

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

## درس ماشین لرنینگ

### پاسخ تمرین/امینی پروژه دوم

نام و نام خانوادگی	علیرضا رضایی
شماره دانشجویی	۴۰۲۰۶۱۰۴
تاریخ	۱۴۰۳ اردیبهشت



## فهرست مطالب

۵	لینک های مربوطه	۱
۵	.....	۱.۱ لینک مربوط به گیت هاب
۵	.....	۲.۱ لینک مربوط به گوگل کلب
۶		سوال اول ۲
۶	.....	۱.۲ بخش اول سوال اول
۷	.....	۲.۲ بخش دوم سوال اول
۸	.....	۳.۲ بخش سوم سوال اول
۱۴		سوال دوم ۳
۱۴	.....	۱.۳
۱۶	.....	۲.۳
۲۲	.....	۳.۳
۲۵	.....	۴.۳
۲۶		سوال سوم ۴
۲۶	.....	۱.۴ تقسیم داده ها به مجموعه های آموزشی و تست
۲۷	.....	۲.۴ شرح پارامترهایتابع train_test_split
۲۷	.....	۳.۴ روش های دیگر برای تقسیم داده ها
۲۸	.....	۴.۴ پارامترهای StratifiedShuffleSplit
۲۸	.....	۵.۴ تقسیم دستی داده ها
۲۹	.....	۶.۴ مدل سازی درخت تصمیم گیری
۲۹	.....	۱.۶.۴ نود ریشه (Root Node)
۳۰	.....	۲.۶.۴ نود چپ فرزند اول
۳۰	.....	۳.۶.۴ نود راست فرزند اول
۳۰	.....	۴.۶.۴ نود چپ فرزند دوم
۳۰	.....	۵.۶.۴ نود راست فرزند دوم
۳۱	.....	۶.۶.۴ نود چپ فرزند سوم
۳۱	.....	۷.۶.۴ نود راست فرزند سوم
۳۱	.....	۸.۶.۴ نود چپ فرزند چهارم
۳۱	.....	۹.۶.۴ نود راست فرزند چهارم
۳۲	.....	۷.۴ درخت تصمیم گیری به صورت دستی
۳۵	.....	۸.۴ ماتریس درهم ریختگی
۳۵	.....	۹.۴ محاسبه دقت (Accuracy)
۳۵	.....	۱۰.۴ محاسبه بازخوانی (Recall)



۳۵	.....	محاسبه دقت (Precision)	۱۱.۴
۳۶	.....	زیان همینگ (Hamming Loss)	۱۲.۴
۳۶	.....	محاسبه امتیاز F1 Score	۱۳.۴
۳۸	.....	اهمیت فرایاپارامترها در درختان تصمیم	۱۴.۴
۳۸	.....	پارامترهای هرس	۱۵.۴
۳۸	.....	بهبود دقت و جلوگیری از بیشبرازش (Overfitting)	۱۶.۴
۳۹	.....	تأثیر و مزایای هرس	۱۷.۴
۳۹	.....	جنگل تصادفی (Random Forest)	۱۸.۴
۴۱	.....	AdaBoost	۱۹.۴
۴۲	.....	سوال چهارم	۵
۴۲	.....	دیتاست	۱.۵
۴۳	.....	تحلیل عملکرد الگوریتم‌های دسته‌بندی شبکه عصبی مصنوعی و بیز ساده برای دسته‌بندی داده‌ها	۲.۵
۴۴	.....	Report Classification	۳.۵
۴۴	.....	True Y	۱.۳.۵
۴۴	.....	Pred Y	۲.۳.۵
۴۵	.....	Labels	۳.۳.۵
۴۵	.....	weight Sample	۴.۳.۵
۴۵	.....	Digits	۵.۳.۵
۴۵	.....	Dict Output	۶.۳.۵
۴۵	.....	Division Zero	۷.۳.۵
۴۵	.....	Report	۸.۳.۵
۴۶	.....	وارد کردن کتابخانه‌ها و بارگذاری داده‌ها	۴.۵
۴۶	.....	بارگذاری و بررسی اولیه داده‌ها	۵.۵
۴۶	.....	تحلیل داده‌های اکتشافی (EDA)	۶.۵
۴۷	.....	تحلیل هیستوگرام‌ها	۱.۶.۵
۴۷	.....	توزیع سن (age)	۲.۶.۵
۴۷	.....	توزیع فشار خون در حالت استراحت	۳.۶.۵
۴۸	.....	توزیع کلسترول (chol)	۴.۶.۵
۴۸	.....	توزیع حداقل ضربان قلب دستیابی (thalach)	۵.۶.۵
۴۸	.....	ماتریس همبستگی	۷.۵
۴۹	.....	تحلیل ماتریس همبستگی	۸.۵
۵۰	.....	پیش پردازش داده‌ها	۹.۵
۵۰	.....	تقسیم‌بندی مجموعه داده	۱۰.۵
۵۰	.....	آموزش و ارزیابی مدل‌ها	۱۱.۵
۵۱	.....	ترسیم منحنی‌های یادگیری	۱۲.۵



۵۳	.....	۱۳.۵ مقایسه نمونه ها
۵۴	.....	۱۴.۵ لینک گیت هاب



## فهرست تصاویر

۶	تابع های فعال ساز ReLU و sigmoid	۱
۶	انواع ReLU	۲
۸	لایه اول شبکه عصبی	۳
۸	شبکه عصبی سوال ۱	۴
۱۱	خروجی شبکه عصبی سوال ۱	۵
۱۳	خروجی شبکه با فعال ساز sigmoid	۶
۱۳	خروجی شبکه با فعال ساز tanh	۷
۱۴	خروجی شبکه با فعال ساز relu	۸
۲۰	ساختار مدل MLP	۹
۲۱	نمودار اتلاف و دقت برای داده های آموزش و اعتبارسنجی	۱۰
۲۱	Classification Report	۱۱
۲۲	Confusion Matrix	۱۲
۲۳	نمودار اتلاف و دقت برای داده های آموزش و اعتبارسنجی برای تابع اتلاف جدید	۱۳
۲۳	Classification Report mse	۱۴
۲۴	Confusion Matrix mse	۱۵
۲۴	loss CBCE	۱۶
۲۵	k-fold cross-validation روش	۱۷
۲۶	stratified k-fold cross-validation روش	۱۸
۲۹	درخت ایجاد شده با توجه به ویژگی ها	۱۹
۳۴	گراف ایجاد شده به کمک الگوریتم دست نویس درخت تصمیم گیرنده	۲۰
۳۶	ماتریس درهم ریختگی	۲۱
۳۷	گزارش دسته بندی	۲۲
۳۷	معیارهای کلی عملکرد مدل	۲۳
۴۰	نتایج پیاده سازی جنگل تصادفی	۲۴
۴۲	نتایج پیاده سازی AdaBoost	۲۵
۴۳	Model ANN	۲۶
۴۴	Parameters ANN	۲۷
۴۷	هیستوگرام ویژگی های کلیدی	۲۸
۴۹	نقشه حرارتی ماتریس همبستگی	۲۹
۵۲	GaussianNB منحنی یادگیری برای	۳۰
۵۲	RandomForest منحنی یادگیری برای	۳۱
۵۳	GradientBoosting منحنی یادگیری برای	۳۲



## ۱ لینک های مربوطه

### ۱.۱ لینک مربوط به گیت هاب

شما می توانید با استفاده از این [لینک \(Github\)](#) به صفحه Github مربوط به این پروژه دسترسی پیدا کنید.

### ۲.۱ لینک مربوط به گوگل کلب

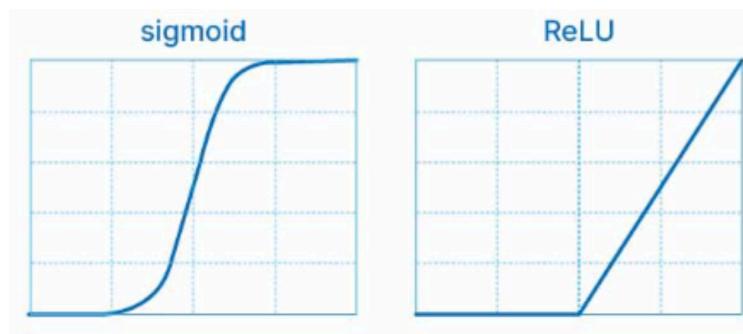
شما می توانید با استفاده از این  
[لینک \(MLproject partOne\)](#)  
[لینک \(MLproject partTwo\)](#)  
[لینک \(MLProject part3\)](#)  
[لینک \(MLproject partFour\)](#)  
به `.notebook` نوشته شده دسترسی پیدا کنید.



## ۲ سوال اول

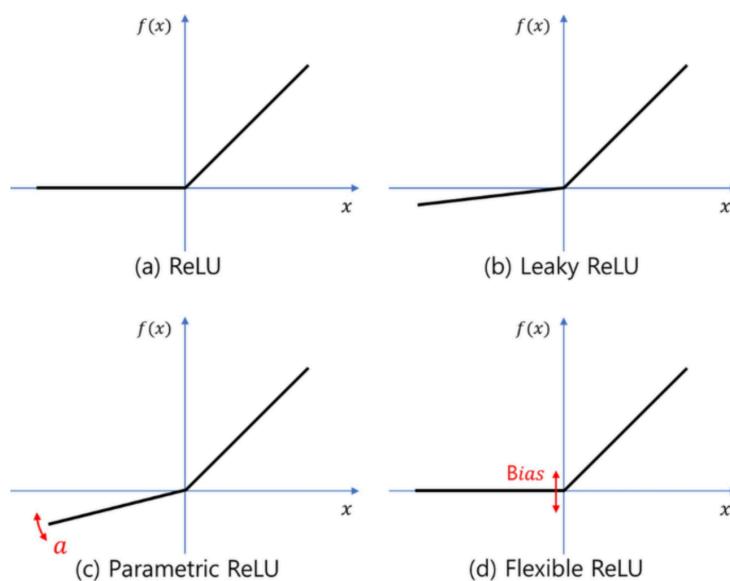
## ۱.۲ بخش اول سوال اول

در این سوال فرض می‌کنیم که تابع فعال‌ساز در لایه یکی مانده به آخر ReLU و در لایه آخر sigmoid است. ابتدا نگاهی به شکل این دو تابع فعال‌ساز می‌اندازیم که در [شکل ۱](#) آمده است.



شکل ۱: تابع‌های فعال‌ساز ReLU و sigmoid

همانطور که در [شکل ۱](#) مشاهده می‌کنید، ReLU می‌تواند مشکل اشباع شدن مقادیر مثبت را نسبت به sigmoid رفع کند. با این حال، مشکلی که این تابع فعال‌ساز دارد، وقتی است که نورون ورودی منفی دریافت کند؛ در این صورت خروجی صفر خواهد بود. این حالت اصطلاحاً به عنوان نورون مرده شناخته می‌شود و در فرآیند یادگیری شرکت نمی‌کند (dying ReLU). برای بهبود عملکرد در شبکه‌های عمیق‌تر، از گزینه‌های دیگری مانند Parametric ReLU یا Leaky ReLU استفاده می‌شود که در [شکل ۲](#) نمایش داده شده است.



شکل ۲: انواع ReLU



فعال‌ساز sigmoid خروجی بین صفر و یک می‌دهد که در طبقه‌بندی دودویی بسیار مفید است و معمولاً در این نوع طبقه‌بندی به عنوان فعال‌ساز لایه آخر استفاده می‌شود تا نگاشت به فضای احتمالاتی ایجاد کند. علاوه بر این، گرادیان همواری به ما می‌دهد که برای بهروزرسانی وزن‌ها مفید است و در فرآیند یادگیری ایده‌آل است. مشکل این تابع فعال‌ساز ناپذید شدن گرادیان است که باعث می‌شود بهروزرسانی وزن‌ها دچار مشکل شود. این مشکل در لایه آخر کمتر دیده می‌شود. نکته دیگر در مورد این تابع فعال‌ساز منطقه اشباع است؛ یعنی وقتی ورودی نورون محدوده وسیعی دارد، خروجی نورون تقاضت زیادی نمی‌کند و ممکن است در فرآیند یادگیری موثر نباشد. در نتیجه، باید توجه کنیم که خروجی نورون‌ها با تابع فعال‌ساز ReLU منفی نشود و همچنین موضوع اشباع شدن فعال‌ساز sigmoid را در نظر بگیریم.

## ۲.۲ بخش دوم سوال اول

تابع فعال‌ساز ReLU در [رابطه ۱](#) نمایش داده شده است:

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

همانطور که دیده می‌شود، گرادیان این تابع برای مقادیر مثبت ۱ (بیانگر رفتار خطی) و برای مقادیر منفی صفر است. گرادیان ELU در [رابطه ۲](#) نمایش داده شده است:

$$\begin{cases} 1 & \text{اگر } x \geq 0, \\ \alpha e^x & \text{اگر } x < 0 \end{cases} \quad (2)$$

همانطور که از [رابطه ۲](#) مشخص است، گرادیان این فعال‌ساز در مقادیر مثبت مانند ReLU برابر با ۱ است که بیانگر رفتار خطی این تابع است. در مقادیر منفی گرادیان برابر با  $\alpha e^x$  است و با منفی تر شدن  $x$  به صفر میل می‌کند. مزایای این تابع نسبت به ReLU عبارتند از:

- ۱- گرادیان غیرصفر در مقادیر منفی به کاهش مشکل ReLU dying گمک می‌کند زیرا به گرادیان اجازه می‌دهد حتی برای ورودی‌های منفی به عقب برگرد (برخلاف ReLU).
- ۲- به دلیل گرادیان غیرصفر برای ورودی‌های منفی، شبکه‌هایی که از ELU استفاده می‌کنند سریع‌تر همگرا می‌شوند زیرا در فرآیند backpropagation بهتر شرکت می‌کنند.
- ۳- چون تمامی مقادیر گرادیان این تابع غیرصفر است، در الگوریتم گرادیان کاهشی تمامی وزن‌ها امکان بهروزرسانی دارند.



