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

درس یادگیری ماشین

پاسخ مینی‌پروژه اول

نام و نام خانوادگی	ایمان گندمی
شمارهٔ دانشجویی	۴۰۲۲۳۷۰۴
تاریخ	فروردین ۱۴۰۳

فهرست مطالب

۱ سوال اول ۸

۱.۱ فرض کنید در یک مسألهٔ طبقه بندی دوکلاسه، دو لایهٔ انتهایی شبکهٔ شما فعال ساز ReLU و سیگموید است. چه اتفاقی می افتد؟ ۸

۲.۱ یک جایگزین برای ReLU در معادله ۱ آورده شده است. ضمن محاسبهٔ گرادیان آن، حداقل یک مزیت آن نسبت به ReLU را توضیح دهید ۹

۳.۱ به کمک یک نورون ساده، پرسپترون و یا نورون Pitts-McCulloch شبکه ای طراحی کنید که ۹

۲ سوال دوم ۲۰

۱.۲ دیتاست CWRU Bearing که در «مینی پروژهٔ شمارهٔ یک» با آن آشنا شدید را به خاطر آورید. علاوه بر دو کلاسی که در آن مینی پروژه در نظر گرفتید، با مراجعه به صفحهٔ داده های عیب در حالت k،۱۲ دو کلاس دیگر نیز از و X-۰۰۷@۶OR اضافه کنید. با انجام این کار یک کلاس دادهٔ سالم و سه کلاس طریق فایل های X-۰۰۷B۱ از داده های دارای سه عیب متفاوت خواهید داشت. در مورد این که هر فایل مربوط به چه نوع عیبی است به صورت کوتاه توضیح دهید. سپس در ادامه، تمام کارهایی که در بخش «۲» سوال دوم «مینی پروژهٔ یک» برای استخراج ویژگی و آماده سازی دیتا انجام داده بودید را روی دیتاست جدید خود پیاده سازی کنید. در قسمت تقسیم بندی داده ها، یک بخش برای «اعتبارسنجی» به بخش های «آموزش» و «آزمون» اضافه کنید و توضیح دهید که کاربرد این بخش چیست. ۲۰

۲.۲ یک مدل MLP ساده با ۲ لایهٔ پنهان یا بیش تر بسازید. بخشی از داده های آموزش را برای اعتبارسنجی کنار بگذارید و با انتخاب بهینه ساز 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 داد های آزمون را با استفاده ماتریس درهم ریختگی و report-classification نشان داده و نتایج به صورت دقیق تحلیل کنید. ۲۷

۳.۲ فرآیند سوال قبل را با یک بهینه ساز و تابع اتلاف جدید انجام داده و نتایج را مقایسه و تحلیل کنید. بررسی کنید که آیا تغییر تابع اتلاف می تواند در نتیجه اثرگذار باشد؟ ۳۲

۴.۲ در موردCross-validation K-Fold Stratified و Cross-validation K-Fold و مزایای هر یک توضیح دهید. سپس با ذکر دلیل، یکی از این روش ها را انتخاب کرده و بخش «۲» سوال سوم را با آن پیاده سازی کنید و نتایج خود را تحلیل کنید. ۳۴

۳ سوال سوم ۳۸

۱.۳ با استفاده از ماتریس درهم ریختگی و حداقل سه شاخصهٔ ارزیابی مربوط به وظیفهٔ طبقه بندی، عمل کرد درخت آموزش داده شدهٔ خود را روی بخش آزمون داده ها ارزیابی کنید و نتایج را به صورت دقیق گزارش کنید. تأثیر مقادیر کوچک و بزرگ حداقل دو فراپارامتر را بررسی کنید. تغییر فراپارامترهای مربوط به هرس کردن چه تأثیری روی نتایج دارد و مزیت آن چیست؟ ۴۳

۲.۳ توضیح دهید که روش هایی مانند جنگل تصادفی و AdaBoost چگونه می توانند به بهبود نتایج کمک کنند. سپس، با انتخاب یکی از این روش ها و استفاده از فراپارامترهای مناسب، سعی کنید نتایج پیاده سازی در مراحل قبلی را ارتقاء دهید. ۴۶

۴ عنوان سوال چهارم ۴۸

فهرست تصاویر

۱ توابع فعال‌ساز Relu و ELU ۹
۲ بلوک دیاگرام مدل طبقه‌بند خطی ۱۵
۳ بلوک دیاگرام مدل طبقه‌بند خطی ۱۶
۴ بلوک دیاگرام مدل طبقه‌بند خطی ۱۶
۵ بلوک دیاگرام مدل طبقه‌بند خطی ۱۷
۶ خروجی هر سه حالت استفاده از دو تابع فعال‌ساز مختلف ۲۰
۷ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۲۲
۸ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۲۲
۹ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۲۳
۱۰ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۲۶
۱۱ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۲۶
۱۲ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۲۷
۱۳ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۲۹
۱۴ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۳۰
۱۵ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۳۱
۱۶ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۳۱
۱۷ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۳۲
۱۸ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۳۳
۱۹ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۳۳
۲۰ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۳۴
۲۱ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۳۷
۲۲ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۳۸
۲۳ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۳۹
۲۴ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۴۰
۲۵ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۴۱
۲۶ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۴۲
۲۷ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۴۳
۲۸ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۴۴
۲۹ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۴۵
۳۰ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۴۶
۳۱ نتیجهٔ آموزش و ارزیابی با استفاده از SGD ۴۸
۳۲ بلوک دیاگرام مدل طبقه‌بند خطی ۴۸
۳۳ بلوک دیاگرام مدل طبقه‌بند خطی ۴۹
۳۴ بلوک دیاگرام مدل طبقه‌بند خطی ۵۰

۳۵ بلوک دیاگرام مدل طبقه‌بند خطی ۵۲
۳۶ بلوک دیاگرام مدل طبقه‌بند خطی ۵۵
۳۷ بلوک دیاگرام مدل طبقه‌بند خطی ۵۶

فهرست جداول

فهرست برنامه‌ها

۱ (Python) libraries import ۹
۲ (Python) libraries import ۱۱
۳ (Python) libraries import ۱۱
۴ (Python) libraries import ۱۲
۵ (Python) libraries import ۱۳
۶ (Python) libraries import ۱۳
۷ (Python) libraries import ۱۴
۸ (Python) libraries import ۱۷
۹ (Python) libraries import ۱۸
۱۰ (Python) libraries import ۱۹
۱۱ library and dataset import ۲۱
۱۲ library and dataset import ۲۲
۱۳ library and dataset import ۲۲
۱۴ library and dataset import ۲۳
۱۵ library and dataset import ۲۴
۱۶ library and dataset import ۲۵
۱۷ library and dataset import ۲۵
۱۸ library and dataset import ۲۶
۱۹ library and dataset import ۲۶
۲۰ library and dataset import ۲۷
۲۱ library and dataset import ۲۸
۲۲ library and dataset import ۲۸
۲۳ library and dataset import ۲۹
۲۴ library and dataset import ۲۹
۲۵ library and dataset import ۳۰
۲۶ library and dataset import ۳۵
۲۷ library and dataset import ۳۵
۲۸ library and dataset import ۳۸
۲۹ library and dataset import ۳۸
۳۰ library and dataset import ۴۰
۳۱ library and dataset import ۴۱
۳۲ library and dataset import ۴۱
۳۳ library and dataset import ۴۲
۳۴ library and dataset import ۴۲

۳۵ library and dataset import ۴۴
۳۶ library and dataset import ۴۷
۳۷ (Python) libraries import ۴۸
۳۸ (Python) libraries import ۴۸
۳۹ (Python) libraries import ۴۹
۴۰ (Python) libraries import ۴۹
۴۱ (Python) libraries import ۵۰
۴۲ (Python) libraries import ۵۱
۴۳ (Python) libraries import ۵۱
۴۴ (Python) libraries import ۵۲
۴۵ (Python) libraries import ۵۲
۴۶ (Python) libraries import ۵۳
۴۷ (Python) libraries import ۵۴
۴۸ (Python) libraries import ۵۶

لینک پوشهٔ گیت‌هاب:
https://github.com/ImanGandomi/MachineLearning2024/tree/main/ml_MiniProject_1
لینک گوگل کولب:
https://colab.research.google.com/drive/1AjG5WVYKpsJYs8rkoI94V5ZuJdnVG3_m?usp=sharing

۱ سوال اول

۱.۱ فرض کنید در یک مسألهٔ طبقه بندی دوکلاسه، دو لایهٔ انتهایی شبکهٔ شما فعال ساز ReLU و سیگموید است. چه اتفاقی می افتد؟

ابتدا در مورد مزیت استفاده از هر یک از این توابع فعال‌ساز می‌توان صحبت کرد. تابع فعال‌ساز ReLU به دلیل سادگی و اثرگذاری در یادگیری عمیق شناخته می‌شود. این تابع باعث ایجاد حالت غیرخطی می‌شود به این صورت که اگر مقدار ورودی مثبت باید آن را عبور داده و در غر این صورت صفر خروجی می‌دهد. این تابع باعث غلبه بر مشکل محوشدگی گرادیان می‌شود. این تابع نسبت به سایر توابع مثل سیگموید به مرحلهٔ backpropagation بهتر کمک کرده و برای ورودی‌های بزرگتر خروجی‌ها را کوچک نمی‌کند و در نتیجه محوشدگی گرادیان کمتر بروز می‌کند. کاهش بار محاسباتی به دلیل سادگی این گرادیان از دیگر مزایای استفاده از آن است. همچنین این سادگی باعث می‌شد که میزان سرعت همگرایی شبکه به شدت افزایش یابد.

تابع فعال‌ساز sigmoid باعث هدایت خروجی به بین مقادیر ۰ و ۱ می‌شود. خروجی این تابع به عنوان احتمال تعلق به کلاس خاص تعبیر می‌شود و معمولا در لایه آخر برای انجام طبقه‌بندی نهایی مورد استفاده قرار می‌گیرد. استفاده از این تابع باعث ایجاد حالت smooth gradient می‌شود و تا حدودی باعث آسانی در مرحلهٔ backpropagation می‌شود. البته استفاده از این تابع به دلیل کوچک‌کردن ورودی‌های خود به مقدار بین ۰ و ۱ باعث ایجاد محوشدگی گرادیان می‌شود.

در مورد سوال مطرح شده می‌توان اینگونه پاسخ داد که استفاده از این نوع ساختار در لایه‌های انتهایی مدل طبقه‌بند به نوعی رایج است. مزایای استفاده از این ساختار را با توجه به توضیحات گفته‌شده می‌توان این‌طور مطرح کرد: برای لایه ماقبل آخر با تابع فعال‌ساز :ReLU

ایجاد غیرخطی‌گری و استخراج ویژگی: پیش از لایهٔ آخر که کار طبقه‌بندی صورت می‌گیرد، استفاده از این تابع باعث می‌شود ویژگی‌هایی که خاصیت غیرخطی دارند به طور مناسب‌تری از یکدیگر قابلیت تفکیک پیدا کنند. همچنین پیش از ورودی به لایهٔ طبقه‌بند، مدل با استفاده ازاین تابع فعال‌ساز قابلیت یادگیری ویژگی‌های سطح بالاتر را به دست می‌آورد

گرادیان و سرعت بالا: همانطور هم که قبلا ذکر شد این تابع باعث عدم ایجاد محوشدگی گرادیان شده و آموزش شبکه را هموار می‌کند. همچنین به دلیل ساختار سادهٔ خود باعث همگرایی سریع‌تر مدل می‌شود

برای لایهٔ آخر با تابع فعال‌ساز :sigmoid

تفسیرپذیری: با استفاده از این تابع در لایهٔ آخر که خروجی بین ۰ و ۱ ایجاد می‌کند، می‌توان از این خروجی به عنوان احتمال تعلق آن داده به کلاس خاص یاد کرد.

انتقال هموار: ترکیب این دو تابع فعال‌ساز باعث ایجاد انتقال هموار بین لایه‌های استخراج ویژگی و لایهٔ طبقه‌بندی می‌شود. این شرایط تضمین می‌کند که ویژگی‌های آموخته‌شده مناسب هستند و منجر به پیش‌بینی مناسب می‌شود.

