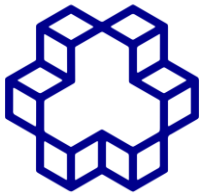
به نام خدا



دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

یادگیری ماشین

مینی پروژه دوم

Github link:

https://github.com/FatemehShokrollahiMoghadam

google drive link:

https://drive.google.com/drive/folders/1b5B582yp_CKuDpapPd8woH40hSIaU1yk?usp=sharing

نگارنده:

فاطمه شکراللهی مقدم

۴۰۲۰۷۳۶۴

بهار ۱۴۰۳

فهرست

پیشگفتار ۳

سوال اول ۴

۱-۱ ..... ۴

۱-۲ ..... ۵

۱-۳ ..... ۶

۱-۳-۱ طراحی شبکه ۶

۱-۳-۲ بررسی اثر اضافه کردن توابع فعال‌ساز ۹

سوال دوم ۱۲

۲-۱ ..... ۱۲

۲-۱-آ ۱۳

۲-۱-ب ۱۵

۲-۱-ج ۱۶

۲-۱-د ۱۷

۲-۲ ..... ۱۸

۲-۳ ..... ۲۲

۲-۴ ..... ۲۵

سوال سوم ۲۸

۳-۱ ..... ۲۸

۳-۲ ..... ۳۴

۳-۲-۱ ارزیابی ۳۷

۳-۲-۲ اثر تغییر فراپارامترها ۳۸

۳-۳ ..... ۴۰

سوال چهارم ۴۲

۴-۱ پیش پردازش داده ۴۳

۴-۲ NaiveBayes Classifier ۴۷

۴-۳ معیارهای ارزیابی ۴۸

۴-۳-۱ ماتریس درهم ریختگی ۴۸

۴-۳-۲ classification_report ۵۰

مراجع ۵۲

پیشگفتار

در صفحه اول گزارش لینگ گیت هاب و لینک گوگل درایو مربوط به نوت بوک های هر سوال آورده شده است.

در اینجا نیز لینک گوگل کولب نوت بوک هر سوال به ترتیب آورده می‌شود:

لینک نوت بوک سوال اول:

https://colab.research.google.com/drive/1pk6nG5853aNegjTSqxyZjNJBstGGhRg7?usp=sharing

لینک نوت بوک سوال دوم:

https://colab.research.google.com/drive/1RRSRa3sDYXwW6QYrc-WznwCcjZzvLhwO?usp=sharing

لینک نوت بوک سوال سوم:

https://colab.research.google.com/drive/1oi7TmkpGlQ1Zl3QMPW2lCOQXThFNdOEj?usp=sharing

لینک نوت بوک سوال چهارم:

https://colab.research.google.com/drive/1rvpLJa3PzRGSJeipE2122UuEzxkzr9tK?usp=sharing

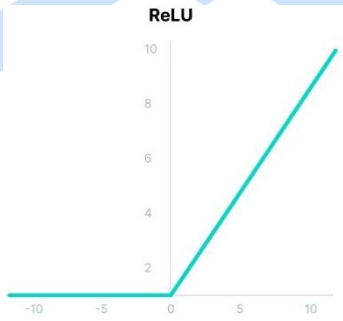
سوال اول

۱-۱

فرض کنید در یک مسألهٔ طبقه بندی دوکلاسه، دو لایهٔ انتهایی شبکهٔ شما فعال سازReLU و سیگموید است. چه اتفاقی می‌افتد؟

فعال‌ساز ReLU:

ReLU(Rectified Linear Unit) با ضابطه $F(x) = \max(0.x)$ به شکل زیر است:

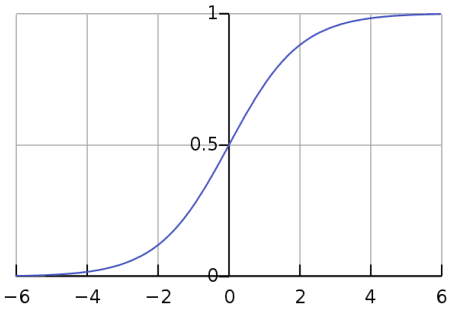


همانطور که پیداست، تابع فعال‌ساز ReLU در صورتی‌که ورودی کمتر از ۰ باشد، صفر (۰) و در غیر اینصورت مقدار خام ورودی را به عنوان خروجی می‌دهد. این تابع فعال ساز تمام نورون‌ها را فعال نمی‌کند و فقط نورون‌هایی که خروجی آنها مثبت است، فعال می شوند. این تابع، بر خلاف توابع خطی، مشتق ثابت ندارند و می توان از آنها در عملیات پس انتشار استفاده کرد.

ایراد اصلی این تابع مشکل مرگ ReLU است .منظور از مرگ ReLU غیرفعال شدن برخی از نورون‌ها و به دنبال آن، صفر شدن خروجی به ازای تمام ورودی هاست. در این حالت، هیچ گرادیانی جریان پیدا نمی‌کند و در صورتی‌که تعداد نورون‌های غیرفعال در شبکه عصبی زیاد شود، عملکرد مدل تحت تأثیر قرار می‌گیرد.

فعال‌ساز sigmoid:

تابع سیگموید با ضابطه $F(x) = \frac{1}{1+e^{-x}}$ به شکل زیر است:



این تابع، ورودی خود را به مقداری در بازه 0 تا 1 تبدیل میکند. هرچه مقدار ورودی بزرگتر باشد، مقدار خروجی این تابع به عدد ۱ و هرچه ورودی منفی تر باشد خروجی به ۰ نزدیکترخواهد بود. ضمنا این تابع نیز مشتق پذیر و مناسب برای استفاده در عملیات پس‌انتشار است. تابع فعالساز سیگموئید به دلیل قرار دادن خروجی بین دو مقدار ۰ و ۱ در طبقه بندی دو کلاسه مورد توجه است.

حال اگر در دو لایه انتهایی، به ترتیب از ReLU و sigmoid استفاده شود، مقادیر منفی با عبور از تابع ReLU مقدار صفر می گیرند و خروجی به ازای سایر مقادیر می تواند در بازه $[0, \infty]$ باشد. حال این خروجی به لایه بعد با تابع فعال ساز سیگموید می‌رود. باتوجه به ضابطه سیگموئید خروجی این تابع به ازای مقادیر صفر که خروجی لایه قبل به ازای وردی منفی بود، 0.5 می شود. از طرفی خروجی های مثبت لایه قبل با ورود به لایه با تابع فعالساز سیگموید، در خروجی مقداری در بازه $[0.5, 1]$ می گیرند. با توجه به شکل تابع سیگموید، این اتفاق موجب کاهش کارایی مدل در طبقه بندی می شود چراکه اطلاعات مقادیر منفی در لایه با فعال ساز ReLU از بین رفته و در خروجی لایه با فعال ساز سیگموئید مقدار 0.5 گرفته و طبقه بندی را دچار اشکال کرده است.

۱-۲

یک جایگزین برای ReLU در معادله ۱ آورده شده است. ضمن محاسبهٔ گرادیان آن، حداقل یک مزیت آن نسبت به ReLU را توضیح دهید.

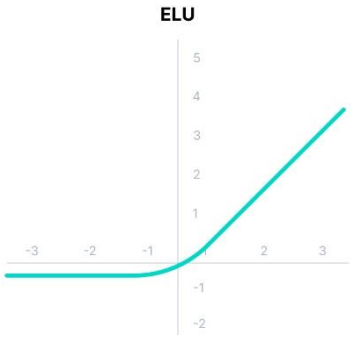
$$\mathrm{ELU}(x) = \begin{cases} x & x >= 0 \\ \alpha\,(e^x - 1) & x < 0 \end{cases} \qquad (۱)$$

این تابع برای حل مشکل مرگ نورون در تابع ReLU که در قسمت اول این سوال اشاره شد، به وجود آمده است. این تابع با نسبت دادن یک شیب به قسمت منفی، از حذف نورون با مقدار منفی جلوگیری می‌کند.

گرادیان F(x)=ELU(x):

$$F(x) = \begin{cases} 1 & x \geq 0 \\ \alpha e^x & x < 0 \end{cases} \Rightarrow F(x) = \begin{cases} 1 & x \geq 0 \\ F(x) + \alpha & x < 0 \end{cases}$$

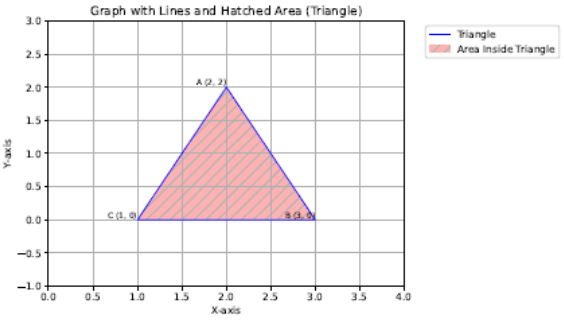
شکل تابع ELU به‌صورت زیر است:



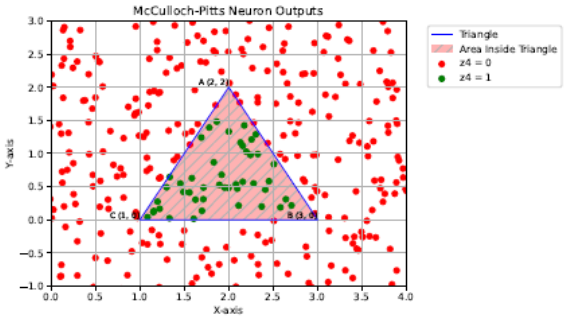
✓ این تابع به صورت smooth خم می‌شود درصورتی که ReLU با یک زاویه تیز خم می‌شود. این خم نرم ELU به وزن‌ها و بایاس‌ها کمک می‌کند تا در مسیر درست و با شیب درست حرکت کنند. همچنین همانطور که گفته شد، ELU از مرگ نورون‌ها نیز جلوگیری می‌کند.

۱-۳

به کمک یک نورون ساده یا پرسپترون یا نورونMcCulloch-Pittsشبکه ای طراحی کنید که بتواند ناحیهٔهاشورزدهٔ داخل مثلثی که در نمودار شکل (آ)نشان داده شده را از سایر نواحی تفکیک کند .پس از انجام مرحلهٔطراحی شبکه که می تواند به صورت دستی انجام شود، برنامه ای که در این دفترچه کد و در کلاس برای نورونMcCulloch-Pittsآموخته اید را به گونه ای توسعه دهید که ۲۰۰۰ نقطهٔ رندوم تولید کند و آن ها را به عنوان ورودی به شبکهٔ طراحی شده توسط شما دهد و نقاطی که خروجی۱ تولید می کنند را با رنگ سبز و نقاطی که خروجی۰تولید می کنند را با رنگ قرمز نشان دهد .خروجی تولیدشده توسط برنامهٔ شما باید به صورتی که در شکل (ب)نشان داده شده است باشد (به محدودهٔ عددی محورهایx وy هم دقت کنید) .اثر اضافه کردن دو تابع فعال ساز مختلف به فرآیند تصمیم گیری را هم بررسی کنید.



(آ) نمودار هاشورزدهٔ مورد سوال



(ب) خروجی مطلوب برنامه

۱-۳-۱ طراحی شبکه

نورونMcCulloch-Pitts در مواقعی که داده خطی تقریب پذیر باشد استفاده می‌شود. در این نورون وزن و آستانه تعریف می‌شود. آستانه تعیین می‌کند که خروجی نورون، صفر یا یک باشد. در اینجا نیز

می‌توان مثلث را با داشتن مختصات رئوس آن، به ۳ خط تقسیم کرد و ناحیه مشترک میان سه خط را به عنوان مثلث مدنظر معرفی کرد. هر خط بیانگر یک نورون McCulloch-Pitts خواهد بود.

معادله‌های خطوط تشکیل دهنده مثلث به‌صورت زیر است:

خط گذرنده از $\binom{2}{2}$و$\binom{3}{0}$: $y = -2x + 6$

خط گذرنده از $\binom{2}{2}$و$\binom{1}{0}$: $y = 2x - 2$

خط گذرنده از $\binom{1}{0}$و$\binom{3}{0}$: $y = 0$

ضرایب x و y به عنوان وزن های نورون و عرض از مبدا معادلات به عنوان آستانه در نظر گرفته می‌شوند.

با استفاده از از دفترچه کد اشاره شده در صورت سوال و مطالب تدریسیاری شبکه را طراحی می‌کنیم.

ابتدا کتابخانه های مورد نیاز را وارد می کنم. سپس یک کلاس برای نورون McCulloch_Pitts تشکیل می‌دهیم این کلاس با توجه به وزن و آستانه برای یک خط برسی می کند که داده در کدام سمت از خط قرار دارد. . اگر حاصل ضرب نقاط در وزن ها از آستانه کمتر باشد خروجی کلاس برابر با یک می شود در غیر این صورت برابر با صفر می شود.

```python
#import library
import numpy as np
import itertools
#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
```

حال مدل شبکه تحت عنوان تابع Area را با استفاده از کلاس تعریف شده و وزن ها و آستانه ها در معادلات خط، طراحی می‌کنیم. به این ترتیب که برای هر خط یک نورون تعریف کرده و خروجی آنها را با وزن ۱ به نورون دیگری به عنوان لایه خروجی می‌فرستیم و خروجی این لایه تعیین تعیین می کند که داده ورودی، درون یا بیرون مثلث است.

```python
#define model
def Area(x, y):
    neur1 = McCulloch_Pitts_neuron([-2, -1], -6)  # weights and threshold in first line
    neur2 = McCulloch_Pitts_neuron([2, -1], 2)    # weights and threshold in second line
    neur3 = McCulloch_Pitts_neuron([0, 1],   0)   # weights and threshold in third line
    neur4 = McCulloch_Pitts_neuron([1, 1, 1], 3)  # output layer

    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])
```

همانطور که پیداست ضرایب x و y در معادلات خطی که پیش‌تر ذکر شد به عنوان وزن و عرض از مبدا خطوط به عنوان آستانه در نظر گرفته شده‌اند. متد model که در کد فوق استفاده شده در بدنه کلاس نورون McCulloch_Pitts تعریف شده است و عملیات آموزش را انجام می‌دهد.

بنابراین تا اینجا توانستیم شبکه‌ای طراحی کنیم که ناحیه درون مثلث را از سایر نواحی تفکیک کند.

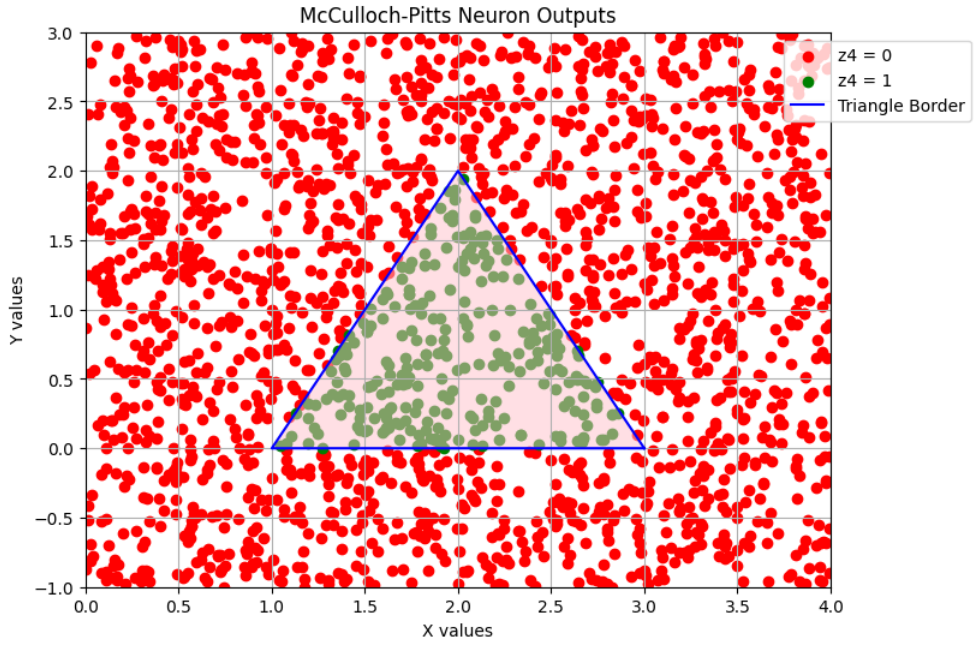
حال ۲۰۰۰ نقطه به‌صورت تصادفی تولید می‌کنیم و به عنوان ورودی به شبکه طراحی شده می‌دهیم.

