

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

درس یادگیری ماشین

مینی پروژه دوم

نام و نام خانوادگی	علی مهرابی
شماره دانشجویی	۴۰۲۲۳۸۵۴
تاریخ	اردیبهشت ۱۴۰۳
استاد درس	دکتر علیاری

فهرست مطالب

۷	لینک های مربوطه	۱
۷	۱.۱ لینک مربوط به گیت هاب
۷	۲.۱ لینک مربوط به گوگل کلب
۸		سوال اول
۸	۱.۲ بخش اول
۹	۲.۲ بخش دوم
۱۰	۳.۲ بخش سوم
۱۶		سوال دوم
۱۶	۱.۳ بخش اول
۱۷	۲.۳ بخش دوم
۱۹	۳.۳ بخش سوم
۲۲	۴.۳ بخش چهارم
۲۵		سوال سوم
۲۵	۱.۴ بخش اول
۲۹	۲.۴ بخش دوم
۳۳	۳.۴ بخش سوم
۳۵		سوال چهارم
۳۵	۱.۵ کاوش و شناخت داده
۳۸	۲.۵ تقسیم داده، پیش پردازش و ساخت مدل
۳۹	۳.۵ بررسی و تحلیل نتایج



فهرست تصاویر

۸	فعال ساز ReLU و sigmoid	۱
۸	انواع ReLU	۲
۱۰	لایه اول شبکه عصبی	۳
۱۰	شبکه عصبی سوال ۱	۴
۱۵	خروجی شبکه عصبی سوال ۱	۵
۱۵	خروجی شبکه با فعال ساز sigmoid	۶
۱۶	خروجی شبکه با فعال ساز tanh	۷
۱۶	خروجی شبکه با فعال ساز relu	۸
۱۸	خروجی مدل اول	۹
۱۸	مقایسه لیبل واقعی و پیش بینی شده مدل اول	۱۰
۱۹	نتیجه طبقه بندی مدل اول	۱۱
۱۹	ماتریس در هم ریختگی مدل اول	۱۲
۲۰	خروجی مدل دوم	۱۳
۲۰	مقایسه لیبل واقعی و پیش بینی شده مدل دوم	۱۴
۲۰	نتیجه طبقه بندی مدل دوم	۱۵
۲۱	ماتریس در هم ریختگی مدل دوم	۱۶
۲۱	خروجی مدل سوم	۱۷
۲۲	مقایسه لیبل واقعی و پیش بینی شده مدل سوم	۱۸
۲۲	نتیجه طبقه بندی مدل سوم	۱۹
۲۲	ماتریس در هم ریختگی مدل سوم	۲۰
۲۳	توزيع کلاس در داده های سوال دوم	۲۱
۲۳	مدل ساخته شده به کمک MLPClassifier	۲۲
۲۴	انجام Cross validation	۲۳
۲۴	کد رسم learning-curve	۲۴
۲۴	انجام CV و رسم منحنی یادگیری	۲۵
۲۵	جدا کردن X و y	۲۶
۲۵	استفاده از Train-Test split	۲۷
۲۵	استفاده از Stratified Train-Test split	۲۸
۲۵	استفاده از CV	۲۹
۲۶	استفاده از Leave one out	۳۰
۲۶	جدا کردن داده ها به صورت دستی	۳۱
۲۶	توزيع برچسب ها در داده	۳۲
۲۷	تقسیم داده ها به مجموعه آموزش و تست	۳۳
۲۷	نوع داده ها	۳۴



۲۷	Label Encoder	۳۵
۲۸	درخت تصمیم ۱	۳۶
۲۸	برچسب واقعی و پیش بینی شده درخت تصمیم اول	۳۷
۲۸	نتایج طبقه بندی درخت تصمیم اول	۳۸
۲۸	ماتریس در هم ریختگی درخت تصمیم اول	۳۹
۲۹	تصویرسازی درخت تصمیم اول	۴۰
۲۹	تحلیل متنی درخت تصمیم اول	۴۱
۲۹	درخت تصمیم دوم	۴۲
۳۰	نتایج طبقه بندی درخت تصمیم دوم	۴۳
۳۱	ماتریس در هم ریختگی درخت تصمیم دوم	۴۴
۳۱	تصویرسازی درخت تصمیم دوم	۴۵
۳۱	تحلیل متنی درخت تصمیم دوم	۴۶
۳۲	درخت تصمیم سوم	۴۷
۳۲	نتایج طبقه بندی درخت تصمیم سوم	۴۸
۳۲	تصویرسازی درخت تصمیم سوم	۴۹
۳۳	درخت تصمیم چهارم	۵۰
۳۳	نتایج درخت تصمیم چهارم	۵۱
۳۳	تصویرسازی درخت تصمیم چهارم	۵۲
۳۴	مدل جنگل تصادفی	۵۳
۳۴	مدل AdaBoost	۵۴
۳۵	داده مربوط به بیماری قلبی	۵۵
۳۵	نوع داده مربوط به بیماری قلبی	۵۶
۳۶	داده های null	۵۷
۳۶	توصیف داده	۵۸
۳۶	ماتریس هم بستگی	۵۹
۳۷	میزان هم بستگی ویژگی ها با متغیر وابسته	۶۰
۳۷	توزیع داده	۶۱
۳۷	توزیع جنسیت در داده	۶۲
۳۷	نحوه توزیع جنسیت، فشار خون، کلسترول و حداقل ضربان در داده	۶۳
۳۸	توزیع نقطه ای داده	۶۴
۳۸	تقسیم داده ها به آموزش و آزمون	۶۵
۳۹	کردن داده ها scale	۶۶
۳۹	مدل GaussianNB در سایکیت لرن	۶۷
۳۹	مدل NaiveBayes ساخته شده در کلاس	۶۸
۳۹	نتایج طبقه بندی	۶۹
۴۰	ماتریس در هم ریختگی	۷۰



فهرست جداول



فهرست برنامه‌ها

۱۱	neurons the building	۱
۱۲	network the of output	۲
۱۳	functions activision different with McCulloch-Pitts	۳
۱۴	functions activision different on based output the plotting	۴
۱۷	split Test Validation, Train.	۵
۱۸	split Test Validation, Train.	۶
۱۹	function) activation different (with ۲ model MLP	۷
۲۱	model) (best ۳ model MLP	۸



۱ لینک های مربوطه

۱.۱ لینک مربوط به گیت هاب

از این لینک گیت هاب ([Github](#)) می توانید برای دسترسی به صفحه Github مربوط به این پروژه استفاده کنید.

۲.۱ لینک مربوط به گوگل کلب

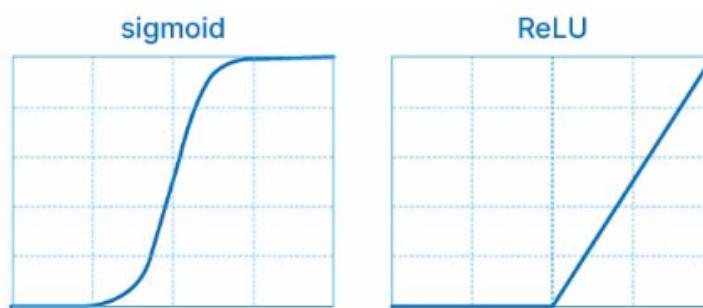
از این لینک گوگل کلب ([Google Colab](#)) می توانید برای دسترسی به notebook نوشته شده دسترسی پیدا کنید.



۲ سوال اول

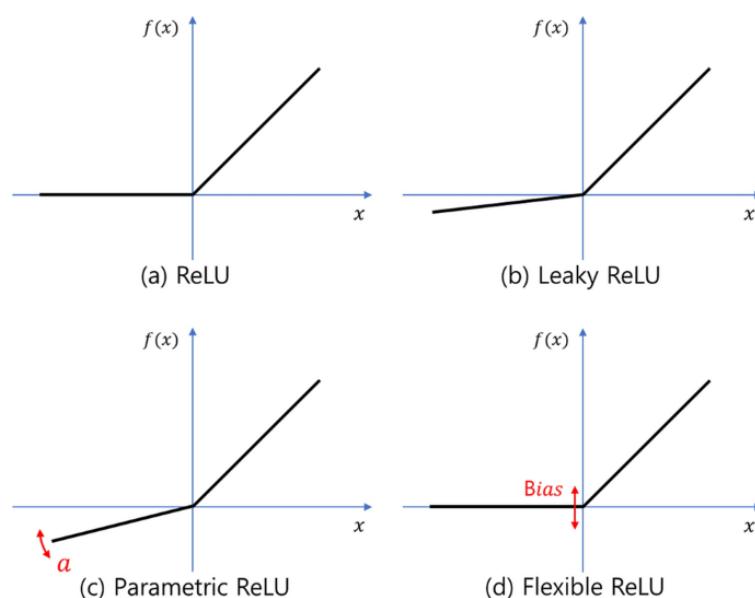
۱.۲ بخش اول

این مسئله را به این صورت درنظر می‌گیریم که تابع فعال ساز (activation function) در لایه یکی مانده به آخر ReLU و در لایه آخر sigmoid است. ابتدا نگاهی به شکل این دو تابع فعال ساز می‌اندازیم که در [شکل ۱](#) آمده است.



شکل ۱: فعال ساز sigmoid 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 شرکتی ندارد (dying ReLU). در شبکه‌های عمیق تر برای بهبود نتایج از گزینه‌های دیگر مانند ReLU و یا Parametric ReLU که در [شکل ۲](#) آمده است.



شکل ۲: انواع ReLU

فعال ساز sigmoid اما یک خروجی بین صفر و یک به ما می دهد که در کار طبقه بنده دو کلاسه بسیار مطلوب است و معمولاً در این نوع طبقه بنده ها به عنوان فعال سازی لایه آخر به کار می رود تا بتواند به ما یک نگاشت به فضای احتمالاتی بدهد. علاوه بر این به ما یک گرادیان smooth می دهد که برای بروز کردن وزن ها کمک می کند و در فرآیند یادگیری ایده آل است. مشکلی که در این تابع فعال ساز با آن روبرو هستیم مسئله ناپدید شدن گرادیان است که باعث می شود عمل بروز شدن وزن ها همراه با تداخل انجام بگیرد. البته وقتی از این تابع در لایه آخر استفاده می شود این مسئله کمتر به چشم می آید. نکته دیگر درباره این فعال ساز منطقه اشباعی است که در آن قرار می گیرد. یعنی در این حالت ورودی های نورون ممکن است رنج زیادی داشته باشد ولی خروجی نورون تفاوت زیادی نمی کند و ممکن است در فرآیند یادگیری موثر نباشد.

پس در نهایت باید این مورد را درنظر داشته باشیم که خروجی نورون ها با تابع فعال ساز ReLU منفی نشود و همانطور موضوع اشباع شدن فعال ساز sigmoid را درنظر بگیریم.

۲.۲ بخش دوم

رابطه ۱ تابع فعال ساز ReLU را نشان می دهد

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

همانطور که دیده می شود گرادیان این تابع برای مقادیر مثبت ۱ (بیانگر رفتار خطی) و برای مقادیر منفی صفر است.

رابطه ۲ گرادیان ELU را نشان می دهد. (برای بدست آوردن این رابطه باید از تابع داده شده نسبت به x مشتق گرفته شود)

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

همانطور که از **رابطه ۲** مشخص است، گرادیان این فعال ساز در مقادیر مثبت مانند ReLU برابر ۱ است که بیانگر رفتار خطی این تابع می باشد. در مقادیر منفی گرادیان برابر با αe^x است و دیده می شود که با منفی تر شدن x به صفر میل می کند.

از برتری های این تابع نسبت به ReLU می توان به موارد زیر اشاره کرد:

- گرادیان غیر صفر در مقادیر منفی به کاهش مشکل dying ReLU کمک می کند زیرا به گرادیان اجازه می دهد حتی برای ورودی های منفی به عقب برگردد (برخلاف ReLU).

- به دلیل گرادیان غیر صفر برای ورودی های منفی، شبکه هایی که از ELU استفاده می کنند سریع تر هم گرا می شوند چون در فرایند backpropagation بهتر شرکت می کنند.

- چون همه مقادیر گرادیان این تابع غیر صفر است در الگوریتم گرادیان کاهشی تمامی وزن ها امکان بروز رسانی دارند.



۳.۲ بخش سوم

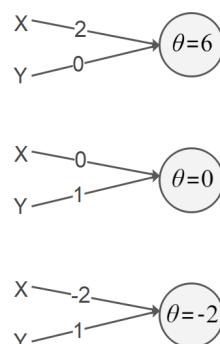
در ابتدا نیاز است شبکه مورد نیاز برای انجام کار طبقه بنده را طراحی کنیم. چون سه شرط در این سوال (شامل سه ضلع مثلث) مطرح است، از ۳ نورون McCulloch-Pitts در لایه اول استفاده می کنیم و از یک نورون نهایی برای عمل رای گیری استفاده می شود. همانطور که گفته شد ابتدا نیاز است تا وزن ها و Threshold سه نورون اول را معلوم کنیم. با بدست آوردن معادلات خط AB, BC, AC به ترتیب داریم:

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

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

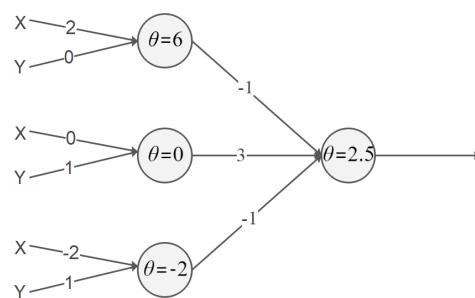
$$-2x + y = -2 \quad (5)$$

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 وزن های 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 آتش کند. به طوری که باید برای ورودی های صفر از لایه قبل پنالتی بیشتری اختصاص دهیم. درنهایت شبکه به همراه Threshold شکل ۴ خواهد بود.



