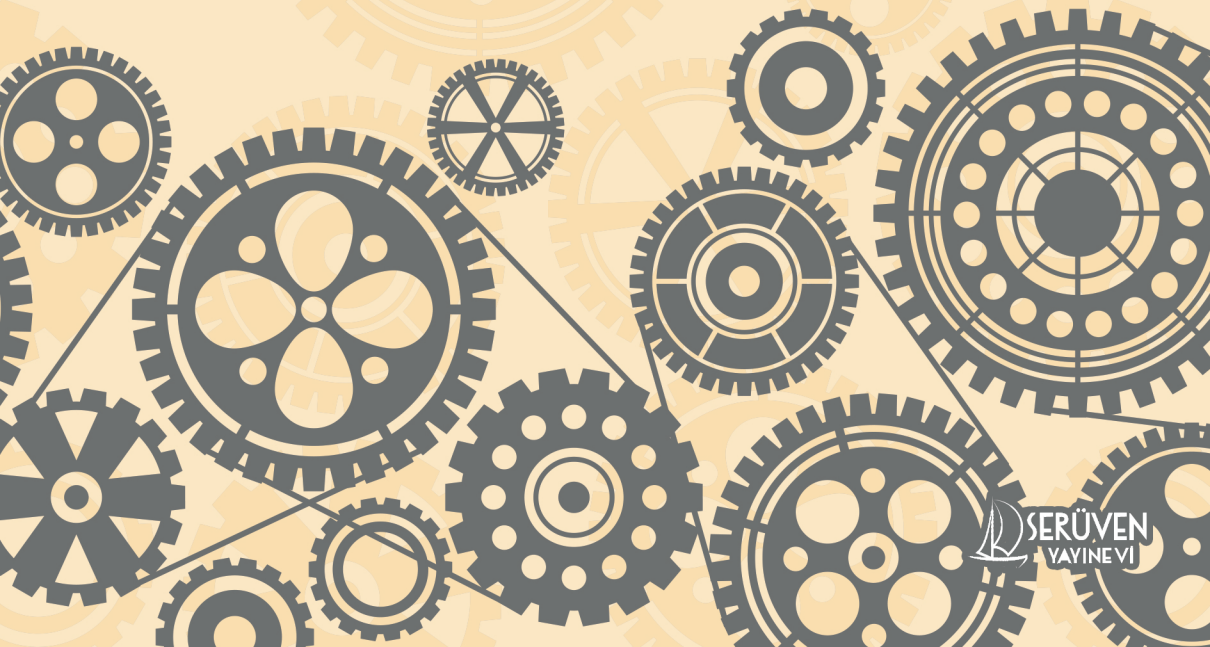
PYCARET İLE MAKİNE ÖĞRENMESİNE GİRİŞ

MEHMET AKİF BUZPINAR



SERÜVEN YAYINEVİ

Genel Yayın Yönetmeni / Editor in Chief • C. Cansın Selin Temana

Kapak & İç Tasarım / Cover & Interior Design • Serüven Yayınevi

Birinci Basım / First Edition • © Aralık 2024

ISBN • 978-625-5552-11-2

Serüven Yayınevi / Serüven Publishing
Türkiye Adres / Turkey Address: Kızılay Mah. Fevzi Çakmak 1. Sokak
Ümit Apt No: 22/A Çankaya/ANKARA
Telefon / Phone: 05437675765
web: www.seruvenyayinevi.com
e-mail: seruvenyayinevi@gmail.com

Baskı & Cilt / Printing & Volume
Sertifika / Certificate No: 47083

Pycaret ile Makine Öğrenmesine Giriş

İÇİNDEKİLER

PyCaret ile Makine Öğrenmesine Giriş

Sayın Okuyucu

Bu kitap, PyCaret kütüphanesinin az bilinen ancak son derece faydalı yönlerini ortaya koymak ve yapay zekâ ile ilgilenen herkese bu süreci kolay ve keyifli hale getirmek için yazıldı. PyCaret'in kullanıcı dostu yapısı ve minimal kodlama gereksinimi, yapay zekanın karmaşıklığını kolayca aşmanıza yardımcı olurken, etkili modeller geliştirme yollarını adım adım uygulamanızı sağlıyor.

Kitap boyunca, gerçek dünyadan sıkça karşılaşılan iki temel problem türüne (sınıflandırma ve değer tahminleme) odaklandım. Bu yöntemlerle sadece kavramları öğrenmekle kalmayacak, aynı zamanda bu bilgileri bilimsel uygulamalarda hayata geçirebileceksiniz.

Biraz merak ve sabırla, bu kitabı takip ederek kendi yapay zeka modellerinizi geliştirmeniz mümkün olacak. Modellerinizle etkili çözümler sunabilecek ve bunları bilimsel bir ortamda savunacak yetkinliğe ulaşacaksınız.

Mehmet Akif BUZPINAR

Önsöz

Yapay zekâ, 21. yüzyılın hızla değişen dünyasında hayatımızın vazgeçilmez bir parçası haline geldi. Eskiden sadece laboratuvarlarda geliştirilen bu teknoloji, bugün gündelik yaşamın her alanında kendini gösteriyor. Ancak, bu büyüleyici dünyaya adım atmak hala birçok kişi için göz korkutucu olabiliyor. Teknik terimler, algoritmalar ya da yazılım bilgisi eksikliği gibi engeller, bu yolda ilerlemek isteyenleri kolayca yıldırabiliyor.

Bu kitap, işte tam da bu engelleri aşmanıza yardımcı olmak için yazıldı. Yapay zekâ ve veri bilimi alanında fark yaratmak isteyen herkes için PyCaret kütüphanesini sade ve anlaşılır bir şekilde anlatmayı amaçlıyor. PyCaret'in kullanıcı dostu yapısı ve minimal kodlama gereksinimi, yapay zekanın karmaşık gibi görünen yönlerini basitleştiriyor ve etkili modeller geliştirme sürecini adım adım uygulamanızı sağlıyor.

Yazım sürecinde, karmaşık kavramları olabildiğince sadeleştirmeye özen gösterdim. Amacım, yalnızca PyCaret'in teknik detaylarını öğretmek değil; bu bilgileri gerçek hayattaki yapay zekâ projelerine nasıl aktarabileceğinizi göstermekti. Kitap boyunca, sınıflandırma ve değer tahminleme gibi sıkça karşılaşılan iki temel problem türüne odaklandım. Bu rehber, sizi karmaşadan uzaklaştırıp, bu problemleri çözmek için gereken yöntemleri kolayca uygulayabileceğiniz bir yapıya kavuşturuyor.

Biraz merak ve sabırla, bu kitap sayesinde kendi yapay zekâ modellerinizi geliştirebilir ve bu çözümleri bilimsel bir ortamda savunacak bilgi ve yetkinliğe ulaşabilirsiniz.

Keyifli okumalar dilerim,

Mehmet Akif BUZPINAR

Teşekkür

Bu kitabın ortaya çıkmasında emeği geçen herkese yürekten teşekkür etmek istiyorum.

Öncelikle, anneme, babama, sevgili eşim Tuğba BUZPINAR'a, en yakın dostlarım Ömer GÖRGÜÇ ve Erhan KAVUNCUOĞLU'na minnettarlığımı sunuyorum. Destekleri, sabırları ve her zaman yanımda olduklarını hissettirmeleri bu kitabı tamamlamamın en büyük nedenlerinden biri oldu.

Ayrıca, yapay zekâ dünyasını anlamama ve bu bilgileri paylaşmama ilham veren açık kaynak gönüllülerine, çalışmayı ve emek vermeyi en büyük erdem kabul etmiş meslektaşlarıma ve bu alandaki öncülere teşekkür ederim. PyCaret kütüphanesini geliştiren ekibe, bu harika aracı herkes için erişilebilir hale getirdikleri için özel bir teşekkür borçluyum.

Kitabın yazım aşamasında fikirlerini, önerilerini ve eleştirilerini paylaşarak daha iyi bir içerik ortaya çıkarmama yardımcı olan herkese sonsuz teşekkürler. Her bir geri bildirim, bu çalışmayı daha faydalı ve anlaşılır kılmak için önemli bir adımdı.

Ve tabii ki, bu kitabı eline alıp okuma zahmetine giren sizlere teşekkür etmek isterim. Bu yolculuğa çıkarken yanımda olduğunuzu bilmek, daha iyisini yapmak için en büyük motivasyonum oldu.

Sevgi ve saygılarımla,

Mehmet Akif BUZPINAR

1. Bölüm

PyCaret ile İkili Sınıflandırma

Giriş

İkili sınıflandırma, makine öğreniminin temel problemlerinden biridir ve bir veri kümesindeki örnekleri iki sınıftan birine atamayı amaçlar. Bu sınıflar genellikle pozitif-negatif veya 1-0 gibi etiketlerle ifade edilir. İkili sınıflandırma algoritmaları, günümüzde e-posta filtreleme, hastalık teşhisi ve kredi kartı dolandırıcılığı tespiti gibi birçok alanda yaygın bir şekilde kullanılmaktadır.

Bu bölümde, ikili sınıflandırma probleminin temel yapısını, çözüm süreçlerini ve bu süreçlerde kullanılan önemli algoritmaları detaylı bir şekilde ele alacağız.

PyCaret, makine öğrenmesi süreçlerini otomatikleştirmek için geliştirilmiş, Python tabanlı, açık kaynaklı ve düşük kod gerektiren bir kütüphanedir. Başından sonuna kadar bir makine öğrenimi modeli oluşturma ve yönetme sürecini hızlandırarak deney döngüsünü ciddi ölçüde kısaltır ve kullanıcıların daha üretken olmalarını sağlar.

PyCaret, özellikle makine öğrenimi iş akışlarını sadeleştirme ve hızlandırma konusunda öne çıkar. Geleneksel yöntemlerde yüzlerce satır kod gerektiren işlemleri yalnızca birkaç satırla gerçekleştirme imkânı sunar. Bu sayede hem zaman tasarrufu sağlar hem de iş akışını daha verimli hale getirir.

Bunun yanı sıra PyCaret, çeşitli popüler makine öğrenimi kütüphanelerini ve çerçevelerini (örneğin, scikit-learn, XGBoost, LightGBM, CatBoost, spaCy, Optuna, Hyperopt ve Ray) kapsayan bir Python paketidir. Bu entegrasyon, kullanıcıların farklı algoritmalar ve araçlarla kolayca çalışmasını sağlayarak, esnek ve güçlü bir modelleme deneyimi sunar.

Sonuç olarak, PyCaret'in sunduğu düşük kod yaklaşımı ve kapsamlı özellikleri hem yeni başlayanlar hem de uzmanlar için etkili bir çözüm ortağı haline gelmiştir. Bu kitap boyunca, PyCaret'i kullanarak ikili sınıflandırma problemlerine nasıl yaklaşabileceğimizi ve bu süreçte hangi adımları izleyebileceğimizi öğrenme fırsatı bulacaksınız.

Bu güçlü araçlardan yararlanarak yapay zekâ modelimizi geliştirmeye başlayalım!

1.1 Çalışma Ortamı ve PyCaret Kurulumu

Makine öğrenmesi algoritmalarının pycaret ile modellenmesi python dili ve jupyter notebook uygulaması üzerinde profesyönelce ve kolaylıkla gerçekleştirilebilir. Çalışma ortamının kurulması için en pratik adımlar aşağıda sunulmuştur.

I. Python 3.11 ve entegre *Anaconda* ortamının kurulumu bağlantısı (*https://repo.anaconda.com/archive/Anaconda3-2023.07-2-Windows-x86_64.exe*) PyCaret'ın desteklediği en üst Python sürümüne göre kurulmalıdır. (Kitap yazım tarihi referans alınırsa *Python 3. 12* Aralık 2024 itibariyle desteklenmektedir)

II. Visual Studio Code (*Jupyter notebook* ve *Python* eklentileri ile kurulmalıdır)

III. Pycaret kütüphanelerinin *Anaconda Promt* üzerinden (Liste 1.1) ortamına kurulması ile sonlandırılmalıdır. (Liste 2.1)

```
# Liste 1.1 Kütüphaneleri Yükleme
pip install pycaret
```

Liste 1.1 *Kütüphaneleri Yükleme*

Varsayılan kurulum gerekli pek çok kütüphaneyi kurmak için yeterlidir ancak tüm alt kütüphaneleri içermez. Tüm farklı amaçlar için ihtiyaç duyulabilecek kütüphaneleri kurmak için:

"pip install pycaret[full]" veya ihtiyacınıza göre aşağıdaki varyantlardan birini yükleyebilirsiniz:

Analiz araçları: *pip install pycaret[analysis]*

Modeller: *pip install pycaret[models]*

Hiperparametre ayarlama: *pip install pycaret[tuner]*

Model operasyonları: *pip install pycaret[mlops]*

Paralel işlem: *pip install pycaret[parallel]*

Test araçları: *pip install pycaret[test]*

```
# Liste 1.2 Sürüm Kontrolü
import pycaret
pycaret.__version__
```

Liste 1.2 *Kütüphaneleri Yükleme*

1.2 Başlangıç

PyCaret'in Sınıflandırma Modülü, verileri belirli gruplara ayırmak için kullanılan, makine öğrenmesi dünyasının güçlü bir aracıdır. Örneğin, bir müşterinin bir ürün satın alıp almayacağını, bir hastanın hasta olup olmadığı veya bir e-postanın spam mı yoksa normal mi olduğunu tahmin etmek gibi durumlarda bu modülü kullanabiliriz.

Bu modül, hem ikili (evet/hayır, var/yok gibi) hem de çoklu sınıf (birden fazla seçenek arasından birini seçme) problemler için uygundur. Veri ön işleme adımlarını otomatik olarak yaparak, model oluşturma sürecini hızlandırır. Ayrıca, 18'den fazla farklı algoritma ve görselleştirme aracı sayesinde, en uygun modeli seçip performansını değerlendirmenizi kolaylaştırır.

Bir yapay zekâ probleminin *PyCaret* aracılığıyla eğitim ve tahminle süreci aşağıdaki gibi özetlenebilir (Şekil 1.1).

1. Sistem gereksinimlerinin *PyCaret* ortamının problem türüne göre kurulması

2. Modellerin eğitilmesi

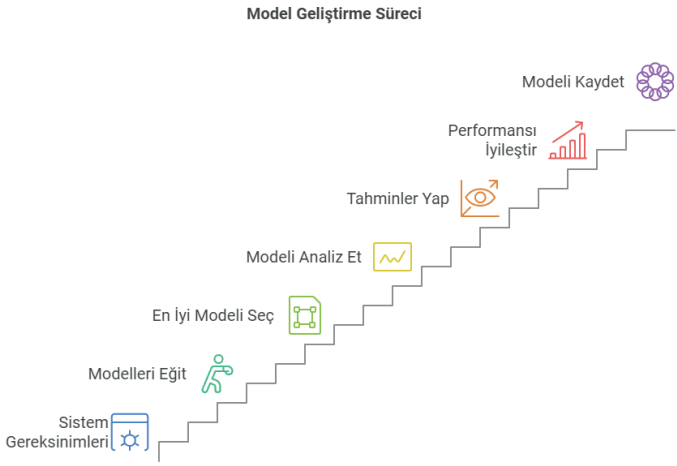
3. En başarılı modelin performans kriterleriyle karşılaştırılarak belirlenmesi

4. En iyi modelin analiz edilmesi

5. Yeni Verilerle Tahminler

6. Model Performans İyileştirme

7. Modelin Kaydedilmesi



Şekil 1.1 *Bir modelin geliştirilme süreci;*

1.3 Pycaret Eğitim Ortamının Hazırlanması

İkili sınıflandırma problemi için örnek veri seti *PyCaret'ın* kendi kütüphanesindeki *'diabetes'* verisetini yükleyerek başlayalım (Liste 1.3)

```
# Liste 1.3 PyCaret veri seti modülünden örnek veri seti yükleniyor
from pycaret.datasets import get_data
data = get_data('diabetes')
```

Liste 1.3 *Kütüphaneleri Yükleme*

Diabetes veri seti, makine öğrenimi ve istatistiksel analiz alanında yaygın olarak kullanılan bir veri setidir. Özellikle diyabet teşhisi ve tahmini için kullanılan bu veri seti, sağlık alanında karar destek sistemlerinin geliştirilmesi için ideal bir örnektir. *Diabetes* veri seti, 768 gözlemden oluşan geniş bir yapı sunar ve toplamda 8 bağımsız değişken ile 1 hedef değişken içerir. Bu değişkenler, diyabet teşhisinde etkili olduğu düşünülen çeşitli tıbbi ve biyolojik ölçümleri kapsamaktadır. Veri setinin bağımsız değişkenleri ve hedef değişkeni şu şekilde açıklanabilir:

Bağımsız Değişkenler (Öznitelikler – Kişilerin hasta olup olmadığını gösterecek ölçümler)

1. **Pregnancies**: Gebelik sayısı, bireyin yaşam boyu kaç kez gebe kaldığını ifade eder.

2. **Glucose**: Oral glikoz tolerans testi sonucu olarak ölçülen kan şekeri seviyesi.

3. **BloodPressure**: Kan basıncı (mmHg) ölçümü, kardiyovasküler risk faktörlerini değerlendirmek için kullanılır.

4. **SkinThickness**: Triseps cilt kıvrımı kalınlığı (mm), bireyin vücut yağ oranını dolaylı olarak ölçer.

5. **Insulin**: İnsülin seviyesi (mIU/mL), vücudun glikoz metabolizmasını nasıl düzenlediğine dair ipuçları sunar.

6. **BMI**: Vücut kitle indeksi (kg/m^2), bireyin vücut ağırlığının boyuna oranıdır ve obezite seviyesini gösterir.

7. **DiabetesPedigreeFunction**: Diyabet aile geçmişi fonksiyonu, bireyin genetik yatkınlığını ölçer.

8. **Age**: Yaş (yıl), diyabet riskinin yaşa göre nasıl değiştiğini anlamak için önemli bir parametredir.

Hedef Değişken (Bağımlı değişken -Tahmin edilecek değer)

Outcome: Diyabet teşhisi sonucu. Bu değişken, ikili sınıflandırma yapısı sunar ve değerleri şu şekildedir:

1: Pozitif diyabet teşhisi.

0: Negatif diyabet teşhisi.

Bu veri seti, ikili sınıflandırma örneği için seçilen numuneye ait kişinin diyabet olup olmadığını ikili sınıflandırma algoritmalarıyla tespit etmeye çalışacaktır. Veri setinin yapısı aşağıda gösterilmektedir (Şekil 1.2).

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)	Class variable
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Şekil 1.2 *Diyabet veri seti özeti*

PyCaret ortamının sınıflandırma için algoritmalarının seçilmesi sürecini Liste 1.4'teki komutla varsayılan ayarlarda gerçekleştirelim. Şekil 1.2 Çıktı olarak karşımıza gelecektir.

```
# Liste 1.4 PyCaret sınıflandırma ayarlarını içe aktarın ve kuruluma başlayın
from pycaret.classification import *
s = setup(data, target = 'Class variable', session_id = 123)
```

Liste 1.4 *Kütüphaneleri Yükleme*

	Description	Value
0	Session id	123
1	Target	Class variable
2	Target type	Binary
3	Original data shape	(768, 9)
4	Transformed data shape	(768, 9)
5	Transformed train set shape	(537, 9)
6	Transformed test set shape	(231, 9)
7	Numeric features	8
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	StratifiedKFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False
16	Log Experiment	False
17	Experiment Name	clf-default-name
18	USI	52db

Şekil 1.2 *Deney seviyesindeki bilgileri içeren bir bilgi tablosu*

Şekil 1.2, çalışmanın temel özelliklerini ve kurulum sırasında gerçekleştirilen işlemleri özetler. Şimdi bu bilgileri açıklayalım:

Oturum Kimliği (Session ID)

Oturum kimliği, tüm fonksiyonlara bir tohum olarak dağıtılan, rastgele bir sayıdan türetilen bir değerdir. Bu sayede çalışmanızın sonuçları daha sonra yeniden üretilebilir. Eğer kurulum sırasında bir *session_id* belirtilmezse, sistem otomatik olarak bir rastgele sayı oluşturur ve bu sayı tüm fonksiyonlara dağıtılır.

Hedef Türü (Target Type)

Hedef değişkenin türü, sistem tarafından otomatik olarak algılanır ve şu üç kategoriden birine atanır:

- **Binary (İkili Sınıflandırma)**: Örneğin, "Evet" ya da "Hayır".
- **Multiclass (Çoklu Sınıflandırma)**: Üç veya daha fazla kategori.

1. **Regression (Regresyon)**: Sürekli sayısal değerler.

Etiket Kodlama (Label Encoding)

- Eğer hedef değişken, sayısal değerler yerine metinsel ifadelerden oluşuyorsa (örneğin, "Evet" veya "Hayır"), sistem otomatik olarak bu metinleri sayısal değerlere çevirir.
- Örneğin: **0** : Hayır / **1** : Evet
- Bu eğitimde, hedef değişken zaten sayısal olduğu için herhangi bir etiket kodlama işlemine ihtiyaç duyulmamaktadır.

Veri Şekilleri (Data Shapes)

- **Orijinal Veri Şekli**: Herhangi bir dönüşüm uygulanmadan önce veri kümesinin boyutları.
- **Dönüştürülmüş Eğitim Veri Şekli**: Eğitim veri kümesi üzerinde yapılan dönüşümlerden sonra elde edilen boyutlar.
- **Dönüştürülmüş Test Veri Şekli**: Test veri kümesi üzerinde yapılan dönüşümlerden sonra elde edilen boyutlar.

Sayısal ve Kategorik Özellikler

- **Sayısal Özellikler**: Sayısal olarak değerlendirilen özelliklerin sayısı.
- **Kategorik Özellikler**: Kategori bazlı olarak değerlendirilen özelliklerin sayısı.

PyCaret'in API Türleri

PyCaret iki farklı API türü sunar:

- **Fonksiyonel API**: Yukarıda gördüğünüz şekilde doğrudan fonksiyonları çalıştırarak kullanılan yöntem.
- **Nesne Yönelimli API (Object Oriented API)**: Bu yöntemde, doğrudan fonksiyon çalıştırmak yerine bir sınıf (class) kullanılır. İlgili sınıfı içe aktardıktan sonra, bu sınıfın yöntemleri aracılığıyla işlemler gerçekleştirilir.

Bu iki yöntem, çalışma şeklinize ve tercihinize bağlı olarak farklı kullanım kolaylıkları sağlar. Örneğin, nesne yönelimli API, daha karmaşık projelerde modüler bir yapı sunarak yönetimi kolaylaştırabilir.

```
# Liste 1.5 Sınıflandırma deney ortamının oluşturulması
from pycaret.classification import ClassificationExperiment
exp = ClassificationExperiment()
# Üretilen deney ortam nesnesinin tip kontrolü
type(exp)
```

Liste 1.5 *Sınıflandırma Deneysel ortamının oluşturulması ve tip testi*

Liste1.5'teki, işlemin ardından tip testi sonucu "*pycaret.classification.oop.ClassificationExperiment*" olmalıdır. Deneysel ortamın *exp* nesnesi üzerinden kurulumu Liste 1.6'daki gibi yapılabilmektedir.

```
# Liste 1.6 exp deneysel ortamının kurulumu
exp.setup(data, target = 'Class variable', session_id = 123)
```

Liste 1.6 *Deneysel ortamda sınıflandırma problem tipinin belirlenmesi*

Deneysel ortamın hazırlanmasının ardından model eğitim parametreleri Şekil 1.3'teki gibi sunulmaktadır.

	Description	Value
0	Session id	123
1	Target	Class variable
2	Target type	Binary
3	Original data shape	(768, 9)
4	Transformed data shape	(768, 9)
5	Transformed train set shape	(537, 9)
6	Transformed test set shape	(231, 9)
7	Numeric features	8
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	StratifiedKFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False
16	Log Experiment	False
17	Experiment Name	clf-default-name
18	USI	0071

Şekil 1.3 *Eğitilecek sistem parametreleri*

Parametreleri özetleyecek olursak;

Session ID (Oturum Kimliği): Çalışmanız için rastgele atanmış bir kimlik numarası. Burada değer **123** olarak belirlenmiştir.

Target (Hedef Değişken): Modelin tahmin etmeye çalıştığı değişkendir. Bu çalışma için hedef, bir **sınıf değişkeni (Class variable)**.

Target Type (Hedef Türü): Hedef değişkenin türü **Binary (İkili Sınıflandırma)** olarak algılanmıştır.

Original Data Shape (Orijinal Veri Şekli): Ham verinin boyutları. Bu çalışma 768 satır ve 9 sütundan oluşan bir veri kümesine sahiptir (**768, 9**).

Transformed Data Shape (Dönüştürülmüş Veri Şekli): Veri dönü-

şüm işlemlerinden sonra boyutlar aynı kalmıştır (**768, 9**).

Transformed Train Set Shape (Dönüştürülmüş Eğitim Veri Şekli): Eğitim veri kümesi, 537 satır ve 9 sütundan oluşmaktadır (**537, 9**).

Transformed Test Set Shape (Dönüştürülmüş Test Veri Şekli): Test veri kümesi, 231 satır ve 9 sütundan oluşmaktadır (**231, 9**).

Numeric Features (Sayısal Özellikler): Veri kümesinde 8 sayısal özellik bulunmaktadır.

Preprocess (Ön İşleme): Veri ön işleme işlemleri uygulanmıştır (**True**).

Imputation Type (Eksik Değer Tamamlama Yöntemi): Eksik değerler için **basit** tamamlama yöntemi kullanılmıştır.

Numeric Imputation (Sayısal Tamamlama): Sayısal verilerdeki eksik değerler, ortalama (**mean**) kullanılarak tamamlanmıştır.

Categorical Imputation (Kategorik Tamamlama): Kategorik verilerdeki eksik değerler, mod (**mode**) kullanılarak tamamlanmıştır.

Fold Generator (Katlama Yöntemi): Veriyi katlamak için **StratifiedKFold** yöntemi tercih edilmiştir.

Fold Number (Katlama Sayısı): Veri kümesi 10 katlama ile çapraz doğrulama için hazırlanmıştır.

CPU Jobs (CPU İşlem Sayısı): Çalışma, tüm işlemcilerde paralel olarak gerçekleştirilmiştir (**-1**).

Use GPU (GPU Kullanımı): GPU kullanılmamıştır (**False**).

Log Experiment (Deney Kaydı): Deney kaydı tutulmamaktadır (**False**).

Experiment Name (Deney Adı): Deney adı varsayılan olarak **clf-default-name** şeklinde belirlenmiştir.

USI (Benzersiz Çalışma Kimliği): Çalışmaya özel benzersiz kimlik numarası **0071** olarak atanmıştır.

Sonraki aşamada bu ayarlara göre sistem eğitimi gerçekleştirilecektir.

1.4 Modellerin Eğitilmesi

Modellerin eğitilmesi sürecinde algoritmaların tek tek seçilerek eğitilmesi gerekmektedir ancak *PyCaret* bu işlemi kolaylaştırmak için kütüphanesindeki tüm sınıflandırma algoritmalarıyla eğitim gerçekleştirir ve performans kriterlerine göre bir liste halinde sunarak bu süreci son derece hızlandırmaktadır (Liste 1.7).

```
# Liste 1.7 Modellerin temel performans karşılaştırması
best = compare_models()
```

Liste 1.7 *Varsayılan sınıflandırma algoritmaları ile model eğitme ve en başarılı modeli atama*

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.7689	0.8047	0.5602	0.7208	0.6279	0.4641	0.4736	1.3810
ridge	Ridge Classifier	0.7670	0.0000	0.5497	0.7235	0.6221	0.4581	0.4690	0.0370
lda	Linear Discriminant Analysis	0.7670	0.8055	0.5550	0.7202	0.6243	0.4594	0.4695	0.0500
rf	Random Forest Classifier	0.7485	0.7911	0.5284	0.6811	0.5924	0.4150	0.4238	0.1940
nb	Naive Bayes	0.7427	0.7955	0.5702	0.6543	0.6043	0.4156	0.4215	0.0400
catboost	CatBoost Classifier	0.7410	0.7993	0.5278	0.6630	0.5851	0.4005	0.4078	0.0890
gbc	Gradient Boosting Classifier	0.7373	0.7918	0.5550	0.6445	0.5931	0.4013	0.4059	0.0770
ada	Ada Boost Classifier	0.7372	0.7799	0.5275	0.6585	0.5796	0.3926	0.4017	0.0870
et	Extra Trees Classifier	0.7299	0.7788	0.4965	0.6516	0.5596	0.3706	0.3802	0.1280
qda	Quadratic Discriminant Analysis	0.7282	0.7894	0.5281	0.6558	0.5736	0.3785	0.3910	0.0510
lightgbm	Light Gradient Boosting Machine	0.7133	0.7645	0.5398	0.6036	0.5650	0.3534	0.3580	0.2440
knn	K Neighbors Classifier	0.7001	0.7164	0.5020	0.5982	0.5413	0.3209	0.3271	0.0570
dt	Decision Tree Classifier	0.6928	0.6512	0.5137	0.5636	0.5328	0.3070	0.3098	0.0460
xgboost	Extreme Gradient Boosting	0.6853	0.7516	0.4912	0.5620	0.5216	0.2887	0.2922	0.0520
dummy	Dummy Classifier	0.6518	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0380
svm	SVM - Linear Kernel	0.5954	0.0000	0.3395	0.4090	0.2671	0.0720	0.0912	0.0410

Şekil 1.4 *Eğitilecek sistem parametreleri*

Deneysel ortam (*exp*) ayarlarına göre sistemin karşılaştırması için Liste 1.8'deki kodu çalıştırdığımızda Şekil 1.5'teki sonuç ile karşılaşırız. Burada üretilen *exp* isimli sınıflandırma ortamı varsayılan ortamdan ayrılmış ve ayarların değişikliği kobaylıkla varsayılan ilk ayarları etkilemeden gerçekleştirilebilir. Böylece iki farklı (*varsayılan* Şekil 1.4 ve *exp* Şekil 1.5) ortamda sınıflandırma algoritma performans karşılaştırması yapılabilir.

```
# Liste 1.8 Modellerin Deneysel ortam ayarlarına göre karşılaştırılması
exp.compare_models()
```

Liste 1.8 *Kütüphaneleri Yükleme*

Fonksiyonel ve OOP API arasındaki çıktının tutarlı olduğunu unutmayın. Bu not defterindeki diğer fonksiyonlar yalnızca fonksiyonel API kullanılarak gösterilecektir.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.7689	0.8047	0.5602	0.7208	0.6279	0.4641	0.4736	0.0450
ridge	Ridge Classifier	0.7670	0.0000	0.5497	0.7235	0.6221	0.4581	0.4690	0.0330
lda	Linear Discriminant Analysis	0.7670	0.8055	0.5550	0.7202	0.6243	0.4594	0.4695	0.0370
rf	Random Forest Classifier	0.7485	0.7911	0.5284	0.6811	0.5924	0.4150	0.4238	0.1320
nb	Naive Bayes	0.7427	0.7955	0.5702	0.6543	0.6043	0.4156	0.4215	0.0360
catboost	CatBoost Classifier	0.7410	0.7993	0.5278	0.6630	0.5851	0.4005	0.4078	0.0340
gbc	Gradient Boosting Classifier	0.7373	0.7918	0.5550	0.6445	0.5931	0.4013	0.4059	0.0730
ada	Ada Boost Classifier	0.7372	0.7799	0.5275	0.6585	0.5796	0.3926	0.4017	0.0750
et	Extra Trees Classifier	0.7299	0.7788	0.4965	0.6516	0.5596	0.3706	0.3802	0.1320
qda	Quadratic Discriminant Analysis	0.7282	0.7894	0.5281	0.6558	0.5736	0.3785	0.3910	0.0380
lightgbm	Light Gradient Boosting Machine	0.7133	0.7645	0.5398	0.6036	0.5650	0.3534	0.3580	0.0390
knn	K Neighbors Classifier	0.7001	0.7164	0.5020	0.5982	0.5413	0.3209	0.3271	0.0490
dt	Decision Tree Classifier	0.6928	0.6512	0.5137	0.5636	0.5328	0.3070	0.3098	0.0390
xgboost	Extreme Gradient Boosting	0.6853	0.7516	0.4912	0.5620	0.5216	0.2887	0.2922	0.0440
dummy	Dummy Classifier	0.6518	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0330
svm	SVM - Linear Kernel	0.5954	0.0000	0.3395	0.4090	0.2671	0.0720	0.0912	0.0310

```
Processing:   0%|          | 0/69 [00:00<?, ?it/s]
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=123, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Şekil 1.5 *Deneysel ortam ayarlarına göre algoritma performanslarının karşılaştırılması*

1.3 Model Analizi

Model eğitimi gerçekleştirildiğinde en başarılı algoritmanın başarısını ayrıntılı incelemek için performans kriterleri ile incelenmesi gerekmektedir.

Confusion matrix, ikili sınıflandırma (binary classification) problemlerinde modelin performansını değerlendirmek için kullanılan bir tablodur. Dört temel değerden oluşur (Şekil 1.6):

1. **True Positive (TP)**: Pozitif olarak tahmin edilip doğru olan örnekler.

2. **True Negative (TN)**: Negatif olarak tahmin edilip doğru olan örnekler.

3. **False Positive (FP)**: Pozitif olarak tahmin edilip yanlış olan (yanlış pozitif).

4. **False Negative (FN)**: Negatif olarak tahmin edilip yanlış olan (yanlış negatif).

Confusion matrix şöyle bir tablo ile gösterilebilir:

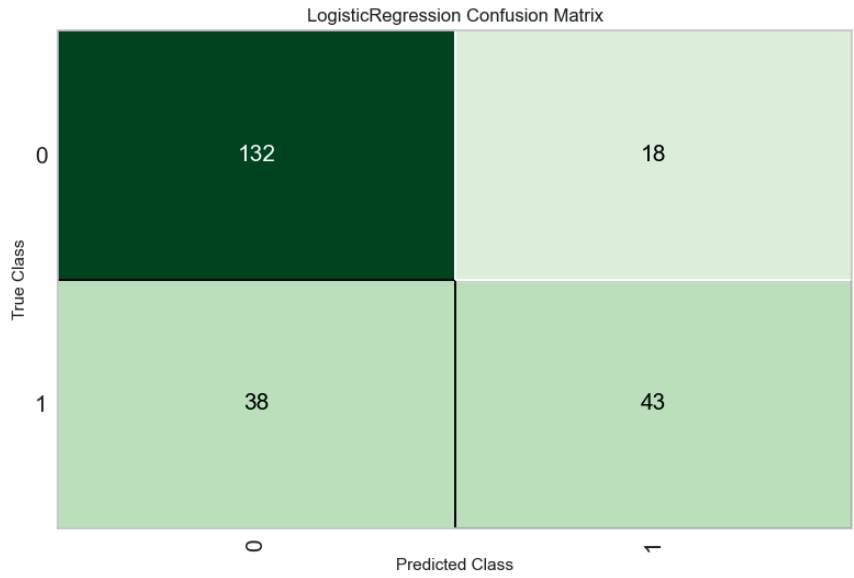
	Gerçek Pozitif	Gerçek Negatif
Tahmin Pozitif	TP	FP
Tahmin Negatif	FN	TN

Şekil 1.6 *Confusion matrix (karmaşıklık matrisi)*

En iyi performansı gösteren model Logistic Regression algoritmasıdır. Liste 1.7'deki kod ile Confusion matrix grafiği çizdirilmiştir. Temel ayarlar üzerinden geliştirilen bu modelin tahmin istatistiği Şekil 1.7'deki tabloda gösterilmiştir.

