*Article*

# DLL ve API Davranış Özelliklerine Dayalı Grafik Tabanlı Makine Öğrenmesi ile Windows Zararlı Yazılım Tespiti

**Khalid Tariq Mahmood *, Özgür Tonkal**

Department of Software Engineering, Samsun University, 55000 Samsun, Türkiye;

ozgur.tonkal@samsun.edu.tr ( Ö.T ); U191118048@samsun.edu.tr

**Abstract:** As malware becomes increasingly complex, machine learning-based techniques have emerged to enhance traditional detection systems. The growing diversity and behavioral intricacy of malware present serious challenges in cybersecurity, as conventional methods often fail to capture the structural and semantic relationships that characterize malicious activity. Therefore, developing innovative, explainable, and computationally efficient detection frameworks that can effectively analyze and interpret such complex patterns has become critical.

This study proposes a novel machine learning-based malware detection framework that integrates API call sequences and DLL information into a unified, graph-based structure to extract behavior-driven features. Rather than relying on deep learning architectures, the proposed method derives statistical and structural attributes from API-DLL relationships and utilizes them as input to various machine learning models, including Logistic Regression, Random Forest, XGBoost, and LightGBM.

The extracted features are comprehensively evaluated across models, and experimental results show that the LightGBM classifier achieves the best performance with an accuracy of 87%. Furthermore, feature importance analysis identifies key API and DLL interactions that contribute most significantly to distinguishing malicious behavior, thereby improving model interpretability. The use of a real-world malware dataset provides a strong foundation for future comparative studies, and this work contributes to the field by offering a lightweight and interpretable machine learning approach for malware detection supported by graph-based behavioral analysis.

**KeyWords:** Malware detection, Machine learning, API call sequence, DLL analysis, Graph-based analysis, LightGBM, Static analysis

## 1. Giriş

Zararlı yazılım (malware), internete bağlı bilgisayar veya cihazlara sızarak devlet kurumlarına, ticari kuruluşlara ya da bireysel kullanıcılara ait hassas bilgileri çalan kötü niyetli yazılım parçaları olarak tanımlanır [1]. Zararlı yazılımlar zamanla daha karmaşık ve gizlenebilir hale gelmiş, gelişmiş teknikler kullanarak tespit edilmelerini zorlaştırmıştır [2,3]. Bu değişken ve dinamik yapı, zararlı yazılımların tespitini hem zorlaştırmakta hem de daha etkili ve esnek tespit yaklaşımlarının geliştirilmesini zorunlu kılmaktadır.

Zararlı yazılım tespit yöntemleri genel olarak iki ana kategoriye ayrılır: **statik analiz** ve **dinamik analiz**. Statik analiz, zararlı yazılımın çalıştırılmasına gerek olmadan kaynak dosya üzerinden analiz yapılmasına olanak tanır [4]. Bu sayede, metin desenleri, opcode'lar ve bayt dizileri gibi düşük seviyeli öznitelikler elde edilir. Bu yöntemler hızlı ve sistem kaynakları açısından verimli olsa da, şifreleme (encryption) ve kod gizleme (obfuscation) tekniklerine karşı savunmasızdır. Bu çalışmada, **statik analiz yöntemi tercih edilmiştir**.

Statik analiz kapsamında, özellikle **API çağrı dizileri** ve **DLL bağımlılıkları**, yazılımın davranışsal izlerini sunar ve zararlı niyetli aktivitelerin saptanmasında kritik rol oynar [6]. API çağrıları; dosya işlemleri, ağ bağlantısı ve kayıt defteri değişiklikleri gibi işlemleri temsil eder. Bu çağrıların sıralı dizilimleri, belirli bağlamlarda zararlı davranış kalıplarını yansıtabilir. Örneğin, "CreateWindow → ShowWindow → CreateFile" şeklinde bir çağrı dizisi, sistemde gizli bir pencere oluşturup ardından dosya yazmaya çalışan kötü niyetli bir yazılımı işaret edebilir [3].

Son yıllarda birçok çalışma, zararlı yazılımların davranışlarını analiz etmek amacıyla **Makine Öğrenmesi (ML)** ve **Derin Öğrenme (DL)** tekniklerini kullanmıştır [7–14]. Ancak bu çalışmada, yüksek işlem gücü gerektiren DL mimarileri yerine, daha açıklanabilir ve hafif yapılı ML modelleri tercih edilmiştir. ML algoritmaları, grafik tabanlı davranışsal verilerden türetilen öznitelikler ile eğitilmiş; model doğruluğu, verimliliği ve yorumlanabilirliği açısından detaylı olarak değerlendirilmiştir.

Bu çalışmada, API ve DLL ilişkileri temel alınarak oluşturulan grafik yapıları üzerinden çeşitli **istatistiksel ve yapısal öznitelikler** çıkarılmıştır. Bu öznitelikler; DLL sayısı, şüpheli API içerip içermediği, dosya veya ağ işlemi sayısı, zincir uzunluğu, çağrı çeşitliliği gibi ölçümleri kapsamaktadır. Elde edilen bu öznitelikler, **Logistic Regression, Random Forest, XGBoost ve LightGBM** gibi yaygın makine öğrenmesi algoritmaları ile değerlendirilmiştir. Özellikle LightGBM modeli, en iyi performansı %87 doğruluk oranı ile elde etmiş, aynı zamanda düşük overfitting eğilimi ve yüksek yorumlanabilirliği ile öne çıkmıştır.

