به نام خدا

دانشگاه جامع انقلاب اسلامی تهران

داود حسنوند

۴۰۲۱۶۶۱۲۰۵

درس یادگیری ماشین

تمرین سری دوم

مدرس: دکتر مهدی علیاری

دانشگاه جامع انقالب اسلامی گروه سامانه های شبکه های

مثلث

https://colab.research.google.com/drive/1BWlY4jTtdOXOw1AJi3UYIxbavvxq1iDi#scrollTo=n_WUpW4wZN8l

https://github.com/ishtarpcamo/ishtar/blob/main/mini_p_2_mosalas_TAMRIN_1_3.ipynb

سوال دوم

https://colab.research.google.com/drive/1aQ1Aq0MZoEmr3-fUtV8-ZHly8oIREC1g#scrollTo=2J8wpOMHleGO

https://github.com/ishtarpcamo/ishtar/blob/main/davood_soal_2_mini_p_2.ipynb

سوال سوم و چهارم

https://colab.research.google.com/drive/10aWgwX9uj5kNDfrCzEsqiQsyCgBV5BR1#scrollTo=mCJDQhCbtKFl

پاسخ سوال ۱

در صورتی که فعال‌سازهای دو لایه انتهایی شبکه‌ی عصبی شما اتفاقی باشند، معمولا این موضوع می‌تواند به مشکلاتی در آموزش شبکه و عملکرد آن منجر شود. مثلا ممکن است باعث شود که شبکه‌ی عصبی شما به داده‌ها به طور نادرستی بیاموزد و عملکرد آن به‌طور کلی کاهش یابد. برای حل این مشکل می‌توانید از تکنیک‌هایی مانند استفاده از تابع فعال‌ساز متمرکز (مثل ReLU یا Leaky ReLU) و یا انجام تنظیمات مناسب برای وزن‌ها و بایاس‌ها استفاده کرد.

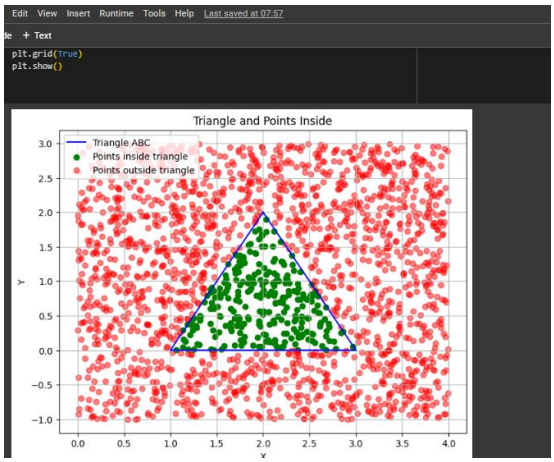
پاسخ قسمت دوم سوال ۱

در تابع فعال‌سازELU ، گرادیان تابع تعریف شده است به صورت زیر :

$$\frac{dELU(x)}{dx} = \begin{cases} 1 & x >= 0 \\ \alpha e^x & x < 0 \end{cases}$$

یکی از مزیت‌های ELU نسبت به ReLU این است که ELU مقدار گرادیان مناسبی برای ورودی‌های منفی نیز دارد، در حالی که در ReLU گرادیان برای ورودی‌های منفی صفر است که ممکن است باعث مشکلاتی مانند مرگ نورون‌ها (dying neurons) شود. به عبارت دیگر، ELU می‌تواند کمک کند تا شبکه‌ی عصبی بهتری آموزش ببیند و از مشکلاتی که در ReLU ممکن است پیش آید جلوگیری کند.

پاسخ قسمت سوم سوال ۱



```python
#import library
import numpy as np
import itertools
import numpy as np
import matplotlib.pyplot as plt
```

این کد ابتدا کتابخانه‌های مورد نیاز را وارد می‌کند، از جمله کتابخانه numpy برای انجام عملیات عددی و matplotlib برای رسم نمودارها. سپس از کلاس itertools برای ایجاد ترکیب‌ها استفاده می‌کند. در انتها نیز از matplotlib برای نمایش نمودارها استفاده می‌شود.

```python
#define muculloch pitts
class McCulloch_Pitts_neuron():

  def __init__(self , weights , threshold):
    self.weights = weights    #define weights
    self.threshold = threshold    #define threshold

  def model(self , x):
    #define model with threshold
    if self.weights @ x >= self.threshold:
        return 1
    else:
        return 0
```

این کد یک کلاس به نام McCulloch_Pitts_neuron تعریف می‌کند که یک نورون مک‌کالاک-پیتس را پیاده‌سازی می‌کند. این کلاس دارای دو ویژگی weights و threshold است که در تابع سازنده تعیین می‌شوند. همچنین این کلاس دارای یک متد به نام modelاست که ورودی x را گرفته و با استفاده از وزن‌ها و آستانه، خروجی نورون را محاسبه می‌کند و آن را برمی‌گرداند. اگر ضرب داخلی بین وزن‌ها و ورودی بیشتر یا مساوی آستانه باشد، خروجی ۱ و در غیر این صورت خروجی ۰ است.

```python
#define model for dataset
def Area(x, y):
  neur1 = McCulloch_Pitts_neuron([-2,-1], -6)
  neur2 = McCulloch_Pitts_neuron([0,1], 0)
  neur3 = McCulloch_Pitts_neuron([6.85,-3.50],6.70
  neur4 = McCulloch_Pitts_neuron([1, 1, 1], 3)

  z1 = neur1.model(np.array([x, y]))
  z2 = neur2.model(np.array([x, y]))
  z3 = neur3.model(np.array([x, y]))
  z4 = neur4.model(np.array([z1, z2, z3]))

  return list([z4])
```

این تابع Area مدلی را برای دیتاست خاصی تعریف می‌کند. ابتدا چهار نورون از نوع McCulloch_Pitts_neuron با وزن‌ها و آستانه‌های مختلف ایجاد می‌شود. سپس با استفاده از این نورون‌ها، خروجی‌های z1 تا z3 محاسبه می‌شوند با ورودی x و .y در نهایت، از این خروجی‌ها برای محاسبه خروجی نورون چهارم استفاده می‌شود و خروجی نهایی به صورت یک لیست از یک عنصر برگردانده می‌شود.

```python
# Generate random data points
num_points = 2000
x_values = np.random.uniform(0, 4, num_points)  # x-axis limits
y_values = np.random.uniform(-1, 3, num_points)  # y-axis limits

# Initialize lists to store data points for different z4 values
red_points = []
green_points = []


# Evaluate data points using the Area function
for i in range(num_points):
    z4_value = Area(x_values[i], y_values[i])
    if z4_value == [0]:  # z4 value is 0
        red_points.append((x_values[i], y_values[i]))
    else:  # z4 value is 1
        green_points.append((x_values[i], y_values[i]))

#Separate x and y values for red and green points
red_x, red_y = zip(*red_points)
green_x, green_y = zip(*green_points)
```

این قطعه کد یک تعداد ۲۰۰۰ نقطه داده‌ای تصادفی ایجاد می‌کند با محدوده‌های x از ۰ تا ۴ و y از -۱ تا ۳. سپس دو لیست red_points و green_points برای ذخیره نقاط داده‌ای با مقادیر مختلف z4 ایجاد می‌شود.

سپس با استفاده از تابعArea ، هر یک از نقاط داده‌ای ارزیابی شده و اگر z4 برابر با [۰] باشد، نقطه به لیست red_points اضافه می‌شود و در غیر این صورت به لیست green_points اضافه می‌شود.

در نهایت، مقادیر x و y برای نقاط قرمز و سبز جداگانه استخراج شده و درred_x ، red_y ،green_x ،green_y ذخیره می‌شوند.

این کد به طور خلاصه داده‌های تصادفی ایجاد می‌کند، آن‌ها را با استفاده از تابع Area ارزیابی می‌کند و نقاط را بر اساس مقدار z4به دو دسته قرمز و سبز تقسیم می‌کند. سپس مقادیر x و y برای هر دسته جداگانه استخراج می‌شوند.

```python
# Define the three points A, B, and C
A = np.array([2, 2])
B = np.array([3, 0])
C = np.array([1, 0])

# Create a line connecting points A, B, and C (to form the triangle)
triangle_x = [A[0], B[0], C[0], A[0]]
triangle_y = [A[1], B[1], C[1], A[1]]

# Plot the triangle and green points
plt.figure(figsize=(8, 6))
plt.plot(triangle_x, triangle_y, 'b-', label='Triangle ABC')
plt.scatter(green_x, green_y, color='green', marker='o', label='Points inside triangle')
plt.scatter(red_x, red_y, color='red', marker='o', label='Points outside triangle',alpha=0.5)

plt.xlabel('X')
plt.ylabel('Y')
plt.title('Triangle and Points Inside')
plt.legend(loc=2)
plt.grid(True)
plt.show()
```

این قطعه کد به تعریف سه نقطه A ، B و C می‌پردازد که با استفاده از آرایه‌های numpy تعریف شده‌اند. سپس یک خط از نقاط A ، B و C ایجاد شده و یک مثلث تشکیل می‌دهد.

سپس مقادیر x و y مربوط به نقاط مثلث در لیست‌های triangle_x و triangle_y ذخیره می‌شوند.