```
# Liste 1.9 confusion matrix'i en başarılı model için çiz
plot_model(best, plot = 'confusion_matrix')
```

Liste 1.9 *Kütüphaneleri Yükleme*



Şekil 1.7 *Logistik Regresyonun ürettiği karmaşıklık matrisi*

Confusion matrix değerlerine göre değerler aşağıda verilmiştir;

True Positive (TP) = 132

False Positive (FP) = 18

False Negative (FN) = 38

True Negative (TN) = 43

Bu değerlere göre Şekil 1.8'deki gibi bir yerleşimle gösterilebilir. Performans kriterleri Şekil 1.9'daki gibi hesaplanmaktadır. PyCaret bu hesapları çok hızlı ve Şekil 1.5'teki algoritmalar için otomatik yapmaktadır.

	Gerçek Pozitif	Gerçek Negatif
Tahmin Pozitif	132	18
Tahmin Negatif	38	43

Şekil 1.8 *Performans hesaplama değerlerinin Confusion matrix üzerinde gösterilmesi*

1. **Doğruluk (Accuracy)**:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{132 + 43}{132 + 43 + 18 + 38} = \frac{175}{231} \approx 0.757\,(75.7\%)$$

2. **Hassasiyet (Precision)**:

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{132}{132 + 18} = \frac{132}{150} \approx 0.88\,(88\%)$$

3. **Duyarlılık (Recall)**:

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{132}{132 + 38} = \frac{132}{170} \approx 0.776\,(77.6\%)$$

4. **F1 Skoru**:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{0.88 \times 0.776}{0.88 + 0.776} \approx 0.824\,(82.4\%)$$

Şekil 1.9 *Karmaşıklık matrisinden elde edilen performasn metrikleri ile hesaplanması*

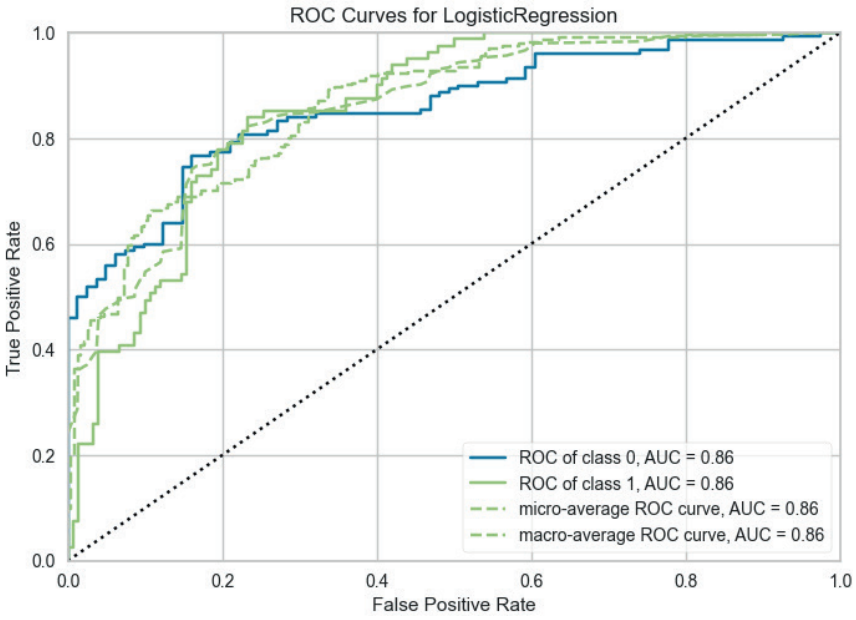
Bu metrikler (Şekil 1.9), modelin güçlü ve zayıf yönlerini anlamak için kullanılır. Örneğin, yanlış pozitiflerin kritik olduğu bir senaryoda **Precision**, yanlış negatiflerin kritik olduğu bir senaryoda ise **Recall** daha önemli olabilir.

Bir diğer performans kriteri de AUC (Area Under the Curve) eğrisidir. AUC, modelin pozitif örnekleri negatif örneklerden ne kadar iyi ayırabildiğini gösterir. Yüksek bir AUC değeri, modelin genelde iyi bir ayırma yeteneğine sahip olduğunu ifade eder. En yüksek değeri 1'dir ve %100 doğruluk anlamına gelir. Yapay zekâ tahminleme sistemlerinde %100'lük bir başarı aşırı doğruluğu ifade ettiği için doğru bir değer olarak kabul

edilmez ve ezberlerme (overfitting) olarak yorumlanır ancak 1'e yakın AUC değerleri çok başarılı sistemleri ifade eder. Liste 1.10'daki kodlarla bu eğri çizdirilerek incelenebilir.

```
# Liste 1.10 AUC
plot_model(best, plot = 'auc')
```

Liste 1.10 *AUC ve ROC çizdirme*



Şekil 1.8 *Logistik regresyonda karmaşıklık matrisi*

ROC Eğrisi (Receiver Operating Characteristic), modelin farklı sınıflandırma eşik değerlerinde, doğru pozitif oranı (True Positive Rate, TPR) ile yanlış pozitif oranı (False Positive Rate, FPR) arasındaki ilişkiyi görselleştiren bir grafiktir. ROC eğrisi, modelin pozitif ve negatif sınıfları ayırma becerisini görselleştirir. ROC ve AUC eğrileri beraber yorumlanmaktadır.

Feature Importance Plot, bir modelin tahmin yaparken hangi özelliklere (değişkenlere) ne kadar önem verdiğini görselleştirir. Bu grafik, modelin her bir özelliği (feature) nasıl değerlendirdiğini anlamak için kullanılır. Yatay eksen, özelliklerin modeldeki önem derecesini (variable importance) ifade eder. Bağımsız değişkenlerin (hastalığı teşhiste etkili olan öznitelikler) tahmin edilen değere olan etkisini göstermek için kullanılan

en pratik yöntemlerden birisi (Şekil 1.9) (feature importance) grafiğidir. Liste 1.11'deki kod kullanılarak çizdirilir.

```
# Liste 1.11 Öznitelik Grafiği
plot_model(best, plot = 'feature')
```

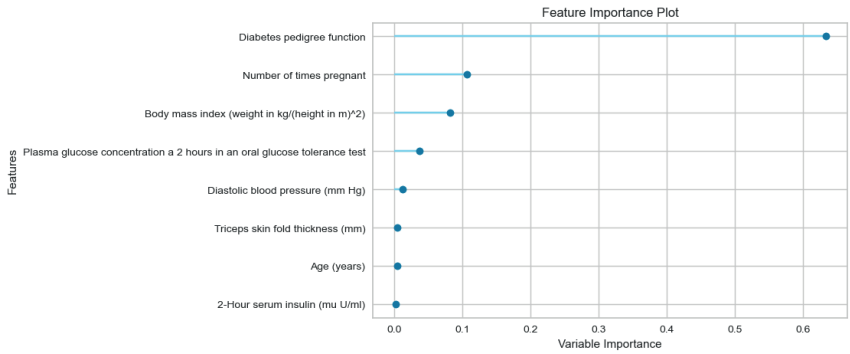
Liste 1.11 *Öznitelik Grafiği*

Feature Importance Plot (Özellik Önem Grafiğinin Önemi):

1. Özellik Seçimi: Daha az önemli özellikler, model karmaşıklığını azaltmak ve hızını artırmak için çıkarılabilir.

2. İş Problemi Anlayışı: Diyabet gibi sağlık sorunlarında hangi faktörlerin daha etkili olduğunu anlamamıza yardımcı olur.

3. Model Yorumlanabilirliği: Modelin tahminlerini nasıl yaptığına dair içgörü sağlar, özellikle sağlık gibi kritik alanlarda güvenilirlik artırır.



Şekil 1.9 *Özellik Önem (Feature Importance) Grafiği*

Şekil 1.19'daki özellik Önem grafiğini incelersek

- **En Önemli Özellikler:**

Diabetes Pedigree Function: En yüksek öneme sahip özelliktir (yaklaşık 0.6). Bu, diyabet riskinin hesaplanmasında genetik geçmişin güçlü bir etkisi olduğunu gösterir.

Number of Times Pregnant: İkinci en önemli özelliktir. Hamilelik sayısının diyabet gelişiminde etkili olduğu vurgulanmaktadır.

- **Orta Düzeyde Önemli Özellikler:**

Body Mass Index (BMI): Üçüncü sırada gelir ve diyabet riskinde önemli bir rol oynar. BMI, kişinin kilosunun boyuna göre orantısını temsil eder.

Plasma Glucose Concentration: Kan glikoz seviyesi de önemli bir faktördür, çünkü yüksek glikoz seviyeleri diyabetin bir göstergesi olabilir.

- **Daha Az Önemli Özellikler:**

Diastolic Blood Pressure, Triceps Skin Fold Thickness, Age, 2-Hour Serum Insulin: Bu özellikler modele daha az katkıda bulunmuştur. Ancak bu, tamamen önemsiz oldukları anlamına gelmez; sadece bu modelde diğer özelliklere kıyasla daha az etkili olmuşlardır.

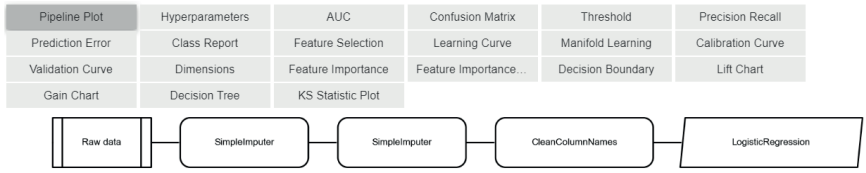
Sonuç olarak bu grafik kapsamında modelin en yüksek önem derecesine sahip özellikleri üzerinde daha derinlemesine analiz yapılabilirken, düşük önem taşıyan özelliklerin neden etkisiz kaldığı incelenerek veri setindeki eksiklikler veya gürültüler tespit edilebilir. Ayrıca, özelliklerin sıralaması, konu uzmanlarıyla birlikte değerlendirilerek modelin iş bağlamına uygunluğu doğrulanabilir. Yüksek sayıda öznitelik içeren veri setlerinde önem derecesi düşük öznitelikler çıkarılabilir. Ancak model performansına etkisi dikkatlice takip edilmelidir.

Jupyter-notebook kullanılarak performans karşılaştırması yapılmak istenirse Liste 1.12'de *plot_model* fonksiyonuna bir alternatif olan *evaluate_model* fonksiyonu kullanılabilir. Bu kod *ipywidget* gerektirdiği için ancak *JupyterNotebook* ortamında kullanılabilir.