نقاط سبز و قرمز را بگونه ای تعریف می‌کنیم که داده های درون مثلث جزو نقاط سبز و داده‌های بیرون مثلث جزو نقاط قرمز باشند.

داده‌هایی که در دسته نقاط سبز قرار دادیم را با رنگ سبز و داده‌های دسته قرمز را با رنگ قرمز رسم می‌کنیم. سپس مثلث مورد نظر را با داشتن مختصات رئوس رسم می کنیم. محدوده عددی محورهای x وy را مطابق شکل سوال در نظر می‌گیریم.

```python
import numpy as np
import matplotlib.pyplot as plt

# Generate random data points
np.random_state=(64)
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)

# Plotting
plt.figure(figsize=(8, 6))
plt.scatter(red_x, red_y, color='red', label='z4 = 0')
plt.scatter(green_x, green_y, color='green', label='z4 = 1')
plt.xlabel('X values')
plt.ylabel('Y values')
plt.title('McCulloch-Pitts Neuron Outputs')
# Enable grid
plt.grid(True)
# Define triangle vertices
triangle_vertices = np.array([(3, 0), (1, 0), (2, 2)])
# Add triangle to plot
plt.gca().add_patch(plt.Polygon(triangle_vertices, color='pink', alpha=0.5))
# Plot triangle border
plt.plot(*zip(*triangle_vertices, triangle_vertices[0]), 'b-', label='Triangle Border')
# Set axis limits
plt.xlim(0, 4)
plt.ylim(-1, 3)
# Add legend
plt.legend()
# Position the legends at the top and right
plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1.0))
```

ناحیه هاشور خورده مثلث را با رنگ صورتی نمایش می‌دهیم. خواهیم دید نقاط سبز رنگ درون مثلث و نقاط قرمز بیرون مثلث قرار می‌گیرند:



۲-۳-۱ بررسی اثر اضافه کردن توابع فعال‌ساز

توابع فعالساز را خارج از کلاس نورون McCulloch_Pitts تعریف می‌کنیم. در اینجا از توابع sigmoid و ReLU استفاده می‌کنیم:

```
#import library
import numpy as np
import itertools

def sigmoid(x):
    return 1/(1+np.exp(-x))

def relu(x):
    return np.maximum(0, x)
```

سپس برای کلاس متغیر ورودی جدیدی برای تعیین تابع فعال‌ساز تعریف می‌کنیم. در داخل کلاس تابع فعال‌ساز را None قرار می‌دهیم. که در این‌صورت نورون بطور خطی رفتار می‌کند.

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

  def __init__(self , weights , threshold, af=None):
    self.weights = weights        #define weights
    self.threshold = threshold    #define threshold
    self.af = af

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

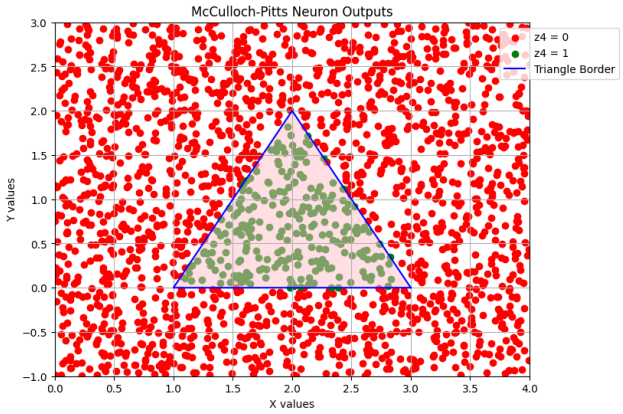
برای اختصاص تابع فعال‌ساز، آن را در مدل شبکه طراحی شده وارد می‌کنیم:

```
#define model for dataset
def Area(x, y):
    neur1 = McCulloch_Pitts_neuron([-2, -1], -6, af=sigmoid)  # weights and threshold in first line
    neur2 = McCulloch_Pitts_neuron([2, -1], 2, af=sigmoid)    # weights and threshold in second line
    neur3 = McCulloch_Pitts_neuron([0, 1],   0, af=sigmoid)   # weights and threshold in third line
    neur4 = McCulloch_Pitts_neuron([1, 1, 1], 3, af=sigmoid)  # output layer

    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])
```

مابقی مراحل مانند قسمت قبل دنبال می‌شود:

```
import numpy as np
import matplotlib.pyplot as plt

# Generate random data points
np.random_state=(64)
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 z5 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)

# Plotting
plt.figure(figsize=(8, 6))
plt.scatter(red_x, red_y, color='red', label='z4 = 0')
plt.scatter(green_x, green_y, color='green', label='z4 = 1')
plt.xlabel('X values')
plt.ylabel('Y values')
plt.title('McCulloch-Pitts Neuron Outputs')
plt.grid(True)  # Enable grid

# Define triangle vertices
triangle_vertices = np.array([(3, 0), (1, 0), (2, 2)])
# Add triangle to plot
plt.gca().add_patch(plt.Polygon(triangle_vertices, color='pink', alpha=0.5))
# Plot triangle border
plt.plot(*zip(*triangle_vertices, triangle_vertices[0]), 'b-', label='Triangle Border')
# Set axis limits
plt.xlim(0, 4)
plt.ylim(-1, 3)
# Add legend
plt.legend()
```

نتیجه به‌صورت زیر است:

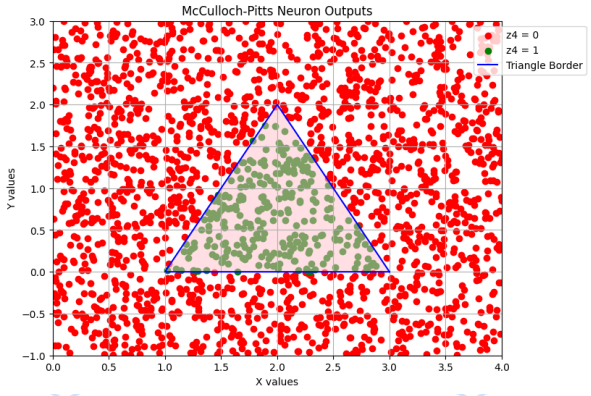


به‌طور مشابه برای تابع فعال‌ساز ReLU داریم:

```python
#define model for dataset
def Area(x, y):
    neur1 = McCulloch_Pitts_neuron([-2, -1], -6, af=relu)  # weights and threshold in first line
    neur2 = McCulloch_Pitts_neuron([2, -1], 2, af=relu)    # weights and threshold in second line
    neur3 = McCulloch_Pitts_neuron([0, 1],   0, af=relu)   # weights and threshold in third line
    neur4 = McCulloch_Pitts_neuron([1, 1, 1], 3, af=relu)  # output layer

    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])
```

```python
import numpy as np
import matplotlib.pyplot as plt

# Generate random data points
np.random_state=(64)
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 z5 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)

# Plotting
plt.figure(figsize=(8, 6))
plt.scatter(red_x, red_y, color='red', label='z4 = 0')
plt.scatter(green_x, green_y, color='green', label='z4 = 1')
plt.xlabel('X values')
plt.ylabel('Y values')
plt.title('McCulloch-Pitts Neuron Outputs')
plt.grid(True)  # Enable grid

# Define triangle vertices
triangle_vertices = np.array([(3, 0), (1, 0), (2, 2)])
# Add triangle to plot
plt.gca().add_patch(plt.Polygon(triangle_vertices, color='pink', alpha=0.5))
# Plot triangle border
plt.plot(*zip(*triangle_vertices, triangle_vertices[0]), 'b-', label='Triangle Border')
# Set axis limits
plt.xlim(0, 4)
plt.ylim(-1, 3)
# Add legend
plt.legend()
# Position the legends at the top and right
plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1.0))
```

نتیجه به‌صورت زیر است:



از آنجایی که این داده‌های درون مثلث را به سه خط تفکیک کردیم و مسئله خطی تفکیک پذیر شد، شبکه طراحی شده بدون توابع فعال‌ساز غیر خطی نیز عملکرد مطلوبی داشت. همانطور که در بالا گزارش شد عملیات تفکیک در حضور توابع sigmoid و ReLU نیز به خوبی انجام گرفت و داده‌های درون مثلث با رنگ سبز و سایر داده‌ها با رنگ قرمز مشخص شده‌اند.

سوال دوم

دیتاست CWRU که در مینی پروژهٔ شمارهٔ یک با آن آشنا شدید را به خاطر آورید .علاوه بر دو کلاسی که در آن مینی پروژه در نظر گرفتید، با مراجعه به صفحهٔ داده های عیب در حالت 12k دو کلاس دیگر نیز از طریق فایل‌های B007_X و IR007@6_X اضافه کنید. با انجام این کار یک کلاس دادهٔ سالم و سه کلاس از داده های دارای سه عیب متفاوت خواهید داشت. در مورد این‌که هر فایل مربوط به چه نوع عیبی است به صورت کوتاه توضیح دهید.

در ادامه آنچه در توضیحات داده‌های این دیتاست در گزارش مینی پروژه اول آمد، می دانیم صفحه داده های عیب در حالت 12k داده های عیب جمع‌آوری شده در فرکانس نمونه‌برداری 12kHz را نشان می‌دهد.

حرف اول نام فایل های داده، موقعیت خطا را نشان می دهد، سه عدد بعدی نشان دهنده قطر عیب و آخرین عدد نشان دهنده بارهای تحمل کننده است. به عنوان مثال، فایل داده B007_0 حاوی داده های خطای بلبرینگ با قطر بلبرینگ معیوب ۰,۰۰۷ اینچ است که بدون بارکار می کند. فایل داده OR007@6_0 شامل داده‌های خطای بلبرینگ نوع بیرونی با قطر ۰,۰۰۷ اینچ است که در مرکز قرار دارد (خطا در موقعیت ساعت ۶) و موتور بدون بار کار می‌کند.

باتوجه به شماره دانشجویی با دورقم آخر۶۴، کلاس های B007_0 و OR007@6_0 علاوه بر کلاس هایی که در مینی پروژه اول داشتیم، مجموعه دیتاست را تشکیل داده‌اند.

۲-۱

در ادامه، تمام کارهایی که در بخش دوم سوال۲مینی پروژه اول برای استخراج ویژگی و آماده سازی دیتا انجام داده بودید را روی دیتاست جدید خود پیاده سازی کنید. در قسمت تقسیم بندی داده ها، یک بخش برای اعتبارسنجی به بخش های آموزش و آزمون اضافه کنید و توضیح دهید که کاربرد این بخش چیست.

ابتدا با دستور gdown، داده های هر کلاس را در محیط گوگل کولب وارد می‌کنیم.

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1NJg1Zod9AZC2a_O3O6Ote8GQ_zxZSNI9
!gdown 1HO-CTWk5ReNXJhjF0QxDvKGrQFm4Y87P
!gdown 1U7wB1Nn4Zn7NgSexQJZC7vQbWCb21G_2
!gdown 1yjH21_vRXq9mggsuJpHYreuHuUHeAlHQ
```

سپس ستون های مورد استفاده از هر کلاس را جدا کرده و مجموعه داده با ۴ کلاس را تشکیل می دهیم.

ستون X097_DE_time از کلاس نرمال و ستون های X105_DE_time و X118_DE_time و X130_DE_time از کلاس های عیب استفاده می‌شود. سپس با دستور ()isnull. وجود داده پوچ را در ستون های انتخاب شده، بررسی می‌کنیم:

```
Normal = pd.read_csv('/content/Normal.csv')

Normal=Normal.X097_DE_time
# Show the number of Normal null values in each column
null_counts1 = Normal.isnull().sum()
print("Number of Normal null values in each column:")
print(null_counts1)
Fault1 = pd.read_csv('/content/Fault1.csv')
Fault1=Fault1.X105_DE_time
# Show the number of null values in each column
null_counts2 = Fault1.isnull().sum()
print("Number of Fault1 null values in each column:")
print(null_counts2)
Fault2 = pd.read_csv('/content/Fault2.csv')
Fault2=Fault2.X118_DE_time
# Show the number of null values in each column
null_counts3 = Fault2.isnull().sum()
print("Number of Fault2 null values in each column:")
print(null_counts3)
Fault3 = pd.read_csv('/content/Fault3.csv')
Fault3=Fault3.X130_DE_time
# Show the number of null values in each column
null_counts4 = Fault3.isnull().sum()
print("Number of Fault3 null values in each column:")
print(null_counts4)
```

نتیجه به‌صورت روبرو است:

```
Number of Normal null values in each column:
0
Number of Fault1 null values in each column:
0
Number of Fault2 null values in each column:
0
Number of Fault3 null values in each column:
0
```

بنابراین مقدار پوچی در هیچ کدام از کلاس ها وجود ندارد.

۲-۱-آ

از هر کلاس ۱۰۰ نمونه با طول ۲۰۰ جدا می‌کنیم. یک ماتریس از داده ها ی هر چهار کلاس به همراه برچسب مربوطه تشکیل می‌دهیم. برای جدا کردن نمونه ها از هر کلاس، از یک حلقه for استفاده شده است که در هر تکرار، یک نمونه با طول ۲۰۰ از داده ها ی کلاسمربوطه را بر می گرداند. در این عملیات ۱۰۰ نمونه تصادفی از هر کلاس انتخاب می شوند و در ماتریسی قرار می گیرند. (برای اینکه انتخاب رندوم

در تکرار بعد ثابت باشد از random.seed(64) استفاده شد.) در نهایت، یک ماتریس داده (X) از ترکیب نمونه های هر چهارکلاس خواهیم داشت(ابعاد این ماتریس200*400 است.) و سپس به بر چسب زدن به نمونه ها می‌پردازیم. داده های نرمال با برچسب 0 و انواع داده های عیب با برچسب های ۱و۲و۳ نمایش داده می‌شوند. الحاق داده ها و برچسب ها با د ستورnp.hstacks انجام می‌شود.