در ادامه، این مثلث و نقاط سبز و قرمز که قبلاً توسط کد قبلی تعیین شده‌اند، روی نمودار نمایش داده می‌شود. نقاط داخل مثلث با رنگ سبز و نقاط خارج از مثلث با رنگ قرمز و با شفافیتی کمتر نمایش داده می‌شوند.

در نهایت، نمودار حاوی مثلث و نقاط داخل و خارج از مثلث به همراه عنوان و توضیحات مناسب نمایش داده می‌شود.

۲ سوال دوم

```
+ Code  + Text

from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

این کد به شما امکان می‌دهد تا به حساب Google Drive خود در Google Colab دسترسی پیدا کنید و فایل‌ها و پوشه‌های موجود در آن را مشاهده و استفاده کنید. با اجرای این کد، یک پنجره باز می‌شود که از شما می‌خواهد تا به حساب Google خود وارد شوید و سپس یک کد تأیید را وارد کنید. سپس Google Colab به حساب Google Drive شما دسترسی پیدا می‌کند و می‌توانید فایل‌ها و پوشه‌های موجود در آن را مشاهده و استفاده کنید.

```
from sklearn.model_selection import train_test_split
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.io
from tensorflow.keras import Sequential
from keras.layers import Dense
from sklearn.preprocessing import LabelEncoder

import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings("ignore", category=ConvergenceWarning)

[5] !pip install --upgrade --no-cache-dir gdown

Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packa
Collecting gdown
  Downloading gdown-5.2.0-py3-none-any.whl (18 kB)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/d
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packag
```

در این قطعه کد، ابتدا بسته‌های مورد نیاز برای اجرای کد فراخوانی شده‌اند. این بسته‌ها شامل بسته‌های مربوط به مدل‌های ماشین لرنینگ، تصویرسازی، آرایه‌های عددی، پردازش داده، خواندن و نوشتن داده‌ها، و پیش‌پردازش داده‌ها می‌شوند.

سپس از ماژول train_test_split از sklearn.model_selection برای تقسیم داده‌ها به داده‌های آموزشی و آزمون استفاده شده است.

در ادامه، از %matplotlib inline برای نمایش نمودارها در خود نوت‌بوک استفاده شده است.

سپس آرایه‌های numpy، فریم داده pandas، ورودی و خروجی شبکه عصبی Sequential از tensorflow.keras، لایه‌های Dense از keras.layers، و LabelEncoder از sklearn.preprocessing فراخوانی شده‌اند.

در ادامه، با استفاده از بسته warnings و ConvergenceWarning از sklearn.exceptions، هشدارهای مربوط به عدم همگرایی مدل‌های ماشین لرنینگ فیلتر شده و نادیده گرفته می‌شود.

بنابراین، این قطعه کد بسته‌ها و تنظیمات اولیه مورد نیاز برای اجرای کدهای مربوط به ماشین لرنینگ و پردازش داده را فراخوانی می‌کند.

```
#Load data from .mat files
data98 = scipy.io.loadmat('/content/drive/MyDrive/mini_projec
data106 = scipy.io.loadmat('/content/drive/MyDrive/mini_proje
data119= scipy.io.loadmat('/content/drive/MyDrive/mini_projec
data131= scipy.io.loadmat('/content/drive/MyDrive/mini_projec
print(data98.keys())
print(data106.keys())
print(data119.keys())
print(data131.keys())
```

```
dict_keys(['__header__', '__version__', '__globals__', 'X098_
dict_keys(['__header__', '__version__', '__globals__', 'X106_
dict_keys(['__header__', '__version__', '__globals__', 'X119_
dict_keys(['__header__', '__version__', '__globals__', 'X131_
```

در این قطعه کد، ابتدا داده‌ها از فایل‌های mat. با استفاده از scipy.io.loadmat خوانده شده و در متغیرهای data98 ، data106، data119 و data131 ذخیره شده‌اند.

سپس با استفاده از keys() ، کلیدهای موجود در هر یک از این داده‌ها چاپ شده‌اند. این کلیدها نشان‌دهنده نام فیلدهای موجود در هر داده هستند و معمولاً برای دسترسی به داده‌های موجود در هر فیلد استفاده می‌شوند.

با چاپ کلیدها، می‌توانید بفهمید که هر داده شامل چه فیلدهایی است و بر اساس آن‌ها می‌توانید داده‌های مورد نیاز خود را از داخل فایل‌های .mat استخراج می کند.

```
normal = data98['X098_DE_time']
fault_Inner = data106['X106_DE_time']
fault_Ball=data119['X119_DE_time']
fault_Centered=data131['X131_DE_time']
print(f'number of data in X098_DE_time(normal) :{normal.size}')
print(f'number of data in X106_DE_time(fault) :{fault_Inner.size}')
print(f'number of data in X098_DE_time(normal) :{fault_Ball.size}')
print(f'number of data in X098_DE_time(normal) :{fault_Centered.size}'
```

```
number of data in X098_DE_time(normal) :483903
number of data in X106_DE_time(fault) :121991
number of data in X098_DE_time(normal) :121410
number of data in X098_DE_time(normal) :122426
```

در این قسمت از کد، داده‌های مربوط به زمان دی ای (DE time) از هر یک از داده‌ها با نام‌های X098_DE_time ، X106_DE_time، X119_DE_time و X131_DE_time از داخل دیکشنری‌های data98 ، data106، data119 و data131 استخراج شده و در متغیرهای normal، fault_Inner، fault_Ball و fault_Centered قرار داده شده‌اند.

سپس با استفاده از ویژگی size ، تعداد داده‌های موجود در هر یک از این متغیرها چاپ شده است. این تعداد داده‌ها نشان‌دهنده تعداد نقاط داده در هر یک از سیگنال‌های زمانی DE می‌باشد.

با این کد، می‌توانید تعداد داده‌های موجود در هر یک از سیگنال‌های زمانی DE مربوط به وضعیت‌های مختلف ماشین‌ها را بدست آورید و برای تحلیل و پردازش بعدی این داده‌ها از آن استفاده کرد .

```
[8]  #Construct a matrix for each data sample
     N = 700
     M = 100
     C1 = np.array([normal[N * i:N * (i + 1)] for i in range(M)]).reshape(N, M
     C2 = np.array([fault_Inner[N * i:N * (i + 1)] for i in range(M)]).reshape
     C3 = np.array([fault_Ball[N * i:N * (i + 1)] for i in range(M)]).reshape(
     C4 = np.array([fault_Centered[N * i:N * (i + 1)] for i in range(M)]).resh

     print(C1.shape)
     print(C2.shape)
     print(C3.shape)
     print(C4.shape)

     (700, 100)
     (700, 100)
     (700, 100)
     (700, 100)
```

در این قسمت از کد، یک ماتریس برای هر نمونه داده ساخته شده است. ابتدا مقادیر ثابت N و M تعریف شده‌اند که به ترتیب نشان‌دهنده تعداد نقاط در هر نمونه داده و تعداد نمونه‌ها هستند.

سپس با استفاده از یک حلقه for، برای هر یک از متغیرهای normal، fault_Inner، fault_Ball و fault_Centered، یک ماتریس N x M ساخته می‌شود. این ماتریس‌ها از زمان DE مربوط به هر یک از وضعیت‌ها ساخته شده‌اند.

در نهایت، با استفاده از ویژگی shape، ابعاد هر یک از این ماتریس‌ها چاپ شده است. این ابعاد نشان‌دهنده تعداد سطرها و ستون‌های مختلف ماتریس‌ها می‌باشد.

با این کد، می‌توانید داده‌های زمانی DE را برای هر یک از وضعیت‌های مختلف ماشین‌ها در قالب ماتریس‌های N x M آماده کنید که برای تحلیل و پردازش بعدی این داده‌ها مناسب می‌باشد.

```
X = np.vstack((C1, C2, C3, C4))
X.shape

(2800, 100)
```

این کد از تابع np.vstack برای ادغام (اتصال اعمال عملیات ریاضی) ماتریس‌های C1، C2، C3 و C4 استفاده می‌کند و نتیجه را در ماتریس X ذخیره می‌کند. سپس با استفاده از ویژگی shape، ابعاد ماتریس X چاپ می‌شود.

