بسمه تعالی

مهدی وحیدمقدم

۴۰۲۱۳۰۷۴

مینی‌پروژه شماره۲ یادگیری ماشین

لینک google colab

https://colab.research.google.com/drive/1f6R66Pw5fYVvqZhsgIIOQvk0WND755Mh?usp=sharing

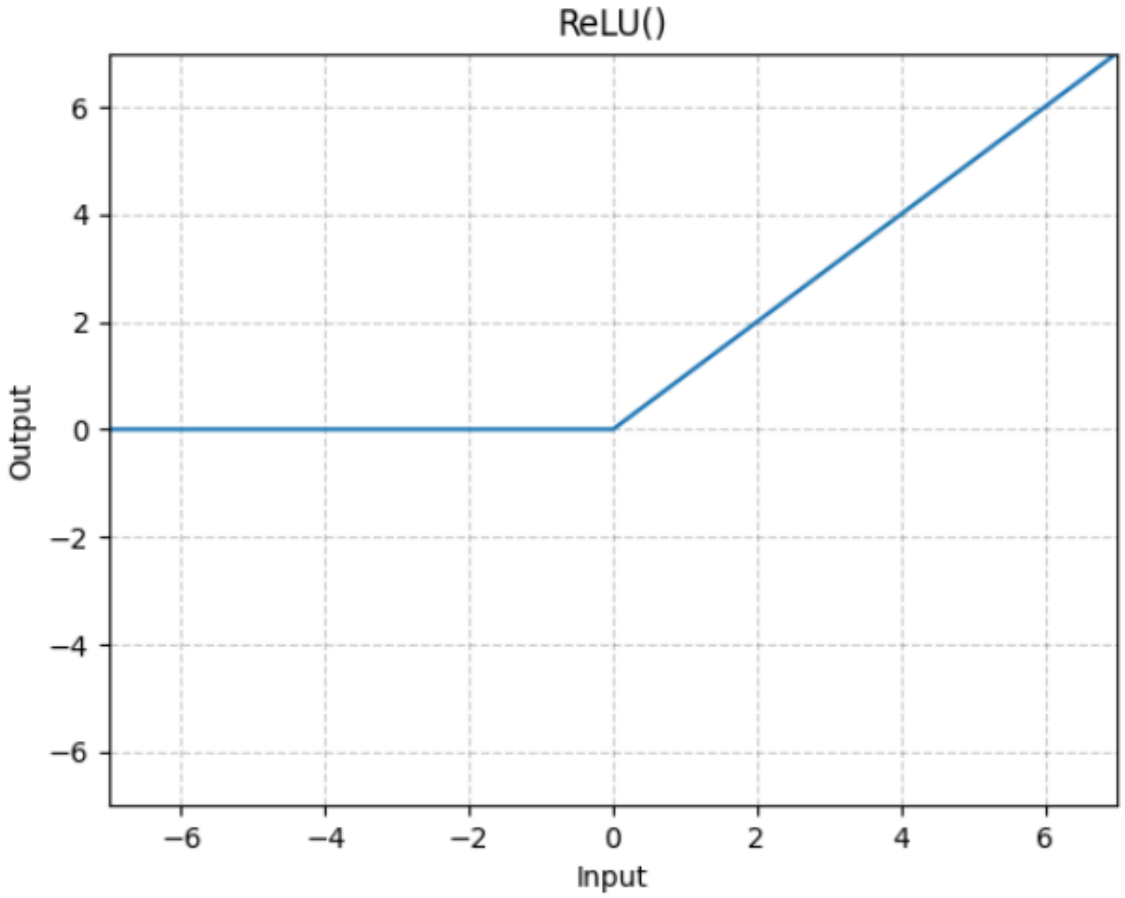
لینک گیت‌هاب

https://github.com/mvmoghadam1999/ML403_MP1_40213074/blob/main/MP2_ML_40213074/ML_MP2.ipynb

سوال ۱)

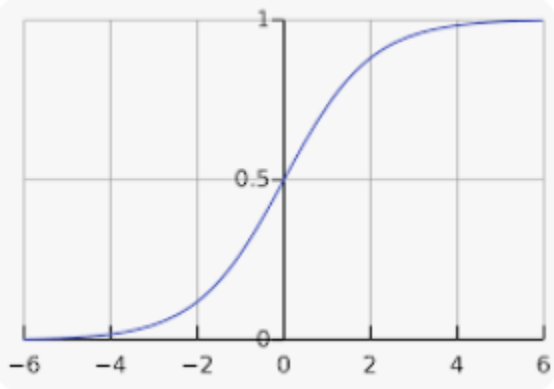
۱. فرض کنید در یک مسألهٔ طبقه‌بندی دوکلاسه، دو لایهٔ انتهایی شبکهٔ شما فعال‌ساز ReLU و سیگموید است. چه اتفاقی می‌افتد؟

تابع ReLU به صورت زیر است:



اگر مقادیر منفی به این تابع داده شود،آن را تبدیل به صفر می‌کند و اگر مقادیر مثبت داده شود،خروجی برابر خود آن عدد خواهد بود.

حال تابع sigmoid به صورت زیر است:



همان‌طور که گفته شد،خروجی تابع ReLU یک عدد بزرگتر مساوی صفر است که وقتی به تابع sigmoid داده می‌شود،خروجی همواره عددی بزرگتر از ۰.۵ خواهد بود.

---

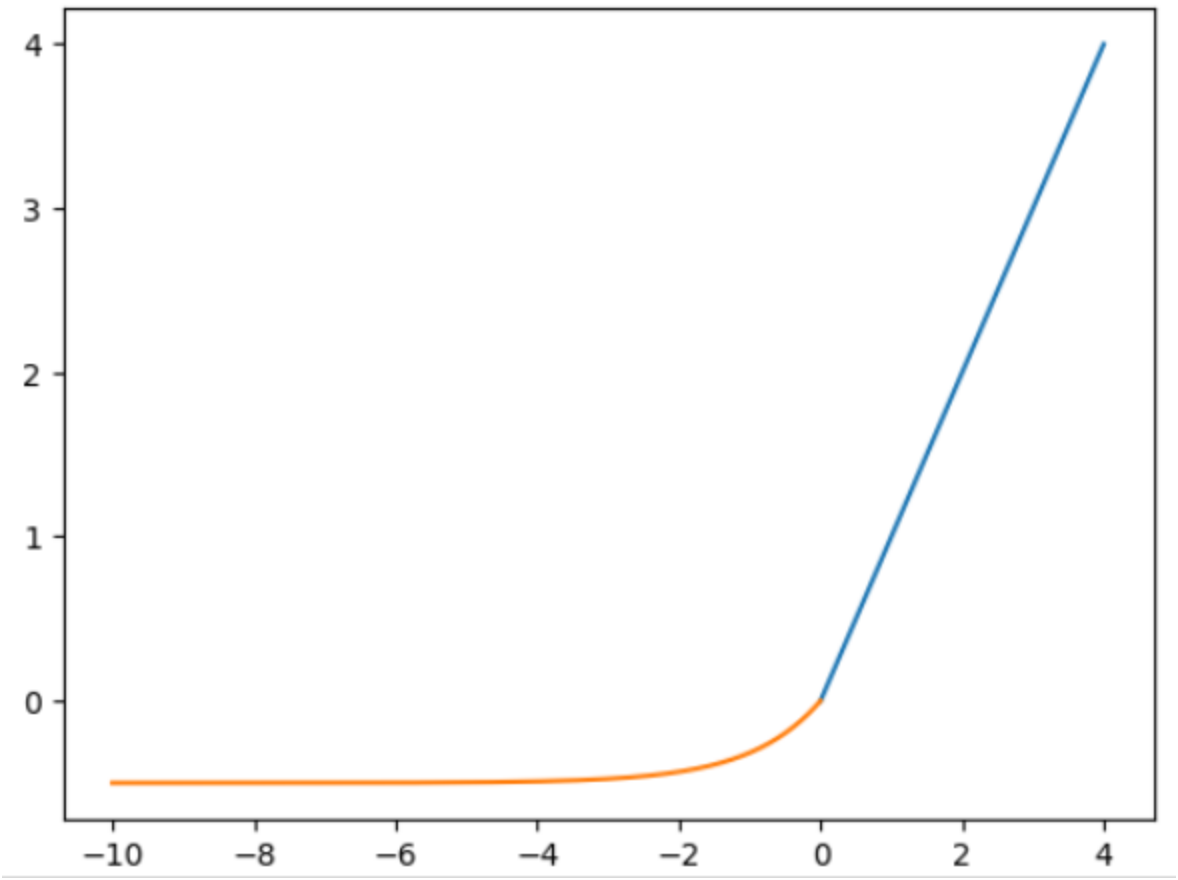
۲. یک جایگزین برای ReLU در معادله ۱ آورده شده است. ضمن محاسبهٔ گرادیان آن، حداقل یک مزیت آن نسبت به ReLU را توضیح دهید.

$$\mathrm{ELU}(x) = \begin{cases} x & x >= 0 \\ \alpha\left(e^{x} - 1\right) & x < 0 \end{cases} \qquad (۱)$$

گرادیان این تابع به صورت زیر است:

$$for\ x \geq 0 \rightarrow \frac{\partial f}{\partial x} = 1 \qquad for\ x \leq 0 \rightarrow \frac{\partial f}{\partial x} = \alpha e^{x}$$

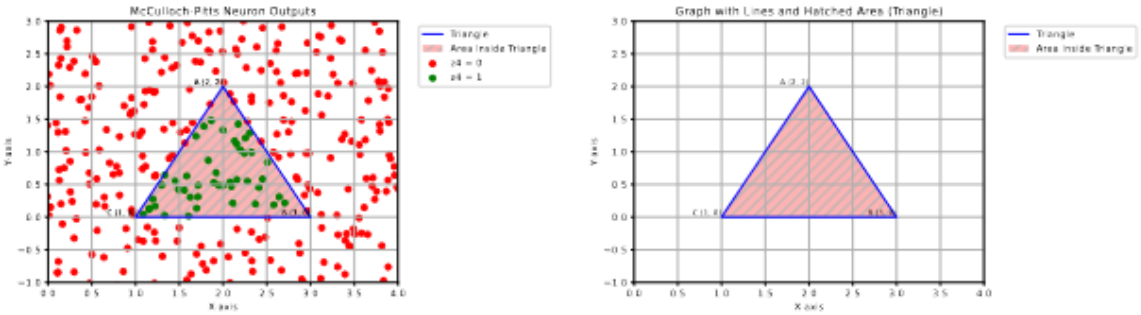
شکل این تابع برای $\alpha = 0.5$ به صورت زیر است:



یکی از مزایای این تابع نسبت به تابع ReLU این است که تابع ReLU مشکل مرگ نورون دارد ؛ یعنی زمانی‌که ورودی صفر یا نزدیک به صفر باشد، تابع ReLU دیگر عملکردی ندارد و به‌بیان دیگر، می‌میرد. در این صورت، مقدار گرادیان تابع صفر می‌شود.اما این تابع با توجه به ضابطه‌ای که برای مقادیر کوچکتر از صفر x در نظر گرفته شده این مشکل را حل کرده است.لازم به ذکر است هر چه مقدار α به صفر نزدیک‌تر شود،شکل تابع به ReLU نزدیک‌تر می‌شود.

---

۳. به کمک یک نورون ساده یا پرسپترون یا نورون McCulloch-Pitts[1] شبکه‌ای طراحی کنید که بتواند ناحیهٔ هاشورزدهٔ داخل مثلثی که در نمودار شکل ۱ (آ) نشان داده شده را از سایر نواحی تفکیک کند. پس از انجام مرحلهٔ طراحی شبکه (که می‌تواند به‌صورت دستی انجام شود)، برنامه‌ای که در این دفترچه‌کد و در کلاس برای نورون McCulloch-Pitts آموخته‌اید را به‌گونه‌ای توسعه دهید که ۲۰۰۰ نقطهٔ رندوم تولید کند و آن‌ها را به‌عنوان ورودی به شبکهٔ طراحی‌شده توسط شما دهد و نقاطی که خروجی «۱» تولید می‌کنند را با رنگ سبز و نقاطی که خروجی «۰» تولید می‌کنند را با رنگ قرمز نشان دهد. خروجی تولیدشده توسط برنامهٔ شما باید به‌صورتی که در شکل ۱ (ب) نشان داده شده است باشد (به محدودهٔ عددی محورهای x و y هم دقت کنید). اثر اضافه‌کردن دو تابع فعال‌ساز مختلف به فرآیند تصمیم‌گیری را هم بررسی کنید.



(آ) نمودار هاشورزدهٔ مورد سوال (ب) خروجی مطلوب برنامه

شکل ۱: نمودارهای مربوط به بخش «۳» سوال اول و خروجی برنامه.

بخش اول کد به صورت زیر است:

```
def ReLU(x):
  return max(0 , x)
def sigmoid(x):
    return 1/(1+np.exp(-x))
class McCulloch_Pitts_neuron():

  def __init__(self , weights , threshold):
    self.weights = weights
    self.threshold = threshold

  def model(self , x):
    if self.weights @ x >= self.threshold:
        return 1
    else:
        return 0
  def model_ReLU(self , x):
    out_relu = ReLU((self.weights @ x) - self.threshold)
    if out_relu == (self.weights @ x) - self.threshold:
      return 1
    if out_relu == 0:
      return 0
  def model_sig(self , x):
    out_sig = sigmoid((self.weights @ x) - self.threshold)
    if out_sig >= 0.5:
```

```
    return 1
  if out_sig <= 0.5:
    return 0
```

در این کد ابتدا دو تابع ReLU و sigmoid تعریف می‌شوند.سپس کلاس McCulloch_Pitts_neuron تعریف می‌شود.تابع اصلی این کلاس دو مقدار ورودی وزن و بایاس را می‌گیرد؛سپس سه تابع برای مدل در این کلاس تعریف شده است.این توابع به ما کمک می‌کنند با توجه به این که نقاط درون ناحیه‌ی مدنظر قرار می‌گیرند یا خیر آن‌ها را به رنگ سبز یا قرمز نشان دهیم.فرق این سه تابع این است که مدل اول فعال‌ساز خاصی ندارد اما در مدل دوم از تابع فعال‌ساز ReLU و در مدل سوم از تابع فعال‌ساز sigmoid استفاده شده است.همان‌طور که مشاهده می‌شود شرایط تعریف شده برای برچسب‌زنی به داده‌ها در هر تابع متفاوت است.

بخش دوم کد به صورت زیر است:

```
def Area(x, y):
  neur1 = McCulloch_Pitts_neuron([2, -1], 2)
  neur2 = McCulloch_Pitts_neuron([-2, -1], -6)
  neur3 = McCulloch_Pitts_neuron([0, 1], 0)
  neur4 = McCulloch_Pitts_neuron([1, 1, 1], 3)

  z1 = neur1.model(np.array([x, y]))
  z2 = neur2.model(np.array([x, y]))
  z3 = neur3.model(np.array([x, y]))
  z4 = neur4.model(np.array([z1, z2, z3]))

  return list([z4]) , neur1 , neur2 , neur3
def Area_ReLU(x, y):
  neur1 = McCulloch_Pitts_neuron([2, -1], 2)
  neur2 = McCulloch_Pitts_neuron([-2, -1], -6)
  neur3 = McCulloch_Pitts_neuron([0, 1], 0)
  neur4 = McCulloch_Pitts_neuron([1, 1, 1], 3)

  z1 = neur1.model_ReLU(np.array([x, y]))
  z2 = neur2.model_ReLU(np.array([x, y]))
  z3 = neur3.model_ReLU(np.array([x, y]))
  z4 = neur4.model_ReLU(np.array([z1, z2, z3]))

  return list([z4]) , neur1 , neur2 , neur3
def Area_sig(x, y):
  neur1 = McCulloch_Pitts_neuron([2, -1], 2)
  neur2 = McCulloch_Pitts_neuron([-2, -1], -6)
  neur3 = McCulloch_Pitts_neuron([0, 1], 0)
  neur4 = McCulloch_Pitts_neuron([1, 1, 1], 3)

  z1 = neur1.model_sig(np.array([x, y]))
```

```
  z2 = neur2.model_sig(np.array([x, y]))
  z3 = neur3.model_sig(np.array([x, y]))
  z4 = neur4.model_sig(np.array([z1, z2, z3]))

  return list([z4]) , neur1 , neur2 , neur3
```

این سه تابع Area تعریف شده هر یک یکی از سه مدل تعریف شده در کلاس بالا را استفاده می-کند تا نتیجه به ازای هر یک از مدل‌ها مشاهده شود. وزن‌ها و بایاس‌های درنظر گرفته شده در نورون‌های تعریف شده نیز به گونه ای است که سه خطی که برای یک مثلث درنظر گرفته می‌شود ارضا شود.

همچنین متغیرهای z1 تا z4 برای تشخیص نقاط سبز و قرمز می‌باشد.(تشخیص این که درون ناحیه قرار داریم یا خارج از آن)

بخش سوم کد به صورت زیر است:

```
import numpy as np
import matplotlib.pyplot as plt

num_points = 2000
x_values = np.random.uniform(0, 4, num_points)
y_values = np.random.uniform(-1, 3, num_points)


red_points = []
green_points = []

for i in range(num_points):
    z4_value = Area(x_values[i], y_values[i])
    if z4_value[0] == [0]:
        red_points.append((x_values[i], y_values[i]))
    else:
        green_points.append((x_values[i], y_values[i]))

red_x, red_y = zip(*red_points)
green_x, green_y = zip(*green_points)
x11 = np.linspace(1 ,2 , 1000)
x22 = np.linspace(2 ,3 , 1000)
x33 = np.linspace(1 ,3 , 1000)
y11 = ((-
z4_value[1].weights[0])*x11+z4_value[1].threshold)/(z4_value[1].weights[1])
y22 = ((-
z4_value[2].weights[0])*x22+z4_value[2].threshold)/(z4_value[2].weights[1])
y33 = ((-
z4_value[3].weights[0])*x33+z4_value[3].threshold)/(z4_value[3].weights[1])
xh =  np.linspace(1 ,y33 , 1000)
plt.figure(figsize=(8, 6))
plt.fill_between(x22, y22 , y33 , color = 'lightcoral')
plt.fill_between(x11, y11 , y33 , color = 'lightcoral')
```

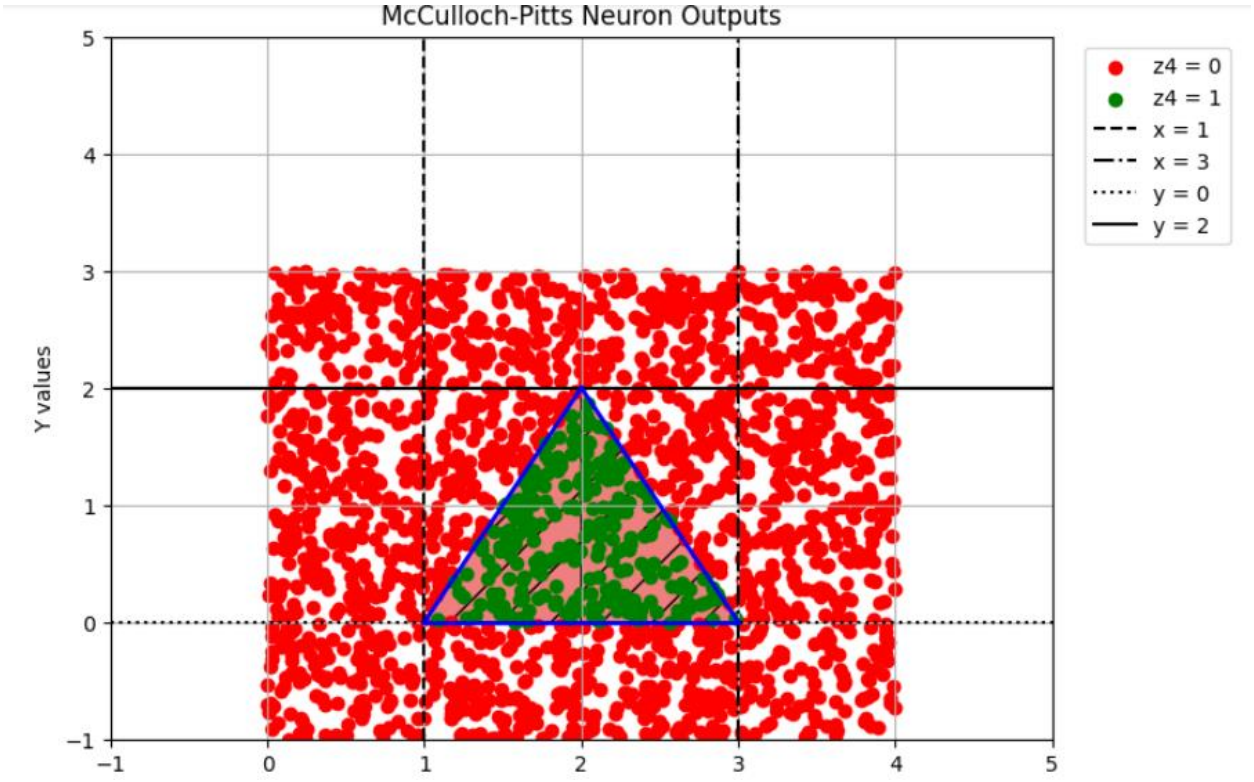
```
plt.fill_between(x22, y22 , y33 , color='none', edgecolor='black', hatch="/")
plt.fill_between(x11, y11 , y33 , color='none', edgecolor='black', hatch="/")
plt.scatter(red_x, red_y, color='red', label='z4 = 0')
plt.scatter(green_x, green_y, color='green', label='z4 = 1')
plt.xlabel('X values')
plt.ylabel('Y values')
plt.title('McCulloch-Pitts Neuron Outputs')

plt.axvline(x=1, color='black', linestyle='--', label='x = 1')
plt.axvline(x=3, color='black', linestyle='-.', label='x = 3')
plt.axhline(y=0, color='black', linestyle=':', label='y = 0')
plt.axhline(y=2, color='black', linestyle='-', label='y = 2')

plt.plot(x11, y11 , color = 'blue' , linewidth = 2)
plt.plot(x22, y22 , color = 'blue' , linewidth = 2)
plt.plot(x33, y33 , color = 'blue' , linewidth = 2)
plt.grid(True)


plt.xlim(-1, 5)
plt.ylim(-1, 5)


plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1.0))
plt.savefig('c.png', bbox_inches='tight')

plt.show()
```

در این کد ابتدا ۲۰۰۰ نقطه با توزیع یونیفرم،با مقادیر x بین ۰ و ۴ و مقادیر y بین منفی ۱ و ۳ تعریف می‌شوند.سپس یک لیست برای نقاط قرمز و یک لیست برای نقاط سبز تعریف می‌شود.سپس در یک حلقه همه نقاط تعریف شده بررسی می‌شوند و تابع Area بر روی آن‌ها اعمال می‌شود.اگر خروجی این تابع صفر باشد،آن نقطه در لیست نقاط قرمز و اگر خروجی تابع صفر نباشد(یعنی یک باشد) آن نقطه در لیست نقاط سبز قرار می‌گیرد.

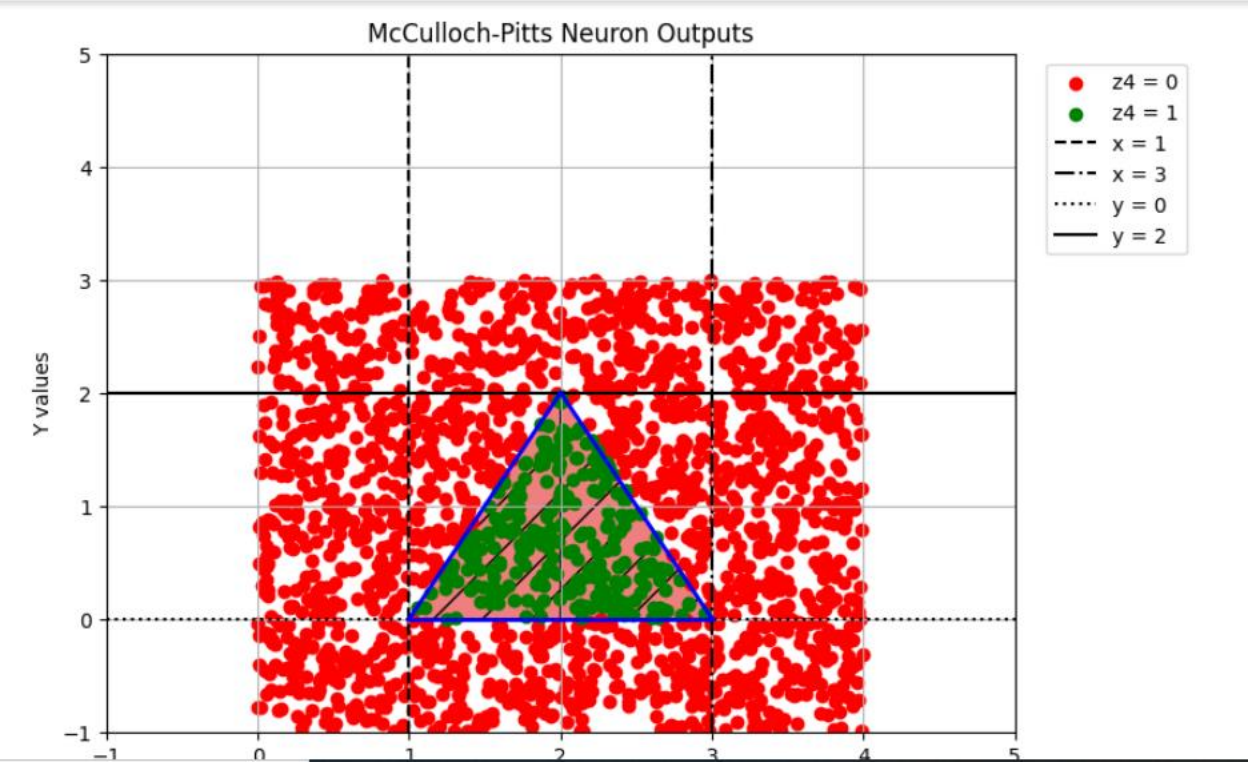
در تابع Area سه خروجی دیگر هم برای دریافت وزن‌ها و بایاسی که به درستی نقاط را تفکیک کرده‌اند،تعریف شده است.در قسمت رسم،ابتدا با توجه به وزن‌ها و بایاس‌ها معادلات خط تعریف شده است.سپس ابتدا ناحیه‌ی بین این سه خط رنگ می‌شود.سپس همین ناحیه با رنگ سیاه هاشور میخورد و در نهایت نقاط سبز و قرمز روی صفحه رسم می‌شوند.