```
Normal_data = Normal
Fault_1 = Fault1
Fault_2 = Fault2
Fault_3 = Fault3
# Extract 100 samples with length of 200
sample_length = 200
num_samples = 100
Normal_samples = []
Fault1_samples = []
Fault2_samples = []
Fault3_samples = []
# Extract samples from class Normal
for i in range(num_samples):
    np.random.seed(64)
    start_idx = np.random.randint(0, len(Normal_data) - sample_length + 1)
    sample = Normal_data[start_idx:start_idx + sample_length]
    Normal_samples.append(sample)
# Extract samples from class Fault1
for i in range(num_samples):
    np.random.seed(64)
    start_idx = np.random.randint(0, len(Fault_1) - sample_length + 1)
    sample = Fault_1[start_idx:start_idx + sample_length]
    Fault1_samples.append(sample)
# Extract samples from class Fault2
for i in range(num_samples):
    np.random.seed(64)
    start_idx = np.random.randint(0, len(Fault_2) - sample_length + 1)
    sample = Fault_2[start_idx:start_idx + sample_length]
    Fault2_samples.append(sample)
    # Extract samples from class Fault3
for i in range(num_samples):
    np.random.seed(64)
    start_idx = np.random.randint(0, len(Fault_3) - sample_length + 1)
    sample = Fault_3[start_idx:start_idx + sample_length]
    Fault3_samples.append(sample)
# Convert lists of samples to numpy arrays
Normal_samples = np.array(Normal_samples)
Fault1_samples = np.array(Fault1_samples)
Fault2_samples = np.array(Fault2_samples)
Fault3_samples = np.array(Fault3_samples)
# Create labels for the samples
Normal_labels = np.zeros((num_samples, 1))  # Assuming class Normal is labeled as 0
Fault1_labels = np.ones((num_samples, 1))   # Assuming class Fault  is labeled as 1
Fault2_labels= np.full((num_samples, 1), 2) # Assuming class Fault  is labeled as 2
Fault3_labels= np.full((num_samples, 1), 3) # Assuming class Fault  is labeled as 3
# Concatenate the data and labels for both classes
data_matrix = np.vstack((Normal_samples, Fault1_samples, Fault2_samples, Fault3_samples))
print("Data Matrix shape:")
print(data_matrix.shape)
labels = np.vstack((Normal_labels, Fault1_labels, Fault2_labels, Fault3_labels))
print("labels Matrix:")
#print(labels.shape)
main_matrix = np.hstack((data_matrix, labels))
print("data & label matrix:")
print(main_matrix)
```

نتیجه به‌صورت زیر است:

```
Data Matrix shape:
(400, 200)
labels Matrix:
data & label matrix:
[[-0.15583569 -0.18483323 -0.15959077 ... -0.07468431 -0.06133292
   0.        ]
 [-0.15583569 -0.18483323 -0.15959077 ... -0.07468431 -0.06133292
   0.        ]
 [-0.15583569 -0.18483323 -0.15959077 ... -0.07468431 -0.06133292
   0.        ]
 ...
 [-0.01502525  0.28629192 -0.05360359 ...  0.62781178 -0.24974401
   3.        ]
 [-0.01502525  0.28629192 -0.05360359 ...  0.62781178 -0.24974401
   3.        ]
 [-0.01502525  0.28629192 -0.05360359 ...  0.62781178 -0.24974401
   3.        ]]
```

ب-۱-۲

در اینجا ویژگی های زیر از مجموعه داده استخراج می‌شوند:

mean, Standard Deviation, Peak, Root Mean Square, Crest Factor, Peak to Peak,

Absolute Mean, Impulse Factor

سپس با دستور pd.DataFrame دیتای جدید با ویژگی های استخراج شده را نمایش می‌دهیم:

```python
#create features
mean_values = np.mean(data_matrix, axis=1)
std_dev_values = np.std(data_matrix, axis=1)
peak_values = np.max(np.abs(data_matrix), axis=1)
rms_value = np.sqrt(np.mean(data_matrix**2, axis=1))
crest_factor = peak_values / rms_value
peak_to_peak_value=np.max(data_matrix, axis=1)-np.min(data_matrix, axis=1)
Abs_Mean_value=np.mean(np.abs(data_matrix), axis=1)
Impulse_Factor=peak_values/Abs_Mean_value
# Concatenate mean and standard deviation as features
features = np.column_stack((mean_values, std_dev_values, peak_values, rms_value, crest_factor, peak_to_peak_value, Abs_Mean_value, Impulse_Factor))
# Create DataFrame
Data = pd.DataFrame(features, columns=['Mean', 'Standard Deviation','Peak', 'RMS', 'Crest Factor', 'Peak to Peak', 'Absolute Mean', 'Impulse Factor'])
print(Data)
```

دیتاست با ویژگی های استخراج شده به‌صورت زیر است:

	Mean	Standard Deviation	Peak	RMS	Crest Factor	Peak to Peak	Absolute Mean	Impulse Factor
0	0.016739	0.086732	0.202148	0.088333	2.288487	0.386982	0.075379	2.681759
1	0.016739	0.086732	0.202148	0.088333	2.288487	0.386982	0.075379	2.681759
2	0.016739	0.086732	0.202148	0.088333	2.288487	0.386982	0.075379	2.681759
3	0.016739	0.086732	0.202148	0.088333	2.288487	0.386982	0.075379	2.681759
4	0.016739	0.086732	0.202148	0.088333	2.288487	0.386982	0.075379	2.681759
..	...	...	...	...	...	...	...	...
395	0.025226	0.637517	2.586373	0.638016	4.053774	5.110615	0.406912	6.356097
396	0.025226	0.637517	2.586373	0.638016	4.053774	5.110615	0.406912	6.356097
397	0.025226	0.637517	2.586373	0.638016	4.053774	5.110615	0.406912	6.356097
398	0.025226	0.637517	2.586373	0.638016	4.053774	5.110615	0.406912	6.356097
399	0.025226	0.637517	2.586373	0.638016	4.053774	5.110615	0.406912	6.356097

[400 rows x 8 columns]

ج-۱-۲

در اینجا ماتریس شامل دیتا همراه با برچسب را با دستور np.random.shuffle مخلوط می‌کنیم. برای تکرار پذیری نتایج از random.seed(64) استفاده شد که باعث می‌شود نتایج آموزش و ارزیابی مدل در هر اجرا یکسان باشد. پس از مخلوط کردن داده‌ها، تقسیم آنها به دسته هایآموزش و آزمون واعتبارسنجی با دستور()train_test_split انجام می‌شود.

مدل روی داده های اعتبارسنجی امتحان می‌شود و دقت مدل محاسبه می شود. تا یه جایی با کم شدن خطای مدل روی داده های آموزش، خطا روی داده های اعتبارسنجی هم پایین می آید. اما از جایی به بعد که مدل داده های آموزش را حفظ می کند اصطلاحا Overfit رخ داده است، خطای مدل روی داده های اعتبارسنجی به جای کم شدن، بیشتر میشود. در این نقطه است که آموزش باید متوقف شود.

از داده های آزمون به عنوان معیاری برای مقایسه عملکرد مدل های مختلف که روی دیتاست واحد آموزش دیده شده اند، استفاده می‌شود. (لفظ داده اعتبار سنجی و داده آزمون ممکن است به‌جای یکدیگر بکاررود و داده اعتبارسنجی آن دسته از داده‌هایی باشد که کارفرما پیش خود نگه داشته باشد و مدل ما هیچ‌گاه این داده ها را ندیده باشد و از آن برای تعیین کارایی مدل ارائه شده استفاده شود.)

ابتدا داده‌ها را با نسبت ۸۵٪ و ۱۵٪ به دسته های آموزش و آزمون تقسیم می کنیم. سپس داده های آموزش را با همان نسبت قبل به دسته های آموزش و اعتبارسنجی تقسیم می‌کنیم.

```
Labeled_Data=np.hstack((Data, labels))
np.random.seed(64)
np.random.shuffle(Labeled_Data)
print(Labeled_Data)
# Remove the header from the variables
data = Data.values
label = labels
# Split the data into train/validation/test sets
train_data, test_data, train_label, test_label = train_test_split(data, label, test_size=0.15, random_state=64)
train_data, val_data, train_label, val_label = train_test_split(train_data, train_label, test_size=0.15, random_state=64)
# Print shape of each set
print(f"train_data shape: {train_data.shape}")
print(f"val_data shape: {val_data.shape}")
print(f"test_data shape: {test_data.shape}")
print(f"train_label shape: {train_label.shape}")
print(f"val_label shape: {val_label.shape}")
print(f"test_label shape: {test_label.shape}")
```

نتیجه به‌صورت زیر است:

```
[[0.02522618 0.63751722 2.58637335 ... 0.40691218 6.35609712 3.        ]
 [0.0167393  0.08673215 0.20214831 ... 0.075379   2.68175906 0.        ]
 [0.01372739 0.26633675 0.90297689 ... 0.19650265 4.59524026 1.        ]
 ...
 [0.0167393  0.08673215 0.20214831 ... 0.075379   2.68175906 0.        ]
 [0.01372739 0.26633675 0.90297689 ... 0.19650265 4.59524026 1.        ]
 [0.01372739 0.26633675 0.90297689 ... 0.19650265 4.59524026 1.        ]]
train_data shape: (289, 8)
val_data shape: (51, 8)
test_data shape: (60, 8)
train_label shape: (289, 1)
val_label shape: (51, 1)
test_label shape: (60, 1)
```

۲-۱-د

دو روش رایج برای نرمال سازی داده‌ها که در مینی پروژه اول بیان شد، عبارت بودند از:

1. MinMaxScaler

همه داده ها را به بازه [0, 1] تبدیل می‌کند. نرمال سازی حداقل_حداکثر در پایتون با استفاده از کتابخانهfrom sklearn.preprocessing و MinMaxScaler امکان‌پذیر است. این کار با تعریف MinMaxScaler() و با دستور scaler.fit_transform(x) انجام می‌شود.

2. StandardScaler

نرمال سازی استاندارد دیتا را به یک توزیع نرمال استاندارد با میانگین صفر و واریانس یک تبدیل می‌کند.

نرمال سازی استاندارد در پایتون با استفاده از کتابخانهfrom sklearn.preprocessing و StandardScaler امکان‌پذیر است. این کار با تعریف StandardScaler () و با دستور scaler.fit_transform(x) انجام می‌شود.

در این سوال از نرمال‌سازی حداقل_حداکثر استفاده شده است:

در این عملیات نباید از داده های آزمون استفاده شود، چراکه باعث نشت اطلاعات از داده های آزمون به مدل یادگیری می‌شود و مدل، تعمیم پذیری و عملکرد خوبی نخواهد داشت. بنابراین ابتدا scaler را روی داده های آموزش فیت کرده و سپس از آن برای مقیاس بندی داده‌های آزمون و اعتبارسنجی استفاده می‌کنیم.

```python
# Scale the data using MinMaxScaler
scaler = MinMaxScaler()
train_data = scaler.fit_transform(train_data)
val_data = scaler.transform(val_data)
test_data = scaler.transform(test_data)
```

۲-۲

یک مدل MLP ساده با دو لایه پنهان یا بیشتر بسازید. بخشی از داده های آموزش را برای اعتبارسنجی کنار بگذارید و با انتخاب بهینه ساز و تابع اتلاف مناسب، مدل را آموزش دهید .نمودارهای اتلاف و Accuracy مربوط به آموزش و اعتبارسنجی را رسم و نتیجه را تحلیل کنید. نتیجهٔ تست مدل روی داده های آزمون را با استفاده ماتریس درهم ریختگی و classification_report نشان داده و نتایج به صورت دقیق تحلیل کنید.

در این قسمت مجموعه دستوراتی شامل تعریف مدل شبکه عصبی با دو لایه می‌نویسیم. مدل شامل دو لایه خطی است که با استفاده از تابع فعالساز غیرخطی ReLU بهم متصل شده‌اند. از ماژول kerasکتابخانه tensorflow استفاده می‌کنیم. در کار طبقه بندی با شبکه عصبی، ابتدا برچسب های دیتاست را با استفاده از تابعto_categorical به‌گونه ای نمایش می دهیم که برچسب های هر کلاس ستون مجزایی داشته باشند:

```
from keras.utils import to_categorical
x_train = train_data
y_train = train_label.ravel()
x_test = test_data
y_test = test_label.ravel()
y_train = to_categorical(y_train, num_classes=4)
y_test = to_categorical(y_test, num_classes=4)
print(x_train.shape, y_train.shape)
```

```
(289, 8) (289, 4)
```

حال به تعریف MLP سه لایه با دو لایه پنهان می‌پردازیم. در این دو لایه از تابع فعال ساز ReLU استفاده می‌شود. در لایه خروجی از تابع فعال‌ساز softmax استفاده می‌کنیم. زیرا در لایه آخر به دلیل داشتن ۴ کلاس، باید از ۴ خروجی استفاده کرد. تابع softmax را می‌توان به شکل ترکیبی از چندین تابع سیگموید در نظر گرفت که در نهایت احتمال مرتبط با هر خروجی را محاسبه می‌کند. لایه های پنهان به ترتیب ۹و ۵ نورون دارند و همانطور که گفته شد برای لایه خروجی ۴ نورون در نظر گرفته می‌شود.

```
import tensorflow as tf
from tensorflow import keras
from keras import preprocessing
from keras.models import Sequential
from keras.layers import Dense
keras.utils.set_random_seed(64)
model = Sequential()
# Add the first hidden layer with 8 neurons and relu activation function
model.add(Dense(9, activation='relu', input_shape=(x_train.shape[1],)))
# Add the second hidden layer with 4 neurons and relu activation function
model.add(Dense(5, activation='relu'))
# Add an output layer with 4 neuron and softmax activation function
model.add(Dense(4, activation='softmax'))
model.summary()
```

تعداد لایه ها و پارامتر های مدل در زیر قابل مشاهده است:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 9)                 81

 dense_1 (Dense)             (None, 5)                 50

 dense_2 (Dense)             (None, 4)                 24

=================================================================
Total params: 155 (620.00 Byte)
Trainable params: 155 (620.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

برای آموزش مدل از تابع بهینه ساز adam با نرخ یادگیری اولیه 0.05 استفاده می‌کنیم. این الگوریتم یک الگوریتم بهینه سازی انباشت گرادیان و نسخه تعمیم یافته SGD است و سرعت همگرایی بالایی دارد. از مزیت های این بهینه ساز می‌توان به سادگی محاسبات، عدم نیاز به حافظه زیاد و نیاز به تنظیم پارامتر اندک اشاره کرد. تابع اتلاف را **BinaryCrossentropy** قرار می‌دهیم که در عملیات طبقه بندی کارآمد است. این تابع اتلاف در مینی پروژه اول مورد بررسی قرار گرفت. همچنین در مسائل طبقه بندی به‌جای استفاده از مقیاس R^2Score از **Accuracy** استفاده می شود. به‌منظور جلوگیری از overfit از ماژول**callbacks** و وارد کردن توابع EarlyStopping، ReduceLROnPlateau استفاده می‌کنیم. تابع EarlyStopping با مانیتور کردن اتلاف داده های اعتبارسنجی، تعداد ایپاکی را صبر می‌کند و درصورت عدم بهبود عملکرد مدل روی داده های اعتبار سنجی، فرآیند آموزش را متوقف می‌کند. تابع ReduceLROnPlateau با مانیتور کردن اتلاف داده های اعتبارسنجی، تعداد ایپاکی(patience) را صبر می‌کند و درصورت عدم بهبود عملکرد، نرخ یادگیری را با مقیاس مشخص factor کاهش می‌دهد(کاهش نرخ یادگیری تا آستانه تعیین شده‌min_lr صورت می‌گیرد.) تا عملکرد مدل را بهبود دهد. همچنین با استفاده از تابع ModelCheckpoint بهترین مدل با در نظر گرفتن شرایط تابع اتلاف، ذخیره می‌شود.

سپس متد فیت با در نظر گرفتن موارد بکار گرفته شده از ماژول callbacks روی داده های آموزش با ۱۰۰ ایپاک و batch_size=50 و استفاده از ۱۰ درصد داده ها برای اعتبار سنجی پیاده‌سازی می‌شود.

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
keras.utils.set_random_seed(64)
initial_lr = 0.05
optimizer = Adam(learning_rate=initial_lr)
model.compile(optimizer=optimizer,
              loss='BinaryCrossentropy',
              metrics=['accuracy'])
np.random_state=64
np.random.seed(64)
early_stop = EarlyStopping(monitor='val_loss', patience=6, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=0.001)
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)

history = model.fit(x_train, y_train, validation_split=0.1, epochs=100, batch_size=50,
                    callbacks=[reduce_lr, early_stop, checkpoint])
```

بخشی از نتایج به‌صورت زیر است:

```
Epoch 1/100
6/6 [==============================] - 1s 56ms/step - loss: 0.6004 - accuracy: 0.3346 - val_loss: 0.5180 - val_accuracy: 0.3793 - lr: 0.0500
Epoch 2/100
6/6 [==============================] - 0s 12ms/step - loss: 0.4546 - accuracy: 0.3500 - val_loss: 0.4178 - val_accuracy: 0.4828 - lr: 0.0500
Epoch 3/100
1/6 [====>.........................] - ETA: 0s - loss: 0.4066 - accuracy: 0.4600/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.
  saving_api.save_model(
6/6 [==============================] - 0s 31ms/step - loss: 0.3827 - accuracy: 0.5077 - val_loss: 0.3616 - val_accuracy: 0.4828 - lr: 0.0500
Epoch 4/100
6/6 [==============================] - 0s 35ms/step - loss: 0.3324 - accuracy: 0.5923 - val_loss: 0.2853 - val_accuracy: 0.8621 - lr: 0.0500
Epoch 5/100
6/6 [==============================] - 0s 37ms/step - loss: 0.2757 - accuracy: 0.7423 - val_loss: 0.2583 - val_accuracy: 0.8621 - lr: 0.0500
Epoch 6/100
6/6 [==============================] - 0s 45ms/step - loss: 0.2500 - accuracy: 0.7192 - val_loss: 0.2417 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 7/100
6/6 [==============================] - 0s 30ms/step - loss: 0.2023 - accuracy: 0.7731 - val_loss: 0.1817 - val_accuracy: 0.8621 - lr: 0.0500
Epoch 8/100
6/6 [==============================] - 0s 47ms/step - loss: 0.1808 - accuracy: 0.7500 - val_loss: 0.1917 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 9/100
6/6 [==============================] - 0s 29ms/step - loss: 0.1713 - accuracy: 0.7654 - val_loss: 0.1846 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 10/100
6/6 [==============================] - 0s 28ms/step - loss: 0.1637 - accuracy: 0.8500 - val_loss: 0.1529 - val_accuracy: 0.8621 - lr: 0.0500
Epoch 11/100
6/6 [==============================] - 0s 22ms/step - loss: 0.1602 - accuracy: 0.8308 - val_loss: 0.1899 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 12/100
6/6 [==============================] - 0s 30ms/step - loss: 0.1485 - accuracy: 0.8231 - val_loss: 0.1410 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 13/100
6/6 [==============================] - 0s 31ms/step - loss: 0.1349 - accuracy: 1.0000 - val_loss: 0.1284 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 14/100
6/6 [==============================] - 0s 37ms/step - loss: 0.1197 - accuracy: 1.0000 - val_loss: 0.1115 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 15/100
6/6 [==============================] - 0s 35ms/step - loss: 0.0913 - accuracy: 1.0000 - val_loss: 0.0821 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 16/100
6/6 [==============================] - 0s 25ms/step - loss: 0.0669 - accuracy: 1.0000 - val_loss: 0.0566 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 17/100
6/6 [==============================] - 0s 21ms/step - loss: 0.0407 - accuracy: 1.0000 - val_loss: 0.0298 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 18/100
6/6 [==============================] - 0s 33ms/step - loss: 0.0224 - accuracy: 1.0000 - val_loss: 0.0177 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 19/100
6/6 [==============================] - 0s 29ms/step - loss: 0.0123 - accuracy: 1.0000 - val_loss: 0.0099 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 20/100
6/6 [==============================] - 0s 37ms/step - loss: 0.0073 - accuracy: 1.0000 - val_loss: 0.0064 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 21/100
6/6 [==============================] - 0s 32ms/step - loss: 0.0049 - accuracy: 1.0000 - val_loss: 0.0045 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 22/100
6/6 [==============================] - 0s 34ms/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 0.0034 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 23/100
6/6 [==============================] - 0s 33ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.0027 - val_accuracy: 1.0000 - lr: 0.0500
Epoch 24/100
6/6 [==============================] - 0s 39ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.0021 - val_accuracy: 1.0000 - lr: 0.0500
```