```
# Liste 1.12 Farklı kriterlerle model hakkında grafikler çizdirme
evaluate_model(best)
```

Liste 1.12 Farklı Grafiklerle Performans İncelemesi

PyCaret'in *pipeline plotu*, bir makine öğrenimi modelinin oluşturulma sürecinde kullanılan tüm adımları görsel olarak sunan bir akış şemasıdır (Şekil 1.10). Bu plot sayesinde, veri ön işleme, özellik mühendisliği ve modelleme gibi aşamaların nasıl birbirini takip ettiği ve modelin nasıl oluşturulduğu net bir şekilde görülebilir.



Şekil 1.10 Modelin eğitim aşamaları

Pipeline'da Gösterilen Adımlar (Şekil 1.10)

- ***Veri Ön İşleme (Raw Data):*** Ham veriyi okunarak *jupyter-notebook* ortamın alınması anlamına gelmektedir. Modelin eğitimi için verilerin hazır hale getirildiği aşamadır. Eksik verilerin tamamlanması, kategorik verilerin sayısal verilere dönüştürülmesi (örneğin, *One-Hot Encoding*), verilerin belirli bir aralıkta ölçeklenmesi gibi işlemler bu aşamada yapılır.

- ***Özellik Mühendisliği (Simple Imputer, CleanColumnNames):*** Modelin performansını artırmak için verideki özelliklerin işlendiği ve yeni özellikler oluşturulduğu aşamadır. Mevcut özelliklerden yeni özellikler türetilmesi veya önemsiz özelliklerin elenmesi gibi işlemler bu kapsamdadır.

- ***Modelleme (Logistic Regression):*** Seçilen makine öğrenimi algoritmasının veriler üzerinde eğitildiği aşamadır. Modelin parametrelerinin ayarlanması ve en iyi performansı elde etmek için optimizasyon işlemleri de bu aşamada gerçekleştirilir.

- ***Değerlendirme*****:** Eğitilen modelin performansının değerlendirildiği aşamadır. Modelin doğruluğu (accuracy) ve hassasiyeti (recall) gibi metrikler hesaplanarak modelin başarısı bu aşamada ölçülür.

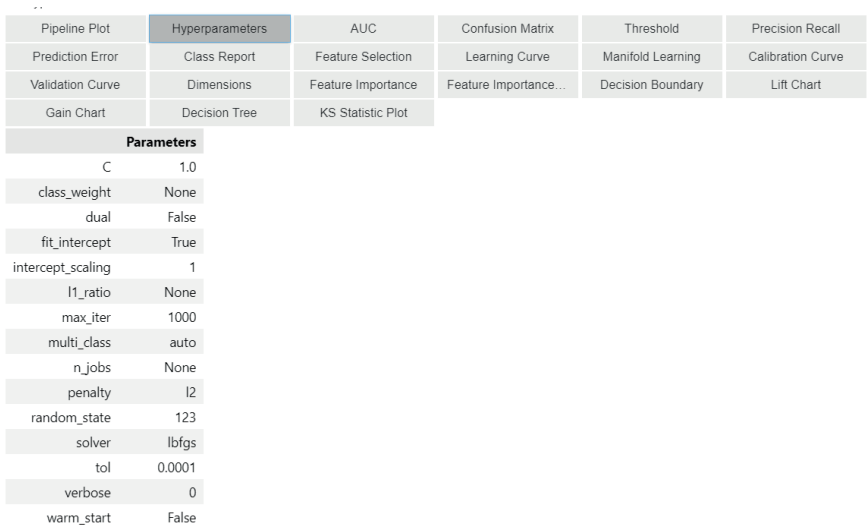
Bu adımlar sayesinde, *PyCaret* kullanıcılara model oluşturma sürecinin her aşamasında şeffaflık sağlar ve modelin performansını analiz etme imkânı sunar.

Logistic Regression Modelinin Parametreleri (Hyperparameters)

En başarılı model olarak seçilen *LogisticRegression* modeli için parametreler varsayılan olarak Şekil 1.11'de sunulmuştur. Bu parametreler modelin nasıl çalışacağına dair önemli detaylar içermektedir. Bu parametrelerin ne anlama geldiğine daha yakından bakalım:

- **c:** Bu parametre, modelin ne kadar karmaşık olacağını belirler. Düşük bir C değeri daha basit bir model, yüksek bir C değeri ise daha karmaşık bir model anlamına gelir.

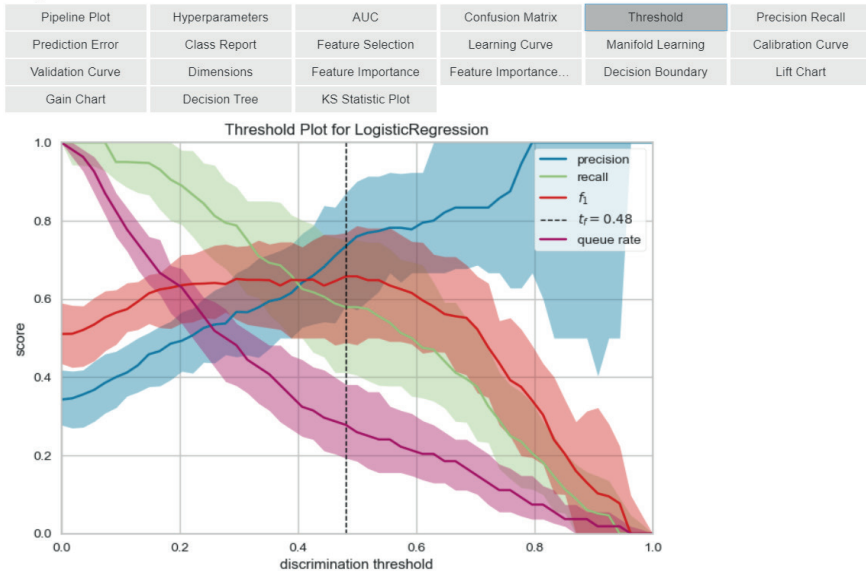
- **class_weight:** Eğer veri setinde bazı sınıflar diğerlerinden daha azsa, bu parametre modelin bu dengesizliği telafi etmesine yardımcı olur.
- **penalty:** Modelin karmaşıklığına bir sınırlama getirmek için kullanılan bir düzenleme terimidir. L1 veya L2 gibi farklı düzenleme türleri olabilir.
- **solver:** Modelin optimizasyon için kullanacağı algoritmayı belirler.
- **max_iter:** Modelin en iyi çözümü bulmak için yapacağı en fazla yineleme sayısıdır.
- **random_state:** Eğer model tekrar çalıştırılırsa, sonuçların her zaman aynı olması için kullanılan bir rastgele sayı üreteci tohumudur.



Şekil 1.11 *Modelin eğitiminde kullanılan algoritmanın eğitim parametreleri*

Böylece, lojistik regresyon modelinin nasıl eğitildiğini ve hangi özelliklere odaklandığını daha ayrıntılı bir şekilde inceleme fırsatımız olmaktadır. Modelin performansı, bu parametrelerin doğru bir şekilde ayarlanmasına bağlıdır. Ancak bu parametrelerin tam olarak ne anlama geldiğini ve hangi değerlerin model için en uygun olduğunu anlamak için daha derinlemesine bir makine öğrenimi bilgisi gerekebilir.

AUC ve Confusion Matrix anlatıldığı için Treshold grafiğini incelemeye geçerek Şekil 1.12'ye dikkatli ve bakalım.



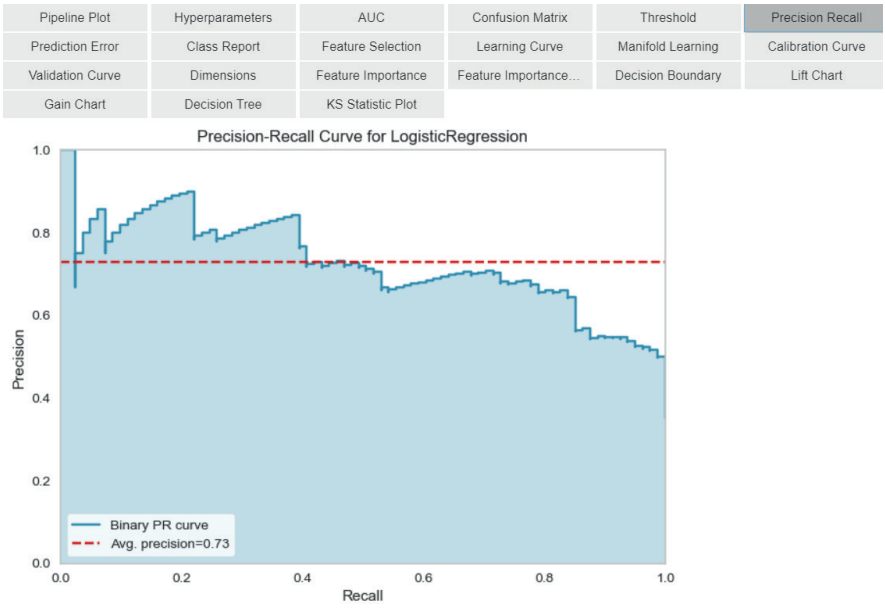
Şekil 1.12 *Modelin Treshold eğitiminde kullanılan algoritmanın eğitim parametreleri*

PyCaret'in **Threshold Plot**'u, önemli bir rehberdir (Şekil 1.12). Modelin ikili sınıflandırma problemlerinde farklı sınıflandırma eşik (*threshold*) değerlerinin model performansına etkisi görselleştirmek ve optimize etmek için önemlidir. *Logistic Regression* gibi olasılık tabanlı modellerde, tahminlerin bir sınıfa atanması için bir eşik değeri kullanılır (örneğin, 0.5).

Threshold Değişimi: Eşik değeri arttıkça (örneğin 0.5'ten 0.7'ye), *Precision* genellikle artar, ancak *Recall* düşer. Eşik değeri azaldıkça (örneğin 0.3'e), *Recall* artar, ancak *Precision* azalabilir.

Denge Seçimi: *F1 Score* veya uygulama bağlamındaki gereksinimlere göre bir eşik değeri seçilir. Örneğin, yanlış negatiflerin kritik olduğu bir durum için daha düşük eşik değeri seçilebilir ancak *Recall* değerinin düşmemesine dikkat edilmelidir. Recall değerinin önemli olduğu unutulmamalıdır. Bu grafik (Şekil 1.12'de), 0.5 eşik (treshold) bölgesinde doğru eşik değerini seçmek için optimize edilmiştir.

Özellikle dengesiz veri setlerinde (örneğin, pozitif ve negatif sınıfların sayısı arasında büyük fark olduğu) model performansını değerlendirmede önemli bir grafik türü de Precision-Recall Curve (Hassasiyet-Duyarlılık Eğrisi) (Şekil 1.13)'dir. Bu grafik, bir sınıflandırma modelinin *Precision* (hassasiyet) ve *Recall* (duyarlılık) değerleri arasındaki dengeyi görselleştirir (Şekil 1.13).



Şekil 1.13 *Hassasiyet-Duyarlılık Eğrisi (Precision-Recall Curve)*

Precision-Recall Curve (Hassasiyet-Duyarlılık Eğrisi)'nden genel olarak:

1. **Dengesiz Veri Setleri:** Precision-Recall Curve, özellikle sınıf dengesizliği durumlarında, ROC eğrisine kıyasla daha iyi performans ölçümü sağladığı,

2. **Precision-Recall Dengesi:** Grafik, farklı threshold (eşik) değerlerinde Precision ve Recall (PR) arasında nasıl bir değişim olduğunu gösterir. Modelin yanlış pozitiflere veya yanlış negatiflere olan toleransına bağlı olarak, uygulamaya uygun bir denge seçilebileceği,

3. **Ortalama Precision (Average Precision):** Grafikteki eğrinin altında kalan alan (AUC-PR), modelin genel performansını ölçer. AUC-PR değeri ne kadar yüksekse, modelin pozitif sınıfları doğru tahmin etme yeteneği o kadar iyi olduğu öğrenilir.

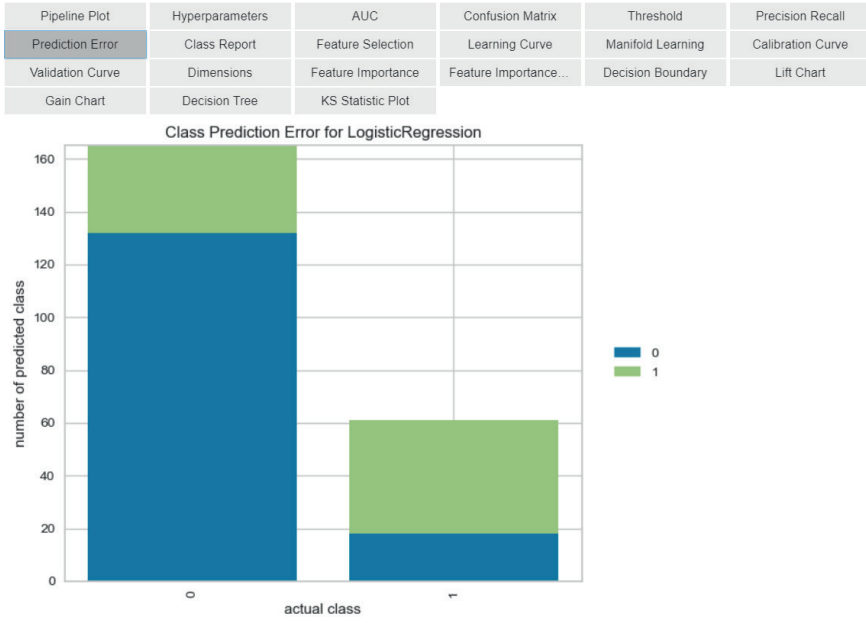
Şekil 1.13'teki grafiği dikkatlice yorumlarsak;

Eğri: Grafik, Precision ve Recall arasındaki ilişkiyi gösterir. Başlangıçta Precision yüksekken (yaklaşık 0.8), Recall arttıkça Precision düşmektedir. Bu, modelin daha fazla pozitif örnek tahmin etmeye çalıştıkça yanlış pozitiflerin arttığını işaret eder.

Ortalama Precision (Average Precision = 0.73): Kırmızı kesikli çizgi, modelin ortalama hassasiyet değerini ifade eder. Ortalama 0.73, mo-

delin genelde pozitif tahminlerde başarılı olduğunu gösterir, ancak mükemmel değildir.

Denge Noktası: Grafik boyunca Precision ve Recall arasında bir denge noktası aranabilir. Örneğin, uygulama bağlamında Precision'ın mı, yoksa Recall'un mu daha önemli olduğu belirlenerek bir eşik seçilebilir.



Şekil 1.14 *Sınıf Tahmin Hata grafiği*

Class Prediction Error(Sınıf Tahmin Hata**) Grafiği (Şekil 1.14)**, bir modelin sınıfları tahmin ederken yaptığı hataları görselleştiren bir grafiktir. Bu grafik, modelin hangi sınıflarda başarılı veya başarısız olduğunu analiz etmeye yardımcı olur.

Grafik, modelin tahmin performansını ve sınıf tahminlerindeki hataları görselleştirmektedir. Mavi renk, modelin 0 sınıfını (negatif) doğru veya yanlış tahmin ettiği örnekleri, yeşil renk ise 1 sınıfını (pozitif) temsil eder. X ekseni gerçek sınıfları (0 ve 1) gösterirken, tahmin hataları çubuklardaki yanlış renk bölgeleriyle belirtilmiştir. Gerçek sınıf "0" için modelin doğru tahmin oranı yüksek olmakla birlikte bazı yanlış pozitif tahminler yapılmıştır. Öte yandan, gerçek sınıf "1" için doğru tahmin sayısı daha sınırlı olup, yanlış negatif tahminler daha belirgin şekilde öne çıkmaktadır. Bu durum, modelin 1 sınıfını tahmin etmede zorlandığını göstermektedir.

Özetle Şekil 1.14'te Sınıf Tahmin Hata grafiği modelin özellikle **1 sınıfını (pozitif)** tahmin etmede **zorlandığını** ve bu nedenle **yanlış negatiflerin** fazla olduğunu gösteriyor. **Gerçek sınıf 0** için ise model genel olarak **başarılıdır**, ancak bazı yanlış pozitifler bulunuyor.

Şekil 1.15 *Logistic Regression (Lojistik Regresyon) sınıflandırma Raporu*

Logistic Regression sınıflandırma raporu (Şekil 1.15), modelin performansını değerlendirmek için yaygın olarak kullanılan performans metriklerini görselleştirmektedir. Sınıflandırma raporu, her bir sınıf (1 ve 0) için aşağıdaki metrikleri içerir:

1. **Precision (Doğruluk)**:

- Bir sınıfa ait olarak tahmin edilen örneklerin ne kadarının doğru olduğunu gösterir.

- Örneğin, sınıf 1 için doğruluk 0.705'tir, yani sınıf 1 olarak tahmin edilen örneklerin %70.5'i doğru sınıflandırılmıştır.

2. **Recall (Duyarlılık)**:

- Gerçek pozitiflerin ne kadarının doğru tahmin edildiğini ifade eder.

- Sınıf 1 için duyarlılık değeri 0.531'dir, yani gerçek sınıf 1 örneklerinin sadece %53.1'i doğru tahmin edilmiştir.

3. **F1-Score**:

- Precision ve Recall'un harmonik ortalamasıdır. Dengeli bir performans metriğidir.
- Sınıf 1 için F1 skoru 0.606'dır.

4. **Support (Destek)**:

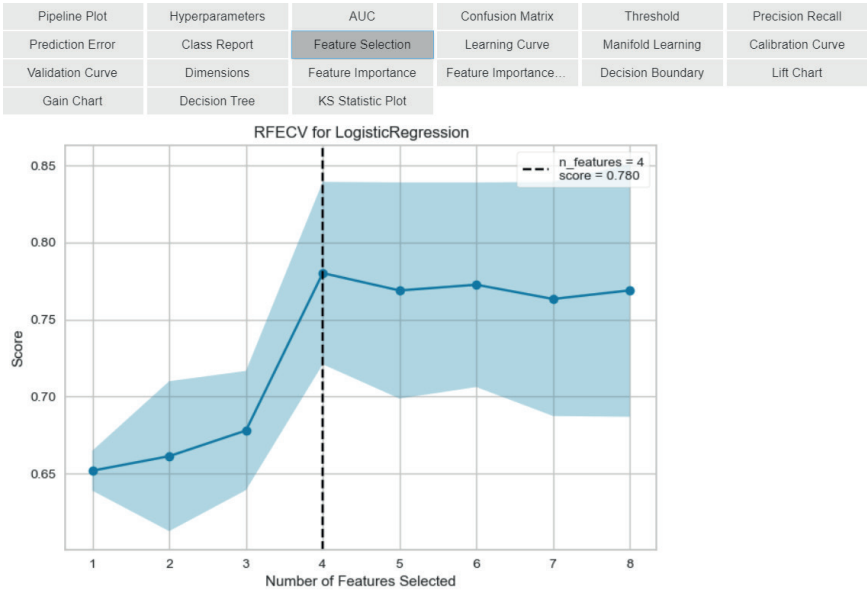
- Her bir sınıfın veri kümesinde kaç örnek içerdiğini gösterir.
- Örneğin, sınıf 1 in destek değeri 81, sınıf 0'ın destek değeri ise 150'dir.

Grafik, bu metriklerin her birinin performansını farklı renk tonlarıyla ifade etmektedir. Daha koyu renkler, yüksek bir performansı; daha açık renkler ise düşük bir performansı simgeler.

Lojistik Regresyon modelinin başarısını genel olarak değerlendirirsek:

- Model sınıf 0'ı daha iyi bir doğruluk (accuracy) (0.776) ve duyarlılık (recall) (0.880) ile tahmin ederken, sınıf 1 için bu değerler görece düşüktür (precision: 0.705, recall: 0.531).
- Bu dengesizlik, modelin sınıf 0'a (muhtemelen daha baskın sınıfa) karşı daha yüksek bir eğilim gösterdiğini ve sınıf 1'i yeterince iyi ayırt edemediğini gösterir.
- Bu durum, sınıf dengesizliği veya modelin sınıf 1 için daha fazla özellik öğrenememesinden kaynaklanabilir.

Bu tür bir analiz, özellikle sınıflandırma modelinin optimize edilmesi gerektiğinde oldukça faydalıdır. Örneğin, sınıf 1 için duyarlılık artırılabilir veya veri dengesizliği azaltılabilir.



Şekil 1.16 *Recursive Feature Elimination with Cross-Validation (RFECV)*

Recursive Feature Elimination with Cross-Validation (RFECV) (Çapraz Doğrulama ile Tekrarlayan Özellik Eleme) yöntemi kullanılarak Şekil 1.16'daki grafik bir lojistik regresyon modeli için özellik seçim sürecini göstermektedir. Bu yöntem, model performansını optimize etmek için gereksiz özellikleri elimine ederken en iyi performansı sağlayan özellik setini belirler.

Grafik Özellikleri ve Yorumlanmasını şu şekilde yapabiliriz:

1. **Yatay Eksen (Number of Features Selected)**:

- Seçilen özellik sayısını gösterir. Grafikte 1'den 8'e kadar değişen özellik sayıları yer almakta.

2. **Dikey Eksen (Score)**:

- Seçilen özellik sayısına göre modelin doğrulama setindeki skorunu ifade eder. Genellikle bu skor, modelin tahmin başarısını ölçen bir metrik (örneğin doğruluk) olarak hesaplanır.

3. **Mavi Çizgi ve Gölgelik Alan**:

- Mavi çizgi, her bir özellik sayısı için ortalama model skorunu gösterir.

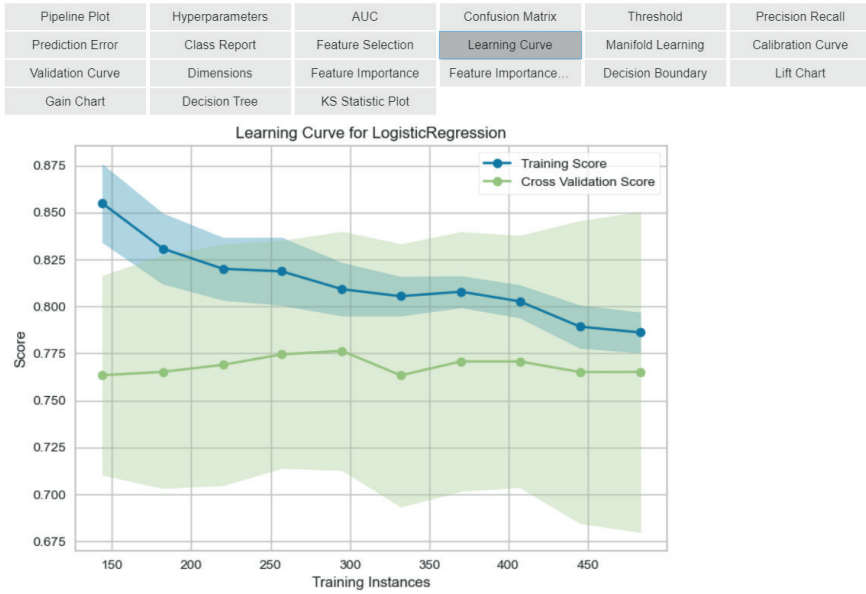
- Gölgelik alan, skorun varyansını (standart sapmayı) ifade eder. Gölgelik alanın geniş olduğu bölgelerde sonuçlar daha az kararlıdır.

4. **Kesikli Siyah Çizgi (Optimal Nokta)**:

- RFECV tarafından seçilen **optimal özellik sayısını** gösterir. Bu örnekte, en iyi performansın **4 özellik ile** elde edildiği belirlenmiştir.

- Bu noktada model skoru yaklaşık **0.780**'dir.

Grafik (1.16), model performansının başlangıçta seçilen özellik sayısı arttıkça yükseldiğini, ancak belirli bir noktadan sonra sabitlendiğini (veya azalmaya başladığını) göstermektedir. Optimal özellik sayısı (4 özellik), daha fazla özellik eklenmesinin modelin performansına anlamlı bir katkı sağlamadığı bir noktadır. Bu, gereksiz özelliklerin modelin aşırı öğrenme (overfitting) veya hesaplama karmaşıklığını artırabileceğini gösterir. Önemli özellikleri belirlemek ve gereksiz olanları elemine ederek daha verimli bir model elde etmek için RFECV etkili yöntemlerden birisidir.



Şekil 1.17 *Lojistik regresyon modeli için bir öğrenme eğrisi (Learning Curve)*

Lojistik regresyon modeli için bir **öğrenme eğrisi (Learning Curve)** Şekil 1.17'de gösterilmiştir. Öğrenme eğrileri, modelin eğitim verisi ve çapraz doğrulama (validation) üzerindeki performansını eğitim örneklerinin sayısına göre görselleştirir.

Grafiğin genel özellikleri:

1. **Yatay Eksen (Training Instances)**:

- Eğitim veri setinde kullanılan örnek sayısını ifade eder. Örnek

sayısı arttıkça modelin performansı nasıl değişiyor, bu eksende izlenir.

2. **Dikey Eksen (Score)**:

- Modelin doğruluk veya başka bir değerlendirme metriği üzerindeki başarısını gösterir.

3. **Mavi Çizgi (Training Score)**:

- Modelin eğitim verisindeki performansını (hata oranı düşük, doğruluk yüksek) gösterir.

- Eğitim verisine tam erişim sağlandıkça skor biraz düşüş göstermiş ve sonrasında dengelenmiştir.

4. **Yeşil Çizgi (Cross Validation Score)**:

- Modelin çapraz doğrulama veri setindeki performansını gösterir. Gerçek performansı temsil eder.

- Daha stabil bir eğri olup, belirli bir eğitim örnek sayısından sonra genellikle sabitlenmiştir.

5. **Gölgeli Alanlar**:

- Her iki skor için varyansı ifade eder. Gölgenin geniş olduğu bölgelerde modelin performansı daha değişkendir.

Bu grafik incelendiğinde şu sonuçlara varılabilir;

1. **Eğitim Skoru ve Doğrulama Skoru Arasındaki Fark (High Bias ve High Variance)**:

- Eğitim skoru (mavi) ile doğrulama skoru (yeşil) arasında bir boşluk bulunmaktadır. Bu durum, modelin **overfitting** (aşırı öğrenme) riski taşımadığını ancak **underfitting** (eksik öğrenme) eğiliminde olduğunu gösterebilir. Model yeterince karmaşık olmayabilir veya veri sayısı yetersizdir.

2. **Eğitim Örnek Sayısının Etkisi**:

- Eğitim örneklerinin başlangıçta artışı, çapraz doğrulama skorunu iyileştirmiştir. Ancak yaklaşık 300 örnekten sonra doğrulama skoru sabitlenmiştir. Daha fazla veri eklemek model performansını artık önemli ölçüde artırmamaktadır.

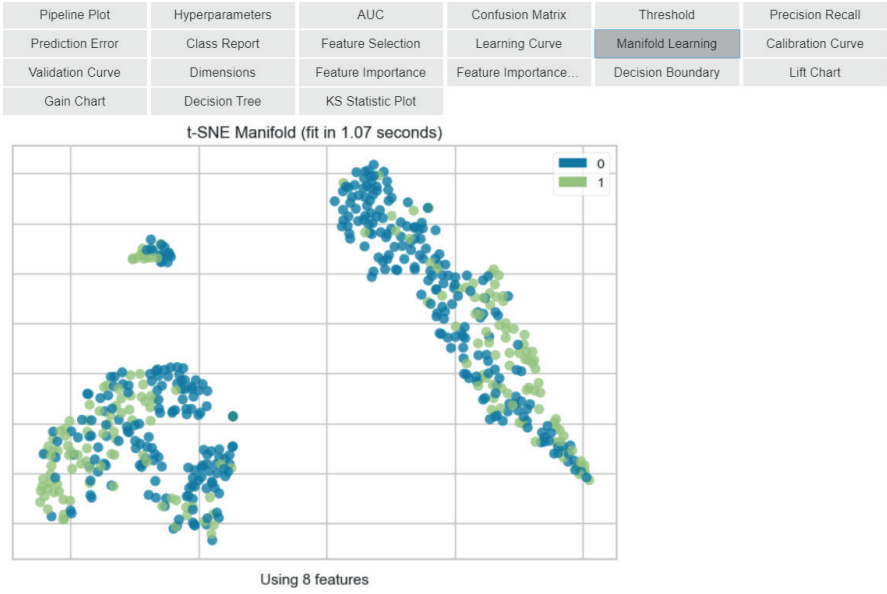
3. **Modelin Kararlılığı**:

- Eğitim ve doğrulama skorlarının birbirine paralel yaklaşması, modelin kararlı bir hale geldiğini ifade eder.

Şekil 1.17'deki grafiğe göre; modelin mevcut verilerle kararlı bir performansa ulaştığı görülmektedir. Ancak doğrulama skoru maksimum 0.775 seviyesinde sabitlenmiştir, bu da modelin genelleme kapasitesini artırmak (**daha başarılı performans göstermesi**) için:

- **Daha karmaşık bir model seçilmesi,**
- **Özellik mühendisliği yapılması,**
- **Veri setinin artırılması (farklı özelliklerle zenginleştirilmesi)**

gibi iyileştirmelerin gerekli olabileceğini göstermektedir.



Şekil 1.18 *t-SNE Manifold Öğrenme Grafiği*

Manifold eğrisi grafiği t-SNE (t-Distributed Stochastic Neighbor Embedding) algoritması kullanılarak verilerin 2 boyutlu uzayda görselleştirildiğini gösteriyor (Şekil 1.18). t-SNE, yüksek boyutlu verilerin daha düşük boyutlu bir uzaya indirgenmesini sağlar ve genellikle veri kümelerindeki örüntüleri veya kümeleri görselleştirmek için kullanılır. Bu grafikte (Şekil 1.18) veriler sınıflandırılırken iki sınıftan hangisine nasıl dağıldığı renkli noktalarla gösterilmiştir.

Grafiği yorumlandığında;

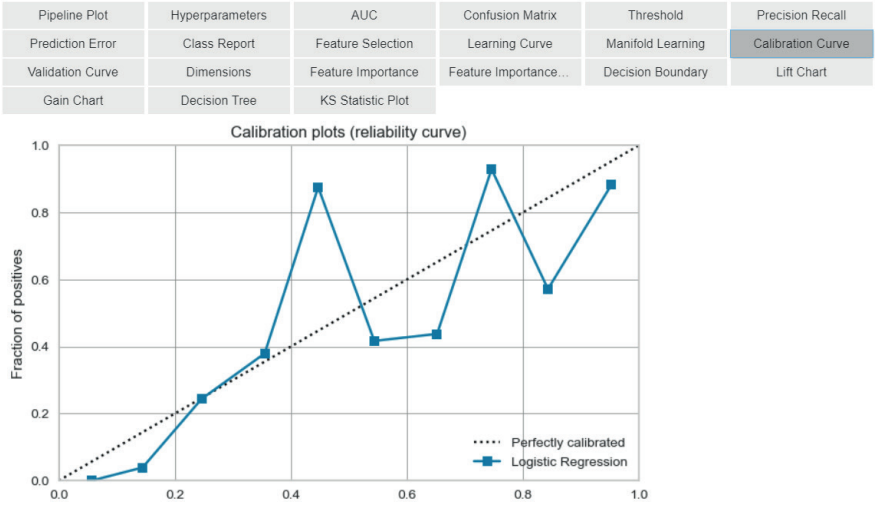
1. **Renkler ve Sınıflar**: Grafikte iki farklı sınıf (mavi: 0, yeşil: 1) gösterilmiştir. t-SNE, bu iki sınıfı kısmen ayrıştırabildiği,

2. **Kümeler**: Veriler üç farklı kümeye ayrılmış durumda. Sağ üstteki

kümelenme büyük ölçüde mavi (sınıf 0) ağırlıklı, sol alttaki küme ise karışık (hem sınıf 0 hem de sınıf 1) olduğu ve

3. **Sınıf Ayrımı**: Bazı bölgelerde sınıflar birbirine daha fazla karışmış, bu da bazı özelliklerin sınıfları ayırmada yeterince güçlü olmadığı görülmüştür.

Bu görselleştirme (Şekil 1.18), veri kümesindeki farklı sınıfların yapısal dağılımını anlamaya yardımcı olur ve özellikle kümeler veya benzerlikler hakkında bilgi verir. Modelin ayırma performansı veya özelliklerin sınıflar üzerindeki etkisi hakkında fikir sahibi olunabilir.



Şekil 1.19 *Kalibrasyon grafikleri (reliability curve)*

Kalibrasyon grafikleri **(reliability curve)** (Şekil 1.19) bir modelin tahmin olasılıklarının ne kadar güvenilir (kalibre edilmiş) olduğunu anlamak için kullanılır. Yorumlanırken odaklanılması gereken nokta, modelin tahmin ettiği olasılıkların ne kadar "güvenilir" olduğudur.

Grafiği analiz edersek;

1. **Eksenler**:

X Ekseni: Modelin tahmin ettiği olasılıklar.

Y Ekseni: Gerçek pozitif oranı (gözlemlerden elde edilen gerçek değerler).

2. Düz Çizgi (Kesikli Çizgi):

Mükemmel Kalibrasyon: Tahmin edilen olasılıklar ile gerçek pozitif oranlar tamamen eşleşiyorsa, noktalar bu çizgi üzerinde yer alır.

3. Mavi Çizgi (Logistic Regression):

Bu çizgi, lojistik regresyon modelinin tahminlerinin kalibrasyon durumunu gösterir.

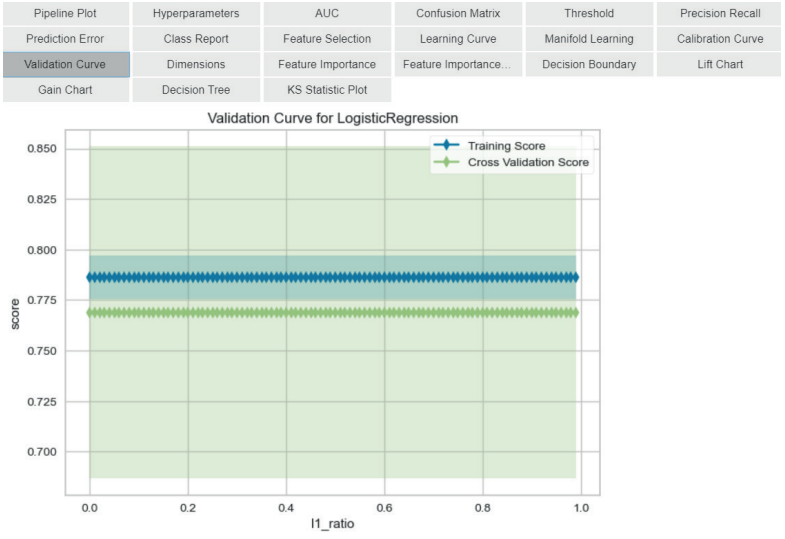
Düşük Olasılık Bölgeleri (0.0-0.4): Modelin tahminleri düşük olasılıkları genellikle hafife alıyor. Pozitif oranlar daha düşük kalmış.

Orta Olasılık Bölgeleri (0.4-0.6): Burada modelin tahminleri dengesiz. Özellikle bir noktada (0.4 civarında) gerçek pozitif oranı çok yüksek çıkmış (aşırı kalibrasyon sapması).

Yüksek Olasılık Bölgeleri (0.8-1.0): Modelin tahminleri bu bölgede nispeten daha tutarlı ancak yine de gerçek pozitif oranlarında dalgalanmalar var.

Lojistik regresyon modeli Şekil 1.18'de görüldüğü üzere mükemmel kalibrasyondan sapmalar göstermektedir. Özellikle düşük ve orta olasılık tahminlerinde tutarsızlık gözleniyor.

Bu durum, modelin tahminlerinin güvenilirliği üzerinde çalışılması gerektiğini gösteriyor. Örneğin, **Platt Scaling** veya **Isotonic Regression** gibi kalibrasyon yöntemleri uygulanarak modelin tahminleri iyileştirilebilir. Kalibrasyon hatalarını azaltarak modelin performansı ve güvenilirliği artırılabilir.



Şekil 1.20 *Doğrulama Eğrisi (***Validation Curve***) grafiği*

Bu **Validation Curve** grafiği (Şekil 1.20), lojistik regresyon modeli için **L1 oranı** (l1_ratio) parametresinin, **eğitim skoru** (Training Score) ve **çapraz doğrulama skoru** (Cross Validation Score) üzerindeki etkisini göstermektedir.

Ayrıntılı bir şekilde grafiğin yorumlarsak:

1. **Eksenler**:

- **X Ekseni**: l1_ratio (L1 ve L2 düzenleme arasındaki dengeyi belirler). Değerler 0.0 (L2 düzenleme) ile 1.0 (L1 düzenleme) arasında değişiyor.
- **Y Ekseni**: Modelin performans skoru (örneğin, doğruluk-accuracy).

2. **Training Score ve Cross Validation Score**:310

- **Mavi Çizgi (Training Score)**: Modelin eğitim seti üzerinde performansını gösteriyor.
- **Yeşil Çizgi (Cross Validation Score)**: Modelin çapraz doğrulama (validation) setindeki performansını gösteriyor.

3. **Eğitim ve Çapraz Doğrulama Arasındaki Fark**:

- Grafikte, **eğitim skoru** ile **çapraz doğrulama skoru** arasında küçük bir fark mevcut.
- Bu durum, modelin aşırı öğrenme (**overfitting**) eğiliminde olmadığını gösteriyor çünkü iki skor arasındaki fark büyük değil.

4. **L1_ratio Etkisi**:

- l1_ratio parametresindeki değişime rağmen, hem **training** hem de **cross validation** skorları sabit kalıyor.
- Bu durum, L1 ve L2 düzenlemeleri arasındaki denge değişse bile model performansının fazla etkilenmediğini göstermektedir.
- Bu, veriye özgü bir durum olabilir: Modelin **düzenlemeye** karşı duyarlılığı düşük olabilir veya veri seti yeterince dengeli olabilir.

Genel Değerlendirme:

- **Model Performansı**: Model genel olarak stabil bir performans sergiliyor.
- **Overfitting veya Underfitting Sorunu**: Grafik, aşırı öğrenme (overfitting) ya da yetersiz öğrenme (underfitting) belirtisi göstermiyor.

- **L1/L2 Düzenleme Etkisi**: L1 ve L2 düzenleme oranı değiştirilse bile performans skorlarında kayda değer bir değişiklik olmuyor. Bu durum, modelin hiperparametre ayarı açısından oldukça dengeli olduğunu gösteriyor.

Bu tür bir grafikte **daha fazla düzenleme (regularization)** gerektiğini veya l1_ratio'nun performansı etkilemediğini görüyorsanız, alternatif hiperparametre optimizasyonlarına odaklanabilirsiniz.

Karar sınırı (decision boundary) grafiği (Şekil 1.21) ile ikili sınıflandırmayı görselleştirmektedir. İki farklı sınıf arasındaki ayrım, iki özellik (**Feature One** ve **Feature Two**) temelinde modellenmiştir.

Grafiği daha yakından incelersek:

1. **Renkler**:

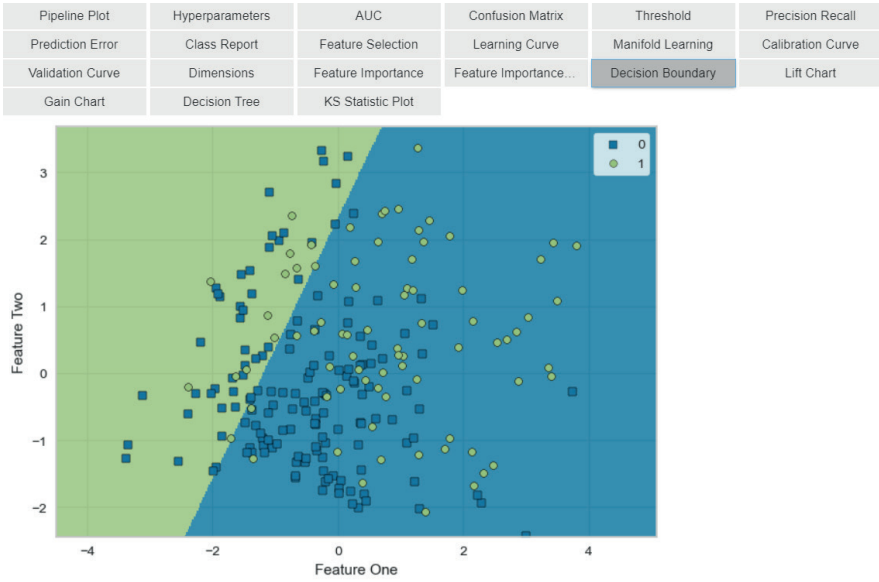
- **Mavi Alan**: Modelin **sınıf 0** olarak tahmin ettiği bölgeleri temsil eder.
- **Yeşil Alan**: Modelin **sınıf 1** olarak tahmin ettiği bölgeleri temsil eder.

2. **Noktalar**:

- **Mavi Kareler**: Gerçek sınıfı **0** olan örnekler.
- **Yeşil Daireler**: Gerçek sınıfı **1** olan örnekler.

3. **Karar Sınırı**:

- **Mavi ve Yeşil Alanların Arasındaki Eğik Çizgi**: Modelin iki sınıfı birbirinden ayırdığı karar sınırını temsil eder.
- Bu çizginin sağ tarafında kalan bölgeler çoğunlukla **sınıf 0**'a ait, sol tarafında kalan bölgeler ise **sınıf 1**'e ait olarak tahmin edilmiştir.



Şekil 1.21 *Karar Sınırı (decision boundary) Grafiği*

Bu durumu modelleme performansı açısından yorumlarsak;

1. **Doğru ve Yanlış Sınıflandırmalar**:

- **Yanlış Sınıflandırılan Örnekler**: Mavi bölgede bulunan yeşil daireler ve yeşil bölgede bulunan mavi kareler, modelin hatalı sınıflandırdığı gözlemlerdir.

- Özellikle karar sınırına yakın noktalarda hatalı sınıflandırma oranı artmaktadır.

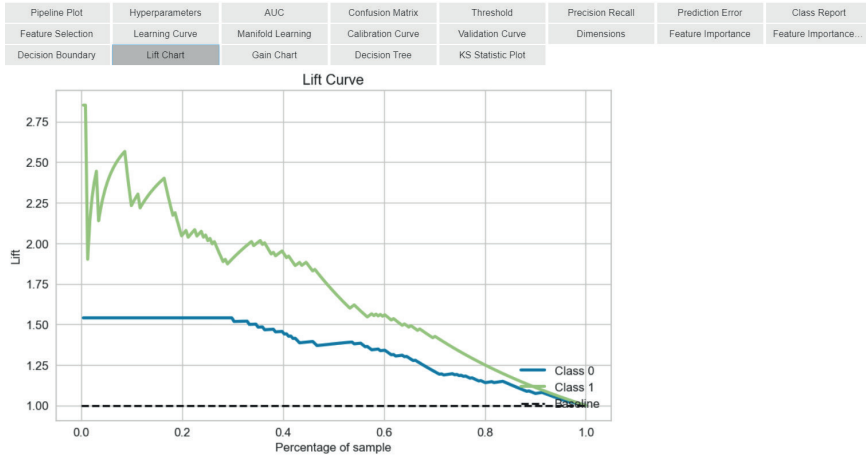
2. **Karar Sınırının Lineerliği**:

- Karar sınırı **lineer (doğrusal)** bir çizgi olarak görünüyor. Bu durum, modelin muhtemelen **lojistik regresyon** gibi doğrusal bir sınıflandırıcı kullandığını gösterir.

3. **Sınıf Ayrımı**:

- Veri setindeki iki sınıf arasındaki ayrım mükemmel değil; bazı bölgelerde sınıflar iç içe geçmiş durumda. Bu nedenle modelin bazı örneklerde hata yapması beklenen bir durumdur.

Karar sınırının doğruluğu, verinin doğasına ve modelin sınıflandırma yeteneğine bağlı olduğu unutulmamalıdır. **Sınıfların iyi ayrıştırılamadığı düşünülüyorsa**, doğrusal olmayan modeller (örneğin, **SVM RBF çekirdeği**, **ağaç tabanlı yöntemler**) tercih edilebilir.



Şekil 1.22 *Lift Eğrisi grafiği*

Lift eğrisi (Şekil 1.22), bir sınıflandırma modelinin, rastgele tahmin yapmaya göre ne kadar daha iyi bir performans sergilediğini görselleştirerek modelin sınıflandırma başarısını değerlendirmeye yönelik bir grafiksel yöntemdir. Özellikle ikili sınıflandırma problemlerinde, modelin iki farklı sınıfı (örneğin, evet/hayır, spam/spam değil) ne kadar iyi ayırt edebildiğini gösterir. Bu eğri sayesinde, modelin hangi seviyede doğru tahminler yaptığını ve hangi seviyede yanlış pozitif veya yanlış negatif sonuçlar ürettiğini daha net bir şekilde anlamak mümkün olur.

Grafiğin genel yorumunu yaparsak:

1. **Eksenler**:

X Ekseni: Örneklerin yüzdesi (örneğin en yüksek olasılık değerinden başlayarak örneklerin sıralanması).

Y Ekseni: Lift değeri (rastgele tahmine kıyasla iyileşme oranı).

2. **Renkler**:

Yeşil Çizgi (Class 1): Sınıf 1 için modelin performansı.

Mavi Çizgi (Class 0): Sınıf 0 için modelin performansı.

Siyah Kesikli Çizgi: Rastgele tahmin için baz çizgi (Lift = 1).

3. **Yüksek Lift Değerleri**:

Grafikte görüldüğü gibi **Class 1** (yeşil çizgi), başlangıçta oldukça yüksek lift değerlerine sahip. Bu durum, modelin sınıf 1 örneklerini başarılı bir şekilde ayırt ettiğini gösterir.

Class 0 (mavi çizgi) ise **daha düşük lift değerleri** ile daha ılımlı bir performans sergiliyor.

4. Lift Zamanla Azalır:

Örnekler sıralandıkça lift değeri azalmaktadır. Bu beklenen bir durumdur çünkü model, en başta **en güvenilir tahmin edilen örnekleri** seçer.

Grafiğin sağ tarafında lift değeri **1'e yaklaşır**. Bu noktada modelin tahmini rastgele tahminle eşdeğer hale gelir.

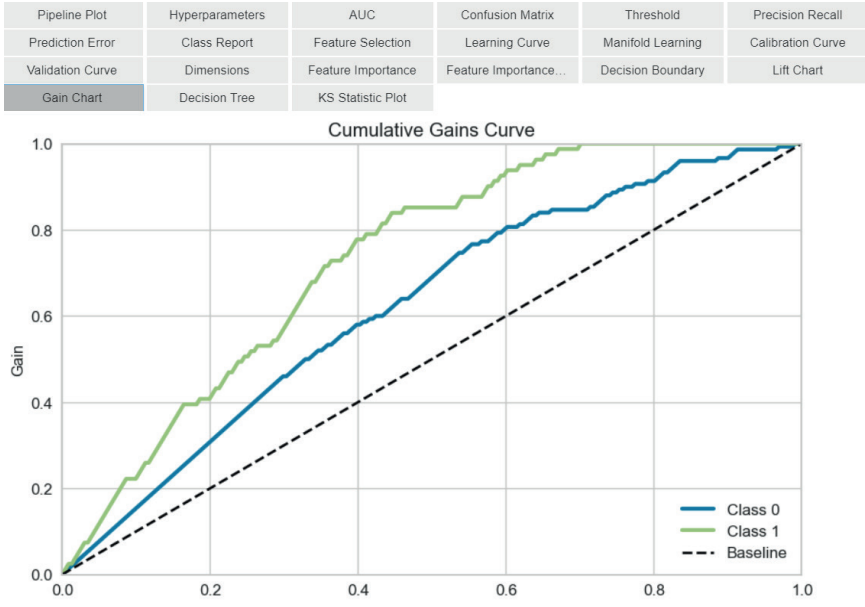
Sonuç olarak;

Class 1 için model oldukça güçlü bir ayrıştırma yapabilmiştir. Başlangıçta lift değeri 2.5'in üzerine çıkarak rastgele tahmine göre büyük bir iyileşme sağlamaktadır.

Class 0 için performans sınırlı olup lift değeri 1.5 civarındadır ve zamanla azalmaktadır.

Eğer lift değeri 1'e yakınsa, modelin rastgele tahminden bir farkı olmadığını gösterir. Ancak bu grafikte özellikle **Class 1** için model performansının güçlü olduğu söylenebilir.

Lift Curve, özellikle **sınıf dengesizliği** olan problemlerde (örneğin nadir olayların tahmini) modelin performansını değerlendirmek için kullanılır.



Şekil 1.23 *Kümülatif Kazanç Eğrisi (***Cumulative Gains Curve***)*

Kümülatif Kazanç Eğrisi (**Cumulative Gains Curve**) (Şekil 1.23), bir modelin sınıflandırma performansını değerlendirmek için kullanılan bir grafiktir. Sınıflandırma modelinin, belirli bir yüzdelik dilimde hedef sınıfın ne kadarını tahmin edebildiğini gösterir. Özellikle **sınıf dengesizliği** olan ikili sınıflandırma problemlerinde performansı görselleştirmek için kullanılır.

Grafiğin Eksenleri:

X Ekseni: Örneklerin yüzdesi (modelin yüksek olasılık tahmininden başlayarak örneklerin sıralanması).

Y Ekseni: Kazanç (tahmin edilen hedef sınıfın yüzdesi).

Grafikteki Çizgiler:

- **Siyah Kesikli Çizgi (Baseline)**: Rastgele tahmin performansı. Eğer model rastgele tahmin yapsaydı, bu çizgi üzerinde ilerlerdi.
- **Yeşil Çizgi (Class 1)**: Modelin **Sınıf 1** için performansı. Bu sınıf için yüksek kazanç oranına ulaşıyor.
- **Mavi Çizgi (Class 0)**: Modelin **Sınıf 0** için performansı. Bu sınıfta kazanç oranı daha düşük.

Kümülatif Kazanç Eğrisi (Şekil 1.23) grafiğini dikkatlice incelersek;

Sınıf 1 (Yeşil Çizgi): Sınıf 1 için model başlangıçta hızlı bir kazanç sağlar. Örneğin, verinin ilk

%20'sinde sınıf 1'in yaklaşık %70-80'ini yakalayabiliyor. Bu, modelin Sınıf 1'e ait örnekleri öncelikli olarak doğru tahmin edebildiğini gösterir. Yeşil çizginin siyah baz çizgisinden belirgin şekilde yukarıda olması, modelin rastgele tahminden çok daha iyi performans gösterdiğini ifade eder.

Sınıf 0 (Mavi Çizgi): Sınıf 0 için kazanç eğrisi daha düşük bir performans sergiliyor. Örneğin, ilk %40'lık dilimde sınıf 0'ın ancak %60'ını yakalayabiliyor. Bu sınıf için modelin performansı sınırlı kalmış ve rastgele tahmin baz çizgisine daha yakın seyrediyor.

Baz Çizgisi (Baseline): Siyah kesikli çizgi, rastgele tahmini temsil eder. Eğer model bu çizgi üzerinde ilerliyorsa, tahminleri rastgele seçimden farksızdır. Ancak, Class 1 ve Class 0 için çizgiler baz çizgisinin üzerinde yer alıyor, bu da modelin rastgele tahmine göre bir iyileşme sağladığını gösterir.

Modelin performansı, sınıflar arasında farklılık göstermektedir. Sınıf 1 için model oldukça başarılı olup, bu sınıfın büyük bir kısmını hızlıca doğru bir şekilde tahmin edebilmektedir. Sınıf 0 için ise performans sınırlı olsa da, yine de rastgele tahmine göre daha iyi sonuçlar vermektedir. Kümülatif Kazanç Eğrisi, modelin her bir sınıf için ne kadar etkili tah-

minler yaptığını görselleştirerek, özellikle sınıf dengesizliği olan ve önceliklendirme gerektiren uygulamalarda daha iyi bir anlayış sunar.

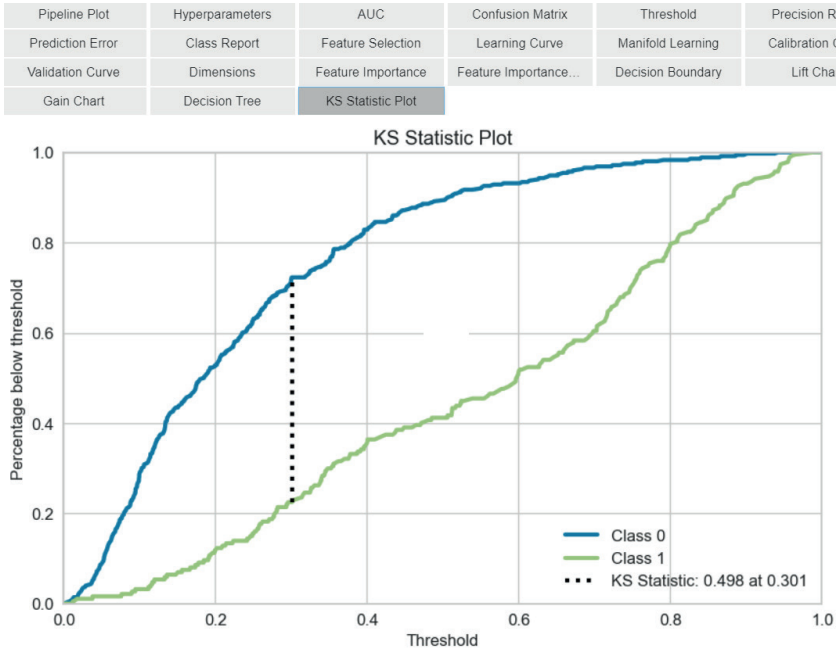
KS (Kolmogorov-Smirnov) istatistik grafiği (Şekil 1.24), bir modelin iki sınıfı ne kadar iyi ayırt ettiğini ölçen bir metriktir. İkili sınıflandırma problemlerinde özellikle modelin tahmin olasılıklarını değerlendirmek için kullanılır.

KS (Kolmogorov-Smirnov) grafiğini incelediğimizde, bir sınıflandırma modelinin iki farklı sınıfı ne kadar iyi ayırt ettiğini görsel olarak gösteren bir araçtır. Bu grafik, modelin tahmin ettiği olasılık değerlerine göre örnekleri sıralayarak, iki sınıfın kümülatif dağılımını karşılaştırır.

- **X ekseni:** Modelin belirlediği bir sınıflandırma eşiği değerini temsil eder. Bu eşik, bir örneğin hangi sınıfa ait olduğuna karar vermek için kullanılır.

- **Y ekseni:** Belirli bir eşik değerinin altında kalan örneklerin toplam yüzdesini gösterir.

KS istatistiği: İki sınıfın kümülatif dağılım eğrileri arasındaki en büyük dikey mesafeyi ifade eder. Bu değer, modelin sınıfları ne kadar iyi ayırt ettiğini ölçer. Değer 0 ile 1 arasında değişir. 1'e yakın değerler, modelin sınıfları çok iyi ayırt ettiğini gösterirken, 0'a yakın değerler modelin zayıf performans gösterdiğini belirtir.



Şekil 1.24 *KS (Kolmogorov-Smirnov) istatistiği grafiği*

Grafikteki çizgiler:

- **Mavi çizgi:** Sınıf 0'a ait örneklerin kümülatif yüzdesini gösterir.
- **Yeşil çizgi:** Sınıf 1'e ait örneklerin kümülatif yüzdesini gösterir.

Grafiğin yorumlanması:

- **KS istatistiği:** Bu örnekte KS değeri 0.498 olarak bulunmuştur. Bu değer, modelin sınıfları oldukça iyi ayırt ettiğini gösterir.
- **Eşik noktası:** KS istatistiğinin maksimum olduğu nokta, yani 0.301 eşiği, modelin en iyi performans gösterdiği noktadır. Bu noktada, iki sınıf arasındaki ayrım en belirgindir.
- **Sınıf dağılımı:** Mavi ve yeşil çizgiler arasındaki mesafe, iki sınıfın ne kadar iyi ayrıldığını gösterir. Çizgiler arasındaki mesafe ne kadar büyükse, model o kadar iyi performans göstermiştir.

Bu grafiğini genel olarak değerlendirirsek **KS değeri (0.498)** ve iki sınıfın ayrıştığı maksimum eşik (0.301), modelin pozitif ve negatif sınıfları ayırma gücünün oldukça iyi olduğunu göstermektedir. Modelin performansı daha da artırılmak istenirse, eşik değeri optimize edilebilir veya modelin tahmin yeteneği farklı metrikler ile iyileştirilebilir.

1.3 Tahminleme (Prediction)

PyCaret'te eğitilmiş bir modelin yeni veri üzerinde tahmin yapmak için predict_model fonksiyonu kullanılır (Liste 1.12). Daha önce eğitilmiş bir modeli kullanarak yeni veriler üzerinde tahminleme yapar. Bu işlem sonucunda, verilen veri çerçevesine (DataFrame) iki yeni sütun eklenir:

1. prediction_label: Model tarafından tahmin edilen sınıf etiketi. Bu, sınıflandırma problemlerinde modelin hangi sınıfı seçtiğini belirtir.

2. prediction_score: Tahmin edilen sınıfın olasılık değeri. Bu değer, modelin tahminine ne kadar güvendiğini gösterir ve genellikle 0 ile 1 arasında bir değer olarak verilir.

Eğer veri seti (data) belirtilmezse, yani *data=None* olarak bırakılırsa, *predict_model* fonksiyonu test setini kullanır. Test seti, PyCaret'in setup fonksiyonu kullanılarak model kurulumu aşamasında otomatik olarak oluşturulan veri kümesidir. Bu durumda modelin performansı test verisi üzerinde değerlendirilir ve çıktı olarak tahminler döndürülür.

Bu fonksiyon, modelin farklı veri setleri üzerinde nasıl performans gösterdiğini analiz etmek için oldukça kullanışlıdır. Ayrıca, yeni veriler üzerinde tahminleme yaparak modelin uygulanabilirliğini ve güvenilirliğini test etme olanağı sağlar.

