DÜZCE ÜNİVERSİTESİ | LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MAKİNE ÖĞRENMESİ İLE ŞİFRELEME ALGORİTMALARININ DİNAMİK SEÇİMİ

FERDİ AZBOY

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN
DOÇ. DR. ESRA ŞATIR

DÜZCE, 2025

T.C.
DÜZCE ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MAKİNE ÖĞRENMESİ İLE ŞİFRELEME ALGORİTMALARININ DİNAMİK SEÇİMİ

Ferdi AZBOY tarafından hazırlanan tez çalışması aşağıdaki jüri tarafından Düzce Üniversitesi Lisansüstü Eğitim Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Doç. Dr. Esra ŞATIR

Düzce Üniversitesi

Jüri Üyeleri

Doç. Dr. Esra ŞATIR
Düzce Üniversitesi _____

Prof. Dr. Ayhan ERDEM
Gazi Üniversitesi _____

Dr. Öğr. Üyesi Enver KÜÇÜKKÜLAHLI
Düzce Üniversitesi
_____

Tez Savunma Tarihi: 17/10/2025

BEYAN

Bu tez çalışmasının kendi çalışmam olduğunu, tezin planlanmasından yazımına kadar bütün aşamalarda etik dışı davranışımın olmadığını, bu tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara kaynak gösterdiğimi ve bu kaynakları da kaynaklar listesine aldığımı, yine bu tezin çalışılması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını beyan ederim.

17/10/2025

Ferdi AZBOY

ÜRETKEN YAPAY ZEKA KULLANIM BEYANI

17/10/2025

Bu tez çalışmasını hazırlarken ChatGPT üretken yapay zekâ programından destek aldığımı beyan ederim. Tezimin hazırlığı aşamasında üretken yapay zekâ programlarından dil çevirisi, literatür taraması ve kodlama desteği aldım. Üretken yapay zekâ programlarından aldığım bilgilerin doğruluğunu kontrol ettiğimi bildiririm.

Herhangi bir zamanda, çalışmamla ilgili yaptığım bu beyana aykırı bir durumun saptanması durumunda, ortaya çıkacak tüm ahlaki ve hukuki sonuçları kabul ettiğimi bildiririm.

Ferdi AZBOY

TEŞEKKÜR

Yüksek lisans öğrenimimde ve bu tezin hazırlanmasında gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Doç. Dr. Esra ŞATIR' a en içten dileklerimle teşekkür ederim.

Bu çalışma boyunca yardımlarını ve desteklerini esirgemeyen sevgili aileme ve arkadaşlarıma sonsuz teşekkürlerimi sunarım.

17 Ekim 2025 **Ferdi AZBOY**

İÇİNDEKİLER

Sayfa No

ŞEKİL LİSTESİ vi
ÇİZELGE LİSTESİ vii
KISALTMALAR vii
ÖZET viii
ABSTRACT ix
1. GİRİŞ 1
2. KURAMSAL TEMELLER 4
2.1. KRİPTOGRAFİ 4
2.1.1. Kriptografinin Tarihsel Gelişimi 4
2.1.2. Şifreleme Algoritmaları 5
2.1.2.1. Blok Şifreleme 5
2.1.2.2. Akış Şifreleme 5
2.1.2.3. Simetrik Şifreleme 6
2.1.2.4. Asimetrik Şifreleme 6
2.2. MAKİNE ÖĞRENMESİ 8
2.2.1. Makine Öğrenmesinin Tarihsel Gelişimi 8
2.2.2. Makine Öğrenmesi Algoritmaları 9
2.2.3. Denetimli Öğrenme 9
2.2.4. Denetimsiz Öğrenme 10
2.2.5. Pekiştirmeli Öğrenme 11
3. LİTERATÜR TARAMASI 12
3.1. ŞİFRELEME ALGORİTMALARI ANALİZLERİ 12
3.2. IOT'DE KRİPTOGRAFİ 13
3.3. KRİPTOGRAFİ VE MAKİNE ÖĞRENMESİ 14
3.4. TEZDEKİ MAKİNE ÖĞRENMESİ ALGORİTMALARI 17
3.4.1. Lojistik Regresyon 17
3.4.2. k-En Yakın Komşular 17
3.4.3. Karar Ağaçları 17
3.4.4. Destek Vektör Makineleri 18
3.4.5. Rastgele Ormanlar 18
3.4.6. Histogram Tabanlı Gradyan Artırma 18
3.4.7. Aşırı Gradyan Artırma 19
3.5. Literatür değerlendirilmesi 19
4. MATERYAL VE YÖNTEM 21
4.1. YÖNTEM AKIŞI 21
4.1.1. Sentetik Veri Üretimi 21
4.1.2. Tarama (Screening) 21
4.1.3. İç İçe Çapraz Doğrulama 22

4.1.4. Eğitim-test Ayrımı (Hold-out) **22**
4.2. VERİ SETİ OLUŞTURMA **22**
4.2.1. Dosya Boyutu (KB): **24**
4.2.2. Şifreleme Süresi (ms) **24**
4.2.3. Deşifreleme Süresi (ms) **24**
4.2.4. CPU Kullanımı (%) **24**
4.2.5. Bellek Tüketimi (MB) **24**
4.2.6. Enerji Tüketimi (mJ) **24**
4.3. TARAMA **25**
4.3.1. Performans Ölçütleri **25**
4.3.1.1. Doğruluk *25*
4.3.1.2. Kesinlik *25*
4.3.1.3. Duyarlılık *25*
4.3.1.4. Makro-F1 ..... *26*
4.4. DOĞRULAMA **26**
4.4.1. İç içe Çapraz Doğrulama **26**
4.4.2. Eğitim-test Ayrımı **26**
4.4.3. Gürbüzlük **27**
5. BULGULAR VE TARTIŞMA **29**
6. SONUÇ VE ÖNERİLER **35**
7. KAYNAKLAR **37**
8. EKLER **40**
8.1. EK 1: JUPYTER ÇALIŞMA ORTAMI HAZIRLIK KODLARI **40**
8.2. EK 2: SENTETİK VERİ ÜRETİMİ KODLARI **41**
8.3. EK 3: HİPERPARAMETRE OPTİMİZASYON KODLARI **43**
8.4. EK 4: TARAMA KODLARI **46**
8.5. EK 5: İÇ İÇE ÇAPRAZ DOĞRULAMA KODLARI **50**
8.6. EK 6: EĞİTİM-TEST AYRIMI KODLARI **52**
8.7. EK 7: GÜRBÜZLÜK KODLARI **53**
8.8. EK 8: PER-CLASS ISI HARİTASI- HİSTGB, 70/30 KODLARI **54**
8.9. EK 9: KARIŞIKLIK MATRİSİ KODLARI **57**
8.10. EK 10: PERMÜTASYON TEMELLİ ÖZNİTELİK ÖNEMİ- HİSTGB, 70/30 KODLARI **58**
ÖZGEÇMİŞ **40**

ŞEKİL LİSTESİ

Sayfa No

Şekil 5.1. Yedi model için Makro-F1 ortalama. 29
Şekil 5.2. Nested CV (5x5) Makro-F1 ortalama. 30
Şekil 5.3. Makro-F1'in eğitim-test oranına göre değişimi. 31
Şekil 5.4. Eğitim süresi ortalama (10 tohum). 32
Şekil 5.5. Çıkarım süresi ortalama (10 tohum). 32
Şekil 5.6. Per-class ısı haritası- HistGB, 70/30. 33
Şekil 5.7. Karışıklık matrisi- HistGB, 70/30. 34
Şekil 5.8.Permütasyon temelli öznitelik önemi- HistGB, 70/30. 34

ÇİZELGE LİSTESİ

Sayfa No

Çizelge 2.1. Blok şifreleme (Stallings (2017)). 5
Çizelge 2.2. Akış şifreleme (Aumasson (2018)). 6
Çizelge 2.3. Simetrik ve simetrik algoritmalar (Stallings (2017); Aumasson (2018); Paar ve Pelzl (2010)). 7
Çizelge 2.4. Denetimli öğrenme algoritmaları (Hastie vd. (2009); Mitchell (1997); Murphy (2012)). 9
Çizelge 2.5. Denetimsiz öğrenme algoritmaları (Mitchell (1997); Murphy (2012)). 10
Çizelge 2.6. Pekiştirmeli öğrenme algoritmaları (Sutton vd. (2020); Mitchell (1997)). 11
Çizelge 3.1. Şifreleme algoritmalarının performans karşılaştırması (Panda (2016); Schneier (1996); Kuznetsov vd. (2024)). 12
Çizelge 3.2. ML tabanlı algoritma seçim performansları (Chinbat vd. (2024); Saleh vd. (2022); Gnatyuk vd. (2023)). 15
Çizelge 4.1. Değişken öznitelikler (Panda (2016), Kuznetsov vd. (2024)). 23
Çizelge 4.2. Sabit özellikler. 23
Çizelge 4.3. Tarama ortalama ± ss. 25
Çizelge 4.4. Nested CV ortalama±ss. 26
Çizelge 4.5. Eğitim-test ayrımı. 26
Çizelge 4.6. Gürbüzlük. 27

KISALTMALAR

AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
CPU	Central Processing Unit
DES	Data Encryption Standard
DT	Decision Tree
IoT	Internet of Things
IoTES	IoT Encryption Selection
KNN	K-Nearest Neighbour
LR	Logistic Regression
LWC	Lightweight Cryptography
ML	Machine Learning
NB	Naive Bayes
RF	Random Forest
RSA	Rivest–Shamir–Adleman Algorithm
SHA	Secure Hash Algorithm
SHAP	SHapley Additive exPlanations
SVM	Support Vector Machine

ÖZET

MAKİNE ÖĞRENMESİ İLE ŞİFRELEME ALGORİTMALARININ DİNAMİK SEÇİMİ

Ferdi AZBOY

Düzce Üniversitesi
Lisansüstü Eğitim Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi
Danışman: Doç. Dr. Esra ŞATIR

Ekim 2025, 58 sayfa

Bu çalışmada, makine öğrenmesi (ML) sınıflandırma algoritmaları kullanılarak sistem parametrelerine göre optimum şifreleme algoritmasının dinamik biçimde seçilmesi amaçlanmaktadır. Literatürde, kullanılan algoritmaların genellikle manuel olarak seçildiği görülmektedir. Bu manuel seçim, özellikle kaynak kısıtlı sistemlerde performans sorunlarına yol açmaktadır. Oysa gerçek dünya uygulamalarında birçok parametre algoritma seçiminde belirleyici olur. Bunlar; veri boyutu, şifreleme süresi, deşifreleme süresi, bellek kullanımı, işlemci kullanımı, enerji tüketimi vb. Özelliklerdir. Bu doğrultuda, dosya boyutu, şifreleme süresi, deşifreleme süresi, işlemci ve bellek kullanımı ve enerji tüketimi gibi 6 kriterden oluşan 2000 satırlık sentetik bir veri seti Python programlama dili kullanılarak üretilmiştir. Veri seti için simetrik şifreleme algoritmalarından Blowfish, Advanced Encryption Standard (AES), Rivest Cipher 6 (RC6), Serpent ve Kalyna seçilmiştir. Modelde veri yedi farklı ML algoritmalarıyla test edilmiş; Rastgele Ormanlar (RF), Aşırı Gradyan Artırma (XGBoost) ve Histogram Tabanlı Gradyan Artırma (HistGB) modelleri en yüksek başarıyı göstermiştir. Sonuçlar, modellerin Makro-F1 $\approx$ 0.916 – 0.918 düzeyinde tutarlı performans sergilediğini göstermiştir. Çıkarım süresi için en hızlı XGBoost, eğitim süresi için en hızlı RF ve Makro-F1 için ise en yüksek HistGB olmuştur. Sonuç olarak tasarlanan model sistemin ihtiyaçlarını odağa alan bir yapı önermekte ve bu bağlamda literatürdeki boşluğu doldurmayı amaçlamaktadır. Oluşturulan mimari; Nesnelerin İnterneti (IoT), mobil sistemler ve bulut bilişim ortamlarında uygulanabileceği gibi gelecekte açıklanabilir yapay zekâ araçlarıyla da kullanılarak modelin karar performansının artırılması sağlanabilir.

Anahtar Sözcükler: Dinamik Seçim, Makine Öğrenmesi, Kriptografi

ABSTRACT

DYNAMIC SELECTION OF ENCRYPTION ALGORITHMS WITH MACHINE LEARNING

Ferdi AZBOY
Düzce University
Graduate School, Department of Computer Engineering
Master's Thesis
Supervisor: Assoc. Prof. Dr. Esra ŞATIR

October 2025, 58 pages

This study aims to dynamically select the optimal encryption algorithm based on system parameters using machine learning (ML) classification algorithms. In the literature, it is observed that the algorithms used are generally selected manually. This manual selection leads to performance issues, especially in resource-constrained systems. However, in real-world applications, many parameters are decisive in algorithm selection. These include data size, encryption time, decryption time, memory usage, processor usage, energy consumption, and other characteristics. Accordingly, a 2000-line synthetic dataset consisting of 6 criteria, such as file size, encryption time, decryption time, processor and memory usage, and energy consumption, was generated using the Python programming language. Blowfish, Advanced Encryption Standard (AES), Rivest Cipher 6 (RC6), Serpent, and Kalyna were selected from symmetric encryption algorithms for the dataset. The data was tested in the model with seven different ML algorithms; Random Forests (RF), Extreme Gradient Boosting (XGBoost), and Histogram-Based Gradient Boosting (HistGB) models showed the highest success. The results show that the models performed consistently at a Macro-F1 level of ≈ 0.916 – 0.918. XGBoost was the fastest for inference time, RF was the fastest for training time, and HistGB was the highest for Macro-F1. In conclusion, the designed model proposes a structure focused on the needs of the system and aims to fill the gap in the literature in this context. The created architecture can be applied in the Internet of Things (IoT), mobile systems, and cloud computing environments, and the model's decision performance can be improved by using explainable artificial intelligence tools in the future.

Keywords: Dynamic Selection, Machine Learning, Cryptography

1. GİRİŞ

Kriptografi, tarih boyunca bireyler ve kurumların birbirleriyle olan iletişim ihtiyaçlarını gizli bir şekilde sağlamak için kullanılmıştır. Şifreleme bilimi ilk çağlarda Sezar şifrelemesi gibi basit yöntemlerle başlamış orta çağda buna kıyasla daha karmaşık olan Vigenère ile devam etmiştir. Yirminci yüzyılın ortalarında Enigma'nın kullanılması ve sonrasında kırılabilmesi kriptografi için bir dönüm noktası sayılmaktadır (Kahn, 1996). Modern çağda ise bu tarihsel bilgi birikimi ile daha gelişmiş teknolojiler ortaya çıkmıştır. Bunlar; açık anahtarlı sistemler, dijital imzalar ve kuantum kriptografi gibi ileri tekniklerdir. Günümüzde kriptografi sadece bilgi gizliliği sağlamanın ötesinde anlamlar ifade etmektedir. Sistem güvenliği, veri bütünlüğü ve kullanıcı gizliliği gibi birçok başlık bu temel altında yer alır. (Stallings, 2017; Katz & Lindell, 2020)

Bu tarihsel gelişim dijital çağda daha karmaşık güvenlik ihtiyaçlarını beraberinde getirmiştir. Çağımızda dijital veri güvenliği bireysel kullanıcıların yanı sıra endüstri, finans ve devlet kurumları için de kritik bir öncelik haline gelmiştir. Veri güvenliğinin öneminin bu boyutta artması şifreleme algoritmalarına olan ihtiyacı da arttırmıştır. Şifreleme algoritmaları, veri güvenliğini sağlamak ve yetkisiz kişilerin verilere erişmesini engellemek amacıyla geliştirilen sistemlerdir. (Stallings, 2017).

Kriptografik sistemler, temelde simetrik şifreleme algoritmaları ve asimetrik şifreleme algoritmaları olarak ikiye ayrılır. Simetrik yapıda hem şifreleme hem de şifre çözme için aynı anahtar kullanılmaktadır. Asimetrik yapıda ise iki anahtar (genel ve özel anahtar) vardır (Schneier, 1996). Simetrik algoritmalar genellikle veri şifrelemede, asimetrik algoritmalar ise çoğunlukla anahtar değişimi ve dijital imza için kullanılır. Bunun nedeni simetrik algoritmaların daha hızlı olmasından kaynaklanır. Asimetrik algoritmalar, karmaşık matematiksel yapıya sahip olduğundan simetriklere göre daha güvenlidir lakin veri şifrelemede daha yavaş kalır. (Katz & Lindell, 2020)

Geleneksel güvenlik sistemlerinde şifreleme algoritması seçimi genellikle güvenlik uzmanlarının deneyimine veya standart varsayımlara göre yapılmaktadır. Bu durumda sistemin ihtiyaçları değişse bile kullanılan şifreleme algoritması dışarıdan manuel bir müdahale olmadığı sürece sabit kalmaktadır. Örneğin AES algoritması yüksek güvenlik

seviyesi sunsa da bazı zaman-kritik senaryolar veya düşük işlem kapasiteli cihazlarda işlem süresinin uzunluğu ve kaynak tüketiminin fazlalığı nedeniyle uygun olmayabilmektedir (Panda, 2016). Literatürdeki çoğu çalışma da algoritma performanslarından bağımsız bir şekilde test edilmiş ve genellikle tek boyutlu yönleriyle incelenmiştir. Ancak sistemin performansında büyük yer kaplayan işlemci gücü, bellek kapasitesi, enerji tüketimi, veri boyutu gibi faktörler doğrudan algoritma tercihine etki etmesi beklenir (Gnatyuk vd., 2023).

