

دانشگاه صنعتی خواجه نصیرالدین طوسی  
دانشکده مهندسی برق - کروه مهندسی گنترل

## درس یادگیری ماشین گزارش مینی پروژه ۲

دانیال عبداللهی نژاد	نام و نام خانوادگی
۴۰۱۰۹۱۶۴	شماره دانشجویی
اردیبهشت ماه ۱۴۰۳	تاریخ



## فهرست مطالب

۴	سوال اول	۱
۴		۱.۱
۵		۲.۱
۶		۳.۱
۱۲	سوال دو	۲
۱۲		۱.۲
۱۴		۲.۲
۲۰		۳.۲
۲۳		۴.۲
۲۶	سوال سوم	۳
۲۶		۱.۳
۳۲		۲.۳
۳۸		۳.۳
۴۰	سوال چهارم	۴
۴۰		۱.۴
۴۶		۲.۴
۴۹		۳.۴



## فهرست تصاویر

۴	.....	فعالساز سیگموید و ReLU	۱
۵	.....	ReLU انواع	۲
۶	.....	ReLU انواع	۳
۷	.....	شبکه طراحی شده	۴
۹	.....	خروجی مدل	۵
۱۰	.....	خروجی مدل McCulloch-Pitts با تابع فعال ساز threshold	۶
۱۰	.....	خروجی مدل McCulloch-Pitts با تابع فعال ساز sigmoid	۷
۱۱	.....	خروجی مدل McCulloch-Pitts با تابع فعال ساز tanh	۸
۱۱	.....	خروجی مدل McCulloch-Pitts با تابع فعال ساز ReLU	۹
۱۸	.....	ساختار مدل MLP	۱۰
۱۹	.....	نمودار اتلاف و دقت برای داده های آموزش و اعتبارسنجی	۱۱
۱۹	.....	Classification Report	۱۲
۲۰	.....	Confusion Matrix	۱۳
۲۱	.....	نمودار اتلاف و دقت برای داده های آموزش و اعتبارسنجی برای تابع اتلاف جدید	۱۴
۲۱	.....	Classification Report mse	۱۵
۲۲	.....	Confusion Matrix mse	۱۶
۲۲	.....	loss CBCE	۱۷
۲۳	.....	روش k-fold cross-validation	۱۸
۲۴	.....	روش stratified k-fold cross-validation	۱۹
۲۵	.....	نتایج cross-validation1	۲۰
۲۵	.....	نتایج cross-validation2	۲۱
۲۷	.....	دقت مدل درخت تصمیمی	۲۲
۲۸	.....	ماتریس درهمریختگی مدل درخت تصمیمی	۲۳
۲۸	.....	درخت تصمیمی ایجاد شده	۲۴
۳۰	.....	دقت مدل درخت تصمیمی	۲۵
۳۰	.....	ماتریس درهمریختگی مدل درخت تصمیمی	۲۶
۳۱	.....	درخت تصمیمی ایجاد شده	۲۷
۳۳	.....	مدل اول	۲۸
۳۳	.....	دقت مدل درخت تصمیمی ۱	۲۹
۳۳	.....	ماتریس درهمریختگی مدل درخت تصمیمی ۱	۳۰
۳۴	.....	درخت تصمیمی ایجاد شده ۱	۳۱
۳۴	.....	مدل دوم	۳۲
۳۵	.....	دقت مدل درخت تصمیمی ۲	۳۳
۳۵	.....	ماتریس درهمریختگی مدل درخت تصمیمی ۲	۳۴

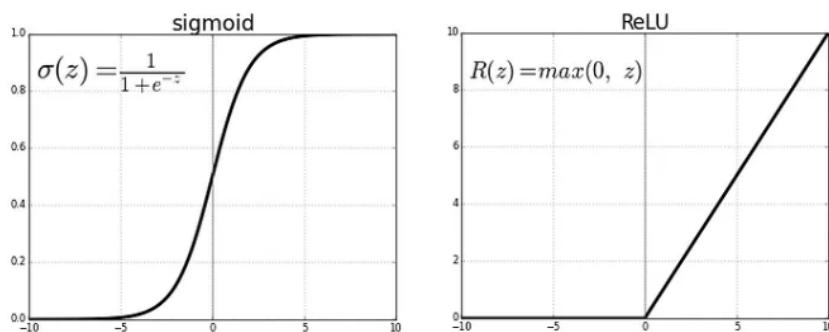


۳۶	.....	درخت تصمیمی ایجاد شده ۲	۳۵
۳۶	.....	مدل سوم	۳۶
۳۷	.....	دقت مدل درخت تصمیمی ۳	۳۷
۳۷	.....	ماتریس درهم ریختگی مدل درخت تصمیمی ۳	۳۸
۳۸	.....	درخت تصمیمی ایجاد شده ۳	۳۹
۳۹	.....	نتایج Random forest	۴۰
۴۰	.....	نتایج Adaboost	۴۱
۴۲	.....	هیستوگرام ویژگی ها	۴۲
۴۵	.....	ماتریس همبستگی	۴۳
۴۷	.....	نتایج مدل ۱	۴۴
۴۸	.....	نتایج مدل ۲	۴۵
۵۰	.....	مقایسه نمونه های مختلف	۴۶

## ۱ سوال اول

۱.۱

فعال ساز sigmoid با یک خروجی بین صفر و یک به ما می‌دهد که در کار طبقه‌بندی دو کلاسه بندی بسیار مطلوب است و معمولاً در این نوع طبقه‌بندی‌ها از این تابع فعال ساز استفاده می‌شود. اگرچه که کار برد دارد اما با یک نگاهی به فضای احتمالاتی، محدوده بزرگی از نقاط ورودی که به سمت مثبت یا منفی بینهایت می‌کنند به خروجی نزدیک به یک یا صفر می‌دهد. به همین دلیل فعال ساز sigmoid برای شبکه‌های عمیق مشکل ساز می‌شود. این مشکل به خاطر وزن‌های ورودی است که در ابتدا مقادیر زیادی داشته باشند. به همین خاطر در فرآیند یادگیری و به روز رسانی وزن‌ها باعث می‌شود گرادیان به مقادیر بسیار کوچک میل کند که اصطلاحاً به آن مشکل گرادیان محو شونده (vanishing gradient) می‌گویند. علاوه بر این، این فعال ساز باعث همگرای بسیار کند مدل می‌شود.

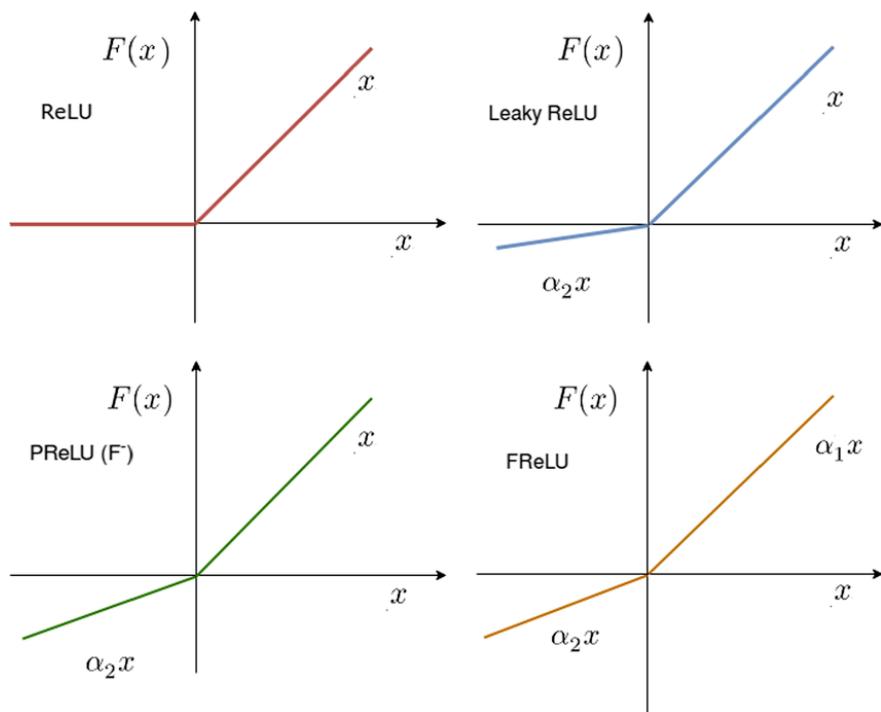


شکل ۱: فعال‌ساز سیگموید و ReLU

همانطور که از شکل ۱ پیداست،ReLU می‌تواند مشکل به اشباع رفتن در مقادیر مثبت را نسبت به sigmoid رفع کند. ولی مشکلی که این فعال ساز وقتی است که نورون یک ورودی منفی دریافت کند، در این صورت خروجی صفر خواهد داشت. در این حالت اصطلاحاً می‌گوییم نورون مرده است و در فرآیند یادگیری شرکت ندارد (dying ReLU). در شبکه‌های عمیق‌تر برای رفع این مشکل از توابعی مانند Leaky ReLU و یا Parametric ReLU که در شکل ۲ آمده است، استفاده می‌شود.

برای رفع مشکل به اشباع رفتن در مقادیر مثبت و منفی، از توابع فعال ساز جدیدتری مانند ReLU Leaky ReLU و Parametric ReLU استفاده می‌شود. در این توابع، مقادیر منفی به جای اینکه به صفر میل کنند، مقدار کمی از شیب را حفظ می‌کنند. این ویژگی باعث می‌شود که نورون‌ها کمتر به حالت مرده (dying) بروند و در فرآیند یادگیری بهبود یابند.

به عنوان مثال، Leaky ReLU با استفاده از شیب کوچک در مقادیر منفی، مشکل مردن نورون‌ها را تا حد زیادی رفع می‌کند. Parametric ReLU (PReLU) به طور مشابه عمل می‌کند، اما با این تفاوت که شیب در مقادیر منفی قابل تنظیم است و می‌توان آن را به عنوان یک پارامتر بهینه‌سازی کرد. Flexible ReLU نوع دیگری است که با افزودن یک بایاس (bias) به تابع، انعطاف‌پذیری بیشتری را فراهم می‌کند.



شکل ۲: انواع ReLU

۲.۱

گرادیان تابع فعالساز ReLU به صورت زیر است.

$$f'(x) = \begin{cases} 0 & \text{اگر } x \leq 0, \\ 1 & \text{اگر } x > 0 \end{cases}$$

همانطور که دیده می‌شود گرادیان این تابع برای مقادیر مثبت ۱ (یعنی رفتار خطی) و برای مقادیر منفی صفر است.  
گرادیان ELU را نشان می‌دهد:

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

همانطور که از رابطه ۲ مشخص است، گرادیان این فعال ساز در مقادیر مثبت مانند ReLU برابر است با ۱ یعنی رفتار این تابع در مقادیر مثبت به صورت خطی است. در مقادیر منفی گرادیان برابر با  $\alpha e^x$  است و در مقادیر خیلی کوچک به صفر میل می‌کند.

از ویژگی های اصلی تابع ReLU و ELU می‌توان به موارد زیر اشاره کرد:

۱. گرادیان غیر صفر در مقادیر منفی:

ReLU: این تابع برای ورودی های منفی گرادیان صفر دارد که منجر به مشکل dying ReLU می‌شود، یعنی نورون ها برای همیشه غیرفعال می‌شوند.

ELU: با داشتن گرادیان غیر صفر در مقادیر منفی، این مشکل را کاهش می‌دهد و اجازه می‌دهد نورون ها حتی با ورودی های منفی به روزرسانی شوند.

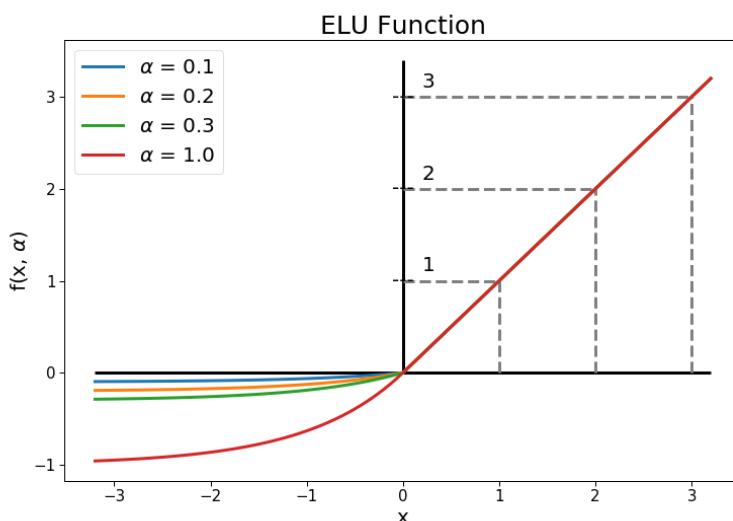
## ۲. مانع از اشباع شدن:

ReLU: مشکل اشباع شدن برای ورودی‌های مثبت را ندارد اما برای ورودی‌های منفی می‌تواند مشکل ساز باشد.

ELU: با حفظ گرادیان غیر صفر، از مشکل اشباع شدن جلوگیری می‌کند و همگراپی سریع‌تر و یادگیری بهتری را فراهم می‌کند.

## ۳. بهبود فرآیند یادگیری:

ReLU و ELU: در هر دو تابع، گرادیان غیر صفر است که باعث بهبود فرآیند یادگیری و به روز رسانی وزن‌ها در الگوریتم گرادیان کاهشی (backpropagation) می‌شود.



شکل ۳: انواع ELU

## ۳.۱

برای این قسمت نیاز است شبکه‌ای برای طبقه‌بندی طراحی کنیم، چون سه شرط در این سوال (شامل سه خط جداکننده برای ایجاد مثلث) مطرح است، از نورون McCulloch-Pitts در لایه اول استفاده می‌کنیم و سپس از یک نورون برای ترکیب نتایج این خط‌ها برای تصمیم‌گیری استفاده می‌کنیم.

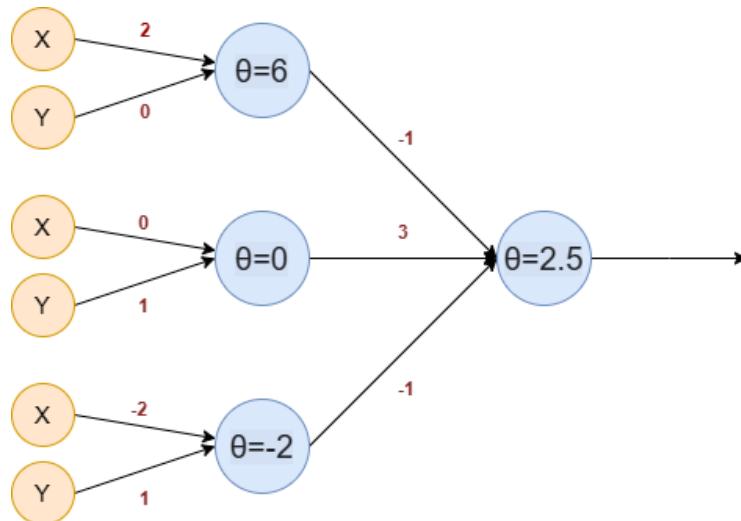
ابتدا نیاز است تا وزن‌ها و Threshold‌ها را مشخص کنیم. با بررسی محاسبات خط‌های AC، AB، BC به ترتیب داریم:

$$2x + y = 6$$

$$y = 0$$

$$-2x + y = -2$$

سپس باید وزن‌های نورون لایه دوم را طوری تنظیم کنیم که وقتی نورون اول خروجی صفر داشت (یعنی خط AB)، و نورون دوم خروجی ۱ داشت (یعنی خط BC)، و نورون سوم خروجی ۰ داشت (یعنی خط AC)، خروجی آخر یک باشد. به طوری که باید برای ورودی‌های صفر از لایه قبل جریمه ورودی بیشتری اختصاص دهیم. در نهایت این نورون با توجه به Treshold به صورت زیر خواهد بود.



شکل ۴: شبکه طراحی شده

کد شبکه طراحی شده در قسمت قبل با استفاده از McCulloch Pitts neuron به صورت زیر است.

```

1 #define McCulloch-Pitts neuron
2 class McCulloch_Pitts_neuron():
3
4     def __init__(self, weights, threshold):
5         self.weights = weights
6         self.threshold = threshold
7
8     def model(self, x):
9         if self.weights @ x >= self.threshold:
10             return 1
11         else:
12             return 0
13
14 #define model for dataset
15 def Area(x, y):
16     neur1 = McCulloch_Pitts_neuron([2, 1], 6)
17     neur2 = McCulloch_Pitts_neuron([0, 1], 0)
18     neur3 = McCulloch_Pitts_neuron([-2, 1], -2)
19     neur4 = McCulloch_Pitts_neuron([-1, 3, -1], 2.5)
20
21     z1 = neur1.model(np.array([x, y]))

```



```

22     z2 = neur2.model(np.array([x, y]))
23     z3 = neur3.model(np.array([x, y]))
24     z4 = neur4.model(np.array([z1, z2, z3]))
25
26     return list([z4])

```

Code 1: Designed Network

با استفاده از مطالب آموزش داده شده به صورت زیر داده رندوم را تولید می‌کنیم.

