Softare Vulnerability Prediction using Extreme Learning Machines Algorithm

Conference Paper · September 2019

CITATIONS
0

READS
37

4 authors, including:

Cagatay Catal
Wageningen University & Research
82 PUBLICATIONS **2,010** CITATIONS

SEE PROFILE

Bedir Tekinerdogan
Wageningen University & Research
276 PUBLICATIONS **2,379** CITATIONS

SEE PROFILE

Software Vulnerability Prediction using Extreme Learning Machines Algorithm

Aşırı Öğrenme Makinesi Algoritmasını Kullanarak Yazılım Zafiyet Kestirimi

Ali Seydi Keçeli[1], Aydın Kaya[1], Çağatay Çatal[2], Bedir Tekinerdoğan[2]

[1] Hacettepe Üniversitesi, Bilgisayar Mühendisliği, Türkiye
{aliseydi, aydinkaya}@cs.hacettepe.edu.tr
[2] Wageningen Üniversitesi, Bilgi Teknolojileri Grubu, Hollanda
{cagatay.catal, bedir.tekinerdogan}@wur.nl

Abstract. Software vulnerability prediction aims to detect vulnerabilities in the source code before the software is deployed into the operational environment. The accurate prediction of vulnerabilities helps to allocate more testing resources to the vulnerability-prone modules. From the machine learning perspective, this problem is a binary classification task which classifies software modules into vulnerability-prone and non-vulnerability-prone categories. Several machine learning models have been built for addressing the software vulnerability prediction problem, but the performance of the state-of-the-art models is not yet at an acceptable level. In this study, we aim to improve the performance of software vulnerability prediction models by using Extreme Learning Machines (ELM) algorithms which have not been investigated for this problem. Before we apply ELM algorithms for selected three public datasets, we use data balancing algorithms to balance the data points which belong to two classes. We discuss our initial experimental results and provide the lessons learned. In particular, we observed that ELM algorithms have a high potential to be used for addressing the software vulnerability prediction problem.

Keywords: Software Vulnurability Prediction, Machine Learning, Extreme Learning Machines

Özet. Yazılım zafiyet kestirimi, yazılımın operasyonel ortama konuşlandırılmadan önce kaynak kod içerisindeki zafiyetlerinin tespit edilmesini amaçlamaktadır. Zafiyetlerin doğru şekilde kestirilmesi daha fazla test kaynağının zafiyet-eğilimli modüllere ayrılmasına yardımcı olmaktadır. Makine öğrenmesi bakışı açısıyla, bu problem yazılım modüllerini, zafiyet-eğilimli ve zafiyet-eğilimli-olmayan seklinde iki kategoriye ayıran ikili bir sınıflandırma işlemidir. Yazılım zafiyet kestirimi için makine öğrenmesi temelli modeller şimdiye kadar geliştirilmiştir ancak su ana kadar teknikte gelinen en son noktada modellerin perfor-

mansı kabul edilebilir seviyede değildir. Bu çalışmada, yazılım zafiyet kestirim modellerinin performanslarını iyileştirmek üzere bu problemde henüz çalışılmamış olan Aşırı Öğrenme Makinesi algoritmalarından yararlanarak performansı iyileştirmeyi amaçlıyoruz. AÖM algoritmalarını 3 açık veri kümesinde uygulamadan önce, iki sınıfa ait olan veri noktalarını dengelemek üzere veri dengeleme algoritmalarını kullanıyoruz. Bu kapsamda yaptığımız deneylerle ilgili ilk deneysel sonuçlarımızı tartışarak öğrenilen dersleri sunuyoruz. AÖM algoritmalarının yazılım zafiyet kestirimi probleminde yüksek potansiyele sahip olduğu gözlemlenmektedir.

Anahtar Kelimeler: Yazılım Zafiyet Kestirimi, Makine Öğrenmesi, Aşırı Öğrenme Makinesi

1 Giriş

Büyük ölçekteki organizasyonları, şirketleri veya KOBİ'leri etkileyebilecek çok sayıda ve farklı seviyelerde zafiyetten söz etmek mümkündür. Örneğin, bu zafiyetler ağ seviyesinde, personel seviyesinde, donanım seviyesinde olabileceği gibi kullanılmakta olan uygulama veya sistem yazılımlarından kaynaklı olarak yazılım seviyesinde de olabilmektedir. Zafiyetlerin doğru zamanda ve etkin şekilde tespit edilmediği durumda, gerek işlenmekte ve saklanmakta olan veri açısından gerekse de sistemin işleyişi açısından, gizlilik, mahremiyet, bütünlük gibi birçok kalite faktörü kapsamında sorunlar yaşanmaktadır.