یک مورد که لازم است که به آن توجه کرد این است که با این که این دو تابع فعال‌ساز به طور معمول در شبکه‌های مختلف مورد استفاده قرار می‌گیرند، اما اگر به ساختار و خروجی هر یک دقت کنیم متوجه می‌شویم که پس از عبود داده‌ها از تابع فعال‌ساز ReLU مقادیر خروجی به ازای مقادیر مثبت، همان مقدار باقی می‌مانند در صورتی که مقادیر منفی در خروجی صفر می‌شود. حال این مقادیر

مثبت و یا صفر در تابع فعال‌ساز بعدی (سیگموید) که به ازای مقادیر مثبت مقداری بین ۰.۵ و ۱ خروجی می‌دهد و به ازای مقادیر منفی مقداری بین ۰ و ۰.۵ خروجی می‌دهد، بین ۰.۵ و ۱ مپ می‌شوند. این فرآیند ممکن است کمی تصمیم‌گیری را لایه آخر محدود کند.

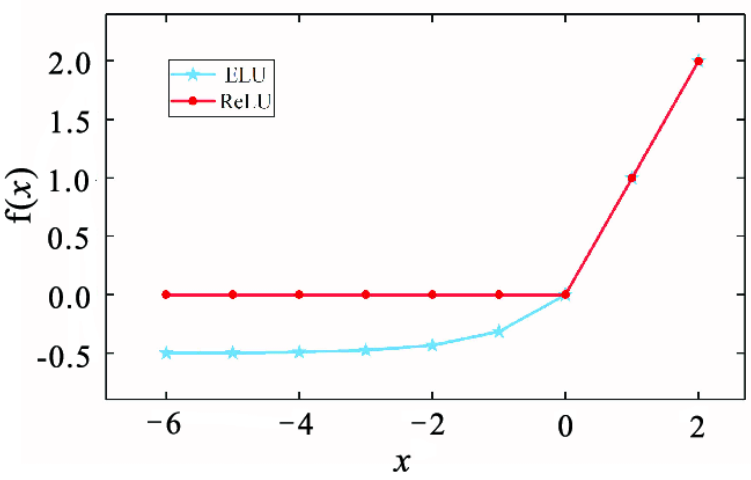
۲.۱ یک جایگزین برای ReLU در معادله ۱ آورده شده است. ضمن محاسبهٔ گرادیان آن، حداقل یک مزیت آن نسبت به ReLU را توضیح دهید

گرادیان این تابع به صورت زیر است.

پارامتر α در این تابع تعیین کنندهٔاندازهٔ خروجی تابع در مقادیر منفی ورودی است. α معمولا مقداری مثبت و کوچک در نظر گرفته می‌شود. مثلا ۰.۱ یا ۰.۰۵.

یک مشکل مهم تابع ReLU مقادیر خروجی این تابع به ازای ورودی‌های منفی است. زمانی که ورودی منفی باشد، تابع ReLU مقدار خروجی نورون را صفر می‌کند و برای همهٔ به‌روزرسانی‌های بعدی همین مشکل باقی می‌ماند. این مشکل با عنوان dying ReLU مطرح است. تابع ELU با اجازه‌دادن عبور مقادیر منفی برای ورودی، این مشکل را بر طرف می‌کند. این کار باعث فعال باقی ماندن نورون می‌شود.

در شکل ۱ تصویری از این دو تابع و تفاوت در خروجی مقادیر منفی آورده شده است.



شکل ۱: توابع فعال‌ساز Relu و ELU

۳.۱ به کمک یک نورون ساده، پرسپترون و یا نورون Pitts-McCulloch شبکه ای طراحی کنید که ...

ابتدا کتابخانه‌های مورد نیاز را فراخوانی می‌کنیم.

```python
#import library
import numpy as np
import itertools
import matplotlib.pyplot as plt
```

Code 1: import libraries (Python)

سه نقطه داده شده برای سه راس مثلث مورد نظر به این صورت هستند.

$$(2,2), \quad (1,0), \quad (3,0)$$

سه خط برای سه ضلع مثلث باید محاسبه شوند که هر یک از این سه خط را به صورت جداگانه به عنوان یک خط جداکنندهٔ دوکلاس می‌توان در نظر گرفت. برای هر یک از این سه طبقه‌بندی، یک نورون با دو ورودی و یک آستانه باید در نظر گرفت.

۱. برای خط گذرنده از دو نقطهٔ $(2,2)$ و $(1,0)$:

$$\begin{aligned} m &= \frac{y_2 - y_1}{x_2 - x_1} \\ &= \frac{0-2}{1-2} \\ &= \frac{-2}{-1} \\ &= 2 \end{aligned}$$

Using point-slope form with $(2,2)$:

$$\begin{aligned} y - 2 &= 2(x-2) \\ y - 2 &= 2x - 4 \\ y &= 2x - 2 \end{aligned}$$

۲. برای خط گذرنده از دو نقطهٔ $(2,2)$ و $(3,0)$:

$$\begin{aligned} m &= \frac{y_2 - y_1}{x_2 - x_1} \\ &= \frac{0-2}{3-2} \\ &= \frac{-2}{1} \\ &= -2 \end{aligned}$$

Using point-slope form with $(2,2)$:

$$\begin{aligned} y - 2 &= -2(x-2) \\ y - 2 &= -2x + 4 \\ y &= -2x + 6 \end{aligned}$$

۳. برای خط گذرنده از دو نقطهٔ $(1,0)$ و $(3,0)$:

$$\begin{aligned} m &= \frac{y_2 - y_1}{x_2 - x_1} \\ &= \frac{0-0}{3-1} \\ &= \frac{0}{2} \\ &= 0 \end{aligned}$$

:$(1, 0)$ with form point-slope Using

$$y - 0 = 0(x - 1)$$

$$y = 0$$

در گام بعدی نورون McCluloch-Pitts را تعریف می‌کنیم. برای این کار یک کلاس تعریف می‌کنیم. در قسمت init این کلاس وزن‌ها و آستانه مورد نظر در مرحله مقداردهی اولیه داده می‌شود. در قسمت model مقادیر وزن و ورودی در هم ضرب شده و پس از مقایسه با مقدار آستانه به صورت دو کلاس ۰ و ۱ طبقه‌بندی می‌شوند.

```
#define muculloch pitts
class McCulloch_Pitts_neuron():
  def __init__(self , weights , threshold):
    self.weights = weights    #define weights
    self.threshold = threshold    #define threshold

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

Code 2: import libraries (Python)

کد آمده در ۱۵ برای فراخوانی کلاس نورون McCluloch-Pitts با وزن‌ها و آستانه‌ها متناسب با هر کد جداکننده نوشته شده است. در این تابع ابتدا کلاس نورون فراخوانی شده و سپس ورودی‌های مورد نظر به کلاس داده می‌شود برای اینکه کار طبقه‌بندی صورت گیرد

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

  z1 = neur1.model(np.array([x, y]))
  z2 = neur2.model(np.array([x, y]))
```

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

  # return str(z1) + str(z2)
  return list([z1])
```

Code 3: import libraries (Python)

حالا ابتدا باید به هر یک از این سه خط به عنوان یک خط جداکنندهٔ دو کلاس نگاه کرد. به طوری که هر بار با نورون McCluloch-Pitts نقاط به دو سمت این خط جدا شوند. بنابراین سه مرتبه باید این طبقه‌بندی دو کلاسه را ابتدا انجام داد.

سه خط محاسبه‌شده مقادیر وزن‌ها و آستانهٔ مورد نظر برای طبقه‌بندی‌های مورد نظر را تشکیل می‌دهند. بنابراین در مرحلهٔ اول سه طبقه بندی با نورون McCluloch-Pitts با این مقادیر اولیه داریم:

([0, 1], 0)

([2, -1], 2)

([-2, -1], -6)

در مرحلهٔ بعد باید نتیجهٔ این سه خط را با هم ترکیب کرد. این کار توسط یک نورون با سه ورودی صورت می‌گیرد. مقادیر اولیه این نورون به صورت زیر درنظر گرفته می‌شود:

([1, 1, 1], 3)

بنابراین با این مراحل ناحیهٔ مورد نظر باید به صورت زیر حاصل شود.

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

# Evaluate data points using the Area function
for i in range(num_points):
    z5_value = Area(x_values[i], y_values[i])
    if z5_value == [0]:  # z5 value is 0
        red_points.append((x_values[i], y_values[i]))
    else:  # z5 value is 1
        green_points.append((x_values[i], y_values[i]))

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

```
green_x, green_y = zip(*green_points)
```

Code 4: import libraries (Python)

ابتدا برای خط

$$y = 0$$

در کد ۱۵ در خط اول قسمت فراخوانی، مقادیر [۰،۱] را برای وزن‌دهی و مقدار ۰ را برای آستانه مقداردهی می‌کنیم. مقدار z۱ را به خروجی کلاس McCluloch-Pitts برای خط اول پس از دادن ورودی‌های مورد نظر نسبت می‌دهیم. در انتهای این تابع مقدار z۱ را که خروجی تابع برای خط اول بود را برمی‌گردانیم.
کد ۵ تعداد ۲۰۰۰ نقطه رندوم در دو بعد x و y تولید می‌کند.

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

Code 5: import libraries (Python)

در قسمت بعدی باید تمامی نقاط تولید شده را به نورون داده تا کار طبقه‌بندی برای این نقاط صورت بگیرد. در حلقه در نظر گرفته شده به تعداد نقاط رندوم کار طبقه‌بندی صورت می‌گیرد که اگر برای هر نقطه خروجی نورون ۱ یا ۰ بود، آن نقطه به هر گروه سبز یا قرمز تعلق می‌یابد.

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

# Evaluate data points using the Area function
for i in range(num_points):
    z5_value = Area(x_values[i], y_values[i])
    if z5_value == [0]:  # z5 value is 0
        red_points.append((x_values[i], y_values[i]))
    else:  # z5 value is 1
        green_points.append((x_values[i], y_values[i]))

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

```
green_x, green_y = zip(*green_points)
```

Code 6: import libraries (Python)

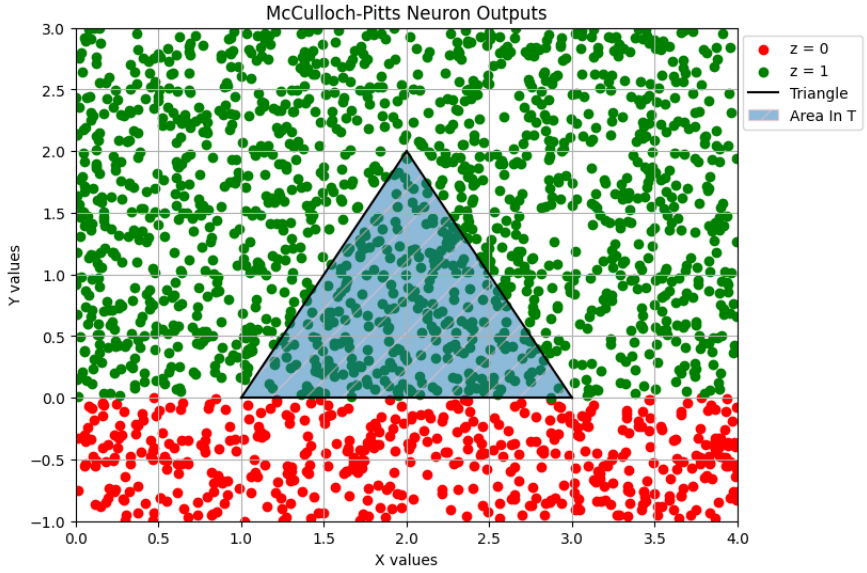
سپس با استفاده از کد ۳۱ نقاط پیش‌بینی‌شده متعلق به هر دو کلاس قرمز و سبز و همچنین مثلث مورد نظر سوال (برای مقایسه درستی خروجی طبقه‌بندی و ناحیه مثلث) رسم شده است

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


y_points = [0, 2, 0]
x_points = [1, 2, 3]

plt.plot(x_points[0:2], y_points[0:2], "black")
plt.plot(x_points[1:3], y_points[1:3], "black")
plt.plot([x_points[0], x_points[-1]], [y_points[0], y_points[-1]], "black",
    label='Triangle')
plt.fill(x_points, y_points, hatch='/', edgecolor='pink', alpha=0.5, label='
    Area In T')  # Add hatch inside the triangle

plt.grid(True)

# Set axis limits
plt.xlim(0, 4)
plt.ylim(-1, 3)

# Position the legends at the top and right
plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1.0))

# Save plot as PDF
plt.savefig('c.png', bbox_inches='tight')
```

```
29
30 plt.show()
```

Code 7: import libraries (Python)

خروجی این طبقه‌بندی به صورت شکل شکل ۲ آمده است. همانطور که قابل مشاهده است، چون می‌خواهیم ناحیه بالای خط جزو کلاس سبز باشد که داخل مثلث قرار بگیرد، بنابراین این کار به درستی صورت گرفته است



شکل ۲: بلوک دیاگرام مدل طبقه‌بند خطی

سپس برای خط

$$2x - y = 2$$

در کد ۱۵ در خط دوم قسمت فراخوانی، مقادیر [۲، -۱] را برای وزن‌دهی و مقدار ۲ را برای آستانه مقداردهی می‌کنیم. مقدار z۲ را به خروجی کلاس McCluloch-Pitts برای خط دوم پس از دادن ورودی‌های مورد نظر نسبت می‌دهیم. در انتهای این تابع مقدار z۲ را که خروجی تابع برای خط دوم بود را برمی‌گردانیم.