```
# Feature Extraction
data_std = np.std(X, axis=1, keepdims=True)
data_rms = np.sqrt(np.mean(np.square(X), axis=1)).reshape(-1, 1)
data_skewness= np.mean(((X - np.mean(X, axis=1, keepdims=True)) *
data_peak=np.max(np.abs(X), axis=1).reshape(-1, 1)
data_crest_factor=np.max(np.abs(X), axis=1).reshape(-1, 1) / np.s
data_absolute_mean=np.mean(np.abs(X), axis=1).reshape(-1, 1)
data_impact_factor = np.max(np.abs(X), axis=1).reshape(-1, 1) / n
data_square_mean_root = np.square(np.mean(np.sqrt(np.abs(X)), axi

X_new = np.hstack([data_std, data_rms,data_skewness,data_peak,dat
```

در این قسمت از کد، انواع ویژگی‌های استخراج شده از داده‌های ورودی X را مشاهده می‌کنیم. این ویژگی‌ها برای هر نمونه داده محاسبه شده و در ماتریس X_new ذخیره می‌شوند.

۱. data_std : انحراف معیار هر نمونه داده محاسبه شده و در یک ماتریس نگهداری می‌شود.

۲. data_rms : میانگین مربعات مقادیر هر نمونه داده محاسبه شده و سپس جذر می‌گیریم و در یک ماتریس نگهداری می‌شود.

۳. data_skewness : شاخص انحراف از توزیع نرمال برای هر نمونه داده محاسبه شده و در یک ماتریس نگهداری می‌شود.

٤. data_peak : بزرگترین مقدار مطلق هر نمونه داده محاسبه شده و در یک ماتریس نگهداری می‌شود.

٥. data_crest_factor : ضریب گوشه برای هر نمونه داده محاسبه شده و در یک ماتریس نگهداری می‌شود.

٦. data_absolute_mean : میانگین مقادیر مطلق هر نمونه داده محاسبه شده و در یک ماتریس نگهداری می‌شود.

۷. data_impact_factor : ضریب تأثیر برای هر نمونه داده محاسبه شده و در یک ماتریس نگهداری می‌شود.

۸. data_square_mean_root :میانگین مربعات جذر مقادیر مطلق هر نمونه داده محاسبه شده و در یک ماتریس نگهداری می‌شود.

سپس تمام این ویژگی‌ها در کنار هم قرار داده شده و در ماتریس X_new ذخیره می‌شوند.

این ویژگی‌ها می‌توانند برای آموزش مدل‌های یادگیری ماشین، تحلیل داده‌ها و یا استفاده در سیستم‌های تشخیص و پیش‌بینی استفاده شوند.

```
ones_array = np.ones((N,1))
y = np.vstack((1*ones_array,2*ones_array,3*ones_array,4*ones_array))
y.shape
```

```
(2800, 1)
```

در این کد، یک آرایه شامل مقادیر یک به ابعاد (Nایجاد می‌شود و در متغیر ones_array ذخیره می‌شود. سپس با استفاده از تابع np.vstack، این آرایه به صورت تکرار شده با ضریب‌های ۱، ۲، ۳ و ٤ به یکدیگر ادغام می‌شود و در متغیر y ذخیره می‌شود. در نهایت با استفاده از ویژگی shape، ابعاد ماتریس y چاپ می‌شود.

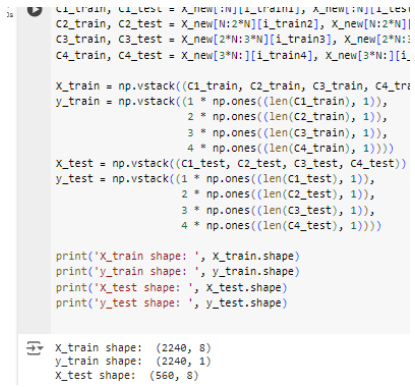
```
# Function to split the data
def data_split(X, per):
    limit = int(len(X) * per)
    idx = np.random.permutation(len(X))
    return idx[:limit], idx[limit:]

# Splitting the data
d_train = 0.8
i_train1, i_test1 = data_split(X_new[:N], d_train)
i_train2, i_test2 = data_split(X_new[N:2*N], d_train)
i_train3, i_test3 = data_split(X_new[2*N:3*N], d_train)
i_train4, i_test4 = data_split(X_new[3*N:], d_train)
```

در این قسمت از کد، یک تابع به نام data_split تعریف شده است که برای تقسیم داده‌ها به دو بخش آموزش و آزمون استفاده می‌شود. این تابع دو ورودی می‌گیرد: ماتریس داده‌ها X و درصد تقسیم داده‌ها برای بخش آموزش. سپس با استفاده از تابع np.random.permutation ابتدا اندیس‌های داده‌ها را تصادفی مخلوط می‌کند و سپس تعداد مشخصی از این اندیس‌ها را برای بخش آموزش و بقیه را برای بخش آزمون انتخاب می‌کند.

سپس داده‌ها به چهار بخش تقسیم

می‌شوند X_new[:N] ،X_new[N:2*N] ،X_new[2*N:3*N] وX_new[3*N:] و برای هر یک از این بخش‌ها اندیس‌های داده‌ها به دو بخش آموزش و آزمون تقسیم می‌شود. اندیس‌های بخش آموزش به ترتیب در i_train1 تا i_train4 ذخیره شده و اندیس‌های بخش آزمون به ترتیب در i_test1 تا i_test4 ذخیره می‌شوند.

این کاربردی است که اغلب در مسائل یادگیری ماشین و ارزیابی مدل‌ها استفاده می‌شود تا داده‌ها به صورت تصادفی و منصفانه بین بخش‌های آموزش و آزمون تقسیم شوند و از بیش‌برازش جلوگیری شود.



```
C2_train, C2_test = X_new[N:2*N][i_train2], X_new[N:2*N][
C3_train, C3_test = X_new[2*N:3*N][i_train3], X_new[2*N:
C4_train, C4_test = X_new[3*N:][i_train4], X_new[3*N:][i_

X_train = np.vstack((C1_train, C2_train, C3_train, C4_tra
y_train = np.vstack((1 * np.ones((len(C1_train), 1)),
                     2 * np.ones((len(C2_train), 1)),
                     3 * np.ones((len(C3_train), 1)),
                     4 * np.ones((len(C4_train), 1))))
X_test = np.vstack((C1_test, C2_test, C3_test, C4_test))
y_test = np.vstack((1 * np.ones((len(C1_test), 1)),
                    2 * np.ones((len(C2_test), 1)),
                    3 * np.ones((len(C3_test), 1)),
                    4 * np.ones((len(C4_test), 1))))

print('X_train shape: ', X_train.shape)
print('y_train shape: ', y_train.shape)
print('X_test shape: ', X_test.shape)
print('y_test shape: ', y_test.shape)
```

```
X_train shape:  (2240, 8)
y_train shape:  (2240, 1)
X_test shape:  (560, 8)
```

در این قسمت از کد، داده‌های آموزش و آزمون برای یک مسئله طبقه‌بندی چهار کلاسه تهیه می‌شود. ابتدا داده‌های هر یک از چهار بخش از داده‌های اصلی X_new که قبلتر به چهار بخش تقسیم شده بودند، بر اساس اندیس‌های آموزش و آزمون که در مراحل قبل مشخص شده بودند، انتخاب می‌شوند.

سپس داده‌های هر یک از این چهار بخش به ترتیب

در C1_train, C1_test, C2_train, C2_test, C3_train, C3_test, C4_train و C4_test ذخیره می‌شوند.

سپس داده‌های آموزش و آزمون برای هر یک از چهار کلاس به صورت جداگانه تهیه می‌شود. برای هر کلاس، داده‌های آموزش و آزمون به صورت جداگانه تهیه شده و برچسب‌های متناظر با آن‌ها (۱ تا ۴) نیز تهیه می‌شود.

در نهایت، داده‌های آموزش و آزمون برای همه چهار کلاس به صورت ترکیبی در X_train و X_test قرار می‌گیرند و برچسب‌های متناظر با آن‌ها نیز در y_train و y_test قرار می‌گیرند.

در انتها ابعاد داده‌های آموزش و آزمون چاپ می‌شود تا مشخص شود که چه تعداد داده‌ها و ویژگی‌ها در هر یک از مجموعه‌ها وجود دارد.

```
train_df = pd.DataFrame(np.hstack((X_train, y_train.reshape(-1, 1))), columns=[f'Feature {i+1}' for i in r
test_df = pd.DataFrame(np.hstack((X_test, y_test.reshape(-1, 1))), columns=[f'Feature {i+1}' for i in rang

train_df.to_csv('/content/drive/MyDrive/mini_project_2/dataset/train_data.csv', index=False)
test_df.to_csv('/content/drive/MyDrive/mini_project_2/dataset/test_data.csv', index=False)
```

```
df_train = pd.read_csv('/content/drive/MyDrive/mini_project_2/dataset/train_data.csv')
df_test = pd.read_csv('/content/drive/MyDrive/mini_project_2/dataset/test_data.csv')
```

```
df_test.head(10)
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8	Label
0	0.065405	0.068522	-0.724796	0.174611	2.548232	0.056850	3.071447	0.049543	1.0
1	0.060611	0.061707	-0.003211	0.143110	2.319196	0.048929	2.924874	0.041277	1.0
2	0.048710	0.062455	0.031869	0.146239	2.341527	0.050229	2.911976	0.042615	1.0

در این بخش از کد، تحلیل توزیع برچسب‌های کلاس‌ها در داده‌های آموزش و آزمون انجام می‌شود.

1. np.unique(y_train, return_counts=True):

- با استفاده از این دستور، مقادیر یکتا (برچسب‌های مختلف کلاس‌ها) در داده‌های آموزش y_trainاستخراج می‌شوند.
- return_counts=Trueباعث می‌شود که تعداد تکرار هر مقدار یکتا نیز برگردانده شود.
- مقادیر یکتا در y_trainدر unique_valuesو تعداد تکرار آن‌ها در countsذخیره می‌شوند.
- همچنین ابعاد y_trainنیز چاپ می‌شود.

2. np.unique(y_test, return_counts=True):

- به همان شکل که در بخش قبلی بود، مقادیر یکتا و تعداد تکرار آن‌ها برای برچسب‌های داده‌های آزمون y_testمحاسبه می‌شود.
- مقادیر یکتا در y_testدر unique_valuesو تعداد تکرار آن‌ها در countsذخیره می‌شوند.
- همچنین ابعاد y_testنیز چاپ می‌شود.