شکل ۴: شبکه عصبی سوال ۱



حال اگر بخواهیم شبکه را به کمک کلاس مدل سازی کنیم به صورت زیر چهار نورون را تعریف می کنیم.

```
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 می کنیم و آن را عنوان ورودی به شبکه می دهیم. باید نقاطی که شبکه به عنوان کلاس ۱ درنظر می گیرد سبز و بقیه نقاط (به عنوان کلاس صفر) قرمز باشد.

```

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

```

Code 2: output of the network

خروجی مدل به صورت [شکل ۵](#) است. همانطور که دیده می شود، به خوبی کار طبقه بندی انجام شده و نقاط داخل مثلث (به مرز نواحی دقت شود) همگی با رنگ سبز و لیبل ۱ مشخص شده اند و نقاط خارج آن با رنگ قرمز و لیبل صفر هستند و نتیجتاً شبکه به خوبی در جدا کردن داده ها عمل می کند.

در قسمت بعد اثر activision function های مختلف را در مدل می بینیم. برای انجام اینکار یک ورودی به ورودی های کلاس McCulloch-Pitts اضافه می کنیم.

همانطور که در بالا دیده می شود، یک ورودی activison به کلاس اضافه شده که در حالتی که به طور خاص تعریف نشود در حالت threshold قرار دارد. همانطور که دیده می شود این توابع در قالب سه متود (method) تعریف شده اند و در نهایت با یک دستور if که با توجه به ورودی activison شرط بندی می کند، به ما خروجی می دهد. در نهایت یک تابع classify and plot تولید شده که تنها ورودی آن نوع تابع فعال ساز است و با توجه به این تابع برای ما کار رسم را انجام می دهد.

نتایج در [شکل ۶](#)، [شکل ۷](#) و [شکل ۸](#) آمده است.

دیده می شود که با تغییر توابع فعال ساز عملکرد شبکه به طور کلی تحت الشعاع قرار می گیرد و شبکه باید با درنظرگیری این توابع



```

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

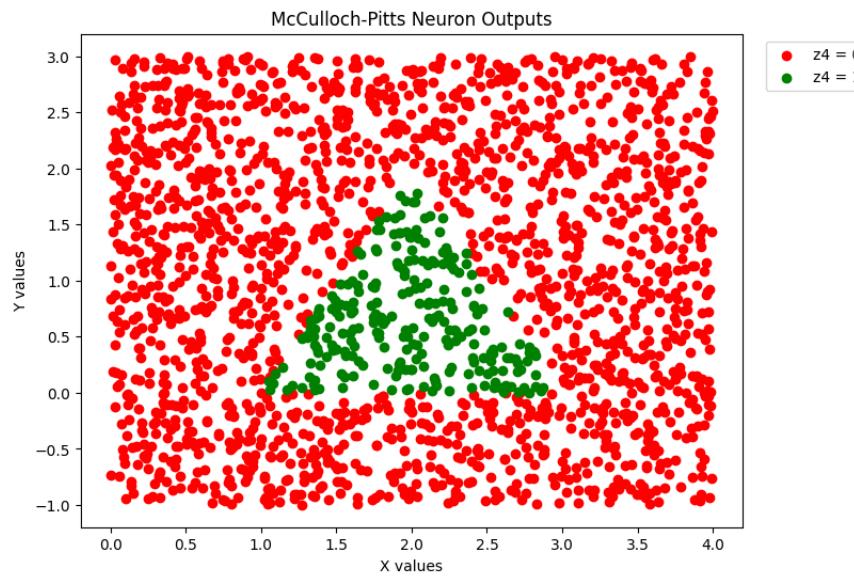


```

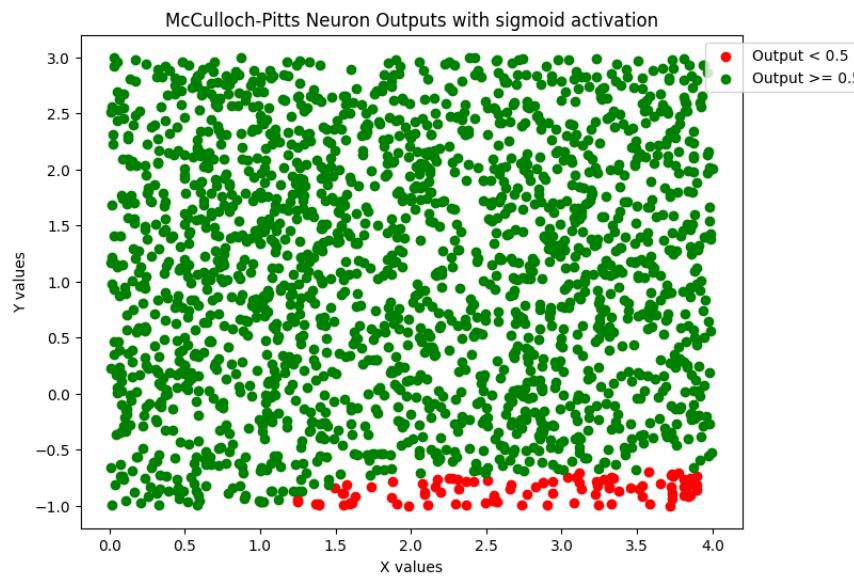
1 def classify_and_plot(activation):
2     num_points = 2000
3     x_values = np.random.uniform(0, 4, num_points)
4     y_values = np.random.uniform(-1, 3, num_points)
5
6     red_points = []
7     green_points = []
8
9     for i in range(num_points):
10         z4_value = Area(x_values[i], y_values[i], 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 activision functions

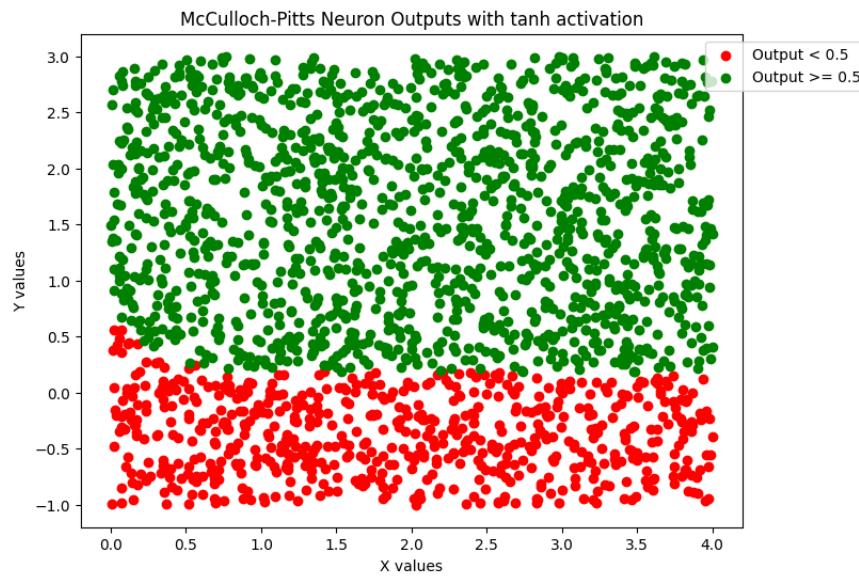


شکل ۵: خروجی شبکه عصبی سوال ۱

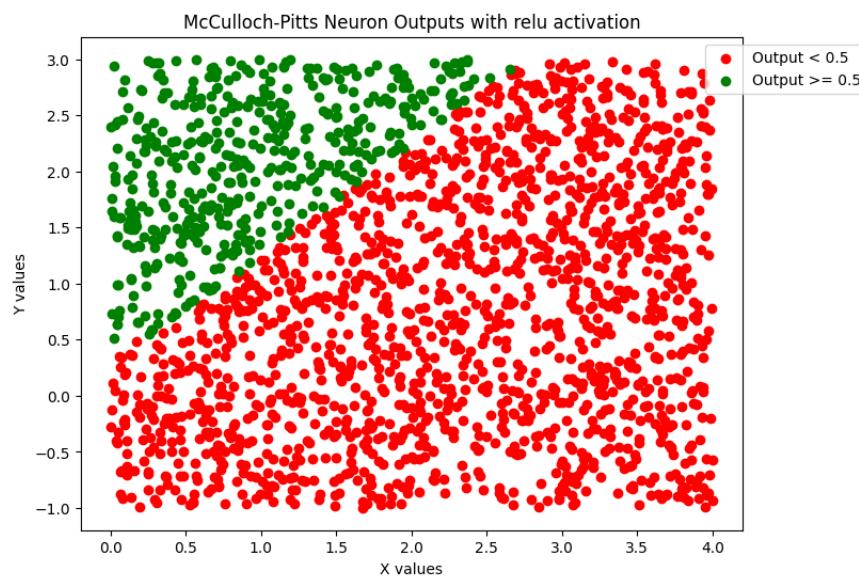


شکل ۶: خروجی شبکه با فعال ساز sigmoid

فعال ساز طراحی شود. در این حالت چون با threshold این شبکه طراحی شده بود، همین فعال سازی نیز بهترین گزینه است و در حالت های دیگر طبقه بند به خوبی کار نمی کند و تنها بخشی از کار طبقه بندی را انجام می دهد.



شکل ۷: خروجی شبکه با فعال ساز tanh



شکل ۸: خروجی شبکه با فعال ساز relu

سوال دوم ۳

بخش اول ۱.۳

همانطور که در گزارش مینی پروژه ۱ آورده شد، این دیتاست شامل ۳ نوع عیب درباره بلبرینگ است. به طور کلی بلرینگ ها سه بخش دارند که شامل کنس داخلی (Inner race)، ساقمه (Ball) و کنس خارجی (Outer race) است. کنس خارجی و داخلی وظیفه نگهداری ساقمه ها بین خود را دارند.



سه دیتاستی که ما در اختیار داریم مربوط به این سه بخش بلبرینگ با مشخصات معلوم است. برای مثال در مورد خاص این گزارش $X = 2$ مربوط به خطای با قطر ۰.۰۰۷ اینچ، بار ۲ اسب بخار و سرعت 1750 rpm است. موقعیت کنس خارجی نسبت به منطقه بار در ساعت ۶ (مرکزی) است. همه خطاهای با فرکانس نمونه برداری ۱۲ کیلوهرتز نمونه برداری شده اند.

در ادامه داده ها را مانند مینی پروژه قبلی می خوانیم و بعد از استخراج ویژگی ها به آن ها برچسب می زنیم. دقت شود که در این مورد با یک طبقه بندی چند کلاسه مواجه هستیم چون سه کلاس خطای داریم و یک کلاس نرمال پس در کل داده ۴ کلاس است. ما از هر کلاس ۴۰۰ نمونه ۳۰۰ تایی بر می داریم. یعنی $400 \times 300 = 120,000$ batch لیبل داده های نرمال است، خطاهای مربوط به کنس داخلی ۱، مربوط به کنس خارجی ۲ و مربوط به توب پ ۳ است.

در نهایت برای 800×800 سطر ویژگی داریم. پس ویژگی های مستقل ما ابعادی برابر با 8×800 دارد و برچسب یک بردar ستونی 800×1 عضوی است. توجه شود که داده ها باید shuffle و scale بشوند که اینکار به کمک StandardScaler انجام می شود.

در قسمت بعدی داده ها را به سه بخش Train، Validation و Test تقسیم می کنیم. خوبی تقسیم داده به سه قسمت به جای دو قسمت این است که عمومیت مدل نسبت به داده های جدید و داده هایی که ندیده است (unseen data) افزایش می یابد. در این حالت مدل به کمک داده های آموزش، آموزش می بیند و اصطلاحا Train می شود و پارامتر های مدل تنظیم می شود. دیتای Validation برای تنظیم بهتر هایپرپارامتر ها مانند نرخ یادگیری، تعداد لایه ها و تعداد نورون ها در هر لایه و به طور کلی مانیتور کردن فرآیند یادگیری مدل به کار می رود. دیتای Test برای ارزیابی فرآیند یادگیری پس از اتمام آن به کار می رود به طوری که این ارزیابی بایاس نیست و مشخص می کند مدل چقدر عمومیت دارد. در نهایت از علل تقسیم دیتا به سه بخش به جای دو بخش می توان به موارد زیر اشاره کرد:

۱- دور کردن مدل از overfit شدن. با اضافه کردن Validation و به کمک منحنی یادگیری برای دیتای Train و Validation و مقایسه آن ها.

۲- مشخص کردن مدل یادگیری ماشین و همینطور انتخاب هایپرپارامتر ها به طوری که متريک مورد نظر بهبود پیدا کند.
اینکار به صورت [برنامه ۵](#) انجام شده است.