Günümüzde geliştirilmekte olan birçok yazılım sistemi, son yıllarda kaynak kod satır sayısı açısından birkaç milyon satır düzeyine ulaşmış durumdadır ve önceden beri organizasyon içerisinde var olan sistemlerle (legacy systems) olan entegrasyonunu da dikkate aldığımız zaman, tipik olarak 50 milyon veya 100 milyon kod satırı düzeyinde yazılımlar karşımıza çıkabilmektedir. Dünyadaki genel trendlere bakıldığında bu tur çok geniş ölçekli yazılım sistemlerinin, onumuzdaki 20-30 yıl suresince ultra geniş ölçekli sistemler haline gelerek, milyon kod satır sayısından milyar kod satırına doğru değişim göstereceği, Carnegie Mellon Üniversitesi, Yazılım Mühendisliği Enstitüsü (SEI) tarafından hazırlanan raporlarda ifade edilmektedir.

Kaynak kod satır sayısının gittikçe artmakta olduğu bu bağlamda, zafiyetlerin ortaya çıkmadan önce kestirilebilmesi gittikçe önem arz eden bir husus haline gelmiştir. Tek bir yazılım dahi, artık tek başına çalışmaktan çok öte, bulunduğu platformdaki diğer altyapılarla koordine ve eş güdüm halinde çalışmak durumundadır. Bu altyapılar; farklı yazılım çerçeveleri, ara katmanlar, veritabanı yönetim sistemleri, yazılım kütüphaneleri şeklinde olabileceği gibi gelişmekte olan farklı teknolojilerden kaynaklı blok zinciri altyapısı, bağlı veri platformu, büyük veri işleme sistemi gibi özelleşmiş düzeyde de olabilmektedir. Tek bir yazılım sisteminin çalışmasını incelerken dahi, etkileşimde olduğu bu tür diğer bileşenleri dikkate aldığımız zaman, her bir bileşenden bir güvenlik tehlikesi veya zafiyeti oluşması oldukça muhtemeldir. Güvenli yazılım geliştirme yaşam çevrimleri, güvenli yazılım tasarımı, statik ve dinamik yazılım analiz araçları, uygulama seviyesindeki güvenlik duvarları bazı sorunları ortadan kaldırsa da bu tür teknikler ve teknolojilerle tamamen tüm güvenlik sorunlarının çözü-

lebildiğini söyleyebilmek söz konusu değildir, aksi halde hemen hemen her gün karşımıza çıkan siber saldırılar, güvenlik ihlalleri ve yetkisiz kişilerin yazılım sistemleri üzerinde denetim kazanarak çıkar veya kazanç elde edilebilmesi mümkün olmazdı.

Sızma testleri (penetration testing) ve statik kod analiz araçları açısından zafiyet tespitini dikkate aldığımız zaman, yanlış pozitif oranının oldukça yüksek olduğu bilinmektedir. Kod gözden geçirme (code review) alternatif bir yöntem seklinde değerlendirilse de, 50-100 milyon kod satırı düzeyindeki yazılımlar için tüm bileşenler bağlamında bu işlemin gerçekleştirilmesi, kaynak ve bütçe açısından çoğu durumda uygulanabilir olmamaktadır.

Bu kısıtlar nedeniyle, son dönemde bazı araştırmacılar makine öğrenmesi algoritmalarından yararlanarak, zafiyet-eğilimli modülleri kestirerek, doğrulama ve geçerleme kaynaklarını bu modüllere daha fazla ayırmakta ve bu sayede, zafiyetleri azaltmayı hedeflemektedir. Bu araştırma alanı, yazılım zafiyet kestirimi olarak adlandırılmakta olup, yazılım kusur kestirimi (software fault prediction) problemiyle kullanılabilecek metrikler ve yöntemler açısından benzerlik göstermektedir ancak zafiyetlerin, kusurlara kıyasla çok daha küçük bir alt küme olması ancak zafiyetlerin ortaya çıkması durumunda, etkilerinin çok daha kritik seviyede olması ve bu zafiyetleri tespit açısından, kaynak kodun ve saldırı örüntülerinin daha derin seviyede bilinmesini gerektirmesi açısından çok daha farklı bir problem olduğunu ifade etmek mümkündür. Araştırmacı bakış açısıyla diğer bir zorluk ise şirketlerin genel olarak zafiyetlere ilişkin olarak verilerini paylaşmak istememesi nedeniyle, bu alanda kamuya açık veri kümesi üretilmesi ve paylaşılması çok fazla görünmemektedir. Bu çalışmada, Stuckman ve arkadaşlarının [3] oluşturulmuş olduğu 3 adet kamuya açık veri kümesinden yararlanılmıştır.

