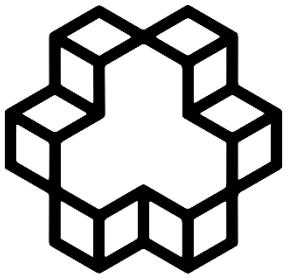


تمرین سری دوم درس یادگیری ماشین

استاد:

دکتر مهدی علیاری شوره‌دلی

دانشجو:

نازنین بنداریان

۴۰۲۱۳۸۷۶

بهار ۱۴۰۳

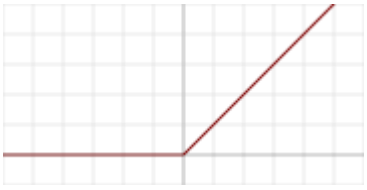
فهرست

لینک گیت هاب: https://github.com/nazaninbondarian/MachineLearning2024/

۱– سوال اول

۱-۱-بخش اول

تابع فعالسازی غیرخطی ReLUs یکی از رایج‌ترین توابع فعالسازی در یادگیری عمیق و شبکه‌های عصبی پیچشی ((Convolutional Neural Network(CNN) محسوب می‌شود. نام این تابع مخفف عبارت Rectified Linear Unit به معنای واحد خطی اصلاح شده است.



شکل ۱ تابع فعالساز ReLU

اگرچه این تابع، ظاهری شبیه به تابع خطی دارد، اما این تابع مشتق‌پذیر است و از آن می‌توان در مرحله انتشار رو به عقب استفاده کرد. این تابع تمامی گره‌ها را همزمان فعال نمی‌کند. به عبارتی، گره‌ها زمانی غیرفعال هستند که مقدار ورودی این تابع کمتر از صفر باشد. به عبارتی، براساس نمودار این تابع، چنانچه ورودی تابع بزرگتر از صفر باشد، خروجی تابع برابر با ورودی تابع و اگر ورودی تابع، مقداری کم‌تر از صفر باشد، خروجی تابع برابر با صفر است. تابع ReLUs و مشتق آن یکنواخت هستند.

مزایای تابع فعالسازی ReLUs به شرح زیر است:

- تابع ReLUs تابعی مشتق‌پذیر است.
- به دلیل ویژگی خطی بودن و غیراشباع‌کنندگی (non-saturating) تابع، همگرایی الگوریتم گرادیان کاهشی (Gradient Descent) سریع‌تر اتفاق می‌افتد. بدین ترتیب، مدل به زمان کمتری برای یادگیری نیاز دارد.
- از آنجا که این تابع فقط تعدادی از گره‌ها را فعال می‌کند، به لحاظ محاسباتی از کارایی بهتری نسبت به سایر توابع فعالسازی در شبکه های عصبی برخوردار است.
- این تابع در مقایسه با توابع سیگموئید و Tanh، تابعی ساده‌تر است و بنابراین محاسبات ریاضیاتی ساده‌تری در طی یادگیری شبکه انجام می‌شود.

- تابع ReLUs مشکل محوشدگی گرادیان توابع سیگموئید و Tanh را ندارد؛ زیرا محدوده مثبت خطی این تابع به گرادیان‌ها اجازه می‌دهد در مسیر فعال گره‌ها در جریان باشند. همچنین، این تابع محدودیتی برای ماکسیمم مقدار ورودی ایجاد نمی‌کند.

محدودیت مهمی که تابع ReLUs با آن مواجه می‌شود، مسئله زوال Dying تابع ReLUs است. این حالت زمانی اتفاق می‌افتد که خروجی چندین تابع RELUs به صورت متوالی برابر با صفر باشد. مسئله زوال تابع ReLUs زمانی رخ می‌دهد که ورودی این تابع مقادیر منفی باشند.

با اینکه ویژگی تابع ReLUs (یعنی نادیده گرفتن مقادیر منفی ورودی) باعث عملکرد بهتر شبکه در یادگیری می‌شود، زمانی که بیشتر ورودی‌های توابع ReLUs در بازه منفی باشند، مشکل زوال را به وجود می‌آورند. در حالتی که اکثر خروجی‌های توابع برابر با صفر شوند، در مرحله انتشار رو به عقب، گرادیان‌ها در طول شبکه جریان پیدا نمی‌کنند و به این ترتیب، وزن‌های شبکه به‌روزرسانی نمی‌شوند. بنابراین، در این حالت، بخش بزرگی از شبکه عصبی غیرفعال باقی می‌ماند و یادگیری شبکه به‌درستی انجام نمی‌شود.

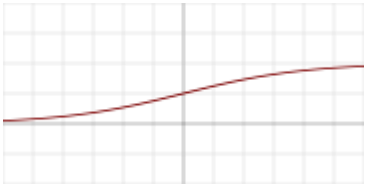
رابطه ریاضی فعالساز ReLUs به صورت زیر است:

$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} = \max(o, x)$$

مشتق تابع فعالسازی ReLUs در ادامه آمده است:

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \\ undefined, & x = 0 \end{cases}$$

تابع سیگموئید تابع فعالسازی غیرخطی، ورودی خود را به مقداری در بازه ۰ تا ۱ تبدیل می‌کند. هرچقدر مقدار ورودی بزرگ‌تر باشد، مقدار خروجی این تابع به عدد ۱ نزدیک‌تر می‌شود. در حالی‌که اگر مقدار ورودی این تابع خیلی کوچک باشد (عدد منفی)، مقدار خروجی تابع سیگموئید به عدد صفر نزدیک‌تر می‌شود. تابع سیگموید، تابع یکنوا (Monotonic) محسوب می‌شود اما مشتق این تابع، تابع یکنوا نیست.



شکل ۲ تابع فعالساز sigmoid

رابطه ریاضی فعالساز سیگموئید به صورت زیر است:

$$f(x) = \frac{1}{1+e^{-x}}$$

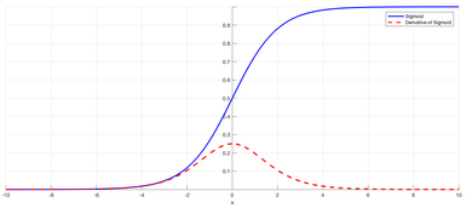
تابع سیگموئید به عنوان یکی از پرکاربردترین توابع فعالسازی غیرخطی محسوب می‌شود. مزایای استفاده از این تابع به شرح زیر هستند:

- معمولاً، از این تابع در مدل‌هایی استفاده می‌شود که خروجی مدل، مقداری احتمالاتی است. از آنجا که مقدار احتمال، عددی بین ۰ تا ۱ است، تابع سیگموئید بهترین انتخاب برای محاسبه احتمال محسوب می‌شود.
- تابع سیگموئید تابعی مشتق‌پذیر به حساب می‌آید و نمودار خروجی این تابع دارای شیب ملایم به شکل S است.

مهم‌ترین محدودیتی که تابع سیگموئید در شبکه های عصبی ایجاد می‌کند، مسئله محوشدگی گرادیان است که به دلیل مشتق این تابع اتفاق می‌افتد. مشتق تابع فعالسازی سیگموئید در ادامه آمده است:

$$f'(x) = f(x)\big(1 - f(x)\big)$$

منحنی مربوط به تابع مشتق سیگموئید در تصویر زیر مشاهده می‌شود.



شکل ۳ تابع فعالساز sigmoid و مشتق تابع فعالساز sigmoid

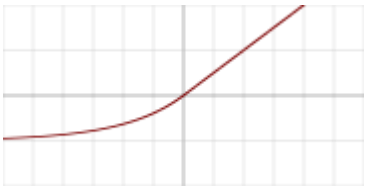
طبق تصویر بالا، مقادیر گرادیان در بازه ۳- تا ۳ مقدار قابل توجهی دارد و در سایر نواحی، مقدار گرادیان به عدد صفر متمایل می‌شود. در این حالت، مقدار گرادیان این تابع برای مقادیر بیشتر از ۳ یا کمتر از ۳- بسیار کوچک است. زمانی که مقدار گرادیان به سمت صفر میل می‌کند، یادگیری شبکه متوقف می‌شود که در این حالت، مسئله محوشدگی گرادیان اتفاق می‌افتد. به عبارتی، از آنجا که تابع سیگموئید فضای ورودی خود را به فضای بین ۰ تا ۱ تبدیل می‌کند، تغییرات بزرگ ورودی این تابع، باعث ایجاد تغییرات کوچک در خروجی می‌شوند و بنابراین، مشتق این تابع مقداری بسیار کوچک خواهد بود.

در شبکه هایی با تعداد لایه‌های کم، این مسئله مشکل‌ساز نیست؛ اما هرچقدر تعداد لایه‌های بیشتری از این تابع فعالسازی در شبکه عصبی استفاده شود، گرادیان تابع هزینه به صفر میل می‌کند و در این حالت، یادگیری شبکه دشوار خواهد شد.

۱-۲-بخش دوم

ReLU برای بخش منفی‌ها مقدار مشتقش صفر است، یعنی در هنگام مشتق‌گیری و بروزرسانی پارامترها برای مقادیر منفی حساسیت نشون نمی‌دهد. در صورت داشتن یک دیتاست که تمام مقادیر آن منفی می‌باشند، دیگر این تابع فعالساز کاربرد ندارد.

تابع فعالسازی ELUss یا تابع فعالسازی واحدهای خطی نمایی یکی از انواع تابع فعالسازی RELUs به حساب می‌آید که برای مقادیر ورودی منفی تابع، شیبی درنظر می‌گیرد. در توابع فعالسازی Leaky RELUs و Parametric RELUs نمودار توابع در ناحیه منفی، به‌صورت خط مستقیم است. تابع فعالسازی ELUs، برای این ناحیه، منحنی لگاریتمی به شکل زیر تعریف می‌کند.



شکل ۴ تابع فعالساز ELU

تابع ریاضی فعالساز ELUs به صورت زیر نوشته می‌شود:

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

مزیت‌های تابع فعالسازی ELUs نسبت به سایر توابع RELUs به شرح زیر است:

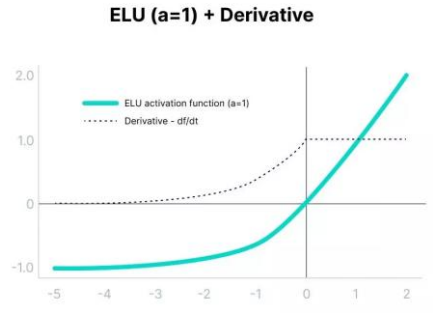
- شیب منحنی تابع ELUs بسیار ملایم است و این ویژگی تا رسیدن به مقدار $-\alpha$ ادامه‌دار است. در تابع فعالسازی RELUs چنین شیب ملایمی دیده نمی‌شود.
- این تابع، با درنظر گرفتن منحنی لگاریتمی برای مقادیر ورودی منفی، مانع بروز زوال تابع می‌شود. همچنین، این ویژگی باعث می‌شود مقادیر وزن‌ها و بایاس به‌درستی یاد گرفته شوند.

محدودیت‌های تابع فعالسازی ELUs به شرح زیر است:

- به دلیل انجام محاسبات نمایی، زمان محاسبات بیشتر می‌شود.
- در تابعELUs ، پارامتری (مشابه با پارامتر a در تابعParametric RELUs) وجود ندارد که در حین یادگیری شبکه عصبی، مقدار آن نیز یاد گرفته شود.
- در هنگام استفاده از این تابع ممکن است مشکل انفجار گرادیان رخ دهد.

مشتق این تابع به صورت زیر است.

$$f'(x) = \begin{cases} \alpha e^x, & x < 0 \\ 1, & x > 0 \\ 1, & x = 0, \alpha = 1 \end{cases} = \begin{cases} f(x) + \alpha, & x < 0 \\ 1, & x \geq 0 \end{cases}$$



شکل ۵ تابع فعالساز ELU و مشتق تابع فعالساز ELU

اگر تابع فعالسازی در یک طبقه‌بندی کننده دو لایه از ReLU در لایه اول و از Sigmoid در لایه خروجی استفاده کند، موارد زیر اتفاق می افتد:

لایه اول (فعالسازی ReLU): فعالسازی ReLU (واحد خطی اصلاح شده) تمام مقادیر منفی را صفر می‌کند در حالی که مقادیر مثبت را بدون تغییر نگه می‌دارد. این به کاهش مشکل ناپدید شدن گرادیان کمک می‌کند، زیرا ReLU اشباع نمی‌شود و به گرادیان‌ها اجازه می‌دهد تا به طور موثر در شبکه جریان داشته باشند، که منجر به آموزش سریع‌تر و کارآمدتر می‌شود.

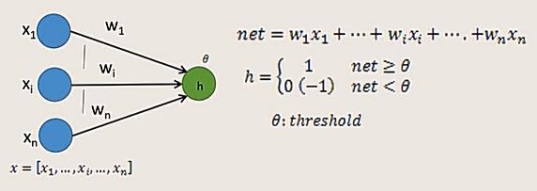
لایه خروجی (فعالسازی سیگموئید): فعالسازی سیگموید خروجی را بین ۰ و ۱ حفظ می‌کند. این به ویژه برای کارهای طبقه بندی باینری مفید است، زیرا می‌توان آن را به عنوان یک احتمال تفسیر کرد. با این حال، توابع سیگموئید در صورت استفاده در لایه‌های پنهان می‌توانند از مشکل ناپدید شدن گرادیان رنج ببرند، اما این موضوع در لایه خروجی کمتر نگران کننده است.

با ترکیب این دو، ReLU در لایه اول انتشار گرادیان کارآمد را در طول آموزش تضمین می‌کند، در حالی که سیگموئید در لایه خروجی با تولید مقادیر بین ۰ و ۱، خروجی نهایی را برای طبقه‌بندی باینری مناسب است.

۱-۳-بخش سوم

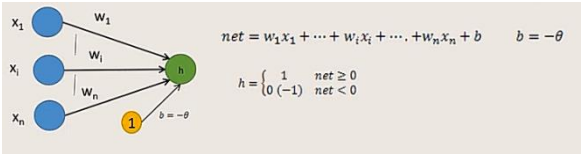
در یک شبکه ساده تک لایه، هر کدام از ورودی‌ها با وزن مرتبط با خود وارد نورون می‌شوند و یک خروجی به اسم net را می‌سازند. خروجی از یک تابع فعالساز عبور می‌کندو خروجی نهایی شبکه عصبی ما را می‌سازد.

در نورون McCullpch-pitts با در نظر گرفتن یک مقدار آستانه، اگر مقدار خروجی وزن‌دار net از آن مقدار آستانه بیشتر و یا مساوی باشد مقدار یک، در صورتی که کمتر از آن باشد مقدار صفر یا منفی یک را باز می‌گرداند. در این شبکه ما باید یک مقدار آستانه و مقادیر اولیه وزن‌ها را مشخص کنیم.



شکل ۶ نورون McCullpch-pitts – آستانه

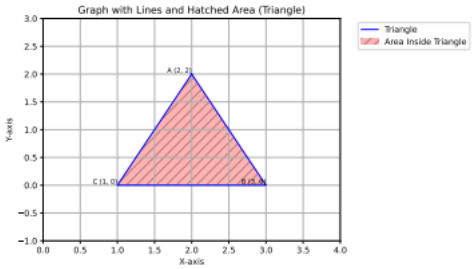
همانطور که در شکل زیر نشان داده شده است می‌توانیم مقدار آستانه را به آن سمت ناتساوی منتقل کنیم و با در نظر گرفتن به عنوان مقدار بایاس ادامه دهیم. با استفاده از بایاس می‌توانیم به معادله یک خط مقدار عرض از مبدأ را بیافزاییم. یکی دیگر از فواید بایاس را می‌توانیم به حالتی اشاره کنیم که تمام ورودی‌ها صفر می‌باشد؛ در نتیجه جمع وزن‌دار آنها صفر می‌گردد، در صورتی که در دیتاست ما برچسب ۱- خورده باشد. در صورت نبودن بایاس صفر بازمی‌گرداند. در نتیجه بایاس به ما یک درجه آزادی می‌دهد.



شکل ۷ نورون McCullpch-pitts – بایاس

در صورتی که دارای ویژگی‌های بیشتر باشیم، دیگر معادله خط را نباید بیابیم. به عنوان مثال برای سه ویژگی باید معادله یک صفحه را بیابیم؛ برای چهار ویژگی باید معادله یک ابر صفحه را بیابیم. درواقع همیشه بعد یکی کمتر از تعداد ویژگی‌ها است.

معادلات سه خط رسم شده در شکل زیر را محاسبه می‌کنیم. در واقع وزن‌ها و بایاس نورون‌ها را با استفاده از معادلات خط به دست میاوریم.



شکل ۸ نمودار هاشورزده‌ی مورد سوال

$$y - y_1 = \frac{y_1 - y_2}{x_1 - x_2}(x - x_1)$$

$$A = (2 \quad 2)\ , B = (3 \quad 0) \rightarrow y - 2 = \frac{2-0}{2-3}(x-2) \rightarrow y - 2 = -2(x-2) \rightarrow y = -2x + 6$$

$$A = (2 \quad 2)\ , C = (1 \quad 0) \rightarrow y - 2 = \frac{2-0}{2-1}(x-2) \rightarrow y - 2 = 2(x-2) \rightarrow y = 2x - 2$$

$$B = (3 \quad 0)\ , C = (1 \quad 0) \rightarrow y - 0 = \frac{0-0}{3-1}(x-3) \rightarrow y - 0 = 0 \rightarrow y = 0$$

در نتیجه می‌توان معادلات را به صورت زیر بازنویسی کرد.

$$net_1 = -2x_1 - 1x_2 + 6\ , net_2 = 2x_1 - x_2 - 2\ , net_3 = 0x_1 - x_2 - 0$$

برای بررسی اینکه داده‌ها در داخل دایره ایت و یا خارج آن می‌توانیم به صورت زیر در نظر بگیریم.

$$u_1 + u_2 + u_3 = 3$$

در ابتدا کتابخانه‌های مورد نیاز را می‌افزاییم.

```
import numpy as np
import random
```

سپس برای اینکه کلاس نورون McCullpch-pitts را تعریف کنیم، باید متغیرهایی مورد استفاده را تعریف کنیم؛ ما از وزن‌ها و بایاس استفاده می‌کنیم. سپس باید تعریف کنیم که مدلمان چگونه کار می‌کند. در این تابع به سادگی تعریف شده است که در صورتی که ضرب ورودی در وزن‌ها مقدار بزرگتر از یک باشد، مقدار یک و در غیراینصورت مقدار صفر را بازگرداند.

```
#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
```

سپس باید خروجی‌های حالت خود را تعریف کنیم. به این منظور یک تابع تعریف شده است. در ابتدا سه شیء از کلاس McCulloch_Pitts_neuron تعریف می‌کنیم. این کلاس دارای دو ورودی می‌باشد؛ وزن‌ها و مقدار آستانه. چون داده‌های ما دارای دو ویژگی است، دو وزن قرار می‌دهیم. شیء چهارم نیز برای بررسی اینکه آیا داخل مثلث و یا خارج آن استفاده شده است. سپس با استفاده از تابع Model بررسی می‌کنیم که نقاط در کدام ناحیه قرار دارند.

```
#define model for dataset
def Area(x, y):
  neur1 = McCulloch_Pitts_neuron([-2, -1], -6)
  neur2 = McCulloch_Pitts_neuron([2, -1], 2)
  neur3 = McCulloch_Pitts_neuron([0, 1], 0)
  neur4 = McCulloch_Pitts_neuron([1, 1, 1], 3)
  z1 = neur1.model(np.array([x, y]))
  z2 = neur2.model(np.array([x, y]))
  z3 = neur3.model(np.array([x, y]))
  z4 = neur4.model(np.array([z1, z2, z3]))
  return list([z4])
```

سپس با استفاده از دستور زیر ۲۰۰۰ داده در بازه‌ی مدنظر تولید می‌کنیم.

```
import matplotlib.pyplot as plt
# Generate random data points
random.seed(76)
num_points = 2000
x_values = np.random.uniform(0, 4, num_points)  # Updated x-axis limits
```

```
y_values = np.random.uniform(-1, 3, num_points)  # Updated y-axis limits
```

سپس لیست‌های خالی را برای ذخیره کردن داده‌های خارج و داخل دایره‌ی واحد ایجاد می‌کنیم.

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

سپس با استفاده از دستورات زیر بررسی می‌کند که داده داخل دایره است و یا خارج آن. درون حلقه for در ابتدا تابع Area را فراخوانی می‌کند، این فراخوانی به تعداد داده‌ها صورت می‌گیرد. در صورتی که مقدار نورون چهارم صفر باشد، داده در خارج مثلث و به رنگ قرمز می‌باشد. در غیر اینصورت داده در داخل مثلث و به رنگ سبز می‌باشد.

```
# 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]))
```

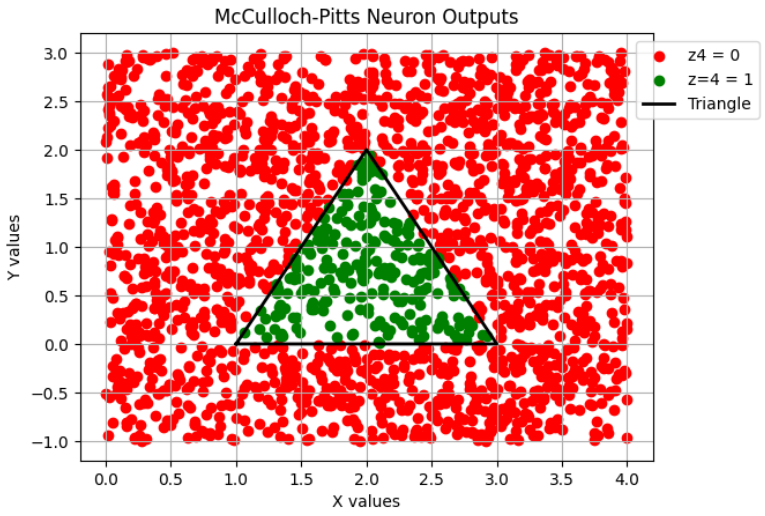
با استفاده از دستور زیر مقادیر x و y خروجی بخش قبل را جداسازی می‌کنیم.