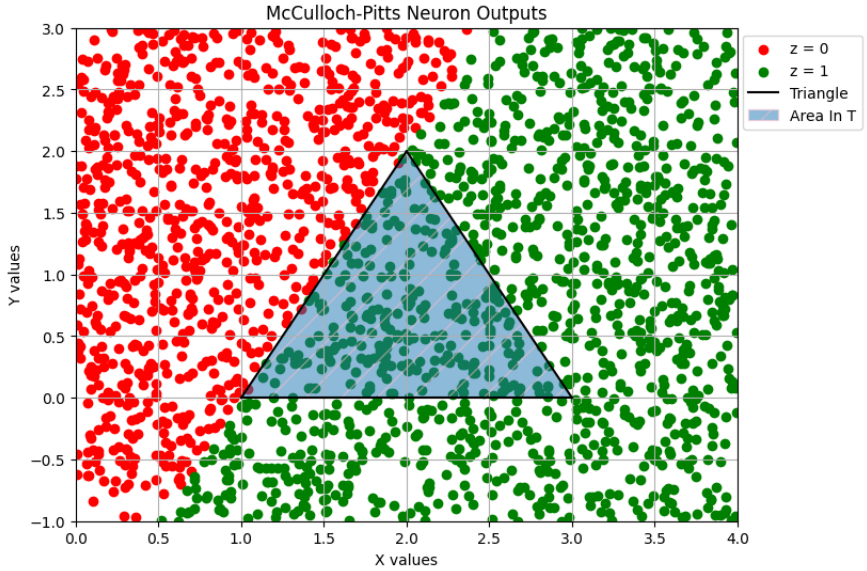
خروجی این طبقه‌بندی به صورت شکل ۳ قابل مشاهده است.

سمت چپ این خط به عنوان کلاس ۰ و سمت راست این خط به عنوان کلاس ۱ پیش‌بینی شده است که مطابق خواسته ما و ناحیهٔ مثلث است.

در نهایت برای خط

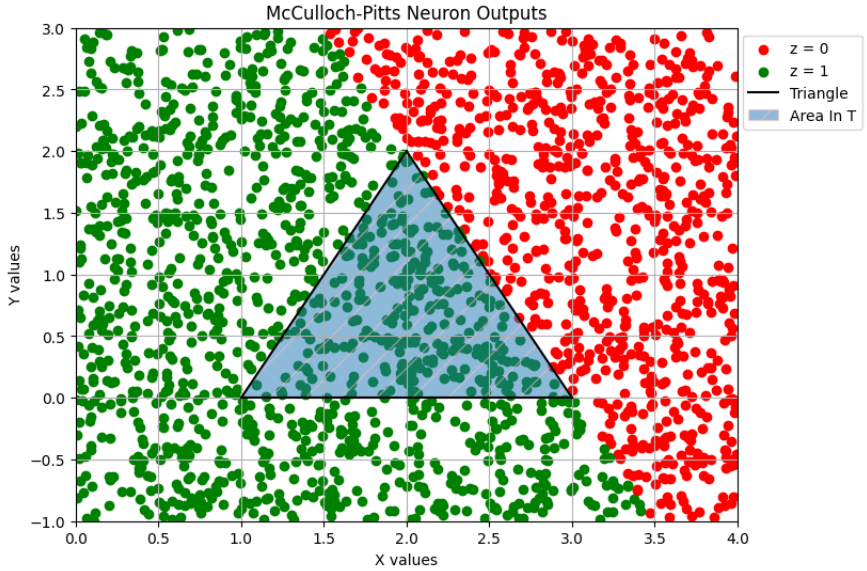
$$-2x - y = -6$$

بنابراین در کد ؟؟ در خط سوم قسمت فراخوانی، مقادیر [-۲، -۱] را برای وزن‌دهی و مقدار -۶ را برای آستانه مقداردهی می‌کنیم. مقدار z۳ را به خروجی کلاس McCluloch-Pitts برای خط سوم پس از دادن ورودی‌های مورد نظر نسبت می‌دهیم. در انتهای این تابع مقدار z۳ را که خروجی تابع برای خط سوم بود را برمی‌گردانیم.



شکل ۳: بلوک دیاگرام مدل طبقه‌بند خطی

خروجی طبقه‌بندی برای خط سوم به صورت زیر آمده است.



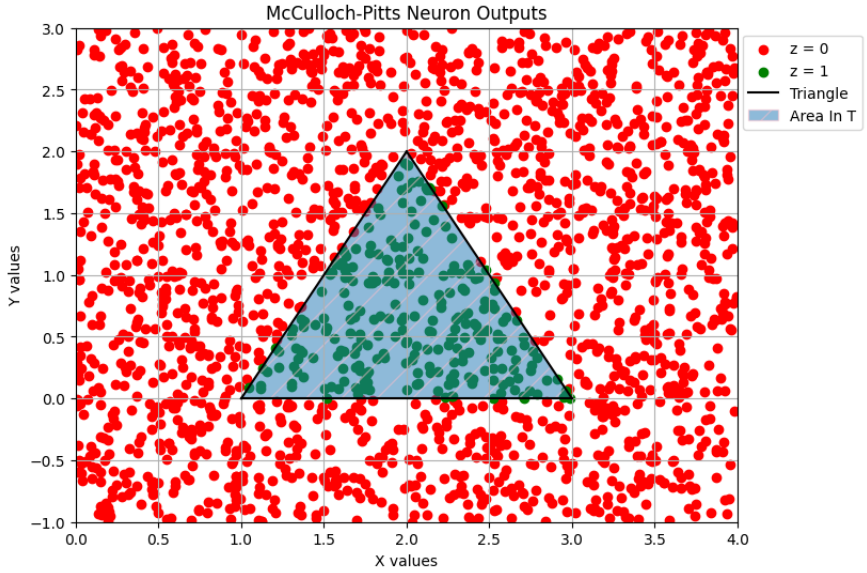
شکل ۴: بلوک دیاگرام مدل طبقه‌بند خطی

سمت راست این خط به عنوان کلاس ۰ و سمت چپ آن به عنوان کلاس ۱ طبقه‌بندی شده است که مورد نظر ماست. باید توجه داشت که اگر برعکس این اتفاق بیوفتد، می‌توان یک منفی در وزن‌ها و آستانه داده شده به نورون ضرب کرد و یا مقداردهی سبز و قرمز را به کلاس های طبقه‌بندی شده عوض کرد.

برای جمع بندی سه طبقه‌بندی انجام شده باید نتیجه هر سه طبقه‌بندی را ترکیب کرد. بنابراین در خط چهارم کد ۱۵ برای یک نورون دارای سه ورودی، وزن‌های [۱، ۱، ۱] و آستانهٔ ۳ را مقداردهی کرده و z۴ را به عنوان خروجی این نورون پس از دادن ورودی‌های آن (سه

خط مورد نظر) نسبت می‌دهیم.

خروجی این طبقه‌بندی را نیز به صورت زیر قابل مشاهده است.



شکل ۵: بلوک دیاگرام مدل طبقه‌بند خطی

در قسمت دوم این سوال خواسته شده است که اثر اضافه کردن دو تابع فعال ساز مختلف به فرآیند تصمیم گیری مورد بررسی قرار گیرد. تابع فعال‌ساز در مورد نورون McClloch-Pitts که مورد آموزش قرار نمی‌گیرد و ما با توجه به استدلال و ناحیهٔ مورد نظر مسئله مقادیر وزن و آستانه را مقداردهی می‌کنیم، زیاد مورد نیاز و جزء اساسی محسوب نمی‌شود. با این حال این کار در ادامه مورد ارزیابی قرار گرفته است.

به عنوان اولین تابع فعال‌ساز از تابع linear استفاده می‌کنیم. بیشترین میزان استفاده در این تابع فعال‌ساز در پروژه‌های مربوط به رگرسیون است اما گاها در مسایل مربوط به طبقه‌بندی نیز مورد استفاده قرار می‌گیرند. از آنجایی که در این سوال نیز کار طبقه‌بندی صورت می‌گیرد، اما به دلیل ماهیت نورون McClloch-Pitts و نتیجه متفاوت اضافه کردن سایر توابع فعال‌ساز،ابتدا از این تابع استفاده می‌کنیم که صرفا خروجی نهایی هر نورون را از قسمت‌های قبلی جدا کند اما تغییری در خروجی ایجاد نکند.

به عنوان تایع فعال‌ساز دوم نیز از تابع leaky relu استفاده می‌کنیم. مزیت این تابع نسبت به ReLU این است که از آنجایی که تابع ReLU خروجی هر ورودی‌ای که مقدار منفی داشته باشد را صفر می‌کند و از طرفی این مقدار خروجی با حد آستانه مورد ارزیابی قرار می‌گیرد، بنابراین مطلوب نیست که مقادیر منفی صفر شوند. با استفاده از تابع leaky relu با ضریب آلفایی که دارد، مقدارهای منفی را صفر نمی‌کند و منفی نگه می‌داد البته با اعمال تضعیف اندازه. این توابع به صورت زیر تعریف می‌شود.

```
def linear(x):
  return x

def leaky_relu(x, alpha=0.01):
```

```
    return max(alpha*x, x)
```

Code 8: import libraries (Python)

کدهای مربوط به تعریف کلاس نورون و تابع فراخوانی هر نورون با توجه به اضافه شدن تابع فعال‌ساز به صورت زیر تغییر پیدا کردند.

```
#define muculloch pitts
class McCulloch_Pitts_neuron_with_AF():
  def __init__(self , weights , threshold, activation_func = None):
    self.weights = weights    #define weights
    self.threshold = threshold    #define threshold
    self.activation_func = activation_func

  def model(self , x):
    #define model with threshold
    y = self.weights @ x
    y = y if self.activation_func == None else self.activation_func(y)
    if y >= self.threshold:
        return 1
    else:
        return 0


#define model for dataset
def Area_with_AF(x, y):
  neur1 = McCulloch_Pitts_neuron_with_AF([0, 1], 0, linear)
  neur2 = McCulloch_Pitts_neuron_with_AF([2, -1], 2, linear)
  neur3 = McCulloch_Pitts_neuron_with_AF([-2, -1], -6, linear)
  # neur3 = McCulloch_Pitts_neuron_with_AF([-2, -1], -0.06, linear)   #for
    leaky relu
  neur4 = McCulloch_Pitts_neuron_with_AF([1, 1, 1], 3, linear)

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

```
30
31    # return str(z1) + str(z2)
32    return list([z4])
```

Code 9: import libraries (Python)

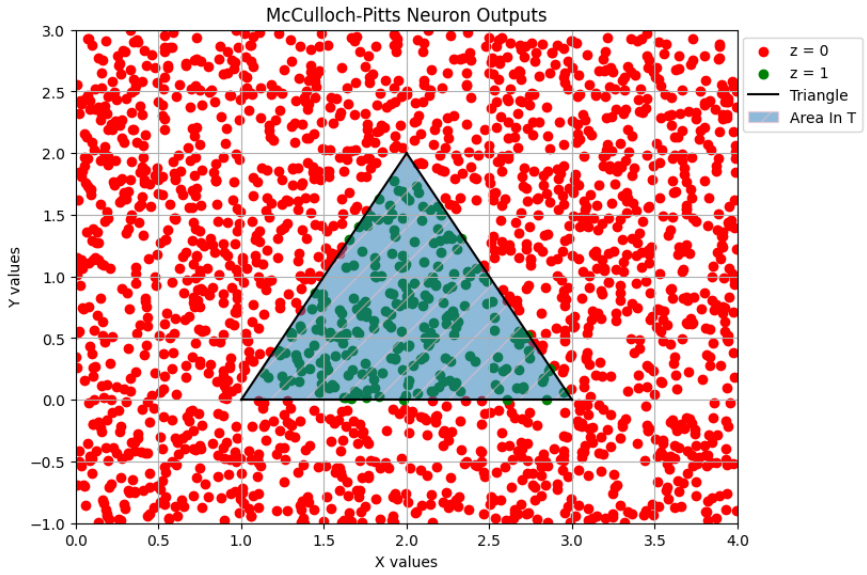
به عنوان تابع فعال‌ساز سوم هم از sigmiod استفاده شده است. مزیت استفاده از این تابع فعال‌ساز به دلیل مپ کردن مقدار خروجی نورون بین بازه ۰ و ۱ است. البته باید توجه داشت که باید برای هر یک از نورون‌ها مقدار آستانه را با توجه به sigmid تغییر داده شود. یعنی به صورت زیر مقادیر حد آستانه باید تغییر پیدا کند.