حال دقت مدل را در داده های آزمون نمایش می‌دهیم:

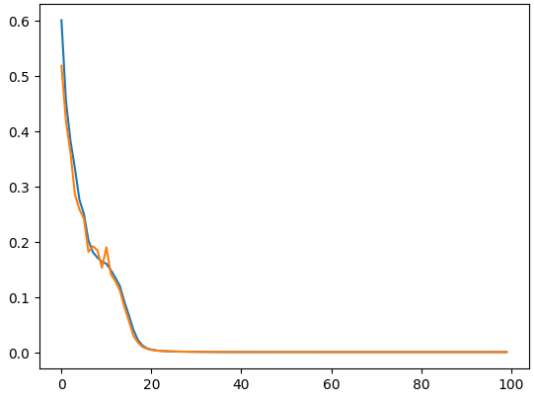
```
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f"Model accuracy on test data: {accuracy:.4f}")

Model accuracy on test data: 1.0000
```

همانطور که پیداست دقت مدل ۱۰۰ درصد گزارش شده‌است.

میزان اتلاف در این داده ها به‌صورت زیر است که مقداری معادل 4.3486e-04 دارد.

```
#Evaluate the model
loss = model.evaluate(x_test , y_test)

2/2 [==============================] - 0s 7ms/step - loss: 4.3486e-04 - accuracy: 1.0000
```

نمودار تابع اتلاف در هر ایپاک برای داده های آموزش و اعتبار سنجی به‌صورت زیر است:

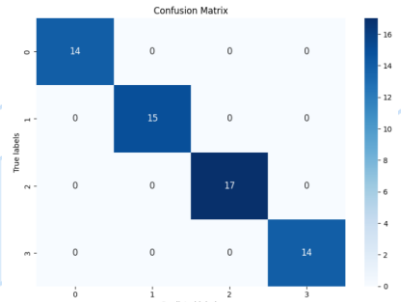


همانطور که از نمودار بالا پیداست مدل عملکرد خوبی در داده های آموزش و اعتبارسنجی داشته و همگرایی رخ داده‌است.

طراحی این مدل با کتابخانه sklearn و ماژول MLPClassifier نیز در دفترچه گوگل کولب این سوال قرار داده شده است.

```python
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

x_train = train_data
y_train = train_label.ravel()
x_test = test_data
y_test = test_label.ravel()
# Assuming you've trained your model already
model = MLPClassifier(hidden_layer_sizes=(9, 5), activation='relu', solver='adam',
                      batch_size=50, learning_rate_init=0.05,
                       max_iter=100, random_state=64,
                      early_stopping=True, validation_fraction=0.1)
model.fit(x_train, y_train)

# Making predictions on the test set
y_pred = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_ylim(len(np.unique(y_test)), 0)  # Fix for matplotlib 3.1.1 and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

نتیجه به‌صورت زیر است:




```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        14
         1.0       1.00      1.00      1.00        15
         2.0       1.00      1.00      1.00        17
         3.0       1.00      1.00      1.00        14

    accuracy                           1.00        60
   macro avg       1.00      1.00      1.00        60
weighted avg       1.00      1.00      1.00        60
```

همانطور که از ماتریس در هم ریختگی و جدول پایین آن پیداست در اینجا نیز همه داده ها به درستی طبقه بندی شده اند و دقت مدل ۱۰۰ درصد است.

۲-۳

فرآیند سوال قبل را با یک بهینه ساز و تابع اتلاف جدید انجام داده و نتایج را مقایسه و تحلیل کنید. بررسی کنید که آیا تغییر تابع اتلاف می تواند در نتیجه اثرگذار باشد؟

در اینجا از بهینه ساز Adagrad و تابع اتلاف Categorical Cross-Entropy که در کتابخانه keras موجود است، استفاده می‌کنیم.

الگوریتم بهینه سازی Adagrad برای بهبود یادگیری، نرخ یادگیری را برای هر پارامتر به‌طور مستقل از سایر پارامتر ها محاسبه می‌کند. یعنی نرخ یادگیری برای هر پارامتر متفاوت است. اگر گرادیان یک پارامتر در گام های قبلی بیشتر بوده باشد، نرخ یادگیری برای آن پارامتر در گام بعد کاهش می‌یابد و برعکس.

تابع Categorical Cross-Entropy با استفاده از رابطه زیر مقدار اتلاف را محاسبه می‌کند:

$$loss = -\sum_{i=1}^{c} y_i log\hat{y}_i$$

C تعداد کلاس در مسئله طبقه بندی و y_i و $\hat{y}_i$ به ترتیب مقدار واقعی و مقدار پیش‌بینی شده است.

برای مقایسه تاثیر بهینه ساز و تابع اتلاف جدید مراحل قسمت قبل را با همان نرخ یادگیری و تعداد ایپاک تکرار و نتایج را گزارش می‌کنیم.(برای مقایسه درست ابتدا تابع اتلاف قسمت قبل را حفظ و بهینه ساز را تغییر داده و نتیجه مشاهده شد و سپس بهینه ساز قبلی را حفظ و تابع اتلاف زا تغییر دادیم. در هر دو مورد عملکرد مدل نسبت به قسمت قبل ضعیف و دقت مدل به 71.67% کاهش یافت.کد مربوط به این قسمت در دفترچه گوگل کولب این سوال موجود است.) در اینجا به گزارش مدل با بهینه ساز و تابع اتلاف جدید بسنده می‌کنیم.

ابتدا مانند قسمت قبل، مدل MLP با تعداد لایه و نورون ذکر شده را تعریف می‌کنیم:

```
import tensorflow as tf
from tensorflow import keras
from keras import preprocessing
from keras.models import Sequential
from keras.layers import Dense
keras.utils.set_random_seed(64)
np.random_state=64
np.random.seed(64)
from keras.utils import to_categorical
x_train = train_data
y_train = train_label.ravel()
x_test = test_data
y_test = test_label.ravel()
y_train = to_categorical(y_train, num_classes=4)
y_test = to_categorical(y_test, num_classes=4)

model = Sequential()
# Add the first hidden layer with 8 neurons and relu activation function
model.add(Dense(9, activation='relu', input_shape=(x_train.shape[1],)))
# Add the second hidden layer with 4 neurons and relu activation function
model.add(Dense(5, activation='relu'))
# Add an output layer with 4 neuron and softmax activation function
model.add(Dense(4, activation='softmax'))
model.summary()
```

سپس با تغییر optimizer از Adam به Adagrad و تغییر loss از BinaryCrossentropy به CategoricalCrossentropy مدل را آموزش می دهیم:

```
from tensorflow.keras.optimizers import Adagrad
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
keras.utils.set_random_seed(64)
initial_lr = 0.05
optimizer = Adagrad(learning_rate=initial_lr)
model.compile(optimizer=optimizer,
              loss='CategoricalCrossentropy',
              metrics=['accuracy'])
np.random_state=64
np.random.seed(64)
early_stop = EarlyStopping(monitor='val_loss', patience=6, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=0.001)
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)

history = model.fit(x_train, y_train, validation_split=0.1, epochs=100, batch_size=50,
                    callbacks=[reduce_lr, early_stop, checkpoint])
```

بخشی از نتایج به‌صورت زیر است:

```
Epoch 1/100
6/6 [==============================] - 1s 84ms/step - loss: 1.3208 - accuracy: 0.4423 - val_loss: 1.2799 - val_accuracy: 0.2414 - lr: 0.0500
Epoch 2/100
6/6 [==============================] - 0s 23ms/step - loss: 1.2525 - accuracy: 0.2500 - val_loss: 1.2262 - val_accuracy: 0.2414 - lr: 0.0500
Epoch 3/100
1/6 [====>.........................] - ETA: 0s - loss: 1.2213 - accuracy: 0.2400/usr/local/lib/python3.10/dist-packages/keras/src/engine/training
  saving_api.save_model(
6/6 [==============================] - 0s 21ms/step - loss: 1.1936 - accuracy: 0.2500 - val_loss: 1.1668 - val_accuracy: 0.2414 - lr: 0.0500
Epoch 4/100
6/6 [==============================] - 0s 25ms/step - loss: 1.1414 - accuracy: 0.2500 - val_loss: 1.1234 - val_accuracy: 0.2414 - lr: 0.0500
Epoch 5/100
6/6 [==============================] - 0s 23ms/step - loss: 1.0982 - accuracy: 0.2500 - val_loss: 1.0909 - val_accuracy: 0.2414 - lr: 0.0500
Epoch 6/100
6/6 [==============================] - 0s 21ms/step - loss: 1.0664 - accuracy: 0.2500 - val_loss: 1.0580 - val_accuracy: 0.2414 - lr: 0.0500
Epoch 7/100
6/6 [==============================] - 0s 20ms/step - loss: 1.0392 - accuracy: 0.2500 - val_loss: 1.0368 - val_accuracy: 0.2414 - lr: 0.0500
Epoch 8/100
6/6 [==============================] - 0s 21ms/step - loss: 1.0171 - accuracy: 0.2500 - val_loss: 1.0175 - val_accuracy: 0.2414 - lr: 0.0500
Epoch 9/100
6/6 [==============================] - 0s 24ms/step - loss: 0.9963 - accuracy: 0.2923 - val_loss: 0.9984 - val_accuracy: 0.2414 - lr: 0.0500
Epoch 10/100
6/6 [==============================] - 0s 20ms/step - loss: 0.9793 - accuracy: 0.2500 - val_loss: 0.9843 - val_accuracy: 0.3793 - lr: 0.0500
Epoch 11/100
6/6 [==============================] - 0s 21ms/step - loss: 0.9645 - accuracy: 0.5346 - val_loss: 0.9711 - val_accuracy: 0.3793 - lr: 0.0500
Epoch 12/100
6/6 [==============================] - 0s 19ms/step - loss: 0.9507 - accuracy: 0.5654 - val_loss: 0.9595 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 13/100
6/6 [==============================] - 0s 24ms/step - loss: 0.9363 - accuracy: 0.7654 - val_loss: 0.9466 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 14/100
6/6 [==============================] - 0s 23ms/step - loss: 0.9239 - accuracy: 0.7654 - val_loss: 0.9343 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 15/100
6/6 [==============================] - 0s 20ms/step - loss: 0.9119 - accuracy: 0.7654 - val_loss: 0.9225 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 16/100
6/6 [==============================] - 0s 22ms/step - loss: 0.8994 - accuracy: 0.7654 - val_loss: 0.9092 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 17/100
6/6 [==============================] - 0s 18ms/step - loss: 0.8871 - accuracy: 0.7654 - val_loss: 0.8963 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 18/100
6/6 [==============================] - 0s 22ms/step - loss: 0.8743 - accuracy: 0.7654 - val_loss: 0.8852 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 19/100
6/6 [==============================] - 0s 19ms/step - loss: 0.8625 - accuracy: 0.7654 - val_loss: 0.8740 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 20/100
6/6 [==============================] - 0s 20ms/step - loss: 0.8511 - accuracy: 0.7654 - val_loss: 0.8618 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 21/100
6/6 [==============================] - 0s 22ms/step - loss: 0.8394 - accuracy: 0.7654 - val_loss: 0.8509 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 22/100
6/6 [==============================] - 0s 23ms/step - loss: 0.8278 - accuracy: 0.7654 - val_loss: 0.8400 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 23/100
6/6 [==============================] - 0s 21ms/step - loss: 0.8158 - accuracy: 0.7654 - val_loss: 0.8290 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 24/100
6/6 [==============================] - 0s 22ms/step - loss: 0.8050 - accuracy: 0.7654 - val_loss: 0.8160 - val_accuracy: 0.6207 - lr: 0.0500
Epoch 25/100
6/6 [==============================] - 0s 11ms/step - loss: 0.7939 - accuracy: 0.7654 - val_loss: 0.8086 - val_accuracy: 0.6207 - lr: 0.0500
```

حال دقت مدل و مقدار تابع اتلاف را در داده های آزمون نمایش می‌دهیم:

```
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f"Model accuracy on test data: {accuracy:.4f}")
```