Model performansları **5 katlı çapraz doğrulama** ile test edilmiş ve öznitelik önem sıralaması (feature importance) ile hangi davranışların sınıflandırma başarısına en çok katkı sağladığı belirlenmiştir. Bu analiz, yalnızca zararlı yazılımların doğru sınıflandırılmasını değil, aynı zamanda hangi davranışların tehdit oluşturduğunu anlaşılır kılmayı da amaçlamaktadır.

Sonuç olarak, bu çalışma grafik tabanlı davranışsal verilerin, açıklanabilir makine öğrenmesi modelleriyle nasıl etkili şekilde işlenebileceğini göstermekte ve hem akademik literatüre hem de pratik güvenlik uygulamalarına katkı sağlamaktadır.

**Bu araştırmanın temel katkıları şunlardır:**

1. API çağrı dizileri ile DLL bağımlılıklarının oluşturduğu grafik yapılar üzerinden elde edilen davranışsal özniteliklere dayalı açıklanabilir, hafif ve etkili bir makine öğrenmesi temelli zararlı yazılım tespit çerçevesi sunulmuştur. Derin gömleme yöntemlerine başvurulmadan, doğrudan çıkarılan yapısal-istatistiksel özelliklerle yüksek doğruluk oranları elde edilmiştir.

2. Grafik temsillerden türetilen öznitelikler, makine öğrenmesi algoritmaları olan Logistic Regression, Random Forest, XGBoost ve LightGBM ile eğitilerek detaylı bir karşılaştırmalı analiz yapılmıştır. Bu algoritmalar, farklı perspektiflerden veriyi yorumlayarak zararlı yazılım davranışlarını başarıyla ayırt etmiştir.

3. LightGBM modeli %87 doğruluk oranı ile en yüksek performansı göstermiş; ayrıca, özellik önem analizi ile hangi özniteliklerin sınıflandırma kararlarına ne derece etki ettiği analiz edilmiştir. Bu sayede, sistemin sadece doğru tahminler yapması değil, aynı zamanda kararlarının anlaşılabilir olması sağlanmıştır.

4. Gerçek dünyadan elde edilen zararsız ve zararlı yazılım örneklerinden oluşturulan özel bir veri seti ile model test edilmiş ve bu veri seti, gelecekteki çalışmalar için de güvenilir bir referans olarak sunulmuştur.

**Makalenin geri kalanı şu şekilde yapılandırılmıştır :**

**Bölüm 2:** Zararlı yazılım tespitine yönelik güncel literatür incelenmiştir.

**Bölüm 3:** Öznitelik çıkarımı, grafik yapılar, makine öğrenmesi algoritmaları ve sistem mimarisi açıklanmıştır.

**Bölüm 4:** Deneysel sonuçlar, performans karşılaştırmaları ve değerlendirme metrikleri sunulmuştur.

**Bölüm 5:** Çalışmanın sınırlamaları ve elde edilen sonuçların yorumları yer almaktadır.

**Bölüm6:** Genel sonuçlar ve gelecekteki araştırmalara yönelik çıkarımlar paylaşılmıştır.

## 2. İlgili Çalışmalar

Son yıllarda zararlı yazılım tespiti alanında yapılan çalışmalar, giderek artan karmaşıklığa sahip zararlı yazılımları tespit edebilmek için özellik mühendisliği ve makine öğrenmesi yöntemlerinin etkinliğini araştırmaktadır. Özellikle API çağrı dizileri ve DLL bağımlılıkları gibi davranışsal veriler üzerinden öznitelikler çıkarılarak, çeşitli makine öğrenmesi algoritmaları ile zararlı yazılım sınıflandırması gerçekleştirilmiştir.

Örneğin, Darem ve arkadaşları [19], öznitelik mühendisliği yöntemleri ile öznitelik uzayını optimize etmiş ve bu sayede Microsoft zararlı yazılım veri setinde %96'nın üzerinde doğruluk elde etmiştir. Bu çalışma, özellikle karmaşık derin öğrenme mimarilerine ihtiyaç duymadan, seçilmiş özniteliklerle etkili tespit yapılabileceğini göstermiştir.

Kumar ve arkadaşları [18], farklı makine öğrenmesi modellerini bir araya getirerek topluluk (ensemble) öğrenme yaklaşımı kullanmış ve görsel temsiller ile çalışarak yüksek doğruluk oranları bildirmiştir. Benzer şekilde, Bensaoud ve Kalita [1], opcode dizileri ve API çağrıları temelinde çeşitli klasik sınıflayıcılarla başarılı sonuçlar elde etmişlerdir.