## ۳.۲ بخش سوم سوال اول

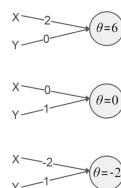
در ابتدا نیاز است شبکه مورد نیاز برای انجام کار طبقه‌بندی را طراحی کیم. چون سه شرط در این سوال (شامل سه ضلع مثلث) مطرح است، از ۳ نورون McCulloch-Pitts در لایه اول استفاده می‌کنیم و از یک نورون نهایی برای عمل رای‌گیری استفاده می‌شود. ابتدا نیاز است تا وزن‌ها و Treshold سه نورون اول را تعیین کنیم. با به دست آوردن معادلات خط AB, BC, AC به ترتیب داریم:

$$2x + y = 6 \quad (3)$$

$$y = 0 \quad (4)$$

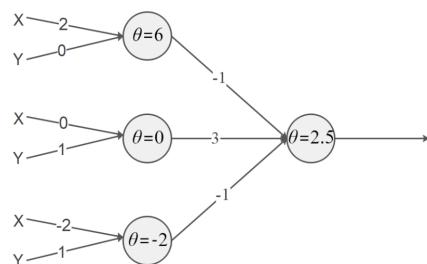
$$-2x + y = -2 \quad (5)$$

با توجه به رابطه ۳، رابطه ۴، رابطه ۵ و دو ورودی  $x$  و  $y$  نمای لایه اول شبکه به صورت شکل ۳ خواهد بود.



شکل ۳: لایه اول شبکه عصبی

سپس باید وزن‌های نورون لایه دوم را طوری تنظیم کنیم که وقتی نورون اول خروجی صفر داشت (سمت چپ خط AB) و نورون دوم خروجی ۱ داشت (بالای خط BC) و نورون سوم خروجی صفر (سمت راست خط AC) خروجی ۱ داشته باشد. برای این منظور باید برای ورودی‌های صفر از لایه قبل پنالتی بیشتری اختصاص دهیم. در نهایت شبکه به همراه Treshold به صورت شکل ۴ خواهد بود.



شکل ۴: شبکه عصبی سوال ۱



حال اگر بخواهیم شبکه را به کمک کلاس مدل‌سازی کنیم، به صورت زیر چهار نورون را تعریف می‌کنیم.

```
1 #define model for dataset
2 def Area(x, y):
3     neur1 = McCulloch_Pitts_neuron([2, 1], 6)
4     neur2 = McCulloch_Pitts_neuron([0, 1], 0)
5     neur3 = McCulloch_Pitts_neuron([-2, 1], -2)
6     neur5 = McCulloch_Pitts_neuron([-1, 3, -1], 2.5)
7
8     z1 = neur1.model(np.array([x, y]))
9     z2 = neur2.model(np.array([x, y]))
10    z3 = neur3.model(np.array([x, y]))
11    z4 = neur5.model(np.array([z1, z2, z3]))
12
13    return list([z4])
```

Code 1: building the neurons



سپس به صورت زیر ۲۰۰۰ نقطه در این محدوده تولید می‌کنیم و آن را به عنوان ورودی به شبکه می‌دهیم. باید نقاطی که شبکه به عنوان کلاس ۱ در نظر می‌گیرد سبز و بقیه نقاط (به عنوان کلاس صفر) قرمز باشد.

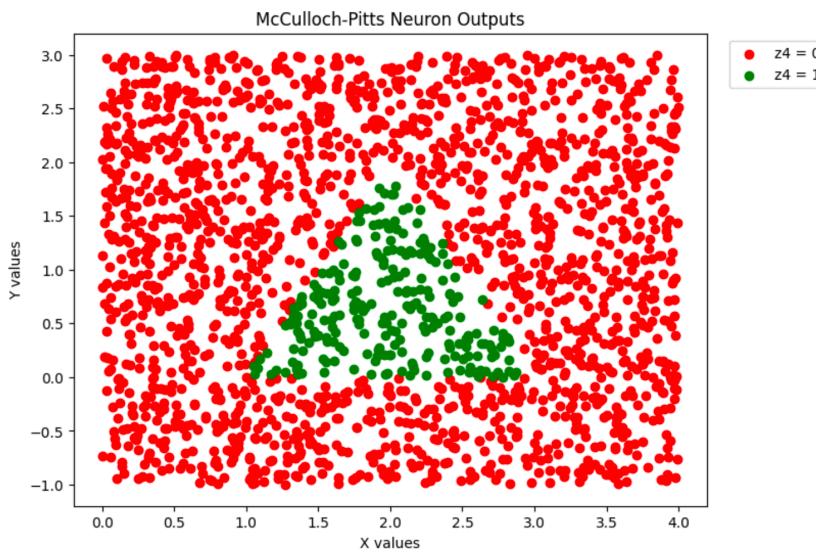
```

1 num_points = 2000
2 x_values = np.random.uniform(0, 4, num_points) # Updated x-axis limits
3 y_values = np.random.uniform(-1, 3, num_points) # Updated y-axis limits
4 # Initialize lists to store data points for different z5 values
5 red_points = []
6 green_points = []
7
8 # Evaluate data points using the Area function
9 for i in range(num_points):
10     z4_value = Area(x_values[i], y_values[i])
11     if z4_value == [0]:
12         red_points.append((x_values[i], y_values[i]))
13     else:
14         green_points.append((x_values[i], y_values[i]))
15
16 # Separate x and y values for red and green points
17 red_x, red_y = zip(*red_points)
18 green_x, green_y = zip(*green_points)
19
20 # Plotting
21 plt.figure(figsize=(8, 6))
22 plt.scatter(red_x, red_y, color='red', label='z4 = 0')
23 plt.scatter(green_x, green_y, color='green', label='z4 = 1')
24 plt.xlabel('X values')
25 plt.ylabel('Y values')
26 plt.title('McCulloch-Pitts Neuron Outputs')
27 plt.legend(loc = 'upper right', bbox_to_anchor=(1.2, 1.0))
28 plt.show()

```

Code 2: output of the network

خروجی مدل به صورت شکل ۱۹ است.



شکل ۵: خروجی شبکه عصبی سوال ۱

همانطور که مشاهده می‌شود، طبقه‌بندی به خوبی انجام شده و نقاط داخل مثلث (به مرز نواحی دقیق شود) همگی با رنگ سبز و برچسب ۱ مشخص شده‌اند و نقاط خارج آن با رنگ قرمز و برچسب صفر هستند و نتیجه می‌گیرد که شبکه به خوبی در جدا کردن داده‌ها عمل می‌کند.

در قسمت بعد اثر activation function های مختلف را در مدل می‌بینیم. برای انجام این کار یک ورودی به ورودی‌های کلاس McCulloch-Pitts اضافه می‌کنیم.

```

1 class McCulloch_Pitts_neuron():
2
3     def __init__(self, weights, activation='threshold', threshold=0):
4         self.weights = np.array(weights)
5         self.activation = activation
6         self.threshold = threshold
7
8     def sigmoid(self, x):
9         return 1 / (1 + np.exp(-x))
10
11    def tanh(self, x):
12        return np.tanh(x)
13
14    def relu(self, x):
15        return np.maximum(0, x)
16
17    def model(self, x):
18        linear_output = self.weights @ x
19        if self.activation == 'threshold':
20            return 1 if linear_output >= self.threshold else 0

```



```

21     elif self.activation == 'sigmoid':
22         return self.sigmoid(linear_output)
23     elif self.activation == 'tanh':
24         return self.tanh(linear_output)
25     elif self.activation == 'relu':
26         return self.relu(linear_output)
27     else:
28         raise ValueError("Unsupported activation function")

```

Code 3: McCulloch-Pitts with different activation functions

همانطور که در بالا مشاهده می‌شود، یک ورودی activation به کلاس اضافه شده که در حالتی که به طور خاص تعریف نشود در حالت threshold قرار دارد. همانطور که دیده می‌شود، این توابع در قالب سه متده (method) تعریف شده‌اند و در نهایت با یک دستور if که با توجه به ورودی activation شرط‌گذاری می‌کند، به ما خروجی می‌دهد. در نهایت یک تابع classify and plot تولید شده که تنها ورودی آن نوع تابع فعال‌ساز است و با توجه به این تابع برای ما کار رسم را انجام می‌دهد.

```

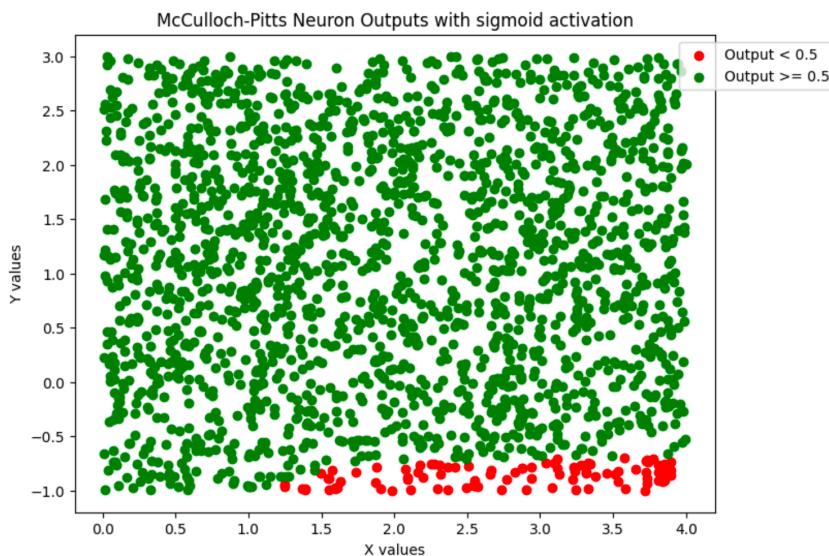
1 def classify_and_plot(activation):
2     num_points = 2000
3     x_values = np.random.uniform(0, 4, num_points)
4     y_values = np.random.uniform(-1, 3, num_points)
5
6     red_points = []
7     green_points = []
8
9     for i in range(num_points):
10        z4_value = Area(x_values[i], y_values[i], activation=activation)
11        if z4_value < 0.5:
12            red_points.append((x_values[i], y_values[i]))
13        else:
14            green_points.append((x_values[i], y_values[i]))
15
16    red_x, red_y = zip(*red_points) if red_points else ([], [])
17    green_x, green_y = zip(*green_points) if green_points else ([], [])
18
19    plt.figure(figsize=(8, 6))
20    plt.scatter(red_x, red_y, color='red', label='Output < 0.5')
21    plt.scatter(green_x, green_y, color='green', label='Output >= 0.5')
22    plt.xlabel('X values')
23    plt.ylabel('Y values')
24    plt.title(f'McCulloch-Pitts Neuron Outputs with {activation} activation')
25    plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1.0))
26    plt.show()
27
28 classify_and_plot('threshold')
29 classify_and_plot('sigmoid')
30 classify_and_plot('tanh')

```

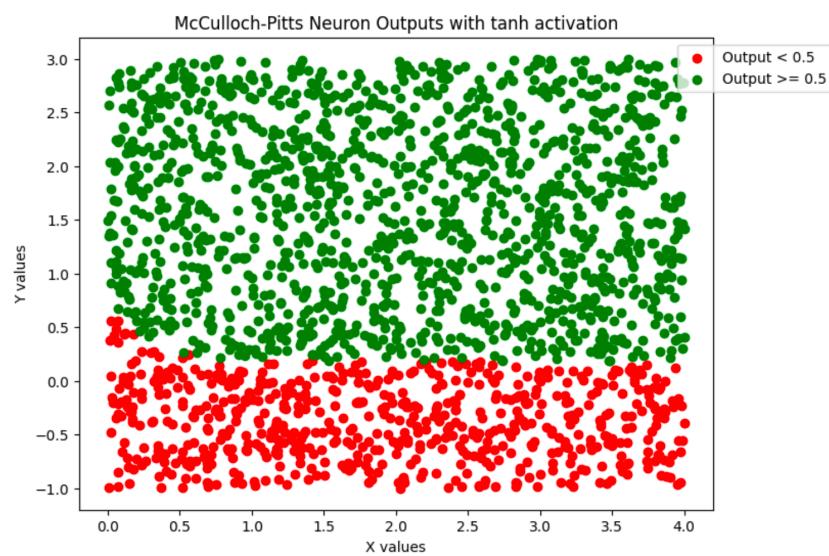
```
31 classify_and_plot('relu')
```

Code 4: plotting the output based on different activation functions

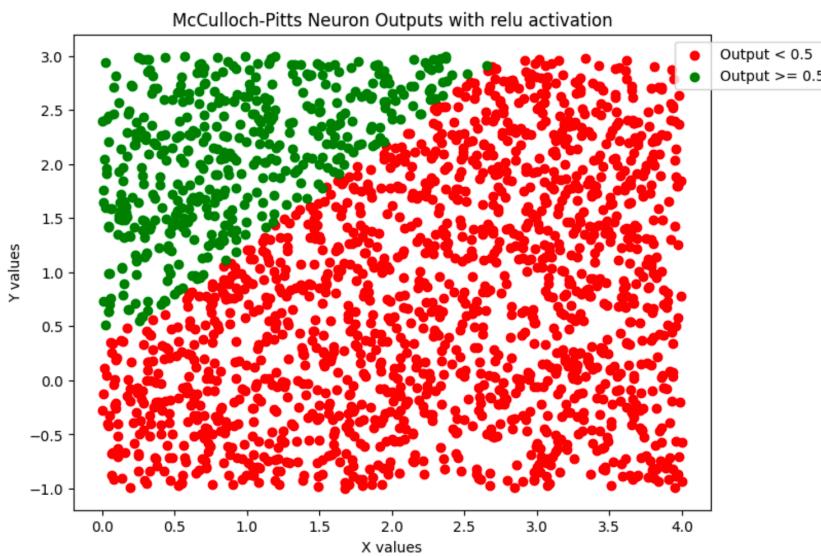
نتایج در شکل ۶، شکل ۷ و شکل ۸ آمده است.



شکل ۶: خروجی شبکه با فعال‌ساز sigmoid



شکل ۷: خروجی شبکه با فعال‌ساز tanh



شکل ۸: خروجی شبکه با فعال‌ساز  $\text{relu}$

همانطور که مشاهده می‌شود، با تغییر توابع فعال‌ساز، عملکرد شبکه به طور کلی تحت تاثیر قرار می‌گیرد و شبکه باید با در نظر گرفتن این توابع فعال‌ساز طراحی شود. در این حالت چون با threshold این شبکه طراحی شده بود، همین فعال‌ساز نیز بهترین گزینه است و در حالت‌های دیگر طبقه‌بند به خوبی کار نمی‌کند و تنها بخشی از کار طبقه‌بندی را انجام می‌دهد.

### ۳ سوال دوم

۱.۳

CWRU Electric Reliance مجموعه داده که در محیط آزمایشگاهی تولید شده، شامل داده‌های بلبرینگ‌ها استفاده شده در موتور القایی با توان دو اسب بخار می‌باشد. این سیستم شامل یک ترانسdiyosr 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 ۱۷۹۷ تا ۱۷۲۰ 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 نرخ



نمونه برداری ۱۲ کیلوهرتز هستند.

