به نام خدا

MP2

دانشجو : مصطفی نبی پور

شماره دانشجویی : ۴۰۱۱۲۸۶۴

نشانی github :

https://github.com/mostafanb77/ML4022_MP2

نشانی google drive :

https://drive.google.com/drive/folders/1agwPAwdYvPxLN8z1QAY6XFcHkL_AhZE_?usp=sharing

سوال ۱) الف)

هنگامی که ReLU با یک تابع Sigmoid دنبال می شود، خروجی لایه ReLU که غیر منفی است، به ورودی تابع Sigmoid تبدیل می شود.

اگر خروجی ReLU بسیار زیاد باشد، تابع Sigmoid این مقدار را به سمت 1 سوق می دهد و باعث اشباع می شود و منجر به گرادیان های بسیار کوچک می شود. اگر خروجی ReLU صفر یا نزدیک به صفر باشد، Sigmoid این مقدار را به سمت 0.5 سوق می دهد، که می تواند منجر به یادگیری ناکارآمد شود، زیرا گرادیان در اینجا نسبتا کوچک است اما غیر صفر است.

تابع Sigmoid در لایه نهایی معمولاً در مسائل طبقه بندی باینری برای خروجی یک امتیاز احتمال مانند بین 0 و 1 استفاده می شود. با این حال، داشتن آن به عنوان یک تابع فعال سازی میانی (درست بعد از ReLU) می تواند مشکل ساز باشد. شبکه ممکن است به دلیل مشکلات گرادیان و اثرات بالقوه اشباع با یادگیری به طور موثر مشکل داشته باشد.

این ترکیب همچنین می تواند منجر به از دست دادن اطلاعات مفید گرادیان شود، که باعث همگرایی کند یا گیر کردن در حداقل های محلی در طول تمرین شود.

ترکیب ReLU به دنبال Sigmoid می‌تواند منجر به نورون‌هایی شود که مقادیر بسیار کوچک (نزدیک به 0.5) یا مقادیر نزدیک به 1 را تولید می‌کنند و به طور موثر قدرت بازنمایی شبکه را محدود می‌کنند.

شبکه ممکن است در یادگیری ویژگی های موثر مشکل داشته باشد، به خصوص در معماری های عمیق تر که این ترکیب می تواند مسائل مربوط به گرادیان را تشدید کند.

ب)

برای محاسبه گرادیان داریم :(با فرض اینکه تابع Loss ما BCE باشد.)

$$\frac{\partial u}{\partial w_1} = x \,, \frac{\partial \hat{y}}{\partial u} = \begin{cases} 1 & u \geq 0 \\ \alpha e^u & u < 0 \end{cases} \,, \frac{\partial L}{\partial \hat{y}} = -\frac{\big((1-\hat{y})y + (1-y)\hat{y}\big)}{\hat{y}(1-\hat{y})Ln(10)}$$

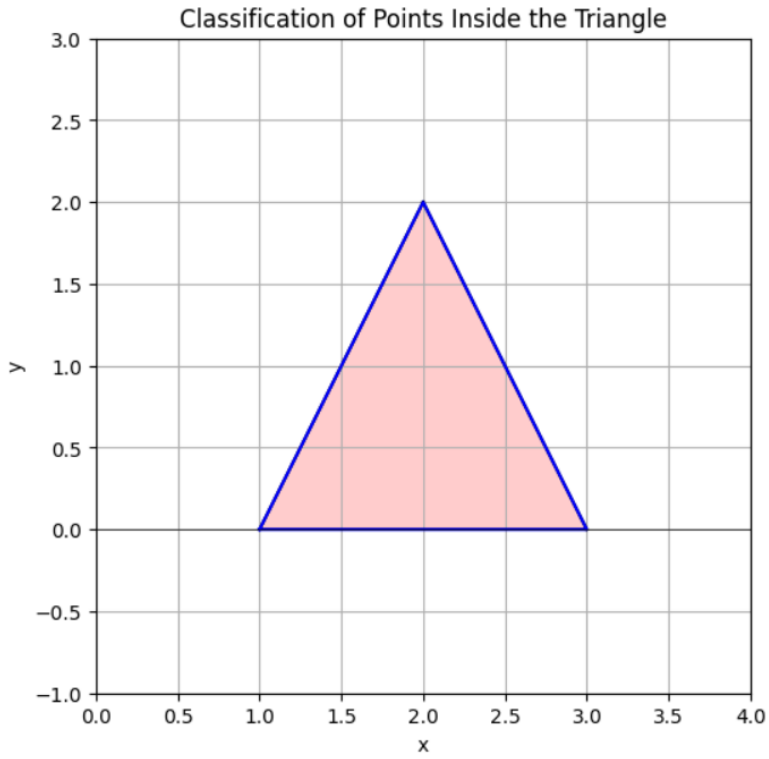
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} \frac{\partial u}{\partial w_1}$$

در اینجا u در واقع ورودی به ELU میباشد.(u = XW) برای w_0 فقط : $\frac{\partial \hat{y}}{\partial u} \cdot \frac{\partial u}{\partial w_0} = 1$ در واقع مشتق تابع ELU میباشد.

وقتی واحدهای ReLU برای ورودی‌های منفی صفر خروجی می‌کنند، گاهی اوقات می‌توانند منجر به نورون‌های «مرده» شوند که هرگز در هیچ نقطه داده‌ای در مجموعه آموزشی فعال نمی‌شوند. این زمانی اتفاق می افتد که وزن ها به گونه ای تنظیم شوند که ورودی ReLU همیشه منفی باشد. هنگامی که یک نورون می میرد، یادگیری را متوقف می کند زیرا گرادیان برای همه ورودی های منفی صفر است.از طرف دیگر ELU دارای یک گرادیان غیر صفر برای ورودی های منفی است : αe^u

ج)

شبکه ای که طراحی کردیم در واقع یک تابع میباشد که نام آن perceptron است و خروجی کدی که زدیم بدین شکل میباشد :



در واقع همینجور که میبینید نقاط به درستی معلوم شده اند.نکته ای که در این بخش وجود داره اینه که در واقع ۳ خط وجود دارند که بدین شکل تعریف شده اند :

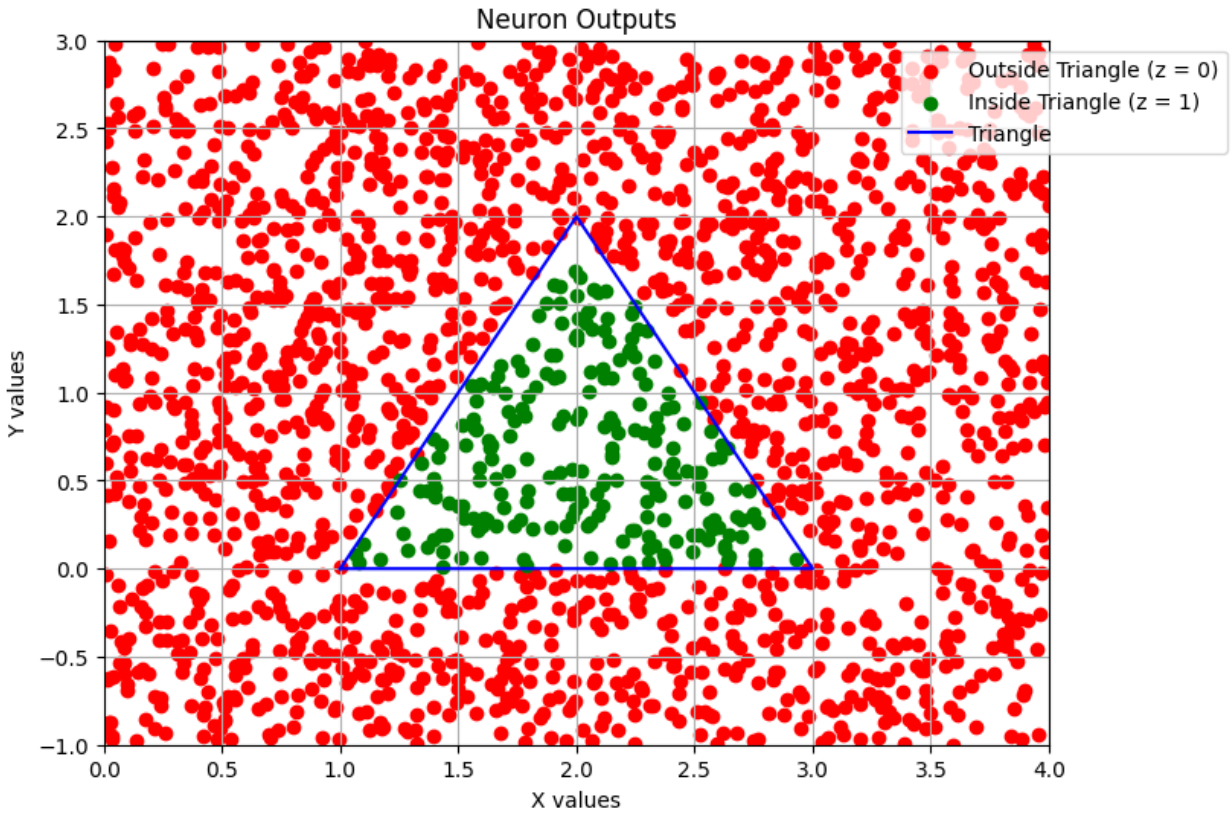
```
def perceptron(x, y):
    # Weights and biases for each condition
    weights_1 = np.array([0, 1])
    bias_1 = 0

    weights_2 = np.array([2, -1])
    bias_2 = -2

    weights_3 = np.array([-2, -1])
    bias_3 = 6

    # Conditions as perceptron linear combinations
    condition_1 = np.dot(weights_1, np.array([x, y])) + bias_1 > 0
    condition_2 = np.dot(weights_2, np.array([x, y])) + bias_2 > 0
    condition_3 = np.dot(weights_3, np.array([x, y])) + bias_3 > 0

    # Return True if all conditions are met
    return condition_1 and condition_2 and condition_3
```

خوب همینجور که میبینید وزن ها و بایاس نیز در این کد معلوم هستند.در واقع کاری که در اینجا کرده ایم بردار X ما به شکل Transpose[X,Y] میباشد و activation function ما نیز به صورت sign میباشد. به عنوان مثال :

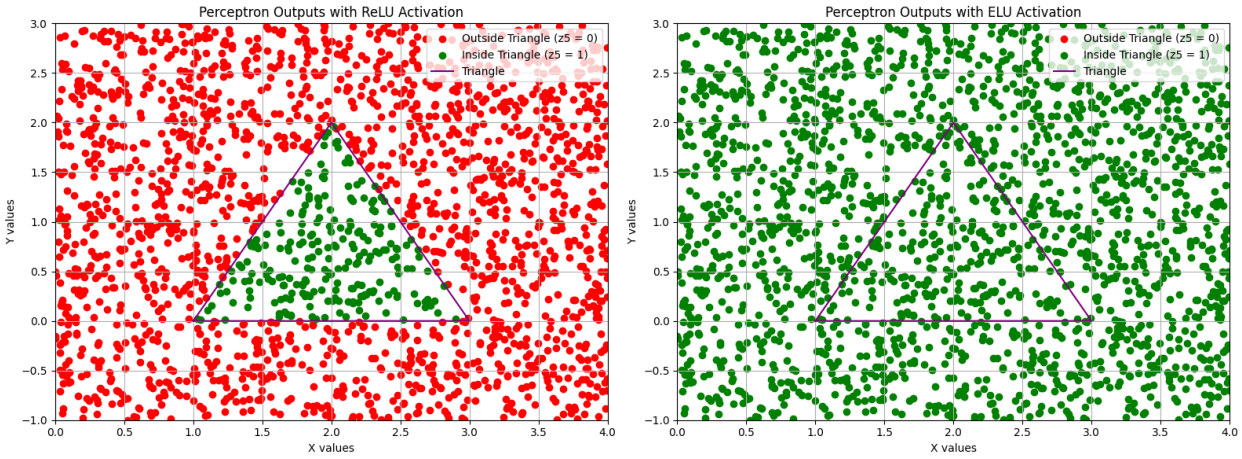
$$-2x - y + 6 \geq 0 \qquad condition\ 3$$

با این پرسپترون ما ۳ خط را معلوم کرده ایم. اگر هر سه شرط برآورده شوند (True)، خروجی را به ما میدهد.

در بخش بعدی سوال خواسته شده که ۲۰۰۰ نقطه رندوم درست کرده و به پرسپترون بدهیم که جواب آن بدین شکل میشود :



و در آخر نیز خواسته که activation function ها را بر روی آن امتحان کنیم که در واقع برای perceptron نیز این کار را کرده بودیم (در واقع همین که بزرگ تر مساوی ها را معلوم کرده بودیم یک جور تابع فعالساز sign میباشد که برای این task استفاده میشد.)خروجی activation function ها بدین شکل میباشد.(Relu,ELU):



خوب همینجور که میبنید برای ELU نمیتوان این کار را انجام داد زیرا ELU هم برای تمام بازه ها خروجی دارد به همین دلیل نمیتواند تابع فعالساز خوبی برای این کار باشد ولی Relu چون برای داده های کوچک تر از 0 نیز 0 میباشد میتوان استفاده کرد.حالت جدید با تابع فعالساز بدین شکل است :

```python
# Define the perceptron function with ReLU activation
def perceptron_relu(x, y):
    weights_1 = np.array([0, 1])
    bias_1 = 0
    weights_2 = np.array([2, -1])
    bias_2 = -2
    weights_3 = np.array([-2, -1])
    bias_3 = 6

    def relu(z):
        return max(0, z)

    output_1 = relu(np.dot(weights_1, np.array([x, y])) + bias_1)
    output_2 = relu(np.dot(weights_2, np.array([x, y])) + bias_2)
    output_3 = relu(np.dot(weights_3, np.array([x, y])) + bias_3)

    return bool(output_1 and output_2 and output_3)

# Define the perceptron function with ELU activation
def perceptron_elu(x, y, alpha=.1):
    weights_1 = np.array([0, 1])
    bias_1 = 0
    weights_2 = np.array([2, -1])
    bias_2 = -2
    weights_3 = np.array([-2, -1])
    bias_3 = 6
```

```
    def elu(z, alpha):
        return z if z > 0 else alpha * (np.exp(z) - 1)

    output_1 = elu(np.dot(weights_1, np.array([x, y])) + bias_1, alpha)
    output_2 = elu(np.dot(weights_2, np.array([x, y])) + bias_2, alpha)
    output_3 = elu(np.dot(weights_3, np.array([x, y])) + bias_3, alpha)

    return bool(output_1 and output_2 and output_3)
```

برای relu واضح است که به چه شکل تعریف شده برای elu بدین شکل میباشد که اگر شروط خط های قبلی که تعریف کرده باشیم بالاتر از ۰ باشند که خودشان اگر نباشند فرمول elu را بر خط پیاده کردیم.

گزارش کد بخش اول :

```
# Generate a grid of points
x_vals = np.linspace(0, 4, 800)
y_vals = np.linspace(-1, 3, 800)
X, Y = np.meshgrid(x_vals, y_vals)
```

شبکه ای از نقاط را در محدوده های مشخص شده ایجاد می کند:

x_vals از 0 تا 4 با 800 امتیاز متغیر است.

y_vals از -1 تا 3 با 800 امتیاز متغیر است.

np.meshgrid یک شبکه دو بعدی از نقاط از x_vals و y_vals ایجاد می کند.

```
# Classify each point in the grid
Z = np.array([[perceptron(x, y) for x in x_vals] for y in y_vals])
```

تابع پرسپترون را در هر نقطه از شبکه اعمال می کند.

Z یک آرایه دو بعدی است که در آن هر عنصر نشان می دهد که نقطه مربوطه در داخل (1) یا خارج (0) مثلث است.

```
# Plot the results
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot([1, 2], [0, 2], 'k-')
ax.plot([1, 3], [0, 0], 'k-')
ax.plot([2, 3], [2, 0], 'k-')
triangle = Polygon([[1, 0], [2, 2], [3, 0]], closed=True,
edgecolor='black', facecolor='#ffcccc' )
```

طرح را با اندازه x68 اینچ تنظیم می کند.

لبه های مثلث را با استفاده از ax.plot با مختصات رسم می کند:

از (1، 0) تا (2، 2).

از (1، 0) تا (3، 0).

از (2، 2) تا (3، 0).

یک شی Polygon ایجاد می کند که مثلث را با رنگ صورت قرمز (#ffcccc) و لبه های سیاه نشان می دهد.

```
# Plot the triangle in blue
triangle_points = np.array([[1, 0], [2, 2], [3, 0], [1, 0]])
plt.plot(triangle_points[:,0], triangle_points[:,1], color='blue',
label='Triangle')

ax.add_patch(triangle)
ax.set_xlim(0, 4)
ax.set_ylim(-1, 3)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Classification of Points Inside the Triangle')
ax.grid(True)
ax.axhline(0, color='black', linewidth=0.5)
ax.axvline(0, color='black', linewidth=0.5)
ax.set_aspect('equal', adjustable='box')
plt.show()
```

علاوه بر این، طرح مثلث را به رنگ آبی برای دید بهتر ترسیم می کند.

وصله مثلثی را به طرح اضافه می کند.

محدودیت های محور x و y را تنظیم می کند.

محورها را برچسب گذاری می کند و عنوان طرح را تعیین می کند.

برای خوانایی بهتر یک شبکه به طرح اضافه می کند.