با این تحلیل، می‌توانید از توزیع برچسب‌های کلاس‌ها در داده‌های آموزش و آزمون مطلع شوید و اطمینان حاصل کنید که داده‌های شما به درستی تقسیم شده‌اند.

داده‌های آموزش و آزمون به دیتافریم‌های Pandas تبدیل شده و سپس در فایل‌های CSV ذخیره شده است. سپس داده‌های از فایل CSVخوانده شده و ۱۰ ردیف اول از داده‌های آزمون نمایش داده شده است.

1. train_df = pd.DataFrame(np.hstack((X_train, y_train.reshape(-1, 1))), columns=[f'Feature {i+1}' for i in range(X_train.shape[1])] + ['Label']):

- داده‌های آموزش (X_train) و برچسب‌های متناظر (y_train) افقی به هم چسبانده شده و به یک دیتافریم Pandasتبدیل می‌شود.

- نام ستون‌ها برای ویژگی‌های ورودی به صورت "Feature 1", "Feature 2", ... و برای برچسب‌ها به صورت "Label" تعیین می‌شود.

2. test_df = pd.DataFrame(np.hstack((X_test, y_test.reshape(-1, 1))), columns=[f'Feature {i+1}' for i in range(X_test.shape[1])] + ['Label']):

- به همان شکل که در بخش قبلی بود، داده‌های آزمون (X_test) و برچسب‌های متناظر (y_test) افقی به هم چسبانده شده و به یک دیتافریم Pandas تبدیل می‌شود.

- نام ستون‌ها نیز برای ویژگی‌های ورودی و برچسب‌ها به صورت مشابه تعیین می‌شود.

3. train_df.to_csv('/content/drive/MyDrive/mini_project_2/dataset/train_data.csv', index=False) و test_df.to_csv('/content/drive/MyDrive/mini_project_2/dataset/test_data.csv', index=False):

- دیتافریم‌های حاوی داده‌های آموزش و آزمون به فایل‌های CSV نام‌های train_data.csv و test_data.csv ذخیره می‌شود.

4. df_train = pd.read_csv('/content/drive/MyDrive/mini_project_2/dataset/train_data.csv') و df_test = pd.read_csv('/content/drive/MyDrive/mini_project_2/dataset/test_data.csv'):

- داده‌های آموزش و آزمون از فایل‌های CSV خوانده می‌شود و در

- دیتافریم‌های df_train و df_test ذخیره می‌شود.

5. df_test.head(10):

- نمایش ۱۰ ردیف اول از داده‌های آزمون با استفاده از متد head() از دیتافریم df_test.

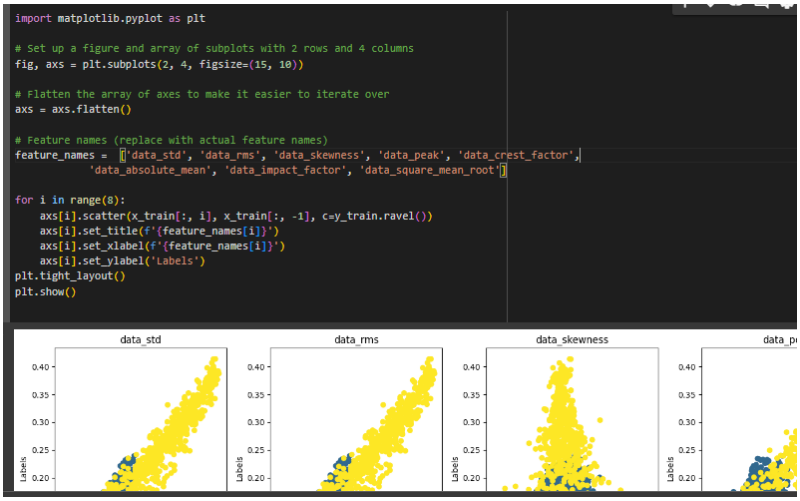
```
def extract_features_labels(df, feature_cols, label_col):
    X = df[feature_cols].values
    y = df[[label_col]].values
    return X, y

features = ['data_std', 'data_rms', 'data_skewness', 'data_peak', 'data_crest
            'data_absolute_mean', 'data_impact_factor', 'data_square_mean_roc
label = 'label'

x_train, y_train = extract_features_labels(df_train, features, label)
x_test, y_test = extract_features_labels(df_test, features, label)

# چاپ ابعاد مجموعه‌های داده
print(f'X_train shape: {x_train.shape}')
print(f'y_train shape: {y_train.shape}')
print(f'X_test shape: {x_test.shape}')
print(f'y_test shape: {y_test.shape}')
```

```
X_train shape: (2240, 8)
y_train shape: (2240, 1)
X_test shape: (560, 8)
y_test shape: (560, 1)
```

در این بخش از کد، یک تابع به نام extract_features_labels تعریف شده است که از یک دیتافریم و نام ستون‌های ویژگی‌ها و برچسب‌ها، ویژگی‌ها و برچسب‌های مورد نیاز را استخراج می‌کند. سپس این ویژگی‌ها و برچسب‌ها برای داده‌های آموزشی و آزمون جداگانه استخراج می‌شوند.

در اینجا، ویژگی‌های مورد نظر برای استخراج از دیتافریم df_train و df_testتعریف شده و سپس با استفاده از تابع extract_features_labels، ویژگی‌ها و برچسب‌ها از دیتافریم‌ها استخراج می‌شوند. سپس ابعاد داده‌های آموزشی و آزمون چاپ می‌شود.

این کد به شما کمک می‌کند تا ویژگی‌های مورد نظر را از داده‌ها استخراج کرده و برای استفاده در مدل‌های آموزشی بعدی آماده کنید.



در این بخش از کد، یک نمودار Scatter برای هر ویژگی (ویژگی‌های ۰ تا ۷) نسبت به برچسب‌ها ایجاد می‌شود. این نمودارها در یک شبکه ۲ در ۴ (۲ ردیف و ۴ ستون) قرار داده می‌شوند. هر نمودار Scatter نشان‌دهنده رابطه بین یک ویژگی و برچسب‌ها است.

متغیرهای اصلی که در این کد استفاده شده‌اند عبارتند از:

Fig :یک شیء شکل (figure) که نمودارها را شامل می‌شود.

Axs :یک آرایه از محورها (axes) که شامل ۲ ردیف و ۴ ستون است.

feature_names :یک لیست از نام‌های ویژگی‌ها که برای هر ویژگی تعیین شده است.

سپس در یک حلقه for از ۰ تا ۷، برای هر ویژگی، یک نمودار Scatter ایجاد می‌شود. هر نمودار Scatter شامل مقادیر ویژگی برای محور x و برچسب‌ها برای محور y است. رنگ‌ها بر اساس برچسب‌ها اختصاص داده شده‌اند.

سپس برای هر نمودار Scatter، عنوان (title)، برچسب محور x و برچسب محور y تنظیم می‌شود. در نهایت، با استفاده از plt.tight_layout() فضای مناسبی بین نمودارها فراهم می‌شود و با فراخوانی plt.show() نمودارها نمایش داده می‌شوند.

```
# Initial split into train and test sets
X_train_combined, X_test_combined, y_train_combined, y_test_comb

)


# Split the training data for validation
X_train_combined,X_valid,y_train_combined,y_valid= train_test_sp

)



print("Train set shape:", X_train_combined.shape)
print("Validation set shape:", X_valid.shape)
print("Test set shape:", X_test_combined.shape)
```

```
Train set shape: (1792, 8)
Validation set shape: (448, 8)
Test set shape: (560, 8)
```

در این بخش از کد، داده‌های آموزش و آزمون ترکیب می‌شوند تا یک مجموعه داده جدید بسازند. این کار با استفاده از تابع np.vstack() انجام می‌شود که به کمک آن داده‌ها به صورت عمودی به یکدیگر اضافه می‌شوند.

- X_combined = np.vstack((x_train, x_test)):

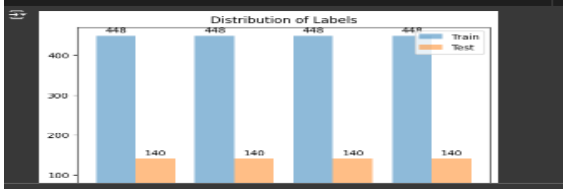
داده‌های ویژگی (X) از داده‌های آموزش (x_train) و داده‌های آزمون (x_test) ترکیب شده و در X_combined ذخیره می‌شود. این کار با اضافه کردن داده‌های آزمون به داده‌های آموزش، یک مجموعه داده جدید برای استفاده در مدلسازی ایجاد می‌شود.

- y_combined = np.vstack((y_train, y_test)):

برچسب‌ها (y) از داده‌های آموزش (y_train) و داده‌های آزمون (y_test) ترکیب شده و در y_combined ذخیره می‌شود. این کار با اضافه کردن برچسب‌های داده‌های آزمون به برچسب‌های داده‌های آموزش، یک مجموعه برچسب جدید برای استفاده در مدلسازی ایجاد می‌شود.

