

**İstanbul Medeniyet Üniversitesi**

**Bilgisayar Mühendisliği Bölümü**

**Örüntü Tanıma Projesi**

**Grup No: 1**

**İsim 1:** Ayşenur Yörür

**İsim 2:** Kaan Berk Duman

**Veri Kümeler :** C2 ve R1

**k-Fold :** 2,5

**Algoritmalar :** Random Forest ve Xgboost (Sınıflandırma); Ridge Regression ve LSTM (regresyon)

**Teslim Tarihi :** 11/01/2025

**Ocak 2025**

**1. GİRİŞ**

**1.1 Veri Kümelerinin Tanıtımı**

**1.1.1. Classification**

Colab Linkleri:

https://colab.research.google.com/drive/1JxFkU5GYWvkpEBktrDLfhucxGCqUCFXM?usp=sharing

https://colab.research.google.com/drive/1z2iW3Br4HUsmGiYAXT227NQM81Q56\_rZ?usp=sharing

Veri setimizin adı Dry Bean Dataset’tir. Veri setimizde 7 farklı kayıtlı bean türüne ait 13.611 tane tanelerin görüntüleri yüksek çözünürlüklü bir kamera ile çekilmiştir. Toplamda 16 özellik, 12 boyut ve 4 şekil formu bu tanelerden elde edilmiştir.

Veri Türü: Çok Değişkenli

Öznitelik Türü: Categorical Integer Real

Biçim Türü: Matris

Veri kümeniz eksik değerler içeriyor mu? Hayır

Örnek Sayısı (veri kümenizdeki kayıtlar): 13611

Öznitelik Sayısı (her kayıttaki alanlar): 16

Bu araştırmada, piyasa koşulları göz önünde bulundurularak form, şekil, tür ve yapı gibi özellikler dikkate alınarak yedi farklı bean türü kullanılmıştır. Benzer özelliklere sahip yedi farklı kayıtlı bean türünü ayırt etmek ve homojen tohum sınıflandırması elde etmek amacıyla bir bilgisayarlı görü sistemi geliştirilmiştir. Sınıflandırma modeli için, yedi farklı bean türüne ait 13.611 tane tanelerin görüntüleri yüksek çözünürlüklü bir kamera ile çekilmiştir. Bilgisayarlı görü sistemi ile elde edilen bean görüntüleri segmentasyon ve özellik çıkarma aşamalarına tabi tutulmuş ve toplamda 16 özellik; 12 boyut ve 4 şekil formu tanelerden elde edilmiştir.

Attribute Information:

1. Area (A): Alan Bir fasulye bölgesinin alanı ve sınırları içindeki piksel sayısı.
2. Perimeter (P): Çevre Fasulyenin çevresi, sınırının uzunluğu olarak tanımlanır.
3. Major axis length (L): Büyük eksen uzunluğu (L): Bir fasulyeden çizilebilecek en uzun çizginin uçları arasındaki mesafe.
4. Minor axis length (l): Küçük eksen uzunluğu (l): Ana eksene dik dururken fasulyeden çizilebilecek en uzun çizgi.
5. Aspect ratio (K): En-boy oranı (K): L ve l arasındaki ilişkiyi tanımlar.
6. Eccentricity (Ec): Eksantriklik (Ec): Aynı momentlere sahip elipsin eksantrikliği.
7. Convex area (C): Konveks alan (C): Fasulye tohumunun alanını içerebilecek en küçük konveks çokgen içindeki piksel sayısı.
8. Equivalent diameter (Ed): Eşdeğer çap (Ed): Alanı bir fasulye tohumu alanı ile aynı olan bir çemberin çapı.
9. Extent (Ex): Kapsam (Ex): Sınır kutusundaki piksellerin fasulye alanına oranı.
10. Solidity (S): Yoğunluk (S): Konvekslik olarak da bilinir. Konveks kabukta bulunan piksellerin fasulyelerde bulunanlara oranı.
11. Roundness (R): Roundness (R): Yuvarlaklık (R): Aşağıdaki formülle hesaplanır: (4πA) / (P²).
12. Compactness (CO): Kompaktlık (CO): Bir nesnenin yuvarlaklığını ölçer: Ed / L.
13. ShapeFactor1 (SF1) Şekil Faktörü 1 (SF1)
14. ShapeFactor2 Şekil Faktörü 2 (SF2)
15. ShapeFactor3 Şekil Faktörü 3 (SF3)
16. ShapeFactor4 Şekil Faktörü 4 (SF4)
17. Class Sınıf: (Şeker, Barbunya, Bombay, Çalı, Dermosan, Horoz ve Sıra).

ekran görüntüsü, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, diyagram, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

eri setimiz 7 sınıflı (multi-class) ve dengesiz bir yapıya sahiptir. Özellikle **Bombay** sınıfına ait verilerin oldukça az olduğu görülmektedir. Bu durum, modeli eğitirken sınıflar arasında dengesizlik kaynaklı performans farklılıklarına yol açabilir.

Ayrıca, veri setinde toplam **68 adet duplicate (tekrarlı)** veri tespit edilmiştir. Bu, aynı özelliklere sahip 68 satırın veri setinde yer aldığı anlamına gelir. Bu durum, eğitim sırasında modelin aynı bilgiyi birden fazla kez öğrenmesine neden olabileceği için potansiyel bir sorun teşkil edebilir.

Veri setindeki özellikler (feature'lar) incelendiğinde, aralarında oldukça yüksek bir korelasyon olduğu görülmektedir. Özellikle birbirinden bağımsız olması beklenen bazı özelliklerin dahi yüksek korelasyon göstermesi, **multicollinearity (çoklu doğrusal bağıntı)** problemine işaret etmektedir.

Boxplot analizleri incelendiğinde, **Bombay bean** sınıfının bazı özellikler (features) açısından diğer sınıflardan belirgin şekilde ayrıldığı gözlemlenmiştir. Özellikle **Area, Perimeter, MajorAxisLength** ve **MinorAxisLength** gibi özellikler üzerinde yapılan incelemelerde, Bombay bean'in boxplotlarının diğer sınıflardan oldukça farklı bir konumda olduğu açıkça görülmektedir.

Bu durum, Bombay bean sınıfının veri setinde diğer sınıflardan ayrılabilmesini kolaylaştırabilir. Dolayısıyla, sınıf dengesizliğine rağmen modelin Bombay bean'i doğru bir şekilde sınıflandırmakta zorluk çekmeyeceği tahmin edilmektedir. Ancak, bu sınıfın diğer sınıflardan bu kadar farklı olması, model performansını artırıcı bir faktör olarak değerlendirilebilir.

metin, ekran görüntüsü, diyagram, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, diyagram, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, diyagram, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, diyagram, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, diyagram, çizgi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, diyagram, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, diyagram, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, diyagram, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu diyagram, metin, plan, dikdörtgen içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, diyagram, ekran görüntüsü, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, diyagram, ekran görüntüsü, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, diyagram, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, diyagram, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, diyagram, çizgi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, diyagram, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, diyagram, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu

**1.1.2. Regresyon**

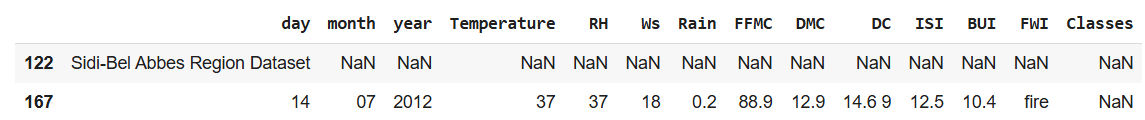
Cezayir Orman Yangınları Veri Seti: Bu veri seti, Cezayir'in iki bölgesinden (kuzeydoğuda bulunan Bejaia bölgesi ve kuzeybatıda bulunan Sidi Bel-abbes bölgesi) alınan 244 gözlemi içermektedir. Her bir bölge için 122 adet gözlem bulunmaktadır.

Veri seti, Haziran 2012 ile Eylül 2012 tarihleri arasındaki dönemi kapsamaktadır. Toplamda 11 bağımsız değişken ve 1 bağımlı değişken (sınıf) içermektedir. Gözlemler, iki sınıfa ayrılmıştır: yangın olan (138 gözlem) ve yangın olmayan (106 gözlem).

Veri Seti Sütunları:

* Tarih: (GG/AA/YYYY) Gün, ay (Haziran'dan Eylül'e kadar) ve yıl (2012). Hava durumu gözlemlerini içerir.
* Sıcaklık (Temp): Öğle saatindeki maksimum sıcaklık (Celsius derece): 22 ile 42 arasında.
* Bağıl Nem (RH): % olarak bağıl nem: 21 ile 90 arasında.
* Rüzgar Hızı (Ws): Rüzgar hızı (km/saat): 6 ile 29 arasında.
* Yağış (Rain): Günlük toplam yağış miktarı (mm): 0 ile 16.8 arasında.
* İnce Yakıt Nem İndeksi (FFMC): FWI (Yangın Hava İndeksi) sistemine ait bir bileşen: 28.6 ile 92.5 arasında.
* Karışık Yakıt Nem İndeksi (DMC): FWI sistemine ait bir bileşen: 1.1 ile 65.9 arasında.
* Kuraklık Kodu (DC): FWI sistemine ait bir bileşen: 7 ile 220.4 arasında.
* Başlangıç Yayılma İndeksi (ISI): FWI sistemine ait bir bileşen: 0 ile 18.5 arasında.
* Birikim İndeksi (BUI): FWI sistemine ait bir bileşen: 1.1 ile 68 arasında.
* Yangın Hava İndeksi (FWI): FWI sistemi genel indeksi: 0 ile 31.1 arasında.
* Sınıflar (Classes): İki sınıf bulunmaktadır: "Yangın" (Fire) ve "Yangın Yok" (Not Fire).

Bu veri seti, yangın tahmini yapmak (Classes) için hava durumu ve yangın hava indeksi (FWI) bileşenlerini analiz etmeyi kullanılabilmektedir. Bizim amacımız labelımız olan FWI hava indeksini tahmin etmek olacaktır.



Bu işlemi, veri setini iki farklı bölgeye ("Bejaia Bölgesi" ve "Sidi-Bel Abbes Bölgesi") ayırabilmek için yaptık. Veri setindeki 122. indexten itibaren kayıtların bölgelere göre farklılık gösterdiğini bildiğimizden, her bir bölgeye ait kayıtları belirlemek amacıyla yeni bir "Region" sütunu ekledik. Bu sütun, "Bejaia Region Dataset" için 0, "Sidi-Bel Abbes Region Dataset" için 1 olarak atanmıştır. Bu sayede, analiz ve modelleme sırasında bölgeler arası ayrım yapılması kolaylaştırılmıştır.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Datada 122. satırda bir object türünde nesneler olduğu için veri türleri obje olarak gözükmektedir. Aşağıda görebileceğiniz üzere dolayısı ile biz 122. satırı da kaldıracağız. Datasetimizde null ve olması gerekenden farklı türde veriler olmadığında hata almadan astype fonksiyonunu kullanarak türlerini değiştirebiliriz. Sütun isimlerinde gereksiz boşluklar da var, bu isimleri düzelteceğiz.

* Örneğin " RH" -> "RH" olmalı.

Preprocess edip temizlenen düzenlenen verimizi github'a yükledikten sonra github üzerinden tekrar çektik.

metin, diyagram, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, diyagram, daire içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, paralel, sayı, numara, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, sayı, numara, paralel, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, diyagram, sayı, numara, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü, çizgi, diyagram, öykü gelişim çizgisi; kumpas; grafiğini çıkarma içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bölgesel olarak farklılıklar görebiliyoruz. Özellikle 7. ve 9. ayda farklılıklar oldukça belirgin durumdadır. 6. ayda çok benzer bir grafik izlemiştir.

metin, çizgi, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, çizgi, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu grafik aslında bize ilginç bir fikir verdi. O gün içerisinde yangın olması bir önceki gün yangın yoksa genel olarak o günün FWI değerini oldukça artırmıştır. Hatta bir önceki değerler bir sonraki değerler ile bir örüntü içerisinde gözüküyor şeklinde de düşünebiliriz. Bu amaçla çerçeveleme yöntemi kullanılabilir ilerleyen aşamalarda.