```
# Liste 1.12 Test seti üzerinde tahminleme
holdout_pred = predict_model(best)
```

Liste 1.12 Test Seti üzerinde en başarılı model ile (Lojistik Regresyon) tahminleme

Liste 1.12'deki kod ile dataframe (*holdout_pred* isimli nesne) formatında bir tahminleme dizisi üretir. Bu testin başarı değeri Şekil 1.25'te gösterildiği gibidir.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.7576	0.8568	0.5309	0.7049	0.6056	0.4356	0.4447

Şekil 1.25 *Tahminleme algoritma modelinin performans değerleri*

```
# Liste 1.13 Test Setinin ilk 5 satırı ekrana yazdır
holdout_pred.head()
```

Liste 1.13 Test Setinin ilk 5 satırı ekrana yazdırma

holdout_pred.head()(Liste 1.13) komutuyla, modelin test seti üzerindeki ilk 5 örneğe ait gerçek sınıf etiketleri ile modelin tahmin ettiği sınıf etiketlerini ve olasılıklarını yan yana görerek hızlı bir değerlendirme yapabiliriz (Şekil 1.26).

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)	Class variable	prediction_label	prediction_score
537	6	114	88	0	0	27.799999	0.247	66	0	0	0.8037
538	1	97	70	15	0	18.200001	0.147	21	0	0	0.9648
539	2	90	70	17	0	27.299999	0.085	22	0	0	0.9393
540	2	105	58	40	94	34.900002	0.225	25	0	0	0.7998
541	11	138	76	0	0	33.200001	0.420	35	0	1	0.6391

Şekil 1.26 *Test Setinin ilk 5 satırı*

Aynı fonksiyon, görülmemiş veri kümeleri üzerindeki etiketleri tahmin etmek için de kullanılabilir. Özgün veri setinin bir kopyasını oluşturup Class değişkenini çıkararak, yeni bir etiketsiz veri çerçevesi elde edebiliriz. Bu yeni veri çerçevesini skorlama işlemi için kullanabiliriz (Liste 1.14).

```
# Liste 1.14 veriyi kopyalama ve Sınıf değişkenini silme
new_data = data.copy()
new_data.drop('Class variable', axis=1, inplace=True)
new_data.head()
```

Liste 1.14 *Eğitimde kullanılan veri setinin tahminleme için kullanımı*

Bu kod (Liste 1.14), orijinal veri setini korumak amacıyla önce *data.copy()* kullanarak bir kopyasını oluşturur ve bu kopyayı *new_data* adı altında saklar. Daha sonra *drop()* fonksiyonu ile *"Class variable"* isimli sütun veri setinden kaldırılır. Bu işlemde *axis=1* parametresi, sütun bazında bir silme işlemi gerçekleştirileceğini belirtirken, *inplace=True* kullanımı değişikliklerin doğrudan *new_data* üzerinde yapılmasını sağlar. Son olarak, *new_data.head()* komutu ile veri setinin güncellenmiş halinin ilk 5 satırı görüntülenir. Bu işlemler sonucunda sınıf değişkeni çıkarılmış, analiz veya modelleme için hazırlanmış yeni bir veri seti elde edilir.

Eğitim veri setinin tamamı Liste 1.15'teki kod ile tahminlenerek modelin başarısı tüm veri üzerinde Şekil 1.27'deki gibi görülebilir.

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

Şekil 1.27 *Tüm veri setinin eğitim öncesi sınıf sütununun çıkarılan özeti*

```
# Liste 1.15 new_data üzerinde modeli tahmin etme
predictions = predict_model(best, data = new_data)
predictions.head()
```

Liste 1.15 *Eğitimde kullanılan veri setinin tahminleme için kullanımı*

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)	prediction_label	prediction_score
0	6	148	72	35	0	33.599998	0.627	50	1	0.6940
1	1	85	66	29	0	26.600000	0.351	31	0	0.9419
2	8	183	64	0	0	23.299999	0.672	32	1	0.7976
3	1	89	66	23	94	28.100000	0.167	21	0	0.9454
4	0	137	40	35	168	43.099998	2.288	33	1	0.8395

Şekil 1.28 *Tüm veri setinin eğitim öncesi sınıf sütununun çıkarılar özeti*

1.4 Modelin Kaydedilmesi (Save Model)

Save Model başlığı altında, PyCaret'in *save_model* fonksiyonu, oluşturulan tüm veri işleme adımları ve makine öğrenimi modelini içeren bir pipeline'ı (eğitim iş akışını) disk üzerinde kaydetmek için kullanır (Liste 1.16).

```
# Liste 1.16 modelin kaydedilmesi
save_model(best, 'my_first_pipeline')
```

Liste 1.16 *Modelin Kaydedilmesi*

Modelin kaydedilmesi, ileride aynı modeli tekrar kullanmak veya başka bir ortama taşımak için büyük kolaylık sağlar. Kaydedilen model, sadece algoritmayı değil, veri ön işleme adımları, dönüşümler ve hiperparametre ayarları gibi tüm pipeline'ı kapsar. Böylece, modelin yeniden eğitilmesine gerek kalmadan doğrudan tahminler yapmak mümkün olur. Kaydedilen model dosyası, daha sonra ***load_model*** fonksiyonu ile tekrar yüklenerek kullanılabilir. Bu özellik, özellikle büyük projelerde veya sürekli model kullanımı gerektiren durumlarda oldukça faydalıdır. Kayıt işlemi başarıyla gerçekleştiğinde Şekil 1.28'deki gibi bir çıktı alınır.

Modelin tekrar çalışma ortamına yüklenmesi, Liste 1.17'deki kod ile gerçekleştirildiğinde (model eğitim aşamalarını gösteren işlem hattı şeması) Şekil 1.29 'daki gibi bir çıktı ile gerçekleşir.

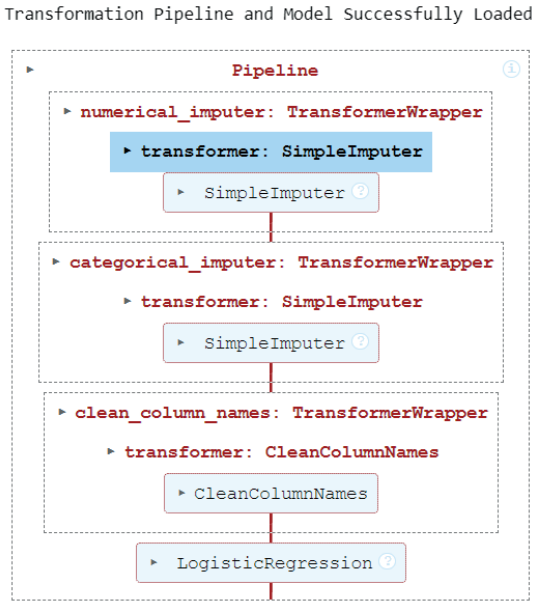
```
# Liste 1.17 modelin yüklenmesi
# load pipeline
loaded_best_pipeline = load_model('my_first_pipeline')
loaded_best_pipeline
```

Liste 1.17 *Modelin Yüklenmesi*

```
Transformation Pipeline and Model Successfully Saved

(Pipeline(memory=Memory(location=None),
          steps=[('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['Number of times pregnant',
                                              'Plasma glucose concentration a 2 '
                                              'hours in an oral glucose '
                                              'tolerance test',
                                              'Diastolic blood pressure (mm Hg)',
                                              'Triceps skin fold thickness (mm)',
                                              '2-Hour serum insulin (mu U/ml)',
                                              'Body mass index (weight in '
                                              'kg/(height in m)^2)',
                                              'Diabetes pedigre...
                  TransformerWrapper(exclude=None, include=None,
                                     transformer=CleanColumnNames(match='[\\]\\[\\,\\{\\}\\"\\:]+'))),
                 ('trained_model',
                  LogisticRegression(C=1.0, class_weight=None, dual=False,
                                     fit_intercept=True, intercept_scaling=1,
                                     l1_ratio=None, max_iter=1000,
                                     multi_class='auto', n_jobs=None,
                                     penalty='l2', random_state=123,
                                     solver='lbfgs', tol=0.0001, verbose=0,
                                     warm_start=False))],
          verbose=False),
 'my_first_pipeline.pkl')
```

Şekil 1.28 *Modelin kaydedilme çıktısı*



Şekil 1.29 *Modelin kaydedilme çıktısı*

Test veri setinde olduğu gibi yüklenen model ile yeni veriler tahminlenebilir. Daha ayrıntılı eğitim ve model karşılaştırma süreçleri PyCaret'ın github sayfasından incelenebilir.

1.5 Modeli Ayarlama (Tune Model)

Makine öğrenimi modellerinin performansını en üst düzeye çıkarmak, genellikle doğru hiperparametrelerin belirlenmesine bağlıdır. PyCaret'in ***tune_model*** fonksiyonu, bu süreci otomatikleştirerek kullanıcıya büyük bir kolaylık sağlar. Bu fonksiyon, modelin hiperparametrelerini ayarlayarak en iyi kombinasyonu bulur ve performansını optimize eder.

Fonksiyonun çıktısı, çapraz doğrulama yöntemiyle katmanlara (fold) göre elde edilen skorların yer aldığı bir değerlendirme tablosudur. Modelin başarısını değerlendirmek için hangi metriğin kullanılacağını belirlemek, ***optimize*** parametresi üzerinden yapılır. Örneğin, doğruluk, F1 skoru veya ROC-AUC gibi çeşitli metrikler kullanılabilir.

Değerlendirme sırasında kullanılan metrikler, ***get_metrics*** fonksiyonu yardımıyla görüntülenebilir. Ancak bazen, projeye özel metriklere ihtiyaç duyulabilir. Bu durumda, ***add_metric*** fonksiyonu ile özel metrikler sisteme eklenebilir ya da gereksiz görülen metrikler ***remove_metric*** fonksiyonu kullanılarak kaldırılabilir. Böylece, model ayarlama süreci tamamen özelleştirilebilir ve projenin ihtiyaçlarına göre şekillendirilir. ***tune_model***, verimlilik ve esneklik sağlarken, en iyi performansı elde etmek için modelin ince ayarını yaparak veri bilimi yolculuğunuzu daha da ileriye taşır. Genel olarak ***tune_model*** komutuyla elde edilen modeller oldukça başarılıdır. Ancak probleme ve veri setine özgü ince ayarlamaların yapılabileceği unutulmamalıdır. Liste 1.18 ile varsayılan parametreler kullanılarak bir karar ağacı (decision tree) modeli üretilmiştir. Şekil 1.29'da eğitim sürecindeki her fold değerine karşılık eğitim performans değerleri gösterilmiştir.

```
# Liste 1.18 varsayılan parametrelerle bir karar ağacı (decision tree) modelin eğitimi
dt = create_model('dt')
```

Liste 1.18 *Modelin üretilmesi*

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7222	0.6774	0.5263	0.6250	0.5714	0.3682	0.3711
1	0.7222	0.7015	0.6316	0.6000	0.6154	0.3982	0.3985
2	0.7407	0.7038	0.5789	0.6471	0.6111	0.4176	0.4190
3	0.5926	0.5053	0.2105	0.3636	0.2667	0.0116	0.0125
4	0.7778	0.7684	0.7368	0.6667	0.7000	0.5242	0.5259
5	0.6296	0.5940	0.4737	0.4737	0.4737	0.1880	0.1880
6	0.6296	0.5699	0.3684	0.4667	0.4118	0.1469	0.1491
7	0.8302	0.7770	0.6111	0.8462	0.7097	0.5940	0.6098
8	0.6604	0.6079	0.4444	0.5000	0.4706	0.2219	0.2227
9	0.6415	0.6206	0.5556	0.4762	0.5128	0.2319	0.2336
Mean	0.6947	0.6526	0.5137	0.5665	0.5343	0.3103	0.3130
Std	0.0720	0.0834	0.1410	0.1310	0.1292	0.1714	0.1739

Şekil 1.29 *dt modelinin performans değerlerinin fold adımlarına göre listelenmesi*

Liste 1.19'daki parametre ayarlaması yapılarak **Karar Ağacı (Decision Tree)** modelinin hiperparametrelerini optimize edebiliriz. Burada dt değişkeni, önceden tanımlanmış bir Karar Ağacı modelini temsil eder ve **tune_model(dt)** fonksiyonu, modelin performansını artırmak için en iyi hiperparametre kombinasyonunu arar. Bu işlem sırasında çapraz doğrulama uygulanır ve değerlendirme, **optimize** parametresinde belirtilen metrik (örneğin doğruluk, ROC-AUC, F1 skoru gibi) üzerinden yapılır. Ayarlanmış hiperparametrelerle elde edilen en iyi model, **tuned_dt** değişkeninde saklanır. Sonuç olarak, **tune_model** fonksiyonu, performansı optimize edilmiş ve kullanıma hazır bir model döner. Şekil 1.30'daki ortama accuracy (doğruluk) değeri incelendiğinde Şekil 1.29'da 0.6947 olan değer Şekil 1.30'da 0.7372'ye yükselmiştir. Alternatif parametre ayarlama gerçekleştirilerek bu doğruluk değeri arttırlmaya çalışılabilir.

```
# Liste 1.19 modelin modelin otomatik ayarlanması
# modelin otomatik ayarlanması (tune)
tuned_dt = tune_model(dt)
```

Liste 1.19 *Modelin parametrelerinin ayarlanması (tuning)*

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8519	0.8135	0.6842	0.8667	0.7647	0.6588	0.6686
1	0.7593	0.6940	0.4737	0.7500	0.5806	0.4236	0.4456
2	0.7593	0.7782	0.8421	0.6154	0.7111	0.5132	0.5318
3	0.7037	0.6511	0.4737	0.6000	0.5294	0.3175	0.3223
4	0.8333	0.7632	0.5263	1.0000	0.6897	0.5902	0.6470
5	0.6296	0.5820	0.4211	0.4706	0.4444	0.1680	0.1685
6	0.7222	0.6654	0.4737	0.6429	0.5455	0.3520	0.3605
7	0.7358	0.6246	0.2778	0.8333	0.4167	0.2973	0.3725
8	0.6604	0.5675	0.2778	0.5000	0.3571	0.1512	0.1633
9	0.7170	0.6643	0.5000	0.6000	0.5455	0.3424	0.3454
Mean	0.7372	0.6804	0.4950	0.6879	0.5585	0.3814	0.4026
Std	0.0653	0.0782	0.1608	0.1611	0.1258	0.1587	0.1654

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

Şekil 1.30 *dt modelinin parametre ayarlaması sonrası performansın fold işlemine göre değişimi*

1.6 Birleştirilmiş (Ensemble) Model

Eğitilen farklı algoritmalar yapılan parametre düzenlemesine ve çeşitli konfigürasyonlarla denemeler yapılmasına rağmen başarısız olursa birden fazla model ile birleştirilebilir ve ensemble (topluluk) bir model ile yeni karma bir model türetilebilir.

ensemble_model fonksiyonu, verilen tahminleyiciyi (estimator) topluluk yöntemleriyle birleştirebilir ve bir ensemble model (topluluk modeli) ortaya çıkarır. Fonksiyonun çıktısı, çapraz doğrulama (**CV**) ile katmanlara göre hesaplanan skorların yer aldığı bir değerlendirme tablosudur. Çapraz doğrulama sırasında değerlendirilen metrikler **get_metrics** fonksiyonu kullanılarak görüntülenebilir. Ayrıca, özel metrikler **add_metric** fonksiyonu ile eklenebilir veya gereksiz metrikler **remove_metric** fonksiyonu ile kaldırılabilir. Bu sayede model değerlendirme süreci ihtiyaçlara göre özelleştirilebilir ve daha başarılı sistemler üretilebilir.

```
# Liste 1.20 Bagging metodu ile model birleştirme
ensemble_model(dt, method = 'Bagging')
```

Liste 1.20 *Modelin parametrelerinin ayarlanması (tuning)*

Bagging (Bootstrap Aggregating) kullanarak topluluk (ensemble) modelinin oluşturulması (Liste 1.20) karar ağaçları algoritma modellerinin bagging yöntemi ile birleştirilmesini ifade eder. Bagging, aynı tahmin-

leyiciyi (estimator) farklı örnekleme veri setleri üzerinde eğiterek birden fazla model oluşturur ve bu modellerin tahminlerini birleştirir. Bu işlem, modelin varyansını azaltarak aşırı öğrenmeyi (ezberleme veya overfitting) önlemeye yardımcı olur ve modelin genel performansını iyileştirir.

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7407	0.8383	0.5263	0.6667	0.5882	0.4028	0.4088
1	0.7963	0.7797	0.7368	0.7000	0.7179	0.5587	0.5591
2	0.7593	0.7669	0.4737	0.7500	0.5806	0.4236	0.4456
3	0.7222	0.7842	0.5263	0.6250	0.5714	0.3682	0.3711
4	0.8148	0.8421	0.7368	0.7368	0.7368	0.5940	0.5940
5	0.6852	0.6759	0.4211	0.5714	0.4848	0.2656	0.2720
6	0.7037	0.7677	0.5263	0.5882	0.5556	0.3344	0.3355
7	0.7925	0.8405	0.4444	0.8889	0.5926	0.4734	0.5245
8	0.6792	0.6659	0.5000	0.5294	0.5143	0.2751	0.2754
9	0.6792	0.6508	0.3333	0.5455	0.4138	0.2103	0.2224
Mean	0.7373	0.7612	0.5225	0.6602	0.5756	0.3906	0.4009
Std	0.0488	0.0695	0.1212	0.1060	0.0924	0.1195	0.1221

Şekil 1.31 *dt modelinin bagging yöntemi ile birleştirilen model performansı*

Tune işlemine göre (Şekil 1.30) sistem performansında az da olsa bir artış olduğu Şekil 1.31'de gözlemlenmiştir.

1.7. Karışım Modelleri (Blend Models)

Birden fazla farklı tür algoritma modeli birleştirerek performansı arttırmak istendiğinde Liste 1.21'de verilen *blend_model* yaklaşımı uygulanabilir. Bu fonksiyon, *estimator_list* parametresi aracılığıyla verilen modeller için Soft Voting/Majority Rule sınıflandırıcısını eğitir. Soft Voting, her modelin tahmin ettiği olasılıkları birleştirerek nihai sınıf tahminini yaparken, Majority Rule (Çoğunluk Kuralı) doğrudan sınıf etiketlerinin çoğunluğuna göre karar verir. Fonksiyonun çıktısı, çapraz doğrulama (Cross Validation) ile katmanlara göre hesaplanan skorların bulunduğu bir değerlendirme tablosudur.

Çapraz doğrulama sırasında değerlendirilen metrikler, *get_metrics* fonksiyonu kullanılarak görüntülenebilir. Ayrıca, özel metrikler *add_metric* fonksiyonu ile eklenebilir veya *remove_metric* fonksiyonu ile kaldırılabilir. Bu sayede, model performansı esnek bir şekilde yönetilebilir.

```
# Liste 1.21 en iyi 3 modelin recall değerine göre çağrılması
best_recall_models_top3
```

Liste 1.21 *Farklı algoritmaların karıştırılması (blend modals)*

Recall değerine göre Şekil 1.31'de en başarılı 3 modelin parametreleri gösterilmiştir. Bu 3 modelin karıştırılarak daha başarılı bir adet karma model oluşturulması (Liste 1.22)'de

```
[GaussianNB(priors=None, var_smoothing=1e-09),
 GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                            learning_rate=0.1, loss='log_loss', max_depth=3,
                            max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_samples_leaf=1,
                            min_samples_split=2, min_weight_fraction_leaf=0.0,
                            n_estimators=100, n_iter_no_change=None,
                            random_state=123, subsample=1.0, tol=0.0001,
                            validation_fraction=0.1, verbose=0,
                            warm_start=False),
 LinearDiscriminantAnalysis(covariance_estimator=None, n_components=None,
                            priors=None, shrinkage=None, solver='svd',
                            store_covariance=False, tol=0.0001)]
```

Şekil 1.31 *Recall değerine göre en başarılı 3 model; Gaussian NB, Gradient Boosting Classifier, Linear Discriminant Analysis*

```
#Liste 1.22 en iyi 3 modelin karıştırılması
blend_models(best_recall_models_top3)
```

Liste 1.22 *Farklı algoritmaların karıştırılması (blend modals)*

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
Fold							
0	0.7963	0.8932	0.6842	0.7222	0.7027	0.5479	0.5484
1	0.7778	0.8120	0.6316	0.7059	0.6667	0.5008	0.5025
2	0.8704	0.9338	0.6842	0.9286	0.7879	0.6976	0.7145
3	0.7037	0.7865	0.4737	0.6000	0.5294	0.3175	0.3223
4	0.8704	0.8962	0.6842	0.9286	0.7879	0.6976	0.7145
5	0.7037	0.6692	0.4737	0.6000	0.5294	0.3175	0.3223
6	0.7407	0.7805	0.6842	0.6190	0.6500	0.4449	0.4463
7	0.7736	0.8667	0.4444	0.8000	0.5714	0.4342	0.4688
8	0.6604	0.6889	0.4444	0.5000	0.4706	0.2219	0.2227
9	0.6981	0.7286	0.4444	0.5714	0.5000	0.2886	0.2933
Mean	0.7595	0.8056	0.5649	0.6976	0.6196	0.4469	0.4555
Std	0.0683	0.0868	0.1103	0.1407	0.1104	0.1575	0.1616

Şekil 1.32 *Gaussian NB, Gradient Boosting Classifier ve Linear Discriminant Analysis modellerinin karıştırılması ile elde edilmiş model performansı*

Karma modellerin daha fazla performans vermesi genel bir beklentidir. Şu ana kadar incelediğimiz tüm modellerden başarılı oluşu bunun bir göstergesidir. Ancak daha fazla başarı için farklı karışımlar oluşturulabilir.Performasn artışıı sağlayabilecek bazı diğer parametreler şunlardır:

- **choose_better**: Model performansını otomatik olarak değerlendiren ve en iyi modeli seçen bir parametredir.
- **method**: Model tahminlerini birleştirme yöntemini belirler. Örneğin, soft (olasılık tabanlı) veya hard (çoğunluk kuralı) gibi seçenekler kullanılabilir.
- **weights**: Modellerin tahminlerine verilecek ağırlıkları ayarlamak için kullanılır. Daha güçlü performansa sahip modellere daha fazla ağırlık verilebilir.
- **fit_kwargs**: Temel modellerin eğitim sürecine ek argümanlar geçirebilmenizi sağlar.
- **probability_threshold**: Sınıflandırma problemlerinde tahmin edilen olasılıklar için bir eşik değeri belirler.
- **return_train_score**: Eğitim setindeki skorların döndürülüp döndürülmeyeceğini belirler.

Fonksiyonun daha fazla detayı için yardım açıklamalarına (docstring) göz atabilirsiniz (Liste 1.23). Bu parametreler, modeli özelleştirmenizi ve performansını iyileştirmenizi sağlar.