برای دو تابع دیگر نیز کد به همین صورت است با این تفاوت که به جای تابع Area از توابع Area_ReLU و Area_sig استفاده شده است.

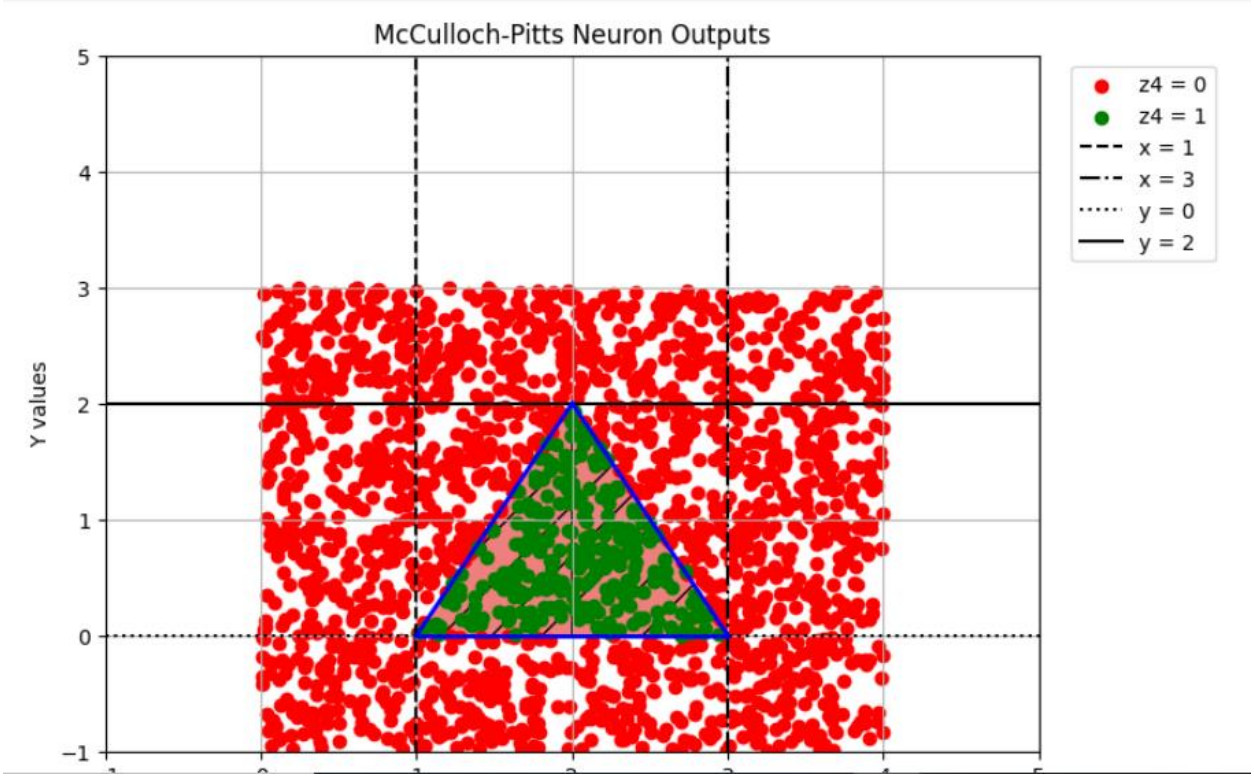
خروجی برای سه تابع به ترتیب به صورت زیر است:



خروجی تابع اول



خروجی تابع دوم



خروجی تابع سوم

سوال ۲)

۱. دیتاست CWRU Bearing که در «مینی‌پروژهٔ شمارهٔ یک» با آن آشنا شدید را به خاطر آورید. علاوه بر دو کلاسی که در آن مینی‌پروژه در نظر گرفتید، با مراجعه به صفحهٔ داده‌های عیب در حالت 12k، دو کلاس دیگر نیز از طریق فایل‌های B007_X[۲] و OR007@6_X اضافه کنید. با انجام این کار یک کلاس دادهٔ سالم و سه کلاس از داده‌های دارای سه عیب متفاوت خواهید داشت. در مورد این که هر فایل مربوط به چه نوع عیبی است به‌صورت کوتاه توضیح دهید.

سپس در ادامه، تمام کارهایی که در بخش «۲» سوال دوم «مینی‌پروژهٔ یک» برای استخراج ویژگی و آماده‌سازی دیتا انجام داده بودید را روی دیتاست جدید خود پیاده‌سازی کنید. در قسمت تقسیم‌بندی داده‌ها، یک بخش برای «اعتبارسنجی» به بخش‌های «آموزش» و «آزمون» اضافه کنید و توضیح دهید که کاربرد این بخش چیست.

دیتاست B007_X مربوط به لرزش‌های مربوط به قسمت ball در خودرو می‌باشد و دیتاست OR007@6_X مربوط به ساچمه رو یا کنس خارجی قرار گرفته در قسمت مرکزی می‌باشد.

قسمت اول کد به صورت زیر است:

```
!gdown 1qDvHzoD7uHy_Wm067KwyNL8g6OPjRByj
!gdown 140WxA8cuZrukyNC30x1UQ17wtdX7zQMc
```

با استفاده از این کد دیتای مربوطه از google drive وارد دفترچه‌ی colab می‌شود.

بخش دوم کد:

```
data = pd.read_csv('/content/data_forq2.csv')
data2 = pd.read_csv('/content/data_forq2_MP2_csv.csv')
X_true = data[['x']].values
y_fault = data[['y']].values
y_fault2 = data2[['fault2']].values
y_fault3 = data2[['fault3']].values


X_true_set = np.array(X_true[:20000])
y_fault_set = np.array(y_fault[:20000])
y_fault2_set = np.array(y_fault2[:20000])
y_fault3_set = np.array(y_fault3[:20000])


X_true_set = X_true_set.reshape(100 , 200)
y_fault_set = y_fault_set.reshape(100 , 200)
y_fault2_set = y_fault2_set.reshape(100 , 200)
y_fault3_set = y_fault3_set.reshape(100 , 200)
list1 = []
list2 = []
list3 = []
list4 = []
for i in range(100):
  list1.append(1)
  list2.append(0)
  list3.append(2)
  list4.append(3)
one_col = np.array(list1)
zero_col = np.array(list2)
two_col = np.array(list3)
three_col = np.array(list4)
one_col = one_col.reshape(100 , 1)
zero_col = zero_col.reshape(100 , 1)
two_col = two_col.reshape(100 , 1)
three_col = three_col.reshape(100 , 1)
X_true_set_with_label = np.append(X_true_set, one_col , axis = 1)
y_fault_set_with_label = np.append(y_fault_set , zero_col , axis = 1)
y_fault2_set_with_label = np.append(y_fault2_set , two_col , axis = 1)
y_fault3_set_with_label = np.append(y_fault3_set , three_col , axis = 1)
dataset = np.vstack((X_true_set, y_fault_set, y_fault2_set, y_fault3_set))
dataset_with_label = np.vstack((X_true_set_with_label ,
y_fault_set_with_label , y_fault2_set_with_label , y_fault3_set_with_label))
```

در این کد ابتدا دیتاهای مربوط به تمرین قبل در data و دو دیتای عیب جدید در data2 ذخیره می‌شوند.(به این صورت که با دستور pd.read_csv اطلاعات از فایل csv آپلود شده در کولب خوانده شده و در data و data2 ذخیره می‌شوند.)

سپس این دیتا به چهار بخش که یک بخش آن داده سالم و سه بخش دیگر داده عیب هستند،تقسیم می‌شود.سپس برای تشکیل دیتای موردنظر ما،از هر یک از این چهار بخش،۲۰۰۰۰ داده استخراج می‌شود و هر کدام تبدیل به یک آرایه ۱۰۰ در ۲۰۰ می‌شود.در واقع با این کار از هر کلاس ۱۰۰ داده با ۲۰۰ ویژگی داریم.

سپس ستون‌هایی ۱۰۰ تایی از اعداد ۰ تا ۳ تعریف می‌شود و به عنوان برچسب در ستون آخر هر یک از داده‌ها یکی از اعداد قرار می‌گیرد.

قسمت سوم کد:

```
def Peak(x):
  return np.max(np.abs(x))
def Standard_deviation(x):
  sum = 0
  for i in range(len(x)):
    sum += np.power((x[i] - np.mean(x)) , 2)
  return(np.sqrt(sum/len(x)))
def Skewness(x):
  sum = 0
  for i in range(len(x)):
    sum += (np.power((x[i] - np.mean(x)) , 3))/len(x)
  return sum/(np.power(Standard_deviation(x) , 3))
def Kurtosis(x):
  sum = 0
  for i in range(len(x)):
    sum += (np.power((x[i] - np.mean(x)) , 4))/len(x)
  return sum/(np.power(Standard_deviation(x) , 4))
def RMS(x):
  sum = 0
  for i in range(len(x)):
    sum += np.power(x[i] , 2)
  return np.sqrt((1/len(x))*sum)
def Crest_Factor(x):
  return((Peak(x))/(RMS(x)))
def SMR(x):
  sum = 0
  for i in range(len(x)):
    sum += np.sqrt(np.abs(x[i]))
```

```
  return (np.power(sum/len(x) , 2))
def Clearance_Factor(x):
  return((Peak(x))/(SMR(x)))
def Peak_to_Peak(x):
  return (np.max(x) - np.min(x))
def Mean(x):
  return np.mean(x)
def feature(x):
  X_feature = []
  for i in range(100):
    X_feature.append([Peak(x[i]) , Standard_deviation(x[i]) , Skewness(x[i])
, Kurtosis(x[i]) , RMS(x[i]) , Crest_Factor(x[i]) , SMR(x[i]) ,
Clearance_Factor(x[i]) , Peak_to_Peak(x[i]) , Mean(x[i])])
  X_feature = np.array(X_feature)
  X_feature = X_feature.reshape(100 , 10)
  return X_feature
```

این قسمت در پروژه قبلی هم به همین صورت وجود داشت.در این قسمت ۱۰ تابع تعریف شده است که برای استخراج ویژگی از دیتای موردنظر است.در آخر نیز تابعی تعریف شده که دیتا را می‌گیرد و دیتای با ویژگی‌های استخراج شده را می‌دهد.

قسمت چهارم کد:

```
from sklearn.utils import shuffle

feature_X_true = feature(X_true_set)
feature_X_true_with_label = np.append(feature_X_true , one_col , axis = 1)
feature_y_fault = feature(y_fault_set)
feature_y_fault_with_label = np.append(feature_y_fault , zero_col , axis = 1)
feature_y_fault2 = feature(y_fault2_set)
feature_y_fault2_with_label = np.append(feature_y_fault2 , two_col , axis =
1)
feature_y_fault3 = feature(y_fault3_set)
feature_y_fault3_with_label = np.append(feature_y_fault3 , three_col , axis =
1)
feature_data_with_label = np.vstack((feature_X_true_with_label ,
feature_y_fault_with_label , feature_y_fault2_with_label ,
feature_y_fault3_with_label))
X_shuffle , y_shuffle = shuffle(feature_data_with_label[: , :10] ,
feature_data_with_label[: , 10])
y_shuffle = y_shuffle.reshape(400 , 1)
shuffled_data = np.append(X_shuffle , y_shuffle , axis = 1)
X_train , X_test , y_train , y_test = train_test_split(shuffled_data[: , :10]
,
                                                       shuffled_data[: , 10]
,
                                                       test_size = 0.2 ,
```

```
                                                        random_state = 74)
X_real_test , X_validation , y_real_test , y_validation =
train_test_split(X_test ,

y_test ,

test_size = 0.2 ,

random_state = 74)
y_train = y_train.reshape(320 , 1)
y_real_test = y_real_test.reshape(64 , 1)
y_validation = y_validation.reshape(16 , 1)
y_test = y_test.reshape(80 , 1)
X_train_with_label = np.append(X_train , y_train , axis = 1)
X_real_test_with_label = np.append(X_real_test , y_real_test , axis = 1)
X_validation_with_label = np.append(X_validation , y_validation , axis = 1)
```

در این قسمت از کد ابتدا ویژگی‌ها از داده موردنظر استخراج می‌شود.سپس به این داده‌ها برچسب مربوط به هر یک نیز زده می‌شود.در این حالت برای هر کلاس یک دیتای ۱۰۰ در ۱۰ داریم.یعنی تعداد ویژگی‌ها از ۲۰۰ به ۱۰ کاهش یافته است.در قدم بعد،دیتا مخلوط می‌شود تا قابلیت اطمینان آموزش بالاتر برود.

در قسمت بعد دیتا به دو بخش آموزش و ارزیابی تقسیم می‌شود.بعد از آن داده‌های تست نیز خود به دو بخش تست و Validation تقسیم می‌شود.در واقع،۲۰ درصد از داده‌های تست به عنوان Validation در نظر گرفته شده است.

داده‌های Validation داده‌هایی هستند که کارفرما یا ناظر پروژه و یا حتی خود کسی که مدل را ساخته است،در اختیار دارد تا تست نهایی را روی مدل با این دیتا انجام دهد تا مطمئن شود که این مدل به درستی عمل می‌کند.

قسمت پنجم کد:

```
from sklearn.preprocessing import MinMaxScaler , StandardScaler
from sklearn.model_selection import train_test_split
scaler = MinMaxScaler()
scaler2 =  StandardScaler()
X_tr_normal = scaler2.fit_transform(X_train)
X_te_normal = scaler2.fit_transform(X_test)
X_val_normal = scaler2.fit_transform(X_validation)
```

```
X_trl_normal = np.append(X_tr_normal , y_train , axis = 1)
X_tel_normal = np.append(X_te_normal , y_test , axis = 1)
X_tvl_normal = np.append(X_val_normal , y_validation , axis = 1)
```

در این قسمت از کد،نرمال‌سازی داده انجام می‌شود.نرمال‌سازی داده‌ها با استفاده از StandardScaler انجام می‌شود.

---

۲. یک مدل Multi-Layer Perceptron (MLP) ساده با ۲ لایهٔ پنهان یا بیش‌تر بسازید. بخشی از داده‌های آموزش را برای اعتبارسنجی کنار بگذارید و با انتخاب بهینه‌ساز و تابع اتلاف مناسب، مدل را آموزش دهید. نمودارهای اتلاف و Accuracy مربوط به آموزش و اعتبارسنجی را رسم و نتیجه را تحلیل کنید. نتیجهٔ تست مدل روی داده‌های آزمون را با استفاده ماتریس درهم‌ریختگی و classification_report نشان داده و نتایج به‌صورت دقیق تحلیل کنید.

مدل تعریف شده برای این قسمت به صورت زیر است:

```
model_keras = Sequential(
    [
        Input(shape=(10,), name='Input'),
        Dense(units=12, activation='relu', name='Hidden_1'),
        Dense(units=8, activation='relu', name='Hidden_2'),
        Dense(units=4, activation='softmax', name='Classification')
    ]
)
```

در این مدل یک لایه ورودی تعریف شده است.برای این لایه اندازه ۱۰ در نظر گرفته است.زیرا تعداد ۱۰ ویژگی از دیتای موردنظر استخراج شده است.همچنین دو لایه‌ی میانی با اندازه ۱۲ و ۸ در نظر گرفته شده است.لایه‌ی آخر که برای طبقه‌بندی در نظر گرفته شده است،۴ خروجی می‌دهد که برابر با تعداد کلاس های دیتا است.برای توابع فعال‌ساز نیز برای لایه‌های میانی تابع ReLU و برای لایه طبقه‌بندی softmax در نظر گرفته می‌شود.

قسمت بعد کد به صورت زیر است:

```
loss_fn = SparseCategoricalCrossentropy()
optim_fn = Adam(learning_rate=0.001)
model_keras.compile(optimizer=optim_fn, loss=loss_fn, metrics=['acc'])
history = model_keras.fit(X_tr_normal, y_train, validation_data =
(X_te_normal, y_test) , epochs=50, batch_size=16)
train_acc = history.history['acc']
train_loss = history.history['loss']

val_acc = history.history['val_acc']
val_loss = history.history['val_loss']
results = model_keras.evaluate(X_te_normal , y_test)
y_pred_keras = model_keras.predict(X_te_normal)
rscore_2 = r2_score(y_test , np.argmax(y_pred_keras , axis = 1))

print(rscore_2)

cf_matrix = confusion_matrix(y_test, np.argmax(y_pred_keras , axis = 1))


print(classification_report(y_test.reshape(80 , ) , np.argmax(y_pred_keras ,
axis = 1)))


plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues', annot_kws={"size":
12})


plt.gca().set_ylim(len(np.unique(y_test)), 0)
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')


plt.tight_layout()
plt.savefig('confusion_matrix1.png', dpi=300)
plt.show()

plt.figure(figsize=(8,5))
plt.plot(train_acc, 'r-o', label='Train Accuracy')
plt.plot(val_acc, 'g-o', label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.legend()
plt.grid()

plt.figure(figsize=(8,5))
plt.plot(train_loss, 'r-o', label='Train Loss')
plt.plot(val_loss, 'g-o', label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.legend()
plt.grid()
```

در این قسمت ابتدا توابع اتلاف و بهینه ساز را تعریف می‌کنیم.در این جا تابع اتلاف SparseCategoricalCrossentropy() و بهینه ساز Adam است.همچنین نرخ یادگیری برای تابع بهینه ساز برابر ۰.۰۰۱ در نظر گرفته شده است.

در بخش با compile در واقع پیکربندی فرایند آموزش و یادگیری اتفاق می‌افتد.

در بخش بعد آموزش با داده‌ی بخش آموزش که از قبل تعیین شد انجام می‌شود.این آموزش با تعداد epochs = 50 و batch_size = 16 اتفاق می‌افتد.پس از آن مقادیر اتلاف و دقت بخش آموزش و اتلاف و دقت بخش ارزیابی برای هر epoch نمایش داده می‌شود.همچنین پیش‌بینی-هایی که مدل از داده‌های تست دارد در یک متغیر ذخیره می‌شود.البته این پیش‌بینی به صورت برچسب هر داده نیست و نیاز است از تابع np.argmax() با axis = 1 استفاده شود تا در هر ردیف بیشترین احتمال به عنوان کلاس پیش‌بینی شده توسط مدل تعیین شود.

در قسمت بعد مقادیر r2_score و بعد از آن به ترتیب ماتریس درهم‌ریختگی و نمودار تغییرات دقت و نمودار تغییرات اتلاف رسم می‌شود.

نتیجه به صورت زیر است:

```
Epoch 1/50
20/20 [==============================] - 1s 20ms/step - loss: 1.4109 - acc:
0.4125 - val_loss: 1.2905 - val_acc: 0.5000
Epoch 2/50
20/20 [==============================] - 0s 7ms/step - loss: 1.2745 - acc:
0.4594 - val_loss: 1.1791 - val_acc: 0.5500
Epoch 3/50
20/20 [==============================] - 0s 6ms/step - loss: 1.1673 - acc:
0.5312 - val_loss: 1.0868 - val_acc: 0.6750
Epoch 4/50
20/20 [==============================] - 0s 7ms/step - loss: 1.0755 - acc:
0.5094 - val_loss: 1.0060 - val_acc: 0.4875
Epoch 5/50
20/20 [==============================] - 0s 7ms/step - loss: 0.9953 - acc:
0.4563 - val_loss: 0.9379 - val_acc: 0.5000
Epoch 6/50
20/20 [==============================] - 0s 6ms/step - loss: 0.9254 - acc:
0.4469 - val_loss: 0.8806 - val_acc: 0.5125
Epoch 7/50
20/20 [==============================] - 0s 6ms/step - loss: 0.8652 - acc:
0.4313 - val_loss: 0.8305 - val_acc: 0.5125
Epoch 8/50
20/20 [==============================] - 0s 7ms/step - loss: 0.8083 - acc:
0.4875 - val_loss: 0.7809 - val_acc: 0.6000
```

```
Epoch 9/50
20/20 [==============================] - 0s 6ms/step - loss: 0.7565 - acc:
0.6812 - val_loss: 0.7397 - val_acc: 0.6125
Epoch 10/50
20/20 [==============================] - 0s 6ms/step - loss: 0.7084 - acc:
0.7125 - val_loss: 0.7009 - val_acc: 0.6250
Epoch 11/50
20/20 [==============================] - 0s 6ms/step - loss: 0.6641 - acc:
0.7094 - val_loss: 0.6678 - val_acc: 0.6375
Epoch 12/50
20/20 [==============================] - 0s 6ms/step - loss: 0.6238 - acc:
0.7094 - val_loss: 0.6395 - val_acc: 0.6375
Epoch 13/50
20/20 [==============================] - 0s 6ms/step - loss: 0.5895 - acc:
0.7156 - val_loss: 0.6130 - val_acc: 0.6375
Epoch 14/50
20/20 [==============================] - 0s 6ms/step - loss: 0.5548 - acc:
0.7219 - val_loss: 0.5876 - val_acc: 0.6500
Epoch 15/50
20/20 [==============================] - 0s 5ms/step - loss: 0.5270 - acc:
0.7406 - val_loss: 0.5621 - val_acc: 0.6625
Epoch 16/50
20/20 [==============================] - 0s 7ms/step - loss: 0.5013 - acc:
0.7469 - val_loss: 0.5405 - val_acc: 0.6875
Epoch 17/50
20/20 [==============================] - 0s 7ms/step - loss: 0.4780 - acc:
0.7656 - val_loss: 0.5130 - val_acc: 0.6750
Epoch 18/50
20/20 [==============================] - 0s 7ms/step - loss: 0.4579 - acc:
0.7875 - val_loss: 0.4923 - val_acc: 0.7000
Epoch 19/50
20/20 [==============================] - 0s 6ms/step - loss: 0.4375 - acc:
0.8094 - val_loss: 0.4758 - val_acc: 0.7375
Epoch 20/50
20/20 [==============================] - 0s 6ms/step - loss: 0.4218 - acc:
0.8156 - val_loss: 0.4495 - val_acc: 0.7500
Epoch 21/50
20/20 [==============================] - 0s 6ms/step - loss: 0.4027 - acc:
0.8469 - val_loss: 0.4337 - val_acc: 0.7625
Epoch 22/50
20/20 [==============================] - 0s 6ms/step - loss: 0.3846 - acc:
0.8469 - val_loss: 0.4126 - val_acc: 0.7750
Epoch 23/50
20/20 [==============================] - 0s 7ms/step - loss: 0.3673 - acc:
0.8656 - val_loss: 0.3986 - val_acc: 0.7875
Epoch 24/50
20/20 [==============================] - 0s 8ms/step - loss: 0.3533 - acc:
0.8750 - val_loss: 0.3807 - val_acc: 0.8000
Epoch 25/50
20/20 [==============================] - 0s 8ms/step - loss: 0.3370 - acc:
0.8750 - val_loss: 0.3583 - val_acc: 0.8250
Epoch 26/50
20/20 [==============================] - 0s 8ms/step - loss: 0.3180 - acc:
0.9031 - val_loss: 0.3391 - val_acc: 0.8375
Epoch 27/50
20/20 [==============================] - 0s 8ms/step - loss: 0.3005 - acc:
0.9062 - val_loss: 0.3176 - val_acc: 0.8625
```

```
Epoch 28/50
20/20 [==============================] - 0s 7ms/step - loss: 0.2837 - acc:
0.9312 - val_loss: 0.3016 - val_acc: 0.8875
Epoch 29/50
20/20 [==============================] - 0s 8ms/step - loss: 0.2672 - acc:
0.9344 - val_loss: 0.2882 - val_acc: 0.9000
Epoch 30/50
20/20 [==============================] - 0s 9ms/step - loss: 0.2527 - acc:
0.9438 - val_loss: 0.2715 - val_acc: 0.9125
Epoch 31/50
20/20 [==============================] - 0s 8ms/step - loss: 0.2381 - acc:
0.9500 - val_loss: 0.2563 - val_acc: 0.9125
Epoch 32/50
20/20 [==============================] - 0s 8ms/step - loss: 0.2251 - acc:
0.9531 - val_loss: 0.2440 - val_acc: 0.9250
Epoch 33/50
20/20 [==============================] - 0s 9ms/step - loss: 0.2114 - acc:
0.9656 - val_loss: 0.2234 - val_acc: 0.9250
Epoch 34/50
20/20 [==============================] - 0s 7ms/step - loss: 0.1985 - acc:
0.9656 - val_loss: 0.2185 - val_acc: 0.9250
Epoch 35/50
20/20 [==============================] - 0s 8ms/step - loss: 0.1874 - acc:
0.9719 - val_loss: 0.2082 - val_acc: 0.9375
Epoch 36/50
20/20 [==============================] - 0s 9ms/step - loss: 0.1753 - acc:
0.9719 - val_loss: 0.1921 - val_acc: 0.9375
Epoch 37/50
20/20 [==============================] - 0s 9ms/step - loss: 0.1662 - acc:
0.9719 - val_loss: 0.1786 - val_acc: 0.9375
Epoch 38/50
20/20 [==============================] - 0s 8ms/step - loss: 0.1564 - acc:
0.9719 - val_loss: 0.1754 - val_acc: 0.9500
Epoch 39/50
20/20 [==============================] - 0s 8ms/step - loss: 0.1460 - acc:
0.9719 - val_loss: 0.1659 - val_acc: 0.9375
Epoch 40/50
20/20 [==============================] - 0s 8ms/step - loss: 0.1396 - acc:
0.9781 - val_loss: 0.1555 - val_acc: 0.9500
Epoch 41/50
20/20 [==============================] - 0s 8ms/step - loss: 0.1309 - acc:
0.9750 - val_loss: 0.1555 - val_acc: 0.9500
Epoch 42/50
20/20 [==============================] - 0s 9ms/step - loss: 0.1241 - acc:
0.9812 - val_loss: 0.1464 - val_acc: 0.9500
Epoch 43/50
20/20 [==============================] - 0s 8ms/step - loss: 0.1167 - acc:
0.9812 - val_loss: 0.1393 - val_acc: 0.9500
Epoch 44/50
20/20 [==============================] - 0s 9ms/step - loss: 0.1108 - acc:
0.9812 - val_loss: 0.1314 - val_acc: 0.9500
Epoch 45/50
20/20 [==============================] - 0s 9ms/step - loss: 0.1057 - acc:
0.9844 - val_loss: 0.1277 - val_acc: 0.9500
Epoch 46/50
20/20 [==============================] - 0s 7ms/step - loss: 0.1004 - acc:
0.9844 - val_loss: 0.1237 - val_acc: 0.9500
```

```
Epoch 47/50
20/20 [==============================] - 0s 7ms/step - loss: 0.0956 - acc:
0.9844 - val_loss: 0.1205 - val_acc: 0.9625
Epoch 48/50
20/20 [==============================] - 0s 7ms/step - loss: 0.0909 - acc:
0.9844 - val_loss: 0.1131 - val_acc: 0.9625
Epoch 49/50
20/20 [==============================] - 0s 9ms/step - loss: 0.0871 - acc:
0.9875 - val_loss: 0.1113 - val_acc: 0.9625
Epoch 50/50
20/20 [==============================] - 0s 8ms/step - loss: 0.0831 - acc:
0.9875 - val_loss: 0.1033 - val_acc: 0.9625
3/3 [==============================] - 0s 11ms/step - loss: 0.1033 - acc:
0.9625
3/3 [==============================] - 0s 5ms/step
```

همان‌طور که مشاهده می‌شود در نهایت به دقت 0.9625 رسیده ایم.