**PreProcess:** Shuffle =False yapıyoruz böylece ilk %80 lik kısım ile son %20 lik kısmı ayırmış oluyoruz bu şekilde sıra bilgisini de kaybetmemiş oluyoruz. Daha sonra iki bölgeyi de böldükten sonra train setlerini uc uca birleştiriyoruz ve test setlerini de sıraya dikkat ederek uc uca birleştiriyoruz böylece train ve set datalarımı hazırlamış olduk.

Bir sonraki adımlarda scale ve PCA işlemlerini yapacağız data leakage olmasını istemediğiz için. Yani test setim hakkında herhangi bir bilgiye train datam sahip olsun istemediğiz için önce bölme işlemini yaptık.

metin, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, çizgi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, çizgi, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Windowing: Zaman serisi verilerinde, ardışık veri noktalarının birbirleriyle ilişkili olması, modelin bu bağımlılıkları öğrenmesini önemli kılmaktadır. Bu nedenle, veri setine **windowing (kaydırmalı pencere)** yöntemi uyguladım. Bu teknik, veriyi belirli bir pencere boyutunda kesitlere ayırarak, her bir pencereyi ilgili bir hedef değerle eşleştirme imkanı sunar. Böylece model, zaman serisinin dinamik yapısını daha iyi öğrenebilir hale gelmektedir.

**Pencere Boyutu (window\_size)**: Her bir pencerenin içerdiği zaman adımı sayısını belirlemek için kullanılmıştır. Bu çalışmada, geçmiş 3 zaman adımını dikkate almak amacıyla window\_size=3 olarak belirledim.

**Adım Boyutu (step\_size)**: Pencereler arasındaki kaydırma mesafesini ifade eder. Her bir zaman adımı için pencere oluşturmak amacıyla step\_size=1 kullanılmıştır.

Bu süreç, zaman serisi verilerinin doğasını koruyarak, hem eğitim hem de test aşamalarında modelin performansını optimize etmek için önemli bir adımdır diye düşünmekteyim.

**1.2 Kullanılan Algoritmaların Özellikleri**

**1.2.1. Classification**

Test verisinin temel amacı, modelin **gerçek dünyadaki performansını** doğru bir şekilde değerlendirmektir. Gerçek hayatta veri genellikle dengesiz olduğu için, test verisini olduğu gibi bırakmak, modelin dengesiz veri üzerinde nasıl bir performans gösterdiğini anlamak açısından kritik öneme sahiptir.

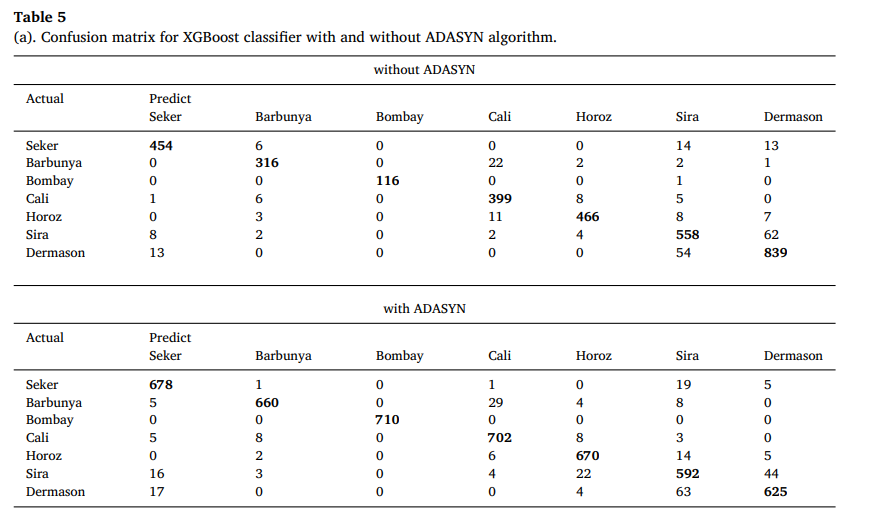
Ancak, bu veri setiyle yapılan diğer çalışmaları incelediğimde, test verisine yapılan manipülasyonların model performansı üzerinde yanıltıcı etkiler oluşturduğunu gözlemledim. Veri setinin tamamına **SMOTE**, **PCA** ve **outlier removal** gibi işlemlerin uygulandığı ve bu süreçlerin model başarılarını yapay olarak artırdığı görülmektedir.

Örneğin, [*Automated Classification of Dry Bean Varieties Using XGBoost and SVM Models*](https://arxiv.org/abs/2408.01244) adlı çalışmada, test verisinin manipüle edilmesiyle yüksek başarı elde edilmiştir. Ancak, bu işlemleri projede denediğimizde, test verisini manipüle etmeden aynı başarıyı tekrarlayamadık. Bu durum, modelin gerçek ayırt edebilme kapasitesinin yanlış değerlendirilmesine neden olabilir.

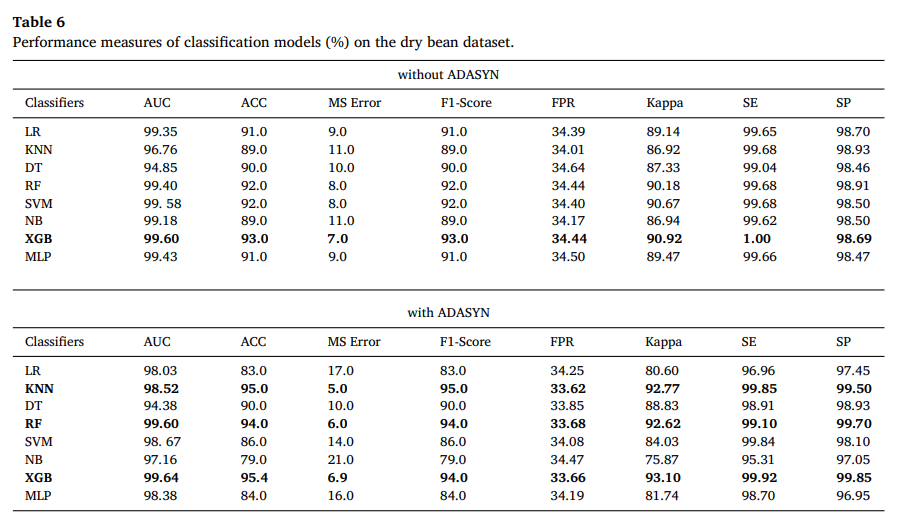
**SMOTE (Synthetic Minority Oversampling Technique)**, azınlık sınıflarını dengelemek için sentetik veri noktaları oluşturan bir yöntemdir. Ancak SMOTE, bizim problemimiz için uygun bir yöntem değildir. Örneğin, iki veri noktası olan A ve B alınarak (A+B)/2 şeklinde bir sentetik veri noktası oluşturulabilir. Bu işlem, A ve B noktalarını yaklaşık olarak 1.5 ağırlıklandırmaya eşdeğer bir sonuç doğurur. Ancak, bu yöntem her zaman tutarlı bir performans artışı sağlamaz. Özellikle *pairplot* analizleri incelendiğinde, SMOTE’un veri setimize uygun olmadığı açıkça görülmektedir. Bu durumun tartışıldığı bir [StackExchange konu başlığı](https://datascience.stackexchange.com/questions/106461/why-smote-is-not-used-in-prize-winning-kaggle-solutions) de mevcuttur.

Buna karşılık, **ADASYN (Adaptive Synthetic Sampling)**, azınlık sınıfındaki daha zor öğrenilen ve çoğunluk sınıfına yakın olan veri noktalarına ağırlık vererek bu bölgelerde daha fazla sentetik veri üretir. Bu yönüyle ADASYN, SMOTE’a kıyasla veri setimiz için daha uygun bir yöntem gibi görünmektedir. Ancak, bu yöntemin de yalnızca eğitim verisine uygulanması gerektiği, test verisinin manipüle edilmemesi gerektiği unutulmamalıdır.

[*Comparison of multiclass classification techniques using dry bean dataset*](https://www.sciencedirect.com/science/article/pii/S2666307423000013) adlı makalede, ADASYN kullanımının etkisi açıkça görülmektedir. Tablolar incelendiğinde, ADASYN kullanılmadan önce **Dermason** ve **Sira** türlerinin sıklıkla karıştırıldığı, ancak **Bombay** türünün ayırt edilmesinde herhangi bir sorun yaşanmadığı gözlemlenmiştir. Bombay fasulyesi, özellikleri bakımından diğer türlerden oldukça farklı olduğu için model tarafından zaten kolaylıkla ayırt edilebilmektedir. Bu nedenle, Bombay fasulyesi için oversampling yapmak veya weighted yöntemler kullanmak mantıklı bir seçenek olabilir.



Tabloya baktığımızda ADASYN kullanıldığında, Bombay fasulyesi için veri noktaları 116’dan 710’a çıkartılmış ve bu durum accuracy değerinin artmasına neden olmuştur. Ancak, bu artış gerçek model performansını yansıtmamaktadır. Özellikle **Dermason** ve **Sira** türleri hala karıştırılmaya devam etmekte, diğer türlerde de hatalar gözlemlenmektedir. Yani accuracy artışı, yalnızca veri manipülasyonundan kaynaklanan bir **ilüzyon** oluşturmaktadır.



En büyük problemlerden biri, test verisine de oversampling yapılması ve bu şekilde yanıltıcı bir başarı elde edilmesidir. Test verisinin manipüle edilmesi, modelin gerçek performansını değerlendirmeyi imkânsız hale getirir ve sonuçların doğruluğunu sorgulatır. Bu tür yöntemlerin projede test edilmesi planlanmakta, ancak test verisinin manipüle edilmediği durumlarda gerçek model başarısının daha iyi anlaşılabileceği vurgulanmaktadır.

Evaluation bölümü için X\_test ve y\_test dataları manipüle edilmeden testlerimizi gerçek dünyaya uygun yapacağız. Hiperparametrelerimizi X\_train ve y\_train üzerinde validation skorlarımızı ve hiperparametrelerimiz Kfold kullanarak yapacağız.

Model hakkında daha çok bilgi edinmek ve modeldeki varolan multicolinearity problemini azaltmak için PCA kullanacağız. Tanımlayıcı İstatistikler bölümününde verilerin pairplotlarında birbirleri ile feature'larımızın yüksek korelasyonlara sahip olduğunu gördük.

PCA modelini eğitirken (fit), yalnızca train veri setini kullanarak bileşenleri belirleriz. Daha sonra aynı model ile test setine sadece transform işlemi uygularız. Böylece her iki veri seti aynı PCA bileşenleri ile temsil edilmiş olur.

metin, ekran görüntüsü, diyagram, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Burada PC1 ve PC2 yi özniteliklerini kullanmak verimizin %88 olarak açıklanabilirliğini yansıtmaktadır. Yüksek boyuttaki verilerde bunları kullanarak bile yapılacak çalışmalar kullanılabilir. Lakin bizim problemimizde veri setimizin büyüklüğü sorun teşkil etmediği için ve **multicolinearity** den kurtulmak istediğimiz için %99.77 açıklanabilirlik ile ilk 7 Principal Component'i kullanacağız.

**Validation Function:** Veri setimizde data leakage'i önlemek ve herhangi bir sınıfın tek bir **veri setine toplanmasını engellemek** için **stratified=y** kullanıyoruz. Bu, her sınıfın oranını hem train hem de validation veri setinde koruyarak dengeli bir dağılım sağlıyor. Böylece model, tüm sınıfları adil bir şekilde öğrenme şansı elde ediyor.

**Evaluation Function:** Bu fonksiyon, bir makine öğrenimi modelinin test verisi üzerindeki performansını değerlendirmek için tasarlanmıştır ve sonuçları metriklerle birlikte görsel olarak sunar. Modelin gerçek dünya, genel performansını test etmek ve genelleme yeteneğini ölçmeyi amaçlar.