```
# Liste 1.23 Daha fazla bilgi almak için dökümantasyon(docstring) çağırma
help(blend_models)
```

Liste 1.23 *Bilgi almak için dokümantasyon (docstring) çağırma*

1.8. Meta Modeller (Stack Models)

Fonksiyonu Açıklaması:

Stacking, birden fazla temel modelin (base estimator) çıktılarını birleştirip (Liste 1.24), bu çıktılar üzerinden bir meta-model eğiterek daha güçlü bir sınıflandırma performansı elde etmeyi amaçlayan bir yöntemdir.

Nasıl Çalışır?

- **Temel Modeller:** *estimator_list* parametresiyle belirlenen temel modeller, veri üzerinde eğitilir ve her biri kendi tahminlerini yapar.
- **Meta-Model:** Temel modellerin ürettiği tahminleri kullanarak bir meta-model eğitilir. Bu model genelde Logistic Regression gibi daha basit bir model olabilir.
- **Sonuç:** *stack_models* fonksiyonu, her bir fold için çapraz doğrulama (CV) sonuçlarını içeren bir değerlendirme tablosu döndürür.

Avantajlar:

- Birden fazla modelin güçlü yönlerini birleştirerek daha iyi genel performans sağlar.
- Özellikle temel modeller farklı algoritmalar veya özellikler üzerine eğitilmişse, çeşitlilik daha iyi sonuçlar doğurur.
- Sınıflandırma görevlerinde, pozitif ve negatif sınıflar arasında daha iyi bir ayrım yapılabilir.

Kısaca: Stacking modeli, temel modellerin tahminlerini birleştirip meta-model ile daha güçlü bir tahminleme yapmayı sağlar. Özellikle ikili sınıflandırma görevlerinde, farklı metrikler (Accuracy, AUC, Recall vb.) üzerinde iyileştirme potansiyeli sunar. Bu yöntem, temel modellerin hatalarını telafi etmek ve genel performansı artırmak için etkili bir stratejidir.

```
# Liste 1.24 Meta model üretmek için en iyi 3 MAE değerli model alınır
stack_models(best_mae_models_top3)
```

Liste 1.24 *Bilgi almak için dökümantasyon (docstring) çağırma*

Şekil 1.33'teki meta modelin başarısını ayrıntılı olarak incelersek;

1. **Accuracy (Doğruluk):** Ortalama doğruluk oranı %76.33. Bu, modelin tahminlerinin yaklaşık %76'sının doğru olduğunu gösterir. Ancak bu metrik, dengesiz veri setlerinde yanıltıcı olabilir.

2. **AUC (ROC Eğrisi Altındaki Alan):** AUC değeri 0.8035, modelin pozitif ve negatif sınıfları iyi ayırt ettiğini gösteriyor. Bu, modelin genel ayırma kapasitesinin iyi olduğunu ifade eder.

3. **Recall (Duyarlılık):** Recall değeri 0.5333, modelin pozitif sınıfı tahmin etme başarısının sınırlı olduğunu gösterir. Model, gerçek pozitiflerin yalnızca %53'ünü doğru tahmin edebilmiştir.

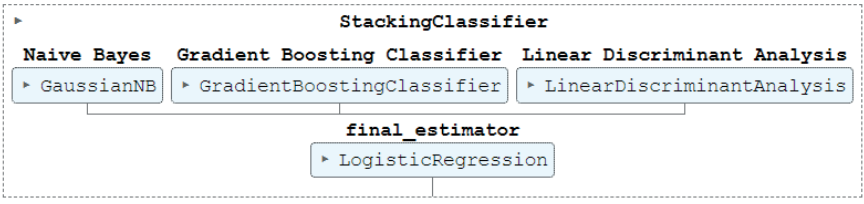
4. **Precision (Kesinlik):** Precision değeri 0.7190, modelin pozitif olarak tahmin ettiği örneklerin yaklaşık %72'sinin doğru olduğunu ifade eder. Bu, modelin yanıltıcı pozitif tahminlerinin düşük olduğunu gösterir.

5. **F1 Skoru:** F1 skoru 0.6096, recall ve precision arasında bir denge sağlandığını gösteriyor. Ancak değer ideal bir seviyede değildir.

6. **Kappa:** Cohen's Kappa değeri 0.4453, sınıflandırma başarısının rastgele tahminden ne kadar farklı olduğunu gösterir. Bu değer, modelin orta seviyede bir performansa sahip olduğunu ifade eder.

7. **MCC (Matthews Correlation Coefficient):** MCC değeri 0.4572, sınıflandırmanın genel doğruluğu ile sınıf dengesini dikkate alan bir ölçüttür. Bu değer de modelin orta düzeyde bir performans sergilediğini gösterir.

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8148	0.9023	0.6316	0.8000	0.7059	0.5735	0.5820
1	0.7963	0.7970	0.6316	0.7500	0.6857	0.5367	0.5410
2	0.8704	0.9233	0.6842	0.9286	0.7879	0.6976	0.7145
3	0.7037	0.7835	0.4737	0.6000	0.5294	0.3175	0.3223
4	0.8519	0.8992	0.6316	0.9231	0.7500	0.6499	0.6736
5	0.6852	0.6722	0.4211	0.5714	0.4848	0.2656	0.2720
6	0.7222	0.7910	0.5263	0.6250	0.5714	0.3682	0.3711
7	0.7547	0.8667	0.3889	0.7778	0.5185	0.3776	0.4184
8	0.6981	0.6810	0.4444	0.5714	0.5000	0.2886	0.2933
9	0.7358	0.7190	0.5000	0.6429	0.5625	0.3775	0.3836
Mean	0.7633	0.8035	0.5333	0.7190	0.6096	0.4453	0.4572
Std	0.0628	0.0879	0.0989	0.1300	0.1061	0.1479	0.1514



Şekil 1.33 *Gaussian NB, Gradient Boosting Classifier ve Linear Discriminant Analysis modellerinin Meta Modele dönüştürülmesi*

Bu sonuçları (Şekil 1.33) ikili (binary) sınıflandırma açısından değerlendirirsek:

Avantajlar:

- **AUC:** AUC'nin yüksek olması, modelin pozitif ve negatif sınıflar arasında iyi bir ayrım yapma yeteneğine sahip olduğunu gösterir.
- **Precision:** Modelin kesinlik değeri yüksek, bu da yanlış pozitif tahminlerin az olduğunu ifade eder.

Dezavantajlar:

- **Recall:** Recall değerinin düşük olması, modelin pozitif sınıfa ait birçok örneği kaçırdığını (false negatives) gösterir.
- **Denge:** F1 skoru, precision ve recall arasında bir denge sağlasa da bu değer daha yüksek olabilirdi.

İyileştirme Önerileri:

1. Hiperparametre Optimizasyonu: Kullanılan temel modellerin (Naive Bayes, Gradient Boosting Classifier, LDA) hiperparametreleri optimize edilebilir.

2. Veri Dengesi: Pozitif ve negatif sınıfların dengesiz olduğu bir veri seti varsa, SMOTE gibi yöntemlerle veri dengesi sağlanabilir.

3. Final Estimator: Logistic Regression yerine daha karmaşık bir model (örneğin, Random Forest veya XGBoost) final tahminci olarak denenebilir.

4. Ek Veri Özellikleri: Daha iyi bir ayrım için veri özellikleri zenginleştirilebilir veya daha önemli özellikler seçilebilir.

Sonuç olarak, modelin genel performansı kabul edilebilir seviyede olsa da, özellikle recall üzerinde çalışılarak sınıf dengesinin daha iyi sağlanabileceği görülmektedir.

2. Bölüm

Regresyon: Sürekli Değer Tahmini

Giriş

Bu bölümde, PyCaret kullanarak regresyon algoritmalarını ve yöntemlerini adım adım nasıl uygulayabileceğinizi öğreneceksiniz. Regresyon, gözetimli makine öğrenmesi problemleri arasında sürekli değerleri tahmin etmek için kullanılan temel yöntemlerden biridir. Örneğin, satış miktarını tahmin etme, sıcaklık öngörüsü yapma veya bir ürünün talep miktarını belirleme gibi gerçek hayatta sıkça karşılaşılan problemler, regresyon yöntemleriyle çözülebilir.

PyCaret'in Regresyon Modülü, bağımlı değişken (hedef ya da çıktı değişkeni) ile bir veya birden fazla bağımsız değişken (özellikler, tahmin ediciler) arasındaki ilişkileri analiz etmek için güçlü bir araç sunar. Bu modül, veri hazırlama sürecinde *setup* fonksiyonu aracılığıyla kapsamlı ön işleme yetenekleri sağlar. Eksik veri doldurma, değişken dönüşümleri, ölçeklendirme ve daha fazlasını otomatikleştirerek, karmaşık işlemleri sadeleştirir ve süreci hızlandırır.

PyCaret, kullanıcı dostu yapısıyla regresyon modellerinin oluşturulmasını ve değerlendirilmesini oldukça kolaylaştırır. Modül içinde kullanıma hazır algoritmaları kullanarak hızlı bir şekilde model analizi gerçekleştirebilir. Ayrıca, modellerin performansını detaylı bir şekilde incelemenize olanak tanıyan çeşitli grafikler ve değerlendirme metrikleri sunar. Bu araçlar, modelin doğruluğunu, tutarlılığını ve genel başarımını değerlendirmenize yardımcı olur.

Bu bölümde, PyCaret'in sunduğu algoritmaların yanı sıra hiperparametre ayarlama yöntemleri ve performans analiz araçlarına da odaklanacağız. Her bir adımı, örnek kodlar ve açıklamalar eşliğinde ele alarak PyCaret ile regresyon problemlerini çözmek için gerekli tüm bilgi ve araçları sağlamayı hedefliyoruz.

2.1 PyCaret Regresyon Modülü

Regresyonun temel amacı, sürekli değerleri tahmin etmektir. Örneğin, satış miktarını öngörmek, sıcaklık değerlerini tahmin etmek veya belirli bir miktarı hesaplamak gibi senaryolarda kullanılır. PyCaret'in regresyon modülü, setup fonksiyonu aracılığıyla veriyi modellemeye hazırlamak için çeşitli ön işleme araçlarını sağlar. Bu araçlar sayesinde eksik veri doldurma, dönüşüm, ölçeklendirme gibi işlemler otomatik olarak

gerçekleştirilir.

PyCaret Regresyon Modülü'nde tipik bir iş akışı sınıflandırma işlemlerinde olduğu gibi 7 adımda gerçekleşir:

8. Sistem gereksinimlerinin *PyCaret* ortamının problem türüne göre kurulması

9. Modellerin eğitilmesi

10. En başarılı modelin performans kriterleriyle karşılaştırılarak belirlenmesi

11. En iyi modelin analiz edilmesi

12. Yeni Verilerle Tahminler

13. Model Performans İyileştirme

14. Modelin Kaydedilmesi

Bu adımlar, bir regresyon problemine sistematik bir şekilde yaklaşmanızı ve PyCaret kullanarak etkili tahmin modelleri oluşturmanızda bir akış oluşturur.

2.2 Veriseti Yükleme

PyCaret kütüphanesinin veri seti modülünü kullanarak hazır bir veri seti yüklemek için Liste 2.1 tasarlanmıştır. İlk olarak, *from pycaret.datasets import get_data* ifadesiyle PyCaret'in hazır veri setlerini yüklemek için kullanılan *get_data* fonksiyonu içe aktarılır. Ardından, *data = get_data('insurance')* komutu çalıştırılarak *'insurance'* adlı örnek veri seti indirilir ve *data* isimli bir değişkene atanır.

Bu veri seti, sigorta prim tahminleriyle ilgili bilgiler içerir ve regresyon problemlerinde kullanılmak üzere hazırlanmıştır. Özellikler yardımıyla bireylerin sigorta masraflarını tahmin etme gibi senaryolar için ideal bir başlangıç noktası sunar. Bu tür veri setleri, PyCaret'in sunduğu yetenekleri öğrenmek ve uygulamak için güçlü bir temel sağlar.

```
# Liste 2.1 Kütüphaneleri Yükleme
from pycaret.datasets import get_data
data = get_data('insurance')
```

Liste 2.1 *Veriseti Yükleme*

'insurance' veri seti, genellikle sigorta maliyetlerini tahmin etmek amacıyla kullanılan bir örnek veri setidir ve regresyon problemleri için idealdir. Bu veri seti, bireylerin çeşitli demografik ve sağlıkla ilgili özelliklerine dayalı olarak sigorta primlerini (maliyetlerini) tahmin etmek için kullanılır. Veri setini tanımak problemi anlamak adına son derece önem-

lidir bu sebeple veri setindeki öznitelikleri kısaca tanıyalım

1. **age:** Bireyin yaşı (sürekli değişken).

2. **sex:** Cinsiyet (kategorik değişken: erkek veya kadın).

3. **bmi:** Vücut kitle indeksi (sürekli değişken), bireyin kilosu ve boyu arasındaki oranı ifade eder.

4. **children:** Bireyin bakmakla yükümlü olduğu çocuk sayısı (sürekli değişken).

5. **smoker:** Sigara kullanımı durumu (kategorik değişken: evet/hayır).

6. **region:** Bireyin yaşadığı coğrafi bölge (kategorik değişken: northeast, northwest, southeast, southwest).

7. **charges:** Sigorta prim maliyeti (hedef değişken, sürekli değer).

Bu veri seti, bireylerin özelliklerini analiz ederek "sigorta maliyetlerini tahmin etmek" amacıyla kullanılır. Özellikle sigara kullanımı, yaş ve bmi (vücut kitle indeksi) gibi değişkenlerin sigorta primine olan etkisini incelemek için idealdir. Regresyon algoritmalarını test etmek ve model performansını değerlendirmek için yaygın olarak kullanılır (Şekil 2.1).

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Şekil 2.1 *Insurance (sigorta) veri seti özeti*

Pycaret ortamının regresyon (değer tahminleme) için algoritmalarının seçilmesi sürecini Liste 2.2'teki kodlarla varsayılan ayarlarda gerçekleştirelim.

```
# Liste 2.2 pycaret kütüphanesinden regresyonun için kurulum başlatılıyor
from pycaret.regression import *
s = setup(data, target = 'charges', session_id = 123)
```

Liste 2.2 *Regresyon tahminleme için veri setinin*

Bu tablo (Şekil 2.2), regresyon problemleri için çalışmanın temel özelliklerini ve kurulum sırasında gerçekleştirilen işlemleri özetler. Şimdi, bu bilgileri açıklayalım:

1. **Session id:** 123

• Modelin yeniden üretilebilirliği (reproducibility) için kullanılan sabit bir rastgele sayı (seed). Aynı session id kullanıldığında, aynı sonuçların elde edilmesini sağlar.

2. **Target:** charges

• Tahmin edilecek bağımlı değişken (hedef değişken). Bu örnekte, charges sütunu hedef olarak belirlenmiştir.

3. **Target type:** Regression

• Hedef değişkenin türü, sürekli bir değer olduğu için regresyon problemi olarak sınıflandırılmıştır.

4. **Original data shape:** (1338, 7)

• Orijinal veri setinin boyutu: 1338 satır ve 7 sütun.

5. **Transformed data shape:** (1338, 10)

• Veri ön işleme adımlarından sonra veri setinin boyutu. Özellikle kategorik değişkenlerin dönüşümü sonucu sütun sayısı artmıştır.

6. **Transformed train set shape:** (936, 10)

• Eğitim setinin dönüşüm sonrası boyutu: 936 satır ve 10 sütun.

7. **Transformed test set shape:** (402, 10)

• Test setinin dönüşüm sonrası boyutu: 402 satır ve 10 sütun.

8. **Numeric features:** 3

• Veri setindeki sayısal özelliklerin sayısı.

9. **Categorical features:** 3

• Veri setindeki kategorik özelliklerin sayısı.

10. **Preprocess:** True

• Veri ön işleme (preprocessing) adımlarının uygulandığını belirtir.

11. **Imputation type:** simple

• Eksik veri doldurma (impute) yöntemi, basit doldurma olarak seçilmiştir.

12. **Numeric imputation:** mean

• Sayısal değişkenlerde eksik değerler, ortalama (mean) ile doldurulmuştur.

13. **Categorical imputation:** mode

• Kategorik değişkenlerde eksik değerler, en sık tekrar eden değer (mode) ile doldurulmuştur.

14. **Maximum one-hot encoding:** 25

• Bir kategorik değişkene uygulanacak maksimum one-hot encoding sınıf sayısı. 25'ten fazla kategori varsa sınırlandırılır.

15. **Encoding method:** None

• Ekstra bir kodlama yöntemi uygulanmamıştır.

16. **Fold Generator:** KFold

• Çapraz doğrulama (cross-validation) sırasında kullanılacak yöntem, K-Fold olarak seçilmiştir.

17. **Fold Number:** 10

• Çapraz doğrulama için kat sayısı. Veri 10 parçaya bölünerek model eğitilir ve doğrulanır.

18. **CPU Jobs:** -1

• Model eğitimi sırasında kullanılacak işlemci çekirdeği sayısı. -1, tüm mevcut işlemci çekirdeklerinin kullanılacağını belirtir.

19. **Use GPU:** False

• GPU kullanımının kapalı olduğunu belirtir. GPU kullanılmaz.

20. **Log Experiment:** False

• Deneyin herhangi bir izleme aracına (örneğin MLflow) kaydedilmediğini belirtir.

21. **Experiment Name:** reg-default-name

• Deney için belirlenen isim. Varsayılan olarak reg-default-name atanmıştır.

22. **USI:** 9111

• Deneye özgü bir benzersiz kimlik numarası (Unique Session ID) otomatik olarak oluşturulmuştur.

	Description	Value
0	Session id	123
1	Target	charges
2	Target type	Regression
3	Original data shape	(1338, 7)
4	Transformed data shape	(1338, 10)
5	Transformed train set shape	(936, 10)
6	Transformed test set shape	(402, 10)
7	Numeric features	3
8	Categorical features	3
9	Preprocess	True
10	Imputation type	simple
11	Numeric imputation	mean
12	Categorical imputation	mode
13	Maximum one-hot encoding	25
14	Encoding method	None
15	Fold Generator	KFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	reg-default-name
21	USI	9111

Şekil 2.2 *Regresyon için varsayılan ayarlar*

PyCaret, kullanıcıların ihtiyaçlarına göre iki farklı API türü sunar: Fonksiyonel API ve Nesne Yönelimli API. Özellikle Nesne Yönelimli (OO) API, bir sınıfın kullanımıyla daha modüler ve yapılandırılmış bir yaklaşım sağlar. Bu yöntem, sınıfa ait metotlar aracılığıyla modelleme sürecinin her aşamasını kolayca takip etmeye ve kontrol etmeye olanak tanır. Büyük projelerde daha esnek bir kullanım sunarak model oluşturma, ayarlama ve değerlendirme işlemlerini sistematik bir şekilde gerçekleştirmenizi sağlar.

Bu yöntem, özellikle farklı veri setleri veya modeller üzerinde tekrarlanabilir süreçler oluşturmak isteyen kullanıcılar için idealdir.

```
# Liste 2.3 Regresyon uygulamasında deney ortamının oluşturulması
from pycaret.regression import RegressionExperiment
exp = RegressionExperiment()
```

Liste 2.3 *Sınıflandırma Deneysel ortamının oluşturulması ve tip testi*

Liste 2.3'teki, işlemin ardından tip testi sonucu "*pycaret.regression.oop.RegressionExperiment*" olmalıdır. Deneysel ortamın *exp* nesnesi üzerinden kurulumu Liste 2.4'teki gibi yapılabilmektedir.

```
# Liste 2.4 exp deneysel ortamının kurulumu
exp.setup(data, target = 'charges', session_id = 123)
```

Liste 2.4 *Deneysel ortamda sınıflandırma problem tipinin belirlenmesi*

Deneysel ortamın hazırlanmasının ardından model eğitim parametreleri Şekil 2.3'teki gibi sunulmaktadır.

	Description	Value
0	Session id	123
1	Target	charges
2	Target type	Regression
3	Original data shape	(1338, 7)
4	Transformed data shape	(1338, 10)
5	Transformed train set shape	(936, 10)
6	Transformed test set shape	(402, 10)
7	Numeric features	3
8	Categorical features	3
9	Preprocess	True
10	Imputation type	simple
11	Numeric imputation	mean
12	Categorical imputation	mode
13	Maximum one-hot encoding	25
14	Encoding method	None
15	Fold Generator	KFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	reg-default-name
21	USI	8662

Şekil 2.3 *Eğitilecek sistem parametreleri*

Parametreleri kısaca özetleyecek olursak;

1. **Session id:** 123 – Tekrar üretilebilirliği sağlamak için kullanılan rastgele sayı (seed).

2. **Target:** charges – Tahmin edilmesi hedeflenen bağımlı değişken.

3. **Target type:** Regression – Hedef değişkenin türü, sürekli bir değer olduğu için regresyon problemi olarak belirlenmiş.

4. **Original data shape:** (1338, 7) – Ham verinin orijinal boyutu; 1338 satır ve 7 sütun.

5. **Transformed data shape:** (1338, 10) – Dönüşümler uygulandıktan sonra veri setinin boyutu; sütun sayısı artmış (muhtemelen kategorik

değişkenler için dönüşüm yapılmış).

6. Transformed train set shape: (936, 10) – Eğitim setinin dönüşüm sonrası boyutu; 936 satır ve 10 sütun.

7. Transformed test set shape: (402, 10) – Test setinin dönüşüm sonrası boyutu; 402 satır ve 10 sütun.

8. Numeric features: 3 – Sayısal özelliklerin sayısı.

9. Categorical features: 3 – Kategorik özelliklerin sayısı.

10. Preprocess: True – Veri ön işleme adımlarının uygulandığını belirtir.

11. Imputation type: simple – Eksik veri doldurma yöntemi basit olarak seçilmiştir.

12. Numeric imputation: mean – Sayısal değişkenlerde eksik değerler ortalama ile doldurulmuştur.

13. Categorical imputation: mode – Kategorik değişkenlerde eksik değerler mod (en sık tekrar eden değer) ile doldurulmuştur.

14. Maximum one-hot encoding: 25 – Bir kategorik değişkene uygulanacak maksimum one-hot encoding sayısı.

15. Encoding method: None – Ek bir kodlama yöntemi uygulanmamış.

16. Fold Generator: KFold – Modelin doğrulama (validation) işlemi için K-Fold çapraz doğrulama yöntemi kullanılmıştır.

17. Fold Number: 10 – Çapraz doğrulamada veriler 10 katmana bölünmüştür.

18. CPU Jobs: -1 – Tüm işlemci çekirdeklerinin kullanıldığını belirtir.

19. Use GPU: False – GPU kullanılmamıştır.

20. Log Experiment: False – Deneyin kayıt altına alınmadığını gösterir.

21. Experiment Name: reg-default-name – Deneye verilen isim.

22. USI: 8662 – Deney için otomatik olarak oluşturulan benzersiz kimlik numarası.

Bu bilgiler, model oluşturma ve değerlendirme sürecinde kullanılan veri setinin detaylarını, uygulanan ön işleme adımlarını ve çapraz doğrulama ayarlarını açık bir şekilde özetlemektedir. Böylece yapılan işlemler tekrar üretilebilir ve analiz edilebilir hale gelir. Sonraki aşamada bu ayar-

lara göre sistem eğitimi gerçekleştirilecektir.

2.3 Modellerin Eğitilmesi

PyCaret'in güçlü araçlarından biri olan *compare_models* fonksiyonu, kütüphanede bulunan tüm tahminleyicileri (*estimators*) aynı anda eğitip çapraz doğrulama (*cross-validation*) ile performanslarını değerlendirir. Bu işlem sonucunda, her bir algoritmanın ortalama çapraz doğrulama skorlarını içeren bir tablo oluşturulur ve en iyi performansı gösteren modeller sıralanır.

Performans değerlendirmesinde kullanılan metrikler, get_metrics fonksiyonu ile görüntülenebilir. Bununla birlikte, özel ihtiyaçlarınıza göre metrik eklemek veya mevcut metrikleri kaldırmak için *add_metric* ve *remove_metric* fonksiyonlarını kullanabilirsiniz.

Normalde, her bir algoritmayı ayrı ayrı seçip eğitmek oldukça zaman alıcı bir süreçtir. Ancak PyCaret, kütüphanesindeki tüm regresyon algoritmalarını tek bir adımda eğiterek bu süreci büyük ölçüde hızlandırır. Sonuçları bir performans tablosu olarak sunar ve böylece hangi algoritmanın en iyi sonucu verdiğini hızlıca belirlemenizi sağlar (Liste 2.5).

```
# Liste 2.5 Modellerin temel performans karşılaştırması ve en başarılısının best'e atanması
best = compare_models()
```

Liste 2.5 *Varsayılan Sınıflandırma algoritmaları ile model eğitimi*

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	2701.9927	23548981.3626	4832.9682	0.8320	0.4447	0.3137	0.0510
rf	Random Forest Regressor	2771.4583	25416502.3827	5028.6343	0.8172	0.4690	0.3303	0.0820
lightgbm	Light Gradient Boosting Machine	2992.1828	25521038.3331	5042.0978	0.8149	0.5378	0.3751	0.0910
et	Extra Trees Regressor	2833.3624	28427844.2412	5305.6516	0.7991	0.4877	0.3363	0.0680
ada	AdaBoost Regressor	4316.0568	29220505.6498	5398.4561	0.7903	0.6368	0.7394	0.0310
lar	Least Angle Regression	4303.5559	38388058.4578	6176.5920	0.7306	0.5949	0.4433	0.0290
llar	Lasso Least Angle Regression	4303.7694	38386824.2786	6176.4846	0.7306	0.5952	0.4434	0.0270
br	Bayesian Ridge	4311.2349	38391950.0874	6176.8896	0.7306	0.5910	0.4447	0.0300
ridge	Ridge Regression	4317.6984	38396435.9578	6177.2329	0.7306	0.5891	0.4459	0.0250
lasso	Lasso Regression	4303.7697	38386797.6709	6176.4824	0.7306	0.5952	0.4434	0.5510
lr	Linear Regression	4303.5559	38388058.4578	6176.5920	0.7306	0.5949	0.4433	0.9490
huber	Huber Regressor	3464.1932	48883463.7074	6969.6284	0.6538	0.4891	0.2210	0.0350
dt	Decision Tree Regressor	3383.4916	47823199.0729	6895.7016	0.6497	0.5602	0.4013	0.0280
par	Passive Aggressive Regressor	4537.0122	67346309.9218	8142.7826	0.5422	0.5276	0.3207	0.0290
en	Elastic Net	7372.5238	90450782.5713	9468.3193	0.3792	0.7342	0.9184	0.0280
omp	Orthogonal Matching Pursuit	9089.9268	133439413.5272	11488.4238	0.0884	0.8790	1.1514	0.0290
knn	K Neighbors Regressor	8007.7997	131387268.8000	11425.3695	0.0859	0.8535	0.9232	0.0290
dummy	Dummy Regressor	9192.5418	148516792.8000	12132.4733	-0.0175	1.0154	1.5637	0.0260

Şekil 2.4 *Eğitilecek sistem parametreleri*

Deneysel ortam (*exp*) ayarlarına göre sistemin karşılaştırması için Liste 2.6'deki kodu çalıştırdığımızda Şekil 2.5'teki sonuç ile karşılaşırız.