در نتیجه، X_combined حاوی داده‌های ترکیب شده و y_combined حاوی برچسب‌های ترکیب شده است که برای استفاده در مدلسازی و آموزش مدل‌های پیش‌بینی استفاده می‌شوند.

```
import numpy as np

# فرض کنید 'y_train_combined' و 'y_test_combined' لیبل‌های شما هستند
# محاسبه توزیع لیبل‌ها در مجموعه آموزش و تست
labels, counts_train = np.unique(y_train_combined, return_counts=True)
_, counts_test = np.unique(y_test_combined, return_counts=True)

# نمایش توزیع لیبل‌ها
fig, ax = plt.subplots()
train_bar = ax.bar(labels - 0.2, counts_train, width=0.4, alpha=0.5, label='Train')
test_bar = ax.bar(labels + 0.2, counts_test, width=0.4, alpha=0.5, label='Test')

ax.legend()
ax.set_title('Distribution of Labels')

# افزودن تعداد داده‌ها برای هر لیبل روی نمودار
def add_labels(bars):
    for bar in bars:
        height = bar.get_height()
        ax.annotate('{}'.format(height),
                    xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

add_labels(train_bar)
add_labels(test_bar)

plt.show()
```



در این بخش از کد، ابتدا توزیع لیبل‌ها بر روی داده‌های آموزشی و تست محاسبه می‌شود و سپس با استفاده از کتابخانه matplotlib، این توزیع‌ها به صورت نمودار مستطیلی نشان داده می‌شود.

1. labels, counts_train = np.unique(y_train_combined, return_counts=True):

این دستور با استفاده از تابع np.unique() تعداد دفعات ظاهر شدن هر لیبل در داده‌های آموزشی را محاسبه می‌کند و این اطلاعات را در labels(لیبل‌ها) و counts_train(تعداد ظاهر شدن هر لیبل در داده‌های آموزشی) ذخیره می‌کند. همین کار برای داده‌های تست نیز انجام می‌شود.

۲. سپس با استفاده از plt.subplots() یک شکل (figure) و یک محور (axes) برای نمودار ایجاد می‌شود.

3. train_bar = ax.bar(labels - 0.2, counts_train, width=0.4, alpha=0.5, label='Train'):

٤. با استفاده از ax.bar() نمودار مستطیلی برای توزیع لیبل‌ها در داده‌های آموزشی ایجاد می‌شود و در train_bar ذخیره می‌شود. همین کار برای داده‌های تست نیز انجام می‌شود.

٥. ax.legend() :افزودن علامت‌ها (legend) به نمودار برای نمایش نوع هر مستطیل (آموزش یا تست).

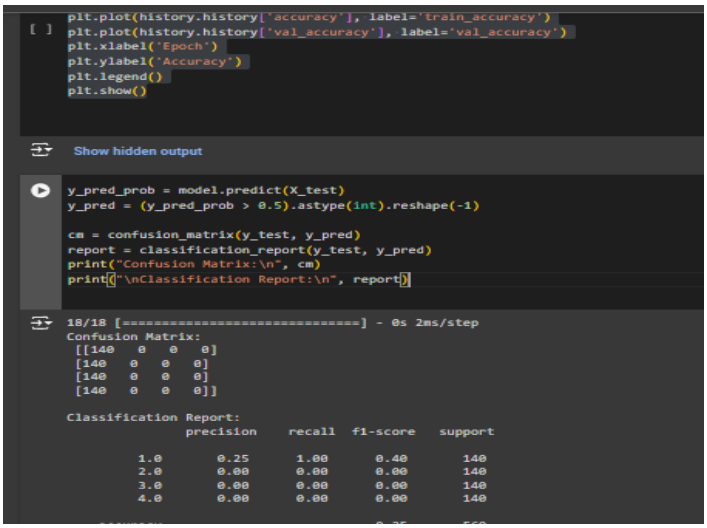
6. ax.set_title('Distribution of Labels'):

تنظیم عنوان نمودار.

۷. تابع add_labels() تعریف شده است که برای افزودن تعداد داده‌ها برای هر کلاس روی نمودار استفاده می‌شود. این تابع برای هر مستطیل، تعداد آن را بر روی نمودار نشان می‌دهد.

۸. با فراخوانی plt.show() نمودار نهایی نمایش داده می‌شود.

این کد به صورت تصویری توزیع لیبل‌ها در داده‌های آموزشی و تست را نشان می‌دهد و اطلاعات مربوط به تعداد داده‌ها برای هر کلاس را نیز نمایش می‌دهد.



در این کد، ابتدا داده‌های آموزشی، تست و اعتبارسنجی از فایل‌های CSV خوانده شده و به دیتافریم‌های train_data، test_data و valid_data تبدیل شده‌اند. سپس داده‌های ورودی و خروجی مدل برای آموزش، تست و اعتبارسنجی آماده می‌شوند.

سپس برچسب‌های کلاس‌ها با استفاده از LabelEncoder از نوع رشته به اعداد صحیح تبدیل شده و سپس به فرمت ONE-HOT تبدیل می‌شوند.

سپس یک مدل شبکه عصبی با استفاده از کتابخانه Keras تعریف می‌شود. این مدل شامل سه لایه با توابع فعال‌سازی مختلف است و برای آموزش با تابع هزینه categorical_crossentropy و بهینه‌ساز adam کامپایل می‌شود.

سپس نمودارهای مربوط به تغییرات هزینه و دقت در طول زمان آموزش رسم می‌شوند.

در نهایت، پیش‌بینی‌های مدل بر روی داده‌های تست انجام شده و ماتریس اشتباهات (confusion matrix) و گزارش طبقه‌بندی (classification report)برای ارزیابی عملکرد مدل چاپ می‌شود.

این کد به طور خلاصه داده‌ها را آماده می‌کند، یک مدل شبکه عصبی را تعریف و آموزش می‌دهد و سپس عملکرد مدل را بر روی داده‌های تست ارزیابی می‌کند.

سوال سوم

این کد Python برای بارگذاری یک دیتاست مربوط به بیماری‌های قلبی با استفاده از کتابخانه pandas و تقسیم داده‌ها به دو بخش آموزش و آزمون با استفاده از تابع train_test_split از کتابخانه scikit-learn استفاده می‌کند. سپس تعداد ردیف‌های هر بخش چاپ می‌شود.

توضیحات کد:

1. `import pandas as pd`

   این خط کتابخانه pandas را وارد می‌کند و آن را با نام pd فراخوانی می‌کند.

2. `from sklearn.model_selection import train_test_split`

   این خط تابع train_test_split از کتابخانه scikit-learn را وارد می‌کند.

3. `data = pd.read_csv('/content/heart.csv'):`

   این خط دیتاست مربوط به بیماری‌های قلبی را از یک فایل CSV بارگذاری می‌کند.

4. `train_data, test_data = train_test_split(data, test_size=0.15, random_state=42):`

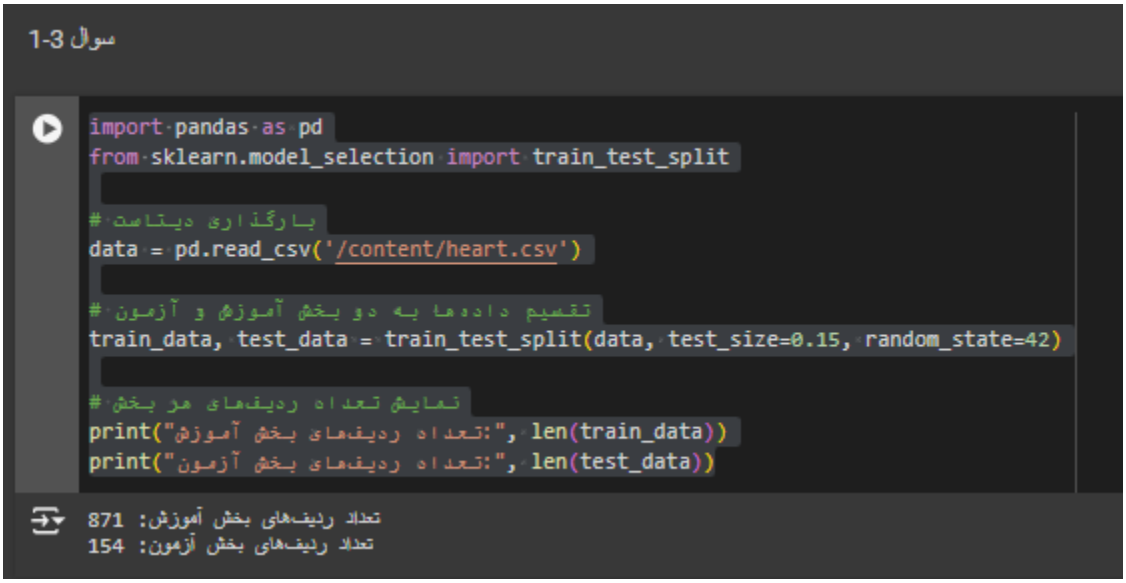
   این خط داده‌ها را به دو بخش آموزش و آزمون تقسیم می‌کند. در اینجا ۱۵٪ داده‌ها به عنوان

   داده‌های آزمون اختصاص داده شده و random_state=42 برای تولید نتایج قابل تکرار استفاده می‌شود.

   این خط تعداد ردیف‌های بخش آموزش را چاپ می‌کند: `print("تعداد ردیف‌های بخش آموزش:", len(train_data))`.

   این خط تعداد ردیف‌های بخش آزمون را چاپ می‌کند: `print("تعداد ردیف‌های بخش آزمون:", len(test_data))`.