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

با استفاده از دستورات زیر نتایج را نشان می‌دهیم. نام هر کدام از نمودارها را مشخص می‌کنیم. و عنوان صفحه تصویر را نیز مشخص می‌کنیم. مرز تصمیم گیری که یک مثلث است را با خطوط مشکی رسم می‌کنیم. نوع هر کدام از داده‌های رسم شده نیز مشخص شده است.

```
# Plotting
plt.scatter(red_x, red_y, color='red', label='z4 = 0')
plt.scatter(green_x, green_y, color='green', label='z=4 = 1')
plt.xlabel('X values')
```

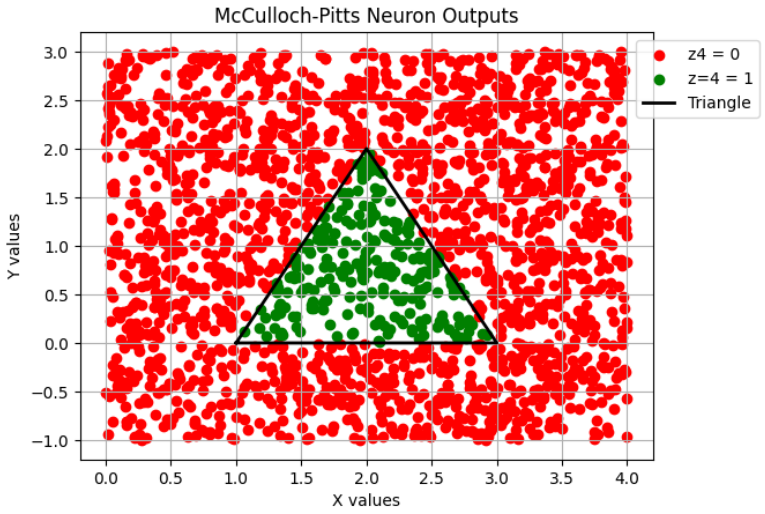
```
plt.ylabel('Y values')
plt.title('McCulloch-Pitts Neuron Outputs')


# Plotting lines with legends
plt.plot([1, 2, 3, 1], [0, 2, 0, 0], color='black', linestyle='-', linewidth=2 ,label='Triangle')
plt.grid(True)
# Position the legends at the top and right
plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1.0))
plt.show()
```



شکل ۹ خروجی برنامه

برای بررسی بیشتر اثر تابع فعالساز sign را به صورت زیر در کلاس McCulloch_Pitts_neuron برای تابع Model تعریف می‌کنیم.

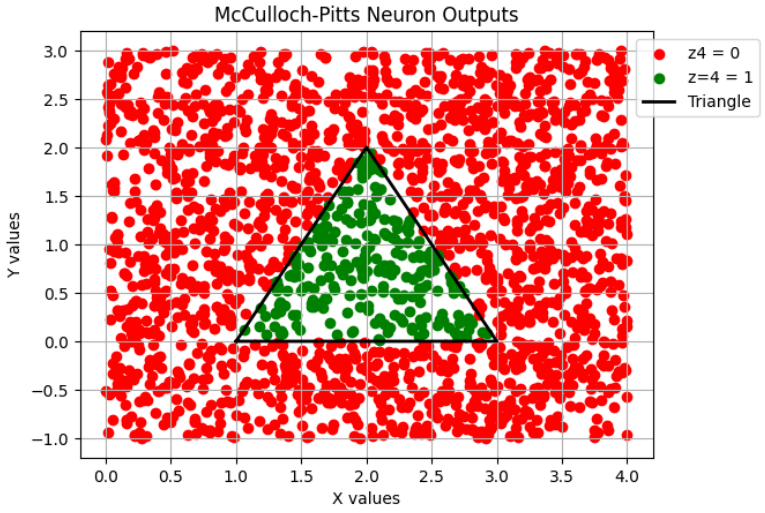
```
def model(self , x):
    #define model with threshold
    if np.sign(self.weights @ x - self.threshold)>=0:
        return 1
    else:
        return 0
```



شکل ۱۰ خروجی برنامه

برای بررسی بیشتر اثر تابع فعالساز سیگموئید را به صورت زیر در کلاس McCulloch_Pitts_neuron برای تابع Model تعریف می‌کنیم.

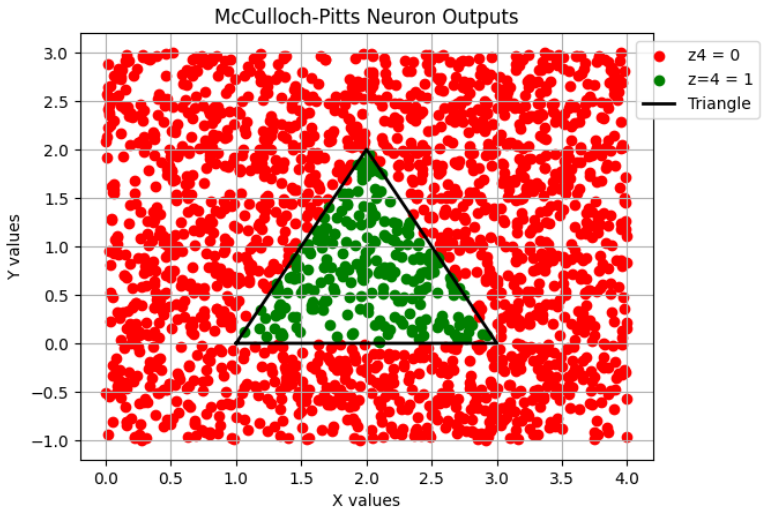
```
def model(self, x):
    # Define model with threshold
    if 1/(1+np.exp(-(self.weights @ x - self.threshold)))>=0.5:
        return 1
    else:
        return 0
```



شکل ۱۱ خروجی برنامه

برای بررسی بیشتر اثر تابع فعالساز tanh را به صورت زیر در کلاس McCulloch_Pitts_neuron برای تابع Model تعریف می‌کنیم.

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



شکل ۱۲ خروجی برنامه

همانطور که در تصاویر می‌توان نشاهده کرد که انتخاب تابع فعالساز گوناگون تاثیری در نحوه کلاس‌بندی این کلاس نداشته است.

۲– سوال دوم

۲-۱-بخش اول

همانطور که در گزارش قبلی نیز به آن اشاره شد داده‌ها شامل انواع مختلفی از عیوب مانند عیوب در شعاع داخلی، توپ و شعاع بیرونی بلبرینگ است. داده‌ها مکان دقیق خطاها را مشخص می‌کنند، مانند خطاهای بلبرینگ انتهای درایو و موقعیت آنها نسبت به منطقه بار می‌باشد. بلبرینگ‌های موتور با استفاده از ماشین‌کاری تخلیه الکتریکی (EDM) با عیوب کاشته شدند. گسل‌هایی با قطر ۰.۰۰۷ اینچ تا قطر ۰.۰۴۰ اینچ به طور جداگانه در مسیر ورودی داخلی، عنصر نورد (یعنی توپ) و مسیر بیرونی معرفی شدند. بلبرینگ‌های معیوب دوباره در موتور آزمایشی نصب شدند و داده‌های ارتعاش برای بارهای موتور ۰ تا ۳ اسب بخار (سرعت موتور ۱۷۲۰ تا ۱۷۹۷ RPM) ثبت شد.

IR007_0 مربوط به خطای شعاع داخلی، B007_0 مربوط به خطای توپ و OR007@6_0 مربوط به خطای شعاع خارجی که در موقعیت ساعت ۶ محل بار اعمال شده است. گسل‌هایی با قطر ۰.۰۰۷ اینچ به طور جداگانه در مسیر ورودی داخلی، عنصر نورد (یعنی توپ) و مسیر بیرونی ایجاد شده‌اند.

در ابتدا یک فایل برای ذخیره سازی داده‌ها ایجاد می‌کنیم.

```
import os
# Define the folder name
folder_name = "Data"
# Check if the folder already exists, and create it if not
if not os.path.exists(folder_name):
   os.makedirs(folder_name)
# Commented out IPython magic to ensure Python compatibility.
%cd Data
```

سپس فایل mat. مربوط به دیتاست را توسط دستورات زیر دریافت می‌کنیم.

```
!wget -q https://engineering.case.edu/sites/default/files/97.mat
!wget -q https://engineering.case.edu/sites/default/files/105.mat
!wget -q https://engineering.case.edu/sites/default/files/118.mat
!wget -q https://engineering.case.edu/sites/default/files/130.mat
```

فایل‌های دریافت شده را توسط دستور زیر آپلود می‌کنیم.

```
from scipy.io import loadmat
NN = loadmat("/content/Data/97.mat")
FF1 = loadmat("/content/Data/105.mat")
FF2 = loadmat("/content/Data/118.mat")
FF3 = loadmat("/content/Data/130.mat")
```

با دستور زیر عنوان‌های کلیدی مربوط یه دیتاست که عنوان‌های ستون می‌باشد، را نمایش می‌دهیم.

```
print(NN.keys())
print(FF1.keys())
print(FF2.keys())
print(FF3.keys())
```

```
dict_keys(['__header__', '__version__', '__globals__', 'X097_DE_time', 'X097_FE_time', 'X097RPM'])
dict_keys(['__header__', '__version__', '__globals__', 'X105_DE_time', 'X105_FE_time', 'X105_BA_time', 'X105RPM'])
dict_keys(['__header__', '__version__', '__globals__', 'X118_DE_time', 'X118_FE_time', 'X118_BA_time', 'X118RPM'])
dict_keys(['__header__', '__version__', '__globals__', 'X130_DE_time', 'X130_FE_time', 'X130_BA_time', 'X130RPM'])
```

شکل ۱۳ عنوان‌های ستون

سپس با عنوان سرستون‌های به دست آمده داده‌ها را به صورت فایل csv. تبدیل کرده‌ایم و ذخیره می‌کنیم.

```
NNa = NN['X097_DE_time']
NNb = NN['X097_FE_time']
NNc = NN['X097RPM']
FF1a = FF1['X105_DE_time']
FF1b = FF1['X105_FE_time']
FF1c = FF1['X105_BA_time']
FF1d = FF1['X105RPM']
FF2a = FF2['X118_DE_time']
FF2b = FF2['X118_FE_time']
FF2c = FF2['X118_BA_time']
FF2d = FF2['X118RPM']
```

```
FF3a = FF3['X130_DE_time']

FF3b = FF3['X130_FE_time']

FF3c = FF3['X130_BA_time']

FF3d = FF3['X130RPM']

Normal = pd.DataFrame({'X097_DE_time': NNa.flatten(), 'X105_BA_time': NNb.flatten()})

Normal.to_csv("/content/Data/Normal.csv",index=False)
```

	X097_DE_time	X105_BA_time
0	0.053197	0.145667
1	0.088662	0.097796
2	0.099718	0.054856
3	0.058621	0.036982
4	-0.004590	0.054445
...	...	...
243933	-0.059664	0.142791
243934	-0.063836	0.148955
243935	-0.034630	0.140531
243936	0.016689	0.095536
243937	0.046938	0.090195

243938 rows × 2 columns

شکل ۱۴ داده‌های نرمال به صورت دیتافرم

```
Fault1 = pd.DataFrame({'X105_DE_time': FF1a.flatten(), 'X105_FE_time': FF1b.flatten(), 'X105_BA_time': FF1c.flatten()})

Fault1.to_csv("/content/Data/Fault1.csv",index=False)
```

	X105_DE_time	X105_FE_time	X105_BA_time
0	-0.083004	-0.402075	0.064661
1	-0.195734	-0.004725	-0.023096
2	0.233419	-0.106631	-0.088522
3	0.103958	-0.074169	-0.093632
4	-0.181115	0.208947	-0.076491
...	...	...	...
121260	0.324545	-0.078484	0.010421
121261	0.142456	-0.012327	0.038306
121262	-0.316424	0.315989	0.096489
121263	-0.063675	0.350916	0.084056
121264	0.267368	0.033078	-0.020159

121265 rows × 3 columns

شکل ۱۵ داده‌های عیب کلاس IR007_0 به صورت دیتافرم

```
Fault2 = pd.DataFrame({'X118_DE_time': FF2a.flatten(), 'X118_FE_time': FF2b.flatten(), 'X118_BA_time': FF2c.flatten()})

Fault2.to_csv("/content/Data/Fault2.csv",index=False)
```

	X118_DE_time	X118_FE_time	X118_BA_time
0	-0.002761	-0.247162	0.015532
1	-0.096324	0.142791	0.016940
2	0.113705	0.003287	-0.036455
3	0.257297	-0.106836	-0.044744
4	-0.058314	0.136011	0.007726
...	...	...	...
122566	0.045157	0.046638	0.023901
122567	0.111106	-0.094304	-0.003621
122568	-0.078294	0.016436	0.031788
122569	-0.149115	-0.038420	-0.032431
122570	0.021117	-0.168062	-0.066834

122571 rows × 3 columns

شکل ۱۶ داده‌های عیب کلاس B007_0 به صورت دیتافرم

```
Fault3 = pd.DataFrame({'X130_DE_time': FF3a.flatten(), 'X130_FE_time': FF3b.flatten(), 'X130_BA_time': FF3c.flatten()})

Fault3.to_csv("/content/Data/Fault3.csv",index=False)
```

	X130_DE_time	X130_FE_time	X130_BA_time
0	0.008528	-0.407005	-0.000040
1	0.423550	0.262776	0.069329
2	0.012995	0.495145	0.030661
3	-0.265175	-0.423442	-0.037461
4	0.237155	-0.307155	-0.116165
...	...	...	...
121986	0.023553	-0.070676	-0.018912
121987	0.028426	0.011300	-0.001690
121988	0.175836	0.059582	0.118298
121989	0.110050	-0.050747	0.056654
121990	-0.102740	0.027325	-0.010502

121991 rows × 3 columns

شکل ۱۷ داده‌های عیب کلاس OR007@6_0 به صورت دیتافرم

سپس بررسی صورت می‌گیردد که آیا داده null وجود دارد یا خیر. از آنجایی که یافت نشد نیاز به حذف ستونی نمی‌باشد.

```
print(Normal.isnull().sum())
```

```
X097_DE_time    0
X105_BA_time    0
dtype: int64
```

شکل ۱۸ بررسی داده‌های null کلاس نرمال

```
print(Fault1.isnull().sum())
```

```
X105_DE_time    0
X105_FE_time    0
X105_BA_time    0
dtype: int64
```

شکل ۱۹ بررسی داده‌های null کلاس عیب IR007_0

```
print(Fault2.isnull().sum())
```

```
X118_DE_time    0
X118_FE_time    0
X118_BA_time    0
dtype: int64
```

شکل ۲۰ بررسی دادههای null کلاس عیب B007_0

```
print(Fault3.isnull().sum())
```

```
X130_DE_time    0
X130_FE_time    0
X130_BA_time    0
dtype: int64
```

شکل ۲۱ بررسی داده‌های null کلاس عیب OR007@6_0

داده‌های ستون اول هر کلاس را در نظر می‌گیریم.

```
Normal_data = Normal.X097_DE_time
Fault1_data = Fault1.X105_DE_time
Fault2_data = Fault2.X118_DE_time
Fault3_data = Fault3.X130_DE_time
```

سپس با توجه به درخواست صورت سوال ماتریس داده‌های با ابعاد 200×100 برای هر کلاس ایجاد می‌کنیم. در ابتدا یک لیست خالی را برای افزودن مقادیر تولید شده ایجاد می‌کنیم. کد np.random.seed(76) را برای مولد اعداد تصادفی NumPy روی ۷۶ تنظیم می‌کند. این تضمین می‌کند که دنباله اعداد تصادفی تولید شده توسط NumPy قابل تکرار خواهند بود اگر یک کد را چندین بار با همان seed اجرا کنید. مقدار seed می‌تواند هر عدد صحیحی باشد. استفاده از یک Seed ثابت برای اهداف اشکال زدایی و آزمایش مفید است، زیرا به شما امکان می دهد هر بار که کد خود را اجرا می‌کنید اعداد تصادفی یکسانی را بدست آورید و از نتایج ثابت اطمینان حاصل کنید.

```
# Extract 100 samples with length of 200
sample_length = 200
num_samples = 100
Normal_samples = []
np.random.seed(76)
# Extract samples from class Normal
for i in range(num_samples):
    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)
    Normal_samples = np.array(Normal_samples)
    Normal_samples.shape
```

لیست ایجاد شده در قبل را با کمک دستور زیر به آرایه تبدیل می‌کنیم و ابعاد آن را نمایش می‌دهیم.

```
    # Convert lists of samples to numpy arrays
    Normal_samples = np.array(Normal_samples)
    Normal_samples.shape
```

```
(100, 200)
```

شکل ۲۲ ابعاد داده‌های نرمال

این فرآیند برای سایر کلاس‌ها نیز تکرار شده است (در دفترچه کد موجود است). سپس آرایه‌های ایجاد شده برای هر کلاس را به هم متصل می‌کنیم. ابعاد ماتریس ایجاد شده را نیز نمایش می‌دهیم.

```
data_matrix = np.vstack((Normal_samples, Fault1_samples, Fault2_samples, Fault3_samples))
```

```
array([[ 0.05945538,  0.08615815,  0.08845292, ...,  0.05528308,
         0.00688431, -0.04464369],
       [-0.00333785, -0.00834462,  0.01001354, ..., -0.11807631,
        -0.09074769, -0.04547815],
       [ 0.06091569,  0.12245723,  0.15103754, ..., -0.01272554,
         0.02628554,  0.03066646],
       ...,
       [ 0.55146727,  0.28588583, -0.53278723, ..., -2.12180888,
        -1.6775488 ,  3.00098902],
       [ 1.7344011 , -2.50596796, -1.06801098, ..., -0.03289311,
         0.37644341,  0.07756277],
       [ 0.84628703, -1.87328313, -0.11654721, ..., -0.05116707,
         0.14375509,  0.16162295]])
```

شکل ۲۳ ماتریس تشکیل شده برای ۴ کلاس

```
data_matrix.shape
```

```
(400, 200)
```

شکل ۲۴ ابعاد ماتریس ایجاد شده

برای داده‌های Normal برچسب صفر، داده‌های عیب کلاس IR007_0 برچسب یک، داده‌های عیب کلاس B007_0 برچسب دو و داده‌های عیب کلاس OR007@6_0 برچسب سه اطلاق می‌شود.

```
Normal_labels = np.zeros((num_samples, 1))
Fault1_labels = np.ones((num_samples, 1))
```

```
Fault2_labels = 2*np.ones((num_samples, 1))
Fault3_labels = 3*np.ones((num_samples, 1))
```

یک ماتریس از برچسب‌ها با استفاده از دستور زیر ایجاد می‌کنیم و در نهایت ماتریس برچسب و داده‌ها را بهم متصل می‌کنیم. ابعاد هر دو را نیز نمایش می‌دهیم.

```
labels = np.vstack((Normal_labels, Fault1_labels, Fault2_labels, Fault3_labels))
labels.shape
```

```
(400, 1)
```

شکل ۲۵ ابعاد ماتریس برچسب

```
main_matrix = np.hstack((data_matrix, labels))
main_matrix.shape
```

```
array([[ 0.05945538,  0.08615815,  0.08845292, ...,  0.00688431,
        -0.04464369,  0.        ],
       [-0.00333785, -0.00834462,  0.01001354, ..., -0.09074769,
        -0.04547815,  0.        ],
       [ 0.06091569,  0.12245723,  0.15103754, ...,  0.02628554,
         0.03066646,  0.        ],
       ...,
       [ 0.55146727,  0.28588583, -0.53278723, ..., -1.6775488 ,
         3.00098902,  3.        ],
       [ 1.7344011 , -2.50596796, -1.06801098, ...,  0.37644341,
         0.07756277,  3.        ],
       [ 0.84628703, -1.87328313, -0.11654721, ...,  0.14375509,
         0.16162295,  3.        ]])
```

شکل ۲۶ ماتریس تشکیل شده از ماتریس‌های داده‌ها و برچسب

```
(400, 201)
```

شکل ۲۷ ابعاد ماتریس تشکیل شده از ماتریس‌های داده‌ها و برچسب

سپس طبق صورت سوال ۹ ویژگی را از ماتریس ساخته شده در گام قبل استخراج می‌کنیم.

```
std_dev_values = np.std(data_matrix, axis=1)
peak_values = np.max(np.abs(data_matrix),axis=1)
crest_factor_values=np.max(np.abs(data_matrix),axis=1)/np.sqrt(np.mean(data_matrix**2, axis=1))
peak_to_peak_values = np.max(data_matrix, axis=1)-np.min(data_matrix, axis=1)
shape_factor_values=np.sqrt(np.mean(data_matrix**2,axis=1))/np.mean(np.abs(data_matrix), axis=1)
mean_values = np.mean(data_matrix, axis=1)
rms_values = np.sqrt(np.mean(data_matrix**2, axis=1))
```

absolute_mean_values = np.mean(np.abs(data_matrix), axis=1)

impulse_factor_values=np.abs(np.max(data_matrix,axis=1))/np.mean(np.abs(data_matrix), axis=1)

سپس ماتریس ویژگی را به کمک دستور زیر ایجاد می‌کنیم.

features=np.column_stack((std_dev_values,peak_values,crest_factor_values,peak_to_peak_values,shape_factor_values,mean_values,absolute_mean_values,rms_values,impulse_factor_values))

با کمک دستور زیر ماتریس ایجاد شده در بالا را در قالب یک دیتافرم ایجاد می‌کنیم.