```
# Liste 2.6 Modellerin OOP deneysel ortam ayarlarına göre karşılaştırılması
exp.compare_models()
```

Liste 2.6 *Nesne kullanarak deneysel eğitim ortamı oluşturma*

Fonksiyonel ve OOP API arasındaki çıktının tutarlı olduğunu unutmayın Şekil 2.5 ve 2.4'ün aynı olması nesne tabanlı programlanın aynı özelliklerde olmasıdır. Ancak her nesne yeni bir özellikler kümesi oluşturacaktır. Bu aşamadan sonraki kodlar nesne üzerinde çalıştırılarak (fonksiyonel API kullanılarak) gerçekleştirilecektir.

Regresyon problemlerinde sınıflandırma problemlerindeki performans kriterleri kullanılamaz ve farklı kriterlerin tanımlanması gerekmektedir. Bu performans parametreleri ve açıklamaları model analiz edilirken incelenecektir.

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	2701.9927	23548981.3626	4832.9682	0.8320	0.4447	0.3137	0.0460
rf	Random Forest Regressor	2771.4583	25416502.3827	5028.6343	0.8172	0.4690	0.3303	0.0850
lightgbm	Light Gradient Boosting Machine	2992.1828	25521038.3331	5042.0978	0.8149	0.5378	0.3751	0.1030
et	Extra Trees Regressor	2833.3624	28427844.2412	5305.6516	0.7991	0.4877	0.3363	0.0710
ada	AdaBoost Regressor	4316.0568	29220505.6498	5398.4561	0.7903	0.6368	0.7394	0.0290
lar	Least Angle Regression	4303.5559	38388058.4578	6176.5920	0.7306	0.5949	0.4433	0.0280
llar	Lasso Least Angle Regression	4303.7694	38386824.2786	6176.4846	0.7306	0.5952	0.4434	0.0290
br	Bayesian Ridge	4311.2349	38391950.0874	6176.8896	0.7306	0.5910	0.4447	0.0270
ridge	Ridge Regression	4317.6984	38396435.9578	6177.2329	0.7306	0.5891	0.4459	0.0300
lasso	Lasso Regression	4303.7697	38386797.6709	6176.4824	0.7306	0.5952	0.4434	0.0280
lr	Linear Regression	4303.5559	38388058.4578	6176.5920	0.7306	0.5949	0.4433	0.6130
huber	Huber Regressor	3464.1932	48883463.7074	6969.6284	0.6538	0.4891	0.2210	0.0320
dt	Decision Tree Regressor	3383.4916	47823199.0729	6895.7016	0.6497	0.5602	0.4013	0.0270
par	Passive Aggressive Regressor	4537.0122	67346309.9218	8142.7826	0.5422	0.5276	0.3207	0.0330
en	Elastic Net	7372.5238	90450782.5713	9468.3193	0.3792	0.7342	0.9184	0.0280
omp	Orthogonal Matching Pursuit	9089.9268	133439413.5272	11488.4238	0.0884	0.8790	1.1514	0.0300
knn	K Neighbors Regressor	8007.7997	131387268.8000	11425.3695	0.0859	0.8535	0.9232	0.0320
dummy	Dummy Regressor	9192.5418	148516792.8000	12132.4733	-0.0175	1.0154	1.5637	0.0250

Şekil 2.5 *Deneysel ortam ayarlarına göre algoritma performanslarının karşılaştırılması*

```
# Liste 2.7 Modellerin deneysel ortam ayarlarına göre karşılaştırılması
best = create_model('gbr')
```

Liste 2.7 *En başarılı modelin oluşturulması*

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	2651.0179	20312968.8483	4506.9911	0.8787	0.4506	0.3416
1	3045.5971	31786370.9368	5637.9403	0.8152	0.4560	0.2991
2	2526.0336	22219684.6628	4713.7761	0.7187	0.4909	0.3007
3	2975.2686	23090143.9001	4805.2205	0.8072	0.4866	0.3810
4	2847.7050	27163724.2109	5211.8830	0.7980	0.5103	0.3367
5	2580.5742	19052273.4486	4364.8910	0.8774	0.3340	0.2437
6	2366.1844	19241130.1231	4386.4713	0.8691	0.3504	0.2649
7	2673.5009	25507898.8469	5050.5345	0.8597	0.4415	0.2750
8	2325.6224	18564300.1288	4308.6309	0.8801	0.3889	0.2888
9	3028.4227	28551318.5202	5343.3434	0.8161	0.5379	0.4058
Mean	2701.9927	23548981.3626	4832.9682	0.8320	0.4447	0.3137
Std	249.8136	4311102.3822	437.4926	0.0488	0.0643	0.0491

Şekil 2.6 *En başarılı modelin k=10 için fold işlemlerindeki performans kriterleri*

2.4 Model Analizi

Regresyon modellerinin başarısını değerlendirmek için çeşitli performans kriterleri kullanılır. Bu kriterler arasında en yaygın olanlar, R^2 (determinasyon katsayısı), MAE (Ortalama Mutlak Hata), MSE (Ortalama Kare Hata) ve RMSE (Kök Ortalama Kare Hata) gibi metriklerdir. R^2, modelin veriyle ne kadar iyi uyum sağladığını ölçerken, MAE, MSE ve RMSE, tahmin edilen değerlerle gerçek değerler arasındaki farkı farklı şekillerde hesaplar. Bu metrikler, modelin doğruluğunu, hassasiyetini ve genel performansını anlamak için temel ölçütlerdir ve model seçiminde kritik bir rol oynar.

1. Mean Absolute Error (MAE)

MAE, tahmin edilen değerlerin gerçek değerlerden ne kadar sapma gösterdiğini ölçen bir hata metriğidir. Modelin tahminlerindeki ortalama mutlak hatayı hesaplar ve tüm sapmaları pozitif olarak değerlendirir.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\text{Gerçek Değer}_i - \text{Tahmin Değeri}_i|$$

MAE, hataların büyüklüğünü anlaşılır ve doğrudan birim cinsinden gösterdiği için kolay yorumlanabilir. Değeri, modelin tahminlerinin gerçeğe ne kadar yakın olduğunu ortaya koyar. Özellikle sıfıra yakın bir MAE, modelin yüksek doğruluğa sahip olduğunu gösterir. Ancak, hataların büyüklüğünü gösterirken yönlerini dikkate almaz ve daha büyük hataları RMSLE gibi metrikler kadar cezalandırmaz.

2. Mean Squared Error (MSE)

MSE, tahmin edilen değerler ile gerçek değerler arasındaki farkların karesinin ortalamasını hesaplayarak bir modelin hata miktarını ölçer.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\text{Gerçek Değer}_i - \text{Tahmin Değeri}_i)^2$$

MSE, hataları kare alarak büyütür, bu da büyük sapmaların daha fazla cezalandırılmasına yol açar. Bu özelliği sayesinde, modelin büyük hatalar yapma eğilimi olup olmadığını anlamak için yararlıdır. Değeri sıfıra ne kadar yakınsa, model o kadar isabetli tahminlerde bulunur. Ancak, MSE'nin birimi tahmin edilen değişkenin karesi cinsindedir, bu yüzden yorumlanması bazen zor olabilir. RMS (Root Mean Squared Error) ile birlikte kullanılarak daha kolay anlaşılabilir hale getirilebilir.

3. Root Mean Squared Error (RMSE)

RMSE, MSE'nin karekökü alınarak elde edilir ve tahmin edilen değerlerle gerçek değerler arasındaki hata büyüklüğünü ölçer.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{Gerçek Değer}_i - \text{Tahmin Değeri}_i)^2}$$

RMSE, hataları aynı birimde ifade ettiği için (örneğin, metre, kilogram gibi), yorumlaması daha kolaydır. Büyük hatalara karşı daha duyarlıdır; bu nedenle, modelin büyük sapmalar yapıp yapmadığını anlamak için kullanışlıdır. Değeri sıfıra ne kadar yakınsa, modelin tahminlerinin doğruluğu o kadar yüksektir. RMSE, özellikle sürekli değişkenlerin tahmin edildiği regresyon problemlerinde yaygın olarak kullanılır.

4. R-Squared (R^2)

Belirleme katsayısı olarak adlandırılır ve modelin hedef değişkendeki varyansı ne kadar açıkladığını gösterir. R^2 değeri 0 ile 1 arasında bir değerdir. 1 modeli mükemmel bir uyum sağladığını, 0 ise hiçbir açıklayıcılığı olmadığını gösterir.

$$R^2 = 1 - \frac{\text{Toplam Hata Kareleri (SSR)}}{\text{Toplam Toplam Varyans (SST)}}$$

5. Root Mean Squared Logarithmic Error (RMSLE)

RMSLE, tahmin edilen değerlerle gerçek değerler arasındaki hatayı logaritmik ölçekte hesaplayan bir hata metriğidir.

$$\text{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(\text{Tahmin Değeri}_i + 1) - \log(\text{Gerçek Değer}_i + 1))^2}$$

Bu metrik, tahmin edilen ve gerçek değerler arasındaki oransal farklılıkları ölçmek için kullanılır. Özellikle tahmin değerleri ve gerçek değerler arasında büyük farklar olduğunda veya değerler küçük ölçeklerde değişiklik gösterdiğinde uygundur.

RMSLE, hatalara logaritmik dönüşüm uyguladığı için, büyük hataları küçültür ve küçük hataların önemini artırır. Bu nedenle, pozitif değerler arasında çalışan ve küçük sapmaların daha önemli olduğu projelerde sıkça tercih edilir. Tahmin edilen ve gerçek değerler arasındaki oranları değerlendirmek istediğiniz durumlar için kullanışlı bir metrik sunar. Değeri sıfıra ne kadar yakınsa, modelin tahmin performansı o kadar iyidir.

6. Mean Absolute Percentage Error (MAPE)

MAPE, bir modelin tahmin ettiği değerlerin gerçek değerlere göre yüzdesel hatasını ölçen bir performans metriğidir. Her tahminin hata oranını (mutlak yüzde hatasını) hesaplar ve bunların ortalamasını alır.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\text{Gerçek Değer}_i - \text{Tahmin Değeri}_i}{\text{Gerçek Değer}_i}\right| \times 100$$

MAPE, tahminlerin genel doğruluğunu anlamak için kullanışlıdır ve sonucu genellikle yüzde olarak ifade edilir. Ancak, gerçek değerlerin sıfıra yakın olduğu durumlarda sorun yaşayabilir, çünkü yüzdesel hata sonsuz olabilir veya yanlış anlamalar yaratabilir. MAPE, finansal analiz, talep tahmini ve işletme verilerinde yaygın olarak kullanılır.

Özet olarak;

- **MAE**: Hataların mutlak ortalamasıdır, yorumlaması kolaydır.
- **MSE**: Hataların karesel ortalamasıdır, büyük hatalara duyarlıdır.
- **R^2**: Modelin açıklayıcısıdır. 1'e yakın olması istenir.
- **RMSE**: Hataların karesel ortalamasının kareköküdür, yorumlaması kolaydır.
- **RMSLE:** RMSLE, logaritmik dönüşüm sayesinde büyük tahmin hatalarını yumuşatır ve küçük hataları daha az cezalandırır. Ancak negatif veya sıfır hedef değerlerle çalışamaz.
- **MAPE**: Hataların yüzde cinsinden ortalamasıdır.

Bu kriterler arasından seçim yapmak, verinin yapısına ve probleme bağlıdır. Örneğin, büyük hataların cezalandırılması isteniyorsa **RMSE** tercih edilirken, hata yüzdelerinin incelenmesi gerekiyorsa **MAPE** kullanılır.

Performans kriterlerinin temel tanımlarını inceledikten sonra en başarılı model olan Gradient Boosting Regressor (gbr) algoritması için elde edilen performans kriterlerini inceleyelim. Gradient Boosting Regressor (GBR) algoritmasının performansını aşağıdaki kriterlere göre yorumlayalım:

1. **Mean Absolute Error (MAE)**: 2701.9927

MAE, tahmin edilen değerler ile gerçek değerler arasındaki farkların mutlak değerlerinin ortalaması olarak ifade edilebilir. GBR modeli için Ortalama hata yaklaşık 2702 birimdir. Bu, modelin tahminlerinin gerçek değerlere oldukça yakın olduğunu gösterir, ancak hata değerinin bağlamına bağlı olarak değerlendirilmelidir.

2. **Mean Squared Error (MSE)**: 23,548,981.3626

MSE, hataların karesinin ortalamasıdır. Büyük hataları daha fazla cezalandırır. MSE değeri oldukça büyük çıkmıştır. Bu, büyük hataların model tarafından üretildiğini gösteriyor olabilir. MSE'nin büyük olması, hataların bazı örneklerde yüksek olabileceğine işaret eder.

3. **Root Mean Squared Error (RMSE)**: 4832.9682

RMSE, MSE'nin karekökü alınarak hesaplanır. Hatanın büyüklüğünü gerçek değerlerle aynı birimde ifade eder. Ortalama hata yaklaşık 4833 birimdir. RMSE değeri MAE'den daha yüksek olduğu için, modelin büyük tahmin hataları ürettiği sonucuna varabiliriz. RMSE'nin büyük çıkması, MSE'deki hataların karelenmesinden kaynaklanır.

4. **R-Squared (R^2)**: 0.8320

R^2, modelin bağımsız değişkenlerle hedef değişkeni ne kadar açıkladığını ölçer. Değer 0 ile 1 arasında değişir. Modelin performansı %83 açıklama oranı ile oldukça iyidir. Bu değer, GBR modelinin hedef değişkendeki değişimin büyük bir kısmını açıkladığını gösteriyor. Kolay anlaşılması açısından bunu sınıflandırmadaki doğruluğa (accuracy) benzetebiliriz.

5. **Root Mean Squared Logarithmic Error (RMSLE)**: 0.4447

RMSLE, logaritmik hata ölçüsüdür. Büyük değerlerin etkisini azaltır ve oransal hatalara odaklanır.

RMSLE değeri 0.4447'dir. Bu, tahminlerin logaritmik ölçeğinde düşük bir hata olduğunu gösterir. Model büyük değerlerdeki hataları yumuşatıyor ve orantısal tahmin doğruluğu açısından iyi bir performans gösteriyor denebilir.

6. **Mean Absolute Percentage Error** (MAPE): 0.3137

MAPE, ortalama yüzde hata oranını verir. Basitçe daha düşük değer daha iyidir. MAPE değeri %31.37'dir. Bu, modelin tahminlerinin ortalama olarak %31.37 oranında hata içerdiğini gösterir. Hata oranı makul kabul edilebilir seviyede, ancak modelin doğruluğunu iyileştirmek için hala parametre düzenleme ve diğer performans arttırma imkanı vardır.

Genel Performans Yorumlaması:

- R^2 değeri %83 ile oldukça iyi bir performans sergilemiştir. Model, hedef değişkenin büyük bir kısmını açıklayabilimektedir.
- MAE ve RMSE değerleri hata büyüklüğünün ortalama 2702 ve 4833 birim civarında olduğunu gösteriyor. RMSE'nin MAE'den yüksek olması, bazı büyük tahmin hatalarının mevcut olduğunu ortaya koymaktadır.
- RMSLE ve MAPE değerleri ise logaritmik hata ölçüsü ve yüzde hata açısından modelin makul seviyede doğruluk sağladığını göstermektedir.
- MSE değeri büyük olsa da R^2'nin yüksekliği modelin genel olarak güçlü bir tahminleme yeteneğine sahip olduğunu ortaya koymaktadır.

Genel olarak performans kriterlerini özetlersek;

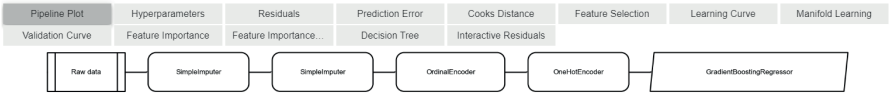
GBR algoritması, veriye oldukça iyi uyum sağlamış ve %83'lük bir açıklama oranı ile güçlü performans göstermektedir. Ancak büyük tahmin hatalarını azaltmak için modelin hiperparametre optimizasyonu veya özellik mühendisliği gibi iyileştirme adımları uygulanabilir. Özellikle RMSE ve MAPE oranlarını düşürmek modelin performansını daha da artıracaktır.

Performans kriterlerini grafiklerle daha görsel bir açıdan incelemek için model değerlendirme gerçekleştirilmelidir. Grafik çizdirmek algoritma ve problem tipine göre uzmanlık gerektirebilirken PyCaret bu konuda bir satır koda ile birazdan bahsedeceğimiz grafikleri otomatik olarak çizdirebilmektedir (Liste 2.8).