```

1 from sklearn.model_selection import train_test_split
2 X_train, X_temp, y_train, y_temp = train_test_split(X, y, train_size=0.6, random_state=54,
   ↪ shuffle=True)
3 X_valid, X_test, y_valid, y_test = train_test_split(X_temp,y_temp, test_size=0.5, random_state
   ↪ =54, shuffle=True)

```

Code 5: Train, Validation, Test split

۲.۳ بخش دوم

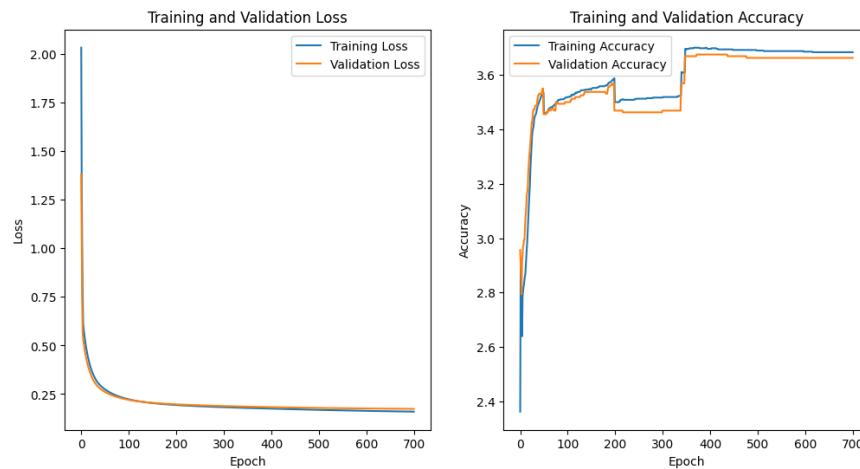
با تعریف توابع فعال ساز و خطای، کلاس مربوط به MLP را به کمک کد های قرار گرفته شده تشکیل می دهیم. برای مدل اول، که با MLPnet1 نام گذاری شده، یک شبکه دولایه با ۴ و ۵ نورون در هر لایه و چهار نورون در لایهنهایی طراحی می کنیم که تابع فعال ساز نورون های پنهان ReLU و تابع فعال ساز نهایی sigmoid است. علاوه بر این تعداد iteration ها ۷۰۰ تنظیم شده و تابع خطای هم Irbce در نظر گرفته شده. نرخ یادگیری نیز $0.5 \times 0.5 = 0.25$ است. توجه شود که چون طبقه بندی multiclass داریم، نیاز است تا برچسب هارا به صورت encoding hot Irone تبدیل کنیم. این مدل در [برنامه ۶](#) آمده است.

نمودار اتلاف و دقت برای Train و Validation در [شکل ۹](#) آمده است. از شکل ۹ مشخص است که فرایند آموزش به خوبی جلو می روید یعنی هم مدل خوب پارامتر هارا یاد می گیرد و هم خطای مربوط به Validation رو به کاهش است. در نتیجه مدل overfit نشده



```
1 mlp_net1 = MLP(hidden_layer_sizes=[4,5], hidden_activation='relu', output_size=4,
2   ↳ output_activation='sigmoid', n_iter=700, loss_fn=bce, eta=0.05, random_state=54)
```

Code 6: Train, Validation, Test split



شکل ۹: خروجی مدل اول

است و دائماً رو به بهبودی است. همانطور که مشخص است از Epoch شماره ۶۰۰ به بعد روند یادگیری خیلی کم می شود. و دقت مدل ثابت است.

علاوه بر این دیگر نتایج مربوط به این مدل در شکل ۱۰ و شکل ۱۱ آمده است. شکل ۱۰ لیل های واقعی و لیل های پیش بینی شده

```
▶ y_hat = mlp_net1.predict(X_test_sc)
y_hat = np.argmax(y_hat, axis=1)
test_labels_onehot = np.argmax(test_labels_onehot, axis=1)
print(f"actual labels: \n {test_labels_onehot} \n predicted labels:\n {y_hat}")

actual labels:
[2 3 3 2 0 2 0 2 1 0 0 1 2 2 2 0 2 3 3 3 1 0 2 0 2 0 2 2 2 3 3 2 3 3 0 3 0
1 3 1 3 0 1 0 3 3 3 3 0 3 1 1 2 0 2 1 3 1 1 2 3 3 1 0 1 3 0 0 1 2 0 3 0
2 1 1 2 1 0 3 2 1 2 1 2 2 1 0 3 2 1 2 0 0 0 1 1 0 3 0 3 1 3 3 0 1 2 2 0 3
1 0 0 2 3 2 0 2 0 1 2 2 2 2 3 3 0 3 2 1 3 2 2 3 2 0 0 0 3 3 2 2 2
1 2 2 1 1 2 0 1 3 3 2 2]
predicted labels:
[2 3 3 2 0 1 0 2 2 0 0 1 2 2 2 0 2 3 3 3 1 0 2 0 2 0 2 2 2 3 3 2 3 3 0 3 0
2 3 2 3 0 2 0 3 3 3 3 0 3 2 1 2 0 2 2 3 2 2 3 3 1 0 2 3 0 0 2 2 0 3 0
2 1 2 2 2 0 3 2 2 2 2 2 1 0 3 2 1 2 0 0 0 2 1 0 3 0 3 2 3 3 0 2 2 2 0 3
2 0 0 2 3 2 0 2 0 1 2 2 2 2 2 3 3 0 3 2 2 3 2 2 3 2 0 0 0 3 3 2 2 2
1 2 2 2 1 2 0 2 3 3 2 2]
```

شکل ۱۰: مقایسه لیل واقعی و پیش بینی شده مدل اول

توسط مدل را نشان می دهد. شکل ۱۱ نتایج مربوط به طبقه بندی را به نمایش می گذارد. همانطور که از شکل ۱۱ دیده می شود، مدل ۱، برای کلاس اول که همان خطاهای مربوط به کنس داخلی بود نسبت به بقیه کلاس ها عملکرد ضعیفی دارد. در کلی مدل accuracy ۸۶ درصد است که جای بهتر شدن دارد و در مدل های بعدی این موضوع بررسی می شود. در کلاس نرمال و خطای مربوط به توب مدل عملکرد بسیار خوبی دارد و مجموع precision و recall که می توان آن را از F1-score دنبال کرد بسیار خوب است. همانطور که گفته شد در کلاس ۱ و ۲ مدل جای بهتر شدن را دارد و این موضوع در ماتریس در هم ریختگی هم بهتر مشخص خواهد شد در قسمت support



```

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
model_report = classification_report(test_labels_onehot, y_hat)
print(model_report)

      precision    recall   f1-score   support
0         1.00     1.00     1.00      37
1         0.92     0.33     0.49      33
2         0.69     0.98     0.81      49
3         1.00     1.00     1.00      41

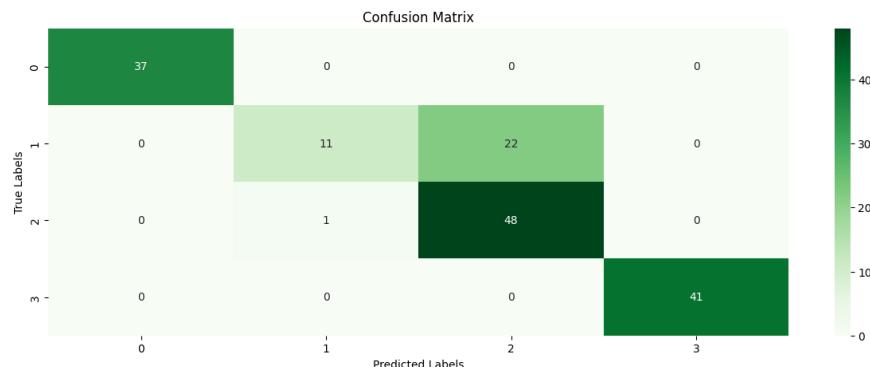
accuracy                           0.86      160
macro avg       0.90     0.83     0.82      160
weighted avg    0.89     0.86     0.84      160

```

شکل ۱۱: نتیجه طبقه بندی مدل اول

می توان تعداد داده های تست برای هر کلاس را مشاهده کرد.

در شکل ۱۲ ماتریس در هم ریختگی مربوط به مدل ۱ نشان داده شده. همانطور که از این ماتریس دیده می شود یک مشکل بزرگ برای ما وجود دارد. دیده می شود که ۲۲ داده مربوط به کلاس ۱ وجود دارد که مدل ما آن را مربوط به کلاس ۲ می داند غیر از این موضوع ۱ داده از کلاس ۲ است که مدل ما آن را در کلاس ۱ طبقه بندی کرده است و سایر داده ها درست طبقه بندی شده اند به طور کلی در این ماتریس باید سعی شود تمام داده هاروی قطر اصلی تجمع پیدا کنند.



شکل ۱۲: ماتریس در هم ریختگی مدل اول

۳.۳ بخش سوم

در این بخش دو مدل دیگر ارائه می شود که تفاوت هایی با مدل ۱ دارند. ابتدا مدل ۲ که در برنامه ۷ آمده است:

```

mlp_net2 = MLP(hidden_layer_sizes=[4,5], hidden_activation='sigmoid', output_size=4,
                output_activation='sigmoid', n_iter=700, loss_fn=mse, eta=0.05, random_state=54)

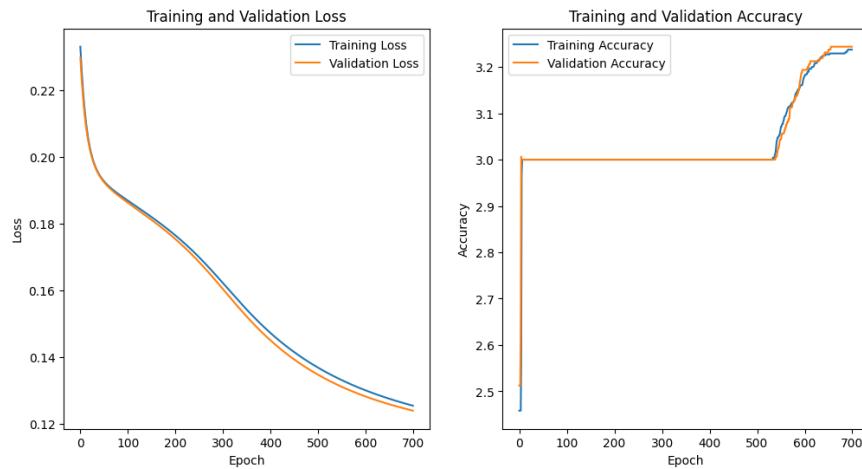
```

Code 7: MLP model 2 (with different activation function)

در برنامه ۷ دیده می شود که برای لایه های میانی به جای Relu از sigmoid استفاده شده است و علاوه بر این تابع خطا هم از bce به



تغییر یافته که عموما برای کارهای regression به کار برده می شود در [شکل ۱۳](#) نمودار اتلاف و accuracy مدل ۲ دیده می شود. می توان در مقایسه با [شکل ۹](#) تفاوت تغییر تابع فعال ساز را مشاهده کرد.



شکل ۱۳: خروجی مدل دوم

تغییر محسوسی در روند کاهش خطأ و افزایش دقت دیده می شود. می توان دید که مقدار خطای داده validation به همراه داده آموزش کم می شود و الگوی نامنظمی دیده می شود. به نظر می رسد اگر تعداد epoch ها بیشتر بود می توانستیم به دقت بیشتری هم برسیم. علاوه بر این [شکل ۱۴](#) مقدار واقعی و پیش بینی شده مدل و [شکل ۱۵](#) نتایج طبقه بندی را نشان می دهد. از [شکل ۱۵](#) دیده می شود که دقت مدل

```
y_hat = mlp_net2.predict(X_test_sc)
y_hat = np.argmax(y_hat, axis=1)
test_labels_onehot = np.argmax(test_labels_onehot, axis=1)
print(f"actual labels: \n {test_labels_onehot} \n predicted labels:\n {y_hat}")

actual labels:
[3 3 3 2 0 2 1 0 0 1 2 2 2 0 2 3 3 3 1 0 2 0 2 0 2 2 2 3 3 2 3 3 0 3 0
1 3 1 3 0 1 0 3 3 3 0 3 1 1 2 0 2 1 3 1 1 2 3 3 1 0 1 3 0 0 1 2 0 3 0
2 1 1 2 1 0 3 2 1 2 1 2 2 1 0 3 2 1 2 0 0 0 1 1 0 3 0 3 1 3 3 0 1 2 2 0 3
1 0 0 2 3 2 0 2 0 1 2 2 2 2 2 3 3 0 3 2 1 3 2 2 3 2 0 0 0 3 3 2 2 2 2
1 2 1 1 2 0 1 3 3 2 2]
predicted labels:
[2 0 0 2 0 2 0 2 1 0 0 1 2 2 2 0 2 0 0 0 1 0 2 0 2 0 2 2 2 3 3 2 0 3 0 0
1 0 1 0 0 1 0 0 0 3 0 0 0 0 1 1 2 0 2 1 3 1 1 2 0 0 1 0 1 0 0 0 1 2 0 0 0
2 1 1 2 1 0 3 2 1 2 1 2 2 1 0 0 2 1 2 0 0 0 1 1 0 0 0 0 1 0 0 0 1 2 2 0 0
1 0 0 2 0 2 0 2 0 1 2 2 2 2 2 3 0 0 0 2 2 0 2 0 2 0 0 0 0 0 2 2 2 2
1 2 2 2 1 2 0 2 0 0 2 2]
```

شکل ۱۴: مقایسه لیبل واقعی و پیش بینی شده مدل دوم

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
model_report = classification_report(test_labels_onehot, y_hat)
print(model_report)

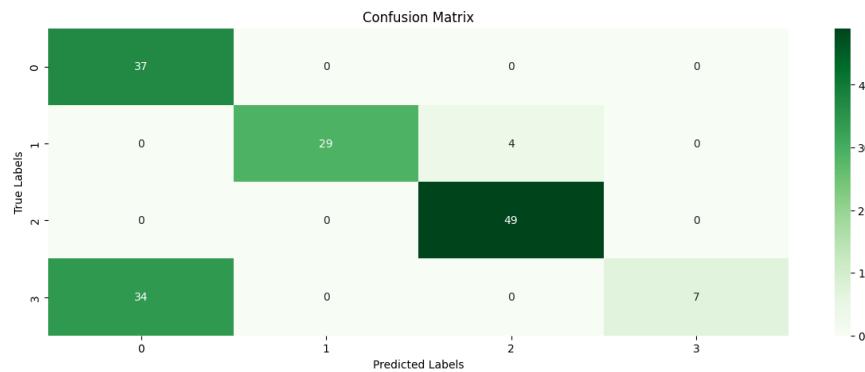
precision    recall   f1-score   support
          0       0.52      1.00      0.69      37
          1       1.00      0.88      0.94      33
          2       0.92      1.00      0.96      49
          3       1.00      0.17      0.29      41

accuracy         0.86      0.76      0.72      160
macro avg       0.86      0.76      0.72      160
weighted avg    0.87      0.76      0.72      160
```

شکل ۱۵: نتیجه طبقه بندی مدل دوم



کاهش یافته و تابع فعال ساز ReLU برایمان عملکرد بهتری را نتیجه می‌داد. علاوه بر این وضعیت مدل در کلاس چهارم بسیار بدتر شده است. **شکل ۱۶** نشان می‌دهد که ۳۴ مورد مربوط به کلاس ۳ بودند را مدل ما در کلاس صفر قرار داده و اصلاً وضعیت جالبی ندارد.