```
1  #define model for dataset
2  def Area_with_AF(x, y):
3    neur1 = McCulloch_Pitts_neuron_with_AF([0, 1], sigmoid(0), sigmoid)
4    neur2 = McCulloch_Pitts_neuron_with_AF([2, -1], sigmoid(2), sigmoid)
5    neur3 = McCulloch_Pitts_neuron_with_AF([-2, -1], sigmoid(-6), sigmoid)
6    neur4 = McCulloch_Pitts_neuron_with_AF([1, 1, 1], sigmoid(3), sigmoid)
7
8    z1 = neur1.model(np.array([x, y]))
9    z2 = neur2.model(np.array([x, y]))
10   z3 = neur3.model(np.array([x, y]))
11   z4 = neur4.model(np.array([z1, z2, z3]))
12
13   # return str(z1) + str(z2)
14   return list([z4])
```

Code 10: import libraries (Python)

خروجی شبکه با استفاده از هر یک از این توابع فعال‌ساز بصورت شکل ۶ است. مشاهده می‌شود که با استفاده از تابع فعال‌ساز linaer خروجی مورد نظر سوال که مشابه حالت قبلی نیز هست، به دست آمده است. تمامی نقاط رندوم تولید شده در داخل مثلث به رنگ سبز و متغلق به کلاس ۱ درآمده‌اند. این خروجی با همان وزن‌های استفاده شده در قسمت قبل بدست آمده است. یعنی با استفاده از این تابع فعال‌ساز که تغییری در خروجی آن نورون ایجاد نمی‌کند، نتیجه مطلوب بدون نیاز به طراحی مجدد حاصل می‌شود.

در حالت استفاده از تابع فعال‌ساز leaky relu به دلیل اینکه با توجه به آستانهٔ در نظر گرفته شده برای خط سوم که مقداری منفی شده بود، و اینکه این تابع اندازهٔ مقادیر منفی را تضعیف می‌کند، در حد آستانهٔ نورون سوم تغییری ایجاد شد که این شیفت مربوط به ضریب آلفای تابع فعال‌ساز اثر داده شود.



شکل ۶: خروجی هر سه حالت استفاده از دو تابع فعال‌ساز مختلف

۲ سوال دوم

۱.۲ دیتاست CWRU Bearing که در «مینی پروژهٔ شمارهٔ یک» با آن آشنا شدید را به خاطر آورید. علاوه بر دو کلاسی که در آن مینی پروژه در نظر گرفتید، با مراجعه به صفحهٔ داده های عیب در حالت k،۱۲ دو کلاس دیگر نیز از و X-۰۰۷@۶OR اضافه کنید. با انجام این کار یک کلاس دادهٔ سالم و سه کلاس طریق فایل های X-۰۰۷B۱ از داده های دارای سه عیب متفاوت خواهید داشت. در مورد این که هر فایل مربوط به چه نوع عیبی است به صورت کوتاه توضیح دهید. سپس در ادامه، تمام کارهایی که در بخش «۲» سوال دوم «مینی پروژهٔ یک» برای استخراج ویژگی و آماده سازی دیتا انجام داده بودید را روی دیتاست جدید خود پیاده سازی کنید. در قسمت تقسیم بندی داده ها، یک بخش برای «اعتبارسنجی» به بخش های «آموزش» و «آزمون» اضافه کنید و توضیح دهید که کاربرد این بخش چیست.

داده‌ها برای یاتاقان‌های معمولی و عیب‌های در درایو تک‌نقطه‌ای و در انتهای فن جمع‌آوری شد. داده‌ها در ۱۲۰۰۰ نمونه در ثانیه و ۴۸۰۰۰ نمونه در ثانیه برای آزمایش‌های یاتاقان انتهایی درایو جمع‌آوری شد. تمام داده های بلبرینگ انتهای فن در ۱۲۰۰۰ نمونه در ثانیه جمع آوری شد

هدف استفاده از این نوع دیتاست، تعمیر و نگهداری پیش‌بینی‌کننده در ماشین‌ها پیش‌بینی خرابی‌ها است. در ماشین های دوار، قطعه ای که بیشترین آسیب را متحمل می شود، بلبرینگ ها هستند. هدف اصلی این تحقیق تشخیص خرابی بلبرینگ با استفاده از حداقل مجموعه مشاهدات و انتخاب حداقل تعداد ویژگی است. مجموعه آزمایشی برای جمع آوری داده ها شامل یک موتور القایی الکتریکی ۰۳.۲ اسب بخار در یک سر، یک مبدل گشتاور در وسط، و یک دینامومتر در انتهای دیگر است که بار را شبیه سازی می کند. سنسورها، به‌ویژه شتاب‌سنج‌ها، روی یاتاقان‌های انتهای شفت موتور و روی فن داخل محفظه موتور قرار گرفتند و ۱۲۰۰۰ و ۴۸۰۰۰ نمونه در ثانیه از عیوب را جمع‌آوری کردند. سرعت و قدرت از طریق مبدل گشتاور بدست آمد و به صورت دستی ثبت شد.

نقص هایی با استفاده از تخلیه الکتریکی به بلبرینگ های SKF اضافه شد که باعث خرابی در تاج داخلی، توپ ها و مسیر بیرونی با قطرهای مختلف از ۰۰۷.۰ اینچ تا ۰۴.۰ اینچ شد. علاوه بر این، این آزمایش ها با تغییر سرعت چرخش و بار انجام شد. مجموعه داده شامل ۱۶۱ رکورد است که به چهار گروه تقسیم می شوند: ۴۸k خط پایه معمولی (داده های بدون خطا)، ۴۸k خطای انتهای درایو، ۱۲k خطای انتهای درایو و ۱۲k خطای انتهای فن. آنها حاوی اطلاعاتی بر اساس بارها و سرعت موتور هستند. در مورد نام فایل ها، حرف اول نشان دهنده موقعیت نقص، سه عدد بعدی قطر خرابی و آخرین عدد نشان دهنده بار است. به عنوان مثال، فایل IR۰۰۷-۰ دارای اطلاعات خرابی تاج داخلی است، با قطر شکست ۰۰۷.۰ اینچ برای بار موتور ۰ اسب بخار. هر فایل داده در مجموعه داده CWRU از داده هایی با طول های مختلف تشکیل شده است و مضرب ۲ نیست، علاوه بر این مجموعه ای بزرگ، متنوع و پیچیده است. [b۴]

ابتدا مجموعه‌داده‌های مورد خواسته مسئله و کتابخانه‌های مورد نیاز را فراخوانی می‌کنیم. سپس چهار دیتاست سالم و خطا را در متغیرهای مورد نظر وارد می‌کنیم. با استفاده از ویژگی key می‌توان به سرستون‌های هر کدام از دستاست‌ها دسترسی پیدا کرد.

```
%cd /content

!gdown 1VoGTdaylo4ytvh6wMOVTxelgxWlQfvyu
!gdown 10IycnfvXVGsMVRQ58-Qy-l4YpvOfYY-_
!gdown 1u45bE0rKvKVNDkqPtmMEQLBTd1wfo6pc
!gdown 1EE1bGnHm5T10pgmgoI99FEQbVyBbY5Ky

from scipy.io import loadmat
import numpy as np
from scipy.stats import skew, kurtosis
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

# Load the .mat file
normal_data = loadmat('/content/97.mat')
fault_data_IR007 = loadmat('/content/105.mat')
fault_data_BB007 = loadmat('/content/118.mat')
fault_data_OR007 = loadmat('/content/130.mat')

print(normal_data.keys())
print(fault_data_IR007.keys())
print(fault_data_BB007.keys())
print(fault_data_OR007.keys())
```

Code 11: import dataset and library

خروجی سرستون‌های دیتاست‌ها بصورت شکل ۷ است.

```
dict_keys(['__header__', '__version__', '__globals__', 'X097_DE_time', 'X097_FE_time', 'X097RPM'])
dict_keys(['__header__', '__version__', '__globals__', 'X105_DE_time', 'X105_FE_time', 'X105_BA_time', 'X105RPM'])
dict_keys(['__header__', '__version__', '__globals__', 'X118_DE_time', 'X118_FE_time', 'X118_BA_time', 'X118RPM'])
dict_keys(['__header__', '__version__', '__globals__', 'X130_DE_time', 'X130_FE_time', 'X130_BA_time', 'X130RPM'])
```

شکل ۷: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

سپس از مجموعه‌دادهٔ سالم ستون X۰۹۷-DE-time و از مجموعه‌داده‌های خطا ستون‌های مربوط به time را به عنوان نمونه با توجه به خواسته مسئله انتخاب می‌کنیم و در متغیرهای مناسب می‌ریزیم. برای داشتن دید مناسب از ابعاد مجموعه‌داده ابتدا نوع دیتای ستون‌های مورد نظر و همچنین ابعاد آن‌ها را خروجی می‌گیریم.

```
normal_data_variable = normal_data['X097_DE_time']
print(type(normal_data_variable))
print(normal_data_variable.shape)

fault_data_IR007_variable = fault_data_IR007['X105_DE_time']
print(fault_data_IR007_variable.shape)

fault_data_BB007_variable = fault_data_BB007['X118_DE_time']
print(fault_data_BB007_variable.shape)

fault_data_OR007_variable = fault_data_OR007['X130_DE_time']
print(fault_data_OR007_variable.shape)
```

Code 12: import dataset and library

خروجی مورد نظر به صورت شکل ۸ است.

```
<class 'numpy.ndarray'>
(243938, 1)
(121265, 1)
(122571, 1)
(121991, 1)
```

شکل ۸: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

در این با انتخاب تعداد نمونه برابر ۱۲۰ و اندازه طول ۲۲۰ از هر کلاس جداسازی شده است. این ۱۲۰ نمونه با طول ۲۲۰ به صورت پشت سر هم قرار گرفته اند. در نهایت ابعاد هر کدام برابر (۱،۲۶۴۰۰) می‌شود. سپس برا تکمیل ابعاد مجموعه‌داده با استفاده از دستور reshape، دیتاها به صورتی درمی‌آیند که سطرهای آن دربرگیرندهٔ ۱۲۰ نمونهٔ مورد خواسته مسئله باشد.

```python
# Extract 10 samples, each containing 5 data points
n_samples = 120
n_data = 220
ext_normal_data = normal_data_variable[:n_samples * n_data, 0]
ext_fault_data_IR007 = fault_data_IR007_variable[:n_samples * n_data, 0]
ext_fault_data_BB007 = fault_data_BB007_variable[:n_samples * n_data, 0]
ext_fault_data_OR007 = fault_data_OR007_variable[:n_samples * n_data, 0]
```

Code 13: import dataset and library

```python
# Reshape the extracted data to have 10 rows and 5 columns
ext_normal_data = ext_normal_data.reshape(n_samples, n_data)
ext_fault_data_IR007 = ext_fault_data_IR007.reshape(n_samples, n_data)
ext_fault_data_BB007 = ext_fault_data_BB007.reshape(n_samples, n_data)
ext_fault_data_OR007 = ext_fault_data_OR007.reshape(n_samples, n_data)


print(ext_normal_data.shape)
print(ext_fault_data_IR007.shape)
print(ext_fault_data_BB007.shape)
print(ext_fault_data_OR007.shape)
```

Code 14: import dataset and library

خروجی این کد نیز بصورت زیر است.