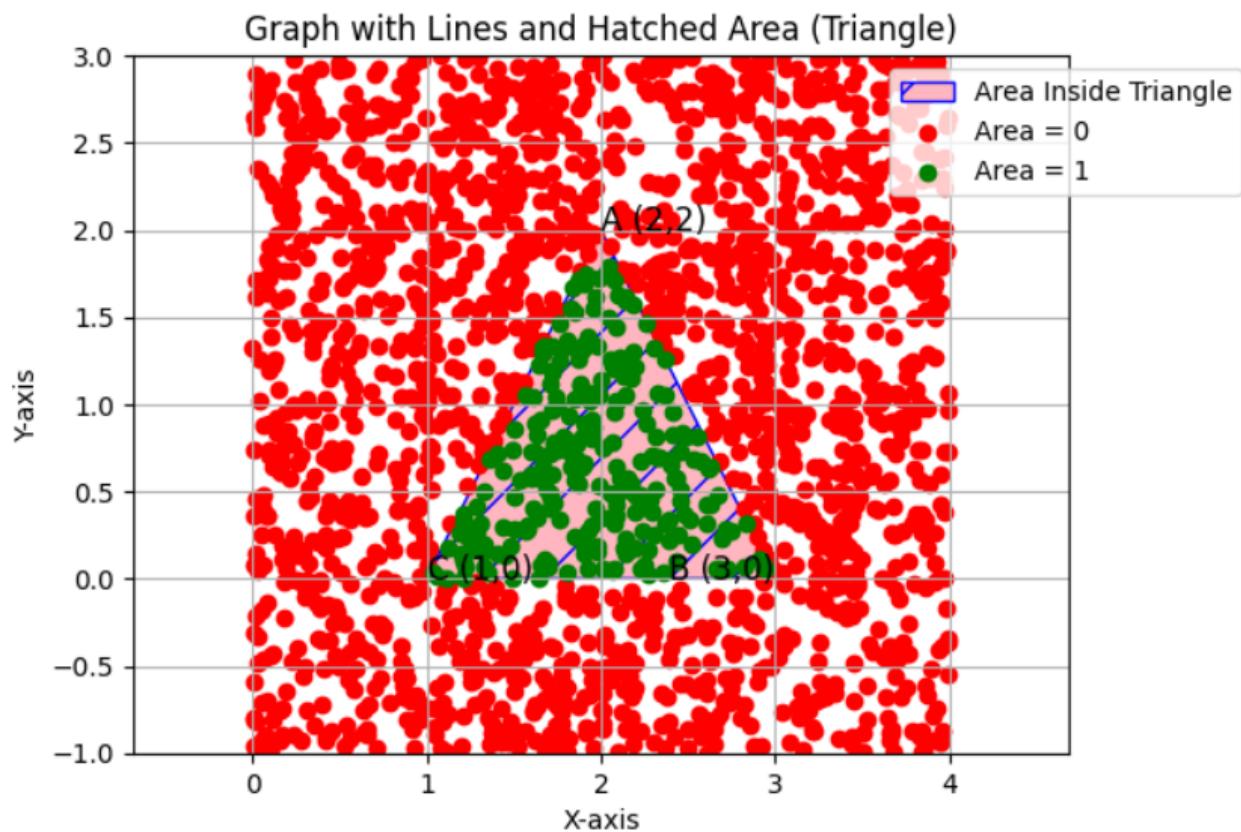
```

1 random_seed = 64
2 np.random.seed(random_seed)
3
4 # Generate random data points
5 num_samples = 2000
6 x_samples = np.random.uniform(0, 4, num_samples)
7 y_samples = np.random.uniform(-1, 3, num_samples)
8
9 area_zero_points = []
10 area_one_points = []
11
12 for i in range(num_samples):
13     area_value = Area(x_samples[i], y_samples[i])
14     if area_value == [0]:
15         area_zero_points.append((x_samples[i], y_samples[i]))
16     else:
17         area_one_points.append((x_samples[i], y_samples[i]))
18
19 # Separate x and y values for area_zero and area_one points
20 area_zero_x, area_zero_y = zip(*area_zero_points)
21 area_one_x, area_one_y = zip(*area_one_points)
22
23 # Plotting
24 plt.figure(figsize=(8, 6))
25 plt.scatter(area_zero_x, area_zero_y, color='red', label='Area = 0')
26 plt.scatter(area_one_x, area_one_y, color='green', label='Area = 1')
27 plt.xlabel('X values')
28 plt.ylabel('Y values')
29 plt.title('McCulloch-Pitts Neuron Outputs')
30 plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1.0))
31 plt.show()

```

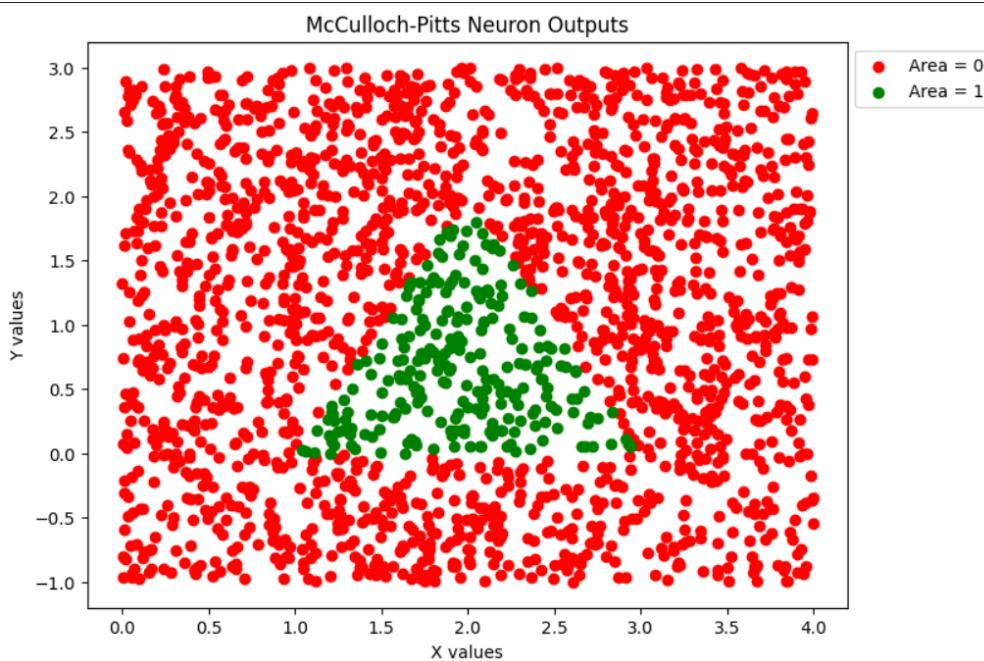
Code 2: Data generation

حال به بررسی نتایج می‌پردازیم. همانطور که دیده می‌شود، مدل به خوبی کار طبقه‌بندی را انجام داده و نقاط داخل مثلث همگی با رنگ سبز و لیبل ۱ مشخص شده‌اند و نقاط خارج از آن با رنگ قرمز و لیبل صفر هستند. شبکه به خوبی در جدا کردن داده‌ها در مثلث عمل می‌کند.

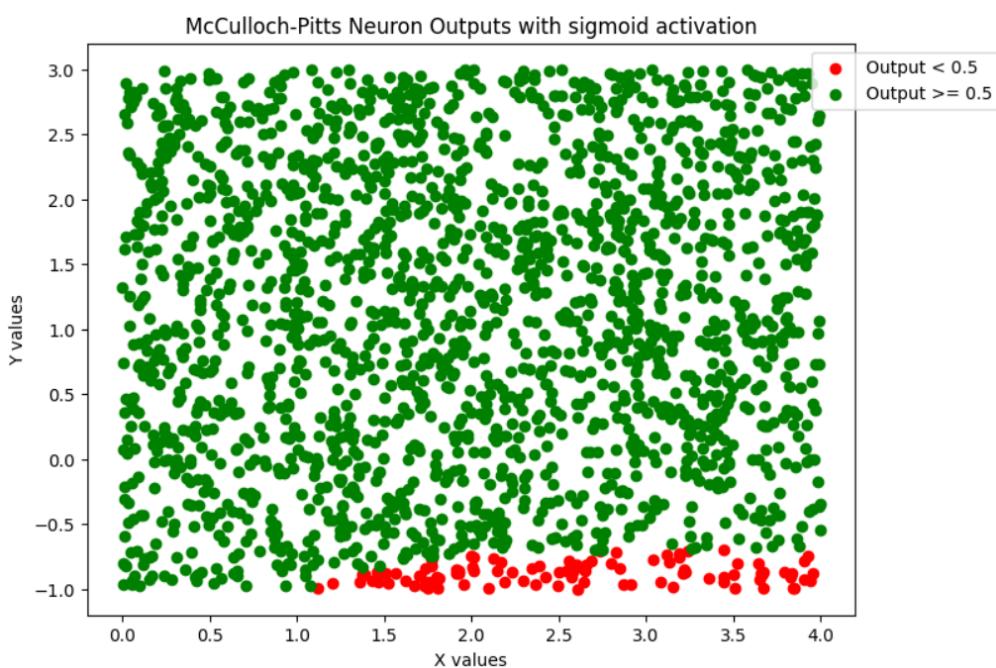


شکل ۵: خروجی مدل

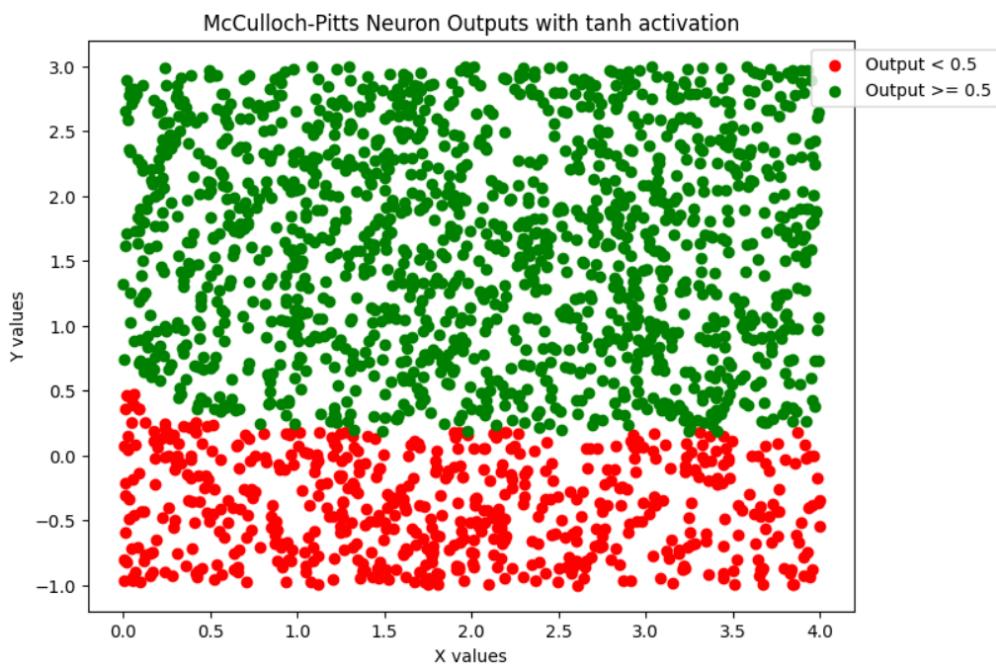
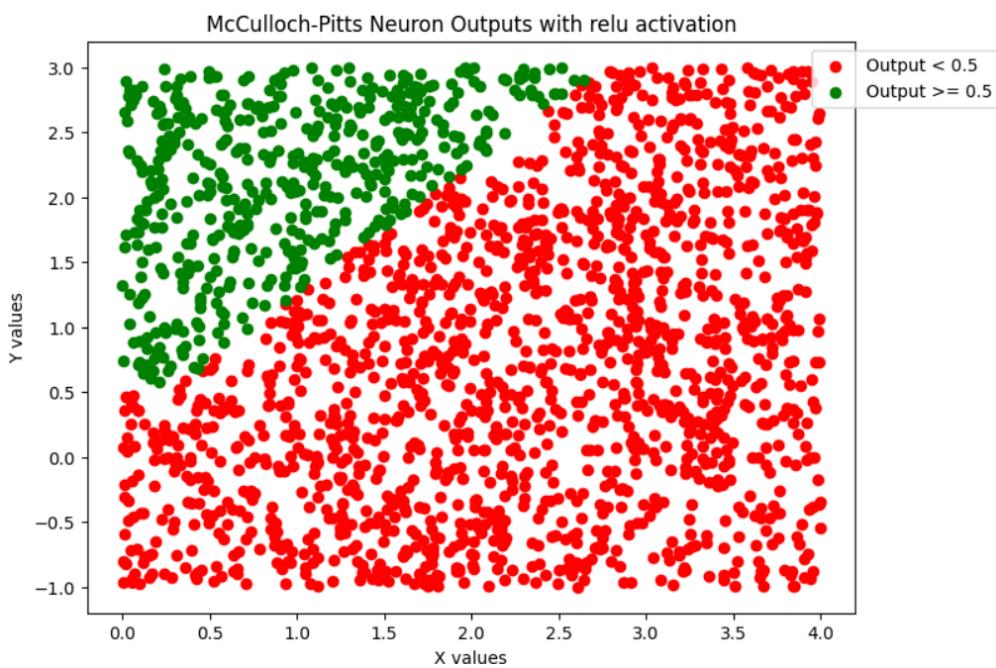
در این بخش، تأثیر انواع activation function های مختلف را بر عملکرد مدل بررسی می کنیم. برای انجام این کار، یک ورودی جدید به کلاس McCulloch-Pitts اضافه می کنیم.  
برای این کار توابع فعالساز sigmoid, ReLU, tanh مورد ارزیابی و مقایسه قرار می دهیم.



شکل ۶: خروجی مدل McCulloch-Pitts با تابع فعال‌ساز threshold



شکل ۷: خروجی مدل McCulloch-Pitts با تابع فعال‌ساز sigmoid

شکل ۸: خروجی مدل McCulloch-Pitts با تابع فعال‌ساز  $\tanh$ 

شکل ۹: خروجی مدل McCulloch-Pitts با تابع فعال‌ساز ReLU

با تغییر توابع فعال‌ساز، عملکرد کلی شبکه تغییر می‌کند و باید در طراحی شبکه به این توابع توجه ویژه‌ای شود. در این حالت، چون شبکه با استفاده از threshold طراحی شده بود، همین تابع فعال‌ساز بهترین عملکرد را ارائه می‌دهد و در حالت‌های دیگر، طبقه‌بندی به خوبی انجام نمی‌شود و تنها بخش‌هایی از عملیات طبقه‌بندی انجام می‌گیرد.



## ۲ سوال دوم

۱.۲

مجموعه داده CWRU که در محیط آزمایشگاهی تولید شده، شامل داده‌های بلبرینگ‌ها استفاده شده در موتور القایی Electric Reliance با توان دو اسب بخار می‌باشد. این سیستم شامل یک ترانسدیوسر گشتاور، یک دینامومتر و واحد کنترلی است. داده‌ها انواع متفاوتی از خرابی‌ها را پوشش می‌دهند و به چهار دسته تقسیم شده‌اند: عادی در ۴۸ کیلوهرتز، عیب در سمت درایو در ۴۸ کیلوهرتز، عیب در سمت درایو در ۱۲ کیلوهرتز، و عیب در سمت فن در ۱۲ کیلوهرتز.

این دسته‌بندی‌ها شامل زیر مجموعه‌هایی برای شناسایی نوع خرابی‌ها هستند: - عیب بلبرینگ (B) - خرابی‌های داخلی - خرابی‌های خارجی، که بر اساس موقعیت نسبی نسبت به ناحیه باردهی دسته‌بندی شده‌اند: مرکزی (موقعیت ساعت ۶:۰۰)، عمودی (موقعیت ساعت ۳:۰۰)، و مخالف (موقعیت ساعت ۱۲:۰۰).

این عیوب با استفاده از فرآیند ماشینکاری الکترو-تخربی (EDM) بر روی بلبرینگ‌های آزمایشی ایجاد شده‌اند و با قطرهای مختلف مشخص می‌شوند، مانند ۷، ۱۴، ۲۱، ۲۸ و ۴۰ میلی‌متر.

داده‌ها با دو فرکانس نمونه‌برداری مختلف، ۱۲ و ۴۸ کیلوهرتز جمع‌آوری شده‌اند، و اطلاعات ارتعاشی برای بارهای موتور در محدوده ۰ تا ۳ اسب بخار، در سرعت‌های موتوری بین ۱۷۲۰ تا ۱۷۹۷ دور در دقیقه (RPM) ثبت شده‌اند.

داده‌های انتخاب شده در سه کلاس عیب‌دار و یک کلاس سالم هستند. کلاس‌های عیب به ترتیب شامل خرابی‌های داخلی، عیب بلبرینگ و خرابی‌های خارجی که در موقعیت مرکزی (موقعیت ساعت ۶) است. همگی این عیوب‌ها در سمت درایو در حالت نرم نمونه‌برداری ۱۲ کیلوهرتز هستند.

● آ) با توجه به باقی مانده شماره دانشجویی بر ۴، داده‌های OR007@6\_B007\_0، IR007\_0، Normal\_0 و ۰@6\_B007\_0 را دانلود می‌کنیم که از این سری داده‌ها، قسمت‌های مربوط به عیب سمت درایو را انتخاب می‌کنیم. برای ایجاد دیتای مورد نظر از  $M=200$  و  $N=300$  استفاده می‌کنیم و داده‌ها را زیر هم قرار می‌دهیم تا یک ماتریس  $800 \times 300$  برسیم. همچنین برای اینکه دیتای انتخاب شده در اجراهای مختلف تغییر نکند از یک random state استفاده می‌کنیم.

```
1 # Load MATLAB file
2 normal_bearings_data = loadmat('/content/drive/MyDrive/ML2024/MP2/Q2/97.mat')
3 faulty_bearings_data1 = loadmat('/content/drive/MyDrive/ML2024/MP2/Q2/105.mat')
4 faulty_bearings_data2 = loadmat('/content/drive/MyDrive/ML2024/MP2/Q2/118.mat')
5 faulty_bearings_data3 = loadmat('/content/drive/MyDrive/ML2024/MP2/Q2/130.mat')
6
7 normal_data = normal_bearings_data['X097_DE_time']
8 fault_data1 = faulty_bearings_data1['X105_DE_time']
9 fault_data2 = faulty_bearings_data2['X118_DE_time']
10 fault_data3 = faulty_bearings_data3['X130_DE_time']
11
12 def select_samples(data, M, N, random_state=None):
13     if random_state is not None:
14         np.random.seed(random_state)
15
16     samples = []
17     for _ in range(M):
```



```

18     start_index = np.random.randint(0, len(data) - N)
19     sample = data[start_index:start_index + N]
20     samples.append(sample)
21
22
23 M = 200
24 N = 300
25 normal_samples = select_samples(normal_data, M, N, 64)
26 faulty_samples1 = select_samples(fault_data1, M, N, 64)
27 faulty_samples2 = select_samples(fault_data2, M, N, 64)
28 faulty_samples3 = select_samples(fault_data3, M, N, 64)
29
30 labels_normal = np.zeros(M)
31 labels_faulty1 = np.ones(M)
32 labels_faulty2 = 2*np.ones(M)
33 labels_faulty3 = 3*np.ones(M)
34
35 normal_samples_flat = normal_samples.reshape(M, -1)
36 faulty_samples_flat1 = faulty_samples1.reshape(M, -1)
37 faulty_samples_flat2 = faulty_samples2.reshape(M, -1)
38 faulty_samples_flat3 = faulty_samples3.reshape(M, -1)
39
40 data_combined = np.vstack((normal_samples_flat, faulty_samples_flat1, faulty_samples_flat2,
41                             faulty_samples_flat3))
42
43 labels_combined = np.concatenate((labels_normal, labels_faulty1, labels_faulty2,
44                                   labels_faulty3))
45 df = pd.DataFrame(data_combined)
46 df['label'] = labels_combined

```

Code 3: Data preparation

- ب) مطابق سری قبل ویژگی‌های داده را استخراج می‌کنیم و با آن را استانداردسازی می‌کنیم. برای تقسیم داده به سه دسته به شکل زیر عمل می‌کنیم.

```

1 from sklearn.model_selection import train_test_split
2
3 X_train, X_rem, y_train, y_rem = train_test_split(X, y,
4                                                 train_size=0.6, random_state=64, shuffle=
5                                                 True)
6
7 X_valid, X_test, y_valid, y_test = train_test_split(X_rem, y_rem,
8                                                 test_size=0.5, random_state=64, shuffle=
9                                                 True)

```

Code 4: Splitting data



همانطور که مشخص است داده‌ها با نسبت ۶۰ درصد داده آموزش، ۲۰ درصد اعتبارسنجی و ۲۰ درصد آزمون به سه دسته تقسیم شده‌اند.

