Black Sea Journal of Artificial Intelligence

doi: 10.5281/zenodo.18213689

Open Access Journal
e-ISSN: 3108-4281

Araştırma Makalesi (Research Article)
Cilt 1 - Sayı 1: 15-24 / Haziran 2025
(Volume 1 - Issue 1: 15-24 / June 2025)

JAVASCRİPT FONKSİYONLARINDA GÜVENLİK AÇIKLARININ MAKİNE ÖĞRENMESİ YÖNTEMLERİYLE TESPİTİ

Hasan Hüseyin ALAV[1]*, Özgür TONKAL[1], Zafer CÖMERT[1]

[1]Samsun University, Faculty of Engineering and Natural Sciences, Department of Software Engineering, 55080, Samsun, Türkiye

Özet: JavaScript, modern web uygulamalarında temel bir programlama dili haline gelmiştir. Ancak dinamik ve zayıf tür denetimine sahip yapısı, ciddi güvenlik açıklarının ortaya çıkmasına neden olmaktadır. Bu çalışmada, JavaScript kod parçacıklarındaki güvenlik açıklarını otomatik olarak tespit etmek amacıyla geliştirilen makine öğrenmesi tabanlı yeni bir çerçeve olan JSVULNDETECT önerilmektedir. Geleneksel manuel kod inceleme yöntemlerinin hata yapmaya eğilimli ve zaman açısından verimsiz olması, otomatik ve ölçeklenebilir analiz araçlarına olan ihtiyacı artırmaktadır. Çalışma kapsamında, özenle ön işlenmiş ve denge sağlanmış bir veri kümesi üzerinde Random Forest, XGBoost, LightGBM ve Destek Vektör Makineleri (SVM) gibi toplamda on iki farklı makine öğrenmesi algoritması değerlendirilmiştir. Veri kümesindeki sınıf dengesizliği SMOTE-Tomek yöntemiyle giderilmiş; modellerin performansı ise Bayesyen hiperparametre optimizasyonu ile artırılmıştır. Ayrıca, Topluluk Öğrenmesi (Ensemble Learning) yaklaşımları olan Oylama (Voting) ve Yığınlama (Stacking) stratejileri uygulanmış, en yüksek doğruluk oranı %97,51 ile Stacking Sınıflandırıcısı kullanılarak elde edilmiştir. Önerilen çerçeve, çoklu öğrenme yaklaşımlarını birleştirerek güvenlik açığı tespit performansını artırmakta ve gerçek zamanlı analiz için web tabanlı bir arayüz sunmaktadır. Bu yönüyle çalışma, mevcut literatürde raporlanan sonuçları aşmakta ve geliştiriciler ile güvenlik analistleri için pratik bir analiz aracı olarak değerli bir katkı sunmaktadır.

Anahtar kelimeler: Siber güvenlik, Makine öğrenmesi, Javascript güvenliği, Güvenlik açığı tespiti, Kod güvenliği

Detection of Security Vulnerabilities in Javascript Functions Using Machine Learning Methods

Abstract: JavaScript has become an essential programming language in modern web applications; however, its dynamic and loosely-typed nature introduces considerable security vulnerabilities. This paper presents JSVULNDETECT, a novel machine learning-based framework designed to automatically detect vulnerabilities in JavaScript code segments. Unlike traditional manual review approaches, which are often error-prone and inefficient, our system offers a scalable, automated solution for static code analysis. We evaluate and compare twelve machine learning algorithms—such as Random Forest, XGBoost, LightGBM, and Support Vector Machines—using a carefully preprocessed and balanced dataset. To address class imbalance, we apply the SMOTE-Tomek method, and we optimize model performance via Bayesian hyperparameter search. Moreover, we integrate ensemble learning strategies (Voting and Stacking), achieving a maximum accuracy of 97.51% with the Stacking Classifier. Our work advances the field by combining multiple learning paradigms, enhancing detection performance, and providing a web-based interface for real-time analysis. The proposed framework not only surpasses previous results reported in similar studies but also serves as a practical tool for developers and security analysts in identifying potential threats in JavaScript functions.

Keywords: Cybersecurity, Machine learning, Javascript security, Vulnerability detection, Code security

***Sorumlu yazar (Corresponding author):** Samsun University, Faculty of Engineering and Natural Sciences, Department of Software Engineering, 55080, Samsun, Türkiye
E mail: hhsynalv@gmail.com (H. H. ALAV)

Hasan Hüseyin ALAV https://orcid.org/0009-0001-1812-2706
Özgür TONKAL https://orcid.org/0000-0001-7219-9053
Zafer CÖMERT https://orcid.org/0000-0001-5256-7648

Gönderi: 14 Nisan 2025 **Received:** April 14, 2025
Kabul: 01 Mayıs 2025 **Accepted:** May 01, 2025
Yayınlanma: 15 Haziran 2025 **Published:** June 15, 2025

Cite as: Alav, H. H., Tonkal, Ö., & Cömert, Z. (2025). Detection of security vulnerabilities in Javascript functions using machine learning methods. *Black Sea Journal of Artificial Intelligence*, *1*(1), 15–24.

1. Giriş

Web tabanlı uygulamaların hızla yaygınlaşmasıyla birlikte, JavaScript programlama dili hem istemci tarafında (client-side) hem de sunucu tarafında (server-side) en çok kullanılan teknolojilerden biri haline gelmiştir. Özellikle Node.js'in yaygınlaşmasıyla birlikte, JavaScript yalnızca dinamik web sayfalarının değil, aynı zamanda sunucu taraflı servislerin de temel yapı taşı olmuştur (Ferenc vd., 2019). Bu yaygın kullanım, beraberinde ciddi güvenlik sorunlarını da gündeme getirmiştir. JavaScript'in dinamik ve zayıf biçimlendirilmiş yapısı, geliştiricilere büyük esneklik sağlarken; aynı zamanda kod enjeksiyonu, siteler arası betik çalıştırma [Cross-Site Scripting (XSS)] ve siteler arası istek sahteciliği [Cross-Site Request Forgery (CSRF)] gibi birçok saldırı türü için de uygun bir zemin hazırlamaktadır (Breiman, 2001; Friedman, 2001).

Geleneksel güvenlik analiz yöntemleri genellikle manuel kod incelemelerine veya klasik statik analiz araçlarına dayanmaktadır. Ancak bu yöntemler hem zaman alıcıdır hem de insan hatasına açık olması nedeniyle yüksek güvenilirlik sağlayamamaktadır (Geurts vd., 2006). Son yıllarda, yapay zekâ ve özellikle makine öğrenmesi tabanlı yaklaşımlar, güvenlik zafiyetlerinin otomatik

tespiti konusunda umut vaat eden çözümler sunmaya başlamıştır. Bu yöntemler, yazılım kodlarını istatistiksel olarak inceleyerek olası açıklara dair tahminlerde bulunabilmekte ve sürekli güncellenebilir modeller sayesinde uyarlanabilir (adaptif) güvenlik analizleri gerçekleştirebilmektedir (Hosmer ve Lemeshow, 2000).

Literatürde bu alanda yapılan çalışmalarda, farklı makine öğrenmesi tekniklerinin JavaScript, C/C++ ve Python gibi programlama dillerindeki güvenlik açıklarının sınıflandırılması amacıyla kullanıldığı görülmektedir. Örneğin, Ferenc vd. (2019) JavaScript fonksiyonları üzerinde çeşitli makine öğrenmesi algoritmalarını karşılaştırarak güvenlik açığı tespiti gerçekleştirmiştir. Benzer şekilde, Chen ve Guestrin (2016) tarafından geliştirilen XGBoost algoritması, yüksek performanslı ağaç tabanlı öğrenme yöntemleriyle birçok sınıflandırma probleminde başarı göstermiştir. Ancak bu çalışmaların çoğu, sınırlı sayıda model ve parametrik yapı ile çalıştığından, sistemin genellenebilirliğini ve doğruluk oranlarını sınırlamaktadır.