	Standard Deviation	Peak	Crest Factor	Peak to Peak	Shape Factor	Mean	Absolute Mean	RMS	Impulse Factor
0	0.081528	0.201940	2.429178	0.402419	1.214257	0.016246	0.068462	0.083131	2.949646
1	0.069435	0.147700	2.084562	0.281214	1.212155	0.014112	0.058453	0.070854	2.526812
2	0.071450	0.175028	2.393741	0.279545	1.188691	0.015534	0.061512	0.073119	2.845418
3	0.070227	0.174194	2.391050	0.347762	1.224943	0.019382	0.059474	0.072852	2.928900
4	0.062739	0.204234	3.253002	0.354020	1.277389	0.002357	0.049150	0.062783	3.047538
...	...	...	...	...	...	...	...	...	...
395	0.735327	2.828402	3.846041	5.639748	1.645999	0.010767	0.446784	0.735406	6.292406
396	0.727805	2.677337	3.677452	5.248279	1.624311	0.018558	0.448215	0.728041	5.735952
397	0.564698	3.000989	5.309943	5.130107	1.771560	0.022954	0.319021	0.565164	9.406883
398	0.707229	2.957944	4.181167	5.532541	1.567473	0.017460	0.451328	0.707444	6.553867
399	0.562784	1.976023	3.507418	3.849306	1.577237	0.025996	0.357197	0.563384	5.532029

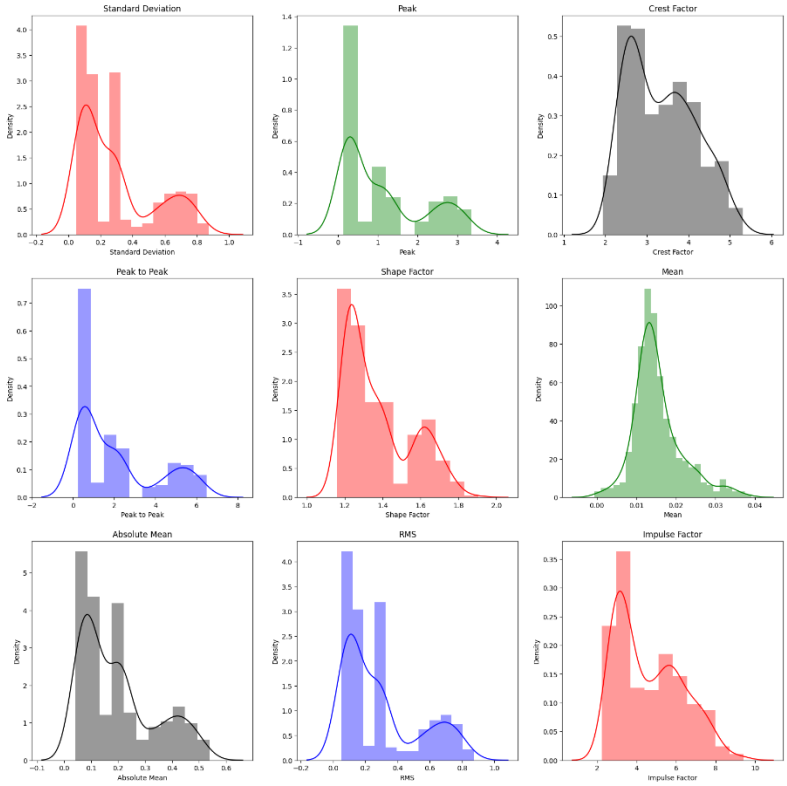
400 rows × 9 columns

شکل ۲۸ دیتافرم ایجاد شده برای ویژگی‌ها

با استفاده از دستورات زیر یک توزیع از نحوه پراکندگی ویژگی‌های ایجاد شده نمایش می‌دهیم.

```
# Create a 3x3 grid of subplots for various numerical variables
plt.figure(figsize=(20, 20))
plt.subplot(3,3,1)
sns.distplot(Data['Standard Deviation'], color="red").set_title('Standard Deviation')
plt.subplot(3,3,2)
sns.distplot(Data['Peak'], color="green").set_title('Peak')
plt.subplot(3,3,3)
sns.distplot(Data['Crest Factor'], color="black").set_title('Crest Factor')
plt.subplot(3,3,4)
sns.distplot(Data['Peak to Peak'], color="blue").set_title('Peak to Peak')
plt.subplot(3,3,5)
sns.distplot(Data['Shape Factor'], color="red").set_title('Shape Factor')
```

```
plt.subplot(3,3,6)
sns.distplot(Data['Mean'], color="green").set_title('Mean')
plt.subplot(3,3,7)
sns.distplot(Data['Absolute Mean'], color="black").set_title('Absolute Mean')
plt.subplot(3,3,8)
sns.distplot(Data['RMS'], color="blue").set_title('RMS')
plt.subplot(3,3,9)
sns.distplot(Data['Impulse Factor'], color="red").set_title('Impulse Factor')
```



شکل ۲۹ نمایش نحوه پراکندگی ویژگی‌های تولید شده

سپس با استفاده از دستورات زیر ماتریس کلی را برش می‌زنیم.

```
Labeled_Data=np.hstack((Data, labels))
np.random.seed(76)
np.random.shuffle(Labeled_Data)
Labeled_Data
```

با استفاده از دستور زیر نیز دیتافرم مربوط به داده‌های برش‌زده را تشکیل می‌دهیم.

	Standard Deviation	Peak	Crest Factor	Peak to Peak	Shape Factor	Mean	Absolute Mean	RMS	Impulse Factor	Target
0	0.740394	3.143120	4.244306	6.160758	1.759954	0.015206	0.420778	0.740550	7.171569	3.0
1	0.287579	1.005473	3.489340	1.969688	1.331168	0.018220	0.216468	0.288156	4.644898	1.0
2	0.314296	1.152802	3.665807	2.125464	1.421538	0.010595	0.221221	0.314474	5.211083	1.0
3	0.304282	1.253187	4.112474	2.212529	1.406439	0.016480	0.216667	0.304728	5.783944	1.0
4	0.599815	1.985363	3.305159	3.905753	1.596989	0.032347	0.376137	0.600686	5.105561	3.0
...	...	...	...	...	...	...	...	...	...	...
395	0.055275	0.144362	2.594539	0.281839	1.251121	0.006365	0.044473	0.055641	3.091284	0.0
396	0.140868	0.422494	2.992044	0.797881	1.263360	0.009754	0.111770	0.141206	3.780029	2.0
397	0.564649	1.958562	3.462364	3.853367	1.585101	0.034000	0.356868	0.565672	5.309543	3.0
398	0.620611	3.081800	4.964288	5.988171	1.663794	0.015072	0.373120	0.620794	8.259552	3.0
399	0.291666	1.194061	4.086810	2.108733	1.350974	0.017220	0.216269	0.292174	5.521173	1.0

400 rows × 10 columns

شکل ۳۰ دیتافرم ایجاد شده برای ویژگی‌ها برش زده شده

ابعاد دیتافرم ایجاد شده را به کمک دستور زیر محاسبه می‌کنیم.

```
Labeled_Data.shape
```

```
(400, 10)
```

شکل ۳۱ ابعاد دیتافرم ایجاد شده

با استفاده از دستور زیر همه ستون‌ها به جز ستون آخر را ورودی، ستون آخر را به عنوان خروجی در نظر می‌گیریم، ابعاد آن را نیز مشخص می‌کنیم.

```
X=np.array(Labeled_Data.loc[:,Labeled_Data.columns!='Target'])
y=np.array(Labeled_Data.loc[:,Labeled_Data.columns=='Target'])
print(X.shape, y.shape)
```

```
(400, 9) (400, 1)
```

شکل ۳۲ ابعاد ماتریس ورودی و خروجی

حال با استفاده از دستورات زیر داده‌های آموزش و آزمون را انتخاب می‌کنیم. در ابتدا کتابخان‌های مورد نیاز در ادامه را می‌افزاییم. در هنگام طراحی شبکه ۰.۲ از داده‌های آموزش را به عنوان داده‌های ارزیابی انتخاب می‌کنیم. داده‌های ارزیابی به این منظور سنجش عملکرد سیتم ما استفاده می‌شود و شبکه تا قبل از آن مشاهده ننموده است.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
```

با استفاده از تابع درون کتابخانه sklearn داده‌ها را پردازش می‌کنیم. نکته‌ی مهم این است که ابتدا باید این scaler را fit کنیم و سپس transform بر روی داده‌ها بزنیم. در واقع scaler نباید داده‌های آزمون را ببیند و فقط باید بر روی داده‌های آموزش fit کردن صورت گیرد. برای نمایش بهتر، تعداد ۵ عدد از داده‌ها را قبل و بعد از sacler را نمایش می‌دهیم؛ که نشان از موفقیت‌آمیز بودن scaler ما می‌باشد. ابعاد ماتریس‌های آموزش و آزمون نیز نمایش داده شده است.

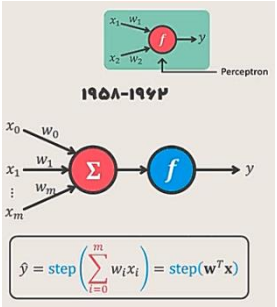
```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
x_train = x_train_scaled
x_test = x_test_scaled
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(320, 9)
(80, 9)
(320, 1)
(80, 1)
```

شکل ۳۳ ابعاد ماتریس آموزش و آزمون

۲-۲-بخش دوم

در شکل زیر می‌توانیم مدل ساده نورون طراحی شده را مشاهده می‌کنیم. تابع فعال‌ساز در ابتدا به صورت پله بود. در واقع مجموع ورودی‌های دارای ضریب را از یک تابع فعالساز پله عبور می‌دهیم. در شبکه عصبی برای بهینه‌سازی نیاز به مشتق‌گیری است ولی تابع پله مشتق‌پذیر نیست.



شکل ۳۴ پیاده‌سازی ریاضی نورون

متریک accuracy برای بررسی مقادیر که به درستی تشخیص داده شده‌اند، استفاده می‌گردد. در ابتدا در صورتی از مقادیر خروجی تولید شده کمتر از آستانه باشد عدد صفر، و اگر بزرگتر باشد مقدار یک را باز می‌گرداند. سپس تعداد مقادیر که به درستی شناسایی شده است را تقسیم بر تعداد کل نمونه‌ها می‌کنیم.

برای ساخت شبکه عصبی در ابتدا کتابخانه‌هاب مورد نیاز را می‌افزاییم.

```
import random
import tensorflow as tf
from tensorflow import keras
from keras import preprocessing
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

سپس درون یک تابع یک شبکه را تعریف می‌کنیم. برای اینکه مقادیر وزن تولید شده در ابتدا ثابت باشد سه خط اول کد نوشته شده است. سپس یک شی از آن را ایجاد می‌کنیم. با استفاده از تابع Dense لایه‌ها را می‌فزاییم. لایه اول دارای ۱۳ نورون به همراه تابع فعالساز ReLU می‌باشد. در لایه دوم که لایه خروجی می‌باشد که به تعداد کلاس‌ها نورون قرار گرفته است و از تابع فعالساز softmax استفاده شده است.

```
def create_model():
  random.seed(76)
  np.random.seed(76)
  tf.random.set_seed(76)
  model = Sequential()
  # Add a hidden layer with 13 neurons and ReLU activation function
  model.add(Dense(13, activation='relu', input_shape=(x_train.shape[1],)))
  # Use 'softmax' for multi-class classification
  model.add(Dense(4, activation='softmax'))
  return model
```

با استفاده از دستورات زیر یک شی از تابع تعریف شده در بالا ایجاد می‌کنیم. با استفاده از دستور summary می‌توانیمتعداد پارامترها را نماش دهیم. در نهایت نیز مقادیر پارامترهای اولیه نماش داده شده است.

```
# Create the model
model_1 = create_model()
model_1.summary()
# Get and print the initial weights
initial_weights = model_1.get_weights()
for layer_num, layer_weights in enumerate(initial_weights):
    print(f"Layer {layer_num + 1} weights:\n{layer_weights}\n")
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 13)                130

 dense_3 (Dense)             (None, 4)                 56

=================================================================
Total params: 186 (744.00 Byte)
Trainable params: 186 (744.00 Byte)
Non-trainable params: 0 (0.00 Byte)
```

شکل ۳۵ پارامترهای شبکه ایجادشده

برای استفاده از تابع هزینه categorical_crossentropy لازم است که برچسب‌ها به کمک دستورات زیر نوع one-hot تبدیل شوند.

```
y_train1 = to_categorical(y_train, num_classes=4)
y_test1 = to_categorical(y_test, num_classes=4)
```

با استفاده از دستورات زیر نوع بهینه‌ساز، تابع هزینه و متریک را مشخص می‌کنیم. سپس شبکه را آموزش می‌دهیم.

```
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history1 = model_1.fit(x_train, y_train1, validation_split=0.2, epochs=60, batch_size=10)
```

با استفاده از دستور زیر شبکه آموزش دیده شده را ارزیابی می‌کنیم.

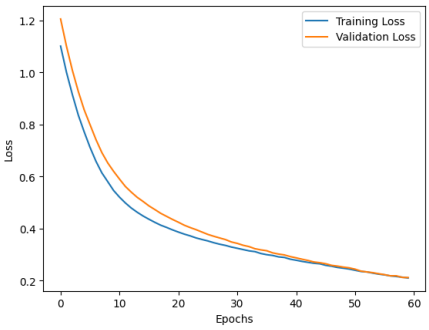
```
loss1 = model_1.evaluate(x_test , y_test1)
```

```
loss: 0.1740 - accuracy: 1.0000
```

شکل ۳۶ ارزیابی داده‌های آزمون

با استفاده از دستورات زیر نمودار مربوط به مقادیر هزینه را رسم می‌کنیم. همانطور که مشاهده می‌شود نمودار هزینه کاهشی بوده است و سیستم دچار overfitting یا underfitting نشود.

```
plt.plot(history1.history['loss'], label='train')   # Training loss
plt.plot(history1.history['val_loss'], label='val')  # Validation loss
plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```



شکل ۳۷ نمودار مقادیر تابع هزینه

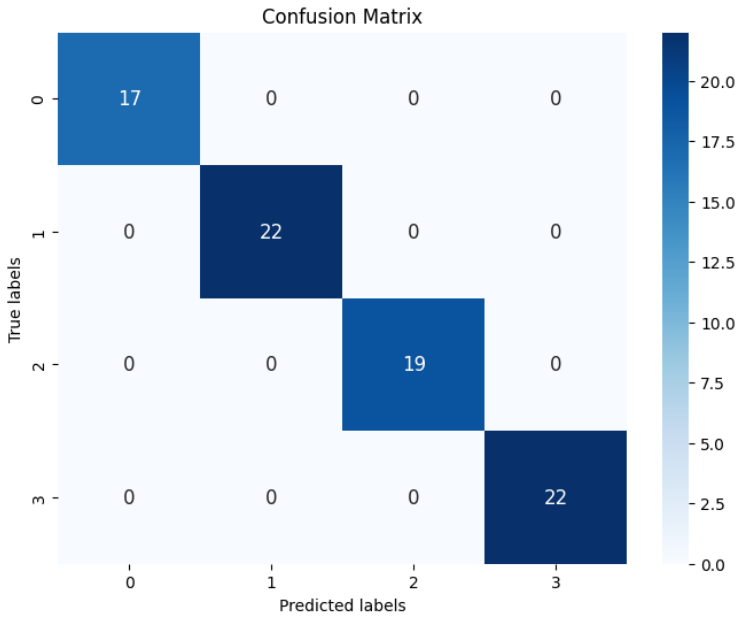
با استفاده از دستورات زیر میزان متریک accuracy را نشان می‌دهیم. از آنجایی که مشاهده می‌شود میزان دقت بالا می‌باشد.

```
# Plot training & validation accuracy values
plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

شکل ۳۸ نمودار accuracy

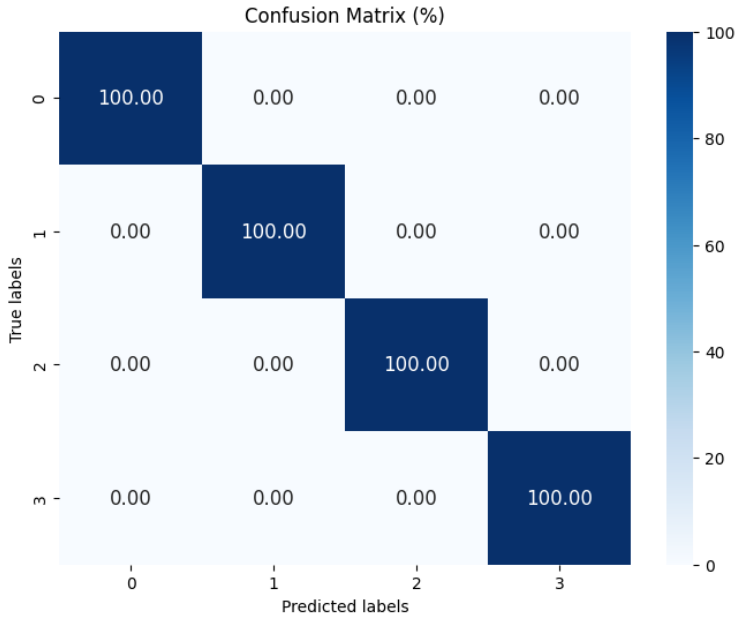
سپس با استفاده از دستور زیر مقادیر آزمایش را محاسبه می‌کنیم و سپس ماتریس درهم‌ریختگی را هم به صورت تعداد و هم دصدی رسم می‌کنیم. همانطور که مشاهده می‌شود مقادیر به درستی کلاس‌بندی شده است.

```
y1_pred = model_1.predict(x_test)
y1_pred_classes = np.argmax(y1_pred, axis=1)
# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y1_pred_classes)
```



شکل ۳۹ ماتریس درهم‌ریختگی

```
# Calculating percentages for each cell
cf_matrix_percent = cf_matrix.astype('float') / cf_matrix.sum(axis=1)[:, np.newaxis] * 100
```



شکل ۴۰ ماتریس درهم‌ریختگی

با استفاده از classification_report می‌توانیم یک گزارش از نحوه عملکرد ماشین دریافت کرد.

```
classification_report(y_test, y1_pred_classes)
```

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

         0.0       1.00      1.00      1.00        17
         1.0       1.00      1.00      1.00        22
         2.0       1.00      1.00      1.00        19
         3.0       1.00      1.00      1.00        22

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80
```

شکل ۴۱ گزارش مربوط به classification_report

۲-۳-بخش سوم

انتخاب بهینه‌ساز مناسب برای عملکرد مدل بسیار مهم است. در اینجا چند بهینه‌ساز رایج آنها آورده شده است.

SGD (Stochastic Gradient Descent): یک رویکرد ساده و کارآمد که پارامترهای مدل را براساس گرادیان تابع اتلاف به روز می‌کند. برای مجموعه داده‌های بزرگ مناسب است و از برازش بیش از حد جلوگیری می‌کند. از معایب آن می‌توان به این مورد اشاره کرد که ممکن است به آرامی همگرا شوند و در حداقل‌های محلی گیر کنند.

Adam (Adaptive Moment Estimation): ترکیبی از مزایای AdaGrad و RMSProp، تطبیق نرخ یادگیری برای هر پارامتر می‌باشد. در عمل، حتی برای مدل‌های پیچیده، به خوبی کار می‌کند. یکی از معایب آن می‌تواند حافظه فشرده باشد. اغلب برای اکثر وظایف یادگیری عمیق توصیه می‌شود.

RMSProp: نرخ یادگیری را برای هر پارامتر تنظیم می‌کند و برای اهداف غیر ثابت به خوبی کار می‌کند. برای شبکه های عصبی بازگشتی کارآمد است. عیب این روش حساس به هایپرپارامترها می‌باشد.

Adagrad: به صورت تطبیقی نرخ یادگیری را برای هر پارامتر تنظیم می‌کند. برای داده‌ها و ویژگی‌های پراکنده خوب است.اما در طول زمان می‌تواند منجر به نرخ یادگیری بسیار کمی شود.

تابع اتلاف برای محاسبه تفاوت بین خروجی تولید شده و خروجی واقعی استفاده می‌شود. در keras، انتخاب تابع اتلاف مناسب برای وظایف طبقه‌بندی برای آموزش مدل‌های موثر بسیار مهم است. در اینجا متداول‌ترین توابع ضرر برای سناریوهای طبقه بندی مختلف وجود دارد:

Categorical Crossentropy: در طبقه‌بندی چند طبقه با برچسب‌های کدگذاری شده یک‌طرفه استفاده می‌گردد. تفاوت بین توزیع برچسب واقعی و توزیع برچسب پیش‌بینی شده را اندازه‌گیری می‌کند.

Sparse Categorical Crossentropy: طبقه‌بندی چند کلاسه با برچسب‌های عدد صحیح مورد استفاده قرار می‌گیرد. مشابه متقاطع طبقه‌ای است، اما زمانی استفاده می‌شود که برچسب‌ها اعداد صحیح هستند، به جای کدگذاری یک‌طرفه.

Binary Crossentropy: برای طبقه‌بندی باینری استفاده می‌شود. تفاوت بین برچسب‌های باینری واقعی و احتمالات پیش‌بینی شده را اندازه گیری می‌کند.

Kullback-Leibler Divergence (KL Divergence): هنگام مقایسه توزیع‌های احتمال استفاده می‌شود. نحوه واگرایی یک توزیع احتمال از توزیع احتمال دوم مورد انتظار را اندازه‌گیری می‌کند.

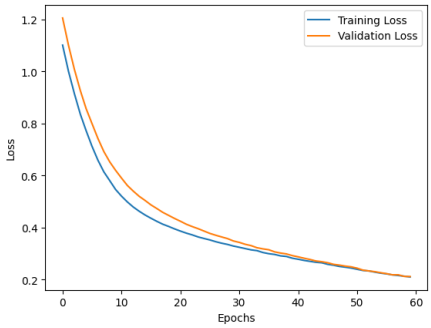
Poisson: شمارش داده‌ها یا وظایف مبتنی بر توزیع پواسون استفاده می‌گردد. اتلاف را براساس توزیع پواسون بین تعداد واقعی و شمارش پیش‌بینی شده اندازه‌گیری می‌کند.

در این بخش از تابع SparseCategoricalCrossentropy استفاده می‌کنیم. این تابع از دست دادن برای مسائل طبقه‌بندی چند کلاسه که برچسب‌های هدف اعداد صحیح هستند، مناسب است. این معادل Categorical Crossentropy است اما با برچسب‌های عدد صحیح به جای برچسب‌های کدگذاری شده one-hot کار می‌کند که می‌تواند کارآمدتر باشد. مدل شامل یک لایه Flatten برای تغییر شکل داده‌های ورودی، به دنبال

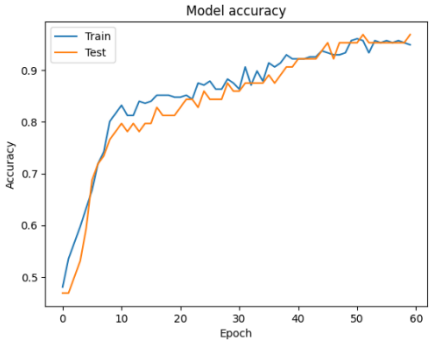
لایه‌های متراکم با فعال سازی ReLU، و یک لایه متراکم نهایی با فعال سازی softmax برای احتمالات کلاس خروجی است. آموزش با استفاده از روش fit و ارزشیابی با روش evaluate انجام می‌شود.

در زیر مدل را به صورت زیر تغییر می‌دهیم و نتایج را نشان می‌دهیم. همانطور که مشاهده می‌شود در این حالت پاخ به مانند قبل می‌باشد و تغییر چندانی نداشته است.

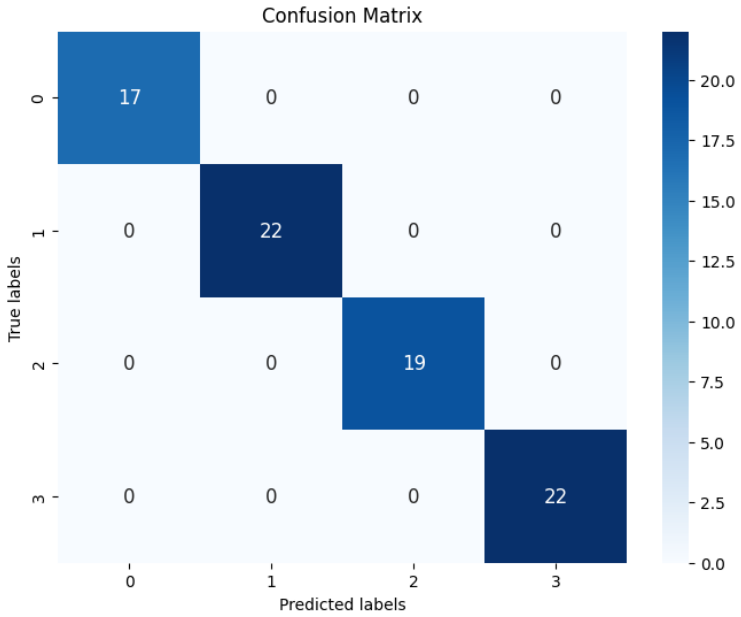
model_2.compile(optimizer='adam',loss='SparseCategoricalCrossentropy',metrics=['accuracy'])



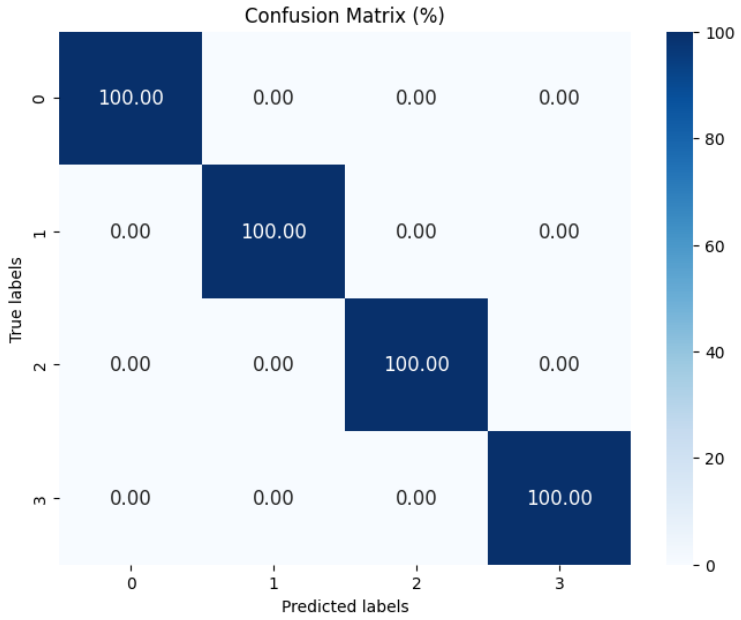
شکل ۴۲ نمودار مقادیر تابع هزینه



شکل ۴۳ نمودار accuracy



شکل ۴۴ ماتریس درهم‌ریختگی



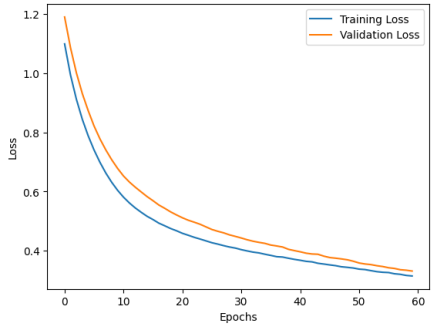
شکل ۴۵ ماتریس درهم‌ریختگی

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

         0.0       1.00      1.00      1.00        17
         1.0       1.00      1.00      1.00        22
         2.0       1.00      1.00      1.00        19
         3.0       1.00      1.00      1.00        22

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80
```