```
(120, 220)
(120, 220)
(120, 220)
(120, 220)
```

شکل ۹: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

با توجه به جدول مسئله تعداد هشت عدد از روش‌های ذکرشده انتخاب شد. با استفاده از این این روش‌ها ویژگی‌های دیتاست مورد استخراج می‌شود.

تابع feature-extraction طوری کدنویسی‌شده است که بتواند محاسبهٔ هر کدام از این روش‌ها و اعمال بر روی دیتاست‌های ورودی را انجام دهد. برای محاسبهٔ هر کدام از این روش‌ها از کتابخانهٔ numpy استفاده شده است. قسمت axis=۱ باعث می‌شود که الگوریتم

مورد نظر بر روی سطرها که نشان‌دهندهٔ نمونه‌های دیتاست هستند اعمال شود. در قسمت بعدی خروجی ویژگی‌های استخراج‌شده با دستور column-stack بصورت ستونی به یکدیگر متصل شدند تا هر دستاست با ابعاد (۸، ۱۲۰) ساخته شود. دی قسمت پایانی نیز یک ستون به دیتاست اضافه شده است که نشان‌دهنده برچسب کلاس است. این ستون با توجه به نوع کلاس از ۰ تا ۳ متغیر است.

```
def feature_extraction (data, class_num):
  #Normal data feature extraction
  standard_deviations = np.std(data, axis=1)
  skewnesses = skew(data, axis=1)
  kurtoses = kurtosis(data, axis=1)
  peak_to_peaks = np.ptp(data, axis=1)
  root_mean_squares = np.sqrt(np.mean(np.square(data), axis=1))
  means = np.mean(data, axis=1)
  absolute_means = np.mean(np.abs(data), axis=1)
  peaks = np.max(data, axis=1)

  print("Standard Deviations:", standard_deviations.shape)
  print("Skewnesses:", skewnesses.shape)
  print("Kurtoses:", kurtoses.shape)
  print("Peak to Peaks:", peak_to_peaks.shape)
  print("Root Mean Squares:", root_mean_squares.shape)
  print("Means:", means.shape)
  print("Absolute Means:", absolute_means.shape)
  print("Peaks:", peaks.shape)

  ext_feature_dataset = np.column_stack((standard_deviations, skewnesses,
   kurtoses, peak_to_peaks,
                                root_mean_squares, means, absolute_means,
   peaks))


  ext_feature_dataset = np.hstack((ext_feature_dataset, class_num * np.ones((
   len(ext_feature_dataset), 1))))
  print("final dataset :",ext_feature_dataset.shape)


```

```
29   return ext_feature_dataset
```

Code 15: import dataset and library

در ادامه برای استفاده از این تابع برای هر کدام از دیتاهای نرمال و خطا، کد زیر به همراه خروجی(مشابه برای هر چهار نوع دیتا) قابل مشاهده است.

```
normal_ext_feature_dataset = feature_extraction(ext_normal_data, 0)
IR007_ext_feature_dataset = feature_extraction(ext_fault_data_IR007, 1)
BB007_ext_feature_dataset = feature_extraction(ext_fault_data_BB007, 2)
OR007_ext_feature_dataset = feature_extraction(ext_fault_data_OR007, 3)


Standard Deviations: (120,)
Skewnesses: (120,)
Kurtoses: (120,)
Peak to Peaks: (120,)
Root Mean Squares: (120,)
Means: (120,)
Absolute Means: (120,)
Peaks: (120,)
final dataset : (120, 9)
```

Code 16: import dataset and library

در نهایت برای این که دیتاست تشکیل شده مانند یک دیتاستی که دارای کلاس‌های مختلف است به صورت استاندارد دربیاید، چهار ماتریس تشکیل شده با هم ادغام شده است. کد و خروجی آن بصورت زیر قابل بررسی است.

```
final_data = np.concatenate((normal_ext_feature_dataset,
    IR007_ext_feature_dataset, BB007_ext_feature_dataset,
    OR007_ext_feature_dataset), axis=0)
final_data.shape
```

Code 17: import dataset and library

در این قسمت پس از مخلوط‌کردن دیتاها با دستور shuffle، دیتاهای مربوط به ویژگی‌های دیتاست (همه ستون‌ها به‌جز ستون اخر) به متغیر X و دیتای برچسب به متغیر y تعلق پیدا می‌کند. در نهایت دیتاها به دو دستهٔ آموزش و ارزیابی تقسیم می‌شوند. نسبت داده‌های

```
(480, 9)
```

شکل ۱۰: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

آموزش به ارزیابی در این قسمت با توجه به اندازه کلی دیتاست برابر ۱۰ درصد در نظر گرفته شده است. در سوال خواسته شده است که یک بخش اعتبارسنجی نیز در نظر گرفته شود. این بخش در ادامه از قسمت دیتاستِ آموزش جدا خواهد شد.

تقسیم به بخش اعتبارسنجی به عنوان یک گام اساسی در یادگیری ماشین شناخته می‌شود. معمول است که این بخش باید از دیتاست آموزش جدا شود، یعنی پس از اینکه دیتاست تست را از آموزش جدا کردیم، از دیتاست باقی‌مانده، بخش اعتبارسنجی را جدا می‌کنیم. عدم اشتراک داده در این سه بخش بسیار مهم است. از دلایل اهمیت و کاربردهای این بخش می‌توان به این موارد اشاره کرد:

ارزیابی مدل: از داده‌های اعتبارسنجی برای ارزیابی مدل هنگام آموزش استفاده می‌شود. کمک به تیون‌کردن هایپرپارامترها و تشخیص اینکه کدام مدل با چه پارامترهایی مناسب‌تر است، از مزایای استفاده از این بخش است.

جلوگیری از اورفیت و آندرفیت شدن مدل: با ارزیابی نمودارهای مربوط به loss داده‌های اعتبارسنجی و مقایسه با نمودار آموزش، می‌توان پیش از تمام شدن کل آموزش به ایجاد شدن این مشکلات پی برد و مشکلات را حل کرد.

توقف زودهنگام: با مقایسهٔ نمودارهای آموزش و اعتبارسنجی می‌توان زمانی را که مقدار loss آموزش و اعتبارسنجی از حدی کمتر نمی‌شوند را تشخیص داده و از آموزش بیشتر و صرف هزینهٔ زمانی و سخت‌افزاری اضافه جلوگیری کرد.

کد مورد نظر و خروجی نهایی بصورت زیر است.

```
shuffled_final_data = shuffle(final_data)

X = shuffled_final_data[:, :-1]
y = shuffled_final_data[:, -1]
print("X :",X.shape)
print("y :",y.shape)

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Code 18: import dataset and library

```
((432, 8), (48, 8), (432,), (48,))
```

شکل ۱۱: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

اما این ابعاد برای برچسب‌ها در ادامه مشکل‌ساز خواهد بود بنابراین نیاز است که با تغییر ابعاد، آن را به صورتی در بیاوریم که بعد ۱ در قسمت ستون وارد شود. با استفاده از دستور reshape این کار را انجام می‌دهیم.

```
# Reshape y_train and y_test
```

```
y_train = np.reshape(y_train, (-1, 1))
y_test = np.reshape(y_test, (-1, 1))
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Code 19: import dataset and library

در نهایت ابعاد دیتاهای مورد نظر به صورت زیر در می‌آید.

```
((432, 8), (48, 8), (432, 1), (48, 1))
```

شکل ۱۲: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

نیاز است که فرایند نرمال‌سازی پس از تقسیم داده‌ها به آموزش و تست صورت بگیرد چرا که داده‌های تست نباید توسط مدل دیده شوند و با اعمال ویژگی آن ها در فرایند نرمال سازی، برخی ویژگی‌ها وارد مدل می‌شود و ارزیابی صورت گرفته درست نخواهد بود. درنتیجه فرآیند نرمال‌سازی با استفاده از اطلاعات داده‌ها آموزش بر روی داده‌های آموزش و ارزیابی صورت می‌پذیرد.

برای نرمال‌سازی داده‌ها در این سوال از روش MinMaxScaler استفاده می‌کنیم. کد مورد استفاده بصورت زیر است.

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Initialize MinMaxScaler
scaler = MinMaxScaler()

scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

Code 20: import dataset and library

۲.۲ یک مدل MLP ساده با ۲ لایهٔ پنهان یا بیش تر بسازید. بخشی از داده های آموزش را برای اعتبارسنجی کنار بگذارید و با انتخاب بهینه ساز 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 داد های آزمون را با استفاده ماتریس درهم ریختگی و report-classification نشان داده و نتایج به صورت دقیق تحلیل کنید.

ابتدا لازم است کتابخانه‌های مورد نیاز را فراخوانی کنیم.

```
from keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer

from sklearn.neural_network import MLPClassifier, MLPRegressor
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

import tensorflow as tf
from tensorflow import keras
from keras import preprocessing
from keras.models import Sequential
from keras.layers import Dense
```

Code 21: import dataset and library

کار طبقه‌بندی در این سوال را با استفاده از یک شبکه MLP با دو لایه پنهان انجام می‌دهیم. از فریم‌ورک کراس استفاده شده است. جزئیات شبکه دولابه به صورت زیر قابل مشاهده است. با دستور Sequential مدلی فراخوانی می‌شود که می‌توان در آن لایه‌های مورد نظر شبکه را در آن وارد کرده و تعریف کرد. سپس در دو نوبت با استفاده از دستور add لایهٔ پنهان اولی با ۱۵ نورون و لایه دوم را با ۸ نورون ایجاد می‌کنیم. با توجه به اینکه تعداد کلاس‌های دیتاست ۴ تاست، باید لایه طبقه‌بندی آخر را با ۴ نورون ایجاد کنیم. با استفاده از دستور summery می‌توان جزئیات شبکه را مشاهده کرد. تابع فعال‌ساز لایه‌های پنهان ReLU و لایه طبقه‌بند softmax در نظر گرفته شده اند.

```
model_2 = Sequential()

# Add the first hidden layer with 50 neurons and linear activation function
model_2.add(Dense(50, activation='relu', input_shape=(x_train.shape[1],)))

# Add the second hidden layer with 30 neurons and linear activation function
model_2.add(Dense(30, activation='relu'))

# Add an output layer with 1 neuron and linear activation function
model_2.add(Dense(4, activation='softmax'))

model_2.summary()
```

Code 22: import dataset and library

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 15)                135

 dense_4 (Dense)             (None, 8)                 128

 dense_5 (Dense)             (None, 4)                 36

=================================================================
Total params: 299 (1.17 KB)
Trainable params: 299 (1.17 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

شکل ۱۳: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

با استفاده از کد ۳ می‌توان شبکه MLP را آموزش داد. با دستور compile و با در نظر گرفتن بهینه‌ساز Adam و تابع اتلاف Spar-seCategoricalCrossentropy به تعداد ۱۰۰ ایپاک مدل را آموزش می‌دهیم. مقدار بخش اعتبارسنجی را برابر ۲۰ درصد قرار میدهیم. مقدار بچ‌سایز را هم برابر ۱۰ قرار می‌دهیم. با این کار داده‌ها در بسته‌های ۱۰ تایی مورد آموزش قرار می‌گیرند.

```
model_2.compile(optimizer='adam', loss='SparseCategoricalCrossentropy',
    metrics=['accuracy'])
history = model_2.fit(x_train, y_train, validation_split=0.2, epochs=100 ,
    batch_size=10)
```

Code 23: import dataset and library

پس از پایان آموزش باید فرآیند آموزش را مورد ارزیابی قرار داد. یک روش معمول و مناسب برای اینکار رسم و مقایسه نمودارهای اتلاف و accuracy داده‌های آموزش و اعتبارسنجی است.

```
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='train')   # Training loss
plt.plot(history.history['val_loss'], label='val')  # Validation loss

plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```

```
Epoch 1/100
35/35 [==============================] - 1s 9ms/step - loss: 1.4030 - accuracy: 0.2522 - val_loss: 1.3863 - val_accuracy: 0.2529
Epoch 2/100
35/35 [==============================] - 0s 3ms/step - loss: 1.3588 - accuracy: 0.2522 - val_loss: 1.3449 - val_accuracy: 0.2989
Epoch 3/100
35/35 [==============================] - 0s 3ms/step - loss: 1.3188 - accuracy: 0.4609 - val_loss: 1.2926 - val_accuracy: 0.4943
Epoch 4/100
35/35 [==============================] - 0s 3ms/step - loss: 1.2701 - accuracy: 0.5304 - val_loss: 1.2431 - val_accuracy: 0.5287
Epoch 5/100
35/35 [==============================] - 0s 3ms/step - loss: 1.2058 - accuracy: 0.5362 - val_loss: 1.1662 - val_accuracy: 0.5747
Epoch 6/100
35/35 [==============================] - 0s 3ms/step - loss: 1.1059 - accuracy: 0.5391 - val_loss: 1.0537 - val_accuracy: 0.5517
Epoch 7/100
35/35 [==============================] - 0s 4ms/step - loss: 0.9851 - accuracy: 0.5710 - val_loss: 0.9337 - val_accuracy: 0.5632
Epoch 8/100
35/35 [==============================] - 0s 3ms/step - loss: 0.8679 - accuracy: 0.5855 - val_loss: 0.8217 - val_accuracy: 0.6552
Epoch 9/100
35/35 [==============================] - 0s 4ms/step - loss: 0.7654 - accuracy: 0.6522 - val_loss: 0.7251 - val_accuracy: 0.7356
Epoch 10/100
35/35 [==============================] - 0s 3ms/step - loss: 0.6766 - accuracy: 0.7333 - val_loss: 0.6381 - val_accuracy: 0.8276
Epoch 11/100
35/35 [==============================] - 0s 3ms/step - loss: 0.5991 - accuracy: 0.8145 - val_loss: 0.5618 - val_accuracy: 0.8851
Epoch 12/100
35/35 [==============================] - 0s 4ms/step - loss: 0.5305 - accuracy: 0.9246 - val_loss: 0.4933 - val_accuracy: 0.9080
Epoch 13/100
35/35 [==============================] - 0s 3ms/step - loss: 0.4709 - accuracy: 0.9507 - val_loss: 0.4385 - val_accuracy: 0.9425
Epoch 14/100
35/35 [==============================] - 0s 3ms/step - loss: 0.4210 - accuracy: 0.9768 - val_loss: 0.3897 - val_accuracy: 0.9655
Epoch 15/100
35/35 [==============================] - 0s 4ms/step - loss: 0.3760 - accuracy: 0.9797 - val_loss: 0.3511 - val_accuracy: 0.9885
Epoch 16/100
```

شکل ۱۴: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

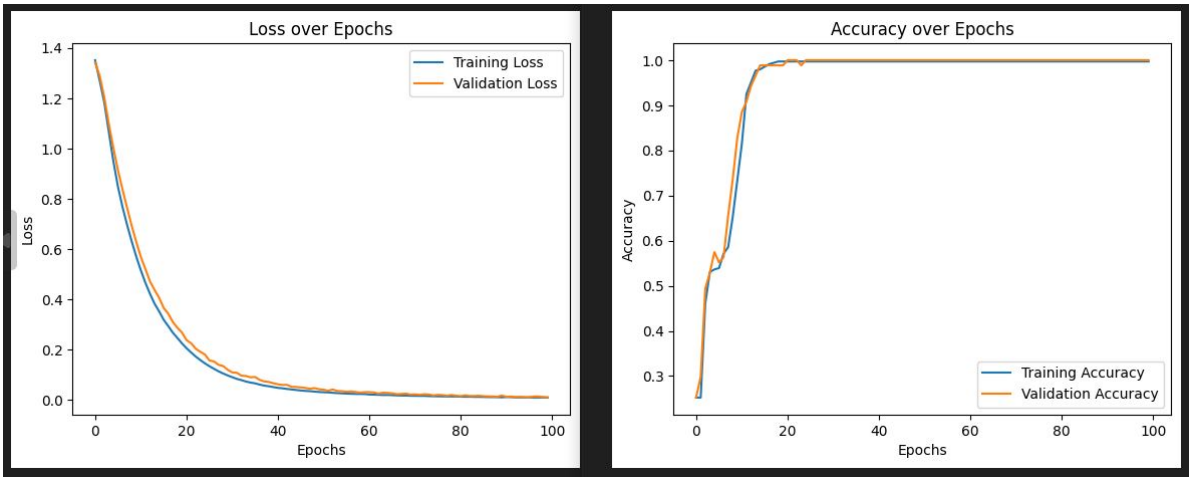
```
10
11 plt.plot(history.history['accuracy'], label='Training Accuracy')
12 plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
13 plt.xlabel('Epochs')
14 plt.ylabel('Accuracy')
15 plt.title('Accuracy over Epochs')
16 plt.legend()
17
18 plt.show()
```

Code 24: import dataset and library

نمودارهای مربوط به اتلاف و accuracy بصورت شکل ۱۵ است. همانطور که قابل مشاهده است نمودرا اتلاف در دوره‌های آموزش مختلف فایل هرگونه علامتی از اورفیت یا اندرفیت است. نمودار اعتبارسنجی فرآیند کاهشی خود را همگام با نمودار آموزش طی کرده است و تا اپاک حدود ۴۰ روند کاهشی با شیب مناسبی داشته و پس از آن میزان کاهش بسیار کم شده است. برای نمودار Accuracy نیز تا اپاک ۱۸ نمودار افزایشی است و سپس به میزان ۱ میرسد که این نیز نشانهٔ آموزش مناسب است. می‌توان نتیجه گرفت که نیازی به آموزش تا ۱۰۰ دورهٔ آموزشی نیست و با توجه به عدم بهبود نمودارها و رسیدن به مقدار نهایی، می‌توان توقف زودهنگام را اجرایی کرد.

در گام بعدی باید نتیجهٔ تست روی داده‌های ارزیابی مورد بررسی قرار گیرد. جهت انجام این‌کار از کد زیر استفاده می‌کنیم. ابتدا داده‌های مربوط به تست را به مدل جهت انجام طبقه‌بندی می‌دهیم. سپس با توجه به احتمالاتی بودن پیش‌بینی مدل، آن طبقه‌ای که ضریب احتمال بالاتری دارد را به عنوان پیش‌بینی نهایی درنظر می‌گیریم. نتیجهٔ تست روی داده‌های ارزیابی به صورت شکل ۱۶ است.

```
1 y_pred_2 = model_2.predict(x_test)
```



شکل ۱۵: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

```python

from sklearn.metrics import accuracy_score

# Assuming y_test contains the true labels
y_pred_classes = np.argmax(y_pred_2, axis=1)  # Convert predicted
    probabilities to class labels
accuracy = accuracy_score(y_test, y_pred_classes)
print("Accuracy:", accuracy)
```

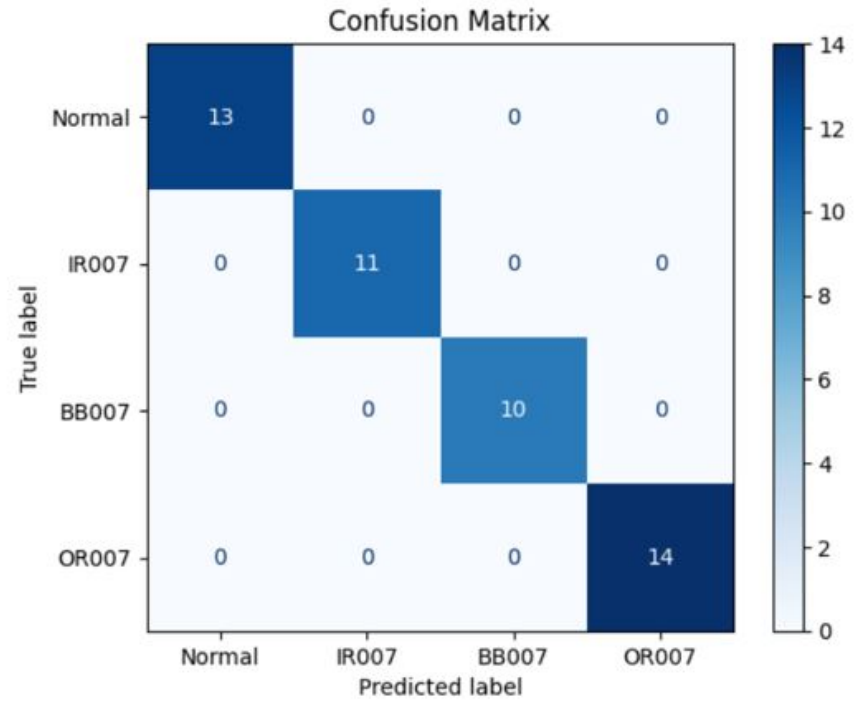
Code 25: import dataset and library

```
2/2 [==============================] - 0s 4ms/step
Accuracy: 1.0
```

شکل ۱۶: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

با استفاده از classification-report و ماتریس درهم‌ریختگی برای داده‌های ازریابی به بررسی نتایج دقیق‌تر می‌پردازیم. تعداد ۴۸ داده از ۴ کلاس مورد ارزیابی قرار گرفت که در تمامی کلاس‌ها پیش‌بینی به درستی اتفاق افتاده است. ماتریس درهم‌ریختگی نیز تمامی داده‌ها روی قطر اصلی هستند و پیش‌بینی اشتباهی صورت نگرفته است.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	13
1.0	1.00	1.00	1.00	11
2.0	1.00	1.00	1.00	10
3.0	1.00	1.00	1.00	14
accuracy			1.00	48
macro avg	1.00	1.00	1.00	48
weighted avg	1.00	1.00	1.00	48



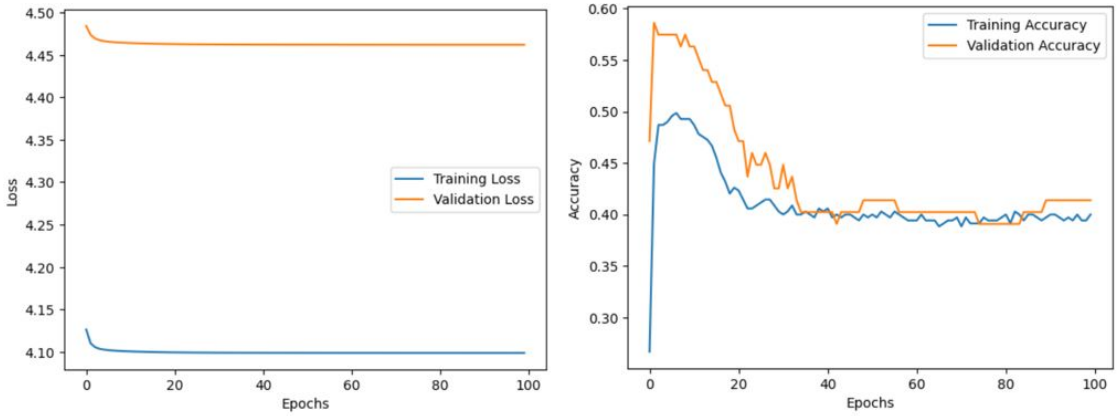
شکل ۱۷: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

۳.۲ فرآیند سوال قبل را با یک بهینه ساز و تابع اتلاف جدید انجام داده و نتایج را مقایسه و تحلیل کنید. بررسی کنید که آیا تغییر تابع اتلاف می تواند در نتیجه اثرگذار باشد؟

برای اینکه بتوان مقایسهٔ مناسبی بین بهینه‌سازها و توابع اتلاف داشت، تمامی پارامترهای قسمت قبل را برای آموزش روی بهینه‌ساز و تابع اتلاف جدید بدون تغییر باقی می‌گذاریم. در این قسمت از بهینه‌ساز SGD و تابع اتلاف KLDivergence استفاده می‌کنیم. باید توجه داشت که تابع اتلاف SparseCategoricalCrossentropy که در قسمت قبل استفاده شد، معمول‌تر و مورداستفاده‌تر است و تابع اتلاف جدید ممکن است به مطلوبی قبلی نباشد.

با آموزش مجدد به نمودارهای اتلاف و معیار شکل ۱۸ می‌رسیم. همانطور که قابل مشاهده است، نمودارهای اتلاف در مرحلهٔ آموزش نتوانسته است به خوبی کاهش پیدا کند و به طور کلی آموزش مناسبی صورت نگرفته است. نمودار اعتبارسنجی نیز با فاصلهٔ زیاد و کاهش کم مواجه شده است. نمودار معیار نیز پس از افزایش نسبی در همان ایپاک‌های اولیه افتاده است. همه چی از آموزش نامناسب خبر می‌دهد

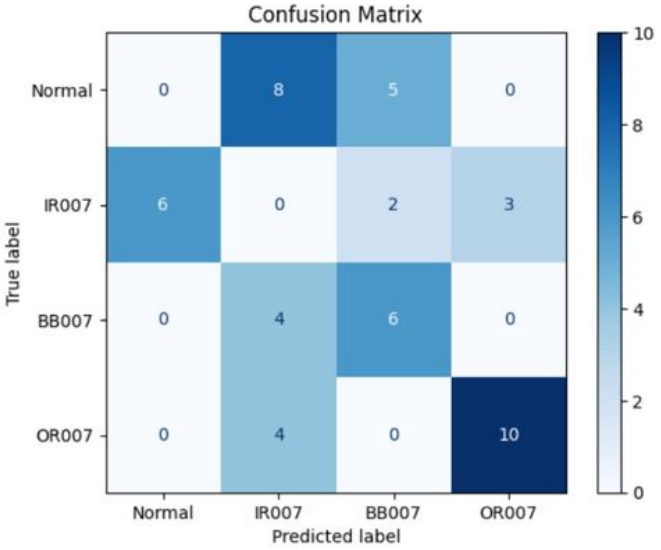
که یا ناشی از بهینه‌ساز است یا تابع اتلاف.



شکل ۱۸: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

در این حالت معیارها در شکل ۱۹ در کلاس‌های مختلف نیز نشان‌دهندهٔ آموزش نامناسب است.

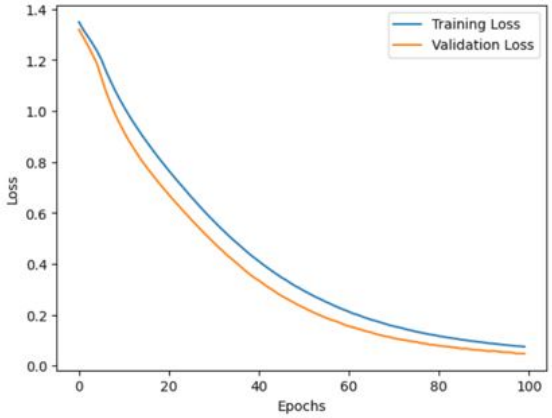
	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	13
1.0	0.00	0.00	0.00	11
2.0	0.46	0.60	0.52	10
3.0	0.77	0.71	0.74	14
accuracy			0.33	48
macro avg	0.31	0.33	0.32	48
weighted avg	0.32	0.33	0.32	48

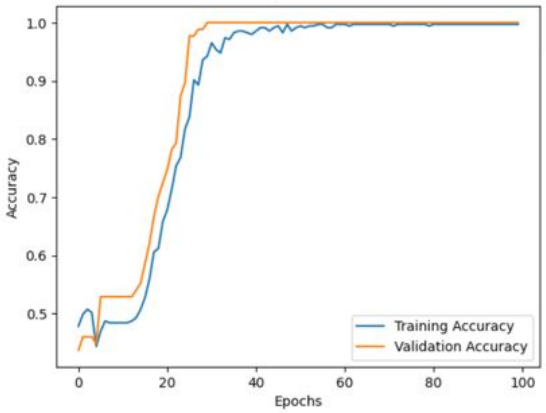


شکل ۱۹: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

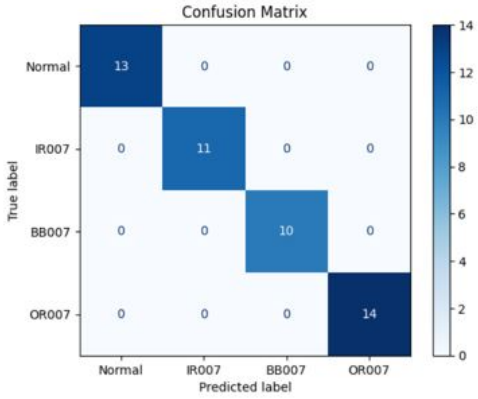
در ادامه برای اینکه بفهمیم مشکل از کجا بوده است، تابع اتلاف را به همان SparseCategoricalCrossentropy برمی‌گردانیم اما بهینه‌ساز را مجددا SGD انتخاب می‌کنیم. در این حالت نمودارها و معیارها به صورت شکل ۲۰ درآمده‌اند. نمودار اتلاف به خوبی

کاهش پیدا کرده و نمودار معیارهم تا مقدار ۱ افزایشی است. ماتریس درهم‌ریختگی و معیارهای دیگر نیز همگی نشان از آموزش مناسب می‌دهند.





	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	13
1.0	1.00	1.00	1.00	11
2.0	1.00	1.00	1.00	10
3.0	1.00	1.00	1.00	14
accuracy			1.00	48
macro avg	1.00	1.00	1.00	48
weighted avg	1.00	1.00	1.00	48



شکل ۲۰: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

نتیجه‌گیری نهایی برای این قسمت را می‌توان اینگونه مطرح کرد که تغییر هم بهینه‌ساز و هم تابع اتلاف می‌توانند اثر متفاوتی در آموزش و عملکرد مدل بگذارند اما انتخاب تابع اتلاف اشتباه می‌تواند حتی در یک مجموعه‌دادهٔ بسیار ساده مانند این مثال، بسیار باعث افت معیارهای عمبکردی شود. در آموزش سوم نیز مشاهده می‌شود که بهینه‌ساز adam سرعت همگرایی بیشتر نسبت به SGD داشته است و چون مجموعه دادهٔ انتخابی آسان بوده است، هر دو به مقدار ۱۰۰ درصد رسیده‌اند.

۴.۲ در مورد Cross-validation K-Fold Stratified و Cross-validation K-Fold و مزایای هر یک توضیح دهید. سپس با ذکر دلیل، یکی از این روش ها را انتخاب کرده و بخش «۲» سوال سوم را با آن پیاده سازی کنید و نتایج خود را تحلیل کنید.

روش K-Fold Cross-validation تکنیکی است که برای ارزیابی عملکرد و تعمیم‌پذیری یک مدل یادگیری ماشین استفاده می‌شود. مجموعه داده به K زیر مجموعه با اندازه مساوی تقسیم می‌شود. ابتدا دیتاست بصورت تصادفی به K بخش با اندازه مساوی تقسیم می‌شود برای هر یک از این K بخش، یک تقسیم‌بندی K-۱ برای داده‌های آموزش صورت می‌گیرد. به این صورت که آن بخش به عنوان دادهٔ اعتبارسنجی استفاده می‌شود و K-۱ بخش دیگر به عنوان دادهٔ آموزش. این فرآیند برای همهٔ K بخش اجرایی می‌شود. معیار نهایی

مانند accuracy بصورت میانگینی از K تکرار محاسبه می‌شود تا یک برآورد کلی از عملکرد مدل و دیتاست به دست آید. این روش اطمینان حاصل می‌کند که یک دیتا حتما یکبار هم در دیتای آموزش باشد و هم در دیتای اعتبارسنجی. که به طور کلی منجر به پیش‌بینی قابل اعتمادتری از عملکرد مدل در مقایسه با تقسیم‌بندی معمولی و یکباره می‌شود.

روش Stratified K-Fold Cross-validation نوعی از K-Fold Cross-validation است که تضمین می کند هر بخش نماینده توزیع کلی کلاس‌ها است. این ویژگی در دیتاست‌هایی که عدم تعادل کلاسی دارند و یک یا چند کلاس کمتر تکرار شده‌اند مناسب است. فرآیند مانند روش قبلی است با این تفاوت که هنگام تقسیم داده به K بخش، اطمینان حاصل می‌شود که توزیع کلاس در هر بخش منعکس کننده توزیع کلاس در کل مجموعه داده باشد. مزیت استفاده از این روش این است که نسبت کلاس‌ها را در هر قسمت حفظ می‌کند، که منجر به تخمین‌های عملکرد قابل اعتمادتر و سازگارتر، به‌ویژه برای مجموعه داده‌های نامتعادل می‌شود. بنابراین با توجه به اینکه تعادل کلاس‌ها در مجموعه‌داده به چه صورت است می‌توان از یک از این دو روش استفاده کرد.

در این قسمت ابتدا نیاز است با استفاده از کتابخانهٔ sklearn، دستور استفاده از k-fold را فراخوانی کنیم. سپس مدلی که در قسمت‌های قبلی مورد استفاده قرار داده بودیم را به شکل تابع درآوریم که در طول آموزش مدل طی دفعات مختلف بشود آن را فراخوانی کرد. مقدار k را هم با توجه به تعداد داده‌ها برابر ۵ قرار داده‌ایم.

```
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shuffle=True, random_state=4)
accuracies = []


def build_model():
    model_2_3 = Sequential()
    model_2_3.add(Dense(15, activation='relu', input_shape=(x_train.shape[1],)
    ))
    model_2_3.add(Dense(8, activation='relu'))
    model_2_3.add(Dense(4, activation='softmax'))
    model_2_3.compile(optimizer='adam', loss='SparseCategoricalCrossentropy',
    metrics=['accuracy'])
    return model_2_3
```

Code 26: import dataset and library

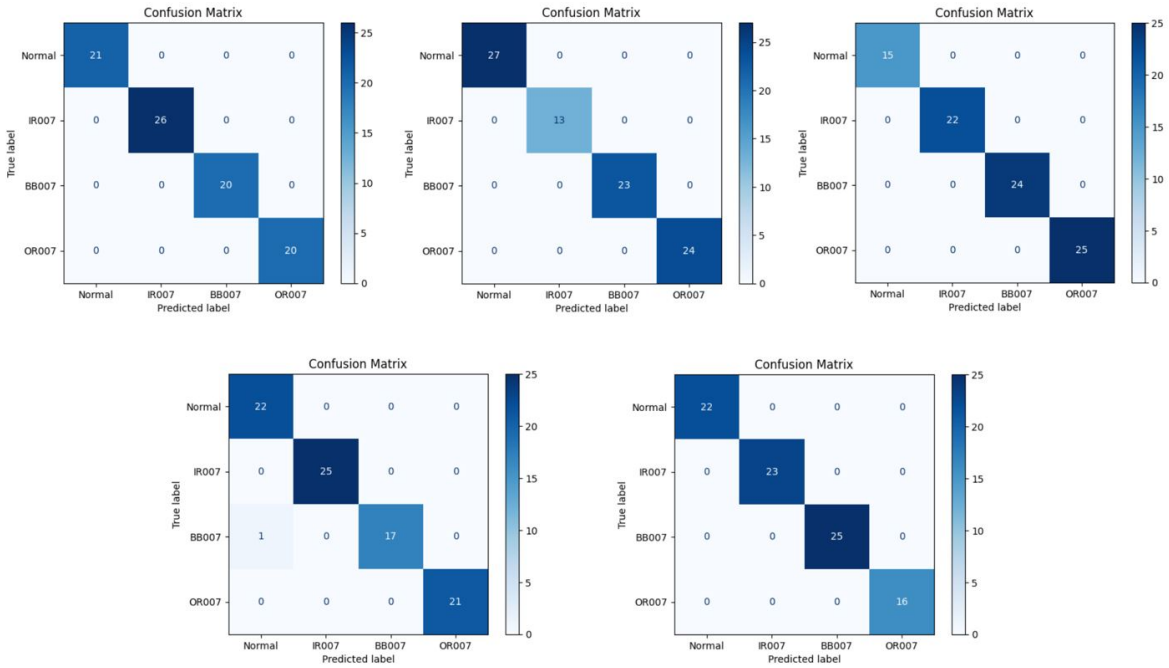
سپس با استفاده از مجموعه‌دادهٔ آموزش که قبلا از دادهٔ ارزیابی جدا شده بود، با استفاده از دستور kf.split و طی ۵ حلقه، آموزش مدل روی دیتاست را انجام می‌دهیم. باید توجه داشت که اینجا مجموعه‌دادهٔ آموزش اولیه به دو قسمت آموزش و اعتبارسنجی جدید تقسیم شده است و برای امکان ارزیابی دقیق‌تر با قسمت‌های قبلی قسمت دیتاست ارزیابی مشابه با مراحل قبلی تمرین است. کد مورد استفاده به صورت ۳۴ است.

```
for train_index, valid_index in kf.split(x_train):
```

```
    # Split the data
    x_train_k, x_valid_k = x_train[train_index], x_train[valid_index]
    y_train_k, y_valid_k = y_train[train_index], y_train[valid_index]

    # Initialize and train the model (RandomForestClassifier in this case)
    # model = RandomForestClassifier(n_estimators=100, random_state=42)
    model_2_3 = build_model()
    model_2_3.fit(x_train_k, y_train_k, validation_data=(x_valid_k, y_valid_k)
    , epochs=100, batch_size=10)


    # Predict on the test set
    y_pred_2_3 = model_2_3.predict(x_valid_k)
    y_pred_classes = np.argmax(y_pred_2_3, axis=1)


    cm = confusion_matrix(y_valid_k, y_pred_classes)

    target_names = ['Normal', 'IR007', 'BB007', 'OR007']

    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=
    target_names)
    disp.plot(cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.show()


    accuracy = accuracy_score(y_valid_k, y_pred_classes)

    accuracies.append(accuracy)

# Calculate the average accuracy
average_accuracy = np.mean(accuracies)
print(f'Average Accuracy: {average_accuracy:.2f}')
```