Bu çalışmanın amacı, JavaScript fonksiyonları içerisindeki potansiyel güvenlik açıklarını otomatik olarak tespit edebilen, yüksek doğruluk ve genellenebilirliğe sahip bir makine öğrenmesi sistemi geliştirmektir. Özgün katkı olarak:

- 12 farklı sınıflandırma modeli sistematik biçimde karşılaştırılmış,
- SMOTE-Tomek ile veri dengesizliği giderilmiş,
- Hyperopt algoritmasıyla hiperparametre optimizasyonu yapılmış,
- Sonuçların kararlılığı Voting ve Stacking gibi topluluk (ensemble) yöntemleri ile güçlendirilmiştir.

Bu yapı sayesinde, doğruluk oranı %97,51 seviyesine ulaşmış ve literatürdeki benzer çalışmalardan daha yüksek performans elde edilmiştir. Ayrıca geliştirilen JSVULNDETECT sistemi, kullanıcı dostu bir web arayüzü ile JavaScript kodlarını otomatik analiz edebilmekte; böylece hem akademik hem de endüstriyel ortamlarda kullanılabilecek bir güvenlik değerlendirme aracı sunmaktadır.

2. Literatür Özeti

Bu bölümde, JavaScript tabanlı yazılımlarda güvenlik açığı tespitine yönelik çalışmalar kapsamlı bir şekilde incelenmiş, makine öğrenmesi yöntemlerinin bu alandaki kullanımı değerlendirilmiştir. İlk olarak, güvenlik açığı tespiti üzerine gerçekleştirilen önceki çalışmalar ele alınarak mevcut yöntemlerin güçlü ve zayıf yönleri tartışılmıştır. Ardından, JavaScript programlama dili özelinde geliştirilen makine öğrenmesi uygulamaları incelenmiş ve bu yaklaşımların siber güvenlik bağlamındaki katkıları ortaya konmuştur. Son olarak, bu çalışmanın literatürdeki yeri belirlenerek mevcut boşluklar ve araştırmanın özgün yönleri vurgulanmıştır. Bu bölüm, önerilen sistemin alana nasıl bir katkı sunduğunu teorik temeller üzerinden açıklamayı amaçlamaktadır.

2.1. Güvenlik Açığı Tespiti Üzerine Yapılan Çalışmalar

Yazılım güvenliği, özellikle web uygulamalarının yaygınlaşmasıyla birlikte hem akademik dünyada hem de endüstride büyük ilgi gören bir araştırma alanı haline gelmiştir. Geleneksel güvenlik testleri genellikle manuel kod incelemesine veya temel statik analiz araçlarına dayanmakta; ancak bu yöntemler, yüksek hata oranı ve düşük ölçeklenebilirlik gibi sınırlamalar nedeniyle günümüz yazılım sistemlerinin ihtiyaçlarını karşılamada yetersiz kalmaktadır (Hosmer ve Lemeshow, 2000). Bu nedenle, güvenlik açıklarının otomatik olarak tespit edilmesine yönelik yapay zekâ ve makine öğrenmesi tabanlı yaklaşımlar son yıllarda öne çıkmıştır.

Gradient Boosting yöntemi, kod örüntülerindeki karmaşık ilişkileri modellemede başarılı sonuçlar elde etmiş ve güvenlik açığı tespitinde sıkça başvurulan modellerden biri olmuştur (Friedman, 2001). Benzer şekilde, Random Forest algoritması yüksek varyansa sahip verilerde dengeli sonuçlar vermesi ve aşırı öğrenmeye (overfitting) karşı dayanıklı yapısı sayesinde yazılım güvenliği alanında tercih edilen algoritmalardan biri haline gelmiştir (Breiman, 2001).

XGBoost algoritması, büyük boyutlu ve dengesiz veri kümeleri üzerinde etkili performans sunarak yazılım güvenliği bağlamında öne çıkmıştır (Chen ve Guestrin, 2016). Bununla birlikte, Extremely Randomized Trees modeli, klasik karar ağaçlarının rastgeleleştirilmiş versiyonu ile daha esnek bir yapı sunarak güvenlik açığı tespiti gibi karmaşık problemler için alternatif bir çözüm getirmiştir (Geurts vd., 2006).

Ferenc vd. (2019), JavaScript fonksiyonları üzerinde gerçekleştirdikleri çalışmalarında, çeşitli makine öğrenmesi algoritmalarının güvenlik açığı tespitindeki başarılarını karşılaştırmalı olarak analiz etmişlerdir. Bu çalışmada, karar ağaçları, Destek Vektör Makineleri (SVM) ve Naive Bayes gibi yöntemlerin farklı doğruluk oranları sergilediği ve sınıflandırma başarısının model seçimi kadar veri ön işleme tekniklerine de bağlı olduğu sonucuna ulaşılmıştır.

Bu alanda yapılan çalışmalar, genel olarak sınırlı sayıda modelle ve sınırlı parametreler üzerinde yoğunlaşmıştır. Ayrıca, verinin dengesiz dağılımı, özellik mühendisliği (feature engineering) süreçlerinin belirsizliği ve kullanılan metriklerin çeşitliliği, sonuçların karşılaştırılabilirliğini zorlaştırmaktadır. Bu durum, daha kapsamlı, çoklu model tabanlı ve entegratif sistemlere olan ihtiyacı doğurmuştur.

2.2. JavaScript Üzerine Makine Öğrenmesi Uygulamaları

JavaScript'in hem istemci hem de sunucu tarafında yaygın biçimde kullanılması, bu dili hedef alan güvenlik açıklarının da daha sık karşılaşılmasına yol açmaktadır. JavaScript, özellikle Belge Nesne Modeli (Document Object Model [DOM]) manipülasyonu, kullanıcı girişleri ve üçüncü parti entegrasyonlar gibi işlemlerde yüksek risk taşıyan bir yapıya sahiptir. Bu nedenle, bu dile özgü

güvenlik açıklarının (örneğin, XSS, DOM tabanlı enjeksiyonlar ve güvensiz API kullanımları) tespit edilmesi için özel çalışmalar yapılmaktadır.

Ferenc vd. (2019), JavaScript fonksiyonları üzerinde gerçekleştirdikleri bir çalışmada, statik analiz araçlarıyla birlikte makine öğrenmesi yöntemlerini entegre ederek güvenlik açığı içeren fonksiyonların sınıflandırılmasını gerçekleştirmiştir. Bu çalışmada, farklı sınıflandırma algoritmalarının performansları karşılaştırılmış ve doğru özellik mühendisliğinin (feature engineering) başarıyı belirgin şekilde etkilediği vurgulanmıştır.

McCallum ve Nigam (1998) tarafından önerilen Naive Bayes modelinin metin tabanlı analizlerdeki başarısı, JavaScript kodları üzerindeki analizlerde de uygulanabilirliğini göstermiştir. JavaScript kodları doğrudan metin olarak değerlendirilebildiğinden, n-gram tabanlı özellik çıkartımı ile bu tür algoritmalar etkili şekilde kullanılabilmektedir.