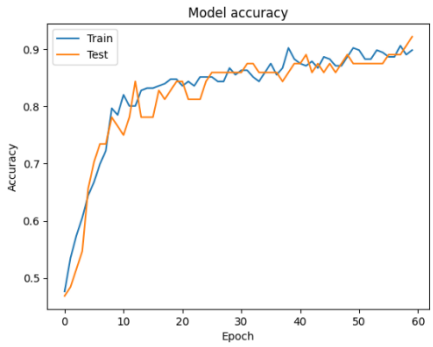
شکل ۴۶ گزارش مربوط به classification_report

در اینجا مدل را به صورت زیر تغییر می‌دهیم. همانطور که مشاهده می‌شود با تغییر نوع بهینه‌ساز عملکرد سیستم کاهش یافته است و داده‌های کلاس ۲ به درستی کلاس‌بندی نشده‌اند و به کلاس ۰ تعلق گرفته‌اند.

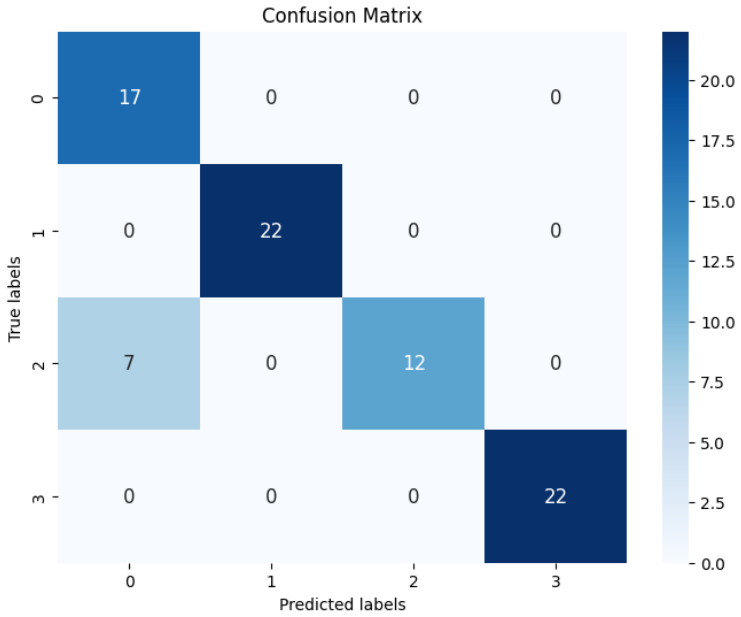
```
model_3.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['accuracy'])
```



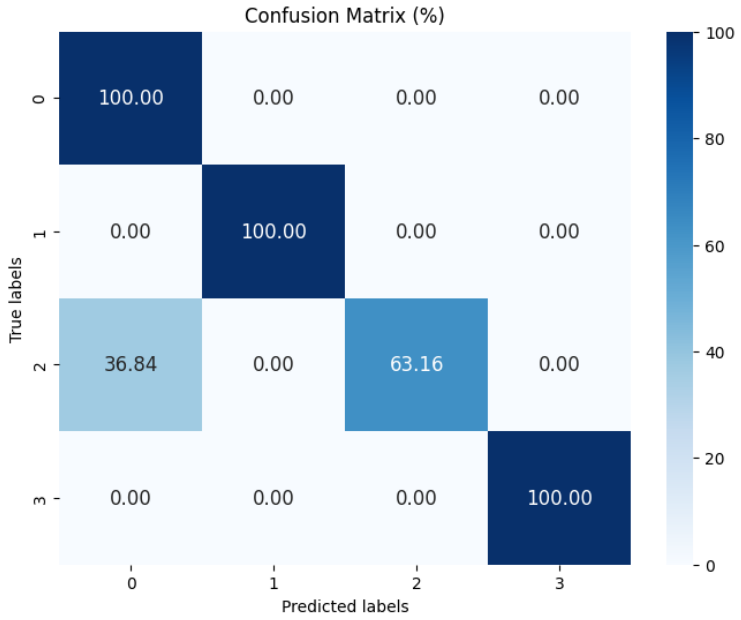
شکل ۴۷ نمودار مقادیر تابع هزینه



شکل ۴۸ نمودار accuracy



شکل ۴۹ ماتریس درهم‌ریختگی



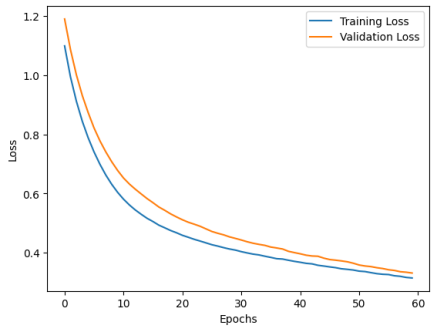
شکل ۵۰ ماتریس درهم‌ریختگی

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

         0.0       0.71      1.00      0.83        17
         1.0       1.00      1.00      1.00        22
         2.0       1.00      0.63      0.77        19
         3.0       1.00      1.00      1.00        22

    accuracy                           0.91        80
   macro avg       0.93      0.91      0.90        80
weighted avg       0.94      0.91      0.91        80
```

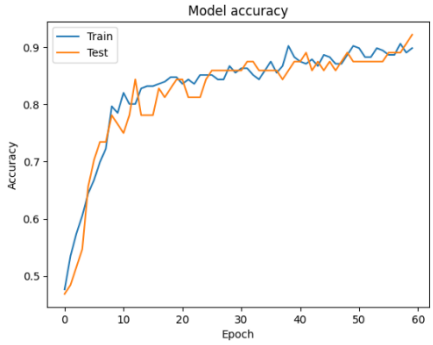
شکل ۵۱ گزارش مربوط به classification_report

و در آخر هم مدل سیستم را به صورت زیر در نظر می‌گیریم. همانطور که مشاهده می‌شود تغییر نو تابع اتلاف در این دو حالت تاثیر چندانی نداشته است.

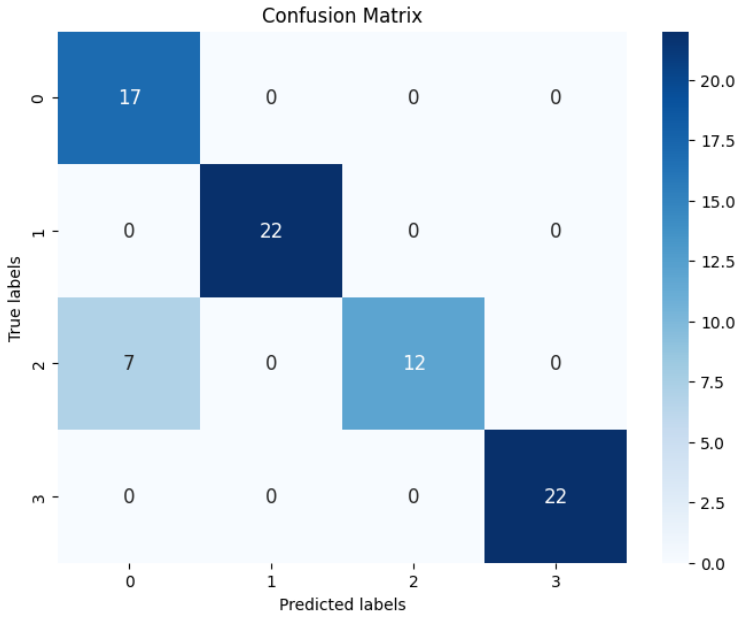
```
model_4.compile(optimizer='SGD',loss='SparseCategoricalCrossentropy',metrics=['accuracy'])
```



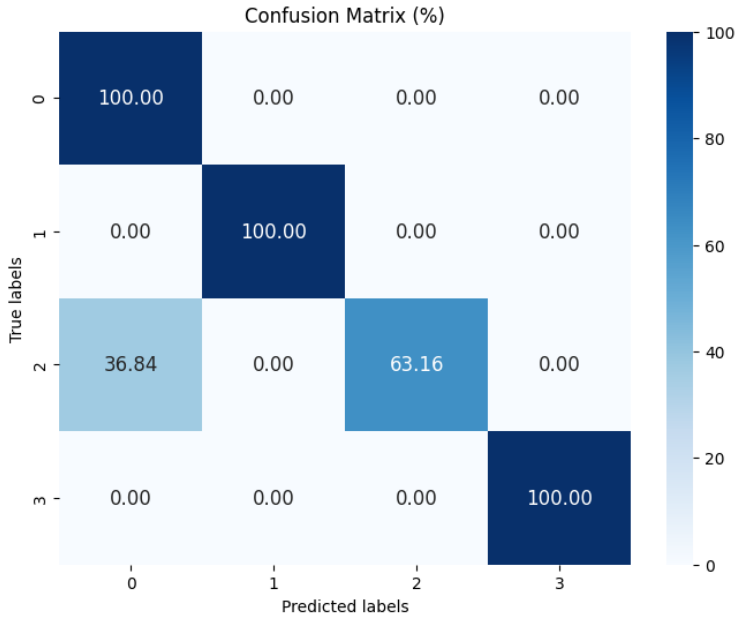
شکل ۵۲ نمودار مقادیر تابع هزینه



شکل ۵۳ نمودار accuracy



شکل ۵۴ ماتریس درهم‌ریختگی



شکل ۵۵ ماتریس درهم‌ریختگی

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

         0.0       0.71      1.00      0.83        17
         1.0       1.00      1.00      1.00        22
         2.0       1.00      0.63      0.77        19
         3.0       1.00      1.00      1.00        22

    accuracy                           0.91        80
   macro avg       0.93      0.91      0.90        80
weighted avg       0.94      0.91      0.91        80
```

شکل ۵۶ گزارش مربوط به classification_report

۲-۴-بخش چهارم

K-Fold Cross-Validation یک روش قوی برای ارزیابی عملکرد مدل‌های یادگیری ماشین، از جمله مدل‌های ساخته شده با Keras است. مجموعه داده به k زیر مجموعه یا folds تقسیم می‌شود. مدل k بار آموزش و تست شده است. در هر تکرار، یک فولد متفاوت به عنوان مجموعه تست استفاده می‌شود، در حالی که k-1 فولد باقیمانده برای آموزش استفاده می‌شود. معیارهای عملکرد (به عنوان مثال، accuracy، precision) از هر تکرار برای ارائه تخمین قابل اعتمادتری از عملکرد مدل، میانگین می‌شوند. از مزایای این روش می‌توان به قابلیت اطمینان بهبود یافته، به معنی واریانس برآورد عملکرد را با میانگین‌گیری نتایج حاصل از تقسیم‌بندی‌های چندگانه آموزش-آزمون کاهش می‌دهد. از همه داده‌ها استفاده می‌کند به این معنی که هر نقطه داده هم برای آموزش و هم برای آزمایش استفاده می‌شود و استفاده از مجموعه داده را به حداکثر می-رساند. این رویکرد کمک می‌کند تا اطمینان حاصل شود که عملکرد مدل سازگار است و به یک تقسیم آموزش-آزمون خاص وابسته نیست.

طبقه‌بندی Stratified K-Fold Cross-Validation بهبود تکنیک سنتی K-Fold Cross-Validation است که به ویژه برای کارهای طبقه‌بندی مفید است. مانند K-Fold Cross-Validation معمولی، مجموعه داده به k زیرمجموعه یا فولد تقسیم می‌شود. این مدل k بار آموزش و ارزیابی می‌شود، هر بار از یک فولد متفاوت به عنوان مجموعه تست و از k-1 فولد باقی مانده به عنوان مجموعه آموزشی استفاده می‌شود. تفاوت اصلی این است که K-Fold طبقه‌بندی شده تضمین می‌کند که هر فولد دارای توزیع مشابهی از کلاس‌ها با مجموعه داده اصلی است. این بدان معنی است که نسبت هر برچسب کلاس در هر فولد تقریباً با کل مجموعه داده یکسان است. از مزایای این روش می‌توان به تخمین‌های عملکرد بهبود یافته اشاره نمود؛ K-Fold طبقه‌بندی شده با حفظ توزیع کلاس در بین فولدها، تخمین‌های عملکرد قابل اعتمادتر و بی طرفانه‌تری را ارائه می‌دهد، به خصوص زمانی که با مجموعه داده‌های نامتعادل سروکار داریم. مدل‌های آموزش دیده و اعتبارسنجی‌شده با فولدهای طبقه‌بندی شده احتمالاً بهتر به داده‌های دیده نشده تعمیم می‌دهند.

در ابتدا کتابخانه‌های مورد نیاز را می‌افزاییم.

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
```

سپس یک شی از kfold ایجاد می‌کنیم و به تعداد ۵ بار داده‌های تست را جابه‌جا مب‌کنبم. در زیر میانگین و مقدار accuracy در هر تکرار را مشاهده می‌کنید (به دلیل حجم بالا از آوردن اطلاعات هر تکرار در گزارش خودداری شده است).

```
kf = KFold(n_splits=5)
scores = []
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]
```

```
Mean accuracy: 0.97 ± 0.03
[1.     0.9375 0.9625 0.925  1.    ]
```

شکل ۵۷ میانگین و مقدار Accuracy در هر تکرار

۳–سوال سوم

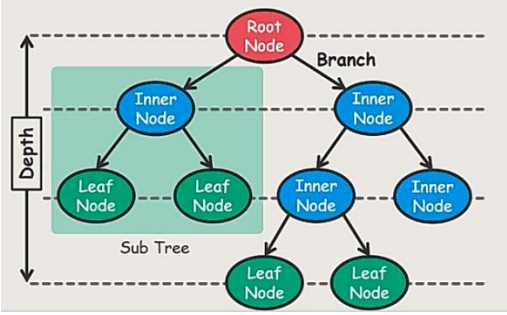
تصور کنید که شما یک محقق پزشکی هستید که داده‌ها را برای یک مطالعه جمع‌آوری می‌کنید. شما داده‌هایی را در مورد مجموعه‌ای از بیماران جمع‌آوری کرده‌اید که همه آنها از یک بیماری رنج می‌بردند. در طول دوره درمان، هر بیمار به یکی از ۵ دارو، داروی A، داروی B، داروی c، داروی x و y پاسخ داد.

بخشی از کار شما این است که مدلی بسازید تا بفهمید کدام دارو ممکن است برای یک بیمار آینده با همان بیماری مناسب باشد. ویژگی‌های این مجموعه داده‌ها سن، جنس، فشار خون و کلسترول بیماران است و هدف دارویی است که هر بیمار به آن پاسخ داده است.

این یک نمونه از طبقه‌بندی‌کننده چندکلاسه است و می‌توانید از بخش آموزشی مجموعه داده برای ساختن درخت تصمیم استفاده کنید و سپس از آن برای پیش‌بینی کلاس یک بیمار ناشناخته یا تجویز دارو برای یک بیمار جدید استفاده کنید.

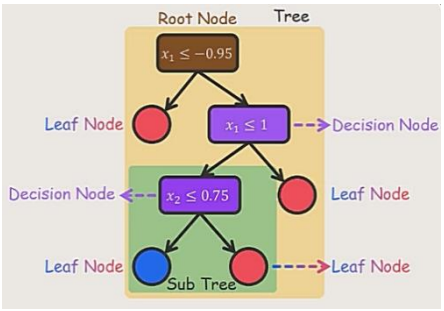
۳-۱-بخش اول

در شکل زیر می‌توانیم ساختار کلی درخت را مشاهده کنیم. عمق درخت می‌تواند زیاد و یا کم باشد. به یک زیرمجموعه از درخت نیز یک Sub Tree می‌گویند.



شکل ۵۸ ساختار کلی درخت

درخت تصمیم شامل ساختاری شامل گره یا node، edge یا شاخه می‌باشد که در شکل زیر می‌توانید مشاهده کنید. تفاوت آن در نقاط گره می‌باشند که بعضی از آن‌ها به صورت دایره و برخی به صورت مستطیل می‌باشند. در گره‌های مستطیل یکسری شرط نوشته شده است، که همان تصمیم‌های ما را مشخص می‌کنند؛ به این گره‌ها، گره‌های تصمیم‌گیری می‌گویند. گره‌های دایره‌ای در واقع برگ‌ها یا گره‌های پیش‌بینی هستند و خروجی را مشخص می‌کنند. گره ابتدایی را نیز Root Node می‌گویند. xها در واقع ورودی‌های ما، که x_i به یکی از ویژگی‌های آن اشاره می‌کند. به نوعی دیگر می‌توان به این صورت بیان کرد که درخت تصمیم مجموعه-ای از دستورات if و else می‌باشد.

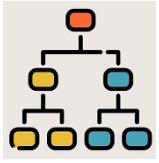


شکل ۵۹ ساختار کلی درخت تصمیم

حال در رابطه با اینکه یادگیری ماشین چگونه درخت تصمیم را تشکیل می‌دهد، چندین سوال وجود دارد. تقسیم‌بندی ویژگی چگونه در درخت تصمیم صورت می‌گیرد؟! ترتیب انتخاب ویژگی چگونه صورت می‌گیرد؟! مقادیر آستانه در مثال‌های اعدادی چگونه صورت می‌گیرد؟ چگونه عمق درخت تصمیم را انتخاب می‌کنیم؟!

این موارد با الگوریتم‌های درخت تصمیم در یادگیری ماشین به این سوالات پاسخ می‌دهیم که معروفترین آنها میتوان به ID3، C4.5 و CART اشاره کرد. C4.5 در واقع یک نوع توسیع یافته ID3 می‌باشد. اکنون درخت تصمیم CART بسیار پرکاربرد می‌باشد، که هم در مسائل رگرسیون و هم کلاس‌بندی مورد استفاده قرار می‌گیرند.

ID3 یکی از اولین الگوریتم‌های مبتنی بر درخت در یادگیری ماشین می‌باشد. این کلمه مخفف Iterative Dichotomiser 3 می‌باشد، که ۳ نشان دهنده‌ی نسخه آن می‌باشد. در واقع به معنی تقسیم ویژگی‌ها به چند گروه به صورت تکراری است. این روش مخصوص دسته‌بندی ویژگی‌های طبقه‌ای است. در واقع به این معنی است که این روش برای روش‌های رگرسیون و دارای ویژگی‌های عددی مورد استفاده قرار نمی‌گیرد.



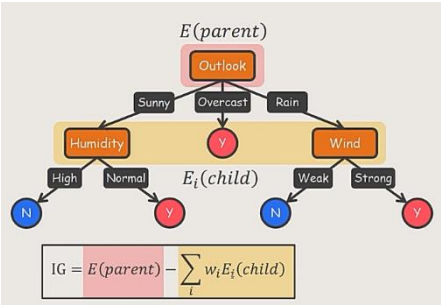
شکل ۶۰ ساختار کلی درخت ID3

این روش دارای چها ویژگی می‌باشد. یک رهیافت بالا به پایین (Top-Down) است. به این معنی است که از Root Node شروع می‌شود به Leaf Root ختم می‌گردد. تقسیم هر گره به تعداد دسته‌های در ویژگی می‌باشد. ویژگی دیگر جست و جوی حریصانه (Greedy Search) می‌باشد. این موضوع مربوط به ترتیب انتخاب ویژگی‌ها می‌باشد؛ در واقع رهیافتی برای انتخاب بهترین گزینه در هر لحظه می‌باشد. که در نتیجه بهترین انتخاب در هر لحظه برای داشتن بهترین نتیجه در کل است. رهیافت بالا به پایین (Top-Down) بدون عقب‌گرد

است. در دو مرحله می‌توانیم الگوریتم جستجوی حریص کردن را تعریف کنیم. اول بررسی تمام گزینه‌های موجود در هر لحظه است. سپس انتخاب بهترین گزینه در حالت ممکن است. لزوماً بهترین گزینه در لحظه منجر به بهترین نتیجه کلی نمی‌شود. و ویژگی آخر Information Gain می‌باشد، که براساس آن انتخاب می-کنیم کدوم ویژگی قرار بگیرد.

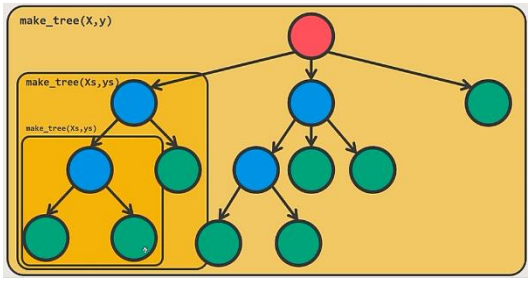
Information Gain در واقع معیاری برای اندازه‌گیری میزان کاهش آنتروپی و کیفیت جداسازی ویژگی می‌باشد. در واقع ما دوست داریم که میزان Information Gain زیاد باشد. رابطه آنتروپی را می‌توان به صورت زیر نوشت. هر چقدر میزان ناخالصی کمتر باشد آنتروپی کمتر می‌گردد و با وجود ناخالصی بیشتر میزان آنتروپی بزرگتر می‌گردد. در واقع می‌توان بیان نمود که کیفیت جداسازی بالا معادل ناخالصی کم و آنتروپی کم می‌باشد.

$$E = -\sum_i p_i \log_2 p_i$$



شکل ۶۱ Information Gain و رابطه مربوط به آن

در یک درخت تصمیم تا یک مسیر به صورت کامل ساخته نشود، به سراغ مسیر بعدی نمی‌زود. نکته دیگر این می‌باشد یک درخت تصمیم دائماً خود را فراخوانی می‌کند.



شکل ۶۲ فراخوانی در درخت تصمیم

برای شروع کار ابتدا باید دیتاست مورد نظر را دانلود کنیم. در ابتدا یک پوشه برای ذخیره سازی داده‌ها ایجاد می‌کند. در ابتدا کتابخانه مدنظر را فراحوانی می‌کنیم. سپس درون حلقه if بررسی می‌شود اگر پوشه‌ای با این نام وجود ندارد، آن را بسازد. و در نهایت به داخل آن پوشه انتقال پیدا می‌کنیم.