این کد به طور خلاصه دیتاست مربوطه را بارگذاری کرده، آن را به دو بخش آموزش و آزمون تقسیم کرده و تعداد ردیف‌های هر بخش را نمایش می‌دهد.



قسمت بعدی

این کد Python برای آموزش یک مدل طبقه‌بندی با استفاده از الگوریتم درخت تصمیمی و ارزیابی دقت مدل استفاده می‌شود. همچنین متغیرهای متنی دیتاست با استفاده از LabelEncoder به متغیرهای عددی تبدیل می‌شوند.

توضیحات کد:

1) from sklearn.tree import DecisionTreeClassifier

این خط الگوریتم Decision Tree Classifier را از کتابخانه scikit-learn وارد می‌کند.

2) from sklearn.metrics import accuracy_score,
3) classification_report

این خط معیارهای ارزیابی دقت مدل مانند accuracy_score و classification_report را وارد می‌کند.

4) from sklearn.preprocessing import LabelEncoder

این خط : LabelEncoder را از scikit-learn

label_encoder = LabelEncoder():

یک نمونه از LabelEncoder ایجاد می‌شود.

5) data['target'] = label_encoder.fit_transform(data['target'])

متغیر هدف دیتاست از متنی به عددی تبدیل می‌شود.

6) X = data.drop('target', axis=1):

متغیرهای وابسته (features) از دیتاست حذف می‌شوند.

7) y = data['target']:

متغیر هدف جدا می‌شود.

8) X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42):

داده‌ها به دو بخش آموزش و آزمون تقسیم می‌شوند.

9) clf = DecisionTreeClassifier():

یک نمونه از Decision Tree Classifier ایجاد می‌شود.

10) clf.fit(X_train, y_train):

مدل با داده‌های آموزش آموزش داده می‌شود.

11) y_pred = clf.predict(X_test):

برچسب‌های پیش‌بینی شده برای داده‌های آزمون محاسبه می‌شود.

12) accuracy = accuracy_score(y_test, y_pred):

دقت مدل بر روی داده‌های آزمون محاسبه می‌شود.

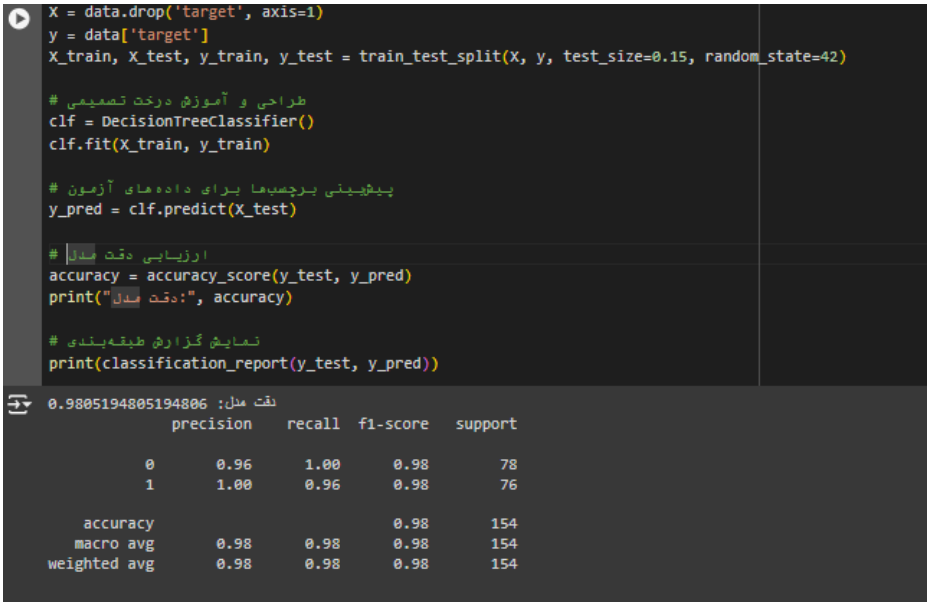
13) print("دقت مدل:", accuracy):

دقت مدل چاپ می‌شود.

14) print(classification_report(y_test, y_pred)):

گزارش طبقه‌بندی شامل معیارهای دقت، بازخوانی و F1 برای داده‌های آزمون چاپ می‌شود.

این کد به طور خلاصه یک مدل طبقه‌بندی با استفاده از الگوریتم Decision Tree Classifier آموزش داده و دقت مدل را برای داده‌های آزمون ارزیابی می‌کند. سپس گزارشی از عملکرد مدل بر روی داده‌های آزمون نمایش داده می‌شود.



```
X = data.drop('target', axis=1)
y = data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)

# طراحی و آموزش درخت تصمیمی
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# پیش‌بینی برچسب‌ها برای داده‌های آزمون
y_pred = clf.predict(X_test)

# ارزیابی دقت مدل
accuracy = accuracy_score(y_test, y_pred)
print("دقت مدل:", accuracy)

# نمایش گزارش طبقه‌بندی
print(classification_report(y_test, y_pred))
```

```
دقت مدل: 0.9805194805194806
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        78
           1       1.00      0.96      0.98        76

    accuracy                           0.98       154
   macro avg       0.98      0.98      0.98       154
weighted avg       0.98      0.98      0.98       154
```

قسمت سوم

این کد Python برای آموزش یک مدل طبقه‌بندی با استفاده از الگوریتم Random Forest و ارزیابی دقت مدل بر روی داده‌های آموزش و آزمون استفاده می‌شود. الگوریتم Random Forest یک مدل مبتنی بر مجموعه از درخت‌های تصمیم است که به صورت موازی آموزش داده می‌شود و از ترکیب پیش‌بینی‌های تک درخت‌ها برای افزایش دقت استفاده می‌کند.

توضیحات کد:

1. from sklearn.ensemble import RandomForestClassifier

   این خط الگوریتم Random Forest Classifier را از scikit-learn وارد می‌کند.

2. from sklearn.metrics import accuracy_score:

   این خط معیار ارزیابی دقت مدل را وارد می‌کند

3. rf_model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42):

   یک نمونه از الگوریتم Random Forest Classifier با تعداد درخت‌های ۱۰۰ و عمق درخت‌ها ۱۰ ایجاد می‌شود.

4. rf_model.fit(X_train, y_train):

مدل با داده‌های آموزش آموزش داده می‌شود.

5. train_pred = rf_model.predict(X_train):

برچسب‌های پیش‌بینی شده برای داده‌های آموزش محاسبه می‌شود.

6. test_pred = rf_model.predict(X_test):

برچسب‌های پیش‌بینی شده برای داده‌های آزمون محاسبه می‌شود.

train_accuracy = accuracy_score(y_train, train_pred):

دقت مدل بر روی داده‌های آموزش محاسبه می‌شود.

7. test_accuracy = accuracy_score(y_test, test_pred):

دقت مدل بر روی داده‌های آزمون محاسبه می‌شود.

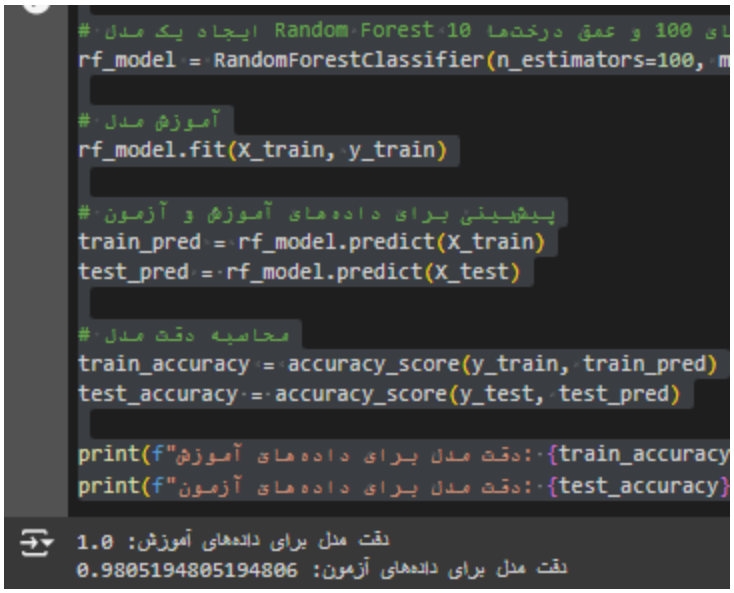
8. print(f"دقت مدل برای داده‌های آموزش: {train_accuracy}"):

دقت مدل بر روی داده‌های آموزش چاپ می‌شود.

print(f"دقت مدل برای داده‌های آزمون: {test_accuracy}"):

دقت مدل بر روی داده‌های آزمون چاپ می‌شود.

این کد به طور خلاصه یک مدل Random Forest با تعداد درخت‌های ۱۰۰ و عمق درخت‌ها ۱۰ آموزش داده و دقت مدل را بر روی داده‌های آموزش و آزمون ارزیابی می‌کند و نتایج را چاپ می‌کند.



قسمت بعدی

این کد Python برای انجام یک Grid Search بر روی مدل Random Forest و انتخاب بهترین فراپارامترها برای افزایش دقت مدل استفاده می‌شود .Grid Search یک روش سرچ سیستماتیک برای انتخاب بهترین مقادیر پارامترها در یک مدل می‌باشد.

توضیحات کد:

1. from sklearn.model_selection import GridSearchCV:

   این خط GridSearchCV را از scikit-learn وارد می‌کند.