Bu çalışmanın genel amacı, mevcut durumda istenen düzeyde olmayan yazılım zafiyet kestirimi modellerinin performansını daha iyi hale getirmek ve bu amaçla, veri dengeleme (data balancing) ve Aşırı Öğrenme Makineleri (Extreme Learning Machines-AÖM) algoritmalarından yararlanmaktır. AÖM algoritmalarının daha hızlı öğrenme hızı, insan müdahalesi açısından daha az gereksinim ihtiyacı ve diğer problemlerde raporlanmış olan daha iyi genelleştirme yeteneği nedeniyle, bu problemde de ilk kez bu çalışma kapsamında incelenmesi hedeflenmiştir. Bu alandaki veri kümelerinin dengesiz (unbalanced) olması nedeniyle (örneğin, %90 oranında zafiyet içermeyen veri noktasının oluşturduğu bir sınıf ve %10 oranında zafiyet içeren veri noktalarının olduğu başka bir sınıf), veri dengeleme algoritmalarının da model performansını iyileştirme açısından değerlendirilmesi planlanmıştır.

Bu çalışmanın literatüre temel katkısı; veri dengeleme algoritmalarının ve Aşırı Öğrenme Makinesi algoritmalarının, yazılım zafiyet kestirimi problemi bağlamında ilk kez uygulanarak gelecek açısından ümit verici sonuçlarının elde edilmiş olmasıdır.

Takip eden bölüm 2'de ilişkili çalışmalar sunulmuştur. Bölüm 3'de önerilmekte olan yöntem açıklanmaktadır. Bölüm 4'de deneysel sonuçlar verilerek Bölüm 5'de sonuçlar ve gelecek çalışmalar ortaya konulmuştur.

2 İlişkili Çalışmalar

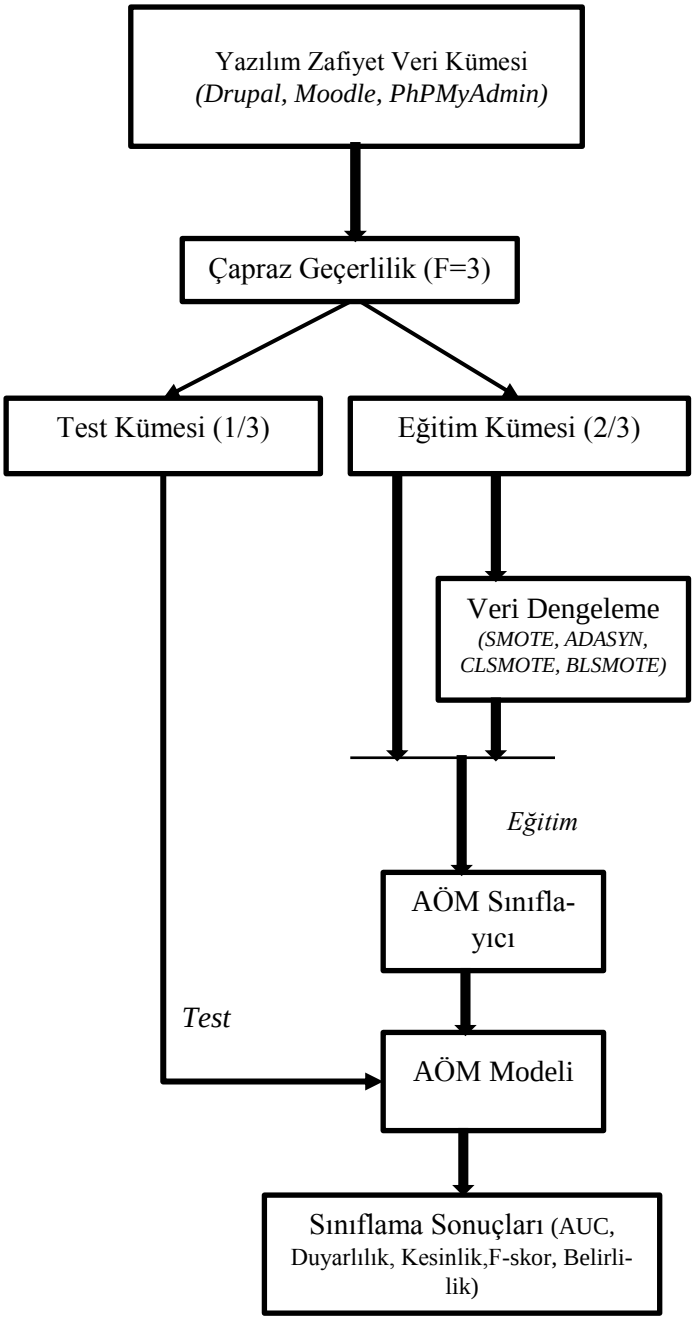
Yazılım zafiyet kestirimi, yazılım güvenliği ile doğrudan ilgili ve yazılım kalitesini etkileyen kalite güvence aktivitelerinden birisidir. Yazılım zafiyet çalışmaları büyük oranda makine öğrenmesi yaklaşımlarından faydalanmaktadır. Geçmiş veriye bağlı olarak oluşturulmuş bu kestirimsel modeller, yazılım test aşamasından önce çalıştırılarak, zafiyet eğilimli modüller ile ilgili sınıflandırmalar gerçekleştirilmektedir. Bu sonuçlar ışığında, zafiyet eğilimli olarak etiketlenen modüller daha detaylı incelemelere tabi tutulmaktadırlar.