Bazı çalışmalarda, grafik temelli yapılar kullanılmış olsa da bu grafiklerden çıkarılan öznitelikler yine geleneksel makine öğrenmesi algoritmaları ile işlenmiştir. Özellikle GraphSAGE veya Node2Vec gibi gömleme yöntemleri yerine, çağrı zinciri uzunluğu, şüpheli API kullanımı, işlem sayıları gibi doğrudan sayısal öznitelikler üzerinden analiz gerçekleştirilmiştir.

Brown ve arkadaşları [20], model seçim sürecini optimize etmek için Otomatik Makine Öğrenimi (AutoML) tekniklerini kullanarak statik analiz temelli sınıflandırıcıların performansını artırmışlardır. Bu yaklaşım, hem açıklanabilirliği hem de verimliliğiyle dikkat çekmiştir.

Bazı güncel çalışmalar halen derin öğrenme veya GNN (Grafik Sinir Ağı) temelli modellere odaklansa da, bu modellerin yüksek işlem maliyetleri ve sınırlı açıklanabilirliği, özellikle güvenlik sistemlerine entegre edilmesi açısından önemli zorluklar doğurmaktadır. Bu nedenle, açıklanabilir, kaynak dostu ve hızlı tahmin sunabilen makine öğrenmesi modelleri, özellikle statik analiz üzerine kurulu sistemler için güçlü bir alternatif olarak değerlendirilmektedir.

Bu çalışmada sunulan yaklaşım, literatürdeki bu yönelimlerle uyumlu şekilde, grafik temsillerden öznitelik çıkartarak bu öznitelikleri makine öğrenmesi algoritmaları ile işleyen açıklanabilir bir sistem olarak konumlandırılmıştır. LightGBM, Logistic Regression, Random Forest ve XGBoost gibi yöntemlerle yapılan karşılaştırmalar sonucunda, özellikle LightGBM modeli yüksek doğruluk ve kararlı sonuçlar göstermiştir.

## 3. Kuramsal Arka Plan

Bu bölümde, önerilen makine öğrenmesi tabanlı zararlı yazılım tespit çerçevesinin kuramsal ve yapısal temelleri sunulmaktadır. Deneysel değerlendirmelere geçmeden önce, API çağrı dizileri ve DLL bağımlılıkları üzerinden elde edilen grafik temsillerin nasıl oluşturulduğu, bu grafiklerden çıkarılan özniteliklerin nasıl tasarlandığı ve makine öğrenmesi algoritmaları ile nasıl sınıflandırıldığı ayrıntılı olarak açıklanacaktır.

Sistem, derin öğrenme tabanlı gömme teknikleri yerine, grafiklerden istatistiksel ve davranışsal öznitelikler çıkartılmasına odaklanmaktadır. Bu öznitelikler arasında şunlar yer almaktadır:

- DLL sayısı
- Şüpheli API içerip içermediği
- Dosya ve ağ işlemi sayısı

- Bağlantı yoğunluğu
- Zincir uzunluğu
- Benzersiz çağrı çeşitliliği

Bu özellikler, geleneksel makine öğrenmesi algoritmalarının kullanımı için doğrudan uygun bir vektör uzayı oluşturur.

Çalışmada kullanılan sınıflandırma algoritmaları şunlardır:

- Logistic Regression
- Random Forest
- XGBoost
- LightGBM

Ayrıca, bu modellerin birlikte değerlendirilmesi amacıyla bir topluluk (ensemble) öğrenme stratejisi olarak soft voting yöntemi kullanılmıştır.

Derin sinir ağları veya GNN tabanlı gömme yöntemlerine ihtiyaç duyulmadan, sistemin düşük kaynakla yüksek doğruluk sağlayabilmesi hedeflenmiştir. Bu çerçeve, özellikle açıklanabilirliği yüksek karar destek sistemleri için uygundur.

Son olarak, çalışmada kullanılan veri kümesi açıklanmıştır. Veri kümesi, çeşitli Windows tabanlı benign ve malicious çalıştırılabilir dosyalardan elde edilen API-DLL ilişkilerini içermektedir. Veri seti, öznitelik çıkarımı öncesinde temizlenmiş ve etiketlenmiş olup, eğitim ve test aşamaları için %80–%20 oranında bölünmüştür. Ayrıca model başarımını daha adil değerlendirebilmek için 5 katlı çapraz doğrulama yöntemi uygulanmıştır.

## 4. Grafik Tabanlı Davranışsal Öznitelik Çıkarımı

Bu çalışmada, Windows çalıştırılabilir dosyalarının davranışlarını analiz etmek amacıyla API çağrı dizileri ve DLL ilişkileri üzerinden grafik benzeri yapılar kullanılmış, ancak doğrudan grafik algoritmaları ya da düğüm tabanlı gömme yöntemleri (örneğin Node2Vec, GNN) kullanılmamıştır. Bunun yerine, bu ilişkilerden elde edilen sayısal ve anlamlı öznitelikler, makine öğrenmesi modellerinde sınıflandırma için kullanılacak veri matrisine dönüştürülmüştür.