- آ) با توجه به باقی مانده شماره دانشجویی بر ۴، داده‌های Normal\_0 و B007\_0، IR007\_0، Normal\_0@6 و ۰@6\_B007\_0 را دانلود می‌کنیم که از این سری داده‌ها، قسمت‌های مربوط به عیب سمت درایور را انتخاب می‌کنیم. برای ایجاد دیتای مورد نظر از  $M=200$  و  $N=300$  استفاده می‌کنیم و داده‌ها را زیر هم قرار می‌دهیم تا به یک ماتریس  $300 \times 800$  برسیم. همچنین برای اینکه دیتای انتخاب شده در اجرای 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     return np.array(samples)
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)
```



```

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
46 df = pd.DataFrame(data_combined)
47 df['label'] = labels_combined

```

Code 5: 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=4, 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=4, shuffle=
9                                                 True)

```

Code 6: 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 وسعی 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 کنیم.

## ۲.۳

در این قسمت به پیاده‌سازی یک مدل MLP می‌پردازیم وسعی 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         self.n_iter = n_iter
35         self.loss_fn = loss_fn
36         self.eta = eta
37         self.random_state = random_state
38         np.random.seed(self.random_state) # Set random seed
39
40     def _init_weights(self):
41         self.ws, self.bs = [], [] # Weight and bias lists for each layer
42         self.as_ = [None] * len(self.hidden_layer_sizes) # Initialize as_ with None
43         all_layers = [self.input_size] + self.hidden_layer_sizes + [self.output_size] # All
44         layer sizes
45         num_layers = len(all_layers)
46         for i in range(1, num_layers):
47             w = np.random.randn(all_layers[i-1], all_layers[i]) # Randomly initialize
48             weights
49             b = np.random.randn(all_layers[i]) # Randomly initialize biases
50             self.ws.append(w)
51             self.bs.append(b)
```



```
51     def fit(self, X, y, X_val=None, y_val=None): # Add optional validation data
52         n, self.input_size = X.shape
53         self._init_weights()
54         train_losses = []
55         val_losses = []
56         train_accs = []
57         val_accs = []
58         for _ in range(self.n_iter):
59             y_hat = self.predict(X)
60             loss = self.loss_fn(y, y_hat)
61             self._gradient_descent(X, y, y_hat)
62             train_losses.append(loss)
63             train_acc = accuracy(y, y_hat)
64             train_accs.append(train_acc)
65             if X_val is not None and y_val is not None:
66                 val_loss = self.loss_fn(y_val, self.predict(X_val))
67                 val_losses.append(val_loss)
68                 val_acc = accuracy(y_val, self.predict(X_val))
69                 val_accs.append(val_acc)
70                 print(f"Train Loss: {loss:.4f} | Train Acc: {train_acc:.4f} | Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.4f}")
71             else:
72                 print(f"Train Loss: {loss:.4f} | Train Acc: {train_acc:.4f}")
73             if X_val is not None and y_val is not None:
74                 self.plot_history(train_losses, val_losses, train_accs, val_accs)
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')
```



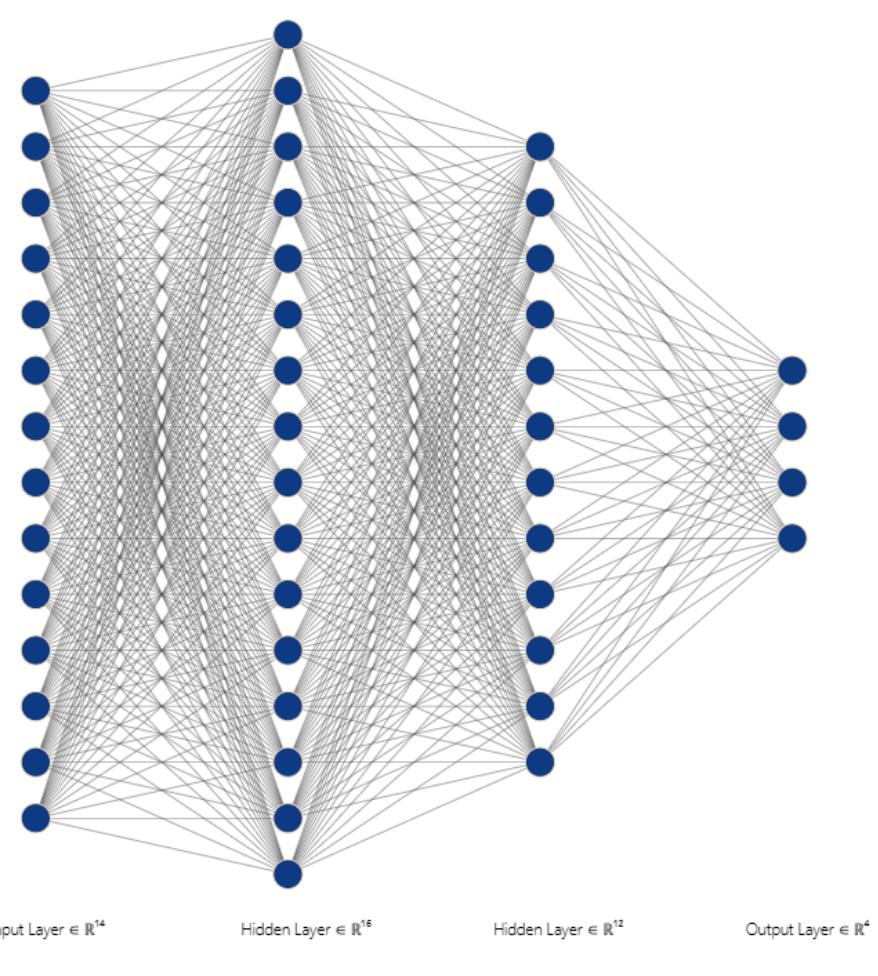
```
95     plt.ylabel('Accuracy')
96     plt.title('Training and Validation Accuracy')
97     plt.legend()
98
99     plt.show()
100
101    def _gradient_descent(self, X, y, y_hat):
102        delta = y_hat - y # Compute difference between predicted and true values
103        for j in range(len(self.ws)-1, 0, -1):
104            w_grad = (self.as_[j-1].T @ delta) / len(y) # Compute weight gradient
105            b_grad = delta.mean(0) # Compute bias gradient
106            self.ws[j] -= self.eta * w_grad # Update weights
107            self.bs[j] -= self.eta * b_grad # Update biases
108            delta = (delta @ self.ws[j].T) * (self._activation_derivative(self.hs[j-1], self.
109 .hidden_activation))
110
111    def predict(self, X):
112        self.hs = [] # Hidden layer outputs
113        self.as_ = [] # Activation function outputs
114        for i, (w, b) in enumerate(zip(self.ws[:-1], self.bs[:-1])):
115            a = self.as_[i-1].copy() if i>0 else X.copy() # Input to the hidden layer
116            self.hs.append(a @ w + b) # Compute hidden layer output
117            self.as_.append(self._activation_function(self.hs[i], self.hidden_activation))
118        # Apply activation function
119        y = self._activation_function(self.as_[-1] @ self.ws[-1] + self.bs[-1], self.
120 .output_activation) # Output layer activation
121
122        return y
123
124
125    def _activation_function(self, x, activation):
126        if activation == 'relu':
127            return np.maximum(0, x) # ReLU activation
128        elif activation == 'sigmoid':
129            return 1 / (1 + np.exp(-x)) # Sigmoid activation
130        else:
131            raise ValueError("Invalid activation function.")
132
133
134    def _activation_derivative(self, x, activation):
135        if activation == 'relu':
136            return np.where(x > 0, 1, 0) # Derivative of ReLU activation
137        elif activation == 'sigmoid':
138            sigmoid = self._activation_function(x, 'sigmoid')
139            return sigmoid * (1 - sigmoid) # Derivative of Sigmoid activation
140        else:
141            raise ValueError("Invalid activation function.")
```

Code 7: MLP

ساختار بهترین مدل ایجاد شده به صورت زیر است.