```
# Liste 2.8 Grafiklerle Performans Analizi
evaluate_model(best)
```

Liste 2.8 *Grafiklerle Performans analizi*

İlk grafiğimiz PyCaret'in iş akışını (pipeline) gösteren yapısı (Şekil 2.7), ham veriden başlayarak modelleme sürecindeki tüm önemli adımları bir araya getirir. Her adım, veriyi bir önceki adımdan daha temiz, anlaşılabilir ve tahminlemeye uygun bir hale getirir. Bu otomatikleştirilmiş süreç, kullanıcıların veri ön işleme ve modelleme arasında zaman kaybetmeden sonuçlara odaklanmasını sağlar.



Şekil 2.7 *performans kriterleri*

1. Raw Data (Ham Veri)

İşlem hattının başlangıç noktası, model eğitimi için kullanılan ham veri setidir. Bu veri genellikle eksik değerler, kategorik değişkenler ve diğer işlenmemiş özellikler içerir. Pipeline, bu ham veri üzerinde işlem yaparak modelin anlamlandırabileceği bir format oluşturur.

*Bu aşamada **Raw Data** (Ham Veri) toplanır ve işlenir.*

2. **SimpleImputer**

Bu adım, eksik verilerin doldurulması (imputation) için kullanılır. PyCaret, eksik değerleri sayısal değişkenlerde ortalama (mean), kategorik değişkenlerde ise mod (en sık tekrar eden değer) ile doldurur. Eksik veriler nedeniyle analiz yapılamayan sütunların doldurulması ve modelin eksiksiz bir veri setiyle eğitilmesi sağlanır.

*Bu aşamada eksik değerler **SimpleImputer** ile doldurulur.*

3. **Ordinal Encoder**

Kategorik değişkenlerin, sıralı bir mantıkla sayısal değerlere dönüştürülmesini sağlar. Örneğin, bir kategorik değişken [“low”, “medium”, “high”] değerlerini içeriyorsa, bunlar sırasıyla [1, 2, 3] gibi sayısal değerlere çevrilebilir. Bu şekilde modelin sayısal işlem yapabilmesi için sıralı kategorik değişkenleri anlamlandırması sağlanır.

*Bu aşamada sıralı kategorik değişkenler **Ordinal Encoder** ile dönüştürülür.*

4. **One Hot Encoder**

Bu adım, kategorik değişkenlerin, her bir kategori için ayrı bir sütun oluşturulmasıyla temsil edilmesini sağlar. Kategorik veriler arasında karşılaştırma yapılmasını ve modelin bu bilgileri öğrenmesini sağlar.

*Bu aşamada diğer kategorik değişkenler **One Hot Encoder** ile sayısal formatta temsil edilir.*

5. **Gradient Boosting Regressor**

Bu adım, işlenmiş veriyi kullanarak tahminleme yapan asıl modeldir. Gradient Boosting Regressor, bir ensemble yöntemidir ve zayıf tahminleyicileri (decision tree gibi) birleştirerek güçlü bir model oluşturur. İşlenmiş verilerden bağımlı değişkenin tahmin edilmesi ve modelin öğrenim sürecinin tamamlanmasıyla tamamlanır.

*Bu aşamada İşlenen veriler **Gradient Boosting Regressor** modeline iletilir ve tahminleme gerçekleştirilir.*

Pipeline Plot	Hyperparameters	Residuals	Prediction Error	Cooks Distance
Validation Curve	Feature Importance	Feature Importance...	Decision Tree	Interactive Residuals

	Parameters
alpha	0.9
ccp_alpha	0.0
criterion	friedman_mse
init	None
learning_rate	0.1
loss	squared_error
max_depth	3
max_features	None
max_leaf_nodes	None
min_impurity_decrease	0.0
min_samples_leaf	1
min_samples_split	2
min_weight_fraction_leaf	0.0
n_estimators	100
n_iter_no_change	None
random_state	123
subsample	1.0
tol	0.0001
validation_fraction	0.1
verbose	0
warm_start	False

Şekil 2.8 *Gradient Boosting Regressor Parametreleri (Hyperparameters)*

Gradient Boosting Regressor için gösterilen hiperparametreleri açıklayalım:

Temel Hiperparametreler

1. **alpha:** 0.9

Quantile kaybı için kullanılır. Hangi kuantili optimize edeceğini belirtir. Varsayılan değerler arasında yer alır.

2. **ccp_alpha:** 0.0

Complexity Parameter. Budama işlemi sırasında düğüm bölünmesi için maliyet karmaşıklığını belirler. Varsayılan olarak kapalı.

3. **criterion:** friedman_mse

Split kriteri. Friedman'ın Mean Squared Error (MSE) optimizasyonunu temel alır.

4. **learning_rate:**0.1

Modelin her iterasyondaki öğrenme hızını belirler. Düşük değerler daha yavaş ama daha iyi genelleme yapabilir.

5. **loss:**squared_error

Kayıp fonksiyonu. Regresyon için kullanılan varsayılan kayıp fonksiyonudur (MSE).

6. **max_depth:**3

Her bir karar ağacının maksimum derinliği. Overfitting'i önlemek için sınırlama getirir.

7. **max_features:**None

Split sırasında değerlendirilecek maksimum özellik sayısı. Varsayılan olarak tüm özellikler kullanılır.

8. **max_leaf_nodes:**None

Her ağacın maksimum yaprak düğüm sayısı. Varsayılan olarak sınırlama yoktur.

9. **min_impurity_decrease:**0.0

Split işlemi sırasında saflık (impurity) azaltımına getirilen minimum eşik.

10. **min_samples_leaf:**1

Bir yaprak düğümde bulunması gereken minimum örnek sayısı.

11. **min_samples_split:**2

Split işlemi yapılması için gereken minimum örnek sayısı.

12. **min_weight_fraction_leaf:**0.0

Yaprak düğümdeki minimum ağırlık fraksiyonu.

13. **n_estimators:**100

Gradient Boosting algoritmasında kullanılacak toplam ağaç sayısı.

14. **n_iter_no_change:**None

Erken durdurma için kullanılan hiperparametre. Değişim olmayan iterasyon sayısını belirtir.

15. **random_state:**123

Rastgelelik kontrolü için kullanılan sabit.

16. **subsample:**1.0

Her iterasyonda kullanılacak veri seti oranı. 1.0 tüm veri setini kullanır.

17. **tol:**0.0001

Kayıp fonksiyonundaki değişikliklerin durdurulması için tolerans değeri.

18. validation_fraction:0.1

Erken durdurma sırasında doğrulama için kullanılan veri seti oranı.

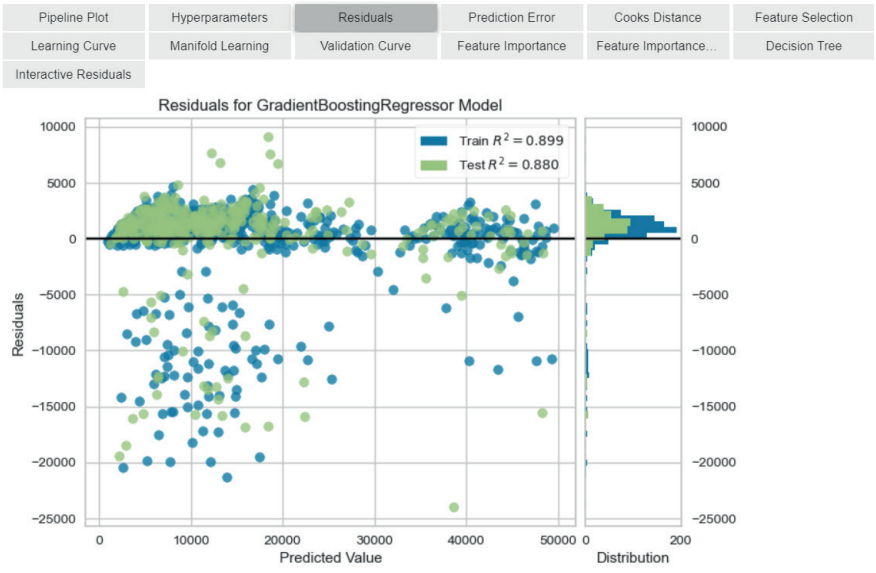
19. verbose:0

Eğitim sırasında çıktı seviyesi. 0, sessiz; daha yüksek değerler daha fazla bilgi verir.

20. warm_start: False

Önceki modelin fit sonuçlarını yeniden kullanıp kullanmama seçeneği.

Bu parametreler, Gradient Boosting Regressor'ın performansını optimize etmek ve farklı veri setlerine uyum sağlamasını sağlamak için ayarlanabilir.



Şekil 2.9 *Gradient Boosting Regressor Residual grafiği*

Gradient Boosting Regressor modeliyle yapılan regresyon analizine ait artık (residual) grafiği (Şekil 2.9) gösterilmiştir. Grafiği analiz ederek şu şekilde yorumlayabiliriz:

1. Grafik Açıklaması

X Ekseni (Predicted Value): Modelin tahmin ettiği değerler.

Y Ekseni (Residuals): Gerçek değer ile modelin tahmin ettiği değer arasındaki fark (artık).

Residual = Gerçek Değer - Tahmin Değeri.

R^2 Değerleri:

Train R^2 = 0.899: Eğitim veri setinde modelin açıklama gücünü temsil eder. Model, eğitim verisinin %89.9'unu açıklayabiliyor.

Test R^2 = 0.880: Test veri setinde modelin açıklama gücüdür. Model, test verisinin %88.0'ini açıklayabiliyor.

Artıkların Dağılımı: Sağda histogram olarak, artıkların frekans dağılımı gösterilmiş.

2. Artık Analizi

- **Merkezleme:** Artıklar çoğunlukla sıfır etrafında yoğunlaşmış durumda. Bu, modelin genellikle doğru tahminlerde bulunduğunu ve sistematik bir hata olmadığını gösterir.

- **Simetri:** Artıkların dağılımı genel olarak simetrik görünüyor, ancak bazı aşırı değerler (outliers) gözlemleniyor.

- **Homoskedastisite (Sabit Varyans):** Artıkların varyansı, tahmin edilen değerlere göre çok değişmiyor. Ancak bazı yüksek tahmin değerlerinde artıkların daha fazla yayıldığı gözleniyor. Bu, varyansın tam sabit olmadığını ama büyük ölçüde kabul edilebilir olduğunu gösterir.

- **Aşırı Değerler:** Bazı tahminlerde, özellikle -10000 ve -20000 bandında büyük artık değerleri görülüyor. Bu, modelin bazı örneklerde gerçek değerlerden oldukça farklı tahmin yaptığını gösterir. Bu durum, veri setindeki ekstrem durumlar veya modelin bu veri noktalarına yeterince uyum sağlayamamasından kaynaklanabilir.

3. Eğitim ve Test Performansı Karşılaştırması

- Eğitim setindeki R^2 değeri (%89.9), test setindeki R^2 değerine (%88.0) oldukça yakın. Bu durum, modelin aşırı öğrenmeden (overfitting) uzak olduğunu ve genelleme kabiliyetinin iyi olduğunu gösterir.

- Test setinde biraz daha düşük R^2 değeri, modelin test verisine eğitim verisinden biraz daha az uyum sağladığını, ancak farkın çok küçük olduğunu işaret eder.

4. Artık Dağılımının Özeti

- Sağ taraftaki histogram, artıkların çoğunun sıfıra yakın olduğunu ve bir kısmının da hafif pozitif veya negatif yönlerde dağıldığını gösteriyor.
- Dağılım genelde düzgün görünüyor, ancak birkaç uç değer bu dağılımı biraz bozuyor.

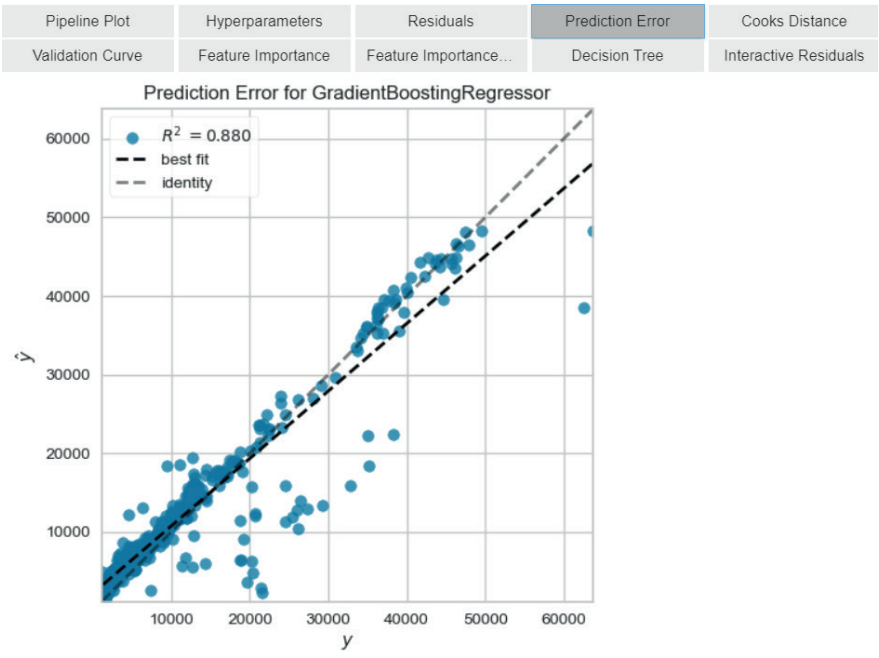
5. Genel Değerlendirme

Bu model, hem eğitim hem de test verisinde yüksek performans gösteriyor. R^2 değerleri ve artık analizi, modelin doğru bir şekilde tahminler yapabildiğini ve veri setindeki çoğu varyasyonu açıklayabildiğini gösteriyor.

Ancak:

- Aşırı artık değerlerin olduğu durumlar incelenmeli. Bu durumlar, belki de veri setindeki anormal gözlemlerden kaynaklanıyor olabilir.
- Modelin performansını artırmak için bu uç değerlerin etkisi incelenebilir veya veri temizliği yapılabilir.

Sonuç olarak, *Gradient Boosting Regressor* bu veri setinde oldukça iyi bir performans sergilemiştir.



Şekil 2.10 *Modelin eğitim aşamaları*

Prediction Error grafiği, Gradient Boosting Regressor modelinin tahmin performansını görselleştiren bir araçtır. Grafikte yer alan unsurları ve genel yorumu şu şekilde açıklayabiliriz:

Grafiğin Unsurları

1. Eksenler:

- **x ekseni (gerçek değerler, y)**: Gerçek hedef (bağımlı değişken) değerlerini ifade eder.
- **y ekseni (tahmin edilen değerler, y^)**: Model tarafından tahmin edilen değerleri ifade eder.

2. R^2 Değeri:

- Grafik sol üstte R^2= 0.880 olarak belirtilmiştir. Bu, modelin veriyi açıklama oranıdır. Burada model %88 oranında değişkenliği açıklayabiliyor, bu da oldukça iyi bir performans gösterir.

3. Çizgiler:

- **Identity (kimlik çizgisi):** Grafikteki düz siyah çizgi, mükemmel tahmin durumunu temsil eder. Eğer tüm noktalar bu çizgi üzerinde yer alsaydı, model hatasız bir performans göstermiş olurdu.
- **Best fit (en iyi uyum çizgisi):** Kesikli çizgi, tahmin edilen ve gerçek değerler arasındaki ilişkiyi gösterir. Bu çizginin eğimi, modelin genel tahmin eğilimini yansıtır.

4. Noktalar:

- Her bir nokta, modelin bir tahmin sonucunu temsil eder. Noktaların kimlik çizgisine olan yakınlığı, modelin tahmin doğruluğunu gösterir. Noktalar kimlik çizgisine ne kadar yakınsa tahmin o kadar doğru olur.

Genel olarak grafiği yorumlarsak;

1. Tahmin Başarısı:

- Noktaların büyük bir kısmı kimlik çizgisine yakın bir şekilde dağıldığından, model genel olarak doğru tahminler yapmıştır.
- R^2 değerinin 0.88 olması, modelin tahminlerinin veri üzerindeki varyansın büyük bir kısmını açıklayabildiğini göstermektedir.

2. Sapmalar:

- Bazı noktalar kimlik çizgisinden uzaklaşmaktadır. Bu, modelin belirli veri noktaları için tahmin hatası yaptığını gösterir.
- Özellikle üst değerlerde ve alt uçlarda sapmalar görülmektedir. Bu durum, modelin uç değerler veya nadir durumlarla başa çıkmakta zorlan-

dığını gösterebilir.

3. **Overfitting veya Underfitting Durumu:**

- Grafik genel olarak iyi bir uyum gösteriyor ve büyük sapmalar olmadığı için overfitting veya underfitting'in belirgin bir göstergesi yoktur.

- Eğitim ve test verisi arasındaki performans farkı minimum düzeyde görünüyor (residuals grafiği ile desteklenebilir).

4. **Geliştirme Alanları:**

- Özellikle uç noktalarda modelin performansı düşüyor gibi görünüyor. Bu, modelin uç değerleri daha iyi öğrenmesi için hiperparametre ayarlarının iyileştirilebileceğini gösterebilir.

Sonuç olarak Prediction Error grafiği (Şekil 2.10), *Gradient Boosting Regressor* modelinin oldukça iyi bir performans sergilediğini göstermektedir. Ancak uç değerlerde gözlenen sapmalar, modelin bu alanlarda daha iyi performans göstermesi için optimizasyon yapılabileceğini işaret etmektedir. Model, genel doğruluğu ve açıklayıcılığı bakımından başarılıdır.

Cook's Distance grafiği (Şekil 2.11), veri setindeki her bir gözlemin model üzerinde ne kadar etkili (influence) olduğunu değerlendiren bir analiz aracıdır. Bu grafiği yorumlarken aşağıdaki unsurları göz önünde bulundurabiliriz:

Grafiğin Unsurları

1. **X Eksenindeki Değerler (Instance Index):**

- Veri setindeki her bir gözlemi temsil eder.

- Grafikte gözlemler sıra numarasıyla (örneğin, 0'dan 900'e kadar) belirtilmiştir.

2. **Y Eksenindeki Değerler (Influence):**

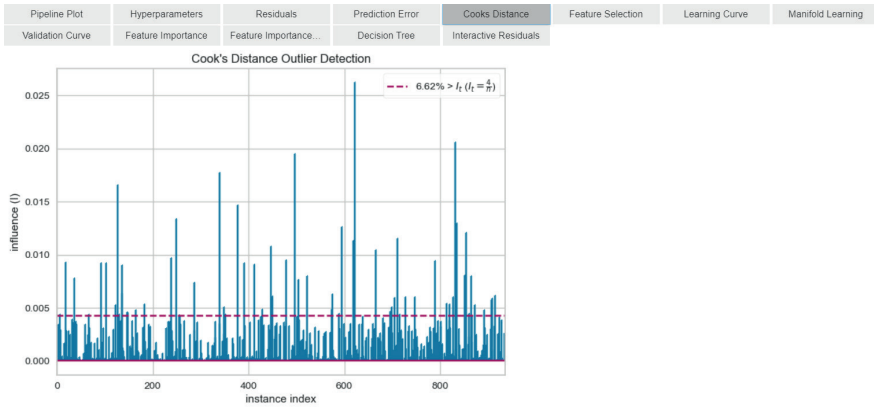
- Cook's Distance metriği ile her bir gözlemin model üzerindeki etkisini ifade eder.

- Yüksek bir Cook's Distance değeri, o gözlemin modelin parametreleri üzerinde daha fazla etkisi olduğu anlamına gelir.

3. **Kırmızı Kesikli Çizgi:**

- *It* (threshold) olarak belirlenen bir sınır değeri gösterir.

- Eğer bir gözlemin etkisi bu sınır değerinin üstündeyse, o gözlem bir **aşırı etkili veri (outlier)** veya **aşırı uç (high leverage point)** olarak değerlendirilir.



Şekil 2.11 *Cooks Distance grafiği*

Genel Yorum

1. **Aşırı Etkili Gözlemler:**

- Grafikte birkaç gözlem, kırmızı kesikli sınır çizgisini aşmaktadır. Bu gözlemler, modelin sonuçlarını önemli ölçüde etkileyen veri noktalarıdır.

- Belirlenen eşik değere göre veri setindeki gözlemlerin %6.62'sinin model üzerinde yüksek etkiye sahip olduğu belirtilmiştir.

2. **Modelin Hassasiyeti:**

- Sınırı aşan gözlemler modelin parametrelerini ve sonuçlarını ciddi şekilde etkileyebilir.

- Bu tür gözlemler, genellikle veri setinde uç değerleri veya aykırı verileri temsil eder. Bu gözlemlerin analizi yapılmalı ve gerekiyorsa modele etkisi azaltılmalıdır (örneğin, bu gözlemler çıkarılabilir, yeniden ağırlıklandırılabilir veya dönüşüm uygulanabilir).

3. **Grafik Dağılımı:**

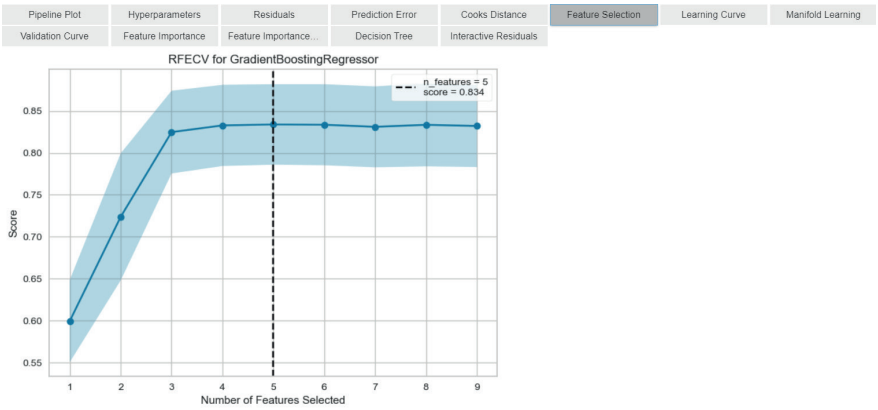
- Gözlemlerin büyük bir kısmının etkisi oldukça düşüktür (0.005 seviyesinin altında).

- Yalnızca birkaç veri noktası dikkat çekici bir etkiye sahiptir. Bu da modelin genelde dengeli bir şekilde öğrenim sağladığını, ancak bazı uç gözlemlerin dikkate alınması gerektiğini gösterir.

4. **Eşik Üstündeki Gözlemlerle Çalışma:**

- Modelin bu gözlemleri düzgün öğrenip öğrenmediği kontrol edilmelidir. Aykırı değerlerin çıkarılması ya da özel bir ağırlıklandırma yapılması, modelin performansını iyileştirebilir.

Sonuç olarak grafikteki (Şekil 2.10) düşük etkili çoğunluk, modelin genelde dengeli ve güvenilir tahminler yaptığını gösterir. Ancak sınırı aşan birkaç gözlem, model üzerinde yüksek etki bırakmaktadır. Bu gözlemler daha detaylı analiz edilmeli ve gerekirse aykırılık durumuna uygun işlemler yapılmalıdır. Bu tür bir iyileştirme modeli daha genel bir performansa yönlendirebilir.



Şekil 2.12 *Feature Selection grafiği*

Recursive Feature Elimination with *Cross-Validation (RFECV)* yönteminin bir *Gradient Boosting Regressor* modeli üzerinde uygulandığını gösteren grafik en iyi özellik sayısını belirlediğini Şekil 2.11'de göstermektedir.

Grafiğin Unsurları

1. **X Ekseni (Number of Features Selected):** Seçilen özellik (feature) sayısını ifade eder. Bu, modelin performansını değerlendirirken kullanılan farklı sayıda özelliği temsil eder. Örneğin, 1'den 9'a kadar olan değerler, her iterasyonda kullanılan özelliklerin sayısını gösteriyor.

2. **Y Ekseni (Score):** Modelin R^2 performans skorunu ifade eder. Bu skor, modelin seçilen özelliklerle ne kadar iyi tahmin yaptığını ölçer. Daha yüksek bir değer, daha iyi bir performansı temsil eder.

3. **Mavi Çizgi ve Gölgelik:** Mavi çizgi, her özellik sayısı için ortalama modeli performans skorunu gösterir. Mavi gölgelik ise skorların varyansını ifade eder, yani model performansındaki belirsizlik aralığını gösterir.

4. **Kesikli Siyah Çizgi:** En iyi model performansı için önerilen özellik sayısını belirtir.Bu grafikte, en iyi performans 5 özellik seçildiğinde elde edilmiştir (n_features = 5).

5. **En Yüksek Performans (Score):** 5 özellik seçildiğinde, modelin en iyi R^2 skoru yaklaşık olarak **0.834** olarak verilmiştir.

Grafiği yorumlayalım;

1. **Performans Artışı:** Özellik sayısı 1'den 5'e doğru arttıkça model performansında belirgin bir artış gözlemleniyor. Bu, daha fazla özellik kullanıldıkça modelin daha iyi bir tahmin yapabildiğini gösteriyor.

2. **Optimal Özellik Sayısı:** 5 özelliğin seçilmesi, modelin en yüksek performansı elde ettiği noktadır. Daha fazla özellik eklenmesi model performansına anlamlı bir katkı sağlamamaktadır.

3. **Varyans:** Mavi gölgeli alanın genişliği, özellikle az sayıda özellik kullanıldığında model performansında daha fazla belirsizlik olduğunu gösteriyor. Bu da daha az özellik seçildiğinde modelin performansının daha değişken olabileceğine işaret eder.

4. **Özellik Fazlalığı:** 5'ten fazla özellik seçildiğinde (örneğin, 6 veya daha fazla), modelin performansı artmamakta, aksine sabit veya hafifçe düşüş eğilimi göstermektedir. Bu durum, fazla özellik kullanmanın gereksiz olduğunu ve potansiyel olarak aşırı öğrenme (overfitting) riskini artırabileceğini işaret eder.

Bu grafiğe (Şekil 2.12) göre, en iyi model performansı için 5 özellik seçilmelidir. Özellik seçimi süreci, modelin karmaşıklığını azaltmak ve daha iyi genelleme sağlayabilmek için etkili bir yaklaşımdır. 5 özellikten fazlasını kullanmak modelin performansına anlamlı bir katkı sağlamadığı için özellik azaltma stratejisi uygun bir yöntemdir.

Şekil 2.13 *Öğrenme Eğrisi (Learning Curve) grafiği*

Öğrenme eğrisi grafiği Şekil (2.13), *Gradient Boosting Regressor* modelinin eğitim ve çapraz doğrulama (cross validation) süreçlerinde performansını değerlendirmektedir.

Grafiğin Unsurları

1. **X Ekseni (Training Instances):**

- Eğitim için kullanılan veri örneği sayısını ifade eder.
- Sol tarafta daha az veriyle yapılan eğitim, sağa doğru gidildikçe daha fazla veriyle yapılan eğitim sonuçlarını gösterir.

2. **Y Ekseni (Score):**

- Modelin performans metriğini (örneğin, R^2 veya başka bir doğruluk metriği) ifade eder.
- Daha yüksek skor, modelin daha iyi performans gösterdiğini ifade eder.

3. **Mavi Çizgi (Training Score):**

- Eğitim verisinde modelin elde ettiği skorları gösterir. Modelin eğitildiği verideki doğruluğu temsil eder.

4. **Yeşil Çizgi (Cross Validation Score):**

- Çapraz doğrulama (validation) verisindeki model performansını gösterir. Modelin genelleme kabiliyetini temsil eder.

5. **Gölgeli Alanlar:**

- Mavi ve yeşil çizgilerin etrafındaki alanlar, ilgili skorların varyansını (güven aralığını) ifade eder. Daha geniş bir alan, skorların daha değişken olabileceğini gösterir.

Grafik Yorumu

1. **Eğitim Skoru (Training Score):**

- Mavi çizgi, az sayıda eğitim örneğiyle başladığında çok yüksek (yaklaşık 0.95) bir performans sergiliyor. Ancak, eğitim veri seti büyüdükçe skor azalmaya başlıyor ve bir plato seviyesine ulaşıyor.
- Bu, modelin küçük veri setinde mükemmel şekilde uyum sağladığını (overfitting eğilimi) ancak veri arttıkça genelleme yapmaya çalıştığını gösteriyor.

2. **Çapraz Doğrulama Skoru (Cross Validation Score):**

- Yeşil çizgi, başlangıçta daha düşük bir skorla (yaklaşık 0.8) baş-

lıyor ancak eğitim veri seti büyüdükçe kademeli olarak artıyor ve daha stabil hale geliyor.

- Bu, daha fazla veriyle modelin genelleme kabiliyetinin iyileştiğini gösteriyor.

3. Eğitim ve Çapraz Doğrulama Skorları Arasındaki Fark:

- Mavi ve yeşil çizgi arasındaki boşluk, modelin overfitting durumunu gösterir.

- Grafiğin sağ tarafına doğru (daha fazla eğitim örneği kullanıldığında), bu fark küçülüyor, bu da modelin genelleme performansının daha iyi hale geldiğini gösteriyor.

4. Genelleme Performansı:

- Çapraz doğrulama skoru, hiçbir zaman eğitim skoruna eşit olmuyor, ancak yeterince yakınlaşıyor. Bu, modelin hâlâ mükemmel bir genelleme yapamadığını, ancak kabul edilebilir bir performans gösterdiğini ifade eder.

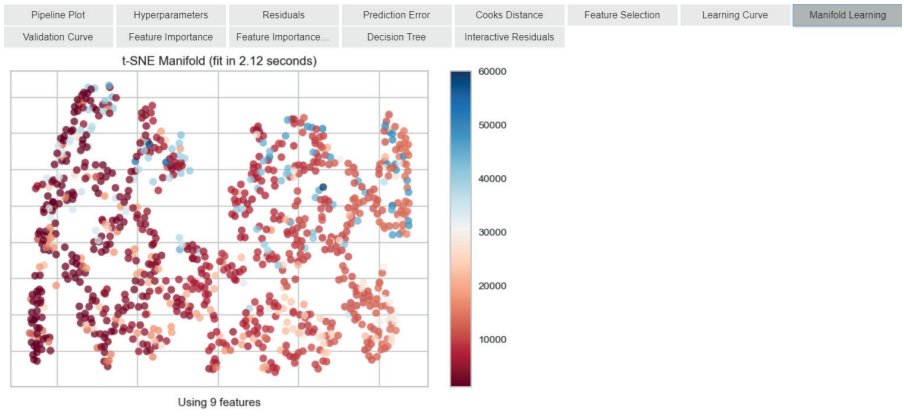
Sonuç

- Model, az sayıda eğitim örneğiyle overfitting eğilimi gösteriyor.

- Daha fazla veri kullanıldıkça, model genelleme performansı iyileşiyor ve çapraz doğrulama skoru daha stabil hale geliyor.

- Eğitim ve çapraz doğrulama skorları arasındaki farkın azaldığı yer, modelin performansını daha iyi optimize ettiği bölgedir.

- Eğitim verisinin artırılması, modelin genelleme kapasitesini daha da iyileştirebilir.



Şekil 2.14 *Manifold Learning grafiği*

Bu grafik, **t-SNE (t-Distributed Stochastic Neighbor Embedding)** yöntemini kullanarak yüksek boyutlu bir veri kümesinin iki boyutlu bir düzleme indirgenmiş temsilini göstermektedir. Bu yöntem, verilerdeki ilişkileri ve yapıları daha görsel bir şekilde analiz etmeyi sağlar.

Grafik Unsurları

1. **Puanların Renk Kodlaması:**

- Sağ taraftaki renk skalası, hedef değişkenin (örneğin, fiyat veya başka bir sürekli değer) dağılımını ifade eder.

- Kırmızıdan maviye geçiş, hedef değişkenin düşükten yükseğe doğru artan değerlerini göstermektedir. Örneğin:

  - Kırmızı tonlar: Daha düşük hedef değerler.

  - Mavi tonlar: Daha yüksek hedef değerler.

2. **x ve y Eksenleri:**

- x ve y eksenleri t-SNE tarafından oluşturulan boyutları temsil eder. Bu eksenlerin fiziksel veya doğrudan bir anlamı yoktur; ancak veri kümelerindeki benzerlikleri göstermek için göreceli bir ölçü sağlar.

3. **Gruplaşmalar (Clusters):**

- Veri noktalarının yoğun olduğu bölgeler, veriler arasında yüksek benzerlikler veya ilişkiler olduğunu gösterebilir.

- Ayrık noktalar veya kümeler, olası anormallikleri ya da farklı veri alt gruplarını ifade edebilir.

Grafik Yorumu

1. **Kümeleşmeler:**

- Grafikte birden fazla grup veya küme görünüyor. Bu, veri setindeki gözlemlerin belirli özellikler bakımından benzerlik gösterdiğini ve doğal olarak bir araya toplandığını işaret eder.

- Küme yapıları, modelinizin belirli alt gruplar üzerinde daha iyi veya daha kötü performans göstermesine yol açabilir.

2. **Hedef Değişkenin Dağılımı:**

- Renk dağılımına göre, düşük değerli hedef değişkenlere sahip gözlemler (kırmızı tonlar) ile yüksek değerlere sahip gözlemler (mavi tonlar) arasında belirgin ayrımlar var.

- Bu, hedef değişkenin veri özelliklerine göre belli başlı bölgelerde yoğunlaştığını ve ayrıştığını gösteriyor.

3. Küme İçindeki Heterojenlik:

- Bazı kümelerde renk tonlarının karışık olduğu görülüyor. Bu, hedef değişkenin bu kümelerde daha homojen olmadığını, yani aynı grupta düşük ve yüksek değerlerin bir arada bulunduğunu işaret edebilir.

4. Outlier (Aykırı Değerler):

- Grafik, t-SNE indirgemesindeki veri dağılımına göre, kümelerden uzak kalan veya diğer noktalardan izole olmuş aykırı değerlere dair ipucu verebilir.

Sonuç ve Öneriler

1. Gruplar Arası Ayrımlar:

- Kümeleşme yapısı, benzer özelliklere sahip veri noktalarını gruplandırmada etkili olduğunu gösteriyor. Bu yapıları daha detaylı analiz etmek için, her bir kümenin özelliklerini inceleyebilirsiniz.

2. Aykırı Değer Analizi:

- Küme yapısından uzak olan noktalar, potansiyel aykırı değerlerdir. Bu değerleri belirlemek ve analiz etmek, model performansını artırabilir.

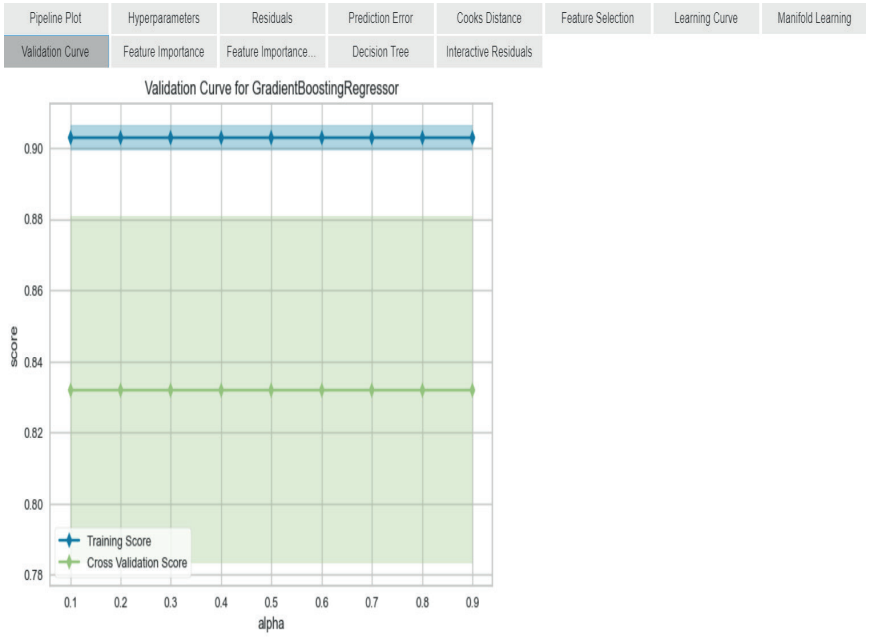
3. Renk Dağılımına Göre İnceleme:

- Hedef değişkenin düşük ve yüksek olduğu bölgeleri daha iyi anlamak için, bu alanlarda hangi özelliklerin belirleyici olduğuna bakabilirsiniz.

4. Küme Bazlı Performans:

- Eğer regresyon modeli, bu tür bir kümelenmeyi tam olarak öğrenemiyorsa, model performansını iyileştirmek için bu grupları dikkate alan özellik mühendisliği veya daha karmaşık modeller düşünülebilir.

Bu grafik, veri kümesindeki yapısal bilgileri anlamak için güçlü bir araçtır ve analiz süreçlerini daha iyi yönlendirebilir.



Şekil 2.15 *Doğrulama Eğrisi (Validation Curve)*

GradientBoostingRegressor modeli için bir **doğrulama eğrisi**ni (Validation Curve) (Şekil 2.15) göstermektedir. Grafik, modelin bir hiperparametresi olan *alpha* değerinin (*quantile.regression*'da kullanılan bir hiperparametre) hem eğitim (training) hem de çapraz doğrulama (cross-validation) üzerindeki etkisini analiz etmek için kullanılmıştır.

Grafiğin (Şekil 2.15) özellikleri:

1. **Eğitim Skoru (Training Score):**

- Mavi çizgi, modelin eğitim verisi üzerindeki performansını göstermektedir.

- Skor, alpha değeri değişse bile neredeyse sabit kalarak 0.90 civarında kalmıştır.

2. **Çapraz Doğrulama Skoru (Cross-Validation Score):**

- Yeşil çizgi, modelin çapraz doğrulama seti üzerindeki performansını göstermektedir.

- Skor, alpha değerine göre bir değişiklik göstermemiş ve yaklaşık 0.83-0.84 civarında sabit kalmıştır.

3. x-Ekseni (Alpha):

- Grafik, alpha hiperparametresinin 0.1 ile 0.9 arasında değiştiği durumları inceler.

4. Y-Ekseni (Score):

- Skor, modelin başarımını ifade eder. Genelde R^2 veya başka bir regresyon metriği (örneğin, MAE ya da MSE'nin ters çevrilmiş hali) kullanılır.

Grafiği Yorumlarsak:

1. Eğitim ve Doğrulama Skorları Arasındaki Fark:

- Eğitim skoru ile doğrulama skoru arasında belirgin bir fark vardır. Eğitim skoru daha yüksektir (yaklaşık 0.90), bu da modelin eğitim verisine aşırı uyum (overfitting) gösterebileceğini düşündürür.

- Çapraz doğrulama skoru daha düşük (0.83 civarında) ve sabittir, bu da modelin genelleme yeteneğinin sınırlı olabileceğini gösterir.

2. Hyperparametre Alpha'nın Etkisi:

- Alpha değerinin değiştirilmesi (0.1'den 0.9'a kadar) hem eğitim hem de doğrulama skorlarını neredeyse hiç etkilememiştir. Bu, alpha hiperparametresinin model performansı üzerinde minimal bir etkisi olduğunu veya verisetinin özellikleri nedeniyle bu hiperparametrenin kritik olmadığını gösterir.

3. Stabil Performans:

- Eğitim ve doğrulama skorlarının alpha boyunca sabit kalması, modelin bu hiperparametreye duyarsız olduğunu ve stabil çalıştığını ifade eder.

Sonuç ve Öneriler

1. Hyperparametre Optimizasyonu:

- Alpha hiperparametresi, mevcut veri setinde veya modelde belirgin bir iyileşme sağlamıyor gibi görünüyor. Bu nedenle, hiperparametre optimizasyonunda diğer parametrelere (örneğin, learning_rate, n_estimators, max_depth) odaklanabilirsiniz.

2. Overfitting Sorunu:

- Eğitim ve doğrulama skorları arasındaki fark, modelin eğitim verisine aşırı uyum sağlamış olabileceğini gösteriyor. Overfitting'i azaltmak

için aşağıdaki stratejiler kullanılabilir:

- Daha fazla veri toplamak.
- Modelin karmaşıklığını azaltmak (örneğin, max_depth veya n_estimators'ı düşürmek).
- Regularization yöntemlerini (örneğin, min_samples_split veya min_samples_leaf) kullanmak.

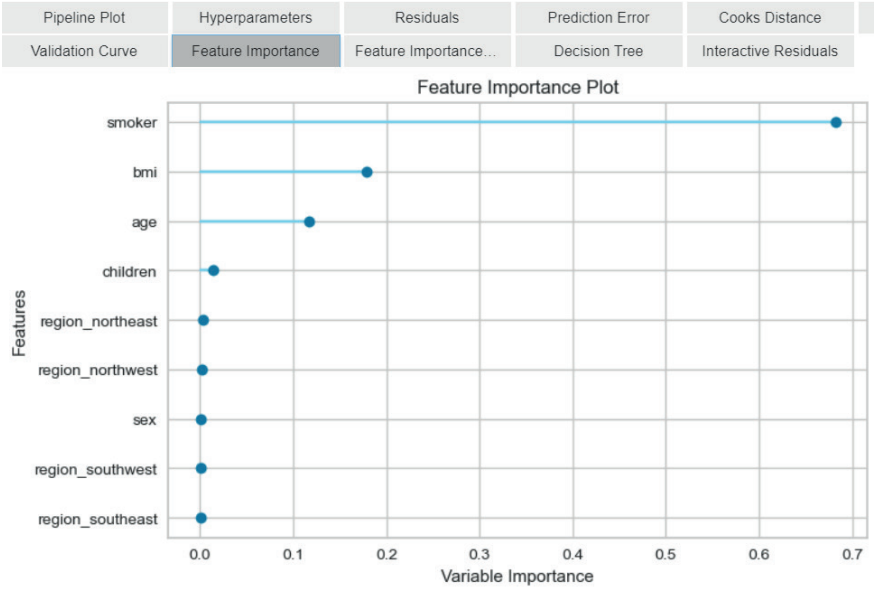
3. Alpha'nın Sabitliği:

- Alpha'nın model üzerindeki etkisi minimal olduğu için, bu hiperparametreyi varsayılan bir değerde bırakabilir veya diğer kritik hiperparametrelere öncelik verebilirsiniz.

4. Genelleme Performansı:

- Çapraz doğrulama skorunun sabitliği, modelin belirli bir genelleme performansına ulaştığını ancak daha iyileştirilebilir alanlar olabileceğini işaret eder. Özellikle, genelleme yeteneğini artırmak için farklı model yapılarını veya başka algoritmaları denemek faydalı olabilir.

Bu doğrulama eğrisi, modelin alpha hiperparametresine duyarlılığını değerlendirmek için önemli bir araçtır ve genel performans hakkında değerli bilgiler sunmaktadır.



Şekil 2.16 *Özellik Etki (Feature Importance) Grafiği*

Özellik Etki (Feature Importance) grafiği (Şekil 2.16), bir modelin özniteliklerinin hedef değişken üzerindeki etkisini gösteren bir grafiktir. Bu açıdan bakıldığında bazı özniteliklerin çok etkili bazılarının ise çok az etkili olduğu görülebilmektedir.

Grafikte yer alan bilgileri incelersek:

1. **Dikey Eksen (Features):** Modelde kullanılan öznitelikler sıralanmıştır. Örneğin: smoker, bmi, age, children, region gibi özellikler bulunmaktadır.

2. **Yatay Eksen (Variable Importance):** Bu, her bir özniteliğin modelin tahminlerine katkısının büyüklüğünü göstermektedir. Daha büyük bir değer, modelin tahminleri üzerinde daha büyük bir etki anlamına gelir.

Yorum:

- **En Önemli Öznitelik:** *smoker* (sigara içen birey) özelliği, açık ara en yüksek önem derecesine sahiptir (~0.7). Bu, modelin tahminlerini en çok etkileyen özelliktir.

- **Diğer Önemli Öznitelikler:** *bmi* (beden kitle indeksi) ve *age* (yaş), model üzerinde orta derecede etkili özelliklerdir.

- **Daha Az Etkili Öznitelikler:** *children* (çocuk sayısı) ve farklı *region* (bölge) değişkenleri model üzerinde çok az bir etkiye sahiptir. *sex* (cinsiyet) de model için önemsiz sayılabilecek düzeydedir.

Sonuç olarak: Bu grafik (Şekil 2.16), modelin özellikle *smoker*, *bmi* ve *age* özelliklerine odaklanarak daha iyi hale getirilebileceğini göstermektedir. Diğer değişkenlerin önem düzeyi düşük olduğu için, bu değişkenler modele fazla katkı sağlamayabilir. Özellik seçimi (*feature selection*) aşamasında, düşük öneme sahip bu özellikler çıkarılabilir.

Grafikler ve bu grafiklere dayalı yorumlar, kullanılan problem ve algoritmaya göre farklılık gösterebilir. Bu nedenle her bir farklılık dikkatlice analiz edilmeli ve problemin başarısını artırmaya yönelik uygun adımlar atılmalıdır. Örneğin, *evaluate_model* komutuyla oluşturulan bazı grafikler *GradientBoostingRegressor* algoritmasıyla çalışmamaktadır. Bu durum, algoritmaların yapısına ve matematiksel modeline bağlı olup tamamen normaldir. PyCaret, algoritmanın bu tür grafiklerin çizilip çizilemeyeceğini otomatik olarak kontrol eder. Bu özellik, PyCaret'in kullanıcıya sunduğu önemli avantajlardan biridir.

2.5 Tahminleme (Prediction)

PyCaret'ta kullanılan *predict_model* fonksiyonu (Liste 2.9), daha önce eğitilmiş bir modeli kullanarak yeni veriler üzerinde tahminleme yapmadan önce verilen veri setine tahmin sütunu (*prediction_label*) ekler. Model tarafından tahmin edilen değerler bu (*prediction_label*) sütun altında listelenir. Eğer veri seti (data) belirtilmezse, yani *data=None* olarak bırakılırsa, *predict_model* fonksiyonu test setini kullanır. Test seti, PyCaret'in setup fonksiyonu kullanılarak model kurulumu aşamasında otomatik olarak oluşturulan veri kümesidir. Bu durumda modelin performansı test verisi üzerinde değerlendirilir ve çıktı olarak tahminler döndürülür. Bu fonksiyon (Liste 2.9), modelin farklı veri setleri üzerinde nasıl performans gösterdiğini analiz etmek için oldukça kullanışlıdır. Ayrıca, yeni veriler üzerinde tahminleme yaparak modelin uygulanabilirliğini ve güvenilirliğini test etme olanağı sağlar.

```
# Liste 2.9 Test Seti üzerinde tahminleme
holdout_pred = predict_model(best)
```

Liste 2.9 *Test Seti üzerinde en başarılı model (GradientBoostingRegressor) ile tahminleme*

Liste 2.9'deki kod ile dataframe (holdout_pred isimli nesne) formatında bir tahminleme dizisi üretir. Bu testin başarı değeri Şekil 2.17'de gösterildiği gibidir.

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Gradient Boosting Regressor	2392.5661	17148355.3169	4141.0573	0.8800	0.3928	0.2875

Şekil 2.17 *Modelin test veri seti üzerindeki tahmin performansı*

Tahmin edilen test setinin ilk 5 satırı da Liste 2.10'daki kod ile çağrılmış ve Şekil 2.18'deki gibi elde edilmiştir.

```
# Liste 2.10 Test setinin ilk 5 satırı tahmin değerleri eklenerek ekrana yazdırılır
holdout_pred.head()
```

Liste 2.10 *Test Setinin ilk 5 satırını ekrana yazdırma*

	age	sex	bmi	children	smoker	region	charges	prediction_label
650	49	female	42.680000	2	no	southeast	9800.888672	10681.513104
319	32	male	37.334999	1	no	northeast	4667.607422	8043.453463
314	27	female	31.400000	0	yes	southwest	34838.871094	36153.097686
150	35	male	24.129999	1	no	northwest	5125.215820	7435.516853
336	60	male	25.740000	0	no	southeast	12142.578125	14676.544334

Şekil 2.18 *Modelinin tahmin (prediction_label altındaki) değerleri*

Aynı fonksiyon, görülmemiş veri kümeleri üzerindeki etiketleri tahmin etmek için de kullanılabilir. Özgün veri setinin bir kopyasını oluşturup *charges* değişkenini çıkararak, yeni bir etiketsiz veri çerçevesi elde edebiliriz. Bu yeni veri çerçevesini skorlama işlemi için kullanabiliriz.

Bu işlemde, orijinal veri setini korumak amacıyla önce *data.copy()* fonksiyonu kullanılarak bir kopyası oluşturulur ve bu kopya *new_data* adı altında saklanır. Daha sonra *drop()* fonksiyonu ile *charges* isimli sütun veri setinden kaldırılır. Bu işlemde *axis=1* parametresi, sütun bazında bir silme işlemi gerçekleştirileceğini belirtirken, *inplace=True* kullanımı değişikliklerin doğrudan *new_data* üzerinde yapılmasını sağlar. Son olarak, *new_data.head()* komutu ile veri setinin güncellenmiş halinin ilk 5 satırı görüntülenir. Bu işlemler sonucunda *charges* değişkeni çıkarılmış ve yeni veri seti, analiz veya modelleme için hazırlanmış olur (Liste 2.11).

```
# Liste 2.11 veriyi kopyala ve 'charges' sütununu sil
new_data = data.copy()
new_data.drop('charges', axis=1, inplace=True)
new_data.head()
```

Liste 2.11 *Eğitimde kullanılan veri setinin tahminleme için kullanımı*

	age	sex	bmi	children	smoker	region
0	19	female	27.900	0	yes	southwest
1	18	male	33.770	1	no	southeast
2	28	male	33.000	3	no	southeast
3	33	male	22.705	0	no	northwest
4	32	male	28.880	0	no	northwest

Şekil 2.19 *Tüm verisetinin tahminleme öncesi 'charges' sütununun çıkarılmış özeti*

```
# Liste 2.12 new_data üzerinde modeli tahmin etme
predictions = predict_model(best, data = new_data)
predictions.head()
```

Liste 2.12 *Eğitimde kullanılan veri setinin tahminleme için kullanımı*

	age	sex	bmi	children	smoker	region	prediction_label
0	19	female	27.900000	0	yes	southwest	18464.334448
1	18	male	33.770000	1	no	southeast	4020.345384
2	28	male	33.000000	3	no	southeast	6555.388388
3	33	male	22.705000	0	no	northwest	9627.045725
4	32	male	28.879999	0	no	northwest	3325.531292

Şekil 2.20 *Tüm veri setinin en başarılı model ile tahminlenmesi*

Eğitim veri setinin tamamını Liste 2.12'deki kod ile tahminlenerek modelin başarısı tüm veri üzerinde Şekil 2.20'daki gibi görülebilir.

2.6 Modelin Kaydedilmesi (Save Model)

PyCaret, *save_model* fonksiyonunu oluşturulan tüm model geliştirme ve veri işleme adımlarını içeren bir pipeline'ı disk üzerinde kaydetmek için kullanır (Liste 2.13).