فرایند آموزش یک فرایند یک مرحله‌ای نیست و برای دستیابی به بهترین مدل ممکن نیاز به تکرار و سعی و خطا وجود دارد. با توجه به اینکه داده آزمون تنها برای سنجش مدل استفاده می‌شود، از یک سری داده تحت عنوان اعتبارسنجی استفاده می‌کنیم تا پیش از آزمودن مدل با استفاده از آن ترکیب‌های مختلف از هایپرپارامترها را بسنجیم و بهترین مدل را پیدا کیم.

## ۲.۲

در این قسمت به پیاده‌سازی یک مدل MLP می‌پردازیم و سعی داریم با استفاده از داده اعتبارسنجی با آزمون و خطا سعی می‌کنیم بهترین مدل را ایجاد کنیم. کد پیاده‌سازی MLP و فرایند آموزش و گرادیان و ... به صورت زیر است.

```
1 ## Activation Function
2 def relu(x):
3     return np.maximum(0, x)
4
5 def sigmoid(x):
6     return 1 / (1 + np.exp(-x))
7
8
9 ## Loss
10 def bce(y, y_hat):
11     return np.mean(-(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat)))
12
13 def mse(y, y_hat):
14     return np.mean((y - y_hat)**2)
15
16 def categorical_cross_entropy(y, y_hat):
17     return -np.mean(np.sum(y * np.log(y_hat), axis=1))
18
19
20 ## Accuracy
21 def accuracy(y, y_hat, t=0.5):
22     y_hat = np.where(y_hat < t, 0, 1)
23     acc = np.sum(y == y_hat) / len(y)
24     return acc
25
26 class MLP:
27     def __init__(self, hidden_layer_sizes, hidden_activation='relu',
28                  output_size=1, output_activation='sigmoid',
29                  n_iter=1000, loss_fn=bce, eta=0.1, random_state=None):
30         self.hidden_layer_sizes = hidden_layer_sizes
31         self.hidden_activation = hidden_activation
32         self.output_size = output_size
```



```

33         self.output_activation = output_activation
34
35         self.n_iter = n_iter
36
37         self.loss_fn = loss_fn
38
39         self.eta = eta
40
41         self.random_state = random_state
42
43         np.random.seed(self.random_state) # Set random seed
44
45
46     def _init_weights(self):
47
48         self.ws, self.bs = [], [] # Weight and bias lists for each layer
49         self.as_ = [None] * len(self.hidden_layer_sizes) # Initialize as_ with None
50
51         all_layers = [self.input_size] + self.hidden_layer_sizes + [self.output_size] # All
52         layer sizes
53
54         num_layers = len(all_layers)
55
56         for i in range(1, num_layers):
57
58             w = np.random.randn(all_layers[i-1], all_layers[i]) # Randomly initialize
59             weights
60
61             b = np.random.randn(all_layers[i]) # Randomly initialize biases
62
63             self.ws.append(w)
64
65             self.bs.append(b)
66
67
68     def fit(self, X, y, X_val=None, y_val=None): # Add optional validation data
69
70         n, self.input_size = X.shape
71
72         self._init_weights()
73
74         train_losses = []
75
76         val_losses = []
77
78         train_accs = []
79
80         val_accs = []
81
82         for _ in range(self.n_iter):
83
84             y_hat = self.predict(X)
85
86             loss = self.loss_fn(y, y_hat)
87
88             self._gradient_descent(X, y, y_hat)
89
90             train_losses.append(loss)
91
92             train_acc = accuracy(y, y_hat)
93
94             train_accs.append(train_acc)
95
96             if X_val is not None and y_val is not None:
97
98                 val_loss = self.loss_fn(y_val, self.predict(X_val))
99
100                val_losses.append(val_loss)
101
102                val_acc = accuracy(y_val, self.predict(X_val))
103
104                val_accs.append(val_acc)
105
106                print(f"Train Loss: {loss:.4f} | Train Acc: {train_acc:.4f} | Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.4f}")
107
108            else:
109
110                print(f"Train Loss: {loss:.4f} | Train Acc: {train_acc:.4f}")
111
112            if X_val is not None and y_val is not None:
113
114                self.plot_history(train_losses, val_losses, train_accs, val_accs)
115
116
117
118
119
120
121
122
123
124

```



```
75     else:
76         self.plot_history(train_losses, None, train_accs, None)
77
78
79     def plot_history(self, train_losses, val_losses=None, train_accs=None, val_accs=None):
80         plt.figure(figsize=(12, 6))
81         plt.subplot(1, 2, 1)
82         plt.plot(train_losses, label='Training Loss')
83         if val_losses is not None:
84             plt.plot(val_losses, label='Validation Loss')
85         plt.xlabel('Epoch')
86         plt.ylabel('Loss')
87         plt.title('Training and Validation Loss')
88         plt.legend()
89
90         plt.subplot(1, 2, 2)
91         plt.plot(train_accs, label='Training Accuracy')
92         if val_accs is not None:
93             plt.plot(val_accs, label='Validation Accuracy')
94         plt.xlabel('Epoch')
95         plt.ylabel('Accuracy')
96         plt.title('Training and Validation Accuracy')
97         plt.legend()
98
99         plt.show()
100    def _gradient_descent(self, X, y, y_hat):
101        delta = y_hat - y # Compute difference between predicted and true values
102        for j in range(len(self.ws)-1, 0, -1):
103            w_grad = (self.as_[j-1].T @ delta) / len(y) # Compute weight gradient
104            b_grad = delta.mean(0) # Compute bias gradient
105            self.ws[j] -= self.eta * w_grad # Update weights
106            self.bs[j] -= self.eta * b_grad # Update biases
107            delta = (delta @ self.ws[j].T) * (self._activation_derivative(self.hs[j-1], self.hidden_activation))
108
109    def predict(self, X):
110        self.hs = [] # Hidden layer outputs
111        self.as_ = [] # Activation function outputs
112        for i, (w, b) in enumerate(zip(self.ws[:-1], self.bs[:-1])):
113            a = self.as_[i-1].copy() if i>0 else X.copy() # Input to the hidden layer
114            self.hs.append(a @ w + b) # Compute hidden layer output
115            self.as_.append(self._activation_function(self.hs[i], self.hidden_activation))
116        # Apply activation function
117        y = self._activation_function(self.as_[-1] @ self.ws[-1] + self.bs[-1], self.output_activation) # Output layer activation
```



```

117     return y
118
119     def _activation_function(self, x, activation):
120         if activation == 'relu':
121             return np.maximum(0, x) # ReLU activation
122         elif activation == 'sigmoid':
123             return 1 / (1 + np.exp(-x)) # Sigmoid activation
124         else:
125             raise ValueError("Invalid activation function.")
126
127     def _activation_derivative(self, x, activation):
128         if activation == 'relu':
129             return np.where(x > 0, 1, 0) # Derivative of ReLU activation
130         elif activation == 'sigmoid':
131             sigmoid = self._activation_function(x, 'sigmoid')
132             return sigmoid * (1 - sigmoid) # Derivative of Sigmoid activation
133         else:
134             raise ValueError("Invalid activation function.")

```

Code 5: MLP

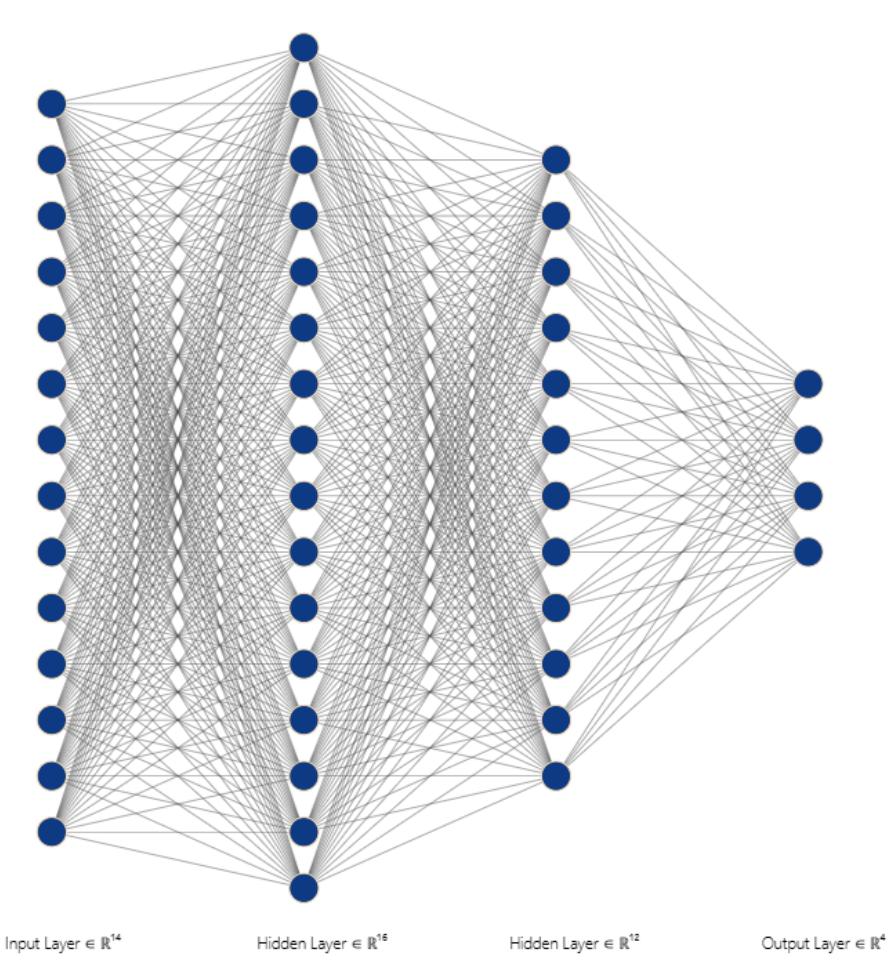
ساختر بهترین مدل ایجاد شده به صورت زیر است.

```

1 mlp = MLP(hidden_layer_sizes=[16,12], hidden_activation='relu', output_size=4,
           output_activation='sigmoid', n_iter=2000, loss_fn=bce, eta=0.06, random_state=64)

```

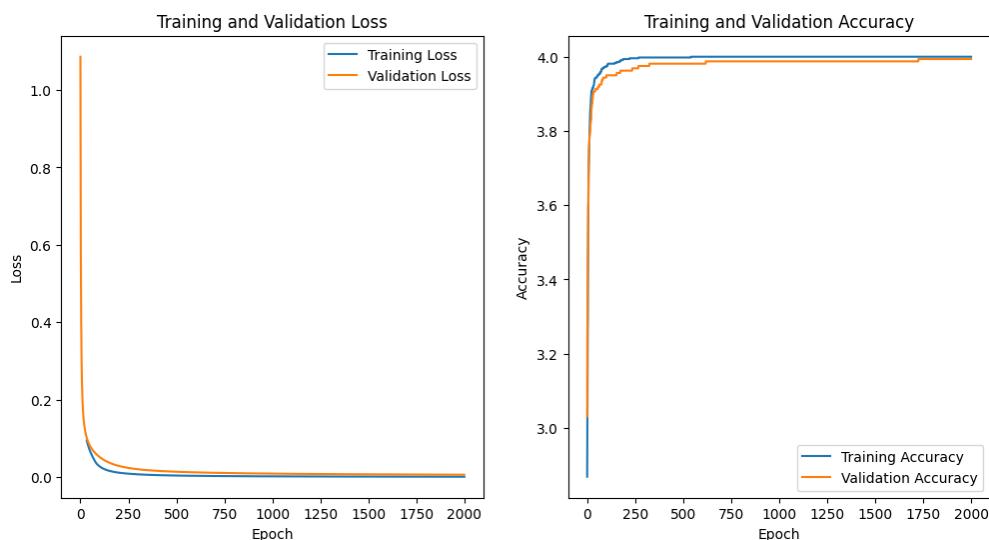
Code 6: Splitting data



شکل ۱۰: ساختار مدل MLP

همانطور که مشخص است مدل از دو لایه پنهان شامل ۱۶ و ۱۲ نod تشکیل شده است. در لایه های میانی از توابع فعال ساز Relu استفاده می کنیم و در لایه خروجی از تابع سیگموید بهره می گیریم.

در لایه خروجی ۴ نod داریم که هر کدام برای مشخص کردن یکی از ۴ کلاس داده ما استفاده می شوند (یک کلاس سالم و سه کلاس عیب). تابع اتلافی که در این قسمت استفاده می شود، تابع BCE (Binary Cross Entropy) است. با توجه به اهمیت داده اعتبارسنجی برای بدست آوردن مقادیر های پارامترها از آن بهره می گیریم و بدین ترتیب مدل را بهبود می دهیم. نتایج مرحله آموزش برای داده های آموزش و اعتبارسنجی به صورت زیر است.



شکل ۱۱: نمودار اتلاف و دقت برای داده‌های آموزش و اعتبارسنجی

مقادیر خطأ و دقت مدل در آخرین تکرار به صورت زیر است:

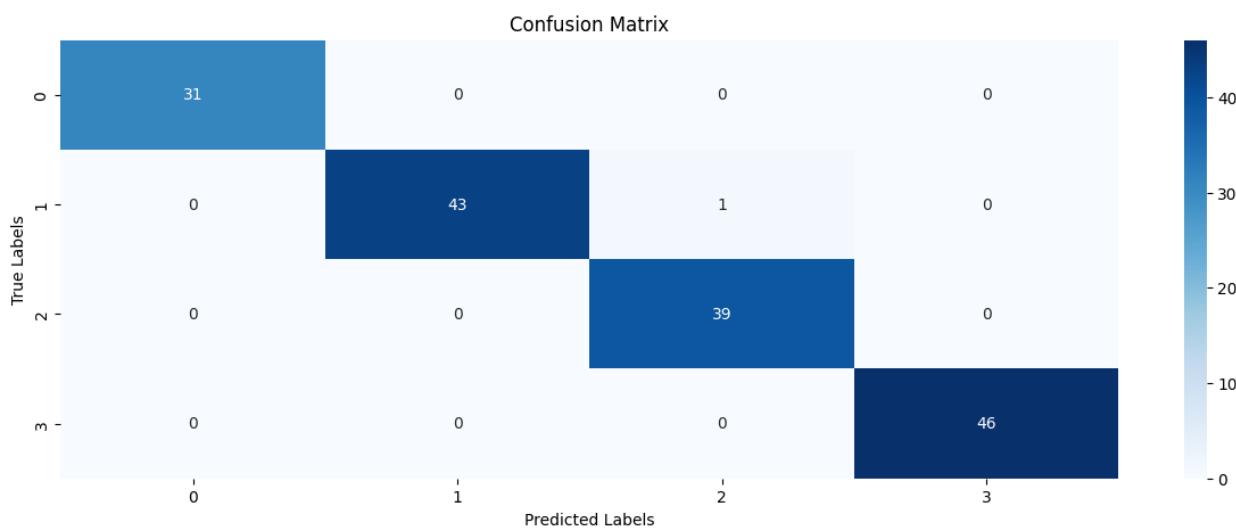
Train Loss: 0.0009 | Train Acc: 4.0000 | Val Loss: 0.0061 | Val Acc: 3.9937

همانطور که از نتایج مشخص است، مدل به خوبی فرایند یادگیری را سپری کرده است و نه دچار Overfit شده‌ایم و نه خطای زیادی داریم. حال برای ارزیابی دقیق از داده آزمون استفاده می‌کنیم.

از مدل آموزش داده شده در مرحله قبل برای ارزیابی نهایی توسط داده آزمون استفاده می‌کنیم که نتایج آن به صورت زیر است:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	31
1	1.00	0.98	0.99	44
2	0.97	1.00	0.99	39
3	1.00	1.00	1.00	46
accuracy			0.99	160
macro avg	0.99	0.99	0.99	160
weighted avg	0.99	0.99	0.99	160

شکل ۱۲ Classification Report :۱۲



شکل ۱۳: Confusion Matrix

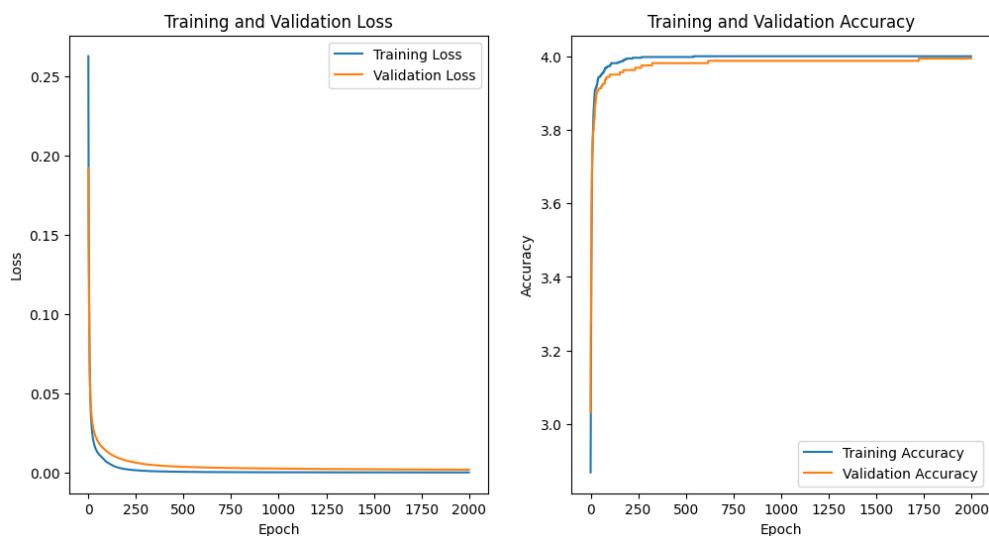
با توجه به نتایج Classification Report مشاهده می‌شود که به طور کلی دقت ۹۹ درصد را در داده آزمون بدست آورده‌ایم. با نگاهی دقیق‌تر در می‌یابیم که در معیار precision در کلاس صفر، یک و سه دقت ۱۰۰ درصد را داریم، این به این معنی است که تمامی نمونه‌های موجود در این سه کلاس را تشخیص داده‌ایم و نمونه‌ای در این سه کلاس نبود که تشخیص داده نشود اما در کلاس ۱ یک نمونه داریم که اشتباه‌آوری کلاس ۰ تشخیص داده شده‌است. معیار recall درست خلاف آن است. در این معیار کلاس‌های ۰، ۱ و ۳ به صورت ۱۰۰ درصد دقت داشته‌اند. دو کلاسی که به در هر دو معیار ۱۰۰ درصد دقت داشته یا دقت کلی ۱۰۰ درصد داشته‌است کلاس ۰ و ۳ است. به این معنی که تمامی نمونه‌های این کلاس به درستی تشخیص داده شده‌اند.