```
import os
```

```
# Define the folder name
folder_name = "Data"
# Check if the folder already exists, and create it if not
if not os.path.exists(folder_name):
  os.makedirs(folder_name)
# Commented out IPython magic to ensure Python compatibility.
%cd Data
```

در مرحله بعد برای اینکه دیتاست مدنظر را از سایت kaggle دانلود کنیم نیاز به نصب پکیج مربوط به آن با دستور زیر می‌باشد.

```
!pip install kaggle
```

سپس با استفاده از دستور زیر که در سایت kaggle با زدن بر روی سه نقطه و انتخاب Copy API command قابل یافتن است، دیتاست مدنظر دانلود می‌گردد. و در نهایت فایل zip دانلود شده را از حالت فشرده خارج می‌کنیم و فایل zip را حذف می‌کنیم.

```
!kaggle datasets download -d pablomgomez21/drugs-a-b-c-x-y-for-decision-trees
!unzip \*.zip && rm *.zip
```

جال باید کتابخانه‌های مرد نیاز پایتون را می‌افزاییم.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
%matplotlib inline
```

سپس فایل csv. دانلود شده در گام ابتدایی را با استفاده از دستور زیر می‌خوانیم. آدرس آن را با کلیک راست کردن بر روی آن و انتخاب copy path یافته‌ایم.

```
df = pd.read_csv(' /content/Data/drug200.csv')
```

سپس با استفاده از دستور زیر ۵ داده ابتدایی (در صورت انتخاب کردن تعداد) به همراه عنوان ویژگی‌ها و برچسب (که ستون آخر می‌باشد) را نمایش می‌دهد.

```
df.head()
```

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

شکل ۶۳ نتایج مربوط به head()

با استفاده از دستور زیر اطلاعات کلی در رابطه با این data-frame لازم است را به ما می‌دهد. از آنجایی که داده null موجود نمی‌باشد، نیاز به حذف آن نیز نمی‌باشد.

```
df.info()
print(df.isnull().sum())
```

```
<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
```

شکل ۶۴ نتیج مربوط به info()

```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

شکل ۶۵ نتیج مربوط به isnull()

با استفاده از دستور زیر ابعاد ماتریس داده‌ها را به دست می‌آوریم.

```
df.shape
```

```
(200, 6)
```

شکل ۶۶ ابعاد ماتریس داده‌ها

با استفاده از دستور زیر اطلاعاتی در رابطه با میانگین، واریانس، مینیمم، ماکزیمم و ... مربوط به داده‌های عددی را به ما می‌دهد.

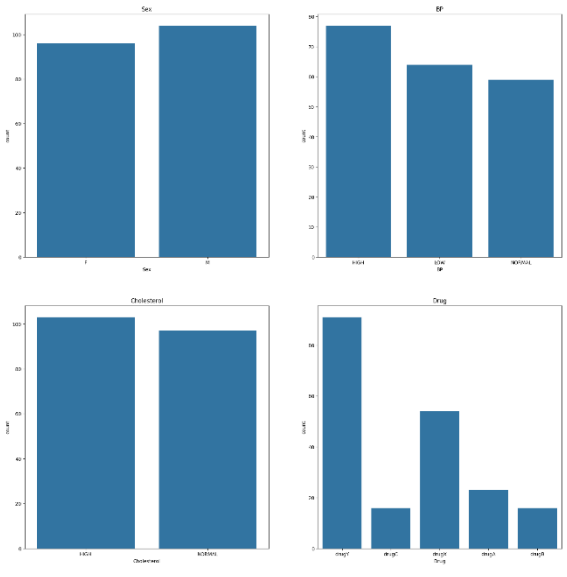
```
df.describe()
```

	Age	Na_to_K
count	200.000000	200.000000
mean	44.315000	16.084485
std	16.544315	7.223956
min	15.000000	6.269000
25%	31.000000	10.445500
50%	45.000000	13.936500
75%	58.000000	19.380000
max	74.000000	38.247000

شکل ۶۷ نتایج مربوط به describe()

تعداد ویژگی‌های غیر عددی را با استفاده از دستورات زیر نمایش می‌دهیم.

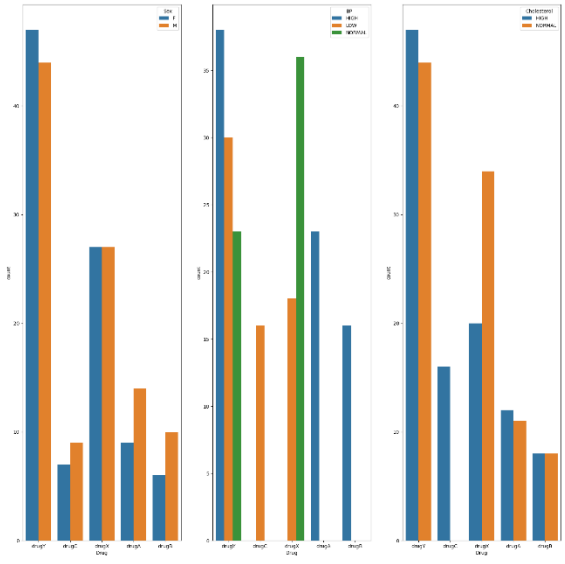
```
plt.figure(figsize=(20, 20))
plt.subplot(2,2,1)
sns.countplot(data=df, x = "Sex").set_title('Sex')
plt.subplot(2,2,2)
sns.countplot(data=df, x = "BP").set_title('BP')
plt.subplot(2,2,3)
sns.countplot(data=df, x = "Cholesterol").set_title('Cholesterol')
plt.subplot(2,2,4)
sns.countplot(data=df, x = "Drug").set_title('Drug')
```



شکل ۶۸ نمایش تعداد ویژگی‌های غیر عددی

با استفاده از دستورات زیر نوع داروی تجویز شده را براساس ویژگی‌های غیر عددی دیگر رسم می‌کنیم.

```
plt.figure(figsize=(20, 20))
plt.subplot(1,3,1)
sns.countplot(data=df, x = "Drug", hue="Sex")
plt.subplot(1,3,2)
sns.countplot(data=df, x = "Drug", hue="BP")
plt.subplot(1,3,3)
sns.countplot(data=df, x = "Drug", hue="Cholesterol")
```



شکل ۶۹ نوع داروی تجویزی براساس سایر ویژگی‌های غیرعددی

با استفاده از دستورات زیر ویژگی‌های غیرعددی را به ویژگی‌های عددی تبدیل می‌کنیم و مجددا عنوان ستون‌ها و داده‌های ۵ ردیف اول را نمایش می‌دهیم.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df["Sex"] = le.fit_transform(df[["Sex"]])
df["BP"] = le.fit_transform(df[["BP"]])
df["Cholesterol"] = le.fit_transform(df[["Cholesterol"]])
df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	0	0	0	25.355	drugY
1	47	1	1	0	13.093	drugC
2	47	1	1	0	10.114	drugC
3	28	0	2	0	7.798	drugX
4	61	0	1	0	18.043	drugY

شکل ۷۰ نتایج مربوط به ()head

Sex: -----> 0 =Female 1 = Male

BP: -----> 0 = High 1 = Low 2 = Normal

Cholesterol: -----> 0 = High 1 = Normal

حال برای ادامه کار باید تعدادی از داده‌ها را به عنوان داده‌های آموزش و تعدادی را به عنوان داده آزمون تقسیم کنیم. به این منظور ابتدا ووردی‌ها و خروجی را با استفاده از دستور زیر مجزا می‌کنیم.

```
X=np.array(df.loc[:,df.columns!='Drug'])
y=np.array(df.loc[:,df.columns=='Drug'])
print('X:', X.shape,'\ny:', y.shape)
```

```
X: (200, 5)
y: (200, 1)
```

شکل ۷۱ ابعاد ماتریس ورودی و خروجی

با استفاده از دستور زیر تعداد داروهای موجوذ در هر دسته از داروها را نمایش می‌دهیم.

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

```
Drug
drugA    23
drugB    16
drugC    16
drugX    54
drugY    91
dtype: int64
```

شکل ۷۲ تعداد داده‌های موجود در هر دسته

در زیر به چندین روش جداسازی داده‌ها اشاره شده است.

Holdout: مجموعه داده را به دو بخش تقسیم کنید: یکی برای آموزش و دیگری برای آزمایش. نسبت‌های رایج ۷۰-۳۰ یا ۸۰-۲۰ هستند.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=76)
```

K-Fold Cross-Validation: داده‌ها را به k زیر مجموعه (فولد) تقسیم کنید. مدل را k بار آموزش دهید، هر بار از یک فولد متفاوت به عنوان مجموعه تست و از k-1 باقی مانده به عنوان مجموعه آموزش استفاده کنید.

```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
scores = cross_val_score(model, X, y, cv=5)
```

Stratified K-Fold Cross-Validation: شبیه به K-Fold Cross-Validation، اما اطمینان حاصل می کند که هر فولد دارای همان نسبت برچسب های کلاس به عنوان مجموعه داده اصلی است.

```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5)
for train_index, test_index in skf.split(X, y):
   X_train, X_test = X[train_index], X[test_index]
   y_train, y_test = y[train_index], y[test_index]
```

Time Series Split: برای داده‌های سری زمانی، داده‌ها را براساس یک برش زمانی خاص تقسیم کنید، اطمینان حاصل کنید که داده‌های آموزشی از نظر زمانی بر داده‌های آزمون مقدم هستند.

```
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)
for train_index, test_index in tscv.split(X):
   X_train, X_test = X[train_index], X[test_index]
   y_train, y_test = y[train_index], y[test_index]
```

در ابتدا کتابخانه‌های مورد نیاز را اضافه می‌کنیم.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import tree
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

سپس با استفاده از کتابخانه sklearn به میزان ۳۰ درصد از داده‌ها را به عنوان داده آزمون انتخاب می‌کنیم. مقدار random state را برابر دو رقم آخر شماره دانشجویی انتخاب شده است. ابعاد آن را نیز نمایش می‌دهیم.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=76)
print('train:',X_train.shape, y_train.shape,'\ntest: ', X_test.shape, y_test.shape)
```

```
train: (140, 5) (140, 1)
test:  (60, 5) (60, 1)
```

شکل ۷۳ ابعاد داده‌های آموزش و آزمون

با استفاده از تابع درون کتابخانه sklearn داده‌ها را پردازش می‌کنیم. نکته‌ی مهم این است که ابتدا باید این scaler را fit کنیم و سپس transform بر روی داده‌ها بزنیم. در واقع scaler نباید داده‌های آزمون را ببیند و فقط باید بر روی داده‌های آموزش fit کردن صورت گیرد. برای نمایش بهتر، تعداد ۵ عدد از داده‌ها را قبل و بعد از sacler را نمایش می‌دهیم؛ که نشان از موفقیت‌آمیز بودن scaler ما می‌باشد. ابعاد ماتریس‌های آموزش و آزمون نیز نمایش داده شده است.

```
# Initialize the StandardScaler
scaler = StandardScaler()
# Fit the scaler on the training data and transform both training and test data
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
x_train = x_train_scaled
x_test = x_test_scaled
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(140, 5)
(60, 5)
(140, 1)
(60, 1)
```

شکل ۷۴ ابعاد ماتریس آموزش و آزمون

سپس یک شی از درخت تصمیم را با استفاده از دستور زیر ایجاد می‌کنیم و آرگومان ورودی random_state را فقط برای آن قرار می‌دهیم. سپس با فراخوانی تابع fit و دادن ورودی و خروجی مربوط به آموزش درخت تصمیم را ایجاد می‌کنیم.

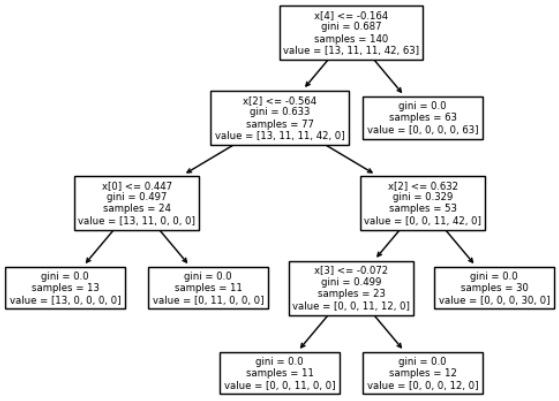
```
clf = tree.DecisionTreeClassifier(random_state=76)
clf.fit(x_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=76)
```

شکل ۷۵ درخت تصمیم ساخته شده

با استفاده از دستورات زیر درخت تصمیم را رسم می‌کنیم. گزاره داخل گره اول بیان می‌کند اگر کوچکتر از ۱۴.۸۹۹ باشد به سمت راست و اگر بیشتر باشد به سمت چپ بروند. ویژگی چهارم مربوط به Na_to_K می‌باشد. همانطور که مشخص است اگر بزرگتر باشد با یک نقطه تصمیم داریم که مشخص می‌کند با وجود ۶۳ نمونه داده مربوط به کلاس دارویی drugB می‌باشد. در واقع به این معنی است که یک کلاس با ویژگی و یک آستانه مشخص شده است. مقدار gini نیز مقدار اهمیت ویژگی‌ها را مشخص می‌کند. هرچقدر شاخه پایین‌تر می‌رویم مقدار آن کاهش می‌یابد. gini برابر با صفر به این معنی است که مقدار ناخالصی در داده‌ها وجود ندارد و آنتروپی برابر صفر است. به این معنی است تعداد داده‌های موجود در آن گره به یک دسته تعلق دارند. همانطور که مشخص است در هیچکدوم از گره‌های برگ مقدار gini غیرصفر وجود ندارد و هیچ داده‌ای miss_class نشده‌اند.

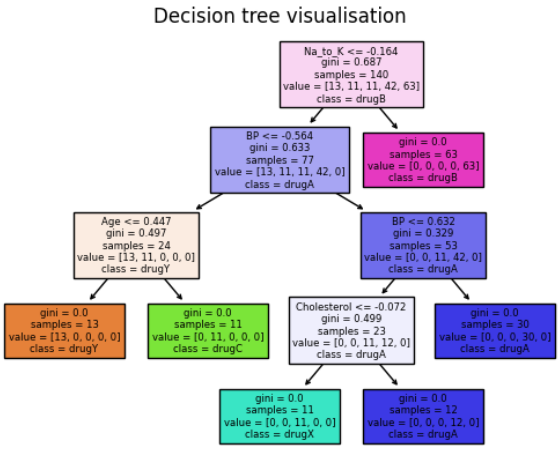
```
tree.plot_tree(clf)
plt.show()
```



شکل ۷۶ نمایش درخت تصمیم

با استفاده از دستورات زیر درخت تصمیم را به همراه نام ویژگی‌ها و کلاس نمایش می‌دهیم.

```
tree.plot_tree(clf,feature_names=df.columns,filled="True",class_names=df["Drug"].unique
())
plt.title("Decision tree visualisation")
plt.show()
```



شکل ۷۷ نمایش درخت تصمیم

دستورات زیر برای نمایش ویژگی‌های درخت تصمیم استفاده می‌شود. در ادامه توضیحات مربوط به هر کدام ذکر شده است.

۱. تعداد گره‌ها
۲. تعداد گره‌های برگ
۳. تعداد نمونه‌های موجود در هر گره. از بالا به پایین و چپ به راست نمایش می‌دهد.
۴. فرزند سمت چپ هر گره کدام است. در صورت نمایش ۱- به این معنی است که فرزند ندارد و یک گره برگ است.
۵. فرزند سمت راست هر گره کدام است.
۶. مقادیر نشان داده شده در هر گره را نمایش می‌دهد.
۷. تعریف می‌کند که هر گره براساس کدام ویژگی شرط گذاری شده است. در صورتی که به گره برگ برسیم مقدار ۲- را بازمی‌گرداند.
۸. مقادیر شرط گذاشته شده را نمایش می‌دهد. در صورتی که به گره برگ برسیم مقدار ۲- را بازمی‌گرداند.
۹. مقادیر مربوط به gini را نمایش می‌دهد.

۱۰. میزان عمق درخت را مشخص می‌کند.

```
print("node_count:", clf.tree_.node_count)
print("n_leaves:", clf.tree_.n_leaves)
print("n_node_samples:", clf.tree_.n_node_samples)
print("children_left:", clf.tree_.children_left)
print("children_right:", clf.tree_.children_right)
print("value:", clf.tree_.value)
print("feature:", clf.tree_.feature)
print("threshold:", clf.tree_.threshold)
print("impurity:", clf.tree_.impurity)
print("max_depth:", clf.tree_.max_depth)
```

```
node_count: 11
n_leaves: 6
n_node_samples: [140  77  24  13  11  53  23  11  12  30  63]
children_left: [ 1  2  3 -1 -1  6  7 -1 -1 -1 -1]
children_right: [10  5  4 -1 -1  9  8 -1 -1 -1 -1]
value: [[[13. 11. 11. 42. 63.]]

 [[13. 11. 11. 42.  0.]]

 [[13. 11.  0.  0.  0.]]

 [[13.  0.  0.  0.  0.]]

 [[ 0. 11.  0.  0.  0.]]

 [[ 0.  0. 11. 42.  0.]]

 [[ 0.  0. 11. 12.  0.]]

 [[ 0.  0. 11.  0.  0.]]

 [[ 0.  0.  0. 12.  0.]]

 [[ 0.  0.  0. 30.  0.]]

 [[ 0.  0.  0.  0. 63.]]]
feature: [ 4  2  0 -2 -2  2  3 -2 -2 -2 -2]
threshold: [-0.16413499 -0.56379378  0.44748352 -2.         -2.          0.63213241
 -0.07161146 -2.         -2.         -2.         -2.        ]
impurity: [0.68653061 0.63315905 0.49652778 0.         0.         0.32894268
 0.49905482 0.         0.         0.         0.        ]
max_depth: 4
```

شکل ۷۸ مقادیر نمایش داده شده برای ویژگی‌های درخت

از دستورات زیر برای نمایش درخت تصمیم به صورت متن استفاده می‌گردد.

```
r = tree.export_text(clf)
print(r)
```

```
|--- feature_4 <= -0.16
|   |--- feature_2 <= -0.56
|   |   |--- feature_0 <= 0.45
|   |   |   |--- class: drugA
|   |   |--- feature_0 >  0.45
|   |   |   |--- class: drugB
|   |--- feature_2 >  -0.56
|   |   |--- feature_2 <= 0.63
|   |   |   |--- feature_3 <= -0.07
|   |   |   |   |--- class: drugC
|   |   |   |--- feature_3 >  -0.07
|   |   |   |   |--- class: drugX
|   |   |--- feature_2 >  0.63
|   |   |   |--- class: drugX
|--- feature_4 >  -0.16
|   |--- class: drugY
```

شکل ۷۹ درخت تصمیم به صورت متن

۳-۲-بخش دوم

با استفاده از دستور زیر مقدار پیش‌بین مربوط به داده آزمون را محاسبه می‌کنیم.

```
y_pred = clf.predict(x_test)
print("score:",clf.score(x_test, y_test))
```

```
score: 0.9833333333333333
```

شکل ۸۰ مقادیر امتیاز داده‌های آموزش

با استفاده از دستور زیر مقادیر احتمال تعلق به هر کلاس را نمایش می‌دهیم.

```
print("proba:", clf.predict_proba(x_test))
```

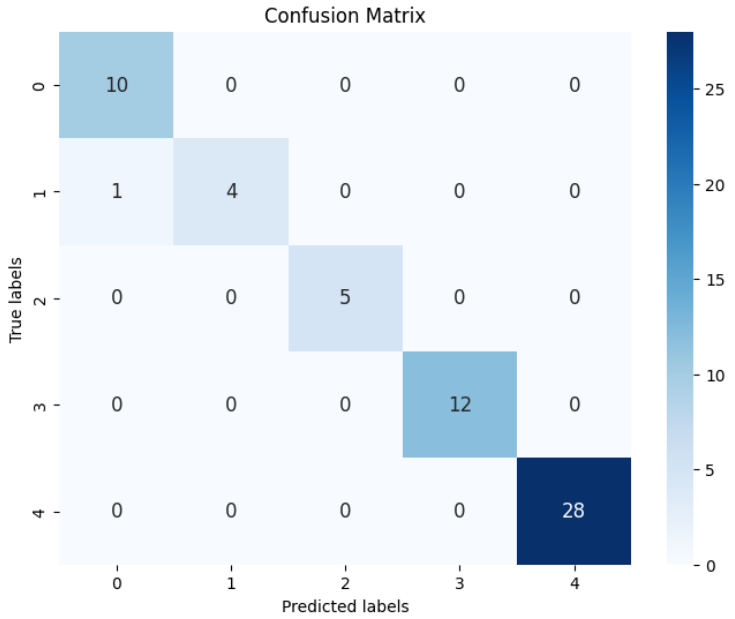
با استفاده از دستورات زیر نشان می‌دهیم که کدام گره‌ها برای رسیدن به داده آزمون ما طی شده است. به صورت یک آرایه می‌باشد که ۰ به معنی عدم عبور و ۱ به معنی عبور می‌باشد.