Yazılım içindeki API çağrıları ve DLL referansları, analiz edilen dosyanın davranışsal profilini yansıtan bir yapı oluşturur. Statik analiz sürecinde elde edilen API çağrı sıralamaları ve DLL kullanımları üzerinden şu tür özellikler çıkarılmıştır:

- Toplam API sayısı

- Şüpheli API içeriyor mu? (örn. WriteProcessMemory, CreateRemoteThread)
- Farklı DLL sayısı
- Dosya sistemi erişimi yapan çağrı sayısı
- Ağ bağlantısı kuran çağrı sayısı
- Registry üzerinde işlem yapan çağrı sayısı
- Aynı API'nin tekrar sayısı (frekans)
- Çağrı dizisi uzunluğu ve çeşitliliği

Bu öznitelikler, hem yazılımın ne tür işlemler yaptığına dair genel bir davranış profili sunmakta hem de zararlı yazılımlar ile benign yazılımlar arasında ayırt edici farkların ortaya çıkmasını sağlamaktadır.

Grafik yapısındaki ilişkilerden yola çıkılarak şu örnek senaryolarda belirli desenler tespit edilmiştir:

- Dosya oluşturma + yazma + kapama işlemlerinin art arda gelmesi
- DLL'lerin çeşitlilik göstermesi ve aynı anda birden fazla DLL'in işlevsel olarak kullanılması
- Zararlı yazılımlar tarafından sıkça kullanılan API kombinasyonlarının bulunması

Bu işlem sonucunda, her analiz edilen yazılım örneği için sabit uzunlukta bir öznitelik vektörü elde edilmiştir. Elde edilen bu yapı, doğrudan **makine öğrenmesi modelleri** için uygun bir giriş veri setine dönüştürülmüş ve model eğitimi bu yapı üzerinde gerçekleştirilmiştir.

Bu yaklaşımla, geleneksel grafik kavramlarına (düğüm/kenar) veya derin gömme yöntemlerine ihtiyaç duyulmadan, açıklanabilir ve hafif bir sistem geliştirilmiştir. Grafik benzeri yapılar sadece davranış ilişkilerini modellemek için arka planda referans olarak kullanılmış, asıl odak öznitelik çıkarımı ve sınıflandırma başarısı olmuştur.

### 4.1 K-means Kümeleme (Unsupervised Learning)

**Temel Mantık:**
K-means, denetimsiz bir öğrenme algoritması olup, veri kümesini kkk sayıda gruba (küme) ayırır. Amaç, her bir veriyi en yakın merkez noktasına atayarak gruplamaktır.

**Projede Uygulanışı:**
Zararlı ve zararsız etiketleri kullanılmadan, API ve DLL ilişkilerine dayalı çıkarılan davranışsal özniteliklerle doğal küme yapısı analiz edilmiştir. Öznitelikler:

- API_ÇEŞİTLİLİĞİ,
- DLL_SAYISI,
- SUPHELI_VAR_MI gibi sayısal ve kategorik değerlerdir.

**Kullanım Amaçları:**

- Veri segmentasyonu
- Şüpheli yazılımların kümelenip kümelenmediğini test etmek
- Ön analiz aracı olarak görsel keşif yapmak

**Sonuç:**
K=2 alındığında zararlı yazılımlar ayrı kümelerde toplanma eğilimi göstermiştir. Model doğruluğu etiketsiz olduğu için dışsal değerlendirme (silhouette score) ile yapılmıştır.

## 4.2 K-Nearest Neighbors (KNN)

**Temel Mantık:**
Her bir örnek, öznitelik uzayındaki en yakın kkk komşusuna bakarak sınıflandırılır. Mesafe ölçütü genellikle Öklidyen mesafedir.

**Projede Uygulanışı:**
Normalize edilmiş öznitelikler (örneğin DLL_SAYISI, API_SAYISI) üzerinden k=5 olarak uygulanmıştır. Eğitim verisi ile test verisi arasında doğrudan karşılaştırma yapılır.

**Modelin Hesaplama Adımı:**

1. Test örneği alınır
2. Tüm eğitim örnekleriyle mesafe hesaplanır
3. En yakın kkk komşu seçilir
4. Komşuların çoğunluğuna göre sınıf atanır

**Avantaj:**

- Model eğitimi gerektirmez
- Küçük veri kümelerinde güçlüdür

**Zorluk:**

- Tüm veriyle karşılaştırma gerektiği için yavaştır
- Büyük veri setlerinde uygun değildir

**Proje Gözlemi:**
Doğruluk ~%80 civarında kalmıştır. Özellikle veri dağılımı homojen olduğunda daha başarılıdır.

## 4.3 LightGBM

**Temel Mantık:**
Gradyan artırmalı karar ağaçlarına dayalıdır. LightGBM, histogram bazlı ve leaf-wise büyüme stratejisi kullanarak yüksek hız ve doğruluk sağlar.