```
# Liste 2.13 modelin kaydedilmesi
save_model(best, 'my_first_pipeline')
```

Liste 2.13 *Modelin Kaydedilmesi*

Bu işlem, ileride aynı modeli tekrar kullanmak veya başka bir ortama taşımak için büyük kolaylık sağlar. Kaydedilen model, sadece algoritmayı değil, veri ön işleme adımları, dönüşümler ve hiperparametre ayarları gibi tüm pipeline'ı kapsar. Böylece, modelin yeniden eğitilmesine gerek kalmadan doğrudan tahminler yapmak mümkün olur. Kaydedilen model dosyası, daha sonra load_model fonksiyonu ile tekrar yüklenerek kullanılabilir. Bu özellik, özellikle büyük projelerde veya sürekli model kullanımı gerektiren durumlarda oldukça faydalıdır. Kayıt işlemi başarıyla gerçekleştiğinde Şekil 2.21'deki gibi bir çıktı alınır.

Modelin tekrar çalışma ortamına yüklenmesi, Liste 2.14'deki kod ile gerçekleştirildiğinde (model eğitim aşamalarını gösteren işlem hattı şeması) Şekil 2.22 'daki gibi bir çıktı ile gerçekleşir.

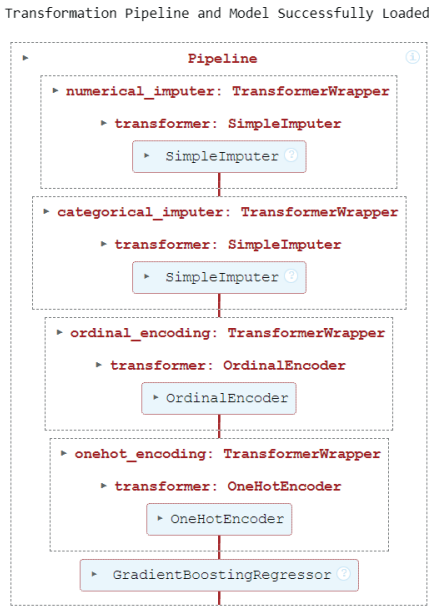
```
# Liste 2.14 modelin yüklenmesi
loaded_best_pipeline = load_model('my_first_pipeline')
loaded_best_pipeline
```

Liste 2.14 *Modelin Yüklenmesi*

```
Transformation Pipeline and Model Successfully Saved

(Pipeline(memory=Memory(location=None),
          steps=[('numerical_imputer',
                  TransformerWrapper(include=['age', 'bmi', 'children'],
                                     transformer=SimpleImputer())),
                 ('categorical_imputer',
                  TransformerWrapper(include=['sex', 'smoker', 'region'],
                                     transformer=SimpleImputer(strategy='most_frequent'))),
                 ('ordinal_encoding',
                  TransformerWrapper(include=['sex', 'smoker'],
                                     transfor...
                                                                    'data_type': dtype('O'),
                                                                    'mapping': female    0
male      1
NaN      -1
dtype: int64},
                                                                   {'col': 'smoker',
                                                                    'data_type': dtype('O'),
                                                                    'mapping': no     0
yes    1
NaN   -1
dtype: int64}])),
                 ('onehot_encoding',
                  TransformerWrapper(include=['region'],
                                     transformer=OneHotEncoder(cols=['region'],
                                                               handle_missing='return_nan',
                                                               use_cat_names=True))),
                 ('trained_model', GradientBoostingRegressor(random_state=123))]),
 'my_first_pipeline.pkl')
```

Şekil 2.21 *Modelin kaydedilme çıktısı*



Şekil 2.22 *Modelin kaydedilme çıktısı*

Test verisetinde olduğu gibi yüklenen model ile yeni veriler tahminlenebilir. Daha ayrıntılı eğitim ve model karşılaştırma süreçleri pycaret github sayfasından incelenebilir (*https://github.com/pycaret/pycaret/blob/master/tutorials/README.md*).

2.7 Modeli Ayarlama (Tune Model)

Regresyon algoritmaları için model performansını en üst düzeye çıkarmak, doğru hiperparametrelerin belirlenmesine bağlıdır. PyCaret'in *tune_model* fonksiyonu, bu süreci otomatikleştirerek kullanıcıya büyük kolaylık sağlar. Bu fonksiyon, regresyon modelinin hiperparametrelerini ayarlayarak en iyi kombinasyonu bulur ve performansını optimize eder.

Fonksiyonun çıktısı, çapraz doğrulama yöntemiyle katmanlara (fold) göre elde edilen skorların yer aldığı bir değerlendirme tablosudur. Modelin başarısını değerlendirmek için kullanılacak metrik, optimize parametresi üzerinden belirlenir. Örneğin, R^2, MAE, MSE veya RMSE gibi çeşitli regresyon metrikleri kullanılabilir. Çapraz doğrulama sırasında değerlendirilen metrikler, *get_metrics* fonksiyonu ile görüntülenebilir. Ancak bazen, projeye özel metriklere ihtiyaç duyulabilir. Bu durumda, *add_metric* fonksiyonu ile özel metrikler sisteme eklenebilir ya da gereksiz görülen metrikler *remove_metric* fonksiyonu ile kaldırılabilir. Bu sayede, model ayarlama süreci tamamen özelleştirilebilir ve projenin ihtiyaçlarına göre şekillendirilebilir. *tune_model*, verimlilik ve esneklik sağlarken, regresyon modelinin en iyi performansı elde etmesi için ince ayar yaparak veri bilimi yolculuğunuzu daha da ileriye taşır (Liste 2.15).

```
# Liste 2.15 varsayılan ayarlarda karar ağacı (Decision Tree) modeli oluşturma
dt = create_model('dt')
```

Liste 2.15 *Modelin Kaydedilmesi*

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	3244.6173	45002914.9978	6708.4212	0.7312	0.5884	0.4883
1	3106.2611	45435728.7536	6740.6030	0.7358	0.5389	0.3271
2	3646.2662	54445682.2627	7378.7318	0.3107	0.6475	0.4752
3	3267.9250	45463401.7749	6742.6554	0.6204	0.5751	0.4339
4	4344.7470	65261429.3013	8078.4546	0.5146	0.7261	0.6008
5	3497.9281	42984919.0254	6556.2885	0.7235	0.4614	0.3208
6	3596.2637	53600704.7298	7321.2502	0.6353	0.5284	0.4126
7	2804.7493	37461859.8541	6120.6094	0.7940	0.4737	0.1787
8	3080.1801	42102090.8846	6488.6124	0.7281	0.5168	0.4537
9	3402.4641	49243302.2625	7017.3572	0.6828	0.5725	0.3613
Mean	3399.1402	48100203.3847	6915.2984	0.6476	0.5629	0.4052
Std	398.2185	7518631.1992	528.0642	0.1348	0.0754	0.1094

Şekil 2.23 *dt modelinin performans değerlerinin fold adımlarına göre listelenmesi*

Liste 1.18'deki paramtre ayarlaması yapılarak **Karar Ağacı (Decision Tree)** modelinin hiperparametrelerini optimize edebiliriz. Burada dt değişkeni, önceden tanımlanmış bir Karar Ağacı modelini temsil eder ve **tune_model (dt)** fonksiyonu, modelin performansını artırmak için en iyi hiperparametre kombinasyonunu arar (Liste 2.16). Bu işlem sırasında çapraz doğrulama uygulanır ve değerlendirme, **optimize** parametresinde belirtilen metrik (Örneğin, R^2, MAE, MSE veya RMSE) üzerinden yapılır. Ayarlanmış hiperparametrelerle elde edilen en iyi model, **tuned_dt** değişkeninde saklanır. Sonuç olarak, **tune_model** fonksiyonu, performansı optimize edilmiş ve kullanıma hazır bir model üretir.

Şekil 2.23'teki ortama R^2 değeri incelendiğinde 0.6476 olan değer Şekil 2.24'te 0.8265'e yükselmiştir. Alternatif parametre ayarlama gerçekleştirilerek bu doğruluk değeri arttırılmaya çalışılabilir.

```
# Liste 2.16 modelin otomatik ayarlanması
tuned_dt = tune_model(dt)
```

Liste 2.16 *Modelin parametrelerinin ayarlanması (tuning)*

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	1745.0008	18073621.2534	4251.3082	0.8920	0.3408	0.1390
1	2380.2671	33969297.4978	5828.3186	0.8025	0.4803	0.1491
2	2005.5481	23477540.5275	4845.3628	0.7027	0.4742	0.1604
3	1986.9419	22156779.8636	4707.0989	0.8150	0.3731	0.1550
4	2255.0797	28517151.4384	5340.1453	0.7879	0.4832	0.1465
5	1961.7810	20794913.6607	4560.1440	0.8662	0.3653	0.1287
6	1649.9559	20053618.6090	4478.1267	0.8635	0.3315	0.1164
7	2049.2066	26281892.4673	5126.5868	0.8555	0.4653	0.1298
8	1991.8599	23667668.4391	4864.9428	0.8471	0.3865	0.1452
9	2159.0994	26013111.3580	5100.3050	0.8324	0.4242	0.1459
Mean	2018.4740	24300559.5115	4910.2339	0.8265	0.4124	0.1416
Std	205.7361	4392006.1282	436.0762	0.0511	0.0570	0.0126

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

Şekil 2.24 *dt modelinin parametre ayarlaması sonrası performansın fold işlemine göre değişimi*

2.8. Birleştirilmiş (Ensemble) Model

Eğitilen farklı algoritmalar yapılan parametre düzenlemesine ve çeşitli konfigürasyonlarla denemeler yapılmasına rağmen başarısız olursa birden fazla model ile birleştirilebilir ve ensemble (birleştirilmiş) bir model ile yeni karma bir model türetilebilir. Ensemble fonksiyonu, verilen tahminleyiciyi (estimator) topluluk yöntemleriyle birleştirilir (ensemble). Fonksiyonun çıktısı, çapraz doğrulama ile katmanlara göre hesaplanan skorların yer aldığı bir değerlendirme tablosudur. Çapraz doğrulama sırasında değerlendirilen metrikler *get_metrics* fonksiyonu kullanılarak görüntülenebilir. Ayrıca, özel metrikler *add_metric* fonksiyonu ile eklenebilir veya gereksiz metrikler *remove_metric* fonksiyonu ile kaldırılabilir. Bu sayede model değerlendirme süreci ihtiyaçlara göre özelleştirilebilir ve daha başarılı sistemler üretilebilir.

```
# Liste 2.17 Bagging metodu ile model birleştirme
ensemble_model(dt, method = 'Bagging')
```

Liste 2.17 *Modelin parametrelerinin ayarlanması (tuning)*

Bagging (Bootstrap Aggregating) kullanarak topluluk (ensemble) modelinin oluşturulması (Liste 1.19) karar ağaçları algoritma modellerinin bagging yöntemi ile birleştirilmesini ifade eder. Bagging, aynı tahminleyiciyi (estimator) farklı örnekleme veri setleri üzerinde eğiterek birden fazla model oluşturur ve bu modellerin tahminlerini birleştirir. Bu işlem, modelin varyansını azaltarak aşırı öğrenmeyi (overfitting) önlemeye yardımcı olur ve modelin genel performansını iyileştirir.

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	2591.8970	23266281.4574	4823.5134	0.8610	0.4637	0.2976
1	2863.6017	30202461.8149	5495.6766	0.8244	0.4882	0.3053
2	2736.5380	24936511.9328	4993.6472	0.6843	0.5148	0.3293
3	2945.2626	27479881.3264	5242.1256	0.7705	0.5164	0.4187
4	3075.1990	30901342.4317	5558.8976	0.7702	0.5670	0.3906
5	2866.8198	25117097.4494	5011.6961	0.8384	0.3711	0.2607
6	2568.9545	22780849.6859	4772.9288	0.8450	0.3730	0.2717
7	2639.4091	26044331.1073	5103.3647	0.8568	0.4710	0.2506
8	2364.6343	19889092.4425	4459.7189	0.8715	0.4108	0.3040
9	2820.2231	31860942.5716	5644.5498	0.7948	0.4605	0.2805
Mean	2747.2539	26247879.2220	5110.6119	0.8117	0.4636	0.3109
Std	198.4899	3670176.0781	359.8965	0.0547	0.0603	0.0521

Şekil 2.25 *dt modelinin bagging yöntemi ile birleştirilen model performansı*

Ensemble işlemi (Liste 2.17), tune işlemi (Liste 2.16) ile karşılaştırıldığında sistem performansında bir düşüşe neden olmuştur. Şekil 2.24'te R^2 değeri 0.8265 iken, Şekil 2.25'te bu değer 0.8117'ye gerilemiştir. Ensemble yöntemi, varsayılan Decision Tree (DT) modelinin performansına kıyasla (Şekil 2.23, $R^2 = 0.6476$) daha başarılı olsa da, tune yöntemiyle elde edilen performansın gerisinde kalmıştır.

2.9. Karışım Modelleri (Blend Models)

Birden fazla farklı tür algoritma modeli birleştirerek performansı arttırmak istendiğinde Liste 2.19'da verilen *blend_model* yaklaşımı uygulanabilir. Bu fonksiyon (*blend_models*), belirtilen *estimator_list* parametresindeki modelleri birleştirerek bir VotingRegressor oluşturur. Bu yöntem, birden fazla regresyon modelinin tahminlerini bir araya getirerek daha güçlü ve dengeli bir performans elde etmeyi amaçlar. Bu fonksiyonun çıktısı, çapraz doğrulama yöntemiyle katmanlara (fold) göre elde edilen skorların yer aldığı bir değerlendirme tablosudur. Bu tablo, her bir katmanda elde edilen metriklerin özetini sunar ve modellerin birleştirilmesinin performansa etkisini analiz etmenize olanak tanır. Çapraz doğrulama sırasında değerlendirilen metrikler, *get_metrics* fonksiyonu ile görüntülenebilir. Ayrıca, projeye özel metrik ihtiyaçları durumunda *add_metric* fonksiyonu ile özel metrikler eklenebilir ya da gereksiz metrikler *remove_metric* fonksiyonu kullanılarak kaldırılabilir. Bu özellik, model birleştirme sürecini tamamen özelleştirilebilir hale getirir. Modellerin karıştırılması (*blend_models*), birden fazla regresyon modelini birleştirerek genel performansı artırmayı ve daha iyi genelleştirme yapmayı hedefleyen esnek ve kullanışlı bir araçtır.

```
# Liste 2.18 MAE değerine göre en iyi 3 modelin çağrılması
best_mae_models_top3
```

Liste 2.18 *MAE değerine göre en iyi 3 modelin çağrılması*

Liste 2.18'in çıktısı olarak, Şekil 2.26'da MAE değerine göre en başarılı 3 model ve bu modellere ait parametreler gösterilmektedir.

```
[GradientBoostingRegressor(random_state=123),
 RandomForestRegressor(n_jobs=-1, random_state=123),
 ExtraTreesRegressor(n_jobs=-1, random_state=123)]
```

Şekil 2.26 *MAE değerine göre en başarılı 3 model; GradientBoostingRegressor, RandomForestRegressor, ExtraTreesRegressor*

Bu 3 modelin karıştırılarak daha başarılı bir karma model oluşturulması (Liste 2.19).

```
# MAE değerine göre en iyi 3 modelin karıştırılması
blend_models(best_mae_models_top3)
```

Liste 2.19 *Farklı algoritmaların karıştırılması (blend modals)*

En başarılı model olarak eğitebildiğimiz bu modeli ayrıntılı olarak bir tahminleme problemini performans kriterleri açısından incelersek;

Bu karıştırma yöntemi, regresyon probleminde *Gradient Boosting Regressor, Random Forest Regressor* ve *Extra Trees Regressor* algoritmalarının birleştirilmesi sonucu elde edilen *Voting Regressor* modelinin çapraz doğrulama (cross-validation) performansını göstermektedir. Performans metrikleri açısından değerleri şu şekilde yorumlayabiliriz:

- **MAE (Mean Absolute Error):** Ortalama mutlak hata, 2715.94 ile modelin tahminlerinin gerçek değerlere ne kadar yakın olduğunu gösterir. Daha düşük bir değer, daha iyi bir performansa işaret eder.

- **MSE (Mean Squared Error):** Ortalama karesel hata, 23991565.26 ile tahmin hatalarının karelerinin ortalamasını ifade eder. Hataların büyüklüğü arttıkça bu değer büyür.

- **RMSE (Root Mean Squared Error):** Kök ortalama kare hata, 4879.83 ile hatanın bir ölçüsü olarak kullanılır. Tahminlerin doğruluğunu değerlendirir ve ölçeklenebilirliği iyidir.

- **R2 (R-squared):** 0.8289, modelin bağımlı değişkendeki (hedef) değişkenliği ne kadar iyi açıkladığını gösterir. Bu değer 1'e ne kadar yakınsa, model o kadar başarılıdır.

- **RMSLE (Root Mean Squared Logarithmic Error):** 0.4545, logaritmik ölçekte tahmin edilen ve gerçek değerler arasındaki farkı gösterir. Hataların ölçeklenmesine karşı daha az duyarlıdır.

- **MAPE (Mean Absolute Percentage Error):** Ortalama yüzdesel hata, %32.18 ile tahmin edilen değerlerin, gerçek değerlere kıyasla yüzde olarak hatasını gösterir. Daha düşük bir oran, daha iyi bir performansa işaret eder.

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	2720.8934	22050841.6103	4695.8324	0.8683	0.4637	0.3425
1	2865.4018	30821460.9279	5551.7079	0.8208	0.4535	0.2800
2	2581.6067	22252661.3019	4717.2727	0.7183	0.5463	0.3794
3	2810.2333	21734211.3564	4661.9965	0.8185	0.4864	0.3873
4	3070.0103	30740150.2464	5544.3801	0.7714	0.5469	0.3679
5	2854.7097	22065332.3136	4697.3750	0.8581	0.3771	0.2778
6	2450.8238	20209907.9911	4495.5431	0.8625	0.3740	0.2901
7	2595.2491	23563676.7364	4854.2432	0.8704	0.3997	0.2334
8	2262.9477	18038706.9975	4247.1999	0.8835	0.3846	0.2937
9	2947.5441	28438703.1674	5332.7951	0.8168	0.5125	0.3660
Mean	2715.9420	23991565.2649	4879.8346	0.8289	0.4545	0.3218
Std	231.8660	4212892.9107	422.8235	0.0491	0.0647	0.0504

Şekil 2.26 *GradientBoostingRegressor, RandomForestRegressor, ExtraTreesRegressor modellerinin karıştırılması ile elde edilmiş model performansı*

Genel olarak yorumlarsak:

- R^2 metriği 0.8289 ile modelin hedef değişkenin yaklaşık %83'ünü açıklayabildiğini göstermektedir ve bu oldukça iyi bir sonuçtur.
- MAE ve RMSE değerleri, modelin tahmin hatalarının kabul edilebilir bir seviyede olduğunu ifade etmektedir.
- MAPE'nin %32.18 olması, modelin tahminlerinin gerçek değerlere göre yaklaşık %32'lik bir sapma gösterdiğini belirtir.

Bu sonuçlar, üç algoritmanın birleştirilmesiyle (blending) elde edilen Voting Regressor'un güçlü bir model olduğunu, ancak hala bazı iyileştirme alanlarının bulunduğunu işaret ediyor. Özellikle hataların düşürülmesi ve genel doğruluğun artırılması için daha fazla hiperparametre optimizasyonu veya farklı veri ön işleme teknikleri değerlendirilebilir.

Bazı önemli **blend_models** parametreleri ve açıklamaları şunlardır:

- **choose_better:** Bu parametre, modelin performansını daha iyi hale getirmek için varsayılan modeli veya önerilen modeli otomatik olarak seçer. (Varsayılan: False)
- **weights:** Birleşimde kullanılan algoritmalara belirli bir ağırlık atamak için kullanılır. Örneğin, bazı modellerin daha fazla etkisi olması gerekiyorsa bu parametre ayarlanabilir. (Örn: [0.5, 0.3, 0.2])
- **optimize:** Modeli optimize etmek için kullanılacak değerlendirme metriğini belirtir. Örneğin, MAE, R2, veya RMSE gibi metrikler kullanılabilir. (Varsayılan: R2)

- **fit_kwargs:** Modellerin eğitim sürecinde eklemek istediğiniz özel argümanları sağlar. Örneğin, eğitim sırasında kullanılan hiperparametre ayarlarını burada geçirebilirsiniz.

- **return_train_score:** Eğitim setindeki değerlendirme metriklerini de geri döndürmek için kullanılır. (Varsayılan: False)

Not: Fonksiyonun diğer ayrıntılı bilgileri için dokümantasyon (docstring) (Liste 2.20) incelenebilir.

```
# Liste 2.20 Daha fazla bilgi almak için dökümantasyon(docstring) çağırma
help(blend_models)
```

Liste 2.20 *Bigli almak için dökümantasyon(docstring) çağırma*

2.10. Meta Modeller (Stack Models)

stack_models fonksiyonu, verilen tahminciler (estimators) üzerinde bir meta-model eğitmek için kullanılır. Meta-model, diğer modellerin tahmin sonuçlarını giriş olarak alır ve bu sonuçları birleştirerek genel performansı artırmayı hedefler. Bu yöntem, genellikle model çeşitliliğinden faydalanarak daha iyi tahminler elde etmek için kullanılır.

Fonksiyonun Detayları:

Meta-modelin üzerine inşa edileceği modellerin listesini *estimator_list* parametresi içerir. Bu modeller daha önce eğitilmiş olmalıdır.

Fonksiyonun çıktısı, çapraz doğrulama (cross-validaiton) yöntemi ile fold bazında elde edilen skorların yer aldığı bir değerlendirme tablosudur. Bu tablo, her fold için elde edilen metrikleri içerir ve modelin performansı hakkında genel bir fikir verir.

Çapraz doğrulama sırasında kullanılan metrikler, *get_metrics* fonksiyonu ile görüntülenebilir. Projeye özel metrikler gerekiyorsa, *add_metric* fonksiyonu ile yeni metrikler eklenebilir. Gereksiz metrikler ise *remove_metric* fonksiyonu ile kaldırılabilir.

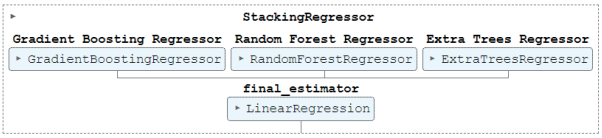
Bu meta-model, temel tahmincilerden daha iyi bir performans sunabilir. Model çeşitliliğinden faydalanarak hataları minimize eder ve daha dengeli bir tahmin performansı sağlar.

Bu yöntemin temel amacı, bir meta-model oluşturarak tahmin performansını artırmaktır. Sürecin çıktısı, fold bazında çapraz doğrulama sonuçlarıdır. Bunun yanı sıra, kullanıcıya metrik ekleme veya çıkarma esnekliği sunarak daha özelleştirilebilir bir yaklaşım sağlar. Bu yöntem, modelleme sürecinde genel performansı artırmak için güçlü bir stratejidir.

```
# Liste 2.21 Meta model üretmek için en iyi 3 MAE değerli model alınır
stack_models(best_mae_models_top3)
```

Liste 2.20 *Bigli almak için dökümantasyon(docstring) çağırma*

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	2609.6884	19941923.0191	4465.6380	0.8809	0.4388	0.3216
1	2980.4083	31017277.7854	5569.3157	0.8197	0.4751	0.2922
2	2546.1494	22498470.1082	4743.2552	0.7151	0.4924	0.2973
3	2847.5662	21076820.8684	4590.9499	0.8240	0.4727	0.3775
4	2921.5377	28163259.1669	5306.9067	0.7905	0.5215	0.3215
5	2677.5306	19787391.3140	4448.3021	0.8727	0.3998	0.2686
6	2369.6118	20267877.6270	4501.9860	0.8621	0.3340	0.2382
7	2693.0703	24841785.8067	4984.1535	0.8634	0.4340	0.2560
8	2229.6840	17762684.4081	4214.5800	0.8853	0.3765	0.2832
9	3001.6387	27582761.2165	5251.9293	0.8223	0.5155	0.3513
Mean	2687.6885	23294025.1320	4807.7016	0.8336	0.4460	0.3007
Std	245.2930	4159112.4380	424.2995	0.0494	0.0582	0.0408



Şekil 2.27 *GradientBoostingRegressor, RandomForestRegressor, ExtraTreesRegressor modellerinin Meta Modele dönüştürülmesi*

Genel olarak, stacking yöntemiyle oluşturulan model, R^2 değerinin yüksek olması nedeniyle hedef değişkeni oldukça iyi açıklamaktadır. MAE ve MAPE değerleri, modelin hata oranının kabul edilebilir seviyede olduğunu gösterirken, düşük RMSLE değeri modelin özellikle daha küçük tahminlerde başarılı olduğunu ortaya koymaktadır. Stacking yöntemi, birden fazla güçlü regresör modelin avantajlarını birleştirerek hata oranlarını azaltmayı ve daha iyi tahminler yapmayı başarmıştır. Modelin performansını daha da artırmak için, temel modellerin hiperparametre ayarları optimize edilebilir veya *XGBoost* ve *Ridge Regression* gibi farklı meta-modeller denenebilir. Bu yöntem, çeşitliliği kullanarak genel performansı başarılı bir şekilde artırmıştır.

Sonuç olarak, *Stacking Regressor* yöntemi, R^2 değeri ve hata oranları göz önüne alındığında, veri seti üzerinde oldukça etkili bir performans sergilemiştir. Bu yöntem hem hata oranını azaltmada hem de genel performansı artırmada başarılıdır.

Sonuç

Bu kitap boyunca, PyCaret kütüphanesini kullanarak sınıflandırma ve regresyon problemlerini her yönüyle ele aldık. PyCaret'in sağladığı kolaylıkları, esnek yapısını ve düşük kod gereksinimiyle sunduğu avantajları detaylı örneklerle inceledik. Amacımız, hem yeni başlayanlar için hızlı bir başlangıç sağlamak hem de uzman kullanıcılar için derinlemesine bir rehber sunmaktı.

Makine öğrenimi modelleri oluşturmanın karmaşıklığı ve zaman alıcı süreçlerini, PyCaret'in işlevselliği sayesinde basitleştirebileceğimizi gördük. Model oluşturma, hiperparametre ayarlama, değerlendirme metrikleri ve görselleştirme gibi birçok adımın kolaylıkla gerçekleştirilebildiğini uygulamalı olarak deneyimledik. Özellikle PyCaret'in fonksiyonel ve nesne yönelimli API'leri, farklı kullanıcı ihtiyaçlarına göre uyarlanabilir yapısıyla dikkat çekti.

Sınıflandırma ve regresyon problemleri özelinde, model karşılaştırma, model seçimi, özellik mühendisliği, hiperparametre optimizasyonu ve ensemble yöntemlerinin nasıl uygulanacağını öğrendik. Ensemble yöntemleri arasında, özellikle stacking ve blending gibi tekniklerin tahmin performansını artırmadaki etkisini vurguladık. Bu yöntemlerle, farklı algoritmaların güçlü yönlerini birleştirerek daha güçlü ve dengeli meta-modeller oluşturmanın mümkün olduğunu gördük.

Ayrıca, PyCaret'in kullanıcıya sunduğu esnek metrik yapılandırması ve otomatik değerlendirme araçlarının, model performansını detaylı bir şekilde analiz etmede ne kadar faydalı olduğunu inceledik. Ensemble yöntemlerinin başarısını, hem R^2 hem de diğer hata metrikleri üzerinden karşılaştırmalı olarak ele alarak, hangi durumlarda bu tekniklerin daha etkili olduğunu netleştirdik.

Sonuç olarak, PyCaret, makine öğrenimi süreçlerini demokratikleştiren, hızlandıran ve daha erişilebilir hale getiren güçlü bir araçtır. Ensemble yöntemleri ve diğer özellikleri sayesinde, veri bilimi projelerinde daha yüksek doğruluk oranlarına ulaşmak ve karmaşık problemleri çözmek mümkün hale gelir.

Bu kitapta paylaşılan bilgi ve tekniklerin, okuyuculara makine öğrenimi projelerinde sağlam bir temel oluşturacağına inanıyoruz. PyCaret'in sunduğu olanaklar, veri bilimi dünyasında daha yaratıcı ve etkili çözümler geliştirme yolunda önemli bir avantaj sağlar. Şimdi sıra, öğrendiklerinizi pratiğe dökerek kendi projelerinizi hayata geçirmekte! Unutmayın, her adım yeni bir keşif ve öğrenme fırsatıdır.

Kaynaklar

Topsakal, O. and Akıncı, T. (2023). Classification and regression using automatic machine learning (automl) – open source code for quick adaptation and comparison. Balkan Journal of Electrical and Computer Engineering, 11(3), 257-261. https://doi.org/10.17694/bajece.1312764

Whig, P., Gupta, K., Jiwani, N. et al. A novel method for diabetes classification and prediction with Pycaret. Microsyst Technol 29, 1479–1487 (2023). https://doi.org/10.1007/s00542-023-05473-2

Moisen, G. G., Freeman, E. A., Blackard, J. A., Frescino, T. S., Zimmermann, N. E., & Edwards, T. C. (2006). Predicting tree species presence and basal area in Utah: A comparison of stochastic gradient boosting, generalized additive models, and tree-based methods. Ecological Modelling, 199(2), 176–187. https://doi.org/10.1016/j.ecolmodel.2006.05.021

Gray, J. B. (2002). Introduction to linear regression analysis. Technometrics, 44(2), 191–192. https://doi.org/10.1198/tech.2002.s714

Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied Logistic Regression. In Wiley series in probability and statistics. https://doi.org/10.1002/9781118548387

Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics, 21(1). https://doi.org/10.1186/s12864-019-6413-7

Freeman, E. A., & Moisen, G. G. (2008). A comparison of the performance of threshold criteria for binary classification in terms of predicted prevalence and kappa. Ecological Modelling, 217(1–2), 48–58. https://doi.org/10.1016/j.ecolmodel.2008.05.015

Dong, J., Yao, Z., Zhu, M., Wang, N., Lu, B., Chen, A. F., Lu, A., Miao, H., Zeng, W., & Cao, D. (2017). ChemSAR: an online pipelining platform for molecular SAR modeling. Journal of Cheminformatics, 9(1). https://doi.org/10.1186/s13321-017-0215-1

Tran, M., Panchal, S., Chauhan, V., Brahmbhatt, N., Mevawalla, A., Fraser, R., & Fowler, M. (2021). Python-based scikit-learn machine learning models for thermal and electrical performance prediction of high-capacity lithium-ion battery. International Journal of Energy Research, 46(2), 786–794. https://doi.org/10.1002/er.7202

Mendez, K. M., Reinke, S. N., & Broadhurst, D. I. (2019). A comparative evaluation of the generalised predictive ability of eight machine learning algorithms across ten clinical metabolomics data sets for binary classification. Metabolomics, 15(12). https://doi.org/10.1007/s11306-019-1612-4

Jungo, A., Scheidegger, O., Reyes, M., & Balsiger, F. (2020). pymia: A Python package for data handling and evaluation in deep learning-based medical image analysis. Computer Methods and Programs in Biomedicine, 198, 105796. https://doi.org/10.1016/j.cmpb.2020.105796

Guerrero, M. C., Parada, J. S., & Espitia, H. E. (2021). EEG signal analysis using classification techniques: Logistic regression, artificial neural networks, support vector machines, and convolutional neural networks. Heliyon, 7(6), e07258. https://doi.org/10.1016/j.heliyon.2021.e07258

Conrad, F., Mälzer, M., Schwarzenberger, M., Wiemer, H., & Ihlenfeldt, S. (2022). Benchmarking AutoML for regression tasks on small tabular data in materials design. Scientific Reports, 12(1). https://doi.org/10.1038/s41598-022-23327-1