به نام خدا

دانشکده مهندسی برق

گزارش درس یادگیری ماشین

مقطع: کارشناسی ارشد گرایش: مهندسی کنترل

گزارش مینی پروژه دوم

توسط:

مرجان محمدی

۴۰۱۱۱۵۳۴

استاد درس:

دکتر علیاری

بهار ۱۴۰۳

فهرست مطالب

https://github.com/marjanMohammadi1375/MachineLearning2023

https://colab.research.google.com/drive/1AXoO7orpa_vnGY-e70Q41NAYOi2mSsa-?usp=drive_link

نکات کلی:

برای همه ی کد ها از رندم استیت ۳۴ که دو رقم آخر شماره دانشجویی است، استفاده شده است.

برای سوال دوم چون باقیمانده دو رقم آخر شماره دانشجویی بر ۸ برابر ۲ است، از دیتاهایی که پسوند ۲ دارند استفاده شده است.

۱ سوال اول

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

۱

---

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

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

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

۱-۱

خصوصیات و کاربردهای ReLU:

ReLU یا واحد خطی اصلاح‌شده، یک فعال‌ساز نامحدود است که مقادیر منفی را به صفر تبدیل کرده و مقادیر مثبت را بدون تغییر انتقال می‌دهد. این خصوصیات ReLU آن را برای استفاده در لایه‌های مخفی شبکه‌های عصبی بسیار مناسب می‌سازد، زیرا:

جلوگیری از مشکل انفجار گرادیان: با حذف مقادیر منفی، ReLU به حفظ ثبات عددی در طول آموزش کمک می‌کند.

فراهم کردن خطی بودن جزئی: این امر به شبکه اجازه می‌دهد که توابع پیچیده‌تری را یاد بگیرد.

خصوصیات و کاربردهای سیگموید:

سیگموید یک فعال‌ساز محدود است که خروجی‌هایی بین ۰ و ۱ تولید می‌کند، و بنابراین اغلب در لایه‌های خروجی شبکه‌های عصبی برای مسائل طبقه‌بندی دوکلاسه به کار می‌رود. این فعال‌ساز:

مدل‌سازی احتمالات: خروجی‌های سیگموید را می‌توان به عنوان احتمال عضویت در یکی از دو کلاس در نظر گرفت.

مشکل اشباع: در مقادیر بسیار بالا یا پایین، شیب تابع سیگموید به شدت کاهش می‌یابد، که می‌تواند منجر به از دست رفتن گرادیان شود.

مشکلات ناشی از ترکیب ReLU و سیگموید:

وقتی خروجی ReLU به عنوان ورودی به سیگموید داده می‌شود:

ریسک اشباع: از آنجا که ReLU می‌تواند مقادیر بسیار بزرگی را تولید کند، احتمال اینکه ورودی‌های سیگموید به سرعت به حداکثر مقدار خود برسند و در نتیجه اشباع شوند، زیاد است. این امر می‌تواند یادگیری را در آن نقاط تقریباً غیرممکن کند، زیرا گرادیان‌های بسیار کوچک یا صفر خواهند بود.

کاهش کارایی یادگیری: به دلیل اشباع گرادیان، شبکه ممکن است در یادگیری ویژگی‌های مفید از داده‌ها ناتوان باشد و در نتیجه دقت طبقه‌بندی پایین آید.

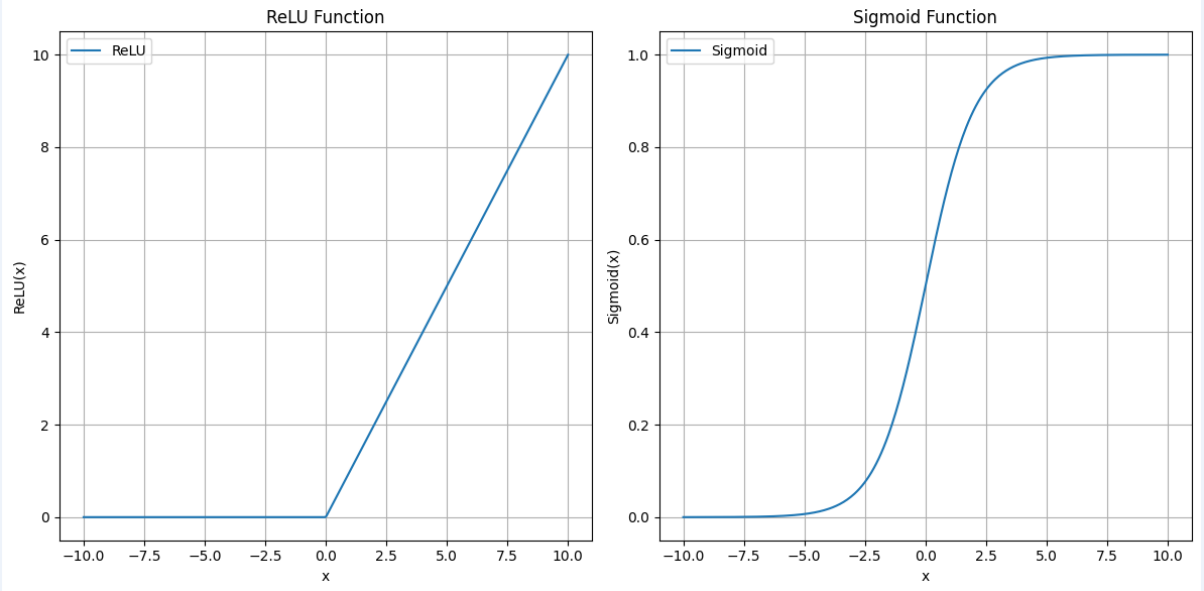
راه‌حل‌ها

استفاده از فعال‌ساز خطی یا بدون فعال‌ساز در لایه‌ی قبل از سیگموید: به جای ReLU، می‌توان از فعال‌ساز خطی (یا هیچ فعال‌ساز) استفاده کرد تا مقادیر ورودی به سیگموید محدود و کنترل‌شده‌تر باشند، که به جلوگیری از اشباع کمک می‌کند و یادگیری مؤثرتری را فراهم می‌آورد.

بررسی جایگزین‌های دیگر: فعال‌سازهای دیگر مانند تانژانت هذلولوی یا حتی فعال‌سازهای جدیدتر مانند ELU یا Leaky ReLU را می‌توان بررسی کرد که ممکن است خصوصیات مطلوب‌تری برای لایه‌های قبل از خروجی داشته باشند.

```
# Define the ReLU function
def relu(x):
    return np.maximum(0, x)

# Define the Sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Generate a range of values from -10 to 10 for x
x = np.linspace(-10, 10, 400)

# Compute the ReLU and Sigmoid values
y_relu = relu(x)
y_sigmoid = sigmoid(x)
```

همانطور که در کد بالا میبینید، تابع sigmoid و ReLU تعریف شده اند که با توجه به کد بالا رسم شده و در زیر نمایش داده شده اند.



۱-۲

تابع فعال‌سازی ELU یا Exponential Linear Unit، یک جایگزین برای ReLU است که به شکل زیر تعریف می‌شود:

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

که در آن alpha یک پارامتر مثبت است که مقدار تابع را در زمانی که x منفی است تنظیم می‌کند.

محاسبه گرادیان ELU:

$$ELU'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha e^{x} & \text{if } x \leq 0 \end{cases}$$

گرادیان تابع ELU به شکل زیر محاسبه می‌شود:

1. **For** $x > 0$:

$$ELU'(x) = 1$$

2. **For** $x \leq 0$:

$$ELU'(x) = \alpha e^{x}$$

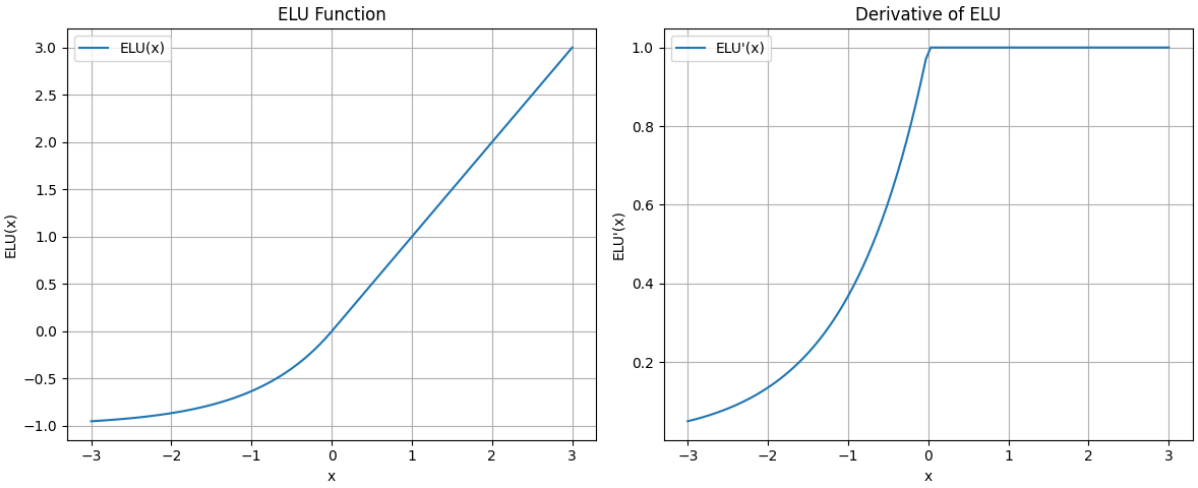
مزایای ELU نسبت به ReLU:

یکی از مزایای اصلی ELU نسبت به ReLU این است که ELU می‌تواند مقادیر منفی را نیز در نظر بگیرد، که این امر می‌تواند به جلوگیری از مشکل مرگ نورون‌ها (neuron dying problem) کمک کند. در ReLU، نورون‌هایی که مقادیر منفی دریافت می‌کنند فعال نمی‌شوند و گرادیان‌ها در این نقاط صفر می‌شوند، که می‌تواند منجر به از دست دادن اطلاعات در طول آموزش شود. ELU با ارائه خروجی‌های منفی برای مقادیر منفی ورودی، اجازه می‌دهد که اطلاعات و گرادیان‌ها حفظ شوند، حتی اگر ورودی منفی باشد.

علاوه بر این، ELU به دلیل داشتن شکل نمایی در بخش منفی، می‌تواند به شبکه کمک کند تا سریعتر همگرا شود. این مزیت به خصوص در مواردی که تابع هزینه دارای مناطقی است که بهینه‌سازی مشکل است، مفید می‌افتد. همچنین استفاده از این تابع فعالساز باعث کاهش مشکل مرگ نورون ها و میانگین خروجی نزدیک به صفرباعث بهبود عملکرد شبکه های عصبی نسبت به ReLU می شود.

```python
def elu(x, alpha=1.0):
    # ELU activation function
    return np.where(x > 0, x, alpha * (np.exp(x) - 1))

def elu_derivative(x, alpha=1.0):
    # Derivative of the ELU function
    return np.where(x > 0, 1, alpha * np.exp(x))

# Generate x values from -3 to 3
x = np.linspace(-3, 3, 100)

# Calculate ELU and its derivative for the range
y_elu = elu(x)
y_elu_derivative = elu_derivative(x)
```

همانطور که در کد بالا مشخص است تابع فعالساز ELU تعریف شده و در شکل زیر همراه با گرادیانش رسم شده است.



۱-۳

نورون مک‌کالوک-پیتس (McCulloch-Pitts Neuron) ، یکی از اولین مدل‌های ریاضی نورون‌های بیولوژیکی است که توسط Warren McCulloch و Walter Pitts در سال ۱۹۴۳ معرفی شد. این مدل یک نورون ساده باینری است که اصول اولیه پردازش اطلاعات در شبکه های عصبی را شبیه سازی می کند. در اینجا برخی از ویژگی ها و اصول این مدل توضیح داده شده است:

اجزاء و عملکرد:

ورودی‌ها : نورون مک‌کالوک-پیتس ورودی باینری دارد که به صورت ۰ یا ۱ هستند. هر ورودی با یک وزن مرتبط است که تاثیر آن را نشان می دهد.

وزن‌ها : به هر ورودی یک وزن عددی تخصیص داده می‌شود که آن ورودی را در تعیین خروجی نورون مشخص می‌کند.

تابع جمع : تمامی ورودی‌ها، ضربدر وزن‌هایشان شده و جمع‌آوری می‌شوند. این جمع به عنوان ورودی یا خالص (ورودی خالص) پتانسیل نورون می‌شود.

آستانه : مقدار آستانه یک مقدار ثابت است که می‌کند نورون فعال شود یا نه. این مقدار معمولاً با یک بایاس نیز می‌شود.

تابع فعالسازی: خروجی نورون مک‌کالوک-پیتس باینری است و به صورت ۰ یا ۱ تعریف می‌شود. اگر ورودی خالص بیشتر از آستانه باشد، خروجی ۱ و در غیر این صورت ۰ خواهد بود.

ویژگی‌ها

باینری بودن: ورودی‌ها و خروجی‌ها به صورت باینری (۰ یا ۱) هستند.

خطی بودن: نورون مک کالوک-پیتس یک مدل خطی است که فقط از جمع وزنی می‌کند.

تصمیم‌گیری ساده: با استفاده از آستانه، نورون می‌تواند تصمیم‌های ساده‌ای را بگیرد (فعال یا غیرفعال شود).

کاربردها

نورون مک‌کالوک-پیتس پایه و اساس شبکه‌های عصبی مدرن را می‌سازد و از آن برای شبیه‌سازی عملیات منطقی پایه (مثل AND, OR, NOT) استفاده می‌شود. این مدل ساده نشان می دهد که حتی یک سیستم ساده باینری قادر به انجام محاسبات منطقی و پردازش اطلاعات است.

محدودیت‌ها

به دلیل ساده بودن بیش از حد، این مدل نمی‌تواند الگوها و اطلاعات پیچیده را پردازش کند.

در این مثال، یک نورون مک‌کالوک-پیتس به عنوان یک دروازه و پیاده‌سازی می‌شود که دو ورودی را دریافت و خروجی ۱ را فقط در تصویر تولید می‌کند که هر دو ورودی ۱ هستند.

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

# Define McCulloch-Pitts Neuron
class McCullochPittsNeuron:
    def __init__(self, weights, threshold):
        self.weights = weights
        self.threshold = threshold

    def model(self, X):
        return 1 if (np.dot(self.weights, X) + self.threshold) >= 0
else 0

# Define function to determine if point is inside triangle
def is_point_in_triangle(x, y):
    neuron1 = McCullochPittsNeuron([-2, -1], 6)
    neuron2 = McCullochPittsNeuron([2, -1], -2)
    neuron3 = McCullochPittsNeuron([0, 1], 0)
    neuron4 = McCullochPittsNeuron([1, 1, 1], -3)

    zone1 = neuron1.model(np.array([x, y]))
    zone2 = neuron2.model(np.array([x, y]))
    zone3 = neuron3.model(np.array([x, y]))
    zone4 = neuron4.model(np.array([zone1, zone2, zone3]))

    return zone4

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

# Classify points
red_points = []  # outside zone
green_points = []  # inside zone

for x, y in zip(x_values, y_values):
    if is_point_in_triangle(x, y) == 0:
        red_points.append((x, y))
    else:
        green_points.append((x, y))

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

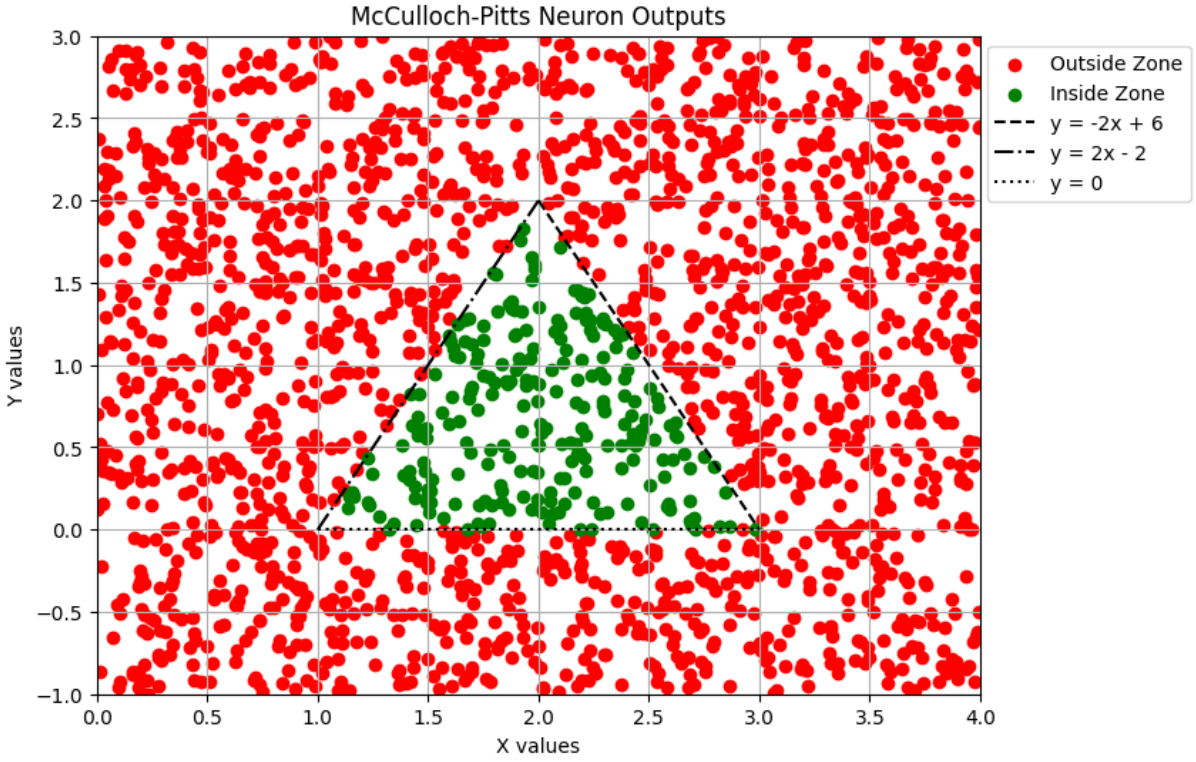
کد بالا یک نمونه از کاربردهای نورون‌های مدل مک‌کالوک-پیتس است که برای تعیین مکان‌ها در یک مکان دو بعدی به یک مثال استفاده می‌شود. ابتدا کتابخانه‌های numpy و matplotlib.pyplot برای محاسبات عددی و ترسیم تصاویر فراخوانی می‌شوند. سپس یک کلاس برای نورون مک‌کالوک-پیتس تعریف می‌شود که دارای وزن‌ها و آستانه است و یک مدل پایه برای فعال یا غیرفعال کردن نورون بر اساس ورودی‌ها می‌دهد.

تابع is_point_in_triangle با استفاده از چهار نورون مک‌کالوک-پیتس تعیین می‌کند که آیا یک نقطه در داخل یا خارج از مثلث تعریف شده است. هر نورون یکی از جنبه‌های منطقی مرزهای مثلث را بررسی می‌کند و در نهایت تصمیم‌گیری می‌کند که نقطه‌ای در یا خارج از مثلث قرار دارد.

برای ارزیابی این سیستم، ۲۰۰۰ نقطه در یک محدوده تولید شده و هر نقطه توسط تابع is_point_in_triangle طبقه‌بندی می‌شود. نقاطی که داخل مثلث هستند به رنگ سبز و نقاط خارج از مثلث به رنگ قرمز نشان داده می‌شوند.

در نهایت، نقاط به خطوط همراه تصمیم‌گیری می‌شوند که مرزهای مثلث هستند، روی نمودار رسم می‌شوند. این نمودار شامل عناصری مانند، محدوده‌های محورها، شبکه‌بندی، افسانه و ذخیره‌سازی نمودار به فایل صورت تصویری است. این مثال می‌دهد که چگونه مدل‌های ساده عصبی می‌توانند برای حل مسائل عملی در روش‌های بصری و شهودی استفاده شوند.

شکل زیر خروجی کد می باشد.

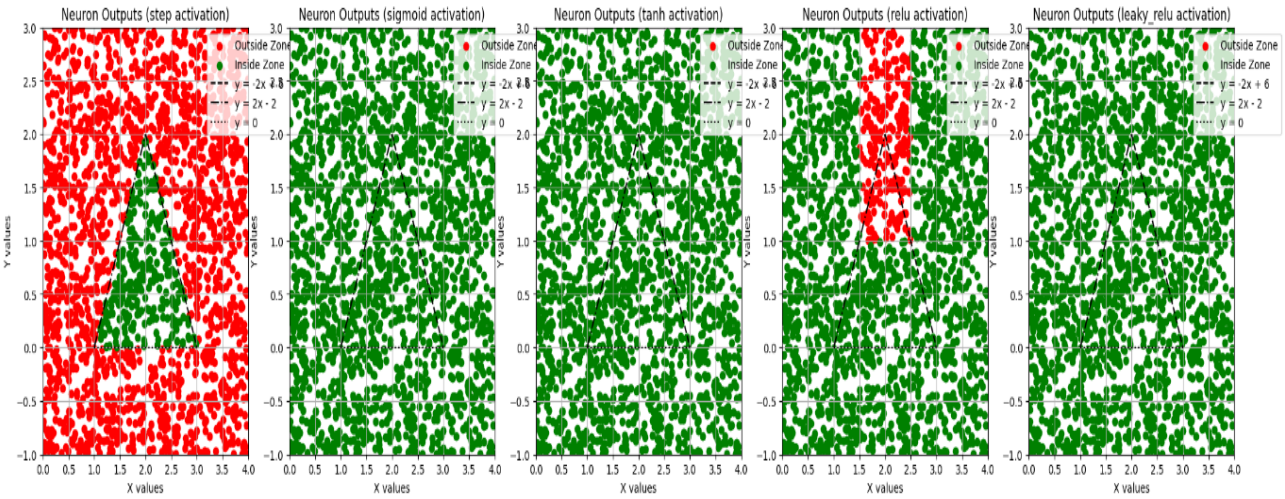


حال در کد زیر ۴ تابع فعالساز sigmoid و tanh و ReLU و Leaky relu تعریف شده اند.