**Projede Uygulanışı:**

- Grafik yapıdan elde edilen davranışsal öznitelikler giriş olarak verildi.
- SUPHELI_FONK_LISTE, API_GENISLIGI, DLL_ORAN, ETKILESIM_OZELLIKLERI gibi sentetik özellikler üretildi.
- Eğitim: %80 eğitim, %20 test
- eval_metric = 'binary_logloss', boosting_type='gbdt'

**Modelin Hesaplama Adımı:**

1. İlk ağaç tahmini yapar
2. Hatalı tahmin edilen örnekler için ağırlık güncellenir
3. Yeni ağaç sadece hataları düzeltmeye çalışır
4. Bu işlem belirli n_estimators kadar sürer

**Avantajları:**

- Büyük veri için uygun
- Feature importance sağlar
- Overfitting riski düşüktür

**Proje Sonucu:**
En iyi sonuç veren modeldir, **%87 doğruluk**, F1-score: 0.86
Ayrıca şüpheli fonksiyon ve DLL çeşitliliği en önemli öznitelikler olarak çıktı.

## 4.4 XGBoost

**Temel Mantık:**
XGBoost, gradyan artırma algoritmasını optimize eder. Her yeni ağaç, önceki ağaçların hatalarını azaltır. Regularizasyon kullanır.

**Projede Uygulanışı:**

- Aynı öznitelik seti kullanıldı
- max_depth=5, learning_rate=0.1, n_estimators=100
- Eğitim: sklearn’s train_test_split ile %80-%20
- ROC-AUC metriği izlendi

**Avantajlar:**

- Regularizasyon ile overfit riski düşer
- Özelleştirilebilir hiperparametreler
- Hızlıdır

**Dezavantaj:**

- LightGBM'den daha yavaş
- Tuning gerektirir

**Proje Sonucu:**
%85 doğruluk, ancak bazı sınıf dengesizliği durumlarında hassasiyet düştü.

## 4.5 Random Forest

**Temel Mantık:**
Çok sayıda karar ağacının oylamasına dayalıdır. Ağaçlar farklı örnekler ve özniteliklerle eğitilir. Topluluk kararı daha sağlamdır.

**Projede Uygulanışı:**

- Varsayılan 100 ağaç kullanıldı
- Özniteliklerin önem sırası çıkarıldı
- criterion = 'gini'

**Hesaplama Mantığı:**

1. Her ağaç farklı örnekler ve özniteliklerle eğitilir
2. Test verisi her ağaca gönderilir
3. En çok oyu alan sınıf tahmin edilir

**Avantajlar:**

- Aşırı öğrenmeye dirençlidir
- Özellik önem sıralaması sunar

**Proje Gözlemi:**
Doğruluk %84
En kararlı modellerden biri, outlier'lara karşı da dayanıklı.

## 4.6 Decision Tree

Temel Mantık:
Veri, her düğümde bir özniteliğe göre bölünerek sınıflandırılır. Ağacın yaprağında karar verilir.

**Projede Uygulanışı:**

- max_depth=10 olarak sınırlı
- Tek başına kullanıldığında overfitting eğilimi gözlemlendi
- Grafiksel olarak karar mantığı açık şekilde görüldü

**Örnek Karar Kuralı**

if DLL_SAYISI > 5 and SUPHELI_FONK > 2 → MALWARE

**Avantaj:**

- Açıklanabilir
- Hızlı

**Dezavantaj:**

- Aşırı öğrenme eğilimi yüksek
- Veri varyasyonundan çok etkilenir

**Proje Gözlemi:**
Doğruluk ~%78
Görsel açıklama ve prototipleme için ideal.

## 4.7 Logistic Regression

**Temel Mantık:**
**Veriyi doğrusal bir karar sınırıyla ayırır. Tahminler sigmoid fonksiyonu ile [0,1] aralığına dönüştürülür.**

**Projede Uygulanışı:**

- **solver='lbfgs', penalty='l2'**
- **Özellik katsayıları çıkarıldı**
- **Özellikle SUPHELI_VAR_MI gibi ikili değişkenlerde etkilidir**

**Formül:**

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1+e^{-z}}, \quad z = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

**Avantaj:**

- Yorumlanabilir
- Hızlı

**Dezavantaj:**

- Doğrusal olmayan ilişkileri yakalayamaz

Proje Gözlemi:
Doğruluk %82
Modelin hangi özniteliklere ne yönde tepki verdiği analiz edilmiştir.

## 4.5 Veri Kümesi

Bu çalışmada zararlı yazılım tespiti amacıyla kullanılan veri kümesi, Windows işletim sistemine ait yürütülebilir dosyaların statik analizinden elde edilmiştir. Veri kümesinde hem zararlı (malicious) hem de zararsız (benign) örnekler yer almaktadır. Her bir örnek, dosya içerisindeki API çağrı dizileri, kullanılan DLL'ler ve bu öğelerin çeşitliliği ve ilişkileri temel alınarak öznitelik çıkarımı sürecine tabi tutulmuştur.