خطوط محور x و y را در 0 برای مرجع اضافه می کند.

نسبت ابعاد برابر را برای طرح تضمین می کند.

نمودار را با استفاده از plt.show() نمایش می دهد.

گزارش کد بخش دوم :

```
# Define the  neuron function for your network
def Area(x, y):
    return perceptron(x, y)
```

تابع Area یک بسته بندی در اطراف تابع پرسپترون است که برای تعیین اینکه یک نقطه در داخل یا خارج مثلث است استفاده می شود.

```
# Generate random data points
num_points = 2000  # Increased number of points for better visualization
x_values = np.random.uniform(0, 4, num_points)
y_values = np.random.uniform(-1, 3, num_points)
```

ایجاد 2000 داده تصادفی با:

x_values به طور یکنواخت بین 0 و 4 توزیع شده است.

y_values به طور یکنواخت بین -1 و 3 توزیع شده است.

```
# Initialize lists to store data points for different z5 values
red_points = []
green_points = []

# Evaluate data points using the Area function
for i in range(num_points):
    z_value = Area(x_values[i], y_values[i])
    if z_value == False:
        red_points.append((x_values[i], y_values[i]))
    else:
        green_points.append((x_values[i], y_values[i]))

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

فهرست ها را برای ذخیره نقاط طبقه بندی شده در داخل (سبز) یا خارج از مثلث (قرمز) راه اندازی می کند.

روی هر نقطه تولید شده تکرار می شود و از تابع Area برای طبقه بندی آن استفاده می کند.

اگر نقطه خارج از مثلث (False) باشد به red_points اضافه می شود.

اگر نقطه داخل مثلث باشد (True) به green_points اضافه می شود.

مختصات x و y نقاط طبقه بندی شده را برای رسم جدا می کند.

مواردی را که ممکن است green_points خالی باشد رسیدگی می کند تا از خطا جلوگیری کند.

```
# Plotting
plt.figure(figsize=(8, 6))
plt.scatter(red_x, red_y, color='red', label='Outside Triangle (z = 0)')
plt.scatter(green_x, green_y, color='green', label='Inside Triangle (z =
1)')
plt.xlabel('X values')
plt.ylabel('Y values')
plt.title('Neuron Outputs')
```

یک طرح با اندازه 8x6 اینچ ایجاد می کند.

نقاطی را که به عنوان خارج از مثلث طبقه بندی شده اند به رنگ قرمز و داخل مثلث با رنگ سبز ترسیم می کند.

محورها را برچسب گذاری می کند و عنوان طرح را تعیین می کند.

کد بخش سوم :

```
# Initialize lists to store data points for different z5 values
red_points_relu = []
red_points_elu = []
green_points_relu = []
green_points_elu = []

# Evaluate data points using the Area functions
for i in range(num_points):
    z5_value_relu = Area_relu(x_values[i], y_values[i])
    z5_value_elu = Area_elu(x_values[i], y_values[i])
    if z5_value_relu == False:
        red_points_relu.append((x_values[i], y_values[i]))
    else:
        green_points_relu.append((x_values[i], y_values[i]))
    if z5_value_elu == False:
        red_points_elu.append((x_values[i], y_values[i]))
```

```
    else:
        green_points_elu.append((x_values[i], y_values[i]))
```

فهرست ها را برای ذخیره نقاط طبقه بندی شده در داخل (سبز) یا خارج (قرمز) مثلث برای هر دو فعال سازی ReLU و ELU، راه اندازی می کند.

روی هر نقطه تولید شده تکرار می شود و از توابع Area_relu و Area_elu برای طبقه بندی آنها استفاده می کند.

سوال ۲) الف)

در اینجا ما دقیقا مانند مینی پروژه قبل عمل کرده و دقیقا با همان feature ها کارمان را انجام میدهیم. در آخر ماتریسی که بدست می آوریم با ابعاد 400x10 میباشد که ستون آخر در واقع label داده ها میباشد.

درک کلاس های نقص جدید

1. Ball Defect (File: 1B007_0)

نقص توپ به آسیب یا ناهنجاری های موجود در عناصر غلتشی (گوی) بلبرینگ اشاره دارد. این عیوب می توانند به صورت گودال، ریزش یا ترک بر روی سطح توپ ظاهر شوند که منجر به بی نظمی در حرکت چرخش و افزایش سیگنال های ارتعاشی می شود. فایل خاص B007_01 داده های ارتعاش یک یاتاقان با نقص توپ را می گیرد.

2. Outer Race Defect (File: OR007@6_0)

یک outer race defect در مسیر بیرونی بلبرینگ رخ می دهد. outer race حلقه ثابتی است که توپ ها یا غلتک ها در برابر آن قرار می گیرند. نقص در این ناحیه می تواند باعث ایجاد الگوهای ارتعاشی متمایز در هنگام عبور عناصر غلتشی از ناحیه آسیب دیده شود. فایل OR007@6_0 داده‌های ارتعاش یک یاتاقان با outer race defect را در بر می‌گیرد.

مجموعه اعتبار سنجی

مجموعه اعتبار سنجی بخش جداگانه ای از داده های مورد استفاده در طول مرحله آموزش برای تنظیم فراپارامترها و تصمیم گیری در مورد معماری مدل است. این یک ارزیابی بی‌طرفانه از عملکرد مدل ارائه می‌کند و با اطمینان از تعمیم مدل به خوبی به داده‌های دیده نشده، به جلوگیری از برازش بیش از حد کمک می‌کند. مجموعه اعتبارسنجی برای انتخاب مدل و تنظیم فراپارامتر ضروری است.

در اینجا ۲۰ درصد داده ها را به test و ۲۰ درصد را برای validation و بقیه برای آموزش انتخاب کردیم.

داده ها را با روش standardization نرمال سازه کردیم.(دستور standardscalar)

ب)

تا الان همه کارها مانند قبل بود الان سراغ درست کردن مدل خود میرویم که بدین شکل است :

```
# Define the MLP model
model = Sequential([
    Dense(64, activation='elu', input_shape=(features_train.shape[1],)),
    Dense(32, activation='elu'),
    Dense(4, activation='softmax')  # 4 classes for classification
])

# Compile the model
model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

همینطور که میبینید مدل در لایه اول ۶۴ (لایه پنهان) و در لایه بعدی ۳۲ و در خروجی ۴ نورون دارد.

optimizer=Adam():

پارامتر بهینه ساز الگوریتم بهینه سازی را برای استفاده در طول آموزش مشخص می کند. در این مورد از بهینه ساز Adam استفاده می شود.

Adam (Adaptive Moment Estimation) یک الگوریتم بهینه‌سازی است که مزایای دو الگوریتم محبوب دیگر را ترکیب می‌کند: AdaGrad و RMSProp. این برای مشکلات با مجموعه داده های بزرگ و فضاهای پارامتر با ابعاد بالا مناسب است.

بهینه ساز Adam نرخ یادگیری را در طول تمرین تنظیم می کند و سرعت همگرایی و عملکرد کلی را بهبود می بخشد.

:'loss='sparse_categorical_crossentropy

پارامتر ضرر تابع ضرر را برای استفاده در طول تمرین مشخص می کند. در این حالت از تلفات متقاطع طبقه بندی شده پراکنده استفاده می شود.

sparse_categorical_crossentropy برای مسائل طبقه‌بندی چند کلاسه استفاده می‌شود که در آن برچسب‌ها اعداد صحیح هستند (نه کدگذاری یک‌طرفه).

این تابع از دست دادن زمانی مناسب است که مشکل طبقه بندی با بیش از دو کلاس دارید و برچسب های هدف شما اعداد صحیح هستند (به عنوان مثال 0، 1، 2، ...).

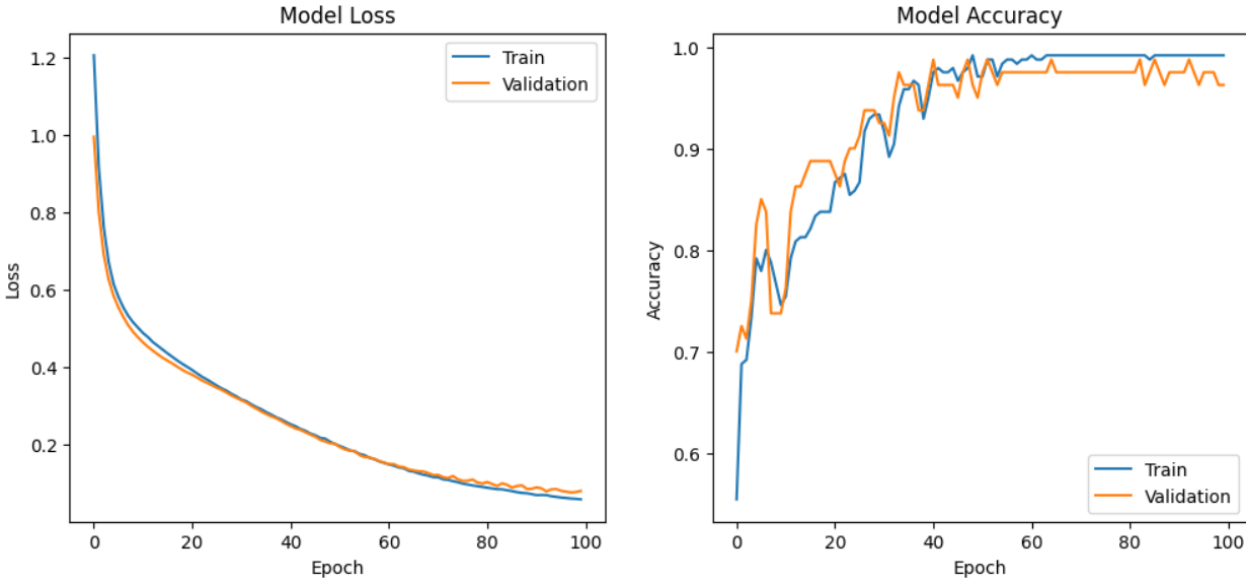
:metrics=['accuracy']

پارامتر متریک لیستی از معیارها را برای ارزیابی در طول آموزش و آزمایش مشخص می کند. در این مورد، دقت به عنوان متریک استفاده می شود.

دقت نسبت نمونه‌های طبقه‌بندی شده صحیح را در بین کل نمونه‌ها اندازه‌گیری می‌کند.

با نظارت بر دقت، می توانید ایده ای از عملکرد مدل خود از نظر پیش بینی صحیح کلاس های هدف داشته باشید.

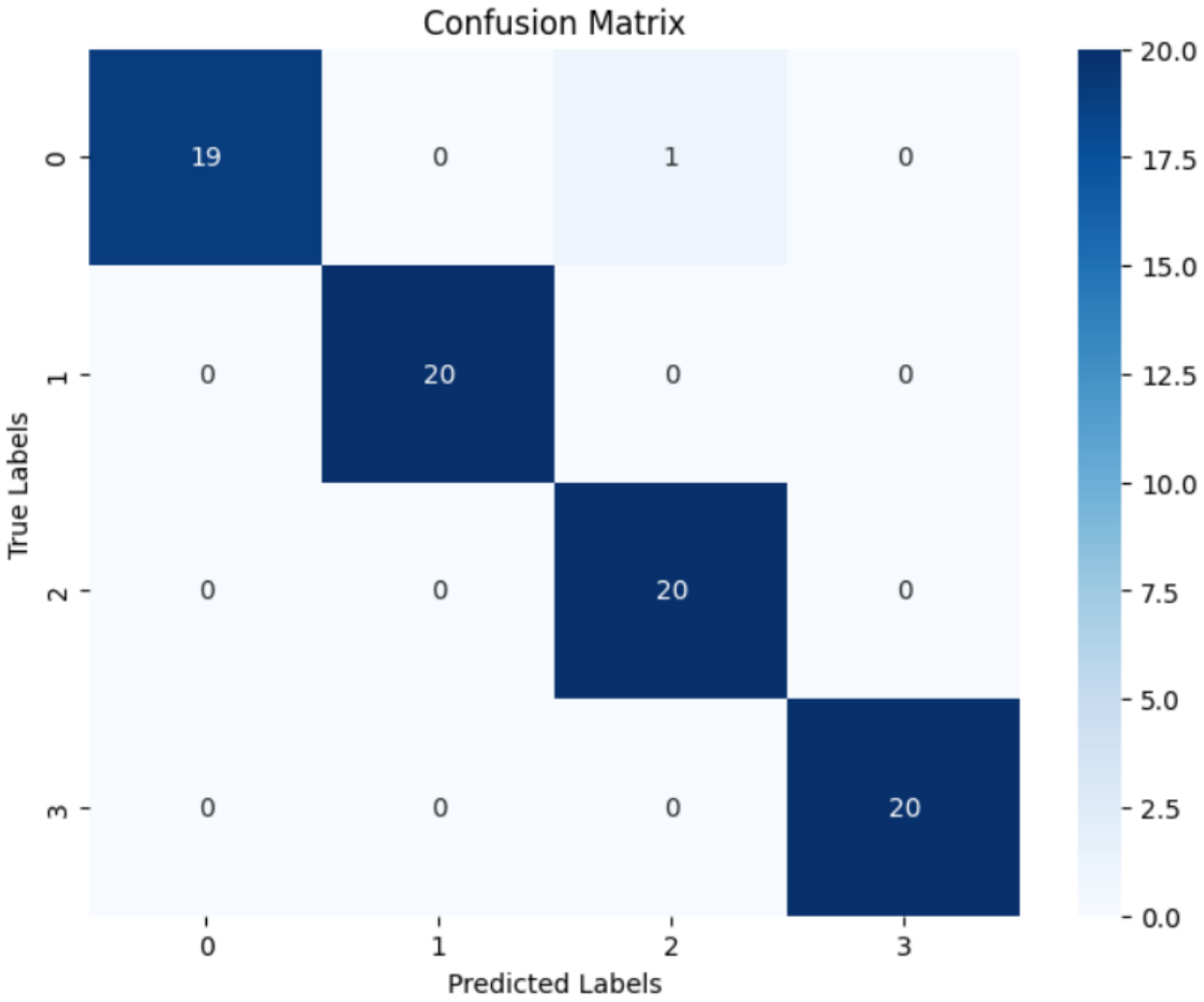
نتیجه بدین شکل میشود :



همینطور که مشاهده میکنید در loss function , داده های validation از train در آخر بیشتر شده که میتواند نشان دهنده این باشد که در واقع در همینتعداد epoch ها مدل باید آموزش ببیند تا overfitting اتفاق نیفتد.(چون مدل ما داده های train را دیده است طبیعی است که برای داده های دیده نشده باید loss function بیشتر باشد.)درواقع برای اینکه مدل ما Overfitting اتفاق بیفتد باید اختلاف زیادی بین تابع ضرر دو داده برقرار باشد که همینجور که میبینید اینطور نمیباشد.

همینطور که برای accuracy مشاهده میکنید طبیعی است که دقت برای داده های train بیشتر از validation باشد زیرا مدل بر روی داده های train آموزش دیده است و همچنین اختلاف زیادی در آخر دقت دو دسته داده وجود ندارد پس همه اینها نشان دهنده این است که مدل ما به خوبی آموزش دیده است و همچنین از یک جا به بعد داده های Train هم accuracy و هم loss function ثابتی را بدست آورده اند که نشان دهنده درست بودن مدل ما میباشد.(در واقع وزن ها ثابت میشوند .)

نتیجه بر روی داده های Test :



Classification Report:

	precision	recall	f1-score	support
0.0	1.00	0.95	0.97	20
1.0	1.00	1.00	1.00	20
2.0	0.95	1.00	0.98	20
3.0	1.00	1.00	1.00	20
accuracy			0.99	80
macro avg	0.99	0.99	0.99	80
weighted avg	0.99	0.99	0.99	80

خوب همینجور که در confusion matrix و classification report مشاهده میکنید دقت مدل بر روی داده های تست به میزان 99. میباشد که در واقع کم تر از دقت بر روی داده های آموزش است :(که باز هم نشان دهنده درست بودن مدل ما میباشد.)

```
0s 20ms/step - loss: 0.0601 - accuracy: 0.9917 - val_loss: 0.0777

0s 15ms/step - loss: 0.0588 - accuracy: 0.9917 - val_loss: 0.0806
```

با نگاه به classification report متوجه میشویم که دقت برای داده های کلاس ۲ کم است(یعنی از بین داده هایی که تخمین میزنه این را از بقیه اشتباه تر تخمین میزنه) به عنوان مثال ۲۱ داده را به عنوان کلاس ۲ تخمین زده ولی فقط ۲۰ داده را درست تخمین زده است که دقت 95. میدهد.

و همچنین recall برای داده های کلاس 0 از همه کم تر است.(در واقع درست تشخیص دادن داده ها از بین تعداد کل داده ای که به عنوان کلاس مد نظر میگیره.) به عنوان مثال ۲۰ داده وجود دارد که برای کلاس 0 است ولی فقط توانسته ۱۹ داده را تخمین بزند که recall در این قسمت 95. میشود.

در مجموع (f1-score) مدل ما داده های کلاس ۳و۱و۲و۰ را به ترتیب بهتر میتواند طبقه بندی کند. که در واقع f1-score میانگین هارمونیک دقت و recall است.

Marco و weighted avg نیز به ترتیب به معنای میانگین precision,recall,f1-score و weighted avg نیز به معنای این است که هر کدام را در تعداد داده هایشان ضرب کرده سپس میانگین بگیریم.