```
Model accuracy on test data: 0.7167
```

```
#Evaluate the model
loss = model.evaluate(x_test , y_test)
```

```
2/2 [==============================] - 0s 7ms/step - loss: 0.5582 - accuracy: 0.7167
```

همانطور که پیداست، دقت مدل 71.67% و میزات تابع اتلاف 0.5582 گزارش شده است.

جدول زیر مقایسه دو حالت بررسی شده در این قسمت و قسمت قبل را نشان می‌دهد:

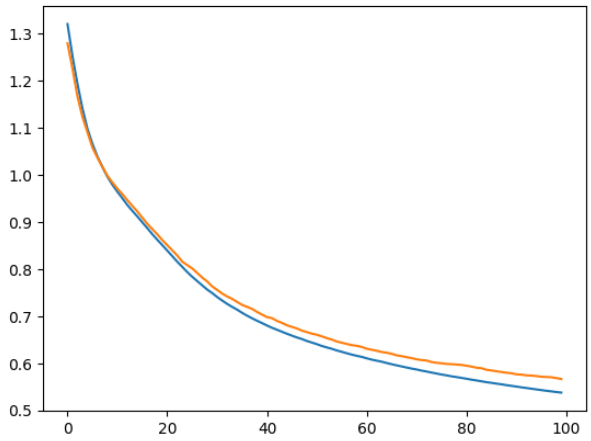
	Adam Optimizer & BinaryCrossentropy Loss Function	**Adagrad Optimizer & CategoricalCrossentropy Loss Function**
Accuracy	100%	71.67%
Loss value	4.3486e-04	0.5582

از مقایسه نتایج در می یابیم بهینه ساز Adam و تابع اتلاف BinaryCrossentropy عملکرد بهتری از بهینه سازAdagrad و تابع اتلاف CategoricalCrossentropy داشتند.

هر دو بهینه ساز الگوریتم های موثری در یادگیری ماشین هستند اما Adagrad در شرایطی که داده پراکنده و تغییرات گرادیان بزرگ است بهترین عملکرد را دارد. در مقابل زمانی که تابع هزینه یکنواخت و گرادیان تغییرات زیادی ندارد Adam عملکرد بهتری دارد.

تابع اتلافBinaryCrossentropy دسته بندی هر کلاس را به طور مستقل انجام می دهد و CategoricalCrossentropy کل مجموعه کلاس ها را به طور همزمان در نظر می گیرد.

نمودار تابع اتلاف در هر ایپاک برای داده های آموزش و اعتبار سنجی به‌صورت زیر است:



همانطور که پیداست در ۱۰۰ ایپاک مدل جدد نتوانسته است به خوبی همگرا شود و به تعداد ایپاک بیشتری برای همگرایی نیاز است.

۲-۴

در مورد K-Fold Cross-validation و Stratified K-Fold Cross-validation و مزایای هریک توضیح دهید. سپس با ذکر دلیل، یکی از این روش ها را انتخاب کرده و بخش۲ این سوال را با آن پیاده سازی کنید و نتایج خود را تحلیل کنید.

به‌طور کلی Cross-validation تکنیکی برای ارزیابی یک مدل یادگیری ماشین و تست عملکرد آن است. این کار به مقایسه و انتخاب یک مدل مناسب و رسیدن به بهترین عملکرد مدل و جلوگیری از overfit کمک می‌کند .

الگوریتم k-Fold روشی برای تقسیم مجموعه داده معرفی می کند که در حل مشکل "تست یکباره" کمک رسان است. در این الگوریتم داده ها به چند حالت می توانند تقسیم شوند. در هر حالت داده را به چند بخش تقسیم می کنیم. یک بخش از آنها به عنوان داده آزمون و سایر بخش ها به عنوان داده آموزش انتخاب می‌شوند.

الگوریتم k-Fold:

۱) تعداد k دسته انتخاب می‌شود . معمولاً k ۵ یا ۱۰است اما می‌تواند هر عددی کمتر از طول مجموعه داده باشد .

۲) مجموعه داده به k قسمت(fold) مساوی (در صورت امکان) تقسیم می‌شود.

۳) k – 1 folds به‌عنوان مجموعه آموزشی انتخاب می‌شود . foldهای باقیمانده مجموعه آزمون خواهند بود .

۴) مدل روی مجموعه آموزشی فیت می‌شود. در هر تکرار از Cross-Validation، باید یک مدل جدید مستقل از مدل آموزش داده شده در تکرار قبلی آموزش داده شود .

۵) در مجموعه آزمایشی تست انجام می‌شود.

۶) مراحل ۳ تا ۵ را k بار تکرار می‌کنیم . هر بار از fold باقی مانده به عنوان مجموعه تست استفاده می‌شود. در پایان، باید مدل روی هر fold تست شده باشد.

۷) برای به دست آوردن نتیجه نهایی ، میانگین تمام نتایج بدست آمده محاسبه و گزارش می‌شود .

از آنجایی که آموزش و آزمایش به‌صورت دقیق، بر روی چندین بخش مختلف مجموعه داده انجام می شود، مقایسه k-Fold نتیجه پایدارتر و قابل اعتمادتری به دست می‌دهد . اگر تعداد fold ها را برای آزمایش مدل بر روی بسیاری از زیر مجموعه‌های مختلف افزایش دهیم، می‌توانیم نتیجه نهایی را بهبود

بخشیم. با این حال، نقطه ضعف روش k-Fold پر هزینه بودن فرآیند در صورت افزایش k که منجر به آموزش مدل های بیشتر می شود است.

الگوریتم Stratified K-Fold زمانی که با عدم تعادل زیادی در target مجموعه داده مواجه هستیم، موثر است. این الگریتم، مجموعه داده را به k دسته تقسیم می کند به طوری که هر دسته دارای درصد مشابهی از نمونه های هر کلاس هدف به عنوان مجموعه کامل است.

الگوریتم Stratified K-Fold Cross-validation:

۱- تعدادی k-fold انتخاب می‌شود .

۲- مجموعه داده را به k دسته تقسیم می‌کنیم. هر fold باید دارای درصد مشابهی از نمونه‌های هر کلاس هدف در مجموعه اصلی باشد .

۳- k–1 folds را به عنوان مجموعه آموزشی انتخاب می‌کنیم. fold باقیمانده مجموعه آزمون خواهد بود.

۴- مدل را روی مجموعه آموزشی فیت می‌کنیم. در هر تکرار یک مدل جدید باید آموزش داده شود.

۵- در مجموعه آزمایشی تست انجام می‌شود.

۶- مراحل ۳ تا ۵ را k بار تکرار می‌کنیم . هر بار از fold باقی مانده به عنوان مجموعه تست استفاده می‌شود. در پایان، باید مدل روی هر fold تست شده باشد.

۷) برای به دست آوردن نتیجه نهایی ، میانگین تمام نتایج بدست آمده محاسبه و گزارش می‌شود .

تنها تفاوت دو الگوریتم در مرحله دوم است.

برای تشخیص اینکه از کدام‌یک از الگوریتم های بالا استفاده کنیم، تعداد لیبل های هر کلاس را در مجموعه داده بررسی می‌کنیم:

```
x = Data.values
y = labels
unique, counts = np.unique(y, return_counts=True)
print("Class counts:", dict(zip(unique, counts)))
```

```
Class counts before balancing: {0.0: 100, 1.0: 100, 2.0: 100, 3.0: 100}
```

همانطور که پیداست، دیتاست دارای تعادل است و تعداد برابری از هر لیبل در دیتاست وجود دارد.
بنابراین از الگوریتمK-Fold Cross-validation استفاده می‌کنیم.

برای پیاده سازی الگوریتم از کتابخانه sklearn و ماژول های KFold و cross_val_score استفاده می‌کنیم. مدل را مانند قسمت دوم تعریف کرده و داده ها را با استفاده از تابع KFold و تعریف 5 fold به داده های آموزش و آزمون تقسیم می‌کنیم. سپس مدل را با داده های آموزش فیت می‌کنیم. در انتها دقت مدل را با داده‌های آزمون می‌سنجیم و در هر بار آموزش مدل با داده های آموزش در هر حالت،دقت را ذخیره و میانگین دقت های محاسبه شده را گزارش می‌کنیم.

```
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
# Set random seed for reproducibility
keras.utils.set_random_seed(64)
np.random.seed(64)
np.random_state=64

#  model definition
def create_model():
    model = Sequential()
    model.add(Dense(9, activation='relu', input_shape=(x.shape[1],)))
    model.add(Dense(5, activation='relu'))  # Second hidden layer
    model.add(Dense(4, activation='softmax'))  # Output layer with 4 classes
    initial_lr = 0.05
    optimizer = Adam(learning_rate=initial_lr)
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

kf = KFold(n_splits=5)
Acc = []
for train_index, test_index in kf.split(x):
    X_train_kf, X_test_kf = x[train_index], x[test_index]
    y_train_kf, y_test_kf = y[train_index], y[test_index]

    # Create a new model instance for each fold
    model = create_model()
    # Train the model
    model.fit(X_train_kf, y_train_kf, epochs=100, verbose=0, validation_split=0.1, batch_size=50)
    # Predict on the test fold
    y_pred = model.predict(X_test_kf)
    # Convert predictions from probabilities to class labels
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_test_kf_classes = np.argmax(y_test_kf, axis=1)
    # Compute accuracy and store it
    Acc.append(accuracy_score(y_test_kf_classes, y_pred_classes))

Acc = np.array(Acc)
print(Acc)
print('mean Acc', Acc.mean())
```

نتیجه به‌صورت زیر است:

```
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
[1. 1. 1. 1. 0.]
mean Acc 0.8
```

بنابراین مدل بطور میانگین دقتی معادل ۸۰٪ دارد.

سوال سوم

دیتاست مربوط به دارو را در نظر بگیرید.

داده هایی در مورد مجموعه ای از بیماران جمع آوری شده‌است که همه آنها از یک بیماری رنج می بردند. در طول دوره درمان، هر بیمار به یکی از ۵ داروی A، B، C، x و y پاسخ داد. ویژگی های این دیتاست سن، جنسیت، فشار خون، کلسترول و میزان سدیم و پتاسیم بیماران است و هدف دارویی است که هر بیمار به آن پاسخ داده است.

۳-۱

با استفاده از بخشی از داده ها، مجموعه داده را به دو بخش آموزش و آزمون تقسیم کنید (حداقل ۱۵ درصد از داده ها را برای آزمون نگه دارید). توضیح دهید که از چه روشی برای انتخاب بخشی از داده ها استفاده کرده اید. آیا روش بهتری برای این کار می شناسید؟

در ادامه، برنامه ای بنویسید که درخت تصمیمی برای طبقه بندی کلاس های این مجموعه داده طراحی کند. خروجی درخت تصمیم خود را با برنامه نویسی و یا به صورت دستی تحلیل کنید.

ابتدا مجموعه داد را در گوگل درایو اپلود و با دستور gdown در محیط گوگل کولب فراخوانی می‌کنیم. دیتا را خوانده و در df ذخیره و آن را نمایش می‌دهیم:

```
import pandas as pd
df = pd.read_csv('/content/drug200.csv')
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    object
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    object
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

	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

در دیتاست ویژگی‌های Age و Na_to_K دارای طیف عددی هستند. برای استفاده از این ویژگی ها با تعیین یک آستانه آنها را به ۲ یا ۳ گروه دسته بندی می‌کنیم:

- برای ویژگی Age بازه بصورت زیر است:

اگر سن فرد کمتر یا مساوی ۴۰ باشد "جوان"، اگر بین ۴۰ تا ۶۵ باشد "بزرگسال" و اگر سنش برابر یا بالاتر از ۶۵ باشد "پیر" است.

- برای ویژگی Na_to_K بازه بصورت زیر است:

اگر داده برابر یا کمتر از ۱۵ باشد "Normal" و اگر بزرگتر از ۱۵ باشد"High" در نظر گرفته می‌شود.

```
data = pd.read_csv('drug200.csv')
age = data['Age']
#replacing labels
for i in range(200):
  if age[i] <= 40:
     age[i]= 'young'
  elif  40 < age[i] < 65:
    age[i] = 'adult'
  elif age[i] >= 65:
    age[i] = 'old'
#replacing in data
data['Age'] = age
```

```
data = pd.read_csv('drug200.csv')
na = data['Na_to_K']
# between 6.27 & 38.2
#replacing labels
for i in range(200):
  if na[i] <=15:
     na[i]= 'Normal'
  elif na[i] > 15:
     na[i] = 'High'
#replacing in data
data['Na_to_K'] = na
```

با اعمال این تغییرات، دیتاست به‌صورت زیر است:

```
print(data)
```

```
       Age Sex      BP Cholesterol Na_to_K   Drug
0    young   F    HIGH        HIGH    High  drugY
1    adult   M     LOW        HIGH  Normal  drugC
2    adult   M     LOW        HIGH  Normal  drugC
3    young   F  NORMAL        HIGH  Normal  drugX
4    adult   F     LOW        HIGH    High  drugY
..     ...  ..     ...         ...     ...    ...
195  adult   F     LOW        HIGH  Normal  drugC
196  young   M     LOW        HIGH  Normal  drugC
197  adult   M  NORMAL        HIGH  Normal  drugX
198  young   M  NORMAL      NORMAL  Normal  drugX
199  young   F     LOW      NORMAL  Normal  drugX

[200 rows x 6 columns]
```

ابتدا داده ها را با دستور () shuffle. مخلوط و سپس با دستور train_test_split با نسبت ۸۵٪ و ۱۵٪ آنها را به آموزش و آزمون تقسیم می‌کنیم:

```
np.random.seed(64)
# Convert DataFrame to a numpy array
array = df.values  #  numpy array
np.random.shuffle(array)  # Shuffle the array
shuffled_df = pd.DataFrame(array, columns=df.columns)
print(shuffled_df)
from sklearn.model_selection import train_test_split
train_data, test_data= train_test_split(data, test_size= 0.15, random_state=64)
# Print shape of each set
print(f"train_data shape: {train_data.shape}")
print(f"test_data shape: {test_data.shape}")
```

ابعاد داده به‌صورت روبرو است:

```
[200 rows x 6 columns]
train_data shape: (170, 6)
test_data shape: (30, 6)
```

تعداد کلاس‌ها در دیتاست به‌صورت زیر است:

```
labels = data['Drug']
len(labels), labels.unique(), labels.value_counts()
```

```
(200,
 array(['drugY', 'drugC', 'drugX', 'drugA', 'drugB'], dtype=object),
 Drug
 drugY    91
 drugX    54
 drugA    23
 drugC    16
 drugB    16
 Name: count, dtype: int64)
```

همانطور که پیداست، دیتاست بالانس نیست.

در اینجا برای تقسیم داده به منظور سادگی از روش بر زدن و تقسیم کردن استفاده شد. از روش های دیگر برای عملکرد بهتر مدل می توان به روش های cross validation که در سوال قبل از آن بهره گرفتیم خصوصا Stratified K-Fold به دلیل عدم تعادل target و همچنین روش های Bootstrapping اشاره کرد.

حال برای طراحی درخت تصمیم گام به گام مطابق کلاس تدریسیاری پیش می‌رویم. ابتدا تابع انتروپی را مطابق رابطه آن تعریف می‌کنیم. در این تابع p نسبت تعداد داده‌های یک کلاس خاص نسب به کل داده‌هاست.

Entropy

$$\text{Entropy}(Y) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

```
def entropy(labels):
    p = labels.value_counts() / len(labels)
    return -sum(p * np.log2(p))
```

در مرحله بعد تابع information gain مطابق رابطه زیر پیاده‌سازی می‌شود:

$$\text{Information Gain}(\text{Feature}) = \text{Entropy}(\text{Parent}) - \sum_{\text{value} \in \text{Feature}} \frac{|\text{Subset with value}|}{|\text{Parent}|} \times \text{Entropy}(\text{Subset with value})$$

```
def information_gain(data, feature, target):
    # Entropy of parent
    entropy_parent = entropy(data[target])

    # Entropy of child
    entropy_child = 0
    for value in data[feature].unique():
        subset = data[data[feature] == value]
        wi = len(subset) / len(data)
        entropy_child += wi * entropy(subset[target])

    return entropy_parent - entropy_child
```

در محاسبه InformationGain باید توجه داشت که این تابع تنها یک ویژگی را دریافت می‌کند. درواقع ابتدا Entropy کلی یک Feature را محاسبه می‌کند. سپس مجموع وزن دار entropy تک تک شاخه‌ها محاسبه و از انتروپی والد(کل دیتاست) کسر می‌شود. این کار را برای تمام ویژگی‌ها انجام شده و ویژگی با بیشترین IG انتخاب می‌شود.

```
data.iloc[:, :-1].columns
```

```
Index(['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K'], dtype='object')
```

```
[information_gain(data, feature, 'Drug') for feature in data.iloc[:, :-1].columns]
```

```
[0.7035715991901172,
 0.007703482714548349,
 0.6201266774024412,
 0.0931061958075754,
 0.994149171480893]
```

همانطور که پیداست IG در آخرین ویژگی یعنی Na_to_K بیشترین مقدار را دارد. برای دریافت این موضوع از دستور np.argmax برای معرفی بیشترین مقدار استفاده می‌کنیم.

```
np.argmax([information_gain(data, feature, 'Drug') for feature in data.iloc[:, :-1].columns])
```

```
4
```

پس ویژگی چهارم(در پایتون شمارش از صفر شروع می‌شود) که همانNa_to_K است، بیشترین مقدار را دارد و ویژگی برنده است و در خت تصمیم از این ویژگی شروع می‌شود.