**1.2.1.1. Model-1: Random Forest Base Model**

Ağaç tabanlı modellerde, özelliklerin ölçeklendirilmesi genellikle gerekli değildir. Bunun temel nedeni, bu modellerin bölünme kararlarını **Gini katsayısı** veya **entropi** gibi kriterlere göre vermesidir. Bu kriterler, en iyi bölünme noktalarını belirlerken yalnızca özelliklerin sırasına ve sınıf dağılımına odaklanır. Özelliklerin mutlak değerleri, bölünme kararlarını etkilemez. Bu durum, ölçeklendirme yapılmış ya da yapılmamış olmasının model performansına bir etkisi olmadığı anlamına gelir.

Bu durumu **Model-1** ve **Model-2** üzerinde yaptığımız analizlerle açıklıyoruz. Örneğin, yaş bilgisi içeren bir sütununuz olduğunu varsayalım:

* **Normalizasyon yapılmadan**: yaş > 18
* **Normalizasyon yapıldıktan sonra**: normalize\_yaş > 0.4

Bu eşikler örnek olarak verilmiştir, ancak dikkat edilmesi gereken husus, normalizasyon yapılıp yapılmadığından bağımsız olarak, bölünme sonuçlarının aynı olmasıdır. Çünkü ağaç tabanlı modeller, yalnızca özelliğin sırasına odaklanır ve ölçek değişikliklerinden etkilenmez.

Yine de, veri işleme süreçlerinde tutarlılığı sağlamak ve diğer modellerle karşılaştırma yaparken uyumlu bir pipeline oluşturmak adına, ölçeklendirme işlemleri bazı durumlarda kullanılabilir. Ancak, bu işlemler ağaç tabanlı modeller için zorunlu değildir.

**1.2.1.2. Model-2: Random Forest + Scale**

Burada scale edilmiş data modele verilmiştir. Default hiperparametreler kullanılmıştır. **Sonuç olarak**, ağaç tabanlı modellerde ölçeklendirme işlemi performans üzerinde bir fark yaratmaz. Bu çalışmada da, bu durumu göstermek amacıyla ölçeklendirilmiş ve ölçeklendirilmemiş veriler üzerinde Model-1 ve Model-2 ile karşılaştırma yapılmıştır.

**1.2.1.3. Model-3: Random Forest + PCA**

Default hiperparametreler kullanılarak Random Forest modeli eğitilmiştir. Data olarak PCA sonucu elde edilmiş ve seçilmiş olan değerler verilmiştir.

**1.2.1.4. Model-4: Random Forest + PCA + Balance Weights**

model\_4 = RandomForestClassifier(random\_state=42, n\_estimators=100, class\_weight='balanced')

Görüldüğü üzere burada class\_weights bölümünde balanced eklenmiştir. Balanced ekleyerek dengesiz verisetimizde güzel değişimler olacak mı kontrol etmek istiyoruz. Belki modelimizi güzel şekilde etkileyebilir. Eğitim datasına PCA ile elde ettiğimiz dönüştürülmüş veriler verilecektir.

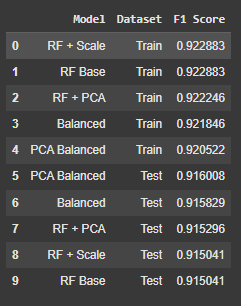
**1.2.1.5. Model-5: Random Forest + Balance Weights**

model\_5 = RandomForestClassifier(random\_state=42, n\_estimators=100, class\_weight='balanced')

Burada da benzer şekilde class\_weight = ‘balanced’ kullanıyoruz lakin eğitim dataseti olarak normal datamızı veriyoruz. PCA işlemi yaptığımız dataseti vermiyoruz.

**1.2.1.6. Random Forest Modellerin Sonuçları**

Modellerde görüldüğü gibi PCA Balanced test üzerinde en yüksek başarıyı gösteriyor bunun olmasının muhtemel sebebi PCA Balanced ile daha genel bir model oluşturulduğu için Test verisini genelleme başarısı da biraz olsun daha yüksek olabilir diye düşünmekteyim.



**1.2.1.7. Hiperparametre Optimizasyonu RF PCA + Balanced**

# Hiperparametreler için grid

param\_grid = {

    'n\_estimators': [50, 100, 200],          # Ağaç sayısı

    'max\_depth': [10, 20, 30],         # Maksimum derinlik

    'min\_samples\_split': [ 5, 10],         # Dallanma için minimum örnek sayısı

    'min\_samples\_leaf': [12, 4],           # Yaprak düğümündeki minimum örnek sayısı

    'max\_features': ['sqrt', 'log2', None],  # Her bir ağaç için kullanılacak maksimum özellik sayısı

}

Hiperparametre optimizasyonu için bir grid belirlenmiş olup bunun üzerinde GridSearchCV fonksiyonu ile optimizasyon yapılmıştır. RandomSearch kullanılmadı çünkü zaten çok uzun süren bir eğitim ilemi yoktu ve datamız çok büyük değildi.

Sonuç olarak: Best Parameters: {'max\_depth': 20, 'max\_features': 'sqrt', 'min\_samples\_leaf': 4, 'min\_samples\_split': 5, 'n\_estimators': 100}

**n\_estimators**: Bu, modelde kaç tane ağaç kullanılacağını belirleyen bir hiperparametre. Ağaç sayısı arttıkça model daha fazla öğrenme yapıyor, ama tabii ki eğitim süresi de uzuyor. Örneğin, 50 ağaçla çalıştırdığımızda model daha hızlı sonuç veriyor ama bazen yeterince iyi genelleme yapamayabiliyor. 200 ağaçta ise sonuçlar genelde daha kararlı oluyor. Bu yüzden burada farklı ağaç sayılarıyla denemeler yapıyoruz.

**max\_depth**: Bu parametre, her bir ağacın maksimum derinliğini belirliyor. Daha derin bir ağaç, daha fazla detay öğrenebiliyor ama bu durum overfitting'e (aşırı öğrenme) yol açabilir. Örneğin, derinliği 10 olan bir ağaç, daha genel bir model üretirken; 30 derinliğinde bir ağaç, veri setindeki en küçük ayrıntılara bile uyum sağlayabilir. Burada, farklı derinliklerle deneme yaparak en uygun değeri bulmaya çalışıyoruz.

**min\_samples\_split**: Bu, bir düğümün dallanabilmesi için gerekli olan minimum örnek sayısını belirliyor. Örneğin, bir düğümde en az 5 örnek varsa dallanıyor, aksi halde dallanma olmuyor. Eğer bu değeri çok küçük yaparsak modelimiz çok karmaşık hale gelebilir. Büyük yaparsak da model daha genel ve dengeli sonuçlar verebilir.

**min\_samples\_leaf**: Bu parametre, yaprak düğümünde bulunması gereken minimum örnek sayısını ifade ediyor. Yaprak düğümünü düşünün, yani artık dallanma yapılmıyor. Buradaki minimum örnek sayısı modelin daha sade mi yoksa daha karmaşık mı olacağını belirliyor. Örneğin, yaprakta 12 örnek olması gerektiğinde model biraz daha basit oluyor ve aşırı öğrenmeyi önlüyor.

**max\_features**: Bu hiperparametre ise her bir ağacın dallanması sırasında kullanabileceği maksimum özellik sayısını belirliyor. Mesela, 'sqrt' dediğimizde toplam özellik sayısının karekökü kadar özelliği kullanıyor. 'log2' dediğimizde ise logaritma (base 2) kadar özellik seçiliyor. Eğer None dersek, tüm özellikler kullanılıyor. Bu parametre aslında çeşitlilik yaratıyor, çünkü her ağaç aynı özelliklere bakmak zorunda kalmıyor.

**1.2.1.8. Model-6: Random Forest PCA + ADASYN**

Çalışmamızın başında bir ADASYN kullanan bir makaleye eleştiride bulunmuştuk. Şimdi bununla ilgili çalışıyoruz. Buradaki bölüm [[Comparison of multiclass classification techniques using dry bean dataset]](https://www.sciencedirect.com/science/article/pii/S2666307423000013) makalesinde yapılan işlemlerin aslında sağlıklı olmadığını göstermek amacıyla yapılmıştır. Öncelikle ADASYN ile oversampling yapıyoruz.

Counter({0: 2881, 2: 2873, 4: 2868, 3: 2837, 1: 2837, 5: 2783, 6: 2749})

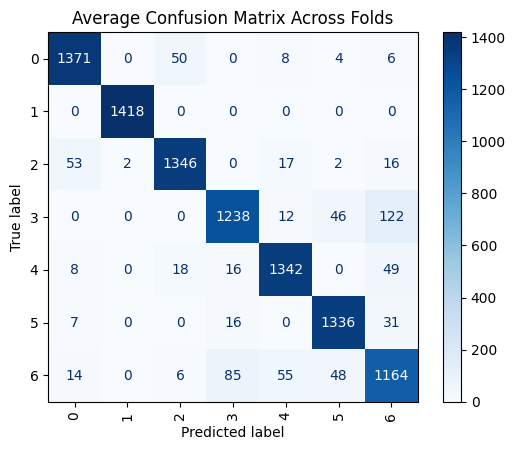
Gördüğümüz gibi ADASYN sonrasında veriler dengeli bir hale geldi. Şimdi Adasyn öncesi ve Adasyn sonrası verileri üzerine PCA dönüşüm işlemleri yapılarak karşılaştırıp Adasyn oversampling işleminin ne yaptığı görselleştirilecektir.

PCA işlemi sonrasında aşağıdaki gibi sonuçlar elde edilmiştir. Oversampling sonrası PCA ile görselleştirme yapıldığında görüldüğü üzere ADASYN aslında sağda görünen turuncu noktalara eklemeler yapmıştır daha çok ve bu şekilde kendi eklediği noktalarla modelin validation bölümünde performansı yükselmiş gibi göstermektedir.

A screenshot of a graph

Description automatically generated

Aşağıdaki Şekil’de görüldüğü üzere Classification Report sonucunda 3 ve 6'nın hala karıştırıldığı görülmektedir. Ve diğerlerinden daha kötü bir sonuç elde edilmektedir. Makaledeki sonuçları elde edememizin sebebi oversampling işleminin test datasına da uygulanmış olmasıdır. Bireysel çalışmalarda datamızın tamamına oversampling yaptığımızda ve diğer işlemleri de uyguladığımızda biz de oldukça yüksek sonuçlar elde ediyoruz ama bunun sebebi modelin test datası hakkında bilgi edinmesi (data leakage) ve testteki zaten tahmin edilen datanın sayısını çoğaltarak başarının artırılmasıdır.

A diagram of a test set

Description automatically generated

**1.2.1.9. Model-1: XGBoost Base Model**

Karşılaştırma yapmak amacıyla bir base model oluşturulmuştur.

xgb\_model\_1 = XGBClassifier(

    objective="multi:softprob",  # Çok sınıflı problemler için

    eval\_metric="mlogloss",     # Değerlendirme metriği

    random\_state=42

)

Çok sınıflı problemlerle çalışıldığı için gerekli ayarlamalar yapılmıştır.

**1.2.1.10. Model-2: XGBoost PCA**

PCA işlemi özellikleri indirgenmiş dataset ile eğitim yapılmıştır.

**1.2.1.11. Model-3: XGBoost PCA + Balancing Weights**

Bu modelde, sınıf dengesizliklerini ele almak için **weight balancing** yöntemi uygulanmış ve boyut azaltımı amacıyla **PCA (Principal Component Analysis)** kullanılmıştır. Ardından, bu işlemden geçirilen veriler üzerinde **XGBoost** modeli eğitilerek sınıflandırma performansı değerlendirilmiştir.

**1.2.12. XGBoost Sonuçlar**

A screenshot of a computer

Description automatically generated

Test datası üzerinde XGB ve PCA işleminin sonuçları daha iyi gözükmektedir çok da fark olmasa da. Bunun üzerine Hiperparametre optimizasyonu yapılacaktır.