```
i = 2
decision_path = clf.decision_path(x_test[[i]])
print("path:", decision_path.toarray())
print("value:", clf.predict(x_test[[i]]))
```

```
path: [[1 1 0 0 0 1 1 1 0 0 0]]
value: ['drugC']
```

شکل ۸۱ نمایش یکی از داده‌های آزمون

در شکل زیر ماتریس درهم‌ریختگی مربوط به این درخت تصمیم را نشان می‌دهد. همانطور که مشاهده می‌شود یکی از داده‌های مربوط به کلاس ۱ به اشتباهی به کلاس ۰ تعلق گرفته است.



شکل ۸۲ ماتریس درهم‌ریختگی

با استفاده از classification_report می‌توانیم یک گزارش از نحوه عملکرد ماشین دریافت کرد. همانطور که مشاهده می‌شود مقادیر accuracy، precision، recall و F1 score را گزارش شده است. تعداد داده‌های آزمون هر دسته نیز مشخص شده است.

```
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

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

       drugA       0.91      1.00      0.95        10
       drugB       1.00      0.80      0.89         5
       drugC       1.00      1.00      1.00         5
       drugX       1.00      1.00      1.00        12
       drugY       1.00      1.00      1.00        28

    accuracy                           0.98        60
   macro avg       0.98      0.96      0.97        60
weighted avg       0.98      0.98      0.98        60
```

شکل ۸۳ گزارش مربوط به classification_report

با استفاده از دستور زیر مقدار ارزیابی میانگین مربوط به این روش را نمایش می‌دهیم.

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
print("\nAccuracy:", accuracy)
```

```
print("Precision:", precision)
print("Recall:", recall)
```

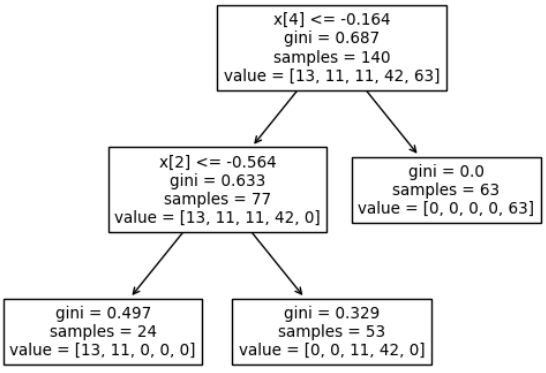
```
Accuracy: 0.9833333333333333
Precision: 0.9818181818181818
Recall: 0.96
```

شکل ۸۴ ارزیابی عملکرد درخت تصمیم

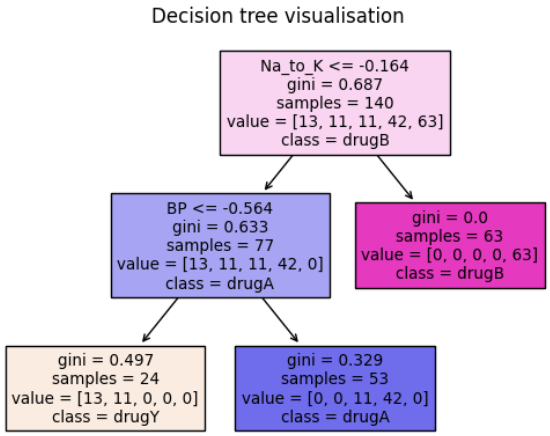
تعداد آرگومان‌های درخت تصمیم بسیار زیاد است. آرگومان criterion برای نوع معیار است که شامل gini، entropy و log_loss می‌باشد. یکی از فراپارامترهای مربوط به هرس کردن max_depth که مربوط به عمق درخت می‌باشد. min_samples_split برای تعیین حداقل تعداد sample مورد نیاز برای اینکه به گره تصمیم تبدیل شود، گره برگ نباشد. min_samples_leaf حداقل تعداد sample مورد نیاز برای اینکه به گره برگ تبدیل شود. با استفاده از max_features می‌گوییم با چه تعداد ویژگی درخت را بسازد. max_leaf_nodes حداکثر تعداد نقاط برگ را مشخص می‌کند؛ که نشان از بزرگ بودن یا کوچک بودن درخت تصمیم است. min_impurity_decrease برای مشخص کردن میزان کاهش آنتروپی می‌باشد. در واقع میزان کاهش ناخالصی را معین می‌کند. برای اینکه هر گره به گره تصمیم‌گیری تبدیل نشود، از این پارامتر استفاده می‌شود. class_weight را نیز برای استفاده در دیتاهای بالانس یا غیربالانس استفاده می‌گردد. آرگومان ccp_alpha مربوط به Cost-Complexity Pruning که میزان نرخ حرص کردن را مشخص می‌کند.

در این بخش مقادیر max_depth و ccp_alpha را تغییر می‌دهیم.

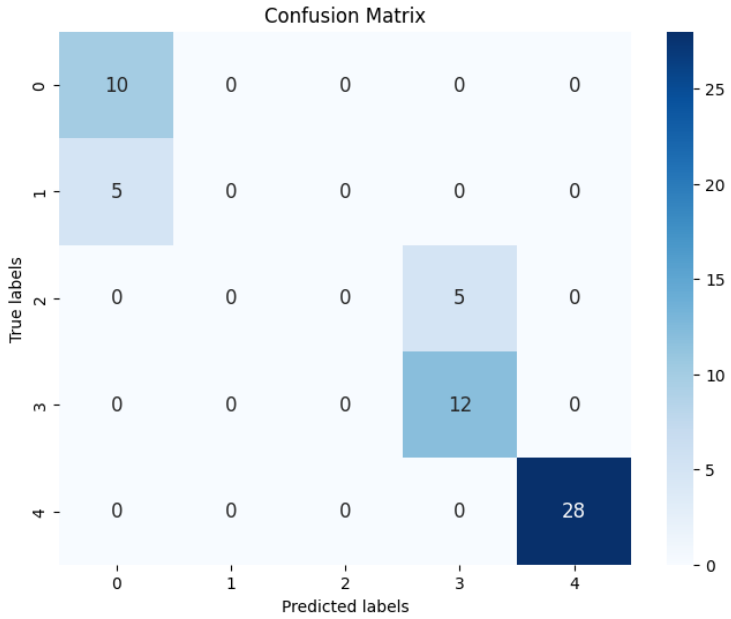
```
max_depth=2
ccp_alpha=0.1
```



شکل ۸۵ نمایش درخت تصمیم



شکل ۸۶ نمایش درخت تصمیم



شکل ۸۷ ماتریس درهم‌ریختگی

```
Accuracy: 0.8333333333333334
Precision: 0.4745098039215686
Recall: 0.6
```

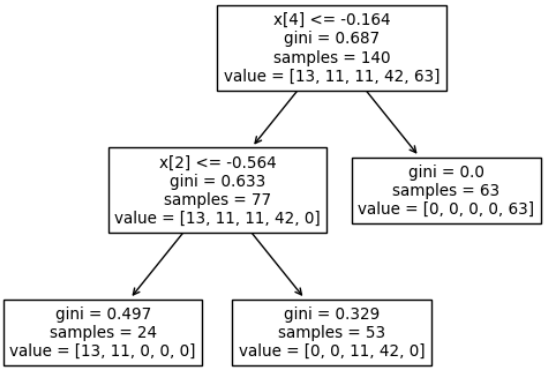
شکل ۸۸ ارزیابی عملکرد درخت تصمیم

همانطور که مشاهده می‌شود در این حالت درخت تصمیم کلاس ۱ و ۲ را نتوانسته است به درستی تشخیص دهد. همانطور که در درخت تصمیم نیز مشاهده می‌گردد گره برگ مربوط به ۳ کلاس تشخیص داده شده‌اند. در گره سوم میزان داده‌ها miss_class یازده و در گره چهارم میزان داده‌ها miss_class یازده می‌باشد.

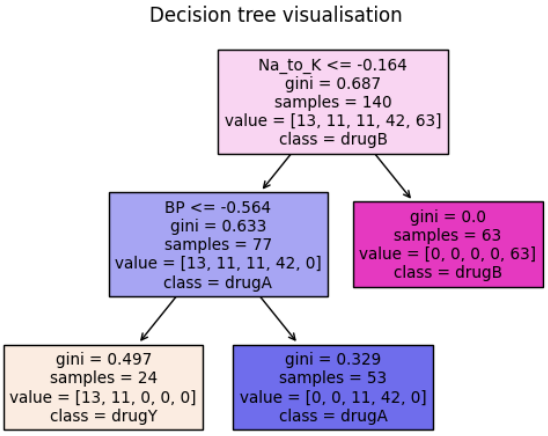
در این بخش مقادیر max_depth را تغییر می‌دهیم. مقدار ccp_alpha با اینکه max_depth را بیشتر انتخاب کرده بودیم، باعث شده درخت ما حرص شود و عمق آن کاهش یابد. در این حالت نیز مجددا کلاس ۱ و ۲ به درستی تشخیص داده نشدند. همانطور که در درخت تصمیم نیز مشاهده می‌گردد گره برگ مربوط به ۳ کلاس تشخیص داده شده‌اند. در گره سوم میزان داده‌ها miss_class یازده و در گره چهارم میزان داده‌ها miss_class یازده می‌باشد.

max_depth=5

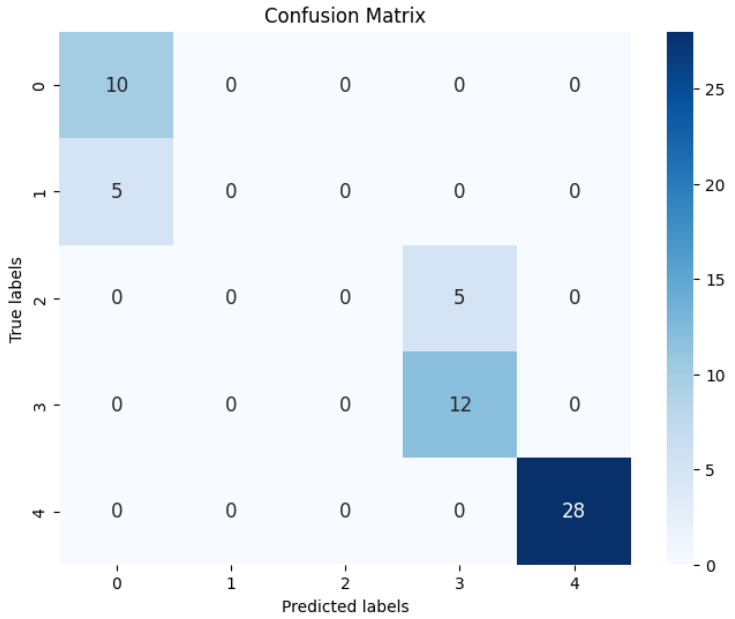
ccp_alpha=0.1



شکل ۸۹ نمایش درخت تصمیم



شکل ۹۰ نمایش درخت تصمیم



شکل ۹۱ ماتریس درهم‌ریختگی

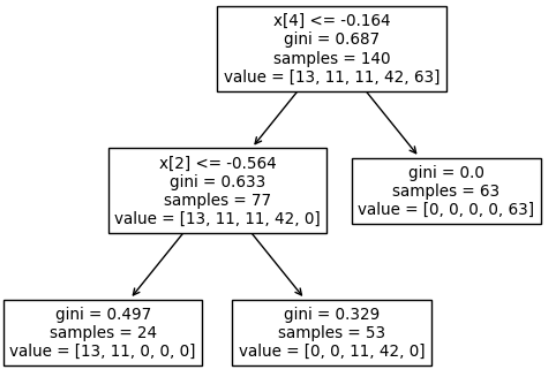
```
Accuracy: 0.8333333333333334
Precision: 0.4745098039215686
Recall: 0.6
```

شکل ۹۲ ارزیابی عملکرد درخت تصمیم

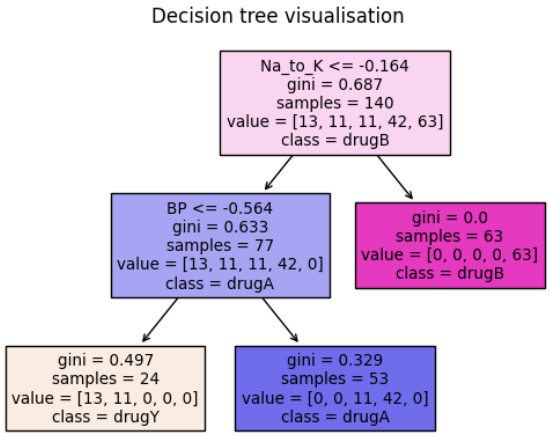
در این بخش مقادیر max_depth و ccp_alpha را تغییر می‌دهیم. همانطور که مشاهده می‌شود کاهش نرخ باعث افزایش تعداد لایه‌ها نمی‌گردد. زیرا عمق درخت ۲ در نظر گرفته شده است.

max_depth=2

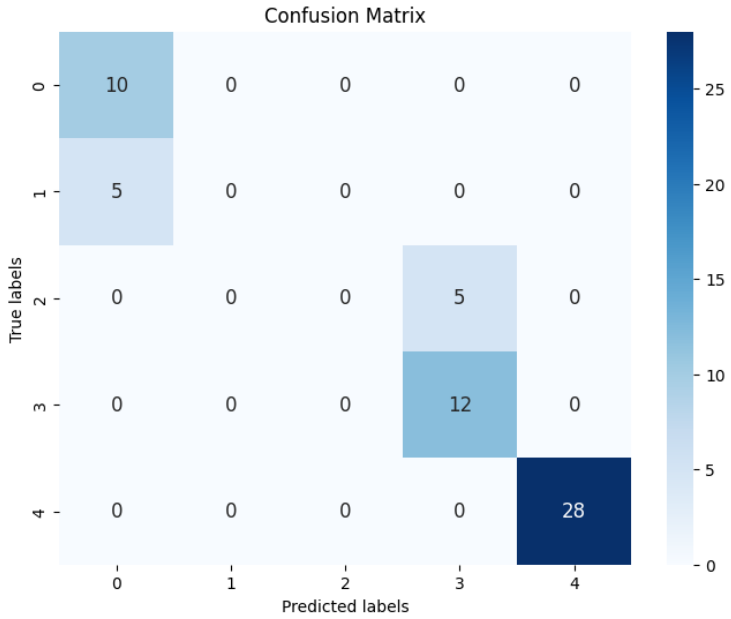
ccp_alpha=0.01



شکل ۹۳ نمایش درخت تصمیم



شکل ۹۴ نمایش درخت تصمیم



شکل ۹۵ ماتریس درهم‌ریختگی

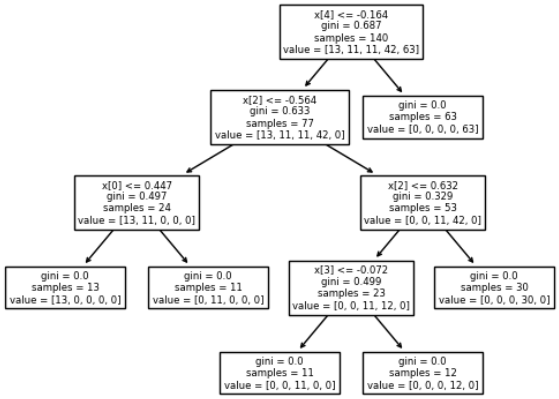
```
Accuracy: 0.8333333333333334
Precision: 0.4745098039215686
Recall: 0.6
```

شکل ۹۶ ارزیابی عملکرد درخت تصمیم

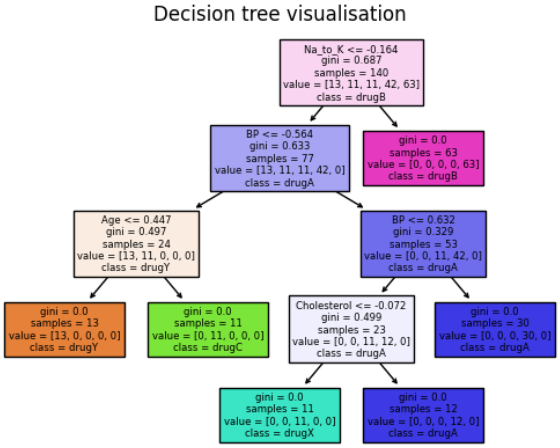
در این بخش مقادیر max_depth را تغییر می‌دهیم. همانطور که مشاهده می‌شود افزایش عمق نسبت به حالت اول، باعث افزایش تعداد لایه‌ها نمی‌گردد.

max_depth=5

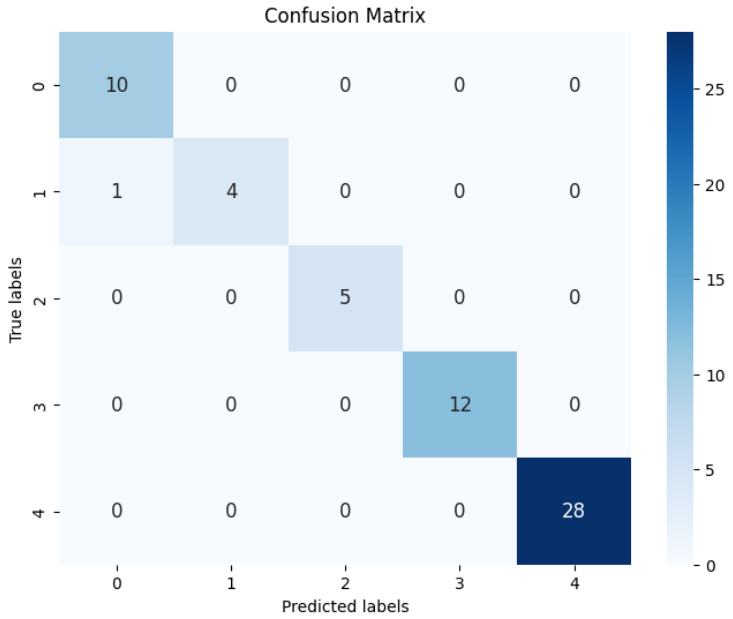
ccp_alpha=0.01



شکل ۹۷ نمایش درخت تصمیم



شکل ۹۸ نمایش درخت تصمیم



شکل ۹۹ ماتریس درهم‌ریختگی

```
Accuracy: 0.9833333333333333
Precision: 0.9818181818181818
Recall: 0.96
```

شکل ۱۰۰ ارزیابی عملکرد درخت تصمیم

۳-۳-بخش سوم

AdaBoost (تقویت تطبیقی) با ترکیب چندین طبقه‌بندی ضعیف برای ایجاد یک طبقه‌بندی قوی کار می‌کند.

AdaBoost، مخفف Adaptive Boosting، یک تکنیک یادگیری گروهی است که چندین یادگیرنده ضعیف، معمولا درخت‌های تصمیم را برای ایجاد یک طبقه‌بندی قوی ترکیب می‌کند. در اینجا نحوه عملکرد AdaBoost با درختان تصمیم آمده است:

AdaBoost چندین درخت تصمیم را به صورت متوالی آموزش می‌دهد، که هر کدام بر تصحیح خطاهای درختان قبلی تمرکز دارند.

در ابتدا، تمام نقاط داده دارای وزن برابر هستند. پس از آموزش هر درخت، وزن نقاط طبقه‌بندی اشتباه افزایش می‌یابد تا درخت بعدی بیشتر روی این نمونه‌های طبقه‌بندی سخت‌تر تمرکز کند.

مراحل AdaBoost با درختان تصمیم:

مقداردهی اولیه: به تمام نمونه های آموزشی وزن های مساوی اختصاص دهید.

تکرار: برای هر دور:

یک درخت تصمیم (اغلب یک درخت ساده که به آن کنده تصمیم می گویند) آموزش دهید.

میزان خطای درخت را محاسبه کنید.

وزن نمونه‌های تمرینی را تنظیم کنید: برای نمونه‌های طبقه‌بندی اشتباه وزن‌ها را افزایش دهید و برای نمونه‌هایی که به درستی طبقه‌بندی شده‌اند، وزن‌ها را کاهش دهید.

وزن درخت را بر اساس دقت آن تعیین کنید.

ترکیب: خروجی های همه درختان را در یک مجموع وزن دار ترکیب کنید که نشان دهنده پیش بینی نهایی است.

از مزایای این روش می‌توان به موارد زیر اشاره نمود:

دقت بهبود یافته: با تمرکز بر موارد دشوار، AdaBoost عملکرد کلی مدل را بهبود می‌بخشد.

تطبیق‌پذیری: با انواع مختلف یادگیرندگان ضعیف به خوبی کار می‌کند، اما به دلیل سادگی و سرعت، معمولاً از خرده تصمیم‌گیری استفاده می‌شود.

Random Forest یک روش یادگیری گروهی است که چندین درخت تصمیم می‌سازد و خروجی‌های آنها را برای بهبود دقت و کاهش بیش از حد برازش ادغام می‌کند.

درخت تصمیم مدلی است که برای طبقه‌بندی و رگرسیون استفاده می‌شود که داده‌ها را به شاخه‌ها تقسیم می‌کند تا براساس ویژگی‌های ورودی پیش‌بینی کند.

جنگل تصادفی از مجموعه‌ای (جنگل) از درختان تصمیم تشکیل شده است. هر درخت بر روی یک زیر مجموعه تصادفی متفاوت از داده‌ها و ویژگی‌ها آموزش داده می‌شود.

نحوه عملکرد جنگل تصادفی به صورت زیر است:

Bootstrapping: چندین زیرمجموعه از مجموعه داده اصلی با استفاده از نمونه‌برداری با جایگزینی ایجاد می‌شوند.

انتخاب ویژگی تصادفی: برای هر زیرمجموعه، یک درخت تصمیم ساخته می‌شود، اما در هر تقسیم فقط یک زیرمجموعه تصادفی از ویژگی‌ها در نظر گرفته می‌شود.

تجمیع: پیش‌بینی‌ها از همه درخت‌ها تجمیع می‌شوند (رای اکثریت برای طبقه‌بندی یا میانگین برای رگرسیون) برای تولید خروجی نهایی.

از مزایای این روش می‌توان به موارد زیر اشاره نمود:

دقت بهبود یافته: ترکیب چندین درخت معمولاً منجر به عملکرد بهتر نسبت به یک درخت می‌شود.

کاهش اضافه برازش: با میانگین گرفتن چندین درخت، Random Forest خطر تطبیق بیش از حد داده-های آموزشی را کاهش می‌دهد.

استحکام: نسبت به داده‌های نویزی و نقاط پرت حساسیت کمتری دارد.