```
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=4)
```

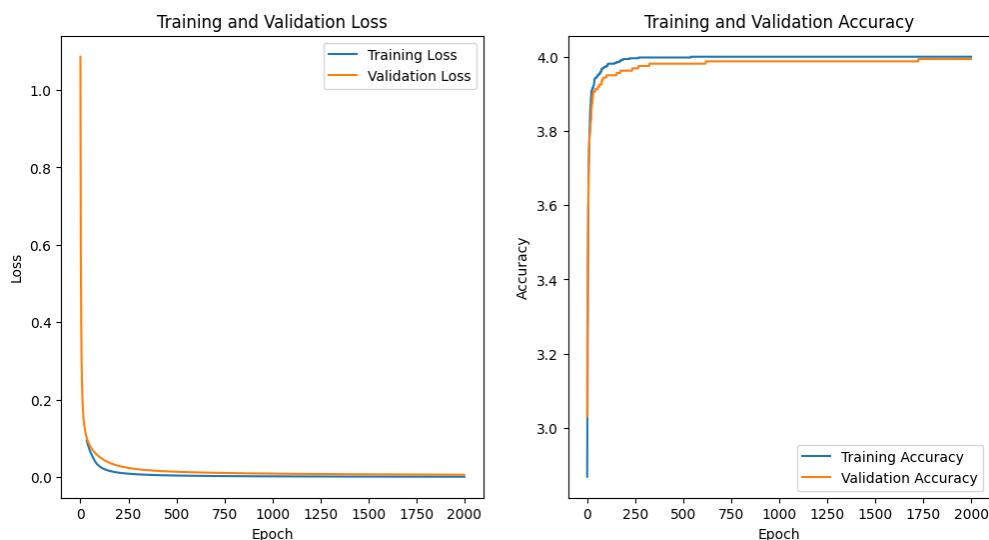
Code 8: Splitting data



شکل ۹: ساختار مدل MLP

همانطور که مشخص است مدل از دو لایه پنهان شامل ۱۶ و ۱۲ نود تشکیل شده است. در لایه‌های میانی از توابع فعال‌ساز Relu استفاده می‌کنیم و در لایه خروجی از تابع سیگموید بهره می‌گیریم.

در لایه خروجی ۴ نود داریم که هرکدام برای مشخص کردن یکی از ۴ کلاس داده ما استفاده می‌شوند (یک کلاس سالم و سه کلاس عیب). تابع اталافی که در این قسمت استفاده می‌شود، تابع BCE (Binary Cross Entropy) است. با توجه به اهمیت داده اعتبارسنجی برای بدست آوردن مقادیر هایپرپارامترها از آن بهره می‌گیریم و بدین ترتیب مدل را بهبود می‌دهیم. نتایج مرحله آموزش برای داده‌های آموزش و اعتبارسنجی به صورت زیر است.



شکل ۱۰: نمودار اتلاف و دقت برای داده‌های آموزش و اعتبارسنجی

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

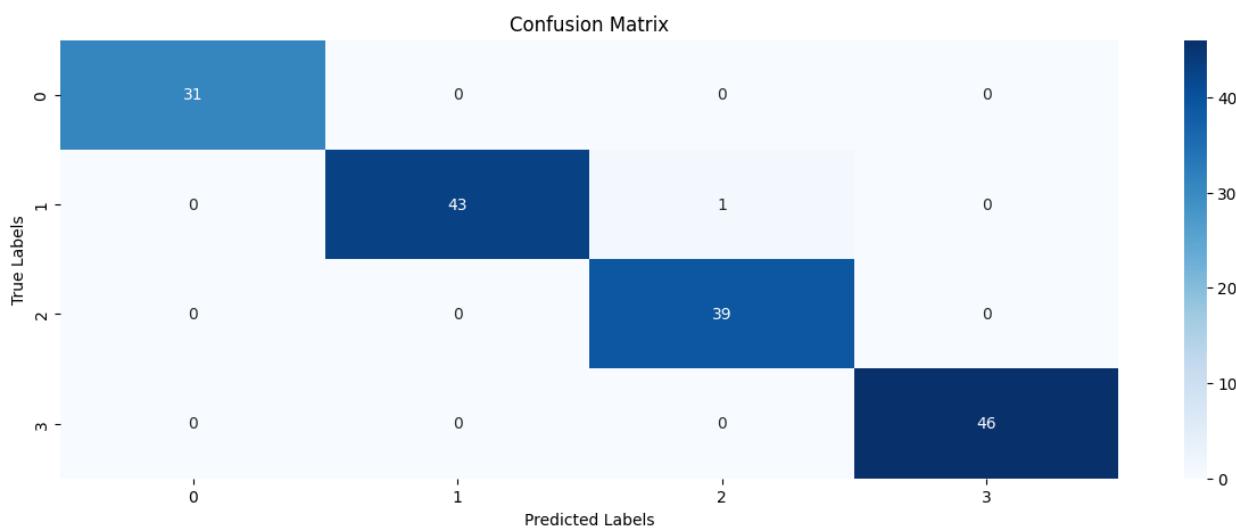
Train Loss: 0.0009 | Train Acc: 4.0000 | Val Loss: 0.0061 | Val Acc: 3.9937

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

Classification Report : ۱۱



شکل ۱۲: Confusion Matrix

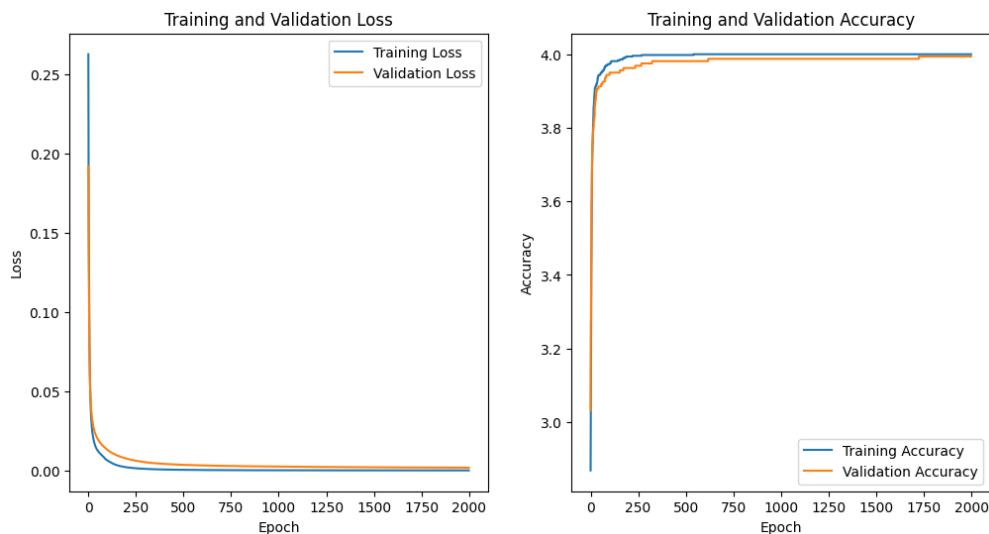
با توجه به نتایج Classification Report مشاهده می‌شود که به طور کلی دقت ۹۹ درصد را در داده آزمون بدست آورده‌ایم. با نگاهی دقیق‌تر در می‌یابیم که در معیار precision در کلاس صفر، یک و سه دقت ۱۰۰ درصد را داریم، این به این معنی است که تمامی نمونه‌های موجود در این سه کلاس را تشخیص داده‌ایم و نمونه‌ای در این سه کلاس نبوده که تشخیص داده نشود اما در کلاس ۲ یک نمونه داریم که اشتباه‌آوری کلاس ۱ تشخیص داده شده‌است. معیار recall درست خلاف آن است. در این معیار کلاس‌های ۰، ۲ 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.

### ۳.۳

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 ۱۳: نمودار اتلاف و دقت برای داده‌های آموزش و اعتبارسنجی برای تابع اتلاف جدید

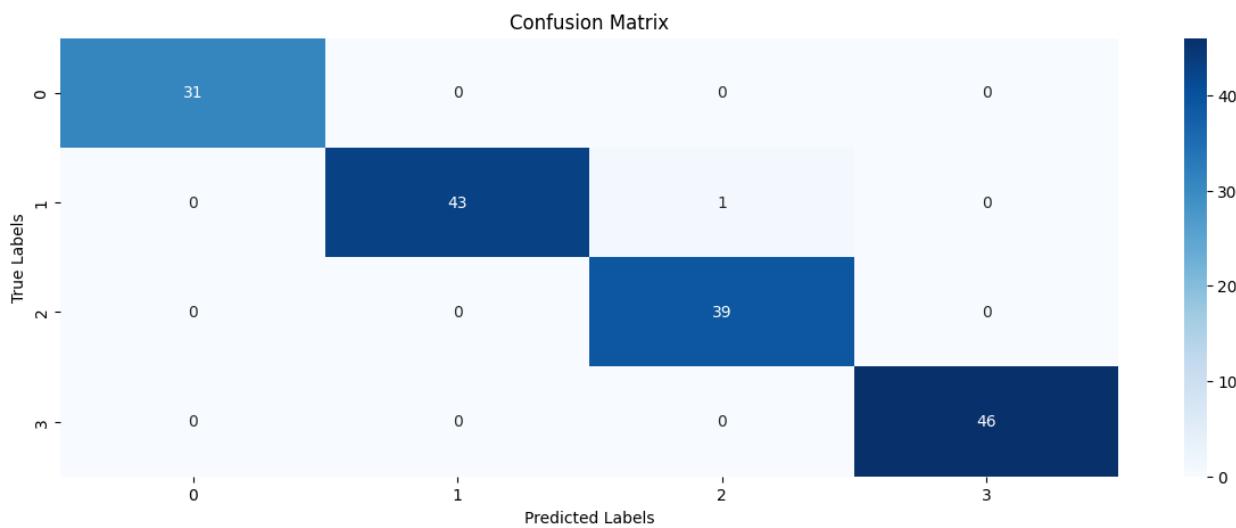
همچنین مقادیر خطأ و دقت برای داده‌های آموزش و اعتبارسنجی به صورت زیر است:

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 ۱۴: Classification Report mse



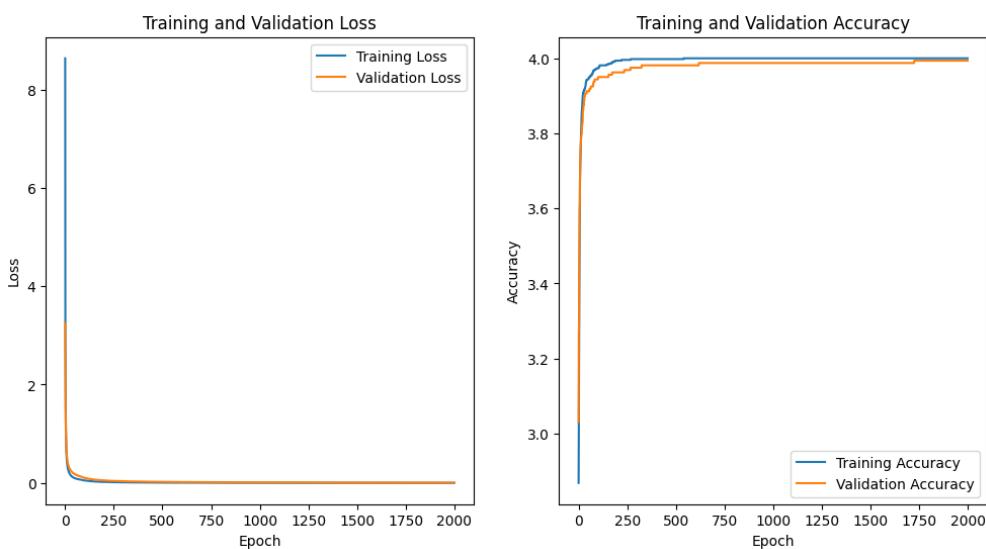
شکل ۱۵: Confusion Matrix mse

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.

$$\text{Categorical Cross-Entropy}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C y_{ij} \log(\hat{y}_{ij}) \quad (6)$$

نتایج و خطاهای به صورت زیر است.



شکل ۱۶: loss CBCE

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



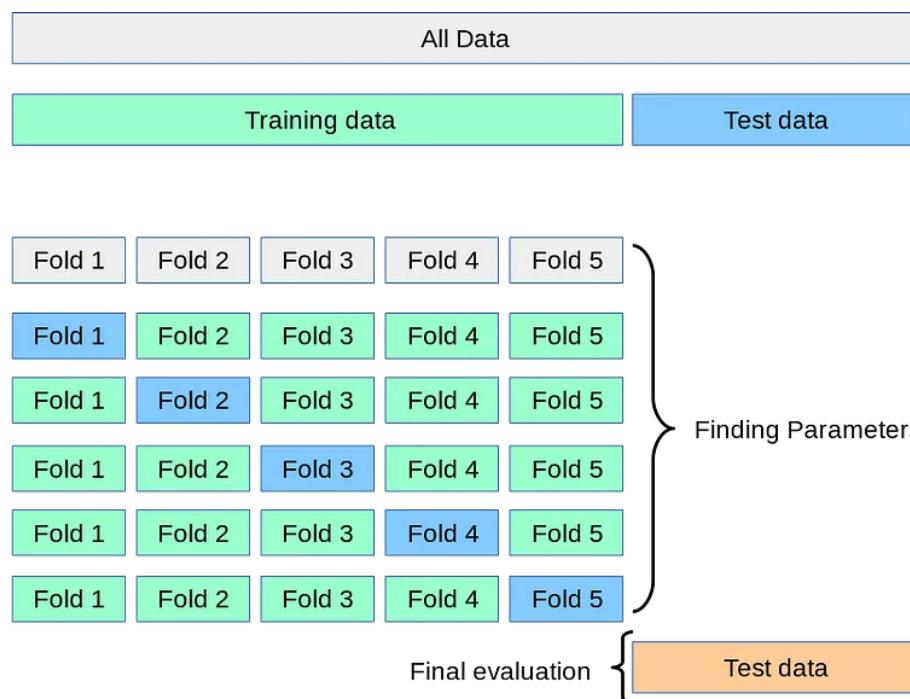
در این مورد هم خطای تغییر زیادی نداشت و نتایج داده آزمون نیز مشابه است.

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.

### ۴.۳

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 ۱۷: 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 تقریباً 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 K-Fold CV روی داده‌های نامتوازن 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 باشند. برای مقابله با این نوع مشکل، Stratified K-Fold با استفاده از فرآیند Stratification استفاده می‌شود.



شکل ۱۸: 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 stratified k-fold cross-validation نیست و با k-fold cross-validation پیاده‌سازی را انجام می‌دهیم.

### ۴ سوال سوم

#### ۱.۴ تقسیم داده‌ها به مجموعه‌های آموزشی و تست

```

1 df = df.sample(frac=1, random_state=4)
2
3 split_idx = int(0.8 * len(df))
4
5 x_train = df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].iloc[:split_idx]
6 x_test = df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].iloc[split_idx:]
7
8 y_train = df['Drug'].iloc[:split_idx]
9 y_test = df['Drug'].iloc[split_idx:]
10
11 print(x_train.shape)
12 print(x_test.shape)

```



تابع `train_test_split` یک ابزار در `sklearn.model_selection` است که برای تقسیم آرایه‌های داده به دوزیر مجموعه: داده‌های آموزشی و تست استفاده می‌شود. این تابع به طور گستردگی در یادگیری ماشین برای اعتبارسنجی مدل‌ها کاربرد دارد.<sup>۱</sup>

#### ۲.۴ شرح پارامترهای تابع `train_test_split`

\*`arrays`: دنباله‌هایی از شاخص‌ها یا داده‌ها. این‌ها می‌توانند لیست‌ها، آرایه‌های نامپای، فریم‌های داده پانداس و غیره باشند.

`test_size`: نسبت داده‌هایی که باید در تقسیم تست گنجانده شوند (عدد اعشاری برای کسر، عدد صحیح برای تعداد نمونه‌های مطلق).

`train_size`: نسبت داده‌هایی که باید در تقسیم آموزشی گنجانده شوند.

`random_state`: برای تضمین تکرارپذیری تقسیم‌ها.

`shuffle`: آیا داده‌ها قبل از تقسیم باید تصادفی شوند. اگر `shuffle=False` باشد، در این صورت `stratify` باید `None` باشد.

`stratify`: اگر `None` نباشد، داده‌ها به صورت طبقه‌بندی شده تقسیم می‌شوند، با استفاده از این به عنوان برچسب‌های کلاس.

#### ۳.۴ روش‌های دیگر برای تقسیم داده‌ها

