1. Scikit-learn (sklearn), Python'da kullanılan popüler bir makine öğrenimi kütüphanesidir ve model parametrelerini ayarlamak için kullanabileceğiniz çeşitli metodlar ve araçlar sunar. İşte scikit-learn kütüphanesinde parametre seçimi için hazırlanmış bazı önemli metodlar:
2. GridSearchCV: GridSearchCV, belirli bir model için tüm kombinasyonları deneyerek en iyi hiperparametre setini bulmanıza yardımcı olan bir hiperparametre ayarlama yöntemidir. GridSearchCV, bir hiperparametre ızgarası (grid) tanımlayarak ve bu ızgara üzerinde çapraz doğrulama (cross-validation) kullanarak çalışır.
3. RandomizedSearchCV: RandomizedSearchCV, GridSearchCV'ye benzer bir parametre ayarlama yöntemidir, ancak rastgele örneklenmiş bir hiperparametre uzayı üzerinde çalışır. Bu, tüm parametre kombinasyonlarını denemek yerine bir rastgele örnekleme yaklaşımı kullanır.
4. Bayesian Optimization: Scikit-learn, Bayesian Optimization için farklı kütüphanelerle (örneğin, scikit-optimize veya GPyOpt) entegre edilebilir. Bayesian Optimization, model performansını optimize etmek için bir olasılık modelini kullanır ve daha az denemeyle en iyi hiperparametreleri bulmanıza yardımcı olur.
5. Cross-Validation (Çapraz Doğrulama): Scikit-learn, parametre ayarlama sırasında modellerin performansını değerlendirmek için kullanabileceğiniz çeşitli çapraz doğrulama stratejileri sunar. Özellikle KFold, StratifiedFold ve TimeSeriesSplit gibi sınıflar kullanılır.
6. Learning Curve Analizi: Scikit-learn, modelin performansının eğitim örneklerinin sayısı ile nasıl değiştiğini anlamak için kullanabileceğiniz **learning\_curve** fonksiyonunu içerir. Bu, aşırı uyum (overfitting) veya yetersiz uyum (underfitting) durumlarını tespit etmenize yardımcı olabilir.
7. Validation Curve: **validation\_curve** fonksiyonu, belirli bir hiperparametrenin farklı değerleri için eğitim ve doğrulama performanslarını değerlendirmenize olanak tanır. Bu, hiperparametrelerin belirli bir değeri için en iyi performansı bulmanıza yardımcı olabilir.
8. Model Selection Modülleri: Scikit-learn, model seçimi ve karşılaştırması için kullanabileceğiniz birçok yardımcı işlev içerir. Örneğin, **GridSearchCV** veya **RandomizedSearchCV** ile birlikte **SelectKBest**, **SelectFromModel**, ve **RFE (Recursive Feature Elimination)** gibi özellik seçimi ve model seçimi yöntemleri kullanılabilir.

Bu metodlar ve araçlar, scikit-learn kütüphanesini kullanarak makine öğrenimi modellerinizi en iyi şekilde ayarlamanıza ve hiperparametrelerinizi optimize etmenize yardımcı olur. Hangi metodun kullanılacağı, veri kümenizin özelliklerine ve sorununuzun gereksinimlerine bağlı olarak değişebilir.

**KOD PARÇALARI VE AÇIKLAMALARI (İRİS FLOWER DATASET)**

Lojistik regresyon için İris Flower Dataset ini kullandım. Şimdi bu kodların açıklamasını yapacağım:

* İlk olarak gerekli Python kütüphanelerini içeri aktardım ve ‘datasets’ kullanarak iris veri setini jupyter notebook’a yükledim.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn import datasets

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

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

* Daha sonra iris veri setini dosyanın içerisine yükledim.

iris=pd.read\_csv("./IRIS.csv")

* Veriyi özellikler ve hedef olarak ayırdım.

X = iris.drop('species', axis=1) 🡪 bu satırda iris veri setinden ‘species’ adlı sütunu çıkarttım ve kalan sütunları ‘X’ adlı veri çerevesine atadım.

y = iris['species'] 🡪 Bu satırda iris veri setindeki ‘species’ sütununu ‘y’ adlı değişkene atadım. Y dediğimiz değişken çiçek türlerini içeren hedef değişkenimi temsil ediyor.

* Şimdi iki özelliğe göre veri setimizi görselleştirdim.

plt.scatter(X[y == 'setosa']['sepal\_length'], X[y == 'setosa']['sepal\_width'], label='Iris setosa', color='r', marker='o') 🡪 Çiçek yaprak uzunluğu ve genişliği arasındaki ilişkiyi göstermek için kullandım.

plt.scatter(X[y == 'versicolor']['sepal\_length'], X[y == 'versicolor']['sepal\_width'], label='Iris versicolor', color='g', marker='x')

plt.scatter(X[y == 'virginica']['sepal\_length'], X[y == 'virginica']['sepal\_width'], label='Iris virginica', color='b', marker='s')

plt.xlabel('Çiçek Yaprak Uzunluğu (cm)')

plt.ylabel('Çiçek Yaprak Genişliği (cm)') 🡪 x ve y eksenlerine etikerler ekledik.

plt.legend(loc='upper right') 🡪 burada grafikteki sınıf etiketlerini ekledik.

plt.title('Çiçek Yapraklarının Uzunluğu ve Genişliği') 🡪 grafiğin başlığını bu şekilde belirttim

plt.show() 🡪 son olarak grafiği görüntüledik.