Bilişim teknolojilerinin 1950'li yıllardan günümüze neredeyse üstel denebilecek bir hızla gelişmesi sonucunda çağımızda üretilen veri miktarı tarihte eşi görülmemiş bir boyuta ulaşmıştır. Bir analizde dünya üzerindeki verilerin yaklaşık %90'ının yalnızca son iki yılda üretildiği ifade edilmiştir (Al-Jarrah vd., 2015). Bu ciddi artış, günümüzde sıkça kullanılan "büyük veri" (big data) kavramını ortaya çıkarmıştır. Veri boyutu bu şekilde arttıkça veri güvenliğini sağlamak da zorlaşmaktadır. Bu durum sistem gereksinimlerine göre uygun şifreleme algoritmasının seçilmesini gerektirir. Dolayısıyla yeni güvenlik yaklaşımlarının geliştirilmesi gerekliliği ortaya çıkmıştır. Bu bağlamda kriptografi ile yapay zekâ tekniklerinin birleştiği yeni nesil güvenlik çözümleri öne çıkmaya başlamıştır (Saleh vd., 2022).

Modern kriptografik sistemlerde bir şifreleme algoritmasının henüz kırılmamış olması tek başına güçlü bir güvenlik göstergesi değildir. Asıl odaklanılan nokta bu algoritmanın neden güvenli olduğu ve bir saldırganın bunu ne kadar sürede kırabileceğidir. Bu nedenle günümüzde bir şifreleme algoritması teorik güvenlik düzeyinin yanında hız, kaynak kullanımı, donanım uyumu gibi kriterler de göz önünde bulunur. Buna en iyi örnek olarak IoT teknolojileri verilebilir. IoT teknolojisinin yaygınlaşması ile bu cihazlarla gerçekleştirilen yüksek boyutlu veri transferini de beraberinde getirmiştir. IoT sistemleri sınırlı kaynakla çalışan bir yapıya sahip olduğu için veri güvenliğini sağlayacak algoritma seçiminde birden fazla parametreyi inceleme ihtiyacı doğar. Bu yüzden tercih edilecek algoritmanın kaynak kullanım verimliliği ve IoT cihazlarda sergileyeceği performans göz önünde bulundurulmalıdır (Stallings, 2017).

Bu bağlamda, günümüz teknolojilerinin karmaşıklığı ve sistem gereksinimlerinin hızlı giderilme ihtiyacı şifreleme algoritmalarını seçilmesi konusunda yeni yaklaşımların bulunması gerekliliğini göstermektedir. Bu çerçevede ML tekniklerinin veri güvenliği alanında kullanımı gittikçe artmaktadır. Özellikle sınıflandırma tabanlı ML modelleri,

sistem parametrelerine dayalı karar destek araçları olarak değerlendirilmektedir (Panda, 2016).

Bu tez, ML' den yararlanarak sistem ihtiyaçlarına göre optimum şifreleme algoritmasının dinamik biçimde seçimini sağlayacak bir model sunmaktadır. Modelde ML sınıflandırma algoritmalarından RF, Destek Vektör Makineleri (SVM), HistGB, XGBoost, Karar Ağaçları (DT), Lojistik Regresyon (LR), k-En Yakın Komşular (KNN) algoritması kullanılmıştır. Bu mimaride Python programlama dili ile oluşturulan sentetik veri seti ML algoritmaları kullanılarak eğitilmiştir. Eğitilen model, çeşitli sistem parametrelerini kullanarak optimum olan şifreleme algoritmasını dinamik bir şekilde seçme yeteneğine sahip olmaktadır.

Literatürde bu yaklaşıma yakın bazı çalışmalar mevcuttur. Chinbat, Madanian, Airehrour & Hassandoust (2024), hafif şifreleme algoritmalarını Raspberry Pi 3 üzerinde test ederek ML algoritmalarıyla analiz etmiştir. Saleh, Jhanjhi, Abdullah & Saher (2022), IoTES adı verdikleri bir platformu aracılığıyla kullanıcı girdilerine dayalı algoritma seçimi yapmışlardır. Shafique, Mehmood, Alawida, Khan ve Khan (2022), Gnatyuk, Prisyazhnuk, Nosov & Nosenko (2023) gibi araştırmalar çeşitli veri tiplerinde ML ile şifreleme algoritması seçimi üzerine odaklanmıştır. Ancak bu araştırmalar yalnızca donanım testine ya da sınırlı sayıda ölçütün değerlendirilmesine dayanmaktadır. Modelin uygulanabilirliği henüz gerçek donanım ortamında test edilmemiştir. Ancak bu çalışma sayesinde benzer araştırmalara da kaynaklık edebilecek temel bir çerçeve önerisi oluşturulmaktadır.

Bu tezde, sistem parametrelerine göre en uygun şifreleme yöntemini dinamik olarak seçen ML tabanlı bir model önerilmektedir. Geliştirilen model, farklı koşulları temsil eden geniş bir sentetik veri seti üzerinde ML algoritmalarıyla eğitilmiştir. Böylece klasik yaklaşımlarda olduğu gibi sabit bir algoritma kullanmanın yerine modelin işlemci gücü, bellek kullanımı, veri boyutu ve güvenlik seviyesi gibi kriterlere göre duruma en uygun şifreleme algoritmasını otomatik olarak belirleyebilmesi sağlanmıştır. Bu yaklaşımın özgün katkısı, birçok sistem değişkenini bir arada dikkate alarak şifreleme algoritmalarını otomatik biçimde sınıflandırıp seçebilen bir mimari sunmasıdır.

Tez, ikinci bölümde kuramsal temeller, üçüncü bölümde literatür taraması, dördüncü bölümde materyal ve yöntem, beşinci bölümde elde edilen bulgular ve son bölümde de sonuçlara yer verilecek şekilde kurgulanmıştır.

2. KURAMSAL TEMELLER

2.1. KRİPTOGRAFİ

2.1.1. Kriptografinin Tarihsel Gelişimi

Kriptografi ve veri güvenliği, antik çağlardan günümüze kadar insanlar arasındaki iletişimin gizliliğini sağlamak amacıyla kullanılmıştır. Kullanılan tekniklerin neler olduğunu anlamak için kriptografinin kilometre taşlarına bakmak gerekmektedir. İlk olarak M.Ö. 1900'lerde Mısırlılar, hiyeroglifleri kullanarak basit gizleme teknikleri uygulamıştır. Sonrasında Roma İmparatoru Julius Caesar tarafından geliştirilen ve adıyla anılan Sezar şifreleme, bir metindeki harflerin sabit bir sayı kaydırılarak değiştirilmesi yöntemine dayanır. Bu yöntem monoalfabetik şifreleme olarak adlandırılır. Orta Çağ'da ise Vigenère gibi polialfabetik şifreleme yöntemleri geliştirilerek daha kompleks yapılar elde edilmiştir (Kahn, 1996).

II. Dünya Savaşı sırasında Almanlar tarafından kullanılan Enigma makinesi, dinamik polialfabetik şifreleme sistemine dayanmaktaydı. Bu makine çok güçlü ve karmaşık bir şifreleme yöntemi kullanıyordu ve bu durum Almanlara savaşta büyük üstünlük sağlamıştı. İngiliz matematikçi Alan Turing ve ekibinin Bombe cihazı adıyla geliştirdikleri sistem sayesinde Enigma'nın şifreleri çözülebildi. Bu olay sadece savaşın seyrini değiştirmekle kalmayıp aynı zamanda bilgisayar biliminin ve modern kriptografinin de doğuşuna yol açmıştır (Singh, 1999).

Savaş sonrasında Claude Shannon 1949 yılında yayımladığı çalışma ile modern kriptografinin teorik temelini oluşturmuştur. Bu dönemde geliştirilen Data Encryption Standard (DES), simetrik şifrelemenin ilk örneklerinden biri olmuş ve uzun yıllar güvenlik protokollerinde kullanılmıştır (Stallings, 2017). Diffie ve Hellman'ın 1976 yılında ortaya koyduğu "açık anahtarlı şifreleme" yaklaşımı ise Rivest-Shamir-Adleman (RSA) ve ECC gibi asimetrik algoritmaların gelişimini sağlamıştır (Diffie ve Hellman, 1976).

Günümüzde ise blok zinciri ve kuantum kriptografi gibi yeni nesil yaklaşımlar geliştirildi. Kriptografi basit yer değiştirme tekniklerinden son derece karmaşık matematiksel algoritmalara uzanan bir dönüşüm geçirmiştir. Çağlar boyunca güvenlik ihtiyacı değiştikçe kriptografik yaklaşımlardaki yeni yaklaşımlar da bunlara bir çözüm olmuştur (Menezes vd., 1996)

2.1.2. Şifreleme Algoritmaları

Şifreleme, dijital verilerin yetkisiz kişiler tarafından ele geçirilmemesi için matematiksel formüller kullanılarak dönüştürülmesi işlemdir. Alınan verinin çözülme işlemine de deşifreleme denir. Deşifreleme için gönderici ve alıcının aynı anahtar sahip olması gerekmektedir (Stallings, 2017). Şifreleme, gizlilik, bütünlük ve kimlik doğrulama gibi üç temel işlev sunar. Şifreleme sistemleri, "veri gizli mi?", "veri değişti mi?" ve "veri doğru kişiden mi geliyor?" sorularına cevap vermeye çalışarak bu üç ilkeyi sağlamayı hedeflerler (Katz ve Lindell, 2020).

Şifreleme algoritmaları, veri işleme biçimine ve anahtar yapısına göre iki farklı şekilde sınıflandırılır. İşleme biçimine göre sınıflandırma da algoritmaların veriyi nasıl parçalayıp şifrelediğine odaklanır. Veri işleme biçimine göre sınıflandırma kendi içinde ikiye ayrılır: blok şifreleme ve akış şifreleme (Katz ve Lindell, 2020).

2.1.2.1. Blok Şifreleme

Blok şifrelemede, veri sabit uzunluktaki bloklara ayrılır ve her blok şifrelendikten sonra veri birleştirilir. Örneğin, sabit olarak AES 128 bit, Blowfish ise 64 bit blok kullanır (Stallings, 2017). Çizelge 2.1'de blok şifreleme türlerinden üç algoritma karşılaştırılmıştır.

Çizelge 2.1. Blok şifreleme (Stallings (2017)).

Algoritma	Blok Boyutu	Anahtar Uzunluğu	Güçlü Yönler	Zayıf Yönler
AES	128 bit	128, 192, 256 bit	Yüksek güvenlik	Yavaş olabilir
Blowfish	64 bit	32–448 bit	Esnek anahtar boyutu, hızlı	Anahtar planı karmaşık
DES	64 bit	56 bit	Basit	Kırılmış, güvensiz

2.1.2.2. Akış Şifreleme

Akış şifrelemede veri bit veya bayt hâlinde küçük parçalara ayrılıp anlık şifrelenir. Akış şifreleme daha az kaynak tükettiğinden IoT gibi zaman-kritik uygulamalarda tercih

edilmektedirler (Aumasson, 2018). Çizelge 2.2 ile bazı blok ve akış şifreleme algoritmalarını karşılaştırılmaları yapılmaktadır.

Çizelge 2.2. Akış şifreleme (Aumasson (2018)).

Algoritma	Blok Boyutu	Anahtar Uzunluğu	Güçlü Yönler	Zayıf Yönler
ChaCha20	32 bit kelime temelli	128 veya 256 bit	Hızlı, mobil cihazlar için ideal	Daha az standartlaştırılmış
RC4	Bayt akışı	40-2048 bit	Basit ve hızlı, düşük kaynak tüketimi	Güvenlik açıkları mevcut
Salsa20	64 bit blok benzer	256 bit	Güvenli ve hızlı, modern tasarım	Yaygın değil, az destek

Anahtar yapısına göre sınıflandırma iki temel başlık altında gelişmiştir. Şifreleme ve deşifreleme işlemi için aynı anahtarı kullananlar simetrik, farklı anahtarı kullananlar ise asimetrik olarak adlandırılmaktadır (Stallings, 2017).

2.1.2.3. Simetrik Şifreleme

Simetrik şifreleme yöntemlerinde tek bir anahtar olduğundan o anahtarın alıcı tarafa güvenli bir şekilde ulaştırılması da kritik öneme sahiptir. DES, AES, RC6, Serpent ve Blowfish gibi algoritmalar bu sınıfa girer (Stallings, 2017). AES, Rijndael mimarisi üzerine inşa edilmiş olup yüksek güvenlik ve verimlilik sağladığı için modern sistemlerde yaygın olarak tercih edilir (Daemen ve Rijmen, 2002).

2.1.2.4. Asimetrik Şifreleme

Asimetrik şifreleme yöntemlerinde ise farklı iki anahtar (genel anahtar ve özel anahtar) mevcuttur. Genel anahtar açıktır ve herkesle paylaşılır. Gönderici, veriyi alıcının genel anahtarı ile şifreler. Alıcı ise özel anahtar sayesinde verinin şifresini çözer. Özel anahtarı sadece alıcı taraf (kullanıcı, sunucu, uygulama vb.) bilmektedir. Bu sayede anahtar paylaşımı daha güvenli bir şekilde gerçekleştirilmiş olur. RSA, Diffie-Hellman ve Eliptik Eğri Kriptografisi (ECC) bu grupta yer alır. Bu algoritmalar, özellikle dijital imza ve

kimlik doğrulama gibi işlemler için uygundur (Menezes vd., 1996). Çizelge 2.3 ile simetrik ve asimetrik algoritmaların karşılaştırılması yapılmaktadır.

Çizelge 2.3. Simetrik ve simetrik algoritmalar (Stallings (2017); Aumasson (2018); Paar ve Pelzl (2010)).

Özellik	Simetrik Şifreleme	Asimetrik Şifreleme
Anahtar Türü	Şifreleme ve deşifreleme için aynı anahtar kullanılır	Şifreleme için genel anahtar, deşifreleme için özel anahtar kullanılır
Hız	Yüksek	Düşük
Güvenlik	Anahtar paylaşımı risklidir	Anahtar paylaşımı daha güvenlidir
Kullanım Alanları	Büyük veri şifreleme, sabit bağlantılar	Dijital imza, anahtar değişimi, e-posta güvenliği
Kaynak Tüketimi	Daha az işlem gücü ve bellek gerekir	Daha fazla işlem gücü ve bellek gerekir
Algoritma Örnekleri	AES, DES, RC6, Serpent, Blowfish	RSA, ECC, Diffie-Hellman

Genel olarak değerlendirildiğinde simetrik algoritmalar yüksek hız ve düşük kaynak kullanımı ile daha çok veri şifrelemede kullanılmaktadır. Asimetrik algoritmalar ise daha çok güvenlik ihtiyacının ön planda olduğu alanlarda kullanılmaktadır. Bu yönleriyle anahtar paylaşımı ve dijital doğrulama için asimetrik algoritmalar tercih sebebi olmuştur. Günümüzde gerçek sistemlerin çoğunda bu iki yapının avantajlarını birleştiren hibrit kriptografik mimariler kullanmaktadır. Bu sayede hem hız hem de güvenlik gereksinimi sağlanmaya çalışılmıştır. Örneğin, Transport Layer Security gibi protokoller; oturum anahtarlarını asimetrik yöntemle paylaştıktan sonra veriyi de simetrik algoritmayla şifrelemektedir. (Katz ve Lindell, 2020; Stallings, 2017).

2.2. MAKİNE ÖĞRENMESİ

2.2.1. Makine Öğrenmesinin Tarihsel Gelişimi

ML, bilgisayarların programlanmadan verilen örneklerden kendi kurallarını öğrenmelerini amaçlayan bir yapay zekâ dalıdır. 1950'li yıllarda Alan Turing'in ortaya koyduğu "Turing Testi" kavramı bu alanın ilk adımlarından sayılır. Bu dönemde bilim insanları "Bir makine düşünebilir mi?" sorusunu sormakta ve tartışmaktaydılar. 1957'de Rosenblatt tarafından geliştirilen ve ilk sinir ağı modeli denebilecek Perceptron, ML' ye bir zemin oluşturmuştur. Ancak Minsky ve Papert'in 1969 yılında yayınladıkları "Perceptron" adlı kitapta bu modelin sınırlılıklarını göstermesiyle alana olan ilgi azalmış ve ML araştırmaları bir süre duraklamıştır (Mitchell, 1997).

1980'li yıllarda çok katmanlı yapay sinir ağları (YSA) ve özellikle geri yayılım algoritmasının geliştirilmesi ile ML' ye olan ilgi tekrar artmıştır. Bu dönemde DT, SVM ve istatistiksel öğrenme gibi farklı öğrenme yöntemleri geliştirilip denenmeye başlanmıştır (Mitchell, 1997).

1990'lı yıllardan itibaren ML hem matematik hem de bilgisayar bilimlerinin birleştiği bir alan hâline gelmiştir. Tom Mitchell (1997) bu dönemde ML' yi "deneyimle performansı artan sistemlerin geliştirilmesi" olarak tanımlamıştır. Bu tanım, öğrenmeyi nicel olarak ölçülebilir hale getiren ilk yaklaşımlardan biri olmuştur.