```
1 import numpy as np
2 import pandas as pd
3
4 def train_test_split_df(dataframe, target_column, test_size=0.2, shuffle=False, random_state=None):
5     if shuffle:
6         dataframe = dataframe.sample(frac=1, random_state=random_state).reset_index(drop=True)
7
8     split_idx = int(len(dataframe) * (1 - test_size))
9
10    X = dataframe.drop(columns=[target_column])
11    y = dataframe[target_column]
12
13    X_train, X_test = X.iloc[:split_idx], X.iloc[split_idx:]
14    y_train, y_test = y.iloc[:split_idx], y.iloc[split_idx:]
15
16    return X_train, X_test, y_train, y_test
```

از کتابخانه `sklearn.model_selection` برای تقسیم داده‌ها به مجموعه‌های آموزشی و تست به صورت طبقه‌بندی شده استفاده می‌شود. این ابزار به خصوص زمانی مفید است که می‌خواهیم نسبت کلاس‌های مختلف در داده‌ها در هر دو مجموعه آموزشی و تست حفظ شود.



## ۴.۴ پارامترهای **StratifiedShuffleSplit**

**:n\_splits** •

- تعداد تقسیم‌هایی که باید انجام شود.
- مقدار پیش‌فرض  $n\_splits=10$  است.
- این مقدار نشان می‌دهد که چند بار باید داده‌ها تقسیم شوند. معمولاً برای اعتبارسنجی متقابل (cross-validation) استفاده می‌شود.

**:test\_size** •

- نسبت یا تعداد نمونه‌هایی که باید در مجموعه تست قرار گیرند.
- اگر عدد اعشاری بین ۰ و ۱ باشد، نشان‌دهنده نسبت داده‌های تست است.
- اگر عدد صحیح باشد، نشان‌دهنده تعداد مطلق نمونه‌های تست است.
- مقدار پیش‌فرض  $test\_size=None$  است.

**:train\_size** •

- نسبت یا تعداد نمونه‌هایی که باید در مجموعه آموزشی قرار گیرند.
- اگر عدد اعشاری بین ۰ و ۱ باشد، نشان‌دهنده نسبت داده‌های آموزشی است.
- اگر عدد صحیح باشد، نشان‌دهنده تعداد مطلق نمونه‌های آموزشی است.
- مقدار پیش‌فرض  $train\_size=None$  است.
- اگر مشخص نشود، به طور خودکار از تفاوت کل داده‌ها با  $test\_size$  محاسبه می‌شود.

**:random\_state** •

- برای تنظیم بذر (seed) تولید اعداد تصادفی استفاده می‌شود.
- این پارامتر تضمین می‌کند که تقسیم داده‌ها در تکرارهای مختلف یکسان باشد (تکرارپذیری).
- مقدار پیش‌فرض  $random\_state=None$  است.

**:shuffle** •

- تعیین می‌کند که آیا داده‌ها قبل از تقسیم تصادفی شوند یا خیر.
- مقدار پیش‌فرض  $shuffle=True$  است.

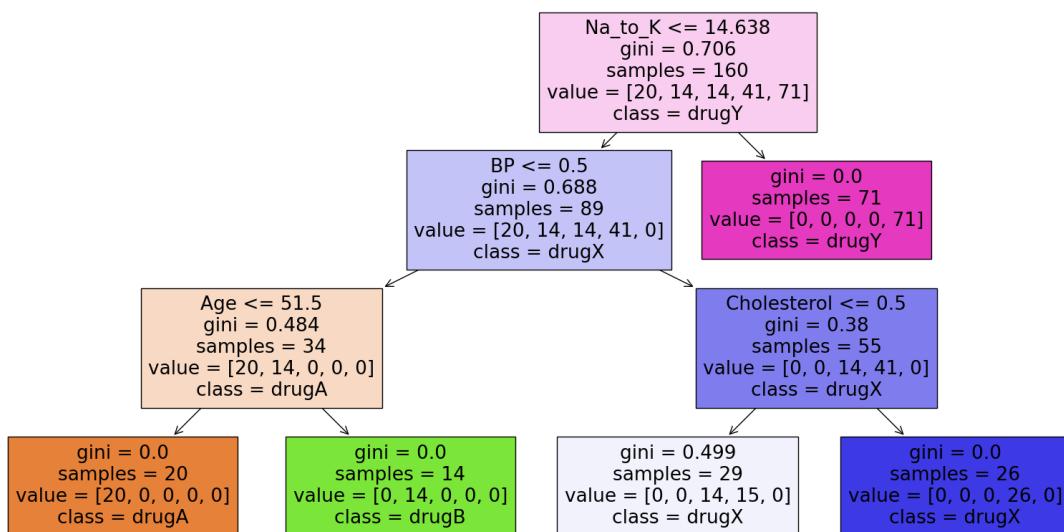
## ۵.۴ تقسیم دستی داده‌ها

می‌توان داده‌ها را به صورت دستی نیز تقسیم نمود. برای این کار می‌توان از ایندکس‌های دیتا فریم یا داده‌ها به صورت آزاده استفاده کرد و سپس به کمک مازولهای موجود مانند `sample` یا `shuffle` یا `sample` ترتیب آن‌ها را تغییر داد و داده‌ها را تقسیم کرد. این روش امکان کنترل دقیق‌تر بر نحوه تقسیم داده‌ها را فراهم می‌کند و می‌تواند در برخی موارد کاربردی باشد.

برای مثال، به صورت زیر می‌توان داده‌ها را دستی تقسیم کرد:  
ابتدا داده‌ها را به صورت آرایه یا دیتا فریم دریافت کنید. از تابع sample shuffle یا برای تغییر ترتیب داده‌ها استفاده کنید. داده‌ها را به نسبت‌های مورد نظر به مجموعه‌های آموزشی و تست تقسیم کنید.

#### ۶.۴ مدل سازی درخت تصمیم گیری

به کمک کتابخانه sklearn مدل درخت تصمیم گیری را فرا می‌خوانیم و بر روی دیتاست train فیت می‌کیم. عملکرد درخت ایجاد شده روی دیتاست تست به شرح زیر می‌باشد.



شکل ۱۹: درخت ایجاد شده با توجه به ویژگی‌ها

شکل ۱۹ درخت ایجاد شده را نمایش می‌دهد. درخت تصمیم‌گیری به کمک معیار Impurity Gini ساخته شده است و در هر نود تقسیم‌بندی‌های مختلفی بر اساس ویژگی‌های داده‌ها انجام شده است. در ادامه، هر نود از درخت به تفصیل تحلیل می‌شود:

#### ۱.۶.۴ نود ریشه (Root Node)

- شرط:  $x[4] \leq 14.638$
- $gini : 0.706$
- تعداد نمونه‌ها: ۱۶۰
- توزیع کلاس‌ها:  $[20, 14, 14, 41, 71]$

این نود نمونه‌ها را به دو گروه تقسیم می‌کند: آن‌هایی که مقدار ویژگی چهارم آن‌ها کمتر یا مساوی ۸۲۹.۱۴ است و آن‌هایی که بیشتر است. impurity در این نود نسبتاً بالاست که نشان‌دهنده تنوع بالای کلاس‌ها در این سطح است.



## ۲.۶.۴ نود چپ فرزند اول

• شرط:  $x[2] \leq 0.5$ •  $gini : 0.688$ 

• تعداد نمونه‌ها: ۸۹

• توزیع کلاس‌ها: [20, 14, 14, 41, 0]

این نود نمونه‌هایی را که ویژگی چهارم آن‌ها کمتر یا مساوی 14.829 است، به دو گروه تقسیم می‌کند: آن‌هایی که مقدار ویژگی دوم آن‌ها کمتر یا مساوی 0.5 است و آن‌هایی که بیشتر است.

## ۳.۶.۴ نود راست فرزند اول

•  $gini : 0.0$ 

• تعداد نمونه‌ها: ۷۱

• توزیع کلاس‌ها: [0, 0, 0, 0, 71]

این نود به صورت برگ (Leaf Node) است که نشان می‌دهد تمامی نمونه‌های این گروه به کلاس ۵ تعلق دارند و impurity صفر است.

## ۴.۶.۴ نود چپ فرزند دوم

• شرط:  $x[0] \leq 51.5$ •  $gini : 0.484$ 

• تعداد نمونه‌ها: ۳۴

• توزیع کلاس‌ها: [20, 14, 0, 0, 0]

این نود نمونه‌هایی را که مقدار ویژگی دوم آن‌ها کمتر یا مساوی 0.5 است، به دو گروه تقسیم می‌کند: آن‌هایی که مقدار ویژگی اول آن‌ها کمتر یا مساوی 50.5 است و آن‌هایی که بیشتر است.

## ۵.۶.۴ نود راست فرزند دوم

• شرط:  $x[2] \leq 0.5$ •  $gini : 0.38$ 

• تعداد نمونه‌ها: ۵۵

• توزیع کلاس‌ها: [0, 0, 14, 41, 0]



این نود نمونه‌هایی را که مقدار ویژگی دوم آن‌ها بیشتر از ۰.۵ است، به دو گروه تقسیم می‌کند: آن‌هایی که مقدار ویژگی دوم آن‌ها کمتر یا مساوی ۰.۵ است و آن‌هایی که بیشتر است.

#### ۶.۶.۴ نود چهارم فرزند سوم

$$gini : 0.0$$

- تعداد نمونه‌ها: ۲۰

- توزیع کلاس‌ها: [20, 0, 0, 0, 0]

این نود به صورت برگ است و نشان می‌دهد تمامی نمونه‌های این گروه به کلاس ۱ تعلق دارند و impurity صفر است.

#### ۷.۶.۴ نود راست فرزند سوم

$$gini : 0.0$$

- تعداد نمونه‌ها: ۱۴

- توزیع کلاس‌ها: [0, 14, 0, 0, 0]

این نود به صورت برگ است و نشان می‌دهد تمامی نمونه‌های این گروه به کلاس ۲ تعلق دارند و impurity صفر است.

#### ۸.۶.۴ نود چهارم فرزند چهارم

$$gini : 0.499$$

- تعداد نمونه‌ها: ۲۹

- توزیع کلاس‌ها: [0, 0, 14, 15, 0]

این نود به دو گروه تقسیم می‌شود: آن‌هایی که به کلاس ۳ تعلق دارند و آن‌هایی که به کلاس ۴ تعلق دارند. impurity در این نود بالاست که نشان‌دهنده تنوع نسبی کلاس‌ها در این گروه است.

#### ۹.۶.۴ نود راست فرزند چهارم

$$gini : 0.0$$

- تعداد نمونه‌ها: ۲۶

- توزیع کلاس‌ها: [0, 0, 0, 26, 0]

این نود به صورت برگ است و نشان می‌دهد تمامی نمونه‌های این گروه به کلاس ۴ تعلق دارند و impurity صفر است.



## ۷.۴ درخت تصمیم‌گیری به صورت دستی

```
1 class Node:
2     """Class to represent a node in the decision tree."""
3     def __init__(self, feature_index=None, threshold=None, left=None, right=None, value=None):
4         self.feature_index = feature_index
5         self.threshold = threshold
6         self.left = left
7         self.right = right
8         self.value = value
9
10
11    def build_tree(data, labels, depth=0, max_depth=3):
12        """Builds the decision tree recursively."""
13        num_samples, num_features = data.shape
14        if num_samples == 0:
15            return None
16        if len(set(labels.tolist())) == 1 or depth == max_depth:
17            return Node(value=most_common_label(labels))
18
19        best_feature, best_threshold = best_split(data, labels)
20
21        if best_feature is None:
22            return Node(value=most_common_label(labels))
23
24        left_idxs = data[:, best_feature] < best_threshold
25        right_idxs = data[:, best_feature] >= best_threshold
26
27        left = build_tree(data[left_idxs], labels[left_idxs], depth+1, max_depth)
28        right = build_tree(data[right_idxs], labels[right_idxs], depth+1, max_depth)
29
30        return Node(best_feature, best_threshold, left, right)
31
32    def most_common_label(labels):
33        return max(set(labels.tolist()), key=list(labels).count)
34
35    def best_split(data, labels):
36        num_samples, num_features = data.shape
37        best_gini = 1.0
38        best_feature, best_threshold = None, None
39
40        for feature_index in range(num_features):
41            thresholds = set(data[:, feature_index])
42            for threshold in thresholds:
43                gini = calculate_gini(data, labels, feature_index, threshold)
44                if gini < best_gini:
45                    best_gini, best_feature, best_threshold = gini, feature_index, threshold
```

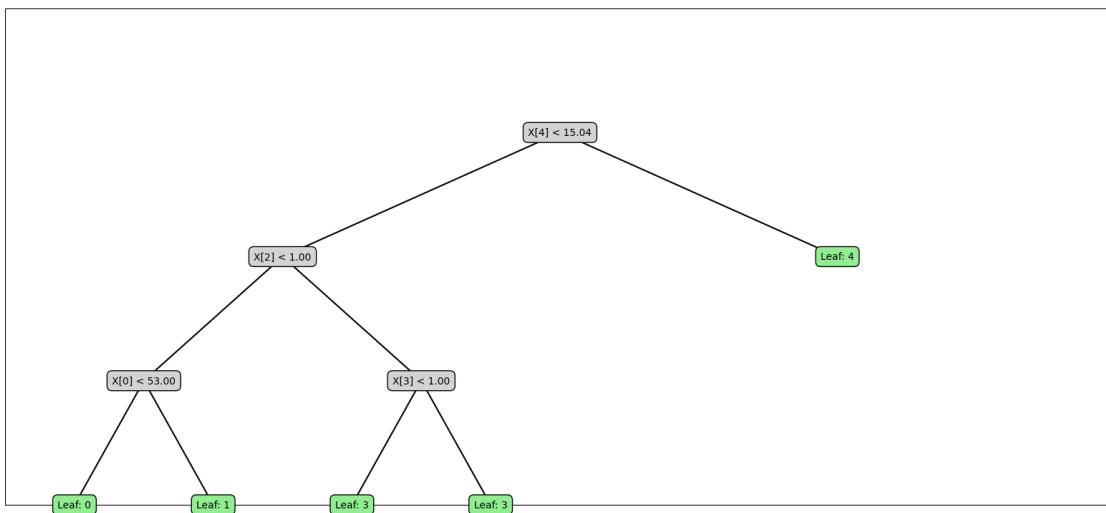