**1.2.12. XGBoost Hiperparametre Optimizaasyonu**

**learning\_rate**: Bu parametre, modelin her adımda ne kadar öğrenmesi gerektiğini belirliyor. Genelde düşük bir öğrenme oranı (mesela 0.01) modelin daha yavaş ama daha dikkatli öğrenmesini sağlıyor. Yüksek bir oran (0.2 gibi) ise modelin daha hızlı öğrenmesini sağlasa da bazen doğruluğu düşürebiliyor. Bu yüzden farklı değerlerle deneme yaparak en uygun oranı bulmaya çalışıyoruz.

**max\_depth** Bu, her bir ağacın maksimum derinliğini belirleyen parametre. Ağaçlar derinleştikçe model, daha karmaşık ilişkileri öğrenebilir ama aşırı öğrenme (overfitting) riski de artar. Örneğin, max\_depth=3 dediğimizde ağaç daha sığ olur ve genelleştirme daha iyi olabilir. Ama max\_depth=7 ile daha detaylı bir öğrenme yapılır.

**Subsample** Bu parametre, her bir ağacın eğitimi sırasında kullanılacak verinin yüzde kaçını belirler. Örneğin, subsample=0.8 dediğimizde model, veri setinin %80’ini kullanır ve bu çeşitlilik sağlar. subsample=1.0 ise tüm veriyi kullanır. Çeşitlilik, overfitting’i azaltmak için faydalı olabilir.

**colsample\_bytree** Bu da, her bir ağacın dallanmasında kullanılacak özelliklerin (features) yüzde kaçını belirler. Mesela, colsample\_bytree=0.8 dediğimizde model her seferinde özelliklerin %80’i ile çalışır, böylece farklı ağaçlar arasında çeşitlilik artar. colsample\_bytree=1.0 ise tüm özellikleri kullanır.

**1.2.2. Regresyon**

Ridge regresyon, **çoklu doğrusal bağıntı** (multicollinearity) gibi sorunların yaşandığı durumlarda tercih edilen bir regresyon yöntemidir. Özellikle, bağımsız değişkenler arasında yüksek korelasyon olduğunda, standart **doğrusal regresyon (Linear Regression)** katsayı tahminlerinde kararsız sonuçlara yol açabilir. Ridge regresyon, bu durumu aşmak için **L2 düzenlileştirme** (L2 regularization) kullanır ve bizim datamızda da mevcut bir multicollinearity vardır.

**R² (Determination Coefficient):** Modelin açıklayabildiği varyans oranını ifade eder.

**MSE (Mean Squared Error):** Ortalama kare hata değeri, modelin tahminlerindeki hata büyüklüğünü ölçer.

**MAE (Mean Absolute Error):** Ortalama mutlak hata değeri, tahmin hatalarının ortalamasını ifade eder.

**MAPE (Mean Absolute Percentage Error):** Tahminlerin hedef değerlerden yüzde olarak ne kadar saptığını gösterir.

**SMAPE (Symmetric MAPE):** Tahminlerin ve gerçek değerlerin yüzdelik farkını simetrik bir şekilde ifade eder.

**1.2.2.1. Ridge Regresyon Base Model**

Model performansını değerlendirmek ve karşılaştırma yapmak için **Ridge Regresyon** kullanılarak bir **base model** oluşturulmuştur. Base model, daha karmaşık modellerin performansını kıyaslamak için başlangıç noktası olarak seçilmiştir.

**1.2.2.2. Ridge Regresyon Hiperparametre Optimizasyonu**

Modelin performansı oldukça yüksek çıktı, hiperparametre optimizasyonuyla pek bir beklentim yok. Lakin 5 kfold kullanarak GridSearch uyguluyorum. Parametrelerim aşağıdaki gibidir.

param\_grid = {

    'alpha': [0.1, 1, 10, 100, 1000],  # L2 regularization strength

}

İşlem sonucunda default alpha değeri çıkıyor. Yani en iyi alpha değeri 1 olarak elde ediyorum. Dolayısı ile sonuçlar değişmemektedir.

**1.2.2.3. Ridge Regresyon + Çerçeveleme Yöntemi**

Çerçeveleme yöntemiyle oluşturulan özellikler kullanılarak Ridge Regresyon modeli eğitilmiş ve test verisi üzerinde değerlendirilmiştir. Modelin performansı oldukça yüksek bir **R² (0.97)** skoru ile doğrulanmıştır, bu da modelin hedef değişkenin varyansının %97'sini açıkladığını gösterir.

Düşük bir **MSE (0.014)** ve **MAE (0.076)**, modelin tahmin hatalarının oldukça küçük olduğunu ifade etmektedir. Ancak, **MAPE (514.27)** ve **SMAPE (107.55)** metriklerinin yüksek olması, hedef değerlerin sıfıra yakın olduğu durumlarda yüzdelik hata metriklerinin yanıltıcı olabileceğini göstermektedir. Bu durum, yüzdelik hata metriklerinin yorumlanmasında dikkatli olunması gerektiğini ortaya koymaktadır.

**1.2.2.4. Ridge Regresyon + Çerçeveleme Yöntemi Hiperparametre Optimizasyonu**

Çerçeveleme yöntemiyle oluşturulan özellikler, Ridge Regresyon modeline uygulanmış ve ardından **GridSearchCV** ile hiperparametre optimizasyonu gerçekleştirilmiştir. Bu süreçte, modelin düzenlileştirme gücünü belirleyen **alpha** parametresi optimize edilmiş ve en iyi değer olarak **alpha=1** bulunmuştur. Bu zaten modelin default değeridir. Sonuçlarda bir değişim olmamıştır.

**1.2.2.5. LSTM Model**

LSTMRegressionModel sınıfı, zaman serisi verileri için **LSTM tabanlı bir regresyon modeli** oluşturmak, eğitmek ve test etmek amacıyla tasarlanmıştır. TensorFlow Keras kullanılarak inşa edilen model, giriş verilerinin zaman bağımlılıklarını öğrenmek için **LSTM katmanı** ile başlar. Ardından, özellikleri işlemek için bir **Dense (tam bağlı)** katman eklenir ve en sonunda regresyon çıktısını üretmek için bir **tek nöronlu çıkış katmanı** bulunur.

Model, **ReLU aktivasyonu**, 'adam' optimizasyon algoritması ve 'mse' (Mean Squared Error) kayıp fonksiyonuyla derlenir. Eğitim sırasında doğrulama verisi kullanılarak genelleştirme performansı izlenir. train metodu, modeli belirli bir **epoch** ve **batch size** ile eğitir ve eğitim geçmişini döndürür. evaluate metodu ise test veri seti üzerinde modelin **MSE** ve **MAE** metriklerini hesaplayarak performansı değerlendirir.

Bu amaçla önce train datası üzerinde validation yapılarak model performansı değerlendirilmiştir. Daha sonra Evaluation yapılarak doğrulama işlemi test verisi üzerinde gerçekleştirilmiştir.

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

**2. DENEYSEL BULGULAR**

**2.1 Sınıflandırma Sonuçları**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Test Verisi üzerinde Sonuçlar** | | | | |
| **accuracy** | **precision** | **recall** | **Macro f1** | **weighted f1** |
| **RF: Base Model** | **0.9150** | **0.9275** | **0.9262** | **0.9268** | **0.9150** |
| **RF: Scaled** | **0.9150** | **0.9275** | **0.9262** | **0.9268** | **0.9150** |
| **RF: PCA** | **0.9150** | **0.9283** | **0.9272** | **0.9277** | **0.9152** |
| **RF: PCA + Balance** | **0.9158** | **0.9290** | **0.9273** | **0.9281** | **0.9160** |
| **RF: Balanced** | **0.9158** | **0.9289** | **0.9274** | **0.9281** | **0.9158** |
| **RF: PCA + Adasyn** | **0.9073** | **0.9159** | **0.9241** | **0.91964** | **0.9072** |
| **XGB: Base Model** | **0.9180** | **0.9316** | **0.9288** | **0.9301** | **0.9181** |
| **XGB: PCA** | **0.9191** | **0.9310** | **0.9301** | **0.9305** | **0.9192** |
| **XGB: PCA + Balance** | **0.9187** | **0.9293** | **0.9286** | **0.9288** | **0.9189** |

**Hata Matrisleri**

A diagram of a test set

Description automatically generatedA diagram of a test set

Description automatically generated

**RF: Base Model RF: Scaled**

A diagram of a test set

Description automatically generatedA diagram of a test set

Description automatically generated

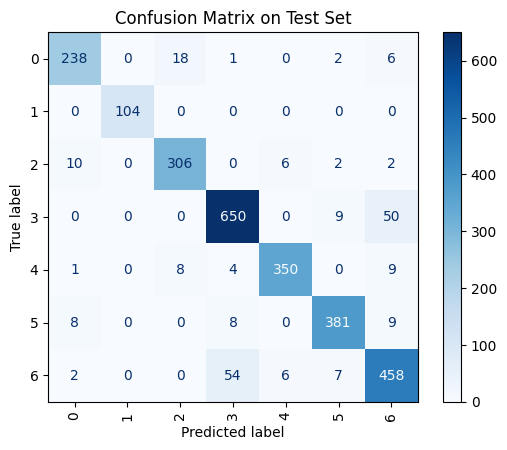
**RF: PCA RF: PCA + Balance**

A diagram of a test set

Description automatically generatedA diagram of a test set

Description automatically generated

**RF: Balanced RF: PCA + Adasyn**

A diagram of a test set

Description automatically generated

**XGB: Base Model XGB: PCA**

A diagram of a test set

Description automatically generated

**XGB: PCA + Balance**

**2.2 Regresyon Sonuçları**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Test Verisi üzerinde Sonuçlar** | | | | |
| **R2** | **MAE** | **MSE** | **MAPE** | **SMAPE** |
| **Ridge Base** | **0.9815** | **0.0765** | **0.0097** | **49.3835** | **18.6026** |
| **Ridge + Window** | **0.9737** | **0.0142** | **0.0765** | **514.2763** | **107.5467** |
| **LSTM + Window** | **0.9693** | **0.0166** | **0.1014** | **63.1526** | **23.2863** |

**LSTM grafiği**

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

**3.SONUÇ**

**3.1. Classification**

**Random Forest (RF):**

* RF modeliyle yapılan deneylerde, ölçeklendirme (scale) işleminin model performansı üzerinde belirgin bir etkisi olmadığı görülmüştür. Bunun sebebi, RF'nin ağaç tabanlı bir yöntem olması ve veri ölçeğinden bağımsız çalışabilmesidir.
* PCA uygulaması, boyut indirgemesi ile model performansında hafif bir artış sağlamıştır (%0.1'lik bir iyileşme).
* Dengesiz veri problemlerini ele almak için kullanılan class\_weight='balanced' yaklaşımı, sınıf dengesizliklerinin etkisini azaltarak modelin genelleme kapasitesini artırmıştır.
* ADASYN ile yapılan denemelerde, veri manipülasyonunun test verisine uygulanması durumunda model performansında yanıltıcı artışlar gözlemlenmiştir. Bu durum, test verisinin manipüle edilmemesi gerektiğinin önemini vurgulamaktadır.

**XGBoost (XGB):**

* XGBoost modelleri, Random Forest'a kıyasla genellikle biraz yüksek performans göstermiştir.
* PCA ve sınıf dengesi ayarlamalarıyla XGB modelinin genelleme başarısında iyileşme sağlanmıştır.

**3.2. Regression**

**Ridge Regresyon:**

* Ridge regresyon, veri setindeki çoklu doğrusal bağıntı problemini ele alarak yüksek R² skorları elde etmiştir (%98.15).
* Çerçeveleme (windowing) yöntemi maalesef beklenen etkiyi yaratamamıştır. Bunun sebebi belki de Ridge Regresyonun kendi çalışma yapısı ile ilgilidir.

**LSTM:**

* LSTM tabanlı zaman serisi modelleri, özellikle geçmiş veri örüntülerini öğrenmede başarılı olmuştur.
* Eğitim ve doğrulama aşamalarında düşük kayıp değerleri elde edilmiştir. Ancak, Ridge regresyon ile kıyaslandığında LSTM'nin performansı regresyon görevinde biraz daha düşük kalmıştır.
* Bu düşük kalmanın sebebinin makine öğrenmesi modellerinin düşük veride yapay sinir ağlarından daha başarılı olmasına bağlıyorum.