2000'li yıllardan sonra veri madenciliği, büyük veri ve güçlü bilgisayarların birleşmesi ile ML geniş çaplı bir alanda kullanılabilir oldu. Bu dönemde RF, KNN, SVM gibi yeni yöntemlerin etkililiği şirketler ve akademisyenlerin bu alana yönelmesini sağlamıştır (Hastie vd., 2009).

2010 sonrası derin öğrenme, ensemble yöntemler ve açıklanabilir yapay zekâ gibi daha gelişmiş yaklaşımların ortaya çıkması ile ML' de farklı bir evreye geçildiği söylenebilir. Zhou (2012) tarafından ortaya konan ensemble yöntemler, birden fazla modeli birleştirerek doğruluk ve dayanıklılık performansını önemli ölçüde iyileştirmiştir.

Bugün ML; sağlık, güvenlik, finans, ulaşım ve bunlara benzer birçok alanda kullanılmaktadır. Bu alanın tarihsel gelişimi öncelikle teorik çalışmalarla başlamış olup zamanla bu birikim sayesinde ortaya çıkan yaklaşımlar günlük hayattaki ihtiyaçlara da yanıt verecek şekilde biçimlenmiştir.

2.2.2. Makine Öğrenmesi Algoritmaları

ML, giriş olarak aldığı verilerdeki örüntüleri öğrenerek genellemeler yapan ve bunlara dayalı olarak karar veren sistemlerdir (Mitchell, 1997). ML yöntemleri, kullanılan algoritmanın nasıl bir veriyle beslendiği ve nasıl karar verdiğine göre üç başlık altında sınıflandırılır: denetimli öğrenme, denetimsiz öğrenme ve pekiştirmeli öğrenme (Murphy, 2012).

2.2.3. Denetimli Öğrenme

Bu yaklaşımında algoritmaya girdi verisi ile çıktı da öğretilerek eğitilir. Amaç, yeni gelen benzer veriler için doğru çıktıyı tahmin etmesidir. Örnek algoritmalardan bazıları; DT, SVM, LR ve RF (Hastie vd., 2009). Tezin yaklaşımına uygun ML algoritmaları da bu başlık altına girmektedir. Bu algoritmaların özellikleri Çizelge 2.4' te detaylı bir şekilde incelenmiştir.

Çizelge 2.4. Denetimli öğrenme algoritmaları (Hastie vd. (2009); Mitchell (1997); Murphy (2012)).

Algoritma	Avantajları	Dezavantajları	Kullanım Alanları
DT	Yorumlaması kolay, hızlı sonuç verir	Aşırı öğrenme riski vardır	Sınıflandırma, karar destek sistemleri
SVM	Karmaşık sınırlar çizebilir, yüksek doğruluk sağlar	Hiperparametre ayarı zordur, zaman alır	Görüntü işleme, biyoinformatik
LR	Basit, hızlı ve yorumlanabilir sonuçlar üretir	Karmaşık ilişkileri modelleyemez	Tıbbi tanı, müşteri segmentasyonu
RF	Aşırı öğrenmeyi azaltır, yüksek doğruluk sağlar	Hesaplama maliyeti yüksektir	Güvenlik, finansal tahminler
KNN	Parametrik olmayan, düzensiz karar sınırlarını iyi yakalar	Tahmin zamanı ve bellek maliyeti yüksek, ölçek duyarlı	Tablosal veride güçlü sınıflandırma

Çizelge 2.4 (devamı). Denetimli öğrenme algoritmaları (Hastie vd. (2009); Mitchell (1997); Murphy (2012)).

HistGB	Her adımda kaybı indirger, düşük öğrenme oranı ve alt örnekleme ile iyi genelleme	Hiperparametrelere duyarlı; dikkatli düzenleme gerekir	Tablosal veride güçlü sınıflandırma/regresyon
XGBoost	Pratikte yüksek doğruluk	Hiperparametre uzayı geniş; yinelemeli eğitim maliyeti	Büyük ölçekli sınıflandırma, yarışma düzeyi uygulamalar

2.2.4. Denetimsiz Öğrenme

Bu yaklaşımda girdi verisine ait doğru cevaplar verilmez. Bu şekilde algoritmanın verilerdeki örüntüyü ya da verinin hangi gruba ait olduğunu kendiliğinden bulması sağlanmaya çalışılmaktadır (Murphy, 2012). Örnek algoritmalar: K-Means, DBSCAN, Principal Component Analysis (PCA). Bu algoritmaların özellikleri Çizelge 2.5'te detaylı bir şekilde incelenmiştir.

Çizelge 2.5. Denetimsiz öğrenme algoritmaları (Mitchell (1997); Murphy (2012)).

Algoritma	Avantajları	Dezavantajları	Kullanım Alanları
K-Means	Basit ve hızlı, büyük verilerde etkili	Küme sayısı önceden bilinmelidir	Pazarlama, müşteri segmentasyonu
DBSCAN	Karmaşık yapıları iyi ayırt eder, küme sayısı gerekmez	Gürültülü verilere duyarlıdır	Jeoloji, IoT ağları
PCA	Boyut indirgeme sağlar, örüntüleri ortaya çıkarır	Veri yorumlaması gerekebilir, anlam karmaşası olabilir	Veri keşfi, öznitelik seçimi

2.2.5. Pekiştirmeli Öğrenme

Bu modellerde algoritmanın deneme-yanılma yöntemiyle öğrenmesi ve zamanla en doğru kararları vermesi amaçlanmaktadır. Bir oyun gibi doğru kararlar ödüllendirilir ve hatalar ise cezalandırılır. Örnek: Q-learning, DQN. Bu algoritmaların özellikleri Çizelge 2.6' da detaylı bir şekilde incelenmiştir.

Çizelge 2.6. Pekiştirmeli öğrenme algoritmaları (Sutton vd. (2020); Mitchell (1997)).

Algoritma	Avantajları	Dezavantajları	Kullanım Alanları
Q-learning	Çevrim içi öğrenmeye uygundur, çevresel geri bildirime duyarlıdır	Ödül tanımı karmaşıktır	Robotik, oyunlar
DQN	Görsel ve çok boyutlu ortamlarda iyi çalışır, sinir ağı ile güçlendirilmiştir	Eğitim süresi uzundur, kaynak tüketimi yüksektir	Görüntü tabanlı öğrenme, kontrol sistemleri

3. LİTERATÜR TARAMASI

3.1. ŞİFRELEME ALGORİTMALARI ANALİZLERİ

İkinci bölümde temel kavramsal çerçevesi sunulan şifreleme algoritmaları, bu bölümde literatürdeki ampirik veriler ışığında performans açısından analiz edilmiştir.

Literatürde, hız, verimlilik ve veri işleme hızı (throughput) gibi performans ölçütleri baz alınarak şifreleme algoritmalarının karşılaştırılması ile çok sayıda çalışma mevcuttur. Birçok çalışmada farklı algoritmalar karşılaştırılarak belirli senaryolarda avantajlı ve dezavantajlı yönleri incelenmiştir. Örneğin, Panda (2016) tarafından yapılan çalışmada metin, ikili ve görsel dosyalar üzerinde AES, DES, Blowfish ve RSA algoritmaları denenmiş ve performansları kıyaslanmıştır. Panda (2016), Schneier (1996) ve Kuznetsov, Zhuk, Taran ve Barabanov (2024) çalışması referans alınarak oluşturulan Çizelge 3.1' de şifreleme algoritmalarının temel performans karşılaştırması yapılmıştır.

Çizelge 3.1. Şifreleme algoritmalarının performans karşılaştırması (Panda (2016); Schneier (1996); Kuznetsov vd. (2024)).

Algoritma	Algoritma Türü	İşlem Süresi	Veri İşleme Hızı	Güvenlik Seviyesi
AES	Simetrik	150	250	Yüksek
Blowfish	Simetrik	140	270	Yüksek
DES	Simetrik	250	160	Düşük
RSA	Asimetrik	980	30	Çok Yüksek

Çalışmaya göre AES ve Blowfish 'in en düşük işlem süresi ve en yüksek veri işleme hızına sahip olduğu anlaşılmıştır. DES 'in ise AES ve Blowfish'ten daha yavaş olduğu görülmüştür. RSA ise güçlü kriptografik yapısı sayesinde yüksek güvenlik sağlamasına karşın büyük veri bloklarında yavaş kalmıştır. Bu da RSA' nın işlem süresi açısından dezavantajlı olduğunu göstermektedir.

Kuznetsov vd. (2024) araştırmalarında, zaman-kritik siber güvenlik uygulamaları için modern simetrik şifreleme algoritmalarının (AES, Strumok, SNOW 2.0, HC-128 vb.) karşılaştırmasını yapmışlardır. Karşılaştırmada algoritmaların veri işleme hızı ve başlangıç gecikmesi gibi performans metrikleri ölçülmüş; Strumok ve SNOW 2.0 algoritmaları yüksek hız ve düşük kurulum süreleriyle öne çıkmıştır. AES algoritması kriptografik sağlamlık açısından üstün olsa da işlem hızı açısından bazı algoritmalara göre yavaş kalmıştır. Bu bulgular, uygulama senaryolarına göre algoritma seçiminde en az güvenlik kadar veri işleme hızı ve donanım uyumunun da göz önünde bulundurulması gerektiğini ortaya koymaktadır.

Literatürde farklı platformlar ve algoritmalar için benzer performans değerlendirmeleri yer almaktadır. Genel olarak birçok çalışmada tek bir şifreleme algoritmasının tüm koşullar için en iyi sonucu veremeyeceğini ifade etmektedir. Seçimin verinin boyutu, işlem süresi ve donanım kapasitesi gibi etkenlere bağlı olarak değişebileceğini ortaya koymaktadır. Şifreleme algoritmalarını karşılaştıran literatüre bakıldığında değişen sistem ihtiyaçlarına göre dinamik seçim mekanizmalarının geliştirilmesi için önemli bir bakış açısı sunmaktadır.

3.2. IOT'DE KRİPTOGRAFİ

IoT sistemlerde kullanılan cihazlar çoğunlukla kısıtlı işlemci gücü, bellek ve batarya kapasitesine sahiptir. Bundan dolayı kriptografik yöntemlerin bu ortamlarda uygulanması geleneksel sistemlere göre farklı bir optimizasyon gerektirir. Yüksek güvenlik seviyesine sahip bir algoritmanın (AES, RSA vb.), sınırlı kaynaklı bir IoT cihazında çalıştırılması sistemde beklenen performansın alınmamasına sebep olabilir. Bu durum, literatürde hafif şifreleme adıyla anılan algoritmalara olan ilgiyi arttırmıştır. Literatür taramasının önceki başlıklarında değinildiği gibi IoT cihazlarında algoritma seçerken güvenliğin yanı sıra hız, bellek kullanımı ve enerji tüketimi gibi metriklerin büyük önem taşıdığı görülmektedir (Radhakrishnan vd, 2024). Örneğin, Radhakrishnan, Jadon ve Honnavalli (2024), IoT cihazlarda AES-128 ve ASCON, SPECK gibi hafif şifreleme algoritmalarının güvenlik, hız ve bellek tüketimi kriterlerine göre kıyaslamıştır. Elde edilen bulgular, bazı hafif algoritmalarının IoT ortamlarında klasik algoritmalara kıyasla daha hızlı çalışabildiğini ve daha az kaynak harcadığını ortaya koymaktadır. Bu da IoT uygulamalarında, AES gibi güvenlik düzeyi yüksek ancak kaynak kullanımı fazla olan

algoritmaların yerine SPECK veya ASCON gibi daha hafif algoritmaların tercih edilebileceğini göstermektedir.

Literatürdeki Chinbat vd. (2024) çalışmasında, daha önce bahsedildiği gibi Raspberry Pi gibi bir IoT donanımı üzerinde hafif blok şifrelerin performansını test etmiş ve RECTANGLE algoritmasının en yüksek performansı sergilediği görülmüştür. Benzer şekilde, Kuznetsov vd. (2024) de IoT uygulamalarında AES' in her zaman yeterince hızlı olmadığını bazı modern algoritmaların (ör. SNOW 2.0) daha uygun olabileceğini vurgulamıştır. Bu çalışmalar ışığında IoT sistemlerinde kriptografi, sadece güvenlik seviyesine göre değil, cihazın işlem kapasitesi ve enerji bütçesine göre optimize edilmesi gereken bir alan olarak karşımıza çıkmaktadır. NIST' in hafif şifreleme standartlarını belirlemek üzere yarışmalar düzenlemesi ve sonucunda ASCON gibi algoritmaları standartlaştırması da bu ihtiyacın bir göstergesidir.

Özetle, IoT ve benzeri kaynak kısıtlı sistemlerde kullanılacak kriptografik çözümlerin seçiminde çok boyutlu bir yaklaşım gerekmektedir. Hem literatürdeki çalışmaların işaret ettiği üzere, güvenlik, hız, bellek kullanımı ve enerji tüketimi gibi kriterlerin dengeli bir şekilde değerlendirilmesi; hem de uygulama senaryosunun gerektirdiği minimum güvenlik düzeyinin altına düşülmemesi önemlidir. Bu nedenle, IoT ekosisteminde kriptografi uygulamaları için geliştirilen modern yaklaşımlar, farklı algoritmaların çeşitli metrikler altında değerlendirilmesini ve sistem gereksinimine en uygun algoritmanın uyarlanmasını amaçlamaktadır. Bu tez çalışmasının motivasyonu da bu noktadan gelmekte olup, IoT gibi dinamik ve kısıtlı ortamlarda uygun şifreleme algoritmasının akıllıca seçilmesi konusunda literatüre katkı sunmayı hedeflemektedir.

3.3. KRİPTOGRAFİ VE MAKİNE ÖĞRENMESİ

Son yıllarda ML teknikleri, şifreleme algoritmalarının performans analizi ve uygun algoritmanın seçiminde kullanılmaya başlanmıştır. Özellikle sınıflandırma algoritmaları, sistemin kaynak kullanımı, veri tipi ve performans gereksinimlerine göre en uygun şifreleme yöntemini dinamik olarak belirleyebilme potansiyeline sahiptir. ML tabanlı dinamik algoritma seçimi konusunda literatürde çeşitli örnekler mevcuttur.

Türk ve Şamlı (2020) tarafından yapılan bir başka çalışmada ise klasik şifreleme algoritmalarıyla şifrelenmiş YSA ile sınıflandırılmıştır. Ötelemeli, Doğrusal, Yer Değiştirme ve Vigenère gibi geleneksel algoritmalar kullanılarak şifrelenmiş metinler

oluşturulmuş ve bu metinler bir YSA modeli ile analiz edilmiştir. Sonuçlar, Ötelemeli şifreleme sınıfının %100 doğruluk oranıyla sınıflandırma yaptığını göstermiştir. Türk ve Şamlı (2020), bu çalışmada klasik kriptografik yapıların YSA ile başarıyla sınıflandırılabileceğini göstermiştir. ML modellerinin şifreli verideki desenleri yakalayabilmesi algoritma seçimi problemlerinde de öğrenme tekniklerinin başarılı olabileceğine işaret etmektedir.

Chinbat vd. (2024), IoT tabanlı bir senaryoda sekiz farklı hafif şifreleme algoritmasını (AES, PRESENT, LEA, RECTANGLE, vb.) çeşitli boyutlardaki dosyalarla deneyerek beş farklı makine öğrenimi (RF, SVM, MLP, DT, KNN), bu algoritmaların performanslarını karşılaştırmışlardır. Chinbat vd. (2024), Saleh vd. (2022) ve Gnatyuk vd. (2023) çalışmaları derlenerek Çizelge 3.2 ML tabanlı algoritma seçim performansları çizelgesi hazırlanmıştır.

Çizelge 3.2. ML tabanlı algoritma seçim performansları (Chinbat vd. (2024); Saleh vd. (2022); Gnatyuk vd. (2023)).

Model	Test Ortamı	Doğruluk (%)	Makro-F1	Seçilen Algortima	Uygulandığı Kaynak
RF	Raspberry Pi 3	94	0.93	RECTANGLE	Chinbat vd. (2024)
SVM	Raspberry Pi 3	89	0.88	SPECK	Chinbat vd. (2024)
XGBoost	Pi Zero / Beagle	96	0.94	SIMON	Saleh vd. (2022)
Gradient Boosting	Pi Zero / Beagle	93	0.90	PRESENT	Saleh vd. (2022)
YSA/ Deep Learning	UAV dataset	91	0.91	AES + RC6	Gnatyuk vd. (2023)

Chinbat vd. (2024) çalışmasında algoritma seçimi için eğitilen modeller arasında RF en yüksek doğruluk ve Makro-F1'e ulaşarak en başarılı makine öğrenimi yöntemi olmuştur. Model Raspberry Pi 3 donanımı üzerinde çalıştırılmış ve en uygun algortima olarak RECTANGLE algoritmasını seçmiştir. Ayrıca, bu sonuçlar, IoT gibi kaynak kısıtlı

ortamlarda makine öğrenimi destekli bir karar mekanizması ile şifreleme algoritması seçiminin mümkün olabileceğini göstermektedir.