در کل مدل به خوبی توانسته است که داده ها را طبقه بندی کند.

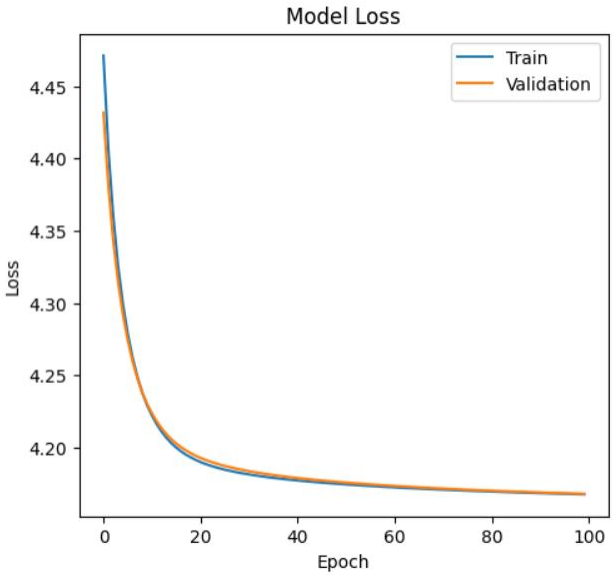
ج)

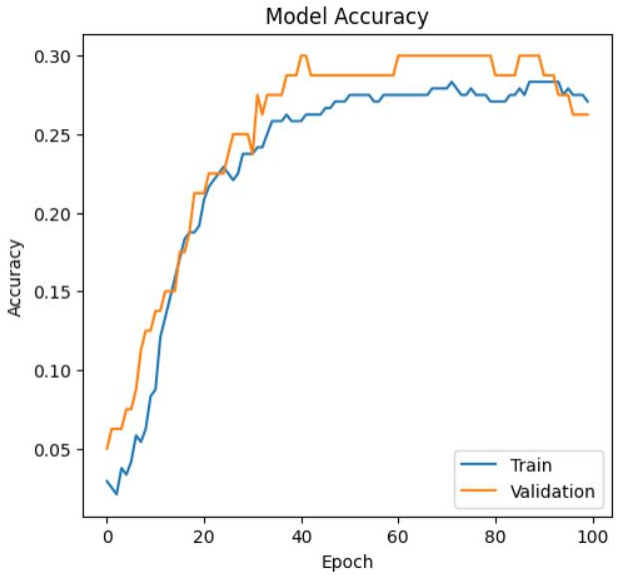
Optimizer و loss function جدید به ترتیب Adagrad و Kullback-Leibler Divergence Loss میباشند.

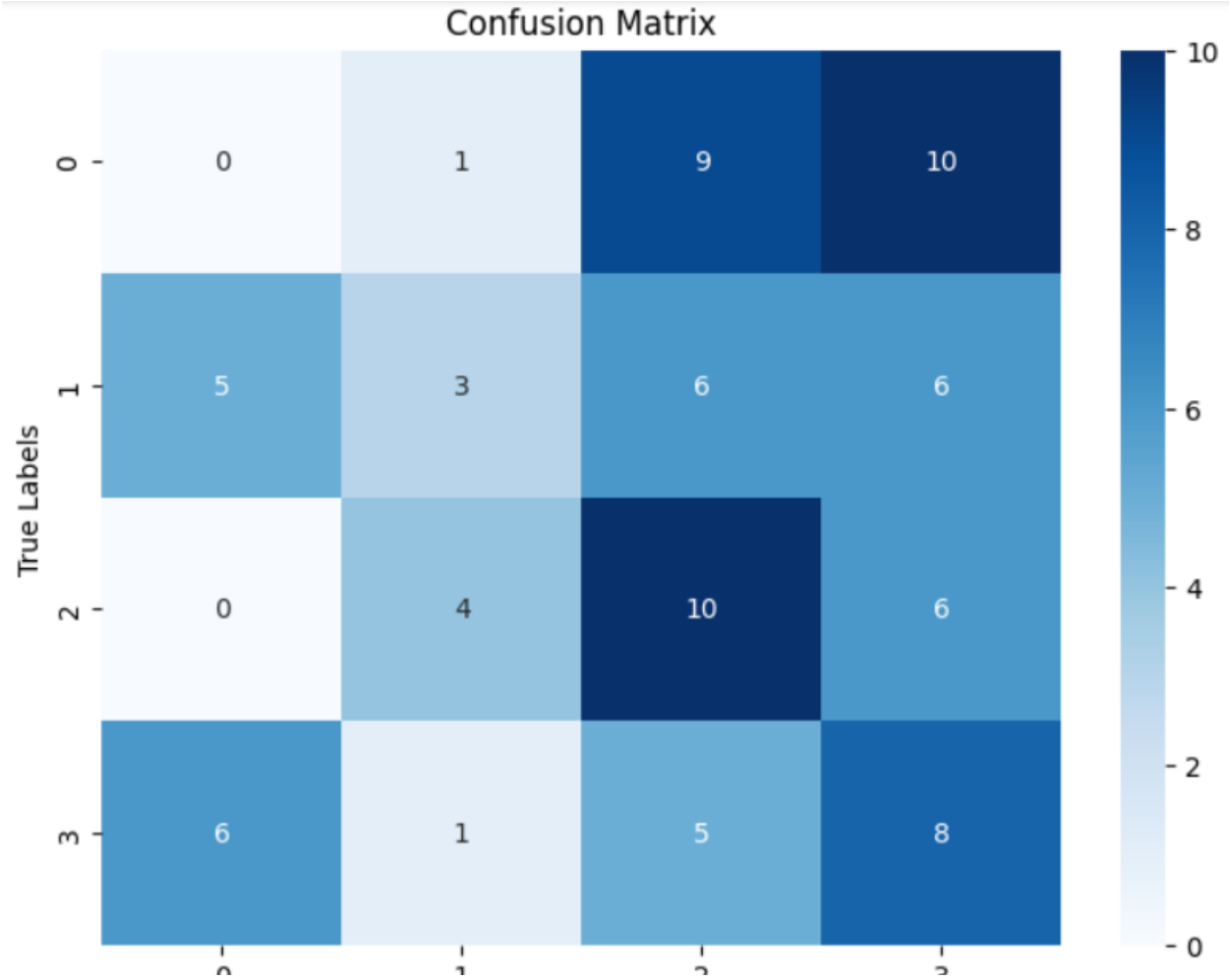
KLDvergence() : واگرایی Kullback-Leibler که به عنوان آنتروپی نسبی نیز شناخته می‌شود، نحوه واگرایی یک توزیع احتمال از توزیع احتمال دوم را اندازه‌گیری می‌کند. معمولاً در مدل‌های احتمالی برای تعیین کمیت تفاوت بین دو توزیع احتمال استفاده می‌شود.

Adagrad یک الگوریتم بهینه‌سازی نرخ یادگیری تطبیقی است که نرخ یادگیری را برای هر پارامتر بر اساس گرادیان‌های تاریخی تطبیق می‌دهد. این به ویژه برای داده های پراکنده مفید است.

نتیجه بدین شکل میشود :







Classification Report:

	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	20
1.0	0.33	0.15	0.21	20
2.0	0.33	0.50	0.40	20
3.0	0.27	0.40	0.32	20
accuracy			0.26	80
macro avg	0.23	0.26	0.23	80
weighted avg	0.23	0.26	0.23	80

خوب هینجور که از نتایج بالا معلوم است تاثیر این ۲ پارامتر بر روی درست کردن مدل به شدت زیاد است زیرا تمام معیار های درست بودن مدل که در بخش قبل راجع به آنها صحبت شد در اینجا نقض و همچنین کم تر شده است.

به عنوان مثال برای loss function همانطور که میبینید تابع loss ما از یک جا به بعد ثابت شده است که این نشان دهنده این میباشد که تقریبا وزن ها دیگر update نخواهند شد پس مدل ما بهتر نمیشود و همچنین تابع ضرر مقدار بیشتری نسبت به قبل دارد.

د)

K-Fold Cross-validation:

در K-Fold Cross-validation، مجموعه داده اصلی به طور تصادفی به تاهای با اندازه K تقسیم می شود.

این مدل K بار آموزش داده می شود، هر بار از K-1 برای آموزش و از باقی مانده برای اعتبار سنجی استفاده می شود.

این فرآیند امکان ارزیابی قوی تر مدل را فراهم می کند، زیرا هر نقطه داده دقیقاً یک بار در مجموعه اعتبار سنجی قرار می گیرد.

با این حال، ممکن است در مجموعه داده های نامتعادل عملکرد خوبی نداشته باشد زیرا توزیع یکسان کلاس ها را در هر قسمت تضمین نمی کند.

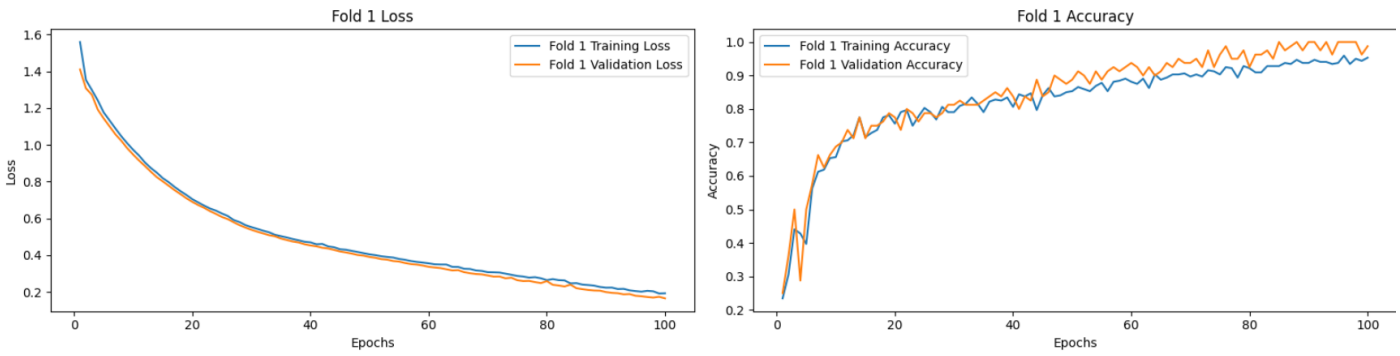
Stratified K-Fold Cross-validation:

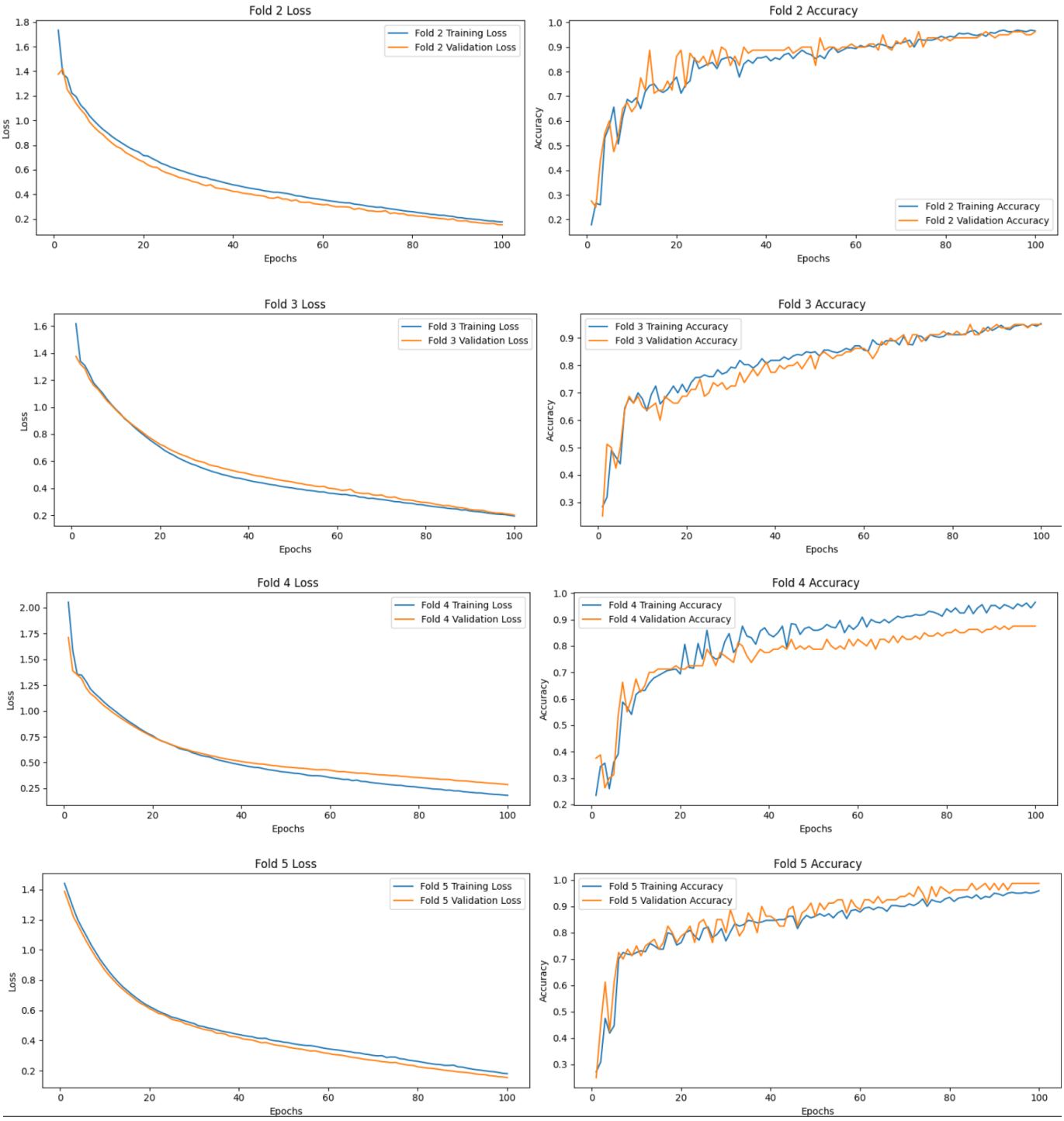
Stratified K-Fold Cross-validation توسعه ای از اعتبارسنجی متقاطع K-Fold است که تضمین می کند هر فولد دارای توزیع کلاسی مشابه با مجموعه داده اصلی است.این به ویژه برای مسائل طبقه بندی با توزیع کلاس نامتعادل مفید است.این روش تضمین می‌کند که هر فولد نشان‌دهنده توزیع کلی کلاس‌ها در مجموعه داده است، که می‌تواند به ارزیابی مدل مطمئن‌تر منجر شود.

حال بیایید یکی از این روش ها را انتخاب کرده و پیاده سازی کنیم:

برای کار داده شده، از آنجایی که ما با یک مشکل طبقه بندی سروکار داریم، عاقلانه است که از اعتبار سنجی متقاطع K-Fold Stratified استفاده کنیم. این تضمین می‌کند که هر فولد توزیع کلاسی مشابه مجموعه داده اصلی را حفظ می‌کند و ارزیابی دقیق‌تری از عملکرد مدل ارائه می‌کند.

نتیجه مدل های ما بدین شکل میشود (ما داده ها را به ۵ دسته تقسیم بندی کرده ایم) :





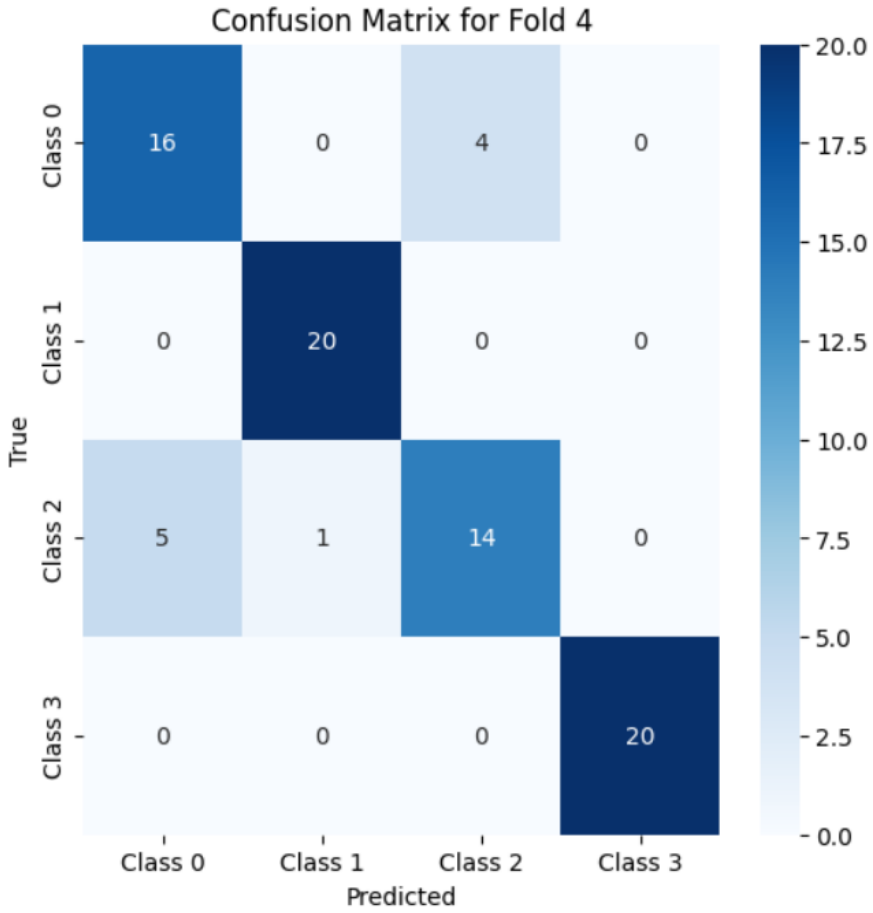
خوب همینجور که مشاهده میکنید دسته 4 بهتر از بقیه هستند.(overfitting اتفاق نیفتاده است.)بقیه داده ها از آنجایی که validation بهتر از train بوده یا convergeکرده اند درست نیستند.(مدل ها)البته برای مدل ها میتوان تعداد epoch را افزایش داد اینگونه باعث میشه که تابع loss کم تر بشود.