Son yıllarda, derin öğrenme modelleri de JavaScript güvenliği bağlamında araştırılmaya başlanmıştır. Örneğin, kodu Soyut Sözdizim Ağacı (Abstract Syntax Tree [AST]) yapısına dönüştürerek kodun yapısal özelliklerini öğrenmeye çalışan projeler, Evrişimsel Sinir Ağı (Convolutional Neural Network [CNN]) veya Tekrarlayan Sinir Ağı (Recurrent Neural Network [RNN]) tabanlı yaklaşımlar kullanarak zararlı davranış örüntülerini sınıflandırma hedefiyle geliştirilmektedir. Ancak bu tür modeller yüksek hesaplama gücü gerektirdiğinden, pratikte kullanımı sınırlı kalabilmektedir.

Son yıllarda yazılım güvenlik açıklarının tespitine yönelik makine öğrenmesi tabanlı yaklaşımlar önemli ilerlemeler kaydetmiştir. Harzevili vd. (2023), 2011–2022 yılları arasındaki 67 çalışmayı inceleyerek veri dengesizliği, özellik çıkarımı, model genellemesi ve açıklanabilirlik konularında süregelen sorunları vurgulamış ve yeni nesil yöntemlere yönelim ihtiyacını ortaya koymuştur. JavaScript özelinde geliştirilen modern yöntemler, geleneksel istatistiksel sınıflandırıcıların ötesine geçmektedir. Örneğin PeerJ Computer Science'ta yayımlanan 2024 tarihli bir çalışma, JavaScript fonksiyonlarındaki güvenlik zafiyetlerini tespit etmek amacıyla çok katmanlı konvolüsyonel sinir ağlarını istifleyerek (stacking-CNN) önemli ölçüde doğruluk artışı sağlamıştır. Benzer şekilde, Liu vd. (2023) tarafından sunulan MFXSS modeli, kaynak kodun soyut sözdizim ağaçları (AST) ve kontrol akış grafikleri (CFG) üzerinden çoklu özellik füzyonu gerçekleştirerek, özellikle XSS açıklarının tespitinde yüksek performans sergilemiştir. Bu güncel çalışmalar, yazılım güvenlik açığı tespitinde derin öğrenme ve grafik tabanlı modelleme tekniklerinin giderek daha fazla benimsendiğini ve geleneksel yöntemlere kıyasla daha üstün sonuçlar sunduğunu göstermekte olup, bu çalışmanın da benzer bir motivasyonu desteklediğini teyit etmektedir.

Güncel çalışmalar, genellikle tek bir modelin performansına odaklanmakta ve topluluk öğrenme (ensemble) yöntemlerinin potansiyelinden yeterince yararlanmamaktadır. Ayrıca, hiperparametre optimizasyonu, veri ön işleme ve sınıf dengesizliği gibi konular çoğu zaman yüzeysel ele alınmakta; bu da sistemlerin genelleme kabiliyetini düşürmektedir.

Bu bağlamda, JavaScript kodlarının güvenlik analizi için optimize edilmiş, çoklu model karşılaştırmalı ve topluluk öğrenme yöntemlerinden faydalanan sistemlere olan ihtiyaç giderek artmaktadır. Bu çalışmanın önemli motivasyon kaynaklarından biri de bu eksikliği giderecek yapısal bir çözüm sunabilmektir.

Literatürdeki mevcut çalışmalar incelendiğinde, güvenlik açığı tespiti konusunda makine öğrenmesi tabanlı yaklaşımların giderek daha fazla tercih edildiği görülmektedir. Bununla birlikte, bu çalışmaların büyük çoğunluğu sınırlı sayıda model üzerinde yoğunlaşmakta; veri dengesizliği, hiperparametre optimizasyonu ve topluluk öğrenme (ensemble) yöntemlerinin birlikte uygulanması gibi önemli boyutlar genellikle göz ardı edilmektedir. Özellikle JavaScript özelinde yapılan çalışmalarda, sınıflandırma modelleri ile güvenlik açığı içeren kodların tespiti hedeflenmiş olsa da, bu sistemlerin çoğu tekil algoritmalarla geliştirilmiş ve geniş kapsamlı karşılaştırmalar yapılmamıştır.

Ayrıca, literatürdeki bazı çalışmalar sınıflandırma başarısını değerlendirmek için yalnızca doğruluk (accuracy) metriğini kullanmakta; bu da özellikle dengesiz veri kümelerinde yanıltıcı sonuçlara yol açabilmektedir. Topluluk öğrenme yöntemlerinin (özellikle çok katmanlı yığınlama [Stacking] gibi yaklaşımların) bu bağlamda sunduğu avantajlar yeterince araştırılmamış; kullanılan veri ön işleme tekniklerinin (örneğin, Sentetik Azınlık Aşırı Örnekleme Tekniği ve Tomek bağlantıları [SMOTE-Tomek] gibi hibrit dengeleme yöntemleri) sınıflandırma performansı üzerindeki etkisi ayrıntılı olarak değerlendirilmemiştir.

Bu çalışma, literatürdeki bu boşlukları doldurmayı hedeflemektedir. Özgün yönleri şunlardır:

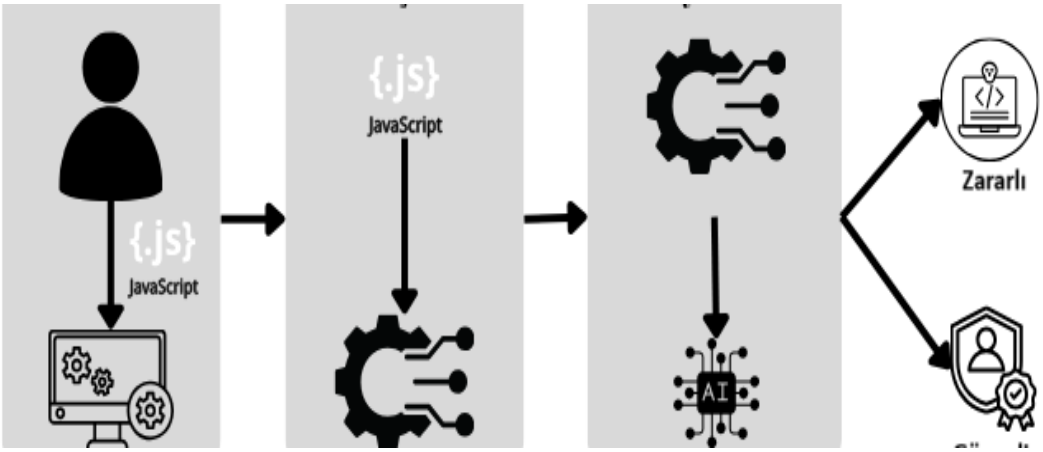
- 12 farklı makine öğrenmesi modeli detaylı şekilde karşılaştırılmıştır.
- Her bir model için Bayes optimizasyonu (Bayesian Optimization) tabanlı hiperparametre ayarı yapılmıştır.
- Veri kümesindeki sınıf dengesizliği SMOTE-Tomek yöntemi ile ele alınmıştır.
- Hem oy birliği temelli (Voting) hem de yığınlama temelli (Stacking) topluluk öğrenme yöntemleri kullanılarak, modellerin yatay ve dikey güçlü yönleri bir araya getirilmiştir.
- Elde edilen sistem, %97,51 doğruluk oranıyla önceki çalışmalardan daha iyi performans göstermiştir.
- Ayrıca, sistemin web arayüzü ile kullanılabilir hale getirilmiş olması, akademik prototipten çok endüstriyel geçerliliği olan bir yapıya işaret etmektedir.

Bu yönleriyle çalışma hem yöntemsel bütünlüğü hem de uygulamaya dönük yapısıyla mevcut literatüre anlamlı bir katkı sunmaktadır.