Başka bir araştırmada ise Saleh vd. (2022), IoTES adıyla bir model geliştirmişlerdir. IoTES, kullanıcıdan alınan CPU gücü, bellek kapasitesi, güvenlik seviyesi gibi parametrelere göre optimum şifreleme algoritması öneren bir karar destek sistemidir. Çalışmada AES, RSA, ChaCha20, TEA gibi toplam 28 farklı algoritma farklı donanımlar (Raspberry Pi 3, Pi Zero, Pocket BeagleBoard vb.) üzerinde test edilmiştir. Geliştirilen sistemin "IoTES with Security" adlı versiyonunda işlem süresinin dışında güvenlik düzeyi de değerlendirmiştir. Algoritma önerisi için RF, Gradient Boosting ve XGBoost gibi makine öğrenimi modelleri tercih edilmiştir. Karşılaştırmalar sonucunda XGBoost %96 ile en yüksek doğruluk oranına ulaşan model olmuştur. Saleh vd. (2022)'nin çalışması, IoT cihazların kısıtlarına uygun şekilde güvenlik ve performansı dengeleyen bir şifreleme seçiminin, makine öğrenimi ile yüksek doğrulukta gerçekleştirilebileceğini ortaya koymuştur.

Makine öğrenimi ve kriptografiyi bir araya getiren bir diğer güncel çalışma da Gnatyuk vd. (2023)'ün sunduğu yapıdır. Bu çalışmada insansız hava aracı (İHA/UAV) sistemlerinde kullanılan görüntü verilerinin güvenliği için bir veri kümesi oluşturulmuştur. Bu veri kümesi çeşitli şifreleme algoritmaları ile çoklu kriterler temelinde puanlanmıştır. AES, RC6, Serpent, Blowfish ve Kalyna dahil 14 farklı simetrik şifreleme algoritması; kriptografik dayanıklılık, şifreleme hızı, anahtar genişletme verimliliği gibi ölçütlere göre değerlendirilmiştir. Daha sonra bu özellikler kullanılarak bir YSA modeli eğitilmiş ve farklı senaryolarda en uygun algoritmayı tahmin etmesi sağlanmıştır. Model, girdi koşullarına göre her bir senaryoda doğru algoritmayı seçmeyi büyük ölçüde başarmış ve dinamik algoritma seçimi için yapay öğrenmenin uygulanabilir olduğunu göstermiştir. Özellikle Kalyna gibi standart dışı algoritmaların da dahil edilmesi, modelin geniş bir yelpazede genelleme yapabildiğini ortaya koymuştur.

Bu başlık altında incelenen çalışmalara bakıldığında, ML tekniklerinin şifreleme algoritmasının dinamik seçiminde başarılı sonuçlar verdiği görülmektedir. Bu tez çalışması literatürdeki bu deneyimleri referans almakta ve kapsamını genişleterek özgün bir model önermektedir. Önerilen model, bahsedilen çalışmalardan farklı olarak daha fazla sistem parametresini (işlemci kullanımı, bellek tüketimi, veri boyutu vb.) değerlendirerek karar vermektedir. Sonuç olarak geliştirilen model, dinamik şifreleme algoritması seçimi literatürüne çok yönlü bir katkı sunmakta; işlemci gücü, bellek, veri

boyutu gibi kriterleri göz önüne alarak sistem bağlamına en uygun algoritmayı öneren bir yapay zekâ destekli çözüm sağlamaktadır.

3.4. TEZDEKİ MAKİNE ÖĞRENMESİ ALGORİTMALARI

3.4.1. Lojistik Regresyon

İkili ve çok-sınıflı sınıflandırmada olasılıkları logit/softmax ile modelleyen doğrusal tabanlı yöntemdir. Çok-sınıflı durumda genellikle "one-vs-rest" ya da softmax lojistik regresyon kullanılır. Amaç fonksiyonu negatif log-olasılık (cross-entropy) olup L2 düzenleme ile genelleme gücü artırılır. Özelliklerin standartlaştırılması katsayılara ortak bir ölçek sağlar ve optimizasyonu (lbfgs) kararlı kılar. Hiperparametre olarak C (düzenleme gücü), penalty (genellikle l2) ve max_iter öne çıkar. Yorumlanabilirlik, hızlı eğitim ve sağlamlık avantajları arasında sayılırken çizgisel ayrılabilirlik varsayımına duyarlılık ve karmaşık karar sınırlarında düşük kapasite de sınırlılıklarından bazılarıdır. En çok tıp, sosyal bilimler ve mühendislikte sınıf olasılığı tahmini için kullanılır. Modern biçimi 1950'ler ve 60'larda istatistik literatüründe yerleşti (Hastie vd., 2009).

3.4.2. k-En Yakın Komşular

KNN, eğitim aşamasında parametre öğrenmeyen, tahminde ise örneğe en yakın k komşunun sınıflarını oylayan tembel (lazy) bir yöntemdir. Öklid uzaklığı yaygındır; ancak farklı ölçekli özniteliklerde standartlaştırma şarttır. Hiperparametreler: k ve weights (uniform/distance). Küçük k gürültüye, büyük k sınıf karışmasına yol açar; sınıf sınırları karmaşık olduğunda iyi, yüksek boyutta ve dengesiz ölçekte zayıftır. Yerel örüntüleri iyi yakalar, küçük-orta veri setlerinde güçlü bir referanstır. İlk çalışmalar 1950'ler; ML' de yaygın anlatımı 1990'lar ve sonrası ders kitaplarıyla yerleşti (Hastie vd., 2009).

3.4.3. Karar Ağaçları

DT, öznitelik alanını yinelemeli bölerek yapraklarda sınıf dağılımı üreten ayrıklaştırıcı modellerdir. Safsızlık ölçütü (Gini/entropi) ile en iyi bölme seçilir; max_depth, min_samples_split gibi budama/sınırlama hiperparametreleri aşırı uyumu engeller. Yorumlanabilirlik yüksektir; doğrusal olmayan etkileşimleri yakalar. Ancak tek ağaç değişkenliğe duyarlıdır ve küçük veri dalgalanmalarında kararsız olabilir. Bu nedenle topluluk yöntemleri (RF, XGBoost) pratikte daha sağlamdır. Kripto senaryosunda DT,

örneğin yüksek throughput ve düşük enerji gibi kuralları açıkça ifade edebilir. Modern CART çerçevesi 1984'te yayımlandı (Hastie vd., 2009).

3.4.4. Destek Vektör Makineleri

SVM, veriyi çekirdek (kernel) hilesiyle yüksek boyuta gömüp marjini maksimize eden bir sınır bulur. RBF çekirdeği, doğrusal olmayan sınırları esnek biçimde temsil eder. Temel hiperparametreler: C (ceza), gamma (çekirdek ölçeği). Büyük C daha az ihlal, aşırı uyum riski; küçük C daha fazla yumuşama sağlar. gamma büyüdükçe sınır kıvrılır. Standartlaştırma şarttır; parametrik olmayan gücü nedeniyle dengeli beş sınıflı veri için güçlü bir adaydır fakat ölçek ve grid aralığına duyarlıdır. Orta ölçekli tablosal verilerde güçlü bir üst seviye sınıflandırıcıdır. Çekirdekli SVM' nin dönüm noktası 1995'tir (Cortes &Vapnik, 1995).

3.4.5. Rastgele Ormanlar

RF, bootstrap örneklem üzerinde büyütülen çok sayıda karar ağacının oylamasıdır. Her düğümde rastgele öznitelik altkümesi seçimi, ağaçlar arası korelasyonu düşürerek genellemeyi iyileştirir. Önemli hiperparametreler: n_estimators, max_depth, max_features, min_samples_leaf. Aşırı ayar gerektirmez, ölçeklemeye az duyarlıdır ve öznitelik önemi üretir. Zayıf yanı: çok yüksek boyutta tahmin süresi ve bellek tüketimi. Kriptografik performans özniteliklerinde RF genellikle güçlü bir "varsayılan iyi" yöntemdir. Az ayarla sağlam performans, değişken önemi ve ölçek duyarsızlığı sağlar. 2001'de önerildi (Breiman, 2001).

3.4.6. Histogram Tabanlı Gradyan Artırma

Gradyan artırma, zayıf öğrenenleri yinelemeli ekleyerek kayıp fonksiyonunun negatif gradyanını takip eder. Histogram tabanlı uygulamalar, öznitelikleri kovalara ayırıp bölme aramasını hızlandırır; bu yaklaşım LightGBM ile popülerleşmiştir. Hiperparametreler: learning_rate, max_depth, max_bins, l2_regularization. Düşük learning_rate daha çok ağaç gerektirir ama genelde daha iyi genelleme sağlar. HistGB, ağaç tabanlı olmasına rağmen iyi ayarlandığında SVM/RF ile rekabetçidir. Gradyan Artırma 2001'de sistematikleşti. Histogram yaklaşımı daha sonra hız/ölçeklenebilirlik için benimsendi (Friedman, 2001; Hastie, Tibshirani ve Friedman, 2009).

3.4.7. Aşırı Gradyan Artırma

XGBoost, ağaç artırmanın ölçeklenebilir ve düzenlemeli bir sürümüdür. İkinci mertebe yaklaşım, sütun örnekleme ve seyrekliğe duyarlı ayarlamalarla hem hız hem doğruluk sağlar. Hiperparametreler: n_estimators, max_depth, learning_rate, subsample, colsample_bytree, reg_lambda. Varsayılanlarda dahi rekabetçidir; ancak küçük veri ve gürültüde aşırı uyum riski olduğundan İç içe çapraz doğrulama (nested CV) içinde grid araması şarttır. Kripto bağlamında, çoklu öznitelik etkileşimlerini yakalamada güçlü bir "üst seviye" ağaç topluluğudur. Ağaç artırmayı ikinci mertebe yaklaşım ve düzenleme ile hızlandıran, sütun örnekleme ve seyrekliğe duyarlı bölmelerle ölçeklenebilirlik sunan kütüphanedir. Kaggle ve endüstride tablosal veride sık tercih edilir. Düzenleme sayesinde aşırı uyumu denetler. 2016'da geniş kabul gördü. (Chen ve Guestrin, 2016)

3.5. LİTERATÜR DEĞERLENDİRİLMESİ

Literatür incelemesi sonucunda şifreleme algoritmalarının seçiminde sadece bir veya iki özelliğin dikkate alınmasının yeterli olmadığı anlaşılmıştır. Yapılan çoğu çalışmada algoritmaların genellikle benzer ortamlarda test edildiği (örneğin güçlü bir bilgisayarda) görülmektedir. Lakin bu algoritmaların farklı cihazlarda (IoT cihazı, cep telefonu vb.) farklı performans gösterebileceği gerçeği göz ardı edilmiştir. Ayrıca kriptografik olarak güçlü ancak işlemciyi çok yoran algoritmaların (AES-256 gibi), özellikle IoT gibi kaynak kısıtlı sistemlerde verimli olmadığı anlaşılmaktadır. Bazı çalışmaların ML kullanılarak algoritma seçimi yaptığı ama modelin karar mekanizmasının açıklanabilirliğine dair bir bilgi vermediği gözlemlenmiştir.

Bu tezin literatüre katkıları aşağıda özetlenmektedir:

1. Dinamik algoritma seçimi için birden çok sistem özelliği (veri boyutu, CPU kullanımı, bellek, enerji tüketimi vb.) kullanılarak geniş bir sentetik veri seti oluşturulmuştur.

2. 7 farklı ML algoritması kullanılarak farklı sistem senaryolarına karşı uyumlu ve sağlam bir şifreleme algoritması seçimi başarıyla uygulanmıştır.

3. AES, Blowfish ve Kalyna gibi popüler algoritmaların yanı sıra AES yarışması finalistlerinden olan RC6 ve Serpent gibi iki algoritma da modele dahil edilerek kapsamlı bir kıyaslama yapılmıştır.

4. IoT cihazlarının donanım kısıtları göz önünde bulundurularak CPU kullanımı ve bellek kullanımı gibi kriterler modelde kullanılmıştır. Böylece modelin önerdiği algoritmaların gerçek senaryolarda işe yarayabilecek çözümler sunması amaçlanmıştır.

Sonuç olarak, bu tez yalnızca farklı algoritmaları kıyaslamakla kalmayıp aynı zamanda sistem özelliklerine dayalı olarak en uygun algoritmayı dinamik olarak seçebilecek bir model önermektedir. Bu mimari hem literatürdeki boşlukları doldurmayı hem de gelecekte gerçek hayat problemlerine uygulanabilir sistem tasarımlarına katkı sunmayı amaçlamaktadır.

4. MATERYAL VE YÖNTEM

Bu bölümde, geliştirilen karar destek modelinin mimarisi, veri seti oluşturma süreci, kullanılan öznitelikler, simülasyon ortamı ve doğrulama tasarımı açıklanmaktadır. Amaç, farklı şifreleme algoritmalarının sistem kaynaklarına etkisini nicel olarak modelleyip ölçülebilir metrikler (süre, CPU, bellek, enerji, dosya boyutu) üzerinden AES, Blowfish, RC6, Serpent, Kalyna sınıflarından uygun olanını dinamik seçebilen bir sınıflandırıcı geliştirmektir. Değerlendirme ölçütü olarak Makro-F1, doğruluk (accuracy), kesinlik (precision), duyarlılık (recall); uygulama açısından eğitim süresi (s) ve tahmin gecikmesi (ms/örnek) olarak belirlenmiştir (Hastie vd., 2009/2017; Zhou, 2012/2021).

Deneyler Windows 11, AMD Ryzen 7 5700U ve 16 GB RAM üzerinde yürütülmüştür. Python 3.12 ortamında scikit-learn, pandas, numpy ve xgboost kütüphaneleri kullanılmıştır. Tekrar üretilebilirlik için rastgelelik tohumları belirtilmiş (varsayılan SEED=42) ve her aşamanın ara çıktıları CSV dosyalarına kaydedilmiştir. (Hastie vd., 2009/2017; Zhou, 2012/2021)

Geliştirme Jupyter Notebook üzerinde yapılmıştır. Ölçek duyarlı modeller (LR, SVM, KNN) için StandardScaler içeren pipeline kullanılmış; ağaç-tabanlı modeller (RF, XGBoost, HistGB için ölçekleme gerekmemiştir. Çalışma akışı ve çıktı dizinleri otomatik üretilmiştir.

4.1. YÖNTEM AKIŞI

4.1.1. Sentetik Veri Üretimi

Literatür aralıklarına dayanarak 2000 gözlemden oluşan sentetik bir veri kümesinin üretilmesi ve betimsel doğrulaması yapılmıştır.

4.1.2. Tarama (Screening)

LR, KNN, DT, SVM, RF, HistGB ve XGBoost modellerinin ortak protokol altında tarama değerlendirmesi yapılmıştır.

4.1.3. İç İçe Çapraz Doğrulama

En güçlü üç aday (RF, XGBoost, HistGB) için iç katmanda ızgara arama (GridSearchCV), dış katmanda 5 katlı olacak şekilde nested CV performans tahmini yapılmıştır.

4.1.4. Eğitim-test Ayrımı (Hold-out)

80/20, 75/25, 70/30, 60/40 eğitim-test ayrımı senaryolarının sabit hiperparametre ve sabit tohum ile kıyaslanması yapılmıştır. Ayrıca 10–20 tohum (seed) tekrarlı gürbüzlük (robustness) analizinin raporlanması yapılmıştır. Bu protokol hem model seçimi yanlılığını azaltmakta hem de bölme oranı, tohum ve donanım etkilerini şeffaflaştırmaktadır. (Hastie vd., 2009/2017; Zhou, 2012/2021)

4.2. VERİ SETİ OLUŞTURMA

Bu çalışmada kullanılan veri seti, gerçek cihazlardan toplanan verilerin yerine Python ortamında yapay olarak oluşturulmuştur. Bu tercih, senaryo çeşitliliğinin daha fazla olmasını sağladığından modellemeyi ve algoritmaların verdiği tepkileri daha iyi analiz etmeyi mümkün kılmıştır. Veri seti 2000 satırla oluşturulmuş ve her bir gözlem için altı öznitelik tanımlanmıştır. Bu sayede her bir algoritma hem yüksek hem de düşük yükte test edilebilmiş ve model tüm sınıfları dengeli bir şekilde öğrenmiştir.

Veri seti oluşturulurken aşağıdaki yol haritası izlenmiştir:

1. İlk olarak modelin tahmin etmeye çalıştığı sınıf (örneğin AES, RC6, Serpent) algoritma tipi sütunu olarak belirlendi.

2. Her bir algoritma için farklı sistem koşulları (büyük veri, düşük CPU vb.), 200-300 arasında rastgele senaryo oluşturularak parametre olarak tanımlandı.

3. Şifreleme süresi, CPU ve bellek kullanımı, enerji tüketimi gibi veriler için normal dağılım, ortalama değerleri bilinmeyen parametreler için de uniform dağılım kullanılmıştır.

4. Modelin sadece “en sık görüleni” tahmin etmesini engellemek için her algoritma sınıfından benzer sayıda örnek üretildi.

5. Özniteliklerin aralıkları belirlendi.

Bu çalışmada kullanılan öznitelik aralıkları, literatürde raporlanan performans verilerinin ortalama değerleri dikkate alınarak belirlenmiştir. Kuznetsov vd. (2024), Panda (2016) ve Gnatyuk vd. (2023) gibi çalışmalardan alınan değerler, kullanılan simülasyon senaryosuna uyarlanmış ve tek bir donanım ortamı varsayımıyla yeniden ölçeklendirilmiştir. Örneğin AES için şifreleme süresi literatürde 640–780 ms aralığında raporlanmış olup, bu çalışmada 650–750 ms olarak alınmıştır.