همینجور که میبینید loss function به جز در مدل ۴ در بقیه یا کم تر است یا برابر است که این نشان میدهد که مدل ما به درستی آموزش ندیده است.

تاثیر داده های test بر روی مدل :

Fold 4 - Classification Report:

	precision	recall	f1-score	support
Class 0	0.76	0.80	0.78	20
Class 1	0.95	1.00	0.98	20
Class 2	0.78	0.70	0.74	20
Class 3	1.00	1.00	1.00	20
accuracy			0.88	80
macro avg	0.87	0.88	0.87	80
weighted avg	0.87	0.88	0.87	80

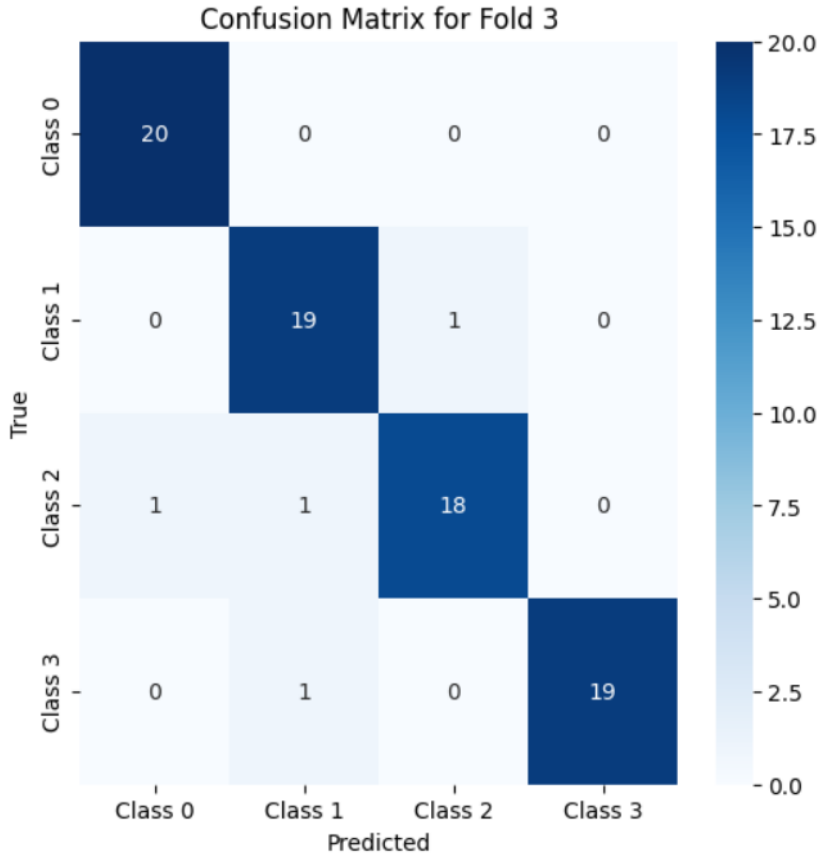


خوب همینطور که میبینید مدل ۴ نسبت به مدل قبلی که بدست آورده ایم دقت کم تری دارد ولی بهتر از بقیه مدل های این بخش میباشد.برای اینکه این مدل بهتر شود میتوان تعداد epoch ها را به طور مثال افزایش داد اگر دقت مدل افزایش و مقدار loss داده های validation (به طوری که هنوز یک مقدار بیشتر از داده های train باشد) کم تر شود مدل ما قابل قبول تر میشود.

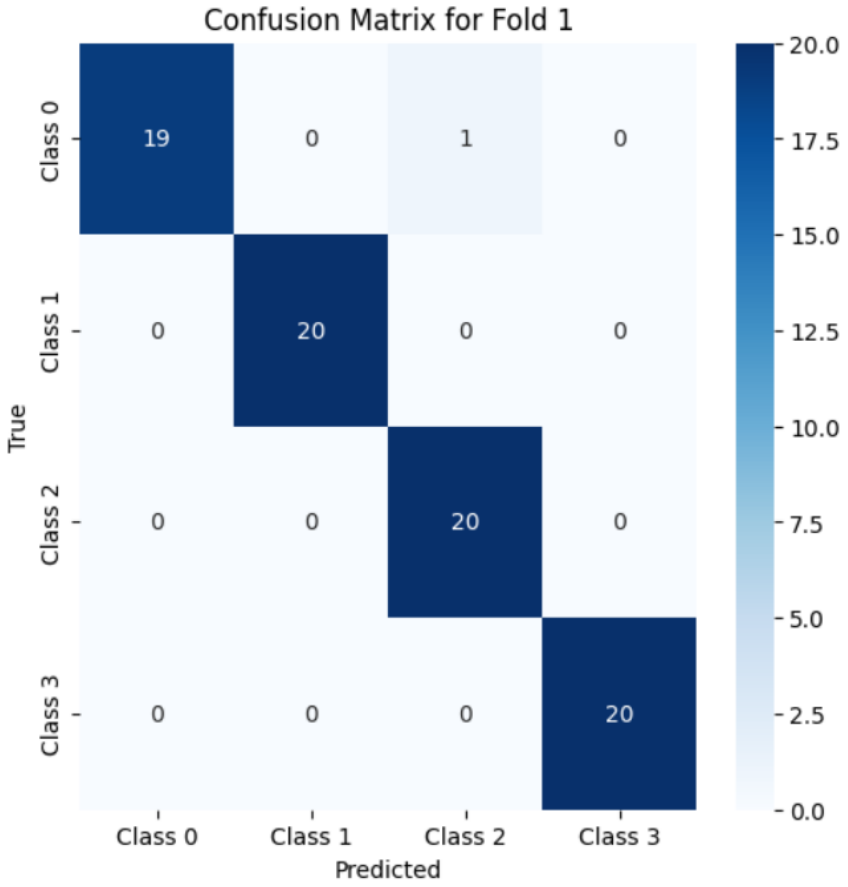
نتیجه بقیه مدل ها :

Fold 3 - Classification Report:

	precision	recall	f1-score	support
Class 0	0.95	1.00	0.98	20
Class 1	0.90	0.95	0.93	20
Class 2	0.95	0.90	0.92	20
Class 3	1.00	0.95	0.97	20
accuracy			0.95	80
macro avg	0.95	0.95	0.95	80
weighted avg	0.95	0.95	0.95	80

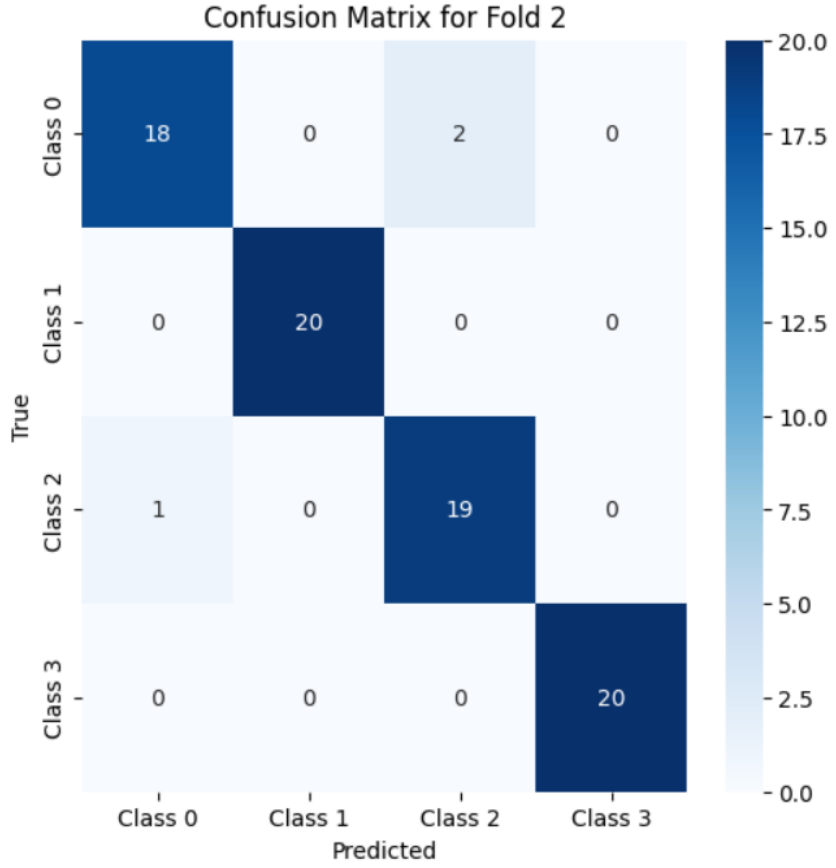


```
Fold 1 - Classification Report:
              precision    recall  f1-score   support

     Class 0       1.00      0.95      0.97        20
     Class 1       1.00      1.00      1.00        20
     Class 2       0.95      1.00      0.98        20
     Class 3       1.00      1.00      1.00        20

    accuracy                           0.99        80
   macro avg       0.99      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80
```

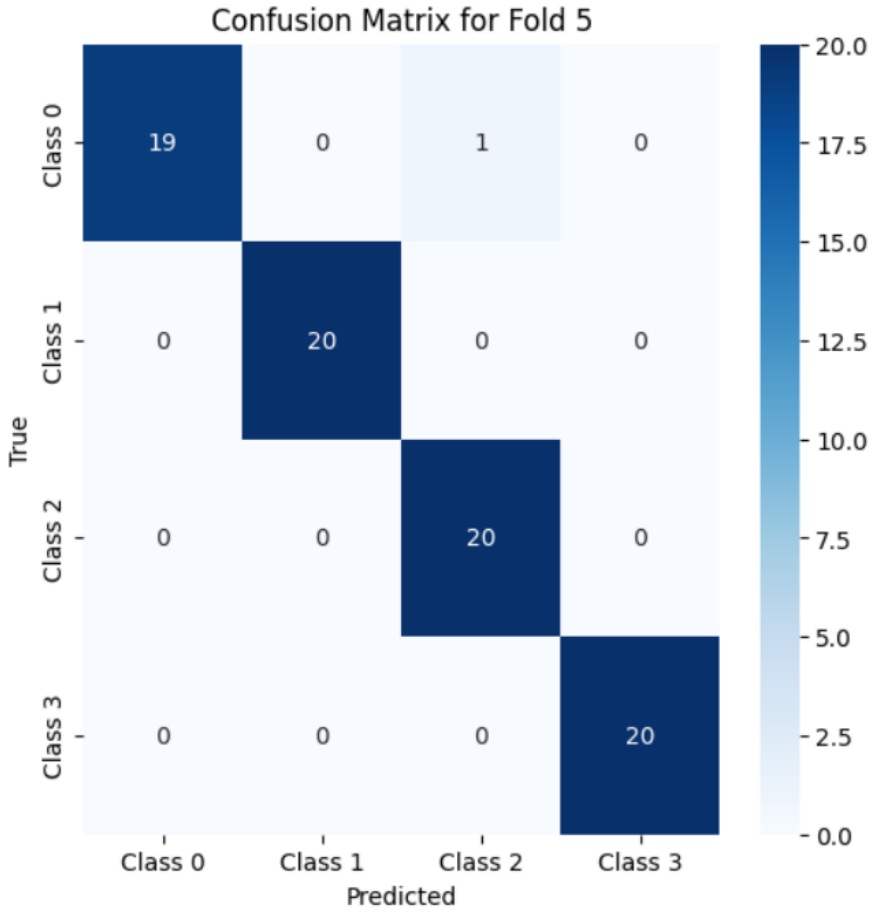


Fold 2 - Classification Report:

	precision	recall	f1-score	support
Class 0	0.95	0.90	0.92	20
Class 1	1.00	1.00	1.00	20
Class 2	0.90	0.95	0.93	20
Class 3	1.00	1.00	1.00	20
accuracy			0.96	80
macro avg	0.96	0.96	0.96	80
weighted avg	0.96	0.96	0.96	80



```
Fold 5 - Classification Report:
              precision    recall  f1-score   support

     Class 0       1.00      0.95      0.97        20
     Class 1       1.00      1.00      1.00        20
     Class 2       0.95      1.00      0.98        20
     Class 3       1.00      1.00      1.00        20

    accuracy                           0.99        80
   macro avg       0.99      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80
```



با اینکه بقیه مدل ها accuracy خیلی خوبی دارند ولی تابع ضرر آنها نشان دهنده اینست که مدل آنها به نادرستی آموزش دیده است.

یک راه برای بهتر کردن این مدل ها افزایش تعداد epoch ها تا Loss fuction برای داده های اعتبار سنجی بیشتر از داده های train شود و همچنین میتوان تعداد Fold ها را کم تر کرد تا داده های بیشتری برای آموزش مدل در دسترس باشد تا مدل بهتر آموزش ببیند.

گزارش کد K fold stratified :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam


# Define the number of splits for Stratified K-Fold
n_splits = 5
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=64)

# Store confusion matrices and classification reports for later analysis
confusion_matrices = []
classification_reports = []

# Lists to store histories of training and validation metrics for each
fold
all_train_losses = []
all_val_losses = []
all_train_accuracies = []
all_val_accuracies = []

# Stratified K-Fold Cross-validation
for fold, (train_index, val_index) in enumerate(skf.split(features,
labels)):
    print(f"Training fold {fold+1}/{n_splits}")
```

```
    # Split data into training and validation sets
    features_train, features_val = features[train_index],
features[val_index]
    labels_train, labels_val = labels[train_index], labels[val_index]

    # Define the MLP model
    model = Sequential([
        Dense(64, activation='elu',
input_shape=(features_train.shape[1],)),
        Dense(32, activation='elu'),
        Dense(4, activation='softmax')  # 4 classes for classification
    ])

    # Compile the model
    model.compile(optimizer=Adam(),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    # Train the model
    history = model.fit(features_train, labels_train, epochs=100,
batch_size=64,
                        validation_data=(features_val, labels_val),
verbose=0)

    # Append fold history
    all_train_losses.append(history.history['loss'])
    all_val_losses.append(history.history['val_loss'])
    all_train_accuracies.append(history.history['accuracy'])
    all_val_accuracies.append(history.history['val_accuracy'])

    # Generate confusion matrix and classification report for this fold
    predictions = model.predict(features_val)
    predicted_labels = np.argmax(predictions, axis=1)
    confusion_matrices.append(confusion_matrix(labels_val,
predicted_labels))
    classification_reports.append(classification_report(labels_val,
predicted_labels, target_names=['Class 0', 'Class 1', 'Class 2', 'Class
3']))
```

اول از همه کتابخانه های مورد نظر را وارد میکنیم.

n_splits: تعداد فولدها برای اعتبارسنجی متقابل.

skf: نمونه ای از StratifiedKFold که تضمین می کند که هر فولد نسبت یکسانی از هر کلاس دارد.

confusion_matrices and classification_reports: برای ذخیره معیارهای عملکرد برای هر فولد.

all_train_losses, all_val_losses, all_train_accuracies, and all_val_accuracies: برای ذخیره معیارهای آموزشی و اعتبارسنجی برای هر فولد.

skf.split (ویژگی ها، برچسب ها): داده ها را به شاخص های آموزشی و اعتبار سنجی تقسیم می کند.

features_train، features_val، labels_train، labels_val: زیر مجموعه‌های مجموعه داده برای فولد فعلی.

Sequential(): یک پشته خطی از لایه ها را راه اندازی می کند.

Dense(64, activation='elu', input_shape=(features_train.shape[1],)): اولین لایه پنهان با 64 واحد و فعال سازی ELU.

متراکم (32، 'activation='elu): دومین لایه پنهان با 32 واحد و فعال سازی ELU.

متراکم (4، 'activation='softmax): لایه خروجی با 4 واحد (یکی برای هر کلاس) و فعال سازی softmax برای طبقه بندی.

model.compile(): مدل را برای آموزش با بهینه ساز Adam، تلفات متقابل آنتروپی طبقه بندی شده و متریک دقت پیکربندی می کند.

model.fit(): مدل را روی داده های آموزشی آموزش می دهد و روی داده های اعتبارسنجی ارزیابی می کند. (verbose=0)

history.history: شامل فقدان آموزش و اعتبارسنجی و دقت برای هر دوره است.

np.argmax(predictions, axis=1): خروجی های softmax را به برچسب های کلاس پیش بینی شده تبدیل می کند.

confusion_matrix() و classification_report(): ماتریس های سردرگمی و گزارش های طبقه بندی را برای هر فولد ایجاد کرده و آنها را ذخیره کنید.

سپس loss function برای هر کدام را بدین شکل نشان داده ایم :