حال کلاس گره را به گونه ای تعریف می کنیم که ویژگی و لیبل را دریافت می کند و زمانی که گره تصمیم گیری داریم، ویژگی به نام همان ویژگی است که گره بر پایه آن ساخته شده است و لیبل None می ماند. و زمانی که گره برگ داریم، ویژگی None می ماند و لیبل به کلاس مورد نظر اشاره می‌کند. در این کلاس متدی برای نمایش بهتر وضعیت درخت تصمیم در نظر گرفته می شود.

```
class Node:

    def __init__(self, feature=None, label=None):
        self.feature = feature
        self.label = label
        self.children = {}

    def __repr__(self):
        if self.feature is not None:
            return f'DecisionNode(feature="{self.feature}", children={self.children})'
        else:
            return f'LeafNode(label="{self.label}")'
```

حال تابع درخت تصمیم را ضمن قرار دادن شرطی را برای بررسی گره برگ بودن، می‌سازیم که داده و لیبل را دریافت می‌کند و IG را برای تمام ویژگی ها محاسبه و ویژگی با بیشترین IG را انتخاب می‌کند و به کلاس گره می‌دهد. سپس حلقه ای روی بهترین ویژگی ایجاد می‌کنیم تا subset ساخته شود. در این حلقه عملیات حذف ستون بهترین ویژگی انجام می‌شود. Subset ساخته شده را به عنوان دیتاست به تابع درخت تصمیم ساخته شده می‌دهیم و آن را به عنوان فرزند معرفی می‌کنیم.

```
def make_tree(data, target):
    # leaf node?
    if len(data[target].unique()) == 1:
        return Node(label=data[target].iloc[0])

    features = data.drop(target, axis=1).columns
    if len(features) == 0 or len(data) == 0:
        return Node(label=data[target].mode()[0])

    # calculate information gain
    gains = [information_gain(data, feature, target) for feature in features]

    # greedy search to find best feature
    max_gain_idx = np.argmax(gains)
    best_feature = features[max_gain_idx]

    # make a node
    node = Node(feature=best_feature)

    # loop over the best feature
    for value in data[best_feature].unique():
        subset = data[data[best_feature] == value].drop(best_feature, axis=1)
        # display(subset)

        node.children[value] = make_tree(subset, target)

    return node
```

حال درخت تصمیم را برای داده‌های آموزش سوال پیاده سازی می‌کنیم:

```
tree = make_tree(train_data, 'Drug')
tree
```

```
DecisionNode(feature="Na_to_K", children={'Normal': DecisionNode(feature="BP", children={'LOW': DecisionNode(feature="Cholesterol", children={'HIGH':
LeafNode(label="drugC"), 'NORMAL': LeafNode(label="drugX")}), 'HIGH': DecisionNode(feature="Age", children={'old': LeafNode(label="drugB"), 'adult':
DecisionNode(feature="Sex", children={'M': DecisionNode(feature="Cholesterol", children={'HIGH': LeafNode(label="drugA"), 'NORMAL': LeafNode(label="drugA")}), 'F':
DecisionNode(feature="Cholesterol", children={'NORMAL': LeafNode(label="drugB"), 'HIGH': LeafNode(label="drugB")})}), 'young': LeafNode(label="drugA")}), 'NORMAL':
LeafNode(label="drugX")}), 'High': LeafNode(label="drugY")})
```

بخشی از نتایج به‌صورت بالا نمایش داده شده است. برای نمایش root node داریم:

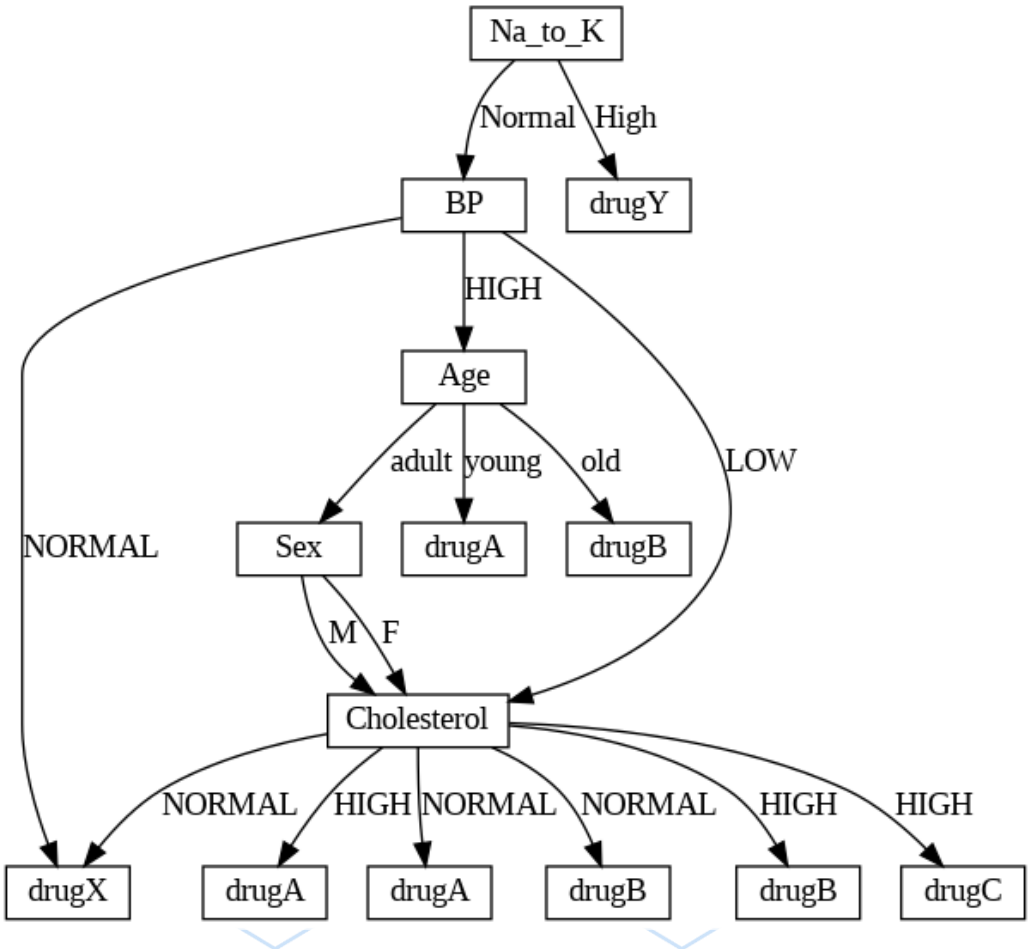
```
tree.feature
```

```
'Na_to_K'
```

با استفاده از graphviz و دستور زیر درخت تصمیم را رسم می‌کنیم:

```python
from graphviz import Digraph, nohtml

g = Digraph('g', filename='decision-tree.gv', node_attr={'shape': 'record', 'height': '.1'})

def plot_tree(tree, g):
    root_node = tree.feature
    if root_node is None:
        return g
    g.node(root_node, nohtml(root_node))
    child_nodes = tree.children.keys()

    for i, child in enumerate(child_nodes):
        node = tree.children[child]
        name = node.feature if node.feature is not None else child+node.label
        label = node.feature if node.feature is not None else node.label
        g.node(name, nohtml(label))
        g.edge(root_node, name, label=child)
        plot_tree(node, g)
    return g

g = plot_tree(tree, g)
g.render('decision_tree', format='png', view=True)
```

درخت تصمیم طراحی شده به‌صورت زیر قابل نمایش است:



همانطور که پیداست ابتدا درخت ویژگی با بیشترین IG را انتخاب و آن را به عنوان root node قرار داد. سپس اگر این ویژگی مقدار high داشت به آن لیبل Drug Y می‌زند. برای گره بعدی با محاسبه IG ویژگی فشار خون را انتخاب می‌کند که اگر normal بود به آن لیبل Drug X و اگر low بود وارد گره کلسترول می‌شود و اگر high بود وارد گره سن می‌شود. ورود به هریک از این گره ها بر مبنای IG و جستجوی

حریصانه است. فشار خون high با ورود به گره سن چنانچه سن در بازه های Young و Old باشد به‌ترتیب لیبل های Drug A و Drug B دریافت می کند و اگر سن در محدوده Adult باشد وارد گره جنسیت می‌شود(این مرحله در عملیات هرس کردن قابل حذف است). هر دو جنسیت وارد گره کلسترول شده و براساس normal یا high بودن لیبل متناظر را دریافت می‌کند.

۳-۲

با استفاده از ماتریس درهم ریختگی و حداقل سه شاخصهٔ ارزیابی مربوط به وظیفهٔ طبقه بندی، عملکرد درخت آموزش داده شدهٔ خود را روی بخش آزمون داده ها ارزیابی کنید و نتایج را به صورت دقیق گزارش کنید.

تأثیر مقادیر کوچک و بزرگ حداقل دو فراپارامتر را بررسی کنید. تغییر فراپارامترهای مربوط به هرس کردن چه تأثیری روی نتایج دارد و مزیت آن چیست؟

برای ارزیابی درخت طراحی شده به‌صورت زیر عمل می‌کنیم:

تابع predict را به‌گونه ای تعریف می‌کنیم که درخت و نمونه را دریافت کرده و ابتدا بررسی گره برگ بودن انجام می‌شود. سپس به گره فرزند رفته و حالت متناظر را انتخاب می‌کند و نمونه را مجددا دریافت می‌کند و دوباره بالا می‌آید تا نهایتا به لیبل مشخصی دست یابد.

```
def predict(node, sample):
    if node.feature is None:
        return node.label

    feature_value = sample[node.feature]

    if feature_value in node.children:
        return predict(node.children[feature_value], sample)
    else:
        return node.label
```

داده آزمون را به این تابع داده و پیش بینی درخت طراحی شده را گزارش می‌کنیم:

```
[predict(tree, sample) for _, sample in test_data.iterrows()]
```

```
['drugX',
 'drugY',
 'drugB',
 'drugA',
 'drugX',
 'drugX',
 'drugY',
 'drugC',
 'drugX',
 'drugY',
 'drugY',
 'drugY',
 'drugX',
 'drugY',
 'drugY',
 'drugC',
 'drugY',
 'drugY',
 'drugX',
 'drugY',
 'drugX',
 'drugX',
 'drugY',
 'drugX',
 'drugY',
 'drugX',
 'drugY',
 'drugX',
 'drugA']
```

برای مقایسه خروجی پیش بینی شده با خروجی های اصلی تست، بتدا باید index های آن‌را ریست کنیم:

```
actual = test_data.reset_index(drop=True)
actual['Drug']
```

```
0     drugX
1     drugY
2     drugB
3     drugA
4     drugX
5     drugX
6     drugY
7     drugC
8     drugX
9     drugY
10    drugY
11    drugY
12    drugX
13    drugY
14    drugY
15    drugC
16    drugY
17    drugY
18    drugX
19    drugY
20    drugX
21    drugX
22    drugY
23    drugX
24    drugY
25    drugX
26    drugY
27    drugX
28    drugX
29    drugA
Name: Drug, dtype: object
```

از مقایسه این مقادیر درمی‌یابیم تمام داده ها به درستی پیش بینی شده است. که این مسئله می‌تواند نشان‌دهنده overfit باشد. برای رفع این مسئله از تکنیک های هرس کردن استفاده می‌شود.

در این بخش از کتابخانه Sklearn و ماژول‌هایtree و DecisionTreeClassifier مجددا درخت تصمیم را این بار با قابلیت های تنظیم موارد مختلفی که کتابخانه در اختیارمان قرار داده‌است، پیاده سازی می‌کنیم:

معیار جستجو، عمق درخت و ccp_alpha (از پارامتر های دخیل در هرس کردن درخت) از جمله موارد قابل تنظیم برای درخت تصمیم اند.

از آنجاکه درخت تصمیم در scikit-learn به ورودی عددی برای آموزش نیاز دارد، در دیتاست اصلی داده های غیر عددی را مطابق قطعه کد زیر به داده‌های عددی تبدیل می‌کنیم:

```
data = pd.read_csv('drug200.csv')
#sex
sex = data['Sex']
#replacing labels
for i in range(200):
  if sex[i] == 'M':
     sex[i]= 0
  else:
     sex[i] = 1
data['Sex'] = sex
#BP
bp = data['BP']
#replacing labels
for i in range(200):
  if bp[i] == 'LOW':
     bp[i]= 0
  elif bp[i] == 'NORMAL':
     bp[i] = 1
  else:
     bp[i] = 2
data['BP'] = bp
#CH
ch = data['Cholesterol']
for i in range(200):
  if ch[i] == 'NORMAL':
     ch[i]= 0
  else:
     ch[i] = 1
data['Cholesterol'] = ch
```

دیتاست به‌صورت زیر خواهد بود:

```
print(data)
```

```
     Age Sex BP Cholesterol  Na_to_K   Drug
0     23   1  2           1   25.355  drugY
1     47   0  0           1   13.093  drugC
2     47   0  0           1   10.114  drugC
3     28   1  1           1    7.798  drugX
4     61   1  0           1   18.043  drugY
..   ...  .. ..         ...      ...    ...
195   56   1  0           1   11.567  drugC
196   16   0  0           1   12.006  drugC
197   52   0  1           1    9.894  drugX
198   23   0  1           0   14.020  drugX
199   40   1  0           0   11.349  drugX

[200 rows x 6 columns]
```

حال پس از تقسیم دیتاست به داده‌های آموزش و آزمون، درخت تصمیم را با استفاده از tree.DecisionTreeClassifier و معیار ’entropy‘ و عمق ۳ و ضریب هرس 0.1 طراحی می‌کنیم:

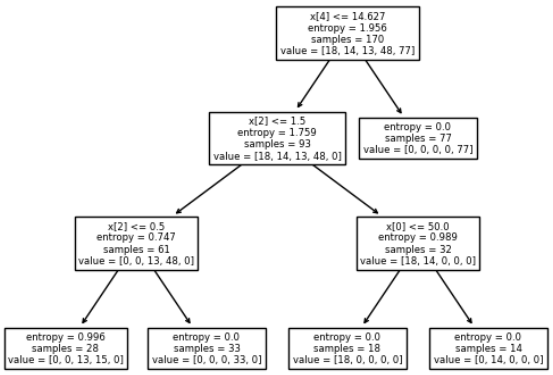
```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
np.random.seed(64)
# Convert DataFrame to a numpy array
array = data.values  #  numpy array
np.random.shuffle(array)  # Shuffle the array
shuffled_df = pd.DataFrame(array, columns=df.columns)
X_train, X_test, y_train, y_test = train_test_split(shuffled_df.drop(['Drug'], axis=1), shuffled_df['Drug'], test_size=0.15, random_state=64)
#tree
clf = tree.DecisionTreeClassifier(criterion='entropy',max_depth=3, random_state=64, ccp_alpha=0.1)
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.1, criterion='entropy', max_depth=3,
                       random_state=64)
```

درخت طراحی شده به‌صورت زیر خواهد بود:

```
tree.plot_tree(clf)
```

```
[Text(0.625, 0.875, 'x[4] <= 14.627\nentropy = 1.956\nsamples = 170\nvalue = [18, 14, 13, 48, 77]'),
 Text(0.5, 0.625, 'x[2] <= 1.5\nentropy = 1.759\nsamples = 93\nvalue = [18, 14, 13, 48, 0]'),
 Text(0.25, 0.375, 'x[2] <= 0.5\nentropy = 0.747\nsamples = 61\nvalue = [0, 0, 13, 48, 0]'),
 Text(0.125, 0.125, 'entropy = 0.996\nsamples = 28\nvalue = [0, 0, 13, 15, 0]'),
 Text(0.375, 0.125, 'entropy = 0.0\nsamples = 33\nvalue = [0, 0, 0, 33, 0]'),
 Text(0.75, 0.375, 'x[0] <= 50.0\nentropy = 0.989\nsamples = 32\nvalue = [18, 14, 0, 0, 0]'),
 Text(0.625, 0.125, 'entropy = 0.0\nsamples = 18\nvalue = [18, 0, 0, 0, 0]'),
 Text(0.875, 0.125, 'entropy = 0.0\nsamples = 14\nvalue = [0, 14, 0, 0, 0]'),
 Text(0.75, 0.625, 'entropy = 0.0\nsamples = 77\nvalue = [0, 0, 0, 0, 77]')]
```



۱-۲-۳ ارزیابی

برای ارزیابی درخت روی داده‌های آزمون، از متد predict استفاده و سپس دقت مدل را گزارش می‌کنیم:

```
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

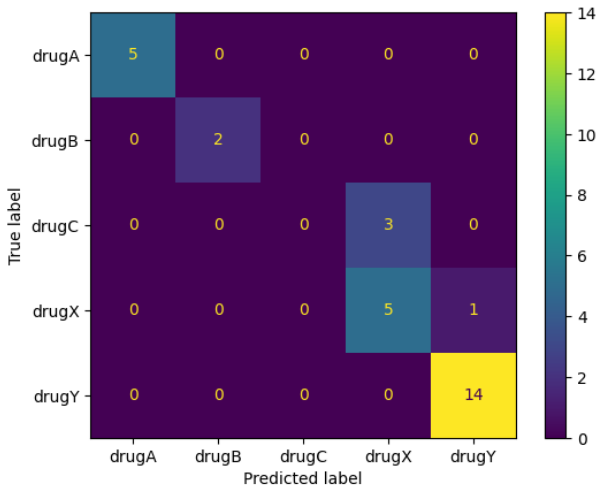
```
Accuracy: 0.87
```

دقت درخت طراحی شده ٪۸۷ است.

ماتریس درهم ریختگی بوسیله تابع metrics.confusion_matrix رسم می‌گردد:

```
# Compute confusion matrix
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

# Determine unique class labels
unique_labels = np.unique(y)
# Create ConfusionMatrixDisplay
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=unique_labels)

# Plot and show the confusion matrix
cm_display.plot()
plt.show()
```

ماتریس درهم ریختگی به‌صورت زیر است:



درباره ماتریس درهم ریختگی در سوال چهارم به‌طور مفصل بحث می‌شود. خروجی این ماتریس چهار قسمت دارد:True Positive, False Negative , False Positive, True Negative

اکثر خروجی ها در بخش True Positive قرار دارند و خروجی قابل قبول است. اما همانطور که پیداست درخت طراحی شده در کلاس Drug C موفق عمل نکرده است. همچنین یک داده متعلق به کلاس

Drug X را به اشتباه در کلاس Drug Y طبقه بندی کرده است. در سایر موارد عملکرد درخت درست است.

شاخصه های ارزیابی نظیر معیار Recall و Precision در ادامه محاسبه می‌شوند:

در رابطه با نحوه بدست آوردن مقادیر این معیارها در سوال چهارم صحبت می‌شود.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
```

نتایج به‌صورت زیر است:

```
Accuracy: 0.87
Precision: 0.79
Recall: 0.87
```

۲-۲-۳ اثر تغییر فراپارامترها

هایپرپارامتر متغیری است که مدل از آن در پیش بینی استفاده می‌کند و بر خلاف پارامترها، مدل با یادگیری تعیین نمی‌شوند، بلکه توسط مهندس طراح تنظیم می‌شوند. در اینجا اثر تغییر دو فراپارامتر max depth وccp_alpha بررسی می‌شوند.