3. Materyal ve Yöntem

Bu bölümde çalışmanın uygulama adımları, kullanılan veri seti, veri ön işleme süreci, özellik çıkarımı, veri dengeleme yöntemleri ve sınıflandırma modelleri ayrıntılı olarak sunulmaktadır. Genel sistem mimarisi Şekil 1'de gösterilmektedir.

Şekil 1' de geliştirilen sistemin genel işleyiş süreci üç aşamalı olarak sunulmuştur. İlk aşamada, kullanıcı sisteme analiz edilmek üzere bir JavaScript kodu yüklemektedir. Bu kod, ikinci aşamada sistem tarafından incelenerek makine öğrenmesi için gerekli olan temel özellikler (nitelikler) otomatik olarak çıkarılmaktadır. Son aşamada ise, bu özellikler kullanılarak kodun güvenlik açığı içerip içermediği analiz edilmekte ve kullanıcıya bilgi verilmektedir. Böylece geliştiriciler, yazdıkları kodlarda potansiyel güvenlik açıklarını hızlı ve etkili bir şekilde tespit etme imkânına sahip olmaktadır.



Şekil 1. Akış diyagramı.

3.1. Veri Seti

Bu çalışmada kullanılan veri seti, Ferenc ve arkadaşları tarafından geliştirilen ve JavaScript fonksiyonlarının güvenlik zafiyeti içerip içermediğini sınıflandırmayı amaçlayan etiketli bir veri kümesidir (Ferenc vd., 2019). Veri seti toplamda 10.000'den fazla JavaScript fonksiyonu içermekte olup, her bir fonksiyon için 45'ten fazla statik özellik (örneğin: kod uzunluğu, döngü karmaşıklığı, hata geçmişi, ifade sayısı vb.) hesaplanmıştır. Fonksiyonlar, güvenli (non-vulnerable) ve güvensiz (vulnerable) olmak üzere iki sınıfa ayrılmıştır. Ancak sınıflar arasında ciddi bir dengesizlik bulunmaktadır: veri setinde güvenli fonksiyonlar yaklaşık %85'lik bir çoğunlukla temsil edilirken, güvensiz fonksiyonlar yalnızca %15 oranındadır. Bu dengesizlik, modelin güvensiz fonksiyonları öğrenmesini zorlaştırmakta ve sınıflandırma başarımını düşürmektedir.

Bu nedenle, veri seti üzerinde Sentetik Azınlık Aşırı Örnekleme Tekniği (SMOTE) ile Tomek bağlantılarının birleştirilmesinden oluşan SMOTE-Tomek yöntemi uygulanarak sınıf dengesi sağlanmış ve öğrenme algoritmalarının pozitif sınıf (güvensiz fonksiyon) performansı artırılmıştır. Veri seti, hem temel hem de türetilmiş özellikleri içermesi sayesinde, yalnızca makine öğrenmesi algoritmalarını test etmek için değil, aynı zamanda özellik mühendisliği (feature engineering) süreçlerini de değerlendirmek açısından zengin bir kaynak sunmaktadır.

3.2. Özellik Çıkarımları

Veri setinde bulunan özellikler, JavaScript fonksiyonlarının güvenlik açıklarını tespit etmek için kritik öneme sahiptir. Bu çalışmada kullanılan bazı temel özellikler Tablo1'de gösterilmiştir.

Tablo 1. Kullanılan özellikler

Özellik Kodu	Açıklama
LOC (Lines of Code)	Bir fonksiyonun toplam kod satırı sayısı
CYCL_DENS(Cyclomatic Density)	Fonksiyonun döngüsel karmaşıklık yoğunluğu
HBUGS (Historical Bugs)	Fonksiyonun geçmişte içerdiği hata sayısı
NOS(Number of Statements)	Fonksiyon içerisindeki toplam ifade sayısı
HDIFF(Historical Diff)	Fonksiyonun geçmişte yapılan değişikliklerin sayısı

Bu özellikler, güvenlik açıklarını tespit etmek için kullanılan temel metriklerdir. Ayrıca, bu çalışmada bu temel özelliklerden türetilen yeni özellikler de Tablo 2'de gösterilmiştir.

Bu özellik çıkarımları, veri setinin zenginleştirilmesi ve daha hassas analizler yapılması için kritik öneme sahiptir.

3.3. Veri Dengesizliğinin Giderilmesi

Veri dengesizliği problemi, güvenlik açığı bulunmayan örneklerin veri setinde baskın olması nedeniyle ortaya çıkmaktadır. Bu sorunu çözmek amacıyla, Sentetik Azınlık Aşırı Örnekleme Tekniği (SMOTE ve Tomek bağlantıları (Tomek Links) birlikte kullanılmıştır. SMOTE, azınlık sınıfa ait (güvensiz) örnekleri sentetik olarak çoğaltarak sınıf dengesizliğini azaltırken; Tomek bağlantıları, birbirine çok yakın fakat farklı sınıflara ait

örnek çiftlerini tespit ederek sınıflar arası karışıklığı azaltır ve daha net karar sınırları elde edilmesini sağlar. Bu iki yöntemin birleşimi olan SMOTE-Tomek, modelin özellikle azınlık sınıfa (vulnerable functions) dair öğrenme başarımını artırarak genel sınıflandırma doğruluğunu iyileştirmeyi hedeflemiştir.

Tablo 2. Türetilen özellikler

Özellik Kodu	Açıklama
CYCLE_DENS_LOC	CYCL_DENS / LOC oranı, fonksiyonun satır başına döngüsel karmaşıklık yoğunluğunu gösterir.
HBUGS_LOC	HBUGS / LOC oranı, fonksiyonun satır başına hata oranını gösterir.
HBUGS_NOS	HBUGS / NOS oranı, fonksiyonun ifade başına hata oranını gösterir.
LOC_NOS	LOC / NOS oranı, fonksiyonun uzunluğunu ifade başına gösterir.
HBUGS_HDIFF	HBUGS / HDIFF oranı, yapılan değişiklikler başına hata oranını gösterir

3.4. Model Seçimi

Bu çalışmada, JavaScript kodlarında güvenlik açığı tespiti problemini farklı perspektiflerden ele alabilmek amacıyla 12 farklı makine öğrenmesi algoritması değerlendirilmiştir. Bu modeller;

- Doğrusal sınıflandırıcılar: Lojistik Regresyon (Logistic Regression), Destek Vektör Makineleri (SVM)
- Karar ağacı tabanlı yaklaşımlar: Rastgele Ormanlar (Random Forest; Breiman, 2001), Aşırı Gradyan Artırmalı Karar Ağaçları (XGBoost; Chen & Guestrin, 2016), LightGBM (Ke vd., 2017)
- İstatistiksel ve örnek tabanlı yöntemler: En Yakın Komşu Algoritması (KNN – K-Nearest Neighbors), Naive Bayes (McCallum ve Nigam, 1998)

gibi çeşitli öğrenme paradigmalarını temsil etmektedir. Her modelin belirli avantajları bulunmakta olup, model performansları ve davranışları Bölüm 4.1'de detaylı olarak sunulmuştur.

3.5. Hiperparametre Optimizasyonu

Makine öğrenmesi modellerinin doğruluk, hassasiyet ve genellenebilirliğini artırmak amacıyla hiperparametre ayarları yapılmıştır. Bu kapsamda, geleneksel ızgara araması (grid search) gibi yöntemlere kıyasla daha az hesaplama maliyetiyle daha etkili sonuçlar sunan Bayes optimizasyonu (Snoek vd., 2012) tercih edilmiştir. Optimizasyon sürecinde, literatürde önerilen hiperparametre aralıkları kullanılmış ve her model için 50 iterasyonluk bir arama gerçekleştirilmiştir. Seçilen aralıklar ve dağılımlar, Tablo 3'te detaylı olarak verilmiştir.