**4.EK**

**4.1 Sınıflandırma Kodları**

## Data

!pip install ydata\_profiling

import pandas as pd

import seaborn as sns

import ydata\_profiling

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split, StratifiedKFold, cross\_val\_score

from sklearn.preprocessing import MinMaxScaler, StandardScaler

from imblearn.over\_sampling import SMOTE

from imblearn.under\_sampling import RandomUnderSampler

from imblearn.combine import SMOTEENN

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, accuracy\_score, ConfusionMatrixDisplay, f1\_score, confusion\_matrix

from sklearn.decomposition import PCA

from xgboost import XGBClassifier

from collections import Counter

import pandas as pd

# Dataset metin bilgisini okuma

data\_url = "https://github.com/AysenurYrr/ML-Lab/raw/refs/heads/main/DryBeanDataset/Dry\_Bean\_Dataset.xlsx"

# Dataseti çekme ve okuma

data = pd.read\_excel(data\_url)

print(data.shape)

data.head()

# Genel bilgi

dataset\_info = data.info()

# Benzersiz sınıf isimlerini kontrol etme

class\_names = data['Class'].unique()

class\_names

## EDA

feature\_array = data.columns[:-1]  # Class hariç tüm sütunlar

# Class Analizi

plt.figure(figsize=(25, 5))

sns.countplot(x='Class', data=data)

plt.show()

# Tanımlayıcı İstatistikler (Descriptive Statistics)

data.describe()

# Box Plot Analizi

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# Boxplot çizmek için fonksiyon

def plot\_boxplot(data, feature, class\_name):

    """

    Verilen bir özellik ve sınıf için boxplot çizer.

    Parameters:

        data: pandas.DataFrame

            Analiz yapılacak veri seti.

        feature: str

            Boxplot oluşturulacak özellik adı.

        class\_name: str

            Belirli bir sınıfın adı.

    """

    plt.figure(figsize=(10, 6))

    sns.boxplot(x=data['Class'], y=data[feature], data=data)

    plt.title(f"Boxplot of {feature} for Class {class\_name}", fontsize=14)

    plt.xlabel("Class", fontsize=12)

    plt.ylabel(feature, fontsize=12)

    plt.xticks(rotation=45)

    plt.grid(axis='y', linestyle='--', alpha=0.7)

    plt.show()

for feature in feature\_array:

    plot\_boxplot(data, feature, "All Classes")

## Klasik PreProcess İşlemleri

# Delete Duplicate

data = data.drop\_duplicates()

data = data.reset\_index(drop=True)

data.shape

data

from sklearn.preprocessing import OneHotEncoder,LabelEncoder

# Label Encoding işlemi

label\_encoder = LabelEncoder()

data\_encoded\_label = data.copy()

data\_encoded\_label['Class\_Encoded'] = label\_encoder.fit\_transform(data['Class'])

data\_encoded\_label.drop("Class", axis=1, inplace = True)

print(data\_encoded\_label.shape)

data\_encoded\_label

# OneHotEncoder işlemi

encoder = OneHotEncoder(sparse\_output=False)

encoded\_classes = encoder.fit\_transform(data[['Class']])

# Yeni DataFrame'e ekleme

data\_encoded\_ohe = pd.DataFrame(encoded\_classes, columns=encoder.get\_feature\_names\_out(['Class']))

data\_encoded\_ohe = pd.concat([data, data\_encoded\_ohe], axis=1)

data\_encoded\_ohe.drop("Class", axis=1, inplace = True)

print(data\_encoded\_ohe.shape)

data\_encoded\_ohe

data\_encoded\_label

X = data\_encoded\_label.drop("Class\_Encoded", axis=1)

y = data\_encoded\_label["Class\_Encoded"]

X.head()

y.head()

## Veriyi Eğitime Hazırlama

# Veriyi test ve eğitim olarak ayırma

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# Eğitim setini ölçekleme

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

print("\nScale edilmemiş")

print(X\_train.shape)

print(y\_train.shape)

print("\nScale edilmiş")

print(X\_train\_scaled.shape)

print(y\_train.shape)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Test Datası, Gerçek dünyaya uygun olması açısından balance gibi şeyler yapılmayacaktır. Sadece Scale işlemi uygulanır")

print(X\_test\_scaled.shape)

print(y\_test.shape)

def visualize\_pca\_explained\_variance(pca\_model):

    """

    PCA açıklanabilirlik oranlarını görselleştirir ve oranları yazdırır.

    Parameters:

        pca\_model (PCA): PCA model nesnesi.

    Returns:

        None

    """

    explained\_variance = pca\_model.explained\_variance\_ratio\_

    cumulative\_variance = np.cumsum(explained\_variance)

    # Açıklanabilirlik oranlarını yazdırma

    for i, (ind\_var, cum\_var) in enumerate(zip(explained\_variance, cumulative\_variance), start=1):

        print(f"Principal Component {i}: Individual Explained Variance = {ind\_var:.4f}, Cumulative Explained Variance = {cum\_var:.4f}")

    # Görselleştirme

    plt.figure(figsize=(8, 5))

    plt.bar(range(1, len(explained\_variance) + 1), explained\_variance, alpha=0.7, color="blue", label="Individual Explained Variance")

    plt.step(range(1, len(explained\_variance) + 1), cumulative\_variance, where="mid", label="Cumulative Explained Variance", color="orange")

    plt.xlabel("Principal Components")

    plt.ylabel("Explained Variance Ratio")

    plt.title("Explained Variance by Principal Components")

    plt.legend()

    plt.tight\_layout()

    plt.show()

pca\_model = PCA(n\_components=10)

X\_train\_pca = pca\_model.fit\_transform(X\_train\_scaled)

X\_test\_pca = pca\_model.transform(X\_test\_scaled)

# PCA açıklanabilirlik oranlarını görselleştirme

visualize\_pca\_explained\_variance(pca\_model)

# İlk 5 feature'ı seçme

X\_train\_reduced = X\_train\_pca[:, :7]

X\_test\_reduced = X\_test\_pca[:, :7]

print("X\_train\_reduced Shape",X\_train\_reduced.shape)

print("X\_test\_reduced Shape",X\_test\_reduced.shape)

## Fonksiyonlar

# Validation Function

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

def KFold\_validation(X, y, model=None, cv=2):

    """

    Label encoded veri ile k-fold çapraz doğrulama uygulayarak bir modeli değerlendirir ve

    confusion matrix bilgilerini görselleştirir.

    Parameters:

        X (pd.DataFrame or np.ndarray): Özellikler (features).

        y (pd.Series or np.ndarray): Sınıf etiketleri (labels, label encoded formatında).

        model: Kullanılacak makine öğrenimi modeli (varsayılan RandomForestClassifier).

        cv (int): Çapraz doğrulama kat sayısı (folds).

    Returns:

        dict: CV doğruluk skorları, ortalama skor, standart sapma ve confusion matrices.

    """

     # Veriyi NumPy dizisine dönüştürme (Pandas uyumsuzluğu önlemek için)

    if isinstance(X, pd.DataFrame):

        X = X.values

    if isinstance(y, pd.Series):

        y = y.values

    skf = StratifiedKFold(n\_splits=cv, shuffle=True, random\_state=42)

    scores = []

    f1\_scores = []

    confusion\_matrices = []

    for train\_index, validation\_index in skf.split(X, y):

        X\_train, X\_validation = X[train\_index], X[validation\_index]

        y\_train, y\_validation = y[train\_index], y[validation\_index]

        # Model eğitimi ve tahmin

        model.fit(X\_train, y\_train)

        y\_pred = model.predict(X\_validation)

        # Skor kaydı

        scores.append(model.score(X\_validation, y\_validation))

        # F1 score kaydı

        f1 = f1\_score(y\_validation, y\_pred, average="weighted")

        f1\_scores.append(f1)

        # Confusion matrix

        cm = confusion\_matrix(y\_validation, y\_pred, labels=np.unique(y))

        confusion\_matrices.append(cm)

    # Ortalama confusion matrix

    mean\_cm = np.mean(confusion\_matrices, axis=0).astype(int)

    mean\_f1 = np.mean(f1\_scores)

    # Görselleştirme

    disp = ConfusionMatrixDisplay(confusion\_matrix=mean\_cm, display\_labels=np.unique(y))

    disp.plot(cmap='Blues', xticks\_rotation='vertical')

    plt.title("Average Confusion Matrix Across Folds")

    plt.show()

    # Sonuçları döndürme

    results = {

        #"CV Scores": scores,

        "Mean Accuracy": np.mean(scores),

        #"Mean Standard Deviation": np.std(scores),

        #"F1 Scores": f1\_scores,

        "Mean F1 Score": np.mean(f1\_scores)

        #"Confusion Matrices": confusion\_matrices

    }

    return results, mean\_f1

# Evaluation Function

def evaluation\_func(X\_test, y\_test, model):

    """

    Test verisi üzerinde bir modelin performansını değerlendirir ve görselleştirir.

    Parameters:

        X\_test (pd.DataFrame or np.ndarray): Test özellikleri (features).

        y\_test (pd.Series or np.ndarray): Test sınıf etiketleri (labels).

        model: Eğitimli makine öğrenimi modeli.

    Returns:

        dict: Classification report, confusion matrix.

    """

    # Veriyi NumPy dizisine dönüştürme (Pandas uyumsuzluğu önlemek için)

    if isinstance(X\_test, pd.DataFrame):

        X\_test = X\_test.values

    if isinstance(y\_test, pd.Series):

        y\_test = y\_test.values

    # Test verisi üzerinde tahmin yap

    y\_pred = model.predict(X\_test)

    # Classification report

    class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

    # Confusion matrix

    cm = confusion\_matrix(y\_test, y\_pred, labels=np.unique(y\_test))

    f1\_score\_ = f1\_score(y\_test, y\_pred, average="weighted")

    # Görselleştirme

    disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=np.unique(y\_test))

    disp.plot(cmap='Blues', xticks\_rotation='vertical')

    plt.title("Confusion Matrix on Test Set")

    plt.show()

    # Sonuçları döndürme

    results = {

        #"Confusion Matrix": cm,

        "f1 score": f1\_score\_,

        "Classification Report": class\_report

    }

    return results, f1\_score\_

## Model Random Forest

# Model-1: Base Model (Scale yok)

# Fonksiyonunu ksiyonu çağır ve sonuçları al

base\_model = RandomForestClassifier(random\_state=42)

results, rf\_base\_train\_f1 = KFold\_validation(X\_train, y\_train, base\_model, cv=2)

print(results)

test\_results, rf\_base\_test\_f1 = evaluation\_func(X\_test, y\_test, base\_model)

print("Classification Report:")

for label, metrics in test\_results["Classification Report"].items():

    print(f"{label}: {metrics}")

# Model-2: Scale Var

print("Scale edilmiş, Balance edilmemiş")

print(X\_train\_scaled.shape)

print(y\_train.shape)

# Fonksiyonunu çağır ve sonuçları al

model\_2 = RandomForestClassifier(random\_state=42)

results, rf\_scale\_train\_f1 = KFold\_validation(X\_train\_scaled, y\_train, model\_2, cv=2)

results

test\_results, rf\_scale\_test\_f1 = evaluation\_func(X\_test\_scaled, y\_test, model\_2)

print("Classification Report:")

for label, metrics in test\_results["Classification Report"].items():

    print(f"{label}: {metrics}")

# Model-3: PCA

# İlk 5 feature'ı seçme

X\_train\_reduced = X\_train\_pca[:, :5]

X\_test\_reduced = X\_test\_pca[:, :5]

print("X\_train\_reduced Shape",X\_train\_reduced.shape)

print("X\_test\_reduced Shape",X\_test\_reduced.shape)

model\_3 = RandomForestClassifier(random\_state=42)

results, rf\_pca\_train\_f1 = KFold\_validation(X\_train\_reduced, y\_train, model\_3, cv=2)

results

test\_results, rf\_pca\_test\_f1 = evaluation\_func(X\_test\_reduced, y\_test, model\_3)

print("Classification Report:")

for label, metrics in test\_results["Classification Report"].items():

    print(f"{label}: {metrics}")