```
# Plot loss and accuracy graphs for each fold
epochs = range(1, 101)
plt.figure(figsize=(16, 20))

for fold in range(len(all_train_losses)):
    plt.subplot(len(all_train_losses), 2, 2*fold+1)
    plt.plot(epochs, all_train_losses[fold], label=f'Fold {fold+1}
Training Loss')
    plt.plot(epochs, all_val_losses[fold], label=f'Fold {fold+1}
Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title(f'Fold {fold+1} Loss')
    plt.legend()

    plt.subplot(len(all_train_losses), 2, 2*fold+2)
    plt.plot(epochs, all_train_accuracies[fold], label=f'Fold {fold+1}
Training Accuracy')
    plt.plot(epochs, all_val_accuracies[fold], label=f'Fold {fold+1}
Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title(f'Fold {fold+1} Accuracy')
    plt.legend()

plt.tight_layout()
plt.show()
```

در حقیقت حلقه درست کرده ایم و تمام loss ها برای داده های validation و train را نمایش داده ایم.

همین کار را نیز برای confusion matrix و classification report انجام داده ایم:

```
# Analyze confusion matrices and classification reports
for i, (cm, report) in enumerate(zip(confusion_matrices,
classification_reports)):

    print(f"Fold {i+1} - Classification Report:")
    print(report)


    # Plot confusion matrix
    plt.figure(figsize=(6, 6))
```

```
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Class
0', 'Class 1', 'Class 2', 'Class 3'], yticklabels=['Class 0', 'Class 1',
'Class 2', 'Class 3'])
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title(f'Confusion Matrix for Fold {i+1}')
    plt.show()
```

سوال ۳)الف) داده مورد استفاده در سوال داده های دارو میباشد.

ما از دستور train_test_split برای تقسیم بندی داده ها به train,test استفاده کردیم :

```
X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.3, random_state=64)
```

همینجور که میبینید اندازه داده های تست ۳۰ درصد کل داده ها میباشد.

یک روش جایگزین و بالقوه بهتر، Stratified Sampling است، به خصوص اگر کلاس‌ها نامتعادل باشند. این روش تضمین می کند که نسبت هر کلاس در مجموعه های آموزشی و آزمایشی مشابه نسبت موجود در مجموعه داده اصلی است. این را می توان با استفاده از روش StratifiedShuffleSplit در scikit-learn انجام داد:

```
from sklearn.model_selection import StratifiedShuffleSplit

sss = StratifiedShuffleSplit(n_splits=1, test_size=0.3, random_state=64)
for train_index, test_index in sss.split(features, labels):
    X_train, X_test = features.iloc[train_index],
features.iloc[test_index]
    y_train, y_test = labels.iloc[train_index], labels.iloc[test_index]
```

(البته ما این کار را برای درست کردن مدل انجام نداده ایم.)

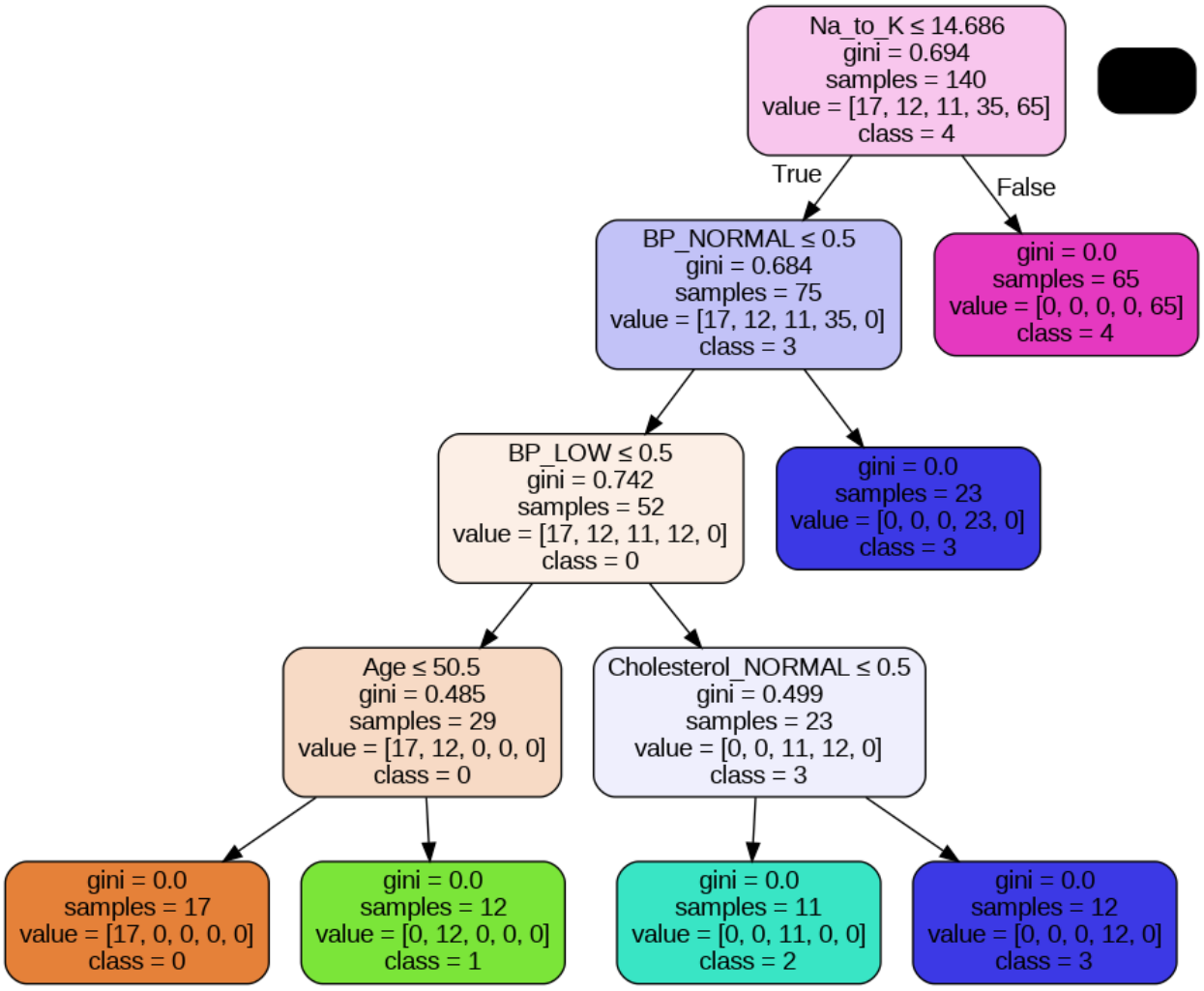
برای درست کردن مدل ما ابتدا باید ستون هایی که عددی نیستند را به نحوی تبدیل میکنیم تا مدل بتواند آنها را تشخیص دهد. در واقع با one-hot encoding ما ویژگی ها را بدین شکل میکنیم :

	Age	Na_to_K	Drug	Sex_M	BP_LOW	BP_NORMAL	Cholesterol_NORMAL
0	23	25.355	4	False	False	False	False
1	47	13.093	2	True	True	False	False
2	47	10.114	2	True	True	False	False
3	28	7.798	3	False	False	True	False
4	61	18.043	4	False	True	False	False
..	...	...	...	...	...	...	...
195	56	11.567	2	False	True	False	False
196	16	12.006	2	True	True	False	False
197	52	9.894	3	True	False	True	False
198	23	14.020	3	True	False	True	True
199	40	11.349	3	False	True	False	True

[200 rows x 7 columns]

همینطور که میبینید کلاس های ما از ۰ تا ۴ میباشند.BP به ۲ دسته BP_LOW و BP_NORMAL تقسیم شده است در واقع اگر ۲ ستون درست شده BP , FALSE باشند در واقع BP برابر HIGH میباشد به همین ترتیب برای بقیه ستون ها میتوان استدلال کرد.

بعد از درست کردن مدل نتیجه ای که بدست می آوریم بدین شکل میباشد :



در واقع عکسی که در بالا میبینید درخت تصمیم ما میباشد که بر روی داده ها درست شده است.تحلیل آن بدین صورت است :

Root Node

ویژگی: Na_to_K (در واقع این نشان دهنده feature ای میباشد که برای دسته بندی داده ها استفاده میکند.)

آستانه: 14.686 ≥ (گره ریشه مجموعه داده را بر اساس ویژگی "Na_to_K" تقسیم می کند. اگر مقدار کمتر یا مساوی 14.686 باشد، به Child سمت چپ می رود. در غیر این صورت، به child مناسب می رسد.)

شاخص جینی: 0.694 (شاخص جینی ناخالصی را اندازه گیری می کند و مقادیر پایین تر نشان دهنده گره های خالص تر (همگن تر) است.)

نمونه: 140 (در واقع این تعداد کل نمونه ها میباشد که در حال بررسی میباشد.)

ارزش: [17، 12، 11، 35، 65] (این تعداد داده هایی است که مربوط به هر کلاس است.)

کلاس: 4 (متداول ترین کلاس در این گره)

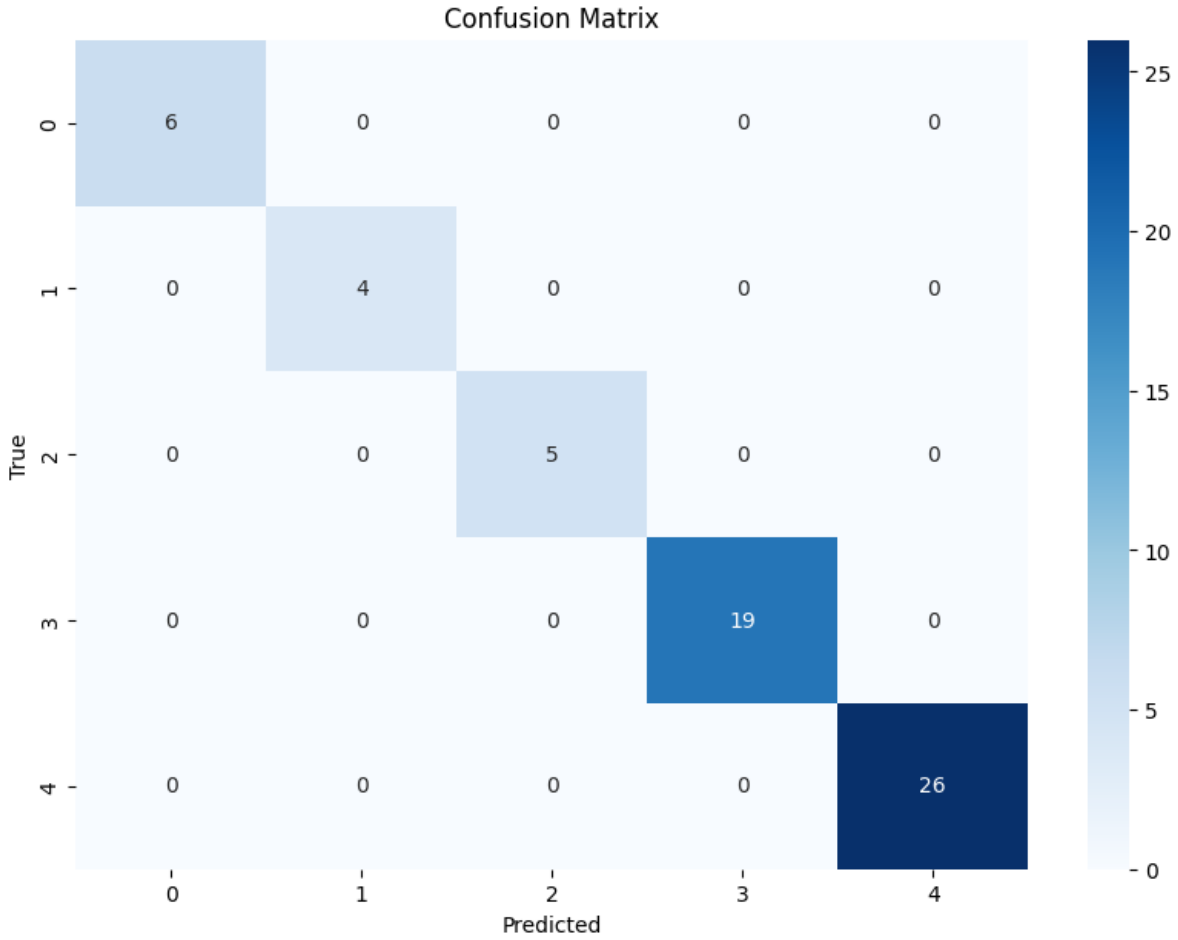
وقتی سمت راست برود تمام داده های آن در واقع نشان دهنده کلاس ۴ میباشد و همچنین تمام داده های کلاس ۴ در همان ابتدا جدا شده اند.

بقیه درخت نیز بدین شکل میباشد که BP_NORMAL≤5. مینویسد بدین معنی است که این ویژگی در اینجا FASLE است.(TRUE=1 , FALSE=0)

معنی کلاس ها : drug Y = 4 و drug X = 3 و drug A = 0 و drug B = 1 و drug C = 2

ب)

Confusion matrix و classification report که شامل شه شاخصه ارزیابی میباشد بدین صورت است :



Classification Report:	precision	recall	f1-score	support
0	1.00	1.00	1.00	6
1	1.00	1.00	1.00	4
2	1.00	1.00	1.00	5
3	1.00	1.00	1.00	19
4	1.00	1.00	1.00	26
accuracy			1.00	60
macro avg	1.00	1.00	1.00	60
weighted avg	1.00	1.00	1.00	60

خوب همینطور که مشاهده میکنید درخت ما توانسته است که همه داده های تست را به درستی شناسایی کند این موضوع از آنجایی که داده هایی که در اختیار ما هستند کم میباشد و مسئله از جهاتی ساده میباشد(مدل به صورت کامل بر روی داده ها آموزش داده شده) میتواند اتفاق بیفتد.

ما در اینجا ۲ پارامتر را تغییر میدهیم : max_depth و min_samples_split

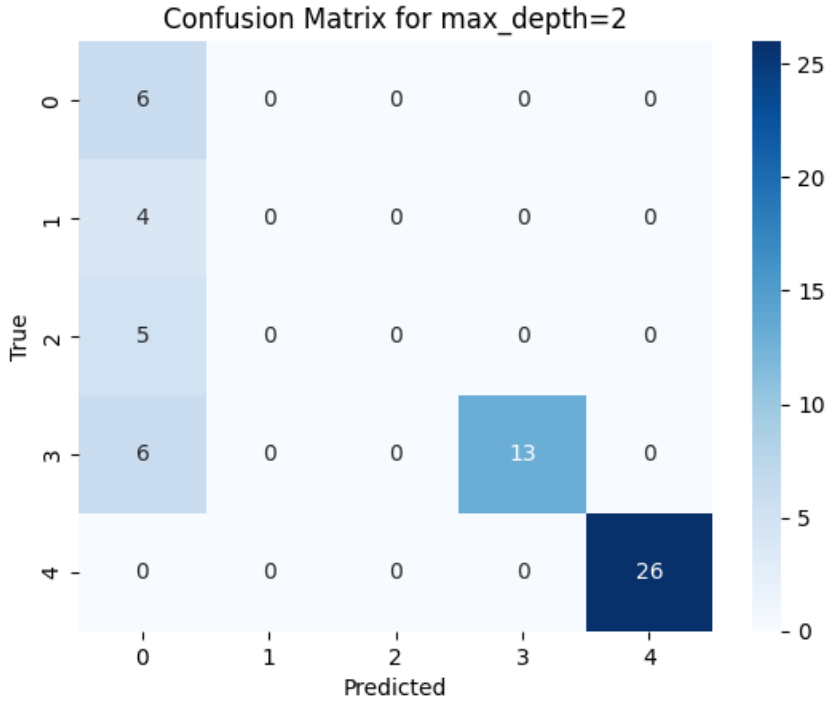
کارایی آنها به ترتیب بدین شکل میباشد :

max_depth: این هایپرپارامتر حداکثر عمق درخت را کنترل می کند. محدود کردن عمق درخت می تواند با حصول اطمینان از اینکه درخت بیش از حد پیچیده نمی شود و با داده های آموزشی خیلی نزدیک نمی شود، از برازش بیش از حد جلوگیری می کند.

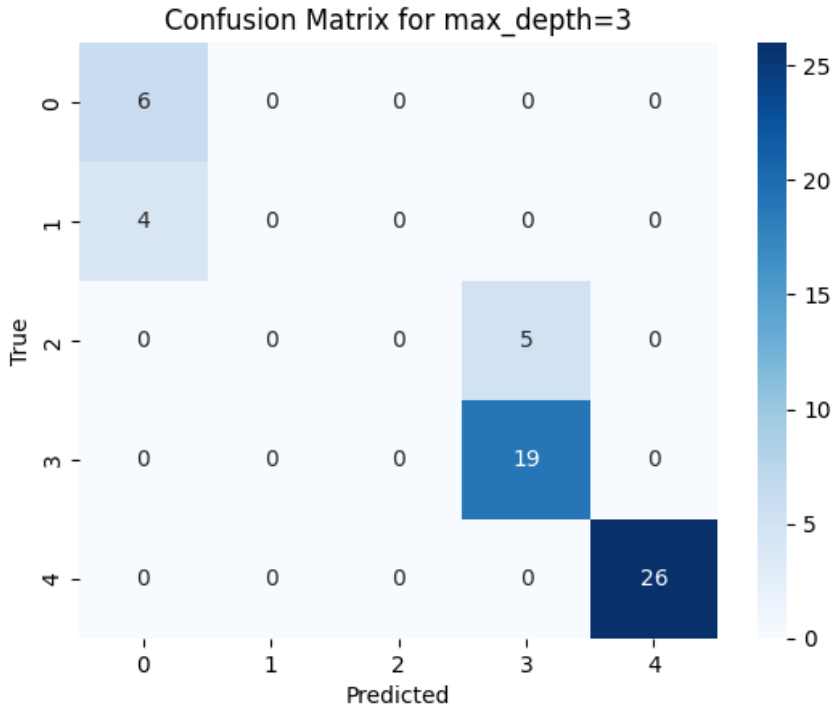
min_samples_split: این هایپرپارامتر حداقل تعداد نمونه های مورد نیاز برای تقسیم یک گره داخلی را مشخص می کند. افزایش این مقدار می تواند منجر به درختی شود که حساسیت کمتری به نویز در داده های آموزشی دارد در واقع منظور این است که شاخه هایی درست نشود که تعداد داده های کم را دارا باشد و این موضوع باعث میشود مدل ما احتمال overfit شدن آن کم شود.

نتیجه تغییر max_depth :

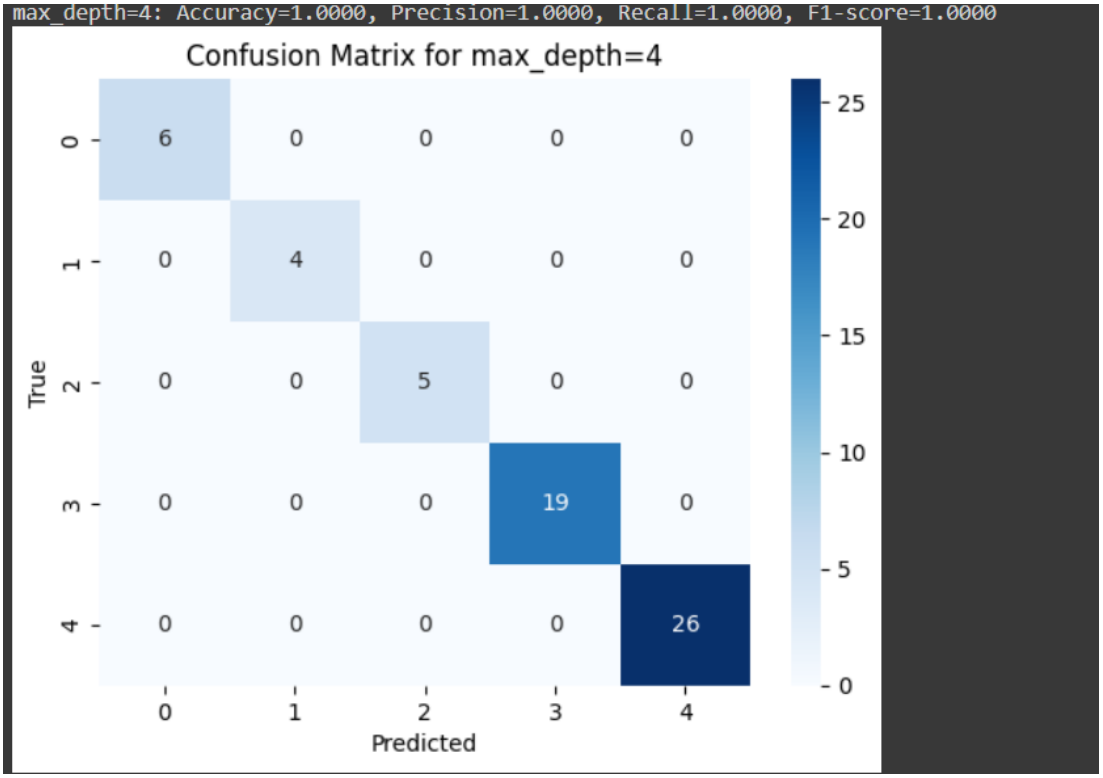
```
max_depth=2: Accuracy=0.7500, Precision=0.7786, Recall=0.7500, F1-score=0.7351
```

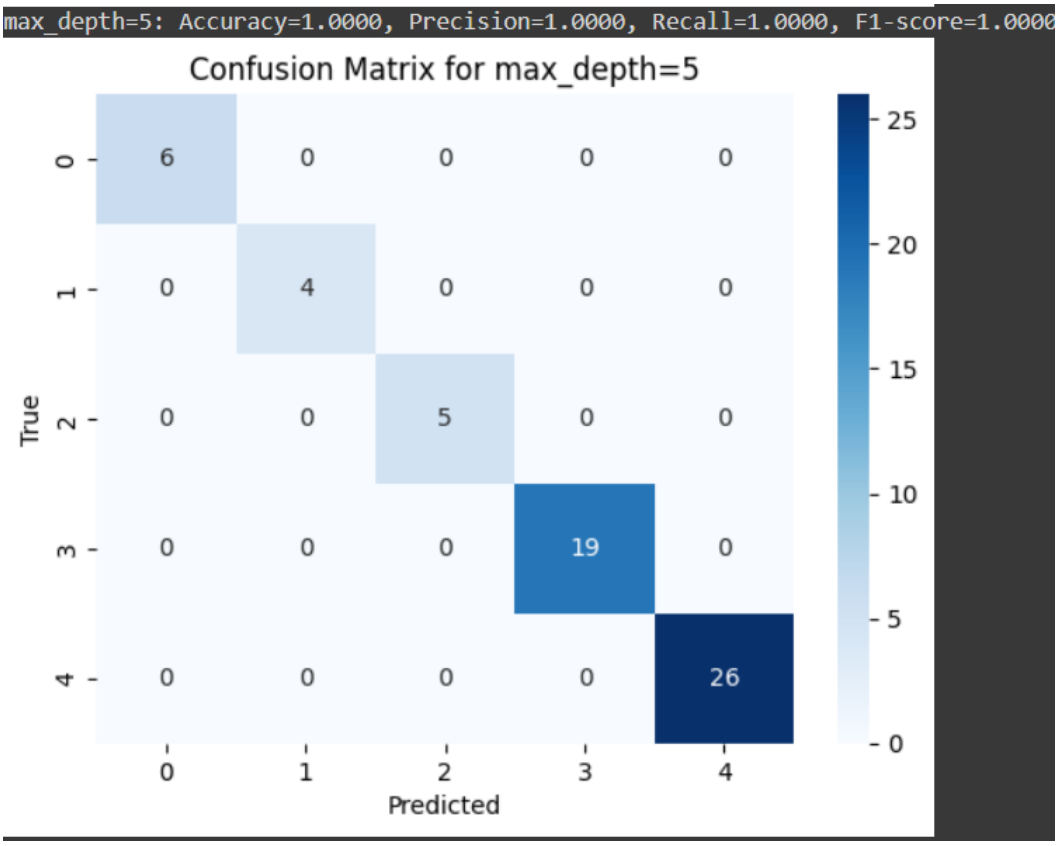