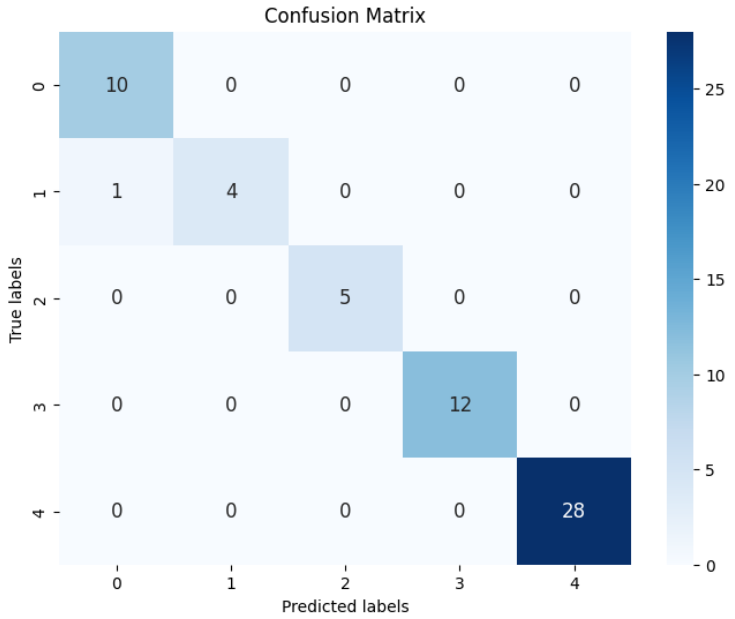
با استفاده از قدرت درخت‌های تصمیم‌گیری چندگانه، Random Forest عملکرد پیش‌بینی و استحکام را در مقایسه با درخت‌های تصمیم فردی افزایش می‌دهد.

در زیر دستورات مربوط به درخت با عمق ۲ با استفاده از AdaBoost قرار دارد. همانطور که از پاسخ مشخص است بهبود یافته است؛ و با استفاده از این درخت تصمیم کلاس‌بندی را انجام دهیم.

```
from sklearn.ensemble import AdaBoostClassifier

base_estimator = tree.DecisionTreeClassifier(max_depth=2)  # Using a weak classifier

clf_adaboost = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50, learning_rate=0.1, random_state=76)

clf_adaboost.fit(x_train, y_train)
```

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

       drugA       0.91      1.00      0.95        10
       drugB       1.00      0.80      0.89         5
       drugC       1.00      1.00      1.00         5
       drugX       1.00      1.00      1.00        12
       drugY       1.00      1.00      1.00        28

    accuracy                           0.98        60
   macro avg       0.98      0.96      0.97        60
weighted avg       0.98      0.98      0.98        60


Accuracy: 0.9833333333333333
Precision: 0.9818181818181818
Recall: 0.96
```

شکل ۱۰۱ ارزیابی عملکرد درخت



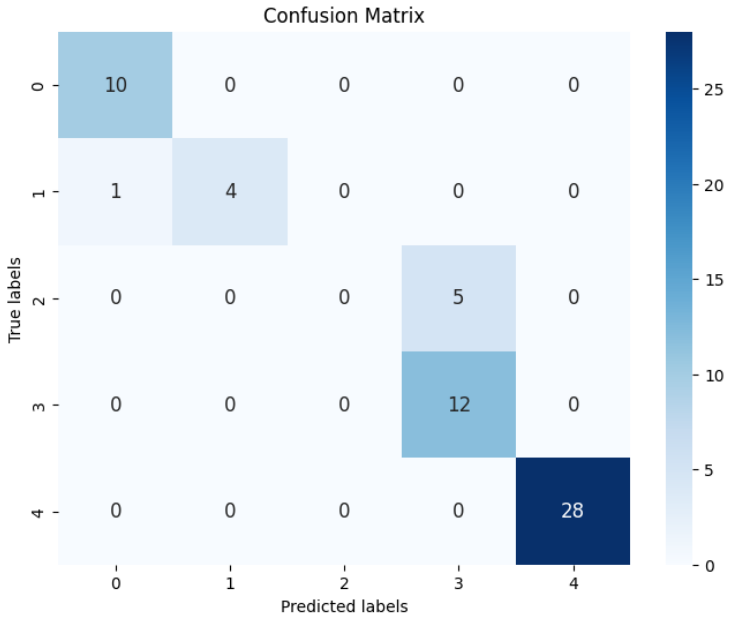
شکل ۱۰۲ ماتریس درهم‌ریختگی

در زیر دستورات مربوط به درخت با عمق ۲ با استفاده از Random Forest قرار دارد. همانطور که از پاسخ مشخص است بهبود یافته است. در این حالت نتوانستیم کلاس ۲ را مجددا به درستی شناسایی کنیم.

```
from sklearn.ensemble import RandomForestClassifier
clf_rf = RandomForestClassifier(n_estimators=200, max_depth=2, random_state=76)
clf_rf.fit(x_train, y_train)
```

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

       drugA       0.91      1.00      0.95        10
       drugB       1.00      0.80      0.89         5
       drugC       0.00      0.00      0.00         5
       drugX       0.71      1.00      0.83        12
       drugY       1.00      1.00      1.00        28

    accuracy                           0.90        60
   macro avg       0.72      0.76      0.73        60
weighted avg       0.84      0.90      0.86        60


Accuracy: 0.9
Precision: 0.7229946524064171
Recall: 0.76
```

شکل ۱۰۳ ارزیابی عملکرد درخت



شکل ۱۰۴ ماتریس درهم‌ریختگی

۴–سوال چهارم

۴-۱-بررسی دیتاست

این مجموعه داده به سال ۱۹۸۸ برمی‌گردد و از چهار پایگاه داده تشکیل شده است: کلیولند، مجارستان، سوئیس، و لانگ بیچ V. شامل ۷۶ ویژگی، از جمله ویژگی پیش بینی شده است، اما تمام آزمایش‌های منتشر شده به استفاده از زیر مجموعه ای از ۱۴ مورد اشاره دارد. فیلد "هدف" به وجود بیماری قلبی در بیمار اشاره دارد. عدد صحیح برابر صفر، بدون بیماری و برابر یک، بیماری است. اطلاعات ویژگی را می‌توان بیان نمود: سن، جنس، نوع درد قفسه سینه (۴ مقدار)، فشار خون در حال استراحت، کلسترول سرم بر حسب میلی گرم در دسی لیتر، قند خون ناشتا > ۱۲۰ میلی گرم در دسی لیتر،نتایج الکتروکاردیوگرافی در حالت استراحت (مقادیر ۰،۱،۲)، حداکثر ضربان قلب به دست آمده، آنژین ناشی از ورزش، Oldpeak = افسردگی ST ناشی از ورزش نسبت به استراحت، شیب بخش ST اوج تمرین، تعداد عروق اصلی (۰-۳) رنگ‌آمیزی شده توسط فلوروسوپی، thal: ۰ = عادی ۱ = نقص ثابت ۲ = عیب قابل برگشت. نام و شماره تامین اجتماعی بیماران اخیراً از پایگاه داده حذف شده و مقادیر ساختگی جایگزین آن شده است.

۴-۲-بررسی کلاس‌بندی Bayes

ایده‌ی کلاس‌بندی $Bayes$ از رابطه‌ی معادله‌ی کلی زیر می‌باشد.

$$P(A \mid B) = \frac{P(B \mid A)P(B)}{P(A)}$$

اگر رابطه بالا رو به صورت زیر بازنویسی کنیم، y در واقع برچسب‌های هر کلاس می‌باشند و می‌خواهیم محاسبه کنیم میزان احتمال هر کلاس با ویژگی‌های ورودی چه میزان است و احتمال ماکزیمم را به عنوان کلاس مدنظر انتخاب می‌کنیم.

$$P(y \mid X) = \frac{P(X \mid y)P(y)}{P(X)}$$

یک شرط مهم دارد این روش که مستقل بودن ویژگی‌های ورودی از یکدیگر می‌باشد. لین شزط امکان بازنویسی رابطه به صورت زیر را فراهم می‌کند.

$$P(y \mid X) = \frac{P(x_1 \mid y)P(x_2 \mid y)P(x_3 \mid y)\dots P(x_n \mid y)P(y)}{P(X)}$$

برای یافتن ماکزیمم مقدار، از طرفین معادله بالا یک ماکزیمم‌گیری انجام می‌دهیم.

$$y = \arg\max_y P(y \mid X) = \arg\max_y \frac{P(x_1 \mid y)P(x_2 \mid y)P(x_3 \mid y)\dots P(x_n \mid y)P(y)}{P(X)}$$

از آنجایی که ماکزیمم‌گیری بر روی y است و $P(X)$ تاثیری بر آن ندارد، می‌توانیم صرف نظر کنیم. برای ما فقط ماکزیمم مهم است و مقدار برای ما اهمیتی ندارد.

$$y = \arg\max_{y} P(x_1 \mid y)P(x_2 \mid y)P(x_3 \mid y)\dots P(x_n \mid y)P(y)$$

از آنجایی که این مقادیر کوچک می‌باشند، ضرب آن‌ها بسیار کوچکتر می‌شود و دقت بسیار کم می‌باشد. برای رفع این موضوع می‌توانیم با استفاده از لگاریتم این مسائل را به جمع تبدیل کنیم و رابطه را بازنویسی کنیم.

$$y = \arg\max_{y} \log\big(P(x_1 \mid y)\big) + \log\big(P(x_2 \mid y)\big) + \log\big(P(x_3 \mid y)\big) + \dots + \log\big(P(x_n \mid y)\big) + \log\big(P(y)\big)$$

برای اینکه یک داده ورودی را کلاس‌بندی کنیم، رابطه بالا را محاسبه می‌کنیم. به $P(y)$ اصطلاحاً prior می‌گویند، که احتمال هر کلاس را تعیین می‌کنند. به رابطه‌ی بالا نیز posteriors می‌گویند. برای $\big(P(x \mid y)\big)$ از PDF گاووسی استفاده می‌کنیم که رابطه آن به صورت زیر است.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

۴-۳-کد پایتون

در ابتدا یک پوشه برای ذخیره سازی داده‌ها ایجاد می‌کند. در ابتدا کتابخانه مدنظر را فراحوانی می‌کنیم. سپس درون حلقه if بررسی می‌شود اگر پوشه‌ای با این نام وجود ندارد، آن را بسازد. و در نهایت به داخل آن پوشه انتقال پیدا می‌کنیم.

```
import os
# Define the folder name
folder_name = "Data"
# Check if the folder already exists, and create it if not
if not os.path.exists(folder_name):
  os.makedirs(folder_name)
# Commented out IPython magic to ensure Python compatibility.
%cd Data
```

در مرحله بعد برای اینکه دیتاست مدنظر را از سایت kaggle دانلود کنیم نیاز به نصب پکیج مربوط به آن با دستور زیر می‌باشد.

```
!pip install kaggle
```

سپس با استفاده از دستور زیر که در سایت kaggle با زدن بر روی سه نقطه و انتخاب Copy API command قابل یافتن است، دیتاست مدنظر دانلود می‌گردد. و در نهایت فایل zip دانلود شده را از حالت فشرده خارج می‌کنیم و فایل zip را حذف می‌کنیم.

```
!kaggle datasets download -d johnsmith88/heart-disease-dataset
!unzip \*.zip && rm *.zip
```

حال باید کتابخانه‌های مرد نیاز پایتون را می‌افزاییم.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
%matplotlib inline
```

سپس فایل csv. دانلود شده در گام ابتدایی را با استفاده از دستور زیر می‌خوانیم. آدرس آن را با کلیک راست کردن بر روی آن و انتخاب copy path یافته‌ایم.

```
df = pd.read_csv('/content/Data/heart.csv')
```

سپس با استفاده از دستور زیر ۵ داده ابتدایی (در صورت انتخاب کردن تعداد) به همراه عنوان ویژگی‌ها و برچسب (که ستون آخر می‌باشد) را نمایش می‌دهد.

```
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

شکل ۱۰۵ نتایج مربوط به ()head

با استفاده از دستور زیر اطلاعات کلی در رابطه با این data-frame لازم است را به ما می‌دهد. از آنجایی که داده null موجود نمی‌باشد، نیاز به حذف آن نیز نمی‌باشد.

```
df.info()
```

```
print(df.isnull().sum())
```

```
<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
```

شکل ۱۰۶ نتایج مربوط به info()

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

شکل ۱۰۷ نتایج isnull()

با استفاده از دستور زیر اطلاعاتی در رابطه با میانگین، واریانس، مینیمم، ماکزیمم و ... مربوط به داده‌های عددی را به ما می‌دهد.

```
df.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366	0.754146	2.323902	0.513171
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.617755	1.030798	0.620660	0.500070
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

شکل ۱۰۸ نتایج مربوط به describe()

با استفاده از دستور زیر مقادیر مجزا در target که در واقع برچسب‌های ما می‌باشد، به همراه تعداد آن‌ها نمایش می‌دهد. از آنجایی که تعداد داده‌ها زیاد می‌باشد و اختلاف بین آن‌ها قابل چشم پوشی است، داده‌ها را به صورت بالانس شده در نظر می‌گیریم.

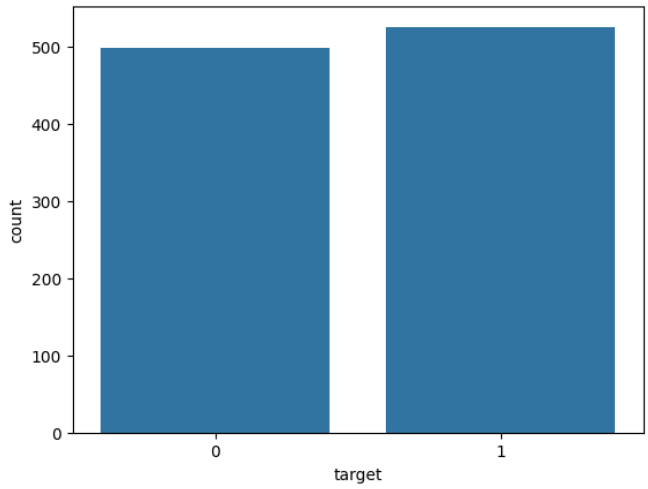
```
df.groupby('target').size()
```

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

شکل ۱۰۹ تعداد داده‌های هر دسته

با استفاده از کتابخانه seaborn که در آن دستور زیر وجود دارد به صورت قابل نمایش تعداد داده‌ها را می‌توانیم رسم نماییم.

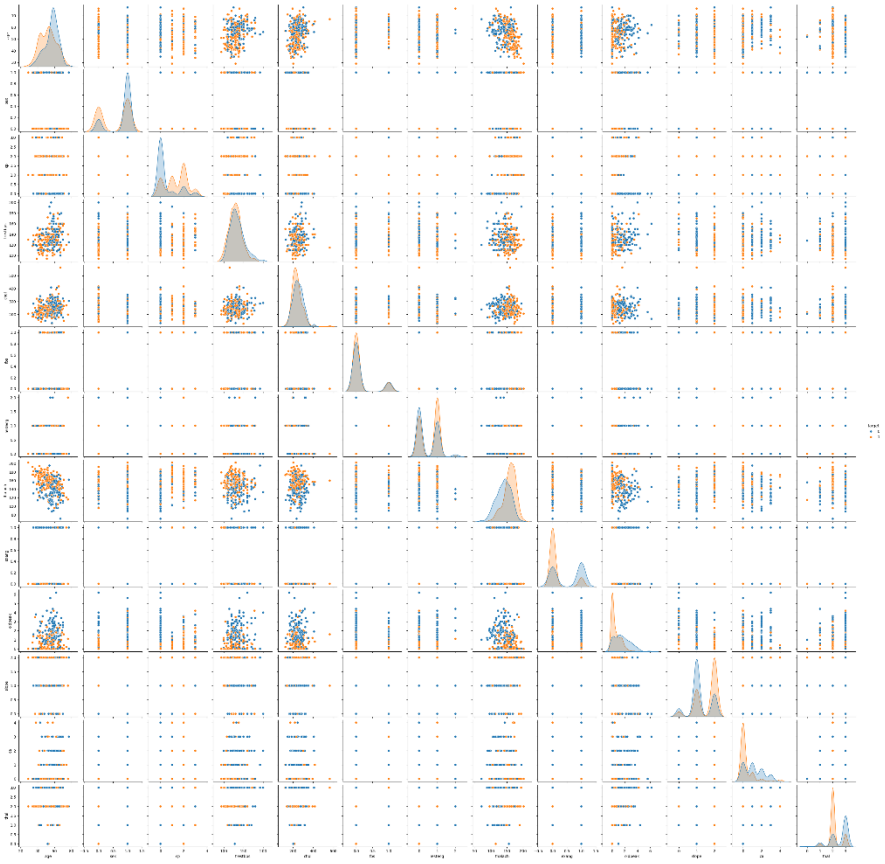
```
sns.countplot(x='target', data=df)
```



شکل ۱۱۰ نمایش تصویری تعداد داده‌ها

با استفاده از دستور زیر می‌توانیم نوع توزیع داده‌ها را رسم کنیم. با توجه به نمودارها می‌توان این حدس را زد که به دشواری بتوان دو کلاس را از یکدیگر مجزا کنیم.

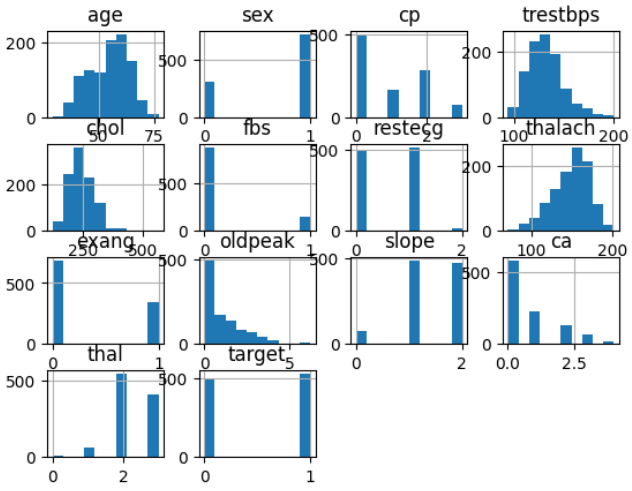
```
sns.pairplot(df,hue='target')
```



شکل ۱۱۱ نوع توزیع هر یکی از ویژگی‌ها

با استفاده از دستور زیر می‌توان مقدار هیستوگرام مربوط به ویژگی‌ها را رسم کنیم و ببینیم این ویژگی‌ها در چه بازه‌ای قرار دارند.

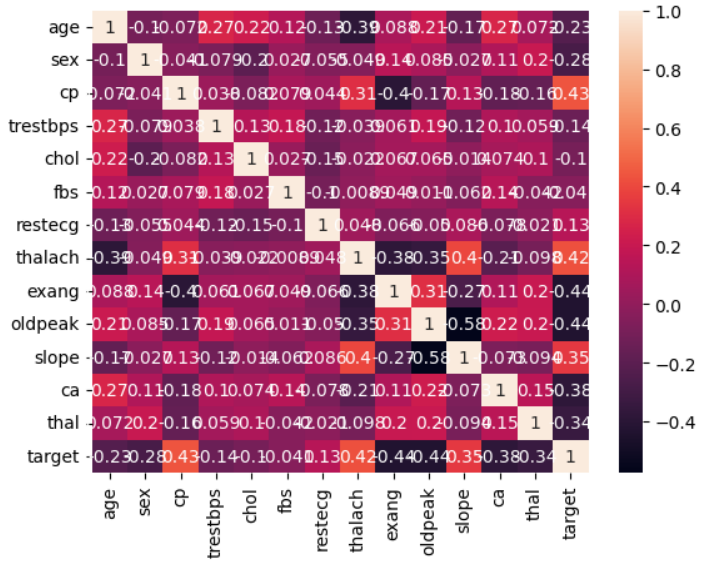
```
df.hist()
```



شکل ۱۱۲ نمودار هیستوگرام داده‌ها

با استفاده از دستور زیر میزان همبستگی بین ویژگی‌ها را نشان می‌دهیم.

```
cor=df.corr()
fig, ax = plt.subplots()
ax = sns.heatmap(cor, annot=True)
```



شکل ۱۱۳ نمودار هیت مپ تمام ویژگی‌ها

حال برای ادامه کار باید تعدادی از داده‌ها را به عنوان داده‌های آموزش و تعدادی را به عنوان داده آزمون تقسیم کنیم. به این منظور ابتدا وروردی‌ها و خروجی را با استفاده از دستور زیر مجزا می‌کنیم.

```
X=np.array(df.loc[:,df.columns!='target'])
y=np.array(df.loc[:,df.columns=='target'])
```

سپس با استفاده از کتابخانه sklearn به میزان ۲۰ درصد از داده‌ها را به عنوان داده آزمون انتخاب می‌کنیم. مقدار random state را برابر دو رقم آخر شماره دانشجویی انتخاب شده است. ابعاد آن را نیز نمایش می‌دهیم.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=76)
print('train:',X_train.shape, y_train.shape,'\ntest: ', X_test.shape, y_test.shape)
```

```
train: (820, 13) (820, 1)
test:  (205, 13) (205, 1)
```

شکل ۱۱۴ ابعاد داده‌های آموزش و آزمون

استفاده از StandardScaler برای اینکه دامنه تمام ویژگی‌ها در یک بازه قرار بگیرند در یادگیری ماشین بسیار رایج است. ویژگی‌ها را با حذف میانگین و مقیاس‌بندی به واریانس واحد استاندارد می‌کند. این استانداردسازی یک نیاز رایج برای بسیاری از برآوردگرهای یادگیری ماشین است، زیرا اگر ویژگی‌های فردی کم و بیش شبیه داده‌های استاندارد توزیع شده معمولی نباشند (مانند گاوسی با میانگین ۰ و واریانس واحد) ممکن است رفتار بدی داشته باشند. عملیات مقیاس بندی از این فرمول پیروی می کند:

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

x مقدار ویژگی، u میانگین نمونه‌های آموزشی، s انحراف معیار نمونه‌های آموزشی است.

مزایای StandardScaler را می‌توانیم چنین نام برد:

نرمالایز کردن: StandardScaler ویژگی‌ها را نرمالایز می‌کند، به این معنی که هر ویژگی دارای میانگین ۰ و انحراف استاندارد ۱ خواهد بود که منجر به توزیع گاوسی می‌شود.

عملکرد بهبود یافته: بسیاری از الگوریتم‌های یادگیری ماشین زمانی که ویژگی‌ها در مقیاس نسبتاً مشابه و نزدیک به توزیع معمولی هستند، بهتر عمل می‌کنند یا سریع‌تر همگرا می‌شوند.

سازگاری: StandardScaler با فرمت‌های داده‌های پراکنده و متراکم سازگار است و می‌تواند به طور یکپارچه در پیش پردازش ادغام شود.

این تابع درون کتابخانه sklearn موجود است. نکته‌ی مهم این است که ابتدا باید این scaler را fit کنیم و سپس transform بر روی داده‌ها بزنیم. در واقع scaler نباید داده‌های آزمون را ببیند و فقط باید بر روی داده‌های آموزش fit کردن صورت گیرد. برای نمایش بهتر، تعداد ۵ عدد از داده‌ها را قبل و بعد از sacler را نمایش می‌دهیم؛ که نشان از موفقیت‌آمیز بودن scaler ما می‌باشد.