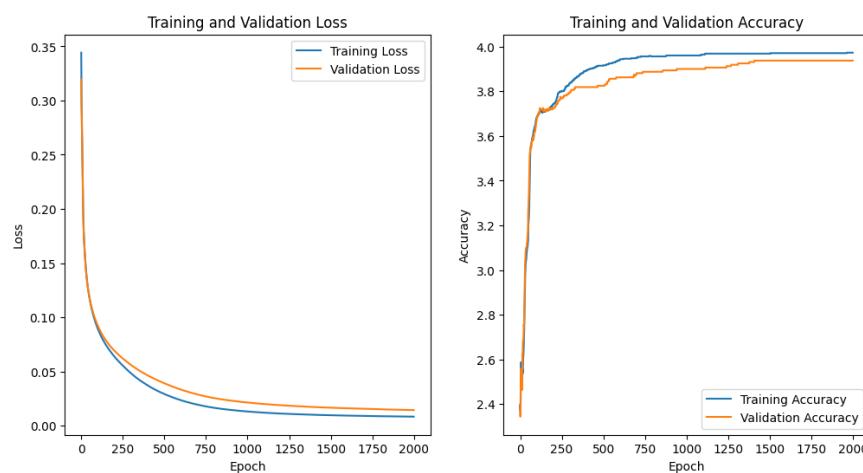
شکل ۱۶: ماتریس در هم ریختگی مدل دوم

علاوه بر این ۴ مورد مربوط به کلاس ۱ داشتیم که مدل آن را در کلاس ۲ طبقه بندی کرده است. در نهایت یک مدل می‌سازیم که از مدل ۱ و ۲ بهتر باشد و بهترین عملکرد را رایه بدهد. این مدل در [برنامه ۸](#) آمده است:

```
mlp_net3 = MLP(hidden_layer_sizes=[5,6], hidden_activation='relu', output_size=4,
→ output_activation='sigmoid', n_iter=2000, loss_fn=mse, eta=0.05, random_state=54)
```

Code 8: MLP model 3 (best model)

در [برنامه ۸](#) دیده می‌شود که برای لایه‌های میانی به جای Relu استفاده شده است و علاوه بر این از تابع خطا mse استفاده شده است. در **شکل ۱۷** نمودار اتلاف و accuracy مدل ۳ دیده می‌شود. از **شکل ۱۷** مشاهده می‌شود که روند یادگیری به خوبی در



شکل ۱۷: خروجی مدل سوم

حال انجام است و دقت هم با هر epoch بیشتر می‌شود. علاوه بر این **شکل ۱۸** مقدار واقعی و پیش‌بینی شده مدل و **شکل ۱۹** نتایج طبقه بندی را نشان می‌دهد. از **شکل ۱۹** دیده می‌شود که دقت مدل افزایش یافته و دقت ۹۹ درصدی را شاهد هستیم. تمامی کلاس‌ها نمره



```

y_hat = mlp_net3.predict(X_test_sc)
y_hat = np.argmax(y_hat, axis=1)
test_labels_onehot = np.argmax(test_labels_onehot, axis=1)
print("actual labels: \n", test_labels_onehot, "\n predicted labels:\n", y_hat)

actual labels:
[2 3 3 2 0 2 0 2 1 0 0 1 2 2 2 0 2 3 3 3 1 0 2 0 2 0 2 2 2 3 3 2 3 3 0 3 0
1 3 1 3 0 1 0 3 3 3 3 0 3 1 2 0 2 1 3 1 1 2 3 3 1 0 1 3 0 0 1 2 0 3 0
2 1 1 2 1 0 3 2 1 2 1 2 2 1 0 3 2 1 2 0 0 0 1 1 0 3 0 3 1 3 3 0 1 2 2 0 3
1 0 0 2 3 2 0 2 0 1 2 2 2 2 2 3 3 3 0 3 2 1 3 2 2 3 2 0 0 0 3 3 2 2 2
1 2 1 1 1 2 0 1 3 3 2 2]

predicted labels:
[2 3 3 2 0 1 0 2 1 0 3 1 2 2 2 0 2 3 3 3 1 0 2 0 2 0 2 2 2 3 3 2 3 3 0 3 0
1 3 1 3 0 1 0 3 3 3 3 0 3 1 2 0 2 1 3 1 1 2 3 3 1 0 1 3 0 0 1 2 0 3 0
2 1 1 2 1 0 3 2 1 2 1 2 2 1 0 3 2 1 2 0 0 0 1 1 0 3 0 3 1 3 3 0 1 2 2 0 3
1 0 0 2 3 2 0 2 0 1 2 2 2 2 2 3 3 3 0 3 2 1 3 2 2 3 2 0 0 0 3 3 2 2 2
1 2 1 1 1 2 0 1 3 3 2 2]

```

شکل ۱۸: مقایسه لیبل واقعی و پیش‌بینی شده مدل سوم

```

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
model_report = classification_report(test_labels_onehot, y_hat)
print(model_report)

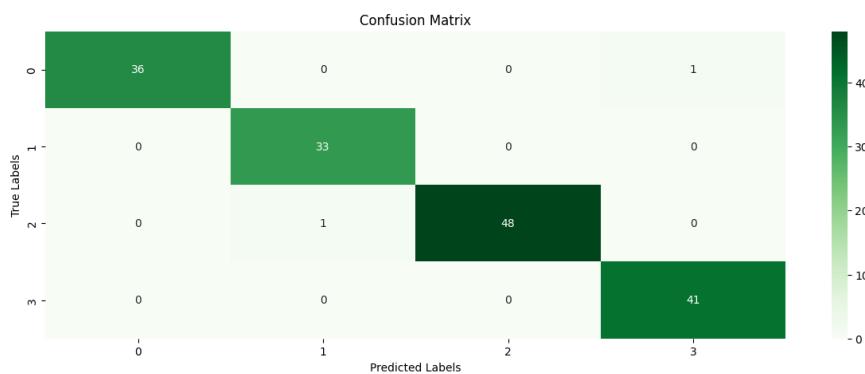
precision    recall    f1-score   support
          0       1.00      0.97      0.99      37
          1       0.97      1.00      0.99      33
          2       1.00      0.98      0.99      49
          3       0.98      1.00      0.99      41

accuracy                           0.99      160
macro avg       0.99      0.99      0.99      160
weighted avg    0.99      0.99      0.99      160

```

شکل ۱۹: نتیجه طبقه‌بندی مدل سوم

بالایی دارند و عملکرد طبقه‌بند در تمامی کلاس‌ها خیلی خوب است.



شکل ۲۰: ماتریس در هم ریختگی مدل سوم

از شکل ۲۰ می‌توان فهمید که مدل فقط دو داده را اشتباه طبقه‌بندی کرده است. در واقع یک داده در کلاس ۰ را در کلاس ۳ طبقه‌بندی کرده و یک داده در کلاس ۱ را در کلاس ۲ طبقه‌بندی کرده و سایر داده‌ها درست طبقه‌بندی شده‌اند.

۴.۳ بخش چهارم

در ابتدا با اشاره به مزایای استفاده از Cross Validation شروع می‌کنیم. استفاده از این تکنیک با جابجای کردن داده آموزش و تست، مارا از overfit شدن دور می‌کند. علاوه بر این مجموعه آموزش و تست روی کل داده‌ها می‌چرخد و از تمام داده هم برای آموزش و هم برای

تست استفاده می شود. علاوه بر این موارد یک تکنیک مقاوم برای تخمین عملکرد سیستم است چون نتایج طی چند بار run گرفتن نشان داده می شود و در نهایت استفاده از این تکنیک باعث حل مشکل bias-variance trade-off می شود و یک مدل با عمومیت بالاتر را برای ما تضمین می کند.

اما تفاوت این دو تکنیک در چیست؟ به طور خلاصه تفاوت این دو تکنیک در نحوه تقسیم داده ها به مجموعه آموزش و تست برای ارزیابی مدل بر می گردد. در CV, K-fold CV، داده ها به K فولد تقسیم می شوند، مدل روی K-1 آموزش می بیند و روی ۱ فولد باقی مانده تست گرفته می شود. این فرآیند K بار انجام می شود. در نهایت متوسط عملکرد مدل در تمامی مراحل گزارش می شود. در تکنیک Stratified K-fold CV هم دقیقا همین اتفاق می افتاد با این تفاوت که هنگام تقسیم بندی داده ها، این الگوریتم در نظر می گیرد که توزیع کلاس ها در هر فولد مشابه توزیع کلاس ها در کل داده باشد. برای مثال فرض کنید در یک طبقه بندی ۷۰ درصد داده ها متعلق به کلاس ۱ و باقی آن متعلق به کلاس ۰ است. در حالت CV, K-fold CV، ممکن است تعدادی از فولد ها تمامًا شامل داده های کلاس ۱ باشد و این موضوع می تواند منجر به بایاس شدن مدل بشود در صورتی که در حالت Stratified K-fold CV این نسبت ۷۰ به ۳۰ در تمامی فولدها رعایت می شود.

در نتیجه در داده هایی که دارای imbalance ای بالا هستند، حتما پیشنهاد می شود که از Stratified K-fold CV استفاده شود. با توجه به [شکل ۲۱](#) دیده می شود که داده های ما به صورت جدی imbalanced نیستند. در نتیجه می توان از همان تکنیک CV استفاده کرد. علاوه بر این در این بخش از مژاول آماده `sklearn.neural-network.MLPClassifier` استفاده می شود تا با قسمت های

```
[ ] y_train.value_counts(), y_valid.value_counts(), y_test.value_counts()
[ ] (8
      1.0    128
      3.0    124
      0.0    118
      2.0    110
      Name: count, dtype: int64,
      8
      0.0    45
      2.0    41
      1.0    39
      3.0    35
      Name: count, dtype: int64,
      8
      2.0    49
      3.0    41
      0.0    37
      1.0    33
      Name: count, dtype: int64)
```

شکل ۲۱: توزیع کلاس در داده های سوال دوم

قبل تفاوت داشته باشد و از این مژاول هم استفاده شده باشد. برای انجام اینکار ابتدا باید کل داده هارا scale کنیم. بعد از اینکار مدل را به کمک `MLPClassifier` و مانند [شکل ۲۲](#) می سازیم. سپس

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
mlp = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', max_iter=1000)
kf = KFold(n_splits=10)
```

شکل ۲۲: مدل ساخته شده به کمک `MLPClassifier`

نیاز است تا Cross Validation را تعریف کنیم، برای انجام اینکار هم می توانیم از Kfold استفاده کنیم و هم از cross-val-score که مورد اولی به ما اندیس های مربوطه را هنگام جدا کردن می دهد و دومی داده را برای ما جدا می کند. ما در اینجا از Kfold استفاده می کنیم که آزادی عمل بیشتری را در اختیار مان قرار می دهد. در [شکل ۲۳](#) این عمل انجام شده است. در یک حلقه for برای هر فولد متغیر شدن و پیش بینی کردن اجرا شده است و در نهایت دقت تست گزارش می شود و برای ۱۰ بار این عمل تکرار می شود.



```

scores = []
for train_index, test_index in kf.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y[train_index], y[test_index]

    mlp.fit(X_train, y_train)

    y_pred = mlp.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred))

scores = np.array(scores)
print("Cross-validation scores:", scores)
print("Mean cross-validation score:", scores.mean())

```

شکل ۲۳: انجام Cross validation

در مرحله بعد به کمک learning-curve این منحنی را رسم می‌کنیم. همچنین مقادیر انحراف معیار و میانگین هم فولد را حساب می‌کنیم تا در نمودار نهایی آن را نشان دهیم. در نهایت منحنی یادگیری را رسم می‌کنیم. **شکل ۲۴** این قسمت از کد را نشان می‌دهد در نهایت

```

train_sizes, train_scores, test_scores = learning_curve(
    mlp, X_scaled, y, cv=kf, n_jobs=-1, train_sizes=np.linspace(0.1, 1.0, 10), scoring='accuracy'
)

train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)

test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title("Learning Curve (MLPClassifier)")
plt.xlabel("Training examples")
plt.ylabel("Score")
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1, color="r")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")

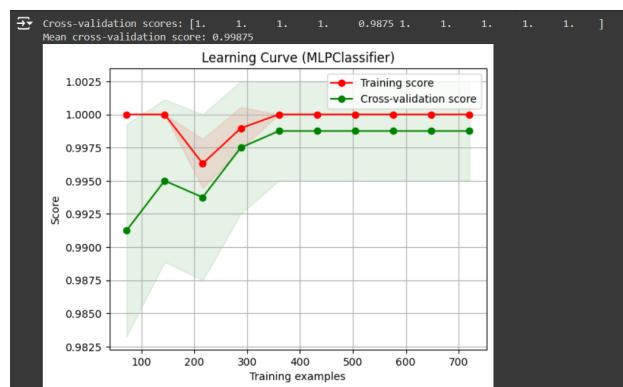
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

plt.legend(loc="best")
plt.show()

```

شکل ۲۴: کد رسم learning-curve

شکل ۲۵ خروجی را به ما نشان می‌دهد. این شکل حاوی دقت در مرحله‌های مختلف تغییر فولد است و در نهایت میانگین این دقت‌ها گرفته شده است. علاوه بر این منحنی یادگیری و امتیاز CV هم در نمودار آورده شده که نشان می‌دهد عملکرد یادگیری مدل بسیار خوب و پایدار است. مطابق نتایج تنها در یک فولد دقت کمتر از ۱۰۰ درصد داشتیم و همین موضوع میانگین کل را تحت تاثیر قرار داده است.



شکل ۲۵: انجام CV و رسم منحنی یادگیری



۴ سوال سوم

برای انجام این سوال از داده مربوط به دارو استفاده شده است. ابتدا مطابق شکل ۲۶ ویژگی ها و برچسب را جدا می کنیم.