Veri kümesinin temel bileşenleri şunlardır:

- Etiket (Label): Her örnek, Benign veya Malicious olarak sınıflandırılmıştır. Bu etiketler, daha önce antivirüs sistemlerince taranmış yazılım örneklerinden alınmıştır.
- Öznitelikler:
  Aşağıda listelenen davranışsal ve yapısal öznitelikler, her bir yazılım örneği için çıkarılmıştır:
    - API_SAYISI: Yazılımda kullanılan toplam API fonksiyonu sayısı
    - DLL_SAYISI: Dosyada çağrılan toplam DLL dosyası sayısı
    - SUPHELI_VAR_MI: Şüpheli kabul edilen API'lerden en az biri kullanılmış mı (evet/hayır)
    - SUPHELI_FONK_LISTE: Şüpheli API fonksiyonlarının toplam sayısı
    - API_DLL_ORANI: API sayısının DLL sayısına oranı
    - ETKILESIM_OZELLIKLERI: PolynomialFeatures ile oluşturulan etkileşimli değişkenler (örn: DLL_SAYISI * SUPHELI_VAR_MI)
    - KARMAŞIKLIK_METRIKLERI: Farklı API çeşitliliği ve frekansına dayalı sayısal göstergeler

Veri Kümesi Dağılımı:

- Toplam örnek sayısı: 601
    - Benign örnek: 307
    - Malicious örnek: 294

Veri Kümesi Kaynağı:

- Benign dosyalar: Güvenilir yazılım arşivlerinden (örneğin WinSxS, Microsoft Store) alınmıştır.
- Malicious dosyalar: Zararlı yazılım veri tabanları ve örnek koleksiyonları (VirusShare, Malicia Dataset vb.) kullanılarak oluşturulmuştur.

Ön İşleme Adımları:

- API ve DLL adları, Unicode karakterlerden arındırılarak normalize edilmiştir.
- Eksik ya da boş alanlar doldurulmuş veya elenmiştir.
- Etiket dengesizliği azaltılmış, sınıflar dengelenmiştir.
- Tüm sayısal öznitelikler MinMaxScaler ile normalize edilmiştir.

Veri kümesi, modellerin hem doğruluğunu hem de genel geçerliliğini test etmek amacıyla %80 eğitim ve %20 test şeklinde bölünmüş ve 5 katlı çapraz doğrulama (5-fold cross-validation) uygulanmıştır.

## 4.6 Performans Ölçütleri

Zararlı yazılım sınıflandırma modelinin başarısını değerlendirmek için çeşitli istatistiksel performans ölçütleri kullanılmıştır. Bu ölçütler, modelin yalnızca doğru sınıflandırma yapma yeteneğini değil, aynı zamanda zararlı yazılımları yanlışlıkla atlamama veya zararsız yazılımları hatalı olarak zararlı olarak işaretlememe duyarlılığını da ölçmektedir.

Bu çalışmada aşağıdaki metrikler dikkate alınmıştır:

### 4.6.1 Doğruluk (Accuracy)

Modelin toplam doğru tahmin sayısının toplam tahmin sayısına oranıdır. Genel başarıyı yansıtır.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

### 4.6.2 Hassasiyet (Precision)

Zararlı olarak tahmin edilen örneklerden kaç tanesinin gerçekten zararlı olduğunu gösterir. Yanlış pozitifleri azaltmak için önemlidir.

$$\text{Precision} = \frac{TP}{TP + FP}$$

### 4.6.3 Duyarlılık / Gerçek Pozitif Oranı (Recall / Sensitivity)

Gerçek zararlı yazılımlardan kaç tanesinin doğru şekilde tespit edildiğini ölçer. Kritik güvenlik uygulamalarında öne çıkar.

$$\text{Recall} = \frac{TP}{TP + FN}$$

#### 4.6.4 F1-Skoru

Precision ve Recall değerlerinin harmonik ortalamasıdır. Sınıflar arasında dengesizlik olduğunda daha dengeli bir ölçüm sağlar.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

#### 4.6.5 ROC-AUC (Eğri Altı Alan)

Modelin pozitif ve negatif sınıflar arasındaki ayrımı ne kadar iyi yaptığını gösteren ölçüttür. 0.5 değeri rastgele tahmin, 1.0 ise kusursuz ayrımı temsil eder.

Bu ölçütler, hem genel sınıflandırma başarımını hem de özellikle zararlı yazılımları tespit etme yeteneğini değerlendirmek açısından kritik öneme sahiptir. Tüm modeller bu metriklere göre karşılaştırılmış, en iyi sonuçları veren modeller LightGBM, XGBoost ve Random Forest olarak belirlenmiştir.

## 4.7 Sonuçlar ve Değerlendirme

Bu çalışmada, Windows platformuna ait zararlı yazılımların statik analizine dayalı olarak davranışsal öznitelikler çıkarılmış ve farklı makine öğrenmesi algoritmaları kullanılarak zararlı yazılım tespiti gerçekleştirilmiştir. API çağrı dizileri, DLL kullanımı, şüpheli fonksiyon içerikleri ve etkileşimli özellikler temelinde oluşturulan öznitelik kümesi, çeşitli sınıflandırma modelleri ile değerlendirilmiştir.