اگر حداکثر عمق خیلی کم تنظیم شود، مدل قادر به یادگیری مسائل پیچیده نخواهد بود که این امر موجب افزایش خطا در داده آموزشی می‌شود. اگر حداکثر عمق بیش از حد زیاد تنظیم شود،ممکن است مدل داده را حفظ کند(overfit) و اگرچه خطا کمتر خواهد شد، اما نمی تواند به خوبی در همه داده های بعدی تعمیم یابد. به عبارت دیگر Generalization را از دست خواهیم داد و در داده های تست خطای بیشتری گزارش می شود.

مطلوب آن است که حداکثر عمق در مقدار بهینه تنظیم شود تا علاوه بر حفظ Generalization، از overfit جلوگیری شود.

در اینجا با در نظر گرفتن مقادیر مختلف max depth و محاسبه دقت مدل در هر حالت مقدار مناسب برای این فراپارامتر را گزارش می‌کنیم:

```
 #different max_depth
depths = [1, 2, 3, 4, 5, 6, 7, 8]
for depth in depths:
    clf =DecisionTreeClassifier(criterion='entropy',max_depth=depth, random_state=64, ccp_alpha=0.1)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Max Depth: {depth}, Accuracy: {accuracy:.2f}')
```

مقدار حداکثر عمق درخت از مقدار ۱ تا ۸ تغییر داده شده است . دقت مدل هر مرحله به‌صورت زیر است:

```
Max Depth: 1, Accuracy: 0.63
Max Depth: 2, Accuracy: 0.80
Max Depth: 3, Accuracy: 0.87
Max Depth: 4, Accuracy: 0.97
Max Depth: 5, Accuracy: 0.97
Max Depth: 6, Accuracy: 0.97
Max Depth: 7, Accuracy: 0.97
Max Depth: 8, Accuracy: 0.97
```

همانطور که پیداست از عمق ۴ به بعد عملکرد مدل ثابت مانده. پس حداکثر عمق بهینه برای مدل ۴ است.

همچنین کوچک بودن این فراپارامتر موجب دقت کم مدل و افزایش آن تا حد بهینه دقت را افزایش می دهد. در صورت افزایش نامعقول این فراپارامتر احتمال overfit افزایش می‌یابد.

ccp_alpha پارامتری است که برای کنترل هرس درخت تصمیم استفاده می شود. هرس به کاهش پیچیدگی مدل نهایی کمک می‌کند و می تواند Generalization آن را با کاهش overfit بهبود بخشد.

مقادیر بالاتر ccp_alpha درخت را ساده‌تر و مقادیر پایین تر آن درخت را پیچیده‌تر می‌سازد. اگرچه این مقدار باید بهینه شود. زیرا مقادیر بسیار بالای این پارامتر خطای مدل آموزشی را زیاد می‌کند و مقادیر بسیار کوچک آن درخت را پیچیده و احتمال overfit را افزایش می‌دهد.

```
 #different max_depth
ccp= [0.001, 0.01, 0.1, 0.5, 1, 3, 6, 9]
for ccpalpha in ccp:
    clf =DecisionTreeClassifier(criterion='entropy',max_depth=4, random_state=64, ccp_alpha=ccpalpha)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'ccp_alpha: {ccpalpha}, Accuracy: {accuracy:.2f}')
```

دقت مدل با مقادیر مشخص شده در قطعه کد بالا به‌صورت زیر است:

```
ccp_alpha: 0.001, Accuracy: 0.97
ccp_alpha: 0.01, Accuracy: 0.97
ccp_alpha: 0.1, Accuracy: 0.97
ccp_alpha: 0.5, Accuracy: 0.80
ccp_alpha: 1, Accuracy: 0.47
ccp_alpha: 3, Accuracy: 0.47
ccp_alpha: 6, Accuracy: 0.47
ccp_alpha: 9, Accuracy: 0.47
```

همانطور که پیداست از مقدار 0.1 به بعد دقت مدل کاهش می‌یابد. تا جایی که در مقدار ۹ دقت مدل از نصف کمتر است.

بنابراین مقدار بهینه در این دو فراپارامتر برابر است با:max depth=4 & ccp_alpha=0.1

دقت مدل در این حالت برابر است با:٪۹۷

```
clf =DecisionTreeClassifier(criterion='entropy',max_depth=4, random_state=64, ccp_alpha=0.1)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.97
```

دقت نسبت به حالت اولیه ۱۰ درصد افزایش یافت.

۳-۳

توضیح دهید که روش هایی مانند جنگل تصادفی و AdaBoost چگونه می توانند به بهبود نتایج کمک کنند. سپس، با انتخاب یکی از این روش ها و استفاده از فراپارامترهای مناسب، سعی کنید نتایج پیاده سازی در مراحل قبلی را ارتقاء دهید.

جنگل‌های تصادفی یک الگوریتم یادگیری متشکل از درختان تصمیم‌گیری است که در آن درخت‌های تصمیم ایجاد می‌شوند تا به جای انتخاب نقاط تقسیم بهینه، تقسیم‌های غیربهینه با معرفی تصادفی ایجاد شوند. هر درخت تصمیم در جنگل تصادفی بر روی یک زیر مجموعه تصادفی از داده‌های آموزشی و یک زیر مجموعه تصادفی از ویژگی‌ها آموزش داده می‌شود. سپس خروجی جنگل تصادفی با تجمیع پیش‌بینی‌های همه درخت‌های تصمیم تعیین می‌شود. این رویکرد به جنگل تصادفی اجازه می‌دهد تا بسیار دقیق و مقاوم در برابر مشکل overfitting ظاهر شود.

AdaBoost ، یک الگوریتم ترکیبی است که از روش‌های تاخیر و تقویت برای توسعه یک پیش‌بینی‌کننده ارتقا یافته، استفاده می‌کند. الگوریتم AdaBoost مشابه جنگل‌های تصادفی است به این معنا که پیش‌بینی‌ها از درختان تصمیم‌گیری زیادی گرفته می‌شوند. با این حال، سه تفاوت اصلی وجود دارد: ۱- AdaBoost به جای درخت جنگلی از درختان کنده‌شده ایجاد می‌شود. استامپ(کنده) درختی است که فقط از یک‌گره و دو برگ تشکیل شده‌است.۲- کنده‌های ایجاد شده در تصمیم‌گیری نهایی به یک اندازه وزن ندارند و تصمیماتی که خطای بیشتری ایجاد می‌کنند در تصمیم نهایی اثر کمتری خواهند داشت.۳- ترتیبی که در آن کنده ایجاد می‌شود مهم است، زیرا هر کنده قصد دارد خطاهای کنده قبلی را کاهش دهد. در این سوال تخمین‌گر پایه این الگوریتم DecisionTreeClassifier است.

ابتدا از الگوریتم جنگل تصادفی برای بهبود مدل استفاده می‌کنیم:

پارامترmax_depth: int, default=None حداکثر عمق درخت را نشـــان می‌دهد. اگر None باشـــد، گره‌ها تا زمانی گســـترش می‌یابند که همه برگ‌ها خالص شـــوند یا تا زمانی که همه برگ‌ها کمتر از min_samples_split داشته باشند.

برای این کار از ماژول RandomForestClassifier در کتابخانه sklearn استفاده می‌کنیم.

```
from sklearn.ensemble import RandomForestClassifier
# Define the Random Forest classifier
rf_clf = RandomForestClassifier(criterion='entropy',n_estimators=10, max_depth=4,ccp_alpha=0.1, random_state=64)

# Train the model
rf_clf.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_clf.predict(X_test)

# Evaluate accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print('Random Forest Accuracy:', accuracy_rf)
```

```
Random Forest Accuracy: 0.9666666666666667
```

دقت مدل حدود 97% بدســت آمد که نســبت به حالت اولیه بهبود داشــته اما وقتی در قســمت قبل هایپر پارامترها را بهینه کردیم به نتیجه مشابه رسیدیم.

برای پیاده سازی AdaBoost از ماژول AdaBoostClassifier در کتابخانه sklearn استفاده می‌کنیم.

با تنظیم پارامتر های مربوطه داریم:

```
from sklearn.ensemble import AdaBoostClassifier
# Define the AdaBoost classifier with Decision Tree as base estimator
base_estimator = DecisionTreeClassifier(criterion='entropy',max_depth=4, random_state=64, ccp_alpha=0.1)
ada_clf = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=10, learning_rate=0.001, random_state=64)

# Train the model
ada_clf.fit(X_train, y_train)

# Make predictions
y_pred_ada = ada_clf.predict(X_test)

# Evaluate accuracy
accuracy_ada = accuracy_score(y_test, y_pred_ada)
print('AdaBoost Accuracy:', accuracy_ada)
```

```
AdaBoost Accuracy: 0.9666666666666667
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed
  warnings.warn(
```

دقت مدل در این حالت نیز حدود ۹۷٪ گزارش می‌شود.

سوال چهارم

دیتاست بیماری قلبی را در نظر بگیرید. داده ها را به دو بخش آموزش و آزمون تقسیم کرده و ضمن انجام پیش پردازش هایی که لازم می دانید و با فرض گاوسی بودن داده ها، از الگوریتم طبقه بندیBayes استفاده کنید و نتایج را در قالب ماتریس درهم ریختگی وclassification_report تحلیل کنید. تفاوت میان دو حالتMacro و Micro را در کتابخانهٔ سایکیت لرن شرح دهید.

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

در گام اول دیتاست را روی گوگل درایو بارگزاری می کنیم. سپس با استفاده از دستور gdown فایل دیتاست را در گوگل کولب فراخوانی می‌کنیم.

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1BFqSrM-Pur8jn-kTB8abwnFI8tcoboI7
```

برای خواندن فایل csv و دریافت اطلاعات(تعداد ردیف و ستون داده، مقادیر پوچ و...) از دستور df.info() از کتابخانه pandas استفاده می‌کنیم.

```
import pandas as pd
df = pd.read_csv('/content/heart.csv')
df.info()
```

نتیجه دستور فوق به‌صورت زیر است:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

دیتاست بیماری قلبی در بردارنده ۱۴ ستون و ۱۰۲۵ داده است. ستون آخر با عنوان target بیانگر سالم یا بیمار بودن است (سالم=۰ و بیمار=۱). ۱۳ ستون دیگر فاکتور هایی مانند سن، جنسیت(زن=۰ و مرد=۱)، نوع درد قفسه سینه، فشارخون، کلسترول، قندخون و ... را نشان می‌دهد. همچنین تمام داده‌ها عددی هستند.

۴-۱ پیش پردازش داده

برای نمایش مقادیر Null یا NaN از دستور df.isna() استفاده می‌شود که برای هر سلول از دیتافریم مقدار True (دارای مقدار خالی) یا False (دارای مقدار غیر خالی)را بر می‌گرداند. برای شناسایی تعداد مقادیر خالی در هر ستون دیتافریم از دستور زیر استفاده و نتیجه را گزارش می‌کنیم:

```
for i in df.columns:
 print('Number of NaN in',i,'=',df.isna().sum().sum())
```

```
Number of NaN in age = 0
Number of NaN in sex = 0
Number of NaN in cp = 0
Number of NaN in trestbps = 0
Number of NaN in chol = 0
Number of NaN in fbs = 0
Number of NaN in restecg = 0
Number of NaN in thalach = 0
Number of NaN in exang = 0
Number of NaN in oldpeak = 0
Number of NaN in slope = 0
Number of NaN in ca = 0
Number of NaN in thal = 0
Number of NaN in target = 0
```

همانطور که پیداست هیچ مقدار خالی در دیتافریم وجود ندارد.

با استفاده از دستور groupby().size() می‌توان تعداد افراد با ویژگی موردنظر را نشان داد. برای مثال برای مشخص کردن تعداد افراد سالم و بیمار داریم:

```
df.groupby('target').size()
```

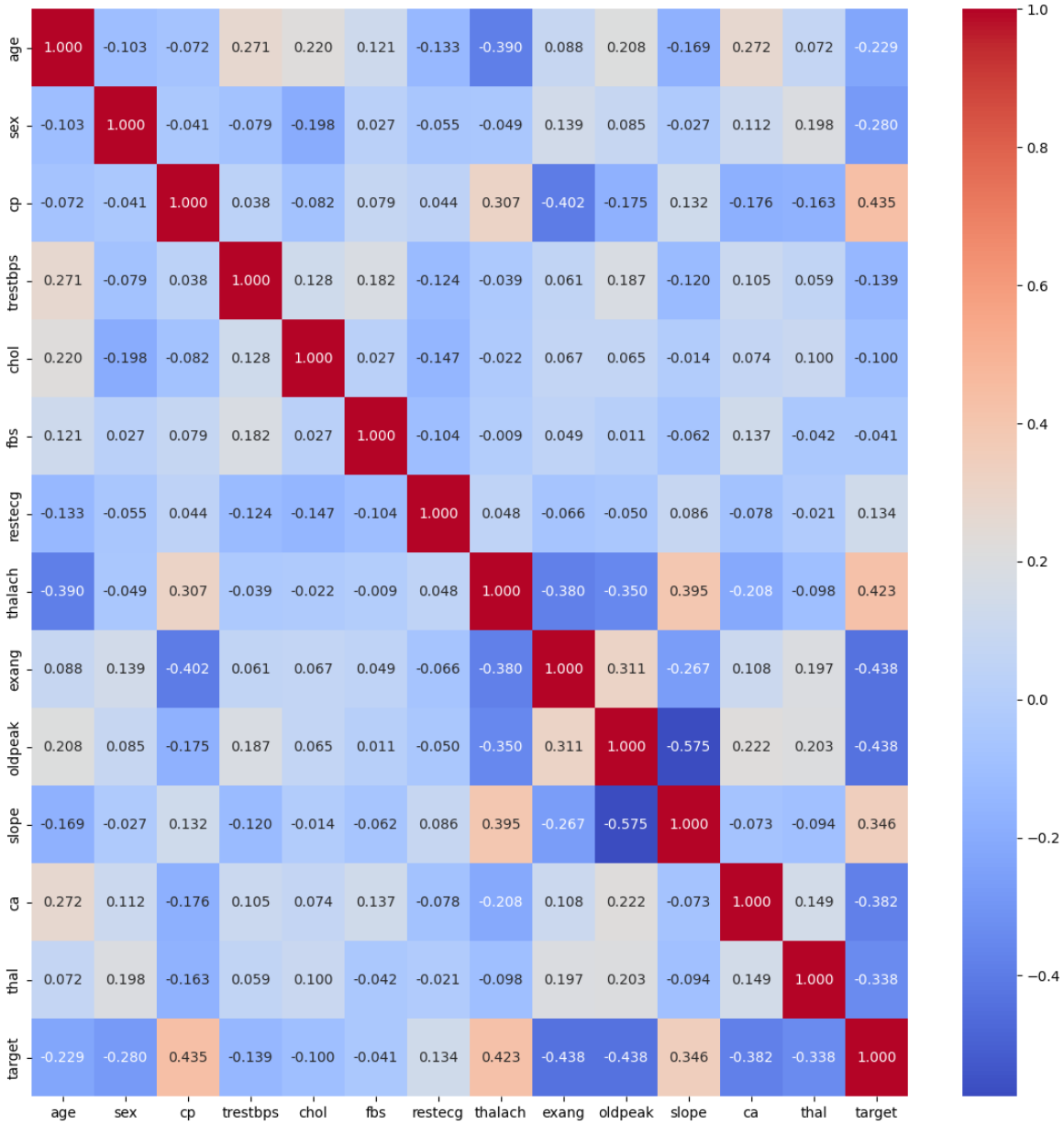
```
target
0    499
1    526
dtype: int64
```

بنابراین در این دیتاست، ۴۹۹ نفر بیمار و ۵۲۶ نفر سالم اند.

با ا ستفاده از د ستور df.corr() ماتریس همب ستگی میان متغیرها محا سبه می شود. بو سیله کتابخانه seaborn و تابع sns.heatmap() از ’coolwarmcmap‘ برای تعیین رنگ ها، از’fmt=.3f’ برای نمایش اعداد تا سه رقم اعشار و از ’yticklabels‘ برای نمایش نام هر ستون استفاده می‌شود. اگر ’annot=True‘مقدار عددی هر درایه ماتریس همبستگی در هر خانه نوشته می‌شود.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate correlation matrix
corr_matrix = df.corr()

# Create heatmap using seaborn
plt.figure(figsize=(14, 14))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.3f', yticklabels=corr_matrix.columns)
```