# Model-4: Balance Weights + PCA

model\_4 = RandomForestClassifier(random\_state=42, n\_estimators=100, class\_weight='balanced')

results, rf\_pca\_bal\_train\_f1 = KFold\_validation(X\_train\_reduced, y\_train, model\_4, cv=2)

results

test\_results, rf\_pca\_bal\_test\_f1 = evaluation\_func(X\_test\_reduced, y\_test, model\_4)

print("Classification Report:")

for label, metrics in test\_results["Classification Report"].items():

    print(f"{label}: {metrics}")

# Model-5: RF Balanced

model\_5 = RandomForestClassifier(random\_state=42, n\_estimators=100, class\_weight='balanced')

results, rf\_bal\_train\_f1 = KFold\_validation(X\_train, y\_train, model\_5, cv=2)

results

test\_results, rf\_bal\_test\_f1 = evaluation\_func(X\_test, y\_test, model\_5)

print("Classification Report:")

for label, metrics in test\_results["Classification Report"].items():

    print(f"{label}: {metrics}")

# Verileri bir tabloya yerleştirme

df\_scores = {

    'Model': ['RF + PCA', 'RF + PCA', 'RF + Scale', 'RF + Scale', 'RF Base', 'RF Base','PCA Balanced','PCA Balanced',"Balanced","Balanced"],

    'Dataset': ['Test', 'Train', 'Test', 'Train', 'Test', 'Train', 'Test', 'Train',"Test", "Train"],

    'F1 Score': [

        rf\_pca\_test\_f1, rf\_pca\_train\_f1,

        rf\_scale\_test\_f1, rf\_scale\_train\_f1,

        rf\_base\_test\_f1, rf\_base\_train\_f1,

        rf\_pca\_bal\_test\_f1, rf\_pca\_bal\_train\_f1,

        rf\_bal\_test\_f1, rf\_bal\_train\_f1

    ]

}

df\_scores = pd.DataFrame(df\_scores)

df\_scores = df\_scores.sort\_values(by='F1 Score', ascending=False).reset\_index(drop=True)

df\_scores

## Hiperparametre Optimizasyonu

# Hiperparametreler için grid

param\_grid = {

    'n\_estimators': [50, 100, 200],          # Ağaç sayısı

    'max\_depth': [10, 20, 30],         # Maksimum derinlik

    'min\_samples\_split': [ 5, 10],         # Dallanma için minimum örnek sayısı

    'min\_samples\_leaf': [12, 4],           # Yaprak düğümündeki minimum örnek sayısı

    'max\_features': ['sqrt', 'log2', None],  # Her bir ağaç için kullanılacak maksimum özellik sayısı

# Grid Search

grid\_search = GridSearchCV(

    estimator=RandomForestClassifier(random\_state=42, class\_weight='balanced'),

    param\_grid=param\_grid,

    cv=2,                    # 5 katlı çapraz doğrulama

    scoring='f1\_macro',      # Makro F1 skoru ile değerlendirme

    n\_jobs=-1                # Paralel işlem kullanımı

)

# Modeli eğitme

grid\_search.fit(X\_train\_reduced, y\_train)

# En iyi parametreleri görüntüleme

print("Best Parameters:", grid\_search.best\_params\_)

# En iyi modeli seçme

best\_model = grid\_search.best\_estimator\_

# Test setinde tahmin yapma

y\_test\_pred = best\_model.predict(X\_test\_reduced)

# Sonuçları değerlendirme

print("Accuracy:", accuracy\_score(y\_test, y\_test\_pred))

print("F1 Score (Weighted):", f1\_score(y\_test, y\_test\_pred, average='weighted'))

print("F1 Score (Macro):", f1\_score(y\_test, y\_test\_pred, average='macro'))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_test\_pred))

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_test\_pred))

# Model-6: RF ADASYN

from imblearn.over\_sampling import ADASYN

# ADASYN uygulama

adasyn = ADASYN(random\_state=42)

X\_adasyn, y\_adasyn = adasyn.fit\_resample(X\_train\_scaled, y\_train)

print("ADASYN sonrası veri dağılımı:")

from collections import Counter

print(Counter(y\_adasyn))

pca\_before\_adasyn = PCA(n\_components=6)  # 6 bileşene indiriyoruz (isteğe bağlı artırılabilir)

X\_before\_adasyn\_train\_pca = pca\_before\_adasyn.fit\_transform(X\_train\_scaled)

# PCA ile açıklanan varyansı görüntüleme

explained\_variance = pca\_before\_adasyn.explained\_variance\_ratio\_

print("Açıklanan varyans oranları:", explained\_variance)

print("Toplam açıklanan varyans:", sum(explained\_variance))

pca\_adasyn = PCA(n\_components=6)  # 6 bileşene indiriyoruz (isteğe bağlı artırılabilir)

X\_adasyn\_train\_pca = pca\_adasyn.fit\_transform(X\_adasyn)

X\_adasyn\_test\_pca = pca\_adasyn.transform(X\_test\_scaled)

# PCA ile açıklanan varyansı görüntüleme

explained\_variance = pca\_adasyn.explained\_variance\_ratio\_

print("Açıklanan varyans oranları:", explained\_variance)

print("Toplam açıklanan varyans:", sum(explained\_variance))

# Yan yana iki grafik oluşturmak için subplots ayarı

fig, axs = plt.subplots(1, 2, figsize=(16, 6))

# İlk grafik: PCA ile 2 boyuta indirgenmiş veriler (ADASYN öncesi)

for label in np.unique(y\_train):

    axs[0].scatter(

        X\_before\_adasyn\_train\_pca[y\_train == label, 0],

        X\_before\_adasyn\_train\_pca[y\_train == label, 1],

        label=f"Class {label}", alpha=0.5

    )

axs[0].set\_title("PCA ile 2 Boyuta İndirgenmiş Veriler (Önce)")

axs[0].set\_xlabel("Principal Component 1")

axs[0].set\_ylabel("Principal Component 2")

axs[0].legend()

axs[0].grid()

# İkinci grafik: PCA ile 2 boyuta indirgenmiş veriler (ADASYN sonrası)

for label in np.unique(y\_adasyn):

    axs[1].scatter(

        X\_adasyn\_train\_pca[y\_adasyn == label, 0],

        X\_adasyn\_train\_pca[y\_adasyn == label, 1],

        label=f"Class {label}", alpha=0.5

    )

axs[1].set\_title("PCA ile 2 Boyuta İndirgenmiş Veriler (Sonra)")

axs[1].set\_xlabel("Principal Component 1")

axs[1].set\_ylabel("Principal Component 2")

axs[1].legend()

axs[1].grid()

# Grafiklerin gösterimi

plt.tight\_layout()

plt.show()

model\_6 = RandomForestClassifier(random\_state=42)

# Model performansını değerlendirme

results, rf\_adasyn\_train\_f1 = KFold\_validation(X\_adasyn\_train\_pca, y\_adasyn, model=model\_6, cv=2)

results, rf\_adasyn\_train\_f1

test\_results, rf\_adasyn\_test\_f1 = evaluation\_func(X\_adasyn\_test\_pca, y\_test, model\_6)

print("Classification Report:")

for label, metrics in test\_results["Classification Report"].items():

    print(f"{label}: {metrics}")

## XGBoost

# Model-1: Base Model

# XGBoost modeli tanımlama

xgb\_model\_1 = XGBClassifier(

    objective="multi:softprob",  # Çok sınıflı problemler için

    eval\_metric="mlogloss",     # Değerlendirme metriği

    random\_state=42

)

# Fonksiyonu çağırma.

results, xgb\_train\_f1 = KFold\_validation(X\_train, y\_train, model=xgb\_model\_1, cv=2)

# Sonuçları yazdırma

print("K-Fold Validation Results:")

print(f"Mean Accuracy: {results['Mean Accuracy']:.4f}")

print(f"Mean F1 Score: {xgb\_train\_f1:.4f}")

xgb\_model\_1.fit(X\_train,y\_train)

test\_results, xgb\_test\_f1 = evaluation\_func(X\_test, y\_test, xgb\_model\_1)

print("Classification Report:")

for label, metrics in test\_results["Classification Report"].items():

    print(f"{label}: {metrics}")

# Model-2: PCA

print("X\_train\_reduced Shape",X\_train\_reduced.shape)

print("X\_test\_reduced Shape",X\_test\_reduced.shape)

# XGBoost modeli tanımlama

xgb\_model\_2 = XGBClassifier(

    objective="multi:softprob",  # Çok sınıflı problemler için

    eval\_metric="mlogloss",     # Değerlendirme metriği

    random\_state=42

)

# Fonksiyonu çağırma.

results, xgb\_train\_pca\_f1 = KFold\_validation(X\_train\_reduced, y\_train, model=xgb\_model\_2, cv=2)

# Sonuçları yazdırma

print("K-Fold Validation Results:")

print(f"Mean Accuracy: {results['Mean Accuracy']:.4f}")

print(f"Mean F1 Score: {xgb\_train\_pca\_f1:.4f}")

xgb\_model\_2.fit(X\_train\_reduced,y\_train)

test\_results, xgb\_test\_pca\_f1 = evaluation\_func(X\_test\_reduced, y\_test, xgb\_model\_2)

print("Classification Report:")

for label, metrics in test\_results["Classification Report"].items():

    print(f"{label}: {metrics}")

# Model-3: Balancing Weights + PCA

xgb\_model\_3 = RandomForestClassifier(random\_state=42, n\_estimators=100, class\_weight='balanced')

results, xgb\_train\_bal\_pca\_f1 = KFold\_validation(X\_train\_reduced, y\_train, xgb\_model\_3, cv=2)

# Sonuçları yazdırma

print("K-Fold Validation Results:")

print(f"Mean Accuracy: {results['Mean Accuracy']:.4f}")

print(f"Mean F1 Score: {xgb\_train\_bal\_pca\_f1:.4f}")

xgb\_model\_3.fit(X\_train\_reduced,y\_train)

test\_results, xgb\_test\_bal\_pca\_f1 = evaluation\_func(X\_test\_reduced, y\_test, xgb\_model\_3)

print("Classification Report:")

for label, metrics in test\_results["Classification Report"].items():

    print(f"{label}: {metrics}")

# Conculusion

# Verileri bir tabloya yerleştirme

df2\_scores = {

    'Model': ['XGB\_Base','XGB\_Base', 'XGB\_PCA','XGB\_PCA','XGB\_Bal','XGB\_Bal'],

    'Dataset': ['Test', 'Train', 'Test', 'Train', 'Test', 'Train'],

    'F1 Score': [

        xgb\_test\_f1, xgb\_train\_f1,

        xgb\_test\_pca\_f1, xgb\_train\_pca\_f1,

        xgb\_test\_bal\_pca\_f1, xgb\_train\_bal\_pca\_f1

    ]

}

df2\_scores = pd.DataFrame(df2\_scores)

df2\_scores = df2\_scores.sort\_values(by='F1 Score', ascending=False).reset\_index(drop=True)

df2\_scores

# HyperParameter Optimization

!pip install scikit-learn==1.5.0

import xgboost as xgb

import pandas as pd

from sklearn.metrics import accuracy\_score, f1\_score

# XGBoost DMatrices (gerekli format)

dtrain = xgb.DMatrix(X\_train\_reduced, label=y\_train)

dtest = xgb.DMatrix(X\_test\_reduced, label=y\_test)