2. from sklearn.ensemble import RandomForestClassifier:

   این خط الگوریتم Random Forest Classifier را وارد می‌کند.

3. rf_model = RandomForestClassifier(): یک نمونه از الگوریتم

   Random Forest Classifier بدون تنظیم پارامترها ایجاد می‌شود.

4. param_grid = {...}:

مجموعه‌ای از فراپارامترهای مختلف برای Grid Search تعیین می‌شود، از جمله تعداد درخت‌ها، عمق درخت‌ها و مقادیر مختلف برای تقسیم داده‌ها.

5. grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, scoring='accuracy'):

یک نمونه از GridSearchCV با مدل Random Forest و فراپارامترهای تعیین شده، ۵ fold cross-validation-و معیار ارزیابی دقت ایجاد می‌شود.

6. grid_search.fit(X_train, y_train):

Grid Search بر روی داده‌های آموزش اجرا شده و بهترین فراپارامترها و دقت مدل به دست آمده است.

7. best_params = grid_search.best_params_:

بهترین فراپارامترهای انتخاب شده توسط Grid Search استخراج می‌شود.

8. best_accuracy = grid_search.best_score_:

دقت مدل با استفاده از بهترین فراپارامترها محاسبه می‌شود.

9. print("بهترین فراپارامترها:"):

بهترین فراپارامترها چاپ می‌شود.

10. print(best_params):

مقادیر بهترین فراپارامترها چاپ می‌شود.

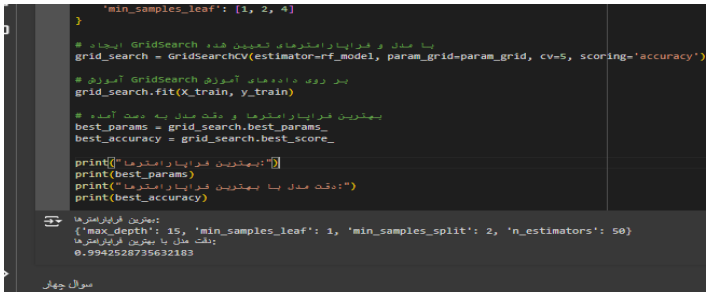
11. print("دقت مدل با بهترین فراپارامترها:"):

دقت مدل با استفاده از بهترین فراپارامترها چاپ می‌شود.

12. print(best_accuracy):

دقت مدل با استفاده از بهترین فراپارامترها چاپ می‌شود.

این کد به طور خلاصه Grid Search بر روی مدل Random Forest اجرا کرده و بهترین فراپارامترها و دقت مدل با استفاده از این فراپارامترها را چاپ می‌کند.



سوال چهارم

سوال چهار

بارگیری داده‌ها و نمایش چند ردیف ابتدایی:

```
import pandas as pd

data = pd.read_csv("drug200.csv")
print(data.head())
```

```
   Age Sex      BP Cholesterol  Na_to_K   Drug
0   23   F    HIGH        HIGH   25.355  drugY
1   47   M     LOW        HIGH   13.093  drugC
2   47   M     LOW        HIGH   10.114  drugC
3   28   F  NORMAL        HIGH    7.798  drugX
4   61   F     LOW        HIGH   18.043  drugY
```

این کد Python برای خواندن و نمایش اطلاعات اولیه یک فایل CSV به نام "drug200.csv" استفاده می‌شود.دیتاست

توضیحات کد:

import pandas as pd:

این خط کتابخانه pandas را وارد می‌کند و با نام pd قابل دسترسی می‌کند.

data = pd.read_csv("drug200.csv"):

این خط فایل CSV به نام "drug200.csv" را از دیسک خوانده و در یک DataFrame pandas به نام data ذخیره می‌کند.

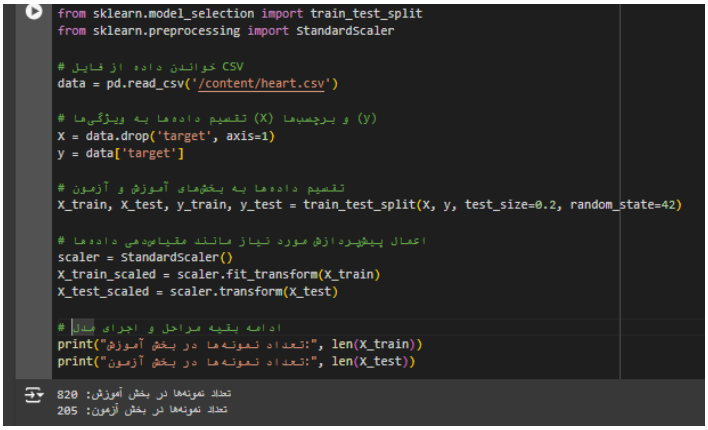
print(data.head()):

این خط اطلاعات اولیه از فایل CSV خوانده شده را نمایش می‌دهد. تابع head() اطلاعات اولیه از فایل را به صورت جدول نشان می‌دهد.

با اجرای این کد، اطلاعات اولیه از فایل "drug200.csv" شامل چند سطر از داده‌ها را مشاهده خواهید کرد. این اطلاعات شامل مقادیر مختلف در ستون‌ها و سطرهای مختلف داده‌ها می‌باشد.

تقسیم داده‌ها به بخش‌های آموزش و آزمون

این کد Python برای پیش‌پردازش داده‌ها و تقسیم آن‌ها به بخش‌های آموزش و آزمون برای استفاده در مدل‌سازی استفاده می‌شود.



توضیحات کد:

1. import pandas as pd:

این خط کتابخانه pandas را وارد می‌کند و با نام pd قابل دسترسی می‌کند.

2. from sklearn.model_selection import train_test_split:

از کتابخانه scikit-learn ، تابع train_test_split برای تقسیم داده‌ها به بخش‌های آموزش و آزمون استفاده می‌شود.

3. from sklearn.preprocessing import StandardScaler:

از کتابخانه scikit-learn ، کلاس StandardScaler برای اعمال مقیاس‌دهی داده‌ها استفاده می‌شود.

4. data = pd.read_csv('/content/heart.csv'):

این خط فایل CSV به نام "heart.csv" را از دیسک خوانده و در یک DataFrame pandas به نام data ذخیره می‌کند.

5. X = data.drop('target', axis=1):

این خط متغیر X را ایجاد می‌کند که شامل ویژگی‌ها همه ستون‌ها به جز ستون ("target")است.

6. y = data['target']:

این خط متغیر y را ایجاد می‌کند که شامل برچسب‌ها ستون ("target")است.

7. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42):

داده‌ها به بخش‌های آموزش و آزمون تقسیم می‌شوند با نسبت ۸۰٪ برای آموزش و ۲۰٪ برای آزمون.

8. scaler = StandardScaler():

یک نمونه از کلاس StandardScaler ایجاد می‌شود.

9. X_train_scaled = scaler.fit_transform(X_train):

داده‌های بخش آموزش مقیاس‌داده می‌شوند.

10. X_test_scaled = scaler.transform(X_test):

داده‌های بخش آزمون مقیاس‌داده می‌شوند.

11. Print ("تعداد نمونه‌ها در بخش آموزش:", len(X_train)):

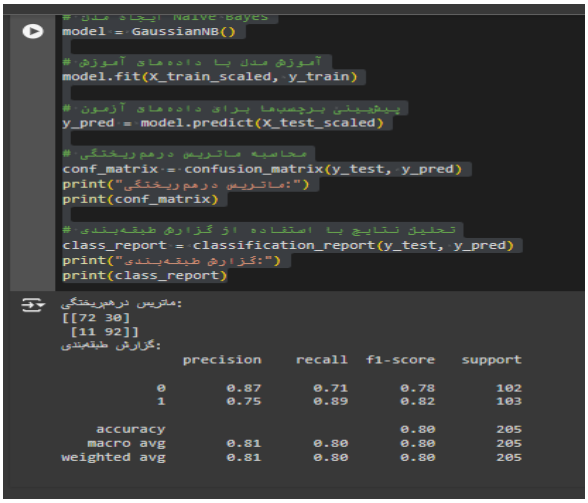
تعداد نمونه‌های موجود در بخش آموزش چاپ می‌شود.

12. Print ("تعداد نمونه‌ها در بخش آزمون:", len(X_test)):

تعداد نمونه‌های موجود در بخش آزمون چاپ می‌شود.

با اجرای این کد، داده‌ها از فایل "heart.csv" خوانده شده، به ویژگی‌ها و برچسب‌ها تقسیم شده و سپس به بخش‌های آموزش و آزمون تقسیم می‌شوند. سپس داده‌ها مقیاس‌داده شده و تعداد نمونه‌های موجود در هر بخش چاپ می‌شود.

ماتریس در هم ریختگی



این کد Python برای ایجاد یک مدل طبقه‌بندیNaive Bayes ، آموزش مدل با داده‌های آموزش، پیش‌بینی برچسب‌ها برای داده‌های آزمون، محاسبه ماتریس درهم‌ریختگی و تحلیل نتایج با استفاده از گزارش طبقه‌بندی استفاده می‌شود.

توضیحات کد:

1. from sklearn.naive_bayes import GaussianNB:

از کتابخانهscikit-learn ، کلاس GaussianNB برای ایجاد مدل Naive Bayes وارد می‌شود.