Yazılım zafiyet kestiriminde (YZK) genel olarak yazılım metrikleri ve yazılım kaynak kodlarından üretilmiş metin öznitelikleri kullanılmaktadır [1]. Jimenez ve arkadaşları tarafından yapılan çalışmada; yazılım metrikleri, metin madenciliği ve istatistiki analiz uyarı yaklaşımları birleştirilmiştir [2]. Yazılım metrikleri ve metin özniteliklerinden faydalanan bir başka çalışma Stuckman ve arkadaşları [3] tarafından önerilmiştir. Bu çalışmada ayrıca birkaç veri dengeleme yaklaşımlarının yazılım zafiyet kestirimi üzerine etkileri de incelenmiştir. Largerstorm ve arkadaşları [4] farklı yazılım metriklerinin zafiyet tespiti üzerindeki etkilerini incelemiştir. Bileşen ve mimari seviyedeki metriklerin, zafiyet tespitinde etkili oldukları görülmüştür.

Shin ve arkadaşları [5] kaynak kod değişim (*code chunk*), karmaşıklık ve geliştirici aktivitelerinin yazılım zafiyet kestirimi üzerindeki etkilerini incelemiştir. Walden ve arkadaşları [6] tarafından yapılan çalışmada metin madenciliği ile üretilen özniteliklerin yazılım metriklerine göre daha başarılı sonuçlar ürettiği gözlemlenmiştir. Sultana [7], istatistiki yöntemler ile makine öğrenmesi yöntemlerini birlikte kullanan modeller önermiştir. Shin ve Williams [8] yazılım hata tespiti modellerinin zafiyet tespitinde de kullanılıp kullanılamayacağı üzerine testler yapmışlardır. Elde edilen sonuçlar ışığında zafiyet ve hata tespiti modellerinin yakın sonuçlar ürettiği gözlemlenmiştir. Morrison ve arkadaşları [9] Windows işletim sistemi üzerinde çalıştırılmak üzere 2 YZK modeli önermiştir. Pang ve arkadaşları [10] derin öğrenme ağı kullanarak zafiyet kestirimi gerçekleştirmişlerdir. Derin öğrenme ağının eğitiminde kaynak kodlardan elde edilen öznitelikler kullanılmıştır. Yaptığımız literatür taramasına göre, Aşırı Öğrenme Makinesi (AÖM) algoritmaları bu problem özelinde henüz uygulanmamış olup ilk kez bu yayınlamakta olduğumuz çalışma kapsamında ele alınmıştır.

3 Yöntem

Yönteme ilişkin genel akış şeması Şekil-1 de gösterilmiştir. İlk adım olarak veri seti test ve eğitim kümelerine ayrılmıştır. Sonraki adımda eğitim kümesi üzerinde dengeleme yapılarak ve yapılmadan AÖM modelleri oluşturulmuştur. Oluşturulan bu modeller üzerinde test kümesi kullanılarak sınıflandırma yapılmış ve sınıflandırma sonuçlarına ait AUC (Area Under ROC Curve-ROC eğrisi altında kalan alan), duyarlık, keskinlik, F-skor ve belirlilik ölçümleri yapılmıştır. Deneyler sırasında üçlü çapraz geçerlik testi uygulanmıştır. Çapraz geçerlik testinde; veri kümesi rastgele 3 parçaya ayrılmakta ve parçalardan 2'si eğitim, kalan tek parça ise test için kullanılmaktadır.



Şekil 1. Uygulanan deney tasarımının genel akışı.

3.1 Veri Dengeleme

Yazılım zafiyet tespiti için kullanılan veri kümeleri son derece dengesiz olması sebebiyle model eğitiminden önce çeşitli veri dengeleme yöntemleri ile daha homojen bir eğitim kümesi oluşturulmuştur. Veri dengeleme için SMOTE [11], CLSMOTE [12], BLSMOTE [13] ve ADASYN [14] yöntemleri kullanılmıştır. SMOTE ve türevleri var olan örnekleri çoğaltmak yerine yeni örnekler üretmektedir. CLSMOTE ve BLSMOTE, SMOTE yaklaşımının geliştirilmiş sürümleridir. CLSMOTE yaklaşımında veri önce kümelenmekte sonrasında ise standart SMOTE uygulanmaktadır. BLSMOTE yönteminde ise sınır (borderline) örnekler ilk olarak tespit edilmektedir. Yeni üretilecek örnekler, bu sınır örnekler üzerinden tespit edilmektedir. Kullanılan son dengeleme yöntemi ise ADASYN'dir. Bu yaklaşımda, azınlık sınıfları için farklı ağırlıklarda dağılımlar kullanılmaktadır.