```

15
16     return best_feature, best_threshold
17
18 def calculate_gini(data, labels, feature_index, threshold):
19     left_labels = labels[data[:, feature_index] < threshold]
20     right_labels = labels[data[:, feature_index] >= threshold]
21     left_gini = 1.0 - sum([(left_labels == v).mean()**2 for v in set(left_labels.tolist())])
22     right_gini = 1.0 - sum([(right_labels == v).mean()**2 for v in set(right_labels.tolist())])
23     left_weight = len(left_labels) / len(labels)
24     right_weight = len(right_labels) / len(labels)
25     return left_gini * left_weight + right_gini * right_weight

```

- تعریف کلاس نود: ابتدا یک کلاس به نام `Node` تعریف می‌شود که نماینده یک نود در درخت تصمیم‌گیری است. این کلاس شامل ویژگی‌هایی برای ذخیره شاخص ویژگی، آستانه، نودهای فرزند چپ و راست، و همچنین مقدار نهایی برای نود برگ است.
- تابع ساخت درخت (`build_tree`): این تابع به صورت بازگشته درخت تصمیم‌گیری را می‌سازد. در هر مرحله از ساخت درخت:
  - اگر داده‌ای وجود نداشته باشد، نود برگ برگردانده می‌شود.
  - اگر تمامی برچسب‌ها مشابه باشند یا به حداقل عمق برسیم، نود برگ با بیشترین برچسب مشترک برگردانده می‌شود.
  - بهترین ویژگی و آستانه برای تقسیم داده‌ها پیدا می‌شود.
  - داده‌ها به دو بخش چپ و راست تقسیم می‌شوند و فرآیند ساخت درخت به صورت بازگشته برای هر بخش ادامه می‌یابد.
- تابع بیشترین برچسب مشترک (`most_common_label`): این تابع برچسبی را که بیشترین فراوانی را در داده‌ها دارد، بر می‌گرداند. از این تابع برای تعیین مقدار نهایی نودهای برگ استفاده می‌شود.
- تابع بهترین تقسیم‌بندی (`best_split`): این تابع بهترین ویژگی و آستانه را برای تقسیم‌بندی داده‌ها با کمترین ناخالصی جینی پیدا می‌کند. برای هر ویژگی و هر مقدار آستانه، ناخالصی جینی محاسبه می‌شود و بهترین تقسیم‌بندی انتخاب می‌شود.
- تابع محاسبه جینی (`calculate_gini`): این تابع ناخالصی جینی را برابر یک تقسیم‌بندی خاص محاسبه می‌کند. ابتدا داده‌ها به دو بخش چپ و راست تقسیم می‌شوند و سپس ناخالصی جینی برای هر بخش محاسبه و میانگین وزنی آنها برگردانده می‌شود.
- تابع پیش‌بینی (`predict`): این تابع برای یک نمونه خاص با استفاده از درخت تصمیم‌گیری، برچسب را پیش‌بینی می‌کند. اگر نود برگ باشد، مقدار آن برگردانده می‌شود؛ در غیر این صورت، نمونه بر اساس ویژگی و آستانه مربوطه به نود فرزند چپ یا راست هدایت می‌شود.
- تابع دقت (`accuracy_score`): این تابع دقت مدل را با مقایسه برچسب‌های واقعی و پیش‌بینی شده محاسبه می‌کند. تعداد پیش‌بینی‌های صحیح تقسیم بر تعداد کل نمونه‌ها، دقت مدل را نشان می‌دهد.
- بارگذاری و پیش‌پردازش داده‌ها: داده‌ها از فایل `drug200.csv` بارگذاری می‌شوند و سپس ویژگی‌های غیر عددی (مانند فشار خون، کلسترول، و جنسیت) با استفاده از `LabelEncoder` به مقادیر عددی تبدیل می‌شوند. سپس داده‌ها به مجموعه‌های آموزش و تست تقسیم می‌شوند.
- ساخت درخت و پیش‌بینی: در نهایت، درخت تصمیم‌گیری با استفاده از داده‌های آموزشی ساخته می‌شود و برچسب‌های نمونه‌های تست با استفاده از درخت پیش‌بینی می‌شوند. دقت مدل نیز محاسبه و چاپ می‌شود.



شکل ۲۰: گراف ایجاد شده به کمک الگوریتم دست نویس درخت تصمیم گیرنده

گراف ایجاد شده برای تست درخت در شکل ۲۰ آورده شده است. و دقت این مدل برابر است با ۰.۹.



#### ۸.۴ ماتریس درهم ریختگی

ماتریس درهم ریختگی نمایشی از پیش‌بینی‌های صحیح و نادرست است که به ما کمک می‌کند تا درک بهتری از عملکرد مدل در هر کلاس داشته باشیم. این ماتریس شامل چهار بخش است:

- تعداد مواردی که به درستی به عنوان مثبت طبقه‌بندی شده‌اند: **True Positives (TP)**
- تعداد مواردی که نادرست به عنوان مثبت طبقه‌بندی شده‌اند: **False Positives (FP)**
- تعداد مواردی که به درستی به عنوان منفی طبقه‌بندی شده‌اند: **True Negatives (TN)**
- تعداد مواردی که نادرست به عنوان منفی طبقه‌بندی شده‌اند: **False Negatives (FN)**

#### ۹.۴ محاسبه دقت (Accuracy)

دقت کلی مدل را محاسبه می‌کند و به ما می‌گوید که چه درصدی از کل پیش‌بینی‌ها به درستی انجام شده‌اند. فرمول آن به صورت زیر است:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (7)$$

#### ۱۰.۴ محاسبه بازخوانی (Recall)

Recall نشان می‌دهد که چه درصدی از داده‌های واقعی مثبت به درستی توسط مدل به عنوان مثبت شناسایی شده‌اند. این شاخص برای مواردی که هزینه اشتباهات منفی بالا است مهم می‌باشد. فرمول آن به صورت زیر است:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (8)$$

در حالت میانگین‌گیری ماکرو، بازیابی برای هر کلاس محاسبه می‌شود و سپس میانگین گرفته می‌شود.

$$\text{Recall (macro)} = \frac{1}{C} \sum_{c=1}^C \frac{TP_c}{TP_c + FN_c} \quad (9)$$

که  $C$  تعداد کلاس‌ها می‌باشد.

#### ۱۱.۴ محاسبه دقت (Precision)

Precision دقت نشان می‌دهد که چه درصدی از پیش‌بینی‌های مثبت مدل واقعاً مثبت هستند. این شاخص زمانی اهمیت پیدا می‌کند که هزینه‌های اشتباهات مثبت بالا باشد. فرمول آن به صورت زیر است:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (10)$$

در حالت میانگین‌گیری ماکرو، دقت برای هر کلاس محاسبه می‌شود و سپس میانگین گرفته می‌شود.

$$\text{Precision (macro)} = \frac{1}{C} \sum_{c=1}^C \frac{TP_c}{TP_c + FP_c} \quad (11)$$

که  $C$  تعداد کلاس‌ها می‌باشد.



## ۱۲.۴ زیان همینگ (Hamming Loss)

$$\text{Loss Hamming} = \frac{1}{n} \sum_{i=1}^n \frac{\text{Number of incorrect labels}}{\text{Number of labels}} \quad (12)$$

که  $n$  تعداد نمونه‌ها می‌باشد.

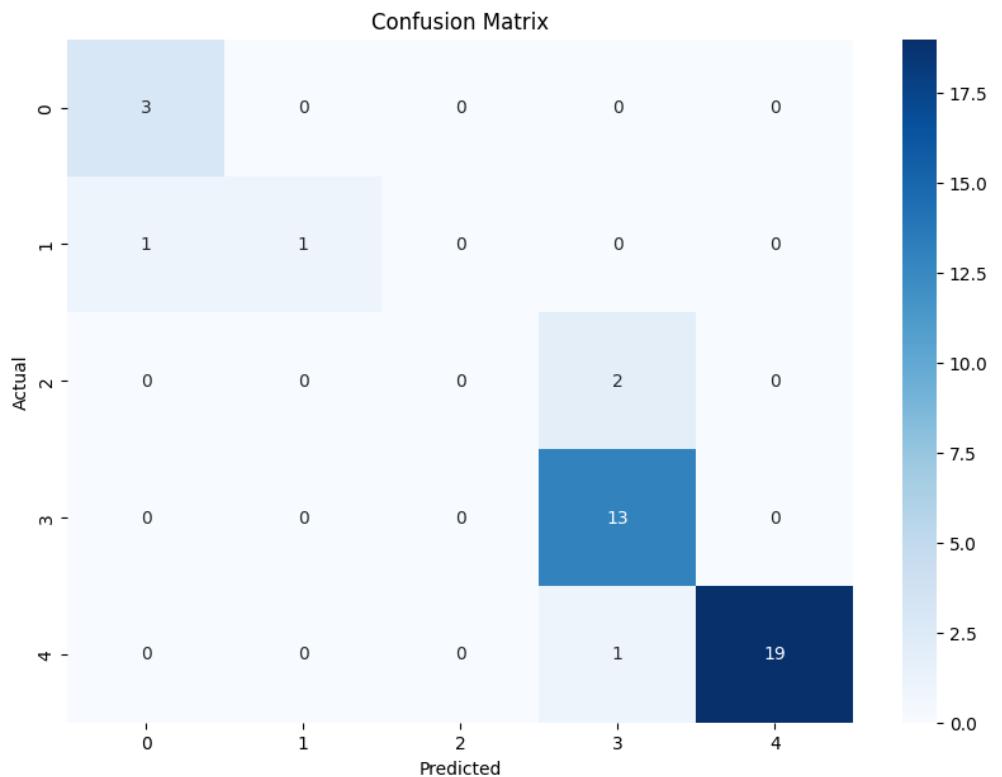
## ۱۳.۴ محاسبه امتیاز F1 Score

امتیاز F1 میانگین هماهنگ دقت و بازخوانی است. فرمول آن به صورت زیر است:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (13)$$

در حالت میانگین گیری ماکرو، امتیاز F1 برای هر کلاس محاسبه می‌شود و سپس میانگین گرفته می‌شود.

$$\text{F1 Score (macro)} = \frac{1}{C} \sum_{c=1}^C 2 \cdot \frac{\text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c} \quad (14)$$



شکل ۲۱: ماتریس درهم ریختگی

با توجه به شکل ۲۱ مدل به خوبی توانسته است همه کلاس‌هارا به جز کلاس‌های C و X تفکیک و کلاس‌بندی کند. دلیل این امر در گراف ۱۹ به خوبی نمایان است.



Classification Report:				
	precision	recall	f1-score	support
0	0.75	1.00	0.86	3
1	1.00	0.50	0.67	2
2	0.00	0.00	0.00	2
3	0.86	0.92	0.89	13
4	0.95	1.00	0.98	20
accuracy			0.90	40
macro avg	0.71	0.68	0.68	40
weighted avg	0.86	0.90	0.87	40

شکل ۲۲: گزارش دسته بندی

