen veri kümelerine ait en iyi bilgi kazancı ve kazanç oranı değerleri verilmiştir. Buna göre en iyi değerlere sahip metrik değerleri yazılmış ve parantez içinde elde edilen sayısal kazanç değerleri verilmiştir. Tüm veri kümeleri için orijinal haline göre en iyi kazanç değerleri değişmiştir. PC1 veri kümesinin bilgi kazancı metriği hariç diğer veri kümelerindeki tüm metrik değerleri değişmiş ve sayısal değerleri yükselmiştir. Bunun sebebi, lOBlank metriğinin kazanç değeri yükselmiş olmasına rağmen daha yüksek bilgi kazançlı bir metrik değerinin olmamasından kaynaklanmaktadır.

Tablo 6: Veri kümesinin en iyi bilgi kazancı oranı yönünden metrikleri.

Yöntem	Bilgi Kazancı	Kazanç Oranı
CM1	lOComment (0.055)	lOComment (0.057)
Smote+CM1	lOCode (**0.470**)	lOCode (**0.168**)
JM1	loc (0.071)	loc (0.026)
Smote+JM1	v(g) (**0.382**)	locCodeAndComment (**0.098**)
KC1	B (0.119)	uniq_Opnd (0.071)
Smote+KC1	uniq_Op (**0.445**)	lOBlank (**0.143**)
PC1	lOBlank (0.045)	lOBlank (0.05)
Smote+PC1	lOBlank (**0.514**)	locCodeAndComment (**0.149**)

Şekil 3'te Smote uygulanmış veri kümelerinde elde edilen en iyi bilgi kazancı metrik değerlerinin değişimleri görülmektedir. Buna göre tüm yazılım hata tahmini veri kümelerinde bilgi kazancı değerleri daha iyi duruma gelmiştir. Bu sonuç metrik ve sınıf değerlerinin dağılım oranlarının daha dengeli duruma gelmesiyle yakından ilgilidir. Tablo 7'deki entropi hesaplaması ile ilgili basit örneklere bakıldığında [24]:



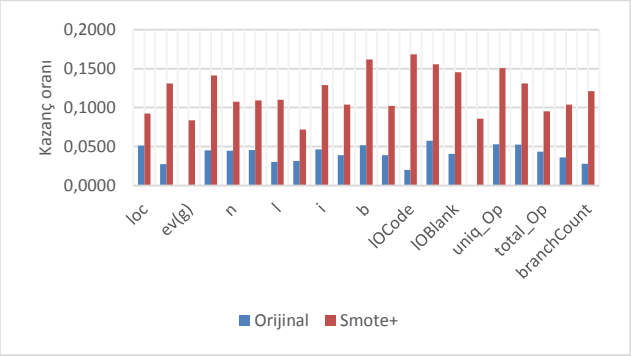
Şekil 3: En iyi metriklerin bilgi kazancı değişimleri.

Tablo 7: Entropi hesabı örneği.

	Düğüm	Sayı	Entropi
Örnek 1	Sınıf 1 = evet Sınıf 2 = hayır	1 5	$-(1/6)\log_2(1/6)-(5/6)\log_2(5/6) = 0.650$
Örnek 2	Sınıf 1 = evet Sınıf 2 = hayır	3 3	$-(3/6)\log_2(3/6)-(3/6)\log_2(3/6) = 1$

Dengeli sınıf dağılımı olan ikinci örneğin entropi değerinin sınıf dağılımı 3 evet, 3 hayır iken 1 olarak hesaplandığı fakat 1 evet, 5 hayır olduğunda daha farklı bir entropi değeri (0.650) hesaplandığı görülmektedir.

Şekil 4'te CM1 veri kümesine Smote algoritması uygulanmadan önceki kazanç oranı değerleri ile Smote algoritması uygulandıktan sonraki kazanç oranı değerleri çizgi grafiği şeklinde görülmektedir. Buna göre tüm metrik değerlerinde artış görülmekte ve özellikle Smote uygulamasından önceki en iyi bilgi değerine sahip Halstead grubuna ait yorum satırı sayısı (lOComment) metriği iken verinin dengeli hale gelmesinden sonra Halstead kod satırı sayısı (lOCode) metriğinin daha değerli olduğu ortaya çıkmıştır. Aynı şekilde çevrimsel karmaşıklık (v(g)) değeri yükselirken, zorluk (d) değerinin ise düştüğü görülmektedir.



Şekil 4: CM1 bilgi kazanç oranı değişimi.

Tüm yazılım hata tahmini sonuçları incelendiğinde hatalı olma veya hatalı olmama durumuna en fazla bilgi sunan metriklerin lOComment, lOCode, loc, locCodeAndComment, lOBlank gibi ilgili yazılım modülüne ait satır sayısı ile ilgili metrik değerlerinin daha etkili olduğu görülmektedir.