# Hiperparametre arama alanı

param\_grid = {

    'learning\_rate': [0.01, 0.1, 0.2],

    'max\_depth': [3, 5, 7],

    'subsample': [0.8, 1.0],

    'colsample\_bytree': [0.8, 1.0],

}

# En iyi sonuçları depolamak için

best\_params = None

best\_score = float("inf")  # Log loss olduğu için düşük skor daha iyi

best\_model = None

# Hiperparametre arama döngüsü

for learning\_rate in param\_grid['learning\_rate']:

    for max\_depth in param\_grid['max\_depth']:

        for subsample in param\_grid['subsample']:

            for colsample\_bytree in param\_grid['colsample\_bytree']:

                params = {

                    'objective': 'multi:softprob',

                    'num\_class': len(set(y\_train)),

                    'eval\_metric': 'mlogloss',

                    'learning\_rate': learning\_rate,

                    'max\_depth': max\_depth,

                    'subsample': subsample,

                    'colsample\_bytree': colsample\_bytree,

                    'seed': 42

                }

                # XGBoost CV

                cv\_results = xgb.cv(

                    params=params,

                    dtrain=dtrain,

                    num\_boost\_round=500,

                    nfold=2,

                    early\_stopping\_rounds=10,

                    verbose\_eval=False

                )

                # En düşük log loss değerini kontrol et

                mean\_mlogloss = cv\_results['test-mlogloss-mean'].min()

                if mean\_mlogloss < best\_score:

                    best\_score = mean\_mlogloss

                    best\_params = params

                    # En iyi modeli yeniden eğit

                    best\_model = xgb.train(

                        params=params,

                        dtrain=dtrain,

                        num\_boost\_round=len(cv\_results)

                    )

# Test verileri üzerinde tahmin yap

preds = best\_model.predict(dtest)

y\_pred = preds.argmax(axis=1)

# Accuracy ve F1 Score hesaplama

accuracy = accuracy\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred, average='weighted')

print("Best Parameters:", best\_params)

print("Best Log Loss:", best\_score)

print("Test Accuracy:", accuracy)

print("Test F1 Score:", f1)

Source: Murat KOKLU Faculty of Technology, Selcuk University, TURKEY. ORCID : 0000-0002-2737-2360 [mkoklu@selcuk.edu.tr](mailto:mkoklu@selcuk.edu.tr)

Ilker Ali OZKAN Faculty of Technology, Selcuk University, TURKEY. ORCID : 0000-0002-5715-1040 [ilkerozkan@selcuk.edu.tr](mailto:ilkerozkan@selcuk.edu.tr)

Relevant Papers: KOKLU, M. and OZKAN, I.A., (2020), “Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques.” Computers and Electronics in Agriculture, 174, 105507. DOI: [https://doi.org/10.1016/j.compag.2020.105507](https://www.google.com/url?q=https%3A%2F%2Fdoi.org%2F10.1016%2Fj.compag.2020.105507)

Citation Requests / Acknowledgements: KOKLU, M. and OZKAN, I.A., (2020), “Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques.” Computers and Electronics in Agriculture, 174, 105507. DOI: [https://doi.org/10.1016/j.compag.2020.105507](https://www.google.com/url?q=https%3A%2F%2Fdoi.org%2F10.1016%2Fj.compag.2020.105507)

**4.2 Regresyon Kodları**

# Load Dataset

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

import numpy as np

# Load the dataset while skipping the first row

url = "https://raw.githubusercontent.com/AysenurYrr/ML-Lab/refs/heads/main/UCI%20Algerian%20Forest%20Fires/Algerian\_forest\_fires\_dataset\_UPDATE.csv"

dataset = pd.read\_csv(url, skiprows=1)

print(dataset.shape)

dataset.head()

# Veri Ön İşleme

## missing values

dataset[dataset.isnull().any(axis=1)]

dataset.loc[:122,"Region"]=0

dataset.loc[122:,"Region"]=1

df=dataset

df[['Region']]=df[['Region']].astype(int)

df.head()

df.info()

df.iloc[[122]]

## Removing the null values

df=df.dropna().reset\_index(drop=True)

##remove the 122nd row

df=df.drop(122).reset\_index(drop=True)

##remove the 122nd row

df=df.drop(122).reset\_index(drop=True)

df.columns

df[['month','day','year','Temperature','RH','Ws']]=df[['month','day','year','Temperature',' RH',' Ws']].astype(int)

df.drop(columns=[' RH',' Ws'],inplace=True)

df.info()

objects=[features for features in df.columns if df[features].dtypes=='O']

objects

for i in objects:

    if i!='Classes  ':

        df[i]=df[i].astype(float)

df.rename(columns={'Rain ':'Rain'},inplace=True)

# Rename the "Class  " column to "Class"

df['Classes']=np.where(df['Classes  '].str.contains('not fire'),0,1)

df.drop(columns=['Classes  '], inplace=True)

df.info()

## Let ave the cleaned dataset

df.to\_csv('Algerian\_forest\_fires\_cleaned\_dataset.csv',index=False)

path = 'https://raw.githubusercontent.com/AysenurYrr/ML-Lab/refs/heads/main/UCI%20Algerian%20Forest%20Fires/Algerian\_forest\_fires\_cleaned\_dataset.csv'

df\_cleaned = pd.read\_csv(path)

print(f"Shape: {df\_cleaned.shape}")

df\_cleaned.head()

# EDA

df\_copy=df\_cleaned.drop(['day','month','year'],axis=1)

df\_copy.head()

## categories in classes

df\_copy['Classes'].value\_counts()

## Density Plot

## Plot desnity plot for all features

df\_copy.hist(bins=50,figsize=(20,15))

plt.show()

## Percentage for Pie Chart

percentage=df\_copy['Classes'].value\_counts(normalize=True)\*100

classlabels=["Fire","Not Fire"]

plt.figure(figsize=(12,7))

plt.pie(percentage,labels=classlabels,autopct='%1.1f%%')

plt.title("Pie Chart of Classes")

plt.show()

## Box Plot

import matplotlib.pyplot as plt

# List of continuous features

continuous\_features = ["Temperature", "RH", "Rain", "FFMC", "DMC", "DC", "ISI", "BUI", "FWI"]

# Create boxplots for continuous features

plt.figure(figsize=(15, 10))

for i, feature in enumerate(continuous\_features, 1):

    plt.subplot(3, 3, i)  # Adjust grid layout for clarity

    df\_cleaned.boxplot(column=feature)

    plt.title(feature)

plt.tight\_layout()

plt.show()

import matplotlib.pyplot as plt

# List of continuous features

continuous\_features = ["Temperature", "RH", "Rain", "FFMC", "DMC", "DC", "ISI", "BUI", "FWI"]

# Create boxplots for Region 0 and Region 1 separately

for region in [0, 1]:

    region\_data = df\_cleaned[df\_cleaned["Region"] == region]

    plt.figure(figsize=(15, 10))

    for i, feature in enumerate(continuous\_features, 1):

        plt.subplot(3, 3, i)  # Adjust grid layout for clarity

        region\_data.boxplot(column=feature)

        plt.title(f'{feature} (Region {region})')

    plt.tight\_layout()

    plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

# Filter data by Region

dftemp\_region\_1 = df\_cleaned[df\_cleaned['Region'] == 1]

dftemp\_region\_0 = df\_cleaned[df\_cleaned['Region'] == 0]

# Create subplots

fig, axes = plt.subplots(1, 2, figsize=(20, 6), sharey=True)

# Sidi-Bel Abbes Region (Region 1)

sns.countplot(ax=axes[0], x='month', hue='Classes', data=dftemp\_region\_1, palette="viridis")

axes[0].set\_title("Fire Analysis of Sidi-Bel Abbes Region", weight='bold')

axes[0].set\_xlabel("Months", weight='bold')

axes[0].set\_ylabel("Number of Fires", weight='bold')

# Bejaia Region (Region 0)

sns.countplot(ax=axes[1], x='month', hue='Classes', data=dftemp\_region\_0, palette="viridis")

axes[1].set\_title("Fire Analysis of Bejaia Region", weight='bold')

axes[1].set\_xlabel("Months", weight='bold')

axes[1].set\_ylabel("")

# Adjust layout

plt.tight\_layout()

plt.show()

df\_cleaned.head()

# Function to plot FWI values over time with fire and no-fire indicators

def plot\_fwi\_with\_fire\_and\_no\_fire(region\_data, region\_name):

    plt.figure(figsize=(14, 6))

    # Plot FWI values over time

    plt.plot(region\_data.index, region\_data['FWI'], label='FWI (Line)', color='blue', linewidth=1.5)

    # Highlight days with fire (Class == 1) as red points

    fire\_days = region\_data[region\_data['Classes'] == 1]

    plt.scatter(fire\_days.index, fire\_days['FWI'], color='red', label='Fire (Class=1)', zorder=5)

    # Highlight days with no fire (Class == 0) as blue points

    no\_fire\_days = region\_data[region\_data['Classes'] == 0]

    plt.scatter(no\_fire\_days.index, no\_fire\_days['FWI'], color='blue', label='No Fire (Class=0)', zorder=5)

    # Formatting the plot

    plt.title(f"FWI Over Time for {region\_name}", weight='bold')

    plt.xlabel("Time (Days)", weight='bold')

    plt.ylabel("FWI", weight='bold')

    plt.legend()

    plt.grid(alpha=0.5)

    plt.tight\_layout()

    plt.show()

# Plot for Region 1 (Sidi-Bel Abbes)

plot\_fwi\_with\_fire\_and\_no\_fire(dftemp\_region\_0, "Sidi-Bel Abbes Region")

# Plot for Region 0 (Bejaia)

plot\_fwi\_with\_fire\_and\_no\_fire(dftemp\_region\_1, "Bejaia Region")

# Preprocess

from sklearn.model\_selection import train\_test\_split

# Split for Region 0

train\_region\_0, test\_region\_0 = train\_test\_split(dftemp\_region\_0, test\_size=0.2, shuffle=False)

# Split for Region 1

train\_region\_1, test\_region\_1 = train\_test\_split(dftemp\_region\_1, test\_size=0.2, shuffle=False)

# Combine train and test data from both regions

train\_combined = pd.concat([train\_region\_0,train\_region\_1], ignore\_index=True)

test\_combined = pd.concat([test\_region\_0,test\_region\_1], ignore\_index=True)

plot\_fwi\_with\_fire\_and\_no\_fire(train\_combined, "Sidi-Bel Abbes and Bejaia Region Region Train")

plot\_fwi\_with\_fire\_and\_no\_fire(test\_combined, "Sidi-Bel Abbes and Bejaia Region Region Test")

## Scale

train = train\_combined.drop(columns=['day','month','year'])

test = test\_combined.drop(columns=['day','month','year'])

# Temporarily store the 'FWI' column and remove it from train and test

fwi\_train = train\_combined['FWI']

train = train\_combined.drop(columns=['FWI'])

train['FWI'] = fwi\_train  # Add 'FWI' back at the end

fwi\_test = test\_combined['FWI']

test = test\_combined.drop(columns=['FWI'])

test['FWI'] = fwi\_test  # Add 'FWI' back at the end

train.head()

# Initialize scalers for features and target

feature\_scaler = StandardScaler()

target\_scaler = StandardScaler()

# Scale features

X\_train\_scaled = feature\_scaler.fit\_transform(train.drop(columns=['FWI']))

X\_test\_scaled = feature\_scaler.transform(test.drop(columns=['FWI']))

# Scale target

y\_train\_scaled = target\_scaler.fit\_transform(train[['FWI']])

y\_test\_scaled = target\_scaler.transform(test[['FWI']])

print("X\_train\_scaled Shape:",X\_train\_scaled.shape)

print("X\_test\_scaled Shape:",X\_test\_scaled.shape)

print("y\_train\_scaled Shape:",y\_train\_scaled.shape)

print("y\_test\_scaled Shape:",y\_test\_scaled.shape)

# Reshape y\_train\_scaled and y\_test\_scaled to 1D arrays for compatibility

y\_train\_scaled\_ = y\_train\_scaled.reshape(-1)

y\_test\_scaled\_ = y\_test\_scaled.reshape(-1)

## Data Windowing

def create\_windows\_with\_targets(data, targets, window\_size, step\_size):

    """

    Create sliding windows for time-series data along with their corresponding target values.

    :param data: Numpy array of features (X).

    :param targets: Numpy array of target values (y).

    :param window\_size: Number of time steps in each window.

    :param step\_size: Step size between windows.

    :return: Sliding windows (X\_windows) and corresponding targets (y\_targets).