باید توجه داشت که مدل ما کلاس‌های عیب و سالم را در هیچ نمونه‌ای اشتباه تشخیص نداده است که می‌توان گفت این مورد اهمیت بیشتری برای ما دارد. تنها خطای مدل اشتباه گرفتن کلاس عیب یک و دو است که در یک مورد کلاس عیب ۱ به اشتباه ۲ تشخیص داده شده‌است.

## ۳.۲

در این قسمت با حفظ ساختار مدل و تغییر توابع فعالساز و اتلاف نتایج را بررسی می‌کنیم.

در این قسمت از تابع اتلاف (MSE) (Mean Square Error) استفاده می‌کنیم. نتایج به صورت زیر است:



شکل ۱۴: نمودار اتلاف و دقت برای داده‌های آموزش و اعتبارسنجی برای تابع اتلاف جدید

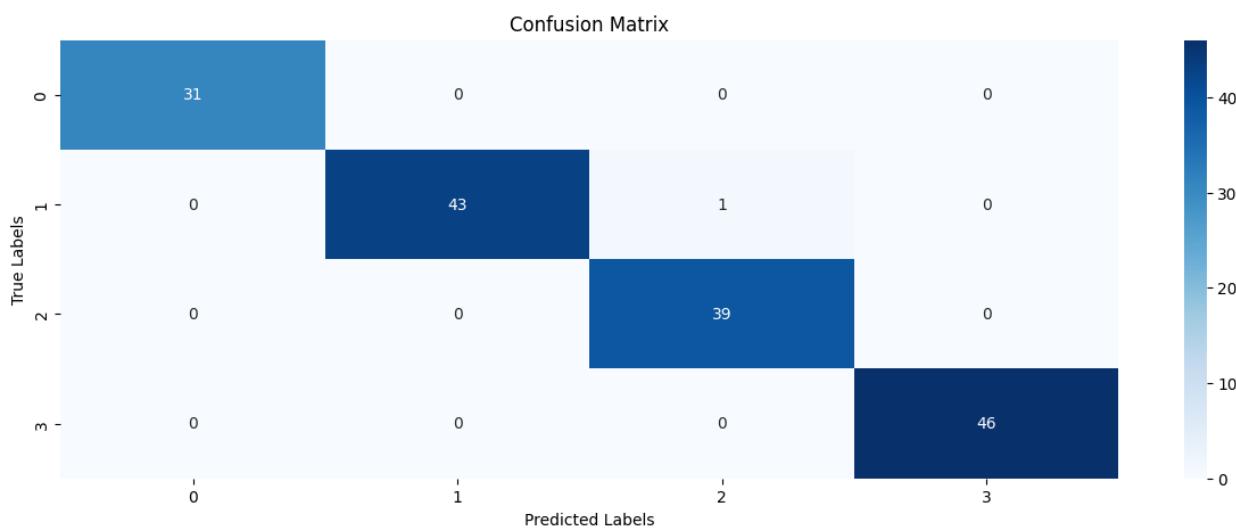
همچنین مقادیر خطأ و دقت برای داده‌های آموزش و اعتبارسنجی به صورت زیر است:

Train Loss: 0.0000 | Train Acc: 4.0000 | Val Loss: 0.0018 | Val Acc: 3.9937

مشاهده می‌شود که مدل به تقریباً به طور مشابه به خوبی فرایند یادگیری را طی می‌کند. تنها تفاوتی که در فرایند یادگیری وجود دارد بهبود جزئی خطأ و دقت در هر دو داده آموزش و اعتبارسنجی است. حال برای بررسی عملکرد مدل با استفاده از داده آزمون، مدل را می‌سنجمیم.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	31
1	1.00	0.98	0.99	44
2	0.97	1.00	0.99	39
3	1.00	1.00	1.00	46
accuracy			0.99	160
macro avg	0.99	0.99	0.99	160
weighted avg	0.99	0.99	0.99	160

شکل ۱۵: Classification Report mse



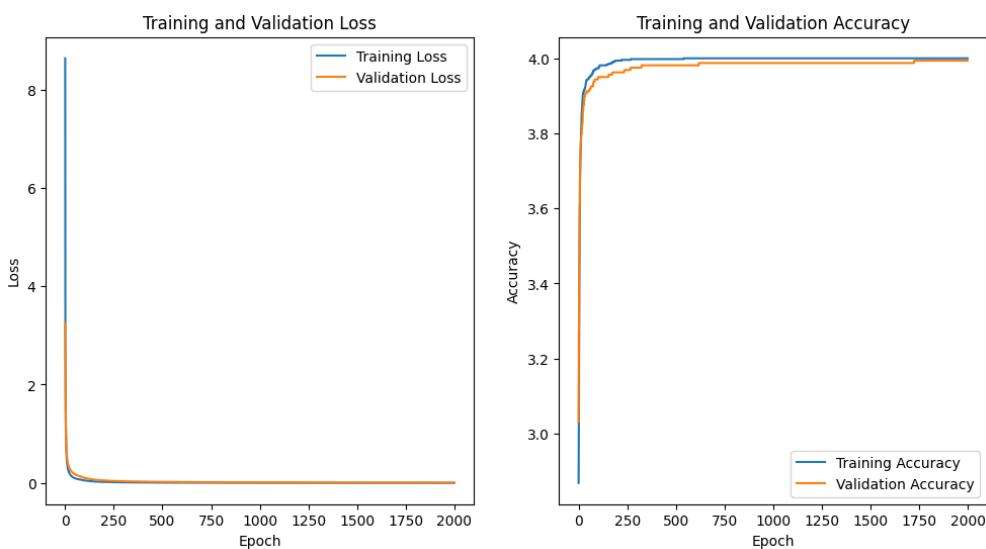
شکل ۱۶: Confusion Matrix mse

همانطور که مشخص است، دقت در داده آزمون تغییری نکرده است و مشابه با قسمت قبل است.

حال با Categorical Cross-Entropy Loss بررسی می‌کنیم.

$$\text{Categorical Cross-Entropy}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C y_{ij} \log(\hat{y}_{ij}) \quad (1)$$

نتایج و خطاهای به صورت زیر است.



شکل ۱۷: loss CBCE

Train Loss: 0.0017 | Train Acc: 4.0000 | Val Loss: 0.0070 | Val Acc: 3.9937

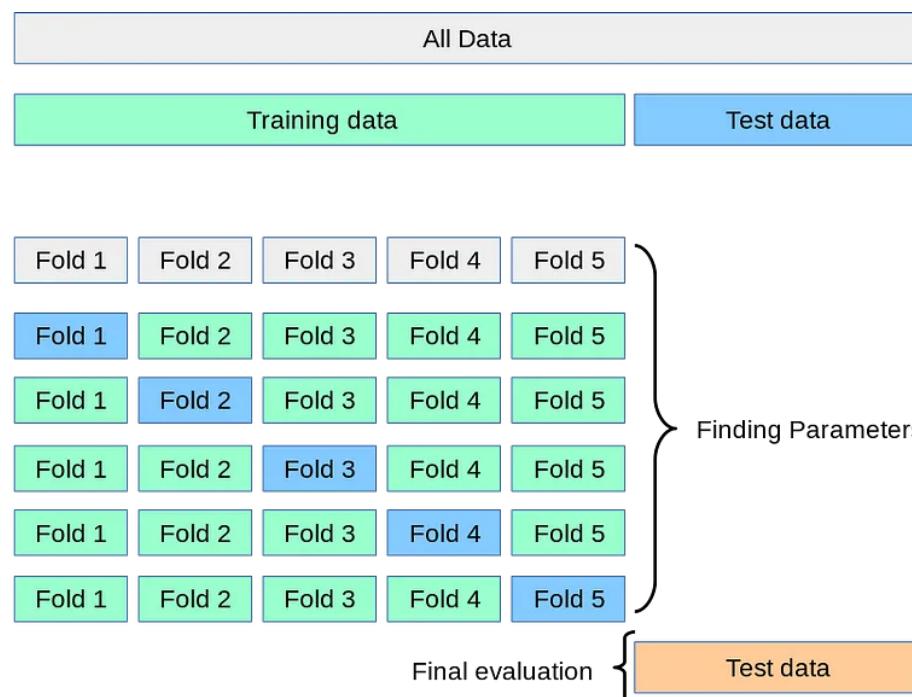
در این مورد هم خطای تغییر زیادی نداشت و نتایج داده آزمون نیز مشابه است.

به طور کلی تغییر تابع اتلاف در موارد مختلف می‌تواند تاثیرگذار باشد. در مدل ما نیز در دقت آموزش تاثیر کمی داشت ولی در مسائل و شبکه‌های متفاوت می‌تواند تاثیرگذارتر باشد.

## ۴.۲

روش k-fold cross-validation تکنیکی است که در آن داده‌ها به  $k$  قسمت یا fold برابر تقسیم می‌شوند. مدل روی  $(k-1)$  fold آموزش داده می‌شود و روی fold باقی‌مانده ارزیابی می‌شود. این فرآیند  $k$  بار تکرار می‌شود و هر fold یکبار به عنوان مجموعه اعتبارسنجی عمل می‌کند. عملکرد متوسط مدل در تمام  $k$  تکرار برای ارزیابی عملکرد کلی مدل استفاده می‌شود.

- این روش تضمین می‌کند که هر مشاهده از مجموعه داده اصلی شناس حضور در مجموعه آموزش و تست را دارد.
- نتایج  $k$  fold می‌توانند به صورت میانگین (یا به روش دیگری) ترکیب شوند تا یک تخمین واحد ایجاد شود. مزیت این روش این است که تمام مشاهدات هم برای آموزش و هم برای اعتبارسنجی استفاده می‌شوند و هر مشاهده دقیقاً یکبار برای اعتبارسنجی استفاده می‌شود.



شکل ۱۸: روش k-fold cross-validation

یک تغییر از stratified k-fold cross-validation است. این روش زمانی استفاده می‌شود که داده‌ها عدم تعادل کلاس داشته باشند، به این معنی که برخی کلاس‌ها به طور قابل توجهی نمونه‌های بیشتری نسبت به سایر کلاس‌ها دارند. در stratified k-fold cross-validation، داده‌ها به گونه‌ای به  $k$  fold تقسیم می‌شوند که هر fold همان نسبت نمونه‌ها از هر کلاس را

داشته باشد که در داده‌های اصلی وجود دارد. این اطمینان می‌دهد که مجموعه‌های آموزش و اعتبارستجو نمونه نماینده‌ای از هر کلاس داشته باشند و خطر بایاس ارزیابی مدل کاهش یابد.

- اگر از روش CV روی داده‌های نامتوارزن استفاده کنیم، ممکن است باعث بایاس شدن آموزش روی یک کلاس شویم. زیرا در K-Fold به طور تصادفی  $k$  زیرمجموعه انتخاب می‌شود و احتمال زیادی وجود دارد که fold‌هایی داشته باشیم که شامل اکثریت کلاس‌ها باشد. برای مقابله با این نوع مشکل، Stratified K-Fold با استفاده از فرآیند Stratification استفاده می‌شود.



شکل ۱۹: روش stratified k-fold cross-validation

در این مساله با توجه به اینکه تعداد داده‌های هر کلاس به طور برابر وارد مجموعه داده نهایی شده، نیازی به استفاده از k-fold cross-validation پیاده‌سازی را انجام می‌دهیم.

```

1 clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
2                      hidden_layer_sizes=(16, 12), random_state=64)
3 clf.fit(train_normalized_data, train_labels_onehot)
4
5 scoring = {'prec_macro': 'precision_macro',
6            'rec_macro': make_scorer(recall_score, average='macro')}
7 scores = cross_validate(clf, train_normalized_data, train_labels_onehot, scoring=scoring,
8                         cv=5, return_train_score=True)
9
10
11 scores['train_prec_macro'], scores['test_prec_macro'], scores['train_rec_macro'], scores['
12   test_rec_macro']
13 #plot
14 train_prec_macro = scores['train_prec_macro']
15 test_prec_macro = scores['test_prec_macro']
16 train_rec_macro = scores['train_rec_macro']
17 test_rec_macro = scores['test_rec_macro']
18
19 plt.figure(figsize=(10, 6))
20

```

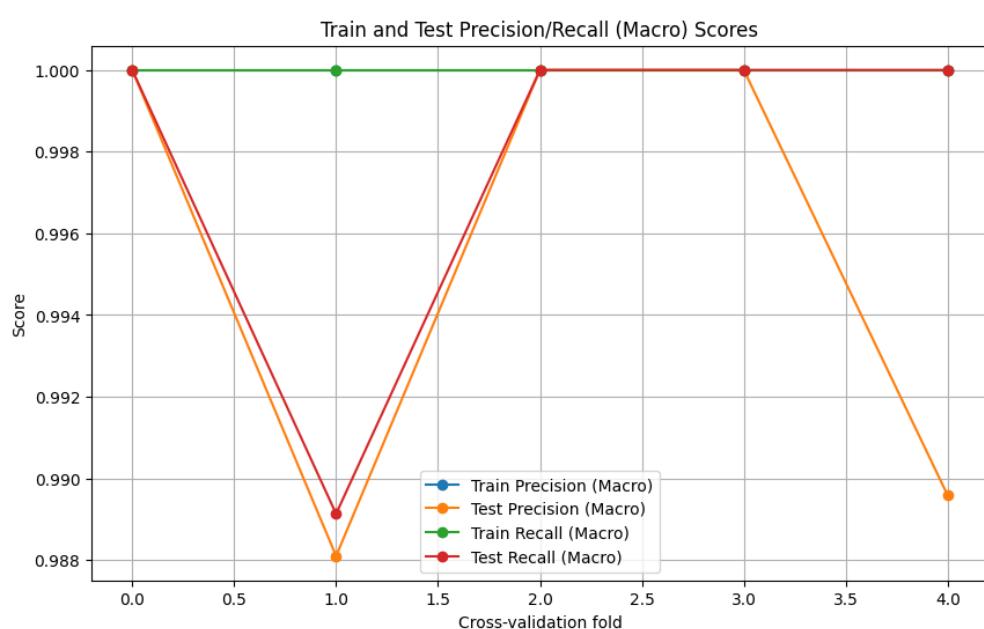


```
21 plt.plot(train_prec_macro, label='Train Precision (Macro)', marker='o')
22 plt.plot(test_prec_macro, label='Test Precision (Macro)', marker='o')
23
24 plt.plot(train_rec_macro, label='Train Recall (Macro)', marker='o')
25 plt.plot(test_rec_macro, label='Test Recall (Macro)', marker='o')
26
27 plt.xlabel('Cross-validation fold')
28 plt.ylabel('Score')
29 plt.title('Train and Test Precision/Recall (Macro) Scores')
30 plt.legend()
31 plt.grid(True)
32 plt.show()
```

حال به بررسی نتایج می پردازیم

```
(array([1., 1., 1., 1., 1.]),
 array([1.          , 0.98809524, 1.          , 1.          , 0.98958333]),
 array([1., 1., 1., 1., 1.]),
 array([1.          , 0.98913043, 1.          , 1.          , 1.        ]))
```

شکل ۲۰: نتایج cross-validation1



شکل ۲۱: نتایج cross-validation2

مشاهده می شود که با استفاده از این روش در نتایج بهبود جزیی داشتیم و همچنین فرایند overfit نبود فرایند آموزش نیز تضمین می شود.



## ۳ سوال سوم

۱.۳

در این قسمت به مقایسه روش‌های مختلف تقسیم داده‌های آموزشی و آزمون پرداخته می‌شود و بهترین روش برای استفاده در درخت تصمیم‌گیری انتخاب می‌شود. روش‌هایی که بررسی کردیم شامل تقسیم تصادفی ساده، تقسیم با استفاده از StratifiedShuffleSplit، و تبدیل ویژگی‌های Categorical به عددی و سپس تقسیم داده‌ها می‌باشد.

تقسیم تصادفی ساده

```
1 from sklearn.model_selection import train_test_split
2
3
4 X_gender = pd.get_dummies(df['Sex'])
5 X_BP = pd.get_dummies(df['BP'])
6 X_chol = pd.get_dummies(df['Cholesterol'])
7 X_other = df.drop(['Sex', 'Drug', 'BP', 'Cholesterol'], axis=1)
8 X = pd.concat([X_gender, X_other, X_BP, X_chol], axis=1).values
9 y = pd.get_dummies(df['Drug']).values
10 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=64, test_size=0.2, shuffle = True)
```

در این روش، داده‌ها به صورت تصادفی تقسیم می‌شوند. از تابع `train_test_split` برای این منظور استفاده شده است. این روش تصمیم نمی‌کند که نسبت نمونه‌ها از هر کلاس در مجموعه‌های آموزشی و تست یکسان باشد.

استفاده از StratifiedShuffleSplit

```
1 from sklearn.model_selection import StratifiedShuffleSplit
2 spl = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=64)
3 for train_index, test_index in spl.split(df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']], df['Drug']):
4     x_train, x_test = df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].iloc[train_index], df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].iloc[test_index]
5     y_train, y_test = df['Drug'].iloc[train_index], df['Drug'].iloc[test_index]
```

در این روش از `StratifiedShuffleSplit` برای تقسیم داده‌ها استفاده می‌شود. این روش تصمیم می‌کند که نسبت نمونه‌ها از هر کلاس در مجموعه‌های آموزشی و تست یکسان باشد.

تبدیل ویژگی‌های Categorical به عددی و سپس تقسیم داده‌ها در این روش، ابتدا ویژگی‌های Categorical مانند جنسیت، فشار خون و کلسترول به مقادیر عددی تبدیل می‌شوند و سپس داده‌ها به مجموعه‌های آموزشی و تست تقسیم می‌شوند.