Bu çalışmalar referans alınarak öznitelik dağılım aralıkları (simülasyon değerleri) için Çizelge 4.1 hazırlanmıştır. Bununla birlikte Çizelge 4.2' de diğer özniteliklerden farklı olan dosya boyutu ve güvenlik seviyesine yer verilmiştir.

Çizelge 4.1. Değişken öznitelikler (Panda (2016), Kuznetsov vd. (2024)).

Algoritma	Şifreleme Süresi (ms)	Deşifreleme Süresi (ms)	CPU Kullanımı (%)	Bellek Kullanımı (MB)	Enerji Tüketimi (mJ)
AES	650-750	680-770	60-80	10-20	80-120
Blowfish	300-400	320-420	40-60	8-16	50-90
RC6	400-600	420-620	50-70	10-18	60-100
Serpent	500-650	520-670	55-75	12-20	70-110
Kalyna	450-600	480-620	55-80	14-22	75-115

Çizelge 4.2. Sabit özellikler.

Öznitelik	Uygulama Açıklaması	Değer / Aralık
Dosya Boyutu	Tüm algoritmalar için sabit dağılımla üretilmiştir. Simülasyonda farklı dosya büyüklüklerini temsil eder.	500–5000 KB (uniform dağılım)

Veri seti oluşturulurken izlenen akış şeması sayesinde farklı sistem koşullarında algoritmaların performanslarının değerlendirilmesi ve ML algoritmalarının eğitimi için dengeli ve çeşitlendirilmiş bir veri kümesi elde edilmesi sağlamıştır.

Oluşturulan sentetik veri setinde her bir gözlem birimi için toplam altı farklı öznitelik tanımlanmıştır. Bu öznitelikler, sistem performansını ve güvenliğini etkileyen temel

faktörler göz önünde bulundurularak belirlenmiş ve modelin karar vermesinde ne kadar etkili oldukları da incelenmiştir. Aşağıda her öznitelik tek tek açıklanacak ve modeldeki rolü özetlenecektir:

4.2.1. Dosya Boyutu (KB):

Şifrelenecek verinin büyüklüğünü ifade eder. Dosya boyutu arttıkça algoritmaların hızı ve verimliliği değişebilir. Modelin, "Büyük boyutlu dosyalar için hangi algoritma tercih edilmelidir?" sorusuna cevap verebilmesi için bilmesi gereken bir girdidir.

4.2.2. Şifreleme Süresi (ms)

Algoritmanın belirli bir veriyi şifrelemesi için harcadığı süredir. Gerçek zamanlı sistemlerde modelin hızlı tepki vermesi bu özniteliğe bağlıdır. Özellikle zamanın öncelendiği uygulamalarda model bu bilgiye göre karar verir.

4.2.3. Deşifreleme Süresi (ms)

Şifrelenmiş verinin çözülmesi için gereken süredir. Performansın ön planda olduğu senaryolarda, şifreleme ve deşifreleme süresi birlikte değerlendirilir ve toplam işlem yükü analiz edilir.

4.2.4. CPU Kullanımı (%)

Şifreleme işlemi sırasında işlemcinin ne ölçüde kullanıldığını gösterir. Bu öznitelik, özellikle IoT ve gömülü sistemler gibi kısıtlı işlem gücüne sahip cihazlar için kritiktir. Seçilen algoritmanın CPU'yu fazla kullanması bu tür cihazlarda yavaşlamaya sebep olabilir.

4.2.5. Bellek Tüketimi (MB)

Algoritmanın işlem sırasında kullandığı ortalama RAM miktarını gösterir. Hafıza kapasitesi sınırlı olan sistemlerde algoritmanın bellek ihtiyacı düşük olmalıdır.

4.2.6. Enerji Tüketimi (mJ)

Şifreleme işlemi sırasında tüketilen enerji miktarını temsil eder. Batarya ile çalışan cihazlar açısından tüketilen enerji çok önemli bir değerdir.

Bu özniteliklerin her biri, modelin algoritma seçimi yapma kararında ölçüt olarak kullanılmıştır. Ayrıca model eğitildikten sonra hangi özniteliğin daha etkili olduğunu

hesaplanabilmektedir. Bu sayede, sistem tasarımı açısından en önemli kriterin hangisi olduğu da sonuç bölümünde tartışılmıştır.

4.3. TARAMA

Bu bölümde model havuzu, tarama amacıyla ortak bir protokol altında 5×tekrarlı katmanlı k-kat çapraz doğrulama (repeated stratified k-fold CV) ile değerlendirilmiştir. Birincil ölçüt Makro-F1' dir. Tarama sonuçları, RF, XGBoost ve HistGB modellerinin istikrarlı biçimde önde olduğunu göstermiş; ayrıntılı doğrulama bu üç adayla sürdürülmüştür. Çizelge 4.3' te de tarama sonuçlarının ortalama ± standart sapma (ss) değerleri görülmektedir.

Çizelge 4.3. Tarama ortalama ± ss.

Model	Doğruluk	Kesinlik	Duyarlılık	Makro-F1
RF	0.917 ± 0.014	0.920 ± 0.013	0.918 ± 0.014	0.918 ± 0.014
HistGB	0.916 ± 0.013	0.918 ± 0.013	0.917 ± 0.013	0.917 ± 0.013
XGBoost	0.915 ± 0.013	0.917 ± 0.012	0.916 ± 0.012	0.916 ± 0.012
DT	0.897 ± 0.012	0.898 ± 0.012	0.898 ± 0.012	0.898 ± 0.012
SVM	0.782 ± 0.016	0.782 ± 0.016	0.784 ± 0.016	0.782 ± 0.017
KNN	0.753 ± 0.020	0.758 ± 0.020	0.756 ± 0.020	0.755 ± 0.020
LR	0.547 ± 0.021	0.544 ± 0.022	0.549 ± 0.021	0.540 ± 0.021

4.3.1. Performans Ölçütleri

4.3.1.1. Doğruluk

Doğru tahmin edilen örneklerin tüm test örneklerine oranıdır.

4.3.1.2. Kesinlik

Pozitif tahminlerin ne kadarının gerçekten doğru olduğunu gösteren değerdir.

4.3.1.3. Duyarlılık

Gerçek pozitiflerin ne kadarının model tarafından doğru tespit edildiğini ifade eder.

4.3.1.4. Makro-F1

Kesinlik ve duyarlılık değerlerinin harmonik ortalamasıdır. Özellikle sınıflar dengeli olmadığında bile genel başarıyı görmede bir gösterge olarak kullanılır.

4.4. DOĞRULAMA

4.4.1. İç içe Çapraz Doğrulama

Hiperparametre yanlılığını azaltmak için nested CV uygulanmıştır. Dış katman 5 katlı, iç katman 5 katlıdır. İç katmanda GridSearchCV ile en iyi hiperparametreler seçilmiş, ölçüt Makro-F1 olarak belirlenmiştir. Her dış katmanda seçilen en iyi iç model dış test kümesinde değerlendirilmiş ve sonuçlar Çizelge 4.4' te ortalama ve ss biçiminde oluşturulmuştur.

Çizelge 4.4. Nested CV ortalama±ss.

Model	Doğruluk	Kesinlik	Duyarlılık	Makro-F1
RF	0.915 ± 0.014	0.918 ± 0.013	0.916 ± 0.014	0.916 ± 0.014
XGBoost	0.916 ± 0.010	0.917 ± 0.010	0.916 ± 0.010	0.917 ± 0.010
HistGB	0.917 ± 0.014	0.918 ± 0.014	0.918 ± 0.014	± 0.014

4.4.2. Eğitim-test Ayrımı

Çizelge 4.5 karşılaştırılabilirlik için 80/20, 75/25, 70/30, 60/40 eğitim-test ayrımı yapıldıktan sonra oluşan sonuçları içermektedir. En iyi üç aday model aynı protokolle eğitilip test edilmiştir.

Çizelge 4.5. Eğitim-test ayrımı.

Model	Bölme	Doğruluk	Kesinlik	Duyarlılık	Makro-F1	Eğitim Süresi (s)	Çıkarım Süresi (ms/örnek)
RF	80/20	0.89	0.898	0.891	0.893	0.348	0.135
XGBoost	80/20	0.915	0.917	0.916	0.916	0.643	0.022
HistGB	80/20	0.9	0.902	0.901	0.901	1.946	0.075
RF	75/25	0.904	0.911	0.905	0.906	0.408	0.108

Çizelge 4.5 (devamı). Eğitim-test ayrımı.

XGBoost	75/25	0.906	0.909	0.907	0.907	0.653	0.017
HistGB	75/25	0.902	0.904	0.903	0.903	1.828	0.066
RF	70/30	0.912	0.918	0.913	0.914	0.402	0.085
XGBoost	70/30	0.913	0.915	0.914	0.914	0.636	0.014
HistGB	70/30	0.917	0.918	0.918	0.918	1.795	0.057
RF	60/40	0.909	0.914	0.91	0.911	0.392	0.067
XGBoost	60/40	0.906	0.91	0.908	0.908	0.618	0.011
HistGB	60/40	0.906	0.908	0.907	0.908	1.739	0.048

4.4.3. Gürbüzlük

Seçilen modellerin farklı tohum değerlerine duyarlılığını görmek için deneyler 10 tohum ile yinelenmiştir. Performans ölçütleri olarak Makro-F1, doğruluk, kesinlik, duyarlılık; uygulama ölçütleri olarak da eğitim süresi (s) ve çıkarım süresi (ms/örnek) belirlenmiştir. Çizelge 4.6' da gürbüzlük işlemi sonrası oluşan sonuçlar verilmiştir.

Çizelge 4.6. Gürbüzlük.

Model	Bolme	Doğruluk	Kesinlik	Duyarlılık	Makro-F1	Eğitim Süresi (s)	Çıkarım Süresi (ms/örnek)
HistGB	60/40	0.912	0.914	0.914	0.914	1.732	0.048
HistGB	70/30	0.916	0.917	0.917	0.917	1.815	0.057
HistGB	75/25	0.917	0.918	0.918	0.918	1.869	0.065
HistGB	80/20	0.917	0.919	0.918	0.918	1.945	0.077
RF	60/40	0.916	0.919	0.918	0.918	0.412	0.068
RF	70/30	0.914	0.917	0.916	0.916	0.422	0.088
RF	75/25	0.918	0.92	0.919	0.919	0.422	0.109
RF	80/20	0.918	0.921	0.919	0.919	0.423	0.176
XGBoost	60/40	0.914	0.915	0.915	0.914	0.62	0.011

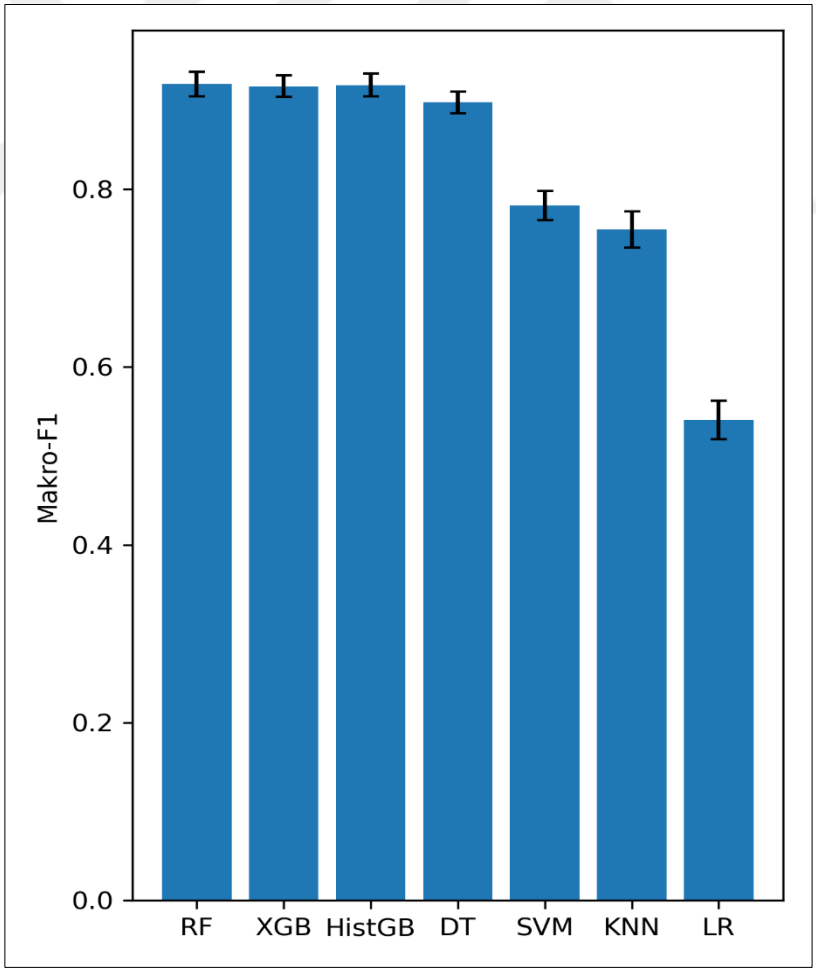
Çizelge 4.6 (devamı). Gürbüzlük.

XGBoost	70/30	0.918	0.92	0.919	0.919	0.642	0.014
XGBoost	75/25	0.921	0.923	0.922	0.922	0.668	0.017
XGBoost	80/20	0.922	0.923	0.922	0.923	0.671	0.021

5. BULGULAR VE TARTIŞMA

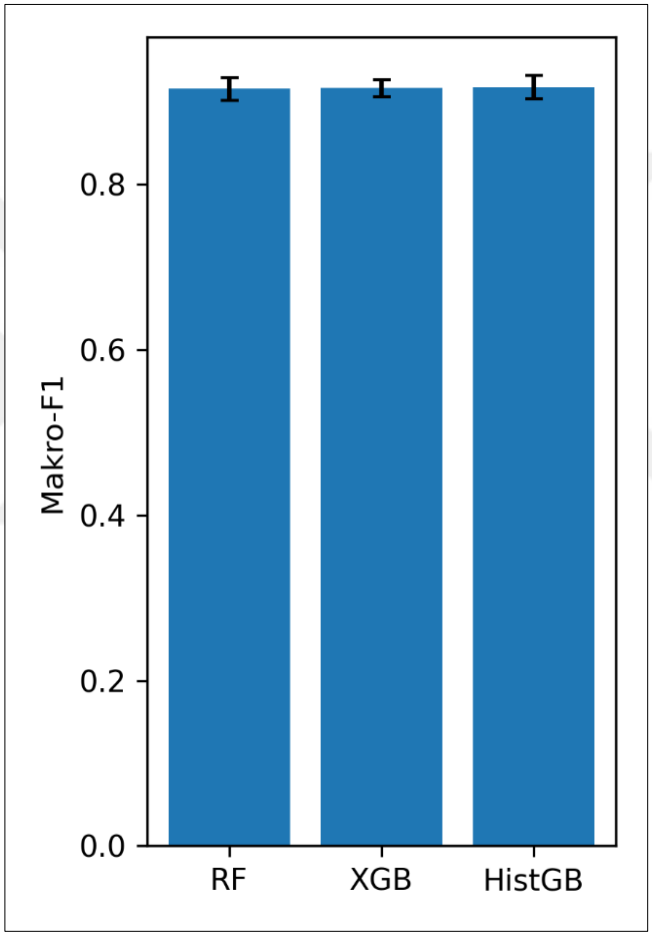
Bu bölümde, geliştirilen modelin test aşamasında elde ettiği bulgular değerlendirilmiştir. Amaç, ML algoritmalarıyla eğitilen modelin, sistem parametrelerine göre şifreleme algoritmalarını doğru şekilde seçip seçemediğini test etmektir. Model performansı; tarama, iç içe çapraz validasyon, eğitim-test ayrımı ve gürbüzlük aşamalarından çıkan sonuçlara göre sınıflandırma ölçütleri, öznitelik katkı etkileri ve sınıflar arasındaki ayırt edebilme yeteneği üzerinden yorumlanmıştır.

Şekil 5.1 ve Çizelge 4.3'te görüldüğü üzere 5×tekrarlı tarama sonuçlarına göre RF, XGBoost ve HistGB Makro-F1 bakımından ilk üçte yer almakta olup, aralarındaki farklar standart sapma düzeyinin altındadır (≤0.002). DT, LR, KNN ve SVM belirgin biçimde daha düşük bantta kalmıştır. Bu nedenle ayrıntılı doğrulama söz konusu üç adayla sürdürülmüştür.