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

# Define McCulloch-Pitts Neuron with different activation functions
class McCullochPittsNeuron:
    def __init__(self, weights, threshold, activation='step'):
        self.weights = weights
        self.threshold = threshold
        self.activation = activation

    def step_function(self, x):
        return 1 if x >= 0 else 0

    def sigmoid_function(self, x):
        return 1 / (1 + np.exp(-x))

    def tanh_function(self, x):
        return np.tanh(x)

    def relu_function(self, x):
        return np.maximum(0, x)

    def leaky_relu_function(self, x):
        return np.where(x > 0, x, x * 0.01)

    def model(self, X):
        net_input = np.dot(self.weights, X) + self.threshold
        if self.activation == 'step':
            return self.step_function(net_input)
        elif self.activation == 'sigmoid':
            return self.sigmoid_function(net_input)
        elif self.activation == 'tanh':
            return self.tanh_function(net_input)
        elif self.activation == 'relu':
            return self.relu_function(net_input)
        elif self.activation == 'leaky_relu':
            return self.leaky_relu_function(net_input)
        else:
            raise ValueError("Unknown activation function")

# Define function to determine if point is inside triangle using
different activations
def is_point_in_triangle(x, y, activation='step'):
    neuron1 = McCullochPittsNeuron([-2, -1], 6, activation)
    neuron2 = McCullochPittsNeuron([2, -1], -2, activation)
    neuron3 = McCullochPittsNeuron([0, 1], 0, activation)
    neuron4 = McCullochPittsNeuron([1, 1, 1], -3, activation)
```

```
    zone1 = neuron1.model(np.array([x, y]))
    zone2 = neuron2.model(np.array([x, y]))
    zone3 = neuron3.model(np.array([x, y]))
    zone4 = neuron4.model(np.array([zone1, zone2, zone3]))

    return zone4

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

# Helper function to classify points based on activation function
def classify_points(activation):
    red_points = []  # outside zone
    green_points = []  # inside zone

    for x, y in zip(x_values, y_values):
        if is_point_in_triangle(x, y, activation) == 0:
            red_points.append((x, y))
        else:
            green_points.append((x, y))

    return red_points, green_points

# Classify points using different activation functions
activations = ['step', 'sigmoid', 'tanh', 'relu', 'leaky_relu']
results = {activation: classify_points(activation) for activation in
activations}
```

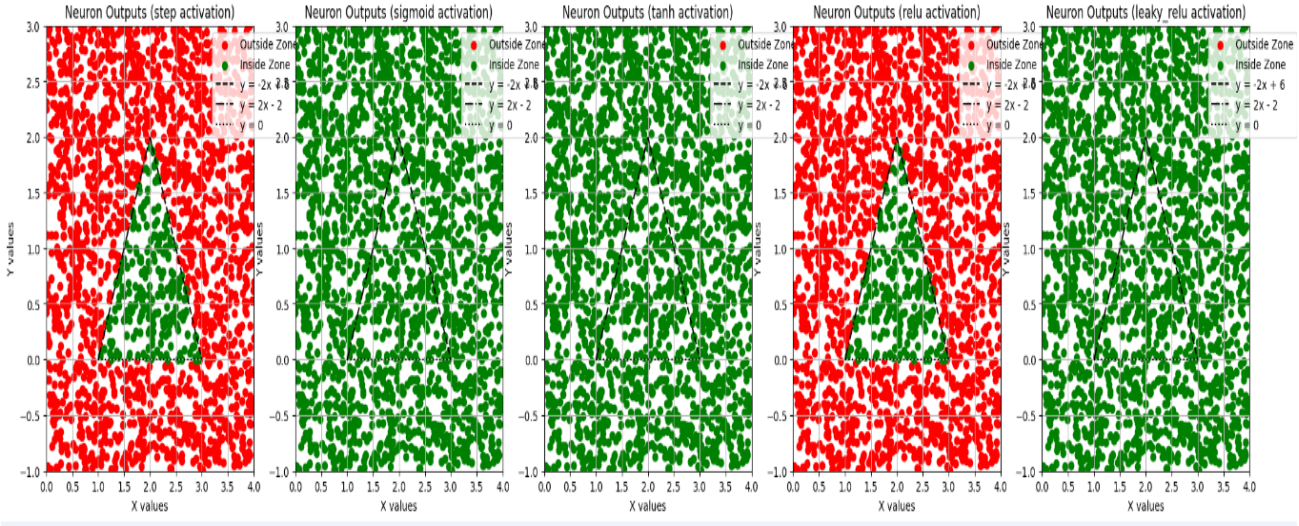
و خروجی ها به صورت زیر درآمده اند که همانطور که مشخص است هیچ کدام نتوانسته اند با شرایط یکسان نقاط را تشخیص دهند.



حال تعدا نورون ها را یکی کاهش داده و شرایط تصمیم گیری را تغییر داده و میبینیم ReLU طبق شکل زیر به جواب رسیده است.

```
    def model(self, X):
        net_input = np.dot(self.weights, X) + self.threshold
        if self.activation == 'step':
            return self.step_function(net_input)
        elif self.activation == 'sigmoid':
            return self.sigmoid_function(net_input)
        elif self.activation == 'tanh':
            return self.tanh_function(net_input)
        elif self.activation == 'relu':
            return self.relu_function(net_input)
        elif self.activation == 'leaky_relu':
            return self.leaky_relu_function(net_input)
        else:
            raise ValueError("Unknown activation function")

# Define function to determine if point is inside triangle using
different activations
def is_point_in_triangle(x, y, activation='step'):
    neuron1 = McCullochPittsNeuron([-2, -1], 6, activation)
    neuron2 = McCullochPittsNeuron([2, -1], -2, activation)
    neuron3 = McCullochPittsNeuron([0, 1], 0, activation)

    zone1 = neuron1.model(np.array([x, y]))
    zone2 = neuron2.model(np.array([x, y]))
    zone3 = neuron3.model(np.array([x, y]))

    # Check if point is inside the triangle
    return zone1 and zone2 and zone3

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

# Helper function to classify points based on activation function
def classify_points(activation):
    red_points = []  # outside zone
    green_points = []  # inside zone

    for x, y in zip(x_values, y_values):
        if is_point_in_triangle(x, y, activation):
            green_points.append((x, y))
        else:
            red_points.append((x, y))
```

```
    return red_points, green_points

# Classify points using different activation functions
activations = ['step', 'sigmoid', 'tanh', 'relu', 'leaky_relu']
results = {activation: classify_points(activation) for activation in
activations}
```



۲ سوال دوم

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

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

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

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

[۱] تشخیص اینکه با کدام روش می‌توانید این کار را انجام دهید با شماست.
[۲] X، باقی‌ماندهٔ تقسیم دو رقم آخر شمارهٔ دانشجویی شما بر ۴ است.

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

۲-۱

دیتاست CWRU Bearing که توسط دانشگاه کیس وسترن ریزرو تهیه و عرضه می‌شود، مجموعه‌ای استاندارد در زمینه تشخیص خطاهای بیرینگ در ماشین آلات دوار است. این دیتاست برای ارزیابی و توسعه روش‌های نوین در مانیتورینگ وضعیت و نگهداری تجهیزات استفاده می‌شود و شامل داده‌های ارتعاشی است که از بیرینگ‌های تحت شرایط عملیاتی مختلف جمع‌آوری شده‌اند.

دیتاست CWRU به دو دسته تقسیم می شود: داده های نرمال و داده های دارای خطا:

داده‌های نرمال: شامل موارد ارتعاشی ثبت شده از بیرینگ‌هایی است که بدون هیچ گونه خطا و در شرایط عملیاتی استاندارد کار می‌کنند. این داده‌ها برای تعریف خط پایه‌ای از عملکرد سالم بیرینگ‌ها استفاده می‌شود.

داده‌های دارای خطا: شامل داده‌های ارتعاشی ثبت شده از بیرینگ‌های دارای انواع خطاها، شامل خطاهای توپ بیرینگ، خطاهای حرکت توپ، خطاهای قفسه بیرینگ.

کاربردها

این دیتاست در توسعه و تست الگوریتم‌های تشخیص خطا و مدل‌های پیش‌بینی ماشین به کار رفته و برای‌بینی عمر مفید بیرینگ‌ها و مانیتورینگ وضعیت ماشین‌آلات کاربرد دارد. مدل‌های توسعه یافته بر اساس این دیتاست، می‌تواند تغییراتی در عملکرد بیرینگ‌ها را ایجاد کند و به خرابی‌های بهبود و افزایش کارایی تعمیر و نگهداری کمک کند.

در دیتاست Case Western Reserve University Bearing، فایل‌ها بر اساس نوع عیب و شرایط عملیاتی توصیف می‌شوند. هر فایل نام‌گذاری خاصی دارد که نوع عیب و بعضی اوقات موقعیت آن عیب را بیان می‌کند. انواع عیب‌های شامل در این دیتاست عبارتند از:

عیب‌های توپ بیرینگ (Ball Faults): این عیب‌ها به فایل‌هایی اشاره دارند که در آن‌ها توپ‌های بیرینگ دچار خرابی شده‌اند، مانند ترک خوردگی یا ساییدگی. این فایل‌ها معمولا با کد BF (Ball Fault) مشخص می‌شوند.

عیب‌های مسیر حرکت داخلی (Inner Raceway Faults): این عیب‌ها در مسیر حرکت داخلی بیرینگ رخ داده و معمولا با کد IR (Inner Race) نشان داده می‌شوند.

عیب‌های مسیر حرکت خارجی (Outer Raceway Faults): این عیب‌ها بر روی مسیر حرکت خارجی بیرینگ اتفاق می‌افتند و با کد OR (Outer Race) مشخص می‌شوند.

ابتدا دیتا را فراخوانی کرده و طبق زیر دانلود می کنیم:

```
# Constants
M, N = 250, 200
np.random.seed(25)
my_ID_Number = 34
```

```
dataset_url = {
    'normal':
'https://engineering.case.edu/sites/default/files/99.mat',
    'faulty1':
'https://engineering.case.edu/sites/default/files/107.mat',
    'faulty2':
'https://engineering.case.edu/sites/default/files/120.mat',
    'faulty3':
'https://engineering.case.edu/sites/default/files/132.mat'
}

# Functions
def load_data(urls):
    data = {}
    for label, url in urls.items():
        !wget -q {url}  # Quiet mode, no output
        data[label] = loadmat(url.split('/')[-1])  # Load and return
data
    return data


# Load all datasets
data = load_data(dataset_url)
cols = {key: list(val.keys())[-4:] for key, val in data.items()}
```

در قسمت سازماندهی داده، داده‌های ارتعاشی که از دیتاست‌های بارگیری شده به دست آمده‌اند، به شکل ماتریس‌های دوبعدی سازماندهی می‌شوند. هر ماتریس نشان‌دهنده یک نوع داده و یک ویژگی خاص است. به طور کلی، مراحل این بخش عبارتند از:

لود داده: ابتدا، داده‌های بارگیری شده از لینک‌های مختلف، با استفاده از تابع loadmat از کتابخانه scipy.ioبه داده‌های قابل استفاده در پایتون تبدیل می‌شوند.

ساختاردهی ماتریس: با توجه به ساختار داده‌های ارتعاشی و ویژگی‌های مورد نیاز، ماتریس‌های دوبعدی با اندازه‌های معین معمولا (MxN) ساخته می‌شوند. این ماتریس‌ها برای ذخیره سازی داده‌های زمانی ارتعاشی از بیرینگ‌ها استفاده می‌شوند.

پرکردن ماتریس: با استفاده از داده‌های بارگیری شده، ماتریس‌ها با مقادیر مربوط به داده‌های ارتعاشی پر می‌شوند. این کار با توجه به نوع و ویژگی‌های مورد نیاز انجام می‌شود.

مدیریت موارد استثنا: گاهی اوقات ممکن است مرزهای ماتریس با محدوده داده‌های ارتعاشی مطابقت نداشته باشد یا برخی از داده‌ها از نوعی از اشکالات یا نویز رنج برده باشند. در این صورت، موارد استثنا به طور معمول با استفاده از ساختارهای کنترل شرایط مورد نظر مدیریت می‌شوند.

با این رویکرد، داده‌های ارتعاشی موجود در دیتاست، به شکل ماتریس‌های دوبعدی با ساختار منظم و مرتب شده تبدیل می‌شوند که برای محاسبه و استفاده در مراحل بعدی پردازش داده مناسب هستند.

```
def organize_data(data, cols):
    all_matrices = {}
    for label, datasets in data.items():
        for col in cols[label]:
            mat = np.zeros((M, N))
            for j in range(M):
                try:
                    mat[j, :] = datasets[col][j:j+N].reshape(-1,)
                except Exception as e:
                    continue  # Handle case where window exceeds bounds
of data
            all_matrices[f"{label}_{col}"] = mat
    return all_matrices


# Organize data into matrices
matrices = organize_data(data, cols)
```

در قسمت استخراج ویژگی‌ها، برای هر ماتریس ارتعاشی که به دست آمده است، انواع مختلفی از ویژگی‌های آماری و سیگنالی محاسبه می‌شود. این ویژگی‌ها معمولا اطلاعات مفیدی از نحوه رفتار و شکل سیگنال‌های ارتعاشی را در طول زمان ارائه می‌دهند. برخی از ویژگی‌های معمولا محاسبه شده در اینجا شامل موارد زیر می‌شوند:

انحراف معیار (Standard Deviation): اندازه انحراف سیگنال‌های ارتعاشی از میانگین آنها که نشان دهنده پراکندگی داده است.

نقطه بیشینه (Peak): بیشترین مقدار سیگنال ارتعاشی در هر زمان.

شکوفه‌گی (Skewness) :انحراف از تقارن سیگنال ارتعاشی.

میانگین (Mean): میانگین مقادیر سیگنال ارتعاشی در هر زمان.

میانگین مطلق (Absolute Mean): میانگین مقادیر مطلق سیگنال ارتعاشی در هر زمان.

میانگین جذر مربعات (Root Mean Square) :میانگین مقادیر جذر مربعی سیگنال ارتعاشی در هر زمان.

مربع میانگین جذر مربعات (Square Root Mean): مربع میانگین مقادیر جذر مربعی مطلق سیگنال ارتعاشی در هر زمان.

کرتوسیتی (Kurtosis): اندازه شیب یا تنگی سیگنال ارتعاشی.

ضریب رویه (Crest Factor): نسبت بیشینه مقدار سیگنال به میانگین جذر مربعات آن.

ضریب مرکز (Clearance Factor): نسبت بیشترین مقدار سیگنال به میانگین جذر مربعی مطلق آن.

پیک به پیک (Peak-to-Peak): محدوده بین مقدار بیشینه و کمینه سیگنال در هر زمان.

فرم سیگنال (Shape Factor): نسبت میانگین جذر مربعات به میانگین مطلق سیگنال.

ضریب تاثیر (Impact Factor): نسبت میانگین جذر مربعات به میانگین مطلق سیگنال.

ضریب تاثیر ناگهانی (Impulse Factor): نسبت میانگین سیگنال به میانگین مطلق آن.

این ویژگی‌ها اطلاعات مهمی از سیگنال‌های ارتعاشی بیرینگ ارائه می‌دهند که می‌توانند به عنوان ورودی‌های مناسب برای الگوریتم‌های یادگیری ماشین برای تشخیص خطا و پیش‌بینی عمر مفید بیرینگ‌ها استفاده شوند.

```
def extract_features(matrix):
    # Compute various statistical features from the matrix
    features = {
        'standard deviation': stats.tstd(matrix, axis=1),
        'peak': np.max(matrix, axis=1),
        'skewness': stats.skew(matrix, axis=1),
        'mean': np.mean(matrix, axis=1),
        'absolute mean': np.mean(np.abs(matrix), axis=1),
        'root mean square': np.sqrt(np.mean(np.square(matrix),
axis=1)),
        'square root mean': np.square(np.mean(np.sqrt(np.abs(matrix)),
axis=1)),
        'kurtosis': stats.kurtosis(matrix, axis=1),
        'crest factor': np.max(matrix, axis=1) /
np.sqrt(np.mean(np.square(matrix), axis=1)),
```

```
        'clearance factor': np.max(matrix, axis=1) /
np.square(np.mean(np.sqrt(np.abs(matrix)), axis=1)),
        'peak to peak': np.max(matrix, axis=1) - np.min(matrix,
axis=1),
        'shape factor': np.sqrt(np.mean(np.square(matrix), axis=1)) /
np.mean(np.abs(matrix), axis=1),
        'impact factor': np.sqrt(np.mean(np.square(matrix), axis=1)) /
np.mean(np.abs(matrix), axis=1),
        'impulse factor': np.abs(np.mean(matrix, axis=1)) /
np.mean(np.abs(matrix), axis=1)
    }
    return features


# Extract features for each dataset and create dataframes
dfs = []
for label, matrix in matrices.items():
    features = extract_features(matrix)
    df = pd.DataFrame(features)
    df['label'] = 0 if 'normal' in label else 1
    dfs.append(df)
```

سپس طبق زیر دیتا ها به دیتافریم تبدیل می شوند

```
# Combine all dataframes
df = pd.concat(dfs, ignore_index=True)
```

در قسمت بعد دیتا ها به قسمت آموزش، آزمایش و اعتبار سنجی تقسیم می شوند که این کار طی دو مرحله انجام می شود. ابتدا ۷۵ درصد دیتا برای آموزش جدا می شود سپس از ۲۵ درصد باقی مانده ۱۵ درصد برای اعتبار سنجی و ۱۰ در صد برای آزمایش جدا می شود.

```
# Split data into training, validation, and test sets
train_ratio = 0.75
validation_ratio = 0.15
test_ratio = 0.10

# First split to separate training and the remaining data
x_train, x_temp, y_train, y_temp = train_test_split(
    df.drop('label', axis=1).values,
    df['label'].values,
    test_size=1 - train_ratio,
    random_state=34,
    shuffle=True
)

# Second split to separate validation and test data
x_val, x_test, y_val, y_test = train_test_split(
    x_temp,
```

```
    y_temp,
    test_size=test_ratio / (test_ratio + validation_ratio),
    random_state=34,
    shuffle=True
)
```

در ادامه دیتاها را نرمالایز می کنیم. از استاندارد اسکیلر استفاده می کنیم که همه دیتاها را به بین منفی یک و مثبت یک تبدیل می کند.

داده‌ها در یادگیری ماشین به سه دسته اصلی تقسیم می‌شوند: داده‌های آموزش ، داده‌های اعتبارسنجی و داده‌های آزمون هر یک از این دسته‌ها دارای نقش وظایف خاصی در مراحل مختلف آموزش و ارزیابی مدل‌های یادگیری ماشین هستند:

داده‌های آموزش :(Train Data)

این داده‌ها برای آموزش مدل استفاده می‌شوند. مدل با دیدن داده‌های آموزشی، الگوها و قوانین موجود در داده‌ها را یاد می‌گیرد.

هدف از استفاده از این داده‌ها، به دست آوردن پارامترهای مدل به گونه‌ای است که مدل بتواند به بهترین شکل ممکن بر روی داده‌های آموزشی عمل کند.

این داده‌ها باید به طور کافی و نمایان‌کننده‌ی مجموعه کل داده‌ها باشند.

داده‌های اعتبارسنجی :(Validation Data)

این داده‌ها برای ارزیابی عملکرد مدل و انتخاب بهترین مدل از بین مدل‌های مختلف استفاده می‌شوند.

هدف از استفاده از این داده‌ها، تنظیم پارامترهای مدل و انتخاب بهترین مدل بر اساس عملکرد آن بر روی داده‌های اعتبارسنجی است.

این داده‌ها نباید برای آموزش مدل استفاده شوند تا مدل به داده‌های آموزشی وابسته نشود.

داده‌های آزمون :(Test Data)

این داده‌ها برای ارزیابی نهایی عملکرد مدل انتخاب شده و اعتبارسنجی نهایی مدل استفاده می‌شوند.

هدف از استفاده از این داده‌ها، ارزیابی دقیق و قابل اعتماد عملکرد مدل نهایی بر روی داده‌هایی که قبلاً مدل آن‌ها را ندیده است، است.

این داده‌ها همچنین می‌توانند برای ارزیابی عملکرد مدل بر روی داده‌های جدید و واقعی در شرایط واقعی استفاده شوند.

با استفاده از این سه دسته داده، می‌توان مدل‌هایی را با دقت و کارآیی بالا آموزش داد و ارزیابی کرد که بر روی داده‌های جدید و واقعی نیز به خوبی عمل کنند.

```
# Standardize data
scaler = StandardScaler()
scaler.fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_val_scaled = scaler.transform(x_val)
x_test_scaled = scaler.transform(x_test)
```

۲-۲

در این قسمت از یک مدل mlp سه لایه استفاده شده است که چون دیتاها را بین منفی یک و یک نرمالایز کرده ایم برای تابع فعالساز لایه های پنهان از Tanh استفاده کرده ایم و همچنین تابع فعالساز لایه آخر softmax می باشد. مدل ما لرنینگ ریت 0.01 با اپتیمایزر sgd دارد.تعداد نورون های لایه های پنهان به ترتیب 10 و 7 می باشد.

در زیر کد مربوط به این بخش آورده شده است.

چون دیتای ما دارای ۴ کلاس است ما برای هر کلاس یک لیبل تعریف می کنیم:

```
# Extract features for each dataset and create dataframes
dfs = []
for label, matrix in matrices.items():
    features = extract_features(matrix)
    df = pd.DataFrame(features)
    if 'normal' in label:
        df['label'] = 0
    elif 'faulty1' in label:
        df['label'] = 1
    elif 'faulty2' in label:
        df['label'] = 2
    elif 'faulty3' in label:
        df['label'] = 3
    dfs.append(df)
```

میسینگ ولیو ها یا داده هایی که معلوم نیستند را با میانگین هر ستون پر میکنیم:

```
imputer = SimpleImputer(strategy='mean')
```

در انتها مدل را طبق زیر تعریف می کنیم:

```
# Create and train the MLP model manually with new optimizer and loss
function
mlp_model = MLPClassifier(hidden_layer_sizes=(10, 7),
activation='tanh', solver='sgd', learning_rate_init=0.01, max_iter=1,
warm_start=True, random_state=34)
```

در ادامه مدل را آموزش می دهیم که طی ۵۰ ایپاک آموزش میبیند و طی هر ایپاک خطا را در یک لیست ذخیره می کنیم که در ادامه نمودار آن را رسم کنیم:

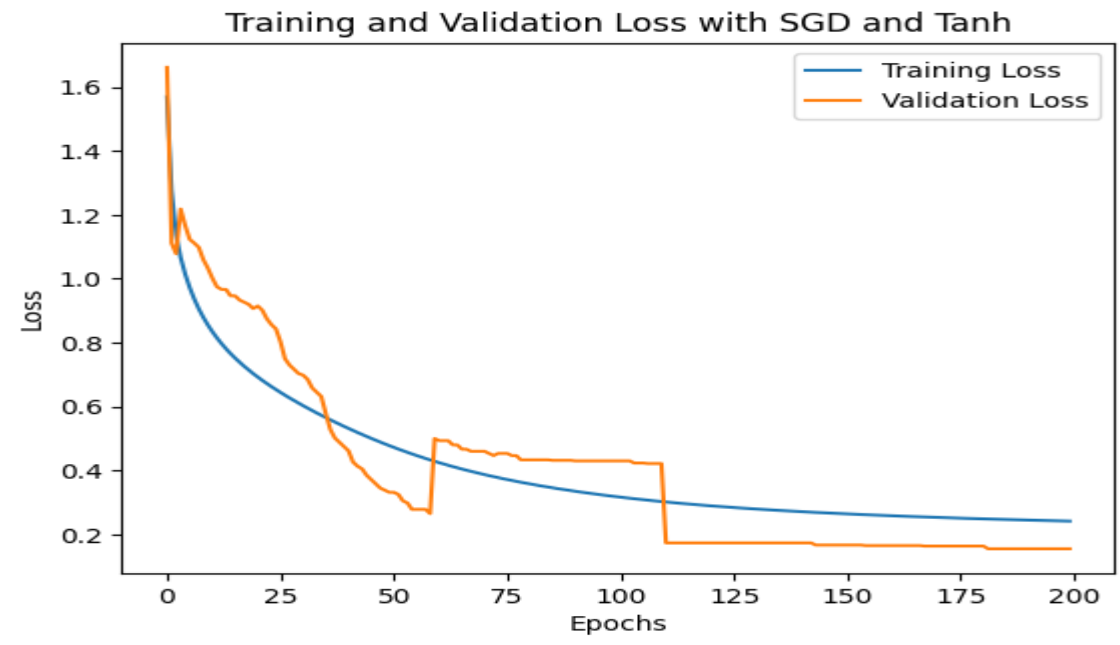
```
train_losses = []
val_losses = []
val_accuracies = []

for epoch in range(50):  # Train for 50 epochs
    mlp_model.fit(x_train_scaled_imputed, y_train)
    train_losses.append(mlp_model.loss_)