* Şimdi giriş ve çıkış değerlerinin boyutlarına bakalım.

print(X.shape) 🡪 bu satır x adlı veri çerçevesinin boyutunu ekrana yazdırdı.

print(y.shape) 🡪 bu satır da aynı şekilde y adlı hedef değişkinin boyutunu ekrana yazdırdı.

* Veriyi eğitim ve test setlerine yazdırdım.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

Train\_test\_split fonksiyonu veri setini rastgele ikiye böler. İlk parametre plarak ‘X’ yani veri özelliklerini, ikinci parametre olarak ‘y’ yani hedef değişkeni alır.

* Şimdi veriyi ölçeklendirelim.

scaler = StandardScaler() 🡪 bu satırda ‘StandardScaler’ adlı bir ölçekleme nesnesi oluşturduk. Veri özelliklerini standartlaştırmak için bunu kullandım.

X\_train = scaler.fit\_transform(X\_train) 🡪 bu satırda ‘X\_train’ üzerinde fit\_transform işlemi yaptım. X\_train eğitim verilerini temsil ediyor. ‘fit’ işlemi StandardScaler taradından eğitim verilerinin standart sapma ve ortalama değerlerini hesaplamak için kullanılır.

X\_test = scaler.transform(X\_test) 🡪 bu satırda test verileri olan ‘X\_test’ üzerinde saddece ‘transform’ işlemi yaptım.

* Lojistik regresyonun modelini oluşturuyoruz ve eğitiyoruz.

model = LogisticRegression(max\_iter=1000, random\_state=42, penalty='l2', C=1.0) 🡪 bu satırda lojistik regresyon modelini oluşturdum ve ‘model’ adlı değişkene atadım.

model.fit(X\_train, y\_train) 🡪 bu satırda ‘model’ adlı lojistik regresyon modelini, eğitim verileri üzerinde eğittim. Bu sayede model, verileri kullanarak sınıflandırma görevini öğrendi.

lr = LogisticRegression(max\_iter=1000) 🡪 bu satırda başka bir lojistik regresyon modeli oluşturuldu.

lr.fit(X, y) 🡪 bu satırda ‘lr’ adlı lojistik regresyon modelini tüm veri seti üzerinde eğittim.

* Parametre ve bias değerlerimizi görelim.

print("Parametre: ", lr.coef\_) 🡪 bu satırda ‘lr.coef\_’ ifadesi lojistik regresyon modelinin öğrenilmiş katsayılarını ifade eder.

print("Bias: ", lr.intercept\_) 🡪 bu satırda ‘lr.intercept\_’ ifadesi lojistik regresyonun modelinin kesme noktasını yani bias değerini ifade eder.

* Test verileri üzerinde tahminler yapıyoruz.

y\_pred = model.predict(X\_test) 🡪 ‘y\_pred’ değişkeni, modelin tahminlerini saklayacak olan değişkendir. ‘model.predict’ ifadesi eğitilmiş lojistik regresyon modelini kullanarak veri özellikleri üzerinde sınıflandırma tahminlerinin yapılmasını sağlar.

* Aşağıdaki kodda modeli diske kaydediyoruz.

import joblib

joblib.dump(model, 'egitilmis\_model.pkl')

* Daha sonra kaydettiğimiz modeli yüklüyoruz.

loaded\_model = joblib.load('egitilmis\_model.pkl') 🡪 bu satırda daha önce diske kaydedilmiş olan bir lojistik regresyon modeli (‘egitilmis\_model.pkl’) yüklenir ve bu öğrenilmiş model ‘loaded\_model’ değişkenine atanır.

y\_pred = loaded\_model.predict(X\_test) 🡪 bu satırda, yüklenen lojistik regresyon modelini kullanarak, yeni veri özellikleri (‘X\_test’) üzerinde sınıflandırma tahminleri yapılır. Sonrasında da bu tahminler ‘y\_pred’ adlı değişkene atandı.

* Model performansını değerlendiriyoruz.

accuracy=accuracy\_score(y\_test, y\_pred) 🡪 bu satırda ‘accuracy\_score’ fonksiyonunu kullanarak doğruluk metriğini hesapladım. Bu metrik, modelin doğru tahminlerinin toplam veriye oranını ifade ediyor.

confusion=confusion\_matrix(y\_test, y\_pred) 🡪 bu satırda ‘confusion\_matrix’ fonksiyonunu kullanarak karışıklık matrisini hesapladım.

report=classification\_report(y\_test, y\_pred) 🡪 bu satırda ise ‘classification\_report’ fonksiyonunu kullanarak sınıflandırma raporunu hesapladım. Bu sayede model, her sınıf için ayrı ayrı değerlendirildi ve genel bir rapor oluşturuldu.

* Son olarak model performans değerlendirilmesini ekrana çıktı olarak yazdırıyoruz.

print(f"Model Doğruluk: {accuracy:.2f}")

print("Confusion Matrix:\n",confusion)

print("Sınıflandırma Raporu:\n",report)