**Model Karşılaştırması**

Gerçekleştirilen deneyler sonucunda aşağıdaki bulgular elde edilmiştir:

- **LightGBM**, tüm ölçütler baz alındığında en yüksek başarıyı gösteren model olmuştur. %87 doğruluk ve %86 F1-Skoru ile en iyi performansı sergilemiştir. Özellikle yüksek boyutlu özniteliklerle çalışabilme ve özellik önem derecelendirmesi sunabilme avantajı, modelin güvenilirliğini artırmıştır.
- **XGBoost** da benzer biçimde yüksek performans göstermiştir. Ancak hiperparametre ayarlarına karşı daha hassas olması nedeniyle eğitim süresi daha uzun sürmüş, belirli sınıf dengesizliklerinde hataya yatkın olmuştur.
- **Random Forest**, %84 doğruluk ile dengeli ve kararlı sonuçlar sunmuştur. Modelin karar ağacı yapısı, sistemin kararlılığını sağlamış ve özellikle overfitting'e karşı dirençli davranmıştır.
- **KNN**, küçük ve homojen veri kümelerinde etkili olmakla birlikte, büyük boyutlu veri setlerinde zaman ve bellek maliyetleri nedeniyle sınırlı kalmıştır.

- **Logistic Regression**, yüksek doğruluk oranı sunmasa da (~%82), özniteliklerin sınıflandırma üzerindeki etkisini yorumlamak açısından oldukça faydalı olmuştur. Güvenlik sistemlerinde karar destek mekanizması olarak kullanılabilir.
- **Decision Tree**, veri üzerinde aşırı öğrenme eğilimi göstermiştir. Ancak modelin açıklanabilir olması, özellikle kurumsal güvenlik politikalarının geliştirilmesinde katkı sağlamaktadır.
- **K-means**, etiketlenmemiş veriler üzerinde zararlı yazılımların kümelenme eğilimini görmek açısından kullanılmış ve doğal ayrışmaların davranışsal olarak analiz edilebileceğini göstermiştir.

### 4.8 Genel Değerlendirme

Model performansları incelendiğinde, gradyan artırmalı ağaç temelli yöntemlerin (LightGBM, XGBoost) zararlı yazılım tespiti konusunda öne çıktığı gözlemlenmiştir. Grafik temelli özniteliklerin doğru modellenmesi ve yüksek ayrıştırıcı güce sahip olması, bu başarının temel nedenlerinden biridir. Ayrıca, çıkarılan etkileşimli özellikler (örneğin DLL_SAYISI * SUPHELI_VAR_MI) sınıflandırma başarımını artırmıştır.

Bu çalışmada ayrıca, her bir modelin açıklanabilirlik, hız, doğruluk ve ölçeklenebilirlik açısından avantajları ve sınırlılıkları değerlendirilmiş; zararlı yazılım tespiti için optimum çözümün bağlamsal ihtiyaçlara göre şekillenmesi gerektiği sonucuna varılmıştır.

### 4.9 Yetenek Değerlendirmesi

Gerçekleştirilen bu çalışma, farklı makine öğrenmesi algoritmalarının zararlı yazılım tespiti konusundaki **sınıflandırma gücü**, **öznitelik duyarlılığı** ve **uyarlanabilirliği** açısından kapsamlı bir değerlendirmesini sunmaktadır. Aşağıda, sistemin yetenekleri çeşitli açılardan analiz edilmiştir:

#### 4.9.1 Sınıflandırma Başarımı

- LightGBM ve XGBoost gibi gradyan artırmalı yöntemler, yüksek doğruluk oranları ve düşük hata payları ile zararlı yazılım sınıflandırmasında güçlü modeller olduğunu göstermiştir.
- Logistic Regression gibi açıklanabilir modeller, özellikle davranışsal özelliklerin risk analizine katkı sunmaktadır.
- K-means gibi denetimsiz yöntemler, ön analiz aşamalarında davranış benzerliklerini ortaya koymak için uygun bir temel oluşturmuştur.

#### 4.9.2 Açıklanabilirlik ve Model Yorumlanabilirliği

- Logistic Regression ve Decision Tree gibi modellerin öznitelik katsayıları ve karar yolları açıkça analiz edilebilir olup, güvenlik analistlerinin neden-sonuç ilişkilerini değerlendirmesine imkan sağlamaktadır.
- LightGBM ile elde edilen "feature importance" verileri, şüpheli API ve DLL davranışlarının tespitinde yol gösterici olmuştur.

#### 4.9.3 Uygulanabilirlik ve Performans

- Uygulanan modeller düşük donanım maliyetleriyle eğitilebilmiş ve test süresi oldukça kısa tutulmuştur (örneklem boyutu 601).
- Genişletilebilir özellik yapısı sayesinde farklı işletim sistemlerine veya yazılım türlerine adapte edilme potansiyeline sahiptir.

#### 4.9.4 Esneklik ve Uyarlanabilirlik

- Çalışma, sadece statik analiz verileri kullanılarak yapılmış olsa da, öznitelik çıkarım süreçleri dinamik analizle de entegre edilebilir durumdadır.
- Modüler yapı sayesinde yeni öznitelikler eklenerek yeniden eğitim yapılabilir ve model güncellenebilir.

#### 4.9.5 Zorluklar ve Sınırlılıklar

- Düşük sayıda veri ile eğitilen bazı modeller (örneğin KNN) sınırlı genelleme başarısı göstermiştir.
- Özellikle sınıf dengesizliği durumlarında (örneğin Benign > Malware) bazı modellerde doğruluk yanıltıcı olabilmektedir.
- Gömleme teknikleri yerine doğrudan istatistiksel öznitelik kullanımı, bazı karmaşık ilişkileri yakalama açısından eksiklik oluşturabilir.