```
[18] x = data.drop(['Drug'], axis = 1)
y = data['Drug']
x.shape, y.shape
→ ((200, 5), (200,))
```

شکل ۲۶: جدا کردن X و y

۱.۴ بخش اول

راه های مختلفی برای انجام این جداسازی وجود دارد که در اینجا به پنج مورد آن می پردازیم:

۱- استفاده از کتابخانه آماده به صورت شکل ۲۷. این تابع به صورت خودکار و رندوم، درصد مشخصی از تابع را جداسازی می کند.

```
[19] from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=54)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
→ ((160, 5), (40, 5), (160,), (40,))
```

شکل ۲۷: استفاده از Train-Test split

۲- استفاده از Stratified Train-Test split به صورت شکل ۲۸. در بخش ۴.۳ به مفهوم Stratified بودن پرداخته شد. به این معنا که

```
▶ from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=54)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
→ ((160, 5), (40, 5), (160,), (40,))
```

شکل ۲۸: استفاده از Stratified Train-Test split

توزیع کلاس ها در داده آموزش و ارزیابی (تست) با داده اصلی یکسان خواهد بود که از مواردی مانند bias شدن جلوگیری می کند.

۳- استفاده از CV که در بخش ۴.۳ به آن اشاره شد و شکل ۲۹ آن را نشان می دهد.

```
[ ] from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold, cross_val_score

kf = KFold(n_splits=5, shuffle=True, random_state=54)
model = DecisionTreeClassifier(random_state=54)
scores = cross_val_score(model, X, y, cv=kf)

print("Cross-validation scores:", scores)
print("Average cross-validation score:", scores.mean())
```

شکل ۲۹: استفاده از CV

۴- استفاده از تکنیک Leave one out که هر داده را یک فولد درنظر میگرد و توان پردازشی بالایی را می طلبد. شکل ۳۰ نشانگر این روش

```
[ ] from sklearn.model_selection import LeaveOneOut, cross_val_score
loo = LeaveOneOut()
scores = cross_val_score(model, X, y, cv=loo)
```

شکل ۳۰: استفاده از Leave one out

است.

۵- استفاده از یک روش دستی. به این صورت که ابتدا داده هارا shuffle می کنیم تا داده ها کاملا توزیع رندوم داشته باشند. سپس به اندازه دلخواه از داده هارا می برمی واندیس آن هارا به صورت **شکل ۳۱** جدا می کنیم. سپس به اندازه این اندیس ها داده های آموزش و ارزیابی را جدا می کنیم. برای انتخاب بین موارد بالا ابتدا توزیع کلاس ها در داده را که به صورت **شکل ۳۲** است را بررسی می کنیم. مطابق **شکل ۳۲**

```
[25] from sklearn.utils import shuffle
data = shuffle(data, random_state=54)

test_size = 0.2
border_idx = int((1-test_size) * len(data))

X_train = data.drop(['Drug'], axis = 1)[:,border_idx:]
X_test = data.drop(['Drug'], axis = 1)[border_idx:]
y_train = data['Drug'][:border_idx]
y_test = data['Drug'][border_idx:]

X_train.shape, X_test.shape, y_train.shape, y_test.shape
→ ((160, 5), (40, 5), (160,), (40,))
```

شکل ۳۱: جدا کردن داده ها به صورت دستی

```
[23] y.value_counts()
```

Drug	
drugY	91
drugX	54
drugA	23
drugB	16
drugC	16
Name:	count, dtype: int64

شکل ۳۲: توزیع برچسب ها در داده

ما یک داده imbalanced داریم و بهتر از روش stratified برای جدا کردن داده های آموزش و تست استفاده کنیم. گزینه دیگر استفاده از CV است که می توان برای تنظیم های پارامتر ها از آن استفاده کرد. ولی در این مرحله به صورت **شکل ۳۳** داده هارا جداسازی می کنیم. در مرحله بعد باید به سراغ تبدیل متغیر های categorical به داده های عددی برویم. ابتدا به **شکل ۳۴** نگاهی بیندازید. همانطور که دیده می شود سه ویژگی Sex، Bp و Cholesterol و همینطور برچسب، داده های categorical هستند. از LabelEncoder **شکل ۳۵** برای این تبدیل استفاده می کنیم. همانطور که از **شکل ۳۵** مشخص است از یک حلقه for استفاده شده تا ویژگی های categorical به انکو در داده شوند. نکته مهم دیگر این است که باید مدل داده های تست را بیند به همین دلیل از transform برای داده های تست استفاده شده و fit نشده است.

در مرحله بعد به تعریف مدل می پردازیم. در ابتدا یک مدل ساده میسازیم و هایپرپارامتر خاصی را عوض نمی کنیم تا ابتدا خروجی را بررسی کنیم. این مدل در **شکل ۳۶** آمده است. در **شکل ۳۷** آمده است و دیده می شود که تمامی برچسب ها درست پیش بینی شده است. در **شکل ۳۸** آمده است و دیده می شود که تمامی برچسب ها درست پیش بینی شده است. در **شکل ۳۸** دیده می شود که تمامی کلاس ها به درستی تقسیم بندی شده اند و مدل هیچ خطای نداشته است. همین مورد می توان نشانه ای از over model بودن درخت تصمیم باشد و ممکن است نیاز به مواردی مانند تنظیم هایپرپارامتر ها و هرس کردن درخت داشته باشیم. در **شکل ۳۹** ماتریس در هم ریختگی مشاهد می شود که نکته در **شکل ۳۸** را تایید می کند و نشان می دهد تمامی کلاس ها درست پیش بینی شده اند. حال به تفسیر مدل می رسیم که از نقاط قوت Decision Tree ها است. **شکل ۴۰** مدل مارا به تصویر می کشد.

```

❶ from sklearn.model_selection import train_test_split
❷ X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=54)
❸ X_train.shape, X_test.shape, y_train.shape, y_test.shape
❹ ((160, 5), (40, 5), (160,), (40,))

[29] y_train.value_counts()
❺ Drug
drugY    73
drugX    43
drugA    18
drugB    13
drugC    13
Name: count, dtype: int64

[30] y_test.value_counts()
❻ Drug
drugY    18
drugX    11
drugA     5
drugB     3
drugC     3
Name: count, dtype: int64

```

شکل ۳۳: تقسیم داده ها به مجموعه آموزش و تست

```

❶ data.info()
❷ <class 'pandas.core.frame.DataFrame'>
Index: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Age         200 non-null    int64  
 1   Sex          200 non-null    object  
 2   BP           200 non-null    object  
 3   Cholesterol 200 non-null    object  
 4   Na_to_K      200 non-null    float64
 5   Drug          200 non-null    object  
dtypes: float64(1), int64(1), object(4)
memory usage: 10.9+ KB

```

شکل ۳۴: نوع داده ها

```

❶ from sklearn.preprocessing import LabelEncoder
❷ label_encoders = {}
❸ def encode_labels(train, test, columns):
    for col in columns:
        le = LabelEncoder()
        train[col] = le.fit_transform(train[col])
        test[col] = le.transform(test[col])
        label_encoders[col] = le
    return train, test
❹ categorical_columns = ['Sex', 'BP', 'Cholesterol']
❺ X_train, X_test = encode_labels(X_train, X_test, categorical_columns)

❻ le_drug = LabelEncoder()
❼ y_train = le_drug.fit_transform(y_train)
❼ y_test = le_drug.transform(y_test)

❽ print("Training set shape:", X_train.shape, y_train.shape)
❽ print("Test set shape:", X_test.shape, y_test.shape)
❾ Training set shape: (160, 5) (160,)
Test set shape: (40, 5) (40,)

```

Label Encoder : ۳۵

مطابق شکل ۴۰ اولین ویژگی که مدل با آن root node را می سازد نسبت سدیم به پتانسیم یا (NA to K) است. انتظار می رویم که خلوص با پیشرفت درخت بیشتر شود پس جینی باید در مراحل بعد کم تر شود. علاوه بر این دیده می شود که ما کلا داریم و درخت تصمیم با این داده آموزش می بینند. با توجه به اینکه بیشتری داده ها مربوط به کلاس drugY هستند، کلاس مربوط به این node این کلاس معروفی شده است. دیده می شود که با یک شرط کل کلاس drugY جدا می شود. در node جینی صفر است و تمامی ۷۳ نمونه مرتبط با کلاس drugY جداسازی شده است. اگر $N_{\text{to}K} \leq 14.829$ بود وارد 1 node می شویم که شامل ۸۷ سمپل غیر از



```
[82] from sklearn import tree
clf = tree.DecisionTreeClassifier(random_state = 54)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

شکل ۳۶: درخت تصمیم ۱

```
[83] print(f"actual labels: \n {y_test} \n predicted labels: \n {y_pred}")
actual labels:
[4 4 1 4 0 4 3 3 3 0 0 3 4 2 1 4 4 3 4 3 4 4 0 0 2 4 2 4 4 3 3 4 4 4 3 4 3
3 4 1]
predicted labels:
[4 4 1 4 0 4 3 3 3 0 0 3 4 2 1 4 4 3 4 3 4 4 0 0 2 4 2 4 4 3 3 4 4 4 3 4 3
3 4 1]
```

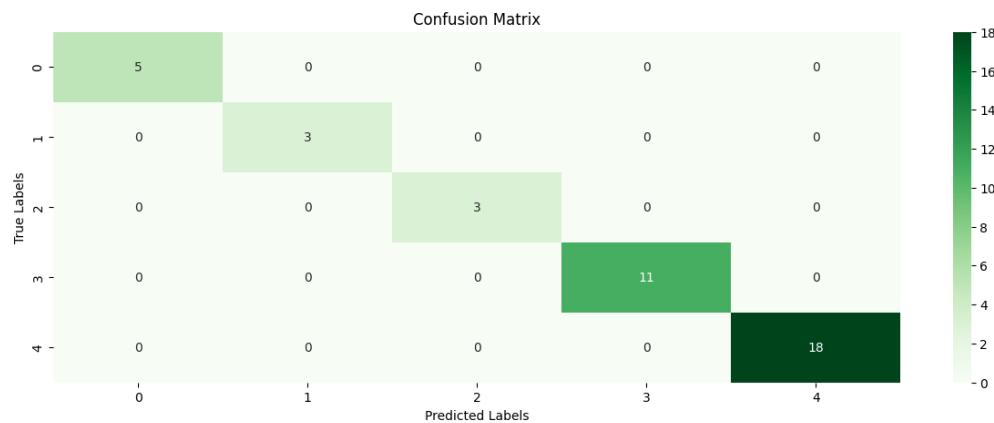
شکل ۳۷: برچسب واقعی و پیش بینی شده درخت تصمیم اول

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
model_report = classification_report(y_test, y_pred)
print(model_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
```

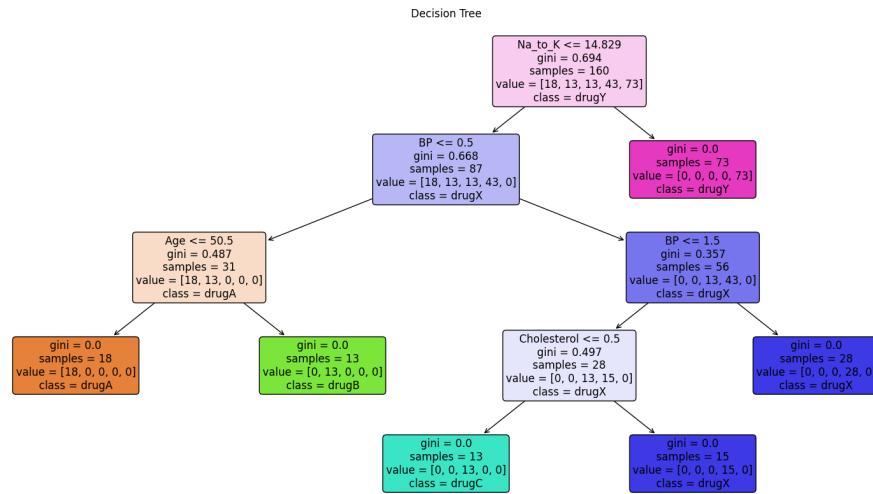
شکل ۳۸: نتایج طبقه بندی درخت تصمیم اول



شکل ۳۹: ماتریس در هم ریختگی درخت تصمیم اول

کلاس drugY است. در این حالت اگر $\text{bloodpressure} \leq 0.5$ node 3 بود وارد node 4 و در غیر این صورت وارد node 4 می شویم. دقیق شود که با پایین رفتن در درخت، میزان خلوص هر Desicion node بیشتر می شود به طوری که در هر ۶ leaf node ۱۰۰٪ خلوص داریم و تمامی کلاس ها به درستی و دقیق ۱۰۰٪ درصد از هم جدا شده اند.

در شکل ۴۱ با کمک یک کد آماده، به صورت متنی درخت را تحلیل شده است که ما از روی شکل ۴۰ خودمان این کار را انجام دادیم.



شکل ۴۰: تصویرسازی درخت تصمیم اول

```

☞ The binary tree structure has 11 nodes and has the following tree structure:
node=0 is a split node with value=[[18, 13, 13, 43, 73]]: go to node 1 if X[:, 4] <= 14.828500278843506 else to node 10.
node=1 is a split node with value=[[18, 13, 13, 43, 0]]: go to node 2 if X[:, 2] <= 0.5 else to node 5.
node=2 is a split node with value=[[18, 13, 0, 0, 0]]: go to node 3 if X[:, 0] <= 50.5 else to node 4.
node=3 is a leaf node with value=[[18, 0, 0, 0, 0]].
node=4 is a leaf node with value=[[0, 13, 0, 0, 0]].
node=5 is a split node with value=[[0, 0, 13, 43, 0]]: go to node 6 if X[:, 2] <= 1.5 else to node 9.
node=6 is a split node with value=[[0, 0, 0, 13, 15, 0]]: go to node 7 if X[:, 3] <= 0.5 else to node 8.
node=7 is a leaf node with value=[[0, 0, 0, 13, 0, 0]].
node=8 is a leaf node with value=[[0, 0, 0, 15, 0, 0]].
node=9 is a leaf node with value=[[0, 0, 0, 28, 0, 0]].
node=10 is a leaf node with value=[[0, 0, 0, 0, 73, 0]].
  
```

شکل ۴۱: تحلیل متی درخت تصمیم اول

۲.۴ بخش دو

معیار های مختلف برای مدل اول در [بخش ۱.۴](#) مورد بررسی قرار گرفت و دیدیم که با عدم تنظیم فرآپارامتر ها مدل به دقت ۱۰۰ درصد می رسد و می تواند تمامی کلاس هارا با خلوص کامل از هم جدا کند. در این بخش، تعدادی فرآپارامتر به مدل اضافه می کنیم و نتایج را بررسی می کنیم.

مدل در [شکل ۴۲](#) نشان داده شده است. فرآپارامتر ها شامل max features و min samples split و max leaf nodes هستند.