3.6. Ensemble Yöntemleri

Modelin genel başarımını artırmak, farklı algoritmaların güçlü yönlerinden faydalanmak ve tekil modellerin zaaflarını dengelemek amacıyla iki farklı topluluk (ensemble) yöntemi uygulanmıştır:

- Voting Classifier: Birden fazla modelin tahminleri oy çokluğu ile birleştirilerek nihai karar verilir (Dietterich, 2000).
- Stacking Classifier: Birinci seviye öğrenicilerden (base learners) elde edilen tahmin çıktıları, ikinci seviyedeki bir öğreniciye (meta-learner) giriş olarak verilerek daha kompleks karar mekanizması kurulmasını sağlar (Wolpert, 1992).

Bu yöntemlerin uygulanma adımları, doğrulama süreçleri ve elde edilen başarı metrikleri Bölüm 4.3'te kapsamlı şekilde sunulmuştur.

Tablo 3. Model hiperparametre aralıkları

Algoritma	Parametre	Aralık	Dağılım	Açıklama
KNN	n_neighbors	1 – 20	Uniform	Komşu sayısı
Random Forest	n_estimators	50 – 200	Uniform	Ağaç sayısı
	max_depth	5 – 30	Uniform	Ağaç derinliği
Gradient Boosting	n_estimators	50 – 200	Uniform	Boost sayısı
	learning_rate	0,01 – 0,3	Log-uniform	Öğrenme hızı
AdaBoost	n_estimators	50 – 200	Uniform	Ağırlıklı öğrenici sayısı
Bagging	n_estimators	10 – 200	Uniform	Bootstrap örnek sayısı
Extra Trees	n_estimators	10 – 200	Uniform	Rastgele ağaç sayısı
Logistic Regression	C	1e-6 – 1e+6	Log-uniform	Düzenleme katsayısı
SVM	C	1e-6 – 1e+6	Log-uniform	Margin kontrolü
Decision Tree	max_depth	3 – 30	Uniform	Ağaç derinliği
Naive Bayes	var_smoothing	1e-9 – 1e-5	Log-uniform	Sayısal kararlılık
XGBoost	n_estimators	50 – 200	Uniform	Boost sayısı
	learning_rate	0,01 – 0,3	Log-uniform	Öğrenme hızı
LightGBM	n_estimators	50 – 200	Uniform	Boost sayısı
	learning_rate	0,01 – 0,3	Log-uniform	Öğrenme hızı

4. Makine Öğrenme Modeli

Bu bölümde, JavaScript fonksiyonlarında güvenlik açıklarını tespit etmek amacıyla geliştirilen makine öğrenmesi tabanlı yaklaşımlar ayrıntılı olarak ele alınmaktadır. İlk olarak, sınıflandırma sürecinde kullanılan temel algoritmalar tanıtılmış; ardından veri setinin eğitime hazırlanması süreci açıklanmıştır. Son olarak, bireysel modellerin yanı sıra, doğruluk oranını artırmak amacıyla uygulanan Voting ve Stacking gibi topluluk (ensemble) öğrenme yöntemleri detaylı şekilde değerlendirilmiştir.

4.1. Kullanılan Modeller

Bu çalışmada tercih edilen makine öğrenmesi modelleri, farklı öğrenme paradigmalarını temsil edecek şekilde dikkatle seçilmiştir. Amaç, her bir modelin güçlü yönlerinden faydalanarak, güvenlik açığı tespiti gibi karmaşık bir problemi daha isabetli çözebilecek bir sistem oluşturmaktır.

Örneğin, Lojistik Regresyon (Logistic Regression) ve Destek Vektör Makineleri (SVM) gibi modeller doğrusal sınıflandırma yetenekleriyle bilinir. Lojistik Regresyon, verilerin lineer olarak ayrılabildiği durumlarda hızlı ve etkili sonuçlar sunarken (Hosmer ve Lemeshow, 2000; Şahin, 2025), SVM ise yüksek boyutlu uzayda karar sınırlarını optimize ederek ayrım performansını artırır (Cortes ve Vapnik, 1995).

Buna karşın, Karar Ağacı tabanlı modeller olan Rastgele Ormanlar (Random Forest; Breiman, 2001), XGBoost (Chen ve Guestrin, 2016) ve LightGBM (Ke vd., 2017), doğrusal olmayan ilişkileri modellemede oldukça başarılıdır. Özellikle Random Forest, overfitting riskini azaltan "bagging" stratejisi sayesinde yüksek varyansa sahip veri kümelerinde kararlı sonuçlar üretmektedir. XGBoost ve LightGBM ise boosting tabanlı yaklaşımlarıyla model hatalarını kademeli olarak azaltmakta ve yüksek doğruluk sağlamaktadır.

Diğer yandan, Naive Bayes gibi olasılıksal modeller, özellikler arasında koşulsuz bağımsızlık varsayımı ile çalışır (McCallum ve Nigam, 1998). Bu yöntem, özellikle metin sınıflandırma gibi alanlarda etkili olsa da, karmaşık bağımlılık ilişkilerini yakalamakta sınırlıdır. KNN gibi örnek-tabanlı algoritmalar ise, düşük boyutlu ve dengeli veri kümelerinde başarılı olmasına rağmen, yüksek boyut veya dengesiz sınıf dağılımlarında performans kaybı yaşayabilir.

Bu çeşitliliği avantaja çevirmek amacıyla bu çalışmada, ensemble yöntemleri olan Voting Classifier ve Stacking Classifier tercih edilmiştir. Ensemble yaklaşımlar, farklı modellerin karar mekanizmalarını birleştirerek genel hata oranını azaltmayı hedefler (Dietterich, 2000). Özellikle Stacking yöntemi, birinci seviye öğrenicilerden (örneğin: ağaç tabanlı ve doğrusal modeller) elde edilen çıktıları ikinci bir öğreniciye (meta-learner) aktarır ve böylece modellerin birbirinin eksik yönlerini tamamlamasına olanak sağlar (Wolpert, 1992).

Burada "yatay" ve "dikey" katkılar kavramsallaştırması kullanılabilir: bazı modeller (örneğin karar ağaçları), verinin iç yapısını ayrıntılı biçimde çözümleyerek dikey analiz yaparken; bazı modeller (örneğin Logistic Regression, SVM), daha genelleyici karar sınırları oluşturarak yüzeysel fakat geniş kapsamlı katkılar sağlar. Bu farklı stratejilerin bir araya getirilmesi, güvenlik açığı tespiti gibi çok boyutlu problemler için yüksek doğruluk ve genelleme kabiliyeti sunar.

Sonuç olarak, sistemde kullanılan modeller hem farklı karar stratejileri hem de farklı matematiksel temelleri ile birbirini tamamlamakta; bu da Stacking gibi yöntemlerin neden en yüksek başarıyı sağladığını açıklamaktadır (Wolpert, 1992).

4.2. Eğitime Hazırlık

Veri seti üzerinde yapılan ön işlemlerin ardından, sınıflandırma modellerinin performansını artırmak için hiperparametre optimizasyonu gerçekleştirilmiştir. Hiperparametreler, modelin öğrenme sürecinde ayarlanması gereken ve genel performansı doğrudan etkileyen parametrelerdir. Bu çalışmada, 12 farklı model için en uygun hiperparametreleri belirlemek amacıyla Bayes Arama (Bayesian Optimization) yöntemi kullanılmıştır. Bayes Arama, grid search ve rastgele arama (random search) yöntemlerinden farklı olarak, önceki denemelerden elde edilen bilgileri kullanarak arama alanını sistematik şekilde daraltır ve böylece optimal parametreleri daha hızlı ve etkili biçimde bulur (Snoek vd., 2012). Her model için belirlenen hiperparametre aralıkları Tablo 3'te sunulmuş olup, Bayes Arama yöntemi 50 iterasyon boyunca en iyi kombinasyonu aramıştır.