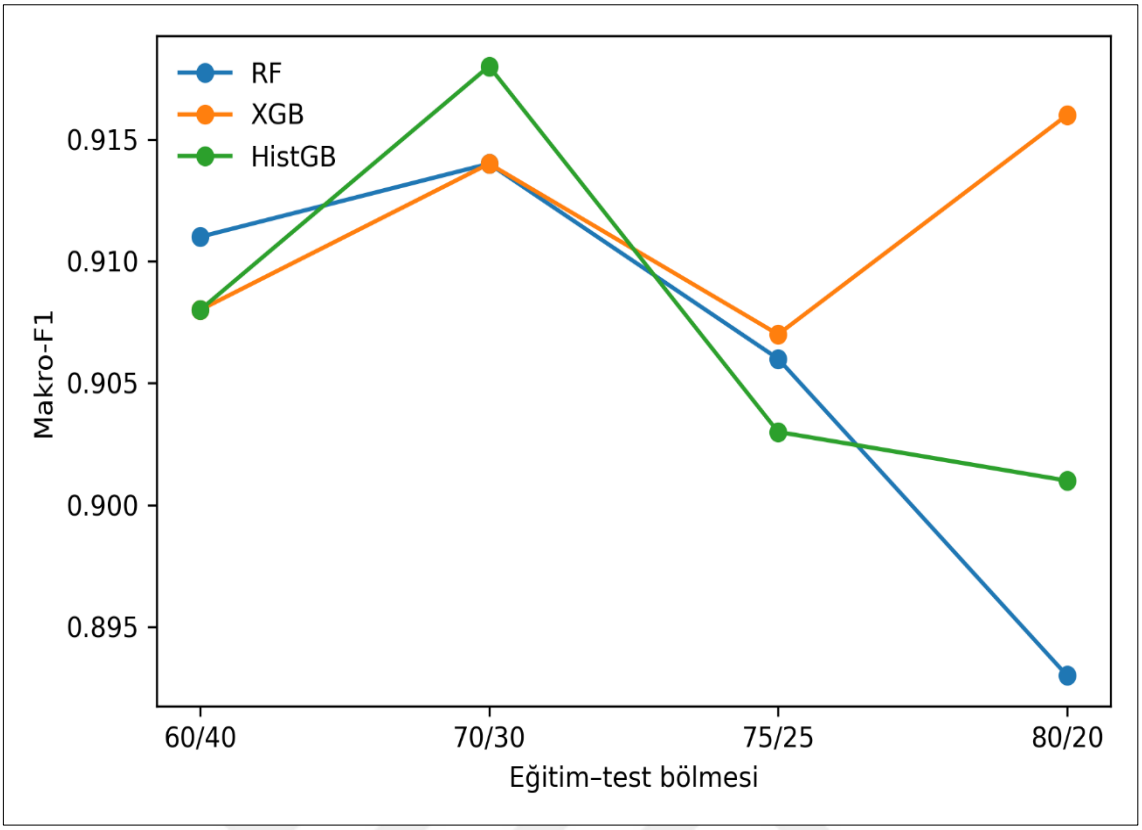
Şekil 5.1. Yedi model için Makro-F1 ortalama.

Şekil 5.2 ve Çizelge 4.4'teki nested CV bulguları, üç aday modelin Makro-F1 ortalamalarının birbirine çok yakın olduğunu göstermektedir (RF 0.916 ± 0.014; XGBoost 0.917 ± 0.010; HistGB 0.918 ± 0.014). Aralarındaki farklar, çoğu durumda standart sapma düzeyinin altındadır. Ayrıca XGBoost' nin sapması daha düşüktür (±0.010), bu da katmanlar arası daha kararlı performansa işaret eder. Doğruluk, kesinlik ve duyarlılıkta da benzer yakınlık gözlenmektedir. Bu nedenle nihai seçim, ikincil ölçütler olan çıkarım süresi, eğitim süresi ve tohumlar arası değişkenlik dikkate alınarak yapılmıştır.



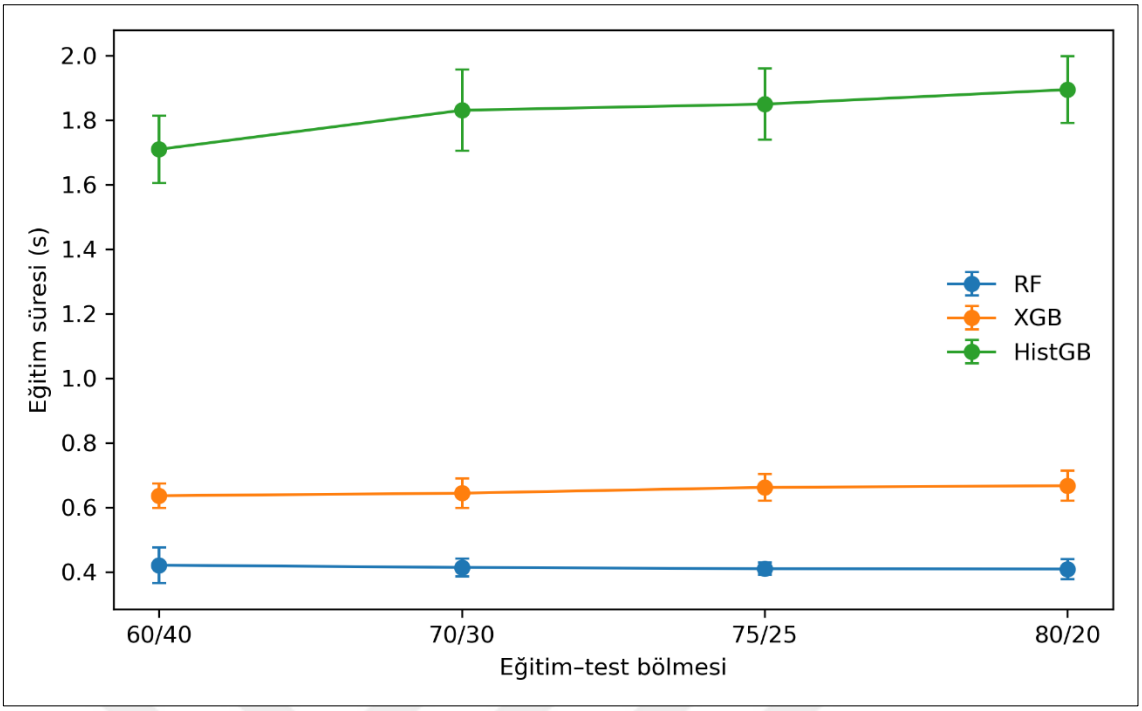
Şekil 5.2. Nested CV (5x5) Makro-F1 ortalama.

Şekil 5.3 ve Çizelge 4.5 incelendiğinde Makro-F1 bakımından 60/40' ta RF öndedir. 70/30' da en yüksek değer HistGB' dedir. 75/25 ve 80/20 oranlarında ise XGBoost' nin birinci olmuştur. Zaman profili tutarlıdır. Buna göre eğitim süresi RF <XGBoost <HistGB; çıkarım süresi XGB <HistGB <RF şeklinde oluşmuştur.

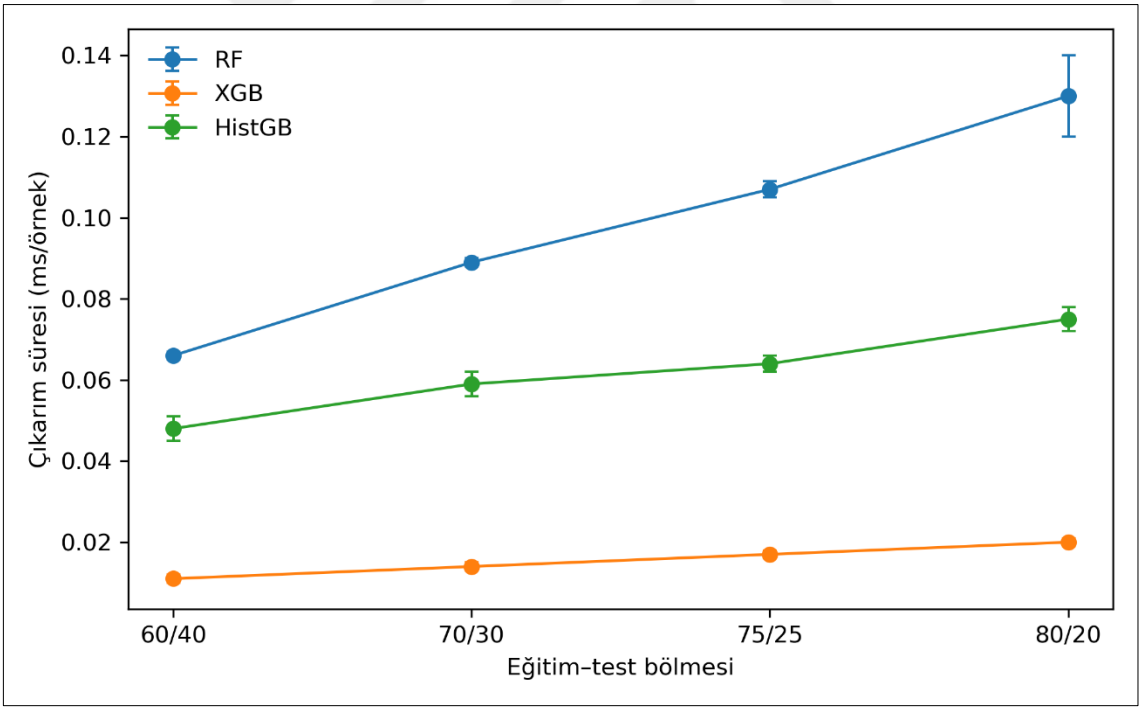


Şekil 5.3. Makro-F1'in eğitim-test oranına göre değişimi.

Şekil 5.4, Şekil 5.5 ve Çizelge 4.6 incelendiğinde dört eğitim-test bölmesi boyunca Makro-F1 değerlerinin dar bir bantta seyretmektedir. XGBoost 0.914-0.923, RF 0.916-0.919 ve HistGB 0.914-0.918 aralıklarında değerler almıştır. Bölmeler arası dalgalanma XGBoost 0.009, RF 0.003 ve HistGB için 0.004' tür. Bu bulgular, tohum tekrarı altında XGBoost' nin yüksek Makro-F1 ve düşük gecikme ile öne çıktığını, RF' nin eğitimde en hızlı olduğunu, HistGB' nin ise performansının kararlı fakat daha yavaş olduğunu gösterir.



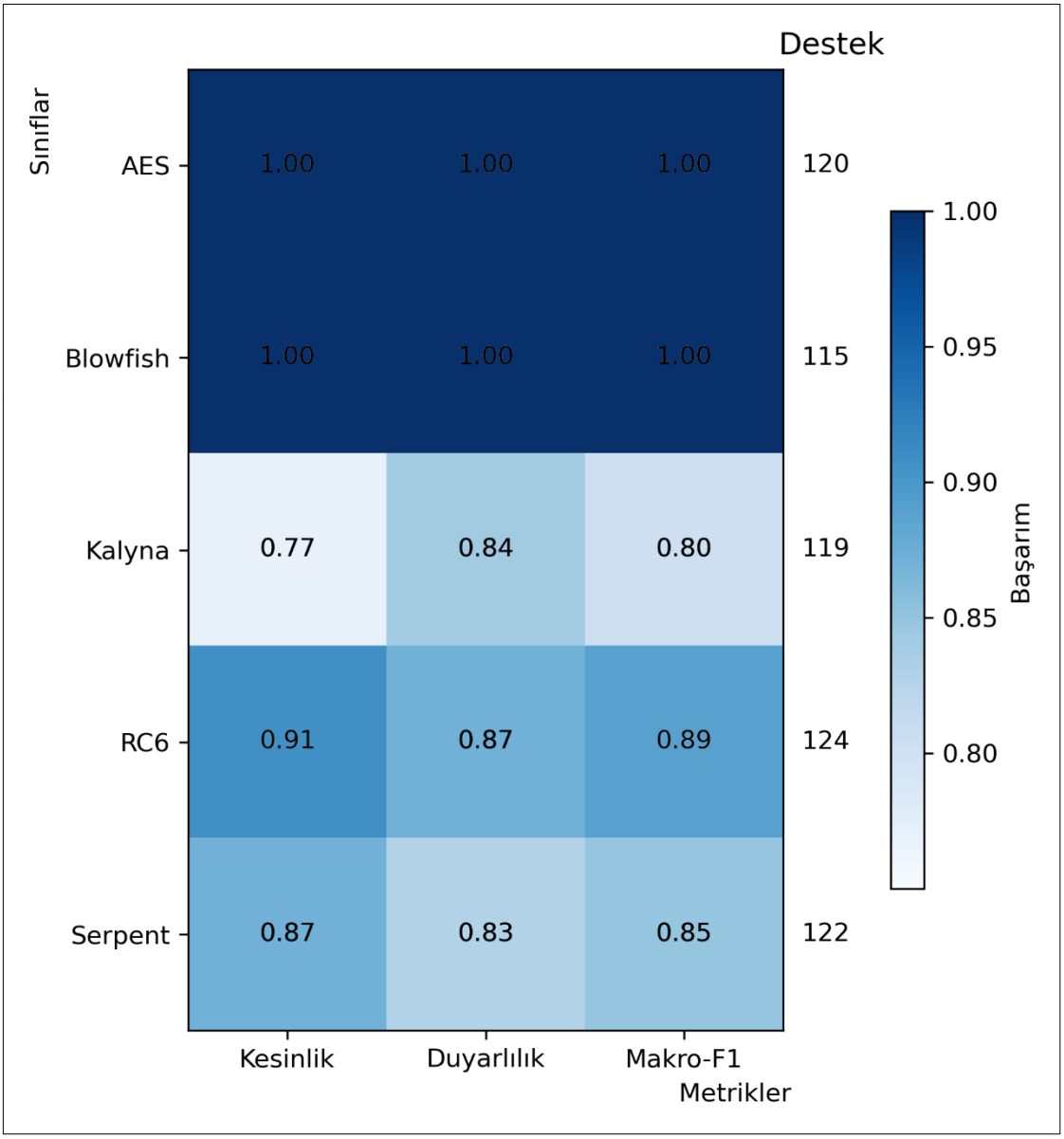
Şekil 5.4. Eğitim süresi ortalama (10 tohum).



Şekil 5.5. Çıkarım süresi ortalama (10 tohum).

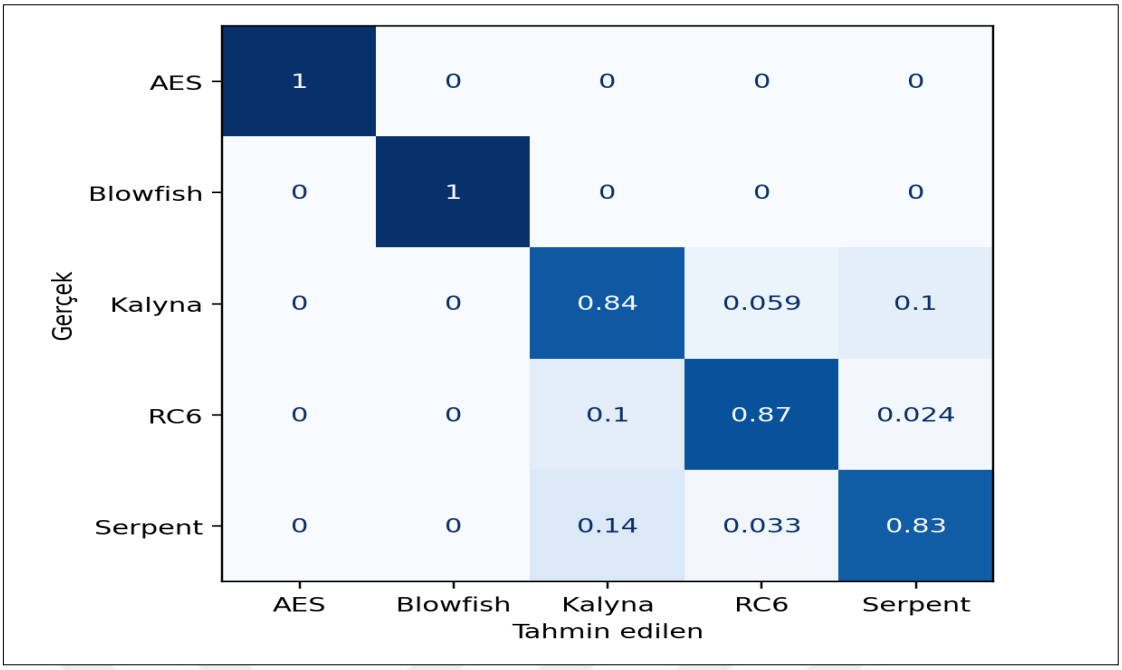
Şekil 5.6'da sınıf bazlı metrikler özetlenmiştir. AES ve Blowfish için kesinlik, duyarlılık ve Makro-F1 değeri 1.00 olup model bu iki sınıfta hatasızdır. Kalyna sınıfında duyarlılık 0.84 iken kesinlik 0.77'ye düşmekte, bu da yanlış-pozitiflerin göreli yüksekliğine işaret etmektedir; Makro-F1 0.80'dir. RC6 (F1=0.89) ve Serpent (F1=0.85) orta-yüksek başarı

düzeyindedir. Destek sütunu, sınıflar arası gözlem sayılarının benzer ölçeklerde olduğunu ve dengesizliğin bulguları çarpıtmadığını göstermektedir.