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.model_selection import train_test_split
4
5 X_gender = pd.get_dummies(df['Sex'])
6 X_BP = pd.get_dummies(df['BP'])
7 X_chol = pd.get_dummies(df['Cholesterol'])
```



```

8 X_other = df.drop(['Sex', 'Drug', 'BP', 'Cholesterol'], axis=1)
9 X = pd.concat([X_gender, X_other, X_BP, X_chol], axis=1).values
10 y = pd.get_dummies(df['Drug']).values
11 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=64, test_size=0.2, shuffle
           = True)

```

### انتخاب بهترین روش برای درخت تصمیم‌گیری

برای درخت تصمیم‌گیری، مهم است که مجموعه‌های آموزشی و تست به طور مناسب نمایانگر کل داده‌ها باشند. این امر به خصوص در مورد داده‌های نامتوابن که تعداد نمونه‌های کلاس‌ها متفاوت است، اهمیت دارد. در چین شرایطی، روش StratifiedShuffleSplit به دلایل زیر بهترین انتخاب است:

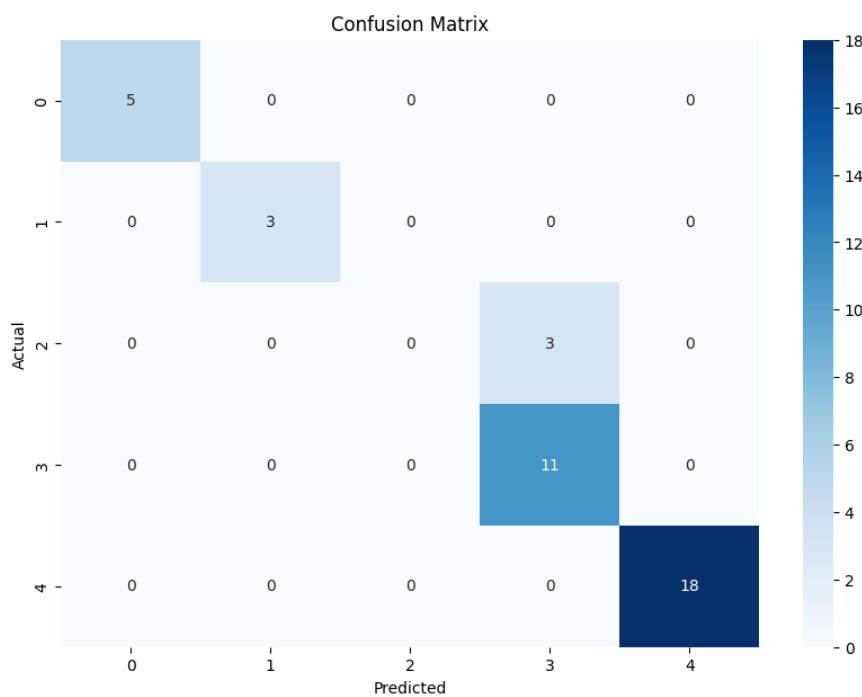
- درخت تصمیم‌گیری می‌تواند نسبت به نامتوابن کلاس‌ها حساس باشد. StratifiedShuffleSplit تصمین می‌کند که هر کلاس به طور مناسب در هر دو مجموعه آموزشی و تست نمایان باشد، که منجر به مدل دقیق‌تر و تعیین‌پذیرتر می‌شود.
- با حفظ نسبت کلاس‌ها در هر دو مجموعه، StratifiedShuffleSplit خطر بایاس مدل به سمت کلاس اکثرب را کاهش می‌دهد. این امر به خصوص در مورد داده‌های نامتوابن بسیار مهم است.
- تقسیم‌های متعادل به دستیابی به معیارهای عملکرد پایدارتر در تقسیم‌های مختلف داده کمک می‌کند.

در این قسمت یک مدل درخت تصمیمی با حداقل عمق ۳ ایجاد می‌کنیم و نتایج آن را بررسی می‌کنیم.

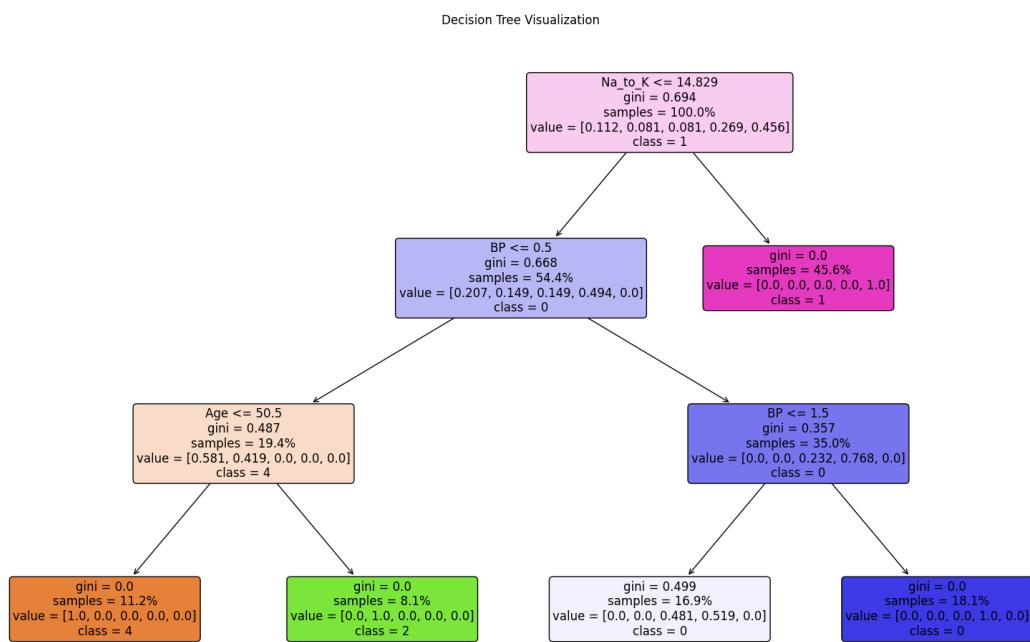


Classification Report:					
	precision	recall	f1-score	support	
0	1.00	1.00	1.00	5	
1	1.00	1.00	1.00	3	
2	0.00	0.00	0.00	3	
3	0.79	1.00	0.88	11	
4	1.00	1.00	1.00	18	
accuracy			0.93	40	
macro avg	0.76	0.80	0.78	40	
weighted avg	0.87	0.93	0.89	40	

شکل ۲۲: دقت مدل درخت تصمیمی



شکل ۲۳: ماتریس درهمریختگی مدل درخت تصمیمی



شکل ۲۴: درخت تصمیمی ایجاد شده

با توجه به نتایج مشاهده می شود که مدل دقت کلی ۹۲.۵ را داشته ولی در کلاس ۲ هیچ نمونه‌ای را درست پیشینی نکرده. این مورد از ماتریس درهمریختگی نیز قابل مشاهده است. حال به بررسی درخت ایجاد شده می پردازیم.



## تحلیل گره‌ها

### گره ریشه

درخت تصمیم با تقسیم بر اساس ویژگی نسبت سدیم به پتاسیم ( $\text{Na\_to\_K}$ ) آغاز می‌شود. آستانه این تقسیم برابر با 14.829 است. ناخالصی جینی این گره برابر با 0.696 بوده و شامل 100% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.131, 0.088, 0.075, 0.075] است و کلاس غالب 0.244، 0.462 می‌باشد.

### زیر درخت چپ

زیر درخت چپ گره‌ای زیر درخت چپ بر اساس ویژگی فشار خون (BP) تقسیم می‌شوند. گره 1 با آستانه 0.5 و ناخالصی جینی 0.689 شامل 53.8% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.244, 0.163, 0.14, 0.453, 0.0] است و کلاس غالب 0 می‌باشد. گره 1a با ویژگی سن (Age) تقسیم می‌شود. آستانه این تقسیم برابر با 50.5 است. ناخالصی جینی این گره برابر با 0.48 بوده و شامل 21.9% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.6, 0.4, 0.0, 0.0, 0.0] است و کلاس غالب 4 می‌باشد.

گره برگ 1a1 دارای ناخالصی جینی 0.0 بوده و شامل 13.1% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [1.0, 0.0, 0.0, 0.0] است و کلاس 4 می‌باشد.

گره برگ 1a2 دارای ناخالصی جینی 0.0 بوده و شامل 8.8% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 1.0, 0.0, 0.0] است و کلاس 2 می‌باشد.

گره 1b با ویژگی فشار خون (BP) تقسیم می‌شود. آستانه این تقسیم برابر با 1.5 است. ناخالصی جینی این گره برابر با 0.36 بوده و شامل 31.9% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.235, 0.765, 0.0] است و کلاس غالب 0 می‌باشد.

گره برگ 1b1 دارای ناخالصی جینی 0.497 بوده و شامل 16.2% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.462, 0.538, 0.0] است و کلاس 0 می‌باشد.

گره برگ 1b2 دارای ناخالصی جینی 0.0 بوده و شامل 15.6% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.0, 1.0, 0.0] است و کلاس 0 می‌باشد.

### زیر درخت راست

گره 2 مستقیماً به یک گره برگ تقسیم می‌شود. ناخالصی جینی این گره برابر با 0.0 بوده و شامل 46.2% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.0, 1.0] است و کلاس 1 می‌باشد.

ویژگی  $\text{Na\_to\_K}$  مهم‌ترین ویژگی برای تقسیم اولیه است و پس از آن ویژگی‌های BP و Age قرار دارند. توزیع کلاس‌ها در هر گره نشان‌دهنده نسبت نمونه‌های هر کلاس است که به درک ناخالصی گره کمک می‌کند. مقادیر پایین‌تر ناخالصی جینی نشان‌دهنده گره‌های خالص‌تر با توزیع همگن‌تر کلاس‌ها هستند. کلاسی که در هر گره بیشترین نسبت را دارد به عنوان کلاس تصمیم‌گیری انتخاب می‌شود. گره‌های برگ نقاطه تصمیم‌گیری نهایی در درخت هستند. هر گره برگ به یک کلاس خاص اختصاص داده شده و نشان‌دهنده خلوص نمونه‌ها در آن گره است. برای مثال، گره برگ 1a1 به صورت خالص متعلق به کلاس 4 است که نشان‌دهنده مرز تصمیم‌گیری دقیق است. درخت تصمیم با ایجاد چندین تقسیم، داده‌ها را به نواحی ای که عمده‌ای به یک کلاس خاص تعلق دارند، تقسیم کرده است. تقسیم اولیه بر اساس  $\text{Na\_to\_K}$  بسیار مهم است که نشان می‌دهد این ویژگی بیشترین تفکیک کلاس‌ها را فراهم می‌کند. تقسیمات بعدی بر اساس BP و Age مرزهای تصمیم‌گیری را بیشتر دقیق‌تر می‌کنند. مقادیر ناخالصی جینی نشان می‌دهند که درخت به طور مؤثر گره‌های خالص ایجاد کرده است، به ویژه در گره برگ 1a1 و گره 2 که ناخالصی جینی صفر دارند. تقسیمات دقیق، به ویژه با مقادیر ناخالصی پایین در

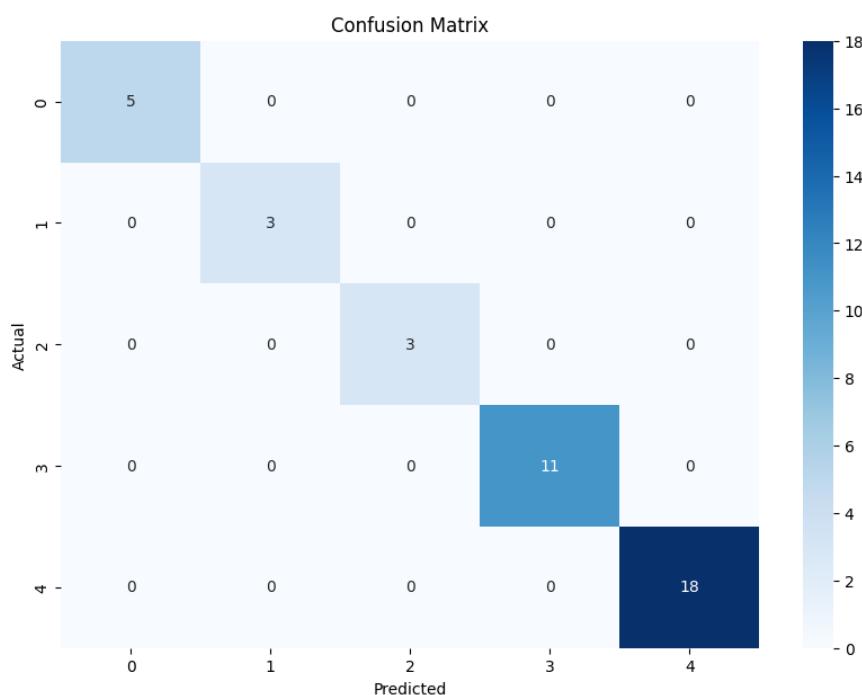
گره‌های برگ، نشان‌دهنده این است که مدل ممکن است به داده‌های آموزشی بیش‌برازش شود که این می‌تواند نشانه‌ای از این باشد که مدل در داده‌های آموزشی خوب عمل می‌کند اما در داده‌های جدید عملکرد کمتری دارد.

درخت تصمیم ارائه شده مدل روشنی از مجموعه داده‌ها فراهم می‌کند و ویژگی‌های کلیدی مانند BP، Na\_to\_K و Age را برجسته می‌کند. با این حال، باید عملکرد آن بر روی مجموعه داده‌های اعتبارسنجی ارزیابی شود تا از تعیین پذیری خوب مدل و عدم بیش‌برازش آن اطمینان حاصل شود.

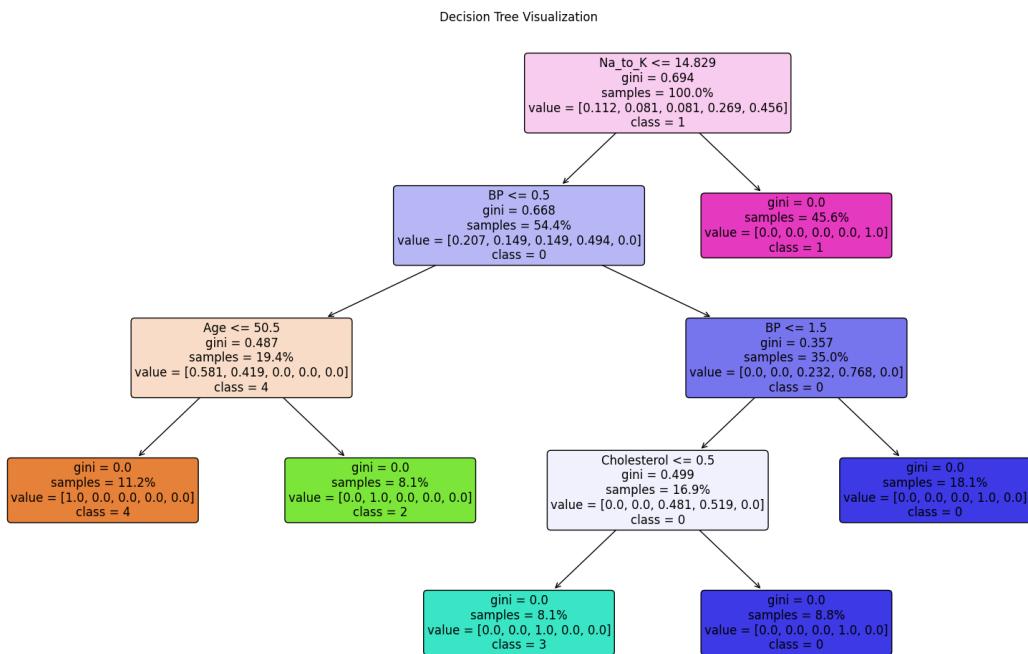
حال مدل را با عمق ۴ ایجاد می‌کنیم. نتایج به صورت زیر خواهد بود.

Classification Report:					
	precision	recall	f1-score	support	
0	1.00	1.00	1.00	5	
1	1.00	1.00	1.00	3	
2	1.00	1.00	1.00	3	
3	1.00	1.00	1.00	11	
4	1.00	1.00	1.00	18	
accuracy			1.00	40	
macro avg	1.00	1.00	1.00	40	
weighted avg	1.00	1.00	1.00	40	

شکل ۲۵: دقیقیت مدل درخت تصمیمی



شکل ۲۶: ماتریس درهم‌ریختگی مدل درخت تصمیمی



شکل ۲۷: درخت تصمیمی ایجاد شده

مشاهده می‌شود که مدل در این حالت دقیق ۱۰۰ درصد را کسب کرده و تمامی نمونه‌ها در کلاس‌های مختلف را به درستی پیش‌بینی کرده است. تحلیل این درخت به صورت زیر است.

درخت تصمیم با تقسیم بر اساس ویژگی نسبت سدیم به پتاسیم (Na\_to\_K) آغاز می‌شود. آستانه این تقسیم برابر با 14.829 است. ناخالصی جینی این گره برابر با 0.694 بوده و شامل ۱۰۰٪ نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.112, 0.081, 0.081, 0.269, 0.456] است و کلاس غالب ۱ می‌باشد.

### زیر درخت چپ

گره‌های زیر درخت چپ بر اساس ویژگی فشار خون (BP) تقسیم می‌شوند. گره ۱ با آستانه ۰.۵ و ناخالصی جینی ۰.۶۶۸ شامل ۵۴.۴٪ نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.207, 0.149, 0.149, 0.494, 0.0] است و کلاس غالب ۰ می‌باشد. گره ۱a با ویژگی سن (Age) تقسیم می‌شود. آستانه این تقسیم برابر با ۵۰.۵ است. ناخالصی جینی این گره برابر با ۰.۴۸۷ بوده و شامل ۱۹.۴٪ نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.581, 0.419, 0.0, 0.0, 0.0] است و کلاس غالب ۴ می‌باشد.

گره برج ۱a1 دارای ناخالصی جینی ۰.۰ بوده و شامل ۱۱.۲٪ نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [1.0, 0.0, 0.0, 0.0, 0.0] است و کلاس ۴ می‌باشد.

گره برج ۱a2 دارای ناخالصی جینی ۰.۰ بوده و شامل ۸.۱٪ نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 1.0, 0.0, 0.0, 0.0] است و کلاس ۲ می‌باشد.

گره ۱b با ویژگی فشار خون (BP) تقسیم می‌شود. آستانه این تقسیم برابر با ۱.۵ است. ناخالصی جینی این گره برابر با ۰.۳۵۷ بوده و شامل ۳۵.۰٪ نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.232, 0.768, 0.0] است و کلاس غالب ۰ می‌باشد. گره برج ۱b1 دارای ناخالصی جینی ۰.۴۹۹ بوده و شامل ۱۶.۹٪ نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.481, 0.519, 0.0] است و کلاس ۰ می‌باشد.



گره برگ 1b2 دارای ناخالصی جینی 0.0 بوده و شامل 18.1% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.0] است و کلاس 0 می‌باشد.

### زیر درخت راست

گره‌های زیر درخت راست بر اساس ویژگی فشار خون (BP) تقسیم می‌شوند. گره 2 با آستانه 1.5 و ناخالصی جینی 0.357 شامل 35.0% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.232, 0.768, 0.0] است و کلاس غالب 0 می‌باشد.

گره 2a با ویژگی کلسترول (Cholesterol) تقسیم می‌شود. آستانه این تقسیم برابر با 0.5 است. ناخالصی جینی این گره برابر با 0.499 بوده و شامل 16.9% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.481, 0.519, 0.0] است و کلاس غالب 0 می‌باشد.

گره برگ 2a1 دارای ناخالصی جینی 0.0 بوده و شامل 8.1% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 1.0, 0.0] است و کلاس 3 می‌باشد.

گره برگ 2a2 دارای ناخالصی جینی 0.0 بوده و شامل 8.8% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.0, 1.0, 0.0] است و کلاس 0 می‌باشد.

گره برگ 2b دارای ناخالصی جینی 0.0 بوده و شامل 45.6% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.0, 0.0, 1.0] است و کلاس 1 می‌باشد.

گره برگ 2b1 دارای ناخالصی جینی 0.0 بوده و شامل 18.1% نمونه‌ها می‌باشد. توزیع کلاس‌ها در این گره به صورت [0.0, 0.0, 0.0, 0.0, 1.0] است و کلاس 1 می‌باشد.

## ۲.۳

معیارهای مختلف برای مدل اول در بخش قبیل مورد بررسی قرار گرفت و مشاهده کردیم که با عدم تنظیم فرآپارامترها، مدل به دقت ۱۰۰ درصد می‌رسد و می‌تواند تمامی کلاس‌ها را با خلوص کامل از هم جدا کند. در این بخش، تعدادی فرآپارامتر به مدل اضافه می‌کنیم و نتایج را بررسی می‌کنیم.

مدل نشان داده شده شامل فرآپارامترهای min samples split max features max leaf nodes است.

split samples min : حداقل تعداد سمپل‌هایی را که نیاز است تا یک نود تقسیم شود را تعیین می‌کند. برای مثال اگر این فرآپارامتر برابر با ۳۰ تنظیم شود و تعداد سمپل‌های یک نود کمتر از این مقدار باشد، نود مذکور دیگر تقسیم نخواهد شد.

features max : تعداد ویژگی‌هایی که درخت برای طبقه‌بندی از آن استفاده می‌کند و در اینجا برابر با sqrt تنظیم شده، یعنی اگر ۴ ویژگی داشته باشیم، درخت تنها از ۲ تای آن‌ها برای طبقه‌بندی استفاده می‌کند.

nodes leaf max : تعداد کل نودهای برگ (leaf node) که درخت می‌تواند داشته باشد را محدود می‌کند. از دیگر فرآپارامترهای خوب درخت تصمیم می‌توان به ccp alpha و depth max اشاره کرد که برای هرس کردن درخت به کار می‌روند.

ابتدا با انتخاب هایپرپارامترهای زیر مدل را مورد ارزیابی قرار می‌دهیم:



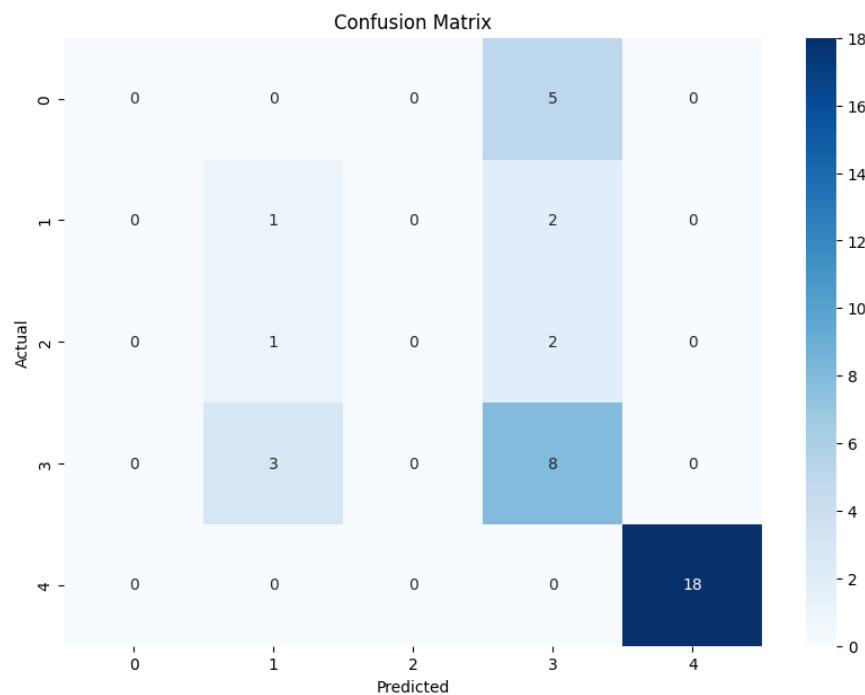
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, max_features='sqrt', max_leaf_nodes=5,
min_samples_split=20, random_state=64)
```