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train)
print(X_train[:5])
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
print(X_train[:5])
```

```
[[ 70.    1.    0.  130.  322.    0.    0.  109.    0.    2.4   1.    3.
    2. ]
 [ 68.    1.    2.  118.  277.    0.    1.  151.    0.    1.    2.    1.
    3. ]
 [ 45.    1.    0.  142.  309.    0.    0.  147.    1.    0.    1.    3.
    3. ]
 [ 59.    1.    1.  140.  221.    0.    1.  164.    1.    0.    2.    0.
    2. ]
 [ 66.    1.    0.  160.  228.    0.    0.  138.    0.    2.3   2.    0.
    1. ]]
[[ 1.69135815  0.65655508 -0.92759127 -0.08736565  1.47377297 -0.42809016
  -1.02030043 -1.75211554 -0.70258435  1.14902955 -0.59754229  2.26073923
  -0.50625059]
 [ 1.47012451  0.65655508  0.99317853 -0.76695304  0.59875732 -0.42809016
   0.87685683  0.08268315 -0.70258435 -0.04828907  1.00371482  0.27593605
   1.09655051]
 [-1.07406234  0.65655508 -0.92759127  0.59222173  1.22099067 -0.42809016
  -1.02030043 -0.09205959  1.42331664 -0.90351665 -0.59754229  2.26073923
   1.09655051]
 [ 0.47457314  0.65655508  0.03279363  0.47895717 -0.49015104 -0.42809016
   0.87685683  0.65059702  1.42331664 -0.90351665  1.00371482 -0.71646554
  -0.50625059]
 [ 1.24889087  0.65655508 -0.92759127  1.61160281 -0.3540375  -0.42809016
  -1.02030043 -0.48523073 -0.70258435  1.06350679  1.00371482 -0.71646554
  -2.1090517 ]]
```

شکل ۱۱۵ نمایش ۵ عدد از داده‌های آموزش قبل و بعد از scaler

سپس یک shape از داده‌های خروجی آموزش می‌گیریم و با استفاده از کد زیر به یک بردار تبدیل می‌کنیم.

```
print(y_train.shape)
print(y_train.ravel().shape)
```

```
(820, 1)
(820,)
```

شکل ۱۱۶ ابعاد بردار خروجی آموزش

حال داده‌ها آماده برای دادن به این کلاس‌بندی می‌باشد.

به صورت یک کلاس، کلاس‌بندی Bayes را تعریف می‌کنیم؛ که شامل دو بخش fit و predict می‌باشد. در ابتدا باید داده آموزش را برای این کلاس‌بندی fit کنیم. x در واقع داده‌های ورودی و y برچسب‌ها می‌باشند. با دستور زیر تعداد ورودی‌ها و تعداد ویژگی‌ها رو بررسی می‌کنیم.

```
n_samples, n_features = X.shape
```

با دستور زیر مقادیر مجزا در y را محاسبه می‌کنیم که در واقع تعداد کلاس‌های ما می‌باشد.

```
self._classes = np.unique(y)
n_classes = len(self._classes)
```

برای محاسبه PDF طبق رابطه بیان شده، ما نیاز به میانگین و واریانس داریم و میزان prior نیز برای هر کلاس به صورت مجزا محاسبه می‌گردد. در ابتدا برای fit کردن چندین آرایه خالی می‌سازیم. برای میانگین و واریانس به سایز تعداد کلاس و ویژگی‌ها می‌باشد و برای prior به سایز تعداد کلاس‌ها می‌باشد.

```
self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
self._var = np.zeros((n_classes, n_features), dtype=np.float64)
self._priors = np.zeros(n_classes, dtype=np.float64)
```

درون حلقه for، داده‌هایی که y برابر با کلاس ما دارند را جدا می‌کنیم و مقادیر میانگین، واریانس و prior را محاسبه می‌کنیم و به آرایه‌ی خالی تعریف شده در بالا اضافه می‌کنیم. این حلقه بر روی تمام کلاس‌ها تکرار می‌شود و محاسبه می‌شود.

```
for idx, c in enumerate(self._classes):
  X_c = X[y == c]
  self._mean[idx, :] = X_c.mean(axis=0)
  self._var[idx, :] = X_c.var(axis=0)
  self._priors[idx] = X_c.shape[0] / float(n_samples)
```

برای بخش predict در واقع ما می‌خواهیم یک ورودی بدهیم و خروجی که برچسب می‌باشد را بگیریم. در واقع می‌خواهیم رابطه مربوط به posteriors را محاسبه کنیم. در ابتدا لگاریتم مربوط به prior هر کلاس را محاسبه می‌کنیم. سپس مجموع لگاریتم مربوط به PDF تعریف شده را محاسبه می‌کنیم. و درنهایت رابطه prior را با مجموع لگاریتم مربوط به PDF جمع می‌کنیم. در مرحله بعدی به آرایه posteriors می‌افزایم. و در نهایت ما می‌خواهیم ماکزیمم گیری کنیم. در واقع ما برای هر کلاس به صورت جداگانه رابطه posteriors را محاسبه می‌کنیم و بررسی می‌کنیم مقدار ماکزیمم مربوط به کدام کلاس می‌باشد.

```
def _predict(self, x):
  posteriors = []
  # calculate posterior probability for each class
  for idx, c in enumerate(self._classes):
        prior = np.log(self._priors[idx])
        posterior = np.sum(np.log(self._pdf(idx, x)))
        posterior = posterior + prior
        posteriors.append(posterior)
  # return class with the highest posterior
   return self._classes[np.argmax(posteriors)]
def _pdf(self, class_idx, x):
```

```
        mean = self._mean[class_idx]
        var = self._var[class_idx]
        numerator = np.exp(-((x - mean) ** 2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator
```

در تابع زیر برای هر یک از داده‌ها با استفاده از تابع بالا محاسبه می‌کند و برچسب مربوطه را در درون یک آرایه قرار می‌دهد.

```
    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)
```

علاوه بر کلاس تعریف شده در بالا می‌توانیم از کلاس GaussianNB مربوط به sklearn نیز استفاده کنیم؛ که با مقایسه خروجی‌ها می‌توان مشاهده کرد که یکسان می‌باشند. در ابتدا یک شی از هر دو کلاس ایجاد می‌کنیم. مقادیر X_train و y_train را به تابع fit مربوط به هر دو کلاس می‌دهیم. با استفاده از تابع predict خروجی ماشین مربوط به X_test را دریافت می‌کنیم.

```
MyNB = NaiveBayes()
MyNB.fit(X_train, y_train.ravel())
pred = MyNB.predict(X_test)
from sklearn.naive_bayes import GaussianNB
SKNB=GaussianNB()
SKNB.fit(X_train,y_train.ravel())
pred2 = SKNB.predict(X_test)
```

با استفاده از دستور زیر خروجی‌ها را برای هر دو کلاس نمایش می‌دهیم و مشاهده می‌کنیم که دارای خروجی‌های یکسان می‌باشند و می‌توان بیان نمود هر دو کلاس مشابه می‌باشند.

```
print(pred)
print(pred2)
```

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

شکل ۱۱۷ خروجی‌های هر دو کلاس

یکی از روش‌های ارزیابی این کلاس‌بندی ماتریس درهم‌ریختگی می‌باشد. خود این ماتریس یک ماتریس مربعی است که به تعداد کلاس‌های ما ابعاد دارد (در اینجا دو کلاس داریم پس ابعاد ماتریس درهم‌ریختگی 2 × 2 می‌باشد). خود ماتریس درهم‌ریختگی یک طرح ساده است که تعداد پیش‌بینی‌ها را به چهار دسته تقسیم می‌کند:

مثبت واقعی (TP): تعداد پیش بینی های صحیح که یک نمونه مثبت است.

منفی واقعی (TN): تعداد پیش بینی های صحیح که یک نمونه منفی است.

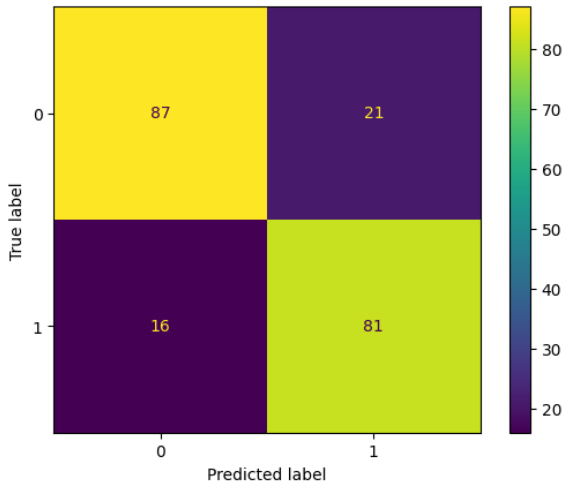
مثبت کاذب (FP): تعداد پیش بینی های نادرست که یک نمونه مثبت است که به عنوان خطای نوع I نیز شناخته می شود (مربوط به کلاس دوم است ولی به نادرستی مربوط به کلاس اول شناسایی شده است).

منفی کاذب (FN): تعداد پیش بینی های نادرست که یک نمونه منفی است که به عنوان خطای نوع II نیز شناخته می شود(مربوط به کلاس اول است ولی به نادرستی مربوط به کلاس دوم شناسایی شده است).

مقادیر بر روی قطر اصلی مقادیری می‌باشند که به درستی پیش‌بینی شده‌اند. این ماتریس روشی ساده برای اندازه‌گیری عملکرد یک مدل طبقه‌بندی ارائه می‌کند که امکان محاسبه معیارهایی مانند accuracy، precision، recall و F1 score را فراهم می‌کند. بین precision و recall یک رابطه معکوس وجود دارد به این معنی که با افزایش یکی، دیگری کاهش می‌یابد. با استفاده از F1 score نقطه بهینه بین این دو را می‌توان یافت. به این معنی که نقطه‌ای که F1 score بیشتری دارد، نقطه بهینه‌تری می‌باشد. برای رسم آن می‌توان از کتابخانه sklearn با دستورات زیر استفاده نمود.

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

```
plt.show()
```



شکل ۱۱۸ ماتریس درهم‌ریختگی

همانطور که مشاهده می‌کنیم ماتریس ما 2 × 2 می‌باشد. همانطور که مشاهده می‌شود ۲۱ داده از کلاس اول به نادرستی به کلاس دوم تشخیص داده شده است و ۱۶ داده از کلاس دوم به نادرستی مربوط به کلاس اول تشخیص داده است.

با استفاده از دستور زیر می‌توانیم یک گزارش از نحوه عملکرد ماشین دریافت کرد، که مقادیر y_test و pred را از ما می‌گیرد و مقادیر accuracy، precision، recall و F1 score را باز می‌گرداند. تعداد داده‌های آزمون دسته اول ۱۰۸ و تعداد داده‌های آزمون دسته دوم ۹۷ می‌باشد.

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

	precision	recall	f1-score	support
0	0.84	0.81	0.82	108
1	0.79	0.84	0.81	97
accuracy			0.82	205
macro avg	0.82	0.82	0.82	205
weighted avg	0.82	0.82	0.82	205

شکل ۱۱۹ گزارش مربوط به classification_report

برخی از معیارها اساساً برای کارهای طبقه‌بندی باینری تعریف شده‌اند (به عنوان مثال F1_Score، ROC_AUC_SCORE). در این موارد، به طور پیش فرض فقط برچسب مثبت ارزیابی می‌شود، با فرض اینکه به طور پیش فرض کلاس مثبت برچسب ۱ خورده است (اگرچه این ممکن است از طریق پارامتر pos_label قابل تنظیم باشد).

در گسترش یک متریک باینری به مشکلات چند برچسبی یا چند خطی ، داده‌ها به عنوان مجموعه‌ای از مشکلات باینری ، یکی برای هر کلاس رفتار می‌شوند. سپس چندین روش برای محاسبات متریک باینری در مجموعه کلاس‌ها وجود دارد که هر یک از آنها ممکن است در برخی از سناریوها مفید باشد. در صورت وجود، باید با استفاده از پارامتر average، بین این موارد را انتخاب کنید.

Macro به سادگی میانگین معیارهای باینری را محاسبه می‌کند و به هر کلاس وزن مساوی می‌دهد. در مسائلی که کلاس‌های نادر با این وجود مهم هستند، میانگین‌گیری macro ممکن است وسیله‌ای برای برجسته کردن عملکرد آنها باشد. از سوی دیگر، این فرض که همه کلاس‌ها به یک اندازه مهم هستند، اغلب نادرست است، به طوری که میانگین‌گیری macro بر عملکرد معمولاً پایین در یک کلاس غیرمتداول بیش از حد تأکید می‌کند.

Micro به هر جفت نمونه-کلاس سهم مساوی در متریک کلی می‌دهد (به جز در نتیجه وزن نمونه). به جای جمع کردن متریک در هر کلاس، این تقسیم‌کننده‌ها و تقسیم‌کننده‌هایی که معیارهای هر کلاس را تشکیل می‌دهند برای محاسبه یک ضریب کلی جمع می‌کند. در تنظیمات چند برچسبی، از جمله طبقه‌بندی چند کلاس که در آن یک کلاس اکثریت نادیده گرفته می‌شود، ممکن است میانگین‌گیری micro ترجیح داده شود.

در زمینه معیارهای ارزیابی مانند F1_score در Scikit-Learn، تفاوت اصلی بین میانگین‌گیری micro و macro در نحوه تجمیع امتیازات در سراسر کلاس‌ها است.

میانگین‌گیری micro با در نظر گرفتن همه نمونه‌های همه کلاس‌ها با هم، متریک را در سطح جهانی محاسبه می‌کند. با هر نمونه به طور مساوی رفتار می‌کند و آن را برای مجموعه داده‌های نامتعادل مناسب می‌کند. دقت و فراخوانی را برای هر کلاس محاسبه می‌کند، سپس میانگین وزنی آنها را محاسبه می‌کند.

میانگین macro متریک را به طور مستقل برای هر کلاس محاسبه می‌کند و سپس بدون در نظر گرفتن عدم تعادل کلاس، میانگین را می‌گیرد. به هر کلاس، بدون توجه به نمایش آنها در مجموعه داده، وزن یکسانی می‌دهد. این رویکرد ممکن است در مجموعه داده‌های نامتعادل عملکرد خوبی نداشته باشد زیرا با همه کلاس‌ها به طور یکسان رفتار می‌کند.

به طور خلاصه، میانگین micro برای مجموعه داده‌های نامتعادل مناسب است ، در حالی که میانگین macro با تمام کلاس‌ها به طور مساوی رفتار می‌کند ، که ممکن است برای مجموعه داده‌های نامتعادل ایده‌آل نباشد.

در ابتدا ۵ ایندکس از مجموعه‌ی داده آزمون را به صورت تصادفی به کمک دستور زیر انتخاب می‌کنیم. از random_seed برای ثابت نگه داشتن ایندکس‌های انتخابی در هر مرحله اجرای برنامه استفاده شده است.

```
import random
# Set the random state
random.seed(76)
# Assuming data_array is your array
array_length = len(y_test)
random_indixes = random.sample(range(array_length), 5)  # Choose 5 random indixes
```

سپس با استفاده از دستور زیر مقادیر واقعی و مقادیر پیش‌بینی شده برای ۵ داده تصادفی را نمایش می‌دهیم. همانطور که مشاهده می‌شود در داده دوم تفاوت وجود دارد.

```
print('true label:',y_test.ravel()[random_indixes])
print('pred label:',pred[random_indixes])
```

```
true label: [1 1 0 0 1]
pred label: [1 0 0 0 1]
```

شکل ۱۲۰ نمایش ۵ داده از مجموعه‌ی آزمون به صورت تصادفی

برای بررسی بیشتر به صورت زیر، تعداد داده‌های در هر کلاس را یکسان می‌کنیم. در ابتدا با استفاده از دستور زیر تعداد داده‌های برچسب‌ها (تعداد کلاس‌ها) را می‌شماریم و نمایش می‌دهیم. zip برای نظیر به نظیر کردن داده‌های دو لیست می‌باشد و dict برای تبدیل به دیکشنری می‌باشد.

```
unique, counts = np.unique(y, return_counts=True)
print("Class counts before balancing:", dict(zip(unique, counts)))
```

با استفاده از کتابخانه زیر به صورت تصادفی داده‌های دیتاست را انتخاب می‌کنیم؛ در واقع کلاسی که تعداد داده کمتر را انتخاب می‌کند و داده‌های همه‌ی کلاس‌ها را به آن تعداد انتخاب می‌کند.

```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=76)
X_resampled, y_resampled = rus.fit_resample(X, y)
unique_resampled, counts_resampled = np.unique(y_resampled, return_counts=True)
print("Class counts after balancing:", dict(zip(unique_resampled, counts_resampled)))
print('\n')
```

```
print("New balanced dataset shape:", X_resampled.shape, y_resampled.shape)
```

```
Class counts before balancing: {0: 499, 1: 526}
Class counts after balancing: {0: 499, 1: 499}


New balanced dataset shape: (998, 13) (998,)
```

شکل ۱۲۱ ابعاد داده‌های بالانس شده

به مانند قبل ۲۰ درصد از داده‌ها را به داده‌های آزمون اختصاص می‌دهیم و ابعاد آن را نمایش می‌دهیم. با استفاده از StandardScaler داده‌های آموزش و آزمون را scaler می‌کنیم.

```
train: (798, 13) (798,)
test:  (200, 13) (200,)
```

شکل ۱۲۲ ابعاد داده‌های آموزش و آزمون

```
[[6.90e+01 1.00e+00 3.00e+00 1.60e+02 2.34e+02 1.00e+00 0.00e+00 1.31e+02
  0.00e+00 1.00e-01 1.00e+00 1.00e+00 2.00e+00]
 [4.70e+01 1.00e+00 2.00e+00 1.08e+02 2.43e+02 0.00e+00 1.00e+00 1.52e+02
  0.00e+00 0.00e+00 2.00e+00 0.00e+00 2.00e+00]
 [7.00e+01 1.00e+00 1.00e+00 1.56e+02 2.45e+02 0.00e+00 0.00e+00 1.43e+02
  0.00e+00 0.00e+00 2.00e+00 0.00e+00 2.00e+00]
 [4.10e+01 1.00e+00 1.00e+00 1.20e+02 1.57e+02 0.00e+00 1.00e+00 1.82e+02
  0.00e+00 0.00e+00 2.00e+00 0.00e+00 2.00e+00]
 [3.50e+01 1.00e+00 1.00e+00 1.22e+02 1.92e+02 0.00e+00 1.00e+00 1.74e+02
  0.00e+00 0.00e+00 2.00e+00 0.00e+00 2.00e+00]]
[[ 1.58487302  0.64801916  2.05969918  1.67644718 -0.24361634  2.37697286
  -1.01948977 -0.79083387 -0.72311878 -0.84652091 -0.62922989  0.21660226
  -0.5719904 ]
 [-0.85177036  0.64801916  1.07312058 -1.35500882 -0.07405993 -0.42070316
   0.85938514  0.15986871 -0.72311878 -0.93203024  1.01708306 -0.74366777
  -0.5719904 ]
 [ 1.69562954  0.64801916  0.08654198  1.44325826 -0.03638073 -0.42070316
  -1.01948977 -0.24757525 -0.72311878 -0.93203024  1.01708306 -0.74366777
  -0.5719904 ]
 [-1.51630946  0.64801916  0.08654198 -0.65544205 -1.69426557 -0.42070316
   0.85938514  1.51801526 -0.72311878 -0.93203024  1.01708306 -0.74366777
  -0.5719904 ]
 [-2.18084857  0.64801916  0.08654198 -0.53884759 -1.03487955 -0.42070316
   0.85938514  1.15584285 -0.72311878 -0.93203024  1.01708306 -0.74366777
  -0.5719904 ]]
```

شکل ۱۲۳ نمایش ۵ عدد از داده‌های آموزش قبل و بعد از scaler

در اینجا نیز ابعاد بردار خروجی آموزش را تغییر می‌دهیم.

```
(798,)
(798,)
```

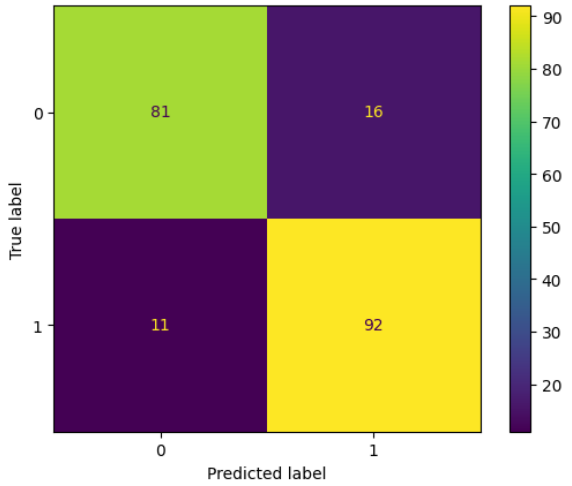
شکل ۱۲۴ ابعاد بردار خروجی آموزش

به مانند قبل با استفاده از کتابخانه و کلاس تعریف شده کلاس‌بندی Bayes را بر روی داده‌ها پیاده‌سازی می‌کنیم. مقادیر پیش بینی شده برای هر دو روش را در زیر می‌توانید مشاهده کنید. همانطور که مشاهده می‌شود یکسان می‌باشند.

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

شکل ۱۲۵ خروجی‌های هر دو کلاس

ماتریس درهم‌ریختگی این حالت را در شکل زیر مشاهده می‌کنید. همانطور که مشاهده می‌کنید نسبت به مرحله قبل تعداد داده‌های نادرست تشخیص داده شده در هر دو کلاس کاهش یافته است.



شکل ۱۲۶ ماتریس درهم‌ریختگی

با استفاده از classification_report می‌توانیم یک گزارش از نحوه عملکرد ماشین دریافت کرد. همانطور که مشاهده می‌شود مقادیر accuracy، precision، recall و F1 score را بهبود یافته است. تعداد داده‌های آزمون دسته اول ۹۷ و دسته‌ی دوم ۱۰۳ می‌باشد.

	precision	recall	f1-score	support
0	0.88	0.84	0.86	97
1	0.85	0.89	0.87	103
accuracy			0.86	200
macro avg	0.87	0.86	0.86	200
weighted avg	0.87	0.86	0.86	200

شکل ۱۲۷ گزارش مربوط به classification_report

در زیر مقادیر واقعی و مقادیر پیش‌بینی شده برای ۵ داده تصادفی با ایندکس انتخابی در قبل را نمایش می‌دهیم. همانطور که مشاهده می‌شود در داده دوم تفاوت وجود دارد.

```
true label: [1 1 0 1 1]
pred label: [1 0 0 1 1]
```

شکل ۱۲۸ نمایش ۵ داده از مجموعه‌ی آزمون به صورت تصادفی