3.2 Aşırı Öğrenme Makinesi

Aşırı öğrenme makineleri (AÖM) sadece tek gizli katmandan oluşan ileri beslemeli bir sinir ağı (neural network) modelidir [15]. Hem sınıflama hem de regresyon problemlerinde kullanılabilmektedir. Örnek bir AÖM modeli Şekil-2'de gösterilmiştir. Şekil-2'de w girdi bağlantılarının ağırlıklarını β çıktı ağırlıklarını ve g aktivasyon fonksiyonunu belirtmektedir. Gizli katmandaki aktivasyon işlemleri Eşitlik-1 ile ifade edilebilir.

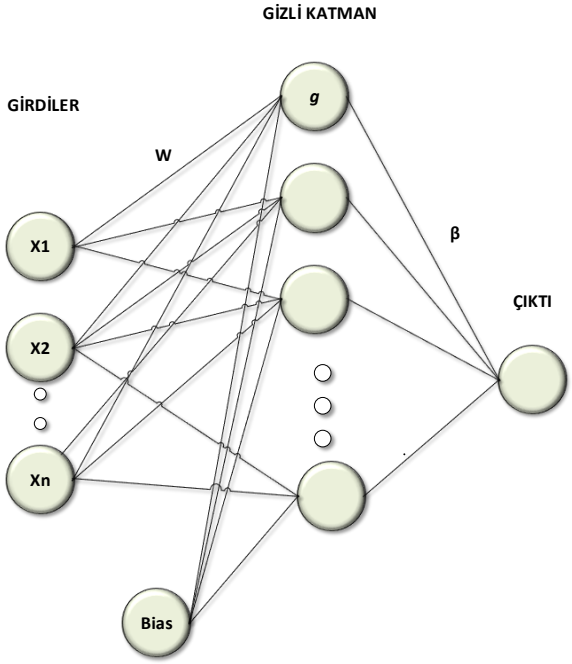
$$\sum_{j=1}^{M} \beta_i \, g(w_j x_j + b_j) \tag{1}$$

Geleneksel yapay sinir ağı yaklaşımlarından farklı olarak giriş katmanına uygulanan ağırlıklar rastgele bir şekilde verilmektedir. AÖM'ye ait rastgele olarak atanan bu değerler daha sonra eğitim aşamasında güncellenmemektedir. Buna karşın gizli katman ve çıktı katmanı arasındaki bağlantılara atanan ağırlık değerleri doğrusal bir model yardımıyla analitik olarak hesaplanır. Geleneksel sinir ağlarında katmanlar arasındaki ağırlıklar iteratif olarak yani hatayı en az yapan değerlere ulaşılana kadar farklı değerlerin sınanmasıyla hesaplanırken, AÖM'lerde bu ağırlıkların doğrusal olarak hesaplanması modelin eğitiminde ciddi bir hızlanmaya sebep olmaktadır [15]. AÖM yapısında çıkış katmanı Eşitlik-2'deki hale dönüştürülüp Eşitlik-3 şeklinde ifade edilebilir. Eşitlik-3'teki ifade artık doğrusal olarak çözülebilir.

$$H(w_{i,j}, x_i, b_i) = \begin{bmatrix} g(w_{1,1}, x_1, b_1) & \cdots & g(w_{1,m}, x_m, b_m) \\ \vdots & \ddots & \vdots \\ g(w_{n,1}, x_1, b_1) & \cdots & g(w_{n,m}, x_m, b_m) \end{bmatrix} \tag{2}$$

$$y = H\,\beta \tag{3}$$

Ayrıca yapılan çalışmalarda AÖM yaklaşımının pek çok uygulama alanında yaygın bir diğer makine öğrenmesi yöntemi olan SVM yaklaşımından daha başarılı olduğu görülmüştür [16,17].