شکل ۲۸: مدل اول

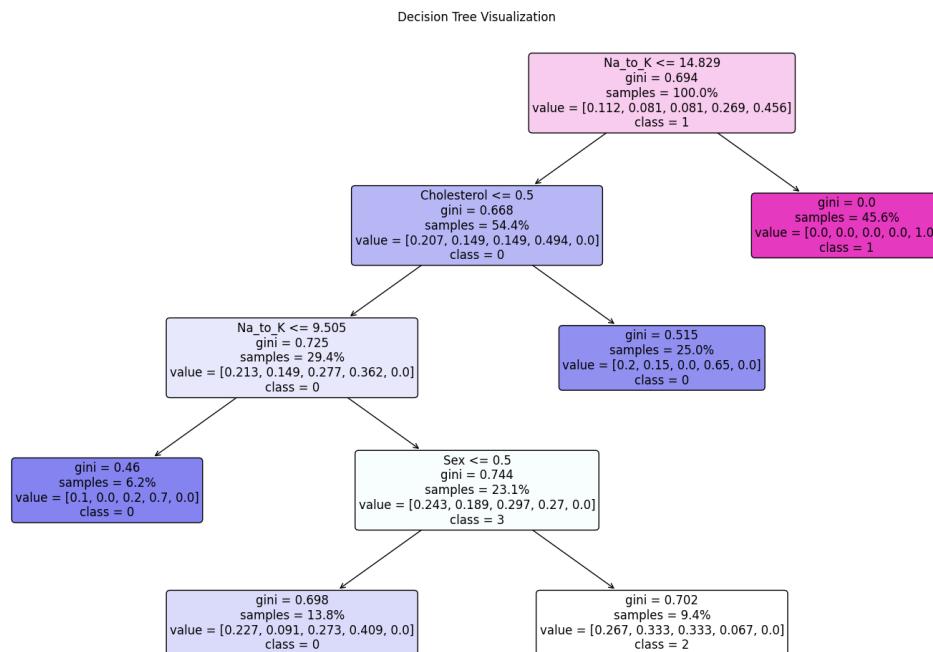
```
Accuracy: 0.675
Classification Report:
precision    recall    f1-score   support
          0       0.00     0.00      0.00       5
          1       0.20     0.33     0.25       3
          2       0.00     0.00      0.00       3
          3       0.47     0.73     0.57      11
          4       1.00     1.00      1.00      18

   accuracy                           0.68      40
  macro avg       0.33     0.41     0.36      40
weighted avg       0.59     0.68     0.63      40
```

شکل ۲۹: دقت مدل درخت تصمیمی ۱



شکل ۳۰: ماتریس درهم ریختگی مدل درخت تصمیمی ۱



شکل ۳۱: درخت تصمیمی ایجاد شده

نتایج طبقه‌بندی درخت تصمیم مشاهده می‌شود. می‌بینیم که دقت به ۶۸ درصد کاهاش پیدا کرده است.

دلیل این امر واضح است؛ مدل ما هیچ داده‌ای را در کلاس‌های ۰ و ۲ پیش‌بینی نکرده که می‌تواند نتیجه محدود کردن زیاد مدل و عدم توانایی مدل در تطبیق خود با پیچیدگی داده‌ها باشد. در واقع، مدل به اندازه کافی پیچیده نیست تا داده‌های ۱ و ۲ را از بقیه داده‌ها جدا کند. بهترین نتیجه را در کلاس چهار که همان Y drug است دارد و هم recall بالایی هم دارد و در نتیجه f1-score خوبی نیز دارد.

در ماتریس درهم‌ریختگی هرچه مقادیر روی قطر تجمع بیشتری داشته باشند، مدل عملکرد بهتری داشته است. در این ماتریس می‌توان دید که مدل ما هیچ داده‌ای را از کلاس ۰ و ۲ در دسته‌بندی جدا قرار نداده است. ۳ داده مربوط به کلاس ۳ را در کلاس یک طبقه‌بندی کرده و ۲ داده مربوط به کلاس ۱ را در کلاس ۳ طبقه‌بندی کرده است. با تنظیم هایپرپارامترها، ما اجازه دادیم که خلوص هر نوک متر بماند و این داده‌ها که اشتباه طبقه‌بندی شده‌اند نتیجه این استراتژی ما بود.

درخت جدید نشان می‌دهد که درخت ما نسبت به قبل کوچکتر شده است و خلوص هر leaf node کمتر شده است. علاوه بر این، می‌توان دید که تمام طبقه‌بندی روی سه ویژگی NA to K, BP و Cholesterol انجام شده است. با توجه به نمودار درخت دیده می‌شود که هیچ کلاسی از drugA و drugC وجود ندارد و این همان موضوع است که درخت هیچ داده‌ای را در کلاس ۰ و ۲ طبقه‌بندی نمی‌کند و پیچیدگی لازم را ندارد.

با کم و زیاد کردن پارامترها مدل جدید به صورت زیر خواهد بود.

```

*          DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, max_features='sqrt', max_leaf_nodes=10,
min_samples_split=5, random_state=64)

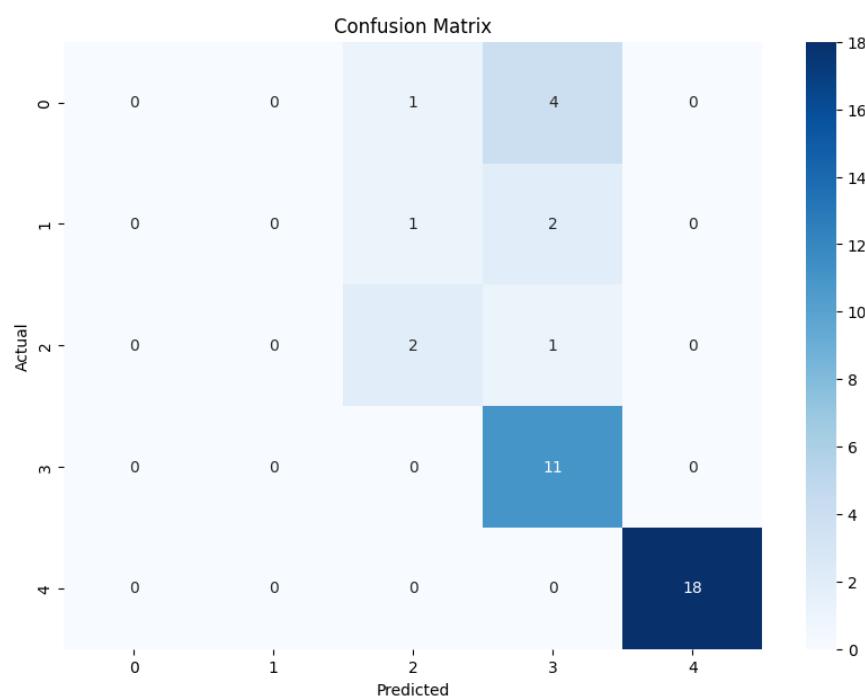
```

شکل ۳۲: مدل دوم

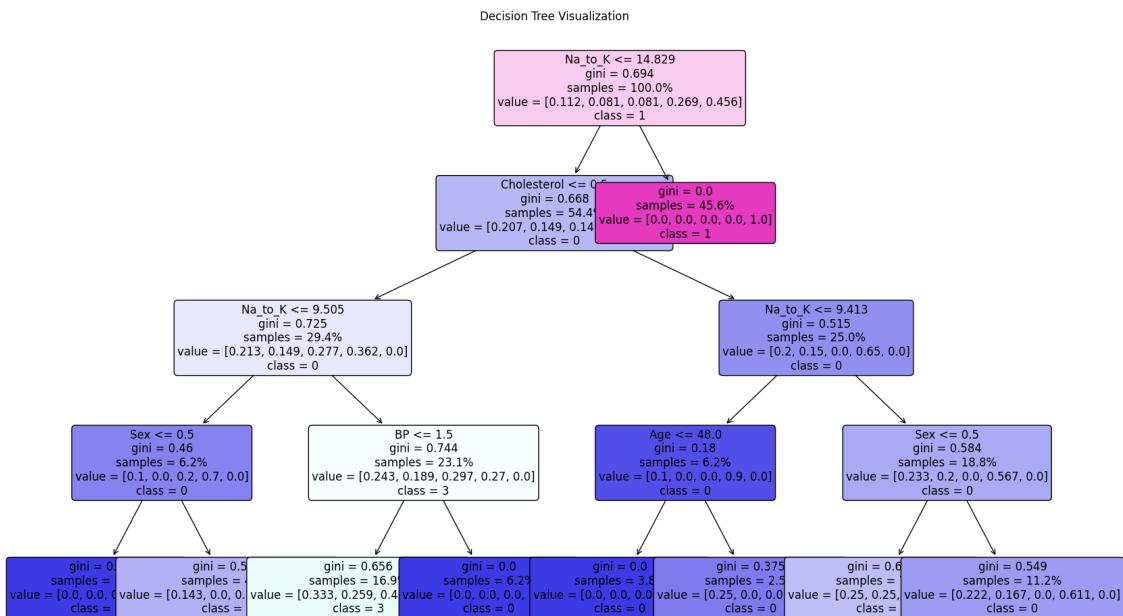


Classification Report:				
	precision	recall	f1-score	support
0	0.00	0.00	0.00	5
1	0.00	0.00	0.00	3
2	0.50	0.67	0.57	3
3	0.61	1.00	0.76	11
4	1.00	1.00	1.00	18
accuracy			0.78	40
macro avg		0.42	0.53	0.47
weighted avg		0.66	0.78	0.70

شكل ۳۳: دقت مدل درخت تصمیمی ۲



شكل ۳۴: ماتریس درهم ریختگی مدل درخت تصمیمی ۲



شکل ۳۵: درخت تصمیمی ایجاد شده

همان‌طور که دیده می‌شود، این درخت کمتر هرس شده و شروط آزادانه‌تر هستند تا مدل پیچیدگی بیشتری نسبت به مدل قبل به خود بگیرد. همان‌طور که دیده می‌شود، دقت بهتر شده و به ۷۸ درصد رسیده است.

مشاهده می‌شود درخت بزرگ‌تر شده و پیچیدگی بیشتری به خود گرفته است.

در حالت بعد پارامتر ccp\_alpha را در مدل قرار می‌دهیم که میزان هرس کردن را مشخص می‌کند. هرچه ccp\_alpha بزرگ‌تر باشد، هرس بیشتری را شاهد خواهیم بود.

```

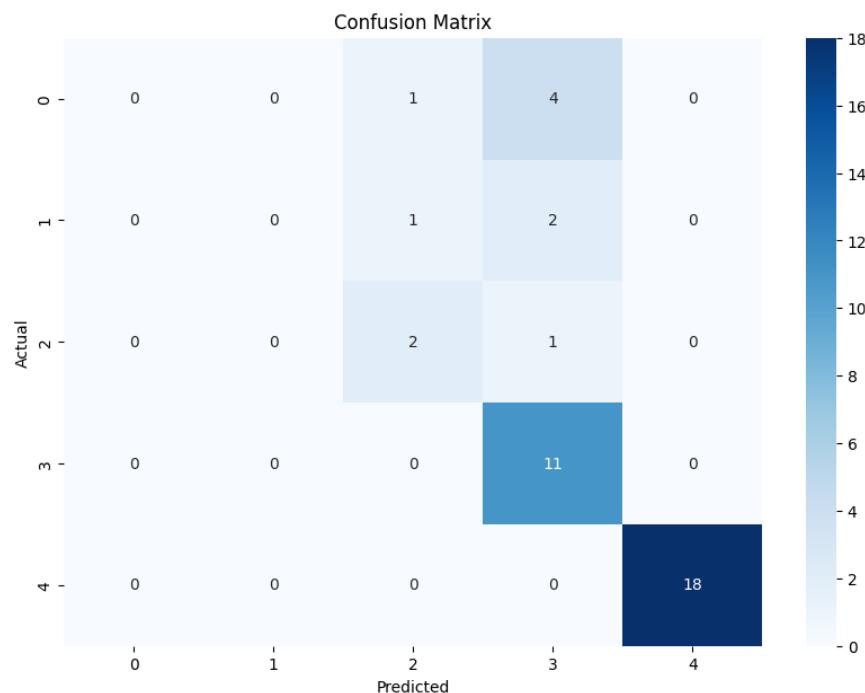
*          DecisionTreeClassifier
DecisionTreeClassifier(ccp_alpha=0.01, max_depth=4, max_features='sqrt',
max_leaf_nodes=10, min_samples_split=5, random_state=64)

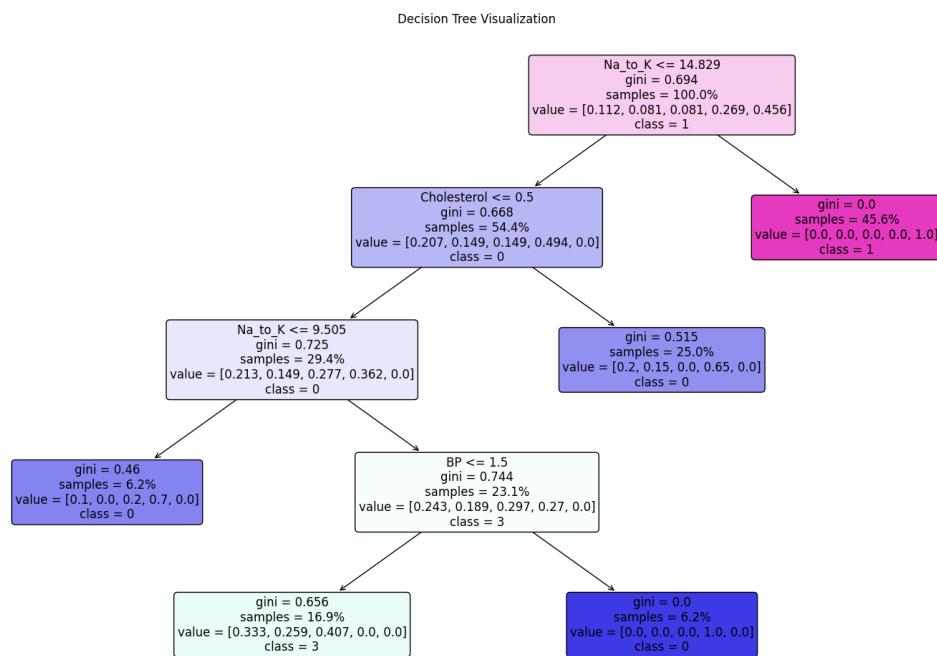
```

شکل ۳۶: مدل سوم



Classification Report:					
	precision	recall	f1-score	support	
0	0.00	0.00	0.00	5	
1	0.00	0.00	0.00	3	
2	0.50	0.67	0.57	3	
3	0.61	1.00	0.76	11	
4	1.00	1.00	1.00	18	
accuracy				0.78	40
macro avg				0.42	40
weighted avg				0.66	40

شکل ۳۷: دقیقیت مدل درخت تصمیمی<sup>۳</sup>شکل ۳۸: ماتریس درهMRIختگی مدل درخت تصمیمی<sup>۳</sup>



شکل ۳۹: درخت تصمیمی ایجاد شده

همان‌طور که دیده می‌شود، با حفظ دقیق‌تر کمی از درخت هرس شده و شکل آن جمع‌وجویرتر شده است که به تنظیم فرآپارامترها برمی‌گردد.