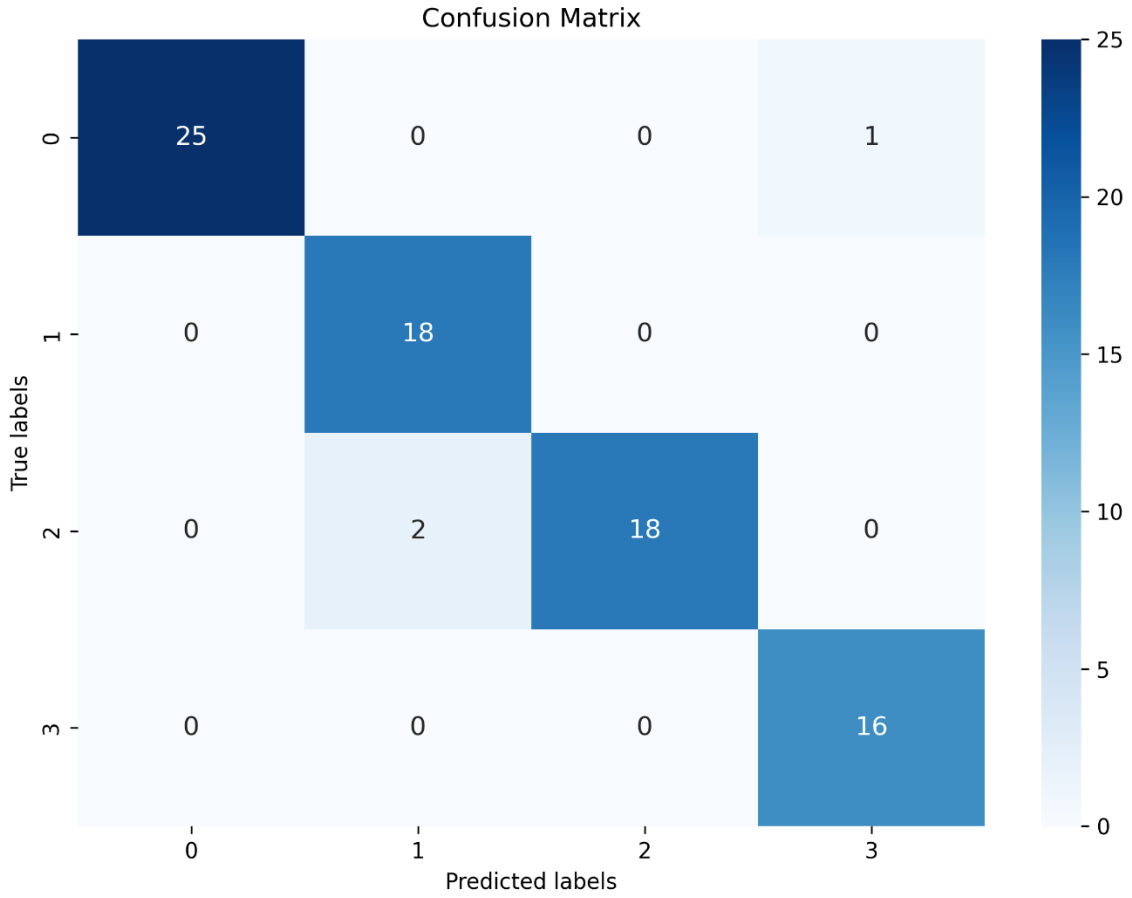
مقدار r2_score به صورت زیر است:

```
0.8916789758739537
```

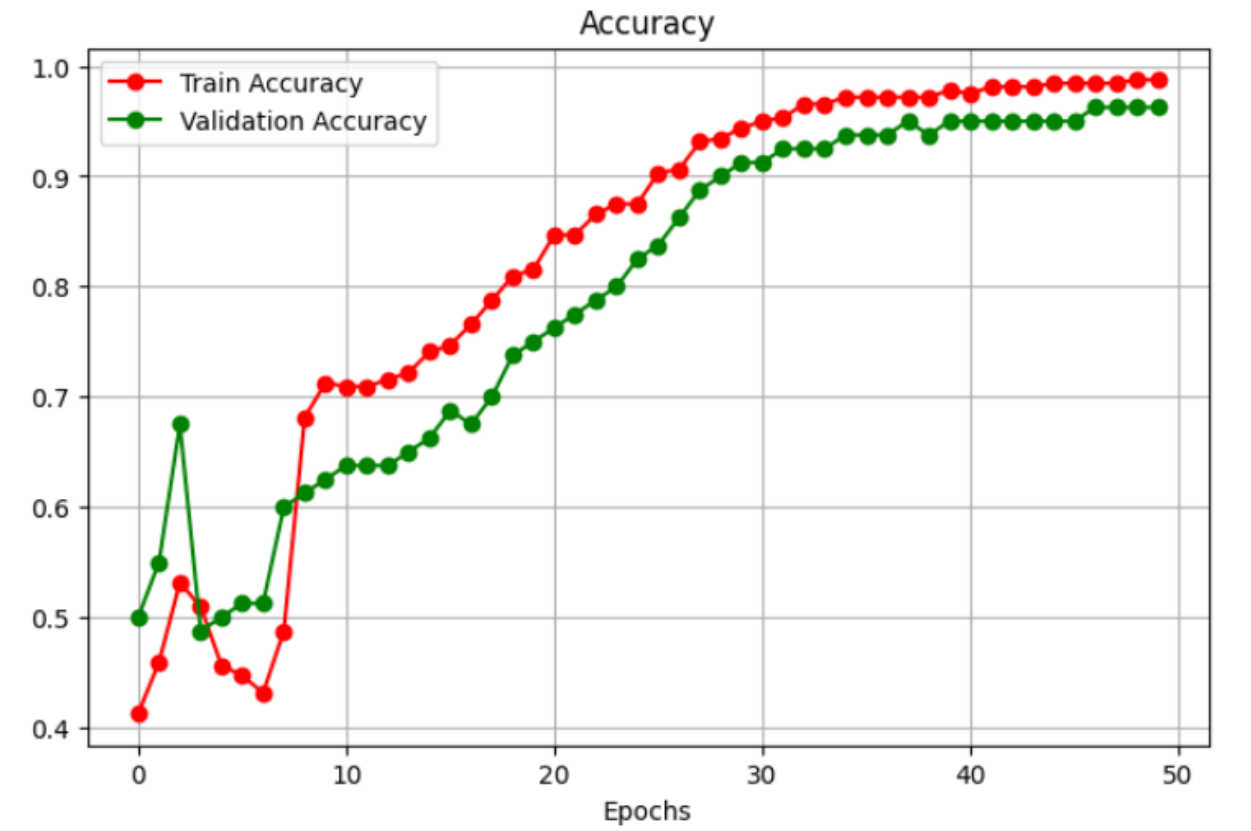
همچنین خروجی classification_report به صورت زیر است:

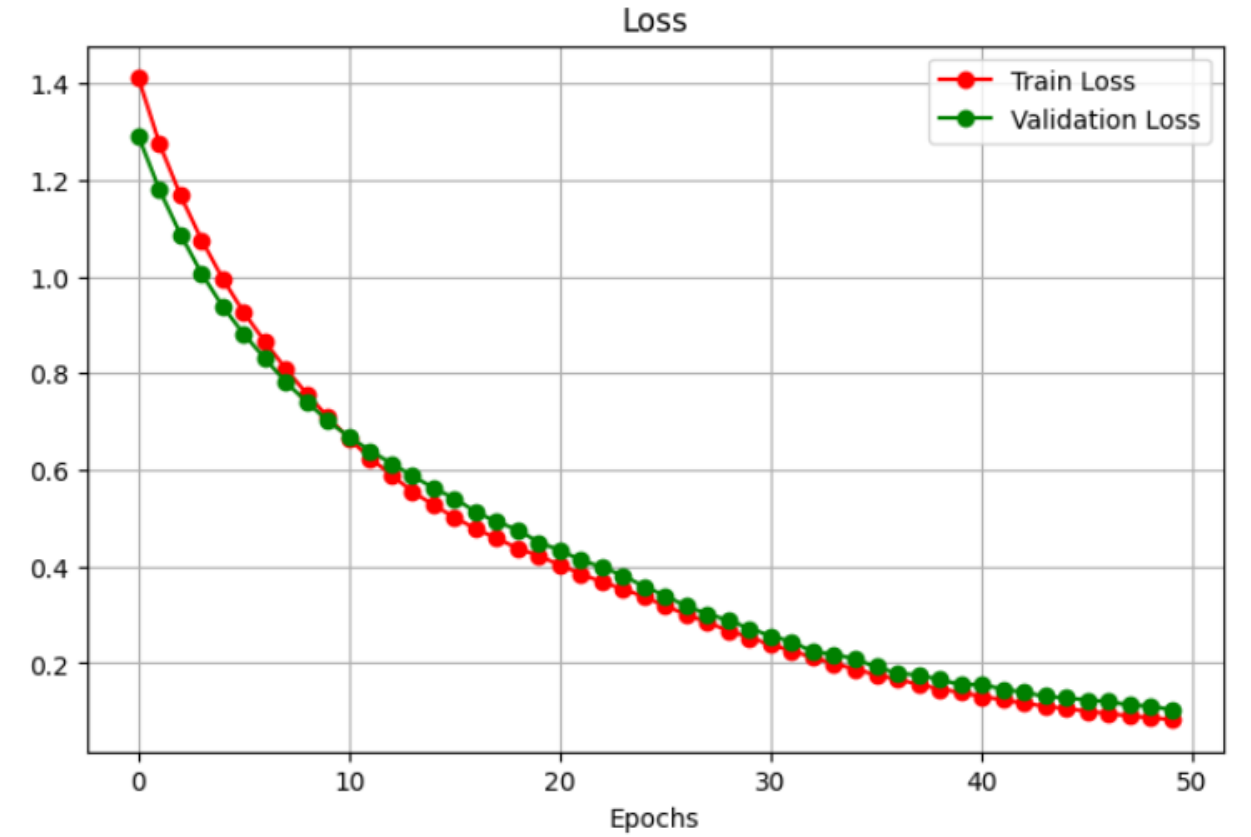
```
precision    recall  f1-score   support

         0.0       1.00      0.96      0.98        26
         1.0       0.90      1.00      0.95        18
         2.0       1.00      0.90      0.95        20
         3.0       0.94      1.00      0.97        16

    accuracy                           0.96        80
   macro avg       0.96      0.97      0.96        80
weighted avg       0.97      0.96      0.96        80
```

ماتریس درهم‌ریختگی به صورت زیر است:



نمودارهای تغییرات دقت و اتلاف نیز به صورت زیر هستند:


Accuracy
Train Accuracy
Validation Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
10
20
30
40
50
Epochs


Loss
Train Loss
Validation Loss
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50
Epochs

همان‌طور که مشاهده می‌شود مقدار دقت با زیاد شدن دوره‌های آموزش غیر از یک نقطه خاص تقریبا روند صعودی دارد.(ممکن است در قسمت چون دوره‌های زیادی از آموزش نگذشته است و همزمان دیتاها قابلیت تفکیک کمتری پیدا کرده اند،فرایند آموزش کمی دچار مشکل شده باشد.)

همچنین نمودار اتلاف نشان می‌دهد که هم در داده‌ی آموزش و هم در داده‌ی ارزیابی کاهشی است و در نهایت به مقداری نزدیک به صفر رسیده است.همچنین این نمودار نوسانی نداشته است و اتلاف به ازای هر دو داده آموزش و ارزیابی این حالت را دارد نه یکی از آن‌ها.پس باعث می‌شود در نهایت دقت خوبی داشته باشیم.

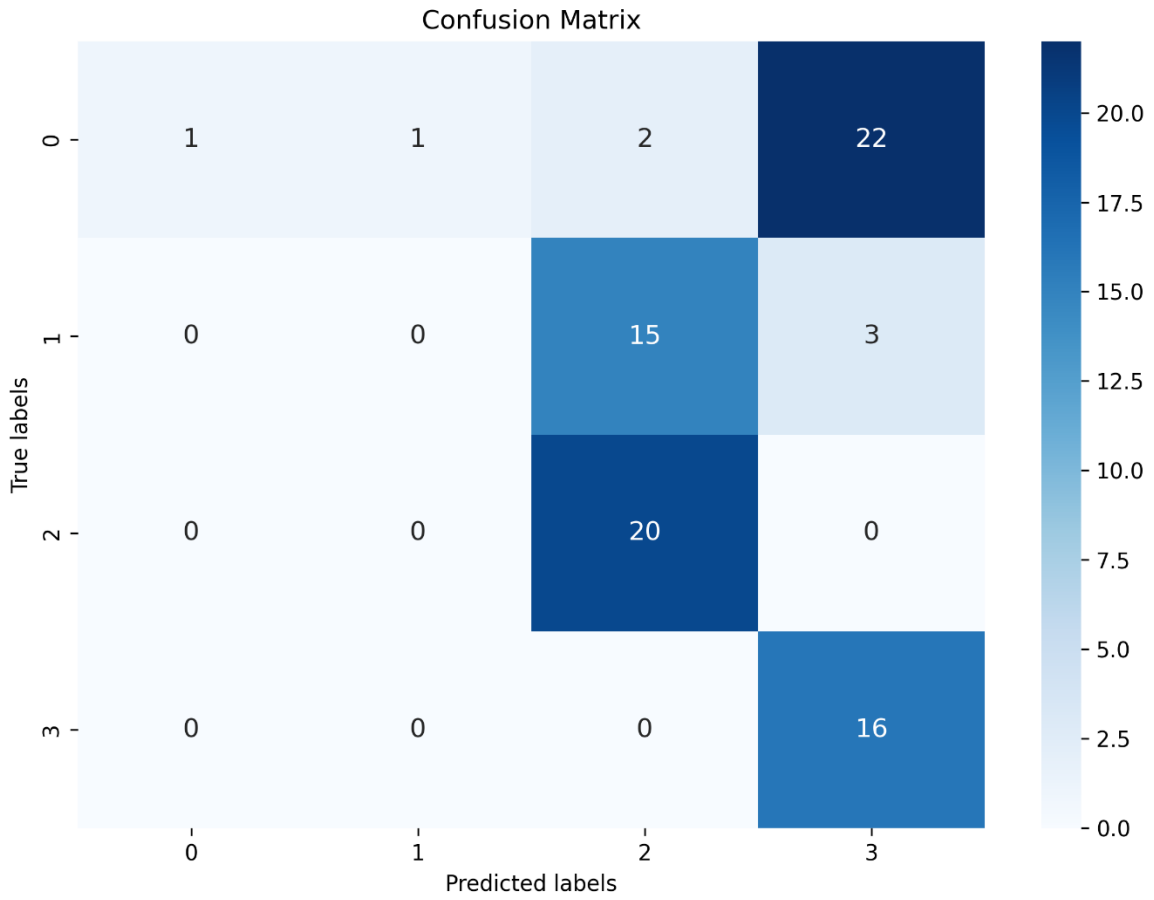
---

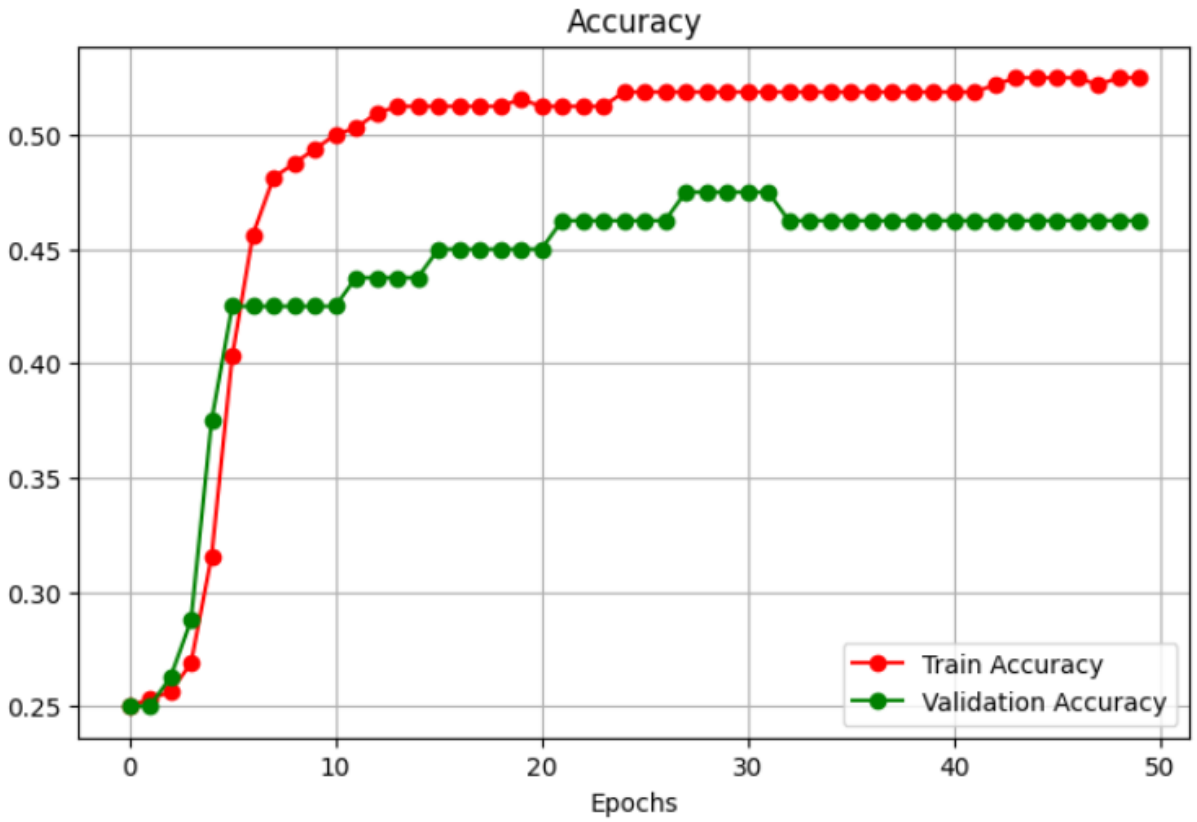
۳. فرآیند سوال قبل را با یک بهینه‌ساز و تابع اتلاف جدید انجام داده و نتایج را مقایسه و تحلیل کنید. بررسی کنید که آیا تغییر تابع اتلاف می‌تواند در نتیجه اثرگذار باشد؟

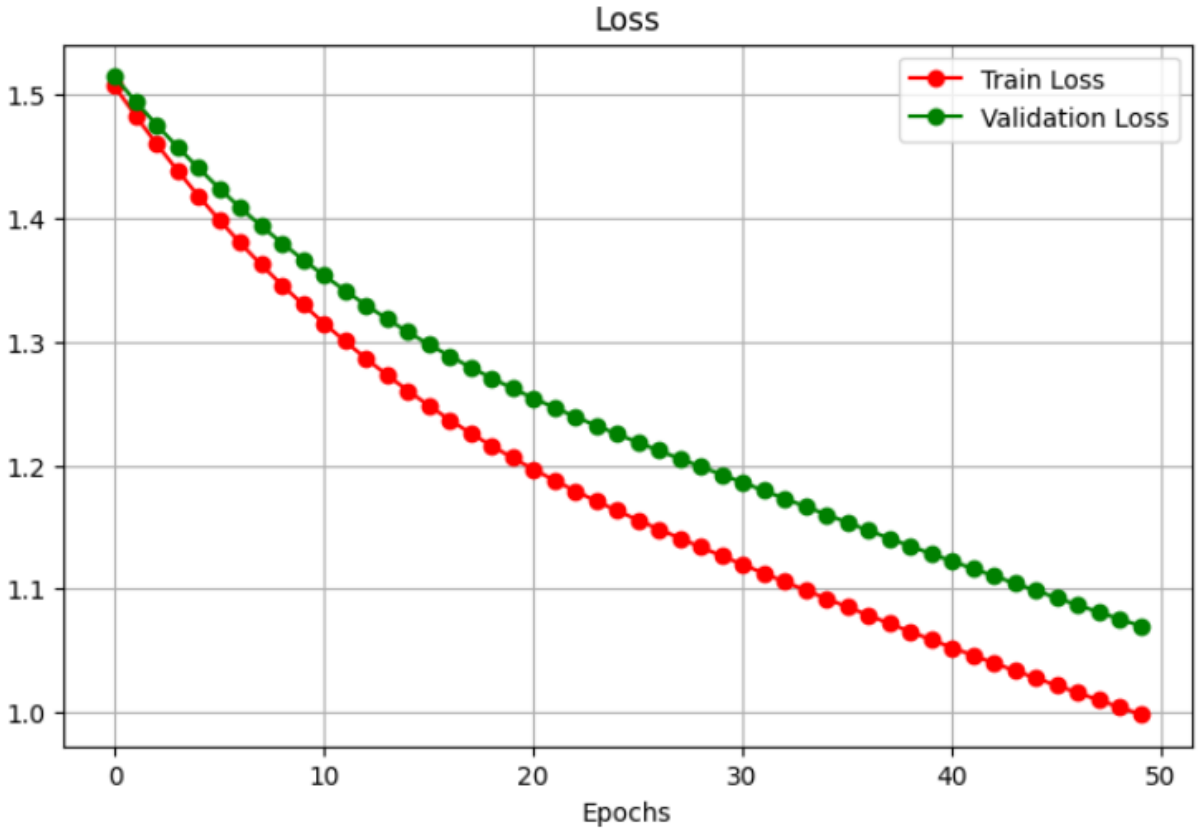
حالت اول:

تابع اتلاف بدون تغییر و تابع بهینه ساز تغییر به SGD

نتایج به صورت زیر است:


Confusion Matrix
True labels
Predicted labels
0
1
2
3
1
1
2
22
0
0
15
3
0
0
20
0
0
0
0
16
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0


Accuracy
0.50
0.45
0.40
0.35
0.30
0.25
0
10
20
30
40
50
Epochs
Train Accuracy
Validation Accuracy