Hiperparametre optimizasyon sürecinde, her algoritma için anlamlı ve etkili olduğu bilinen hiperparametreler belirlenmiştir. Örneğin, n_neighbors parametresi yalnızca KNN modeli için uygulanabilirken, n_estimators parametresi Random Forest, XGBoost ve LightGBM gibi topluluk öğrenme (ensemble learning) modelleri için kritik öneme sahiptir. Parametre aralıkları, daha önce yapılan çalışmalar (Wolpert, 1992; Breiman, 2001; Chen ve Guestrin, 2016; Ke vd., 2017) ve ampirik gözlemler doğrultusunda belirlenmiş; dağılım türü ise parametrenin ölçeğine göre uniform ya da log-uniform olarak seçilmiştir.

Bayes optimizasyonu, 50 iterasyon boyunca bu hiperparametre uzayını tarayarak her model için en yüksek doğruluğu sağlayan kombinasyonu tespit etmiştir. Bu süreçte değerlendirme ölçütü olarak doğruluk (accuracy), hassasiyet (precision), duyarlılık (recall) ve F1 skoru dikkate alınmıştır.

4.3. Model Eğitimi

4.3.1. Voting classifier

Voting Classifier, birden fazla makine öğrenmesi modelinin tahminlerini birleştirerek nihai kararı verir (Dietterich, 2000). Bu çalışmada, KNN, Random Forest, Gradient Boosting, AdaBoost, Bagging, Extra Trees, Logistic Regression, SVM, Decision Tree, Naive Bayes, XGBoost ve LightGBM modelleri, SMOTE-Tomek ile dengelenmiş veri seti üzerinde eğitilmiştir. Her model için Bayes Arama yöntemiyle en iyi hiperparametreler belirlendikten sonra, bu modeller Voting Classifier içinde

bir araya getirilmiş, çapraz doğrulama yöntemiyle performansları değerlendirilmiş ve nihai model test veri seti üzerinde eğitilerek performans metrikleri (doğruluk, hassasiyet, duyarlılık ve F1 skoru) hesaplanmıştır. Voting Classifier, farklı modellerin güçlü yönlerini birleştirerek genel performansı artırmayı amaçlamakta olup, bu çalışmada hard voting yaklaşımı kullanılmıştır.

4.3.2. Stacking classifier

Stacking Classifier, birinci seviye öğrenicilerin tahminlerini ikinci seviye öğreniciye (meta-öğrenici) aktararak nihai kararı verir (Wolpert, 1992). Bu çalışmada, KNN, Random Forest, Gradient Boosting, AdaBoost, Bagging, Extra Trees, Logistic Regression, SVM, Decision Tree, Naive Bayes, XGBoost ve LightGBM modelleri birinci seviye öğrenici olarak kullanılmış; SMOTE-Tomek ile dengelenmiş veri seti üzerinde eğitilmiştir. Birinci seviye modellerin tahminleri, ikinci seviye öğrenici olarak seçilen LightGBM'e aktarılmış; çapraz doğrulama yöntemiyle performans değerlendirilmiş ve test veri seti üzerinde nihai performans metrikleri hesaplanmıştır. Bu yöntem, farklı modellerin güçlü yönlerini birleştirerek daha yüksek doğruluk ve genelleme kapasitesi sağlamayı hedefler.

5. Bulgular

Bu bölümde model eğitimi aşaması sonrası kullanılan modelin doğrulama ve test işlemleri gerçekleştirilmiştir. Gerçekleştirilen testler sonucunda elde edilen sonuçlar karşılaştırmalı olarak verilmiştir.

5.1. Çapraz Doğrulama

Model doğrulama sürecinde, veri seti 5 katlı Stratified K-Fold yöntemiyle eşit parçalara bölünmüş ve her fold sırayla test seti olarak kullanılmıştır. Bu yöntem, sınıf dağılımını dengeli tutarken, cross_val_score fonksiyonu ile modelin doğruluk (accuracy) skorunu hesaplamayı sağlamıştır. Voting ve Stacking Classifier modelleri için uygulanan bu çapraz doğrulama sonucunda, modelin genel performansını yansıtan ortalama doğruluk skoru 0,9397 elde edilmiştir.

5.2. Test Performansı

Model eğitimi ve doğrulama tamamlandıktan sonra, eğitimde kullanılmayan test veri seti üzerinde modellerin performansı değerlendirildi. Voting ve Stacking Classifier modelleri, SMOTE-Tomek ile dengelenmiş eğitim verisiyle eğitildikten sonra, test veri setinde tahminlerde bulunarak doğruluk (accuracy), hassasiyet (precision), duyarlılık (recall) ve F1 skoru gibi metrikler hesaplandı.

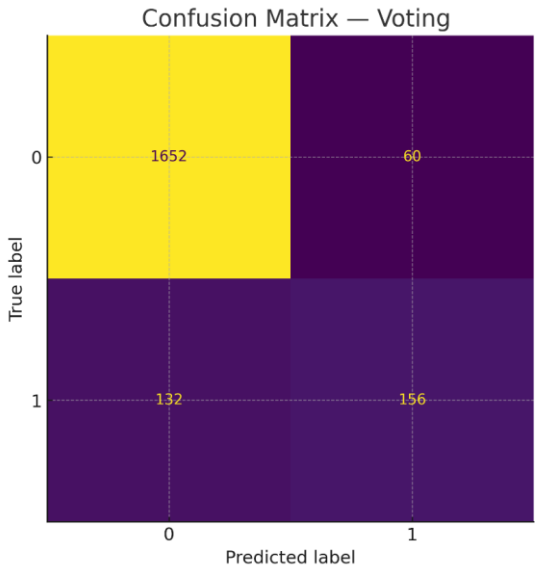
Tablo 4'de verilen tüm model performansları değerlendirildiğinde, Extra Trees modeli en yüksek doğruluk oranını sağlamıştır. Bu modelin doğruluğu 0,9694 olarak hesaplanmıştır.

Confusion matrix (karmaşıklık matrisi), modelin doğru ve yanlış sınıflandırmalarını görselleştirmek için kullanılır. Bu matris, gerçek sınıflar ile modelin tahmin ettiği sınıflar arasındaki ilişkiyi gösterir.

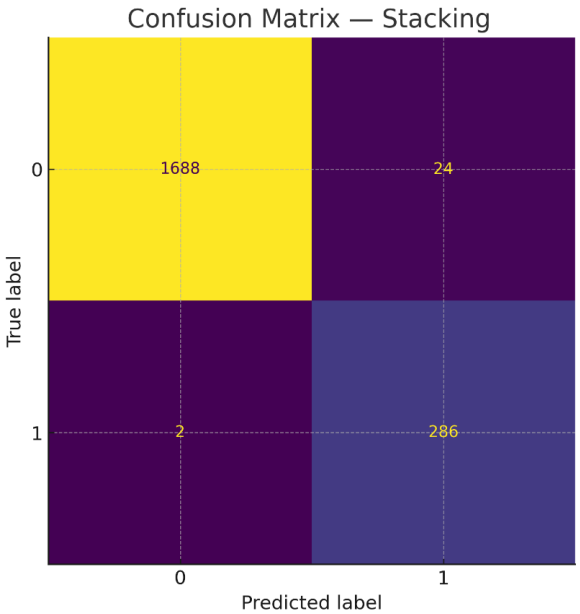
Şekil 2'de gösterilen karmaşıklık matrisinde, Voting Classifier modelinin yüksek doğruluk oranına sahip olduğunu görülmektedir. Model, negatif ve pozitif sınıfları büyük ölçüde doğru sınıflandırırken, düşük yanlış pozitif (FP) değeriyle yanlış alarm üretme olasılığını minimize etmiştir. Ancak, nispeten yüksek yanlış negatif (FN) değeri, bazı pozitif örneklerin hatalı sınıflandırıldığını göstermektedir.