    # Calculate validation loss and accuracy
    val_preds = mlp_model.predict(x_val_scaled_imputed)
    val_accuracy = accuracy_score(y_val, val_preds)
    val_loss = np.mean((val_preds - y_val) ** 2)

    val_accuracies.append(val_accuracy)
    val_losses.append(val_loss)
```

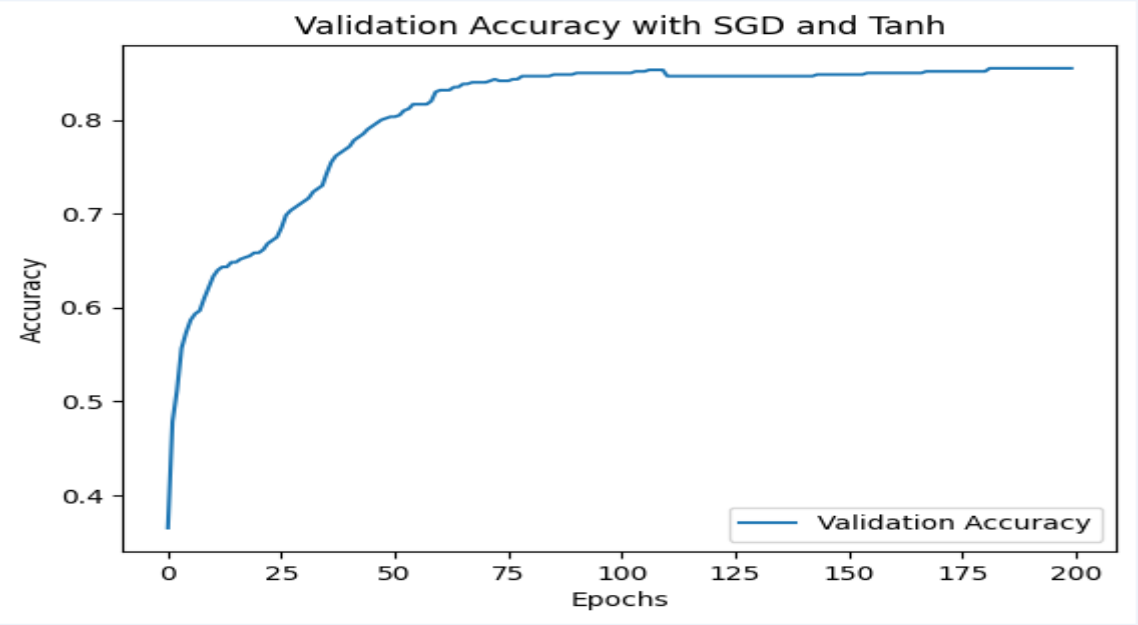
در انتها به بخش رسم خطاها و دقت و کانفیوژن ماتریکس دیتاهای آزمون می رسیم که طبق زیر است:

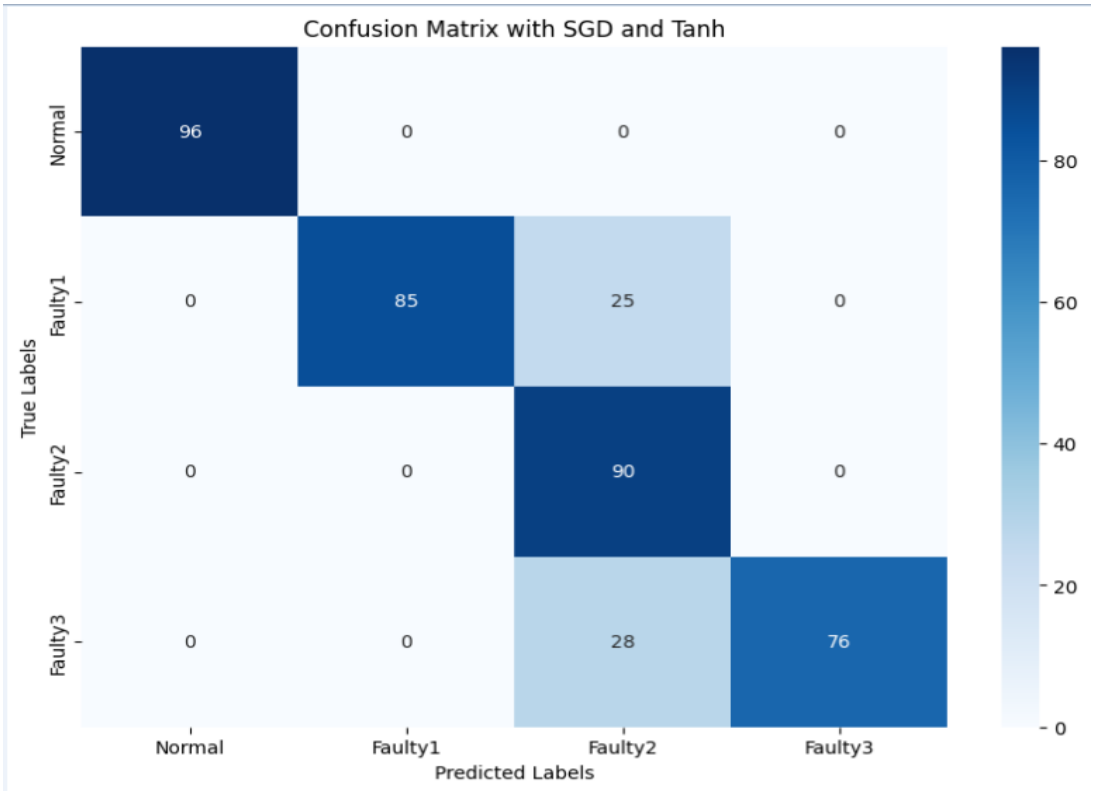
```
# Plotting training and validation loss
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.title('Training and Validation Loss with SGD and Tanh')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plotting validation accuracy
plt.plot(val_accuracies, label='Validation Accuracy')
plt.title('Validation Accuracy with SGD and Tanh')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Predictions on test set
test_preds = mlp_model.predict(x_test_scaled_imputed)
```

```
# Classification report
print("Classification Report for Test Data with SGD and Tanh:")
print(classification_report(y_test, test_preds))

# Confusion matrix
cm = confusion_matrix(y_test, test_preds)
print("Confusion Matrix for Test Data with SGD and Tanh:")
print(cm)

# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Normal', 'Faulty1', 'Faulty2', 'Faulty3'],
yticklabels=['Normal', 'Faulty1', 'Faulty2', 'Faulty3'])
plt.title('Confusion Matrix with SGD and Tanh')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Analysis
print("Final Train Accuracy with SGD and Tanh:",
accuracy_score(y_train, mlp_model.predict(x_train_scaled_imputed)))
print("Final Validation Accuracy with SGD and Tanh:",
accuracy_score(y_val, mlp_model.predict(x_val_scaled_imputed)))
print("Test Accuracy with SGD and Tanh:", accuracy_score(y_test,
test_preds))
```

نتایج به صورت زیر است:




Validation Accuracy with SGD and Tanh
Accuracy
0.8
0.7
0.6
0.5
0.4
0
25
50
75
100
125
150
175
200
Epochs
Validation Accuracy


Confusion Matrix with SGD and Tanh
True Labels
Normal
Faulty1
Faulty2
Faulty3
96
0
0
0
0
85
25
0
0
0
90
0
0
0
28
76
Normal
Faulty1
Faulty2
Faulty3
Predicted Labels
80
60
40
20
0

```
Classification Report for Test Data with SGD and Tanh:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        96
           1       1.00      0.77      0.87       110
           2       0.63      1.00      0.77        90
           3       1.00      0.73      0.84       104

    accuracy                           0.87       400
   macro avg       0.91      0.88      0.87       400
weighted avg       0.92      0.87      0.87       400

Confusion Matrix for Test Data with SGD and Tanh:
[[96  0  0  0]
 [ 0 85 25  0]
 [ 0  0 90  0]
 [ 0  0 28 76]]

Final Train Accuracy with SGD and Tanh: 0.8763333333333333
Final Validation Accuracy with SGD and Tanh: 0.855
Test Accuracy with SGD and Tanh: 0.8675
```

همانطور که طبق بالا مشخص است، با sgd طی ۲۰۰ ایپاک به دقت آزمون حدود ۸۷ درصد رسیده ایم.

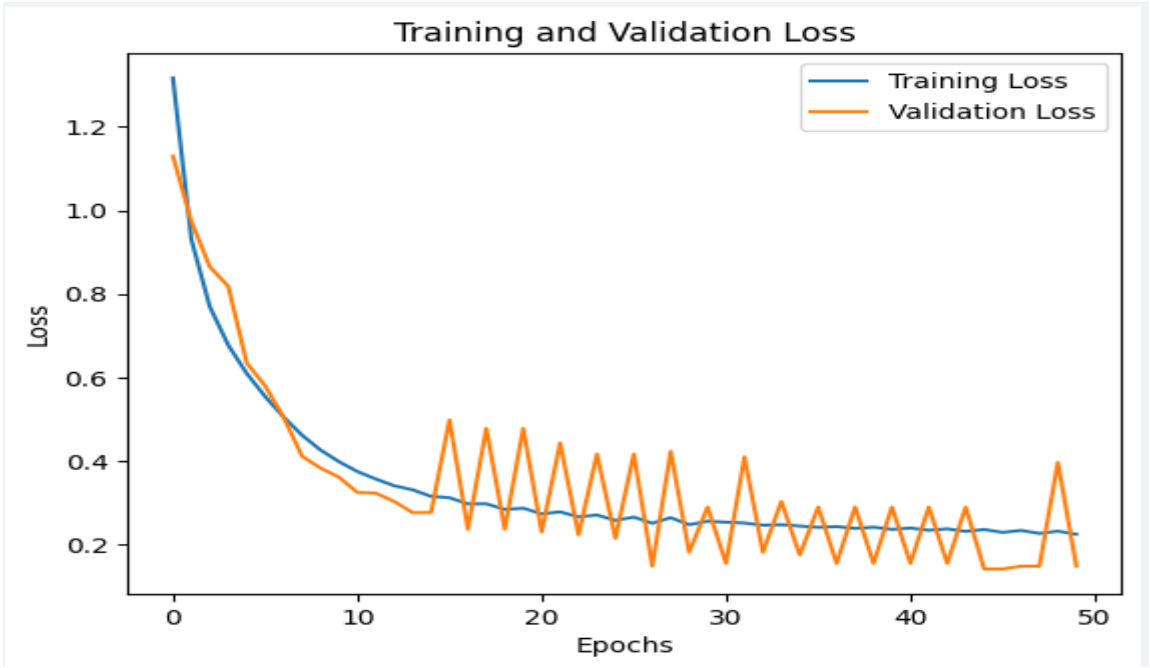
۲-۳

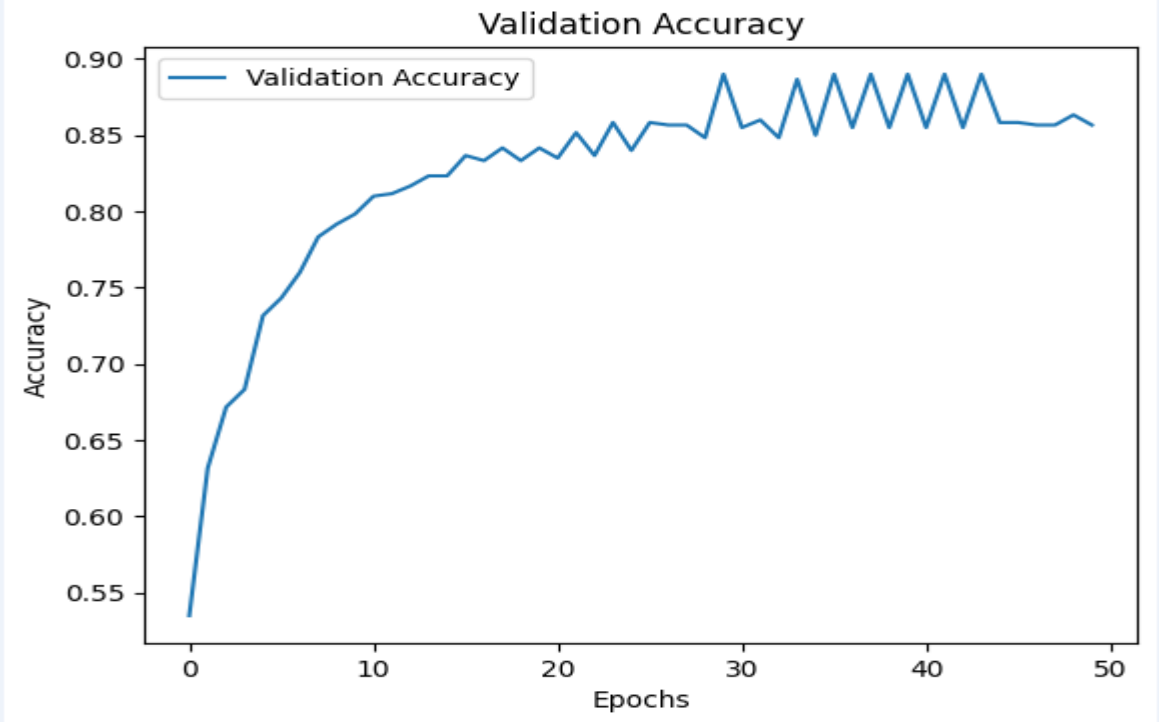
برای این بخش از دو اپتیمایزر adam و momentum استفاده شده است که مدل مانند زیر تعریف می شود:

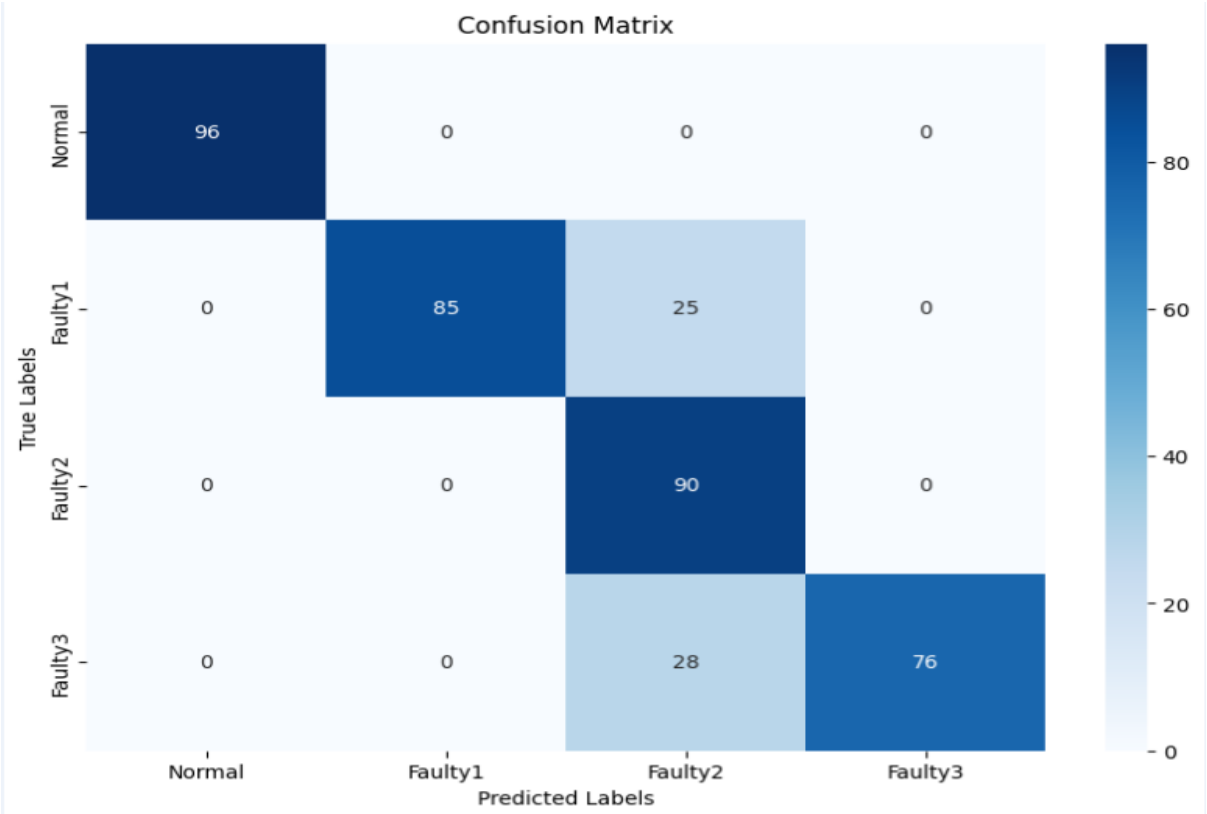
Adam with log loss:

```
# Create and train the MLP model manually
mlp_model = MLPClassifier(hidden_layer_sizes=(20, 10),
activation='tanh', solver='adam',learning_rate_init=0.01, max_iter=1,
warm_start=True, random_state=34)
```

همانطور که طبق زیر مشاهده می شود، این اپتیمایزر با ۵۰ ایپاک به دقت حدود ۸۷ درصد رسیده است.


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epochs
0
10
20
30
40
50
0.2
0.4
0.6
0.8
1.0
1.2


Validation Accuracy
Validation Accuracy
Accuracy
Epochs
0
10
20
30
40
50
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90




```
Classification Report for Test Data:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        96
           1       1.00      0.77      0.87       110
           2       0.63      1.00      0.77        90
           3       1.00      0.73      0.84       104

    accuracy                           0.87       400
   macro avg       0.91      0.88      0.87       400
weighted avg       0.92      0.87      0.87       400

Confusion Matrix for Test Data:
[[96  0  0  0]
 [ 0 85 25  0]
 [ 0  0 90  0]
 [ 0  0 28 76]]
Final Train Accuracy: 0.8786666666666667
Final Validation Accuracy: 0.8566666666666667
Test Accuracy: 0.8675
```

Adam with crossentropy loss:

ابتدا کتابخانه های مورد نیاز را فراخوانی میکنیم که اضافه بر کتابخانه های قبل کتابخانه torch نیز برای ساختن و آموزش شبکه عصبی فراخوانی شده است.

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

```
import pandas as pd
from scipy.io import loadmat
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.impute import SimpleImputer
import seaborn as sns
import torch
import torch.nn as nn
import torch.optim as optim
from scipy import stats  # Import scipy.stats
```

در ادامه دیتا لود شده و خوانده می شود سپس ورودی و تارگت جدا می شود.

```
# Constants
M, N = 250, 200
np.random.seed(25)
my_ID_Number = 34
dataset_url = {
    'normal':
'https://engineering.case.edu/sites/default/files/99.mat',
    'faulty1':
'https://engineering.case.edu/sites/default/files/107.mat',
    'faulty2':
'https://engineering.case.edu/sites/default/files/120.mat',
    'faulty3':
'https://engineering.case.edu/sites/default/files/132.mat'
}

# Functions
def load_data(urls):
    data = {}
    for label, url in urls.items():
        !wget -q {url}  # Quiet mode, no output
        data[label] = loadmat(url.split('/')[-1])  # Load and return
data
    return data


# Load all datasets
data = load_data(dataset_url)
cols = {key: list(val.keys())[-4:] for key, val in data.items()}
```

این تابع داده ها را در ماتریس ها سازماندهی می کند. ماتریسی از صفرها را راه‌اندازی می‌کند و آن را با داده‌های مجموعه داده‌ها پر می‌کند و مواردی را که پنجره داده‌ها از مرزها فراتر می‌رود، مدیریت می‌کند.

```
def organize_data(data, cols):
    all_matrices = {}
    for label, datasets in data.items():
        for col in cols[label]:
            mat = np.zeros((M, N))
            for j in range(M):
                try:
                    mat[j, :] = datasets[col][j:j+N].reshape(-1,)
                except Exception as e:
                    continue  # Handle case where window exceeds bounds
of data
            all_matrices[f"{label}_{col}"] = mat
    return all_matrices


# Organize data into matrices
matrices = organize_data(data, cols)
```

در ادامه این تابع ویژگی های آماری مختلفی را از هر ماتریس استخراج می کند، مانند انحراف معیار، اوج، چولگی، میانگین و موارد دیگر. فرهنگ لغت این ویژگی ها را برمی گرداند.

```
def extract_features(matrix):
    # Compute various statistical features from the matrix
    features = {
        'standard deviation': stats.tstd(matrix, axis=1),
        'peak': np.max(matrix, axis=1),
        'skewness': stats.skew(matrix, axis=1),
        'mean': np.mean(matrix, axis=1),
        'absolute mean': np.mean(np.abs(matrix), axis=1),
        'root mean square': np.sqrt(np.mean(np.square(matrix),
axis=1)),
        'square root mean': np.square(np.mean(np.sqrt(np.abs(matrix)),
axis=1)),
        'kurtosis': stats.kurtosis(matrix, axis=1),
        'crest factor': np.max(matrix, axis=1) /
np.sqrt(np.mean(np.square(matrix), axis=1)),
        'clearance factor': np.max(matrix, axis=1) /
np.square(np.mean(np.sqrt(np.abs(matrix)), axis=1)),
        'peak to peak': np.max(matrix, axis=1) - np.min(matrix,
axis=1),
        'shape factor': np.sqrt(np.mean(np.square(matrix), axis=1)) /
np.mean(np.abs(matrix), axis=1),
        'impact factor': np.sqrt(np.mean(np.square(matrix), axis=1)) /
np.mean(np.abs(matrix), axis=1),
```

```
        'impulse factor': np.abs(np.mean(matrix, axis=1)) /
np.mean(np.abs(matrix), axis=1)
    }
    return features


# Extract features for each dataset and create dataframes
dfs = []
for label, matrix in matrices.items():
    features = extract_features(matrix)
    df = pd.DataFrame(features)
    if 'normal' in label:
        df['label'] = 0
    elif 'faulty1' in label:
        df['label'] = 1
    elif 'faulty2' in label:
        df['label'] = 2
    elif 'faulty3' in label:
        df['label'] = 3
    dfs.append(df)

# Combine all dataframes
df = pd.concat(dfs, ignore_index=True)
```

در ادامه دیتا در دو مرحله به بخش آموزش و آزمون و اعتبارسنجی تقسیم شده و بعد شافل می شوند.

سپس برحسب میانگین و واریانس هر فیچر، نرمالایز شده و میسینگ دیتاها پر می شوند.

```
# Split data into training, validation, and test sets
train_ratio = 0.75
validation_ratio = 0.15
test_ratio = 0.10

# First split to separate training and the remaining data
x_train, x_temp, y_train, y_temp = train_test_split(
    df.drop('label', axis=1).values,
    df['label'].values,
    test_size=1 - train_ratio,
    random_state=34,
    shuffle=True
)

# Second split to separate validation and test data
x_val, x_test, y_val, y_test = train_test_split(
    x_temp,
    y_temp,
    test_size=test_ratio / (test_ratio + validation_ratio),
    random_state=34,
    shuffle=True
```

```
)

# Standardize data
scaler = StandardScaler()
scaler.fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_val_scaled = scaler.transform(x_val)
x_test_scaled = scaler.transform(x_test)

# Impute missing values
imputer = SimpleImputer(strategy='mean')
x_train_scaled_imputed = imputer.fit_transform(x_train_scaled)
x_val_scaled_imputed = imputer.transform(x_val_scaled)
x_test_scaled_imputed = imputer.transform(x_test_scaled)
```

سپس این بخش داده های از پیش پردازش شده را به تانسور PyTorch برای استفاده در شبکه عصبی تبدیل می کند.

```
# Convert data to PyTorch tensors
x_train_tensor = torch.tensor(x_train_scaled_imputed,
dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
x_val_tensor = torch.tensor(x_val_scaled_imputed, dtype=torch.float32)
y_val_tensor = torch.tensor(y_val, dtype=torch.long)
x_test_tensor = torch.tensor(x_test_scaled_imputed,
dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)
```

در زیر مدل شبکه عصبی در پایتورچ تعریف می شود.

```
# Define the neural network model
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(x_train_tensor.shape[1], 10)
        self.fc2 = nn.Linear(10, 7)
        self.fc3 = nn.Linear(7, 4)
        self.tanh = nn.Tanh()

    def forward(self, x):
        x = self.tanh(self.fc1(x))
        x = self.tanh(self.fc2(x))
        x = self.fc3(x)
        return x
```

این بخش نمونه ای از مدل شبکه عصبی را ایجاد می کند، تابع تلفات (آنتروپی متقابل برای وظایف طبقه بندی) را مشخص می کند و بهینه ساز (آدام) را تنظیم می کند.

```
# Instantiate the model, loss function, and optimizer
model = SimpleNN()
criterion = nn.CrossEntropyLoss()  # This is the default loss function
for classification
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

بخش بعد بخش آموزش مدل و اعتبار سنجی مدل است.

```
# Training loop
num_epochs = 200
train_losses = []
val_losses = []
val_accuracies = []

for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(x_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    loss.backward()
    optimizer.step()
    train_losses.append(loss.item())

    model.eval()
    with torch.no_grad():
        val_outputs = model(x_val_tensor)
        val_loss = criterion(val_outputs, y_val_tensor)
        val_losses.append(val_loss.item())

        _, val_preds = torch.max(val_outputs, 1)
        val_accuracy = accuracy_score(y_val_tensor.numpy(),
val_preds.numpy())
        val_accuracies.append(val_accuracy)
```

در ادامه خطای ترین و ولیدیشن و همچنین دقت ولیدیشن رسم می شوند.

```
# Plotting training and validation loss
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()
plt.show()

# Plotting validation accuracy
plt.plot(val_accuracies, label='Validation Accuracy')
plt.title('Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

سپس این بخش مدل را روی مجموعه تست ارزیابی می کند تا پیش بینی کند.

```
# Predictions on test set
model.eval()
with torch.no_grad():
    test_outputs = model(x_test_tensor)
    _, test_preds = torch.max(test_outputs, 1)
```

در نتیجه هم این بخش یک گزارش طبقه بندی و ماتریس سردرگمی را برای داده های آزمایش چاپ می کند. همچنین ماتریس سردرگمی را برای تجسم بهتر ترسیم می کند.

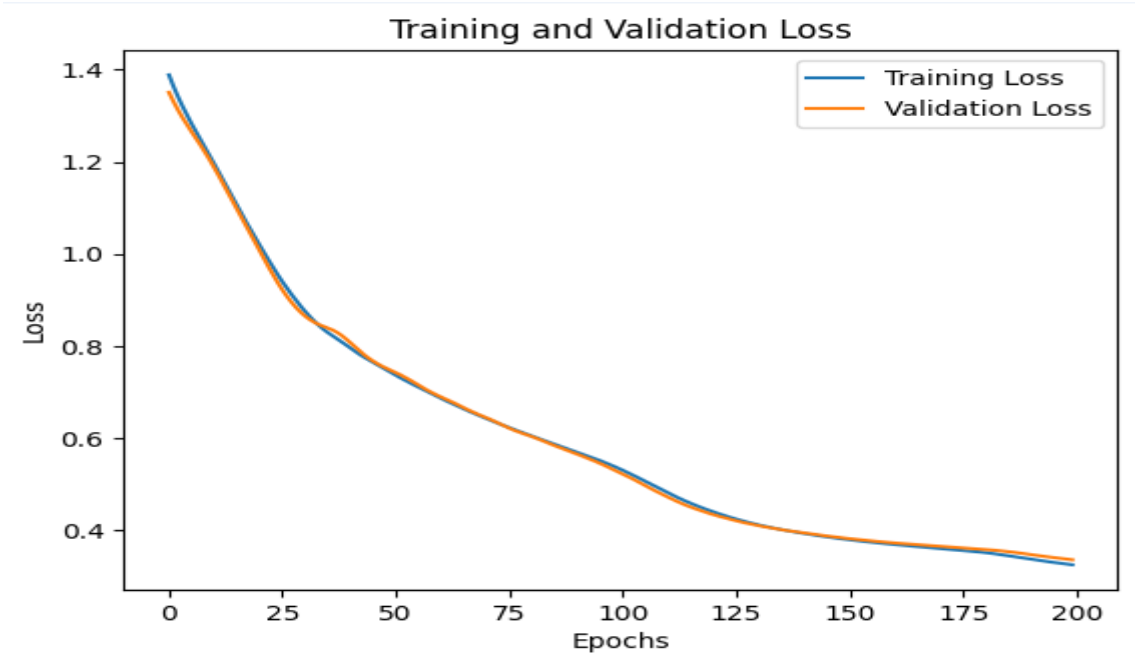
```
# Classification report
print("Classification Report for Test Data:")
print(classification_report(y_test_tensor.numpy(), test_preds.numpy()))

# Confusion matrix
cm = confusion_matrix(y_test_tensor.numpy(), test_preds.numpy())
print("Confusion Matrix for Test Data:")
print(cm)

# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Normal', 'Faulty1', 'Faulty2', 'Faulty3'],
yticklabels=['Normal', 'Faulty1', 'Faulty2', 'Faulty3'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

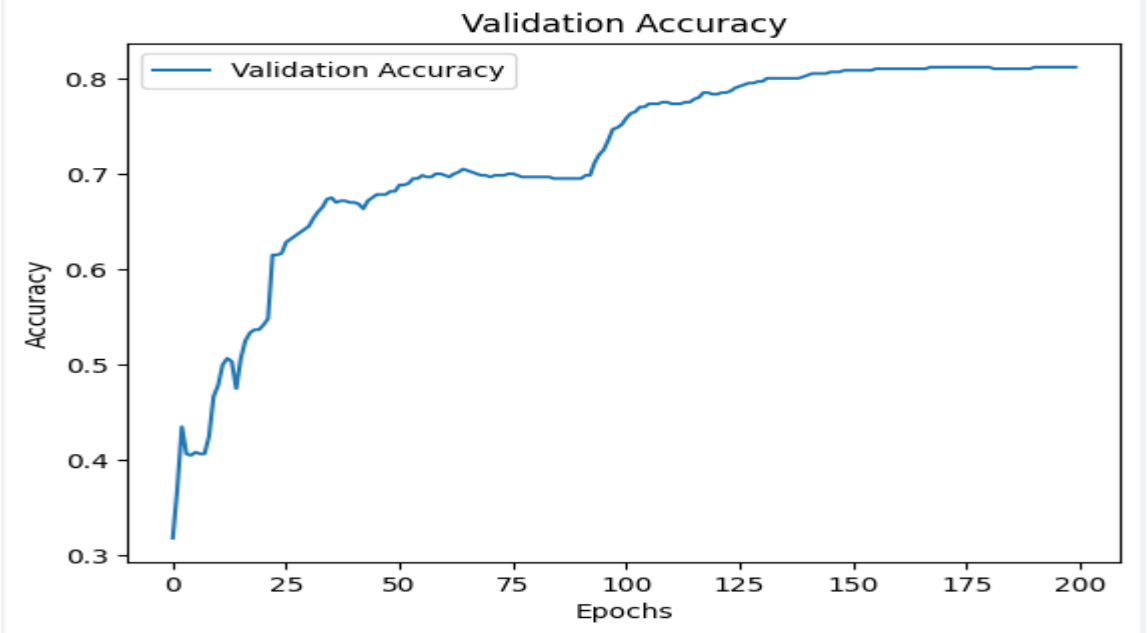
این بخش آموزش نهایی، اعتبار سنجی و دقت تست را محاسبه و چاپ می کند و خلاصه ای از عملکرد مدل را ارائه می دهد.

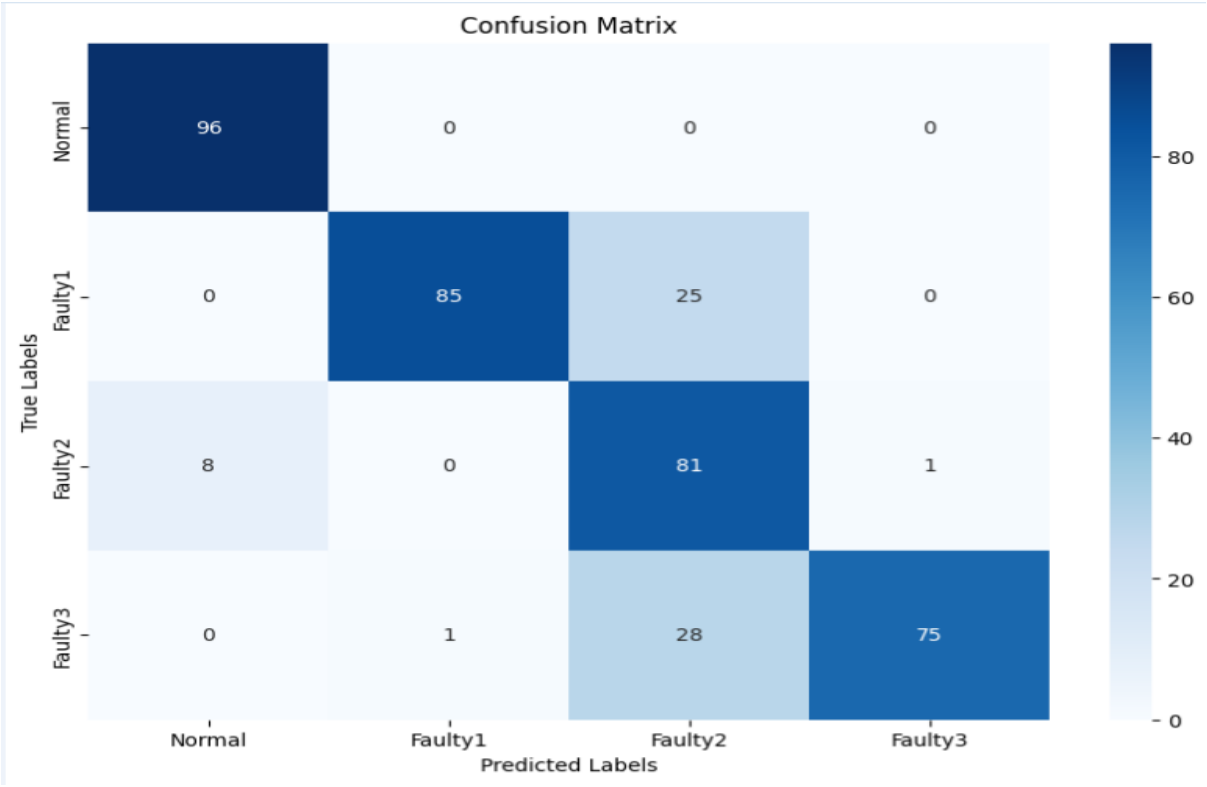
```
# Analysis
train_preds = model(x_train_tensor)
```

```
_, train_preds = torch.max(train_preds, 1)
print("Final Train Accuracy:", accuracy_score(y_train_tensor.numpy(),
train_preds.numpy()))

val_preds = model(x_val_tensor)
_, val_preds = torch.max(val_preds, 1)
print("Final Validation Accuracy:",
accuracy_score(y_val_tensor.numpy(), val_preds.numpy()))

print("Test Accuracy:", accuracy_score(y_test_tensor.numpy(),
test_preds.numpy()))
```

نتایج به صورت زیر است:








```
Classification Report for Test Data:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        96
           1       0.99      0.77      0.87       110
           2       0.60      0.90      0.72        90
           3       0.99      0.72      0.83       104

    accuracy                           0.84       400
   macro avg       0.88      0.85      0.85       400
weighted avg       0.89      0.84      0.85       400

Confusion Matrix for Test Data:
[[96  0  0  0]
 [ 0 85 25  0]
 [ 8  0 81  1]
 [ 0  1 28 75]]
```

```
Final Train Accuracy: 0.832
Final Validation Accuracy: 0.8116666666666666
Test Accuracy: 0.8425
```

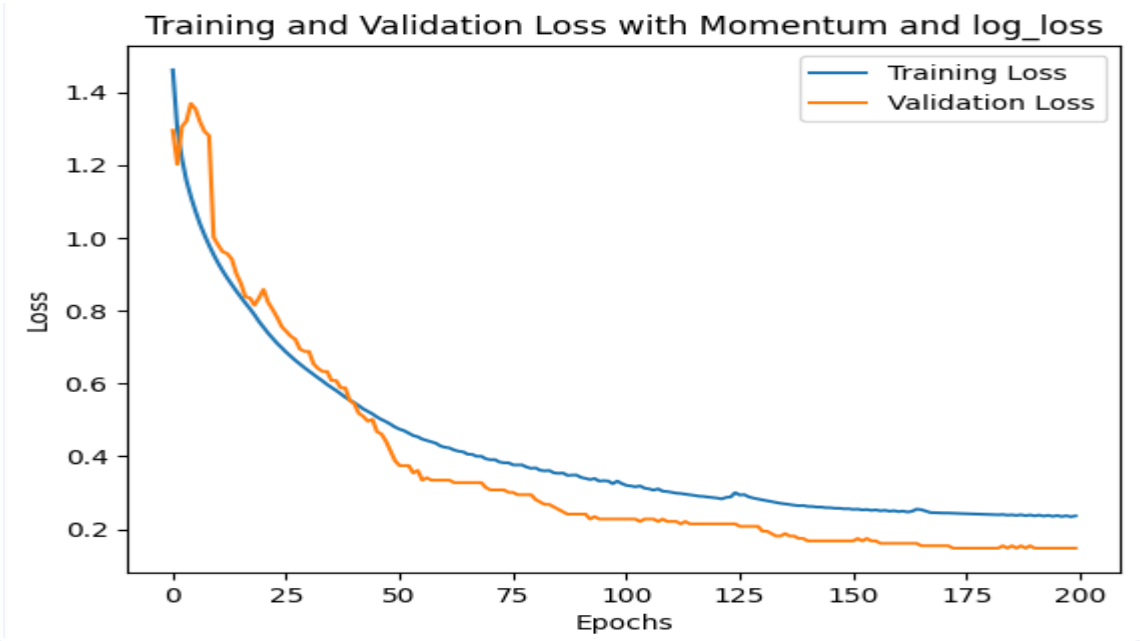
همانطور که مشاهده میکنیم طی ۲۰۰۲ ایپاک شبکه به دقت آزمون ۸۴ رسیده است.

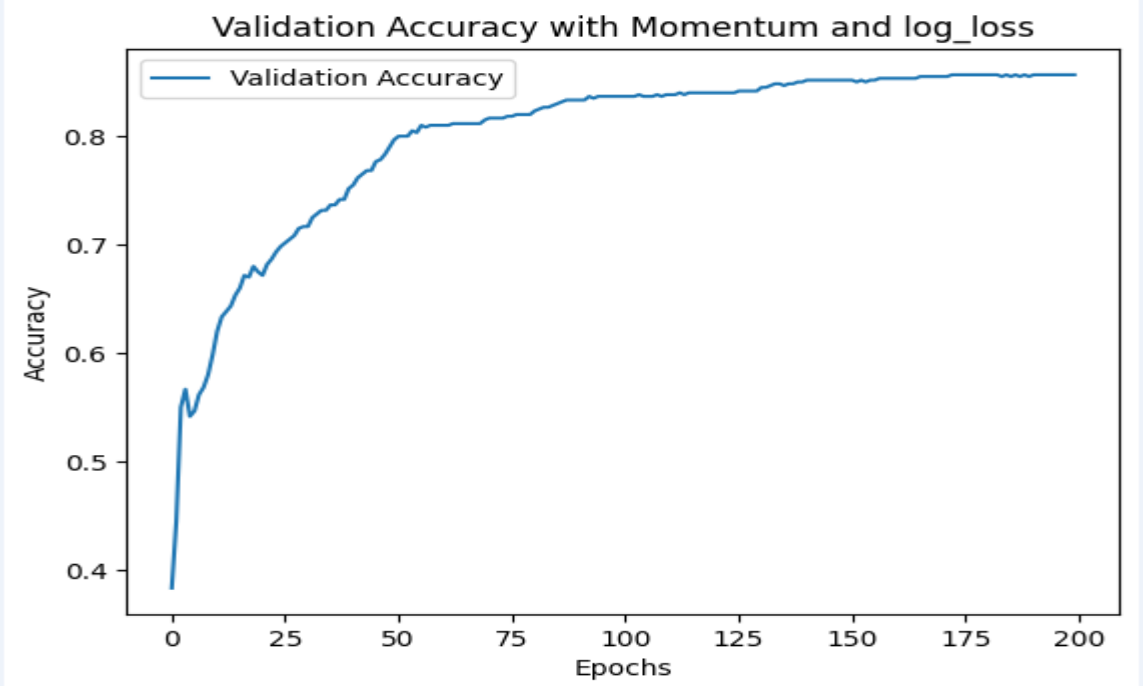
Momentum with log loss:

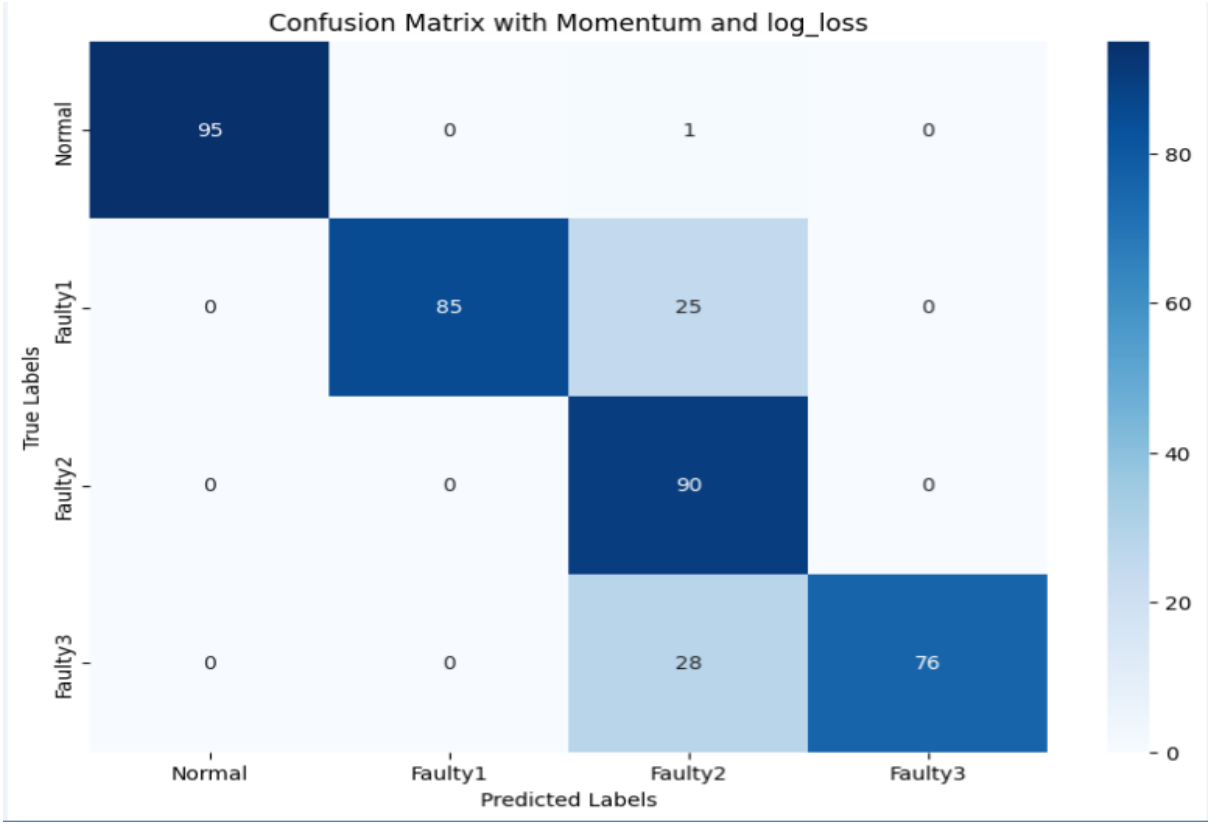
تنها فرق این قسمت با قسمت قبل این است که از اپتیمایزر ممنتم استفاده شده است بقیه قسمت های کد مانند ادام است.

```
# Create and train the MLP model manually with Momentum optimizer
mlp_model = MLPClassifier(hidden_layer_sizes=(10, 7),
activation='relu', solver='sgd', learning_rate_init=0.01,
momentum=0.98, max_iter=1, warm_start=True, random_state=34)
```

نتایج:








```
Classification Report for Test Data with Momentum and log_loss:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        96
           1       1.00      0.77      0.87       110
           2       0.62      1.00      0.77        90
           3       1.00      0.73      0.84       104

    accuracy                           0.86       400
   macro avg       0.91      0.87      0.87       400
weighted avg       0.92      0.86      0.87       400

Confusion Matrix for Test Data with Momentum and log_loss:
[[95  0  1  0]
 [ 0 85 25  0]
 [ 0  0 90  0]
 [ 0  0 28 76]]

Final Train Accuracy with Momentum and log_loss: 0.878
Final Validation Accuracy with Momentum and log_loss: 0.8566666666666667
Test Accuracy with Momentum and log_loss: 0.865
```

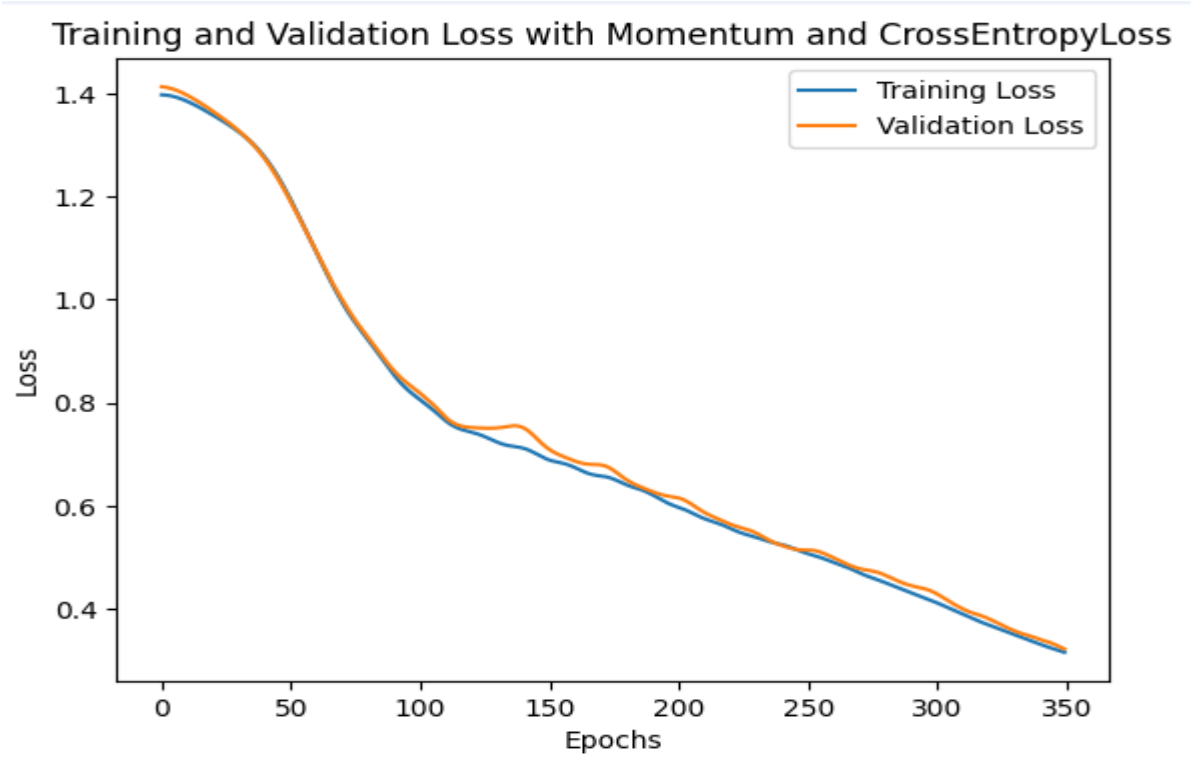
همانطور که مشخص است دقت آزمون در این بخش به حدود 87 درصد رسیده است.

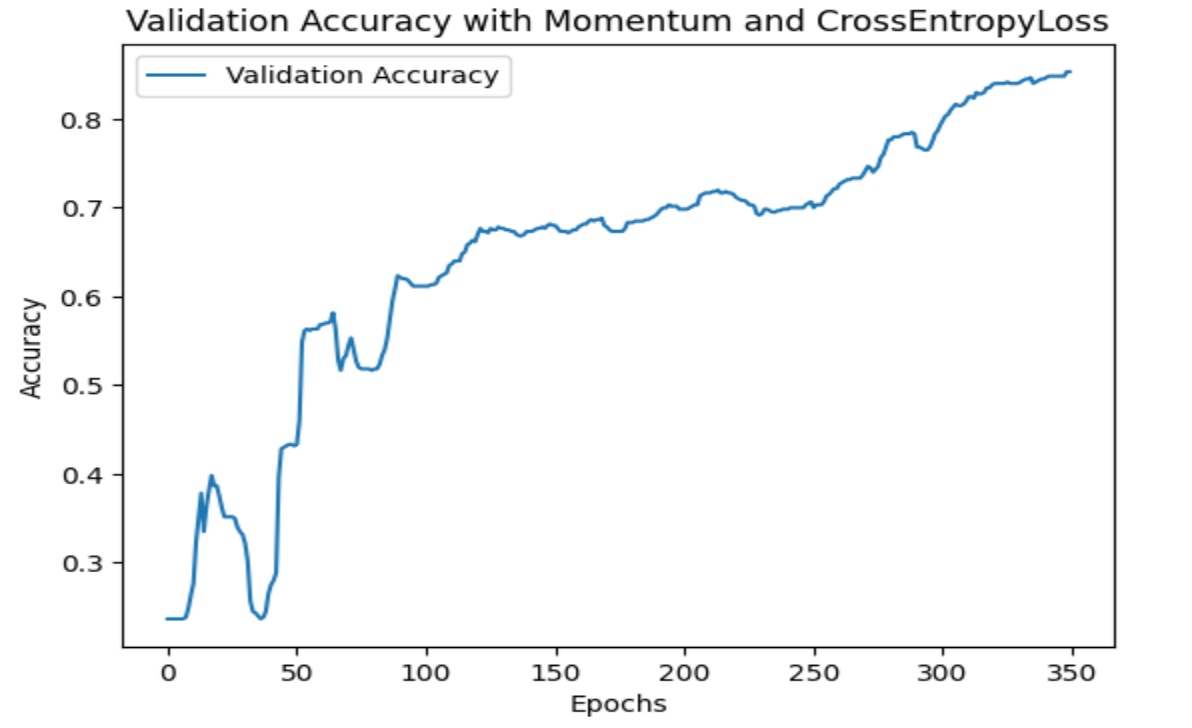
Momentum with crossentropy loss:

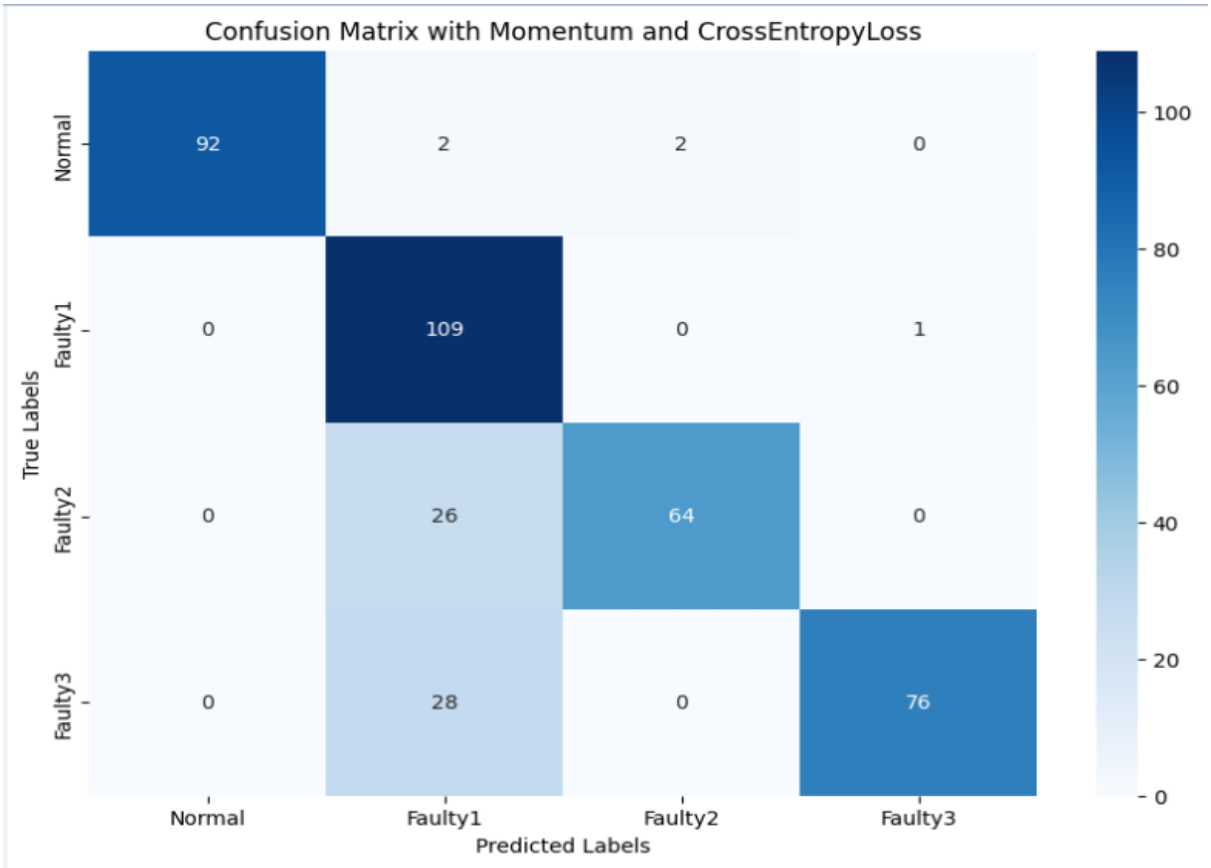
همه ی قسمت های این بخش نیز مانند قسمت adam با تابع اتلاف کراس انتروپی می باشد فقط تنها فرق در اپتیمایزر است.

```
# Instantiate the model, loss function, and optimizer
model = SimpleNN()
criterion = nn.CrossEntropyLoss()  # This is the default loss function
for classification
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.98)
```

نتایج:








```
Classification Report for Test Data with Momentum and CrossEntropyLoss:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        96
           1       0.66      0.99      0.79       110
           2       0.97      0.71      0.82        90
           3       0.99      0.73      0.84       104

    accuracy                           0.85       400
   macro avg       0.90      0.85      0.86       400
weighted avg       0.90      0.85      0.86       400

Confusion Matrix for Test Data with Momentum and CrossEntropyLoss:
[[ 92   2   2   0]
 [  0 109   0   1]
 [  0  26  64   0]
 [  0  28   0  76]]

Final Train Accuracy with Momentum and CrossEntropyLoss: 0.8603333333333333
Final Validation Accuracy with Momentum and CrossEntropyLoss: 0.8533333333333334
Test Accuracy with Momentum and CrossEntropyLoss: 0.8525
```

همانطور که از نتایج بالا مشخص است، نتیجه میگیریم که ممنتم با تعداد ایپاک بیشتری به دقتی نزدیک دقت آدام رسیده است و عملکرد آدام نسبت به ممنتم بهتر بوده است.

۲-۴

دراین قسمت می آییم و روی اپتیمایزر adam که جواب بهتری نسبت به بقیه داشت مراحل را انجام می دهیم:

1. K-Fold Cross-Validation:

K-Fold Cross-Validation یک تکنیک ارزیابی مدل است که به منظور بررسی مدل عملکرد بر روی داده‌های ناشناخته استفاده می‌شود. در این روش:

داده‌ها به k بخش (fold) تقسیم می‌شوند.

در هر تکرار ، یکی از این بخش‌ها به عنوان داده‌های آزمون (test) و بقیه به عنوان داده‌های آموزش (train) استفاده می‌شود.

این فرآیند k بار تکرار می‌شود و هر بخش یک بار به عنوان داده‌های آزمون استفاده می‌شود.

در نهایت، عملکرد مدل در k تکرار به عنوان عملکرد نهایی مدل گزارش می‌شود.

2 .Stratified K-Fold Cross-Validation:

Stratified K-Fold Cross-Validation یک نسخه بهبود یافته از K-Fold Cross-Validation است که کلاس‌ها را در هر بخش حفظ می‌کند. در این روش:

داده‌ها به k تقسیم می‌شوند، به طوری که نمونه‌های هر کلاس در هر بخش مشابه کل داده‌ها باشد.

این تکنیک برای داده‌ها با استفاده از نامتوازن کلاس‌ها بسیار مفید است، زیرا تضمین می‌کند که هر نماینده مناسبی از کل داده‌ها باشد.

انتخاب روش:

با توجه به اینکه داده‌های ما ممکن است دارای نامتوازن کلاس‌ها باشند، استفاده از Stratified K-Fold Cross-Validation توصیه می‌شود. این روش تضمین می‌کند که هر کدام از نمایندگان مناسبی از کل داده‌ها داده می‌شود و باید از عملکرد مدل ارائه شده دقیق‌تر شود.

طبق مراحل قبل ابتدا کتابخانه های مورد نیاز ایمپورت می شوند و دیتا فراخوانی میشود سپس دیتا دانلود شده و به دیتا فریم تبدیل می شود و فیچرهای آن استخراج می شود و اسپلیت انجام می گیرد. در این بخش فقط قسمت جدید کد را توضیح می دهیم.

دورهای تقسیم بندی و آموزش:

skf.split(X_scaled_imputed, y) داده‌های X_scaled_imputed و y را به ۴ بخش تقسیم می‌کند. در هر تکرار، train_index و val_index شاخص‌های مربوط به داده‌های آموزشی و اعتبارسنجی را برمی‌گرداند.

x_train, x_val: داده‌های آموزشی و اعتبارسنجی برای ویژگی‌ها.

y_train, y_val: داده های آموزشی و اعتبارسنجی برای برچسب ها.

ایجاد و آموزش مدل:

MLPClassifier: یک مدل شبکه عصبی چند لایه (MLP) با دو لایه مخفی (۲۰ نورون و ۱۰ نورون) و تابع فعال‌ساز tanh ایجاد می‌شود. الگوریتم بهینه سازی آدم با ارزیابی اولیه ۰,۰۱ و تکرار ۵۰ استفاده می‌شود.

mlp_model.fit(x_train, y_train): مدل با استفاده از داده‌های آموزشی آموزش داده می‌شود.

پیش بینی و ارزیابی مدل:

val_preds = mlp_model.predict(x_val) : مدل داده‌های اعتبارسنجی را پیش‌بینی می‌کند.

val_accuracy = accuracy_score(y_val, val_preds): دقت پیش‌بینی‌های مدل بر روی داده‌های اعتبارسنجی محاسبه می‌شود.

val_loss = np.mean((val_preds - y_val) ** 2): خطای مربعی میان پیش‌بینی‌ها و برچسب‌های واقعی محاسبه می‌شود.

نتایج ذخیره شده:

val_accuracies.append(val_accuracy): دقت اعتبارسنجی به لیست val_accuracies اضافه می‌شود.

val_losses.append(val_loss): خطای اعتبارسنجی به لیست val_losses اضافه می‌شود. سپس میانگین دقت ها و خطاهای ولیدیشن را پرینت می کنیم.

```
# Stratified K-Fold Cross-Validation
skf = StratifiedKFold(n_splits=4)
val_accuracies = []
val_losses = []

for train_index, val_index in skf.split(X_scaled_imputed, y):
    x_train, x_val = X_scaled_imputed[train_index],
X_scaled_imputed[val_index]
    y_train, y_val = y[train_index], y[val_index]

    mlp_model = MLPClassifier(hidden_layer_sizes=(10, 7),
activation='tanh', solver='adam', learning_rate_init=0.01, max_iter=50,
random_state=34)
    mlp_model.fit(x_train, y_train)

    val_preds = mlp_model.predict(x_val)
    val_accuracy = accuracy_score(y_val, val_preds)
    val_loss = np.mean((val_preds - y_val) ** 2)

    val_accuracies.append(val_accuracy)
    val_losses.append(val_loss)

# Report average accuracy and loss
print("Average Validation Accuracy:", np.mean(val_accuracies))
print("Average Validation Loss:", np.mean(val_losses))
```

در این بخش بطور کلی:

تقسیم‌بندی‌ها: داده‌ها به ۴ بخش تقسیم می‌شوند، به طوری که هر نماینده‌ای نماینده از کل داده‌ها باشد.

آموزش و اعتبارسنجی: مدل برای هر یک از بخش ۴ آموزش داده می‌شود و بر روی آن ارزیابی می‌شود.

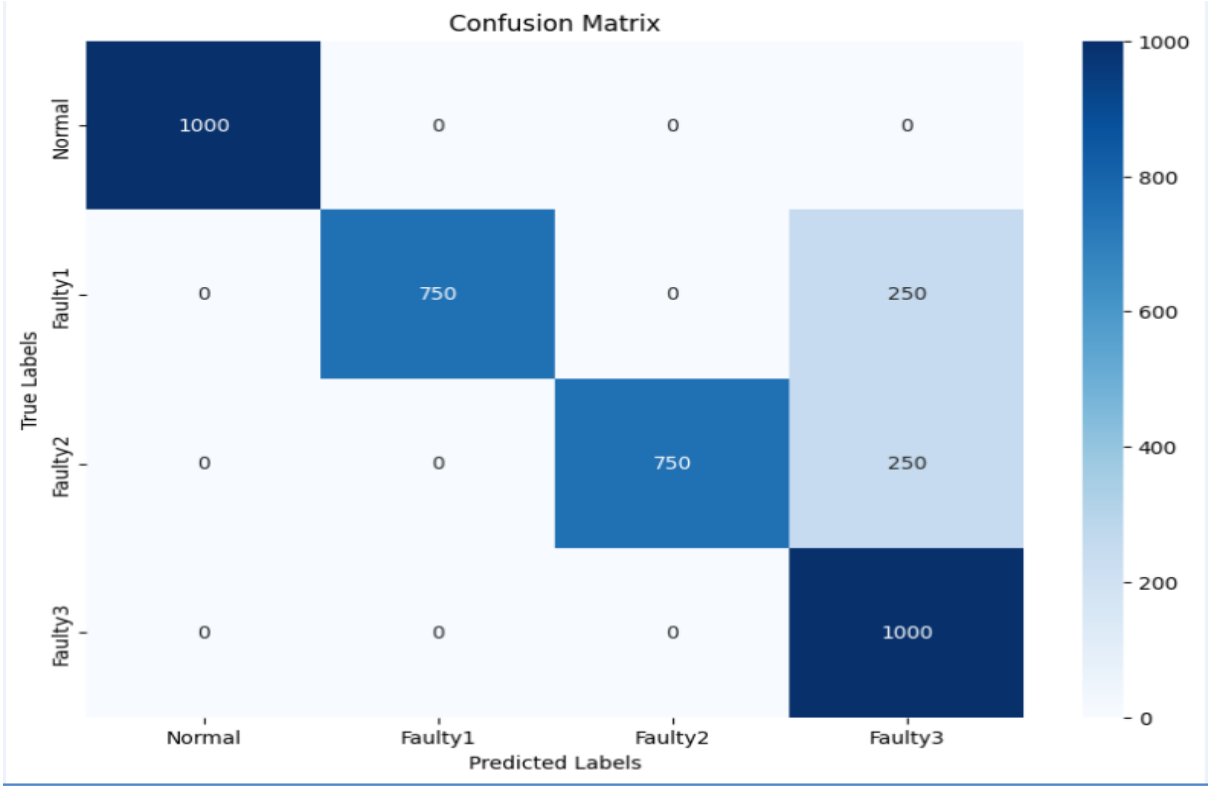
نتایج جمع آوری: دقت و خطای مدل در هر بار اعتبارسنجی جمع آوری می شود.

محاسبه میانگین: میزان دقت و خطاهای گزارش ارزیابی به عنوان کیفیت نهایی عملکرد مدل می‌شود.

این روش تضمین می‌کند که مدل به طور جامع و دقیق ارزیابی می‌شود و از تمام داده‌ها برای ارزیابی استفاده می‌شود، در حالی که نسبت‌های کلاس‌ها در هر بخش حفظ می‌شود. این به ویژه در مواردی که نامتوازن داده‌ها مفید هستند و به ارزیابی‌های قابل اعتماد از عملکرد مدل کمک می‌کند.

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

           0       1.00      1.00      1.00      1000
           1       1.00      0.75      0.86      1000
           2       1.00      0.75      0.86      1000
           3       0.67      1.00      0.80      1000

    accuracy                           0.88      4000
   macro avg       0.92      0.88      0.88      4000
weighted avg       0.92      0.88      0.88      4000

Confusion Matrix:
[[1000    0    0    0]
 [   0  750    0  250]
 [   0    0  750  250]
 [   0    0    0 1000]]
```

```
Final Model Accuracy: 0.875
```



همانطور که میبینیم، دقت نسبت به روشهای قبل کمی بهتر شده است.

۳ سوال سوم

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

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

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

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

- sklearn.ensemble.RandomForestClassifier
- sklearn.ensemble.AdaBoostClassifier

اگر به دقت کلی آزمون زیر ۸۰ درصد رسیده‌اید یا تحلیل درخت تصمیم به‌صورت دستی برایتان مشکل شده است لازم است با ذکر توضیحات، پیاده‌سازی‌هایی علاوه بر پیاده‌سازی‌های قبلی و با فراپارامترهای جدید جهت حل این مشکلات انجام دهید. هم‌چنین می‌توانید حداقل چهار فراپارامتر برای درخت تصمیم خود در نظر بگیرید و این فراپارامترها را با روش‌هایی مانند GridSearch بهینه کنید.

۳-۱

مجموعه داده "Forest Cover Type" از مجموعه داده‌های معروف و رایج در زمینه یادگیری ماشین است که برای مسئله طبقه‌بندی استفاده می‌شود. این مجموعه داده توسط سازمان زمین‌شناسی ایالات متحده (USGS)فراهم شده و شامل اطلاعات مربوط به نوع پوشش جنگلی مناطق مختلف در منطقه راواهل، کلرادو است. این مجموعه داده به طور گسترده در پژوهش‌های مرتبط با یادگیری ماشین و داده‌کاوی استفاده می‌شود.

ویژگی‌های مجموعه داده

این مجموعه داده شامل ۵۸۱۰۱۲ نمونه است که هر نمونه دارای ۵۴ ویژگی می‌باشد. ویژگی‌های این مجموعه داده به دو دسته ویژگی‌های عددی (continuous) و ویژگی‌های گسسته (categorical) تقسیم می‌شوند.

ویژگی‌های عددی:(Continuous)

Elevation: ارتفاع از سطح دریا (به متر)

Aspect: جهت (به درجه)

Slope: شیب (به درجه)

Horizontal Distance To Hydrology: فاصله افقی تا نزدیک‌ترین منبع آبی (به متر)
Vertical Distance To Hydrology: فاصله عمودی تا نزدیک‌ترین منبع آبی (به متر)
Horizontal Distance To Roadways: فاصله افقی تا نزدیک‌ترین جاده (به متر)
Hillshade 9am: سایه‌ی تپه ساعت ۹ صبح (مقدار بین ۰ تا ۲۵۵)
Hillshade Noon: سایه‌ی تپه ظهر (مقدار بین ۰ تا ۲۵۵)
Hillshade 3pm: سایه‌ی تپه ساعت ۳ بعد از ظهر (مقدار بین ۰ تا ۲۵۵)
Horizontal Distance To Fire Points:فاصله افقی تا نزدیک‌ترین نقطه آتش‌سوزی (به متر)

ویژگی‌های گسسته:(Categorical)

Wilderness Area: چهار منطقه وحشی که به صورت متغیرهای دامی (dummy variables) ارائه شده‌اند.

Soil Type: چهل نوع مختلف خاک که به صورت متغیرهای دامی ارائه شده‌اند.

برچسب(Target)

برچسب این مجموعه داده، نوع پوشش جنگلی است که به صورت اعداد ۱ تا ۷ کدگذاری شده است:

Spruce/Fir: صنوبر/نراد
Lodgepole Pine: کاج لاج‌پول
Ponderosa Pine: کاج پاندوروسا
Cottonwood/Willow: پنبه‌چوب/بید
Aspen: صنوبر
Douglas-fir: داگلاس-نراد
Krummholz: جنگل‌های خمیده و درهم

ما در این سوال از کل داده ها استفاده کرده ایم.

توضیح روش انتخاب بخشی از داده‌ها

برای انتخاب بخشی از داده‌ها به منظور تقسیم به دو بخش آموزش و آزمون، از روش تقسیم تصادفی طبقه‌بندی شده (Stratified Random Split) استفاده شده است. این روش به کمک تابع train_test_split از کتابخانه sklearn پیاده‌سازی شده است.

روش استفاده شده Stratified Random Split:

در این روش، داده‌ها به صورت تصادفی به دو بخش تقسیم می‌شوند، اما به گونه‌ای که نسبت کلاس‌ها در هر دو بخش آموزش و آزمون حفظ می‌شود. این روش به خصوص برای مجموعه داده‌هایی که دارای کلاس‌های نامتوازن هستند بسیار مفید است. در کد زیر از این روش استفاده شده است:

```
from sklearn.model_selection import train_test_split
```

تقسیم داده‌ها به دو بخش آموزش و آزمون

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15,
random_state=34, stratify=y)
```

در اینجا:

X و y داده‌ها و برچسب‌های آنها هستند.

test_size=0.15 نشان می‌دهد که ۱۵٪ از داده‌ها به عنوان داده‌های آزمون و ۸۵٪ به عنوان داده‌های آموزش در نظر گرفته می‌شوند.

random_state=34 برای تضمین بازتولیدپذیری نتایج استفاده شده است.

stratify=y تضمین می‌کند که نسبت کلاس‌ها در هر دو بخش آموزش و آزمون حفظ شود.

مزایای روش Stratified Random Split

حفظ نسبت کلاس‌ها: این روش اطمینان می‌دهد که نسبت کلاس‌ها در مجموعه‌های آموزش و آزمون مشابه باشد و از بروز عدم تعادل در کلاس‌ها جلوگیری می‌کند.

بازتولیدپذیری: با تنظیم random_state می‌توان نتایج را تکرار کرد.

سادگی و کارایی: این روش به راحتی قابل پیاده‌سازی است و به طور گسترده در مسائل طبقه‌بندی استفاده می‌شود.

روش‌های دیگر برای تقسیم داده‌ها

علاوه بر روش Stratified Random Split، روش‌های دیگری نیز برای تقسیم داده‌ها وجود دارند که بسته به شرایط و نوع داده‌ها ممکن است مناسب‌تر باشند:

K-Fold Cross-Validation:

در این روش، داده‌ها به k بخش مساوی تقسیم می‌شوند و مدل به تعداد k بار آموزش داده می‌شود، هر بار یک بخش به عنوان داده‌های آزمون و بقیه به عنوان داده‌های آموزش استفاده می‌شود. این روش به بهبود ارزیابی مدل کمک می‌کند.

Stratified K-Fold Cross-Validation:

این روش مشابه K-Fold Cross-Validation است با این تفاوت که نسبت کلاس‌ها در هر کدام از k بخش حفظ می‌شود.

Time Series Split:

اگر داده‌ها به ترتیب زمانی مرتب شده باشند (مانند داده‌های سری زمانی)، می‌توان از این روش استفاده کرد که در آن داده‌ها به ترتیب زمانی به بخش‌های آموزش و آزمون تقسیم می‌شوند.

انتخاب روش بهینه

انتخاب بهترین روش برای تقسیم داده‌ها بستگی به نوع داده‌ها و مسئله مورد نظر دارد. در اینجا روش Stratified Random Split انتخاب شده است زیرا مجموعه داده دارای کلاس‌های مختلفی است و حفظ نسبت کلاس‌ها در هر دو بخش آموزش و آزمون اهمیت دارد. اگر داده‌ها دارای ترتیب زمانی بودند یا ارزیابی دقیق‌تری از مدل نیاز بود، می‌توانستیم از روش‌های دیگری مانند K-Fold Cross-Validation استفاده کنیم.

برای کد ابتدا کتابخانه های مورد نیاز فراخوانی می شوند سپس طبق زیر مجموعه داده "Forest Cover Type" از کتابخانه sklearn.datasets بارگذاری می‌شود. این مجموعه داده شامل اطلاعات مربوط به نوع پوشش جنگلی مناطق مختلف است. ویژگی‌های داده‌ها در متغیر X و برچسب‌ها (نوع پوشش جنگلی) در متغیر y ذخیره می‌شوند.

```
# Load the dataset
data = fetch_covtype()
X, y = data.data, data.target
```

در مرحله بعد برای تقسیم داده‌ها از تابع train_test_split استفاده شده است. ٪۸۵ داده‌ها برای آموزش و ۱۵٪ داده‌ها برای آزمون در نظر گرفته شده‌اند. پارامتر stratify=y تضمین می‌کند که نسبت کلاس‌ها در هر دو بخش آموزش و آزمون حفظ شود. این کار از بروز عدم تعادل در کلاس‌ها جلوگیری می‌کند.

```
# Split the d ata into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.15, random_state=34, stratify=y)
```

در ادامه ک مدل درخت تصمیم با استفاده از کلاس DecisionTreeClassifier ایجاد شده و بر روی داده‌های آموزش (X_train) و (y_train) آموزش داده می‌شود.

```
# Create and train the decision tree classifier
clf = DecisionTreeClassifier(random_state=34)
clf.fit(X_train, y_train)
```

پس از آموزش مدل، برچسب‌های داده‌های آزمون (X_test) پیش‌بینی می‌شوند و نتایج پیش‌بینی شده در متغیر y_pred ذخیره می‌شوند.

```
# Predict the labels for the test set
y_pred = clf.predict(X_test)
```

برای ارزیابی مدل، دقت (Accuracy) محاسبه شده و گزارش طبقه‌بندی (Classification Report) که شامل معیارهای مختلفی مانند دقت(Precision) ، فراخوانی (Recall) و امتیاز F1 (F1-score) برای هر کلاس است، نمایش داده می‌شود.

```
# Evaluate the model
```

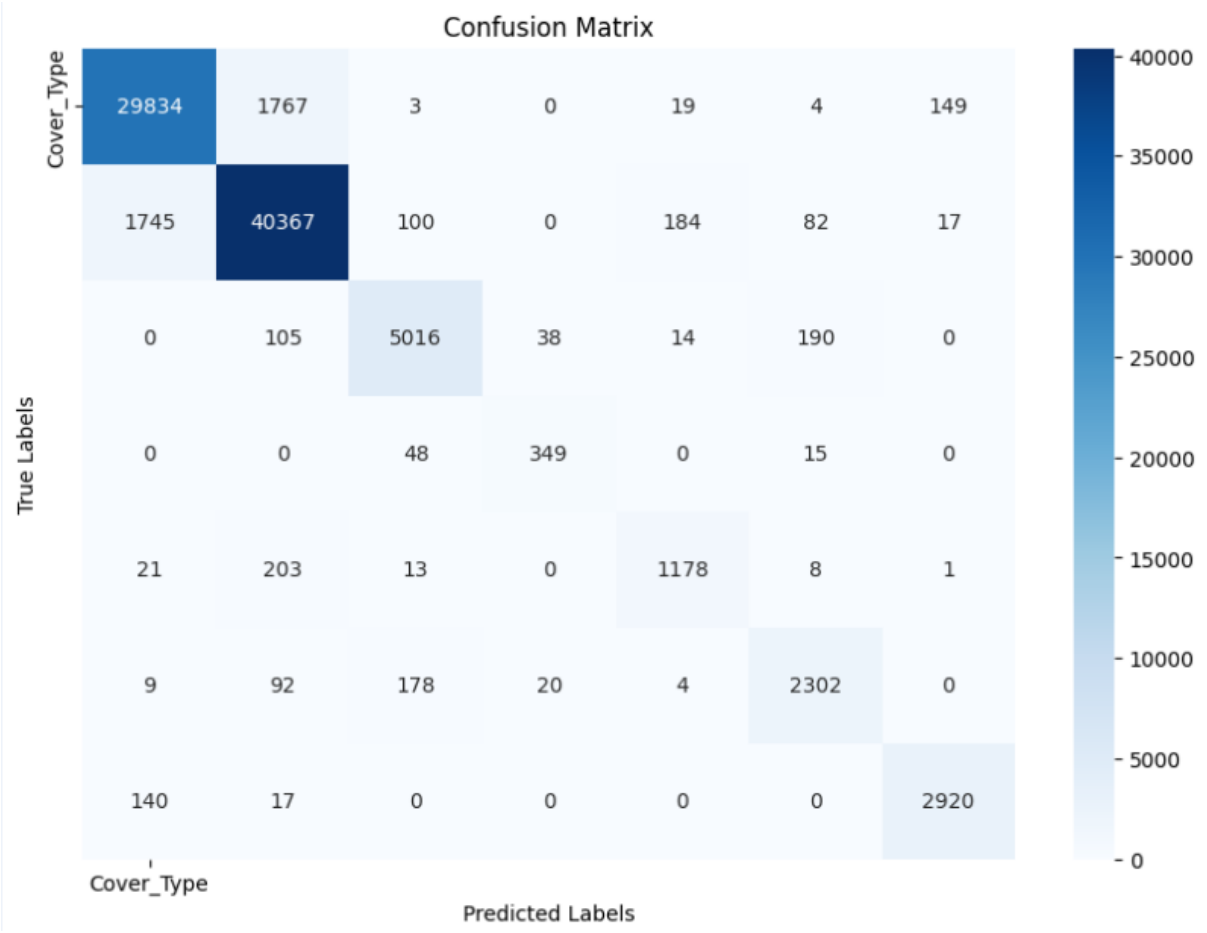
```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

ماتریس درهم‌ریختگی (Confusion Matrix) محاسبه شده و نمایش داده می‌شود. این ماتریس نشان می‌دهد که مدل به چه تعداد از نمونه‌های هر کلاس به درستی و نادرستی طبقه‌بندی کرده است.

برای نمایش بهتر نتایج، ماتریس درهم‌ریختگی با استفاده از کتابخانه seaborn رسم شده و به صورت نمودار حرارتی (heatmap) نمایش داده می‌شود. این نمودار به تشخیص بهتر عملکرد مدل در طبقه‌بندی نمونه‌ها کمک می‌کند.

```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=data.target_names, yticklabels=data.target_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
Accuracy: 0.9404947677620709
Classification Report:
              precision    recall  f1-score   support

           1       0.94      0.94      0.94     31776
           2       0.95      0.95      0.95     42495
           3       0.94      0.94      0.94      5363
           4       0.86      0.85      0.85       412
           5       0.84      0.83      0.83      1424
           6       0.89      0.88      0.88      2605
           7       0.95      0.95      0.95      3077

    accuracy                           0.94     87152
   macro avg       0.91      0.90      0.91     87152
weighted avg       0.94      0.94      0.94     87152

Confusion Matrix:
[[29834  1767     3     0    19     4   149]
 [ 1745 40367   100     0   184    82    17]
 [    0   105  5016    38    14   190     0]
 [    0     0    48   349     0    15     0]
 [   21   203    13     0  1178     8     1]
 [    9    92   178    20     4  2302     0]
 [  140    17     0     0     0     0  2920]]
```



در این قسمت درخت تصمیم آموزش دیده با استفاده از تابع plot_tree از کتابخانه sklearn.tree رسم می‌شود. ویژگی‌ها و کلاس‌ها به صورت متنی نمایش داده می‌شوند و درخت به صورت تصویری نمایش داده می‌شود.

```
# Plot the tree
plt.figure(figsize=(20, 10))
plot_tree(clf, filled=True, feature_names=data.feature_names,
class_names=[str(i) for i in np.unique(y)])
plt.title('Decision Tree')
plt.show()
```

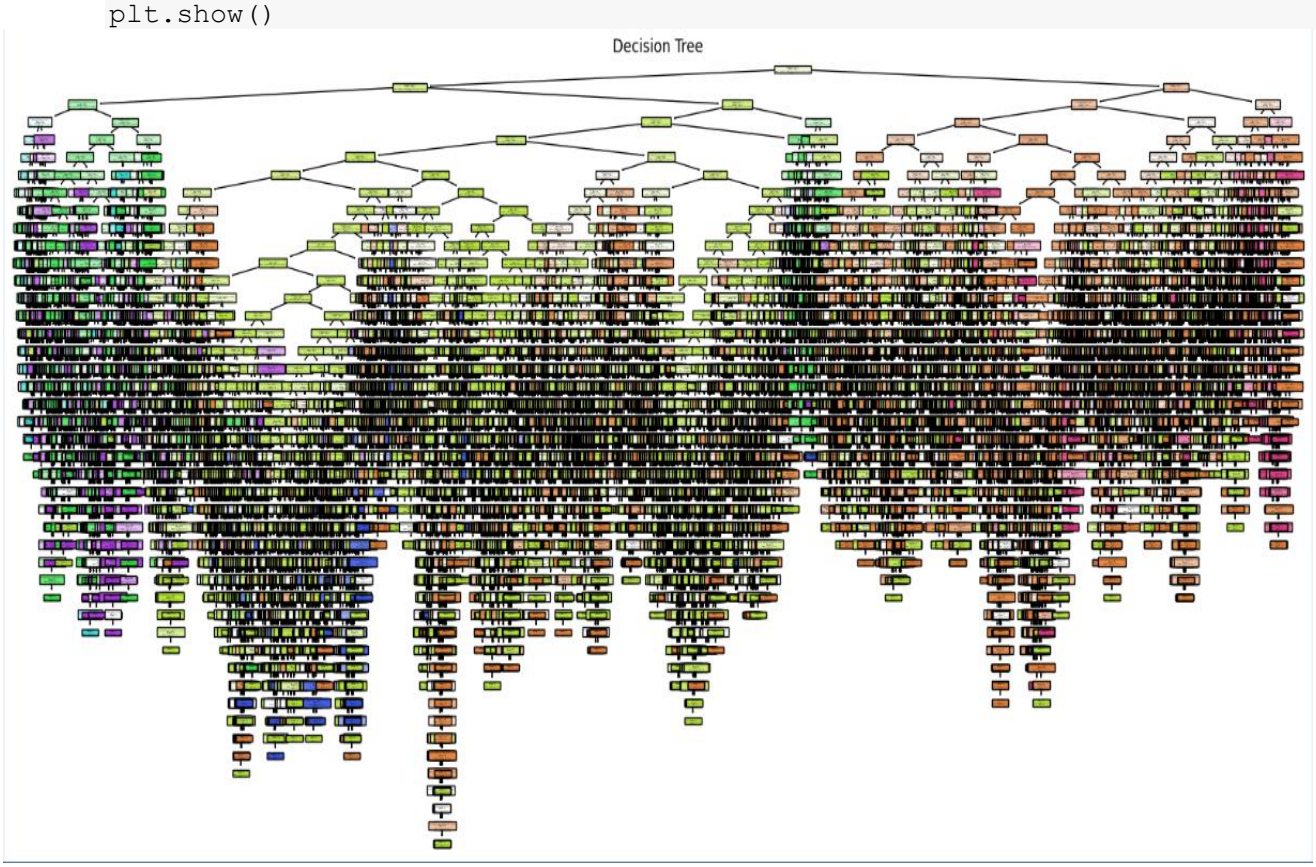
اهمیت ویژگی‌ها با استفاده از ویژگی feature_importances_ مدل محاسبه می‌شود. ویژگی‌ها بر اساس اهمیت‌شان مرتب شده و چاپ می‌شوند.

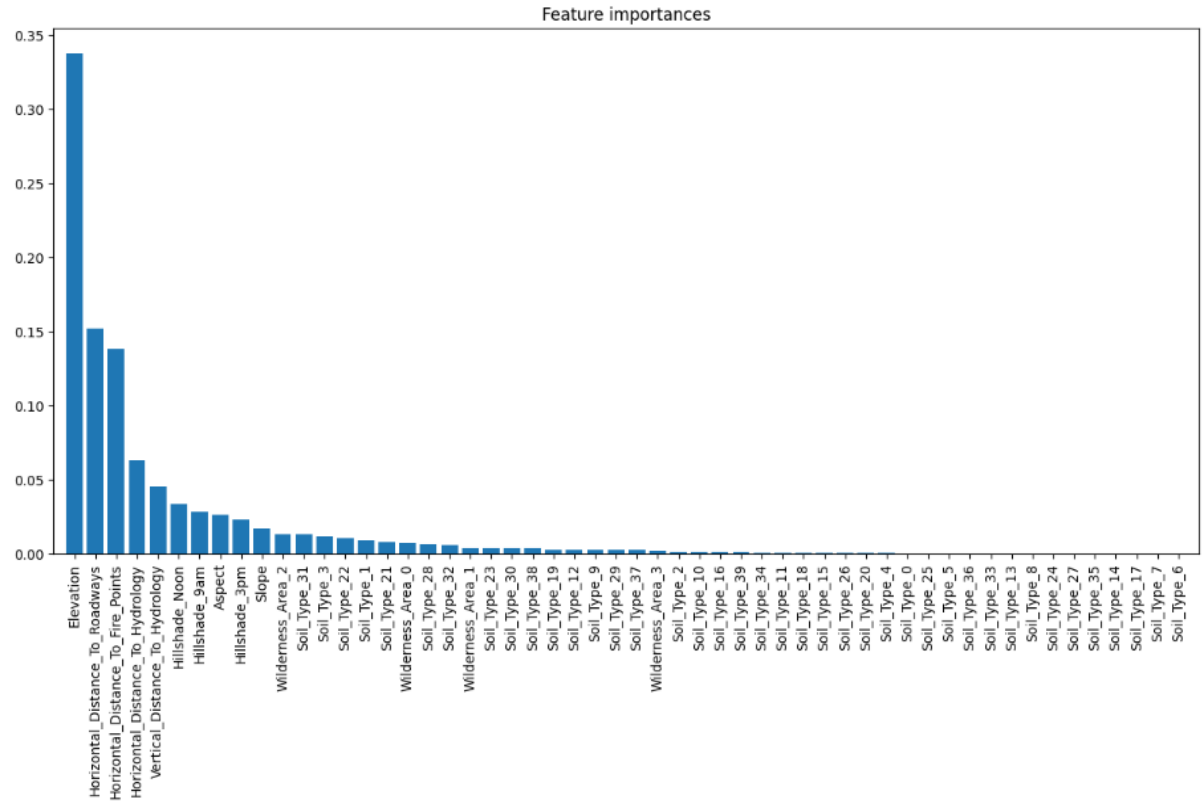
```
# Feature importances
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]

# Print the feature ranking
```

```
print("Feature ranking:")
for f in range(X.shape[1]):
    print(f"{f + 1}. feature {indices[f]} ({importances[indices[f]]})")
```

در این بخش، نمودار اهمیت ویژگی‌ها با استفاده از کتابخانه matplotlib رسم می‌شود. این نمودار نشان می‌دهد که کدام ویژگی‌ها بیشترین تأثیر را در مدل درخت تصمیم دارند.

```
# Plot the feature importances
plt.figure(figsize=(15, 7))
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices], align="center")
plt.xticks(range(X.shape[1]), np.array(data.feature_names)[indices],
rotation=90)
plt.xlim([-1, X.shape[1]])
plt.show()
```



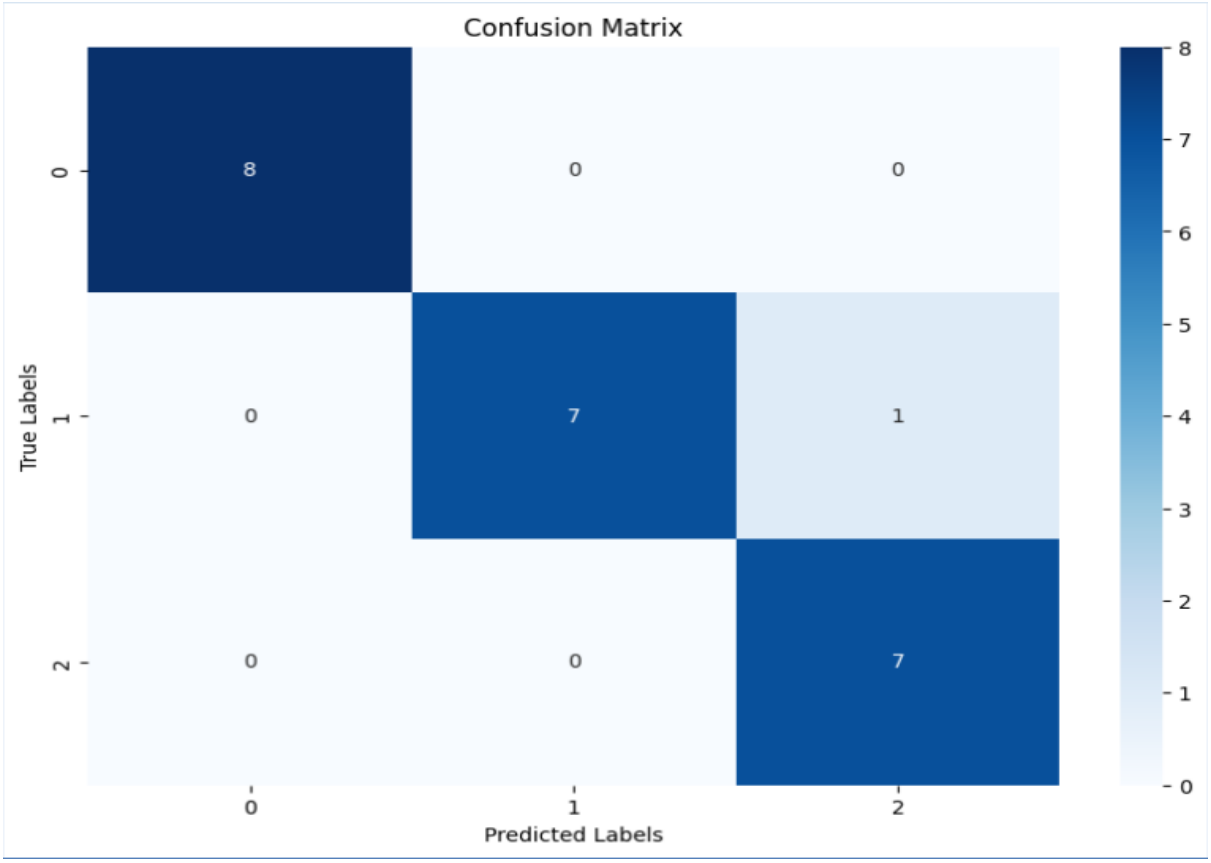



```
Feature ranking:
1. feature 0 (0.3376556161990264)
2. feature 5 (0.1519290672400148)
3. feature 9 (0.1380263270675986)
4. feature 3 (0.06334506676784567)
5. feature 4 (0.04540900227722515)
6. feature 7 (0.03349188616335191)
7. feature 6 (0.02878890915335089)
8. feature 1 (0.02619841655013067)
9. feature 8 (0.02336733092309916)
10. feature 2 (0.017453381449526816)
11. feature 12 (0.013382201898255692)
12. feature 45 (0.013047936038031246)
13. feature 17 (0.011838734050749427)
14. feature 36 (0.010419597124092015)
15. feature 15 (0.00956827096238418)
16. feature 35 (0.008159857127361074)
17. feature 10 (0.007097299913000778)
18. feature 42 (0.006738085562562137)
19. feature 46 (0.006020777733801313)
20. feature 11 (0.00436555353369152)
21. feature 37 (0.004361394545892849)
22. feature 44 (0.004335092631680153)
23. feature 52 (0.004109989816740735)
24. feature 33 (0.0031416851065254733)
25. feature 26 (0.0030153873844853954)
26. feature 23 (0.002896589177672205)
27. feature 43 (0.002743570884943826)
28. feature 51 (0.0024931013554964025)
29. feature 13 (0.0024326045863167283)
30. feature 16 (0.0017709859029266812)
31. feature 24 (0.0016841202041210958)
32. feature 30 (0.0014687072260499607)
```

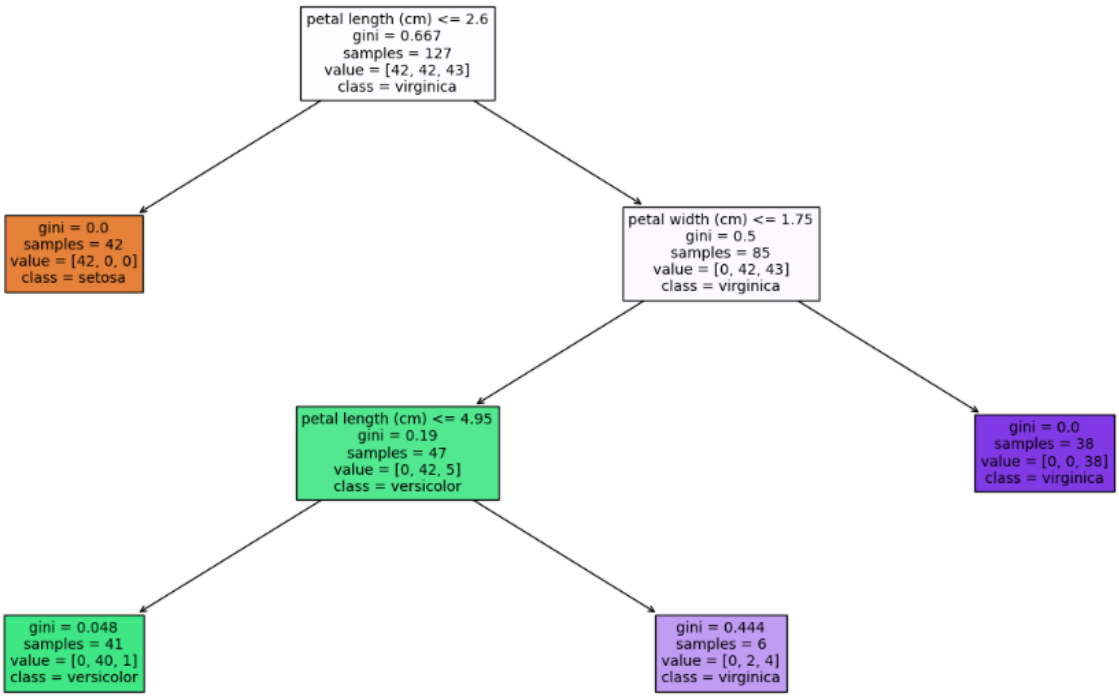
```
33. feature 53 (0.0014050640188691113)
34. feature 48 (0.0010029893090315906)
35. feature 25 (0.0009867894082744599)
36. feature 32 (0.0009809544241215426)
37. feature 29 (0.0009538667360067438)
38. feature 40 (0.0007354913572394137)
39. feature 34 (0.0007221776913517797)
40. feature 18 (0.0006596450326860124)
41. feature 14 (0.0004929602588998845)
42. feature 39 (0.0002433326942995121)
43. feature 19 (0.00020420682560257233)
44. feature 50 (0.00018703704568486012)
45. feature 47 (0.0001741228960878123)
46. feature 27 (0.0001543668573627652)
47. feature 22 (0.00012515768551893037)
48. feature 38 (8.940054993099521e-05)
49. feature 41 (8.673794608245365e-05)
50. feature 49 (1.7766974344686346e-05)
51. feature 28 (1.1318736791950832e-05)
52. feature 31 (5.1924494656291005e-06)
53. feature 21 (4.874544396304868e-06)
54. feature 20 (0.0)
```

چون درخت تصمیم خیلی بزرگ است ما عمق درخت را حداکثر ۳ در نظر می گیریم که قابل دیدن باشد و طبق زیر میبینیم که کدام فیچر ها رنک بوده اند.

```
Accuracy: 0.9565217391304348
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       1.00      0.88      0.93         8
           2       0.88      1.00      0.93         7

    accuracy                           0.96        23
   macro avg       0.96      0.96      0.96        23
weighted avg       0.96      0.96      0.96        23

Confusion Matrix:
[[8 0 0]
 [0 7 1]
 [0 0 7]]
```



Decision Tree



تحلیل کلی:

- مدل دقت بالایی دارد که نشان‌دهنده عملکرد خوب مدل است.

- گزارش طبقه‌بندی نشان می‌دهد که مدل در پیش‌بینی تمامی کلاس‌ها دقت بالایی دارد.

- ماتریس سردرگمی نشان می‌دهد که اکثر پیش‌بینی‌ها صحیح بوده‌اند و تعداد کمی از نمونه‌ها اشتباه طبقه‌بندی شده‌اند.

- درخت تصمیم به‌خوبی نشان می‌دهد که مدل چگونه تصمیم می‌گیرد و ویژگی‌های مهم در تصمیم‌گیری را می‌توان از آن استخراج کرد.

- نمودار اهمیت ویژگی‌ها نشان می‌دهد که کدام ویژگی‌ها بیشترین تاثیر را در تصمیم‌گیری دارند.

این تحلیل‌ها به شما کمک می‌کنند تا بفهمید مدل چگونه عمل می‌کند و چه ویژگی‌هایی در پیش‌بینی‌ها مهم هستند.

در هر بلوک اطلاعاتی مانند نام ویژگی، مقدار threshold، مقدار gini و تعداد نمونه‌های موجود در آن گره به نمایش درآمده‌اند.

که در تصویر بالا ، valueیک آرایه با۳ عنصر عددی است که نشان دهنده فراوانی یا احتمال هر کلاس/برچسب در این گره از درخت است. مجموع این عناصر برابر با ۱ است. به عنوان مثال، اگر برچسب های داده ها [۰، ۱] باشند، اولین دو عنصر ممکن است فراوانی کلاس ۰ و ۱ در این گره باشند.. gini=0.667 یک شاخص ناهمگنی داده ها در این گره است که. یک مقدار gini بالا نشان می دهد که داده ها در این گره متنوع هستند.

همینطور درخت به ترتیب از بالا به پایین و از چپ به راست شکل میگیرد.

۳-۲

ابتدا کتابخانه ها را فراخوانی می کنیم سپس دیتا را لود کرده و دیتا را به بخش آموزش و آزمون تقسیم می کنیم و با استفاده از استاندارد اسکیلر به دو بخش آموزش و آزمون تقسیم می کنیم.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_covtype
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import StandardScaler
import seaborn as sns

# Load the dataset
data = fetch_covtype()
X, y = data.data, data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.15, random_state=34, stratify=y)

# Standardize data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

در ادامه تابعی که مدل را ارزیابی می‌کند را تعریف می کنیم. ابتدا پیش‌بینی‌های مدل روی داده‌های آزمایشی انجام می‌شود. سپس معیارهای ارزیابی مانند دقت، دقت ویژه، بازخوانی، و امتیاز F1 محاسبه و چاپ می‌شوند. همچنین گزارش طبقه‌بندی و ماتریس اغتشاش چاپ و به صورت تصویری نمایش داده می‌شود.

```
# Function to evaluate the model
def evaluate_model(clf, X_test, y_test):
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.show()
```

در ادامه مدل درخت تصمیم با پارامترهای پیش‌فرض آموزش داده می‌شود و سپس ارزیابی می‌گردد.

```
# Train and evaluate with default parameters
clf_default = DecisionTreeClassifier(random_state=34)
clf_default.fit(X_train_scaled, y_train)
print("Default Parameters:")
evaluate_model(clf_default, X_test_scaled, y_test)
```

طبق زیر مدل درخت تصمیم با پارامتر max_depth برابر با ۱۰ آموزش داده می‌شود و سپس ارزیابی می‌گردد.

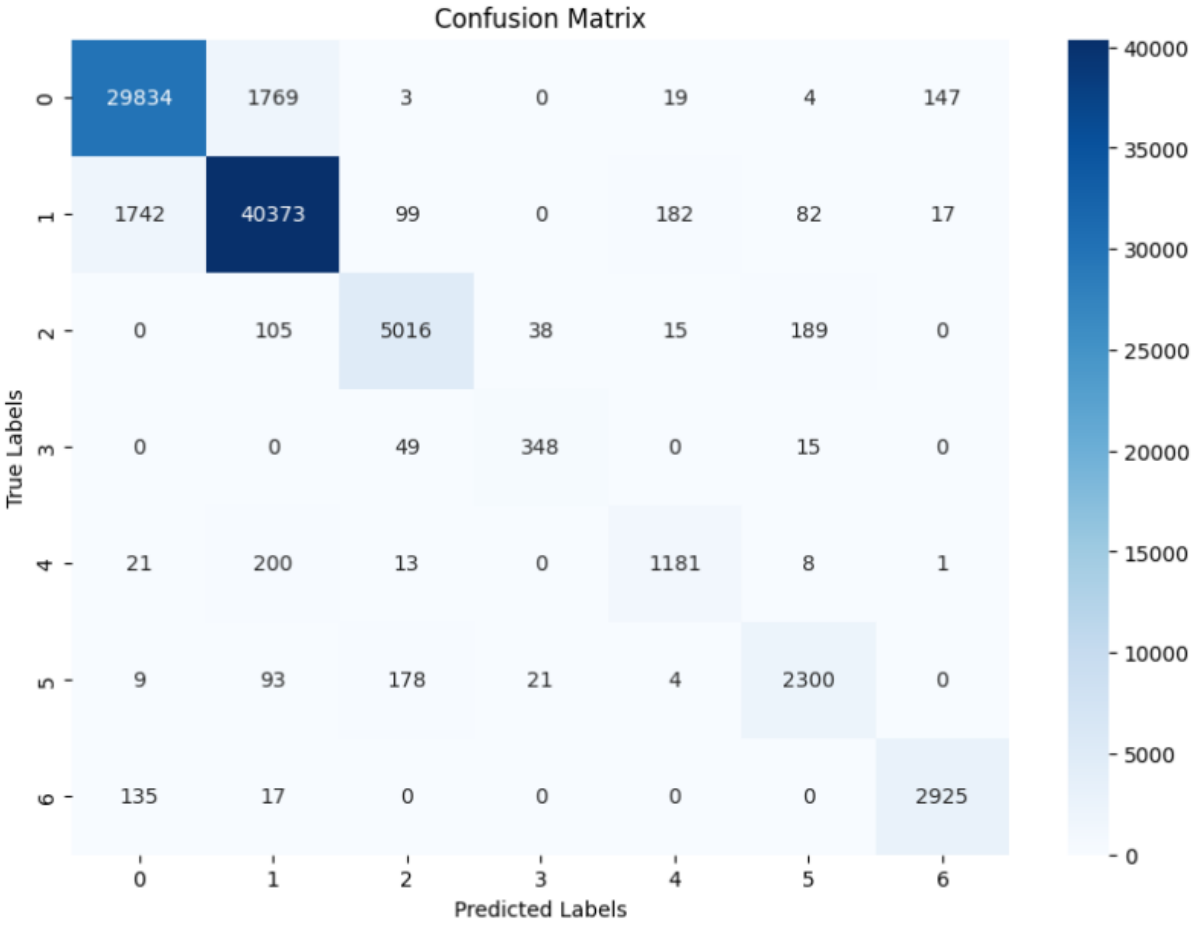
```
# Train and evaluate with max_depth=10
clf_max_depth = DecisionTreeClassifier(random_state=34, max_depth=10)
clf_max_depth.fit(X_train_scaled, y_train)
print("Max Depth = 10:")
evaluate_model(clf_max_depth, X_test_scaled, y_test)
```

مدل درخت تصمیم با پارامتر min_samples_split برابر با ۲۰ آموزش داده می‌شود و سپس ارزیابی می‌گردد.

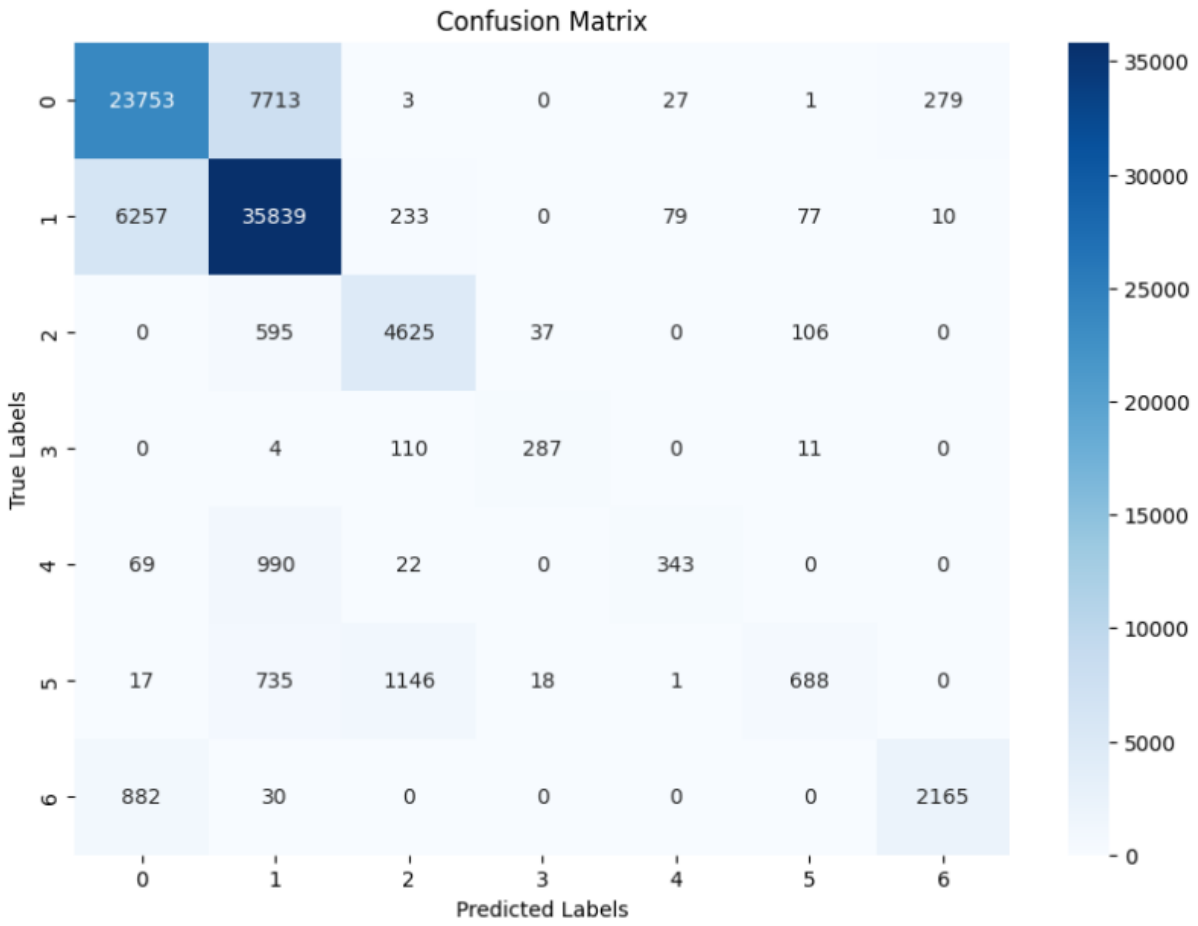
```
# Train and evaluate with min_samples_split=20
clf_min_samples_split = DecisionTreeClassifier(random_state=34,
min_samples_split=20)
clf_min_samples_split.fit(X_train_scaled, y_train)
print("Min Samples Split = 20:")
evaluate_model(clf_min_samples_split, X_test_scaled, y_test)
```

نتایج شاخص ها به ترتیب به صورت زیر نمایش داده می شوند.

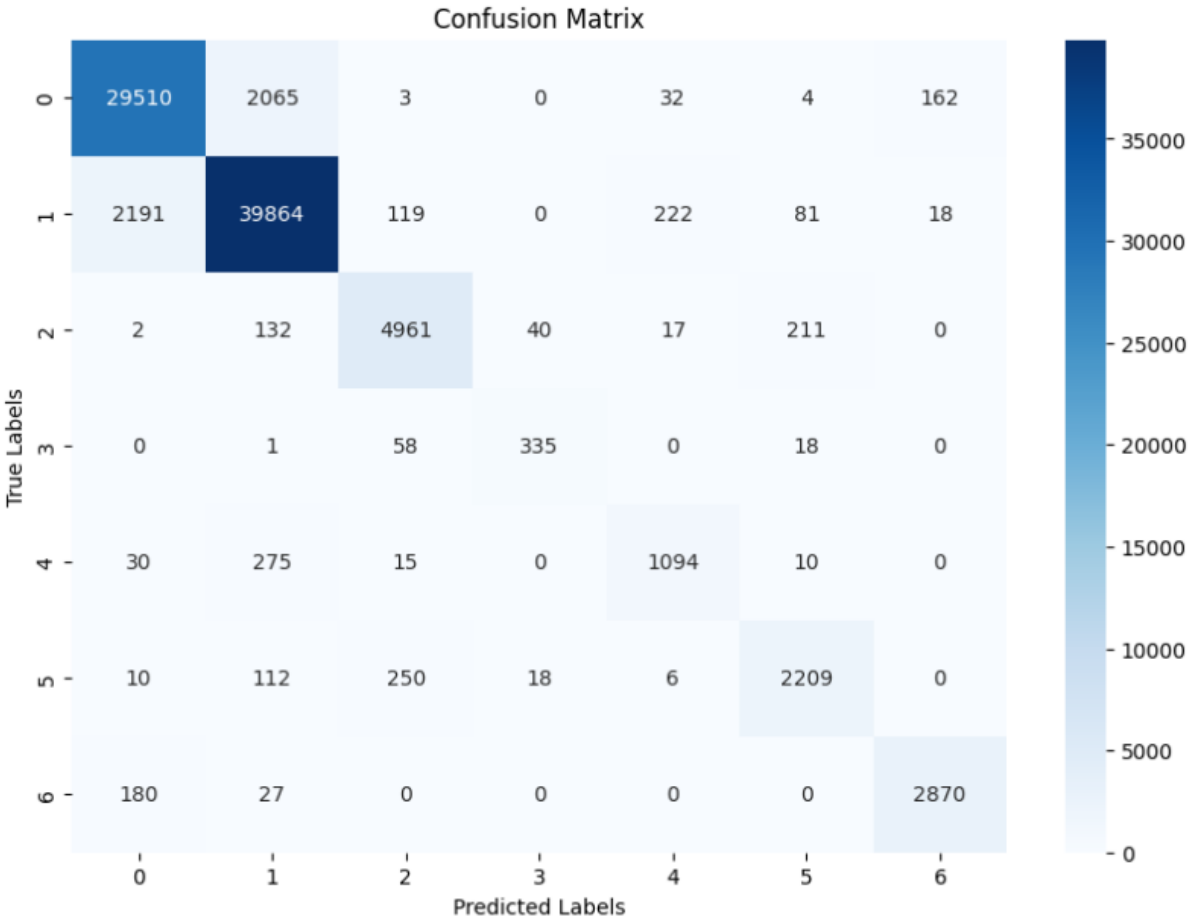
```
Default Parameters:
Accuracy: 0.9406
Precision: 0.9406
Recall: 0.9406
F1 Score: 0.9406
Classification Report:
              precision    recall  f1-score   support

           1       0.94      0.94      0.94     31776
           2       0.95      0.95      0.95     42495
           3       0.94      0.94      0.94      5363
           4       0.86      0.84      0.85       412
           5       0.84      0.83      0.84      1424
           6       0.89      0.88      0.88      2605
           7       0.95      0.95      0.95      3077

    accuracy                           0.94     87152
   macro avg       0.91      0.90      0.91     87152
weighted avg       0.94      0.94      0.94     87152

Confusion Matrix:
[[29834  1769     3     0    19     4   147]
 [ 1742 40373    99     0   182    82    17]
 [    0   105  5016    38    15   189     0]
 [    0     0    49   348     0    15     0]
 [   21   200    13     0  1181     8     1]
 [    9    93   178    21     4  2300     0]
 [  135    17     0     0     0     0  2925]]
```




```
Max Depth = 10:
Accuracy: 0.7768
Precision: 0.7775
Recall: 0.7768
F1 Score: 0.7699
Classification Report:
              precision    recall  f1-score   support

           1       0.77      0.75      0.76     31776
           2       0.78      0.84      0.81     42495
           3       0.75      0.86      0.80      5363
           4       0.84      0.70      0.76       412
           5       0.76      0.24      0.37      1424
           6       0.78      0.26      0.39      2605
           7       0.88      0.70      0.78      3077

    accuracy                           0.78     87152
   macro avg       0.79      0.62      0.67     87152
weighted avg       0.78      0.78      0.77     87152

Confusion Matrix:
[[23753  7713     3     0    27     1   279]
 [ 6257 35839   233     0    79    77    10]
 [    0   595  4625    37     0   106     0]
 [    0     4   110   287     0    11     0]
 [   69   990    22     0   343     0     0]
 [   17   735  1146    18     1   688     0]
 [  882    30     0     0     0     0  2165]]
```




```
Min Samples Split = 20:
Accuracy: 0.9276
Precision: 0.9275
Recall: 0.9276
F1 Score: 0.9275
Classification Report:
              precision    recall  f1-score   support

           1       0.92      0.93      0.93     31776
           2       0.94      0.94      0.94     42495
           3       0.92      0.93      0.92      5363
           4       0.85      0.81      0.83       412
           5       0.80      0.77      0.78      1424
           6       0.87      0.85      0.86      2605
           7       0.94      0.93      0.94      3077

    accuracy                           0.93     87152
   macro avg       0.89      0.88      0.89     87152
weighted avg       0.93      0.93      0.93     87152

Confusion Matrix:
[[29510  2065     3     0    32     4   162]
 [ 2191 39864   119     0   222    81    18]
 [    2   132  4961    40    17   211     0]
 [    0     1    58   335     0    18     0]
 [   30   275    15     0  1094    10     0]
 [   10   112   250    18     6  2209     0]
 [  180    27     0     0     0     0  2870]]
```



تحلیل نتایج

تغییر فراپارامترهای max_depth و min_samples_split می‌تواند تأثیر قابل توجهی بر دقت، دقت ویژه، بازخوانی، و امتیاز F1 مدل داشته باشد. معمولاً:

max_depth مثلاً ۵ و min_samples_split مثلاً ۵ می‌تواند مدل را بیش‌برازش کند، به این معنی مدل جزییات زیادی را از داده‌های آموزشی یاد می‌گیرند و ممکن است در با داده‌های عملکرد جدید ضعیفی داشته باشند.

max_depthمثلاً ۲۰ و min_samples_split مثلاً ۵۰ می‌توان مدل را کم‌برازش کرد، به این معنی که مدل جزییات کافی از داده‌ها یاد نمی‌گیرد و نمی‌تواند به خوبی پیش‌بینی کند.

مزیت هرس کردن

هرس کردن درخت تصمیم با محدود کردن عمق درخت (max_depth) و حداقل تعداد نمونه‌های لازم برای تقسیم یک گره (min_samples_split) برای جلوگیری از بیش‌برازش می‌کند. این کار باعث می‌شود

که مدل ساده‌تر و عمومی‌تر باشد و عملکرد بهتری روی داده‌های آزمایشی (و داده‌های جدید) باشد. همچنین هرس می‌تواند باعث کاهش پیچیدگی مدل و افزایش سرعت محاسباتی شود.

مزایا:

کاهش پیچیدگی مدل: مدلهای هرس شده ساده تر و تفسیر پذیرتر هستند.

بهبود عمومی‌سازی: مدل‌های هرس‌شده به احتمال کمتری بیش‌برازش می‌شوند و در اثر داده‌های جدید عملکرد بهتری دارند.

کاهش زمان محاسبات: مدلهای ساده تر زمان کمتری برای آموزش و پیش بینی نیاز دارند.

۳-۳

جنگل تصادفی(Random Forest)

جنگل تصادفی یک تکنیک ترکیبی است که از مجموعه‌ای از درخت‌های تصمیم استفاده می‌کند. هر درخت تصمیم به صورت مستقل از یک نمونه تصادفی از داده‌ها و ویژگی‌ها ساخته می‌شود و نتایج نهایی با میانگین‌گیری یا رای‌گیری از تمام درخت‌ها به دست می‌آید.

مزایا:

کاهش واریانس (Variance Reduction) : ترکیب چندین درخت تصمیم باعث کاهش واریانس مدل می‌شود و مدل نهایی کمتر به داده‌های آموزشی حساس است.

کاهش بیش‌برازش (Overfitting Reduction) : به دلیل ترکیب مدل‌ها و استفاده از نمونه‌های تصادفی، جنگل تصادفی کمتر احتمال دارد به داده‌های آموزشی بیش‌برازش شود.

پایداری بیشتر: نتایج جنگل تصادفی به تغییرات کوچک در داده‌های آموزشی حساس نیستند، بنابراین مدل پایدارتر است.

AdaBoost:

AdaBoost (Adaptive Boosting) یک تکنیک تقویتی است که به ترتیب مدل‌های پایه را آموزش می‌دهد و به هر مدل وزن می‌دهد. در هر مرحله، داده‌هایی که مدل قبلی به درستی طبقه‌بندی نکرده است، وزن بیشتری می‌گیرند تا مدل جدید بیشتر روی این داده‌ها تمرکز کند.

مزایا:

بهبود دقت (Accuracy Improvement) : با توجه به تمرکز بیشتر روی نمونه‌هایی که به درستی طبقه‌بندی نشده‌اند، AdaBoostمی‌تواند دقت مدل را بهبود بخشد.

ترکیب مدل‌های ضعیف به یک مدل قوی: حتی اگر مدل‌های پایه (مثلاً درخت‌های تصمیم ساده) عملکرد ضعیفی داشته باشند، ترکیب آن‌ها با AdaBoost می‌تواند یک مدل قوی ایجاد کند.

پیش‌بینی‌های وزنی AdaBoost : از پیش‌بینی‌های وزنی استفاده می‌کند که باعث می‌شود مدل نهایی عملکرد بهتری داشته باشد.

مقایسه و انتخاب روش مناسب

جنگل تصادفی بیشتر بر کاهش واریانس و بهبود پایداری مدل تمرکز دارد و معمولاً در داده‌های با نویز زیاد یا پیچیده کاربرد دارد.

AdaBoost بیشتر بر کاهش بایاس تمرکز دارد و می‌تواند برای داده‌هایی که مدل‌های ساده‌تر نمی‌توانند به خوبی آنها را طبقه‌بندی کنند، مناسب باشد.

نتیجه‌گیری

هر دو روش جنگل تصادفی و AdaBoost می‌توانند به بهبود نتایج مدل‌ها کمک کنند. جنگل تصادفی با کاهش واریانس و پایداری بیشتر، و AdaBoost با کاهش بایاس و تمرکز بیشتر بر نمونه‌های دشوار. انتخاب روش مناسب بستگی به نوع داده‌ها و مسئله مورد نظر دارد. در بسیاری از موارد، ترکیب این روش‌ها با هم نیز می‌تواند به بهبود عملکرد کلی مدل‌ها کمک کند.

ما از جنگل تصادفی استفاده کرده ایم. ابتدا کتابخانه ها فراخوانی شده، دیتا لود شده و به قسمت تست و ترین جدا شده و نرمالایز میشود.

در ادامه این تابع مدل را ارزیابی می‌کند. ابتدا پیش‌بینی‌های مدل روی داده‌های آزمایشی انجام می‌شود. سپس معیارهای ارزیابی مانند دقت، دقت ویژه، بازخوانی، و امتیاز F1 محاسبه و چاپ می‌شوند. همچنین گزارش طبقه‌بندی و ماتریس اغتشاش چاپ و به صورت تصویری نمایش داده می‌شود.

```
# Function to evaluate the model
def evaluate_model(clf, X_test, y_test):
    y_pred = clf.predict(X_test)
```

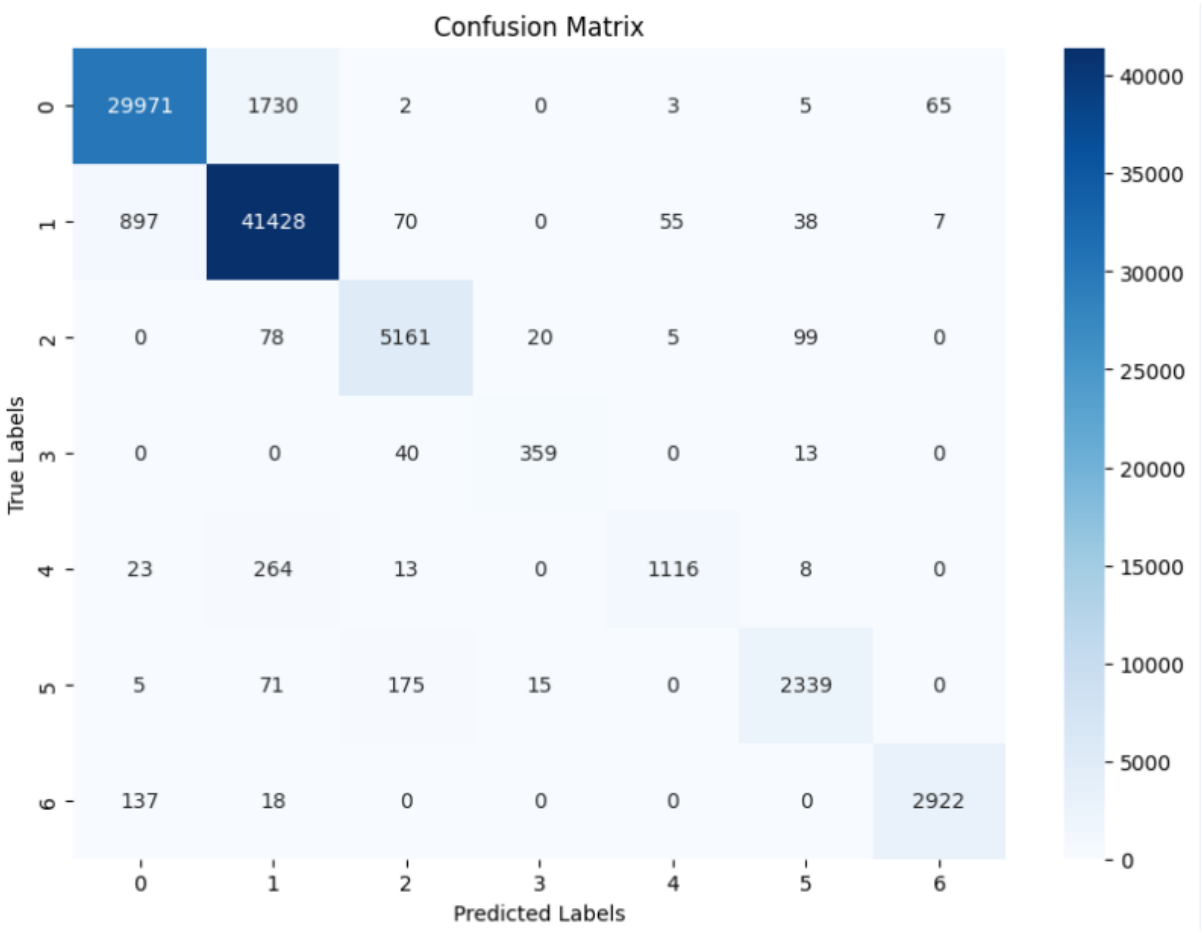
```
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.show()
```

سپس مدل جنگل تصادفی با استفاده از پارامترهای مشخص شده (تعداد ۱۰۰ درخت، بدون محدودیت در عمق درخت، و حداقل ۲ نمونه برای تقسیم یک گره) آموزش داده می‌شود. سپس مدل با استفاده از تابع evaluate_model ارزیابی می‌شود و نتایج ارزیابی شامل دقت، دقت ویژه، بازخوانی، امتیاز F1 ، گزارش طبقه‌بندی، و ماتریس اغتشاش نمایش داده می‌شود.

```
# Train and evaluate with Random Forest
clf_rf = RandomForestClassifier(random_state=34, n_estimators=100,
max_depth=None, min_samples_split=2)
clf_rf.fit(X_train_scaled, y_train)
print("Random Forest Classifier:")
evaluate_model(clf_rf, X_test_scaled, y_test)
```

در نهایت نتایج به صورت زیر آورده شده است.

```
Random Forest Classifier:
Accuracy: 0.9558
Precision: 0.9559
Recall: 0.9558
F1 Score: 0.9555
Classification Report:
              precision    recall  f1-score   support

           1       0.97      0.94      0.95     31776
           2       0.95      0.97      0.96     42495
           3       0.95      0.96      0.95      5363
           4       0.91      0.87      0.89       412
           5       0.95      0.78      0.86      1424
           6       0.93      0.90      0.92      2605
           7       0.98      0.95      0.96      3077

    accuracy                           0.96     87152
   macro avg       0.95      0.91      0.93     87152
weighted avg       0.96      0.96      0.96     87152

Confusion Matrix:
[[29971  1730     2     0     3     5    65]
 [  897 41428    70     0    55    38     7]
 [    0    78  5161    20     5    99     0]
 [    0     0    40   359     0    13     0]
 [   23   264    13     0  1116     8     0]
 [    5    71   175    15     0  2339     0]
 [  137    18     0     0     0     0  2922]]
```



۴ سوال چهارم

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

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

توضیحات جایگزین در مورد مجموعه داده‌ها

این مجموعه داده از سال ۱۹۸۸ آغاز شده و شامل چهار پایگاه داده مختلف از مناطق کلیولند، مجارستان، سوئیس و لانگ بیچ است. در کل، این مجموعه داده شامل ۷۶ ویژگی است که یکی از این ویژگی‌ها برای پیش‌بینی بیماری قلبی استفاده می‌شود. با این حال، تمامی تحقیقات منتشر شده از زیرمجموعه‌ای از ۱۴ ویژگی برای تحلیل استفاده می‌کنند. هدف اصلی این مجموعه داده، تشخیص وجود بیماری قلبی در بیماران است.

ویژگی‌های کلیدی و هدف مجموعه داده

فیلد target نشان‌دهنده وجود یا عدم وجود بیماری قلبی است و دارای دو مقدار صحیح می‌باشد: ۰ به معنای عدم بیماری و ۱ به معنای وجود بیماری قلبی.

۱۴ ویژگی اصلی استفاده شده:

Age
Sex
Chest Pain Type
Resting Blood Pressure
Serum Cholesterol
Fasting Blood Sugar
Resting ECG Results
Maximum Heart Rate Achieved
Exercise Induced Angina
ST Depression
Slope of the Peak Exercise ST Segment
Number of Major Vessels
Thalassemiaبا مقادیر ۰ (نرمال)، ۱ (نقص ثابت)، و ۲ (نقص قابل برگشت)
Predicted Attribute

این مجموعه داده‌ها برای تحقیقات و توسعه مدل‌های تشخیصی و پیش‌بینی بیماری‌های قلبی استفاده می‌شود و ابزار مهمی برای پزشکان و محققان در بهبود تشخیص و درمان بیماری‌های قلبی است.هدف اصلی از استفاده این مجموعه داده، توسعه و ارزیابی مدل‌های یادگیری ماشین برای پیش‌بینی احتمال ابتلا به بیماری قلبی است. این مدل‌ها می‌توانند به تشخیص سریع‌تر و دقیق‌تر بیماری‌ها کمک کنند و در نتیجه باعث بهبود درمان و کاهش مرگ و میر ناشی از بیماری‌های قلبی شوند.

الگوریتم طبقه‌بندی بیز (Bayes Classifier) یک روش آماری برای طبقه‌بندی داده‌ها است که بر اساس نظریه احتمال بیز عمل می‌کند. این الگوریتم به خصوص در مسائل یادگیری ماشین و پردازش زبان طبیعی کاربرد فراوان دارد. در ادامه یک توضیح کلی و جامع در مورد این الگوریتم ارائه می‌شود:

نظریه احتمال بیز

نظریه بیز (Bayes' Theorem) یک روش ریاضی برای محاسبه احتمال وقوع یک رویداد بر اساس اطلاعات موجود در مورد سایر رویدادهای مرتبط است. این نظریه به شکل زیر بیان می‌شود:

$$\frac{P(B|A)\cdot P(A)}{P(B)} = P(A|B)$$

در اینجا:

$P(A|B)$ احتمال وقوع رویداد A به شرط وقوع رویداد B است.

$P(B|A)$ احتمال وقوع رویداد B به شرط وقوع رویداد A است.

$P(A)$ احتمال وقوع رویداد A است.

$P(B)$ احتمال وقوع رویداد B است.

فرضیه ساده(Naive Assumption)

الگوریتم طبقه‌بندی بیز ساده (Naive Bayes Classifier) بر این فرض استوار است که ویژگی‌های ورودی (خصیصه‌ها) مستقل از یکدیگر هستند. این فرض ساده‌سازی شده است، زیرا در بسیاری از موارد این ویژگی‌ها ممکن است به هم وابسته باشند. با این حال، این الگوریتم با وجود این فرض ساده، عملکرد خوبی در بسیاری از مسائل نشان می‌دهد.

نحوه کار الگوریتم بیز ساده

الگوریتم بیز ساده برای طبقه‌بندی یک نمونه جدید به یکی از کلاس‌های موجود، از مراحل زیر پیروی می‌کند:

محاسبه احتمال پیشین هر کلاس :ابتدا احتمال وقوع هر کلاس (احتمال پیشین) را از داده‌های آموزشی محاسبه می‌کنیم. این احتمال برابر است با نسبت تعداد نمونه‌های هر کلاس به تعداد کل نمونه‌ها.
محاسبه احتمال شرطی ویژگی‌ها :سپس احتمال وقوع هر ویژگی به شرط وقوع هر کلاس را محاسبه می‌کنیم. برای داده‌های عددی، معمولاً از توزیع نرمال استفاده می‌شود و برای داده‌های دسته‌بندی از تعداد دفعات وقوع هر مقدار استفاده می‌شود.

محاسبه احتمال پسین :برای هر کلاس، احتمال تعلق نمونه جدید به آن کلاس را با استفاده از فرمول بیز محاسبه می‌کنیم. این احتمال برابر است با حاصل‌ضرب احتمال پیشین کلاس و احتمال شرطی ویژگی‌ها.

انتخاب کلاس با بالاترین احتمال :نمونه جدید را به کلاسی که بالاترین احتمال را دارد، اختصاص می‌دهیم.

کاربردها

الگوریتم بیز ساده به دلیل سادگی و کارایی در بسیاری از کاربردها مورد استفاده قرار می‌گیرد، از جمله:

طبقه‌بندی متون :مانند فیلتر کردن ایمیل‌های اسپم، طبقه‌بندی اسناد و تحلیل احساسات.

تشخیص بیماری‌ها :بر اساس علائم بیمار و داده‌های پزشکی.

سیستم‌های توصیه‌گر :برای پیشنهاد محصولات یا محتوا به کاربران بر اساس تاریخچه فعالیت‌های آنان.

مزایا و معایب

مزایا:

سادگی :الگوریتم بیز ساده به سادگی قابل پیاده‌سازی است.

سرعت :این الگوریتم به دلیل محاسبات ساده و فرضیه‌های ساده، سرعت بالایی در آموزش و پیش‌بینی دارد.

کارایی :حتی با وجود فرضیه استقلال ساده، در بسیاری از مسائل عملی عملکرد خوبی دارد.

معایب:

فرضیه استقلال :فرضیه استقلال ویژگی‌ها همیشه درست نیست و می‌تواند منجر به کاهش دقت در برخی موارد شود.

حساسیت به داده‌های نادرست :اگر داده‌های آموزشی نادرست یا دارای نویز باشند، ممکن است الگوریتم عملکرد ضعیفی داشته باشد.

نتیجه‌گیری

الگوریتم طبقه‌بندی بیز ساده یکی از روش‌های قدرتمند و کارآمد برای طبقه‌بندی داده‌ها است که با وجود فرضیه‌های ساده‌سازی شده، در بسیاری از مسائل واقعی عملکرد خوبی دارد. این الگوریتم به خصوص در مسائل پردازش زبان طبیعی و تشخیص الگو بسیار کاربردی است.

ابتدا کتابخانه های مورد نیاز را فراخوانی کرده و سپس با نصب gdown دیتا را از درایو دانلود کرده و در مرحله بعد دیتا را میخوانیم.

```
# Import libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, ConfusionMatrixDisplay
import gdown

# Install gdown if not already installed
!pip install --upgrade --no-cache-dir gdown

# Download the dataset
url = 'https://drive.google.com/uc?id=1l5QpRb_WwVv4_A2gEyyU1ykzW44J-
HYb'
output = 'heartdataset.csv'
gdown.download(url, output, quiet=False)

# Load the dataset
dataset = pd.read_csv(output)
print(dataset.head())
list_of_column_names = list(dataset.columns)
print(list_of_column_names)
```

در ادامه داده ها را به صورت تصادفی به هم میریزیم (shuffle)، و با استفاده از train-test-split به دو بخش آموزش و آزمون تقسیم می کنیم.

```
# Shuffle the dataset
dataset = shuffle(dataset, random_state=34)
```

```
# Split the data into features and target
X = dataset.drop('target', axis=1)
y = dataset['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=34, stratify=y)
```

در مرحله بعد دیتا را با استفاده از standard scaler بین منفی یک و مثبت یک نرمالایز می کنیم.(تغییر اسکیل دیتا بین منفی یک و یک)

```
# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

در ادامه مدل Gaussian Naive Bayes ایجاد شده و با استفاده از داده‌های آموزشی آموزش داده می‌شود.

```
# Initialize the Gaussian Naive Bayes classifier
gnb = GaussianNB()

# Fit the model
gnb.fit(X_train_scaled, y_train)
```

سپس برچسب‌های داده‌های آزمایشی پیش‌بینی شده و دقت مدل محاسبه و چاپ می‌شود. همچنین گزارش طبقه‌بندی و ماتریس اغتشاش تولید و نمایش داده می‌شوند.

```
# Predict the labels for the test set
y_pred = gnb.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Generate the classification report
report = classification_report(y_test, y_pred, target_names=['No
Disease', 'Disease'])
print("Classification Report:")
print(report)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No
Disease', 'Disease'])
```

```
cmd.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

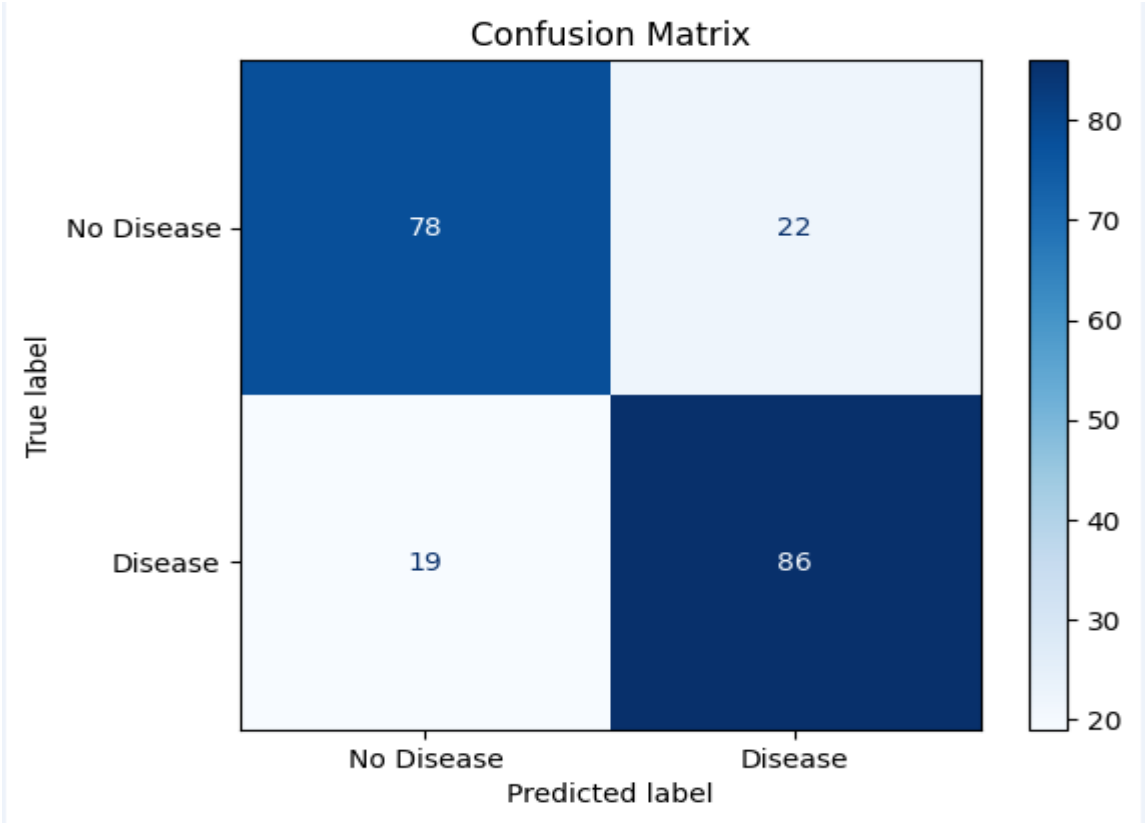
در زیر تفاوت بین میانگین‌های ماکرو و میکرو توضیح داده می‌شود. میانگین ماکرو به هر کلاس به طور مساوی وزن می‌دهد در حالی که میانگین میکرو به کلاس‌های با نمونه‌های بیشتر وزن بیشتری می‌دهد.

```
# Explanation of Macro vs. Micro
print("Macro-averaged metrics calculate the metric independently for
each class and then take the average, treating all classes equally.")
print("Micro-averaged metrics aggregate the contributions of all
classes to compute the average metric, giving more weight to classes
with more samples.")
```

در نهایت پنج نمونه تصادفی از داده‌های آزمایشی انتخاب شده و برچسب‌های آن‌ها پیش‌بینی می‌شوند. سپس این پیش‌بینی‌ها با برچسب‌های واقعی مقایسه و نمایش داده می‌شوند.

```
# Select five random samples from the test set
np.random.seed(34)
random_indices = np.random.choice(len(X_test), size=5, replace=False)
random_samples = X_test.iloc[random_indices]
random_samples_scaled = X_test_scaled[random_indices]

# Predict the labels for the random samples
random_preds = gnb.predict(random_samples_scaled)

# Compare the actual and predicted labels
comparison = pd.DataFrame({
    'Actual': y_test.iloc[random_indices].values,
    'Predicted': random_preds
})

print("Comparison of Actual and Predicted Labels for Random Samples:")
print(comparison)
```

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

   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0
['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']
Accuracy: 0.8000
Classification Report:
              precision    recall  f1-score   support

  No Disease       0.80      0.78      0.79       100
     Disease       0.80      0.82      0.81       105

    accuracy                           0.80       205
   macro avg       0.80      0.80      0.80       205
weighted avg       0.80      0.80      0.80       205
```




```
Macro-averaged metrics calculate the metric independently for each class and then take the average, treating all classes equally.
Micro-averaged metrics aggregate the contributions of all classes to compute the average metric, giving more weight to classes with more samples.
Comparison of Actual and Predicted Labels for Random Samples:
   Actual  Predicted
0       1          1
1       0          0
2       1          1
3       1          1
4       1          1
```

در این شبکه به دقت ۸۰ درصد رسیده ایم.