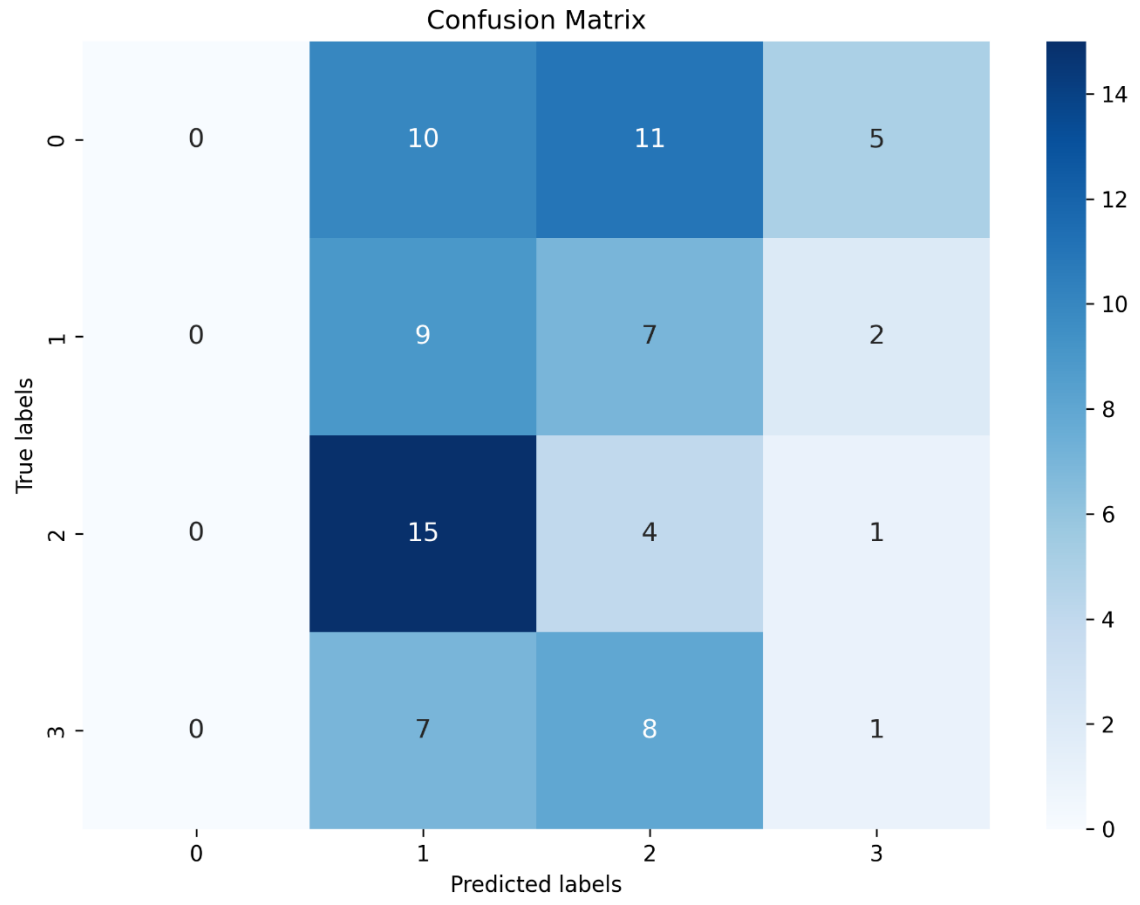

Loss
1.5
1.4
1.3
1.2
1.1
1.0
0
10
20
30
40
50
Epochs
Train Loss
Validation Loss

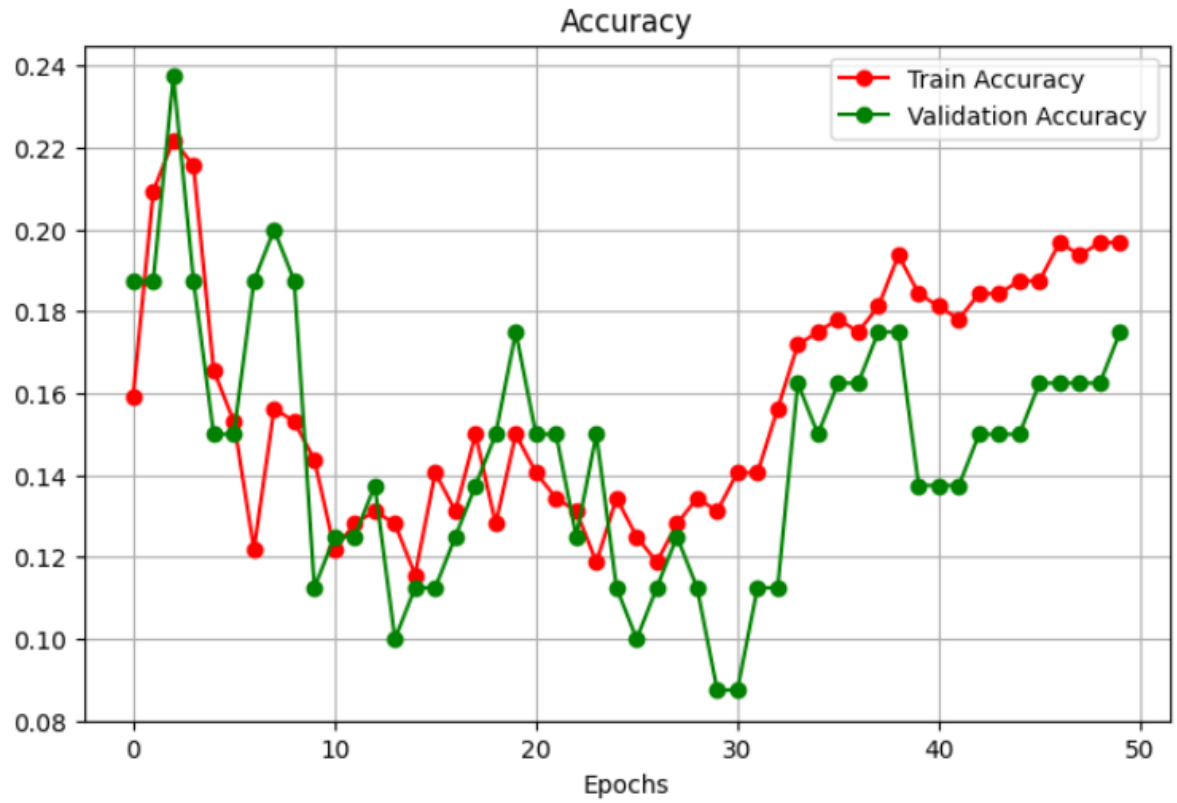
همان‌طور که مشاهده می‌شود در این حالت نیز دقت افزایشی و اتلاف کاهشی است اما اصلا نتیجه قابل قبولی به دست نیامده است.به نظر می‌رسد اگر تعداد دوره‌های آموزش افزایش یابد،مدل می‌تواند عملکرد بهتری داشته باشد و مشکل این باشد که بهینه‌ساز نتوانسته با سرعت خوبی کار کند.

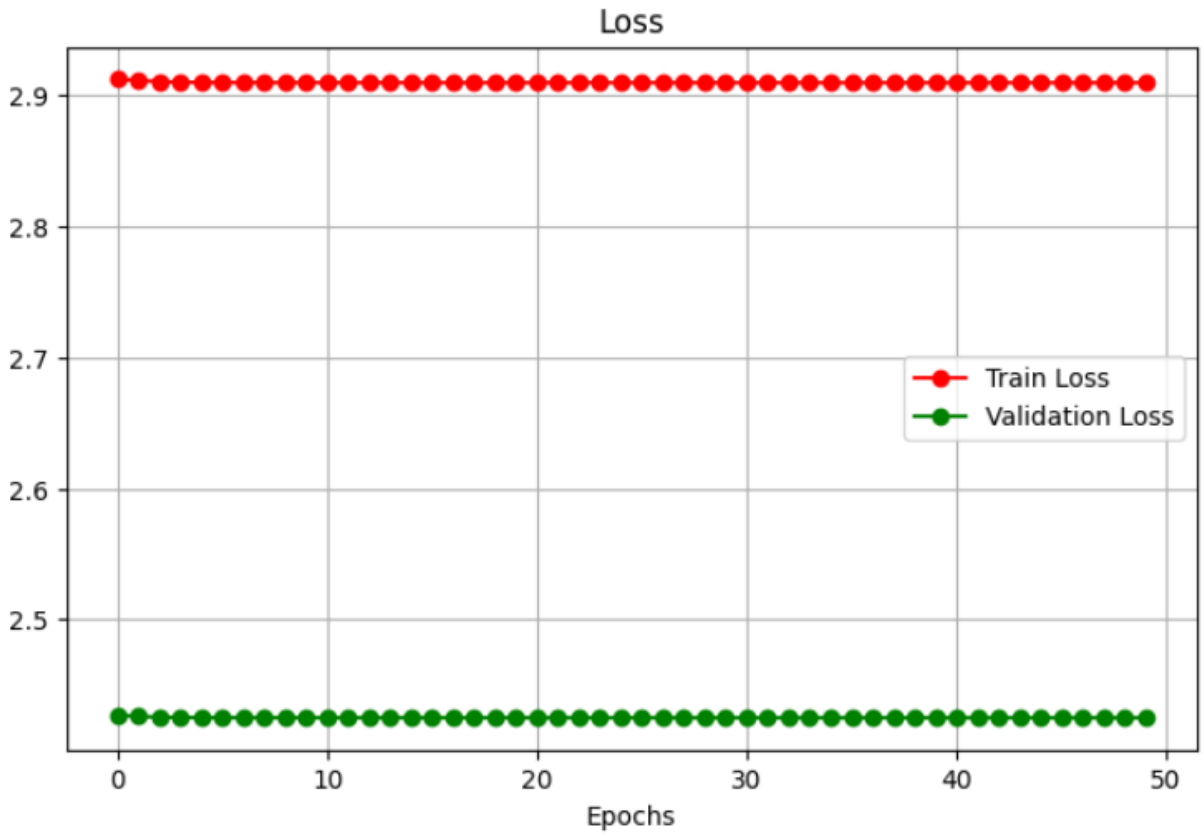
حالت دوم:

تابع اتلاف تغییر به MeanSquaredError() و تابع بهینه‌ساز بدون تغییر

نتیجه به صورت زیر است:



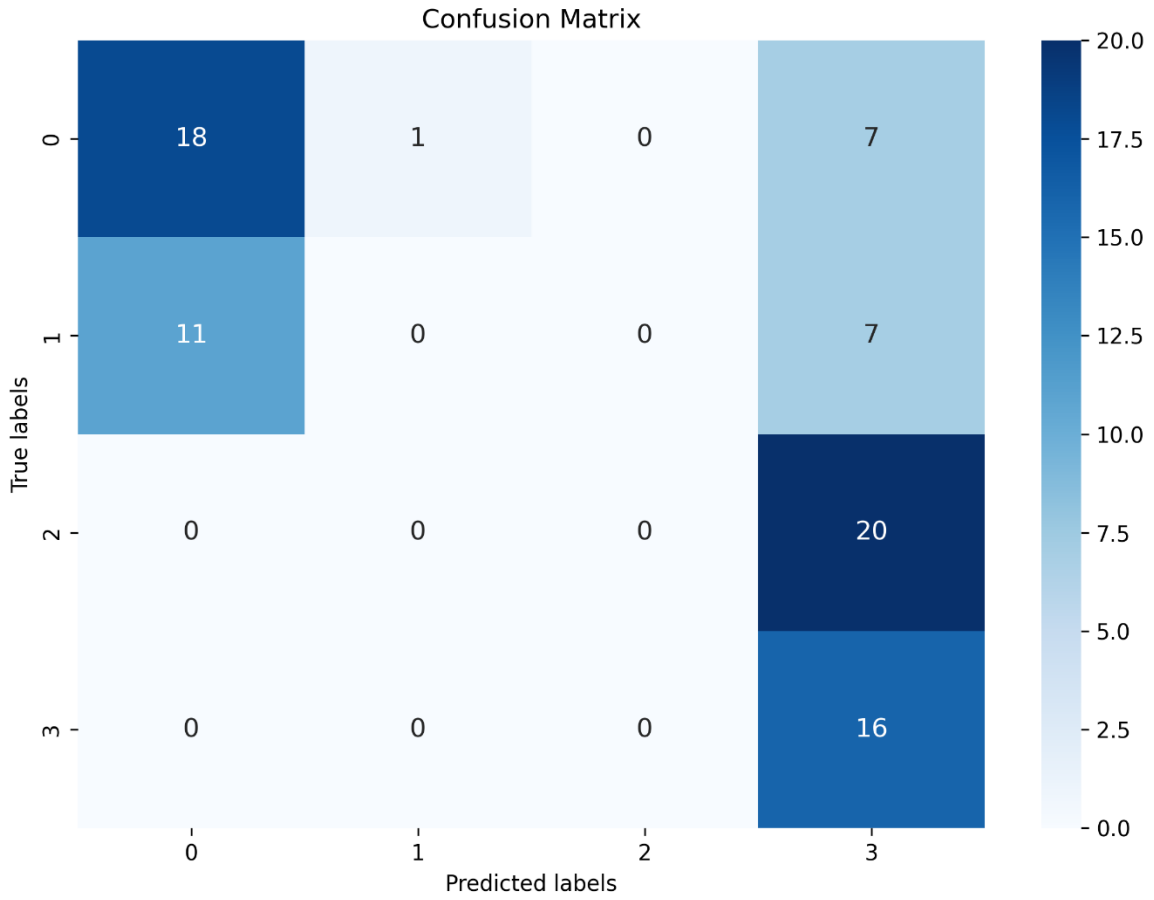

Accuracy
Train Accuracy
Validation Accuracy
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0
10
20
30
40
50
Epochs

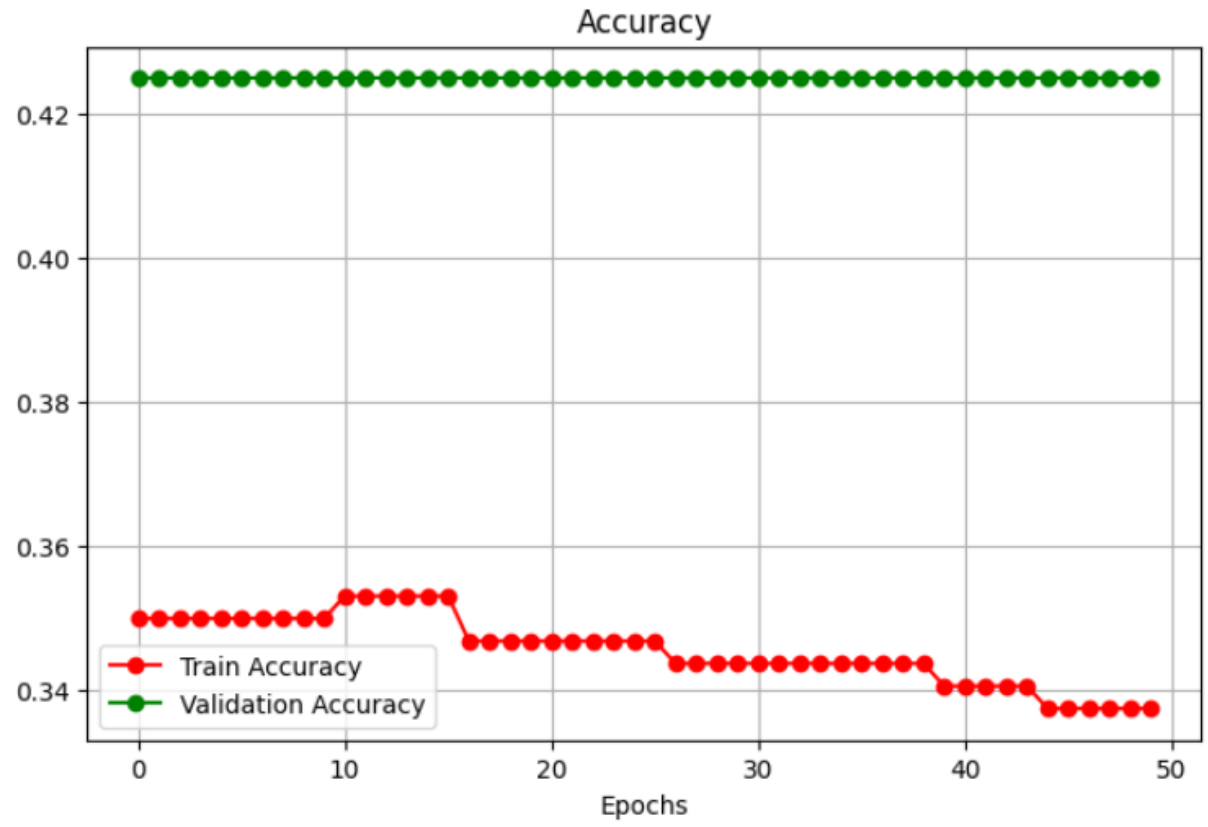

Loss
Train Loss
Validation Loss
2.9
2.8
2.7
2.6
2.5
0
10
20
30
40
50
Epochs

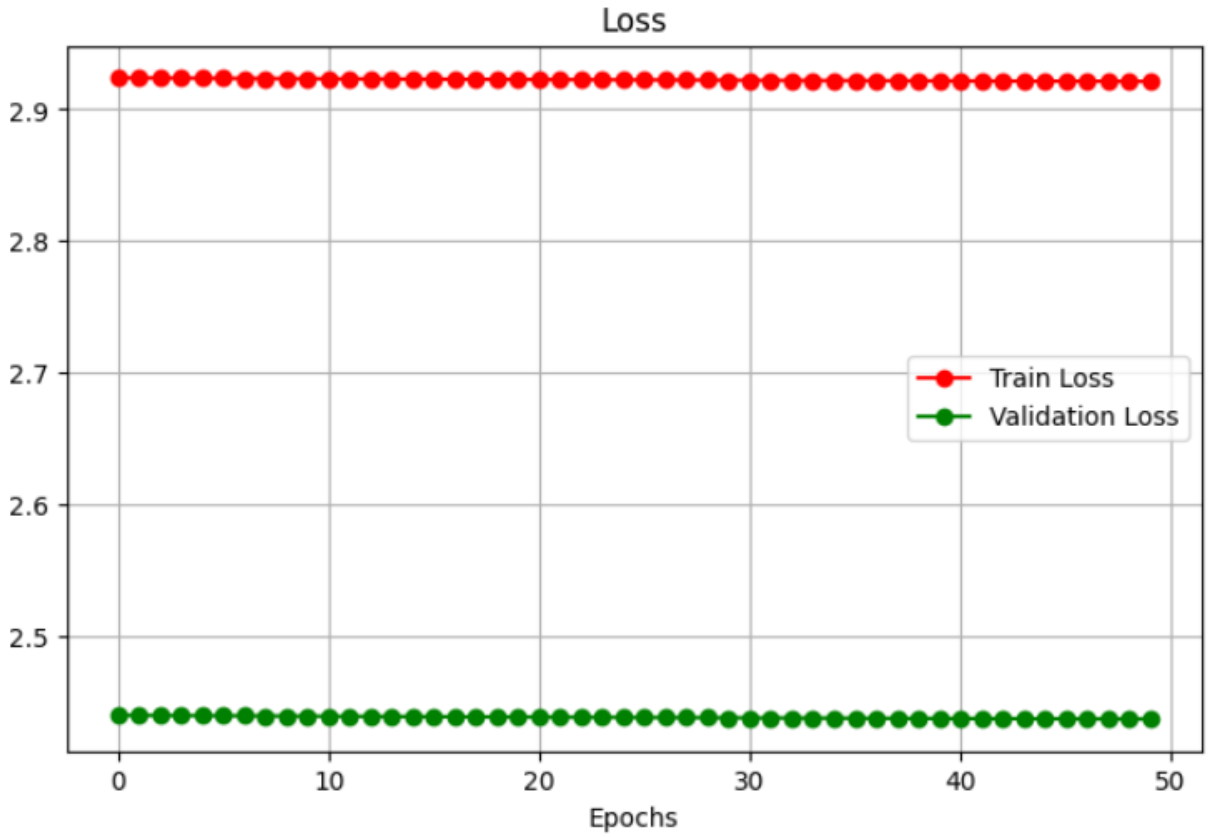
همان طور که مشاهده می‌شود در این حالت اصلا آموزش به خوبی صورت نگرفته و عملکرد بسیار بد است.

حالت سوم:

تابع اتلاف تغییر به MeanSquaredError() و تابع بهینه‌ساز تغییر به SGD




Accuracy
0.42
0.40
0.38
0.36
0.34
Train Accuracy
Validation Accuracy
0
10
20
30
40
50
Epochs


Loss
2.9
2.8
2.7
2.6
2.5
Train Loss
Validation Loss
0
10
20
30
40
50
Epochs

در این حالت هم همان‌طور که مشاهده می‌شود مثل حالت قبل اصلا آموزش به خوبی صورت نگرفته و عملکرد ضعیفی داریم.به نظر می‌رسد این حالت و حالت قبل افت عملکرد به دلیل تغییر تابع اتلاف باشد.

همچنین برا این قسمت به عنوان کاری دیگر،تعریف مدل و آموزش و ارزیابی آن با Pytorch انجام شده که کد به صورت زیر است:

```
def accuracy_fn(y_true, y_pred):
    correct = torch.eq(y_true, y_pred).sum().item()
    acc = (correct / len(y_pred)) * 100
    return acc
torch.manual_seed(74)

from torch import nn
device = "cuda" if torch.cuda.is_available() else "cpu"
class fault(nn.Module):
    def __init__(self, input_features, output_features, hidden_units=8):
        super().__init__()
        self.linear_layer_stack = nn.Sequential(
            nn.Linear(in_features=input_features,
out_features=hidden_units),
            nn.RReLU(),
            nn.Linear(in_features=hidden_units,
out_features=hidden_units),
            nn.RReLU(),
            nn.Linear(in_features=hidden_units,
out_features=output_features),
        )

    def forward(self, x):
        return self.linear_layer_stack(x)


model_pytorch = fault(input_features=10,
                      output_features=4,
                      hidden_units=8).to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_pytorch.parameters(),
                             lr=0.01)
print(y_train.shape)
torch.manual_seed(42)
```

```
device = "cuda" if torch.cuda.is_available() else "cpu"
X_tr_normal = torch.Tensor(X_tr_normal)
X_tr_normal = X_tr_normal.type(torch.float)
X_te_normal = torch.Tensor(X_te_normal)
X_te_normal = X_te_normal.type(torch.float)
y_train = torch.Tensor(y_train)
y_train = y_train.reshape(320 ,).type(torch.float)
y_test = torch.Tensor(y_test)
y_test = y_test.reshape(80 ,).type(torch.float)
X_tr_normal, y_train = X_tr_normal.to(device), y_train.to(device)
X_te_normal, y_test = X_te_normal.to(device), y_test.to(device)

epochs = 70
for epoch in range(epochs):

    model_pytorch.train()



    y_logits = model_pytorch(torch.Tensor(X_tr_normal))
    y_pred = torch.softmax(y_logits, dim=1).argmax(dim=1)

    y_train = torch.Tensor(y_train)
    y_train = y_train.reshape(320 ,).type(torch.long)

    loss = loss_fn(y_logits, torch.Tensor(y_train))
    acc = accuracy_fn(y_true=torch.Tensor(y_train),
                      y_pred=y_pred)


    optimizer.zero_grad()


    loss.backward()


    optimizer.step()


    model_pytorch.eval()
    with torch.inference_mode():

      test_logits = model_pytorch(torch.Tensor(X_te_normal))
      test_pred = torch.softmax(test_logits, dim=1).argmax(dim=1)
```

```
      y_test = torch.Tensor(y_test)
      y_test = y_test.reshape(80 ,).type(torch.long)
      test_loss = loss_fn(test_logits, y_test)
      test_acc = accuracy_fn(y_true=torch.Tensor(y_test),
                             y_pred=test_pred)


    if epoch % 10 == 0:
        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Acc: {acc:.2f}% | Test
Loss: {test_loss:.5f}, Test Acc: {test_acc:.2f}%")
```

و نتیجه به صورت زیر است:

```
Epoch: 0 | Loss: 1.39343, Acc: 25.62% | Test Loss: 1.38521, Test Acc:
22.50%
Epoch: 10 | Loss: 1.05705, Acc: 51.88% | Test Loss: 1.06839, Test Acc:
42.50%
Epoch: 20 | Loss: 0.68278, Acc: 60.94% | Test Loss: 0.70015, Test Acc:
56.25%
Epoch: 30 | Loss: 0.48987, Acc: 73.44% | Test Loss: 0.50146, Test Acc:
71.25%
Epoch: 40 | Loss: 0.39812, Acc: 82.50% | Test Loss: 0.38940, Test Acc:
80.00%
Epoch: 50 | Loss: 0.32620, Acc: 87.50% | Test Loss: 0.31704, Test Acc:
86.25%
Epoch: 60 | Loss: 0.25738, Acc: 91.88% | Test Loss: 0.23098, Test Acc:
95.00%
```

همان‌طور که مشاهده می‌شود،در نهایت به دقت ۹۵ درصد رسیده‌ایم که دقت خوبی است.

---

۴. در مورد K-Fold Cross-validation و Stratified K-Fold Cross-validation و مزایای هریک توضیح دهید. سپس با ذکر دلیل، یکی از این روش‌ها را انتخاب کرده و بخش «۲» سوال سوم را با آن پیاده‌سازی کنید و نتایج خود را تحلیل کنید.

K_fold Cross_validation

در این روش دیتاست به K دسته‌ی مساوی تقسیم می‌شود و مدل روی هر یک از این دسته‌ها آموزش می‌بیند و در نهایت میانگین دقت‌ها در نظر گرفته می‌شود.مثلا اگر ۱۰۰۰ داده داشته

باشیم و بخواهیم به 10 fold آن را تقسیم کنیم،در هر کدام ۱۰۰ داده‌ی تست در نظر گرفته می‌شود و مابقی داده‌ها که داده‌ی تست نیستند،برای آموزش استفاده می‌شوند.یعنی ۱۰۰۰ داده‌ی ما به ۱۰ دسته‌ی ۱۰۰تایی تقسیم و هربار یکی از این دسته‌های ۱۰۰تایی به عنوان داده‌ی تست و ۹۰۰ داده‌ی باقی مانده برای آموزش استفاده می‌شود.به این معنی که ۱۰ دقت مختلف به دست می‌آید و از آن‌ها میانگین گرفته می‌شود.

مزیت این روش این است که داده به شکل‌های مختلف به دو دسته‌ی آموزش و ارزیابی تقسیم می‌شود عملکرد مدل بررسی می‌شود و نتیجه‌ی نهایی بسیار قابل اطمینان تر نسبت به زمانی است که فقط به یک صورت این کار انجام می‌شود.اما در این روش ممکن است توزیع آماری دسته‌ها همانند داده اصلی نباشد و نتایج مختلفی از دسته‌های مختلف به دست آید که میانگین آن‌ها معیار خوبی به ما ندهد.

Stratified K_fold Cross_validation

این روش نیز همانند روش قبل است ولی با این تفاوت که دسته‌هایی که در این روش انتخاب می‌شوند به این صورت است که اگر مثلا در دیتای اصلی ۲۰ درصد از داده‌ها از کلاس۱ و ۸۰ درصد از کلاس۲ باشند در این دسته‌ها نیز این توزیع وجود دارد.یعنی در هر دسته ۲۰ درصد از کلاس۱ و ۸۰ درصد از کلاس۲ هستند.

مزیت این روش این است که در هر سری آموزش با یکی از دسته داده‌ها توزیع آماری همانند داده اصلی وجود دارد که باعث می‌شود مدل نسبت به داده موردنظر آموزش خیلی بهتری ببیند اما عیب این است که ممکن است روی داده‌ی دیگر با توزیع آماری متفاوت،عملکرد ضعیف‌تری داشته باشد.

به همین دلیل برای سوال۳ بخش۲ از روش اول استفاده می‌شود.

سوال۳)

یکی از مجموعه‌داده‌های مربوط به طبقه‌بندی پوشش جنگلی یا دارو را در نظر بگیرید.