Tablo 4. Model performans sonuçları

Metric	Voting Classifier	Stacking Classifier
Accuracy	0,9495	0,9751
Precision	0,9736	0,9755
Recall	0,9227	0,9741
F1 Score	0,9475	0,9748



Şekil 2. Voting classifier için karmaşıklık matrisi.



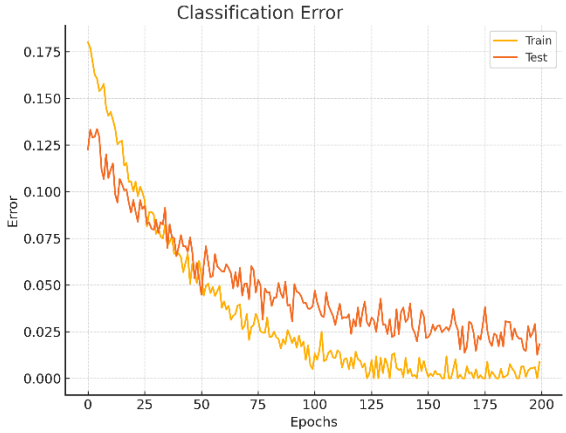
Şekil 3. Stacking classifier için karmaşıklık matrisi.

Şekil 3'te Stacking Classifier için oluşturulan karmaşıklık matrisi verilmiştir. Model, oldukça başarılı bir performans göstermektedir. Özellikle FN değerinin oldukça düşük olması, modelin pozitif örnekleri kaçırma oranının çok az olduğunu göstermektedir. FP değeri de

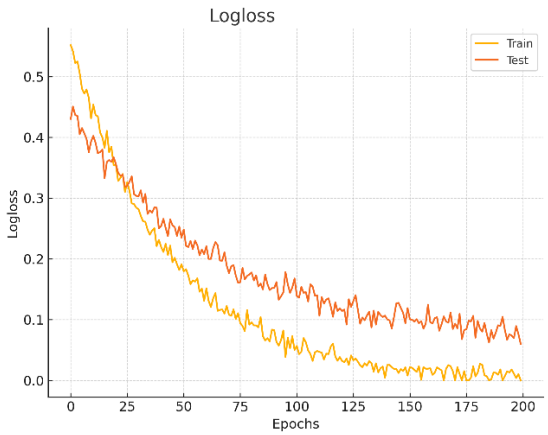
düşük olduğu için model yanlış alarm üretme konusunda da başarılıdır. Genel olarak, Stacking Classifier modeli Voting Classifier' a kıyasla daha yüksek doğruluk ve daha düşük hata oranları ile daha iyi bir performans sergilemektedir.

Şekil 4, Eğitim ve doğrulama hataları dönem boyunca düzenli biçimde azalmakta; iki eğri birbirine yakın seyrettiği için belirgin bir ayrışma (overfitting) gözlenmemektedir. Bu durum, modelin öğrenirken genelleme kabiliyetini koruduğunu gösterir.

Şekil 5, Her iki logloss eğrisi de istikrarlı şekilde düşmektedir ve doğrulama eğrisi yukarı yönlü sapma göstermemektedir. Bu, olasılık tahminlerinin giderek daha iyi kalibre olduğunu ve modelin eğitim sürecinin sağlıklı ilerlediğini ortaya koyar.

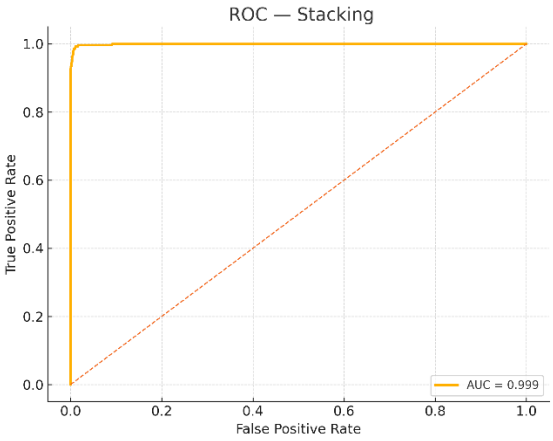


Şekil 4. Eğitim ve doğrulamahata oranı (epoch).

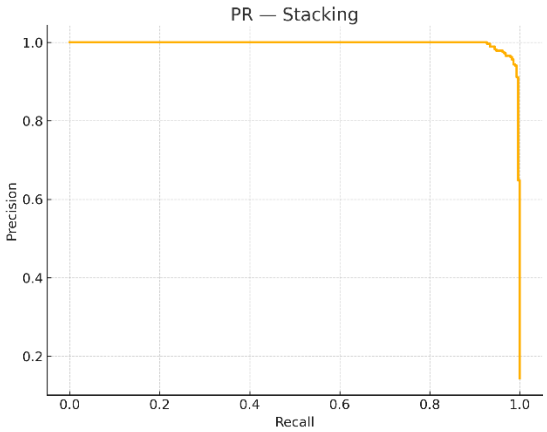


Şekil 5. Eğitim ve doğrulama logloss (epoch).

Şekil 6 ve Şekil 7 Stacking Classifier için ROC ve PR eğrileri. ROC (Receiver Operating Characteristic) eğrisi, modelin yanlış pozitif oranı (False Positive Rate) ile doğru pozitif oranı (True Positive Rate) arasındaki ilişkiyi göstererek sınıflandırma performansını ölçer. Eğrinin altında kalan alan (AUC – Area Under Curve) değeri ne kadar 1'e yakınsa, modelin genel performansı o kadar iyidir. Bu çalışmada Stacking Classifier modeli için elde edilen AUC değeri 0,99 seviyesindedir, bu da modelin hem pozitif hem negatif sınıfları ayırt etme kabiliyetinin yüksek olduğunu göstermektedir.



Şekil 6. ROC eğrisi.



Şekil 7. PR eğrisi.

PR (Precision-Recall) eğrisi ise özellikle dengesiz veri setlerinde daha anlamlı bir değerlendirme sunar. Bu eğri, modelin doğru pozitif oranını (recall) ve yanlış alarm üretmeden ne kadar doğru tahmin yaptığını (precision) görselleştirir. Elde edilen PR eğrisinin yüksek bir eğimde ve geniş alanda seyretmesi, modelin pozitif sınıfı doğru şekilde tahmin ederken aynı zamanda düşük oranda yanlış alarm ürettiğini göstermektedir.

Bu grafikler, Stacking Classifier modelinin özellikle azınlık (vulnerable) sınıfı doğru tahmin etmedeki başarısını kanıtlamakta ve çalışma kapsamında uygulanan ensemble yaklaşımının etkinliğini desteklemektedir.

6. Tartışma ve Sonuç

Bu çalışmada, JavaScript programlama dili içerisinde yer alan fonksiyonların güvenlik açıklarını tespit edebilmek amacıyla JSVULNDETECT adlı makine öğrenmesi tabanlı bir sistem geliştirilmiştir. Sistem, statik analiz teknikleri kullanarak fonksiyonlardan çıkarılan özellikler üzerinden güvenli/güvensiz sınıflandırması yapmakta ve bu süreci otomatikleştirerek hem zamandan tasarruf sağlamakta hem de insan hatasını minimize etmektedir.