### ۳.۳

به طور کلی، تکنیک‌های AdaBoost و RandomForest در دسته‌بندی روش‌های Ensemble learning قرار می‌گیرند. این تکنیک‌ها نتایج حاصل از چندین مدل ساده‌تر را ترکیب کرده و به ما ارائه می‌دهند. این فرآیند شبیه به این است که بر اساس نظر جمعی تصمیم‌گیری کنیم. به دلیل استفاده از چندین مدل ضعیفتر، این تکنیک‌ها می‌توانند ما را از overfitting دور نگه داشته و مشکل bias-variance را تا حدی کاهش دهند. تفاوت اصلی بین این دو روش این است که RandomForest بر اساس روش‌های bagging عمل می‌کند در حالی که AdaBoost بر اساس روش‌های boosting کار می‌کند.

در RandomForest، داده‌های آموزشی به تعدادی subset تقسیم شده و هر کدام به صورت تصادفی برای تشکیل یک درخت استفاده می‌شود. سپس نتیجه نهایی طبقه‌بندی همه درخت‌ها به ما ارائه می‌شود. این روش می‌تواند به دلیل نیازمندی به توان محاسباتی بالا، هزینه‌بر باشد. اما در روش AdaBoost، مدل ابتدا یک طبقه‌بندی انجام می‌دهد و سپس با توجه به خطا و وزن دهی بیشتر به داده‌هایی که اشتباه طبقه‌بندی شده‌اند، نتیجه را در چند مرحله بهبود می‌بخشد.

```

1 from sklearn.ensemble import RandomForestClassifier
2
3 # Initialize the RandomForestClassifier with specific hyperparameters
4 model_rf = RandomForestClassifier(
5     n_estimators=3,          # Number of trees in the forest
6     min_samples_split=4,      # Minimum number of samples required to split an internal node
  
```



```
7     max_depth=10,           # Maximum depth of the tree
8     random_state=64         # Random seed for reproducibility
9 )
```

Code 7: RandomForestClassifier Configuration

	precision	recall	f1-score	support
0	0.83	1.00	0.91	5
1	1.00	0.67	0.80	3
2	0.60	1.00	0.75	3
3	1.00	0.82	0.90	11
4	1.00	1.00	1.00	18
accuracy			0.93	40
macro avg	0.89	0.90	0.87	40
weighted avg	0.95	0.93	0.93	40

شکل ۴۰: نتایج Random forest

در ابتدا، یک مدل جنگل تصادفی می‌سازیم. این مدل تعداد ۳ درخت تشکیل می‌دهد و حداقل نمونه‌هایی که باید وجود داشته باشد تا node تقسیم شود برابر با ۴ است. حداقل عمق درخت‌ها ۱۰ است و مدل. دیده می‌شود که دقت مدل ۹۳ درصد است و مدل در تمامی کلاس‌ها عملکرد خوبی دارد. در کلاس ۲ از کل نمونه‌ها، مدل توانسته ۷۵ درصدشان را به درستی طبقه‌بندی کند. در بخش بعدی، از AdaBoost استفاده می‌کنیم. با استفاده از کد زیر با استفاده از پارامترهای مختلف نتایج را بررسی می‌کنیم.

```
1 from sklearn.ensemble import AdaBoostClassifier
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.model_selection import cross_val_score
4 from sklearn.metrics import f1_score, make_scorer
5
6 # Hyperparameters
7 n_estimators = [3, 5, 10, 30, 100]
8 learning_rates = [0.1, 0.05, 0.01]
9
10 f1_scoring = make_scorer(f1_score, average='weighted')
11
12 weak_learner = RandomForestClassifier(random_state=64)
13
14 results = []
15
16 for ne in n_estimators:
17     for lr in learning_rates:
18         model = AdaBoostClassifier(estimator=weak_learner, n_estimators=ne, learning_rate=lr)
19
20         # Cross-validation
```



```
21     scores = cross_val_score(model, x_train, y_train, cv=3, scoring=f1_scorer)
22     mean_f1_score = np.mean(scores)
23
24     # Append results
25     results.append({
26         'n_estimators': ne,
27         'learning_rate': lr,
28         'mean_f1_score': mean_f1_score,
29     })
30
31 results_df = pd.DataFrame(results)
32
33 print(results_df)
```

Code 8: Adaboost Classifier Configuration

n_estimators	learning_rate	mean_f1_score
0	0.10	0.973911
1	0.05	0.980137
2	0.01	0.980137
3	0.10	0.967558
4	0.05	0.980137
5	0.01	0.987282
6	0.10	0.981056
7	0.05	0.987282
8	0.01	0.973911
9	0.10	0.973515
10	0.05	0.973911
11	0.01	0.973911
12	0.10	0.973911
13	0.05	0.980137
14	0.01	0.973911

شکل ۴۱: نتایج Adaboost

مشاهده می شود که نتایج بهبود پیدا کرده.

## ۴ سوال چهارم

۱.۴

- با توجه به توضیحات منبع داده، داده های استفاده شده برای سال ۱۹۸۸ هستند که شامل ۷۶ ویژگی مختلف می باشد. در این داده ها، مقدار نشان دهنده عدم بیماری قلبی و مقدار ۱ نشان دهنده وجود بیماری قلبی است.



ویژگی‌های این سری داده به صورت زیر هستند.

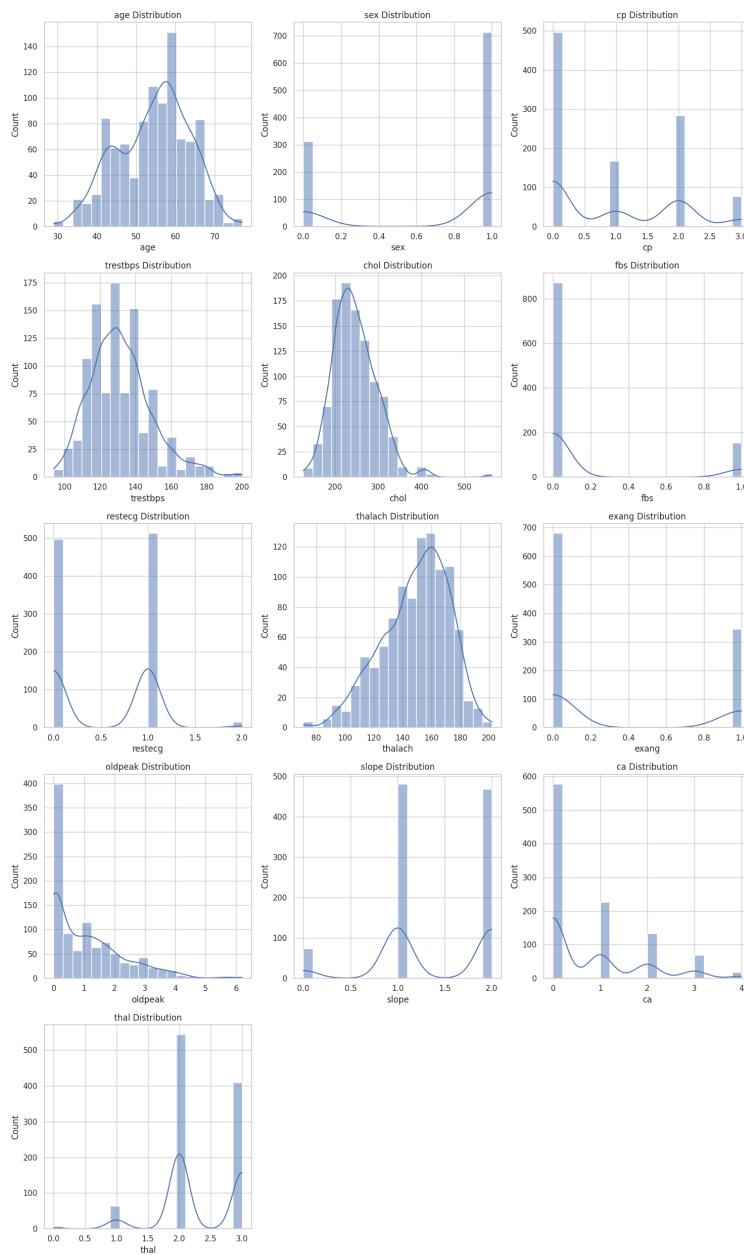
- سن: سن فرد بیمار به سال.
- جنسیت: جنسیت فرد بیمار (۱ برای مرد و ۰ برای زن).
- نوع درد قفسه سینه (۴ مقدار):
  - مقدار ۱: درد قفسه سینه تپیکال آنژین.
  - مقدار ۲: درد قفسه سینه آتیپیکال آنژین.
  - مقدار ۳: درد قفسه سینه غیر مرتبط با قلب.
  - مقدار ۴: بدون درد قفسه سینه.
- فشار خون در حالت استراحت: فشار خون فرد در حالت استراحت (میلی متر جیوه).
- کلسترول سرم به میلی گرم بر دسی لیتر: میزان کلسترول در خون (میلی گرم بر دسی لیتر).
- قند خون ناشتا بیشتر از ۱۲۰ میلی گرم بر دسی لیتر: آیا قند خون ناشتا فرد بالای ۱۲۰ میلی گرم بر دسی لیتر است (۱ برای بله و ۰ برای خیر).
- نتایج الکتروکاردیوگرافی استراحتی (مقادیر ۰، ۱، ۲):
  - مقدار ۰: عادی.
  - مقدار ۱: وجود موج ST-T غیرعادی (شامل وارونگی موج T و ارتفاعات یا افسردگی‌های ST).
  - مقدار ۲: نمایش چپ بطن قلب (LVH) بر اساس معیارهای استراسی.
- حداکثر ضربان قلب دستیافته: بالاترین میزان ضربان قلب در حین ورزش.
- آنژین ناشی از ورزش: آیا فرد در حین ورزش دچار آنژین می‌شود (۱ برای بله و ۰ برای خیر).
- افت ST با ورزش نسبت به حالت استراحت: (oldpeak) تغییرات در سطح بخش ST در ECG پس از ورزش نسبت به استراحت.
- شیب بخش ST در حداکثر تمرینات:
  - مقدار ۱: شیب بالا.
  - مقدار ۲: شیب صاف.
  - مقدار ۳: شیب پایین.
- تعداد رگ‌های عمدۀ (۰-۳) رنگ‌آمیزی شده با فلوروسکوپی: تعداد رگ‌های اصلی (۰-۳) که به وسیله فلوروسکوپی رنگ‌آمیزی شده‌اند.

در این قسمت از GaussianNB برای طبقه‌بندی بهره می‌گیریم. GaussianNB یکی از مدل‌های معروف در دسته‌بندی بیز ساده است که در پکیج scikit-learn پیاده‌سازی شده است. در اینجا به بررسی پارامترهای این مدل می‌پردازیم:

• priors: این پارامتر می‌تواند آرایه‌ای از مقادیر احتمال قبلی برای هر کلاس باشد. اگر تنظیم نشده باشد، مقادیر قبلی براساس فرکانس‌های کلاس در داده‌های آموزشی تخمین زده می‌شوند.

• var\_smoothing: این پارامتر به منظور پایداری محاسباتی به واریانس داده‌ها اضافه می‌شود. مقدار پیش‌فرض آن  $10^{-9}$  است و این مقدار کوچک به جلوگیری از تقسیم بر صفر کمک می‌کند.

نمودارهای هیستوگرام برای ویژگی‌های کلیدی:



شکل ۴۲: هیستوگرام ویژگی‌ها

هیستوگرام‌های زیر توزیع ویژگی‌های مختلف در مجموعه داده بیماری قلبی را نشان می‌دهند.



## توزیع سن

توزیع سن به صورت تقریبی دونموداری است و دارای دو پیک اصلی در حدود ۵۰-۴۵ سال و پیک ثانویه در حدود ۶۰-۵۵ سال است. بیشتر بیماران در بازه سنی ۴۰ تا ۷۰ سال قرار دارند و توزیع به سمت سنین بالاتر متمایل است.

## توزیع جنسیت

این ویژگی دوتایی نشان می‌دهد که تعداد زیادی از بیماران مرد (نشان داده شده با ۱) هستند.

## توزیع نوع درد قفسه سینه

ویژگی نوع درد قفسه سینه دارای چهار دسته (۰ تا ۳) است. شایع‌ترین نوع، نوع ۰ است که پس از آن نوع ۱ قرار دارد، در حالی که نوع‌های ۲ و ۳ کمتر شایع هستند.

## توزیع فشار خون در حالت استراحت

مقادیر فشار خون در حالت استراحت عمدتاً بین ۱۱۰ تا ۱۴۰ میلی‌متر جیوه قرار دارد و یک پیک در حدود ۱۲۰ میلی‌متر جیوه دارد. چند مورد با مقادیر بالاتر نیز وجود دارند.

## توزیع کلسترول

سطح کلسترول سرم بیشتر بیماران بین ۲۰۰ تا ۳۰۰ میلی‌گرم بر دسی‌لیتر قرار دارد و پیک در حدود ۲۵۰ میلی‌گرم بر دسی‌لیتر مشاهده می‌شود. مقادیر بالاتر نیز کمتر دیده می‌شوند.

## توزیع قند خون ناشتا

این ویژگی دوتایی نشان می‌دهد که بیشتر بیماران سطح قند خون ناشتا کمتر از ۱۲۰ میلی‌گرم بر دسی‌لیتر (نشان داده شده با ۰) دارند و تعداد کمتری سطح بالاتر (نشان داده شده با ۱) دارند.

## نتایج الکتروکاردیوگرافی استراحتی

نتایج الکتروکاردیوگرافی در حالت استراحت عمدتاً نرمال (۰) هستند و تعداد کمتری از موارد دارای موج‌های ST-T غیرعادی (۱) و هیپرتروفی بطن چپ (۲) هستند.

## توزیع حداکثر ضربان قلب دستیافته

حداکثر ضربان قلب دستیافته بیماران عمدتاً بین ۱۳۰ تا ۱۷۰ ضربه در دقیقه است و یک پیک در حدود ۱۶۰ ضربه در دقیقه دارد.

## آنژین ناشی از ورزش

بیشتر بیماران آنژین ناشی از ورزش ندارند (۰)، در حالی که تعداد کمتری آنژین دارند (۱).

## توزیع افت ST



مقادیر افت ST عمدتاً پایین هستند و بیشتر مقادیر کمتر از ۲ است. مقادیر بالاتر نیز کمتر دیده می‌شوند.

### شیب بخش ST

ویژگی شیب دارای سه دسته است و شیب تخت (۱) و شیب بالارونده (۲) شایع‌تر هستند، در حالی که شیب پایین‌رونده (۰) کمتر شایع است.

### تعداد رگ‌های عمدۀ رنگ‌آمیزی شده با فلوروسکوپی

تعداد رگ‌های عمدۀ رنگ‌آمیزی شده با فلوروسکوپی از ۰ تا ۴ متغیر است. بیشتر بیماران هیچ رگی (۰) ندارند و با افزایش تعداد رگ‌ها تعداد بیماران کاهش می‌یابد.

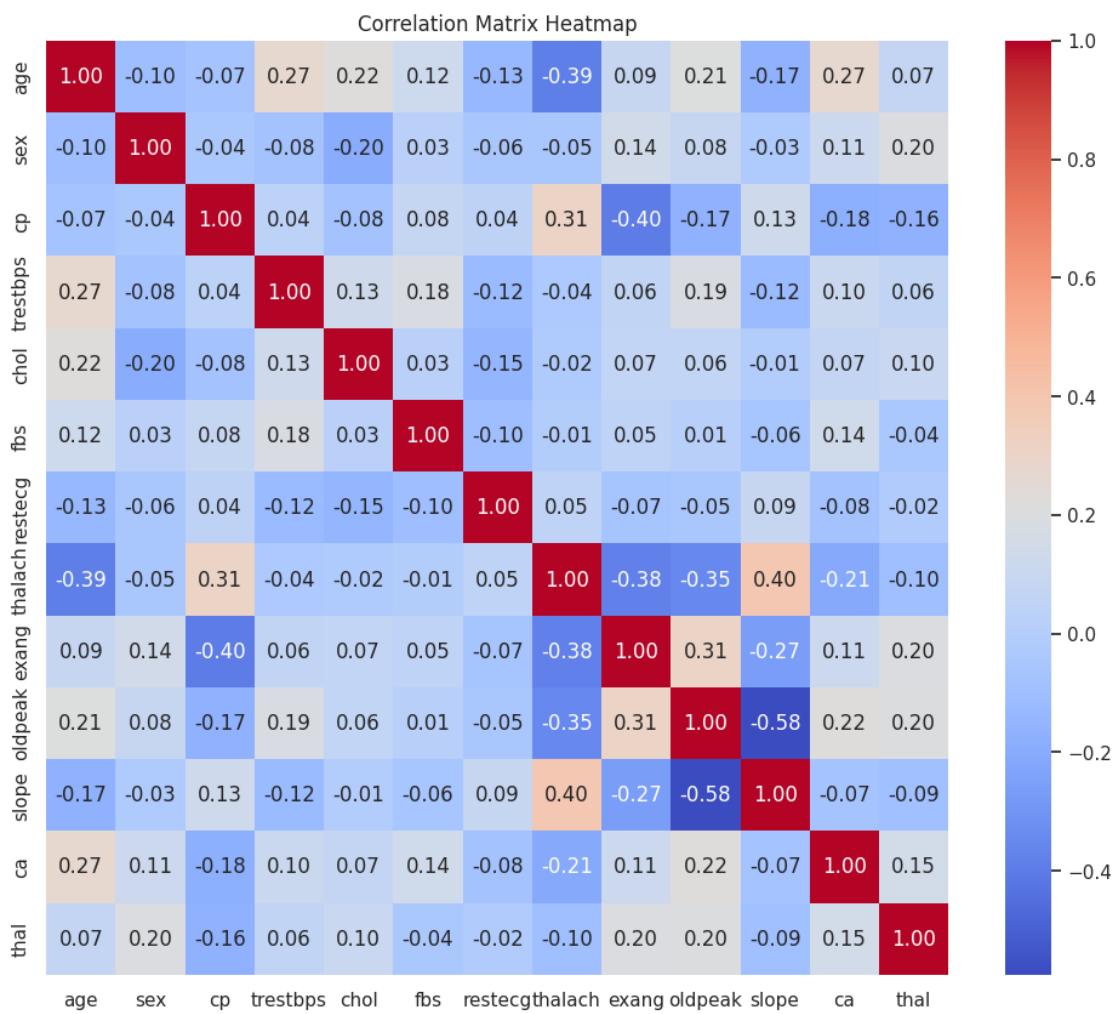
### توزیع تالاسمی

ویژگی تالاسمی دارای سه دسته است و تالاسمی نرمال (۲) شایع‌تر است، پس از آن نقص ثابت (۳) و نقص قابل برگشت (۱) کمتر شایع است.