```

[125] from sklearn import tree
clf = tree.DecisionTreeClassifier(min_samples_split = 30, max_features = 'sqrt', max_leaf_nodes = 4, random_state = 54)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("actual labels:\n", y_test, "\npredicted labels:\n", y_pred)

☞ actual labels:
[4 4 1 4 0 4 3 3 3 0 0 3 4 2 1 4 4 3 4 4 0 0 2 4 2 4 4 3 3 4 4 4 3 4 3
3 4 1]
predicted labels:
[4 4 0 4 0 4 3 3 3 0 0 3 4 3 0 4 4 3 4 3 4 0 0 3 4 3 4 3 3 4 4 4 3 4 3
3 4 0]
  
```

شکل ۴۲: درخت تصمیم دوم

: حداقل تعداد سمپل هایی را که نیاز است تا یک نود split شود را برای درخت تعیین می کند. برای مثال اگر این



فراپارامتر برابر با ۳۰ تنظیم شود و تعداد سمپل های یک leaf node decision node به ۲۸ برسد، این یک leaf node خواهد شد و دیگر تقسیم نمی شود.

max features: تعداد ویژگی هایی که درخت برای طبقه بندی از آن استفاده می کند و در این برابر \sqrt{N} تنظیم شده، یعنی اگر ۴ ویژگی داشته باشیم درخت از تنها دو تای آنها برای طبقه بندی استفاده می کند.

max leaf nodes: تعداد کل leaf node هایی که درخت می تواند داشته باشد را محدود می کند. از دیگر فراپارامتر های خوب درخت تصمیم می توان به max depth و ccp alpha اشاره کرد که برای هرس کردن درخت به کار می روند. در شکل ۴۳ نتایج طبقه بندی درخت تصمیم ۲ دیده می شود. از شکل ۴۳ دیده می شود که دقت کل به ۶۰ درصد کاهش یافته است.

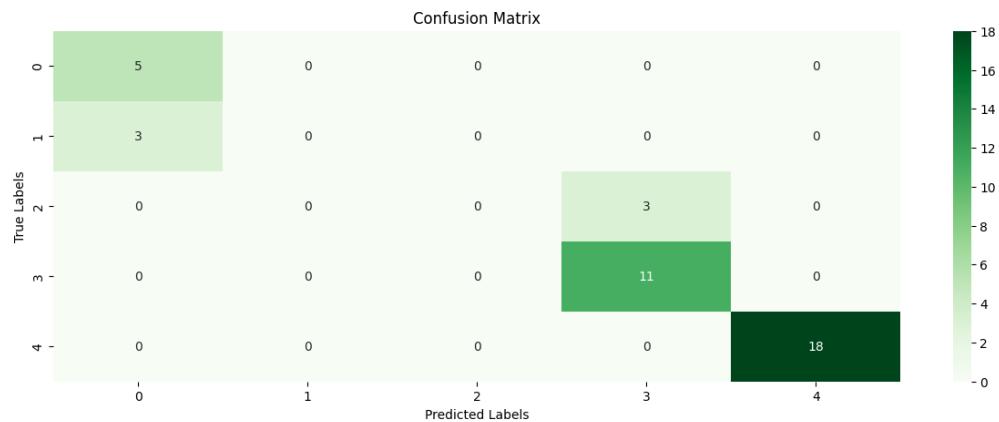
	precision	recall	f1-score	support
0	0.62	1.00	0.77	5
1	0.00	0.00	0.00	3
2	0.00	0.00	0.00	3
3	0.79	1.00	0.88	11
4	1.00	1.00	1.00	18
accuracy			0.85	40
macro avg	0.48	0.60	0.53	40
weighted avg	0.74	0.85	0.79	40

شکل ۴۳: نتایج طبقه بندی درخت تصمیم دو

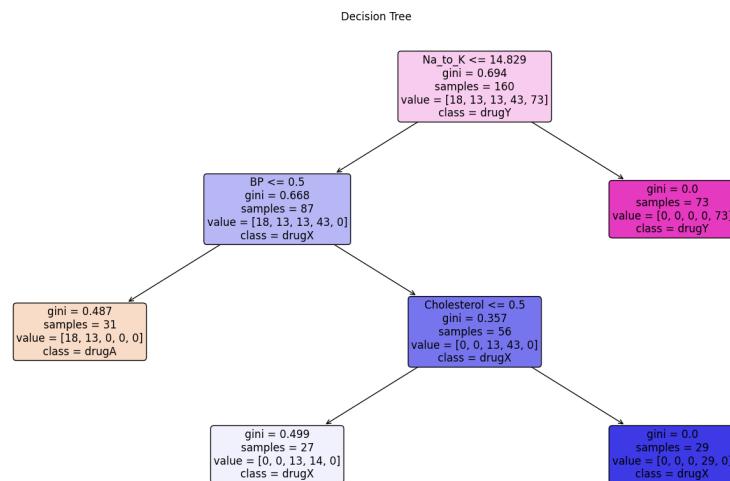
دلیل آن هم واضح است. مدل ما هیچ داده ای را در کلاس های ۱ و ۲ پیش بینی نکرده که می تواند نتیجه محدود کردن زیاد مدل و عدم توانایی مدل در تطبیق خود با پیچیدگی داده ها باشد. در واقع مدل به اندازه کافی پیچیده نیست تا داده های ۱ و ۲ را از بقیه داده ها جدا کند. بهترین نتیجه را در کلاس چهار که همان Y drug است دارد و هم recall precision و هم خوبی دارد و در نتیجه f1-score بالایی هم خواهد داشت. در کلاس ۰ precision حدود ۶۰ درصد است به این معنا که از ۸ نمونه ای که مدل گفته است به کلاس ۰ تعلق دارد، ۶۰ درصد آن واقعاً به کلاس ۰ تعلق دارد و حدود ۴۰ درصد خطأ داریم. در نقطه مقابل recall این کلاس ۱۰۰ است چون از ۵ موردی که متعلق به کلاس صفر بوده همگی به درستی تشخیص داده شده است.

در شکل ۴۴ ماتریس در هم ریختگی مشاهد می شود. هرچه مقادیر روی قطر تجمع بیشتری داشته باشند، مدل عملکرد بهتری داشته. در این ماتریس می توان دید که مدل ما هیچ داده ای را از کلاس ۱ و ۲ در دسته بندی جدا قرار نداده است. ۳ داده مربوط به کلاس ۱ را در کلاس صفر طبقه بندی کرده و ۳ داده مربوط به کلاس ۲ را در کلاس ۳ طبقه بندی کرده است. با تنظیم های پردازشی دادیم که خلوص هر نود کمتر بماند و این ۶ داده که اشتباه طبقه بندی شده اند نتیجه این استراتژی ما بود. در شکل ۴۵ می توان کاری که درخت تصمیم انجام داده است را مشاهده کرد. شکل ۴۵ نشان می دهد که درخت ما نسبت به شکل ۴۰ جمیوجور تر شده است و خلوص هر leaf node کم تر از شکل ۴۰ است. در نگاه بعدی می توان دید که ما کلاس ۴ leaf node در مقایسه با شکل ۴۰ مربوط به مدل ۱ که ۶ leaf node داشت. علاوه بر این می توان دید که تمام طبقه بندی روی سه ویژگی انجام شده است که K, NA to Bp, Cholesterol و drugC هستند. با توجه به شکل ۴۵ دیده می شود که هیچ کلاسی از drugB و drugC در leaf node در نهاده وجود ندارد و این همان موضوع است که درخت هیچ داده ای را در کلاس ۱ و ۲ طبقه بندی نمی کند و پیچیدگی لازم را ندارد. با نگاه کردن به ۳ node می توان دید که مدل می توانسته با توجه به شرط min samples split max leaf nodes که برابر با ۴ تنظیم شده بود، جلوی اینکار را می گیرد. در نهایت شکل ۴۶ می توان خلوصیت lead node هارا مشاهده کرد.

اما برای تغییرات، مدل ۳ در شکل ۴۷ آمده است. همانطور که دیده می شود، این درخت کمتر هرس شده و شرط آزادانه تر هستند تا مدل پیچیدگی بیشتری نسبت به مدل ۲ به خود بگیرد. نتایج در شکل ۴۸ هستند. همانطور که دیده می شود دقت بسیار بهتر شده و به ۹۳ درصد رسیده است. ۸۳ درصد از پیش بینی های مدل مربوط به کلاس صفر درست هستند و مدل توانسته ۶۷ درصد از داده های کلاس ۱ را درست طبقه بندی کند. یعنی نسبت به مدل قبلی توانسته دو تا از سه تا داده ای که اشتباه طبقه بندی شده بودند را در کلاس درست قرار



شکل ۴۴: ماتریس در هم ریختگی درخت تصمیم دوم



شکل ۴۵: تصویرسازی درخت تصمیم دوم

```

The binary tree structure has 7 nodes and has the following tree structure:
node=0 is a split node with value=[[18, 13, 13, 43, 73]]: go to node 1 if X[:, 4] <= 14.828500270843506 else to node 2.
node=1 is a split node with value=[[18, 13, 13, 43, 0]]: go to node 3 if X[:, 2] <= 0.5 else to node 4.
node=2 is a leaf node with value=[[ 0, 0, 0, 0, 73]].
node=3 is a leaf node with value=[[18, 13, 0, 0, 0]].
node=4 is a split node with value=[[ 0, 0, 13, 43, 0]]: go to node 5 if X[:, 3] <= 0.5 else to node 6.
node=5 is a leaf node with value=[[ 0, 0, 13, 14, 0]].
node=6 is a leaf node with value=[[ 0, 0, 0, 29, 0]].
  
```

شکل ۴۶: تحلیل متنی درخت تصمیم دوم

دهد. در نهایت شکل ۴۹ درخت تصمیم ۳ را تصویر سازی می کند. در این شکل می بینیم که درخت همچنان بزرگ است (مانند مدل ۱) و پیچیدگی بیشتری به خود گرفته است. در مدل ۴، پارامتر ccp-alpha را در مدل قرار می دهیم که میزان هرس کردن را مشخص می



```
[141] from sklearn import tree
      clf = tree.DecisionTreeClassifier(min_samples_split = 2, max_features = 5, max_leaf_nodes = 10, random_state = 54)
      clf.fit(x_train, y_train)
      y_pred = clf.predict(x_test)
      print(f'actual labels: \n{y_test} \n predicted labels:\n{y_pred}')

→ actual labels:
[4 4 1 4 0 4 3 3 3 0 0 3 4 2 1 4 4 3 4 3 4 4 0 0 2 4 2 4 4 3 3 4 4 4 3 4 3
3 4 1]
predicted labels:
[4 4 1 4 0 4 3 3 3 0 0 3 4 2 0 4 4 3 4 3 4 4 0 0 2 4 2 4 4 3 3 4 4 4 3 4 3
3 4 1]
```

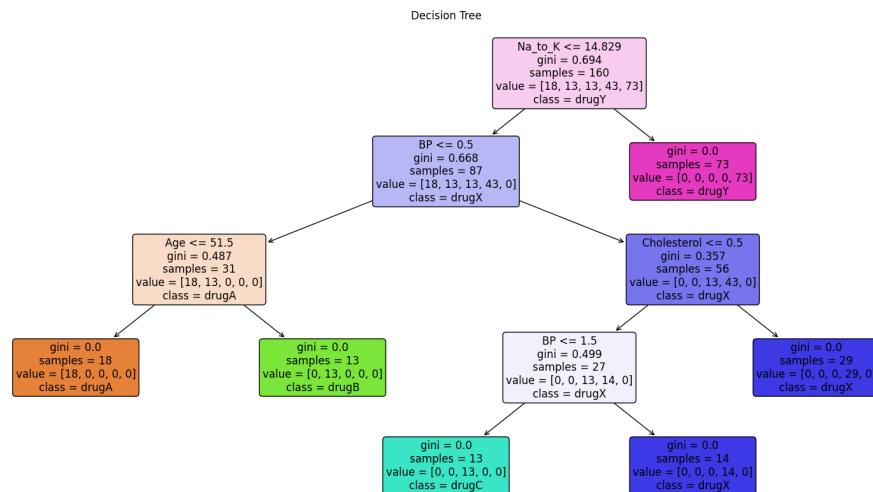
شکل ۴۷: درخت تصمیم سوم

```
[142] from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
      model_report = classification_report(y_test, y_pred)
      print(model_report)

→
      precision    recall   f1-score   support
      0            0.83     1.00     0.91      5
      1            1.00     0.67     0.80      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          0.97
      macro avg       0.97     0.93     0.94      40
      weighted avg    0.98     0.97     0.97      40
```

شکل ۴۸: نتایج طبقه بندی درخت تصمیم سوم



شکل ۴۹: تصویرسازی درخت تصمیم سوم

کند شکل ۵۰ این مدل را نشان می دهد. هرچه ccp-alpha بزرگتر باشد، هرس بیشتری را شاهد خواهیم بود. نتایج این مدل در شکل ۵۱ آمده است. دیده می شود که به دقت ۹۳ درصد رسیده ایم و مشکل در کلاس ۲ است که طبقه بند هیچ کدام از داده های مربوط به این کلاس را در کلاس ۲ طبقه بندی نکرده است. داده های مربوط به این کلاس تمام در طبقه ۳ مطابق ماتریس در هم ریختگی طبقه بندی شده اند. در شکل ۵۲ تصویرسازی مدل انجام شده است. همانطور که دیده می شود، کمی از درخت هرس شده و شکل آن جموجور تر شده است که به تنظیم فراپارامتر ها برمیگردد.