۱. با استفاده از بخشی از داده‌ها، مجموعه‌داده را به دو بخش آموزش و آزمون تقسیم کنید (حداقل ۱۵ درصد از داده‌ها را برای آزمون نگه دارید). توضیح دهید که از چه روشی برای انتخاب بخشی از داده‌ها استفاده کرده‌اید. آیا روش بهتری برای این کار می‌شناسید؟
در ادامه، برنامه‌ای بنویسید که درخت تصمیمی برای طبقه‌بندی کلاس‌های این مجموعه‌داده طراحی کند. خروجی درخت تصمیم خود را با برنامه‌نویسی و یا به‌صورت دستی تحلیل کنید.

کد به صورت زیر است:

```
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn import tree
data_test_drug = pd.read_csv('/content/drug_data.csv')
y_drug = data_test_drug[['Drug']].values
X_drug = data_test_drug[['Age' , 'Sex' , 'BP' , 'Cholesterol' ,
'Na_to_K']].values
X_pick , X_not_pick , y_pick , y_not_pick = train_test_split(X_drug ,
                                                             y_drug ,
                                                             test_size =
0.3)
X_train_drug , X_test_drug , y_train_drug , y_test_drug =
train_test_split(X_pick ,

 y_pick ,

 test_size = 0.2)
```

لازم به ذکر است فایل csv دارئ برای این کار انتخاب شده است.در این فایل ستون‌هایی که در آن‌ها از اسامی استفاده شده است،این اسامی با مقادیر عددی جایگزین شده‌اند تا بتوان با استفاده از درخت تصمیم آن‌ها را دسته‌بندی کرد.

در این کد ابتدا فایل csv مربوط به داده‌ها از google drive فراخوانی می‌شود.سپس ۵ستون اول این فایل به عنوان داده‌ها و ستون آخر به عنوان برچسب در نظر گرفته می‌شود.

سپس به صورت تصادفی ۷۰درصد از داده‌ها به عنوان داده‌ی موردنظر برای انجام فرایند آموزش و ارزیابی انتخاب می‌شوند.می‌توان این روند را طوری انجام داد که دیتاهای انتخاب شده توزیع آماری کل داده را حفظ کنند که روش بهتری برای انتخاب داده است اما در این جا از روش تصادفی استفاده شده است.

سپس این داده‌ها به دو دسته‌ی آموزش و ارزیابی با نسبت ۸ به ۲ تقسیم می‌شوند.

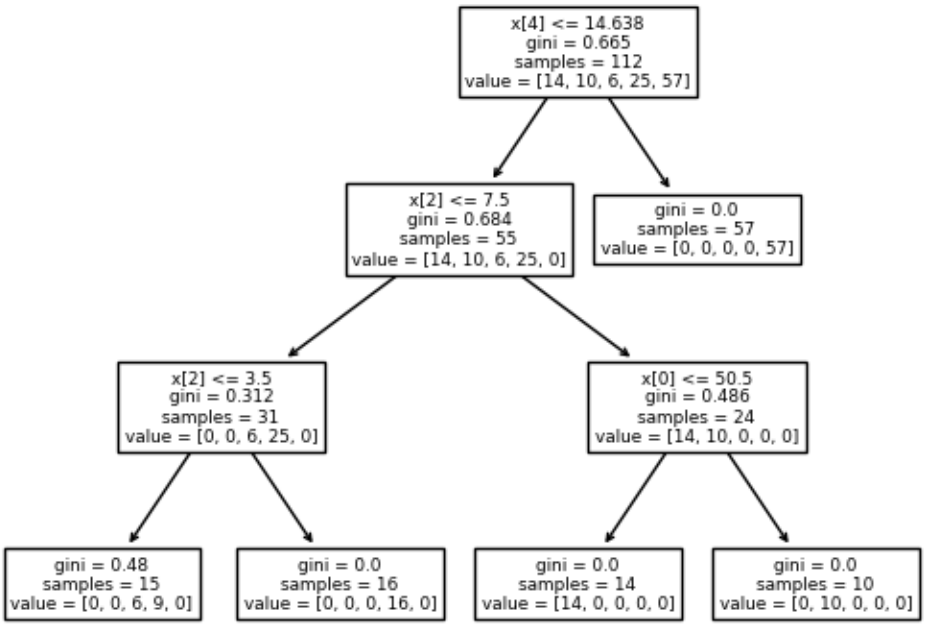
قسمت دوم کد:

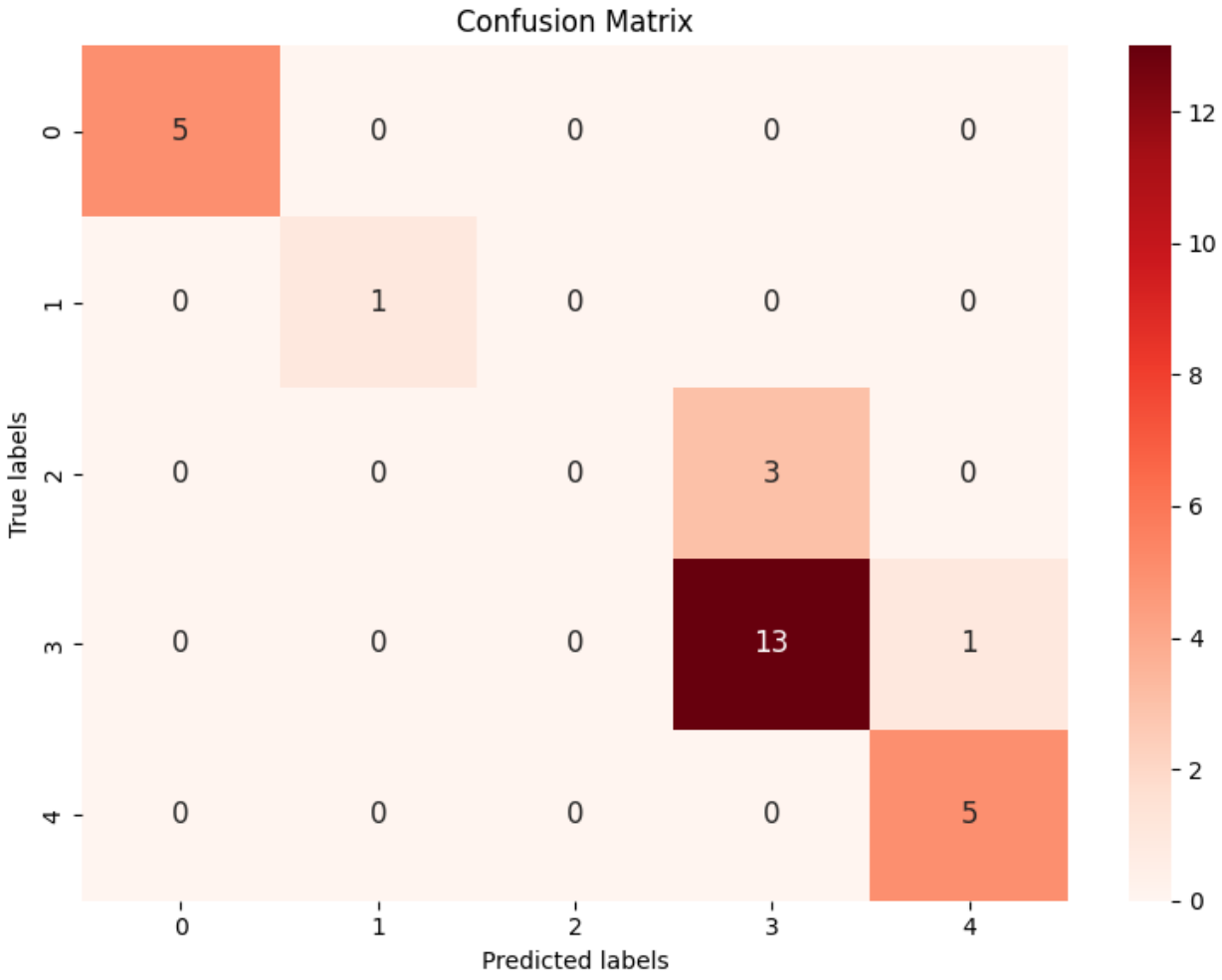
```
clf_dt1 = tree.DecisionTreeClassifier(max_depth=3, random_state=74,
ccp_alpha=0)
clf_dt1.fit(X_train_drug, y_train_drug)
tree.plot_tree(clf_dt1);
print(clf_dt1.score(X_test_drug, y_test_drug))
y_pred_drug1 = clf_dt1.predict(X_test_drug)
print(precision_score(y_test_drug , y_pred_drug1 ,average='micro'))
print(recall_score(y_test_drug , y_pred_drug1 ,average='micro'))
print(f1_score(y_test_drug , y_pred_drug1 ,average='micro'))
print(jaccard_score(y_test_drug , y_pred_drug1 ,average='micro'))
cf_matrix_tree1 = confusion_matrix(y_test_drug, y_pred_drug1)
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix_tree1, annot=True, fmt='d', cmap='Reds',
annot_kws={"size": 12})


plt.gca().set_ylim(len(np.unique(y_test_drug)), 0)
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.tight_layout()
plt.savefig('confusion_matrix_tree1.png', dpi=300)
```

در این قسمت از کد درخت تصمیم با فراپارامترهایی که مشاهده می‌شود ساخته می‌شود.بعد از آن این درخت تصمیم روی داده‌ی ما آموزش داده می‌شود و درخت تصمیم مربوط به آن رسم می‌شود.سپس دقت این مدل گزارش شده و پس از آن چند معیار دیگر نیز برای این مدل چاپ می‌

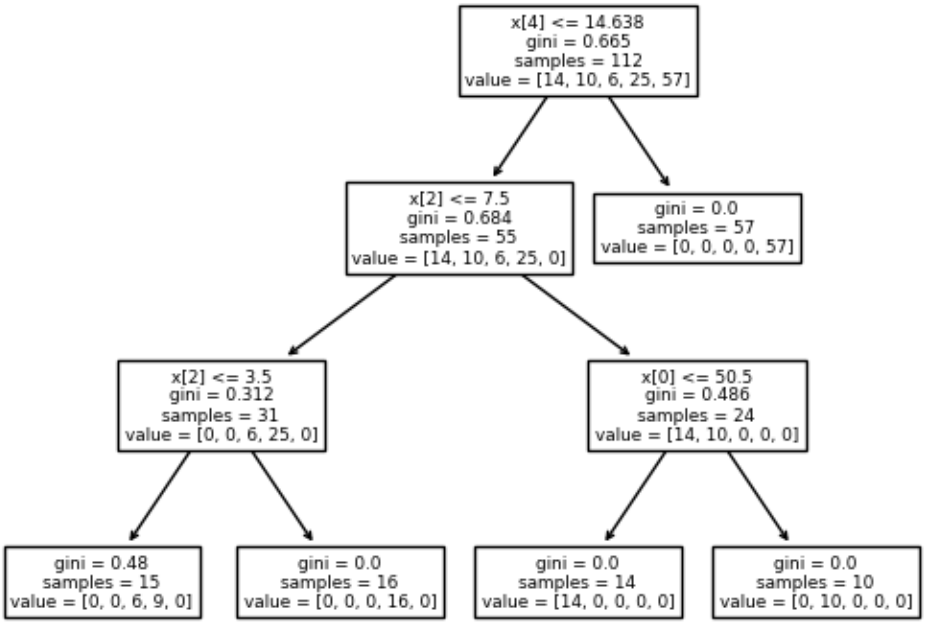
شوند.در نهایت هم ماتریس در هم ریختگی برای این مدل رسم می‌شود که نتیجه به صورت زیر است:

```
0.8571428571428571
0.8571428571428571
0.8571428571428571
0.8571428571428571
0.75
```





تحلیل درخت تصمیم:



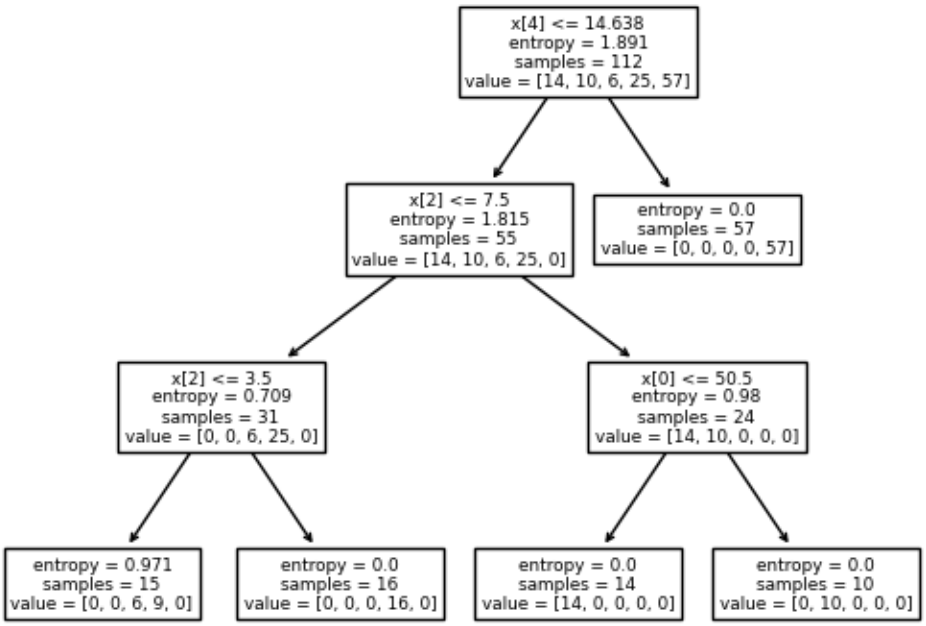
همان‌طور که مشاهده می‌شود،در این درخت ابتدا با توجه به مقدار ویژگی پنجم،اگر مقدار این ویژگی از ۱۴.۶۳۸ بیشتر باشد،داده از کلاس۴ است و در غیر این صورت از کلاس‌های دیگر است.همان‌طور که می‌بینیم،مقدار gini هم برای آن صفر است.پس از آن درخت به سراغ کارکردن با ویژگی سوم می‌رود.طبق آن چه می‌بینیم اگر این ویژگی مقداری بزرگتر از ۷.۵ داشته باشد،داده متعلق به کلاس صفر یا ۱ و اگر کوچکتر از آن باشد،داده متعلق به کلاس ۲ یا ۳ است.در مرحله بعد ابتدا به سراغ شاخه‌ی سمت چپ می‌رویم.در این شاخه،باز هم درخت روی ویژگی سوم کار می‌کند.اگر این ویژگی مقداری بزرگتر از ۳.۵ داشته باشد،داده متعلق به کلاس ۳ خواهد بود و اگر مقدار کمتر از ۳.۵ باشد،داده یا متعلق به کلاس۲ و یا متعلق به کلاس۳ است.در شاخه‌ی طرف دیگر،درخت با ویژگی اول کار می‌کند؛اگر این ویژگی مقداری بزرگتر از ۵۰.۵ داشته باشد،داده متعلق به کلاس۱ و اگر مقداری کوچکتر از آن داشته باشد،متعلق به کلاس صفر است.پس این درخت با سه لایه پیشروی،عملکرد نسبتا خوبی داشته است و مقدار خطای آن نسبتا کم است.

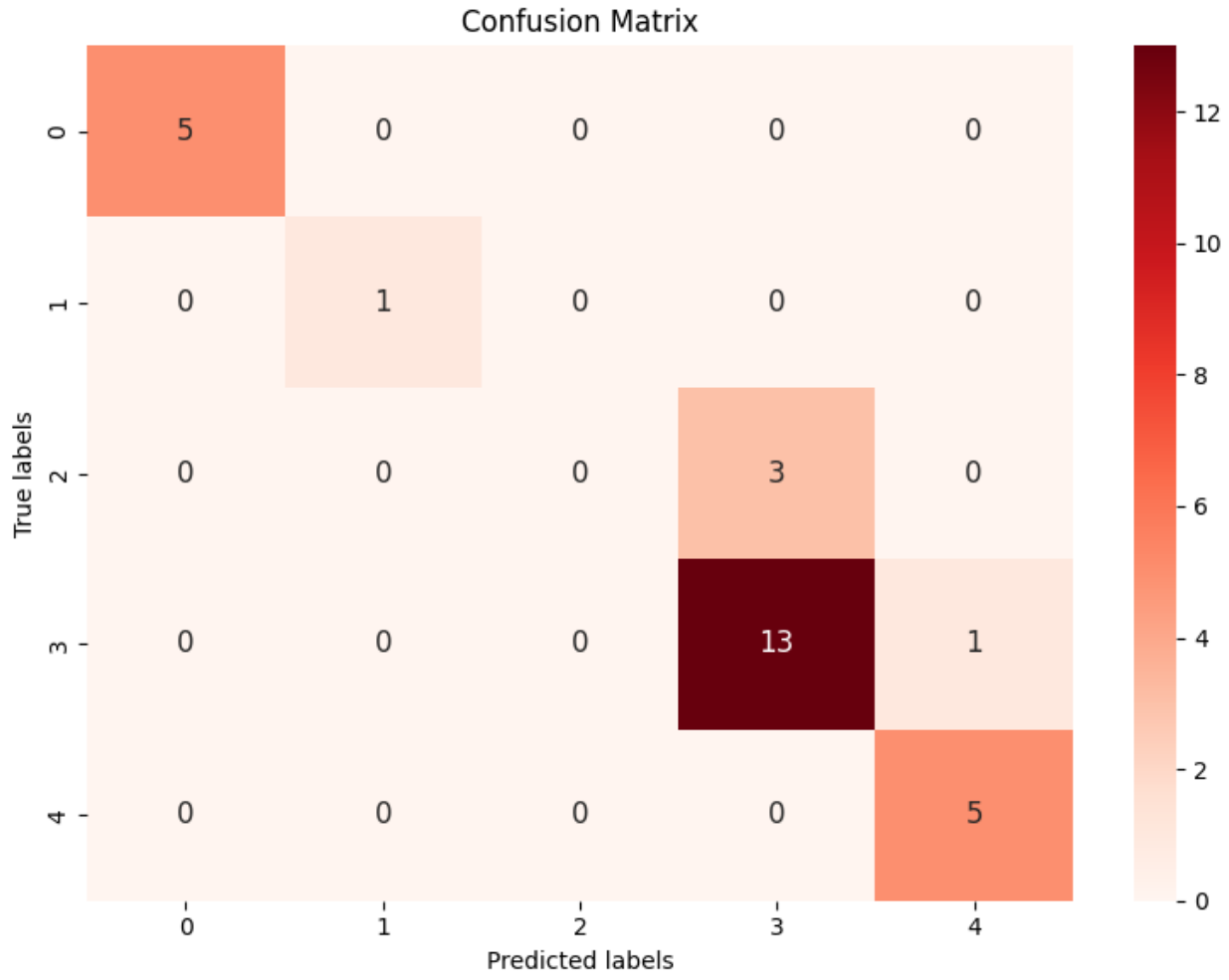
۲. با استفاده از ماتریس درهم‌ریختگی و حداقل سه شاخصهٔ ارزیابی مربوط به وظیفهٔ طبقه‌بندی، عملکرد درخت آموزش‌دادهشدهٔ خود را روی بخش آزمون داده‌ها ارزیابی کنید و نتایج را به‌صورت دقیق گزارش کنید.
تأثیر مقادیر کوچک و بزرگ حداقل دو فراپارامتر را بررسی کنید. تغییر فراپارامترهای مربوط به هرس‌کردن چه تأثیری روی نتایج دارد و مزیت آن چیست؟

در این قسمت،ماتریس درهم‌ریختگی و پنج شاخصه ارزیابی برای ۷ حالت مختلف دیگر غیر از حالت اول از فراپارامترهای درخت تصمیم رسم و محاسبه شده است که در زیر دیده می‌شود:

۱.

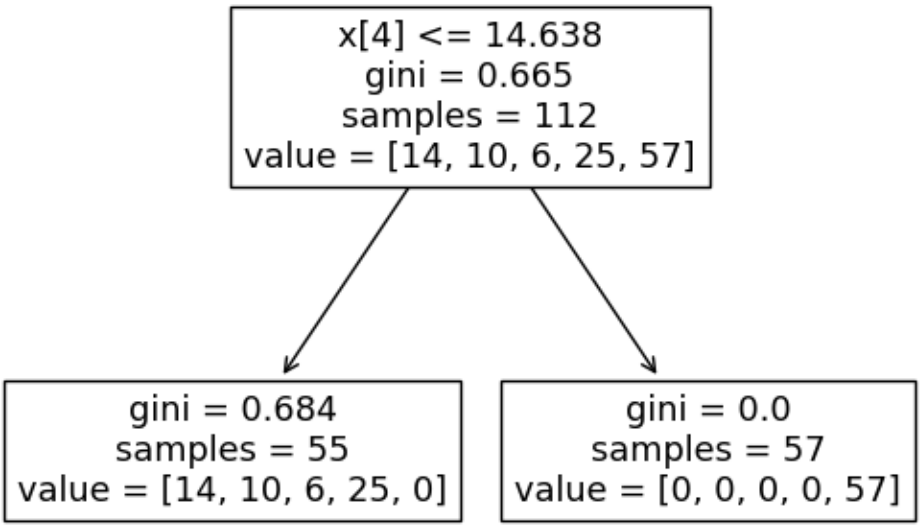
```
clf_dt2 = tree.DecisionTreeClassifier(max_depth=3, random_state=74,
ccp_alpha=0 , criterion = 'entropy')
0.8571428571428571
0.8571428571428571
0.8571428571428571
0.8571428571428571
0.75
```

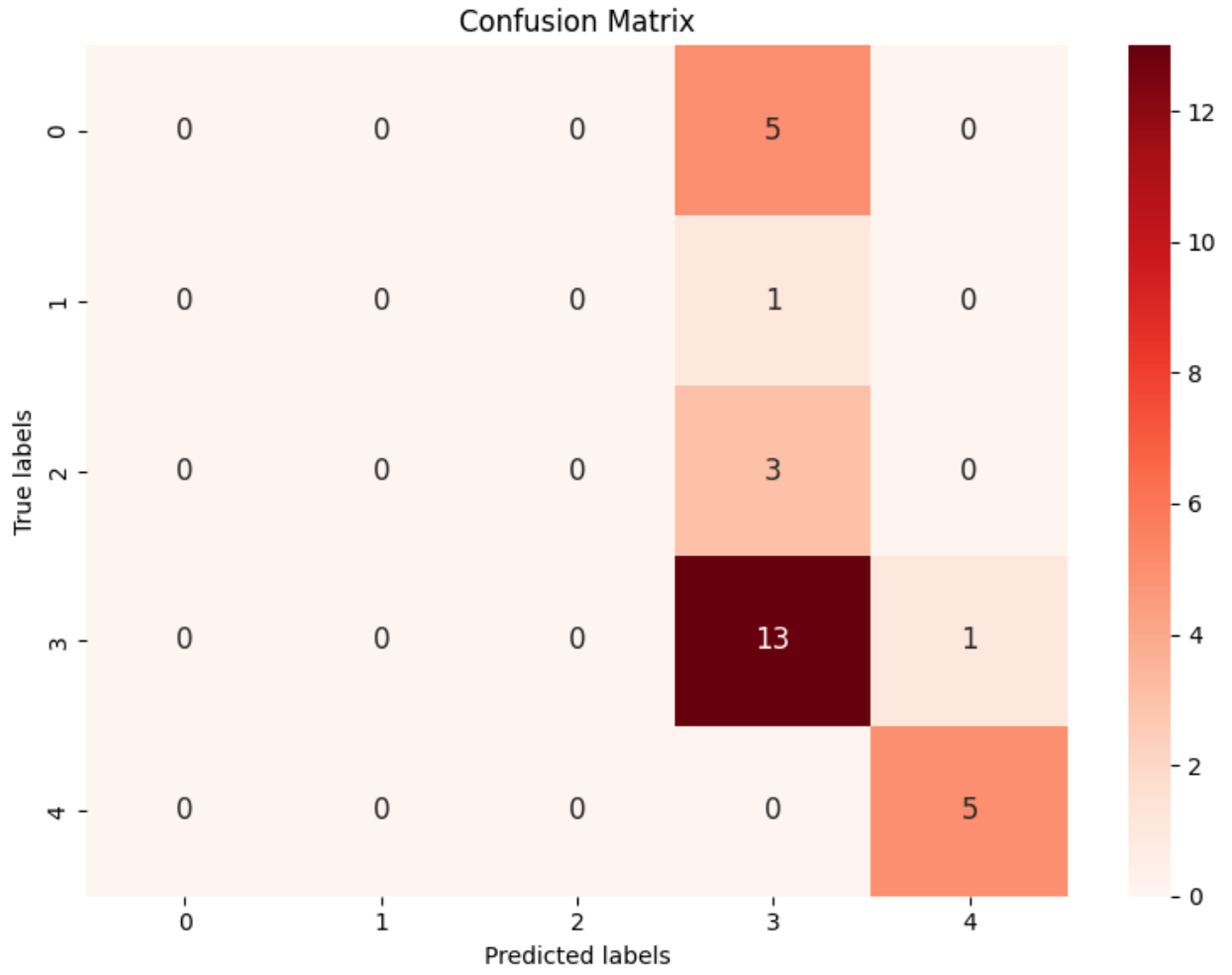




.۲

```
clf_dt3 = tree.DecisionTreeClassifier(max_depth=3, random_state=74,
ccp_alpha=0.3)
0.6428571428571429
0.6428571428571429
0.6428571428571429
0.6428571428571429
0.47368421052631576
```

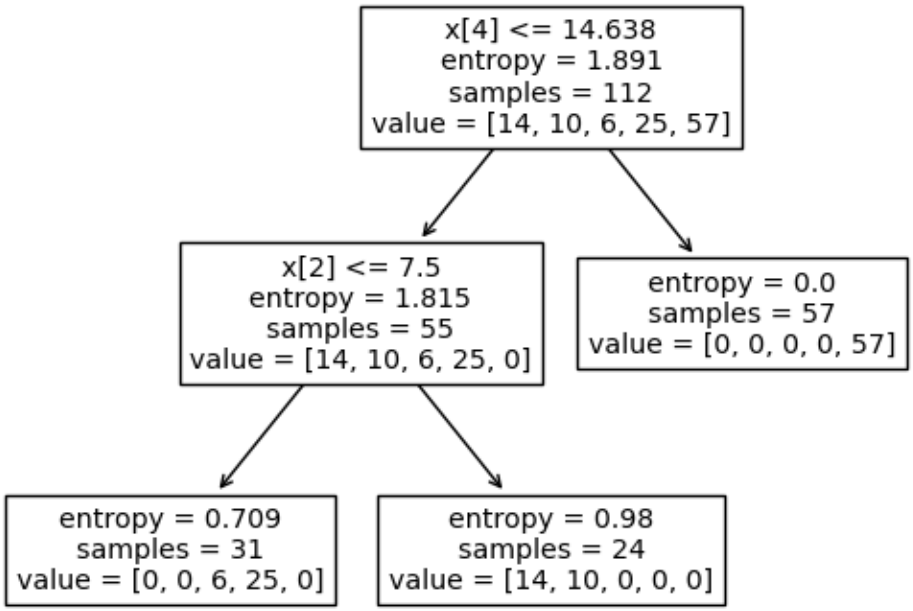

x[4] <= 14.638
gini = 0.665
samples = 112
value = [14, 10, 6, 25, 57]
gini = 0.684
samples = 55
value = [14, 10, 6, 25, 0]
gini = 0.0
samples = 57
value = [0, 0, 0, 0, 57]

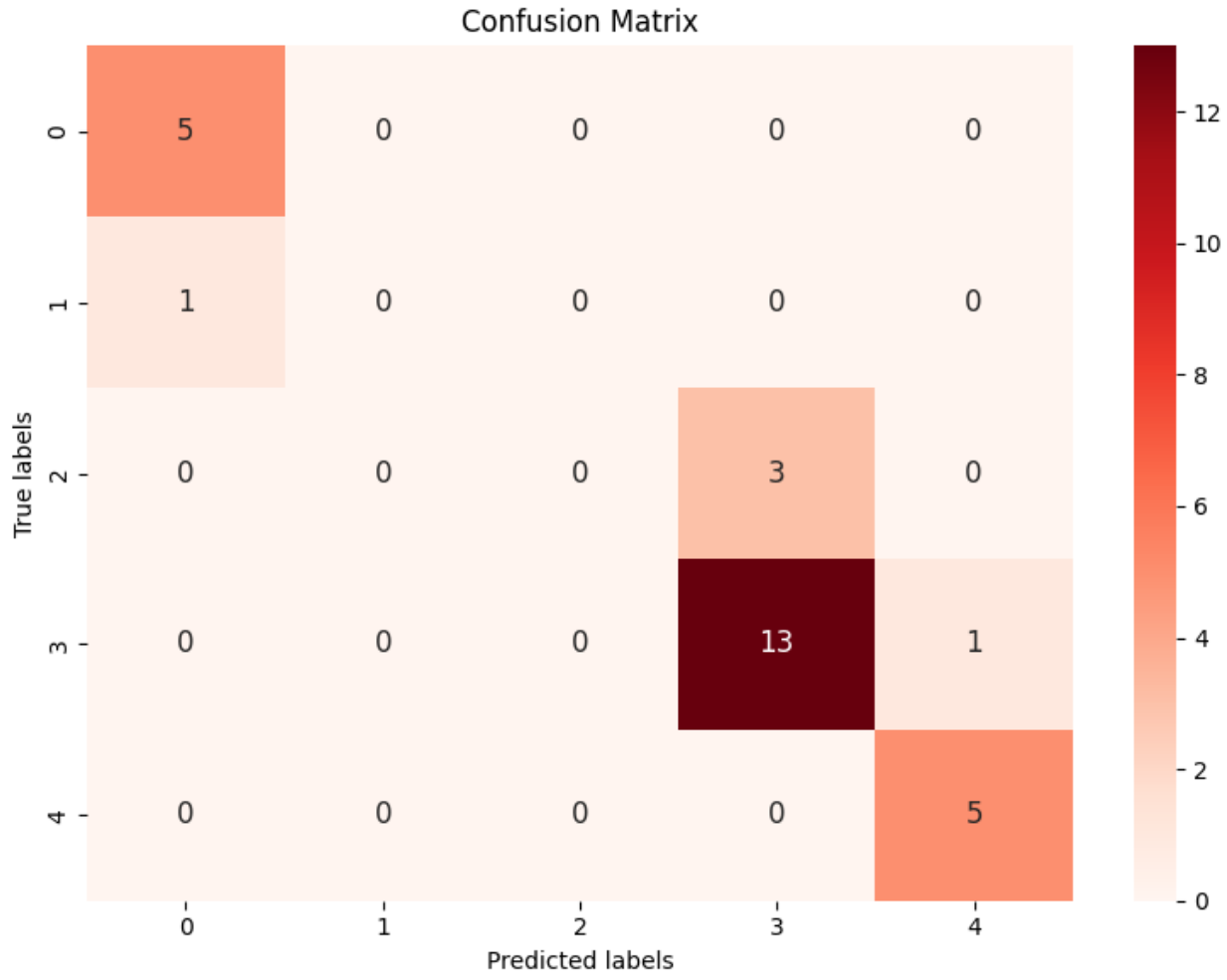


.۳

```
clf_dt4 = tree.DecisionTreeClassifier(max_depth=3, random_state=74,
ccp_alpha=0.3 , criterion = 'entropy')
```

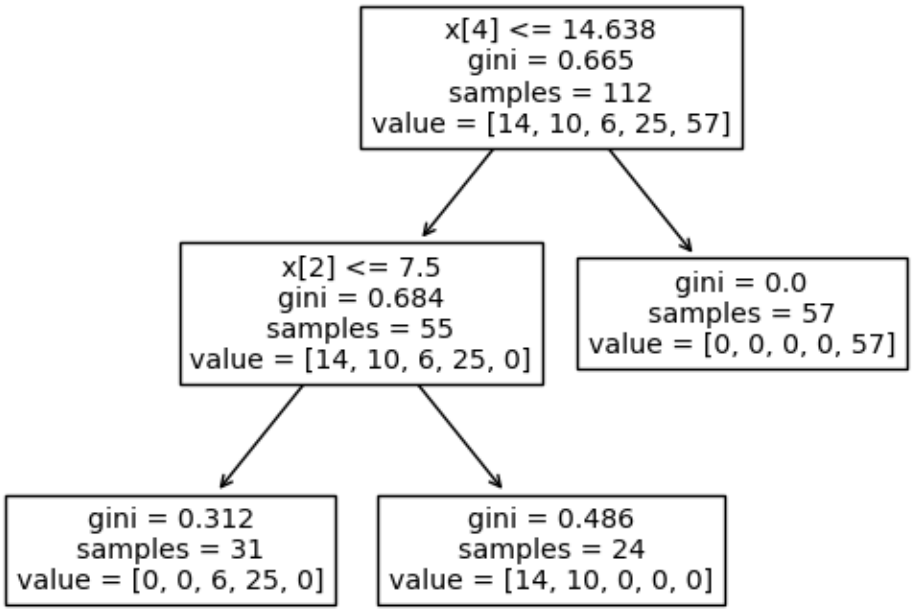
```
0.8214285714285714
0.8214285714285714
0.8214285714285714
0.8214285714285714
0.696969696969697
```

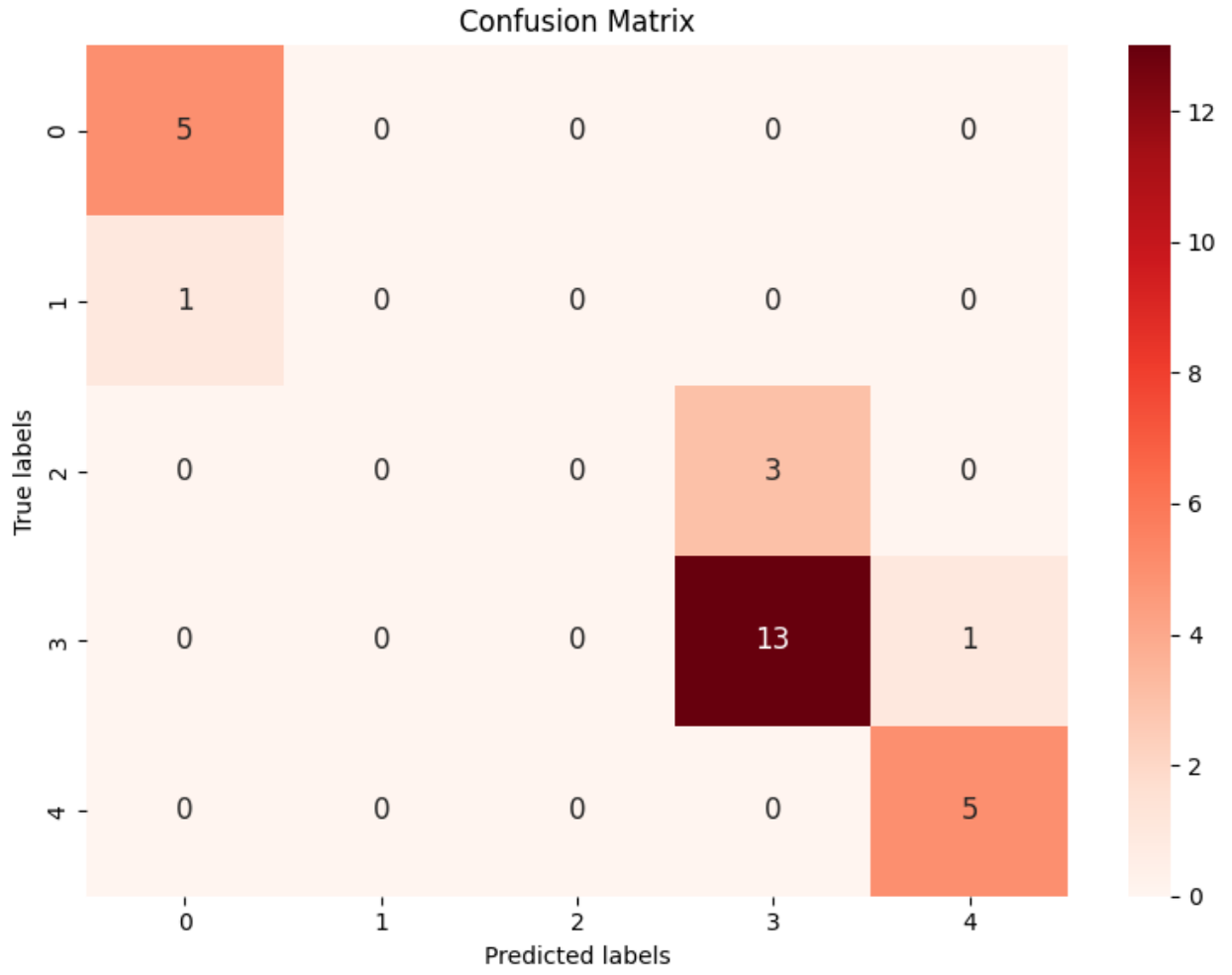

x[4] <= 14.638
entropy = 1.891
samples = 112
value = [14, 10, 6, 25, 57]
x[2] <= 7.5
entropy = 1.815
samples = 55
value = [14, 10, 6, 25, 0]
entropy = 0.0
samples = 57
value = [0, 0, 0, 0, 57]
entropy = 0.709
samples = 31
value = [0, 0, 6, 25, 0]
entropy = 0.98
samples = 24
value = [14, 10, 0, 0, 0]



.۴

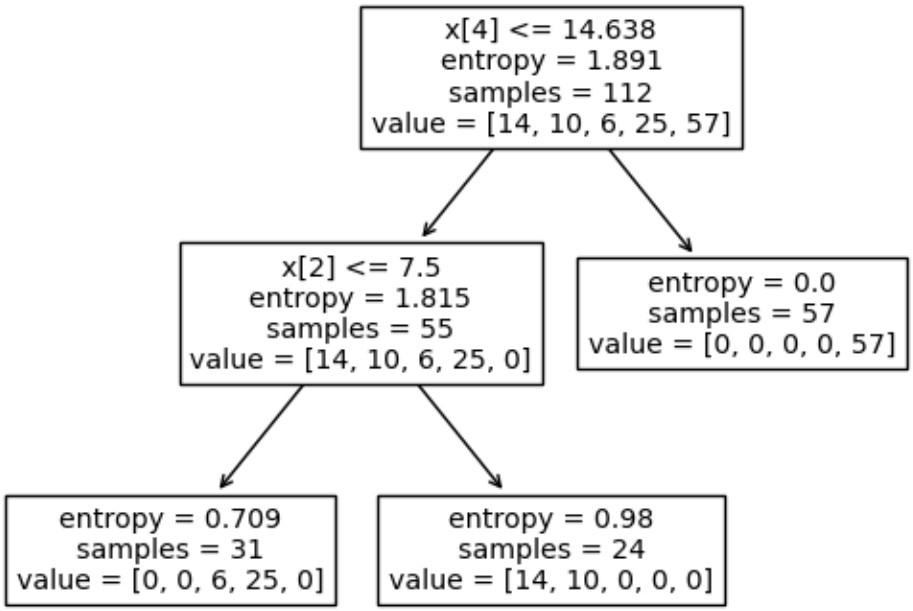
```
clf_dt5 = tree.DecisionTreeClassifier(max_depth=2, random_state=74,
ccp_alpha=0)
0.8214285714285714
0.8214285714285714
0.8214285714285714
0.8214285714285714
0.696969696969697
```

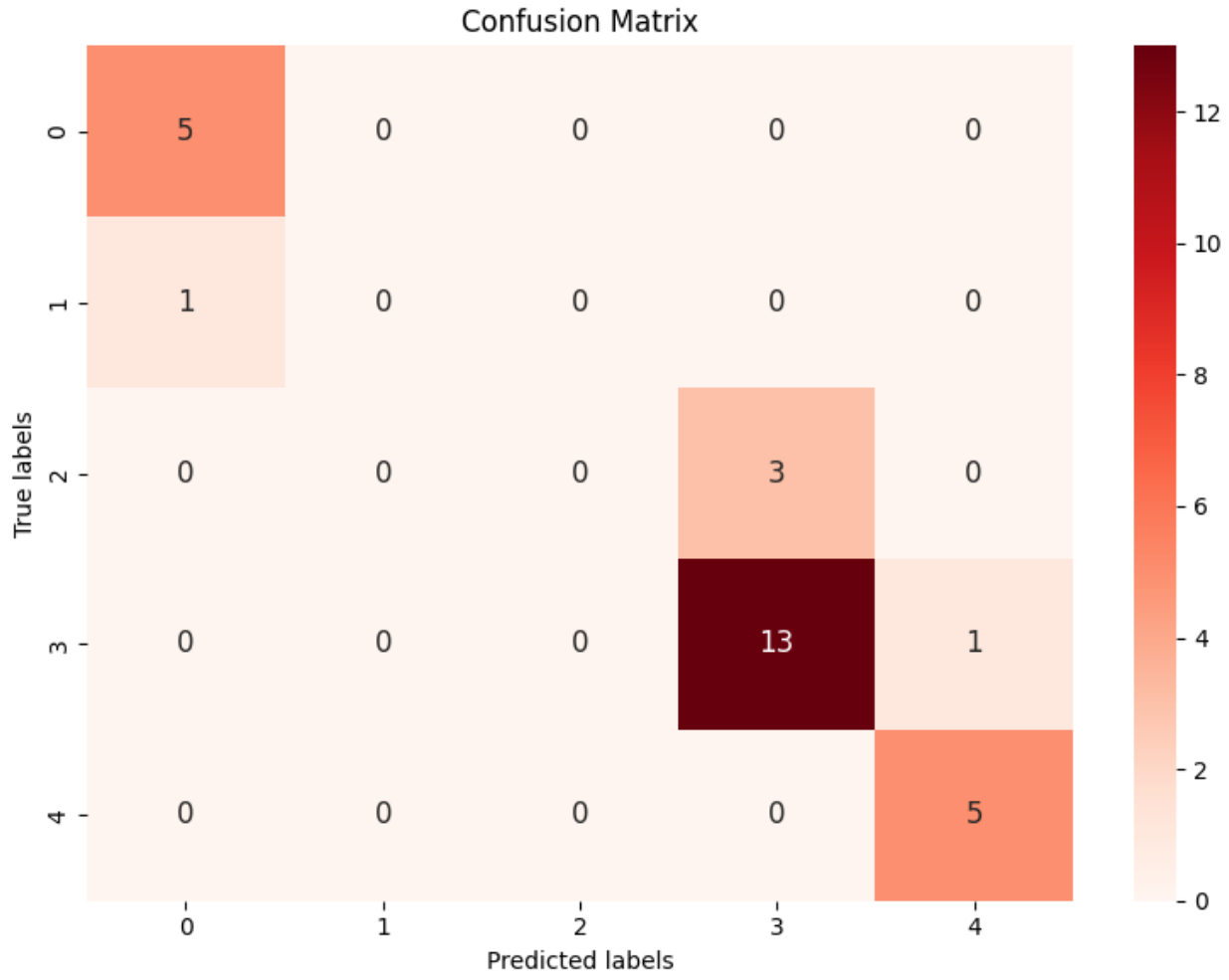

x[4] <= 14.638
gini = 0.665
samples = 112
value = [14, 10, 6, 25, 57]
x[2] <= 7.5
gini = 0.684
samples = 55
value = [14, 10, 6, 25, 0]
gini = 0.0
samples = 57
value = [0, 0, 0, 0, 57]
gini = 0.312
samples = 31
value = [0, 0, 6, 25, 0]
gini = 0.486
samples = 24
value = [14, 10, 0, 0, 0]



.۵

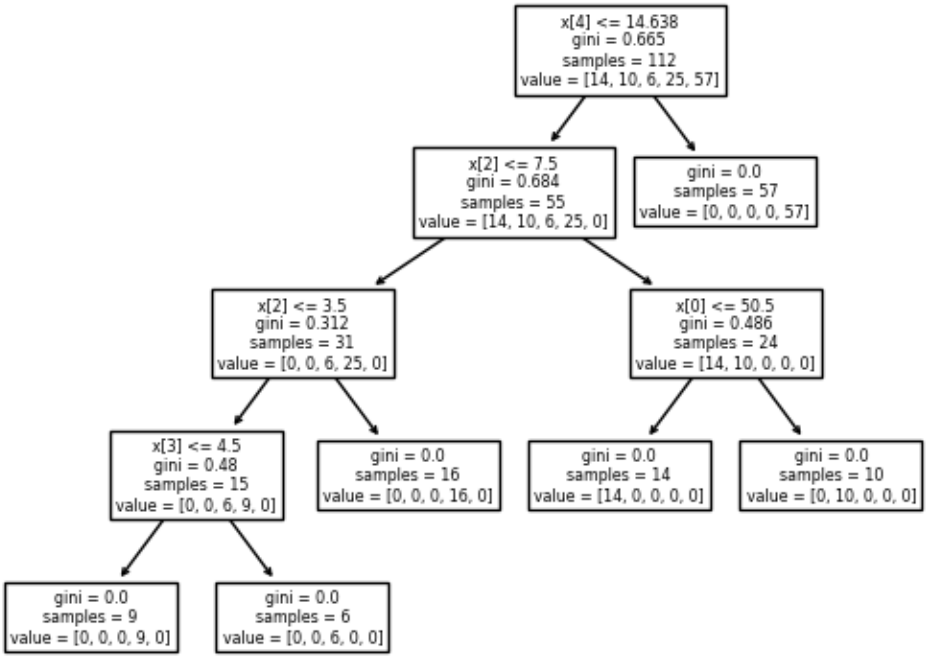
```
clf_dt6 = tree.DecisionTreeClassifier(max_depth=2, random_state=74,
ccp_alpha=0 , criterion = 'entropy')
0.8214285714285714
0.8214285714285714
0.8214285714285714
0.8214285714285714
0.696969696969697
```

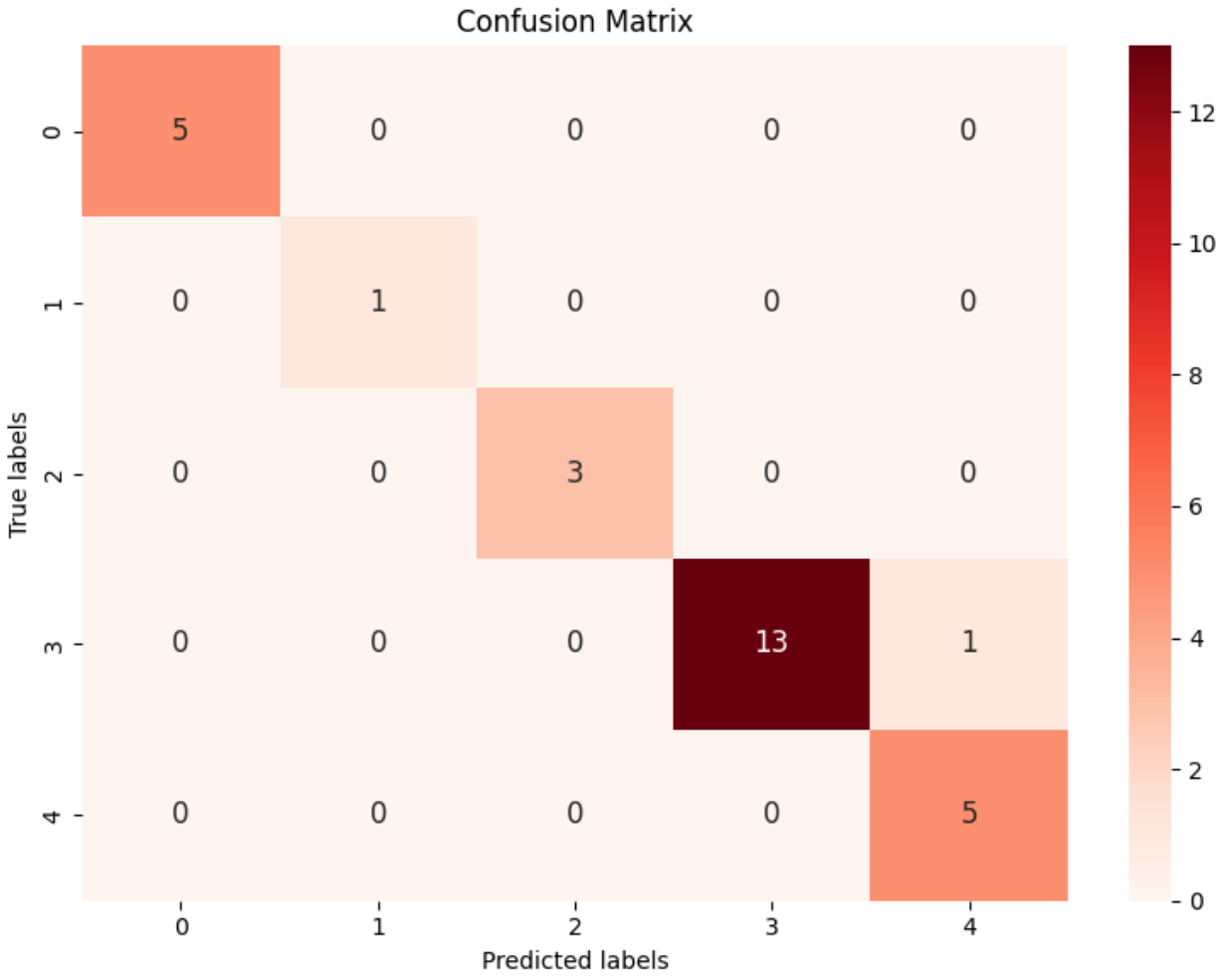

x[4] <= 14.638
entropy = 1.891
samples = 112
value = [14, 10, 6, 25, 57]
x[2] <= 7.5
entropy = 1.815
samples = 55
value = [14, 10, 6, 25, 0]
entropy = 0.0
samples = 57
value = [0, 0, 0, 0, 57]
entropy = 0.709
samples = 31
value = [0, 0, 6, 25, 0]
entropy = 0.98
samples = 24
value = [14, 10, 0, 0, 0]



۶.

```
clf_dt7 = tree.DecisionTreeClassifier(max_depth=4, random_state=74,
ccp_alpha=0)
0.9642857142857143
0.9642857142857143
0.9642857142857143
0.9642857142857143
0.9310344827586207
```

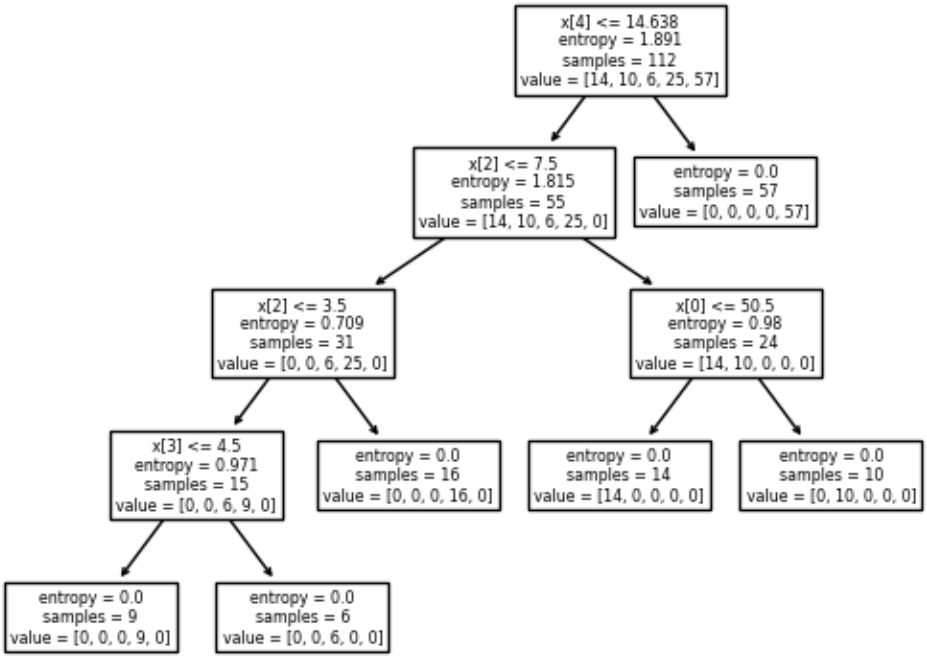

x[4] <= 14.638
gini = 0.665
samples = 112
value = [14, 10, 6, 25, 57]
x[2] <= 7.5
gini = 0.684
samples = 55
value = [14, 10, 6, 25, 0]
gini = 0.0
samples = 57
value = [0, 0, 0, 0, 57]
x[2] <= 3.5
gini = 0.312
samples = 31
value = [0, 0, 6, 25, 0]
x[0] <= 50.5
gini = 0.486
samples = 24
value = [14, 10, 0, 0, 0]
x[3] <= 4.5
gini = 0.48
samples = 15
value = [0, 0, 6, 9, 0]
gini = 0.0
samples = 16
value = [0, 0, 0, 16, 0]
gini = 0.0
samples = 14
value = [14, 0, 0, 0, 0]
gini = 0.0
samples = 10
value = [0, 10, 0, 0, 0]
gini = 0.0
samples = 9
value = [0, 0, 0, 9, 0]
gini = 0.0
samples = 6
value = [0, 0, 6, 0, 0]

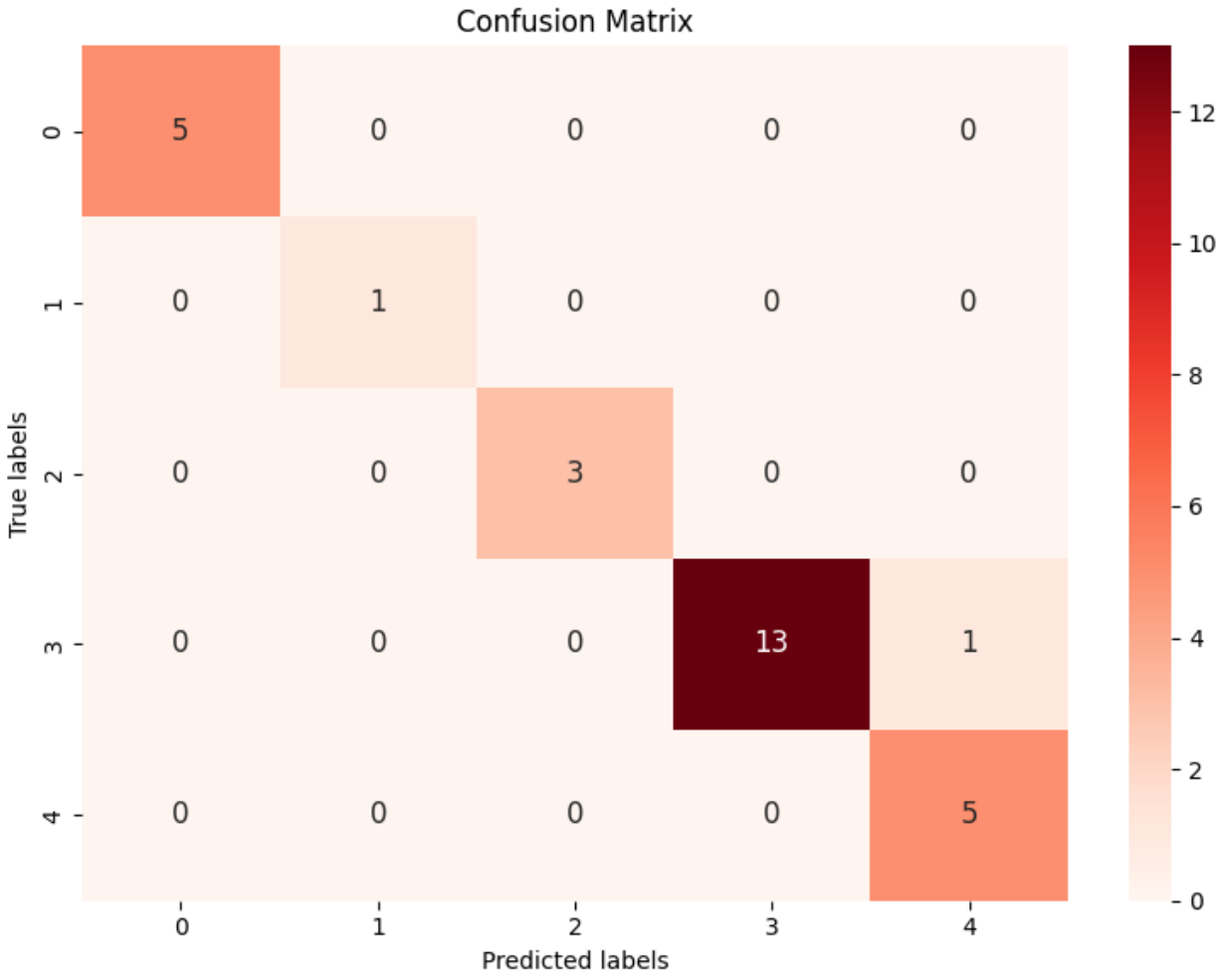


.٧

```
clf_dt8 = tree.DecisionTreeClassifier(max_depth=4, random_state=74,
ccp_alpha=0 , criterion = 'entropy')
```

```
0.9642857142857143
0.9642857142857143
0.9642857142857143
0.9642857142857143
0.9310344827586207
```


x[4] <= 14.638
entropy = 1.891
samples = 112
value = [14, 10, 6, 25, 57]
x[2] <= 7.5
entropy = 1.815
samples = 55
value = [14, 10, 6, 25, 0]
entropy = 0.0
samples = 57
value = [0, 0, 0, 0, 57]
x[2] <= 3.5
entropy = 0.709
samples = 31
value = [0, 0, 6, 25, 0]
x[0] <= 50.5
entropy = 0.98
samples = 24
value = [14, 10, 0, 0, 0]
x[3] <= 4.5
entropy = 0.971
samples = 15
value = [0, 0, 6, 9, 0]
entropy = 0.0
samples = 16
value = [0, 0, 0, 16, 0]
entropy = 0.0
samples = 14
value = [14, 0, 0, 0, 0]
entropy = 0.0
samples = 10
value = [0, 10, 0, 0, 0]
entropy = 0.0
samples = 9
value = [0, 0, 0, 9, 0]
entropy = 0.0
samples = 6
value = [0, 0, 6, 0, 0]



همان‌طور که مشاهده می‌شود تغییر `criterion` تاثیر زیادی در عملکرد درخت ندارد و آن چه که اثر زیادی در عملکرد دارد،تعداد لایه‌های پیشروی درخت می‌باشد.این تعداد بستگی به پارامتر `max_depth` دارد.همان‌طور که می‌بینیم با افزایش این پارامتر،دقت نیز بیشتر می‌شود.البته باید دقت شود که این تعداد خیلی زیاد نشود.چون در آن صورت ممکن است Overfit رخ دهد.همچنین پارامتر `ccp_alpha` که مقدار هرس کردن درخت را کنترل می‌کند بر این تعداد اثر می‌گذارد که در نتیجه بر نتیجه کلی اثر خواهد گذاشت.

K-Fold Cross-validation

همچنین عملکرد هر یک از درخت‌ها با این روش تست شده که کد به صورت زیر است:

```
om sklearn.model_selection import cross_val_score
# with 10 folds

#model_1
scores1 = crossfr_val_score(clf_dt1, X_pick, y_pick, cv=10)
mean_acc1 = 0
for i in range(len(scores1)):
  mean_acc1 += scores1[i]

print(f"scores for decision tree 1 is \n{scores1} \nand mean of accuracy
is \n{mean_acc1/len(scores1)}")
print(f"\n ----------------------------------")

#model_2
scores2 = cross_val_score(clf_dt2, X_pick, y_pick, cv=10)
mean_acc2 = 0
for i in range(len(scores2)):
  mean_acc2 += scores2[i]

print(f"scores for decision tree 2 is \n{scores2} \nand mean of accuracy
is \n{mean_acc2/len(scores2)}")
print(f"\n ----------------------------------")

#model_3
scores3 = cross_val_score(clf_dt3, X_pick, y_pick, cv=10)
mean_acc3 = 0
for i in range(len(scores3)):
  mean_acc3 += scores3[i]

print(f"scores for decision tree 3 is \n{scores3} \nand mean of accuracy
is \n{mean_acc3/len(scores3)}")
print(f"\n ----------------------------------")

#model_4
scores4 = cross_val_score(clf_dt4, X_pick, y_pick, cv=10)
mean_acc4 = 0
for i in range(len(scores4)):
  mean_acc4 += scores4[i]

print(f"scores for decision tree 4 is \n{scores4} \nand mean of accuracy
is \n{mean_acc4/len(scores4)}")
print(f"\n ----------------------------------")
```

```
#model_5
scores5 = cross_val_score(clf_dt5, X_pick, y_pick, cv=10)
mean_acc5 = 0
for i in range(len(scores5)):
  mean_acc5 += scores5[i]

print(f"scores for decision tree 5 is \n{scores5} \nand mean of accuracy
is \n{mean_acc5/len(scores5)}")
print(f"\n ----------------------------------")

#model_6
scores6 = cross_val_score(clf_dt6, X_pick, y_pick, cv=10)
mean_acc6 = 0
for i in range(len(scores6)):
  mean_acc6 += scores6[i]

print(f"scores for decision tree 6 is \n{scores6} \nand mean of accuracy
is \n{mean_acc6/len(scores6)}")
print(f"\n ----------------------------------")

#model_7
scores7 = cross_val_score(clf_dt7, X_pick, y_pick, cv=10)
mean_acc7 = 0
for i in range(len(scores7)):
  mean_acc7 += scores7[i]

print(f"scores for decision tree 7 is \n{scores7} \nand mean of accuracy
is \n{mean_acc7/len(scores7)}")
print(f"\n ----------------------------------")

#model_8
scores8 = cross_val_score(clf_dt8, X_pick, y_pick, cv=10)
mean_acc8 = 0
for i in range(len(scores8)):
  mean_acc8 += scores8[i]

print(f"scores for decision tree 8 is \n{scores8} \nand mean of accuracy
is \n{mean_acc8/len(scores8)}")
print(f"\n ----------------------------------")
```

در این کد به روشی که در قسمت آخر سوال۲ گفته شد،داده‌ها به ۱۰ فولد تقسیم شده و آموزش انجام می‌شود.سپس دقت به ازای هر یک از این ۱۰ حالت برای هر مدل چاپ شده و در نهایت میانگین آن‌ها نیز نمایش داده می‌شود:

```
scores for decision tree 1 is
[0.92857143 1.         0.92857143 0.92857143 0.85714286 0.92857143
 0.85714286 0.92857143 0.92857143 0.92857143]
and mean of accuracy is
0.9214285714285714

 ---------------------------------
scores for decision tree 2 is
[0.92857143 1.         0.92857143 0.92857143 0.85714286 0.92857143
 0.85714286 0.92857143 0.92857143 0.92857143]
and mean of accuracy is
0.9214285714285714

 ---------------------------------
scores for decision tree 3 is
[0.64285714 0.78571429 0.78571429 0.71428571 0.71428571 0.71428571
 0.64285714 0.71428571 0.71428571 0.71428571]
and mean of accuracy is
0.7142857142857144

 ---------------------------------

scores for decision tree 4 is
[0.78571429 0.92857143 0.85714286 0.85714286 0.85714286 0.85714286
 0.78571429 0.85714286 0.85714286 0.85714286]
and mean of accuracy is
0.8499999999999999

 ---------------------------------
scores for decision tree 5 is
[0.78571429 0.92857143 0.85714286 0.85714286 0.85714286 0.85714286
 0.78571429 0.85714286 0.85714286 0.85714286]
and mean of accuracy is
0.8499999999999999

 ---------------------------------
scores for decision tree 6 is
[0.78571429 0.92857143 0.85714286 0.85714286 0.85714286 0.85714286
 0.78571429 0.85714286 0.85714286 0.85714286]
and mean of accuracy is
0.8499999999999999

 ---------------------------------
scores for decision tree 7 is
[1.         1.         1.         1.         0.92857143 1.
 0.92857143 1.         1.         1.        ]
and mean of accuracy is
```

```
0.9857142857142858

 ---------------------------------
scores for decision tree 8 is
[1.         1.         1.         1.          0.92857143 1.
 0.92857143 1.         1.         1.        ]
and mean of accuracy is
0.9857142857142858
```

۳. توضیح دهید که روش‌هایی مانند جنگل تصادفی و AdaBoost چگونه می‌توانند به بهبود نتایج کمک کنند. سپس، با انتخاب یکی از این روش‌ها و استفاده از فراپارامترهای مناسب، سعی کنید نتایج پیاده‌سازی در مراحل قبلی را ارتقاء دهید.

روش AdaBoost

در این روش ابتدا یک مدل بر روی مجموعه دیتای آموزش،آموزش داده می‌شود.سپس یک مدل دوم برای اصلاح اشتباهات مدل اول ساخته می‌شود.این کار تا جایی ادامه می‌یابد تا آموزش دقیق‌تر و به دقت موردنظر برسیم.

روش جنگل تصادفی

در این روش خروجی چندین درخت تصمیم ترکیب می‌شود تا به یک نتیجه‌ی واحد برسیم. سهولت و انعطاف‌پذیری این روش و همچنین توانایی حل مشکلات طبقه‌بندی و رگرسیون از دلایلی است که باعث استفاده‌ی زیاد از آن شده است.

در این جا از روش جنگل تصادفی استفاده می‌شود.

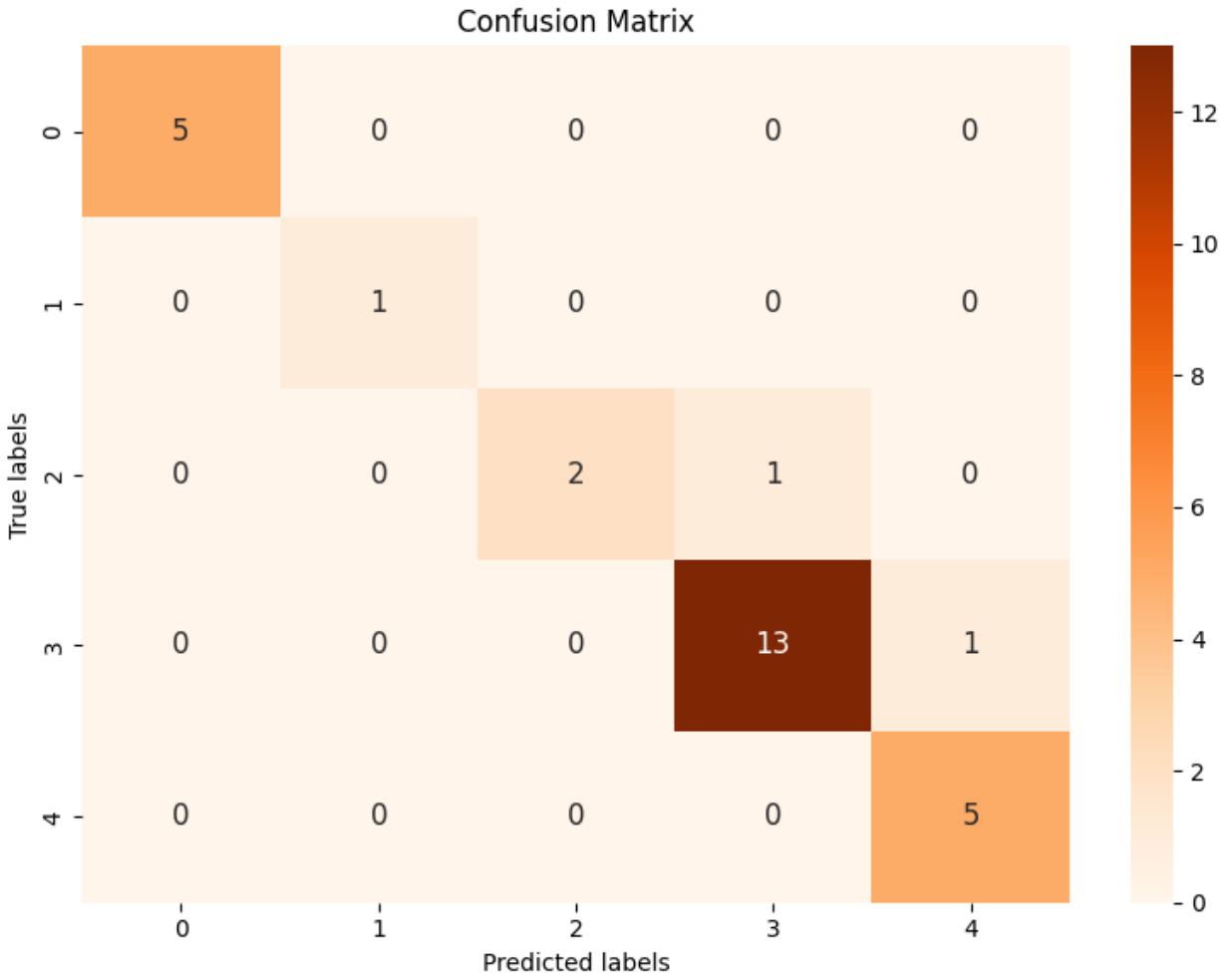
کد به صورت زیر است:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
clf_random_forest = RandomForestClassifier(max_depth=6, random_state=74)
clf_random_forest.fit(X_train_drug , y_train_drug)
clf_random_forest.score(X_test_drug , y_test_drug)
y_pred_drug_forest = clf_random_forest.predict(X_test_drug)

cf_matrix_forest = confusion_matrix(y_test_drug, y_pred_drug_forest)
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix_forest, annot=True, fmt='d', cmap='Oranges',
annot_kws={"size": 12})


plt.gca().set_ylim(len(np.unique(y_test_drug)), 0)
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.tight_layout()
plt.savefig('confusion_matrix_forest.png', dpi=300)
```

در این کد ابتدا طبقه‌بند جنگل تصادفی مورنظر با حداکثر عمق ۶ تعریف می‌شود.(حداکثر عمق همان تعداد پیش‌روی لایه‌های درخت موردنظر است)سپس این طبقه‌بند روی مجموعه داده‌ی آموزش،آموزش داده می‌شود و سپس دقت این طبقه‌بند و ماتریس درهم‌ریختگی مربوط به آن رسم می‌شود که به صورت زیر است:

```
0.9285714285714286
```



همان‌طور که مشاهده می‌شود به دقت تقریبا ۹۳ درصد رسیده‌ایم که دقت خوبی است.

سوال۴)

دیتاست بیماری قلبی را در نظر بگیرید. داده‌ها را به دو بخش آموزش و آزمون تقسیم کرده و ضمن انجام پیش‌پردازش‌هایی که روی آن لازم می‌دانید و با فرض گاوسی‌بودن داده‌ها، از الگوریتم طبقه‌بندی Bayes استفاده کنید و نتایج را در قالب ماتریس درهم‌ریختگی و classification_report تحلیل کنید. تفاوت میان دو حالت Macro و Micro را در کتابخانهٔ سایکیت‌لرن شرح دهید.

درنهایت، پنج داده را به‌صورت تصادفی از مجموعهٔ آزمون انتخاب کنید و خروجی واقعی را با خروجی پیش‌بینی‌شده مقایسه کنید.

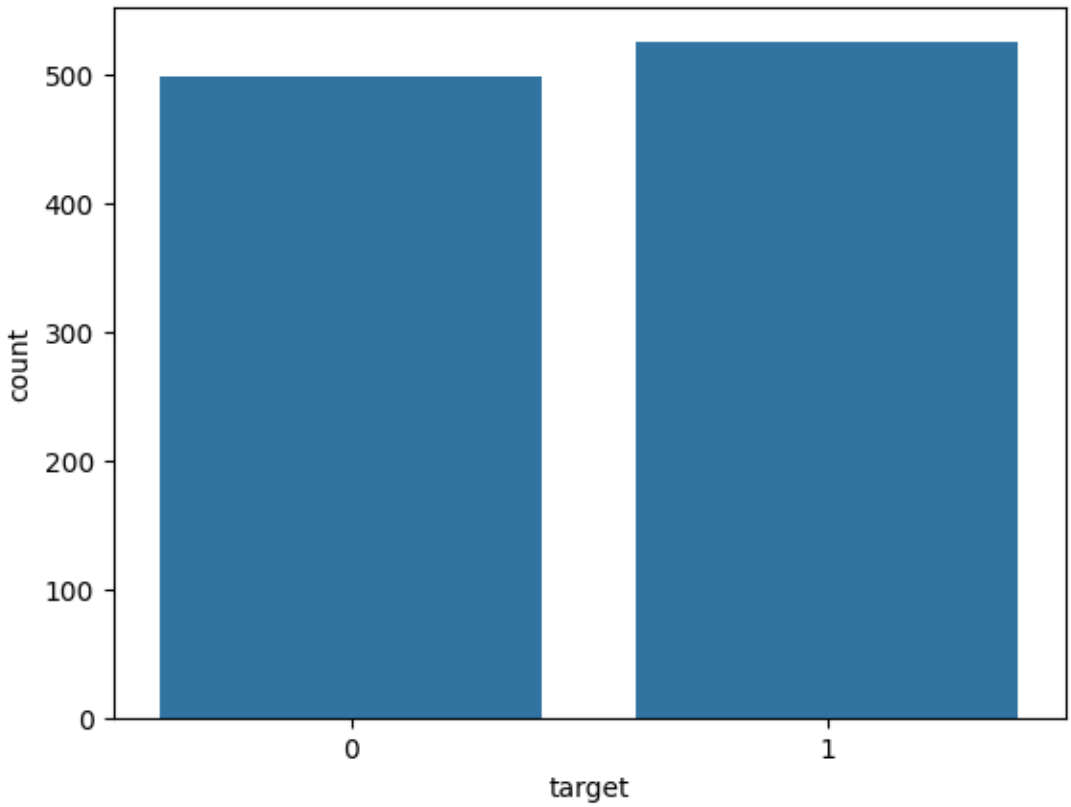
تفاوت میان Micro و Macro این است که در حالت میکرو با شمارش همه‌ی مقادیر صحیح مثبت،مقادیر مثبت کاذب و مقادیر منفی کاذب معیارهای موردنظر را در سطح جهانی و به صورت global محاسبه می‌کند اما در حالت ماکرو،معیارها را برای هر کلاس محاسبه می‌کند و میانگین غیر وزن‌دار آن‌ها را محاسبه می‌کند.این حالت عدم تعادل در کلاس‌ها را در نظر نمی‌گیرد.

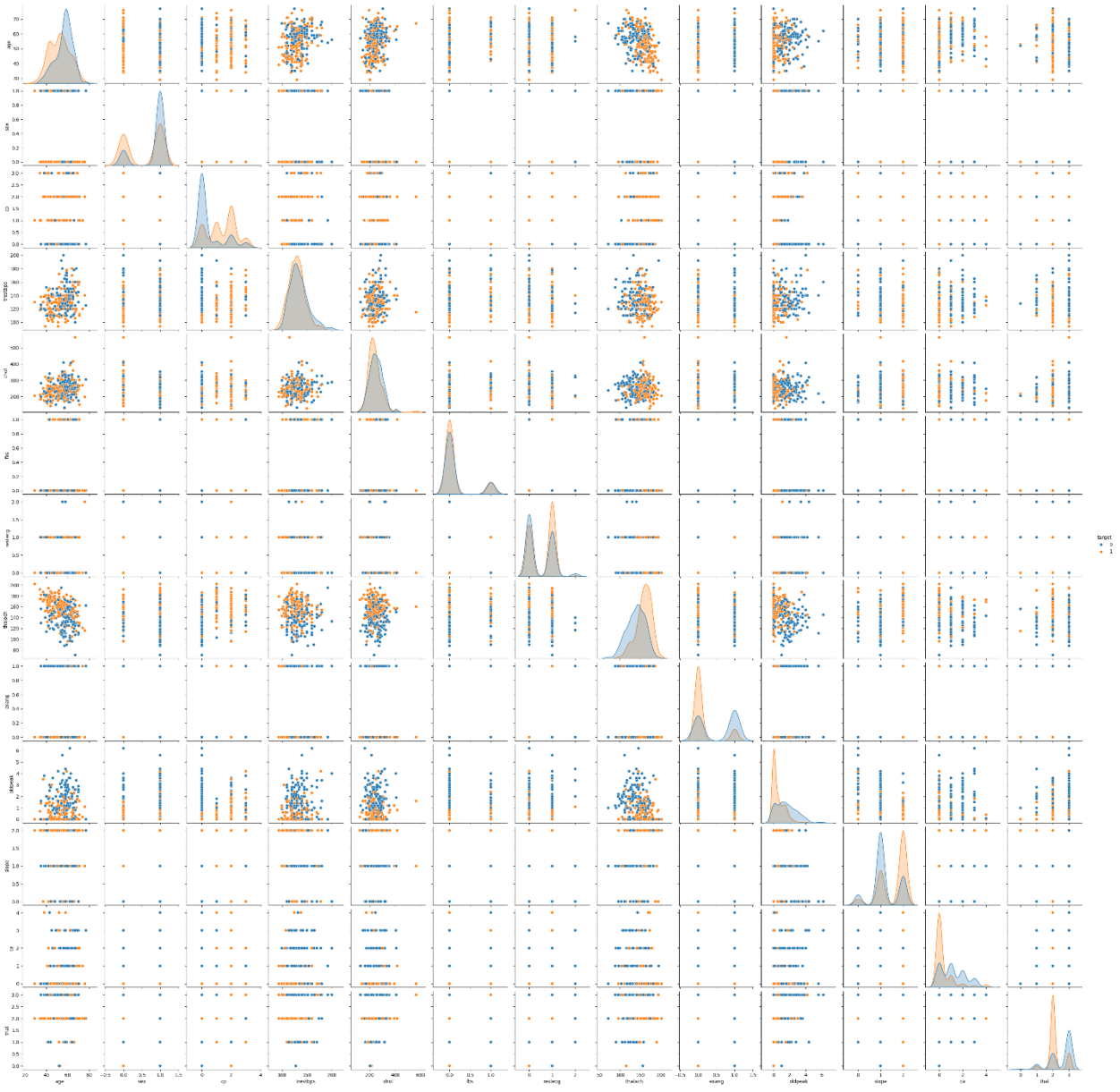
قسمت اول کد به صورت زیر است:

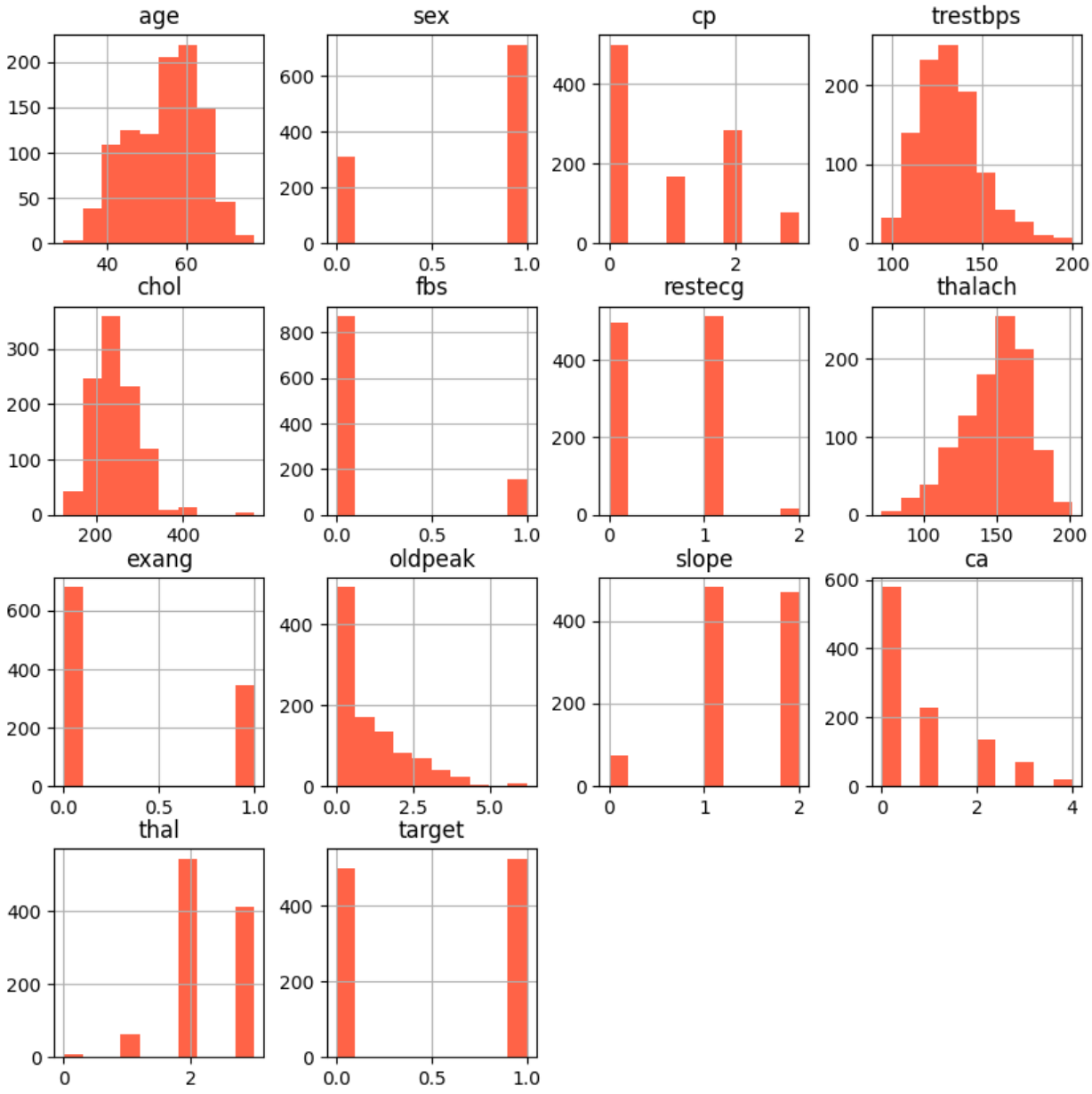
```
data_heart = pd.read_csv("/content/heart_Data.csv")
print(data_heart.groupby('target').size())
sns.countplot(x='target', data=data_heart)
sns.pairplot(data_heart,hue='target')
data_heart.hist(figsize=(10, 10),color='tomato')
plt.show()
```

در این کد دیتا از فایل csv که به این دفترچه از google drive آورده شده است،خوانده می‌شود و سپس فراوانی دیتا نمایش داده می‌شود.خروجی به صورت زیر است:

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


500
400
300
200
100
0
count
0
1
target


age
sex
cp
trestbps
chol
fbs
restecg
thalach
exang
oldpeak
slope
ca
thal
target
0
1



همان‌طور که مشاهده می‌شود،دیتاهای دو کلاس برابر نیستند و این مسئله می‌تواند باعث اخلال در آموزش شود.

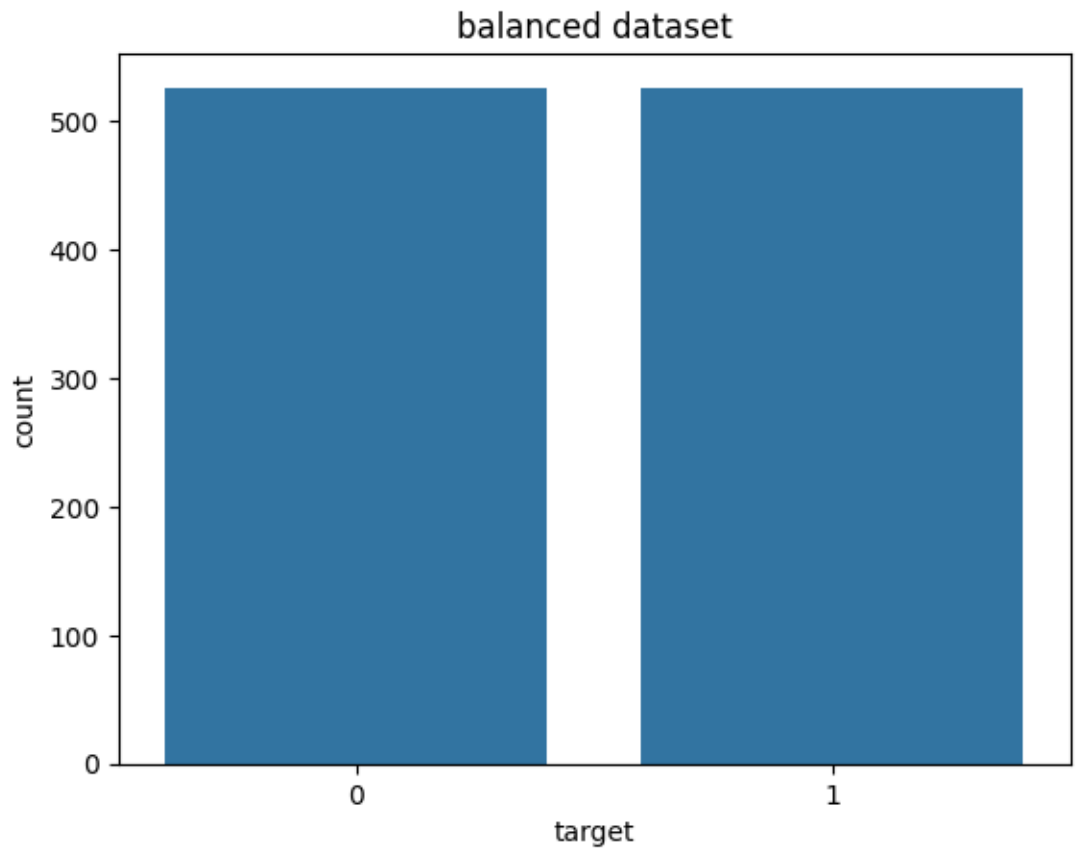
قسمت دوم کد:

```
X = data_heart.drop('target', axis=1)
```

```
y = data_heart['target']

smote = SMOTE(random_state=74)
X_resample, y_resample = smote.fit_resample(X, y)


X_resample_df = pd.DataFrame(X_resample, columns=X.columns)
y_resample_df = pd.DataFrame(y_resample, columns=['target'])

balanced_data = pd.concat([X_resample_df, y_resample_df], axis=1)

sns.countplot(data=balanced_data, x='target')
plt.title('balanced dataset')
plt.xlabel('target')
plt.ylabel('count')
plt.show()

print(balanced_data.groupby('target').size())
```

این قسمت از کد در واقع دیتاهایی نزدیک به دیتاهای موجود در کلاسی که اقلیت دارد ایجاد می‌کند تا توزیع داده‌ها برای کلاس‌های مختلف یکسان شود.

خروجی به صورت زیر است:



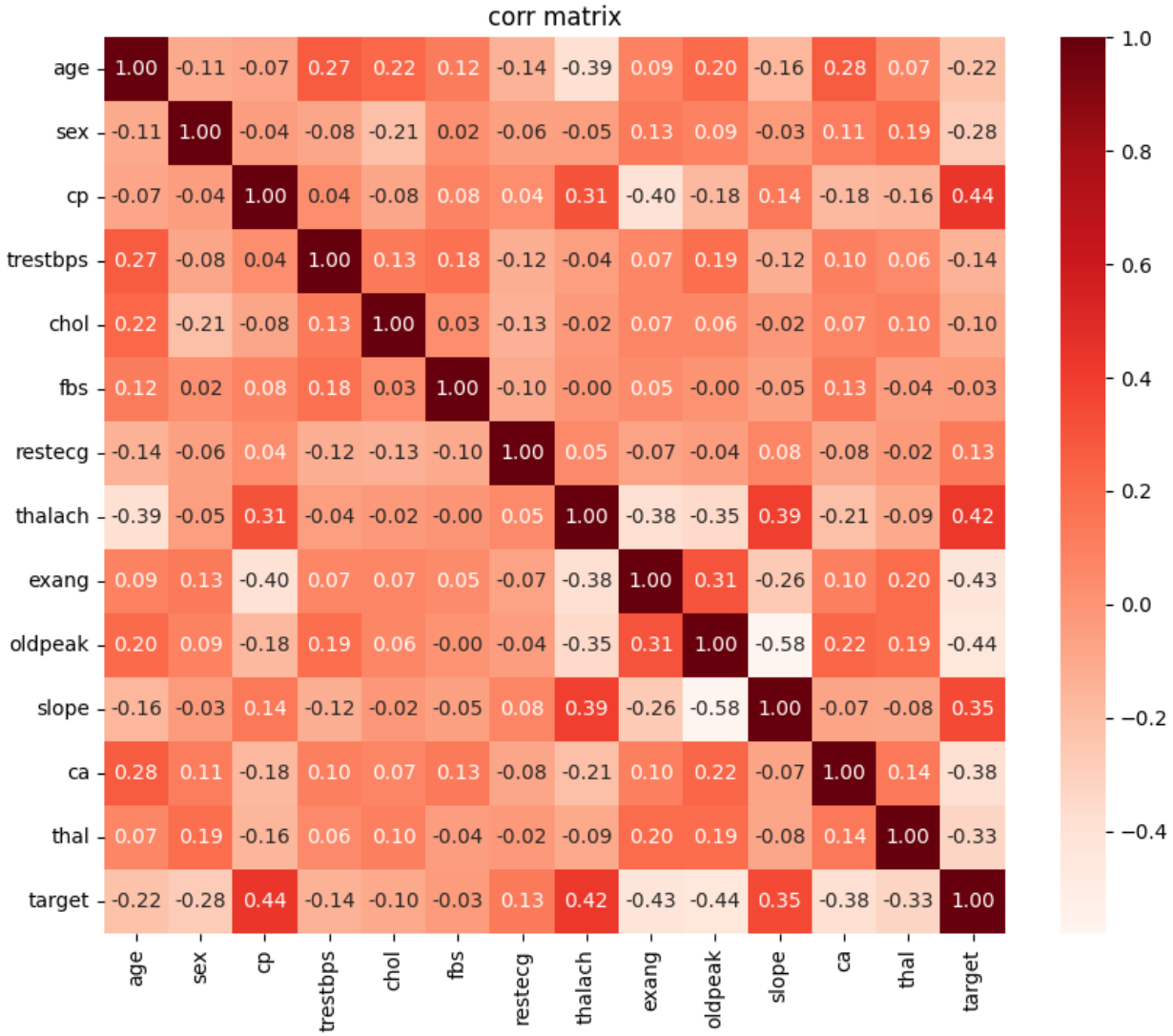

```
target
0    526
1    526
dtype: int64
```

همان‌طور که مشاهده می‌شود تعداد نمونه‌های هر کلاس برابر شده است و به نوعی داده‌ها متعادل شده‌اند.

قسمت سوم کد:

```
correlation_matrix = balanced_data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='Reds', fmt=".2f")
plt.title('corr matrix')
plt.show()
```

در این قسمت ماتریس پراکندگی داده‌های متعادل شده رسم می‌شود که به صورت زیر است:



همان‌طور که مشاهده می‌شود تقریبا وابستگی ویژگی‌های مختلف خیلی زیاد نیست و تقریبا همه زیر ۵۰ درصد است.

قسمت چهارم کد:

```
shuffled_data_heart = shuffle(balanced_data, random_state=74)
shuffled_data_heart_arr = np.array(shuffled_data_heart)
X_heart = np.array(shuffled_data_heart_arr[: , :13])
y_heart = np.array(shuffled_data_heart_arr[: , 13])
print(X_heart.shape)
X_train_heart, X_test_heart, y_train_heart, y_test_heart =
train_test_split(X_heart, y_heart, test_size=0.2, random_state=74)
scaler = StandardScaler()
scaler.fit(X_train_heart)
X_train_norm = scaler.transform(X_train_heart)
X_test_norm = scaler.transform(X_test_heart)
```

در این کد ابتدا دیتا مخلوط می‌شود.سپس داده‌ها با نسبت ۸ به ۲ به دو دسته‌ی آموزش و ارزیابی تقسیم می‌شوند و در نهایت با نرمال‌ساز StandardScaler دیتا نرمال‌سازی می‌شود.

قسمت پنجم کد:

```
class NaiveBayes:
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self._classes = np.unique(y)
        n_classes = len(self._classes)

        # calculate mean, var, and prior for each class
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors = np.zeros(n_classes, dtype=np.float64)

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


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

    def _predict(self, x):
```

```
        posteriors = []

        # calculate posterior probability for each class
        for idx, c in enumerate(self._classes):
            prior = np.log(self._priors[idx])
            posterior = np.sum(np.log(self._pdf(idx, x)))
            posterior = posterior + prior
            posteriors.append(posterior)

        # return class with the highest posterior
        return self._classes[np.argmax(posteriors)]

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

این قسمت از کد در واقع یک کلاس است که مدل بیز ما را تعریف می‌کند.این کلاس دارای توابعی مانند fit و predict مانند توابع آماده sklearn است که مثلا تابع fit آموزش مدل بر اساس داده‌های آموزش را انجام می‌دهد و تابع predict برای پیش‌بینی داده‌های تست استفاده می‌شود.

قسمت ششم کد:

```
model_bayes = NaiveBayes()
model_bayes.fit(X_train_norm, y_train_heart)
y_pred_bayes = model_bayes.predict(X_test_norm)
model_gauss = GaussianNB()
model_gauss.fit(X_train_norm, y_train_heart)
y_pred_gauss = model_gauss.predict(X_test_norm)
```

در این قسمت از کد با استفاده از مدل بیز آموزش انجام می‌شود.همچنین برای آموزش با استفاده از مدل گاوسی،از کتابخانه آماده استفاده شده است.(بر خلاف مدل بیز که نوشته شد).

قسمت هفتم کد:

```
print(classification_report(y_test_heart,y_pred_bayes))
print(classification_report(y_test_heart,y_pred_gauss))
```

این قسمت از کد اطلاعاتی راجع به دقت مدل و برخی پارامترهای دیگر می‌دهد.همچنین برخی از این معیارها برای هر کلاس به صورت مجزا نیز بیان شده است.خروجی به ترتیب به صورت زیر است:

بیز:

```
precision    recall  f1-score   support

         0.0       0.83      0.83      0.83        92
         1.0       0.87      0.87      0.87       119

    accuracy                           0.85       211
   macro avg       0.85      0.85      0.85       211
weighted avg       0.85      0.85      0.85       211
```

گاوسی:

```
precision    recall  f1-score   support

         0.0       0.83      0.83      0.83        92
         1.0       0.87      0.87      0.87       119

    accuracy                           0.85       211
   macro avg       0.85      0.85      0.85       211
weighted avg       0.85      0.85      0.85       211
```

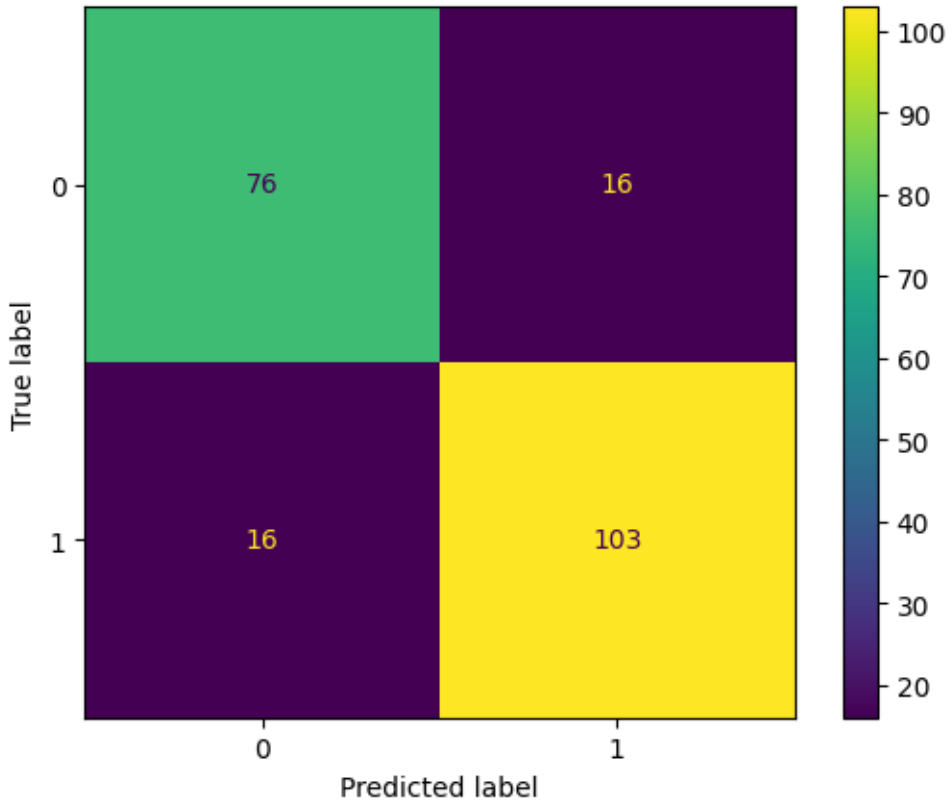
برای مثال دقت برای هر دو مدل برابر ۸۵ درصد است.همچنین F1_score برای هر دو مدل برای داده‌های کلاس اول ۸۳ درصد و برای کلاس دوم ۸۷ درصد است.

قسمت هشتم کد:

```
conf_matrix_bayes = confusion_matrix(y_test_heart,y_pred_bayes)
names = list(shuffled_data_heart.groupby('target').groups.keys())
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_bayes,
display_labels=names)
disp.plot()
plt.show()
print('Accuracy_bayes_model :',accuracy_score(y_test_heart,y_pred_bayes))
print('Precision_bayes_model
:',precision_score(y_test_heart,y_pred_bayes,average='micro'))
print('Recall_bayes_model
:',recall_score(y_test_heart,y_pred_bayes,average='micro'))
print('F1 score_bayes_model
:',f1_score(y_test_heart,y_pred_bayes,average='micro'))
print('Jaccard score_bayes_model
:',jaccard_score(y_test_heart,y_pred_bayes,average='micro'))
```

این قسمت از کد ماتریس درهم‌ریختگی و برخی معیارها را برای مدل بیز نمایش می‌دهد که نتیجه به صورت زیر است:




```
Accuracy_bayes_model : 0.8483412322274881
Precision_bayes_model : 0.8483412322274881
Recall_bayes_model : 0.8483412322274881
F1 score_bayes_model : 0.8483412322274881
```

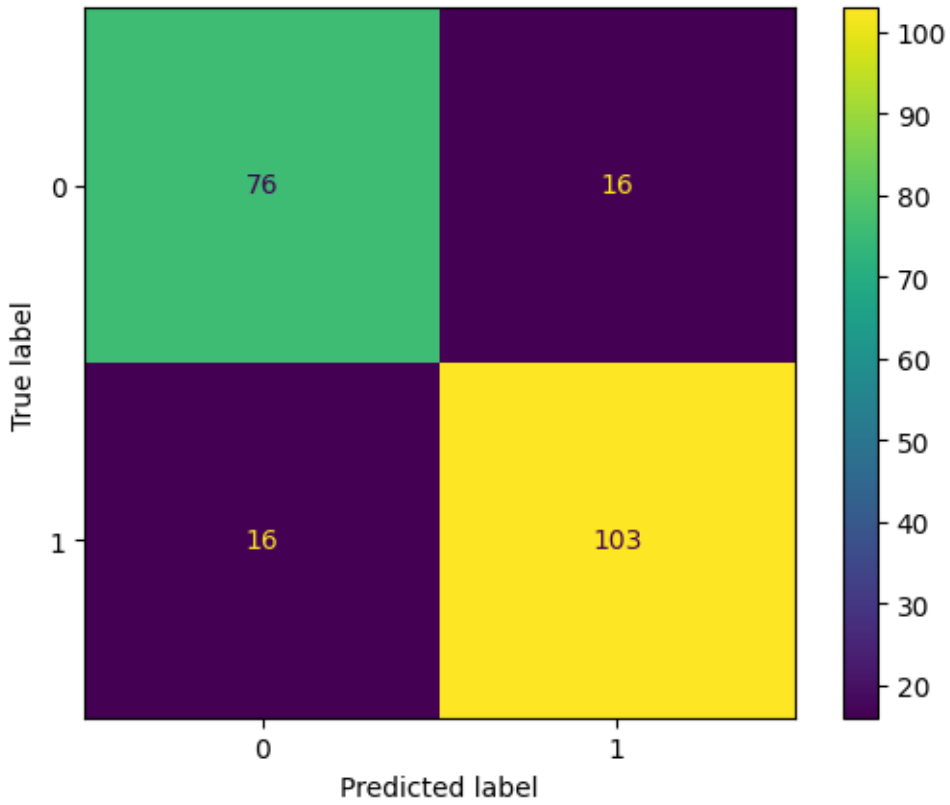
```
Jaccard score_bayes_model : 0.7366255144032922
```

همان‌طور که می‌بینیم عملکرد این مدل عالی نبوده است اما عملکرد نسبتا خوبی داشته است.

این روند برای مدل گاوسی هم عینا تکرار شده که کد و نتایج به صورت زیر است:

```
conf_matrix_gauss = confusion_matrix(y_test_heart,y_pred_gauss)
names = list(shuffled_data_heart.groupby('target').groups.keys())
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_gauss,
display_labels=names)
disp.plot()
plt.show()
print('Accuracy_gauss_model :',accuracy_score(y_test_heart,y_pred_gauss))
print('Precision_gauss_model
:',precision_score(y_test_heart,y_pred_gauss,average='micro'))
print('Recall_gauss_model
:',recall_score(y_test_heart,y_pred_gauss,average='micro'))
print('F1 score_gauss_model
:',f1_score(y_test_heart,y_pred_gauss,average='micro'))
print('Jaccard score_gauss_model
:',jaccard_score(y_test_heart,y_pred_gauss,average='micro'))
```

نتایج:




```
Accuracy_gauss_model : 0.8483412322274881
Precision_gauss_model : 0.8483412322274881
Recall_gauss_model : 0.8483412322274881
F1 score_gauss_model : 0.8483412322274881
Jaccard score_gauss_model : 0.7366255144032922
```

نتایج عینا شبیه به مدل بیز است و این دو مدل عملکرد یکسانی داشته اند.

قسمت نهم کد:

```
import random
data_for_s = np.append(X_test_norm , y_test_heart.reshape(211 , 1) , axis
= 1)
samples = random.sample(list(data_for_s) , 5)
X_sample = (np.array(samples))[: , :13]
y_sample = (np.array(samples))[: , 13]
```

در این قسمت ۵ داده به طور تصادفی از میان داده‌ها انتخاب شده تا خروجی واقعی و خروجی پیش‌بینی شده توسط مدل برای این ۵ داده مقایسه شوند.

برای مقایسه‌ی این خروجی‌ها از کد زیر استفاده شده است:

```
y_pred_sample = model_bayes.predict(X_sample)
print(f"our model predict is {y_pred_sample} and true label is
{y_sample}")
acc = 0
for i in range(len(y_pred_sample)):
  if (y_pred_sample[i] == y_sample[i]):
    acc += 1

print(f"{(100*acc)/len(y_pred_sample)}% of predicts are true")
```

در این کد ابتدا پیش‌بینی برای دیتاهای موجود در نمونه انجام می‌شود.سپس خروجی واقعی و خروجی پیش‌بینی شده مقایسه می‌شود.همچنین به ازای هر دیتا از نمونه که خروجی پیش‌بینی شده و خروجی واقعی برابر هستند،متغیر acc یک واحد اضافه می‌شود و در نهایت این متغیر تقسیم بر ۵ می‌شود تا دقت پیش‌بینی این ۵داده به دست آید.

نتیجه به صورت زیر است:

```
our model predict is [1. 1. 1. 0. 1.] and true label is [1. 1. 1. 0. 1.]
100.0% of predicts are true
```

همان‌طور که مشاهده می‌شود تمام پیش‌بینی‌ها برای این ۵داده‌ی تصادفی درست بوده و دقت ۱۰۰درصد داشته است.

پایان