```
Effect of max_depth:
max_depth=3: Accuracy=0.8500, Precision=0.7440, Recall=0.8500, F1-score=0.7882
```




```
max_depth=4: Accuracy=1.0000, Precision=1.0000, Recall=1.0000, F1-score=1.0000
```

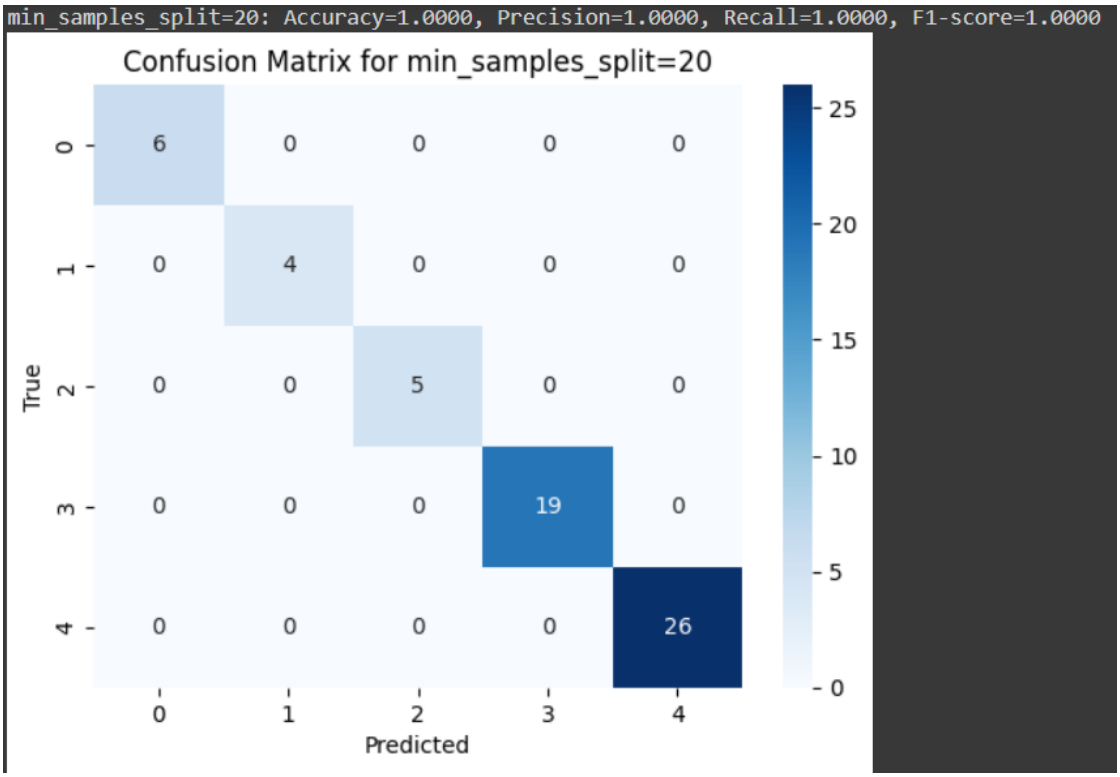


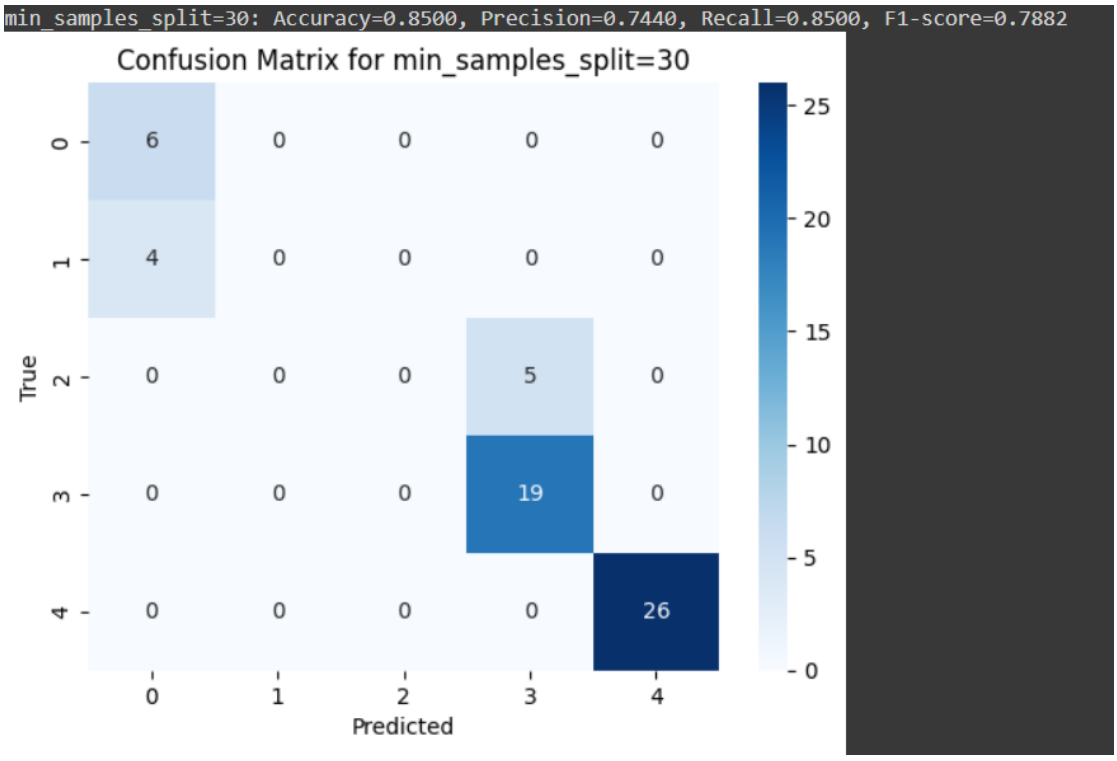


خوب همینطور که مشاهده میکنید در واقع با کم کردن max_depth (تعداد برابر ۳) درخت کارایی خود را بر روی داده های test از دست میدهد و عملکرد آن کم تر میشود.همینطور که در درختی که در بخش قبل رسم کردیم دیدیم کلاس ۴ در عمق دوم درخت ما جدا شد به همین دلیل نیز در اینجا تمام داده ها را توانسته به درستی تشخیص دهد.هر چقدر عمق درخت تصمیم خود را افزایش دهیم بیشتر آموزش میبینند(احتما overfitting نیز در این بین وجود دارد.) و در نتیجه بهتر میتواند بر روی داده های تست عمل کند.طبق نتیجه بالا بهترین عمق درخت ۴ میباشد.

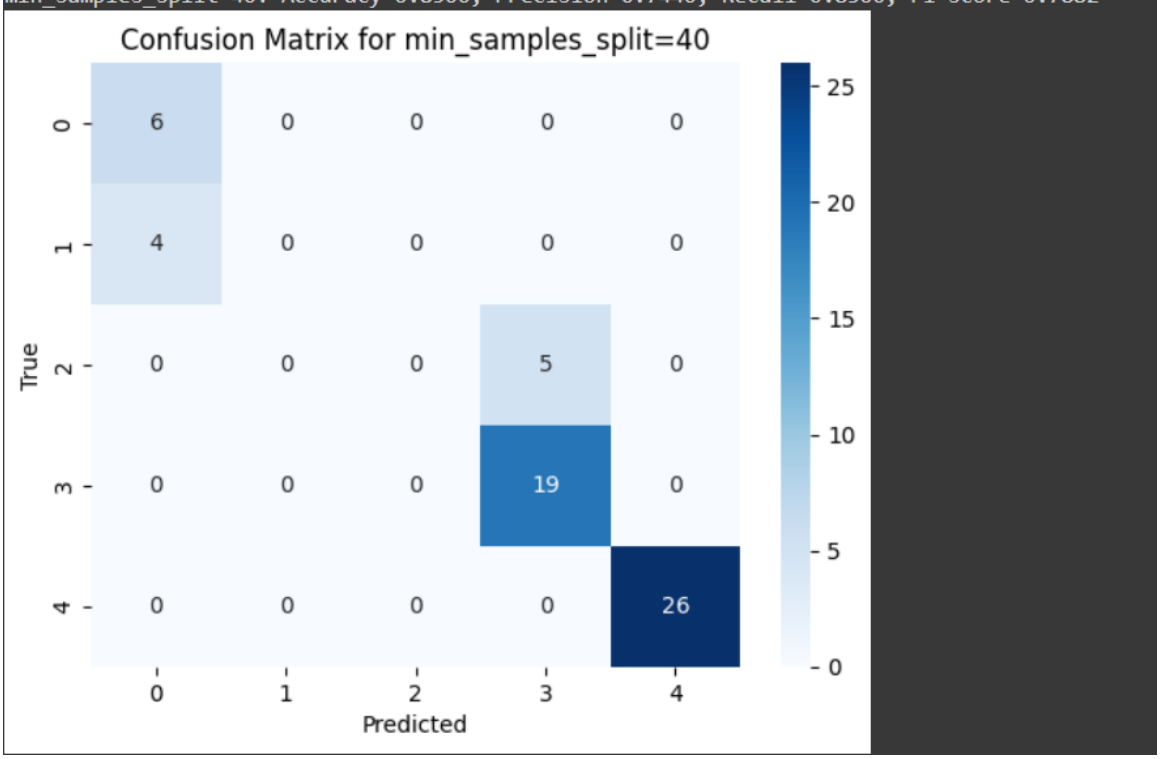
همینطور که میبینید بیشترین کلاسی که توانسته به درستی پیش بینی کند کلاس ۴ و سپس ۳ میباشند دلیل دیگر این ماجرا این است که بیشترین داده موجود بعد از کلاس ۴ این کلاس است و در همان لایه های اولیه میتوان دید که بیشترین داده متعلق به این کلاس است برای همین توانسته به درستی تشخیص دهد.

نتیجه تغییر min_samples_split :

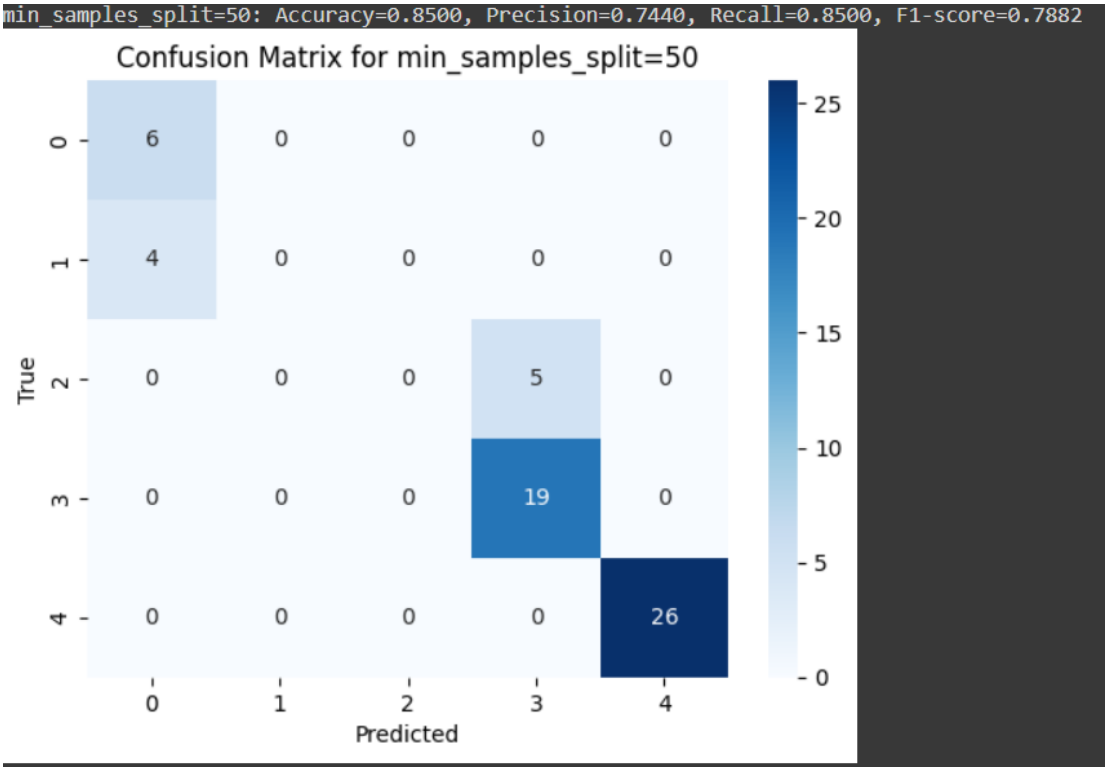