```
Accuracy Score: 0.9
Hamming Loss: 0.1
Precision Score (macro): 0.7125
Recall Score (macro): 0.6900000000000001
F1 Score (macro): 0.6789440444612859
```

شکل ۲۳: معیارهای کلی عملکرد مدل

محاسبه امتیازهای کلی و دیگر نیز در شکل ۲۳ آورده شده است. که نشان دهنده عملکرد متوسط مدل است.



#### ۱۴.۴ اهمیت فرآپارامترها در درختان تصمیم

فرآپارامترها در درختان تصمیم جنبه‌های مانند عمق درخت، حداقل تعداد نمونه‌های مورد نیاز در یک گره برگ و حداقل تعداد نمونه‌های لازم برای تقسیم یک گره داخلی را کنترل می‌کنند. تنظیم این موارد می‌تواند از بیش‌برازش جلوگیری کند (جایی که مدل در داده‌های آموخته عملکرد خوبی دارد اما در داده‌های دیده نشده ضعیف است)، که مشکل رایجی در درختان تصمیم است که تمایل دارند ساختارهای داده‌ای بسیار دقیق را یاد بگیرند.

#### ۱۵.۴ پارامترهای هرس

هرس برای کاهش اندازه یک درخت تصمیم با حذف بخش‌هایی از درخت که قدرت پیش‌بینی متغیرهای هدف را ندارند، استفاده می‌شود. دو فرآپارامتر اصلی مرتبط با هرس در درختان تصمیم عبارتند از:

- `max_depth`: حداکثر عمق درخت را محدود می‌کند. عمق کمتر می‌تواند منجر به یک مدل ساده‌تر شود که ممکن است بهتر تعیین داده شود اما اگر بیش از حد کم باشد ممکن است دچار کم‌برازش شود.
- `min_samples_leaf`: حداقل تعداد نمونه‌هایی که یک گره برگ باید داشته باشد را مشخص می‌کند. تنظیم آن روی مقدار زیاد می‌تواند از رشد بیش از حد درخت در عمق جلوگیری کند، اما ممکن است باعث عدم یادگیری آن شود. مقدار بهینه آن به تعداد نمونه‌های آموخته و تعداد برگ‌ها (`num_leaves`) بستگی دارد.

#### ۱۶.۴ بهبود دقت و جلوگیری از بیش‌برازش (Overfitting)

برای بهتر کردن دقت می‌توان:

- از `learning_rate` کوچکتری استفاده نمود.
- تعداد `iteration` را افزایش داد.
- از `max_depth` بزرگتری استفاده نمود.
- مقدار `min_samples_leaf` را کاهش داد.

برای جلوگیری از over-fitting و افزایش سرعت یادگیری:

- از روش‌های `regularization` استفاده نمود.
- مقدار `max_depth` را کاهش داد و لیمیت در نظر گرفت.
- مقدار `iteration` را کاهش داد.
- از `learning_rate` خیلی کوچک پرهیز کرد.
- مقدار `min_samples_leaf` را افزایش داد.



جدول ۱: نتایج با تغییر دادن های پر پارامترها

runtime	mean_f1_score	min_samples_leaf	max_depth
0.036437	0.870081	1	3
0.032688	0.870081	5	3
0.039604	0.870081	10	3
0.035311	0.750309	20	3
0.033853	1.000000	1	5
0.034720	1.000000	5	5
0.032800	1.000000	10	5
0.039748	0.750309	20	5
0.032864	1.000000	1	7
0.036705	1.000000	5	7
0.032009	1.000000	10	7
0.038044	0.750309	20	7
0.034949	1.000000	1	10
0.033467	1.000000	5	10
0.038080	1.000000	10	10
0.032789	0.750309	20	10

## ۱۷.۴ تأثیر و مزایای هرس

- کاهش پیچیدگی: هرس پیچیدگی مدل را کاهش می دهد که می تواند به کاهش بیش برازش کمک کند.
- بهبود یادگیری: با محدود کردن عمق و افزایش حداقل نمونه ها در هر برگ، درخت نسبت به نویز در داده ها کمتر حساس می شود.
- پیش بینی های سریع تر: درختان کوچکتر در انجام پیش بینی ها کارآمدتر هستند.

تنظیم این پارامترها شامل معامله ای بین توانایی مدل برای درک الگوی زیرین داده ها و تعمیم آن به داده های جدید است. استفاده از اعتبارسنجی متقابل برای یافتن تنظیمات بهینه این پارامترها معمولاً یک روش خوب است.

## ۱۸.۴ جنگل تصادفی (Random Forest)

جنگل تصادفی مجموعه ای از درختان تصمیم (غالباً هزاران درخت) است که هر کدام بر روی زیرمجموعه ای از داده ها آموزش دیده اند. این روش به کاهش بیش برازش کمک کرده و تعمیم بهتری نسبت به یک درخت تصمیم ساده دارد.

n\_estimators: تعداد درختان در جنگل.

max\_features: حداکثر تعداد ویژگی ها مورد استفاده برای تقسیم هر گره.

max\_depth: حداکثر عمق هر درخت.



حداقل تعداد نمونه‌ها لازم برای تقسیم یک گره: min\_samples\_split •

حداقل تعداد نمونه‌ها لازم در یک برگ: min\_samples\_leaf •

Σ	precision	recall	f1-score	support
0	0.75	1.00	0.86	3
1	1.00	0.50	0.67	2
2	1.00	1.00	1.00	2
3	1.00	1.00	1.00	13
4	1.00	1.00	1.00	20
accuracy			0.97	40
macro avg	0.95	0.90	0.90	40
weighted avg	0.98	0.97	0.97	40

شکل ۲۴: نتایج پیاده‌سازی جنگل تصادفی

**AdaBoost ۱۹.۴**

یکی دیگر از الگوریتم های بوستینگ است که با تمرکز بر نمونه های دشوارتر و تصحیح پیوسته خطاهای کار می کند. این روش وزن های بیشتری را به نمونه هایی که به درستی طبقه بندی نشده اند اختصاص داده و مدل های پی در پی را آموزش می دهد تا دقیق آنها را بهبود بخشد.

**:base\_estimator •**

- نوع: estimator object، اختیاری، پیش فرض: None
- توضیح: این پارامتر تعیین می کند که از کدام مدل پایه به عنوان تخمین گر ضعیف استفاده شود. به طور پیش فرض اگر مقداری تعیین نشود، از درخت تصمیم ساده DecisionTreeClassifier با حداقل عمق ۱ استفاده می شود.

**:n\_estimators •**

- نوع: int
- پیش فرض: 50
- توضیح: تعداد تخمین گرهای ضعیف که در نهایت در الگوریتم AdaBoost استفاده می شوند. مقدار بالاتر می تواند دقیق مدل را افزایش دهد اما منجر به زمان بیشتر برای آموزش نیز می شود.

**:learning\_rate •**

- نوع: float
- پیش فرض: 1.0
- توضیح: نرخ یادگیری برای به روز رسانی وزن های هر تخمین گر ضعیف. این مقدار تأثیر وزن هر تخمین گر ضعیف را در مدل نهایی تعیین می کند.

**:algorithm •**

- نوع: string
- پیش فرض: 'SAMME.R'
- توضیح: نوع الگوریتم AdaBoost که قرار است استفاده شود. دو مقدار ممکن است:
  - \* 'SAMME': الگوریتم اصلی AdaBoost برای طبقه بندی چند کلاسه.
  - \* 'SAMME.R': نسخه اصلاح شده از SAMME که از احتمال های کلاس ها به جای برچسب های کلاس استفاده می کند.

**:random\_state •**

- نوع: int, RandomState instance، یا None
- پیش فرض: None
- توضیح: این پارامتر برای تنظیم دانه هی تصادفی استفاده می شود تا بتوان نتایج تکرار پذیر به دست آورد. اگر مقداری تنظیم نشود، از وضعیت تصادفی پیش فرض استفاده خواهد شد.



	n_estimators	learning_rate	mean_f1_score	runtime
0	3	0.10	1.0	0.052096
1	3	0.05	1.0	0.042915
2	3	0.01	1.0	0.043041
3	5	0.10	1.0	0.061624
4	5	0.05	1.0	0.048191
5	5	0.01	1.0	0.047810
6	10	0.10	1.0	0.044437
7	10	0.05	1.0	0.044629
8	10	0.01	1.0	0.055737
9	30	0.10	1.0	0.047787
10	30	0.05	1.0	0.057851
11	30	0.01	1.0	0.047629
12	100	0.10	1.0	0.049968
13	100	0.05	1.0	0.056535
14	100	0.01	1.0	0.045439

شکل ۲۵: نتایج پیاده‌سازی AdaBoost

## ۵ سوال چهارم

### ۱.۵ دیتاست

دیتای مورد نظر برای سال ۱۹۸۸ هست که شامل ۷۶ ویژگی است. آزمایش های انجام شده از ۱۴ ویژگی از این ۷۶ ویژگی استفاده شده است. در این دیتا ۰ به منظور نداشتن بیماری قلبی و یک به منظور داشتن بیماری قلبی است.

- سن

- جنسیت

- نوع درد قفسه سینه (۴ مقدار)

- فشار خون استراحتی

- کلسترول سرم به میلی گرم بر دسی لیتر

- قند خون ناشتا > ۱۲۰ میلی گرم بر دسی لیتر

- نتایج الکتروکاردیوگرافی استراحتی (مقادیر ۰، ۱، ۲)

- حداکثر ضربان قلب دست یافته

- آنژین ناشی از ورزش

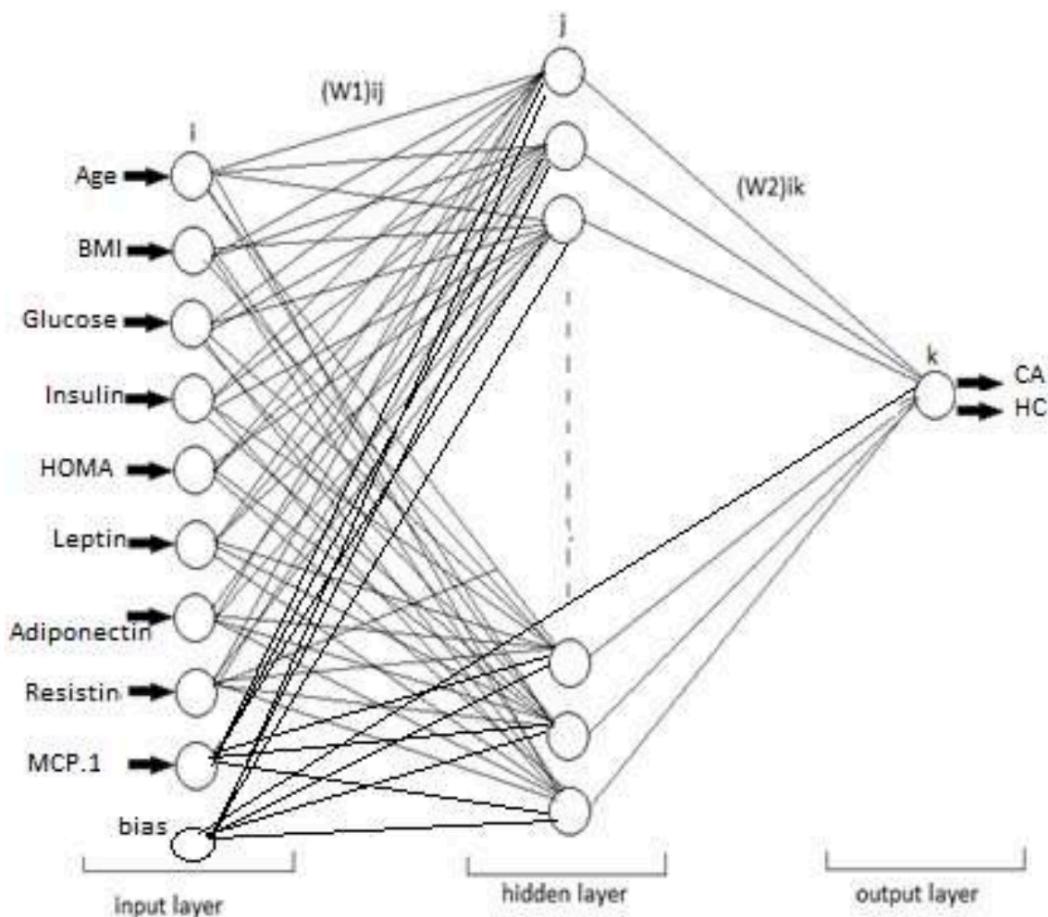
- افت ST با ورزش نسبت به استراحت القا شده oldpeak

- شیب بخش ST در حداکثر تمرینات

- تعداد رگ های عمدہ (۰-۳) رنگ آمیزی شده با فلوروسکوپی

## ۲.۵ تحلیل عملکرد الگوریتم‌های دسته‌بندی شبکه عصبی مصنوعی و بیز ساده برای دسته‌بندی داده‌ها

برای تحلیل بهتر این روش در الگوریتم طبقه بندی Bays مقاله‌ای تحت عنوان Performance Analysis of ANN and Naive Bayes Classification Algorithm for Data Classification مطالعه شد. این مقاله با استفاده از شبکه‌های عصبی مصنوعی و طبقه بندی بیز برای پیش‌بینی سرطان پستان با استفاده از داده‌های آنتروپومتریک و روتین عمکرد خون تحقیق شده است. این مقاله در سال ۲۰۱۹ در ژورنال International Journal of Intelligent Systems and Applications in Engineering منتشر شده است، داده‌هایی که در این مطالعه می‌بینیم شامل متغیرهایی چون سن، شاخص توده بدنی، سطوح گلوکز، انسولین، و پارامترهای خونی دیگر استفاده شده است که از داده‌های مجموعه داده‌های سرطان پستان کوئیمبرا بدست آمده‌اند. این مجموعه داده‌ها از مخزن یادگیری ماشین UCI فراهم شده‌اند که یک منبع عمومی برای تحقیقات علمی در زمینه یادگیری ماشین است. در بخش متودولوژی، این تحقیق از الگوریتم‌های موجود در نرم افزار Matlab استفاده می‌کند، که شامل ابزارهایی برای آموزش و آزمایش مدل‌های ANN و بیز ساده است. تنظیمات مختلف شبکه عصبی مانند تعداد لایه‌ها، تعداد نورون‌ها در هر لایه، نرخ یادگیری وتابع فعال‌سازی مورد بررسی و تنظیم قرار گرفته‌اند. در مقاله از یک مدل شامل ۹ ورودی و ۱ خروجی استفاده شده است و بر ارزیابی عملکرد طبقه‌بندی این الگوریتم‌ها تمرکز شده است. در قسمت نتیجه‌ای گیری که از این پژوهش انجام شده است نشان میدهد که هر دو الگوریتم به طور موثری در تشخیص سرطان پستان عمل کرده‌اند، با این حال ANN دقیق‌تر از بیز را نتیجه داده است. نحوه‌ی ساختار ANN را می‌توانید در شکل زیر مشاهده کنید:



شکل ۲۶: Model ANN



همانطور که در شکل جدول زیر مشاهده میشود پارامتر هایی که برای مدل شبکه عصبی ایجاد شده است را میتوان مشاهد کرد. در پارامتر ها از لونبرگ و مومنتوم استفاده شده که از آپتیمیزرهای قوی در سیستم های هوشمند هستند،

Parameters	Properties
Number of neurons in the input layer	10
Number of the hidden layers	1
Number of neurons in the hidden layer	10
Number of neurons in the output layer	1
Learning rate ( $\alpha$ )	0,2
Coefficient of momentum ( $\beta$ )	0,3
Learning algorithm	Levenberg-Marquardt (trainlm)
Transfer function	Logarithmic      sigmoid (logsig)

شکل ۲۷ : Parameters ANN

هدف این مقاله در واقع مقایسه‌ی شبکه عصبی دیپ لرنینگ با بیز ماشین لرنینگ است که همانطور که گفته شد و در مقاله ذکر شده ANN بهتر از بیز عمل کرده است.

### Report Classification ۳.۵

حال نگاهی به سایت سایکیت لرن می اندازیم و پارامتر هارا تحلیل میکنیم:

True Y ۱.۳.۵

این پارامتر نمایانگر مقادیر تارگت واقعی است. این ها برچسب های واقعی داده ها هستند که برای ارزیابی پیش‌بینی انجام شده توسط طبقه‌بند استفاده می‌شوند.

Pred Y ۲.۳.۵

این پارامتر شامل تارگت های تخمین زده شده توسط یک طبقه‌بند است. این برچسب پیش‌بینی شده توسط مدل طبقه‌بندی ما هستند که با برچسب های واقعی مقایسه می‌شوند تا عملکرد مدل ارزیابی شود



## Labels ۳.۳.۵

این پارامتر لیستی از شاخص‌های برچسب را برای گزارش مشخص می‌کند. معمولاً لیبل برای تارگتی است که قرار است پیش‌بینی شود.

## weight Sample ۴.۳.۵

این پارامتر وزن‌های نمونه‌ها را فراهم می‌کند. برای دادن وزن‌های متفاوت به نمونه‌های مختلف استفاده می‌شود.

## Digits ۵.۳.۵

این پارامتر تعداد اعشار برای مقادیر اعشاری را مشخص می‌کند. در فرمت بندی خروجی استفاده می‌شود، و با گرد کردن مقادیر اعشاری به تعداد مشخص شده از ارقام این کار را انجام میدهد.

## Dict Output ۶.۳.۵

این پارامتر به صورت بولین است. اگر روی True تنظیم شود، گزارش طبق بندی به جای قالب رشته‌ای معمول، به عنوان دیکشنری بازگردانده می‌شود. از این میتوان برای پردازش بیشتر نتایج به صورت برنامه‌ای به جای نشان دادن آنها استفاده کرد. بسته به شرایط میتوان از این پارامتر استفاده کرد.

## Division Zero ۷.۳.۵

این پارامتر تعیین می‌کند که چه اتفاقی می‌افتد زمانی که یک تقسیم صفر در طول محاسبه رخ می‌دهد (یعنی زمانی که هیچ پیش‌بینی مثبت یا نادرست درستی برای یک برچسب مشخص وجود ندارد).

- Warn: عدد صفر را بر می‌گرداند و وارنینگ میدهد
- 0: بدون وارنینگ ۰ را بر می‌گرداند.
- 1: یک را بر می‌گرداند
- np.nan: NAN را بر می‌گرداند

## Report ۸.۳.۵

- دقت: نسبت تعداد پیش‌بینی‌های صحیح مثبت به کل پیش‌بینی‌های مثبت را نشان میدهد.
- حساسیت: نسبت تعداد پیش‌بینی‌های صحیح مثبت به کل موارد مثبت واقعی در کلاس. حساسیت بالا نشان می‌دهد که کلاس به درستی و به طور جامعی شناسایی شده است.
- نمره‌ی F1: میانگین وزن دار دقت و حساسیت را نشان میدهد. این نمره هم خطاهای مثبت کاذب و هم خطاهای منفی کاذب را در نظر می‌گیرد.



## ۴.۵ وارد کردن کتابخانه ها و بارگذاری داده ها

ابتدا کتابخانه های مورد نیاز را وارد می کنیم:

```
1 from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
2 from sklearn.naive_bayes import GaussianNB
3 from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
4 from sklearn.model_selection import learning_curve, train_test_split
5 from sklearn.preprocessing import StandardScaler
6 import pandas as pd
7 import matplotlib.pyplot as plt
8 import seaborn as sns
9 import numpy as np
```

Code 9: ها کتابخانه کردن وارد

این کتابخانه ها برای پیش پردازش داده ها، مصورسازی و پیاده سازی مدل های یادگیری ماشین ضروری هستند و باید ایمپورت شوند.

## ۵.۵ بارگذاری و بررسی اولیه داده ها

مجموعه داده را با استفاده از تابع `pandas.read_csv` از بارگذاری می کنیم:

```
1 file_path = '/mnt/data/heart.csv'
2 heart_data = pd.read_csv(file_path)
```

Code 10: ها داده بارگذاری

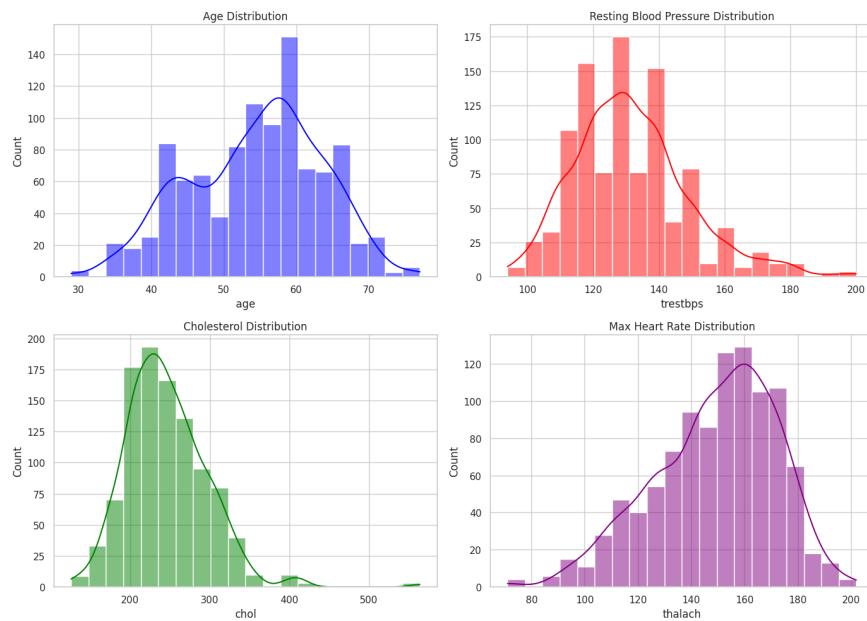
## ۶.۵ تحلیل داده های اکتشافی (EDA)

نمودا های هیستوگرام برای ویژگی های کلیدی هستند.