```
[156] from sklearn import tree
      clf = tree.DecisionTreeClassifier(min_samples_split = 2, max_features = 4, ccp_alpha = 0.07, random_state = 54)
      clf.fit(X_train, y_train)
      y_pred = clf.predict(X_test)
      print(f'actual labels: \n {y_test} \n predicted labels:\n {y_pred}')

actual labels:
[4 4 1 4 0 4 3 3 3 0 0 3 4 2 1 4 4 3 4 3 4 4 0 0 2 4 2 4 4 3 3 4 4 4 3 4 3
3 4 1]
predicted labels:
[4 4 1 4 0 4 3 3 3 0 0 3 4 3 0 4 4 3 4 3 4 4 0 0 3 4 3 4 4 3 3 4 4 4 3 4 3
3 4 1]
```

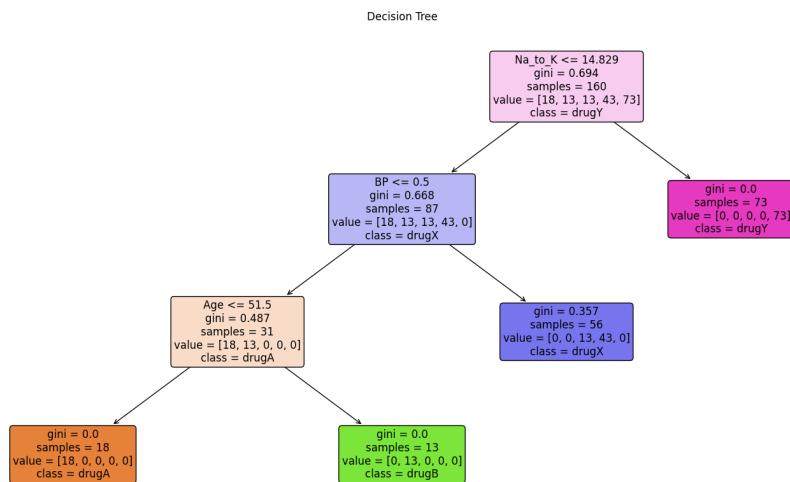
شکل ۵۰: درخت تصمیم چهارم

```
[157] from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
      model_report = classification_report(y_test, y_pred)
      print(model_report)

precision    recall   f1-score   support
          0       0.83     1.00     0.91      5
          1       1.00     0.67     0.80      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                           0.90      40
macro avg       0.72     0.73     0.72      40
weighted avg    0.85     0.90     0.87      40
```

شکل ۵۱: نتایج درخت تصمیم چهارم



شکل ۵۲: تصویرسازی درخت تصمیم چهارم

۳.۴ بخش سوم

به طور کلی RandomForest و AdaBoost در خانواده تکنیک های مبنی بر آنده از چندین یادگیرنده ضعیف تر را باهم جمع می کند و نتیجه اش را به ما می دهد. اینکار مانند این است که برای تشخیص بیماری، به جای اتکا به نظر یک دکتر، به سراغ چندین دکتر برویم و بر اساس برآمد صحبت چند متخصص تصمیم گیری کنیم. چون این روش های متکی به استفاده از چندین weak learner هستند، می توان نگفت که در نهایت مارا بهتر از overfit دور می کنند و مشکل bias-variance را تا حدی بهبود می بخشنند. نکته ای که در رابطه با تفاوت این دو روش مطرح است مبتنی بودن RandomForest بر روش های bagging و



بر روشنایی AdaBoost است.

کارکرد RandomForest به این صورت است که داده آموزش را به تعدادی subset با جاگذاری تقسیم می کند و برای هر کدام یک درخت تشکیل می دهد. سپس در نهایت نتیجه کار طبقه بندی درخت هارا برای گیری به ما می دهد. همین مسئله می تواند آن را به یک روش با نیازمندی به توان محاسباتی بالا تبدیل کند. اما کارکرد AdaBoost به این صورت است که یک طبقه بند درست می کند و با توجه به خطای وزن بیشتر دادن به داده هایی که اشتباه طبقه بندی شده اند نتیجه را در چند مرحله به ما می دهد.

در ابتدا یک مدل جنگل تصادفی می سازیم که در [شکل ۵۳](#) آمده است. مدل ما تعداد ۱۰۰ درخت تشکیل می دهد و حداقل سه پل هایی

```
from sklearn.metrics import classification_report
clf_rf = RandomForestClassifier(n_estimators = 100, min_samples_split = 2, max_depth= 10, max_features = 5, random_state = 54, verbose = True)
clf_rf.fit(X_train, y_train)

y_pred = clf_rf.predict(X_test)

print("Actual labels: \n{}\n".format(y_test))
print("Predicted labels: \n{}\n".format(y_pred))

print(classification_report(y_test, y_pred))

Actual labels:
[4 4 1 4 0 4 3 3 0 0 3 4 2 1 4 4 3 4 3 4 4 0 0 2 4 2 4 4 3 3 4 4 4 3 4 3
 3 4 1]

Predicted labels:
[4 4 1 4 0 4 3 3 3 0 0 3 4 2 0 4 4 3 4 3 4 4 0 0 2 4 2 4 4 3 3 4 4 4 3 4 3
 3 4 1]

precision    recall   f1-score   support
          0       0.83      1.00      0.91       5
          1       1.00      0.67      0.80       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                           0.97      40
macro avg       0.97      0.93      0.94      40
weighted avg    0.98      0.97      0.97      40
```

شکل ۵۳: مدل جنگل تصادفی

که باید وجود داشته باشد تا node تقسیم بشود ۲ است. حداقل عمق درخت ها ۱۰ است و مدل می تواند از ۵ ویژگی استفاده بکند. در نهایت با Verbose از مدل گزارش می گیریم. مطابق [شکل ۵۳](#) دیده می شود که دقت مدل ۹۳ درصد است و مدل در تمامی کلاس ها به خوبی عمل می کند. در کلاس ۱ از کل تعداد نمونه هایی که مدل به ملاس اتلاف کرده ۶۷ درصدشان درست است در بخش بعدی از AdaBoost استفاده می کنیم و مطابق [شکل ۵۴](#) برای تنظیم فرآپارامتر ها از GridSearch استفاده می کنیم. برای انجام

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import GridSearchCV

model = AdaBoostClassifier(random_state=42)
parameters = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 1]
}

grid_search = GridSearchCV(estimator=model, param_grid=parameters, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Best Parameters:", best_params)
print("Test Accuracy:", accuracy)

Best Parameters: {'learning_rate': 0.01, 'n_estimators': 50}
Test Accuracy: 0.85
```

شکل ۵۴: مدل AdaBoost

GridSearch باید یک مجموعه از هایپرپارامتر هایی که میخواهیم تغییر دهیم درست کنیم که در parameters تعريف شده است. در نهایت می گوییم که برای بهترین دقت، کدام یک از این دو زوج تعريف شده بهتر هستند. همانطور که در نتایج می بینیم بهتری نرخ یادگیری ۰.۰۱ است و بهترین n-estimators ۵۰ است. بهترین دقت ما برای این مدل هم ۸۵ درصد است.

۵ سوال چهارم

۱.۵ کاوش و شناخت داده

دیتاستی که در این سوال با آن کار می کنیم، مجموعه ای از داده ها در ارتباط با وضعیت سلامت بیماران در حوزه تشخیص بیماری قلبی است. این دیتاست شامل داده های مربوط به ۴ دیتابیس کلیولند، مجارستان، سویس و لانگ بیچ است و شامل ۷۶ ویژگی است که ما در اینجا از ۱۴ تا از آن ها استفاده می کنیم. این ویژگی ها شامل مواردی از قبلی سن، جنسیت، نوع درد سینه (شامل ۴ نوع)، اطلاعات فشار خون و ... می شود.

داده ای که با آن کار می کنیم به صورت **شکل ۵۵** است. همانطور که در **شکل ۵۵** دیده می شود حدود ۱۰۰۰ سطر داریم و ۱۳ ویژگی که

	data = pd.read_csv('/content/heart.csv')
1	data
2	age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal target
3	0 52 1 0 125 212 0 1 168 0 1.0 2 2 3 0
4	1 53 1 0 140 203 1 0 155 1 3.1 0 0 3 0
5	2 70 1 0 145 174 0 1 125 1 2.6 0 0 3 0
6	3 61 1 0 148 203 0 1 161 0 0.0 2 1 3 0
7	4 62 0 0 138 294 1 1 106 0 1.9 1 3 2 0
8	...
9	1020 59 1 1 140 221 0 1 164 1 0.0 2 0 2 1
10	1021 60 1 0 125 258 0 0 141 1 2.8 1 1 3 0
11	1022 47 1 0 110 275 0 0 118 1 1.0 1 1 2 0
12	1023 50 0 0 110 254 0 0 159 0 0.0 2 0 2 1
13	1024 54 1 0 120 188 0 1 113 0 1.4 1 1 3 0
14	1025 rows × 14 columns

شکل ۵۵: داده مربوط به بیماری قلبی

شرایط فرد را ثبت کرده، در نهایت برچسب target به ما می گوید که مشکل قلبی وجود دارد (برچسب = ۱) و یا وجود ندارد (برچسب = ۰).

در **شکل ۵۶** اطلاعات مربوط به انواع داده ها در دیتاست دیده می شود. می توان دید که همه داده ها از نوع عددی هستند و نیازی به استفاده از روش های encoding نداریم. علاوه بر این هیچ داده null ای نداریم. این موضوع را از **شکل ۵۷** هم می توان دید. در **شکل ۵۸**

[1]:	data.info()
2	<class 'pandas.core.frame.DataFrame'>
3	RangeIndex: 1025 entries, 0 to 1024
4	Data columns (total 14 columns):
5	# column Non-Null Count Dtype
6	0 age 1025 non-null int64
7	1 sex 1025 non-null int64
8	2 cp 1025 non-null int64
9	3 trestbps 1025 non-null int64
10	4 chol 1025 non-null int64
11	5 fbs 1025 non-null int64
12	6 restecg 1025 non-null int64
13	7 thalach 1025 non-null int64
14	8 exang 1025 non-null int64
15	9 oldpeak 1025 non-null float64
16	10 slope 1025 non-null int64
17	11 ca 1025 non-null int64
18	12 thal 1025 non-null int64
19	13 target 1025 non-null int64
20	dtypes: float64(1), int64(13)
21	memory usage: 112.7 kB

شکل ۵۶: نوع داده مربوط به بیماری قلبی

نیز داده به طور کلی توصیف شده. اطلاعاتی که می توان از این شکل یافت مواردی مانند این است که سن افراد در این داده بین ۲۹ تا ۷۷ سال است و میانگین این سن ۵۶ سال است. در **شکل ۵۹** ماتریس هم بستگی آورده شده است. اطلاعات مربوط به **شکل ۵۹** را می توان در **شکل ۶۰** بهتر خواند. از **شکل ۶۰** مشخص است که ویژگی هایی مانند cp، thalach، exang، oldpeak، fbs هم بستگی بیشتری با target دارند و مدل با آن ها به نتیجه بهتر و دقیق تری خواهد رسید. در مقابل مواردی مانند chol و fbs هم بستگی کمی با target دارند و به بیانی دیگر رابطه تصادفی با متغیر هدف دارند.