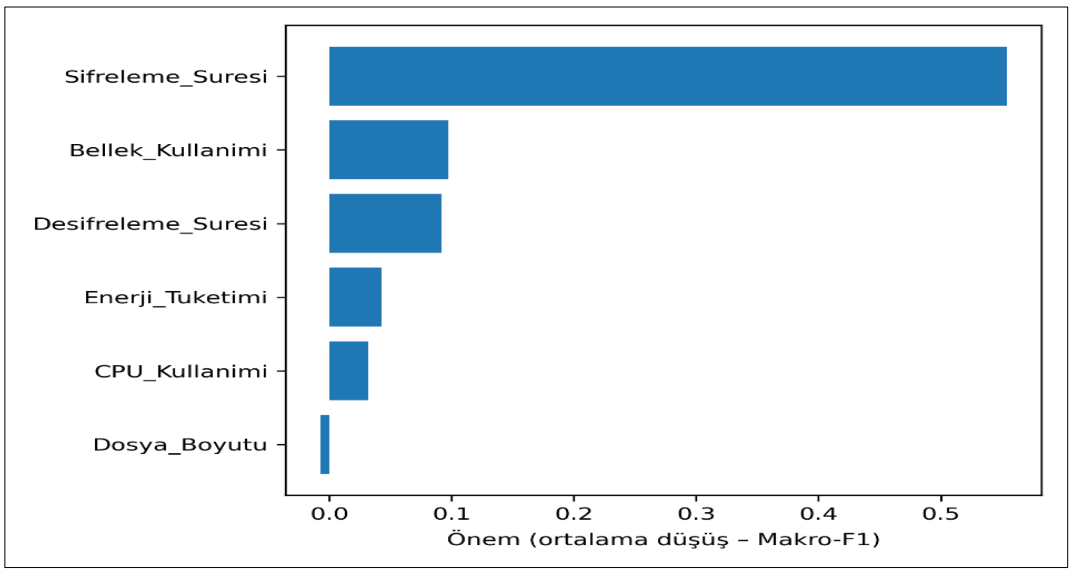
Şekil 5.6. Per-class ısı haritası- HistGB, 70/30.

Şekil 5.7, sınıflar arası karışmaların yönünü göstermektedir. Diyagonal değerler AES ve Blowfish için 1.00'dır. Kalyna örneklerinin %5,9'u RC6'ya, %10'u Serpent'e gitmektedir; Serpent örneklerinin %14'ü Kalyna' ya, %3,3'ü RC6'ya kaymaktadır. RC6 tarafında %10 Kalyna, %2,4 Serpent yönlü hatalar görülür. Bu desen, "Kalyna–Serpent–RC6" üçlüsü arasında özellik uzayında kısmi örtüşmeye işaret eder ve Kalyna için kesinliği düşüren ana kaynağın Serpent karışması olduğunu doğrular.



Şekil 5.7. Karışıklık matrisi- HistGB, 70/30.

Şekil 5.8'da özniteliklerin Makro-F1'de yol açtığı ortalama düşüşler yer almaktadır. Şifreleme süresi açık ara birincidir (~0.55 eşdeğeri etki), sınıflar arasında ayrıştırıcılığın ana kaynağıdır. Orta etki düzeyinde bellek kullanımı (~0.10) ve deşifreleme süresi (~0.09) gelir. Enerji tüketimi ve CPU kullanımı düşük katkıdadır (~0.02–0.03). Dosya boyutu'nun etkisi ihmal edilebilir düzeydedir. Bu profil, zaman ve bellek karakteristiklerinin model kararlarını belirlediğini ve Kalyna–Serpent–RC6 arasındaki karışmaların bu değişkenlerdeki benzerliklerle ilişkili olabileceğini düşündürür.



Şekil 5.8.Permütasyon temelli öznitelik önemi- HistGB, 70/30.

6. SONUÇ VE ÖNERİLER

Bu tezde, şifreleme algoritmalarının performans ve kaynak gereksinimleri temelinde dinamik seçimi için bir karar destek modeli geliştirilmiştir. Model, AES, Blowfish, RC6, Serpent ve Kalyna sınıfları arasında, şifreleme/deşifreleme süresi, CPU ve bellek kullanımı, enerji tüketimi ve dosya boyutu gibi ölçülebilir özniteliklerden yararlanarak çok-sınıflı tahmin yapmaktadır. Literatürde çoğu çalışma algoritmaları tek boyutlu veya donanımdan bağımsız ele alırken, bu çalışma çoklu metrik ve tekrarlanabilir doğrulama ile ayrışmaktadır.

Çalışmanın sonuçları değerlendirildiğinde taramada yedi modelden RF, XGBoost ve HistGB sürekli olarak üst bantta performans sergilemiş, kalan modeller belirgin biçimde geride kalmıştır. Ardından uygulanan nested CV (5×5), hiperparametre seçimini adil kılarak tarafsız performans kestirimi sunmuş ve üç adayın Makro-F1 sonuçlarının yakın bir aralıkta olduğunu göstermiştir. Eğitim-test deneylerinde RF 60/40'ta, HistGB 70/30'da XGBoost 75/25 ve 80/20'de en yüksek Makro-F1'i vermiştir. 10 tohum tekrarı altında ortalama eğitim süresi RF < XGB <XGB; ortalama çıkarım süresi XGB <XGB < RF şeklinde oluşmuştur. Sınıf düzeyinde AES ve Blowfish hatasızdır; Kalyna-Serpent-RC6 üçlüsü arasında kısmi örtüşme vardır. Bu desen, karışıklık matrisinde doğrulanmıştır. Permütasyon temelli önem profili, modelin kararında en yüksek etkiyi şifreleme süresinin sağladığını göstermiş. Bununla birlikte bellek kullanımı ve deşifreleme süresi orta etkide ve dosya boyutunun da düşük etkide kaldığını göstermiştir. Bu desen, kaynak kısıtlı ortamlarda zaman ve bellek maliyetlerinin belirleyici olduğu yönündeki bulgularla uyumludur.

Tezin sonuçları bazı önerileri sunmayı gerekmektedir. Çalışma içi seçim politikasında eğer gecikme kritikse XGB; hızlı yinelemeli eğitim gerekiyorsa RF; 70/30 benzeri dengeli kurulumlarda HistGB tercih edilebilir. Kalyna- Serpent ayrımını güçlendirmek için süre temelli özniteliklere ek olarak bellek erişim örüntüsü ve blok boyu gibi öznitelikler eklenebilir. Genelleme performansını arttırmak için veri kapsamı, farklı mesaj uzunlukları ve donanım profilleriyle genişletilerek yeniden eğitim yapılabilir. (4) Dağıtıma hazırlık: XGB için model kalibrasyonu ve olasılık eşiği ayarı; RF için ağaç sayısı-derinliğiyle çıkarım gecikmesi optimizasyonu raporlanmalı. Permütasyon önemine ek olarak SHAP özetleri eklenerek modelin açıklanabilirliği arttırılabilir.

Özetle, bu çalışma çoklu model ve çoklu doğrulama düzeniyle, literatürdeki tasarım bağımlı yaklaşımları genişleten ve tekrarlanabilir bir çerçeve sunmuştur. Sonuçlar; özellikle gerçek zaman kısıtları bulunan veya batarya odaklı sistemlerde, şifreleme algoritması seçiminin zaman ve bellek maliyetleri etrafında rasyonelleştirilebildiğini göstermektedir.

7. KAYNAKLAR

Al Jarrah, O. Y., Yoo, P. D., Muhaidat, S., Karagiannidis, G. K. ve Taha, K. (2015). Efficient Machine Learning for Big Data: A Review. *arXiv preprint, arXiv:1503.05296*. [Çevrim içi]. Erişim: https://arxiv.org/abs/1503.05296

Aumasson, J.-P. (2018). *Serious Cryptography: A Practical Introduction to Modern Encryption.* San Francisco: No Starch Press.

Breiman, L. (2001). Random Forest. *Machine Learning*, *45*(1), 5–32.

Chen, T. ve Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *KDD* '16.

Chinbat, T., Madanian, S., Airehrour, D. ve Hassandoust, F. (2024). Machine learning cryptography methods for IoT in healthcare. *BMC Medical Informatics and Decision Making*, *24*, 153.

Cortes, C. ve Vapnik, V. (1995). Support-vector networks, *Machine Learning*, 20(3), ss. 273–297.

Daemen, J. ve Rijmen, V. (2002). T*he Design of Rijndael: AES – The Advanced Encryption Standard*. Berlin: Springer.

Diffie, W. ve Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, *22*(6), ss. 644–654.

Gnatyuk, S., Prisyazhnuk, A., Nosov, A. ve Nosenko, K. (2023). Dataset of cryptographic algorithms for UAV image encryption based on artificial neural networks. *CEUR Workshop Proceedings: Classic, Quantum, and Post-Quantum Cryptography (CQPC 2023)*, *3504*, ss. 63–71.

Hastie, T., Tibshirani, R. ve Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2. baskı). New York: Springer.

Kahn, D. (1996). *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. New York: Scribner.

Katz, J. ve Lindell, Y. (2020). *Introduction to Modern Cryptography* (2. baskı). Boca Raton: CRC Press.

Kuznetsov, O., Zhuk, Y., Taran, V. ve Barabanov, M. (2024). Performance analysis of symmetric encryption algorithms for time-critical cybersecurity application. *Proceedings of the CPITS-II Workshop on Cybersecurity in Information and Telecommunication Systems*, *CEUR Workshop Proceedings*, 3700, 26–31.

Menezes, A. J., van Oorschot, P. C. ve Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. Boca Raton: CRC Press.

Mitchell, T. (1997). *Machine Learning.* New York: McGraw-Hill.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT Press.

Paar, C. ve Pelzl, J. (2010). *Understanding Cryptography: A Textbook for Students and Practitioners.* Heidelberg: Springer.

Panda, M. (2016). Performance analysis of encryption algorithms for security. *Proceedings of the SCOPES 2016 Conference*, ss. 278–284. https://doi.org/10.1109/SCOPES.2016.7955835

Radhakrishnan, I., Jadon, S. ve Honnavalli, P. B. (2024). Efficiency and security evaluation of lightweight cryptographic algorithms for resource-constrained IoT devices. *Sensors*, 24(3), Makale No. 1287

Saleh, M., Jhanjhi, N. Z., Abdullah, A. B. ve Saher, R. (2022). Message security level integration with IoTES: A design dependent encryption selection model for IoT devices. *International Journal of Computer Science and Network Security*, *22*(8), ss. 328–335.

Schneier, B. (1996*). Applied Cryptography: Protocols, Algorithms, and Source Code in C (2. baskı)*. New York: John Wiley & Sons.

Shafique, A., Mehmood, A., Alawida, M., Khan, A. N. ve Khan, A. U. R. (2022). A novel machine learning technique for selecting suitable image encryption algorithms for IoT applications. *Wireless Communications and Mobile Computing*, 2022, makale no. 5108331. https://doi.org/10.1155/2022/5108331

Singh, S. (1999). *The code book: The science of secrecy from ancient Egypt to quantum cryptography*. London: Fourth Estate.

Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice* (7. baskı). Boston: Pearson Education.

Sutton, R. S. ve Barto, A. G. (2020). *Reinforcement Learning: An Introduction* (2. baskı). Cambridge: The MIT Press.

Türk, S. ve Şamlı, R. (2020). Yapay sinir ağları ile klasik kriptografi algoritmalarının şifreli veriler üzerinden sınıflandırılması. *Uludağ Üniversitesi Mühendislik Fakültesi Dergisi*, 25(2), 651–664.

Zhou, Z.-H. (2012). *Ensemble Methods: Foundations and Algorithms.* Boca Raton: CRC Press.

8. EKLER

8.1. EK 1: JUPYTER ÇALIŞMA ORTAMI HAZIRLIK KODLARI

```
# %% Ortam ve klasör hazırlığı
import os, time, json, platform, glob, warnings, math
from pathlib import Path
import numpy as np
import pandas as pd

warnings.filterwarnings("ignore")

# === Kök klasör: runs_v3 ===
ROOT = Path("runs_v3")
ROOT.mkdir(exist_ok=True)

# Benzersiz koşu klasörü: run_YYYYMMDD_HHMMSS
def make_run_dir(root: Path):
    base = time.strftime("run_%Y%m%d_%H%M%S")
    d = root / base
    i = 1
    while d.exists():
        d = root / f"{base}_{i}"; i += 1
    for sub in ["nested_cv","holdout","robustness","data","report"]:
        (d/sub).mkdir(parents=True, exist_ok=True)
    return d
```

```
RUN_DIR = make_run_dir(ROOT)
print("Çalışma klasörü:", RUN_DIR.as_posix())

# Yardımcılar
def save_csv(df: pd.DataFrame, subdir: str, name: str):
    p = RUN_DIR / subdir / name
    p.parent.mkdir(parents=True, exist_ok=True)
    df.to_csv(p, index=False)
    print("✓ CSV yazıldı:", p.as_posix())
    return p

def pack_mean_std(df: pd.DataFrame, by_cols, metric_cols):
    g = df.groupby(by_cols, as_index=False)
    mean = g[metric_cols].mean().rename(columns={c: f"{c}_mean" for c in metric_cols})
    std = g[metric_cols].std(ddof=1).rename(columns={c: f"{c}_std" for c in metric_cols})
    return mean.merge(std, on=by_cols)

print("Yardımcılar hazır.")
```

8.2. EK 2: SENTETİK VERİ ÜRETİMİ KODLARI

```
# %% Veri: mevcut CSV'yi bul & yükle, etiket kodla
from sklearn.preprocessing import LabelEncoder

# 1) DF zaten bellekte mi?
if 'df' not in globals():
```

```
    candidates = []
    # Önce runs_v3 içerisindeki en güncel dataset'i ara
    candidates += sorted(glob.glob("runs_v3/run_*/data/*.csv"))
    # Eski çalışmalardan muhtemel adlar
    candidates += glob.glob("sentetik_sifreleme_*.csv")
    candidates += glob.glob("dataset_*.csv")
    candidates += glob.glob("*.csv")  # son çare: bulunduğu klasördeki csv'ler
    # Algoritma sütunu olan ilk dosyayı seç
    df = None
    for p in reversed(sorted(candidates)):
        try:
            tmp = pd.read_csv(p)
            if "Algoritma" in tmp.columns:
                df = tmp.copy()
                DATA_PATH = p
                break
        except Exception:
            pass
    if df is None:
        raise FileNotFoundError("Veri bulunamadı. Lütfen 'Algoritma' sütunu olan dataset CSV'yi klasöre koyun.")
else:
    DATA_PATH = "(bellekten)"

print("Veri kaynağı:", DATA_PATH)
print("Şekil:", df.shape)
```

```
print("Sütunlar:", list(df.columns))


# Hedef ve öznitelikler
X = df.drop(columns=["Algoritma"])
y = df["Algoritma"].to_numpy()
le = LabelEncoder().fit(y)
y_enc = le.transform(y)
n_classes = len(le.classes_)
print("Sınıflar:", dict(enumerate(le.classes_)))


# Raporlama için veri kopyasını koşu klasörüne kaydet
save_csv(df, "data", "dataset_used.csv")
```

8.3. EK 3: HİPERPARAMETRE OPTİMİZASYON KODLARI

```
# %% Modeller ve ızgaralar
from sklearn.ensemble import RandomForestClassifier, HistGradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.model_selection import StratifiedKFold, GridSearchCV, train_test_split


try:
    from xgboost import XGBClassifier
except Exception as e:
    raise RuntimeError("XGBoost bulunamadı. Anaconda Prompt: pip install xgboost==2.1.*") from e


SEED = 42
np.random.seed(SEED)
```

```
# Varsayılan konfigürasyonlar (nested dışında tek atışlarda kullanılacak)
RF_CFG = dict(
    n_estimators=200, max_depth=None, max_features="sqrt",
    min_samples_split=2, min_samples_leaf=1, criterion="gini",
    random_state=SEED, n_jobs=-1
)
XGB_CFG = dict(
    n_estimators=300, max_depth=6, learning_rate=0.1,
    subsample=0.8, colsample_bytree=0.8,
    objective="multi:softprob", num_class=n_classes, eval_metric="mlogloss",
    random_state=SEED, n_jobs=-1
)
HGB_CFG = dict(
    max_iter=300, learning_rate=0.1, max_depth=None,
    l2_regularization=0.0, max_leaf_nodes=31, random_state=SEED
)

# A priori hiperparametre ızgaraları (nested CV için)
RF_GRID = {
    "n_estimators": [160, 240, 320],
    "max_depth": [None, 12, 20],
    "max_features": ["sqrt"],
    "min_samples_split": [2, 4],
    "min_samples_leaf": [1, 2],
    "criterion": ["gini", "entropy"],
```

```
}
XGB_GRID = {
    "n_estimators": [200, 300, 400],
    "max_depth": [4, 6, 8],
    "learning_rate": [0.1, 0.05],
    "subsample": [0.8, 1.0],
    "colsample_bytree": [0.8, 1.0],
}
HGB_GRID = {
    "max_iter": [200, 300],
    "learning_rate": [0.1, 0.05],
    "max_depth": [None, 8],
    "max_leaf_nodes": [31, 63],
    "l2_regularization": [0.0, 0.1],
}

# Deney meta bilgisi (JSON)
meta = {
    "timestamp": time.strftime("%Y-%m-%d %H:%M:%S"),
    "python": platform.python_version(),
    "data_path": DATA_PATH,
    "n_samples": int(len(df)),
    "n_classes": int(n_classes),
    "label_mapping": {int(i): cls for i, cls in enumerate(le.classes_)},
    "grids": {"RF":RF_GRID, "XGB":XGB_GRID, "HistGB":HGB_GRID},
    "seed": SEED
```

```
}
(RUN_DIR/"report"/"run_metadata.json").write_text(json.dumps(meta,
ensure_ascii=False, indent=2), encoding="utf-8")
print("✓ run_metadata.json yazıldı:",
(RUN_DIR/"report"/"run_metadata.json").as_posix())
```