Şekil-2 Aşırı Öğrenme Makinesi

Geleneksel sinir ağlarından daha az parametreye sahip olması ve eğitim parametrelerinin belirlenmemesi gibi avantajlara sahiptir. Ancak ara katmanın nöron sayısı ve kullanılacak olan aktivasyon fonksiyonu seçimi, kullanıcı tarafından belirlenmektedir. Aktivasyon fonksiyonları bir nörona gelen girdiye uygulanan ve bu bağlamda oluşturulacak çıktıyı hesaplayan fonksiyonlardır. Bu modellerin doğrusal olmama özelliği, aktivasyon fonksiyonlarının da doğrusal olmamasından kaynaklanmaktadır. AÖM tarafından kullanılan aktivasyon fonksiyonları Tablo-1'de belirtilmiştir.

Tablo-1 AÖM'de kullanılan bazı aktivasyon fonksiyonları [18]

Aktivasyon Fonksiyonu	***Eşitlik***
Sigmoid	$a = \frac{1}{1+\exp(-n)}$
Radyal Bazlı	$a = \exp(-n^2)$

Normalleştirilmiş Radyal Bazlı	$a = \frac{\exp(-n^2)}{\sum \exp(-n^2)}$
Elliot Sigmoid	$a = \frac{0.5 * \mathrm{n}}{1 + abs(n)} + 0.5$
Hard Limit	$a = \begin{cases} 1, eğer\ n \geq 0 \\ 0, diğer \end{cases}$
Sinüs	$\mathrm{a} = \sin(e)$
Kosinüs	$\mathrm{a} = \cos(e)$
Dogrusal	$a = n$
SoftMax	$a = \frac{\exp(n)}{\sum \exp(n)}$
Ters	$a = \frac{1}{n}$
Pozitif Doğrusal	$a = \begin{cases} n, eğer\ n \geq 0 \\ 0, n < 0 \end{cases}$

4 Deneysel Sonuçlar

Bu bölümde, Sekil 1'de detayları verilmekte olan deney tasarımına ilişkin tüm deneysel sonuçlar sunulmaktadır.

Yazılım metriklerinin nitelik olarak kullanıldığı makine öğrenmesi modellerine ilişkin sonuçlar, Tablo 2-3-4'de sunulmuştur. Bu kapsamda kullanılan yazılım metrikleri; HTML içermeyen kod satır sayısı, PHP dosyası içerisindeki kod satır sayısı, metod sayısı, çevrimsel karmaşıklık (cyclomatic complexity), maksimum iç içe geçme karmaşıklığı (maximum nesting complexity), Halstead hacim, fan-in, fan-out, çağrılan iç metotların sayısı, toplam dış çağrıların sayısı, dış metotların sayısı, metotlara olan dış çağrıların sayısı olarak 12 tanedir. Kamuya açık veri kümelerinde bu metriklerin detayları açıklanmaktadır.

Yazılım kaynak kodunun metin olarak değerlendirilerek oluşturulan sözcük frekans analizi temelli veri kümelerine ilişkin analiz sonuçları ise Tablo 5-6-7'de verilmektedir. Kaynak kod içerisindeki her bir belirteç (token) bir nitelik olarak veri kümelerinde temsil edilmektedir. Tablo 8-9-10'da ise hem yazılım metriklerinin hem de metinsel niteliklerin birleştirildiği durumdaki veri kümelerinde (bütünleşik) yapılan analizlerin sonuçları sunulmuştur.

AUC değerlendirme parametresi referans alınarak yukarıda belirtilen tablolar incelendiğinde; metin seviyesindeki niteliklerle kurulan modellerin, yazılım metrik seviyesindeki niteliklerle kurulan modellerden daha iyi performans verdiği, bütünleşik modellerin performansıyla metin seviyesindeki modellerin performansı kıyaslandığı durumda bütünleşik nitelikler seviyesinde performansın çoğu durumda biraz daha iyi olduğu ancak aradaki farkın çok da fazla olmadığı gözlemlenmiştir. Bu nedenle, sade-

ce metin seviyesindeki niteliklerle model kurulmasının daha uygun olacağı önerilmektedir.

Metin seviyesindeki nitelikler ve bütünleşik seviyedeki niteliklerle elde edilen performanslar incelendiğinde, çoğu durumda ADASYN isimli veri dengeleme yöntemiyle daha başarılı kestirim modellerin kurulabildiği gözlemlenmiştir. Bu nedenle, veri dengeleme algoritmalarını da kestirim modellerine eklemek isteyen araştırmacılar için ADASYN algoritması önerilmektedir.