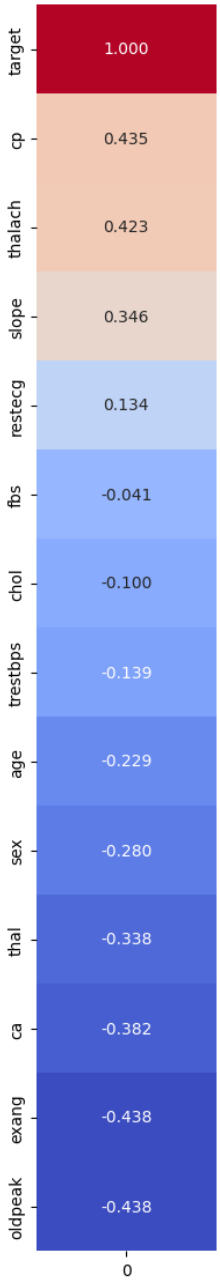
نتیجه به‌صورت زیر قابل نمایش است:



همانطور که پیداست، هر متغیر با خودش بیشترین همبستگی را دارد. برای بررسی بیشترین مقدار همبستگی با target از دستورات ‘df.corrwith()’ و ‘sort_values’ به ترتیب برای محاسبه همبستگی ستون ها با متغیر دلخواه (target) و مرتب سازی بر اساس مقدار(به‌طور پیشفرض از کمتر به بیشتر مرتب می‌شود. برای مرتب سازی از بزرگ به کوچک ‘ascending=False’ قرار می‌دهیم.) استفاده می‌کنیم.

```
corr_matrix = df.corrwith(df['target']).sort_values(ascending=False)

# Create heatmap using seaborn
plt.figure(figsize=(2,14))
sns.heatmap(corr_matrix.to_frame(), annot=True, cmap='coolwarm', fmt='.3f', cbar=False)
```

نتیجه به‌صورت روبرو است:



همانطور که پیداست فاکتور های درد سینه، ماکزیمم ضربان قلب و تغییرات قطعهST ، پس از target بیشترین مقدار همبستگی را با وضعیت سلامتی دارند.

برای مخلوط کردن داده ابتدا دیتافریم را به‌صورت آرایه numpy درآورده و با استفاده از دستور ‘np.random.shuffle()’ داده ها را مخلوط می‌کنیم ((random.seed(64) در نظر گرفته شده است.).

```
import pandas as pd
import numpy as np
np.random.seed(64)
# Convert DataFrame to a numpy array
array = df.values  #  numpy array
np.random.shuffle(array)  # Shuffle the array
shuffled_df = pd.DataFrame(array, columns=df.columns)
```

ستون target را به عنوان برچسب و سایر ستون ها را به عنوان داده در نظر می‌گیریم. پس از مخلوط کردن داده ها، آنها را به دو دسته آموزش و آزمون با نسبت ۲۰٪ : ۸۰٪ تقسیم می‌کنیم. این کار با استفاده از train_test_split() انجام می‌شود. در نهایت مجموعه داده و برچسب آموزش و آزمون را در فایل های جدیدی ذخیره می‌کنیم.

```
#Test & Train
from sklearn.model_selection import train_test_split
train_data, test_data, train_label, test_label = train_test_split(shuffled_df.drop(['target'], axis=1), shuffled_df['target'], test_size=0.2, random_state=64)
# Save train and test data to new CSV files
train_data.to_csv('train_data.csv', index=False)
test_data.to_csv('test_data.csv', index=False)
# Save train and test labels to new files
train_label.to_csv('train_label.csv', index=False)
test_label.to_csv('test_label.csv', index=False)

# Print shape of each set
print(f"train_data shape: {train_data.shape}")
print(f"test_data shape: {test_data.shape}")
print(f"train_label shape: {train_label.shape}")
print(f"test_label shape: {test_label.shape}")
```

ابعاد دیتا و برچسب آموزش و آزمون به‌صورت زیر است:

```
[1025 rows x 14 columns]
train_data shape: (820, 13)
test_data shape: (205, 13)
train_label shape: (820,)
test_label shape: (205,)
```

حال با فرض گوسی بودن توزیع داده، با استفاده از کتابخانه sklearn و دستور StandardScaler() داده ها را استاندارد سازی می‌کنیم. این دستور از معادله زیر برای استاندارد سازی استفاده می‌کند:

$$z = \frac{(x - u)}{s}$$

در عبارت بالا، u میانگین و s انحراف معیار داده های آموزش است.

در این عملیات نباید از داده های آزمون استفاده شود، چراکه باعث نشت اطلاعات از داده های آزمون به مدل یادگیری می‌شود و مدل، تعمیم پذیری و عملکرد خوبی نخواهد داشت. بنابراین ابتدا scaler را روی داده های آموزش فیت کرده و سپس از آن برای مقیاس بندی داده آزمون استفاده می‌کنیم.

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(train_data)
X_train = scaler.transform(train_data)
# scale the test data using the same scaler used for training data
X_test = scaler.transform(test_data)
y_train=train_label
y_test=test_label
print('train:',X_train.shape, y_train.shape,'\ntest: ', X_test.shape, y_test.shape)
```

```
train: (820, 13) (820,)
test:  (205, 13) (205,)
```

NaiveBayes Classifier ۴-۲

حال مطابق خواسته سوال با استفاده از کلاس GaussianNB (طبقه بند بر مبنای قضیه بیز) درکتابخانه sklearn کلاس بندی را انجام می‌دهیم. ابتدا مدل را برای داده ها و برچسب های آموزش فیت می‌کنیم.

سپس با استفاده از ()predict. مدل را با داده های آزمون، تست می‌نمائیم.

```
from sklearn.naive_bayes import GaussianNB
SKNB=GaussianNB()
SKNB.fit(X_train,y_train.ravel())
pred = SKNB.predict(X_test)
print(pred)
```

```
[0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 1. 1. 1. 0.
 0. 1. 1. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0. 1.
 0. 1. 0. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1. 0. 1. 0. 1. 1. 0.
 0. 1. 1. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0.
 1. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 1. 1. 0. 0.
 1. 0. 1. 1. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 1. 1.
 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1.
 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 1. 1. 1. 0.
 0. 0. 0. 1. 1. 0. 1. 0. 1. 1. 0. 1. 1.]
```

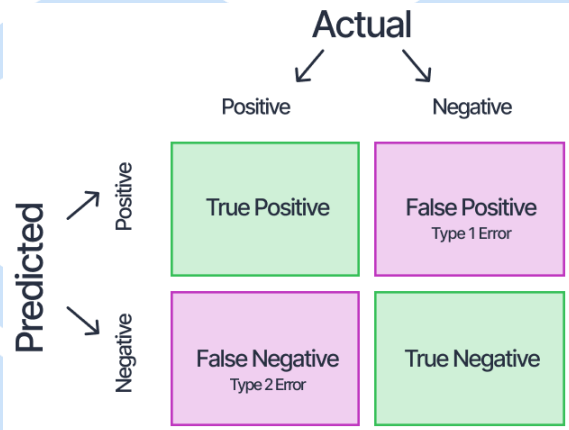
در قطعه کد بالا از ()reval. برای تصحیح ابعاد استفاده شده است.

۴-۳ معیارهای ارزیابی

۴-۳-۱ ماتریس درهم ریختگی

ماتریس درهم ریختگی، نتایج حاصل از تست مدل طبقه‌بند را بر اساس اطلاعات واقعی موجود، نمایش می‌دهد. بر اساس این مقادیر می‌توان معیارهای مختلف ارزیابی و اندازه‌گیری دقت را تعریف کرد.

ماتریس در هم ریختگی به‌صورت زیر است:

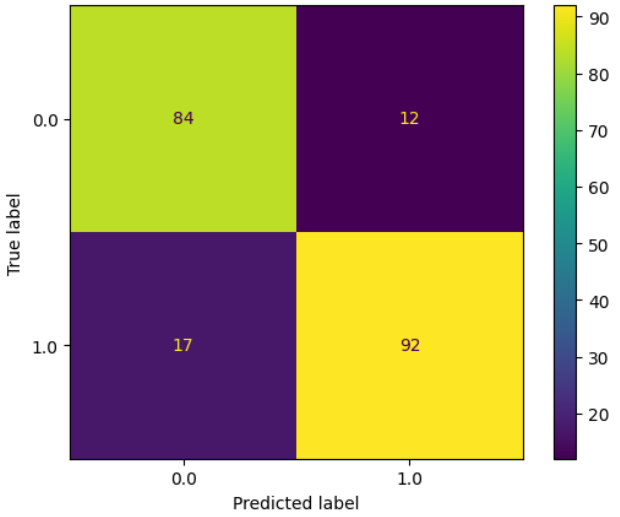


- نمونه عضو کلاس مثبت باشد و عضو همین کلاس تشخیص داده شود(مثبت صحیح یا True Positive)
- نمونه عضو کلاس مثبت باشد و عضو کلاس منفی تشخیص داده شود(منفی کاذب یا False Negative)
- نمونه عضو کلاس منفی باشد و عضو همین کلاس تشخیص داده شود (منفی صحیح یا True Negative)
- نمونه عضو کلاس منفی باشد و عضو کلاس مثبت تشخیص داده شود (مثبت کاذب یا False Positive)

برای رسم ماتریس درهم ریختگی از کتابخانهsklearn استفاده می‌شود. داده های واقعی و داده های پیش بینی شده را به confusion_matrix() می‌دهیم و نمایش ماتریس را بگونه ای تنظیم می کنیم تا ماتریس برای ارزیابی پیش بینی target ترسیم شود.

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test,pred)
names = list(shuffled_df.groupby('target').groups.keys())
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=names)
disp.plot()
plt.show()
```

نتیجه به‌صورت زیر قابل نمایش است:



باتوجه به مطالب ذکر شده و ماتریس بالا می‌توان گفت۸۴ مورد به درستی و ۱۲ مورد به غلط، سالم تشخیص داده شده است. همچنین ۹۲ مورد به درستی و ۱۷ مورد به غلط بیمار تشخیص داده شده است.

معیار های قابل استخراج از ماتریس درهم ریختگی عبارتند از:

۱- معیار صحت: Accuracy = (TP+TN) / (TP+FN+FP+TN)

۲- معیار دقت: Precision = TP / (TP+FP)

۳- معیار حساسیت یا Recall:Recall =TP / (TP+FN)

۴- معیار F1 Score: $F1Score=2\times\frac{Precision\times Recall}{\text{Precision+Recall}}$

با وارد کردن توابع accuracy_score, precision_score, recall_score, f1_score از کتابخانه sklearn می‌توان مدل را با شاخص های بالا ارزیابی نمود:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print('Accuracy :',accuracy_score(y_test,pred))
print('Precision :',precision_score(y_test,pred))
print('Recall :',recall_score(y_test,pred))
print('F1 score :',f1_score(y_test,pred))
```

```
Accuracy : 0.8585365853658536
Precision : 0.8846153846153846
Recall : 0.8440366972477065
F1 score : 0.863849765258216
```

همانطور که پیداست مدل، با معیار صحت عملکرد 85.8% داشته است.

۴-۳-۱-۱ بررسی حالت Macro و Micro در sklearn

کتابخانه sklearn در معیار های ارزیابی نظیر معیار های precision_score, recall_score , f1_score ، پارامتر average را به منظور تعیین نوع میانگین‌گیری روی داده ها در نظر گرفته است.

اگر average= *'micro'* به این معناست که معیار مورد نظر بطور کلی با مقادیر مثبت واقعی، منفی کاذب و مثبت کاذب محاسبه می‌شود.

اگر average= *'macro'* به این معناست که معیار مورد نظر برای هر برچسب محاسبه می شود و میانگین بدون وزن پیدا در نظر گرفته می‌شود. در اینجا عدم تعادل برچسب ها نادیده گرفته می‌شود.

برای مقایسه هر دو رویکرد micro وmacro معیار ها را با در نظر گرفتن پارامتر average مجددا محاسبه می‌کنیم:

```
print('Precision_micro :',precision_score(y_test,pred,average='micro'))
print('Precision_macro :',precision_score(y_test,pred,average='macro'))

print('Recall_micro :',recall_score(y_test,pred,average='micro'))
print('Recall_macro :',recall_score(y_test,pred,average='macro'))

print('F1 score_micro :',f1_score(y_test,pred,average='micro'))
print('F1 score_macro :',f1_score(y_test,pred,average='macro'))
```

```
Precision_micro : 0.8585365853658536
Precision_macro : 0.8581492764661081
Recall_micro : 0.8585365853658536
Recall_macro : 0.8595183486238532
F1 score_micro : 0.8585365853658536
F1 score_macro : 0.8583208217154024
```

همانطور که پیداست در این دیتاست هر سه معیار ذکر شده در هر دو رویکرد micro و macro نتایج نزدیک به همی گزارش می‌کنند.

۴-۳-۲ classification_report

در این قسمت با استفاده از کتابخانه sklearn و وارد کردن تابع classification_report، گزارشی شامل صحت(Accuracy) ، دقت(Precisions) ، Recall، f1_score و پشتیبانی (Support) است. با سه معیار اول در قسمت های قبل آشنا شدیم. معیار پشتیبانی را می توان در قالب تعداد نمونه های پاسخ صحیح که در هر کلاس از مقادیر هدف قرار می گیرد، تعریف کرد.

```
from sklearn.metrics import classification_report
print(classification_report(y_test,pred))
```

نتیجه به‌صورت زیر است:

	precision	recall	f1-score	support
0.0	0.83	0.88	0.85	96
1.0	0.88	0.84	0.86	109
accuracy			0.86	205
macro avg	0.86	0.86	0.86	205
weighted avg	0.86	0.86	0.86	205

جدول بالا عملکرد مدل را برای پیش بینی هر کلاس با معیار های صحت(Accuracy) ، دقت (Precisions)، Recall، f1_score و پشتیبانی(Support) بیان می‌کند. همچنین اثر پارامتر average با رویکرد macro و weighted در معیارها قابل مشاهده است. برای مثال، مدل توانسته است با معیار recall ۸۸ درصد از نمونه‌های کلاس افراد سالم را به درستی پیش‌بینی کند.

در قسمت قبل باMacro و Micro آشنا شدیم. اگر ’average=’weighted باشد، معیار برای هر برچسب با میانگین وزنی آنها محاسبه می‌شود. این رویکرد ’macro‘ را برای درنظر گرفتن عدم تعادل برچسب‌ها تغییر می‌دهد.

مراجع

[1] https://medium.com/@500087551/all-you-need-to-know-about-cwru-dataset-8d391577d8f2.
[2] https://virgool.io/@Zeynab.moradi/cross-validation-in-machine-learning-wf7u3nmrvvnu
[3] https://github.com/MJAHMADEE/MachineLearning2024W