8.4. EK 4: TARAMA KODLARI

```
# === Screening (7 model, 5× RS-KFold) ===
import os, glob, time
from pathlib import Path
import numpy as np, pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

from sklearn.ensemble import RandomForestClassifier, HistGradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# -- Çalışma klasörü (mevcut RUN_DIR varsa onu kullan; yoksa runs_v3 içindeki en son
koşu) --
def latest_run(root="runs_v3"):
    r = Path(root)
    runs = sorted([d for d in r.glob("run_*") if d.is_dir()])
    return runs[-1] if runs else None
```

```
if "RUN_DIR" in globals() and Path(RUN_DIR).exists():
    RUN_DIR = Path(RUN_DIR)
else:
    RUN_DIR = latest_run() or (Path("runs_v3")/time.strftime("run_%Y%m%d_%H%M"))
    (RUN_DIR/"screening").mkdir(parents=True, exist_ok=True)

def save_csv_local(df, name):
    p = RUN_DIR/"screening"/name
    p.parent.mkdir(parents=True, exist_ok=True)
    df.to_csv(p, index=False); print("✓ CSV:", p.as_posix())
    return p

# -- Veri: en güncel dataset_used.csv'yi tercih et; bulunamazsa klasördeki CSV'lere bak --
cands = []
cands += glob.glob(str(RUN_DIR/"data"/"dataset_used.csv"))
cands += glob.glob("runs_v3/run_*/data/dataset_used.csv")
cands += glob.glob("runs/run_*/data/dataset_used.csv")
cands += glob.glob("*Synthetic*_Encryption*_Dataset*.csv") + glob.glob("*Encryption*_Dataset*.csv") + glob.glob("*.csv")

df = None
for p in sorted(cands):
    try:
        tmp = pd.read_csv(p)
```

```
        if "Algoritma" in tmp.columns:
            df = tmp.copy(); DATA_PATH = p
    except Exception:
        pass
if df is None:
    raise FileNotFoundError("Veri bulunamadı. 'Algoritma' sütunu olan bir CSV gerekli.")

# -- Etiket ve özellikler --
le = LabelEncoder().fit(df["Algoritma"])
y_enc = le.transform(df["Algoritma"])
X_df  = df.drop(columns=["Algoritma"]).reset_index(drop=True)
n_classes = len(le.classes_)
print("Veri:", DATA_PATH, "| Şekil:", df.shape, "| Sınıflar:", dict(enumerate(le.classes_)))

# -- Modeller --
try:
    from xgboost import XGBClassifier
except Exception as e:
    raise RuntimeError("XGBoost eksik: Anaconda Prompt'ta `pip install xgboost==2.1.*`") from e

MODELS = [
    ("RF", RandomForestClassifier(n_estimators=200, max_features="sqrt", random_state=42, n_jobs=-1)),
    ("XGB", XGBClassifier(n_estimators=300, max_depth=6, learning_rate=0.1,
                          subsample=0.8, colsample_bytree=0.8, objective="multi:softprob",
```

```
                num_class=n_classes, eval_metric="mlogloss", random_state=42, n_jobs=-1)),
    ("HistGB", HistGradientBoostingClassifier(max_iter=300, learning_rate=0.1, random_state=42)),
    ("SVM", SVC(kernel="rbf", C=10, gamma="scale", probability=False, random_state=42)),
    ("DT", DecisionTreeClassifier(max_depth=None, random_state=42)),
    ("LR", LogisticRegression(max_iter=2000, n_jobs=-1)),
    ("KNN", KNeighborsClassifier(n_neighbors=7, weights="distance")),
]

# -- 5× tekrarlı stratified k-fold screening --
rskf = RepeatedStratifiedKFold(n_splits=5, n_repeats=5, random_state=42)
rows = []
for name, est in MODELS:
    for tr, te in rskf.split(X_df, y_enc):
        est.fit(X_df.iloc[tr], y_enc[tr])
        yp = est.predict(X_df.iloc[te])
        acc = accuracy_score(y_enc[te], yp)
        p, r, f1, _ = precision_recall_fscore_support(y_enc[te], yp, average="macro", zero_division=0)
        rows.append({"Model":name, "Accuracy":acc, "Precision":p, "Recall":r, "F1":f1})

screen_raw = pd.DataFrame(rows)
screen_sum = screen_raw.groupby("Model").agg(['mean','std']).round(3)

def pack(col):
```

```
    return (screen_sum[(col,'mean')].map(lambda v: f"{v:.3f}") + " ± " +
            screen_sum[(col,'std')].map(lambda v: f"{v:.3f}"))

out = pd.DataFrame({
    "Model": screen_sum.index,
    "Accuracy": pack("Accuracy"),
    "Precision": pack("Precision"),
    "Recall":    pack("Recall"),
    "F1":        pack("F1"),
}).reset_index(drop=True)

save_csv_local(screen_raw, "screening_5xRS_raw.csv")
save_csv_local(out,        "screening_5xRS_summary.csv")
display(out.sort_values("F1", ascending=False))
print("Kayıt klasörü:", RUN_DIR.as_posix())
```

8.5. EK 5: İÇ İÇE ÇAPRAZ DOĞRULAMA KODLARI

```
# %% Nested CV (5×5)
outer = StratifiedKFold(n_splits=5, shuffle=True, random_state=SEED)
inner = StratifiedKFold(n_splits=5, shuffle=True, random_state=SEED)

def nested_one(name, est, grid):
    rows = []
    for k, (tr, te) in enumerate(outer.split(X, y_enc), start=1):
        gs = GridSearchCV(est, grid, scoring="f1_macro", cv=inner, n_jobs=-1, refit=True)
        gs.fit(X.iloc[tr], y_enc[tr])
```

```
        best = gs.best_estimator_
        yp = best.predict(X.iloc[te])
        acc = accuracy_score(y_enc[te], yp)
        p, r, f1, _ = precision_recall_fscore_support(y_enc[te], yp, average="macro", zero_division=0)
        rows.append({"Fold":k, "Model":name, "Accuracy":acc, "Precision":p, "Recall":r, "F1":f1})

    folds = pd.DataFrame(rows).round(3)
    save_csv(folds, "nested_cv", f"nested_{name}_folds.csv")

    # >>> SADECE SAYISAL METRİKLERİ ÖZETLE <<<
    metric_cols = ["Accuracy","Precision","Recall","F1"]
    summ = folds[metric_cols].agg(['mean','std']).round(3)

    save_csv(summ.reset_index(), "nested_cv", f"nested_{name}_summary.csv")
    return summ

summ_RF  = nested_one("RF",    RandomForestClassifier(**RF_CFG),  RF_GRID)
summ_XGB = nested_one("XGB",   XGBClassifier(**XGB_CFG),          XGB_GRID)
summ_HGB = nested_one("HistGB", HistGradientBoostingClassifier(**HGB_CFG), HGB_GRID)

def ms(s, m): return f"{s.loc['mean',m]:.3f} ± {s.loc['std',m]:.3f}"
final_nested = pd.DataFrame([
    {"Model":"RF",   "Accuracy":ms(summ_RF,"Accuracy"),
"Precision":ms(summ_RF,"Precision"),
```

```
    "Recall":ms(summ_RF,"Recall"),  "F1":ms(summ_RF,"F1")},
   {"Model":"XGB",                  "Accuracy":ms(summ_XGB,"Accuracy"),
"Precision":ms(summ_XGB,"Precision"),
    "Recall":ms(summ_XGB,"Recall"), "F1":ms(summ_XGB,"F1")},
   {"Model":"HistGB",               "Accuracy":ms(summ_HGB,"Accuracy"),
"Precision":ms(summ_HGB,"Precision"),
    "Recall":ms(summ_HGB,"Recall"), "F1":ms(summ_HGB,"F1")},
])
save_csv(final_nested, "nested_cv", "final_nested_comparison_mean_std.csv")
display(final_nested)
```

8.6. EK 6: EĞİTİM-TEST AYRIMI KODLARI

```
# %% Hold-out çoklu bölmeler
from time import perf_counter as T

def holdout_once(model, test_size, seed):
   Xtr, Xte, ytr, yte = train_test_split(X, y_enc, test_size=test_size,
                                         random_state=seed, stratify=y_enc)
   t0 = T(); model.fit(Xtr, ytr); fit_s = T()-t0
   t1 = T(); yp = model.predict(Xte); pred_ms = (T()-t1)/len(Xte)*1e3
   acc = accuracy_score(yte, yp)
   p, r, f1, _ = precision_recall_fscore_support(yte, yp, average="macro",
zero_division=0)
   return acc, p, r, f1, fit_s, pred_ms

rows = []
splits = [0.20, 0.25, 0.30, 0.40]  # 80/20, 75/25, 70/30, 60/40
```

```
for ts in splits:
    for name, mdl in [
        ("RF",     RandomForestClassifier(**RF_CFG)),
        ("XGB",    XGBClassifier(**XGB_CFG)),
        ("HistGB", HistGradientBoostingClassifier(**HGB_CFG)),
    ]:
        a,p,r,f,fit_s,ms = holdout_once(mdl, ts, SEED)
        rows.append({"Model":name, "Bolme":f"{int((1-ts)*100)}/{int(ts*100)}",
                     "Accuracy":a, "Precision":p, "Recall":r, "F1":f,
                     "fit_time_s":fit_s, "predict_time_ms":ms})

hold = pd.DataFrame(rows).round(3)
save_csv(hold, "holdout", "holdout_all_splits.csv")
display(hold)
```

8.7. EK 7: GÜRBÜZLÜK KODLARI

```
# %% Hücre 5 — Gürbüzlük
N_SEEDS = 10
rows = []
splits = [0.20, 0.25, 0.30, 0.40]

for seed in range(SEED, SEED+N_SEEDS):
    rf  = RandomForestClassifier(**{**RF_CFG,  "random_state": seed})
    xgb = XGBClassifier(**{**XGB_CFG,          "random_state": seed})
    hgb = HistGradientBoostingClassifier(**{**HGB_CFG,      "random_state": seed})
    for ts in splits:
```

```
        for name, mdl in [("RF", rf), ("XGB", xgb), ("HistGB", hgb)]:
            a,p,r,f,fit_s,ms = holdout_once(mdl, ts, seed)
            rows.append({"Model":name, "Bolme":f"{int((1-ts)*100)}/{int(ts*100)}",
                         "Seed":seed, "Accuracy":a, "Precision":p, "Recall":r, "F1":f,
                         "fit_time_s":fit_s, "predict_time_ms":ms})

robust_raw = pd.DataFrame(rows).round(3)
save_csv(robust_raw, "robustness", "robust_all_models_raw_10seeds.csv")

robust_sum = pack_mean_std(
    robust_raw,
    by_cols=["Model","Bolme"],
    metric_cols=["Accuracy","Precision","Recall","F1","fit_time_s","predict_time_ms"]
).round(3)
save_csv(robust_sum, "robustness", "robust_all_models_summary_10seeds.csv")

display(robust_sum.sort_values(["Bolme","Model"]))
```

8.8. EK 8: PER-CLASS ISI HARİTASI- HİSTGB, 70/30 KODLARI

```
# %% Per-class ısı haritası
import numpy as np, pandas as pd, matplotlib.pyplot as plt, glob
from pathlib import Path
from mpl_toolkits.axes_grid1 import make_axes_locatable
from sklearn.metrics import classification_report
from matplotlib import colors
```

```
# 1) Son koşuyu ve klasörleri bul
BASE = Path("runs_v3")
RUNS = sorted([d for d in BASE.glob("run_*") if d.is_dir()])
assert RUNS, "runs_v3 içinde koşu bulunamadı."
RUN_DIR = RUNS[-1]
REPORT = RUN_DIR / "report"
FIGS   = RUN_DIR / "figs"; FIGS.mkdir(parents=True, exist_ok=True)

# 2) Per-class CSV varsa onu kullan, yoksa y_te/y_pr'dan üret
cands = sorted(REPORT.glob("per_class_report_*_7030.csv"))
metric_cols = ["precision","recall","f1-score"]

if cands:
    f = cands[-1]
    rep_df = pd.read_csv(f, index_col=0)
    class_rows = [i for i in rep_df.index if i not in ("accuracy","macro avg","weighted avg")]
    Z = rep_df.loc[class_rows, metric_cols].to_numpy()
    support = rep_df.loc[class_rows, "support"].astype(int).to_numpy()
    class_names = class_rows
    model_tag = f.stem.replace("per_class_report_","").replace("_7030","")
else:
    assert 'y_te' in globals() and 'y_pr' in globals(), "y_te/y_pr yok; önce 70/30 tahminlerini üretin veya CSV'yi kaydedin."
    assert 'classes' in globals(), "classes yok."
    rep = classification_report(y_te, y_pr, target_names=classes, output_dict=True, zero_division=0)
```

```
    class_names = list(classes)
    Z = np.array([[rep[c][m] for m in metric_cols] for c in class_names])
    support = np.array([int(rep[c]["support"]) for c in class_names])
    model_tag = "model"

# 3) Isı haritası
fig, ax = plt.subplots(figsize=(5.8, 6.2), dpi=300)
cmap = plt.get_cmap("Blues")                # mavi tonları
norm = colors.Normalize(vmin=0.75, vmax=1.0) # 0.75–1.00 aralığına odaklan
im = ax.imshow(Z, cmap=cmap, norm=norm, aspect="auto", interpolation="nearest")
cbar = fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04);
cbar.set_label("Başarım")
cbar.set_ticks([0.80, 0.85, 0.90, 0.95, 1.00])

# Hücre içi metinlerde kontrast
thr = 0.875
for i in range(Z.shape[0]):
    for j in range(Z.shape[1]):
        ax.text(j, i, f"{Z[i,j]:.2f}", ha="center", va="center",
                color=("white" if Z[i,j] >= thr else "black"))

ax.set_xlabel("                                    Metrikler")
ax.set_ylabel("                                                Sınıflar")
ax.set_xticks(range(3)); ax.set_xticklabels(["Kesinlik","Duyarlılık","Makro-F1"])
ax.set_yticks(range(len(class_names))); ax.set_yticklabels(class_names)
```

```
# Hücre değerleri
for i in range(Z.shape[0]):
    for j in range(Z.shape[1]):
        ax.text(j, i, f"{Z[i,j]:.2f}", ha="center", va="center")

# Sağda “Destek” sütunu
divider = make_axes_locatable(ax)
ax_sup = divider.append_axes("right", size="10%", pad=0.10)
ax_sup.set_ylim(ax.get_ylim()); ax_sup.axis("off"); ax_sup.set_title("Destek", pad=6)
for i, s in enumerate(support):
    ax_sup.text(0, i, str(s), va="center", ha="left")

fig.tight_layout()
out = FIGS / f"per_class_heatmap_{model_tag}_7030.png"
fig.savefig(out, dpi=300)
print("✓ PNG:", out.as_posix())
```

8.9. EK 9: KARIŞIKLIK MATRİSİ KODLARI

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt, numpy as np, joblib

# Önceki hücrede eğitilmiş est, y_te, y_pr, classes zaten hazırsa bu bloğu doğrudan çalıştır.
cm = confusion_matrix(y_te, y_pr, normalize="true")
disp = ConfusionMatrixDisplay(cm, display_labels=classes)
fig, ax = plt.subplots(figsize=(5.6,4.6))
```

```
disp.plot(ax=ax, cmap="Blues", colorbar=False)
ax.set_xlabel("Tahmin edilen"); ax.set_ylabel("Gerçek")  # başlık yok
fig.tight_layout();    fig.savefig(FIGS/f"confmat_norm_{best_model_name}_7030.png", dpi=220)
```

8.10. EK 10: PERMÜTASYON TEMELLİ ÖZNİTELİK ÖNEMİ- HİSTGB, 70/30 KODLARI

```
# === Permutation Importance (70/30) ===
from sklearn.inspection import permutation_importance

r = permutation_importance(est, X_te, y_te, n_repeats=20, random_state=SEED, scoring="f1_macro")
imp = pd.DataFrame({"feature": feature_cols, "importance": r.importances_mean})
imp = imp.sort_values("importance", ascending=False).reset_index(drop=True)
imp.to_csv(REPORT / f"perm_importance_{best_model_name}_7030.csv", index=False)
print("✓ CSV:", REPORT / f"perm_importance_{best_model_name}_7030.csv")

# Grafik
fig, ax = plt.subplots(figsize=(6, 4.6))
ax.barh(imp["feature"][::-1], imp["importance"][::-1])
ax.set_xlabel("Önem (ortalama düşüş – Makro-F1)")
#ax.set_title(f"Permutation Importance • {best_model_name} • 70/30")
fig.tight_layout()
fig.savefig(FIGS / f"perm_importance_{best_model_name}_7030.png", dpi=300)
plt.show()
print("✓ PNG:", FIGS / f"perm_importance_{best_model_name}_7030.png")
```

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Ferdi AZBOY

Yabancı Dili : İngilizce

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Y. Lisans	Bilgisayar Müh.	Düzce Üniversitesi	2025
Lisans	Bilgisayar Müh.	Kocaeli Üniversitesi	2014
Lise	Fen Bilimleri	Hıdır Sever Lisesi	2007

TEZDEN ÇIKAN YAYIN

Azboy, F. ve Şatır, E. (2025). Makine öğrenmesi tabanlı şifreleme algoritması seçimi için kriptografik algoritmalarının çok kriterli karşılaştırması. İçinde: Sallı Bideci, Ö. (Ed.), Mühendislik Alanında Akademik Araştırmalar. Ankara: Platanus Publishing, ss. 97–107.

TEZ DIŞI YAYIN

Şatır, E., Azboy, F., Aydın, A., Arslan, H. ve Hacıefendioğlu, Ş. (2016). Veri indirgeme ve sınıflandırma teknikleri ile glokom hastalığı teşhisi, *ECJSE*, 3(3).