## 5. Sonuçlar

Bu çalışmada, Windows zararlı yazılımlarını tespit etmek amacıyla, statik analiz verilerinden elde edilen davranışsal öznitelikler kullanılarak çeşitli makine öğrenmesi algoritmaları uygulanmıştır. API çağrı dizileri, DLL kullanımı, şüpheli fonksiyon içerikleri ve etkileşimli özelliklerden oluşan öznitelik kümesi, güçlü sınıflandırma modelleri ile analiz edilmiştir.

Deneysel sonuçlara göre:

- **LightGBM**, %87 doğruluk oranı ile en iyi performansı sergilemiştir.
- **XGBoost** ve **Random Forest**, yüksek doğruluk ve F1-skoru ile güçlü alternatifler olarak öne çıkmıştır.
- **Logistic Regression** ve **Decision Tree**, açıklanabilirlik açısından avantaj sağlamış ve hangi özniteliklerin sınıflandırma kararına etkili olduğunu açık biçimde sunmuştur.

- **K-means** gibi denetimsiz algoritmalar ise, davranışsal kümelenmeleri ortaya koyarak veri yapısının analizinde katkı sunmuştur.

Model karşılaştırmaları, öznitelik mühendisliği ve sınıflandırma sürecinde doğru kombinasyonların tespit başarısını doğrudan etkilediğini göstermektedir. Ayrıca, açıklanabilir ve hafif modellerin (örneğin Logistic Regression) gerçek zamanlı güvenlik sistemleri için önemli avantajlar sunduğu görülmüştür.

Genel olarak bu çalışma, grafik yapılarından elde edilen özniteliklerle çalışan klasik makine öğrenmesi modellerinin, yüksek doğrulukla zararlı yazılım tespiti yapılabileceğini ortaya koymaktadır. Derin öğrenmeye kıyasla daha düşük kaynak tüketimi ve daha yüksek açıklanabilirlik avantajları sayesinde, makine öğrenmesi yöntemlerinin zararlı yazılım analizinde güçlü bir alternatif sunduğu sonucuna varılmıştır.

References

[1] Bensaoud, S., & Kalita, J. (2020). *Deep Learning for Malware Analysis: A Survey*. Computers & Security, 92, 101762.

[2] Feng, Q., et al. (2022). *DawnGNN: A Graph Neural Network Based Approach for Windows Malware Detection*. Computers & Security, 113, 102584.

[3] Nataraj, L., et al. (2011). *Malware Images: Visualization and Automatic Classification*. Proceedings of the 8th International Symposium on Visualization for Cyber Security.

[4] Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2008). *A Survey on Automated Dynamic Malware Analysis Techniques and Tools*. ACM Computing Surveys (CSUR), 44(2), 1–42.

[5] Bayer, U., et al. (2006). *Dynamic Analysis of Malicious Code*. Journal in Computer Virology, 2(1), 67–77.

[6] Gandotra, E., Bansal, D., & Sofat, S. (2014). *Malware Analysis and Classification: A Survey*. Journal of Information Security, 5(2), 56–64.

[7] Ye, Y., et al. (2017). *Survey of Malware Detection Techniques*. IEEE Communications Surveys & Tutorials, 16(1), 167–177.

[8] Chen, Y., et al. (2020). *A Multimodal Android Malware Detection Framework Based on CSCG and GAT*. IEEE Access, 8, 141151–141163.

[9] Zhang, X., et al. (2021). *Semantics-Preserving Reinforcement Learning Attack Against Graph Neural Network-Based Malware Detection*. In: Proceedings of the ACM CCS 2021, 2435–2449.

[10] Lin, Z., et al. (2020). *A Comparative Study of GCN and GAT for Ransomware Detection Based on API Call Graphs*. In: Proceedings of the 2020 IEEE TrustCom.

[11] Wu, L., et al. (2023). *MINES: Malware Identification via Neural Embeddings and Similarity using Graph Contrastive Learning*. Computers & Security, 126, 103047.

[12] Brown, N., et al. (2021). *AutoML for Malware Classification*. Computers & Security, 102, 102137.

[13] Zhou, Y., et al. (2019). *API-MalDetect: A Semantic Feature-Aware Deep Learning Framework for Malware Detection*. Expert Systems with Applications, 138, 112790.

[14] Darem, A. B., et al. (2020). *Feature Engineering for Effective Malware Detection*. In: Journal of Computer Virology and Hacking Techniques, 16(2), 127–137.

[15] Roy, S., et al. (2015). *Chronological Behavior-Based Malware Detection Using Machine Learning*. In: IEEE ICC.

[16] Xu, K., et al. (2021). *Graph-Based Malware Detection with Node2Vec and GNNs*. In: ACM Transactions on Privacy and Security, 24(2), 1–32.

[17] Kumar, S., et al. (2021). *Transfer Learning for Malware Classification Using CNNs*. In: IEEE Access, 9, 35498–35507.

[18] Microsoft Malware Classification Challenge (BIG 2015) Dataset. [Online]. Available: