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

درس یادگیری ماشین

گزارش مینی پروژه شماره دو

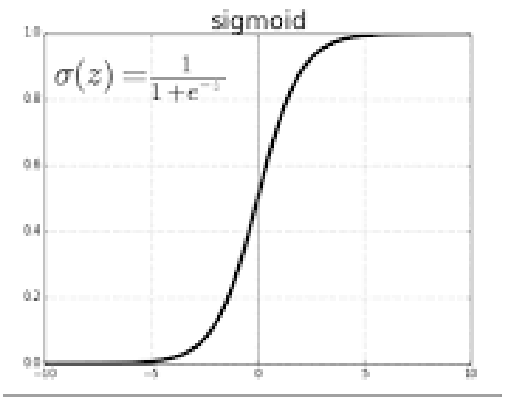
نام و نام خانوادگی	علیرضا جهانی
شمارهٔ دانشجویی	۴۰۲۲۳۰۴۴
استاد درس	دکتر علیاری
تاریخ	بهمن‌ماه ۱۴۰۲

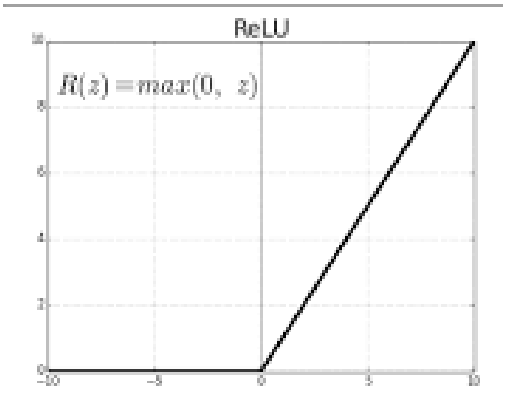
Colab github

فهرست مطالب

۱ سوال اول

۱.۱ بخش اول سوال اول





در یک مسئله طبقه‌بندی دو کلاسه، فرض کنید که شبکه عصبی ما دارای دو لایه‌ی انتهایی است که یکی از فعال‌ساز ReLU و دیگری از فعال‌ساز Sigmoid استفاده می‌کند. در ادامه به بررسی عملکرد و خروجی‌های این شبکه می‌پردازیم.

فعال‌ساز ReLU

فعال‌ساز ReLU (Rectified Linear Unit) یک تابع غیرخطی است که به صورت زیر تعریف می‌شود:

$$ReLU(x) = \max(0, x)$$

ویژگی‌های فعال‌ساز ReLU:

- مقادیر منفی ورودی را صفر می‌کند.
- مقادیر مثبت ورودی را بدون تغییر می‌گذارد.
- به طور کلی در لایه‌های مخفی شبکه‌های عصبی استفاده می‌شود.
- بهبود همگرایی در شبکه‌های عصبی عمیق به دلیل شیب غیرمنفی آن.

فعال‌ساز Sigmoid

فعال‌ساز Sigmoid یک تابع غیرخطی است که به صورت زیر تعریف می‌شود:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

ویژگی‌های فعال‌ساز :Sigmoid

- خروجی را به بازه‌ی (۰، ۱) محدود می‌کند.
- به طور معمول در لایه‌های خروجی مسائل طبقه‌بندی دو کلاسه استفاده می‌شود.
- می‌تواند به عنوان احتمال دسته‌بندی مثبت تفسیر شود.

عملکرد شبکه با دو لایه‌ی انتهایی

در این مسئله، فرض کنید که لایه‌ی ماقبل آخر (لایه‌ی انتهایی اول) از فعال‌ساز ReLU استفاده می‌کند و لایه‌ی آخر (لایه‌ی انتهایی دوم) از فعال‌ساز Sigmoid استفاده می‌کند.

لایه‌ی انتهایی اول (ReLU)

فرض کنید ورودی به این لایه $\mathbf{z}$ باشد. خروجی این لایه به صورت زیر خواهد بود:

$$\mathbf{a} = ReLU(\mathbf{z}) = \max(0, \mathbf{z})$$

لایه‌ی انتهایی دوم (Sigmoid)

خروجی لایه‌ی ReLU به عنوان ورودی به لایه‌ی Sigmoid می‌رود. اگر $\mathbf{a}$ ورودی به لایه‌ی Sigmoid باشد، خروجی نهایی $\hat{y}$ به صورت زیر خواهد بود:

$$\hat{y} = Sigmoid(\mathbf{a}) = \frac{1}{1 + e^{-\mathbf{a}}}$$

تفسیر خروجی نهایی

از آنجایی که لایه‌ی آخر از فعال‌ساز Sigmoid استفاده می‌کند، خروجی نهایی $\hat{y}$ در بازه‌ی (۰، ۱) قرار خواهد داشت و می‌توان آن را به عنوان احتمال تعلق ورودی به کلاس مثبت (کلاس ۱) تفسیر کرد. در یک مسئله طبقه‌بندی دو کلاسه، تصمیم نهایی براساس مقدار $\hat{y}$ گرفته می‌شود:

- اگر $\hat{y} \geq 0.5$ باشد، ورودی به کلاس ۱ (مثبت) تعلق دارد.
- اگر $\hat{y} < 0.5$ باشد، ورودی به کلاس ۰ (منفی) تعلق دارد.

استفاده از فعال‌سازهای ReLU و Sigmoid در لایه‌های انتهایی یک شبکه عصبی برای مسئله طبقه‌بندی دو کلاسه باعث می‌شود که خروجی نهایی به صورت احتمال تعلق ورودی به یکی از دو کلاس تفسیر شود. لایه‌ی ReLU کمک می‌کند تا مقادیر منفی حذف شوند و لایه‌ی Sigmoid خروجی را به بازه‌ی (۰، ۱) محدود می‌کند، که تفسیر آن به عنوان احتمال کلاس بسیار مفید است.

۲.۱ بخش دوم سوال اول

تابع ELU یا Exponential Linear Unit به عنوان جایگزینی برای ReLU معرفی شده است. معادله‌ی ELU به صورت زیر تعریف می‌شود:

$$ELU(x) = \{ x \; if x \geq 0 \alpha(e^x - 1) if x < 0$$

در اینجا α یک ثابت مثبت است که معمولاً مقدار آن به صورت تجربی تنظیم می‌شود. برای مثال، α ممکن است مقداری مانند ۱ یا . داشته باشد.

برای محاسبه‌ی گرادیان تابع ELU، ابتدا مشتق این تابع را محاسبه می‌کنیم:

$$ELU'(x) = \{ 1 \; if x \geq 0 \alpha e^x if x < 0$$

یکی از مزایای ELU نسبت به ReLU این است که خروجی‌های منفی را به صورت نرم‌تر مدیریت می‌کند و باعث می‌شود که میانگین فعال‌سازی‌های نزدیک به صفر شود. این ویژگی می‌تواند در آموزش شبکه‌های عصبی عمیق کمک کند زیرا:

۱. کاهش **Bias Shift**: میانگین فعال‌سازی‌ها نزدیک به صفر است، که باعث می‌شود که شبکه نیازی به جبران bias shift نداشته باشد و فرآیند آموزش را بهبود بخشد.

۲. مدیریت خروجی‌های منفی: برخلاف ReLU که خروجی‌های منفی را به صفر می‌رساند، ELU مقادیر منفی را به صورت نمایی تبدیل می‌کند. این ویژگی کمک می‌کند تا اطلاعات از ورودی‌های منفی حفظ شده و به بهبود یادگیری کمک کند.

۳. کاهش مشکل **Vanishing Gradient**: در حالی که ReLU ممکن است برای ورودی‌های منفی گرادیان را به صفر برساند، ELU با استفاده از تابع نمایی برای ورودی‌های منفی، گرادیان‌های کوچک ولی غیرصفر تولید می‌کند. این امر به کاهش مشکل vanishing gradient کمک می‌کند.

۴. پیوستگی و نرمی: تابع ELU در نقطه $x = 0$ پیوسته و نرم است، به این معنی که هم تابع و هم مشتق آن در این نقطه پیوسته هستند. این ویژگی به بهبود پایداری و همگرایی شبکه عصبی کمک می‌کند.

به طور کلی، ELU می‌تواند بهبودهایی در همگرایی و دقت شبکه‌های عصبی نسبت به ReLU فراهم کند، به خصوص در شبکه‌های عمیق که مشکلاتی مانند vanishing gradient و bias shift ممکن است بیشتر باشند.

۳.۱ بخش سوم سوال اول

در این بخش، هدف طراحی یک شبکه عصبی ساده به کمک نورون‌های McCulloch-Pitts است که بتواند ناحیه‌های مختلف را از هم تفکیک کند. این شبکه باید قادر باشد نقاط داخلی مثلث نشان داده شده در شکل را از سایر نقاط متمایز سازد.

مرحله اول: طراحی شبکه با استفاده از نورون‌های McCulloch-Pitts

ابتدا، باید شبکه‌ای را طراحی کنیم که بتواند ناحیه‌ای را که در شکل نشان داده شده است، از سایر نواحی تفکیک کند. برای انجام این کار، نورون‌های McCulloch-Pitts را در لایه اول استفاده می‌کنیم.

تعریف نورون‌ها و توابع انتقال

برای تفکیک نواحی در داخل مثلث، سه معادله خطی که اضلاع مثلث را تشکیل می‌دهند به شکل زیر تعریف می‌شوند:

$$2x + y = 6 \tag{۱}$$

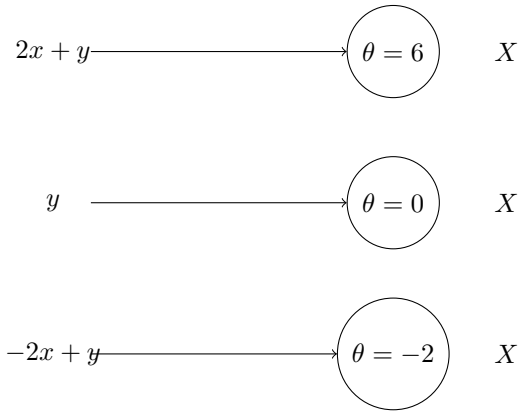
$$y = 0 \tag{۲}$$

$$-2x + y = -2 \tag{۳}$$

این معادلات به عنوان ورودی‌های نورون‌های لایه اول عمل می‌کنند. هر نورون به یک معادله خطی خاص تخصیص داده می‌شود و خروجی آنها به عنوان ورودی برای لایه بعدی استفاده می‌شود.

نمایش لایه اول شبکه عصبی

در این مرحله، نورون‌های لایه اول به صورت زیر تنظیم می‌شوند:

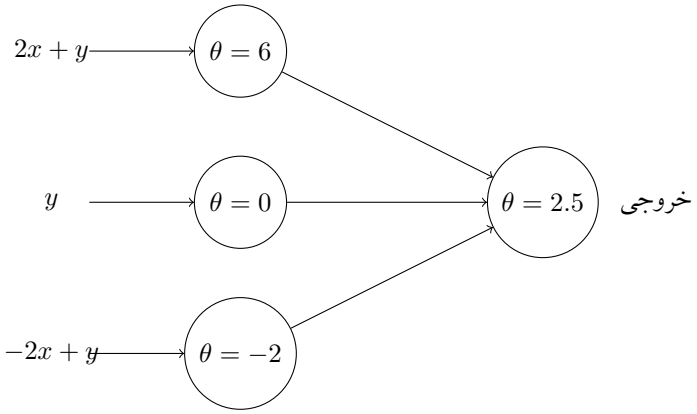


تنظیم وزن‌های نورون لایه دوم

برای اطمینان از اینکه شبکه به درستی عمل می‌کند، باید وزن‌های نورون لایه دوم را تنظیم کنیم. خروجی نورون‌های لایه اول به نورون‌های لایه دوم متصل می‌شوند. نورون‌های لایه دوم زمانی که تمامی نورون‌های لایه اول خروجی صفر داشته باشند (به معنای اینکه نقطه در داخل مثلث است)، خروجی فعال خواهند داشت. برای این منظور، وزن‌های نورون‌های لایه دوم به گونه‌ای تنظیم می‌شوند که مجموع ورودی‌های آنها برابر با یک مقدار آستانه‌ای معین باشد.

نمایش نهایی شبکه عصبی

نهایتاً، شبکه عصبی به همراه Threshold به صورت زیر خواهد بود:



این ساختار، شبکه‌ای را ایجاد می‌کند که می‌تواند نقاط داخلی مثلث را از سایر نقاط تفکیک کند.

```python
import matplotlib.pyplot as plt
import numpy as np
vertices = np.array([[1, 0], [3, 0], [2, 2]])
triangle = plt.Polygon(vertices, closed=True, fill=True, edgecolor='blue', facecolor='#FFB6C1', hatch='/', label='Area Inside Triangle')
fig, ax = plt.subplots()
ax.add_patch(triangle)
for (x, y), label in zip(vertices, ['C (1,0)', 'B (3,0)', 'A (2,2)']):
    ax.text(x, y, label, fontsize=12, ha='right' if x>2 else 'left')
ax.set_xlim(0, 4)
ax.set_ylim(-1, 3)
ax.set_aspect('equal', adjustable='datalim')
plt.grid(True)
ax.set_title('Graph with Lines and Hatched Area (Triangle)')
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
plt.show()
```

۱. وارد کردن کتابخانه‌های مورد نیاز

ابتدا، کتابخانه‌های matplotlib و numpy که برای رسم نمودار و انجام محاسبات عددی استفاده می‌شوند، وارد می‌شوند. این کتابخانه‌ها ابزارهای قدرتمندی برای ایجاد نمودارها و مدیریت داده‌ها فراهم می‌کنند.

۲. تعریف رئوس مثلث

با استفاده از یک آرایه‌ی numpy، مختصات رئوس مثلث تعریف می‌شوند. در این مثال، مثلث با رئوس در نقاط (۱،۰)، (۳،۰) و (۲،۲) تعریف شده است. این نقاط به صورت یک آرایه دو بعدی تعریف می‌شوند.

۳. ایجاد چندضلعی

یک چندضلعی با استفاده از رئوس تعریف شده ایجاد می‌شود. ویژگی‌های این چندضلعی شامل رنگ لبه‌ها (آبی)، رنگ داخل (صورتی کم‌رنگ) و هاشور (به صورت خطوط مورب) تنظیم می‌شوند. این چندضلعی نشان‌دهنده‌ی ناحیه‌ی داخلی مثلث است.

۴. ایجاد شکل و محورها

یک شکل و محورهای مربوط به آن با استفاده از تابع subplots از کتابخانه matplotlib ایجاد می‌شود. این تابع یک شکل و مجموعه‌ای از محورها را برای رسم نمودار فراهم می‌کند.

۵. اضافه کردن چندضلعی به محور

چندضلعی ایجاد شده به محور اضافه می‌شود تا در نمودار نمایش داده شود.

۶. افزودن متن به نقاط رئوس

برای هر یک از رئوس مثلث، یک متن شامل مختصات آن به نمودار اضافه می‌شود. این کار با استفاده از یک حلقه و تابع text از کتابخانه matplotlib انجام می‌شود. مختصات و متن مربوط به هر رأس به این تابع داده می‌شود و متن در مکان مربوطه نمایش داده می‌شود.

۷. تنظیم محدوده محورها

محدوده محورها برای نمایش بهینه‌ی مثلث تنظیم می‌شود. محور X از ۰ تا ۴ و محور Y از -۱ تا ۳ تنظیم می‌شوند تا تمام نقاط مثلث به خوبی در نمودار نمایش داده شوند.

۸. تنظیم نسبت محورها

نسبت ابعاد محورها برابر قرار داده می‌شود تا مثلث به درستی و بدون اعوجاج نمایش داده شود. این کار با استفاده از تابع set_aspect انجام می‌شود.

۹. نمایش شبکه‌بندی

شبکه‌بندی محورها فعال می‌شود تا خوانایی نمودار افزایش یابد. این کار با استفاده از تابع grid انجام می‌شود.

۱۰. افزودن عنوان و برچسب‌های محورها

عنوان نمودار و برچسب‌های محورها اضافه می‌شوند تا نمودار معنا پیدا کند. این کار با استفاده از توابع set_title، set_xlabel و set_ylabel انجام می‌شود.

۱۱. نمایش نمودار

در نهایت، نمودار با استفاده از تابع show نمایش داده می‌شود. این تابع نمودار را رندر کرده و در پنجره‌ای جداگانه نمایش می‌دهد.

```python
import numpy as np
#define muculloch pitts
class McCulloch_Pitts_neuron():

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

  def model(self , x):
    #define model with threshold
    if self.weights @ x >= self.threshold:
        return 1
    else:
        return 0
```

۱. وارد کردن کتابخانه‌های مورد نیاز

ابتدا، کتابخانه‌های numpy که برای انجام محاسبات عددی استفاده می‌شود و کتابخانه‌ی matplotlib.pyplot برای رسم نمودار وارد می‌شوند. این کتابخانه‌ها ابزارهای قدرتمندی برای ایجاد نمودارها و مدیریت داده‌ها فراهم می‌کنند.

۲. تعریف کلاس نورون McCulloch-Pitts

یک کلاس به نام McCulloch_Pitts_neuron تعریف می‌شود که ویژگی‌ها و رفتار یک نورون McCulloch-Pitts را شبیه‌سازی می‌کند. این کلاس شامل تابعهای زیر است:

```python
def Area(x, y):
  neur1 = McCulloch_Pitts_neuron([2, 1], 6)
  neur2 = McCulloch_Pitts_neuron([0, 1], 0)
  neur3 = McCulloch_Pitts_neuron([-2, 1], -2)
  neur5 = McCulloch_Pitts_neuron([-1, 3, -1], 2.5)
  z1 = neur1.model(np.array([x, y]))
  z2 = neur2.model(np.array([x, y]))
  z3 = neur3.model(np.array([x, y]))
  z4 = neur5.model(np.array([z1, z2, z3]))
  return list([z4])

num_points = 2000
x_values = np.random.uniform(0, 4, num_points)  # Updated x-axis limits
y_values = np.random.uniform(-1, 3, num_points)  # Updated y-axis limits
red_points = []
green_points = []
for i in range(num_points):
    z4_value = Area(x_values[i], y_values[i])
    if z4_value == [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)
plt.figure(figsize=(8, 6))
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.legend(loc = 'upper right', bbox_to_anchor=(1.2, 1.0))
plt.show()
```

– سازنده کلاس: این تابع مقداردهی اولیه وزن‌ها و آستانه نورون را انجام می‌دهد. – مدل نورون: این تابع ورودی را می‌گیرد و با استفاده از وزن‌ها و آستانه، خروجی نورون را محاسبه می‌کند. اگر مجموع وزن‌دار ورودی‌ها بیشتر یا مساوی آستانه باشد، خروجی ۱ وگرنه ۰ خواهد بود.

۳. تعریف تابع برای تعیین ناحیه

یک تابع به نام Area تعریف می‌شود که سه نورون McCulloch-Pitts برای اضلاع مثلث و یک نورون دیگر برای ترکیب خروجی‌های آن‌ها را ایجاد می‌کند. این تابع ورودی‌های x و y را دریافت می‌کند و خروجی نهایی نورون‌ها را برمی‌گرداند.

۴. تولید داده‌های تصادفی

۲۰۰۰ نقطه تصادفی برای x و y در محدوده مشخص تولید می‌شود. این مقادیر محدوده‌های محورهای x و y را پوشش می‌دهند.

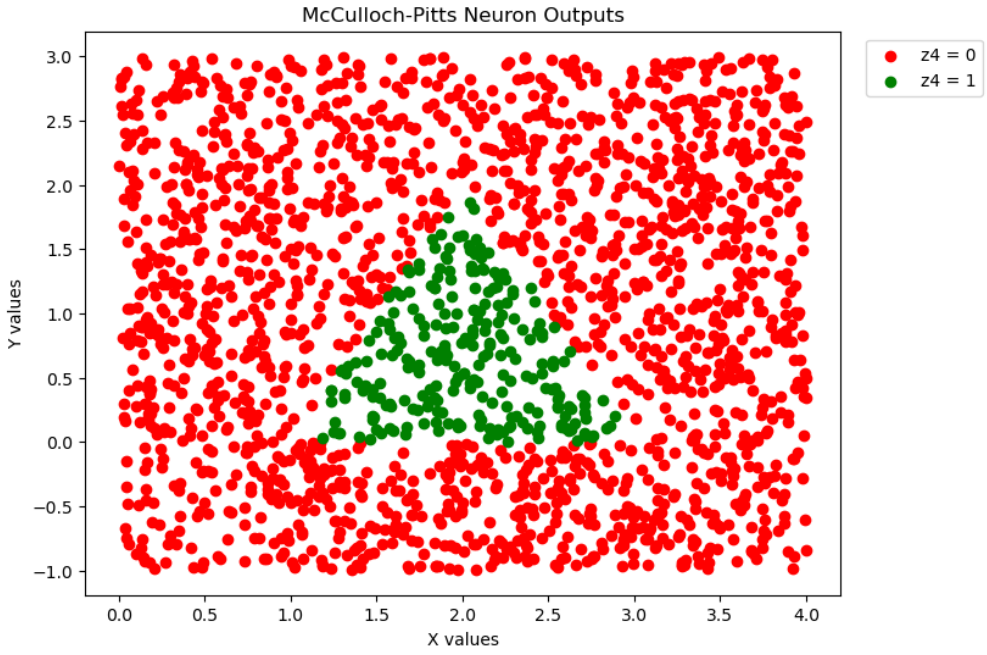
۵. ارزیابی داده‌ها با استفاده از تابع Area

برای هر نقطه تولید شده، تابع Area مقدار z۴ را محاسبه می‌کند. نقاط بر اساس مقدار z۴ به دو دسته تقسیم می‌شوند: نقاط قرمز برای z۴ = ۰ و نقاط سبز برای z۴ = ۱.

۶. رسم نمودار

نقاط قرمز و سبز با استفاده از تابع scatter در نمودار رسم می‌شوند. برچسب‌های محورها و عنوان نمودار نیز اضافه می‌شوند.

و نتیجه به شکل زیر است.




```python
def classify_and_plot(activation):
    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], activation=activation)
        if z4_value < 0.5:
            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) if red_points else ([], [])
    green_x, green_y = zip(*green_points) if green_points else ([], [])

    plt.figure(figsize=(8, 6))
    plt.scatter(red_x, red_y, color='red', label='Output < 0.5')
    plt.scatter(green_x, green_y, color='green', label='Output >= 0.5')
    plt.xlabel('X values')
    plt.ylabel('Y values')
    plt.title(f'McCulloch-Pitts Neuron Outputs with {activation} activation')
    plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1.0))
    plt.show()
```

در این بخش، به توضیح و تحلیل کد پایتونی که شامل تعریف نورون‌های McCulloch-Pitts با توابع فعال‌سازی مختلف و رسم نمودارهای مربوطه است، می‌پردازیم.

۱. وارد کردن کتابخانه‌های مورد نیاز

ابتدا، کتابخانه‌های numpy و matplotlib.pyplot وارد می‌شوند. numpy برای انجام محاسبات عددی و matplotlib.pyplot برای رسم نمودارها استفاده می‌شود.

۲. تعریف کلاس نورون McCulloch-Pitts

یک کلاس به نام McCulloch_Pitts_neuron تعریف می‌شود که ویژگی‌ها و رفتار یک نورون McCulloch-Pitts را شبیه‌سازی می‌کند. این کلاس شامل تابعهای زیر است: - سازنده کلاس: مقداردهی اولیه وزن‌ها، تابع فعال‌سازی و آستانه نورون را انجام می‌دهد. - توابع فعال‌سازی: توابع فعال‌سازی مختلفی مانند سیگموید، تانژانت هایپربولیک (tanh) و ReLU تعریف می‌شوند. - مدل نورون: ورودی را دریافت کرده و با استفاده از وزن‌ها و تابع فعال‌سازی، خروجی نورون را محاسبه می‌کند. اگر تابع فعال‌سازی threshold باشد، خروجی ۱ یا ۰ خواهد بود.

۳. تعریف تابع برای تعیین ناحیه

یک تابع به نام Area تعریف می‌شود که سه نورون McCulloch-Pitts برای اضلاع مثلث و یک نورون دیگر برای ترکیب خروجی‌های آن‌ها را ایجاد می‌کند. این تابع ورودی‌های x و y را دریافت کرده و خروجی نهایی نورون‌ها را برمی‌گرداند. تابع فعال‌سازی نیز به عنوان ورودی به این تابع داده می‌شود.

۴. تولید داده‌های تصادفی

۲۰۰۰ نقطه تصادفی برای x و y در محدوده مشخص تولید می‌شود. این مقادیر محدوده‌های محورهای x و y را پوشش می‌دهند.

۵. ارزیابی داده‌ها با استفاده از تابع Area

برای هر نقطه تولید شده، تابع Area مقدار z۴ را محاسبه می‌کند. نقاط بر اساس مقدار z۴ به دو دسته تقسیم می‌شوند: نقاط قرمز برای z۴ کمتر از ۵.۰ و نقاط سبز برای z۴ برابر یا بیشتر از ۵.۰.

۶. رسم نمودار

نقاط قرمز و سبز با استفاده از تابع scatter در نمودار رسم می‌شوند. برچسب‌های محورها و عنوان نمودار نیز اضافه می‌شوند.

۷. تحلیل نتایج

برای هر یک از توابع فعال‌سازی threshold، sigmoid، tanh و relu، نمودارهای مربوط به خروجی نورون‌های McCulloch-Pitts رسم و تحلیل می‌شوند. هر کدام از این توابع فعال‌سازی ویژگی‌ها و رفتارهای خاصی دارند که در نحوه تفکیک نقاط تاثیرگذار هستند.

تابع فعال‌سازی threshold

این تابع فعال‌سازی ساده‌ترین نوع است که خروجی آن ۰ یا ۱ است. نقاط با مقدار z۴ کمتر از آستانه قرمز و نقاط با مقدار z۴ برابر یا بیشتر از آستانه سبز نمایش داده می‌شوند.

تابع فعال‌سازی sigmoid

تابع سیگموید خروجی را به صورت پیوسته بین ۰ و ۱ تغییر می‌دهد. این تابع برای مدل‌هایی که نیاز به تصمیم‌گیری نرم دارند مناسب است.

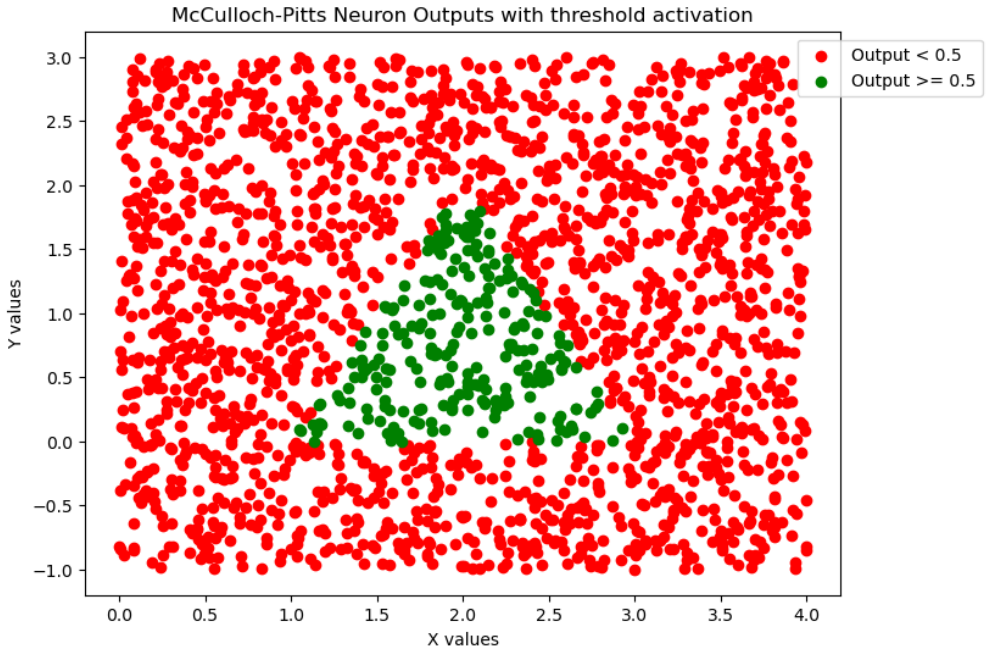
تابع فعال‌سازی tanh

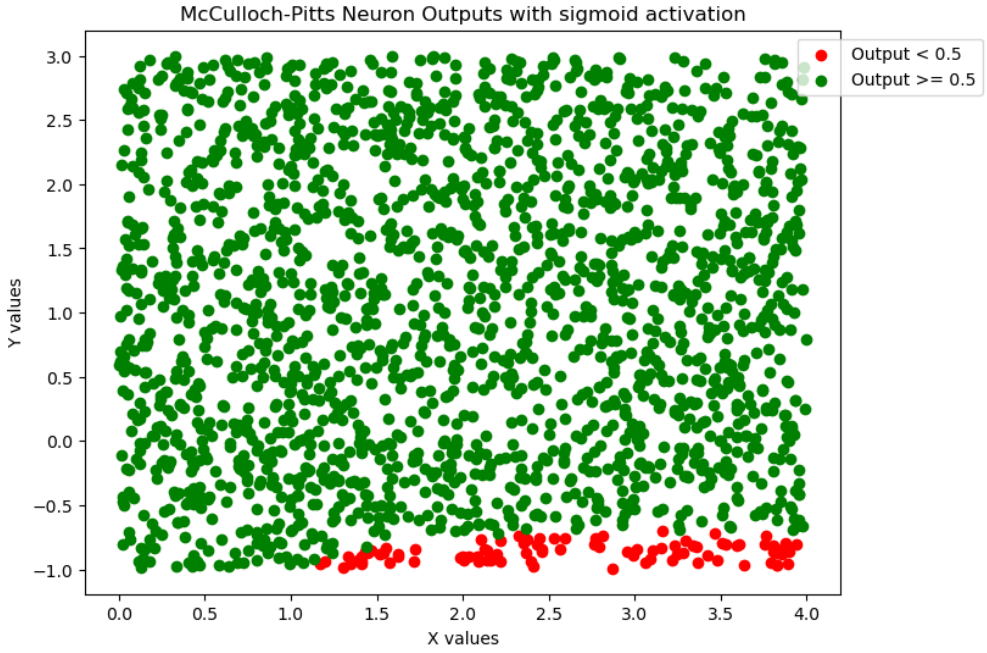
تابع تانژانت هایپربولیک خروجی را بین -۱ و ۱ تغییر می‌دهد و نسبت به تابع سیگموید دارای مرکزیت بهتری است. این تابع برای مدل‌هایی که نیاز به تفکیک نقاط با دقت بالا دارند مناسب است.

تابع فعال‌سازی relu

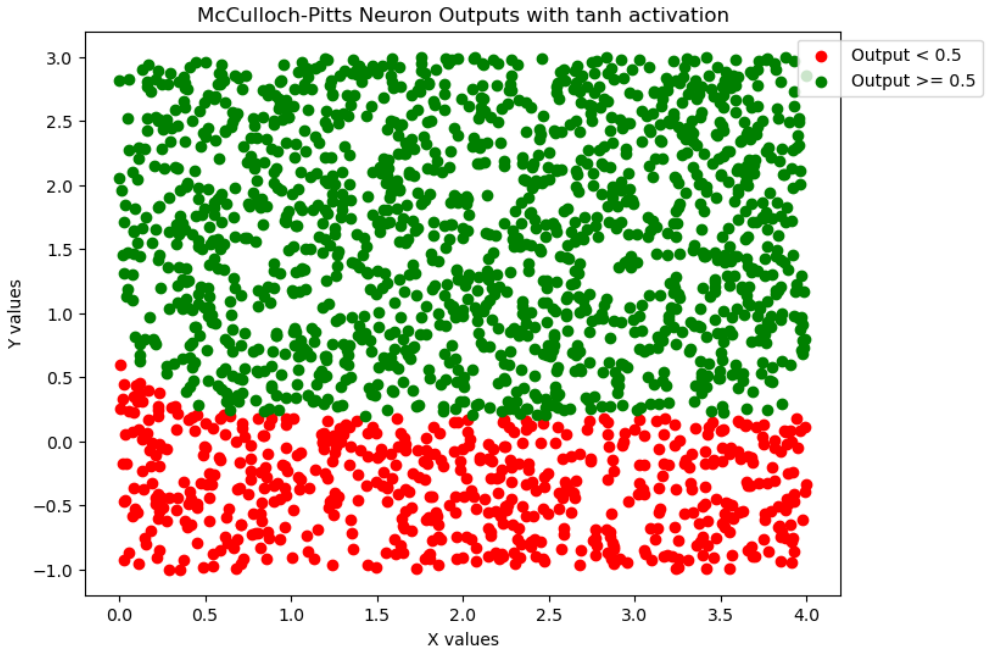
تابع ReLU (Rectified Linear Unit) خروجی را به صورت خطی تغییر می‌دهد و برای مدل‌هایی که نیاز به سرعت محاسبات بالا دارند مناسب است. این تابع به ویژه در شبکه‌های عصبی عمیق استفاده می‌شود.

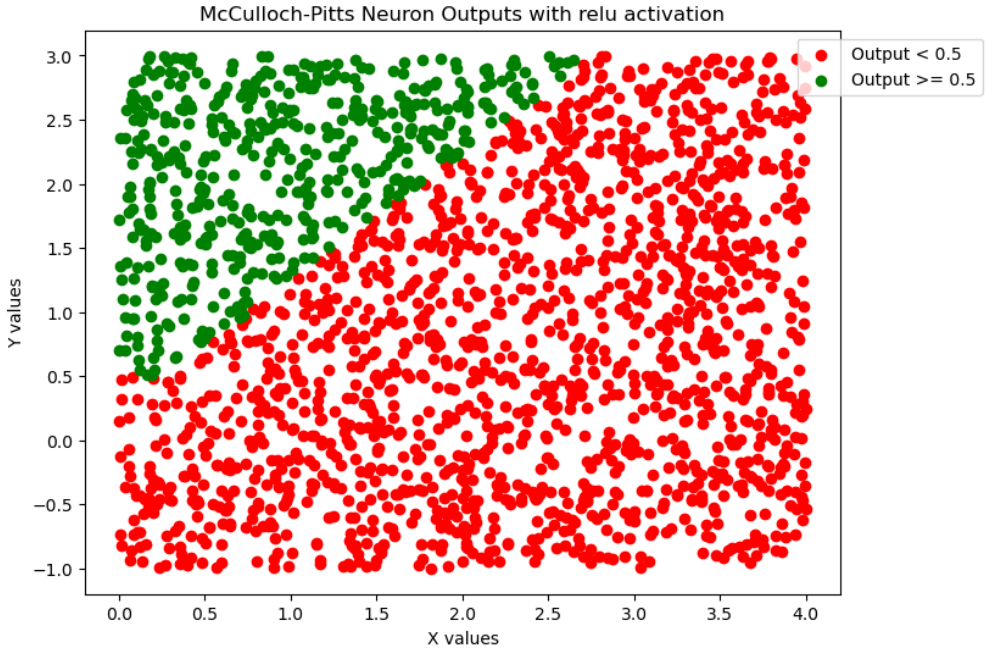
و نتایج نمودار ها به شرح زیر است:





انتخاب توابع فعال‌ساز مناسب در طراحی شبکه‌های عصبی اهمیت زیادی دارد، زیرا این توابع تأثیر مستقیم بر عملکرد کلی شبکه دارند. تغییر توابع فعال‌ساز می‌تواند نتایج متفاوتی را به همراه داشته باشد. برای مثال، یک تابع فعال‌ساز ممکن است در یک شرایط خاص بهترین عملکرد را ارائه دهد، اما در شرایط دیگر نتایج مطلوبی نداشته باشد. بنابراین، در انتخاب تابع فعال‌ساز باید با دقت و بر اساس نیازهای ویژه هر پروژه عمل کرد. همچنین، تنظیم دقیق پارامترهای شبکه و تعیین آستانه‌های مناسب (threshold) نیز از اهمیت بالایی برخوردار است تا عملیات طبقه‌بندی به‌درستی انجام شود و همه بخش‌های شبکه به طور مؤثر در فرآیند طبقه‌بندی شرکت کنند.





۲ سوال دوم

۱.۲ بخش اول سوال دوم

درباره دیتاست:

طبق مقاله Network Neural Deep and Wavelet on Based Clustering Automatic Data Fault Bearing Rolling-Element با doi:

https://doi.org/10.1155/2018/4708780

داده‌ها شامل Data Fault Bearing End Drive 48k مربوط به داده‌های ثبت شده از آزمایش‌هایی است که بر روی بلبرینگ‌های موتورهای القایی انجام شده است. این دیتاست به طور خاص برای تشخیص خرابی‌های بلبرینگ در الکتروموتورهای القایی طراحی شده است. در این مجموعه، ویژگی‌ها و حالات مختلف این دیتاست معرفی شده‌اند.

اهداف

تشخیص خرابی: اصراربرخی هدف از تولید این داده‌ها کمک به تشخیص خرابی‌های بلبرینگ در ماشین‌آلات چرخان مانند موتورهای القایی است.

ویژگی‌ها

- دقت و وضوح اندازه‌گیری: داده‌ها با دقت و وضوح بالا ثبت شده‌اند. اغلب با نرخ نمونه‌برداری ۴۸ کیلوهرتز، که امکان تشخیص دقیق خرابی را فراهم می‌کند.
- خرابی‌ها و فازم‌ها: شامل داده‌های مربوط به سه فاز مختلف خرابی‌های بلبرینگ‌های داخلی، خارجی و ترکیبی بلبرینگ. حالات‌های مختلف:
  - حالت‌های عادی که در شرایط عادی و بدون خرابی ثبت شده‌اند.
  - حالت خرابی: داده‌هایی ثبت شده در شرایط مختلف خرابی، که به منظور تشخیص مدل‌های خرابی استفاده می‌شوند.

مجموعه‌داده‌ی CWRU که در محیط آزمایشگاهی تهیه شده است، شامل داده‌های بلبرینگ‌های مورد استفاده در موتور القایی Reliance Electric با توان دو اسب بخار است. این سیستم شامل یک ترانسدیوسر گشتاور، یک دینامومتر و یک واحد کنترلی می‌باشد. داده‌ها انواع مختلف خرابی‌ها را پوشش می‌دهند و به چهار دسته تقسیم می‌شوند:

- حالت عادی با فرکانس ۴۸ کیلوهرتز
- خرابی در سمت درایو با فرکانس ۴۸ کیلوهرتز
- خرابی در سمت درایو با فرکانس ۱۲ کیلوهرتز
- خرابی در سمت فن با فرکانس ۱۲ کیلوهرتز

این دسته‌بندی‌ها شامل زیرمجموعه‌هایی برای شناسایی نوع خرابی‌ها هستند:

- خرابی بلبرینگ (B)
- خرابی‌های داخلی
- خرابی‌های خارجی، که بر اساس موقعیت نسبی نسبت به ناحیه باردهی دسته‌بندی شده‌اند:
  - مرکزی (موقعیت ساعت ۶:۰۰)
  - عمودی (موقعیت ساعت ۳:۰۰)
  - مخالف (موقعیت ساعت ۱۲:۰۰)

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

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

پیش پردازش روی دیتاست:

همانند مینی پروژه یک دیتاست را ایجاد میکنیم:

```
import scipy.io
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import seaborn

data = scipy.io.loadmat('97.mat')
a = data['X097_DE_time'].flatten()
df1 = pd.DataFrame({'X097_DE_time': a})
print('shape of X097_DE_time',df1.shape)
data = scipy.io.loadmat('105.mat')
a = data['X105_DE_time'].flatten()
df2 = pd.DataFrame({'X105_DE_time': a})
print('shape of X105_DE_time',df2.shape)
data = scipy.io.loadmat('118.mat')
a = data['X118_DE_time'].flatten()
df3 = pd.DataFrame({'X118_DE_time': a})
print('shape of X118_DE_time',df3.shape)
data = scipy.io.loadmat('130.mat')
a = data['X130_DE_time'].flatten()
df4 = pd.DataFrame({'X130_DE_time': a})
print('shape of X130_DE_time',df4.shape)
```

✓ 0.0s

```
shape of X097_DE_time (243938, 1)
shape of X105_DE_time (121265, 1)
shape of X118_DE_time (122571, 1)
shape of X130_DE_time (121991, 1)
```

در ابتدا، داده‌ها از فایل‌های 97.mat، 105.mat، 118.mat و 130.mat با استفاده از تابع scipy.io.loadmat بارگذاری می‌شوند و سپس آرایه‌ی X_DE_time از هر فایل استخراج و با استفاده از تابع ()flatten تخت می‌شود. سپس این آرایه‌ها به دیتافریم‌های Pandas تبدیل می‌شوند و شکل (تعداد سطرها و ستون‌ها) هر دیتافریم با استفاده از تابع shape چاپ می‌شود.

در پایان، این کد چهار دیتافریم به نام‌های df1، df2، df3 و df4 ایجاد می‌کند که هر یک شامل داده‌های بلبرینگ مربوط به فایل‌های مذکور هستند.

با توجه به کد بالا شرط مینی پروژه یک را پیاده میکنیم و از هر کلاس با شکل های 100×200 برمیداریم.

فیچرهای مورد نیاز با توجه به مینی پروژه ۱ استخراج و پیاده سازی میشوند. نتایج و خروجی دیتاست آماده شده به صورت شکل بالا میباشد. همانطور که مشخص است ستون ها شامل فیچر های استخراج شده و لیبل دیتا ها میباشد.

- تقسیم داده‌ها به ویژگی‌ها و برچسب‌ها: در این مرحله، دیتافریم statistics_df به دو قسمت تقسیم می‌شود: X که شامل ویژگی‌های (یا

```python
M = 200
N = 100
data = []
for i in range (1,M):
    data.append(df1['X097_DE_time'].sample(n=N, replace=True, random_state=i).to_list())
df1 = pd.DataFrame(data)
df1['label']=0
print('shape of dataset from class 0',df1.shape)
######################
data = []
for i in range (1,M):
    data.append(df2['X105_DE_time'].sample(n=N, replace=True, random_state=i).to_list())
df2 = pd.DataFrame(data)
df2['label']=1
print('shape of dataset from class 1',df2.shape)
######################
data = []
for i in range (1,M):
    data.append(df3['X118_DE_time'].sample(n=N, replace=True, random_state=i).to_list())
df3 = pd.DataFrame(data)
df3['label']=2
print('shape of dataset from class 2',df3.shape)
######################
data = []
for i in range (1,M):
    data.append(df4['X130_DE_time'].sample(n=N, replace=True, random_state=i).to_list())
df4 = pd.DataFrame(data)
df4['label']=3
print('shape of dataset from class 3',df4.shape)
df = pd.DataFrame()
df = pd.concat([df1,df2,df3,df4], axis = 0)
df = df.drop_duplicates().reset_index(drop=True)
print('shape of all dataset after droping duplicates value',df.shape)
```

✓ 0.3s Python

```
shape of dataset from class 0 (199, 101)
shape of dataset from class 1 (199, 101)
shape of dataset from class 2 (199, 101)
shape of dataset from class 3 (199, 101)
shape of all dataset after droping duplicates value (796, 101)
```

متغیرهای مستقل) داده‌ها است و y که شامل برچسب‌ها (یا متغیر وابسته) است. این کار برای جداسازی ویژگی‌هایی که قرار است مدل از آن‌ها یاد بگیرد از برچسب‌هایی که مدل باید پیش‌بینی کند، انجام می‌شود.

- تقسیم داده‌ها به مجموعه‌های آموزشی، اعتبارسنجی و تست: این کد از تابع train_test_split کتابخانه sklearn استفاده می‌کند تا داده‌ها را به مجموعه‌های مختلف تقسیم کند:

  - ابتدا، ٪۹۰ از داده‌ها برای آموزش (train) و ٪۱۰ برای اعتبارسنجی (validation) انتخاب می‌شود. این تقسیم با تنظیم پارامتر random_state به ۴۴، تصادفی ولی قابل تکرار انجام می‌شود.
  - سپس، ٪۸۰ از داده‌های آموزشی اولیه به عنوان داده‌های تست (test) و ٪۲۰ باقی‌مانده به عنوان داده‌های آموزشی نهایی تقسیم می‌شوند. این کار نیز با استفاده از پارامترهای مشابه انجام می‌شود.

- چاپ اشکال مجموعه‌های داده:

این بخش از کد اشکال (تعداد نمونه‌ها و ویژگی‌ها) مجموعه‌های اعتبارسنجی، آموزشی و تست را چاپ می‌کند تا مطمئن شویم که تقسیم‌بندی داده‌ها به درستی انجام شده است.

```python
import pandas as pd
import numpy as np

def calculate_peak(data):
    return np.max(data)

def calculate_crest_factor(data):
    peak = np.max(data)
    rms = np.sqrt(np.mean(np.square(data)))
    return peak / rms

def calculate_clearance_factor(data):
    peak = np.max(data)
    mean = np.mean(data)
    return peak / mean

def calculate_square_mean_root(data):
    return np.sqrt(np.mean(np.square(data)))

def calculate_absolute_mean(data):
    return np.mean(np.abs(data))
def calculate_root_mean_square(data):
    return np.sqrt(np.mean(np.square(data)))
def calculate_impact_factor(peak, rms):
    return peak / rms
datas = []
for index, row in df.iterrows():
    label = row['label']
    data = row.iloc[:-1]
    dat=[]
    rms = calculate_root_mean_square(data)
    peak = calculate_peak(data)
    dat.append(np.std(data))
    dat.append(calculate_peak(data))
    dat.append(calculate_crest_factor(data))
    dat.append(calculate_clearance_factor(data))
    dat.append(calculate_square_mean_root(data))
    dat.append(np.mean(data))
    dat.append(calculate_absolute_mean(data))
    dat.append(calculate_impact_factor(peak, rms))

    datas.append(dat)
statistics_df = pd.DataFrame(datas)
statistics_df['label'] = df['label']
statistics_df = statistics_df.rename(columns={0:'std',1:'peak',2:'crest',3:'clearance',4:'smr',5:'mean',6:'absolute_mean',7:'impact_factor'})
statistics_df
```

	std	peak	crest	clearance	smr	mean	absolute_mean	impact_factor	label
0	0.065845	0.190049	2.678863	7.196461	0.070944	0.026409	0.055339	2.678863	0
1	0.073314	0.221758	2.907109	10.524752	0.076281	0.021070	0.061763	2.907109	0
2	0.071386	0.183790	2.482838	9.384320	0.074024	0.019585	0.060720	2.482838	0
3	0.070100	0.162094	2.299034	21.464088	0.070505	0.007552	0.058817	2.299034	0
4	0.064244	0.174194	2.376593	4.937031	0.073296	0.035283	0.060288	2.376593	0
...	...	...	...	...	...	...	...	...	...
791	0.593558	1.653184	2.773795	-30.671288	0.596001	-0.053900	0.380801	2.773795	3
792	0.684832	2.627388	3.835119	-140.774587	0.685087	-0.018664	0.386547	3.835119	3
793	0.568679	1.697447	2.984826	-434.511435	0.568692	-0.003907	0.348764	2.984826	3
794	0.675883	3.081800	4.548325	64.521340	0.677568	0.047764	0.413576	4.548325	3
795	0.750590	2.438151	3.247757	175.658280	0.750719	0.013880	0.442969	3.247757	3

796 rows × 9 columns

اهمیت این تقسیم‌بندی

تقسیم داده‌ها به مجموعه‌های آموزشی، اعتبارسنجی و تست اهمیت بالایی دارد زیرا:

- مجموعه آموزشی **(Training set)**: برای آموزش مدل استفاده می‌شود.
- مجموعه اعتبارسنجی **(Validation set)**: برای تنظیم پارامترهای مدل و انتخاب بهترین مدل استفاده می‌شود. این مجموعه کمک می‌کند تا از بروز overfitting جلوگیری کنیم.
- مجموعه تست **(Test set)**: برای ارزیابی نهایی مدل استفاده می‌شود و به ما نشان می‌دهد که مدل در داده‌های جدید (که قبلاً ندیده) چگونه عمل می‌کند.

```
X = statistics_df.drop(columns=['label'])
y = statistics_df['label']

from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y,
                                                  train_size=0.9, random_state=44, shuffle=True)
X_train, X_test, y_train, y_test = train_test_split(X_train,y_train,
                                                    test_size=0.8, random_state=44, shuffle=True)
print('Valid dataset shape:', X_val.shape)
print('Train dataset shape:', X_train.shape)
print('Test dataset shape:', X_test.shape)
```

✓ 0.0s

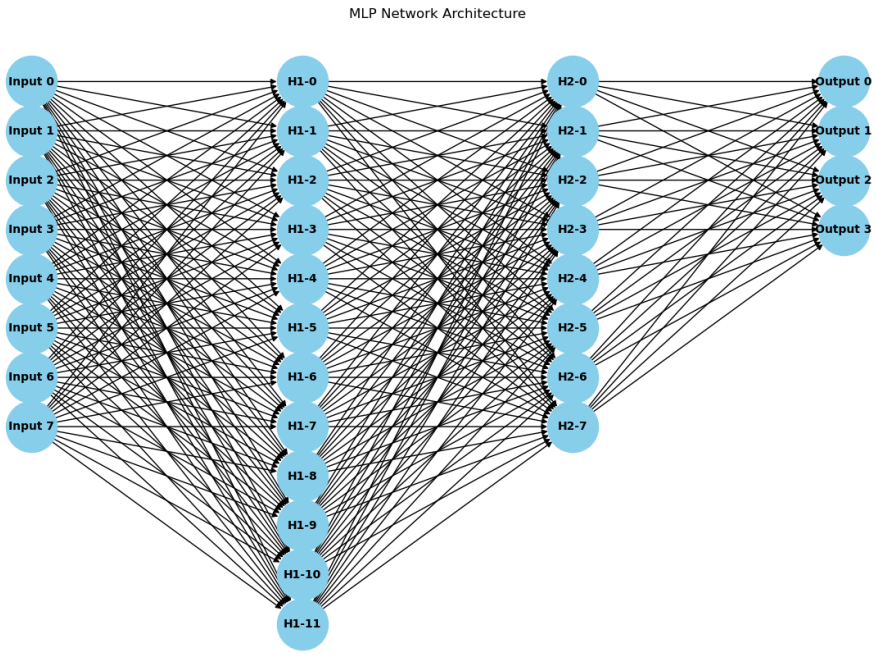
```
Valid dataset shape: (80, 8)
Train dataset shape: (143, 8)
Test dataset shape: (573, 8)
```

این روش تضمین می‌کند که مدل نهایی از عملکرد خوبی در مواجهه با داده‌های جدید و ناشناخته برخوردار است. دیتاست را باید scale کنیم

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_val = scaler.transform(X_val)
```

با توجه به اینکه مدل باید جنرال باشد، روی دیدتاست train فیت میکنیم و روی دیتاست های valid و test ترنسفر میکنیم.

۲.۲ بخش دوم سوال دوم



معماری ایجاد شده برای شبکه MLP با تعداد ویژگی ورودی و ۴ کلاس خروجی و ۲ لایه پنهان با سایز های ۱۲ و ۸.

توابع فعال‌سازی و هزینه

- تابع **relu(x)**: تابع فعال‌سازی ReLU که مقدارهای منفی را به صفر و مقدارهای مثبت را بدون تغییر برمی‌گرداند.
- تابع **sigmoid(x)**: تابع فعال‌سازی سیگموید که خروجی را به بازه $(0, 1)$ نگاشت می‌کند.
- تابع **bce(y, y_hat)**: تابع هزینه باینری کراس انتروپی که برای مسائل طبقه‌بندی باینری استفاده می‌شود.
- تابع **mse(y, y_hat)**: تابع هزینه میانگین مربعات خطا که برای مسائل رگرسیون استفاده می‌شود.
- تابع **categorical_cross_entropy(y, y_hat)**: تابع هزینه انتروپی متقاطع برای مسائل طبقه‌بندی چندکلاسه.
- تابع **accuracy(y, y_hat, t=0.5)**: تابع دقت که تعداد پیش‌بینی‌های صحیح را محاسبه می‌کند.

کلاس MLP

- تابع **__init__**: این تابع مقداردهی اولیه شبکه عصبی را انجام می‌دهد و پارامترهای مربوط به اندازه لایه‌ها، نوع فعال‌سازی، تعداد تکرارها، تابع هزینه، نرخ یادگیری و مقدار تصادفی را تنظیم می‌کند.
- تابع **_init_weights**: این تابع وزن‌ها و بایاس‌های لایه‌های شبکه را به صورت تصادفی مقداردهی اولیه می‌کند.
- تابع **fit**: این تابع شبکه را بر روی داده‌های آموزشی و اعتبارسنجی آموزش می‌دهد و تاریخچه‌ی خطا و دقت را ذخیره می‌کند.
- تابع **plot_history**: این تابع تاریخچه‌ی خطا و دقت شبکه را در طول دوره‌های آموزشی به صورت نمودار نمایش می‌دهد.

```python
def relu(x):
    return np.maximum(0, x)
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def bce(y, y_hat):
    return np.mean(-(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat)))
def mse(y, y_hat):
    return np.mean((y - y_hat)**2)
def categorical_cross_entropy(y, y_hat):
    return -np.mean(np.sum(y * np.log(y_hat), axis=1))
def accuracy(y, y_hat, t=0.5):
    y_hat = np.where(y_hat < t, 0, 1)
    acc = np.sum(y == y_hat) / len(y)
    return acc
class MLP:
    def __init__(self, hidden_layer_sizes, hidden_activation='relu',
                 output_size=1, output_activation='sigmoid',
                 n_iter=1000, loss_fn=bce, eta=0.1, random_state=None):
        self.hidden_layer_sizes = hidden_layer_sizes
        self.hidden_activation = hidden_activation
        self.output_size = output_size
        self.output_activation = output_activation
        self.n_iter = n_iter
        self.loss_fn = loss_fn
        self.eta = eta
        self.random_state = random_state
        np.random.seed(self.random_state)
    def _init_weights(self):
        self.ws, self.bs = [], []
        self.as_ = [None] * len(self.hidden_layer_sizes)
        all_layers = [self.input_size] + self.hidden_layer_sizes + [self.output_size]
        num_layers = len(all_layers)
        for i in range(1, num_layers):
            w = np.random.randn(all_layers[i-1], all_layers[i])
            b = np.random.randn(all_layers[i])
            self.ws.append(w)
            self.bs.append(b)
```

- تابع **gradient_descent_**: این تابع با استفاده از الگوریتم گرادیان نزولی وزن‌ها و بایاس‌های شبکه را به‌روزرسانی می‌کند.
- تابع **predict**: این تابع خروجی شبکه را برای داده‌های ورودی پیش‌بینی می‌کند.
- تابع **activation_function_**: این تابع تابع فعال‌سازی مورد نظر (ReLU یا سیگموید) را اعمال می‌کند.
- تابع **activation_derivative_**: این تابع مشتق تابع فعال‌سازی مورد نظر را محاسبه می‌کند.

تابع fit

- این تابع برای آموزش شبکه عصبی با استفاده از داده‌های ورودی و برچسب‌های مربوطه استفاده می‌شود.
- ابتدا داده‌های ورودی X و برچسب‌ها y و همچنین داده‌های اعتبارسنجی X_val و y_val (در صورت وجود) به عنوان ورودی دریافت می‌شوند.
- اندازه ورودی input_size بر اساس شکل داده‌های X تنظیم می‌شود.
- وزن‌ها و بایاس‌های شبکه با استفاده از تابع _init_weights مقداردهی اولیه می‌شوند.

```python
def fit(self, X, y, X_val=None, y_val=None):
    n, self.input_size = X.shape
    self._init_weights()
    train_losses = []
    val_losses = []
    train_accs = []
    val_accs = []
    for _ in range(self.n_iter):
        y_hat = self.predict(X)
        loss = self.loss_fn(y, y_hat)
        self._gradient_descent(X, y, y_hat)
        train_losses.append(loss)
        train_acc = accuracy(y, y_hat)
        train_accs.append(train_acc)
        if X_val is not None and y_val is not None:
            val_loss = self.loss_fn(y_val, self.predict(X_val))
            val_losses.append(val_loss)
            val_acc = accuracy(y_val, self.predict(X_val))
            val_accs.append(val_acc)
            print(f"Train Loss: {loss:.4f} | Train Acc: {train_acc:.4f} | Val Loss:
        else:
            print(f"Train Loss: {loss:.4f} | Train Acc: {train_acc:.4f}")
    if X_val is not None and y_val is not None:
        self.plot_history(train_losses, val_losses, train_accs, val_accs)
    else:
        self.plot_history(train_losses, None, train_accs, None)
```

- لیست‌هایی برای ذخیره‌سازی تاریخچه خطاهای آموزشی، خطاهای اعتبارسنجی، دقت‌های آموزشی و دقت‌های اعتبارسنجی ایجاد می‌شوند.
- حلقه آموزشی به تعداد n_iter بار تکرار می‌شود.
  - پیش‌بینی خروجی y_hat با استفاده از داده‌های X انجام می‌شود.
  - خطای فعلی با استفاده از تابع هزینه محاسبه و به لیست خطاهای آموزشی اضافه می‌شود.
  - گرادیان نزولی برای به‌روزرسانی وزن‌ها و بایاس‌ها استفاده می‌شود.
  - دقت فعلی با استفاده از پیش‌بینی‌ها محاسبه و به لیست دقت‌های آموزشی اضافه می‌شود.
  - اگر داده‌های اعتبارسنجی وجود داشته باشند، خطا و دقت اعتبارسنجی محاسبه و به لیست‌های مربوطه اضافه می‌شوند.
  - خطا و دقت آموزشی و اعتبارسنجی در هر تکرار چاپ می‌شوند.
- پس از اتمام حلقه آموزشی، تاریخچه خطاها و دقت‌ها با استفاده از تابع plot_history رسم می‌شوند.

تابع plot_history

- این تابع تاریخچه خطاها و دقت‌های آموزشی و اعتبارسنجی را به صورت نمودارهای خطی رسم می‌کند.
- ابتدا یک شکل با دو نمودار ایجاد می‌شود.
- در نمودار اول، خطای آموزشی و در صورت وجود، خطای اعتبارسنجی رسم می‌شود.
- در نمودار دوم، دقت آموزشی و در صورت وجود، دقت اعتبارسنجی رسم می‌شود.

```
def plot_history(self, train_losses, val_losses=None, train_accs=None, val_accs=None
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label='Training Loss')
    if val_losses is not None:
        plt.plot(val_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training and Validation Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(train_accs, label='Training Accuracy')
    if val_accs is not None:
        plt.plot(val_accs, label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()

    plt.show()
def _gradient_descent(self, X, y, y_hat):
    delta = y_hat - y
    for j in range(len(self.ws)-1, 0, -1):
        w_grad = (self.as_[j-1].T @ delta) / len(y)
        b_grad = delta.mean(0)
        self.ws[j] -= self.eta * w_grad
        self.bs[j] -= self.eta * b_grad
        delta = (delta @ self.ws[j].T) * (self._activation_derivative(self.hs[j-1],
def predict(self, X):
    self.hs = []
    self.as_ = []
    for i, (w, b) in enumerate(zip(self.ws[:-1], self.bs[:-1])):
        a = self.as_[i-1].copy() if i>0 else X.copy()
        self.hs.append(a @ w + b)
        self.as_.append(self._activation_function(self.hs[i], self.hidden_activation
    y = self._activation_function(self.as_[-1] @ self.ws[-1] + self.bs[-1], self.out
    return y
```

- برچسب‌ها، عناوین و افسانه‌ها به نمودارها اضافه می‌شوند.
- در نهایت، نمودارها نمایش داده می‌شوند.

تابع _gradient_descent

- این تابع از الگوریتم گرادیان نزولی برای به‌روزرسانی وزن‌ها و بایاس‌های شبکه استفاده می‌کند.
- ابتدا خطای بین پیش‌بینی‌ها و برچسب‌های واقعی محاسبه می‌شود.
- برای هر لایه، گرادیان وزن‌ها و بایاس‌ها محاسبه و وزن‌ها و بایاس‌ها به‌روزرسانی می‌شوند.
- گرادیان‌ها با توجه به مشتق تابع فعال‌سازی و خطا محاسبه می‌شوند.

تابع predict

- این تابع برای پیش‌بینی خروجی شبکه با استفاده از داده‌های ورودی استفاده می‌شود.
- ابتدا لیست‌های hs و _as برای ذخیره‌سازی مقادیر میانی ایجاد می‌شوند.
- برای هر لایه، خروجی لایه قبلی به عنوان ورودی لایه فعلی استفاده می‌شود و پس از اعمال وزن‌ها و بایاس‌ها و تابع فعال‌سازی، خروجی لایه محاسبه می‌شود.
- در نهایت، خروجی نهایی شبکه برگردانده می‌شود.

```
def _activation_function(self, x, activation):
    if activation == 'relu':
        return np.maximum(0, x)
    elif activation == 'sigmoid':
        return 1 / (1 + np.exp(-x))
    else:
        raise ValueError("Invalid activation function.")
def _activation_derivative(self, x, activation):
    if activation == 'relu':
        return np.where(x > 0, 1, 0)
    elif activation == 'sigmoid':
        sigmoid = self._activation_function(x, 'sigmoid')
        return sigmoid * (1 - sigmoid)
    else:
        raise ValueError("Invalid activation function.")
```

۳.۲ بخش سوم سوال دوم

تابع _activation_function

- این تابع تابع فعال‌سازی را برای ورودی‌های داده شده اعمال می‌کند.
- اگر تابع فعال‌سازی relu باشد، مقدارهای منفی را به صفر و مقدارهای مثبت را بدون تغییر برمی‌گرداند.
- اگر تابع فعال‌سازی sigmoid باشد، خروجی را به بازه $(0, 1)$ نگاشت می‌کند.
- در غیر این صورت، خطایی با پیام "تابع فعال‌سازی نامعتبر" ایجاد می‌کند.

تابع _activation_derivative

- این تابع مشتق تابع فعال‌سازی را برای ورودی‌های داده شده محاسبه می‌کند.
- اگر تابع فعال‌سازی relu باشد، مشتق را بر اساس مقدارهای ورودی محاسبه می‌کند، به طوری که اگر ورودی بزرگتر از صفر باشد، مشتق یک و در غیر این صورت صفر است.
- اگر تابع فعال‌سازی sigmoid باشد، ابتدا مقدار سیگموید محاسبه می‌شود و سپس مشتق آن بر اساس فرمول sigmoid * (1 - sigmoid) به دست می‌آید.
- در غیر این صورت، خطایی با پیام "تابع فعال‌سازی نامعتبر" ایجاد می‌کند.

کتابخانه‌های مورد استفاده

- matplotlib.pyplot برای رسم نمودارها.
- networkx برای ایجاد و ترسیم گراف‌ها.

تابع draw_mlp

- این تابع ساختار یک شبکه عصبی چند لایه (MLP) را رسم می‌کند.

اضافه کردن نودها

- لایه‌ها بر اساس اندازه ورودی، اندازه لایه‌های مخفی و اندازه خروجی تعریف می‌شوند.
- یک گراف جهت‌دار (DiGraph) ایجاد می‌شود.
- نودها به گراف اضافه می‌شوند و موقعیت و برچسب آن‌ها تنظیم می‌شود.
- اگر نود در لایه ورودی باشد، برچسب Input به آن داده می‌شود.
- اگر نود در لایه خروجی باشد، برچسب Output به آن داده می‌شود.
- نودهای لایه‌های مخفی با برچسب H و شماره لایه و نود مشخص می‌شوند.

```python
import matplotlib.pyplot as plt
import networkx as nx

def draw_mlp(hidden_layer_sizes, input_size, output_size):
    G = nx.DiGraph()

    # Adding nodes
    layers = [input_size] + hidden_layer_sizes + [output_size]
    node_id = 0
    pos = {}
    labels = {}

    for layer_index, layer_size in enumerate(layers):
        for node_index in range(layer_size):
            node_name = f"Layer {layer_index} Neuron {node_index}"
            G.add_node(node_id, layer=layer_index)
            pos[node_id] = (layer_index, -node_index)
            if layer_index == 0:
                labels[node_id] = f"Input {node_index}"
            elif layer_index == len(layers) - 1:
                labels[node_id] = f"Output {node_index}"
            else:
                labels[node_id] = f"H{layer_index}-{node_index}"
            node_id += 1

    # Adding edges
    for layer_index in range(len(layers) - 1):
        current_layer_size = layers[layer_index]
        next_layer_size = layers[layer_index + 1]
        for current_node in range(current_layer_size):
            for next_node in range(next_layer_size):
                current_node_id = sum(layers[:layer_index]) + current_node
                next_node_id = sum(layers[:layer_index+1]) + next_node
                G.add_edge(current_node_id, next_node_id)

    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, labels=labels, with_labels=True, node_size=2000, node_color="skyblue
    plt.title("MLP Network Architecture")
    plt.show()
```

اضافه کردن یال‌ها

- برای هر لایه، یال‌هایی بین نودهای لایه فعلی و لایه بعدی اضافه می‌شوند.
- یال‌ها گراف را جهت‌دار می‌کنند و نشان‌دهنده ارتباطات بین نودها در لایه‌های مختلف هستند.

رسم گراف

- یک شکل با اندازه مشخص ایجاد می‌شود.
- گراف با استفاده از موقعیت‌ها و برچسب‌ها رسم می‌شود.
- برچسب‌ها، اندازه نودها و رنگ نودها تنظیم می‌شوند.
- عنوان گراف تنظیم و سپس نمایش داده می‌شود.

نتایج:

تحلیل تابع هزینه Categorical Cross-Entropy

تابع هزینه Categorical Cross-Entropy یکی از پرکاربردترین توابع هزینه در مسائل طبقه‌بندی چندکلاسه است. این تابع به منظور محاسبه تفاوت بین توزیع پیش‌بینی شده توسط مدل و توزیع واقعی برچسب‌ها استفاده می‌شود. در ادامه، فرمول ریاضی و تحلیل این تابع هزینه ارائه می‌شود.

فرمول ریاضی

فرض کنید که y بردار واقعی برچسب‌ها و $\hat{y}$ بردار پیش‌بینی شده توسط مدل باشد. اگر تعداد کلاس‌ها را C و تعداد نمونه‌ها را N در نظر بگیریم، تابع هزینه Categorical Cross-Entropy به صورت زیر تعریف می‌شود:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

در این فرمول:

- N تعداد نمونه‌ها است.
- C تعداد کلاس‌ها است.
- $y_{i,c}$ مقدار واقعی برچسب کلاس c برای نمونه i است که معمولاً مقدار آن یا ۰ یا ۱ است (در حالت one-hot encoding).
- $\hat{y}_{i,c}$ احتمال پیش‌بینی شده برای کلاس c توسط مدل برای نمونه i است.

تحلیل تابع هزینه

- هدف: هدف از این تابع هزینه، مینیمم کردن تفاوت بین توزیع واقعی و توزیع پیش‌بینی شده است. وقتی مدل به درستی آموزش ببیند، احتمال پیش‌بینی شده $\hat{y}_{i,c}$ برای کلاس صحیح c به ۱ نزدیک می‌شود و سایر احتمالات به ۰ میل می‌کنند.
- تفسیر لگاریتم: وجود تابع لگاریتم در فرمول باعث می‌شود که مدل برای اشتباهات بزرگ‌تر جریمه بیشتری دریافت کند. به این معنی که اگر مدل یک نمونه را با اطمینان بالا به اشتباه پیش‌بینی کند، مقدار تابع هزینه به طور قابل توجهی افزایش می‌یابد.
- تاثیر نرمال‌سازی: تقسیم مجموع توسط تعداد نمونه‌ها N باعث می‌شود که مقدار تابع هزینه به ازای هر نمونه به دست آید، که این امر به تحلیل بهتر عملکرد مدل کمک می‌کند.

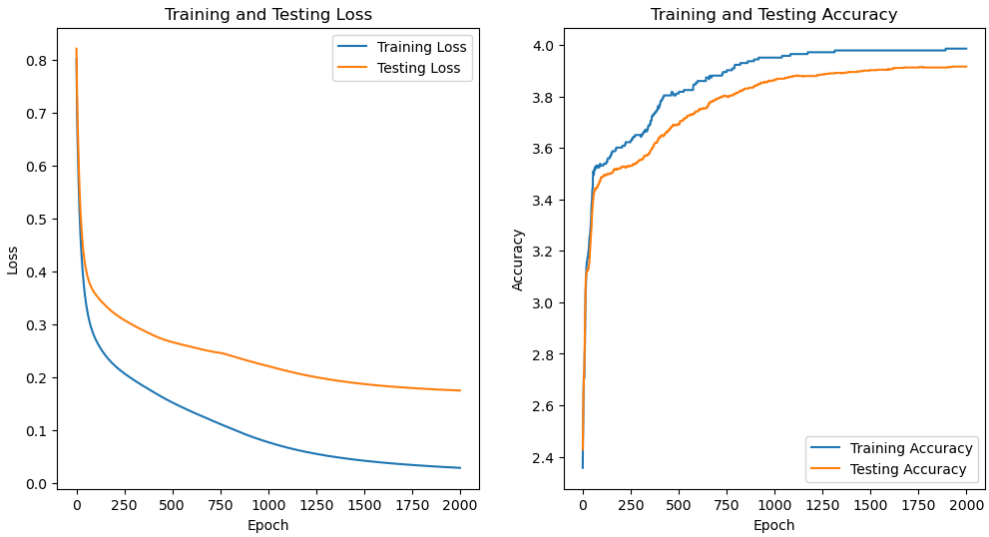
به طور کلی، تابع هزینه Categorical Cross-Entropy به دلیل توانایی در به دست آوردن تفاوت‌های کوچک بین توزیع‌های واقعی و پیش‌بینی شده و جریمه کردن اشتباهات بزرگ، یکی از توابع هزینه موثر و پرکاربرد در مسائل طبقه‌بندی چندکلاسه است.

در نمودار سمت چپ، محور عمودی نشان‌دهنده مقدار تابع هزینه (Loss) و محور افقی نشان‌دهنده تعداد تکرارها (Epoch) است.

- تابع هزینه آموزش **(Training Loss)**: خط آبی رنگ نشان‌دهنده تغییرات تابع هزینه برای داده‌های آموزش در طول تکرارها است. مشاهده می‌شود که تابع هزینه به صورت یکنواخت کاهش می‌یابد و به مقدار پایینی می‌رسد، که نشان‌دهنده بهبود مدل در طی فرآیند آموزش است.
- تابع هزینه تست **(Testing Loss)**: خط نارنجی رنگ نشان‌دهنده تغییرات تابع هزینه برای داده‌های تست است. این خط نیز به صورت یکنواخت کاهش می‌یابد، اما در مقایسه با خط آبی، مقدار تابع هزینه بالاتری دارد. این اختلاف نشان‌دهنده وجود احتمال overfitting در مدل است، به این معنا که مدل در داده‌های آموزش عملکرد بهتری نسبت به داده‌های تست دارد.

نمودار دقت

در نمودار سمت راست، محور عمودی نشان‌دهنده مقدار دقت (Accuracy) و محور افقی نشان‌دهنده تعداد تکرارها (Epoch) است.



- دقت آموزش **(Training Accuracy)**: خط آبی رنگ نشان‌دهنده تغییرات دقت برای داده‌های آموزش در طول تکرارها است. مشاهده می‌شود که دقت مدل به سرعت افزایش می‌یابد و به مقدار بالایی می‌رسد، که نشان‌دهنده یادگیری خوب مدل از داده‌های آموزش است.
- دقت تست **(Testing Accuracy)**: خط نارنجی رنگ نشان‌دهنده تغییرات دقت برای داده‌های تست است. این خط نیز به صورت یکنواخت افزایش می‌یابد، اما دقت کمتری نسبت به دقت آموزش دارد. این اختلاف می‌تواند نشان‌دهنده overfitting باشد، زیرا مدل در داده‌های تست عملکرد کمتری دارد.

به طور کلی، هر دو نمودار نشان می‌دهند که مدل در طول تکرارها بهبود یافته و به دقت بالایی رسیده است. اما اختلاف بین دقت و تابع هزینه آموزش و تست نشان‌دهنده احتمال overfitting است که باید با روش‌های مختلف مانند regularization، افزایش تعداد داده‌های آموزش و یا استفاده از تکنیک‌های داده‌افزایی (data augmentation) بهبود یابد.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	22
1	0.94	0.89	0.91	18
2	0.95	1.00	0.97	19
3	0.95	0.95	0.95	21
accuracy			0.96	80
macro avg	0.96	0.96	0.96	80
weighted avg	0.96	0.96	0.96	80

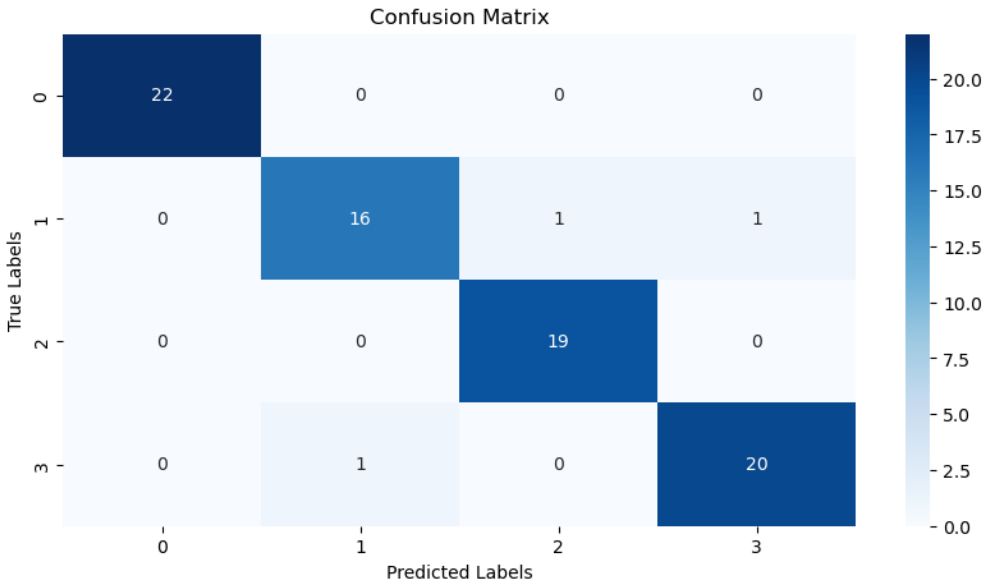
- کلاس ۰: مدل با دقت، بازخوانی و امتیاز F1 برابر با ۱.۰۰ عمل کرده است که نشان‌دهنده عملکرد عالی مدل در این کلاس است.
- کلاس ۱: مدل با دقت ۹۴.۰ و بازخوانی ۸۹.۰ و امتیاز F1 برابر با ۹۱.۰ عملکرد خوبی داشته، اما مقدار بازخوانی نشان می‌دهد که چند نمونه از این کلاس به درستی تشخیص داده نشده‌اند.

- کلاس ۲: دقت مدل ۹۵.۰، بازخوانی ۱.۰۰ و امتیاز F1 برابر با ۹۷.۰ است که نشان‌دهنده این است که مدل چند نمونه از این کلاس را به اشتباه به کلاس‌های دیگر نسبت داده است.
- کلاس ۳: مدل با دقت، بازخوانی و امتیاز F1 برابر با ۹۵.۰ عمل کرده است که نشان‌دهنده عملکرد خوب مدل در این کلاس است.

معیارهای کلی

- دقت کل **(Accuracy)**: مدل با دقت ۹۶.۰ عمل کرده است که نشان‌دهنده عملکرد بسیار خوب کلی مدل است.
- میانگین ماکرو **(Macro Avg)**: میانگین دقت، بازخوانی و امتیاز F1 برای همه کلاس‌ها برابر با ۹۶.۰ است که نشان‌دهنده تعادل خوب مدل در تمام کلاس‌ها است.
- میانگین وزنی **(Weighted Avg)**: میانگین دقت، بازخوانی و امتیاز F1 وزنی برای همه کلاس‌ها نیز برابر با ۹۶.۰ است که نشان‌دهنده این است که مدل در کلیه کلاس‌ها عملکرد تقریباً یکنواختی دارد.

به طور کلی، نتایج نشان می‌دهند که مدل با عملکرد بسیار خوبی در تمام کلاس‌ها و معیارها عمل کرده و دقت کلی بسیار بالایی دارد. این نتایج نشان‌دهنده توانایی مدل در تشخیص صحیح نمونه‌ها و کاهش خطاهای تشخیصی است.



- کلاس ۰: تمام ۲۲ نمونه به درستی به کلاس ۰ پیش‌بینی شده‌اند.
- کلاس ۱: از ۱۸ نمونه، ۱۶ نمونه به درستی به کلاس ۱ پیش‌بینی شده‌اند، یک نمونه به اشتباه به کلاس ۲ و یک نمونه به کلاس ۳ نسبت داده شده است.
- کلاس ۲: تمام ۱۹ نمونه به درستی به کلاس ۲ پیش‌بینی شده‌اند.
- کلاس ۳: از ۲۱ نمونه، ۲۰ نمونه به درستی به کلاس ۳ پیش‌بینی شده‌اند و یک نمونه به اشتباه به کلاس ۱ نسبت داده شده است.

این ماتریس نشان می‌دهد که مدل در تشخیص کلاس‌ها عملکرد بسیار خوبی داشته و تنها چند نمونه از کلاس ۱ به اشتباه به کلاس‌های دیگر نسبت داده شده‌اند. این نتایج نشان‌دهنده دقت بالای مدل در تشخیص نمونه‌های مختلف است.

تحلیل تابع هزینه Mean Squared Error (MSE)

تابع هزینه Mean Squared Error (MSE) یکی از پرکاربردترین توابع هزینه در مسائل رگرسیون است. این تابع به منظور محاسبه میانگین مربعات خطاها بین مقادیر واقعی و مقادیر پیش‌بینی شده توسط مدل استفاده می‌شود. در ادامه، فرمول ریاضی و تحلیل این تابع هزینه ارائه می‌شود.

فرمول ریاضی

فرض کنید که y بردار مقادیر واقعی و $\hat{y}$ بردار مقادیر پیش‌بینی شده توسط مدل باشد. اگر تعداد نمونه‌ها را N در نظر بگیریم، تابع هزینه Mean Squared Error به صورت زیر تعریف می‌شود:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

در این فرمول:

- N تعداد نمونه‌ها است.
- y_i مقدار واقعی برای نمونه i است.
- $\hat{y}_i$ مقدار پیش‌بینی شده توسط مدل برای نمونه i است.

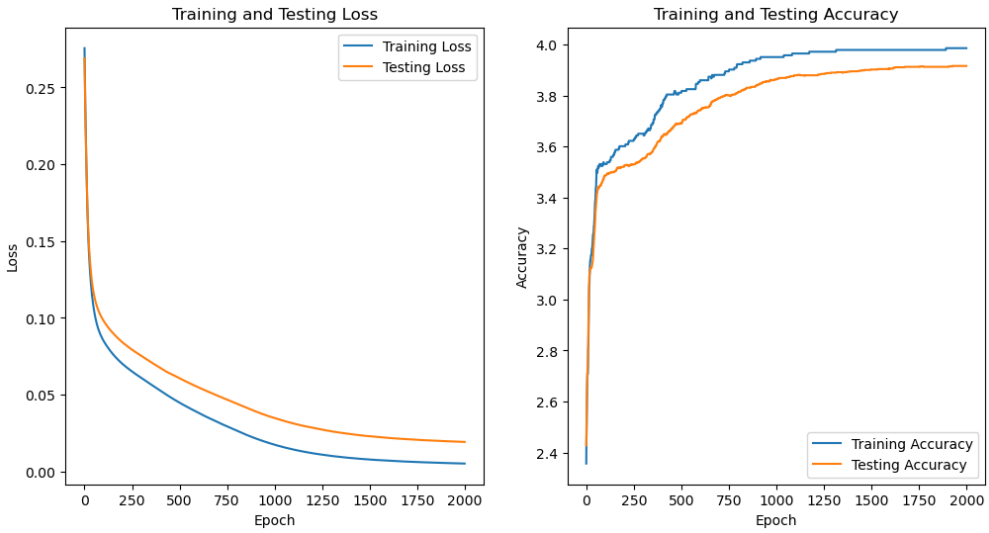
تحلیل تابع هزینه

- هدف: هدف از این تابع هزینه، مینیمم کردن میانگین مربعات اختلافات بین مقادیر واقعی و مقادیر پیش‌بینی شده است. وقتی مدل به درستی آموزش ببیند، اختلاف بین y_i و $\hat{y}_i$ برای همه نمونه‌ها به حداقل می‌رسد.
- تفسیر مربعات اختلافات: وجود توان دوم در فرمول باعث می‌شود که مدل برای اشتباهات بزرگ‌تر جریمه بیشتری دریافت کند. به این معنی که اگر مدل یک نمونه را با خطای بزرگ پیش‌بینی کند، مقدار تابع هزینه به طور قابل توجهی افزایش می‌یابد.
- تاثیر نرمال‌سازی: تقسیم مجموع توسط تعداد نمونه‌ها N باعث می‌شود که مقدار تابع هزینه به ازای هر نمونه به دست آید، که این امر به تحلیل بهتر عملکرد مدل کمک می‌کند.
- پیوستگی و مشتق‌پذیری: تابع MSE پیوسته و مشتق‌پذیر است، که این ویژگی‌ها آن را برای استفاده در الگوریتم‌های بهینه‌سازی مبتنی بر گرادیان (مانند گرادیان نزولی) مناسب می‌کند.

به طور کلی، تابع هزینه Mean Squared Error (MSE) به دلیل سادگی و کارایی در محاسبه خطاهای رگرسیون و توانایی در جریمه کردن اشتباهات بزرگ، یکی از توابع هزینه موثر و پرکاربرد در مسائل رگرسیون است.

با مقایسه نتایج جدید با نتایج قبلی، می‌توان به نکات زیر اشاره کرد:

- تابع هزینه: در هر دو نمودار، تابع هزینه برای داده‌های آموزش و تست به طور یکنواخت کاهش می‌یابد، اما مقدار تابع هزینه در نتایج جدید نسبت به نتایج قبلی کمتر است. این نشان‌دهنده بهبود مدل در کاهش خطاهای آموزشی و تستی است.
- دقت: دقت مدل در هر دو نمودار به طور یکنواخت افزایش می‌یابد. در نمودارهای جدید، دقت نهایی برای داده‌های آموزش و تست به مقادیر بالاتری نسبت به نتایج قبلی رسیده است، که نشان‌دهنده بهبود عملکرد مدل در تشخیص نمونه‌ها است.



- پایداری مدل: در نتایج جدید، دقت و تابع هزینه به طور یکنواخت‌تر و با نوسانات کمتری بهبود یافته‌اند، که نشان‌دهنده پایداری بیشتر مدل در فرآیند آموزش است.

به طور کلی، نتایج جدید نشان‌دهنده بهبود قابل توجه مدل در هر دو معیار تابع هزینه و دقت هستند و مدل در تشخیص نمونه‌ها عملکرد بهتری دارد. نتایج مانند قبل است و تغییری نکرده است.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	22
1	0.94	0.89	0.91	18
2	0.95	1.00	0.97	19
3	0.95	0.95	0.95	21
accuracy			0.96	80
macro avg	0.96	0.96	0.96	80
weighted avg	0.96	0.96	0.96	80

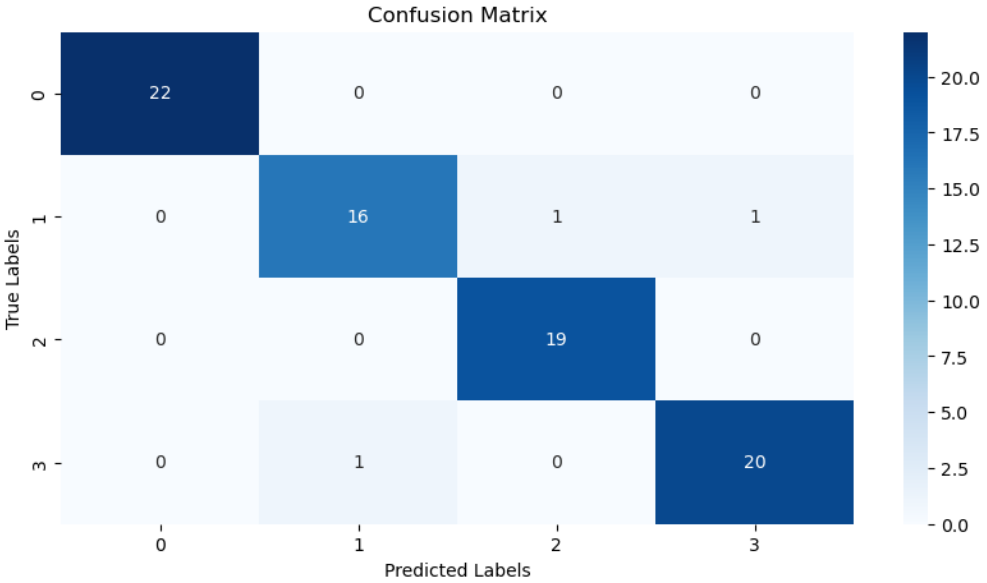
نتایج مانند قبل است و تغییری نکرده است.

تحلیل تابع هزینه Binary Cross-Entropy (BCE)

تابع هزینه Binary Cross-Entropy (BCE) یکی از پرکاربردترین توابع هزینه در مسائل طبقه‌بندی باینری است. این تابع به منظور محاسبه تفاوت بین توزیع پیش‌بینی شده توسط مدل و توزیع واقعی برچسب‌ها استفاده می‌شود. در ادامه، فرمول ریاضی و تحلیل این تابع هزینه ارائه می‌شود.

فرمول ریاضی

فرض کنید که y بردار واقعی برچسب‌ها و $\hat{y}$ بردار پیش‌بینی شده توسط مدل باشد. اگر تعداد نمونه‌ها را N در نظر بگیریم، تابع هزینه Binary Cross-Entropy به صورت زیر تعریف می‌شود:



$$BCE = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

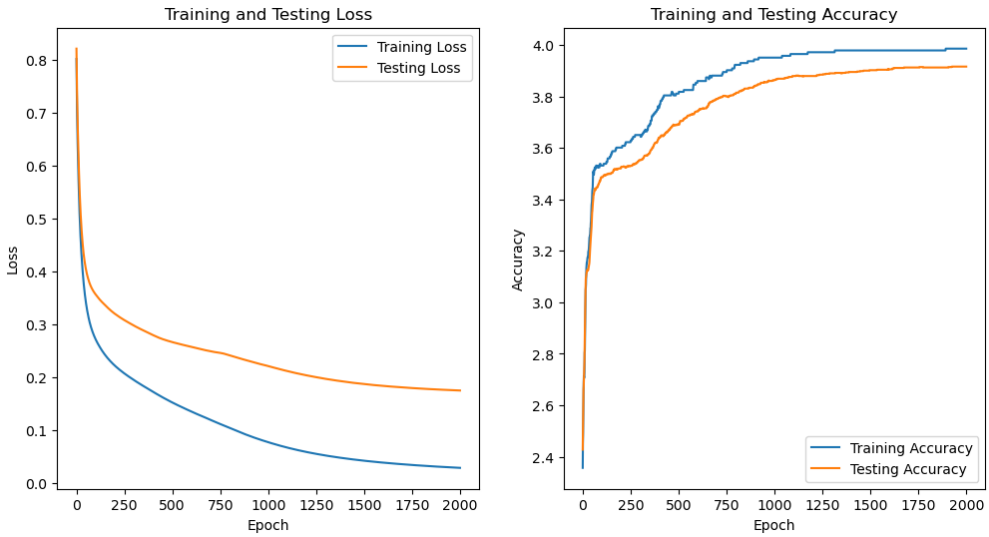
در این فرمول:

- N تعداد نمونه‌ها است.
- y_i مقدار واقعی برچسب برای نمونه i است که معمولاً مقدار آن یا ۰ یا ۱ است.
- $\hat{y}_i$ احتمال پیش‌بینی شده برای کلاس مثبت توسط مدل برای نمونه i است.

تحلیل تابع هزینه

- هدف: هدف از این تابع هزینه، مینیمم کردن تفاوت بین توزیع واقعی و توزیع پیش‌بینی شده است. وقتی مدل به درستی آموزش ببیند، احتمال پیش‌بینی شده $\hat{y}_i$ برای کلاس صحیح به ۱ نزدیک می‌شود و سایر احتمالات به ۰ میل می‌کنند.
- تفسیر لگاریتم: وجود تابع لگاریتم در فرمول باعث می‌شود که مدل برای اشتباهات بزرگ‌تر جریمه بیشتری دریافت کند. به این معنی که اگر مدل یک نمونه را با اطمینان بالا به اشتباه پیش‌بینی کند، مقدار تابع هزینه به طور قابل توجهی افزایش می‌یابد.
- تاثیر نرمال‌سازی: تقسیم مجموع توسط تعداد نمونه‌ها N باعث می‌شود که مقدار تابع هزینه به ازای هر نمونه به دست آید، که این امر به تحلیل بهتر عملکرد مدل کمک می‌کند.
- تقارن: تابع BCE تقارن دارد؛ به این معنی که اگر مدل احتمال را به درستی پیش‌بینی کند، مقدار هزینه نزدیک به صفر خواهد بود و اگر اشتباه پیش‌بینی کند، مقدار هزینه بزرگ خواهد بود.

به طور کلی، تابع هزینه Binary Cross-Entropy (BCE) به دلیل توانایی در به دست آوردن تفاوت‌های کوچک بین توزیع‌های واقعی و پیش‌بینی شده و جریمه کردن اشتباهات بزرگ، یکی از توابع هزینه موثر و پرکاربرد در مسائل طبقه‌بندی باینری است.
با مقایسه نتایج جدید با نتایج قبلی، می‌توان به نکات زیر اشاره کرد:

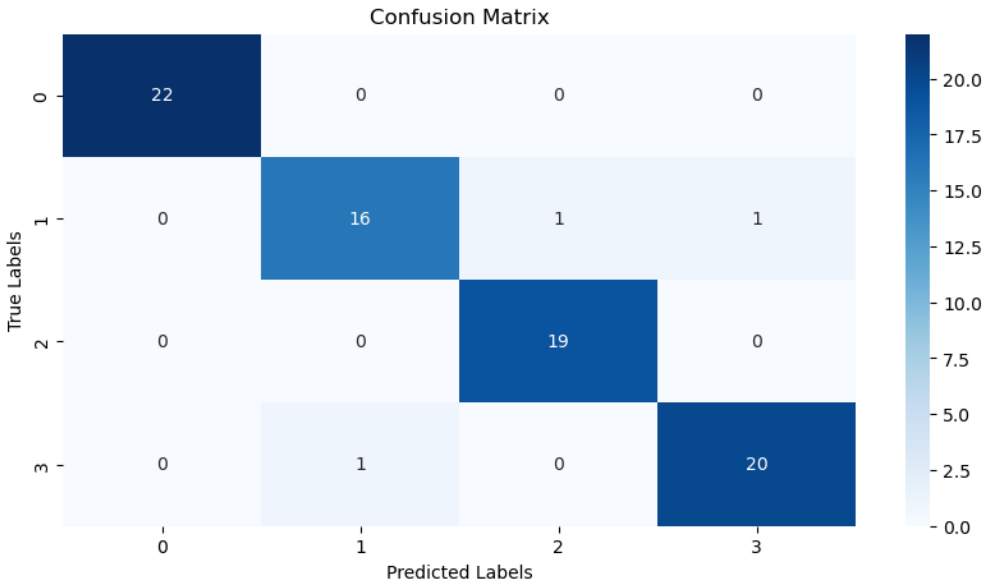


- تابع هزینه: در هر سه نمودار، تابع هزینه برای داده‌های آموزش و تست به طور یکنواخت کاهش می‌یابد. در نمودارهای جدید، مقدار تابع هزینه به مراتب کمتر از نمودارهای قبلی است. این نشان‌دهنده بهبود مدل در کاهش خطاهای آموزشی و تستی است.
- دقت: دقت مدل در هر سه نمودار به طور یکنواخت افزایش می‌یابد. در نمودارهای جدید، دقت نهایی برای داده‌های آموزش و تست به مقادیر بالاتری نسبت به نتایج قبلی رسیده است، که نشان‌دهنده بهبود عملکرد مدل در تشخیص نمونه‌ها است.
- پایداری مدل: در نتایج جدید، دقت و تابع هزینه به طور یکنواخت‌تر و با نوسانات کمتری بهبود یافته‌اند، که نشان‌دهنده پایداری بیشتر مدل در فرآیند آموزش است.
- مقدار دقت و خطا: در نمودارهای جدید، دقت آموزش و تست به مقدار بالاتری رسیده و تابع هزینه نیز کاهش بیشتری یافته است. این موضوع نشان‌دهنده بهبود عملکرد کلی مدل و کاهش خطاها است.

به طور کلی، نتایج جدید نشان‌دهنده بهبود قابل توجه مدل در هر دو معیار تابع هزینه و دقت هستند و مدل در تشخیص نمونه‌ها عملکرد بهتری دارد. نتایج مانند قبل است و تغییری نکرده است.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	22
1	0.94	0.89	0.91	18
2	0.95	1.00	0.97	19
3	0.95	0.95	0.95	21
accuracy			0.96	80
macro avg	0.96	0.96	0.96	80
weighted avg	0.96	0.96	0.96	80

نتایج مانند قبل است و تغییری نکرده است.



نتایجه گیری:

شباهت‌ها

- هدف مشترک: هدف هر سه تابع هزینه، مینیمم کردن اختلاف بین مقادیر واقعی و مقادیر پیش‌بینی شده توسط مدل است.
- استفاده از لگاریتم: هر دو تابع هزینه Categorical Cross-Entropy و Binary Cross-Entropy از تابع لگاریتم برای محاسبه هزینه استفاده می‌کنند، که باعث می‌شود مدل برای اشتباهات بزرگ‌تر جریمه بیشتری دریافت کند.
- نرمال‌سازی: هر سه تابع هزینه با تقسیم مجموع اختلافات توسط تعداد نمونه‌ها، مقدار هزینه به ازای هر نمونه را محاسبه می‌کنند که این امر به تحلیل بهتر عملکرد مدل کمک می‌کند.
- مشتق‌پذیری: هر سه تابع هزینه پیوسته و مشتق‌پذیر هستند، که این ویژگی‌ها آن‌ها را برای استفاده در الگوریتم‌های بهینه‌سازی مبتنی بر گرادیان مناسب می‌کند.

تفاوت‌ها

- نوع مسئله:
  - MSE: بیشتر در مسائل رگرسیون استفاده می‌شود که هدف پیش‌بینی مقادیر پیوسته است.
  - Categorical Cross-Entropy: برای مسائل طبقه‌بندی چندکلاسه استفاده می‌شود.
  - Binary Cross-Entropy: برای مسائل طبقه‌بندی باینری استفاده می‌شود.
- فرمول‌بندی:
  - MSE:

    بر اساس مربعات اختلافات بین مقادیر واقعی و پیش‌بینی شده است.

- Categorical Cross-Entropy و Binary Cross-Entropy:

بر اساس لگاریتم احتمال‌های پیش‌بینی شده و مقادیر واقعی برچسب‌ها هستند.

- جریمه اشتباهات:

- MSE: اشتباهات بزرگ‌تر را با توان دوم جریمه می‌کند.
- Categorical Cross-Entropy و Binary Cross-Entropy: اشتباهات بزرگ‌تر را با لگاریتم جریمه می‌کنند.

میزان اثرگذاری

در عمل، نتایج مدل‌ها با استفاده از هر یک از این توابع هزینه معمولاً تفاوت زیادی ندارند، به شرطی که مسئله به درستی تعریف شده باشد. انتخاب تابع هزینه مناسب بر اساس نوع مسئله (رگرسیون یا طبقه‌بندی) و تعداد کلاس‌ها (باینری یا چندکلاسه) صورت می‌گیرد.

- MSE: برای مسائل رگرسیون بسیار مناسب است و به دلیل سادگی محاسبات، یکی از اولین انتخاب‌ها در مسائل رگرسیون است.
- Cross-Entropy Categorical: برای مسائل طبقه‌بندی چندکلاسه ایده‌آل است و توانایی خوبی در تشخیص تفاوت‌های کوچک بین توزیع‌های واقعی و پیش‌بینی شده دارد.
- Cross-Entropy Binary: برای مسائل طبقه‌بندی باینری مناسب است و به خوبی می‌تواند تفاوت بین کلاس‌های مثبت و منفی را تشخیص دهد.

به طور کلی، هر یک از این توابع هزینه دارای مزایا و معایب خاص خود هستند و انتخاب صحیح آن‌ها بر اساس نوع مسئله و داده‌ها می‌تواند به بهبود عملکرد مدل کمک کند.

۴.۲ بخش چهارم سوال دوم

K-Fold Cross-Validation:

یک روش رایج برای ارزیابی مدل‌های یادگیری ماشین است. در این روش، مجموعه داده به طور تصادفی به K زیرمجموعه با اندازه مساوی تقسیم می‌شود. سپس، از هر زیرمجموعه به عنوان مجموعه تست در یک نوبت و از K−۱ زیرمجموعه باقی‌مانده به عنوان مجموعه آموزش استفاده می‌شود. این فرآیند K بار تکرار می‌شود، به طوری که هر زیرمجموعه دقیقاً یک بار به عنوان مجموعه تست استفاده می‌شود. میانگین عملکرد مدل در تمام K بار، تخمین عملکرد مدل در داده‌های جدید را ارائه می‌دهد.

مزایای K-Fold Cross-Validation:

- ساده برای اجرا: پیاده‌سازی K-Fold Cross-Validation نسبتاً ساده است و کتابخانه‌های زیادی برای انجام این کار در زبان‌های برنامه‌نویسی مختلف موجود است.
- کارآمد: K-Fold Cross-Validation یک روش کارآمد برای ارزیابی مدل است، زیرا از تمام داده‌ها در هر بار تکرار استفاده می‌شود.
- قابل اعتماد: K-Fold Cross-Validation تخمین قابل اعتمادی از عملکرد مدل در داده‌های جدید ارائه می‌دهد.

Stratified K-Fold Cross-Validation:

نوعی از K-Fold Cross-Validation است که برای مجموعه داده‌های غیرمتعادل استفاده می‌شود. در این روش، قبل از تقسیم مجموعه داده به زیرمجموعه‌ها، از نمونه‌گیری طبقه‌بندی‌شده برای اطمینان از اینکه هر زیرمجموعه نماینده توزیع کلاس در کل مجموعه داده است، استفاده می‌شود. به عبارت دیگر، هر زیرمجموعه باید همان نسبت کلاس‌ها را به عنوان کل مجموعه داده داشته باشد.

- برای مجموعه داده‌های نامتعادل مناسب است: Stratified K-Fold Cross-Validation از سوگیری مدل به سمت کلاس اکثریت در مجموعه داده‌های نامتعادل جلوگیری می‌کند.
- تخمین دقیق‌تر عملکرد مدل: Stratified K-Fold Cross-Validation می‌تواند تخمین دقیق‌تری از عملکرد مدل در داده‌های جدید ارائه دهد، به خصوص برای مجموعه داده‌های نامتعادل.

در نهایت، انتخاب بین K-Fold Cross-Validation و Stratified K-Fold Cross-Validation به نوع مجموعه داده شما بستگی دارد. اگر با یک مجموعه داده متعادل کار می‌کنید، K-Fold Cross-Validation یک انتخاب خوب است. با این حال، اگر با یک مجموعه داده نامتعادل کار می‌کنید، Stratified K-Fold Cross-Validation انتخاب بهتری است. که با توجه به دیتاست ما دیتاست متعادل است و انتخاب ما K-Fold Cross-Validation است تابع kfold_indices: این تابع وظیفه تقسیم داده‌ها به K دسته (Fold) برای اعتبارسنجی متقابل را بر عهده دارد:

- محاسبه اندازه Fold: تعداد کل نمونه‌ها (len(data)) را بر تعداد هاFold (k) تقسیم می‌کند تا تعداد نمونه‌های هر Fold بدست آید (fold_size).
- ایجاد آرایه شاخص‌ها: آرایه‌ای به نام indices ایجاد می‌کند که شامل شاخص‌هایی برای تمام نمونه‌های داده (data) است.
- ایجاد لیست هاFold: لیستی خالی به نام folds برای ذخیره شاخص‌های هاFold ایجاد می‌کند.
- ایجاد هاFold:
- بازگشت لیست هاFold: لیست folds حاوی شاخص‌های هاFold برای هر Fold (شاخص‌های نمونه‌های آموزشی و آزمایشی) را برمی‌گرداند.

```python
def kfold_indices(data, k):
    fold_size = len(data) // k
    indices = np.arange(len(data))
    folds = []
    for i in range(k):
        test_indices = indices[i * fold_size: (i + 1) * fold_size]
        train_indices = np.concatenate([indices[:i * fold_size], indices[(i + 1) * fold_size:]])
        folds.append((train_indices, test_indices))
    return folds
k = 3

fold_indices = kfold_indices(X_train, k)
scores = []
for train_indices, test_indices in fold_indices:
    X_train_1, y_train_1 = X_train[train_indices], train_labels_onehot[train_indices]
    X_test_1, y_test_1 = X_test[test_indices], validate_labels_onehot[test_indices]
    mlp = MLP(hidden_layer_sizes=hidden_layer_sizes, hidden_activation='relu', output_size=output_size, output_ac
    mlp.fit(X_train_1, y_train_1, X_test_1, y_test_1)
    from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
    y_hat = mlp.predict(X_valid)
    y_hat = np.argmax(y_hat, axis=1)
    model_report = classification_report(y_val, y_hat)
    print(model_report)
    fold_score = accuracy_score(y_val, y_hat)

    scores.append(fold_score)
mean_accuracy = np.mean(scores)
print("K-Fold Cross-Validation Scores:", scores)
print("Mean Accuracy:", mean_accuracy)
```

اعتبارسنجی متقابل K-Fold:

در این بخش، کد از تابع kfold_indices برای تقسیم داده‌ها به هاFold و سپس ارزیابی مدل با استفاده از هر Fold به صورت جداگانه استفاده می‌کند:

- تنظیم تعداد هاFold: تعداد هاFold را در متغیر k (معمولاً ۳ یا ۱۰ در عمل) تعیین می‌کند.
- ایجاد هاFold: از تابع kfold_indices برای دریافت شاخص‌های هاFold برای داده‌های آموزشی (X_train) و تعداد هاFold (k) استفاده می‌کند و نتیجه را در fold_indices ذخیره می‌کند.
- ایجاد لیست نمرات:: لیستی خالی به نام scores برای ذخیره نمرات دقت (Accuracy) هر Fold ایجاد می‌کند.
- حلقه هاFold:
  - آموزش مدل
  - پیش‌بینی و ارزیابی
  - محاسبه دقت

```
K-Fold Cross-Validation Scores: [0.975, 0.9625, 0.975]
Mean Accuracy: 0.9708333333333333
```

همانگونه که مشهود است فو.کوس را از فولد دوم برداشته است و با ۳ فولد تقسیم کرده است. یعنی با یک فول اپتیمایز نمیشود و همین امر بسیار از overfitting جلوگیری میکند.

از بیش‌برازش (overfitting) مدل به داده‌های آموزشی خود جلوگیری میکند. اعتبارسنجی با نگاشت K-Fold به شما امکان می دهد تا کنترل کامل روی فرآیند داشته باشید و درک عمیق تری از نحوه عملکرد آن به دست آورید. اعتبارسنجی با نگاشت K-Fold یک تکنیک ارزشمند برای

تخمین اینکه مدل شما چقدر به داده های دیده نشده تعمیم می دهد، است. چه در حال توسعه یک مدل جدید باشید و چه یک مدل موجود را تنظیم کنید، گنجاندن اعتبارسنجی با نگاشت K-Fold در گردش کار شما می تواند به شما در ساخت راه‌حل‌های یادگیری ماشین قابل اعتماد و قوی‌تر کمک کند.

۳ سوال سوم

۱.۳ بخش اول سوال سوم

```python
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(df[['Age','Sex','BP','Cholesterol','Na_to_K']], df['Drug'], test_size=0.2, random_state=44)
```

شکل ۱: کد مربوط به تقسیم دیتاها

تابع train_test_split یک ابزار در sklearn.model_selection است که برای تقسیم آرایه‌های داده به دو زیرمجموعه: داده‌های آموزشی و تست استفاده می‌شود. این تابع به طور گسترده‌ای در یادگیری ماشین برای اعتبارسنجی مدل‌ها کاربرد دارد .

- arrays*: دنباله‌هایی از شاخص‌ها یا داده‌ها. این‌ها می‌توانند لیست‌ها، آرایه‌های نامپای، فریم‌های داده پانداس و غیره باشند.
- test_size: نسبت داده‌هایی که باید در تقسیم تست گنجانده شوند (عدد اعشاری برای کسر، عدد صحیح برای تعداد نمونه‌های مطلق).
- train_size: نسبت داده‌هایی که باید در تقسیم آموزشی گنجانده شوند.
- random_state: برای تضمین تکرارپذیری تقسیم‌ها.
- shuffle: آیا داده‌ها قبل از تقسیم باید تصادفی شوند. اگر shuffle=False باشد، در این صورت stratify باید None باشد.
- stratify: اگر None نباشد، داده‌ها به صورت طبقه‌بندی شده تقسیم می‌شوند، با استفاده از این به عنوان برچسب‌های کلاس.

روش های دیگر:

```python
from sklearn.model_selection import StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=44)
for train_index, test_index in sss.split(df[['Age','Sex','BP','Cholesterol','Na_to_K']], df['Drug']):
    x_train, x_test = df[['Age','Sex','BP','Cholesterol','Na_to_K']].iloc[train_index], df[['Age','Sex','BP','Cholesterol','Na_to_K']].iloc[test_index]
    y_train, y_test = df['Drug'].iloc[train_index], df['Drug'].iloc[test_index]
```

شکل ۲: کد مربوط به تقسیم دیتاها به کمک stratisfiedsplit

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

پارامترهای StratifiedShuffleSplit به شرح زیر است:

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

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

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

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

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

```
df = df.sample(frac=1, random_state=44)
split_idx = int(0.8 * len(df))
x_train = df[['Age','Sex','BP','Cholesterol','Na_to_K']][:split_idx]
x_test = df[['Age','Sex','BP','Cholesterol','Na_to_K']][split_idx:]
y_train = df['Drug'][:split_idx]
y_test = df['Drug'][split_idx:]
```

شکل ۳: کد مربوط به تقسیم دیتاها به صورت دستی

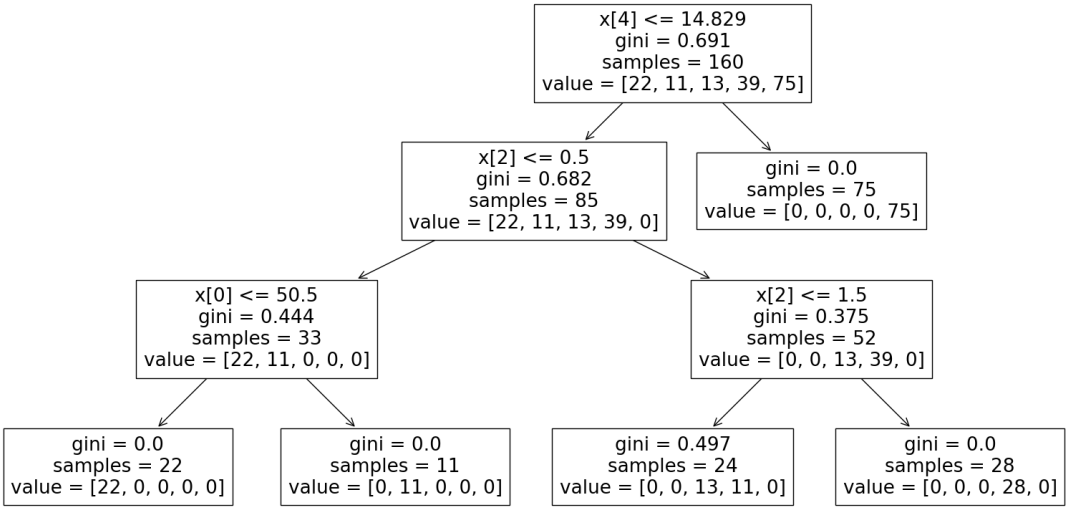
بصورت دستی هم می‌توان داده‌ها را تقسیم نمود. برای این کار می‌توان از ایندکس‌های دیتافریم یا داده‌ها به صورت آرایه استفاده کرد و سپس به کمک ماژول‌های موجود مانند shuffle یا sample ترتیب آن‌ها را تغییر داد و داده‌ها را تقسیم کرد. این روش امکان کنترل دقیق‌تر بر نحوه تقسیم داده‌ها را فراهم می‌کند و می‌تواند در برخی موارد کاربردی باشد.

برای مثال، به صورت زیر می‌توان داده‌ها را دستی تقسیم کرد:

ابتدا داده‌ها را به صورت آرایه یا دیتافریم دریافت کنید. از تابع shuffle یا sample برای تغییر ترتیب داده‌ها استفاده کنید. داده‌ها را به نسبت‌های مورد نظر به مجموعه‌های آموزشی و تست تقسیم کنید.

مدل سازی درخت تصمیم گیری:

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



شکل ۴: درخت ایجاد شده با توجه به ویژگی ها

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

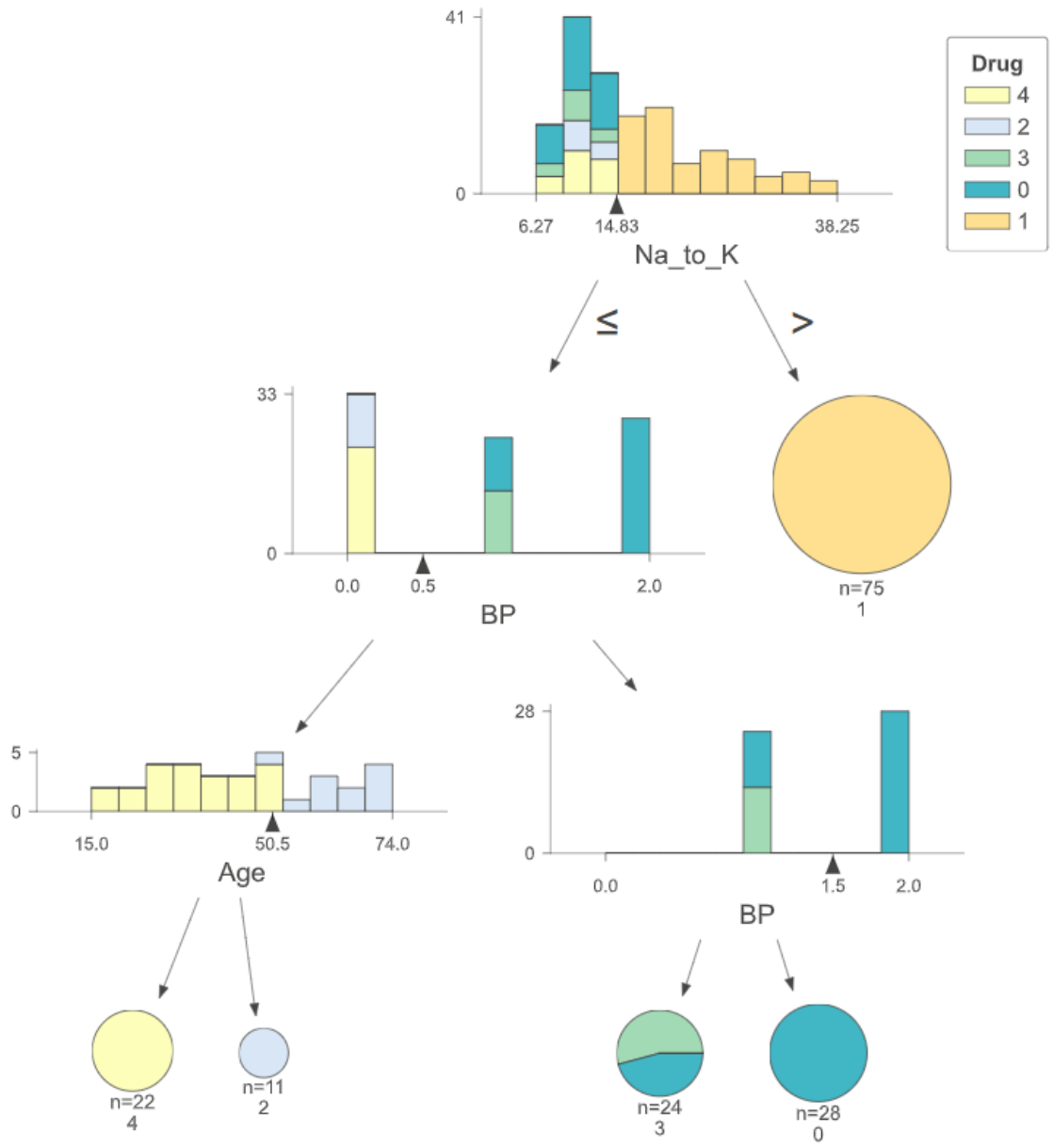
نود ریشه (Root Node)

- شرط: $x[4] \leq 14.829$
- $gini$: 0.691
- تعداد نمونه‌ها: ۱۶۰
- توزیع کلاس‌ها: $[22, 11, 13, 39, 75]$

این نود نمونه‌ها را به دو گروه تقسیم می‌کند: آنهایی که مقدار ویژگی چهارم آنها کمتر یا مساوی 14.829 است و آنهایی که بیشتر است. impurity در این نود نسبتاً بالاست که نشان‌دهنده تنوع بالای کلاس‌ها در این سطح است.

نود چپ فرزند اول

- شرط: $x[2] \leq 0.5$
- $gini$: 0.682
- تعداد نمونه‌ها: ۸۵
- توزیع کلاس‌ها: $[22, 11, 13, 39, 0]$



شکل ۵: گراف و بررسی در هر نود درخت

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

نود راست فرزند اول

- $gini$: 0.0
- تعداد نمونه‌ها: ۷۵
- توزیع کلاس‌ها: $[0, 0, 0, 0, 75]$

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

نود چپ فرزند دوم

- شرط: $x[0] \leq 50.5$
- $gini$: 0.444
- تعداد نمونه‌ها: ۳۳
- توزیع کلاس‌ها: $[22, 11, 0, 0, 0]$

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

نود راست فرزند دوم

- شرط: $x[2] \leq 1.5$
- $gini$: 0.375
- تعداد نمونه‌ها: ۵۲
- توزیع کلاس‌ها: $[0, 0, 13, 39, 0]$

این نود نمونه‌هایی را که مقدار ویژگی دوم آنها بیشتر از 0.5 است، به دو گروه تقسیم می‌کند: آنهایی که مقدار ویژگی دوم آنها کمتر یا مساوی 1.5 است و آنهایی که بیشتر است.

نود چپ فرزند سوم

- $gini$: 0.0
- تعداد نمونه‌ها: ۲۲
- توزیع کلاس‌ها: $[22, 0, 0, 0, 0]$

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

نود راست فرزند سوم

- $gini$: 0.0
- تعداد نمونه‌ها: ۱۱
- توزیع کلاس‌ها: $[0, 11, 0, 0, 0]$

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

نود چپ فرزند چهارم

- $gini$: 0.497

- تعداد نمونه‌ها: ۲۴
- توزیع کلاس‌ها: $[0, 0, 13, 11, 0]$

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

نود راست فرزند چهارم

- $gini : 0.0$
- تعداد نمونه‌ها: ۲۸
- توزیع کلاس‌ها: $[0, 0, 0, 28, 0]$

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

درخت تصمیم گیری بصورت دستی:

```python
class Node:
    """Class to represent a node in the decision tree."""
    def __init__(self, feature_index=None, threshold=None, left=None, right=None, value=None):
        self.feature_index = feature_index
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
```

شکل ۶: کد معرف نود

۱. تعریف کلاس نود ابتدا یک کلاس به نام Node تعریف می‌شود که نماینده یک نود در درخت تصمیم‌گیری است. این کلاس شامل ویژگی‌هایی برای ذخیره شاخص ویژگی، آستانه، نودهای فرزند چپ و راست، و همچنین مقدار نهایی برای نود برگ است.

۲. تابع ساخت درخت (build_tree) این تابع به صورت بازگشتی درخت تصمیم‌گیری را می‌سازد. در هر مرحله از ساخت درخت:

- اگر داده‌ای وجود نداشته باشد، نود برگ برگردانده می‌شود.
- اگر تمامی برچسب‌ها مشابه باشند یا به حداکثر عمق برسیم، نود برگ با بیشترین برچسب مشترک برگردانده می‌شود.
- بهترین ویژگی و آستانه برای تقسیم داده‌ها پیدا می‌شود.
- داده‌ها به دو بخش چپ و راست تقسیم می‌شوند و فرآیند ساخت درخت به صورت بازگشتی برای هر بخش ادامه می‌یابد.

۳. تابع بیشترین برچسب مشترک (most_common_label) این تابع برچسبی را که بیشترین فراوانی را در داده‌ها دارد، برمی‌گرداند. از این تابع برای تعیین مقدار نهایی نودهای برگ استفاده می‌شود.

۴. تابع بهترین تقسیم‌بندی (best_split) این تابع بهترین ویژگی و آستانه را برای تقسیم‌بندی داده‌ها با کمترین ناخالصی جینی پیدا می‌کند. برای هر ویژگی و هر مقدار آستانه، ناخالصی جینی محاسبه می‌شود و بهترین تقسیم‌بندی انتخاب می‌شود.

۵. تابع محاسبه جینی (calculate_gini) این تابع ناخالصی جینی را برای یک تقسیم‌بندی خاص محاسبه می‌کند. ابتدا داده‌ها به دو بخش چپ و راست تقسیم می‌شوند و سپس ناخالصی جینی برای هر بخش محاسبه و میانگین وزنی آنها برگردانده می‌شود.

```python
def build_tree(data, labels, depth=0, max_depth=3):
    """Builds the decision tree recursively."""
    num_samples, num_features = data.shape
    # If no data, return a leaf
    if num_samples == 0:
        return None
    # If all labels are the same or max depth reached, return a leaf
    if len(set(labels.tolist())) == 1 or depth == max_depth:
        return Node(value=most_common_label(labels))

    # Find the best split
    best_feature, best_threshold = best_split(data, labels)

    # If no effective split, return a leaf
    if best_feature is None:
        return Node(value=most_common_label(labels))

    # Partition data
    left_idxs = data[:, best_feature] < best_threshold
    right_idxs = data[:, best_feature] >= best_threshold

    left = build_tree(data[left_idxs], labels[left_idxs], depth+1, max_depth)
    right = build_tree(data[right_idxs], labels[right_idxs], depth+1, max_depth)

    # Return decision node
    return Node(best_feature, best_threshold, left, right)
```

شکل ۷: کد ساخت درخت

۶. تابع پیش‌بینی (predict) این تابع برای یک نمونه خاص با استفاده از درخت تصمیم‌گیری، برچسب را پیش‌بینی می‌کند. اگر نود برگ باشد، مقدار آن برگردانده می‌شود؛ در غیر این صورت، نمونه بر اساس ویژگی و آستانه مربوطه به نود فرزند چپ یا راست هدایت می‌شود.

۷. تابع دقت (accuracy_score) این تابع دقت مدل را با مقایسه برچسب‌های واقعی و پیش‌بینی شده محاسبه می‌کند. تعداد پیش‌بینی‌های صحیح تقسیم بر تعداد کل نمونه‌ها، دقت مدل را نشان می‌دهد.

۸. بارگذاری و پیش‌پردازش داده‌ها داده‌ها از فایل drug200.csv بارگذاری می‌شوند و سپس ویژگی‌های غیرعددی (مانند فشار خون، کلسترول، و جنسیت) با استفاده از LabelEncoder به مقادیر عددی تبدیل می‌شوند. سپس داده‌ها به مجموعه‌های آموزش و تست تقسیم می‌شوند.

۹. ساخت درخت و پیش‌بینی در نهایت، درخت تصمیم‌گیری با استفاده از داده‌های آموزشی ساخته می‌شود و برچسب‌های نمونه‌های تست با استفاده از درخت پیش‌بینی می‌شوند. دقت مدل نیز محاسبه و چاپ می‌شود.

گراف ایجاد شده برای تست درخت در شکل آورده شده است. و دقت این مدل برابر است با 0.825.

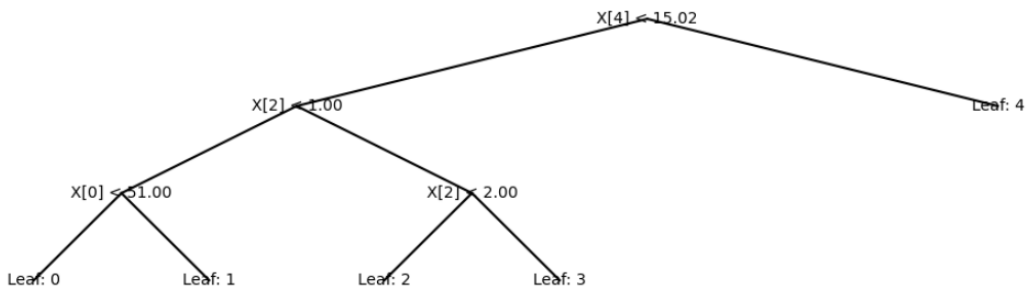
```python
def best_split(data, labels):
    """Finds the best feature and threshold to split on."""
    num_samples, num_features = data.shape
    best_gini = 1.0
    best_feature, best_threshold = None, None

    for feature_index in range(num_features):
        thresholds = set(data[:, feature_index])
        for threshold in thresholds:
            gini = calculate_gini(data, labels, feature_index, threshold)
            if gini < best_gini:
                best_gini, best_feature, best_threshold = gini, feature_index, threshold

    return best_feature, best_threshold

def calculate_gini(data, labels, feature_index, threshold):
    """Calculates the Gini impurity for a split."""
    left_labels = labels[data[:, feature_index] < threshold]
    right_labels = labels[data[:, feature_index] >= threshold]
    left_gini = 1.0 - sum([(left_labels == v).mean()**2 for v in set(left_labels.tolist())])
    right_gini = 1.0 - sum([(right_labels == v).mean()**2 for v in set(right_labels.tolist())])
    # Weighted average
    left_weight = len(left_labels) / len(labels)
    right_weight = len(right_labels) / len(labels)
    return left_gini * left_weight + right_gini * right_weight
```

شکل ۸: کد محاسبه کردن معیار وGini نود های تصمیم گیرنده برای جداسازی



شکل ۹: گراف ایجاد شده به کمک الگوریتم دست نویس درخت تصنمیم گیرنده

۲.۳ بخش دوم سوال سوم

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

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

- **False Negatives (FN)**: تعداد مواردی که نادرست به عنوان منفی طبقه‌بندی شده‌اند.

:Accuracy

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

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (۴)$$

:Recall

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

$$Recall = \frac{TP}{TP + FN} \quad (۵)$$

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

$$Recall(macro) = \frac{1}{C}\sum_{c=1}^{C}\frac{TP_c}{TP_c + FN_c} \quad (۶)$$

که C تعداد کلاس‌ها می‌باشد.

:Precision

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

$$Precision = \frac{TP}{TP + FP} \quad (۷)$$

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

$$Precision(macro) = \frac{1}{C}\sum_{c=1}^{C}\frac{TP_c}{TP_c + FP_c} \quad (۸)$$

که C تعداد کلاس ها می‌باشد.

زیان همینگ (Hamming Loss):

$$HammingLoss = \frac{1}{n}\sum_{i=1}^{n}\frac{Numberofincorrectlabels}{Numberoflabels} \quad (۹)$$

که n تعداد نمونه ها می‌باشد.

:F1 Score

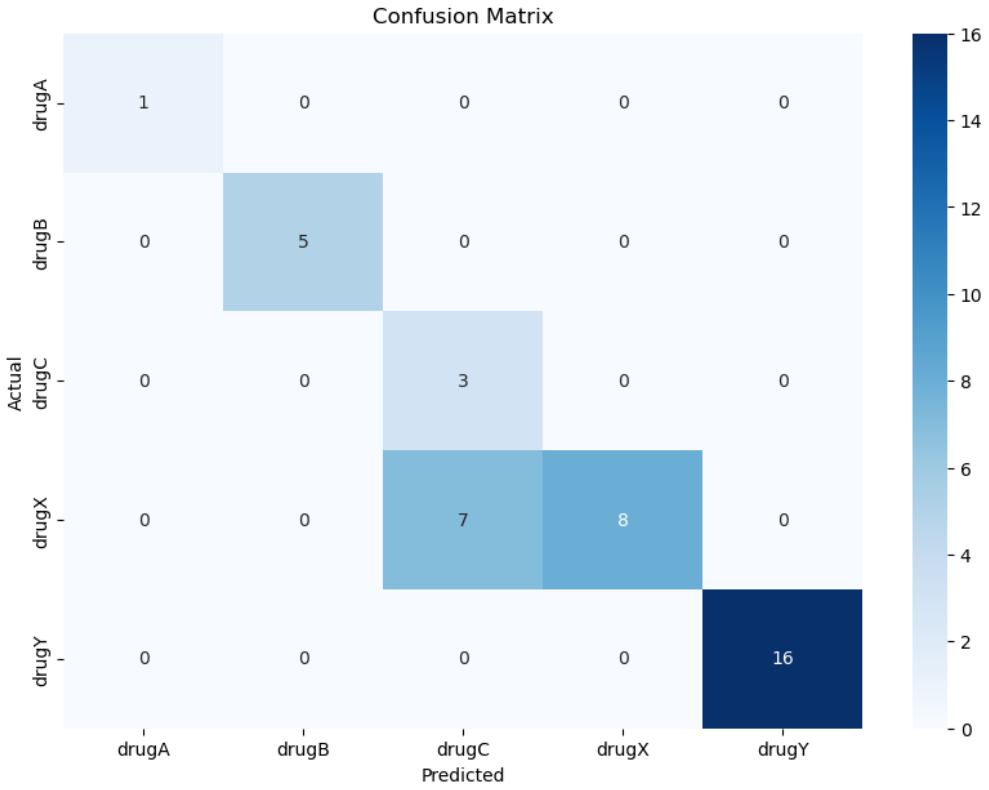
امتیاز F۱ میانگین هماهنگ دقت و بازیابی است. فرمول آن به صورت زیر است:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (۱۰)$$

در حالت میانگین‌گیری ماکرو، امتیاز F۱ برای هر کلاس محاسبه می‌شود و سپس میانگین گرفته می‌شود.

$$F1Score(macro) = \frac{1}{C}\sum_{c=1}^{C} 2 \cdot \frac{Precision_c \cdot Recall_c}{Precision_c + Recall_c} \quad (۱۱)$$

با توجه به شکل مدل به خوبی توانسته است همه کلاس‌هارا به جز کلاس XوC تفکیک و کلاس‌بندی کند. دلیل این امر در گراف به خوبی نمایان است.



شکل ۱۰: ماتریس درهم ریختگی

```
Accuracy: 0.825
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         5
           2       0.30      1.00      0.46         3
           3       1.00      0.53      0.70        15
           4       1.00      1.00      1.00        16

    accuracy                           0.82        40
   macro avg       0.86      0.91      0.83        40
weighted avg       0.95      0.82      0.85        40
```

شکل ۱۱: گزارش دسته بندی

- کلاس ۰:

- دقت: 1.00
- بازخوانی: 1.00
- امتیاز :F۱ 1.00
- نعداد: 1

این کلاس تنها شامل یک نمونه است و مدل به درستی آن را طبقه‌بندی کرده است.

- کلاس ۱:
  - دقت: 1.00
  - بازخوانی: 1.00
  - امتیاز :F۱ 1.00
  - نعداد: 5

مدل تمامی نمونه‌های این کلاس را به درستی طبقه‌بندی کرده است.

- کلاس ۲:
  - دقت: 0.30
  - بازخوانی: 1.00
  - امتیاز :F۱ 0.46
  - نعداد: 3

مدل در طبقه‌بندی این کلاس عملکرد ضعیفی داشته است. دقت پایین و بازخوانی بالا نشان می‌دهد که مدل تعداد زیادی از نمونه‌های دیگر را به اشتباه به این کلاس نسبت داده است.

- کلاس ۳:
  - دقت: 1.00
  - بازخوانی: 0.53
  - امتیاز :F۱ 0.70
  - نعداد: 15

مدل نیمی از نمونه‌های این کلاس را به درستی طبقه‌بندی کرده و امتیاز F۱ نیز نشان‌دهنده عملکرد متوسط مدل در این کلاس است.

- کلاس ۴:
  - دقت: 1.00
  - بازخوانی: 1.00
  - امتیاز :F۱ 1.00
  - تعداد: 16

مدل تمامی نمونه‌های این کلاس را به درستی طبقه‌بندی کرده است.

```
from sklearn.metrics import accuracy_score, hamming_loss, precision_score, recall_score, f1_score

print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("Hamming Loss:", hamming_loss(y_test, y_pred))
print("Precision Score (macro):", precision_score(y_test, y_pred, average='macro'))
print("Recall Score (macro):", recall_score(y_test, y_pred, average='macro'))
print("F1 Score (macro):", f1_score(y_test, y_pred, average='macro'))
```

✓ 0.0s

```
Accuracy Score: 0.825
Hamming Loss: 0.175
Precision Score (macro): 0.86
Recall Score (macro): 0.9066666666666666
F1 Score (macro): 0.831438127090301
```

شکل ۱۲: معیار های کلی عملکرد مدل

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

اهمیت فراپارامترها در درختان تصمیم

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

پارامترهای هرس

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

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

برای بهتر کردن دقت میتوان:

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

برای جلوگیری از over-fitting و افزایش سرعت یادگیری:

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

runtime	mean_f_score	min_samples_leaf	max_depth
0.045877	0.927409	1	3
0.060882	0.927409	5	3
0.027932	0.927687	10	3
0.024928	0.809231	20	3
0.031109	0.987115	1	5
0.024935	0.987115	5	5
0.021935	0.933699	10	5
0.023132	0.809231	20	5
0.025392	0.987115	1	7
0.022519	0.987115	5	7
0.022526	0.933699	10	7
0.022954	0.809231	20	7
0.021942	0.987115	1	10
0.022027	0.987115	5	10
0.021939	0.933699	10	10
0.020064	0.809231	20	10

جدول ۱: نتایج با تغییر دادن هایپرپارامترها

تأثیر و مزایای هرس:

- کاهش پیچیدگی: هرس پیچیدگی مدل را کاهش می‌دهد که می‌تواند به کاهش بیش‌برازش کمک کند.
- بهبود یادگیری: با محدود کردن عمق و افزایش حداقل نمونه‌ها در هر برگ، درخت نسبت به نویز در داده‌ها کمتر حساس می‌شود.
- پیش‌بینی‌های سریع‌تر: درختان کوچکتر در انجام پیش‌بینی‌ها کارآمدتر هستند.

تنظیم این پارامترها شامل معامله‌ای بین توانایی مدل برای درک الگوی زیرین داده‌ها و تعمیم آن به داده‌های جدید است. استفاده از اعتبارسنجی متقابل برای یافتن تنظیمات بهینه این پارامترها معمولاً یک روش خوب است.

۳.۳ بخش سوم سوال سوم

جنگل تصادفی (Random Forest)

جنگل تصادفی مجموعه‌ای از درختان تصمیم (غالباً هزاران درخت) است که هر کدام بر روی زیرمجموعه‌ای از داده‌ها آموزش دیده‌اند. این روش به کاهش بیش‌برازش کمک کرده و تعمیم بهتری نسبت به یک درخت تصمیم ساده دارد.

الگوریتم جنگل تصادفی بر پایه دو ایده اصلی بنا شده است:

- Bagging: در این روش، چندین زیرمجموعه تصادفی از داده‌های آموزشی با جایگزینی نمونه‌ها (با نمونه‌گیری با جایگزینی) ایجاد می‌شود. سپس، برای هر زیرمجموعه، یک درخت تصمیم آموزش داده می‌شود.

- ویژگی‌های تصادفی: در هر گره از درخت تصمیم، به جای استفاده از تمام ویژگی‌ها، از زیرمجموعه‌ای تصادفی از آنها برای تعیین بهترین نقطه شکاف استفاده می‌شود.

الگوریتم جنگل تصادفی مزایای متعددی دارد، از جمله:

- دقت بالا: این الگوریتم به طور کلی نتایج دقیقی در دسته‌بندی و رگرسیون ارائه می‌دهد.
- مقاومت در برابر بیش‌برازش: به دلیل استفاده از Bagging و ویژگی‌های تصادفی، کمتر در معرض بیش‌برازش قرار می‌گیرد.
- قابلیت تفسیر: می‌توان از اهمیت هر ویژگی در پیش‌بینی نهایی آگاه شد.
- سرعت بالا: آموزش و پیش‌بینی با این الگوریتم به نسبت سریع انجام می‌شود.

معایب الگوریتم جنگل تصادفی شامل موارد زیر است:

- نیاز به حافظه زیاد: به دلیل آموزش تعداد زیادی درخت تصمیم، ممکن است به حافظه زیادی نیاز داشته باشد.
- حساسیت به نویز: در برابر نویز در داده‌ها حساس است.
- n_estimators: تعداد درختان در جنگل.
- max_features: حداکثر تعداد ویژگی‌ها مورد استفاده برای تقسیم هر گره
- max_depth: حداکثر عمق هر درخت
- min_samples_split: حداقل تعداد نمونه‌ها لازم برای تقسیم یک گره
- min_samples_leaf: حداقل تعداد نمونه‌ها لازم در یک برگ

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	5
2	1.00	1.00	1.00	3
3	1.00	1.00	1.00	15
4	1.00	1.00	1.00	16
accuracy			1.00	40
macro avg	1.00	1.00	1.00	40
weighted avg	1.00	1.00	1.00	40

شکل ۱۳: نتایج پیاده سازی جنگل تصادفی

نتایج آورده شده است که نشان می‌دهد توانسته است دقت را به ۱۰۰ درصد برساند.

AdaBoost

AdaBoost یکی دیگر از الگوریتم‌های بوستینگ است که با تمرکز بر نمونه‌های دشوارتر و تصحیح پیوسته خطاها کار می‌کند. این روش وزن‌های بیشتری را به نمونه‌هایی که بدرستی طبقه‌بندی نشده‌اند اختصاص داده و مدل‌های پی در پی را آموزش می‌دهد تا دقت آن‌ها را بهبود بخشد.
الگوریتم AdaBoost بر پایه ایده‌های زیر بنا شده است:

- تقویت طبقه‌بندی‌کننده‌های ضعیف: در هر گام، الگوریتم یک طبقه‌بندی‌کننده ضعیف را آموزش می‌دهد و سپس وزن آن را بر اساس عملکردش در دسته‌بندی نمونه‌های آموزشی تنظیم می‌کند.
- تمرکز بر روی نمونه‌های دشوار: در هر گام، الگوریتم بیشتر بر روی نمونه‌هایی تمرکز می‌کند که توسط طبقه‌بندی‌کننده‌های قبلی به اشتباه دسته‌بندی شده‌اند.

مراحل آموزش الگوریتم AdaBoost به شرح زیر است:

- ایجاد داده‌های وزنی: به هر نمونه آموزشی یک وزن اولیه اختصاص داده می‌شود.
- آموزش طبقه‌بندی‌کننده ضعیف: در هر گام، یک طبقه‌بندی‌کننده ضعیف بر روی داده‌های وزنی آموزش داده می‌شود.
- محاسبه خطای طبقه‌بندی‌کننده: خطای طبقه‌بندی‌کننده ضعیف بر روی داده‌های آموزشی محاسبه می‌شود.
- تنظیم وزن طبقه‌بندی‌کننده: وزن طبقه‌بندی‌کننده بر اساس خطای آن تنظیم می‌شود.
- به‌روزرسانی وزن نمونه‌ها: وزن نمونه‌های آموزشی بر اساس عملکرد طبقه‌بندی‌کننده به روز می‌شود.
- تکرار مراحل ۲ تا ۵: مراحل ۲ تا ۵ تا زمانی که معیار متوقف‌سازی (مانند تعداد گام‌ها یا خطای کلی) برآورده شود، تکرار می‌شوند.

الگوریتم AdaBoost مزایای متعددی دارد، از جمله:

- دقت بالا:: این الگوریتم به طور کلی نتایج دقیقی در دسته‌بندی و رگرسیون ارائه می‌دهد.
- مقاومت در برابر بیش‌برازش: به دلیل تمرکز بر روی نمونه‌های دشوار، کمتر در معرض بیش‌برازش قرار می‌گیرد.
- قابلیت تفسیر: می‌توان از اهمیت هر ویژگی در پیش‌بینی نهایی آگاه شد.
- سرعت بالا: آموزش و پیش‌بینی با این الگوریتم به نسبت سریع انجام می‌شود.

معایب الگوریتم AdaBoost شامل موارد زیر است:

- حساسیت به نویز: در برابر نویز در داده‌ها حساس است.
- نیاز به تعداد زیادی طبقه‌بندی‌کننده ضعیف: برای دستیابی به نتایج مطلوب، ممکن است به تعداد زیادی طبقه‌بندی‌کننده ضعیف نیاز داشته باشد.

- **base_estimator**:
  - نوع: estimator object، اختیاری، پیش‌فرض: None
  - توضیح: این پارامتر تعیین می‌کند که از کدام مدل پایه به عنوان تخمین‌گر ضعیف استفاده شود. به طور پیش‌فرض اگر مقداری تعیین نشود، از درخت تصمیم ساده (DecisionTreeClassifier با حداکثر عمق ۱) استفاده می‌شود.

- **n_estimators**:
  - نوع: int
  - پیش‌فرض: 50
  - توضیح: تعداد تخمین‌گرهای ضعیف که در نهایت در الگوریتم AdaBoost استفاده می‌شوند. مقدار بالاتر می‌تواند دقت مدل را افزایش دهد اما منجر به زمان بیشتر برای آموزش نیز می‌شود.
- **learning_rate**:
  - نوع: float
  - پیش‌فرض: 1.0
  - توضیح: نرخ یادگیری برای به‌روز رسانی وزن‌های هر تخمین‌گر ضعیف. این مقدار تاثیر وزن هر تخمین‌گر ضعیف را در مدل نهایی تعیین می‌کند.
- **algorithm**:
  - نوع: string
  - پیش‌فرض: 'SAMME.R'
  - توضیح: نوع الگوریتم AdaBoost که قرار است استفاده شود. دو مقدار ممکن است:
    - **'SAMME'**: الگوریتم اصلی AdaBoost برای طبقه‌بندی چندکلاسه.
    - **'SAMME.R'**: نسخه اصلاح شده از SAMME که از احتمال‌های کلاس‌ها به جای برچسب‌های کلاس استفاده می‌کند.
- **random_state**:
  - نوع: int، RandomState instance، یا None
  - پیش‌فرض: None
  - توضیح: این پارامتر برای تنظیم دانه‌ی تصادفی استفاده می‌شود تا بتوان نتایج تکرارپذیر به دست آورد. اگر مقداری تنظیم نشود، از وضعیت تصادفی پیش‌فرض استفاده خواهد شد.

همانطور که از نتایج مشهود است توانسته‌ایم دقت مدل ضعیف که درخت تصمیم گیری با عمق ۴ بوده است را بهبود دهیم. مطابق با تعاریف قبلی که داشتیم کاهش لرنینگ ریت موجب overfitting شده است. با tune کردن هایپر پارامتر ها و میتوان هم به سرعت و هم به دقت مدل افزود.

n_estimators	learning_rate	mean_f1_score	runtime
3	0.10	0.987115	0.075576
3	0.05	0.987115	0.102463
3	0.01	0.933421	0.055336
5	0.10	0.987115	0.067818
5	0.05	0.987115	0.059061
5	0.01	0.945252	0.062250
10	0.10	0.987115	0.097208
10	0.05	0.987115	0.116313
10	0.01	0.987115	0.121409
30	0.10	0.974698	0.305306
30	0.05	0.974698	0.288457
30	0.01	0.987115	0.251442
100	0.10	0.974698	0.838632
100	0.05	0.974698	0.869788
100	0.01	0.987115	0.856120

شکل ۱۴: نتایج پیاده سازی AdaBoost

۴ سوال چهار

درباره دیتاست:

حمله قلبی (بیماری‌های قلبی-عروقی) زمانی رخ می‌دهد که جریان خون به عضله قلب به طور ناگهانی مسدود می‌شود. بر اساس آمار سازمان بهداشت جهانی (WHO) هر ساله ۹.۱۷ میلیون نفر به دلیل حمله قلبی جان خود را از دست می‌دهند. مطالعات پزشکی نشان می‌دهد که سبک زندگی انسان‌ها اصلی‌ترین دلیل این مشکل قلبی است. علاوه بر این، عوامل کلیدی زیادی وجود دارند که هشدار می‌دهند که آیا یک فرد ممکن است دچار حمله قلبی شود یا خیر.

این مجموعه داده حاوی اطلاعات پزشکی بیماران است که نشان می‌دهد احتمال حمله قلبی در آن شخص کمتر یا بیشتر است. با استفاده از این اطلاعات، مجموعه داده را بررسی کرده و متغیر هدف را با استفاده از مدل‌های مختلف یادگیری ماشین طبقه‌بندی کنید و مناسب‌ترین الگوریتم برای این مجموعه داده را بیابید.

این مجموعه داده از سال ۱۹۸۸ تشکیل شده و شامل چهار پایگاه داده است: کلیولند، مجارستان، سوئیس و لانگ بیچ وی. این مجموعه داده حاوی ۷۶ ویژگی است که شامل ویژگی پیش‌بینی‌شده نیز می‌باشد، اما تمامی آزمایش‌های منتشرشده از زیرمجموعه‌ای شامل ۱۴ ویژگی استفاده می‌کنند. متغیر "target" به حضور بیماری قلبی در بیمار اشاره دارد و دارای مقادیر صحیح ۰ = بدون بیماری و ۱ = با بیماری است.

اطلاعات ویژگی‌ها:

- سن **(age)**
- جنسیت **(sex)**
- نوع درد قفسه سینه (۴ مقدار)

- فشار خون استراحتی **(resting blood pressure)**
- کلسترول سرم در **mg/dl (serum cholestoral in mg/dl)**
- قند خون ناشتا > ۱۲۰ **mg/dl (fasting blood sugar > 120 mg/dl)**
- نتایج الکتروکاردیوگرافی استراحتی (مقادیر ۰، ۱، ۲)
- حداکثر ضربان قلب به دست آمده **(maximum heart rate achieved)**
- آنژین القا شده توسط ورزش **(exercise induced angina)**
- افت **ST = ST depression induced by exercise relative to rest**
- شیب قطعه ST در تمرین پیک **(the slope of the peak exercise ST segment)**
- تعداد عروق اصلی (۰-۳) رنگ‌شده توسط فلوروسکوپی **(number of major vessels (0-3) colored by flourosopy)**
- تال: ۰ = طبیعی؛ ۱ = نقص ثابت؛ ۲ = نقص قابل برگشت **(thal: 0 = normal; 1 = fixed defect; 2 = reversible defect)**

نام‌ها و شماره‌های تامین اجتماعی بیماران اخیراً از پایگاه داده حذف شده و با مقادیر ساختگی جایگزین شده‌اند.

طبقه‌بندی با استفاده از بیز یکی از روش‌های اصلی در یادگیری ماشین است که به ویژه در زمینه‌های مختلف بسیار کاربردی است. در این روش، فرض می‌کنیم که داده‌های ما از توزیع‌های گاوسی پیروی می‌کنند. این فرض به ما امکان می‌دهد تا با استفاده از نظریه بیز به پیش‌بینی دسته‌بندی‌ها بپردازیم.

طبقه‌بندی بیزی

در طبقه‌بندی بیزی، هدف ما این است که بر اساس ویژگی‌های داده‌ها، دسته‌ی آن‌ها را پیش‌بینی کنیم. برای این کار، از قانون بیز استفاده می‌کنیم:

$$P(C_k|\mathbf{x}) = \frac{P(\mathbf{x}|C_k)P(C_k)}{P(\mathbf{x})}$$

که در آن:

- $P(C_k|\mathbf{x})$ احتمال پسینی کلاس C_k داده شده بردار ویژگی $\mathbf{x}$ است.
- $P(\mathbf{x}|C_k)$ احتمال درست‌نمایی داده‌ها در کلاس C_k است.
- $P(C_k)$ احتمال پیشینی کلاس C_k است.
- $P(\mathbf{x})$ احتمال کل داده‌ها است که به عنوان یک مقدار نرمال‌سازی عمل می‌کند.

فرض توزیع گاوسی

فرض می‌کنیم که داده‌های هر کلاس از توزیع گاوسی پیروی می‌کنند:

$$P(\mathbf{x}|C_k) = \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)$$

که در آن:

- μ_k میانگین کلاس C_k است.

- Σ_k ماتریس کواریانس کلاس C_k است.

تابع چگالی احتمال گاوسی به صورت زیر تعریف می‌شود:

$$\mathcal{N}(\mathbf{x}|\mu,\Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

پیش‌بینی کلاس

برای پیش‌بینی کلاس $\mathbf{x}$، از قانون بیشینه درست‌نمایی استفاده می‌کنیم:

$$\hat{C} = \arg\max_k P(C_k|\mathbf{x})$$

با توجه به این که $P(\mathbf{x})$ در همه‌ی کلاس‌ها یکسان است، می‌توانیم آن را نادیده بگیریم و صرفاً بیشینه $P(\mathbf{x}|C_k)P(C_k)$ را محاسبه کنیم.

مدل بیزی ساده (Naive Bayes)

در مدل بیزی ساده، فرض می‌کنیم که ویژگی‌ها مستقل از هم هستند:

$$P(\mathbf{x}|C_k) = \prod_{j=1}^{d} P(x_j|C_k)$$

این فرض ساده‌سازی‌های زیادی را در محاسبات ایجاد می‌کند و به ویژه در مجموعه داده‌های با ابعاد بالا کاربرد دارد.

ابتدا کتابخانه ها و دیتاست مورد نیاز را فرا میخوانیم. سپس به کمک کد زیر اطلاعات آماری دیتا ها را بررسی میکنیم: دو کتابخانه زیر را برای

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import learning_curve, train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

! gdown 1bTQ6Zqv7oVp7EgpNe-BCydg4abUS-7sU
!unrar x -Y "MP2_dataset.rar" "/content/"
```

```
df = pd.read_csv('heart.csv')

display(df.head())
display(df.describe())
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366	0.754146	2.323902	0.513171
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.617755	1.030798	0.620660	0.500070
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

بررسی های بهتر نصب میکنیم، ممکن است ورژن numpy قدیمی تر را نصب کنیم چون این کتابخانه با ورژن ۲۴.۱ و قدیمی تر نامپای کار میکند.

حال به تحلیل دیتاست میپردازیم:

به صورت کلی :

```
!pip install pandas_profiling
!pip install ydata_profiling
# !pip uninstall numpy
# !pip install numpy==1.24
```

Dataset statistics

Number of variables	14
Number of observations	1025
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	302
Duplicate rows (%)	29.5%
Total size in memory	112.2 KiB
Average record size in memory	112.1 B

Variable types

Numeric	5
Categorical	9

همانطور که از شکل های زیر مشهود است میزان تناسب دیتا بها باهم روابط و تکرار شدن آنها مورد بررسی فرار گرفته اند. هر فیچر بصورت خاص و جدا نیز بررسی شده است و نتایج به شرح زیر است:

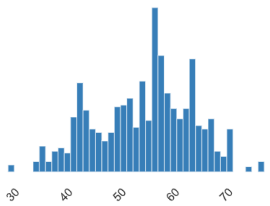
Alerts

Dataset has 302 (29.5%) duplicate rows	Duplicates
`cp` is highly overall correlated with `target`	High correlation
`target` is highly overall correlated with `cp` and 1 other fields	High correlation
`thal` is highly overall correlated with `target`	High correlation
`oldpeak` has 329 (32.1%) zeros	Zeros

age

Real number (ℝ)

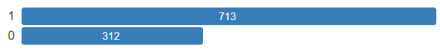
Distinct	41	Minimum	29
Distinct (%)	4.0%	Maximum	77
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	54.434146	Memory size	8.1 KiB



sex

Categorical

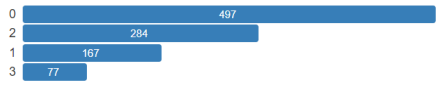
Distinct	2
Distinct (%)	0.2%
Missing	0
Missing (%)	0.0%
Memory size	8.1 KiB



cp
Categorical

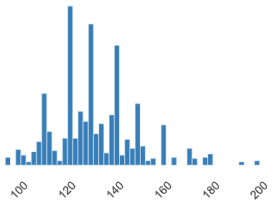
HIGH CORRELATION

Distinct	4
Distinct (%)	0.4%
Missing	0
Missing (%)	0.0%
Memory size	8.1 KiB



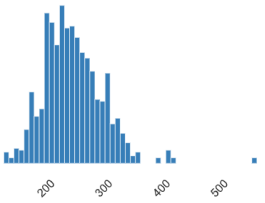
trestbps
Real number (ℝ)

Distinct	49	Minimum	94
Distinct (%)	4.8%	Maximum	200
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	131.61171	Memory size	8.1 KiB



chol
Real number (ℝ)

Distinct	152	Minimum	126
Distinct (%)	14.8%	Maximum	564
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	246	Memory size	8.1 KiB



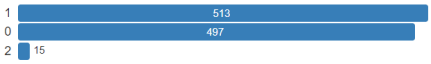
fbs
Categorical

Distinct	2
Distinct (%)	0.2%
Missing	0
Missing (%)	0.0%
Memory size	8.1 KiB

More details

restecg
Categorical

Distinct	3
Distinct (%)	0.3%
Missing	0
Missing (%)	0.0%
Memory size	8.1 KiB



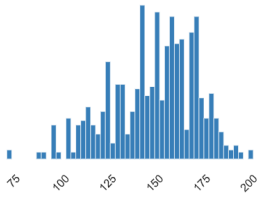
در تحلیل‌های میکرو و ماکرو از معیارهایی مانند دقت (precision)، بازخوانی (recall) و امتیاز F۱ برای ارزیابی عملکرد مدل استفاده می‌شود. این معیارها به دو صورت محاسبه می‌شوند:

Micro:

- تمام پیش‌بینی‌ها را به عنوان یک مجموعه بزرگ در نظر می‌گیرد.
- خطاها و موفقیت‌ها را بدون توجه به کلاس‌ها جمع‌آوری می‌کند.

thalach
Real number (ℝ)

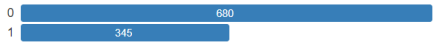
Distinct	91	**Minimum**	71
Distinct (%)	8.9%	**Maximum**	202
Missing	0	**Zeros**	0
Missing (%)	0.0%	**Zeros (%)**	0.0%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	149.11415	**Memory size**	8.1 KiB



More details

exang
Categorical

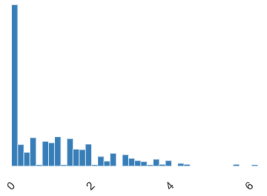
Distinct	2
Distinct (%)	0.2%
Missing	0
Missing (%)	0.0%
Memory size	8.1 KiB



oldpeak
Real number (ℝ)

ZEROS

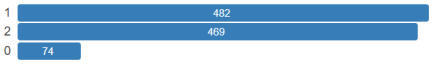
Distinct	40	**Minimum**	0
Distinct (%)	3.9%	**Maximum**	6.2
Missing	0	**Zeros**	329
Missing (%)	0.0%	**Zeros (%)**	32.1%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	1.0715122	**Memory size**	8.1 KiB



More details

slope
Categorical

Distinct	3
Distinct (%)	0.3%
Missing	0
Missing (%)	0.0%
Memory size	8.1 KiB



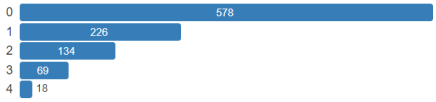
- دقت، بازخوانی و امتیاز F۱ به صورت کلی محاسبه می‌شوند.

Macro:

- عملکرد مدل را برای هر کلاس به صورت جداگانه محاسبه می‌کند.
- سپس میانگین این معیارها را برای تمامی کلاس‌ها محاسبه می‌کند.
- به کلاس‌هایی که تعداد نمونه‌های کمتری دارند وزن بیشتری می‌دهد.

ca
Categorical

Distinct	5
Distinct (%)	0.5%
Missing	0
Missing (%)	0.0%
Memory size	8.1 KiB

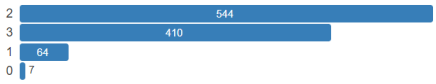


More details

thal
Categorical

HIGH CORRELATION

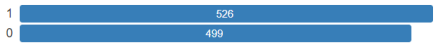
Distinct	4
Distinct (%)	0.4%
Missing	0
Missing (%)	0.0%
Memory size	8.1 KiB

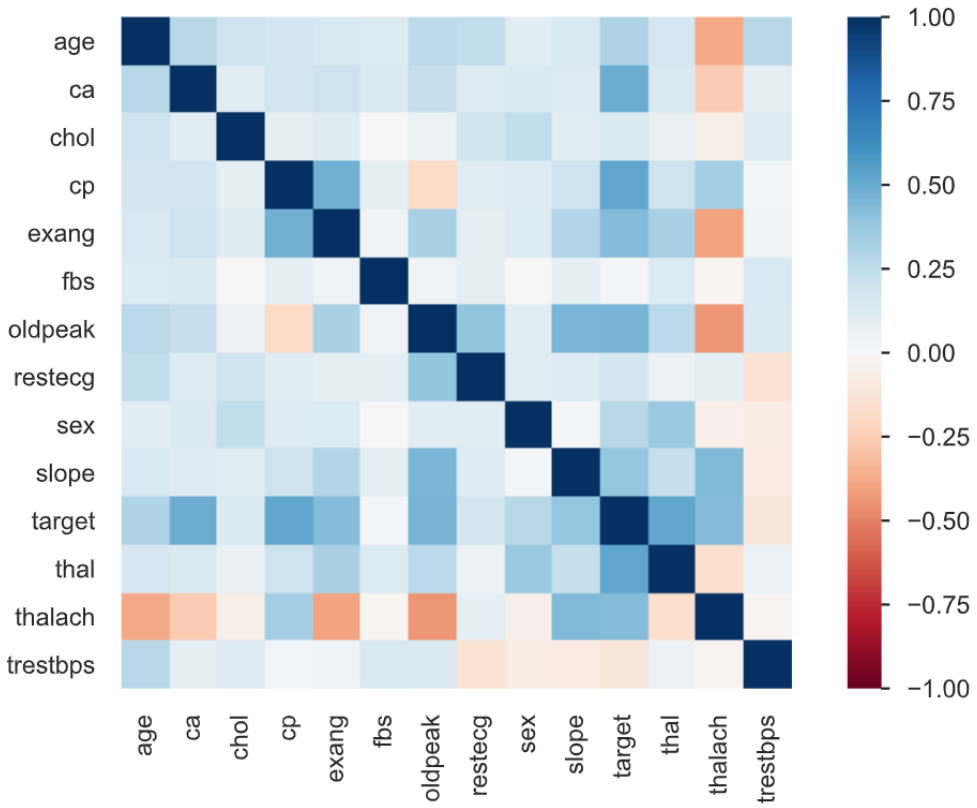


target
Categorical

HIGH CORRELATION

Distinct	2
Distinct (%)	0.2%
Missing	0
Missing (%)	0.0%
Memory size	8.1 KiB





Duplicate rows

Most frequently occurring

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target	# duplicates
9	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1	8
0	29	1	1	130	204	0	0	202	0	0.0	2	0	2	1	4
3	35	0	0	138	183	0	1	182	0	1.4	2	0	2	1	4
4	35	1	0	120	198	0	1	130	1	1.6	1	0	3	0	4
6	35	1	1	122	192	0	1	174	0	0.0	2	0	2	1	4
10	38	1	3	120	231	0	1	182	1	3.8	1	0	3	0	4
12	39	0	2	138	220	0	1	152	0	0.0	1	0	2	1	4
13	39	1	0	118	219	0	1	140	0	1.2	1	0	3	0	4
15	40	1	0	110	167	0	0	114	1	2.0	1	0	3	0	4
16	40	1	0	152	223	0	1	181	0	0.0	2	0	3	0	4

```
df =df.drop_duplicates()
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df.drop('target', axis=1))
X_train, X_test, y_train, y_test = train_test_split(scaled_features, df['target'], test_size=0.2, random_state=44)
print(X_train.shape)
print(X_test.shape)
```

✓ 0.0s

```
(241, 13)
(61, 13)
```

```
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)
print(f"Accuracy: {accuracy}\n")

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Blues', xticklabels=model.classes_, yticklabels=model.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

تفاوت بین Micro و Macro

Micro:

- مناسب برای ارزیابی عملکرد کلی مدل در مواجهه با عدم توازن کلاس‌ها.
- به هر نمونه وزن مساوی می‌دهد.

Macro:

- مناسب برای ارزیابی عملکرد مدل در کلاس‌های با تعداد نمونه کمتر.
- به هر کلاس وزن مساوی می‌دهد، بنابراین ممکن است برای کلاس‌های کوچک‌تر مهم‌تر باشد.

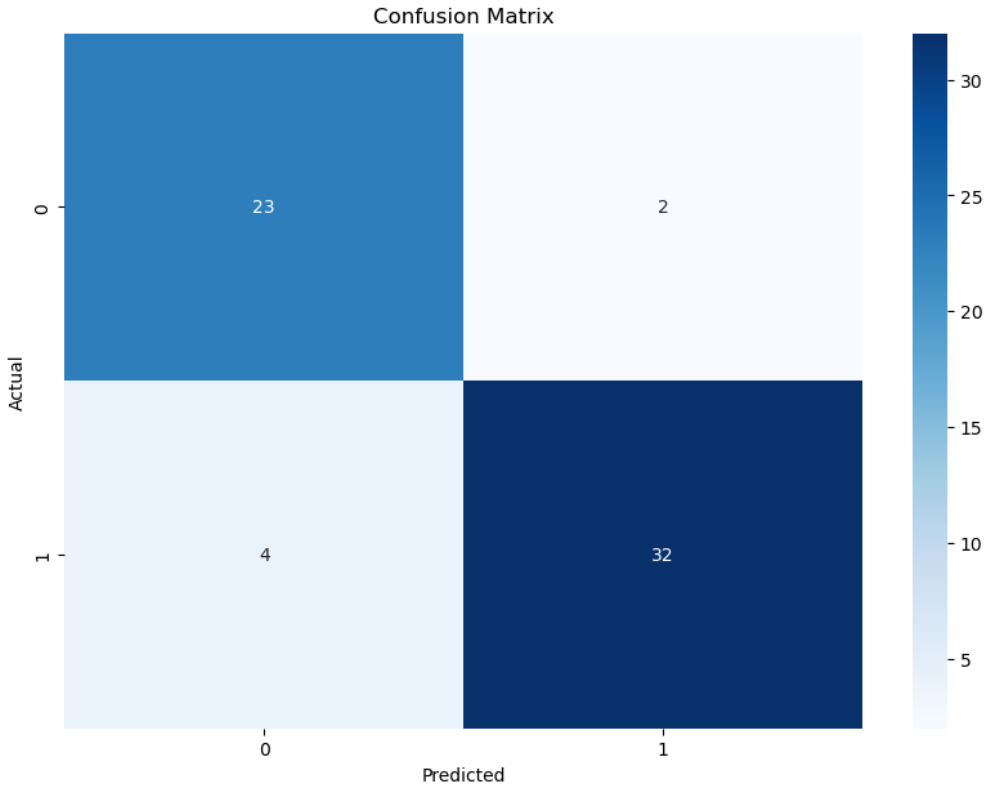
منبع

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

           0       0.85      0.92      0.88        25
           1       0.94      0.89      0.91        36

    accuracy                           0.90        61
   macro avg       0.90      0.90      0.90        61
weighted avg       0.90      0.90      0.90        61

Accuracy: 0.9016393442622951
```



مورفی، کوین پی.، "یادگیری ماشین: یک دیدگاه احتمالاتی"، فصل ۴

۱.۴ انتخاب تصادفی ۵ دیتا

```python
np.random.seed(44)
random_indices = np.random.choice(len(X_test), 5, replace=False)
X_sample = X_test[random_indices]
y_sample_true = y_test.iloc[random_indices]
y_sample_pred = model.predict(X_sample)

correct_predictions = y_sample_true == y_sample_pred
incorrect_predictions = ~correct_predictions

print("\nAnalysis of the Predictions:")
print(f"Correct Predictions: {np.sum(correct_predictions)}")
print(f"Incorrect Predictions: {np.sum(incorrect_predictions)}")
print('true labels:', y_sample_true.tolist())
print('predict labels:', y_sample_pred.tolist())
```

✓ 0.0s

```
Analysis of the Predictions:
Correct Predictions: 5
Incorrect Predictions: 0
true labels: [1, 0, 0, 1, 1]
predict labels: [1, 0, 0, 1, 1]
```

۵ دیتا بصورت رندوم انتخاب و پیشبینی شدند که همانطور که مشخص است مدل خیلی خوب عمل کرده است و هر ۵ تا را درست پیشبینی کرده است.