```
min_samples_split=40: Accuracy=0.8500, Precision=0.7440, Recall=0.8500, F1-score=0.7882
```




```
min_samples_split=50: Accuracy=0.8500, Precision=0.7440, Recall=0.8500, F1-score=0.7882
```



خوب همینجور که میبینید با افزایش تعداد حداقل داده برای تصمیم گیری دقت مدل ما کاهش پیدا میکند این موضوع واضح است زیرا تعداد داده مورد نیاز برای تضمیم گیری در لایه ۴ کم تر از ۳۰ و بیشتر از ۲۰ میباشد برای همین بهترین مدل برای ۲۰ و بقیه نیز مانند یک دیگر شده اند.

نکته کلی : یکی از دلایلی که نمیتوان گفت که مدل ما در حالت کلی overfitting دارد زیرا برای داده های test توانسته به خوبی عمل کند.(همچنین تعداد داده های کم نیز میتواند مستعد یادگیری خیلی خوب داده های train باشد و نمیتوان گفت که به جای الگو ی بین داده ها نویز داده های آموزشی را یاد گرفته باشد.)

ج)

Random Forest و AdaBoost دو روش یادگیری گروهی محبوب هستند که نتایج طبقه‌بندی را با ترکیب پیش‌بینی‌های چندین مدل فردی بهبود می‌بخشند. هر روش رویکرد منحصر به فرد خود را برای ساخت و جمع‌آوری این مدل‌های فردی دارد که منجر به افزایش دقت و استحکام در مقایسه با مدل‌های منفرد مانند درخت‌های تصمیم می‌شود. در اینجا توضیحی درباره نحوه عملکرد هر روش و چگونگی بهبود نتایج ارائه شده است:

Random Forest (جنگل تصادفی)

بررسی اجمالی:

Random Forest یک روش مجموعه ای است که چندین درخت تصمیم می سازد و پیش بینی های آنها را برای بهبود دقت و کنترل بیش از حد برازش(overfitting) ادغام می کند.

چگونه کار می کند:

Bootstrap Aggregation (Bagging): Random Forest از bagging استفاده می کند، که در آن زیر مجموعه های متعددی از داده های آموزشی با نمونه گیری تصادفی با جایگزینی ایجاد می شود. هر زیر مجموعه برای آموزش یک درخت تصمیم جداگانه استفاده می شود.

انتخاب ویژگی تصادفی: در هر تقسیم در درخت تصمیم، یک زیرمجموعه تصادفی از ویژگی ها انتخاب می شود که از بین آنها بهترین ویژگی انتخاب می شود. این تصادفی بودن درختان منفرد را تضمین می کند که همبستگی کمتری دارند، در نتیجه واریانس را کاهش می دهد.

تجمیع(aggregation): پیش‌بینی نهایی با میانگین‌گیری پیش‌بینی‌های همه درختان منفرد (برای رگرسیون) یا با رأی اکثریت (برای طبقه‌بندی) انجام می‌شود.

مزایا:

کاهش بیش از حد برازش: تجمع چندین درخت در مقایسه با یک درخت تصمیم گیری، خطر بیش از حد برازش را کاهش می دهد.

استحکام در برابر نویز: Random Forest به دلیل میانگین درختان متعدد، حساسیت کمتری به نویز در داده های آموزشی دارد.

مدیریت ابعاد بالا: انتخاب تصادفی ویژگی در هر تقسیم باعث می شود که در مدیریت مجموعه داده ها با تعداد زیادی ویژگی موثر باشد.

AdaBoost

بررسی اجمالی:

AdaBoost (تقویت تطبیقی) یک روش مجموعه‌ای است که چندین طبقه‌بندی ضعیف (معمولاً کنده‌های تصمیم، که درخت‌های تصمیم تک سطحی هستند) را برای تشکیل یک طبقه‌بندی قوی ترکیب می‌کند. بر روی نمونه‌های طبقه‌بندی اشتباه طبقه‌بندی‌کننده‌های قبلی تمرکز می‌کند و وزن آنها را برای بهبود دقت تنظیم می‌کند.

چگونه کار می کند:

ابتدایی سازی(Initialization): در ابتدا به تمام نمونه های آموزشی وزن های مساوی اختصاص دهید.

آموزش تکراری:

یک طبقه‌بندی ضعیف بر روی داده‌های تمرین وزنی آموزش دهید.

میزان خطای طبقه بندی کننده را محاسبه کنید.

وزن نمونه های آموزشی را به روز کنید: وزن نمونه های طبقه بندی اشتباه را افزایش دهید و وزن نمونه هایی که به درستی طبقه بندی شده اند را کاهش دهید.

اهمیت طبقه بندی کننده را بر اساس میزان خطای آن محاسبه کنید.

تجمیع: طبقه‌بندی‌کننده‌های ضعیف را با استفاده از مجموع وزنی پیش‌بینی‌های آن‌ها، که در آن وزن‌ها متناسب با اهمیت هر طبقه‌بندی کننده است، ترکیب کنید.

مزایای:

تمرکز بر نمونه‌های سخت: با افزایش وزن نمونه‌های طبقه‌بندی‌شده اشتباه، AdaBoost بر روی نمونه‌های طبقه‌بندی سخت تمرکز می‌کند و عملکرد کلی را بهبود می‌بخشد.

انعطاف‌پذیری: AdaBoost را می‌توان با انواع مختلفی از طبقه‌بندی‌کننده‌های ضعیف مورد استفاده قرار داد، اگرچه قطع‌های تصمیم رایج هستند.

دقت بهبود یافته: با ترکیب بسیاری از طبقه‌بندی‌کننده‌های ضعیف، AdaBoost می‌تواند به طور قابل توجهی دقت را در مقایسه با یک طبقه‌بندی ضعیف منفرد بهبود بخشد.

چگونه آنها نتایج را بهبود می بخشند:

Bias-variance Trade-off :

جنگل تصادفی: با میانگین‌گیری درخت‌های تصمیم چندگانه، واریانس را کاهش می‌دهد که منجر به پیش‌بینی‌های پایدارتر و دقیق‌تر می‌شود.

AdaBoost: با تمرکز مکرر بر روی نمونه هایی که طبقه بندی آنها دشوار است و ترکیب یادگیرندگان ضعیف، هم سوگیری(Bias) و هم واریانس را کاهش می دهد.

استحکام نسبت به overfitting:

جنگل تصادفی: با جمع آوری بسیاری از درختان و ترکیب تصادفی در انتخاب ویژگی، از تطبیق بیش از حد(overfitting) به داده های آموزشی جلوگیری می کند.

AdaBoost: اگرچه AdaBoost می تواند مستعد تطبیق بیش از حد با داده های noise باشد، وزن مجدد تکراری آن به آن کمک می کند تا روی آموزنده ترین قسمت های داده تمرکز کند.

مدیریت داده های پیچیده:

Random Forest: به دلیل انتخاب تصادفی ویژگی برای داده های با ابعاد بالا موثر است و می تواند تعاملات پیچیده بین ویژگی ها را ثبت کند.

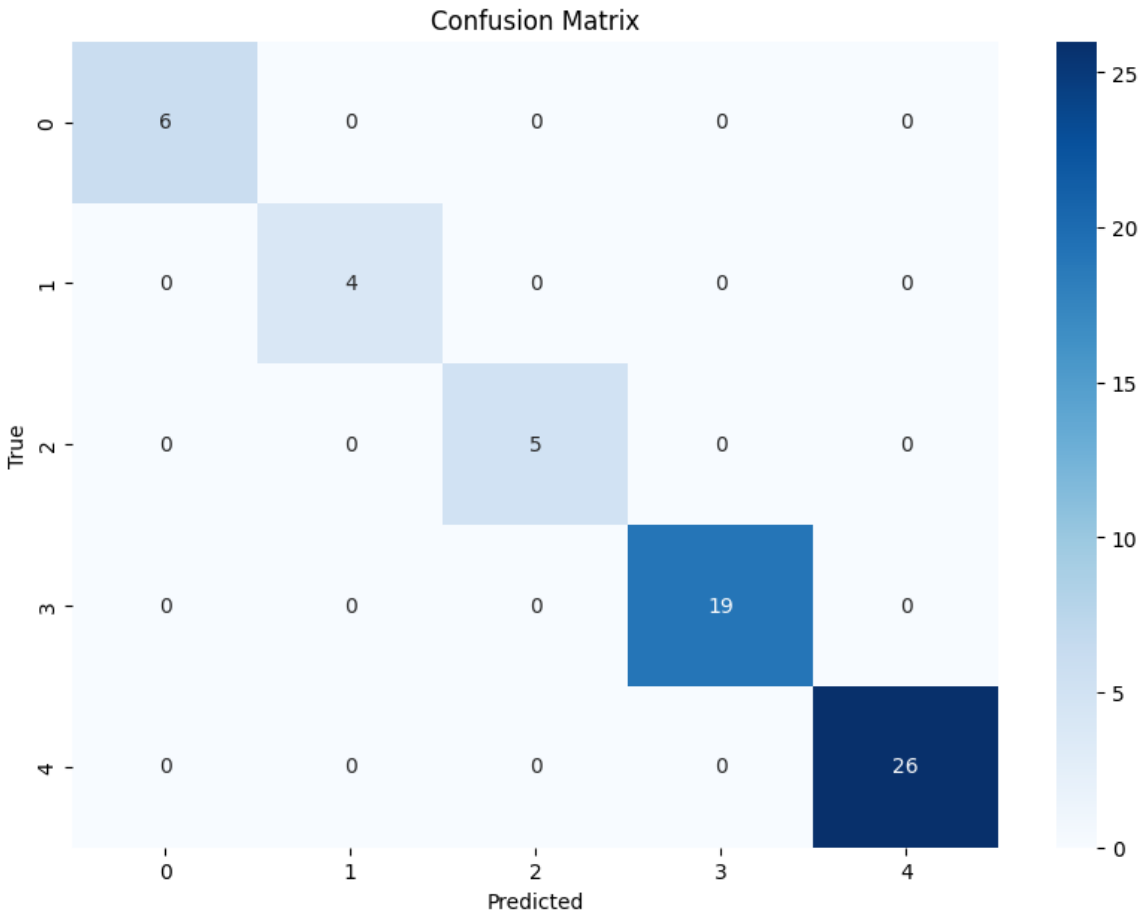
AdaBoost: با انواع مختلف داده ها سازگار است و می تواند عملکرد طبقه بندی کننده های ضعیف را در مجموعه داده های پیچیده بهبود بخشد.

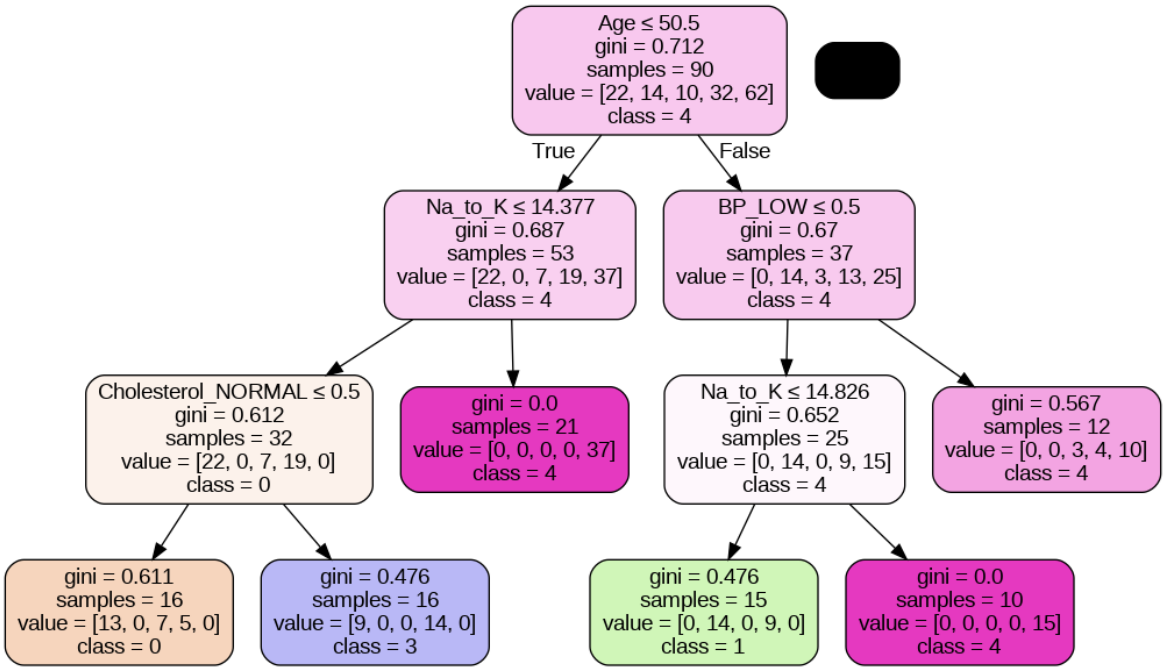
در داده های ما که داده های ساده ای میباشند کار راحتی است که عملکررد بهتری داشته باشند.ما در اینجا از random forest استفاده میکنیم :

مدل ما بدین شکل است :