Tablo-2. Drupal metrik veri kümesi için sınıflama sonuçları

Dengeleme	AUC	Duyarlılık	Kesinlik	F-skor	Belirlilik
Dengelenmemiş	0.6206	0.4242	0.6077	0.4988	0.8777
ADASYN	0.6210	0.6594	0.5386	0.5923	0.7501
SMOTE	0.6166	0.6406	0.5455	0.5889	0.7637
BL SMOTE	**0.6232**	0.6574	0.5336	0.5886	0.7451
CL SMOTE	0.6159	0.6300	0.5311	0.5760	0.7536

Tablo-3. Moodle metrik veri kümesi için sınıflama sonuçları

Dengeleme	AUC	Duyarlılık	Kesinlik	F-skor	Belirlilik
Dengelenmemiş	0.6072	0.0000	NaN	NaN	0.9997
ADASYN	0.6171	0.5200	0.0225	0.0431	0.8145
SMOTE	0.6204	0.5108	0.0225	0.0432	0.8182
BL SMOTE	0.6173	0.5292	0.0233	0.0446	0.8177
CL SMOTE	**0.6242**	0.5317	0.0237	0.0454	0.8202

Tablo-4. PhPMyadmin metrik veri kümesi için sınıflama sonuçları

Dengeleme	AUC	Duyarlılık	Kesinlik	F-skor	Belirlilik
Dengelenmemiş	0.6641	0.1526	0.6019	0.2402	0.9902
ADASYN	0.6989	0.3289	0.1437	0.1998	0.8203
SMOTE	0.6960	0.2970	0.1396	0.1896	0.8327
BL SMOTE	**0.7015**	0.3081	0.1405	0.1928	0.8264
CL SMOTE	0.6987	0.3163	0.1477	0.2009	0.8331

Tablo-5. Drupal metin veri kümesi için sınıflama sonuçları

Dengeleme	AUC	Duyarlılık	Kesinlik	F-skor	Belirlilik
Dengelenmemiş	0.7071	0.4732	0.5552	0.5091	0.8311
ADASYN	**0.7152**	0.4858	0.5320	0.5070	0.8100
SMOTE	0.7115	0.4871	0.5372	0.5097	0.8136
BL SMOTE	0.7132	0.4913	0.5256	0.5068	0.8034
CL SMOTE	0.7125	0.4784	0.5456	0.5089	0.8234

Tablo-6. Moodle metin veri kümesi için sınıflama sonuçları

Dengeleme	AUC	Duyarlılık	Kesinlik	F-skor	Belirlilik
Dengelenmemiş	0.7218	0.0708	0.2119	NaN	0.9978
ADASYN	0.7325	0.2042	0.0314	0.0543	0.9481
SMOTE	0.7287	0.2142	0.0322	0.0560	0.9472
BL SMOTE	**0.7336**	0.2167	0.0325	0.0565	0.9470
CL SMOTE	0.7247	0.2450	0.0318	0.0562	0.9384

Tablo-7. PhPMyadmin metin veri kümesi için sınıflama sonuçları

Dengeleme	AUC	Duyarlılık	Kesinlik	F-skor	Belirlilik
Dengelenmemiş	0.7510	0.1748	0.1922	NaN	0.9327
ADASYN	**0.7654**	0.2593	0.1057	0.1499	0.7981
SMOTE	0.7622	0.2733	0.1072	0.1536	0.7927
BL SMOTE	0.7574	0.2852	0.1144	0.1630	0.7977
CL SMOTE	0.7509	0.2741	0.1156	0.1623	0.8077

Tablo-8. Drupal bütünleşik (metin + metrik) veri kümesi için sınıflama sonuçları

Dengeleme	AUC	Duyarlılık	Kesinlik	F-skor	Belirlilik
Dengelenmemiş	0.7002	0.5026	0.5527	0.5250	0.8196
ADASYN	**0.7181**	0.4861	0.5160	0.4991	0.7971
SMOTE	0.7107	0.5003	0.5286	0.5127	0.8013
BL SMOTE	0.7084	0.5126	0.5208	0.5151	0.7894
CL SMOTE	0.7091	0.4961	0.5256	0.5093	0.8019

Tablo-9. Moodle bütünleşik (metin + metrik) veri kümesi için sınıflama sonuçları

Dengeleme	AUC	Duyarlılık	Kesinlik	F-skor	Belirlilik
Dengelenmemiş	0.7276	0.0675	0.2370	NaN	0.9981
ADASYN	**0.7495**	0.1908	0.0241	0.0427	0.9365
SMOTE	0.7441	0.1942	0.0243	0.0432	0.9356
BL SMOTE	0.7414	0.1983	0.0244	0.0434	0.9344
CL SMOTE	0.7346	0.2233	0.0245	0.0441	0.9269

Tablo-10. PhPMyadmin bütünleşik (metin + metrik) veri kümesi sınıflama sonuçları

Dengeleme	AUC	Duyarlılık	Kesinlik	F-skor	Belirlilik
Dengelenmemiş	0.7282	0.1519	0.2377	0.1835	0.9561
ADASYN	**0.7446**	0.2630	0.1271	0.1709	0.8359
SMOTE	0.7385	0.2578	0.1341	0.1756	0.8472
BL SMOTE	0.7367	0.2837	0.1342	0.1817	0.8329
CL SMOTE	0.7356	0.2570	0.1334	0.1752	0.8478