2. from sklearn.metrics import confusion_matrix, classification_report:

از کتابخانه scikit-learn ، توابع confusion_matrix و classification_report برای محاسبه ماتریس درهم‌ریختگی و تحلیل نتایج طبقه‌بندی وارد می‌شوند.

3. model = GaussianNB():

یک نمونه از مدل Naive Bayes ایجاد می‌شود.

4. model.fit(X_train_scaled, y_train):

مدل با داده‌های آموزش آموزش داده می‌شود.

5. y_pred = model.predict(X_test_scaled):

برچسب‌های پیش‌بینی شده برای داده‌های آزمون با استفاده از مدل محاسبه می‌شود.

6. conf_matrix = confusion_matrix(y_test, y_pred):

ماتریس درهم‌ریختگی برای برچسب‌های واقعی و پیش‌بینی شده محاسبه می‌شود.

7. print("ماتریس درهم‌ریختگی:"):

عبارت "ماتریس درهم‌ریختگی:" چاپ می‌شود.

8. print(conf_matrix):

ماتریس درهم‌ریختگی چاپ می‌شود.

9. class_report = classification_report(y_test, y_pred):

گزارش طبقه‌بندی برای برچسب‌های واقعی و پیش‌بینی شده محاسبه می‌شود.

10. print("گزارش طبقه‌بندی:"):

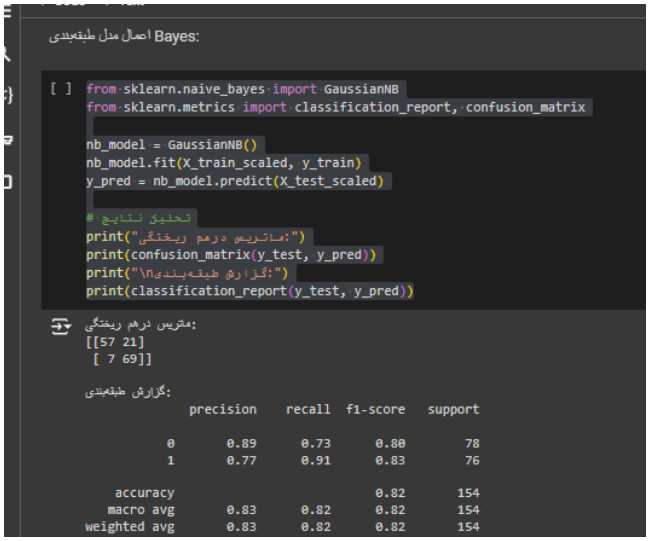
عبارت "گزارش طبقه‌بندی:" چاپ می‌شود.

11. print(class_report):

گزارش طبقه‌بندی چاپ می‌شود.

با اجرای این کد، یک مدل Naive Bayes آموزش داده می‌شود، سپس برچسب‌های پیش‌بینی شده برای داده‌های آزمون محاسبه شده و ماتریس درهم‌ریختگی و گزارش طبقه‌بندی برای ارزیابی عملکرد مدل چاپ می‌شود.

اصول مدل طبقه بندی Bayes



این کد Python برای ایجاد یک مدلNaive Bayes ، آموزش مدل با داده‌های آموزش، پیش‌بینی برچسب‌ها برای داده‌های آزمون، محاسبه ماتریس درهم‌ریختگی و تحلیل نتایج با استفاده از گزارش طبقه‌بندی استفاده می‌شود.

توضیحات کد:

1. from sklearn.naive_bayes import GaussianNB:

از کتابخانهscikit-learn ، کلاس GaussianNB برای ایجاد مدل Naive Bayes وارد می‌شود.

2. from sklearn.metrics import classification_report, confusion_matrix:

از کتابخانهscikit-learn ، توابع classification_report و confusion_matrix برای محاسبه گزارش طبقه‌بندی و ماتریس درهم‌ریختگی وارد می‌شوند.

3. nb_model = GaussianNB():

یک نمونه از مدل Naive Bayes ایجاد می‌شود.

4. nb_model.fit(X_train_scaled, y_train):

مدل با داده‌های آموزش آموزش داده می‌شود.

5. y_pred = nb_model.predict(X_test_scaled):

برچسب‌های پیش‌بینی شده برای داده‌های آزمون با استفاده از مدل محاسبه می‌شود.

6. Print ("ماتریس درهم ریختگی:"):

عبارت "ماتریس درهم ریختگی:" چاپ می‌شود.

Print (confusion_matrix(y_test, y_pred)):

ماتریس درهم‌ریختگی برای برچسب‌های واقعی و پیش‌بینی شده چاپ می‌شود.

7. Print ("\nگزارش طبقه‌بندی:"):

عبارت"\n" گزارش طبقه‌بندی:" چاپ می‌شود (با یک خط خالی).

8. Print (classification_report(y_test, y_pred)):

گزارش طبقه‌بندی برای برچسب‌های واقعی و پیش‌بینی شده چاپ می‌شود.

با اجرای این کد، یک مدل Naive Bayes آموزش داده می‌شود، سپس برچسب‌های پیش‌بینی شده برای داده‌های آزمون محاسبه شده و ماتریس درهم‌ریختگی و گزارش طبقه‌بندی برای ارزیابی عملکرد مدل چاپ می‌شود. این اطلاعات می‌تواند به کاربر کمک کند تا فهمیده که مدل چقدر خوب عمل کرده است و چگونه برچسب‌های پیش‌بینی شده با برچسب‌های واقعی متفاوتند.

پنج داده را به صورت تصادفی از مجموعهٔ آزمون انتخاب کنید و خروجی واقعی را با خروجی پیش بینی شده مقایسه کنید.

این کد Python برای اتصال به حساب Google Drive و مونت کردن آن در محیط اجرایی Google Colab استفاده می‌شود. این کد به کمک کتابخانه google.colab از Google Colab ، تابع drive.mount() را صدا می‌زند تا به حساب Google Drive کاربر متصل شود و محتویات آن در محیط اجرایی Google Colab مونت شود.

توضیحات کد:

1. from google.colab import drive:

از کتابخانهgoogle.colab ، تابع drive وارد می‌شود.

2. drive.mount('/content/drive'):

تابع mount با پارامتر '/content/drive' فراخوانی می‌شود. این تابع باعث می‌شود یک پنجره پاپ‌آپ باز شود و کاربر باید از طریق آن اجازه دسترسی به حساب Google Drive خود را بدهد. سپس Google Colab این اتصال را برقرار می‌کند و محتویات Google Drive در مسیر '/content/drive' در محیط اجرایی Google Colab قابل دسترسی می‌شود.

با اجرای این کد در Google Colab ، یک پنجره پاپ‌آپ باز می‌شود که از کاربر اجازه دسترسی به حساب Google Drive خود را می‌خواهد. پس از اتصال موفق، محتویات Google Drive قابل دسترسی در محیط اجرایی Google Colab خواهد

بود. این کار می‌تواند برای بارگذاری و استفاده از داده‌ها یا فایل‌های موجود در Google Drive در محیط Google Colab مفید باشد.

ادامه

این کد Python برای انتخاب پنج داده تصادفی از مجموعه آزمون، پیش‌بینی برچسب‌های این داده‌ها با استفاده از مدل آموزش دیده شده، و مقایسه برچسب‌های واقعی و پیش‌بینی شده استفاده می‌شود.

توضیحات کد:

1. import numpy as np:

   کتابخانه numpy با نام np وارد می‌شود.

2. random_indices = np.random.choice(len(X_test), 5, replace=False):

   پنج اندیس تصادفی از

   مجموعه آزمون X_test انتخاب می‌شود و در random_indices ذخیره می‌شود.

3. random_X_test = X_test.iloc[random_indices]:

   داده‌های متناظر با اندیس‌های تصادفی انتخاب شده از X_test در random_X_test ذخیره می‌شود.

4. random_y_test = y_test.iloc[random_indices]:

   برچسب‌های متناظر با اندیس‌های تصادفی انتخاب شده از y_test در random_y_test ذخیره می‌شود.

5. random_y_pred = model.predict(random_X_test):

   برچسب‌های پیش‌بینی شده برای داده‌های انتخاب شده با استفاده از مدل آموزش دیده شده، در random_y_pred ذخیره می‌شود.

   از یک حلقه for به اندازه ۵ برای مقایسه خروجی واقعی و پیش‌بینی شده برای هر داده استفاده می‌شود:

   - print("داده شماره", i+1):

     شماره داده مورد نظر چاپ می‌شود.

   - print("خروجی واقعی:", random_y_test.iloc[i]):

     برچسب واقعی برای داده مورد نظر چاپ می‌شود.

   - Print ("خروجی پیش‌بینی شده:", random_y_pred[i]):

برچسب پیش‌بینی شده برای داده مورد نظر چاپ می‌شود.

یک خط تیره برای جداکردن خروجی‌ها چاپ می‌شود :(Print ("----------

این کد با استفاده از مدل آموزش دیده شده، پنج داده تصادفی را از مجموعه آزمون انتخاب کرده، برچسب‌های واقعی و پیش‌بینی شده برای این داده‌ها را مقایسه کرده و نتایج را چاپ می‌کند. این کار می‌تواند به کاربر کمک کند تا بفهمد که مدل چقدر موفق بوده و چگونه برچسب‌های پیش‌بینی شده با برچسب‌های واقعی متفاوتند.