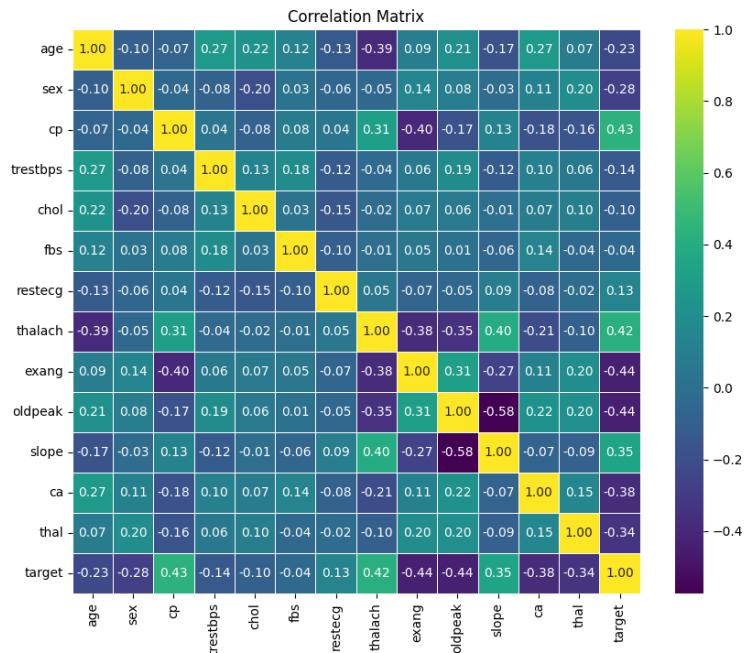
```
[33] data.isna().sum()
```

	age	sex	cp	trestbps	chol	fb	restecg	thalach	exang	oldpeak	slope
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.000000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.617755
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000
50%	56.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	275.000000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000

شکل ۵۷: داده های null

data.describe()												
	age	sex	cp	trestbps	chol	fb	restecg	thalach	exang	oldpeak	slope	
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	
mean	54.434146	0.695610	0.942439	131.611707	246.000000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366	
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.617755	
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	
25%	48.000000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000	
50%	56.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000	
75%	61.000000	1.000000	2.000000	140.000000	275.000000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000	
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	

شکل ۵۸: توصیف داده



شکل ۵۹: ماتریس هم بستگی

نکته دیگری که باید مورد بررسی قرار بگیرد بالا نسبت بودن یا نبودن داده ها است. مطابق شکل ۶۱ می یک داده متعادل داریم که برچسب های ۱ و غیر ۰ توزیع یکسانی دارند.

مورد بعدی توزیع بین زن و مرد در داده است که در شکل ۶۲ آمده است. مطابق این نمودار تعداد مردان حدود ۵.۲ برابر زنان در این دیتاست است.

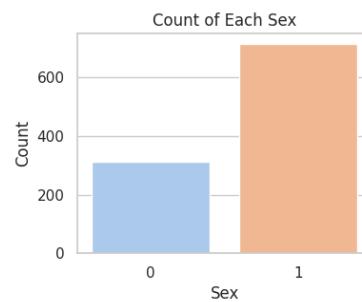
برای نشان دادن سایر توزیع ها در داده از شکل ۶۳ استفاده می کنیم. مطابق شکل ۶۳ چولگی نمودار در کلسیتروول و فشار خون در حالت



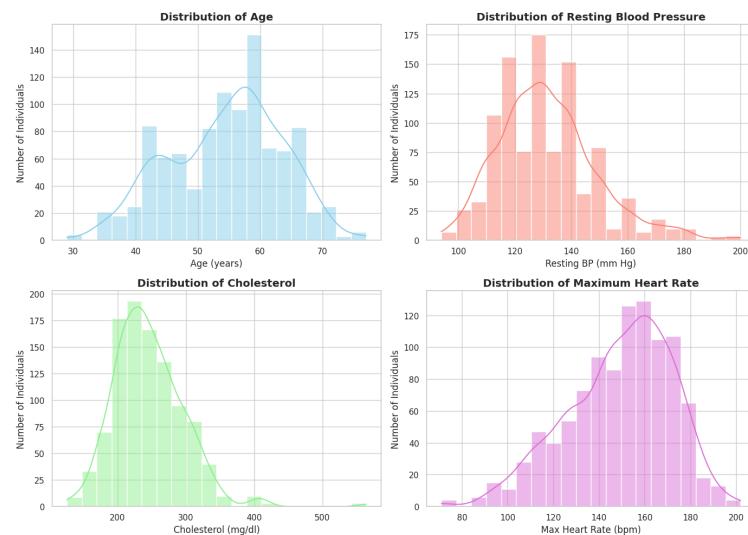
شکل ۶۰: میزان هم بستگی ویژگی ها با متغیر وابسته

```
[18]: proportion = len(data[data['target'] == 1]) / len(data['target'])
print(f'the percentage of label 1 in the whole data is {proportion:.2f} thus dataset is balanced')
# The percentage of label 1 in the whole data is 0.51 thus dataset is balanced
```

شکل ۶۱: توزیع داده



شکل ۶۲: توزیع جنسیت در داده

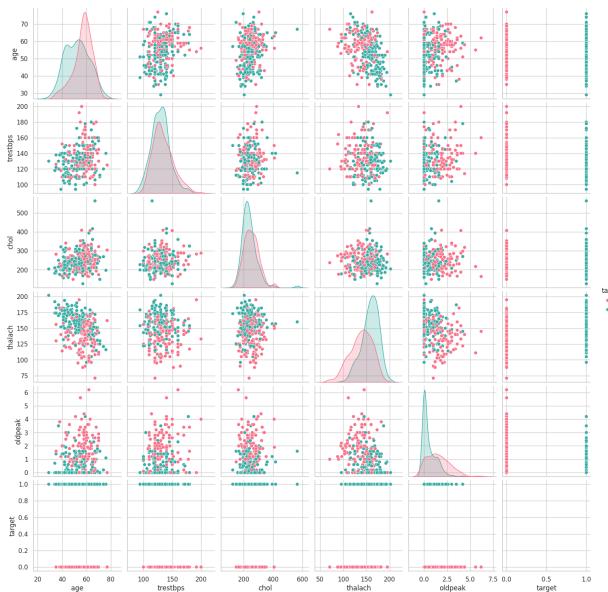


شکل ۶۳: نحوه توزیع جنسیت، فشار خون، کلسترول و حداکثر ضربان در داده



استراحت به سمت چپ (مقادیر کمتر) و چولگی در حداکثر ضربان قلب به سمت راست (مقادیر بیشتر) است. میانگین ضربان قلب حدود ۱۵۵ است. نمودار سن نیز توزیع نرمالی دارد.

نمودار توزیع نقطه ای نیز در **شکل ۶۴** آمده است. بالا انس بودن داده ها در این نمودار مشخص است و از روی نمودار های روی قطر اصلی می توان متوجه شد غیر از ویژگی oldpeak که برای بیماران خیلی کمتر است در سایر ویژگی ها نزدیکی شاهد هستیم. علاوه بر این می توان برخی ویژگی هایی که برای جداسازی به ما کمک می کنند را به صورت چشمی دید برای مثال توجه به thalach که هم بستگی زیادی هم دارد مهم است.



شکل ۶۴: توزیع نقطه ای داده

۲.۵ تقسیم داده، پیش پردازش و ساخت مدل

داده را مطابق **شکل ۶۵** جدا سازی و به دو بخش آموزش و آزمون تقسیم می کنیم. سپس مطابق **شکل ۶۶** داده های آموزش و آزمون را به

```
[41]: X = data.drop(['target'], axis=1)
y = data['target']
X_train, y_train = X[0:800], y[0:800]
X_test, y_test = X[800:], y[800:]

#(42): from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

شکل ۶۵: تقسیم داده ها به آموزش و آزمون

کمک standard scaler می کنیم.

در نهایت مدل را مطابق **شکل ۶۷** می سازیم. علاوه بر استفاده از مدل آماده می توانیم از کلاس ساخته شده به صورت دستی هم استفاده کنیم که در **شکل ۶۸** آمده است. توجه شود که این مدل یک فرآپارامتر var-smoothing دارد که به میزان variance smoothing می پردازد و چون در این مسئله تاثیر خاصی ندارد از آن استفاده نمی کنیم.



```
[4]: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train
```

array([[0.38472981, 0.66417225, 1.89718082, ..., -0.58884277,
 | 0.19910989, 0.66417225, 1.89718082, ..., 1.91989644,
 | 2.108216, -0.31786361, ..., 0.58884277,
 | 1.91989644, 0.66417225, 0.000156, ..., -0.58884277,
 | -0.70710285, 1.0002316], ...]

شکل ۶۶: scale کردن داده ها

```
[5]: from sklearn.naive_bayes import GaussianNB
SGBM = GaussianNB()
SGBM.fit(X_train, y_train)
y_pred = SGBM.predict(X_test)
```

شکل ۶۷: مدل GaussianNB در سایکیت لرن

شکل ۶۸: مدل NaiveBayes ساخته شده در کلاس

۳.۵ بررسی و تحلیل نتایج

در شکل ۶۹ نتایج طبقه بندی مدل آمده است. می توان دید از کل لیل هایی که توسط مدل در کلاس صفر طبقه بندی شده اند، درصدشان واقعا به ۰ تعلق دارند و مدل توانسته است ۷۸ درصد از داده های کلاس صفر را به درستی در طبقه صفر قرار دهد. تعداد داده های تست متعلق به کلاس صفر ۹۱ است. برای کلاس ۱ داریم که از تمام داده هایی که مدل در کلاس ۱ قرار داده ۸۳ درصدشان درست است و توانسته ۸۸ درصد از داده های کلاس ۱ را به درستی در کلاس ۱ قرار دهد. از f1-score می توان گفت که عملکرد طبقه بند در کلاس ۱ بهتر از کلاس ۰ است. چون ما اینجا کار تشخیص بیماری انجمام می دهیم باید recall در کلاس ۱ را افزایش دهیم. این متريک به ما می گويد ما دقیقاً توانسته ايم چند بیمار واقعی را در دسته بیماران قرار دهیم و در این حالت حدود ۱۲ درصد خطأ داریم. یعنی ۱۲ درصد از بیماران را در دسته افراد سالم قرار داده ایم که ایده آل نیست. همانطور که در شکل ۶۹ دیده می شود دقت کلی مدل ۸۳ درصد

```
[6]: from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
model_report = classification_report(y_test, pred_my)
print(model_report)
```

	precision	recall	f1-score	support
0	0.84	0.78	0.81	91
1	0.83	0.88	0.85	114
accuracy	0.83	0.83	0.83	205
macro avg	0.83	0.83	0.83	205
weighted avg	0.83	0.83	0.83	205

شکل ۶۹: نتایج طبقه بندی

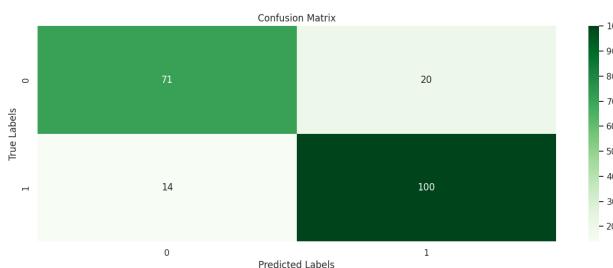
است. می بینیم که دو نوع گزارش macro و weighted macro در این نتیجه داریم که با رجوع به documentation به نتایج زیر می رسیم. دقت کلی یا accuracy برابر میزان نسبت برچسب های درست تشخیص داده شده بر کل پیش بینی های مدل است. در واقع می گوید چند درصد از کل پیش بینی های انجام شده درست هستند. مورد بعدی macro average است که به ما یک میانگین بدون وزن از precision، recall و f1-score می دهد. مثلا برای این مسئله برای recall ۸۳ درصد گزارش شده که میانگین ۷۸ درصد و ۸۸ درصد است. مورد بعدی weighted average است که با توجه به تعداد داده های support در هر کلاس به مقادیر بالا وزن می دهد و میانگین می



گیرد. رابطه ۶ نحوه محاسبه این متريک را نشان می دهد.

$$\text{Average Weighted} = \frac{\sum_{i=1}^n (\text{Metric}_i \times \text{Support}_i)}{\sum_{i=1}^n \text{Support}_i} \quad (6)$$

مورد بعدی micro average است. برای طبقه بندی چند کلاسه یا چند برجسبی با زیرمجموعه ای از کلاس ها، میانگین خرد مشارکت همه کلاس ها را برای محاسبه میانگین متريک در سطح جهانی جمع آوری می کند. با اين حال، برای طبقه بندی با يك رويه یا چند کلاسه با همه کلاس ها، میانگين ميكرو با دقت مطابقت دارد، بنابراین در اينجا به طور جداگانه گزارش نشده است. مورد بعدی ماتریس در هم ریختگی است که در شکل ۷۰ آمده است. میتوان دید که ۱۴ بیمار بودند که مدل ما از طبقه بندی درست آن ها باز مانده و اين همان ۱۲ درصد خطای است که در تحلیل نتایج از آن گفتیم.



شکل ۷۰: ماتریس در هم ریختگی

در بخش بعدی باید چند داده تصادفی انتخاب کنیم و مقدار واقعی و پیش بینی شده برای آن را بررسی کنیم. به طور کلی دو کلاس صفر و یک داریم به همین دلیل برچسب واقعی و پیش بینی شده یا یک است یا صفر. این کار در شکل ۷۱ انجام شده است. ابتدا انديس های

```

❶ import random
random_indices = random.sample(range(len(y_test)), 5)
print('True labels:', y_test.iloc[random_indices].values)
print('Predicted labels:', y_pred[random_indices])
❷ True labels: [0 1 1 0 1]
Predicted labels: [0 1 1 0 0]

```

شکل ۷۱: مقایسه کلاس واقعی و پیش بینی شده

رندهم به تعداد ۵ تا و در فاصله ۰ تا تعداد سمپل های تست تولید میکنیم. مثلا انديس های ۲۰، ۳۴، ۱۰۰ و ۱۶۹ تولید می شوند. حالا باید مقادیر تست و پیش بینی شده برای اين داده ها را ببینیم. همانطور که دیده می شود از اين ۵ داده، کلاس ۰ تا از آن ها درست پیش بینی شده ولی داده آخر که متعلق به کلاس ۱ بوده، در کلاس صفر طبقه بندی شده است.