5 Sonuç ve Gelecek Çalışmalar

Bu çalışmada, yazılım zafiyet kestirimi probleminde veri dengeleme algoritmaları Aşırı Öğrenme Makinesi algoritmalarıyla birlikte ilk kez uygulanarak, metrik, metin ve bütünleşik (metin+metrik) nitelik modellerinden yararlanılarak 3 farklı kamuya açık veri kümesinde çok sayıda deney gerçekleştirilmiştir. Elden edilen deneysel sonuçlara göre, Aşırı Öğrenme Makinesi algoritmalarının bu problem özelinde potansiyele sahip olduğu sonucuna varılmış olup daha iyi performans elde edebilmek üzere ek araştırmaların yapılması planlanmıştır. Bu kapsamda, gelecek çalışmalarda daha farklı veri dengeleme algoritmalarının incelenmesi ve ek olarak daha farklı Aşırı Öğrenme Makinesi algoritmalarının deneylere entegre edilmesi söz konusu olabilecektir. Ayrıca, daha fazla veri kümesi elde edilmesi mümkün olduğu durumda, deneyler daha fazla sayıda veri kümesinde gerçekleştirilebilecektir.

Referanslar

1. Hovsepyan A, Scandariato R, Joosen W, Walden J Software vulnerability prediction using text analysis techniques. In: Proceedings of the 4th international workshop on Security measurements and metrics, 2012. ACM, pp 7-10

2. Jimenez M, Papadakis M, Le Traon Y Vulnerability prediction models: A case study on the linux kernel. In: 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), 2016. IEEE, pp 1-10

3. Stuckman J, Walden J, Scandariato R (2017) The Effect of Dimensionality Reduction on Software Vulnerability Prediction Models. Ieee T Reliab 66 (1):17-37. doi:10.1109/Tr.2016.2630503
4. Lagerström R, Baldwin C, MacCormack A, Sturtevant D, Doolan L Exploring the relationship between architecture coupling and software vulnerabilities. In: International Symposium on Engineering Secure Software and Systems, 2017. Springer, pp 53-69
5. Shin Y, Meneely A, Williams L, Osborne JA (2011) Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. Ieee T Software Eng 37 (6):772-787. doi:10.1109/Tse.2010.81
6. Walden J, Stuckman J, Scandariato R (2014) Predicting Vulnerable Components: Software Metrics vs Text Mining. Proc Int Symp Softw:23-33. doi:10.1109/Issre.2014.32
7. Sultana KZ (2017) Towards a Software Vulnerability Prediction Model using Traceable Code Patterns and Software Metrics. Ieee Int Conf Autom:1022-1025
8. Shin Y, Williams L (2013) Can traditional fault prediction models be used for vulnerability prediction? Empir Softw Eng 18 (1):25-59. doi:10.1007/s10664-011-9190-8
9. Morrison P, Herzig K, Murphy B, Williams L Challenges with applying vulnerability prediction models. In: Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, 2015. ACM, p 4
10. Pang Y, Xue X, Wang H Predicting vulnerable software components through deep neural network. In: Proceedings of the 2017 International Conference on Deep Learning Technologies, 2017. ACM, pp 6-10
11. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) SMOTE: Synthetic minority over-sampling technique. J Artif Intell Res 16:321-357. doi:DOI 10.1613/jair.953
12. Cieslak DA, Chawla NV, Striegel A (2006) Combating imbalance in network intrusion datasets. 2006 Ieee International Conference on Granular Computing:732-+. doi:Doi 10.1109/Grc.2006.1635905
13. Han H, Wang WY, Mao BH (2005) Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. Advances in Intelligent Computing, Pt 1, Proceedings 3644:878-887
14. He HB, Bai Y, Garcia EA, Li ST (2008) ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning. 2008 Ieee International Joint Conference on Neural Networks, Vols 1-8:1322-1328. doi:Doi 10.1109/Ijcnn.2008.4633969
15. Huang GB, Zhu QY, Siew CK (2006) Extreme learning machine: Theory and applications. Neurocomputing 70 (1-3):489-501
16. Huang G-B (2015) What are extreme learning machines? Filling the gap between Frank Rosenblatt's dream and John von Neumann's puzzle. Cogn Comput 7 (3):263-278
17. Huang GB (2014) An Insight into Extreme Learning Machines: Random Neurons, Random Features and Kernels. Cogn Comput 6 (3):376-390
18. ÖZÇALICI M (2017) AŞIRI ÖĞRENME MAKİNELERİ İLE HİSSE SENEDİ FİYAT TAHMİNİ. Hacettepe Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi 35 (1):67-88