6 Sonuçlar

Bu çalışmada, azınlık sınıf değerlerini uygun bir biçimde çoğaltabilen Smote algoritması kullanılarak sınıf dağılımı dengesiz olan yazılım hata tahmini veri kümeleri, dengeli hale getirilmiştir. Bu şekilde yazılım hata tahmini yapılan veri kümelerinin G-ortalama ve F-ölçütü gibi sınıflama başarı ölçütleri yükseltilmiştir. C4.5, Part, JRip, REPTree gibi algoritmaların dengeli yazılım tahmini veri kümesi ile kullanılması üzerine yapılmış testlerde elde edilen sonuçların daha iyi olduğu görülmüştür. Yazılım hata tahmininde kullanılan metriklerin, bilgi kazancı yönünden daha yüksek sayısal değerli sonuçları elde edilmiştir. İleride yapılacak yazılım hata tahmini çalışmalarında, veri kümeleri üzerinde özellik seçimi ön işlemesi yapılarak, bilgi kazancı ve kazanç oranı değerleri üzerindeki etkileri incelenebilir. Böylece yazılım hata tahmini metrikleri sadeleştirilmek suretiyle daha güçlü sınıflama kuralları ve karar ağaçları oluşturulması sağlanabilir. Bu şekildeki güçlü sınıflama kuralları ile yazılım kalitesinin ölçme ve değerlendirme süreçleri daha da iyi hale getirebilir.

7 Kaynaklar

[1] Gupta D, Vinay K, Mittal GH. "Comparative study of soft computing techniques for software quality model". *International Journal of Software Engineering Research & Practices*, 1(1), 33-37, 2011.

[2] Hall T, Beecham S, Bowes D, Gray D, Counsell S. "A systematic literature review on fault prediction performance in software engineering". *IEEE Transactions on Software Engineering*, 38(6), 1276-1304, 2012.

[3] Catal C, Diri B. "A systematic review of software fault prediction studies". *Expert Systems with Applications*, 36(4), 7346-7354, 2009.

[4] Pal B, Hasan A, Aktar M, Shahdat N. "Cluster ensemble and probabilistic neural network modeling of class ımbalance learning in software defect prediction". *Artificial Intelligence and Applications*, In Press.

[5] Shirabad S, Menzies TJ. School of Information Technology and Engineering, University of Ottawa. "The PROMISE repository of software engineering databases". http://promise.site.uottawa.ca/SERepository (01.10.2017).

[6] Koru A, Liu H. "Building effective defect-prediction models in practice". *IEEE Software*, 22(6), 23-29, 2005.

[7] Menzies T, Dekhtyar A, Distefano J, Greenwald J. "Problems with precision: A response to comments on data mining static code attributes to learn defect predictors". *IEEE Transactions on Software Engineering*, 33(9), 637-640, 2007.

[8] Sahana DC. Software Defect Prediction Based on Classication Rule Mining. MSc Thesis, National Institute of Technology Rourkela, Rourkela, India, 2013.

[9] Menzies T, Greenwald J, Frank A. "Data mining static code attributes to learn defect predictors". *IEEE Transactions on Software Engineering*, 33(1), 2-13, 2007.

[10] Lessmann S, Baesens B, Mues C, Pietsch S. "Benchmarking classification models for software defect prediction: A proposed framework and novel findings". *IEEE Transactions on Software Engineering*, 34(4), 485-496, 2008.

[11] Mertik M, Lenic M, Stiglic G, Kokol P. "Estimating software quality with advanced data mining techniques". *International Conference on Software Engineering Advances,* Tahiti, 29 October-3 November 2006.

[12] Pelayo L, Dick S. "Applying novel resampling strategies to software defect prediction". *Fuzzy Information Processing Society NAFIPS '07*, San Diego, USA, 24-27 June, 2007.

[13] Magal K, Jacob SG. "Improved random forest algorithm for software defect prediction through data mining techniques". *International Journal of Computer Applications,* 117(23), 18-22, 2015.

[14] Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. "SMOTE: Synthetic minority over-sampling technique". *Journal of Artificial Intelligence Research*, 16, 321-357, 2002.

[15] Quinlan JR. "Induction of decision trees". *Machine Learning*, 1(1), 81-106, 1986.

[16] Quinlan JR. *C4.5: Programs for Machine Learning*. San Francisco, USA, Morgan Kaufmann Publishers Inc., 1993.

[17] Harris E. "Information gain versus gain Ratio: a study of split method biases". *7th International Symposium on Artificial Intelligence and Mathematics*, Florida, USA, 2-4 January 2002.

[18] Frank E, Witten IH. "Generating accurate rule sets without global optimization". *15th International Conference on Machine Learning*, Wisconsin, USA, 24-27 July 1998.

[19] Tan KC, Tay A, Lee TH, Heng CM. "Mining multiple comprehensible classification rules using genetic programming". *Proceedings of the Congress Evolutionary Computation,* Hawaii, USA, 12-17 May 2002.

[20] Li K, Zhang W, Lu Q, Fang X. "An improved SMOTE imbalanced data classification method based on support degree". *International Conference on Identification, Information and Knowledge in the Internet of Things*, Beijing, China, 17-18 October 2014.
[21] Jiang K, Lu J, Xia K. "A novel algorithm for imbalance data classification based on genetic algorithm improved SMOTE". *Arabian Journal for Science and Engineering*, 41(8), 3255-3266, 2016.
[22] Hu Y, Guo D, Fan Z, Dong C, Huang Q, Xie S, Liu, G, Tan J, Li B, Xie Q. "An Improved algorithm for ımbalanced data and small sample size classification". *Journal of Data Analysis and Information Processing*, 3, 27-33, 2015.
[23] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. "The WEKA Data Mining Software: An Update". *SIGKDD Explorations*, 11(1), 10-18, 2009.
[24] Tan PN, Steinbach M, Kumar V. *Introduction to Data Mining*, 1st ed. Boston, USA, Addison-Wesley Longman Publishing Co. Inc. 2005.
[25] Watson AH, Mccabe TJ. *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. Washington, USA, National Institute of Standards and Technology Special Publication 500-235, 1996.
[26] Tomar D, Agarwal S. "Prediction of Defective Software Modules Using Class Imbalance Learning". *Applied Computational Intelligence and Soft Computing*, Article ID 7658207, 12 pages, 2016.
[27] Stone M. "Cross-validatory choice and assessment of statistical predictions". *Journal of the Royal Statistical. Society,* 36(2), 111–147, 1974.
[28] Paramshetti P, Phalke DA. "Survey on software defect prediction using machine learning techniques". *International Journal of Science and Research*, 3, 1394-1397, 2014.
[29] Hall T, Beecham S, Bowes D, Gray D, Counsell S. "A systematic literature review on fault prediction performance in software engineering". *IEEE Transactions on Software Engineering*. 38(6), 1276-304, 2012.
[30] Wang S, Yao X. "Using Class Imbalance Learning for Software Defect Prediction". *IEEE Transactions on Reliability*, 62(2), 434-43, 2013.
[31] Aleem S, Capretz LF, Ahmed F. "Benchmarking machine learning techniques for software defect detection". *International Journal of Software Engineering & Applications,* 6(3), 11-23, 2015.
[32] Prasad M, Florence L, Arya, A. "A Study on Software Metrics Based Software Defect Prediction using Data Mining and Machine Learning Techniques". *International Journal of Database Theory and Application*, 8(3), 179-190, 2015.
[33] Menzies T, Krishna, R, Pryor, D. North Carolina State University, Department of Computer Science. "The Promise Repository of Empirical Software Engineering Data". http://openscience.us/repo, (01.10.2017).
[34] Frank E, Witten IH. "Generating accurate rule sets without global optimization". *15th International Conference on Machine Learning*, San Francisco, USA, 24-27 July 1998.
[35] Martin B. Instance-Based learning: Nearest Neighbor With Generalization. MSc Thesis, University of Waikato, Hamilton, New Zealand, 1995.
[36] Roy S. Nearest Neighbor with Generalization. MSc Thesis, University of Canterbury, Christchurch, New Zealand, 2002.
[37] Cendrowska J. "Prism - an Algorithm for Inducing Modular Rules". *International Journal of Man-Machine Studies*, 27(4), 349-70, 1987.
[38] Japkowicz N, Stephen S. "The class imbalance problem: A systematic study". *Intelligent Data Analysis*, 6(5), 429-449, 2002.
[39] Batista G, Prati R, Monard M, "A Study of the Behavior of several methods for balancing machine learning training data". *ACM SIGKDD Explorations Special issue on learning from imbalanced datasets*, 6(1), 20-29, 2004.
[40] He H, Bai Y, Garcia EA, Li S. "ADASYN: Adaptive synthetic sampling approach for imbalanced learning". *2008 IEEE International Joint Conference on Neural Networks,* Hong Kong, China, 1-8 June 2008.
[41] Holte RC. "Very simple classification rules perform well on most commonly used datasets". *Machine Learning*, 11, 63-90, 1993.
[42] John GH, Langley P. "Estimating Continuous Distributions in Bayesian Classifiers". *11th Conference on Uncertainty in Artificial Intelligence*, Montreal, Canada, 18-20 August 1995.
[43] Breiman L, Friedman J, Stone CJ, Olshen RA. *Classification and Regression Trees,* England, Taylor & Francis, 1984.
[44] Wang S, Yao X. "Multiclass imbalance problems: analysis and potential solutions". *IEEE Transactions on Man, Systems and Cybernetics Part B*, 42(4), 1119-1130, 2012.
[45] Breiman L. "Random Forests". *Machine Learning*, 45(1), 5-32, 2001.
[46] Specht DF. "Probabilistic neural networks", *Neural Networks*, 3(1), 109-118, 1990.
[47] Catal C, Diri B. "Investigating the effect of dataset size, metrics sets and feature selection techniques on software fault prediction problem". *Information Sciences*, 179(8), 1040-1058, 2009.
[48] Catal, C, Diri, B. "Software defect prediction using artificial ımmune recognition system". *The IASTED Int'l Conference on Software Eng*, Innsbruck, Austria, 13-15 February 2007.
[49] Cohen WW. "Fast effective rule ınduction". *12th International Conference on Machine Learning*, California, USA, 09-12 July 1995.
[50] Han J, Kamber M, Pei J. *Data Mining: Concepts and Techniques*, 3rd ed. Massachusetts, USA, Morgan Kaufmann, 2011.