    """

    X\_windows = []

    y\_targets = []

    for i in range(0, len(data) - window\_size + 1, step\_size):

        X\_windows.append(data[i:i + window\_size])  # Extract the window

        y\_targets.append(targets[i + window\_size - 1])  # Target corresponds to the last time step in the window

    return np.array(X\_windows), np.array(y\_targets)

# Define window and step size

window\_size = 3

step\_size = 1

# Perform windowing on the scaled train data

X\_train\_windows, y\_train\_windows = create\_windows\_with\_targets(X\_train\_scaled, y\_train\_scaled, window\_size, step\_size)

X\_test\_windows, y\_test\_windows = create\_windows\_with\_targets(X\_test\_scaled, y\_test\_scaled, window\_size, step\_size)

# Display shapes of the resulting windows and targets

print("Train Shapes:",X\_train\_windows.shape, y\_train\_windows.shape)

print("Test Shapes:",X\_test\_windows.shape, y\_test\_windows.shape)

# Reshape y\_train and y\_test to 1D arrays for compatibility with scikit-learn

y\_train\_reshaped = y\_train\_windows.reshape(-1)

y\_test\_reshaped = y\_test\_windows.reshape(-1)

# Flatten X\_train and X\_test to 2D arrays for compatibility with logistic regression

X\_train\_flattened = X\_train\_windows.reshape(X\_train\_windows.shape[0], -1)

X\_test\_flattened = X\_test\_windows.reshape(X\_test\_windows.shape[0], -1)

print(f"x\_train\_reshaped {X\_train\_flattened.shape}")

print(f"y\_train\_reshaped {y\_train\_reshaped.shape}")

print(f"x\_test\_reshaped {X\_test\_flattened.shape}")

print(f"y\_test\_reshaped {y\_test\_reshaped.shape}")

y\_train\_reshaped.shape

# Modelling Ridge Regression

##Base Model

# Ridge Regressor'ı başlatma

ridge\_model\_base = Ridge()

# Modeli ölçeklendirilmiş eğitim verisi üzerinde eğitme

ridge\_model\_base.fit(X\_train\_scaled, y\_train\_scaled\_)

# Test verisi üzerinde tahmin yapma

y\_test\_pred = ridge\_model\_base.predict(X\_test\_scaled)

# Model performansını değerlendirme

test\_mse = mean\_squared\_error(y\_test\_scaled\_, y\_test\_pred)

test\_mae = mean\_absolute\_error(y\_test\_scaled\_, y\_test\_pred)

test\_r2 = r2\_score(y\_test\_scaled\_, y\_test\_pred)

print(f"Test MSE: {test\_mse}")

print(f"Test MAE: {test\_mae}")

print(f"Test R2 Score: {test\_r2}")

# Performans metriklerini tuple olarak döndürme

(test\_mse, test\_mae, test\_r2)

# Hyper Parameter Optimization

# Ridge Regression modelini başlat

ridge = Ridge()

# Hiperparametre aralığını tanımla

param\_grid = {

    'alpha': [0.1, 1, 10, 100, 1000],  # L2 regularization strength

}

# GridSearchCV ile hiperparametre optimizasyonu

grid\_search = GridSearchCV(estimator=ridge, param\_grid=param\_grid, cv=5, scoring='r2')

grid\_search.fit(X\_train\_scaled, y\_train\_scaled\_)

# En iyi modeli seç

best\_params = grid\_search.best\_params\_

best\_params

ridge\_best\_model = Ridge(\*\*best\_params)

ridge\_best\_model.fit(X\_train\_scaled, y\_train\_scaled\_)

# Tahmin yap ve performansı değerlendir

y\_test\_pred = ridge\_best\_model.predict(X\_test\_scaled)

# Performans değerlendirmesi

test\_mse = mean\_squared\_error(y\_test\_scaled\_, y\_test\_pred)

test\_mae = mean\_absolute\_error(y\_test\_scaled\_, y\_test\_pred)

test\_r2 = r2\_score(y\_test\_scaled\_, y\_test\_pred)

# Sonuçları göster

{

    "Best Parameters": best\_params,

    "Test MSE": test\_mse,

    "Test Mae": test\_mae,

    "Test R2": test\_r2

}

# Windowing Technique

# Ridge Regression modelini başlat

ridge\_model\_window = Ridge()

# Ridge Regression modeli eğit

ridge\_model\_window.fit(X\_train\_flattened, y\_train\_windows)

# Tahmin yap

y\_train\_pred = ridge\_model\_window.predict(X\_train\_flattened)

y\_test\_pred = ridge\_model\_window.predict(X\_test\_flattened)

# Performans değerlendirmesi

train\_mse = mean\_squared\_error(y\_train\_windows, y\_train\_pred)

test\_mse = mean\_squared\_error(y\_test\_windows, y\_test\_pred)

train\_r2 = r2\_score(y\_train\_windows, y\_train\_pred)

test\_r2 = r2\_score(y\_test\_windows, y\_test\_pred)

# Sonuçları göster

print("Train MSE:", train\_mse)

print("Train R2:", train\_r2)

print("Test MSE:", test\_mse)

print("Test R2:", test\_r2)

#HyperParameter Optimization

# Define the parameter grid

param\_grid = {

    'alpha': [0.1, 1, 10, 50, 100, 200, 500, 1000]  # Example range of alpha values

}

# Define a Ridge model

ridge\_model = Ridge()

# Set up GridSearchCV

grid\_search = GridSearchCV(

    estimator=ridge\_model,

    param\_grid=param\_grid,

    scoring='r2',

    cv=5,  # Cross-validation folds

    verbose=1,

    n\_jobs=-1

)

# Flatten the training data if necessary

# Fit the model

grid\_search.fit(X\_train\_flattened, y\_train\_windows)

# Get the best parameters and the best estimator

best\_params\_ridge\_window = grid\_search.best\_params\_

best\_model\_ridge\_window = Ridge(\*\*best\_params\_ridge\_window)

best\_model\_ridge\_window.fit(X\_train\_flattened, y\_train\_windows)

# Print the best parameters

print("Best parameters:", best\_params)

# Evaluate the model on the test set

y\_test\_pred = best\_model\_ridge\_window.predict(X\_test\_flattened)

test\_mse = mean\_squared\_error(y\_test\_windows, y\_test\_pred)

test\_r2 = r2\_score(y\_test\_windows, y\_test\_pred)

print("Test MSE with best model:", test\_mse)

print("Test R2 with best model:", test\_r2)

# Modelling LSTM with Windowed Data

# Building Simple LSTM Model

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

class LSTMRegressionModel:

    def \_\_init\_\_(self, input\_shape, lstm\_units=64, dense\_units=32, optimizer='adam', loss='mse', metrics=['mae']):

        """

        Initializes the LSTM Regression Model.

        Parameters:

        - input\_shape: Tuple, shape of the input data (time\_steps, features).

        - lstm\_units: Integer, number of LSTM units.

        - dense\_units: Integer, number of Dense units.

        - optimizer: String, optimizer for the model.

        - loss: String, loss function.

        - metrics: List of metrics to monitor during training.

        """

        self.model = Sequential([

            LSTM(lstm\_units, activation='relu', input\_shape=input\_shape),

            Dense(dense\_units, activation='relu'),

            Dense(1)  # Output layer

        ])

        self.model.compile(optimizer=optimizer, loss=loss, metrics=metrics)

    def train(self, X\_train, y\_train, X\_valid, y\_valid, epochs=50, batch\_size=16):

        """

        Trains the LSTM model.

        Parameters:

        - X\_train: Training features.

        - y\_train: Training targets.

        - X\_valid: Validation features.

        - y\_valid: Validation targets.

        - epochs: Integer, number of epochs.

        - batch\_size: Integer, batch size for training.

        Returns:

        - history: Training history object.

        """

        history = self.model.fit(

            X\_train, y\_train,

            epochs=epochs,

            batch\_size=batch\_size,

            validation\_data=(X\_valid, y\_valid)

        )

        return history

    def evaluate(self, X\_test, y\_test):

        """

        Evaluates the LSTM model on test data.

        Parameters:

        - X\_test: Test features.

        - y\_test: Test targets.

        Returns:

        - loss: Loss value on the test data.

        - metrics: List of metric values on the test data.

        """

        loss, mae = self.model.evaluate(X\_test, y\_test)

        print("Test Loss (MSE):", loss)

        print("Test MAE:", mae)

        return loss, mae

#Validation

# 80-20 train-validation split

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(

    X\_train\_windows, y\_train\_windows, test\_size=0.2, shuffle=True, random\_state=42

)

# Initialize the model

input\_shape = (X\_train.shape[1], X\_train.shape[2])

lstm\_model = LSTMRegressionModel(input\_shape=input\_shape)

# Train the model

history = lstm\_model.train(X\_train, y\_train, X\_valid, y\_valid, epochs=50, batch\_size=16)

# Evaluate the model

loss, mae = lstm\_model.evaluate(X\_test\_windows, y\_test\_windows)

# Eğitim geçmişinden değerleri çekme

train\_loss = history.history['loss']

val\_loss = history.history['val\_loss']

train\_mae = history.history['mae']

val\_mae = history.history['val\_mae']

# Loss grafiği

plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)  # İlk grafik

plt.plot(train\_loss, label='Train Loss (MSE)')

plt.plot(val\_loss, label='Validation Loss (MSE)')

plt.xlabel('Epochs')

plt.ylabel('Loss (MSE)')

plt.title('Loss Over Epochs')

plt.legend()

# MAE grafiği

plt.subplot(1, 2, 2)  # İkinci grafik

plt.plot(train\_mae, label='Train MAE')

plt.plot(val\_mae, label='Validation MAE')

plt.xlabel('Epochs')

plt.ylabel('Mean Absolute Error (MAE)')

plt.title('MAE Over Epochs')

plt.legend()

plt.tight\_layout()

plt.show()

#Evaluation

history\_2 = lstm\_model.model.fit(X\_train\_windows, y\_train\_windows, epochs=15, batch\_size=16)

#Final

# Predictions on validation set

y\_valid\_pred = lstm\_model.model.predict(X\_test\_windows)

# Compute validation MAPE, MAE, and MSE

mae = np.mean(np.abs(y\_test\_windows - y\_valid\_pred))

mse = np.mean((y\_test\_windows - y\_valid\_pred) \*\* 2)

r2 = r2\_score(y\_test\_windows, y\_valid\_pred)

# Print metrics

print("Validation MAE:", mae)

print("Validation MSE:", mse)

print("Validation R2:", r2)

from sklearn.model\_selection import KFold

import numpy as np

# K-Fold Cross-Validation

k = 5  # Number of folds

kf = KFold(n\_splits=k, shuffle=True, random\_state=42)

fold\_no = 1

results = []

for train\_index, valid\_index in kf.split(X\_train\_windows):

    print(f"Training on Fold {fold\_no}...")

    # Train-validation split

    X\_train\_fold, X\_valid\_fold = X\_train\_windows[train\_index], X\_train\_windows[valid\_index]

    y\_train\_fold, y\_valid\_fold = y\_train\_windows[train\_index], y\_train\_windows[valid\_index]

    # Model initialization

    input\_shape = (X\_train\_fold.shape[1], X\_train\_fold.shape[2])

    lstm\_model = LSTMRegressionModel(input\_shape=input\_shape)

    # Model training

    history = lstm\_model.train(X\_train\_fold, y\_train\_fold, X\_valid\_fold, y\_valid\_fold, epochs=50, batch\_size=16)

    # Evaluate on validation set

    val\_loss, val\_mae = lstm\_model.evaluate(X\_valid\_fold, y\_valid\_fold)

    print(f"Fold {fold\_no} - Validation Loss: {val\_loss}, Validation MAE: {val\_mae}")

    results.append((val\_loss, val\_mae))

    fold\_no += 1

# Average results across folds

avg\_loss = np.mean([r[0] for r in results])

avg\_mae = np.mean([r[1] for r in results])

print(f"\nK-Fold Cross-Validation Results:")

print(f"Average Validation Loss: {avg\_loss}")

print(f"Average Validation MAE: {avg\_mae}")