```
clf = RandomForestClassifier(random_state=64 , n_estimators = 20,
max_depth = 5 , min_samples_split =20)
```

نتایج نیز مانند زیر میباشد :





خوب هیمنجور که میبینید درخت ما نسبت به روش decision tree ساده تر شده (عمق آن کم تر شده) و همچنین توانسته بهترین عملکرد را بر روی داده های تست داشته باشد پس این بهترین درخت میباشد.

(از آنجایی که تعداد داده های ما کم بوده و مسئله ساده میباشد نیازی به استفاده از ۲ طبقه بندی کننده این بخش نبود.)

گزارش کد برای Decision tree :

در اینجا ما فقط برای یک بخش توضیح میاریم چون بقیه کد ها شبیه به هم هستند :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import graphviz
import pydotplus
from IPython.display import Image

# Step 1: Load the data
data = pd.read_csv('drug200.csv')

# Display the first few rows of the dataset
print(data.head())

# Determine the number of unique classes in the label
label_column = 'Drug'  # Replace 'label' with the actual name of the label
column
num_classes = data[label_column].nunique()
print(f"Number of classes: {num_classes}")
print(data[['BP' , 'Cholesterol']].nunique())

# Step 2: Convert categorical features to numerical using one-hot encoding
# Identify categorical columns (excluding the label column)
categorical_columns = data.select_dtypes(include=['object',
'category']).columns.tolist()
categorical_columns = [col for col in categorical_columns if col !=
label_column]

# Apply one-hot encoding to categorical columns
data_encoded = pd.get_dummies(data, columns=categorical_columns,
drop_first=True)

# Step 2.1: Ensure the label is also properly handled if it's categorical
if data[label_column].dtype == 'object' or data[label_column].dtype.name
== 'category':
    data_encoded[label_column] =
data[label_column].astype('category').cat.codes
```

```
print(data_encoded)
# Split the data into training and testing sets
features = data_encoded.drop(columns=[label_column])
labels = data_encoded[label_column]  # Ensure that labels remain intact
X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.3, random_state=64)

# Step 3: Build a decision tree classifier
clf = DecisionTreeClassifier(random_state=64)
clf.fit(X_train, y_train)

# Step 4: Analyze the output
# Predict the classes for the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(report)
print("Confusion Matrix:")
print(conf_matrix)


# Visualization 2: Confusion matrix heatmap
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=clf.classes_, yticklabels=clf.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Visualization 1: Plot the decision tree
dot_data = export_graphviz(clf, out_file=None,
                           feature_names=features.columns,
                           class_names=[str(c) for c in clf.classes_],
                           filled=True, rounded=True,
                           special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

خوب اول از همه داده ای که دانلود کرده بودیم Loadمیکنیم.

Display Data: چند ردیف اول مجموعه داده را برای درک ساختار آن چاپ می کند.

کلاس های منحصر به فرد: تعداد کلاس های منحصر به فرد را در ستون "دارو" که برچسب طبقه بندی است، مشخص می کند. این به درک پیچیدگی مشکل طبقه بندی کمک می کند.

مقادیر منحصر به فرد در ویژگی ها: تعداد مقادیر منحصر به فرد ستون های "BP" و "کلسترول" را برای درک تنوع آنها چاپ می کند.

Identify Category Columns: ستون هایی از نوع شی یا دسته را انتخاب می کند، به استثنای ستون برچسب.

One-Hot Encoding: ستون های دسته بندی را با استفاده از رمزگذاری یک داغ به فرمت عددی تبدیل می کند. این باعث ایجاد ستون های باینری برای هر دسته می شود.

Handle Label Column: اگر ستون برچسب دسته بندی باشد به کدهای عددی تبدیل می شود. این برای طبقه بندی کننده برای پردازش صحیح برچسب ها بسیار مهم است.

Print Encoded Data: DataFrame را پس از رمزگذاری برای تأیید تغییرات نمایش می دهد.

Feature-Label Split: ویژگی ها و برچسب ها را از داده های کدگذاری شده جدا می کند.

Train-Test Split: داده ها را به مجموعه های آموزشی و آزمایشی (70٪ آموزش، 30٪ تست) با استفاده از یک حالت تصادفی ثابت برای تکرارپذیری تقسیم می کند.

Classifier Initialization: یک طبقه بندی درخت تصمیم را با حالت تصادفی ثابت راه اندازی می کند.

آموزش مدل: طبقه بندی کننده را بر روی داده های آموزشی آموزش می دهد.

پیش‌بینی‌ها: از طبقه‌بندی‌کننده آموزش‌دیده برای پیش‌بینی برچسب‌ها برای مجموعه آزمایشی استفاده می‌کند.

معیارهای ارزیابی:

دقت: دقت کلی پیش بینی ها را محاسبه می کند.

گزارش طبقه بندی: معیارهای دقیق (دقت، یادآوری، امتیاز F1) را برای هر کلاس ارائه می دهد.

Confusion Matrix: عملکرد طبقه‌بندی‌کننده را به صورت ماتریسی نشان می‌دهد که نشان‌دهنده طبقه‌بندی‌های درست در مقابل پیش‌بینی‌شده است.

Heatmap: از Seaborn برای ترسیم یک نقشه حرارتی از ماتریس سردرگمی(confusion matrix) استفاده می کند.

Annotations: سلول های نقشه حرارتی را با اعداد واقعی حاشیه نویسی می کند.

برچسب ها و عنوان: برچسب ها و عنوان مناسب را برای وضوح تنظیم می کند.

Export Graphviz: درخت تصمیم آموزش دیده را به فرمت DOT که یک زبان توصیف گراف است، تبدیل می کند.

Create Graph: از pydotplus برای تبدیل داده های DOT به نمودار استفاده می کند.

Display Graph: تصویری از درخت تصمیم را تولید و نمایش می دهد.

سوال ۴)

در ابتدا کتابخانه های مورد نظر را وارد میکنیم.در اینجا جدید ترین کتابخانه :

```
from sklearn.naive_bayes import GaussianNB
```

سپس طبق روش های گفته شده داده مورد نظر را (heart.csv) در google drive آپلود کرده سپس آنرا بر روی google colab دانلود میکنیم.

بعد از آن میبینیم که تعداد داده های ۰ و ۱ ما چه تعداد میباشد :

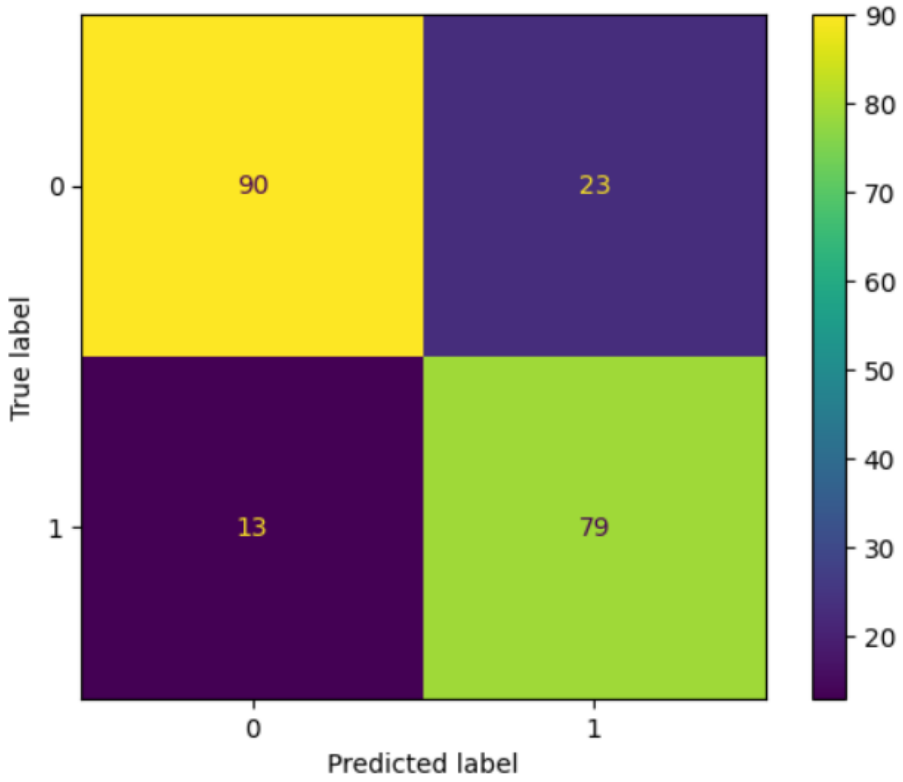
```
target
0    499
1    526
dtype: int64
```

خوب همینطور که مشاهده میکنید تعداد داده ها تقریبا تزدیک به هم میباشد پس نیازی به نگرانی برای این نیست که داده های یک کلاس کم تر از دیگری باشند.(برای یادگیری مهم هستند تا داده ها را بتوان تشخیص داد که متعلق به کدام کلاس هستند.)

سپس ماتریس correlation را تشکیل میدهیم.(البته در اینجا زیاد به کار ما نمی آید چون داده ما یک کار تخصصی میباشد و ما نمیدانیم که کدام feature ها را میتوانیم حذف کنیم.)

سپس داده ها را با روش train_test_split جدا کرده(۲۰ درصد برای داده های تست) سپس با روش standardscalar آنها را normalize میکنیم.

سپس مدل خود را درست کرده و بعد از آن بر روی داده ها test آنرا fit میکنیم سپس نتایج را با confusion matrix و classification report نشان میدهیم که بدین شکل میشوند :(در اینجا ما random state=64 گرفتیم به دلیل شماره دانشجویی بنده)



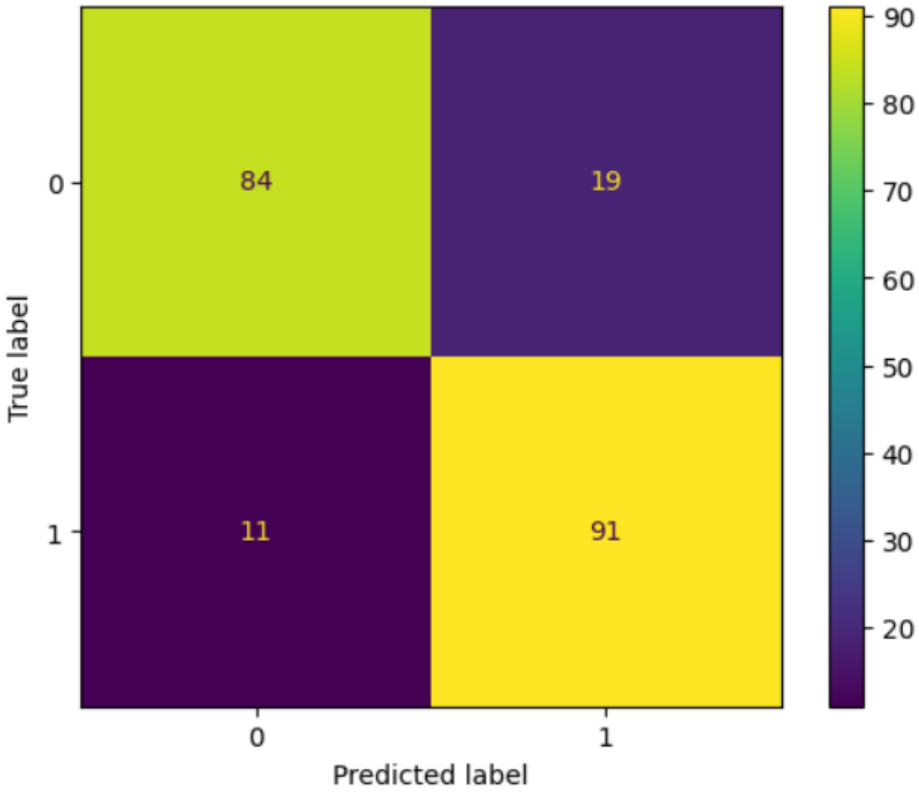
Classification Report:	precision	recall	f1-score	support
0	0.87	0.80	0.83	113
1	0.77	0.86	0.81	92
accuracy			0.82	205
macro avg	0.82	0.83	0.82	205
weighted avg	0.83	0.82	0.82	205

خوب همینطور که مشاهده میکنید precision برای داده های ۰ بهتر از ۱ میباشد در واقع این بدین معنی میباشد که مدل ما داده های ۰ را بهتر میتواند تشخیص دهد(یعنی از بین پیش بینی های ما داده های ۰ را بهتر میتواند پیش بینی کند.) ولی این موضوع برای recall بر عکس میباشد. به راحتی میتوان با جایگذاری در فرمول های هر ۲ این موضوع را نشان داد.

در واقع recall بدین معناست که چقدر مدل ما توانایی این را دارد که داده ها را بین تمام داده های موجود از کلاس مورد نظر بدرستی پیش بینی کند.

F1-score نیز میانگین وزنی Precision و Recall میباشد که در ۰ بهتر از ۱ میباشد که در واقع در آخر دقت مدل را نشان داده که برابر 82. میباشد.

نتایج برای random state = 32 :



Classification Report:

	precision	recall	f1-score	support
0	0.88	0.82	0.85	103
1	0.83	0.89	0.86	102
accuracy			0.85	205
macro avg	0.86	0.85	0.85	205
weighted avg	0.86	0.85	0.85	205

خوب همینطور که میبینید در مدل دوم تقریبا همه چی بهتر از مدل اول میباشد.در اینجا ۲۰۵ داده داریم که باید در قسمت test آنهارا بررسی کنیم.در مدل اول کلاس ۰ و ۱ به ترتیب ۱۱۳ و ۹۲ داده دارند و در مدل دوم ۱۰۳ و ۱۰۲ داده برای بررسی موجود میباشد.به همین دلیل همینجور که میبینید دقت در مدل ۱ برابر 77. ولی

در مدل ۲ برابر 83. میباشد.(بقیه پارامتر ها هم افزایش پیدا کرده اند ولی precision بیشتر از بقیه بیشتر شده است.)در واقع هرچه داده ها بالانس بیشتری داشته باشند کار classification بهتر میباشد.

تفاوت دو حالت Macro و Micro :

Micro average (میانگین مجموع True positives، False negatives و False Positives) فقط برای چند برچسب یا چند کلاس با زیرمجموعه ای از کلاس ها نشان داده می شود، زیرا در غیر این صورت با دقت مطابقت دارد و برای همه معیارها یکسان خواهد بود.(multi class)

Micro :

این معیار با جمع‌آوری مشارکت‌های همه کلاس‌ها با هم محاسبه می‌شود، نه اینکه هر طبقه به طور مستقل رفتار شود.

حالت Micro به کلاس‌های بزرگ‌تر وزن بیشتری می‌دهد، زیرا آنها به معیار کلی کمک بیشتری می‌کنند.

حالت Micro زمانی مفید است که بخواهید عملکرد کلی را ارزیابی کنید و در عین حال عدم تعادل کلاس را در نظر بگیرید.

Macro :

پس از محاسبه متریک برای هر کلاس، the unweighted mean (میانگین) این معیارها برای به دست آوردن متریک کلی محاسبه می شود.

به عبارت دیگر، هر کلاس بدون توجه به عدم تعادل کلاس، به طور مساوی در متریک نهایی مشارکت می کند.

حالت Macro زمانی مفید است که می خواهید عملکرد کلی طبقه بندی کننده را بدون در نظر گرفتن عدم تعادل کلاس ارزیابی کنید.

نشان دادن خروجی برای ۵ داده رندوم :

برای اینکار ابتدا ۵ داده بین ۱ تا ۲۰۵ انتخاب میکنیم سپس این ها را در یک ماتریکس ذخیره کرده و سپس وارد مدل خود میکنیم :

```python
import random

# Initialize an empty matrix to store the random numbers
matrix = []

# Generate 5 random numbers and store them in the matrix
for _ in range(5):
    row = []
    random_number = random.randint(1, 205)
    row.append(random_number)
    matrix.append(row)

# Print the matrix
print(matrix)
```

```
[[89], [26], [123], [34], [80]]
```

همینطور که میبینید داده های مورد نظر بدین شکل میباشند.

مدل ۱ :

```python
print('true label:',y_test.ravel()[matrix])
print('pred label:',y_pred[matrix])
```

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

مدل ۲ :

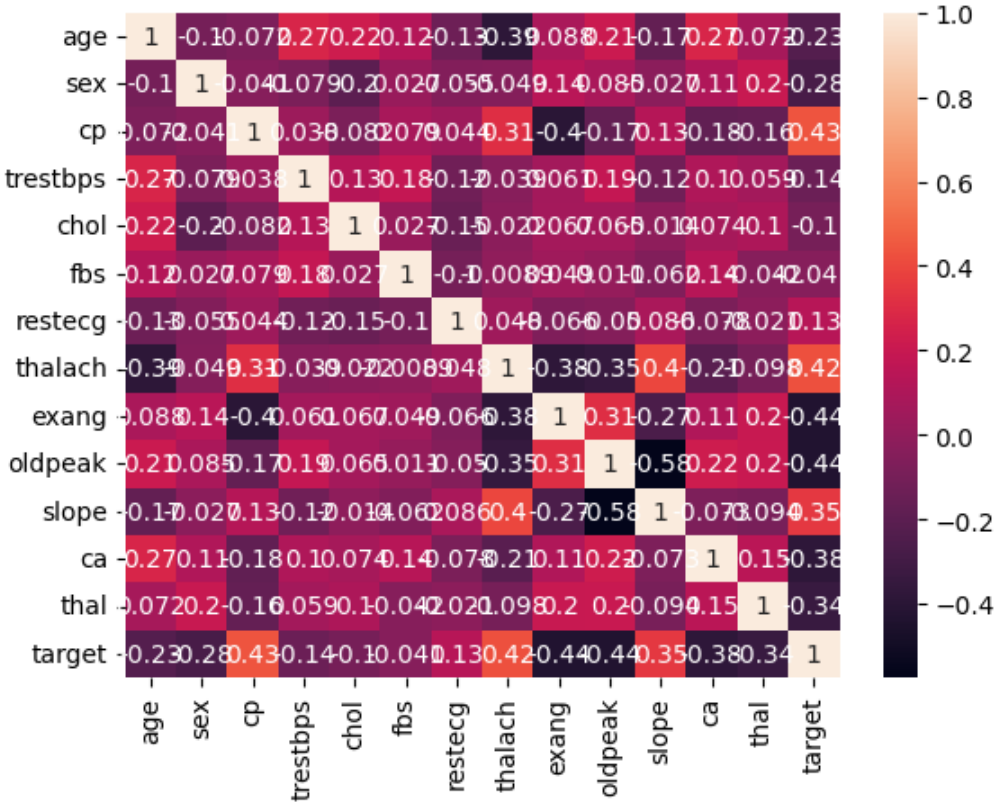
```
print('true label:',y_test.ravel()[matrix])
print('pred label:',y_pred[matrix])
```

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

همینجور که میبینید مدل ۲ آخرین پیش بینی آن به درستی انجام نشده است.

در این روش همینجور که میبنید مدل توانسته به طور کلی عملکرد خوبی داشته باشد.یکی از دلایل آن به این دلیل میباشد که بین feature ها ارتباط زیادی وجود ندارد زیرا این یکی از ارکان اصلی روش bayes برای طبقه بندی میباشد.

البته همینطور که در ابتدا اشاره شد اگر correlation نیز بین ویژگی ها وجود داشته باشد ما اجازه حذف کردن آنها را نداریم زیرا این داده ها داده های تخصصی میباشد و در واقع ارتباط بین ویژگی ها را correlation به ما نمیدهد در واقع correlation یک ارتباط خطی بین ویژگی ها به ما میدهد ولی ممکن است بین ویژگی ها ارتباط غیر خطی باشد.



حداکثر ارتباط خطی برابر 6. میباشد.