Model geliştirme sürecinde karşılaşılan sınıf dengesizliği problemi, SMOTE-Tomek yöntemi ile etkin biçimde çözülmüş, bu sayede azınlık sınıf olan "güvensiz fonksiyonlar" için sınıflandırma başarımı artırılmıştır.

Ayrıca, her biri farklı öğrenme stratejilerine sahip 12 makine öğrenimi modeli hiperparametre optimizasyonu ile değerlendirilmiş ve bu kapsamda Bayesian Search yaklaşımı kullanılmıştır. Voting ve Stacking gibi ensemble öğrenme teknikleri sayesinde model performansı daha da artırılmış ve özellikle Stacking Classifier modeli ile %97,51 doğruluk oranı elde edilmiştir.

Geliştirilen sistem, yalnızca yüksek doğrulukla sınıflandırma yapmakla kalmamış, aynı zamanda önceki literatür çalışmalarının ötesine geçerek çok sayıda modelin karşılaştırmalı analizini sağlamıştır. Örneğin Ferenc vd. (2019) çalışmasında yalnızca birkaç temel model kullanılarak daha düşük doğruluklar elde edilmişken, bu çalışmada kapsamlı modelleme yaklaşımı ve optimizasyon süreci sayesinde daha güçlü sonuçlar alınmıştır. Bu durum, önerilen sistemin hem teknik hem de yöntemsel açıdan literatüre anlamlı bir katkı sunduğunu ortaya koymaktadır.

Ayrıca, JSVULNDETECT sisteminin web tabanlı bir arayüzle erişilebilir olması, onu sadece akademik araştırmalar için değil, aynı zamanda gerçek dünya uygulamaları için de uygun hale getirmektedir. Sistem, yazılım geliştiriciler ve güvenlik uzmanları için kullanılabilir bir araç olarak işlev görmektedir.

Bu çalışma kapsamında sunulan temel katkılar aşağıda özetlenmiştir:

- Güvenlik açığı tespitinde çok modellerli yaklaşım ve hiperparametre optimizasyonunun birlikte uygulanması,
- SMOTE-Tomek ile veri dengesizliği probleminin çözülmesi ve bunun model başarımı üzerindeki etkisinin gösterilmesi,
- Kullanıcı dostu web arayüzü ile geliştiricilere pratik kullanım imkânı sunan bir sistem tasarımı,
- Güvenlik alanında açıklanabilirlik ve genellenebilirlik açısından örnek teşkil edecek açık ve sistematik bir yaklaşımın sunulması.

Gelecek çalışmalarda sistemin kapsamı ve etkinliği daha da artırılabilir. Öncelikli olarak, GitHub gibi açık kaynaklı platformlardan elde edilecek daha güncel ve çeşitli JavaScript projeleri ile veri setinin genişletilmesi, modelin genelleme yeteneğini ve farklı güvenlik senaryolarına adaptasyon kabiliyetini güçlendirecektir. Ayrıca sistemin yalnızca JavaScript ile sınırlı kalmaksızın Python, PHP ve Ruby gibi farklı programlama dillerini de kapsayacak şekilde genişletilmesi, çok dilli güvenlik analizi için önemli bir adım olacaktır. Bununla birlikte, mevcut sistem statik analiz temelli çalışmakta olup, ilerleyen süreçte dinamik analiz teknikleriyle desteklenmesi; özellikle çalışma zamanında ortaya çıkan saldırgan davranışların tespiti açısından fayda sağlayacaktır. Modelin karar verme süreçlerini şeffaflaştırmak amacıyla SHAP ve LIME gibi açıklanabilir yapay zeka (XAI) yöntemlerinin entegrasyonu planlanmakta; böylece kullanıcıya hangi özelliklerin hangi güvenlik risklerine yol açtığı daha anlaşılır biçimde sunulabilecektir. Son olarak, geliştirilecek sistemin RESTful API mimarisi ile yapılandırılarak sürekli entegrasyon/sürekli dağıtım (CI/CD) süreçlerine, yaygın kullanılan geliştirme ortamlarına (örneğin Visual Studio Code) veya tarayıcı uzantılarına entegre edilmesi hedeflenmektedir. Bu sayede geliştiricilerin gerçek zamanlı güvenlik uyarıları alabilmesi ve yazılım yaşam döngüsü boyunca güvenlik farkındalığının artırılması mümkün olacaktır.

Bu çalışma, yazılım güvenliği alanında hem teknik hem de pratik açıdan güçlü bir çözüm sunmakta; özellikle yazılım geliştirme sürecine güvenlik perspektifini dahil etmek isteyen araştırmacı ve profesyoneller için yenilikçi bir yaklaşım ortaya koymaktadır.

Katkı Oranı Beyanı

Yazarların katkı yüzdeleri aşağıda verilmiştir, Yazarlar makaleyi incelemiş ve onaylamıştır.

%	H.H.A.	Ö.T.	Z.C.
K	50	50	
T	50	50	
Y	40	30	30
VTI		50	50
VAY	100		
KT	50	50	
YZ	100		
GR		50	50
PY		50	50

K= kavram, T= tasarım, Y= yönetim, VTI= veri toplama ve/veya işleme, VAY= veri analizi ve/veya yorumlama, KT= kaynak tarama, YZ= Yazım, GR= gönderim ve revizyon, PY= proje yönetimi.

Çatışma Beyanı

Yazarlar bu çalışmada hiçbir çıkar ilişkisi olmadığını beyan etmektedirler.

Etik Onay Beyanı

Bu araştırmada hayvanlar ve insanlar üzerinde herhangi bir çalışma yapılmadığı için etik kurul onayı alınmamıştır.

Kaynaklar

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). https://doi.org/10.1145/2939672.2939785

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018

Dietterich, T. G. (2000). Ensemble methods in machine learning. In J. Kittler & F. Roli (Eds.), *Multiple Classifier Systems* (pp. 1–15). Springer. https://doi.org/10.1007/3-540-45014-9_1

Ferenc, R., Hegedűs, P., Gyimesi, P., Antal, G., Bán, D., & Gyimóthy, T. (2019). Challenging machine learning algorithms in predicting vulnerable JavaScript functions. In *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)* (pp. 8–14). IEEE. https://doi.org/10.1109/RAISE.2019.00010

Friedman, J. H. (2001). Greedy function approximation: A

gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189–1232. https://doi.org/10.1214/aos/1013203451

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, *63*(1), 3–42. https://doi.org/10.1007/s10994-006-6226-1

Harzevili, N. S., Belle, A. B., Wang, J., Wang, S., Jiang, Z. M., & Nagappan, N. (2023). *A survey on automated software vulnerability detection using machine learning and deep learning* (arXiv:2306.11673). arXiv. https://doi.org/10.48550/arXiv.2306.11673

Hosmer, D. W., & Lemeshow, S. (2000). *Applied logistic regression* (2nd ed.). John Wiley & Sons.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., & Liu, T. Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, *30*, 3146–3154. https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

Liu, Z., Fang, Y., Huang, C., & Xu, Y. (2023). MFXSS: An effective XSS vulnerability detection method in JavaScript based on multi-feature model. *Computers & Security*, *124*, 103015. https://doi.org/10.1016/j.cose.2022.103015

McCallum, A., & Nigam, K. (1998). A comparison of event models for naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization* (pp. 41–48). AAAI Press.

Şahin, M. (2025). Binary logistic regression procedure with an application. *Black Sea Journal of Statistics*, *1*(1), 22–26. https://blackseapublishers.online/index.php/statistics/article/view/21

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, *25*, 2951–2959.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, *5*(2), 241–259. https://doi.org/10.1016/S0893-6080(05)80023-1