این توزیع‌ها بینشی از ویژگی‌های بیماران در مجموعه داده ارائه می‌دهند و به درک فراوانی و تغییرپذیری ویژگی‌های مرتبط با بیماری قلبی کمک می‌کنند.

ماتریس همبستگی برای درک روابط بین ویژگی‌ها محاسبه می‌شود:

در این بخش، ماتریس همبستگی ویژگی‌های مختلف بیماری مجموعه داده بیماری قلبی نمایش داده شده است. تحلیل این ماتریس به ما کمک می‌کند تاروابط بین ویژگی‌ها را بهتر درک کنیم.



شکل ۴۳: ماتریس همبستگی

**سن (age):** همبستگی متوسط مثبت با ویژگی های trestbps (فشار خون در حالت استراحت) و chol (کلسترول) دارد. این همبستگی ها نشان می دهد که با افزایش سن، معمولاً فشار خون و کلسترول افزایش می یابد.

**جنسیت (sex):** همبستگی ضعیفی با بیشتر ویژگی ها دارد. همچنین، همبستگی منفی ضعیفی با thalach دارد که نشان می دهد مردان ممکن است حداکثر ضربان قلب کمتری نسبت به زنان داشته باشند.

**نوع درد قفسه سینه (cp):** همبستگی منفی متوسطی با age و trestbps دارد و همبستگی مثبت با thalach دارد. این همبستگی ها نشان می دهد که نوع درد قفسه سینه می تواند با این ویژگی ها مرتبط باشد.

**فشار خون در حالت استراحت (trestbps):** همبستگی مثبتی با age و chol (کلسترول) دارد. این نشان می دهد که فشار خون در حالت استراحت با افزایش سن و سطح کلسترول افزایش می یابد.

**کلسترول (chol):** همبستگی مثبتی با age و trestbps دارد و همبستگی مثبت ضعیفی با fbs دارد. این به این معناست که کلسترول با افزایش سن و فشار خون در حالت استراحت افزایش می یابد و همچنین ممکن است با قند خون ناشتا ارتباط ضعیفی داشته باشد.

**حداکثر ضربان قلب دستیافته (thalach):** همبستگی منفی با age دارد و همبستگی مثبتی با cp دارد. این همبستگی ها نشان می دهند که حداکثر ضربان قلب دستیافته با افزایش سن کاهش می یابد و ممکن است با نوع درد قفسه سینه مرتبط باشد.



آنژین ناشی از ورزش (exang): همبستگی منفی با thalach دارد. این به این معناست که افرادی که آنژین ناشی از ورزش دارند، معمولاً حداکثر ضربان قلب کمتری دارند.

## ۲.۴

داده‌ها را به نسبت ۲۰-۸۰ به دو دسته آموزش و ارزیابی تقسیم می‌کنیم و سپس برای برای پیش‌پردازش داده‌ها از StandardScaler استفاده می‌کنیم.

با استفاده از کد زیر مدل Gaussian Naive Bayes و Gradient Boosting مورد مقایسه قرار می‌دهیم و سپس نتایج را بررسی می‌کنیم.

```
1 from sklearn.naive_bayes import GaussianNB
2 from sklearn.ensemble import GradientBoostingClassifier
3 from sklearn.preprocessing import LabelEncoder
4 from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
5 import seaborn as sns
6 import matplotlib.pyplot as plt
7
8 models = {
9     "GaussianNB": GaussianNB(),
10    "GradientBoosting": GradientBoostingClassifier(random_state=64)
11 }
12
13 results = {}
14 LE = LabelEncoder()
15 LE.fit(y_train)
16
17 for model_name, model in models.items():
18     model.fit(X_train, y_train)
19     y_pred = model.predict(X_test)
20     conf_matrix = confusion_matrix(y_test, y_pred)
21     class_report = classification_report(y_test, y_pred)
22     accuracy = accuracy_score(y_test, y_pred)
23     results[model_name] = {
24         "class_report": class_report,
25         "accuracy": accuracy
26     }
27
28     print(f"Results for {model_name}:")
29     print("\nClassification Report:")
30     print(class_report)
31     print(f"Accuracy: {accuracy}\n")
32
33 # Generate and display the confusion matrix heatmap
34 class_labels = LE.classes_
35 plt.figure(figsize=(10, 7))
```

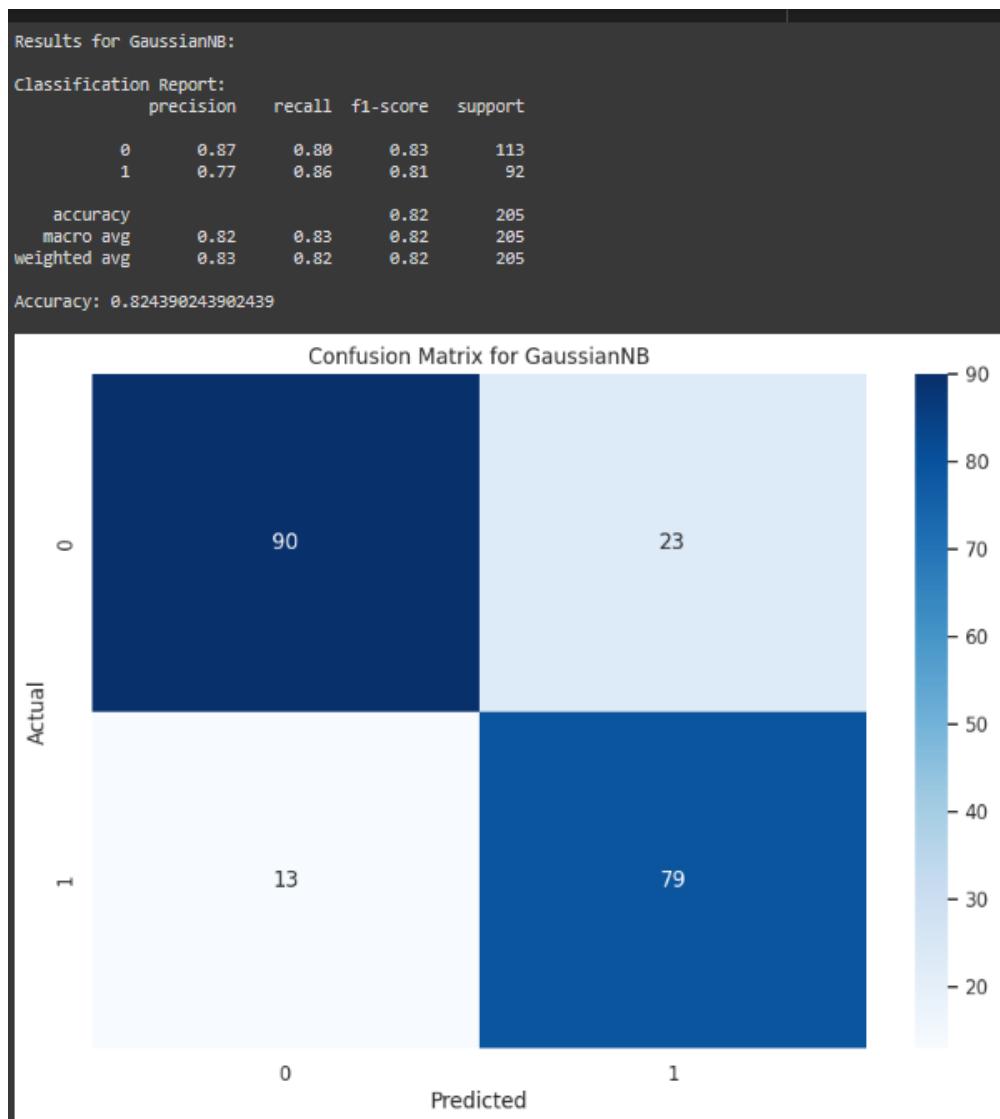
```

36     sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Blues', xticklabels=class_labels,
37         yticklabels=class_labels)
38     plt.xlabel('Predicted')
39     plt.ylabel('Actual')
40     plt.title(f'Confusion Matrix for {model_name}')
41     plt.show()

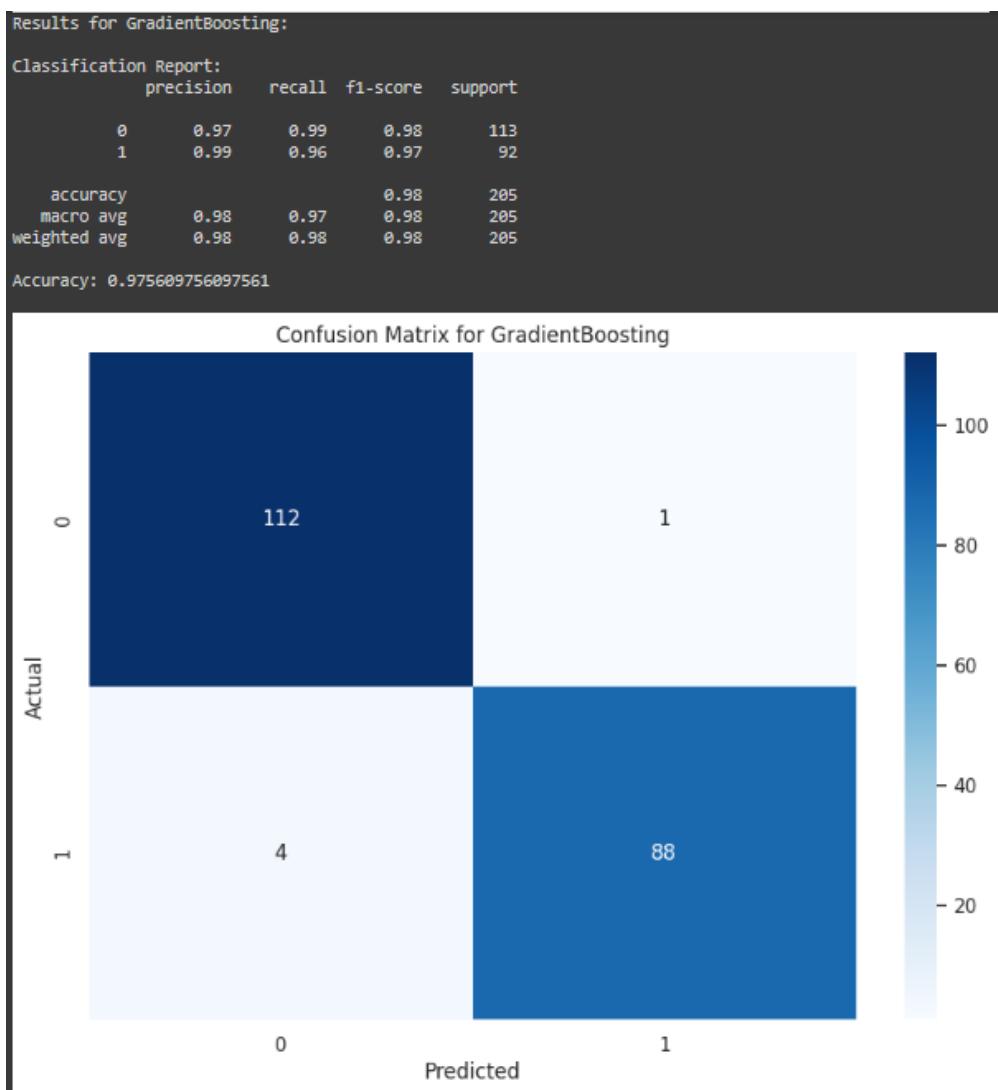
```

Code 9: Gaussian Naive Bayes and Gradient Boosting

نتایج به صورت زیر است.



شکل ۴۴: نتایج مدل ۱



شکل ۴۵: نتایج مدل ۲

## GaussianNB مدل

دقت کلی مدل (Accuracy) 82.4% :

ماتریس درهم‌ریختگی (Confusion Matrix): تعداد نمونه‌های واقعی کلاس 0 که درست طبقه‌بندی شده‌اند: 90، تعداد نمونه‌های واقعی کلاس 0 که اشتباه طبقه‌بندی شده‌اند: 23، تعداد نمونه‌های واقعی کلاس 1 که درست طبقه‌بندی شده‌اند: 79، تعداد نمونه‌های واقعی کلاس 1 که اشتباه طبقه‌بندی شده‌اند: 13.

گزارش طبقه‌بندی (Classification Report): کلاس 0: Precision : 0.83 : F1-score , 0.80 : Recall , 0.87 : Precision ; کلاس 1: Precision : 0.86 : F1-score , 0.82 : Recall , 0.83 : Precision . میانگین وزنی: 0.81 : F1-score , 0.82 : Recall , 0.83 : Precision . در مدل GaussianNB، عملکرد مدل در تشخیص کلاس 1 نسبت به کلاس 0 بهتر بوده است. دقت مدل در کلاس 0 برابر با 87% و در کلاس 1 برابر با 77% است. این نشان می‌دهد که مدل در تشخیص کلاس 1 (بیماری قلبی) عملکرد بهتری دارد.

## GradientBoosting مدل



دقت کلی مدل (Accuracy) : 97.6%

ماتریس درهم‌ریختگی (Confusion Matrix): تعداد نمونه‌های واقعی کلاس 0 که درست طبقه‌بندی شده‌اند: 112، تعداد نمونه‌های واقعی کلاس 0 که اشتباه طبقه‌بندی شده‌اند: 1، تعداد نمونه‌های واقعی کلاس 1 که درست طبقه‌بندی شده‌اند: 88، تعداد نمونه‌های واقعی کلاس 1 که اشتباه طبقه‌بندی شده‌اند: 4.

گزارش طبقه‌بندی (Classification Report): کلاس 0: Precision: 0.97, Recall: 0.98, F1-score: 0.98; کلاس 1: Precision: 0.96, Recall: 0.99, F1-score: 0.97; میانگین وزنی: Precision: 0.98, Recall: 0.98, F1-score: 0.98.

در مدل GradientBoosting، عملکرد مدل به طور قابل توجهی بهتر از مدل GaussianNB بوده است. دقتهای این مدل برابر با 97.6% است. در کلاس 0، مدل با دقتهای 97% و یادآوری 99% عمل کرده است و در کلاس 1 دقتهای 99% و یادآوری 96% عملکرد بسیار خوبی داشته است. نمره F1 نیز در هر دو کلاس بالا بوده و نشان می‌دهد که مدل GradientBoosting در تشخیص داده‌ها بسیار قوی عمل کرده است. به طور کلی، مدل GradientBoosting با دقتهای بالاتر نسبت به مدل GaussianNB عملکرد بهتری داشته و داده‌ها را با دقتهای بالاتری طبقه‌بندی کرده است.

می‌بینیم که در نوع گزارش weighted macro و weighted average در این نتیجه داریم که با رجوع به documentation به نتایج زیر می‌رسیم. دقتهای کلی accuracy برابر میزان نسبت برچسب‌های درست تشخیص داده شده بر کل پیش‌بینی‌های مدل است. در واقع می‌گوید چند مورد از کل پیش‌بینی‌های انجام شده درست هستند. مورد بعدی macro average است که با یک میانگین بدون وزن از precision و recall و f1-score می‌دهد. مثلاً برای این مسئله recall برای ۸۳ درصد گزارش شده که میانگین ۷۸ درصد و ۸۸ درصد است. مورد بعدی weighted average است که با توجه به تعداد داده‌های support در هر کلاس به مقادیر بالا وزن می‌دهد و میانگین می‌گیرد. رابطه زیر نحوه محاسبه این متریک را نشان می‌دهد.

$$\text{Weighted Average} = \frac{\sum_{i=1}^n (\text{Metric}_i \times \text{Support}_i)}{\sum_{i=1}^n \text{Support}_i} \quad (2)$$

برای محاسبه micro average که برای کلاس‌های چندگانه با چند برچسب یا زیرمجموعه‌ای از کلاس‌ها میانگین خرد مشارکت هر کلاس‌ها را برای محاسبه متریک جمع‌آوری می‌کند. به این حال، برای طبقه‌بندی باینری با چند کلاس با macro چند کلاسه گزارش شده است. در اینجا به طور جداگانه گزارش نشده است.

### ۳.۴

پنج نقطه داده تصادفی از مجموعه داده آزمایشی انتخاب می‌شوند تا خروجی‌های واقعی و پیش‌بینی شده مقایسه شوند:

```
1 np.random.seed(64)
2 random_indices = np.random.choice(len(X_test), 5, replace=False)
3 sample_features = X_test[random_indices]
4 sample_true = y_test.iloc[random_indices]
5 sample_preds = {model_name: model.predict(sample_features) for model_name, model in models.items()
6     ()}
7
8 sample_comparison = pd.DataFrame({
9     "True Labels": sample_true.values
10 }, index=random_indices)
11 for model_name, preds in sample_preds.items():
```



```
12     sample_comparison[f"{model_name} Predicted"] = preds  
13  
14 print("\nSample Comparison:")  
15 print(sample_comparison)
```

Code 10: Data sample comparison

Sample Comparison:				
	True Labels	GaussianNB Predicted	GradientBoosting Predicted	
74	0	0	0	0
112	1	1	1	1
42	0	1	1	0
150	0	0	0	0
55	0	0	0	0

شکل ۴۶: مقایسه نمونه‌های مختلف

[لينك Github](#)  
[لينك Colab](#)