```
1 sns.set(style="whitegrid")
2
3 fig, axes = plt.subplots(2, 2, figsize=(14, 10))
4 sns.histplot(heart_data['age'], kde=True, ax=axes[0, 0], color="blue", bins=20)
5 axes[0, 0].set_title('Age Distribution')
6 sns.histplot(heart_data['trestbps'], kde=True, ax=axes[0, 1], color="red", bins=20)
7 axes[0, 1].set_title('Resting Blood Pressure Distribution')
8 sns.histplot(heart_data['chol'], kde=True, ax=axes[1, 0], color="green", bins=20)
9 axes[1, 0].set_title('Cholesterol Distribution')
10 sns.histplot(heart_data['thalach'], kde=True, ax=axes[1, 1], color="purple", bins=20)
11 axes[1, 1].set_title('Max Heart Rate Distribution')
12
13 plt.tight_layout()
```

```
14 plt.show()
```

Code 11: ها هیستوگرام



شکل ۲۸: هیستوگرام ویژگی های کلیدی

### ۱.۶.۵ تحلیل هیستوگرام ها

در این بخش، تحلیل هر یک از هیستوگرام ها آورده شده است:

#### ۲.۶.۵ توزیع سن (age)

هیستوگرام سن (age) نشان می دهد که بیشتر افراد در بازه‌ی سنی ۴۰ تا ۶۰ سال قرار دارند. توزیع سنی به سمت سنین بالا میل دارد که نشان می دهد داده ها بیشتر شامل افراد مسن تر است که بیشتر در معرض خطر بیماری قلبی هستند.

#### ۳.۶.۵ توزیع فشار خون در حالت استراحت

هیستوگرام فشار خون در حالت استراحت (trestbps) نشان می دهد که بیشتر افراد دارای فشار خون در محدوده ۱۲۰ تا ۱۴۰ میلی متر جیوه هستند. توزیع فشار خون به سمت مقادیر بالاتر میل دارد که ممکن است نشان دهنده وجود فشار خون بالا در بسیاری از افراد نمونه باشد. فشار خون بالا یک عامل خطر مهم برای بیماری های قلبی و عروقی است.



#### ۴.۶.۵ توزیع کلسترول (chol)

هیستوگرام کلسترول (chol) نشان می‌دهد که مقدار کلسترول بیشتر افراد در محدوده ۲۰۰ تا ۳۰۰ میلی‌گرم بر دسی لیتر قرار دارد. توزیع کلسترول نیز به سمت مقادیر بالاتر میل دارد که می‌تواند نشان دهنده وجود هایپرکلسترولمی در نمونه باشد. کلسترول بالا یکی از عوامل خطر اصلی برای بیماری قلبی است.

#### ۵.۶.۵ توزیع حداکثر ضربان قلب دستیابی (thalach)

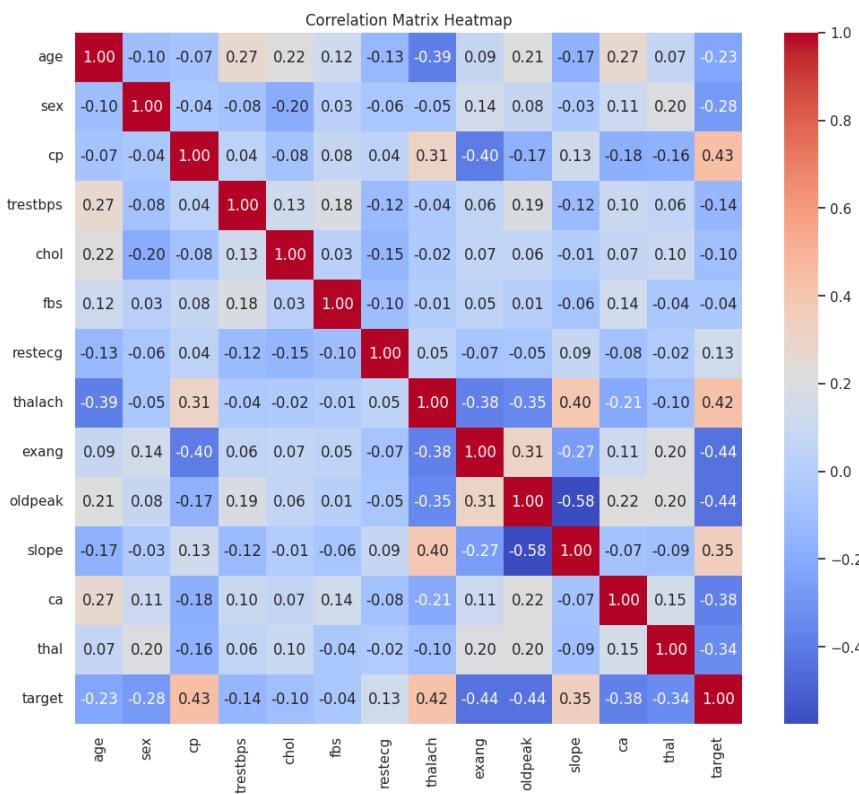
هیستوگرام حداکثر ضربان قلب دستیابی (thalach) نشان می‌دهد که بیشتر افراد دارای حداکثر ضربان قلب در محدوده ۱۴۰ تا ۱۷۰ ضربه در دقیقه هستند. توزیع ضربان قلب دستیابی به سمت مقادیر پایین تر میل دارد که ممکن است نشان دهنده کاهش ظرفیت فیزیکی در بسیاری از افراد نمونه باشد. حداکثر ضربان قلب دستیابی پایین می‌تواند نشان دهنده وجود بیماری قلبی یا کاهش ظرفیت فیزیکی باشد.

### ۷.۵ ماتریس همبستگی

ماتریس همبستگی برای درک روابط بین ویژگی‌ها محاسبه می‌شود:

```
1 correlation_matrix = heart_data.corr()
2 plt.figure(figsize=(12, 10))
3 sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
4 plt.title("Correlation Matrix Heatmap")
5 plt.show()
```

Code 12: همبستگی ماتریس



شکل ۲۹: نقشه حرارتی ماتریس همبستگی

## ۸.۵ تحلیل 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 میابند. 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 دستیابی (thalach): همبستگی منفی با age دارد و همبستگی مثبتی با cp دارد. این همبستگی ها نشان می دهد که حداکثر ضربان قلب دستیابی با افزایش سن کاهش می یابد و ممکن است با نوع درد قفسه سینه مرتبط باشد.

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



## ۹.۵ پیش‌پردازش داده‌ها

مقیاس‌بندی و بیزگی‌ها با استفاده از StandardScaler انجام می‌شود:

```
1 scaler = StandardScaler()  
2 scaled_features = scaler.fit_transform(heart_data.drop('target', axis=1))
```

Code 13: ها و بیزگی‌بندی مقیاس

## ۱۰.۵ تقسیم‌بندی مجموعه داده

مجموعه داده به مجموعه‌های آموزشی و آزمایشی با نسبت ۲۰-۸۰ تقسیم می‌شود:

```
1 X_train, X_test, y_train, y_test = train_test_split(scaled_features, heart_data['target'],  
2 test_size=0.2, random_state=4)
```

داده مجموعه‌بندی تقسیم: Code 14

## ۱۱.۵ آموزش و ارزیابی مدل‌ها

سه مدل مختلف: Gaussian Naive Bayes، Random Forest و Gradient Boosting، اولیه‌می شوند و آموزش دیده و ارزیابی می‌شوند:

```
1 models = {  
2     "GaussianNB": GaussianNB(),  
3     "RandomForest": RandomForestClassifier(random_state=4),  
4     "GradientBoosting": GradientBoostingClassifier(random_state=4)  
5 }  
6  
7 results = {}  
8 for model_name, model in models.items():  
9     model.fit(X_train, y_train)  
10    y_pred = model.predict(X_test)  
11    conf_matrix = confusion_matrix(y_test, y_pred)  
12    class_report = classification_report(y_test, y_pred)  
13    accuracy = accuracy_score(y_test, y_pred)  
14    results[model_name] = {  
15        "conf_matrix": conf_matrix,  
16        "class_report": class_report,  
17        "accuracy": accuracy  
18    }  
19  
20    print(f"Results for {model_name}:")  
21    print("Confusion Matrix:")  
22    print(conf_matrix)
```



```

23     print("\nClassification Report:")
24     print(class_report)
25     print(f"Accuracy: {accuracy}\n")

```

ها مدل ارزیابی و آموزش: Code 15:

## ۱۲.۵ ترسیم منحنی های یادگیری

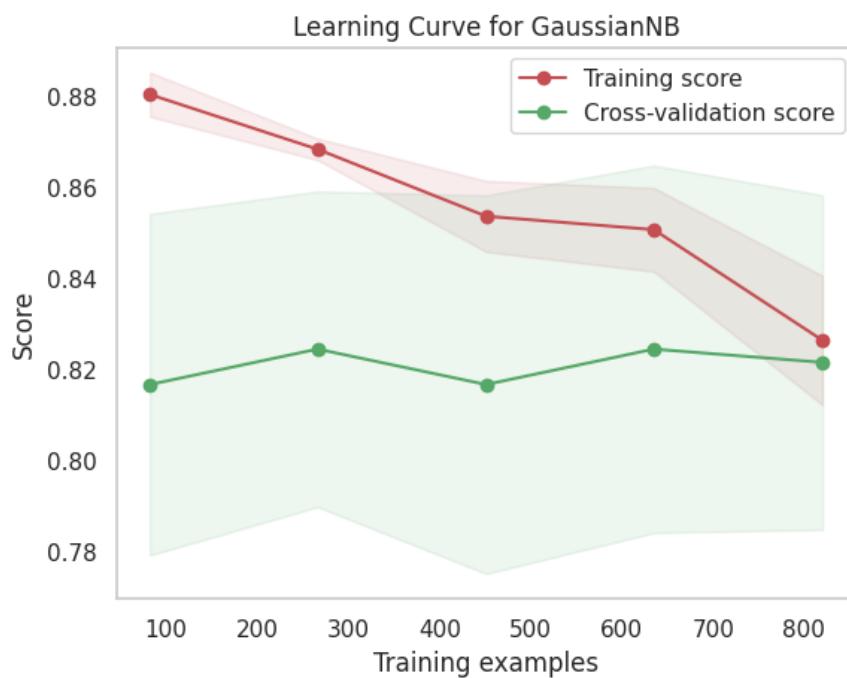
تابع plot\_learning\_curve برای ترسیم منحنی های یادگیری مدل ها تعریف می شود:

```

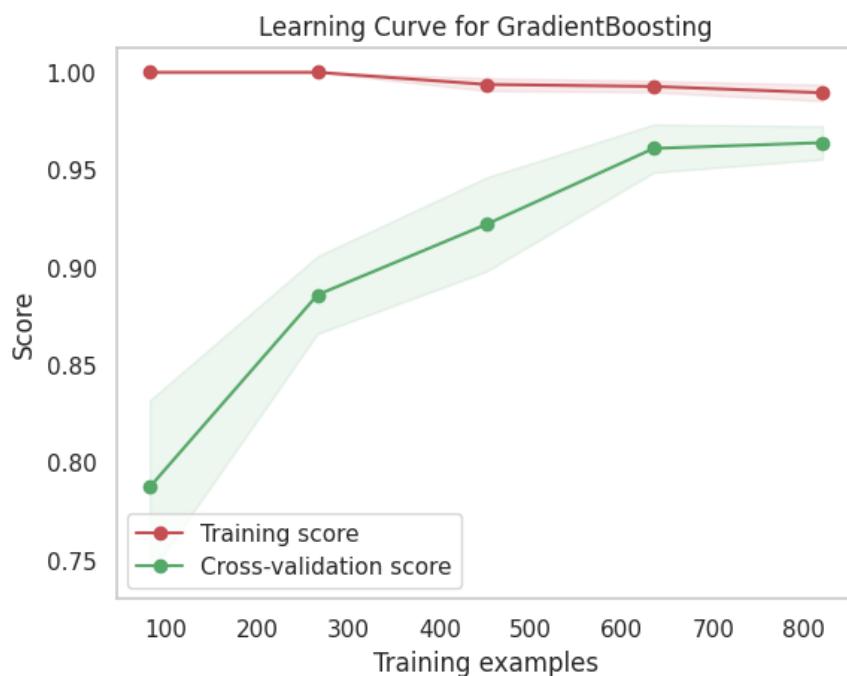
1 def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=None, train_sizes=np.linspace(.1,
2     1.0, 5)):
3     plt.figure()
4     plt.title(title)
5     plt.xlabel("Training examples")
6     plt.ylabel("Score")
7
8     train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs
9     , train_sizes=train_sizes)
10    train_scores_mean = np.mean(train_scores, axis=1)
11    train_scores_std = np.std(train_scores, axis=1)
12    test_scores_mean = np.mean(test_scores, axis=1)
13    test_scores_std = np.std(test_scores, axis=1)
14
15    plt.grid()
16
17    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
18                     train_scores_mean + train_scores_std, alpha=0.1,
19                     color="r")
20    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
21                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
22    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
23    plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")
24
25    plt.legend(loc="best")
26    return plt
27
28
29 for model_name, model in models.items():
30     plot_learning_curve(model, f"Learning Curve for {model_name}", scaled_features, heart_data['
31     target'], cv=5)
32     plt.show()

```

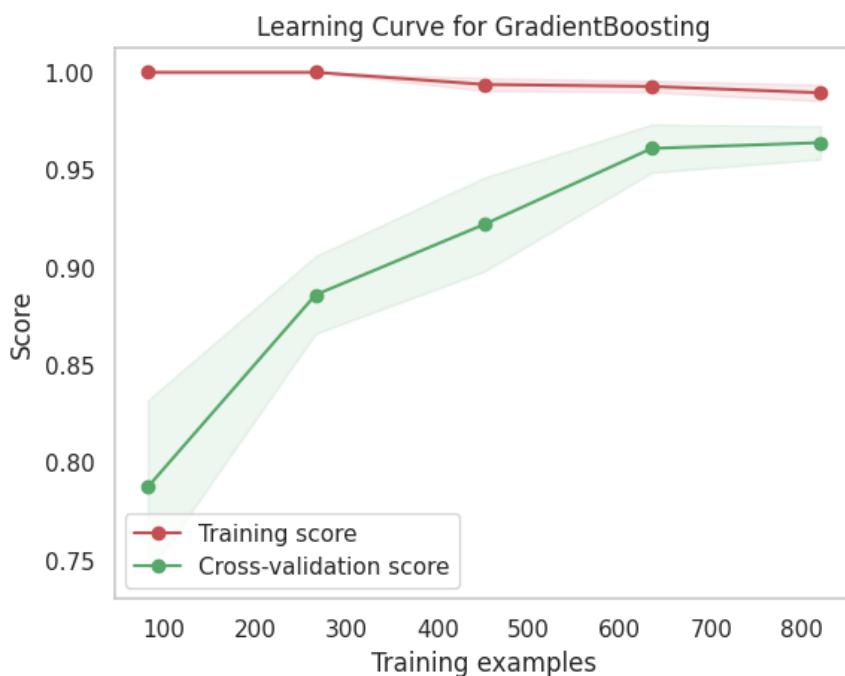
یادگیری های منحنی ترسیم: Code 16:



شکل ۳۰: منحنی یادگیری برای GaussianNB



شکل ۳۱: منحنی یادگیری برای RandomForest



شکل ۳۲: منحنی یادگیری برای GradientBoosting

همانطور که نشان دادیم میبینیم که در الگوریتم gradient boosting و random forest با over fitting مواجه شدیم و مشاهده میشود که الگوریتم GaussianNB بهتر از همه عملکرد است.

### ۱۳.۵ مقایسه نمونه ها

پنج نقطه داده تصادفی از مجموعه داده آزمایشی انتخاب می شود تا خروجی های واقعی و پیش‌بینی شده مقایسه شوند:

```

1 np.random.seed(4)
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 17: نمونه مقایسه

Predicted GradientBoosting	Predicted RandomForest	Predicted GaussianNB	Labels True
۱	۱	۱	۱
.	۱	.	.
.	.	.	.
.	.	.	.
.	۱	.	.

جدول ۲: مقایسه نمونه ها: برچسب های واقعی و پیش بینی شده

## ۱۴.۵ لینک گیت هاب

گیتھاب (GitHub).

## منابع

[scikit-learn](#) [1]

[2] Performance Analysis of ANN and Naive Bayes Classification Algorithm for Data Classification