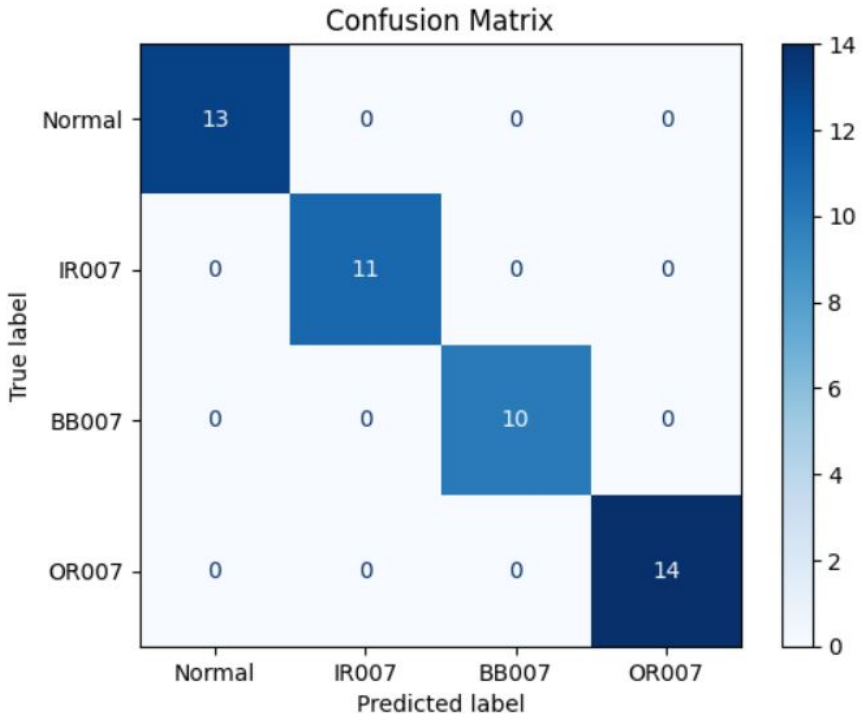
Code 27: import dataset and library

نتایج ماتریس‌های درهم‌ریختگی به صورت شکل ۲۱ قایل مشاهده است.



شکل ۲۱: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

درنهایت نتیجهٔ ارزیابی روی مجموعه‌دادهٔ ارزیابی نیز به صورت شکل ۲۲ است.



شکل ۲۲: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

۳ سوال سوم

ابتدا کتابخانه‌های مورد نیاز را فراخوانی می‌کنیم. سپس فایل CSV دانلودی از مرجع مورد ذکر سوال را با دستور gdown فراخوانی می‌کنیم و آن را بصورت یک دیتافریم درمی‌آوریم. دیتافریم تشکیل‌شده بصورت زیر است. همانطور که قابل مشاهده است دیتاست دارای ۵ ویژگی است که برخی به صورت عددی و برخی به صورت توصیفی هستند. همچنین تعداد کل داده‌ها برابر ۲۰۰ است.

```
!gdown 1XI-a6XbUk23bh0dNirGN8-drYX1ZUtFj
drug_data = pd.read_csv("/content/drug200.csv")

drug_data.head()
```

Code 28: import dataset and library

در صورت سوال ذکرشده است که بخشی از داده‌ها مورد استفاده قرار گیرد. بنابراین لازم است ابتدا اطلاعاتی در مورد داده‌ها به دست آوریم تا بتوانیم در صورت لزوم این کار را انجام دهیم. ابتدا مقدار داده‌های موجود در هر کلاس را مورد بررسی قرار می‌دهیم. کد و خروجی به صورت زیر است.

```
column_name = 'Drug'
num_unique_values = drug_data[column_name].nunique()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY

شکل ۲۳: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

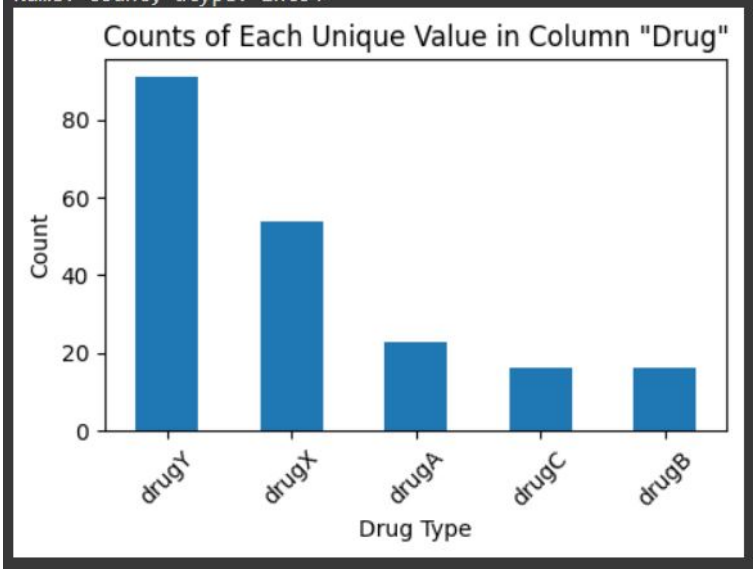
```
value_counts = drug_data[column_name].value_counts()
print(f'The number of unique values in the column "{column_name}" is {
    num_unique_values}')
print(value_counts)

# Plotting the value counts
plt.figure(figsize=(10, 6))
value_counts.plot(kind='bar')
plt.title(f'Counts of Each Unique Value in Column "{column_name}"')
plt.xlabel('Drug Type')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```

Code 29: import dataset and library

تعداد داده‌ها موجود در دیتاست متناسب با یکدیگر نیست. بیشترین مقدار داده در یک طبقه برابر ۹۱ و کمترین برابر ۱۶ است. ممکن است این عدم تناسب برای شبکه مشکل ایجاد کند. البته می‌توان ابتدا پیش از متناسب کردن تعداد داده‌ها با یکدیگر پس از آموزش، مدل را مورد ارزیابی قرار داد و در صورت لزوم این چنین پیش پردازشی روی مدل انجام دهیم.

پیش از تقسیم داده‌ها نیاز است که ویژگی‌های دیتاست را مطابق با مدل درخت تصمیم اماده کنیم. لازم است که نوع دیتای ویژگی‌های

```
The number of unique values in the column "Drug" is 5
Drug
drugY    91
drugX    54
drugA    23
drugC    16
drugB    16
Name: count, dtype: int64
```



شکل ۲۴: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

دیتاست از یک نوع باشد. بنابراین ویژگی‌هایی که از جنس توصیفی هستند را به صورتی مجازی تغییر می‌دهیم و به هر کلاس یک عدد نسبت می‌دهیم. ویژگی‌های Sex, BP, Cholesterol توصیفی هستند. بنابراین ابتدا کلاس‌های هر کدام از ویژگی‌ها را لازم است بدانیم. کد زیر نحوه انجام این فرآیند و خروجی مورد نظر را نشان می‌دهد.

```
unique_values_Sex = np.unique(drug_data["Sex"])
unique_values_BP = np.unique(drug_data["BP"])
unique_values_Cholesterol = np.unique(drug_data["Cholesterol"])

print("unique_values_Sex = " ,unique_values_Sex)
print("unique_values_BP = ", unique_values_BP)
print("unique_values_Cholesterol", unique_values_Cholesterol)


unique_values_Sex =  ['F' 'M']
unique_values_BP =  ['HIGH' 'LOW' 'NORMAL']
```

```
unique_values_Cholesterol ['HIGH' 'NORMAL']
```

Code 30: import dataset and library

سپس با توجه به کلاس های هر ویژگی توصیفی، به هر نوع کلاس یک عدد به خصوص نسبت می‌دهیم که مناسب دادن به مدل درخت تصمیم باشد. خروجی این تغییر در داده‌ها نیز به صورت شکل ۲۵ است.

```
Sex_mapping = {'F': 1, 'M': 2}
BP_mapping = {'HIGH': 1, 'LOW': 2, 'NORMAL': 3}
Cholesterol_mapping = {'HIGH': 1, 'NORMAL': 2}

drug_data['Sex'] = drug_data['Sex'].replace(Sex_mapping)
drug_data['BP'] = drug_data['BP'].replace(BP_mapping)
drug_data['Cholesterol'] = drug_data['Cholesterol'].replace(
    Cholesterol_mapping)

drug_data.head()
```

Code 31: import dataset and library

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	1	1	1	25.355	drugY
1	47	2	2	1	13.093	drugC
2	47	2	2	1	10.114	drugC
3	28	1	3	1	7.798	drugX
4	61	1	2	1	18.043	drugY

شکل ۲۵: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

در قدم بعدی باید از مجموعه‌دادهٔ مورد نظر مقادیر مربوط به ویژگی‌ها را در قالب X و برچسب‌ها در قالب y درآورده شود. سپس داده‌های مربوط به آموزش و ارزیابی از یکدیگر جدا شده‌اند. نسبت داده‌های آرزیابی ۱۵ درصد در نظر گرفته شده است.

```
X_drug = drug_data.drop(columns=['Drug']).values
y_drug = drug_data['Drug'].values
```

```

X_train_d, X_test_d, y_train_d, y_test_d = train_test_split(X_drug, y_drug,
    random_state=4, test_size=0.15)
X_train_d.shape, X_test_d.shape

((170, 5), (30, 5))
```

Code 32: import dataset and library

گام بعدی طراحی درخت تصمیمی است که کلاس‌های مورد نظر داده‌های ما را از یکدیگر جدا کند. این درخت تصمیم را با استفاده از دستور DecisionTreeClassifier فراخوانی کرده‌ایم. در این دستور لازم است برخی پارامترها تعریف شوند. یکی از پارامترها criterion است که تابعی است که میزان کیفیت تقسیم‌بندی را اندازه‌گیری می‌کند. این مقادیر برای این پارامتر قابل استفاده هستند: "gini", "entropy", "log-loss" همچنین پارامتر ccp-alpha نیز یک پارامتر مهم است. این پارامتر در مورد میزان پیچیدگی مورد استفاده برای هرس حداقلی Cost-Complexity است. به این صورت که در فرآیند آموزش، درختی فرعی با بیشترین میزان Cost-Complexity که از مقدار ccp-alpha کمتر باشد، انتخاب خواهد شد. مقادیر مورد استفاده در این مرحله در کد آمده است.

```
clf = tree.DecisionTreeClassifier(random_state=4, ccp_alpha=0.1, criterion='
    entropy')
clf.fit(X_train_d, y_train_d)
tree.plot_tree(clf)
```

Code 33: import dataset and library

```
DecisionTreeClassifier
DecisionTreeClassifier(ccp_alpha=0.5, criterion='entropy', random_state=4)
```

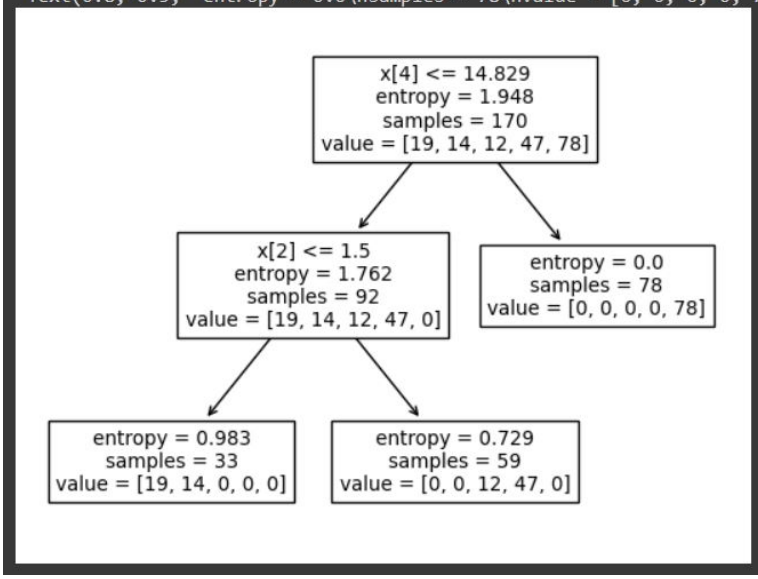
شکل ۲۶: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

درخت تصمیم طراحی شده با توجه به پارامترهای انتخابی به صورت شکل ۲۷ است. در این حالت ۲ مرحله تصمیم‌گیری در درخت تصمیم صورت گرفته است. در مرحله اول با آستانه گذاری برای ویژگی Na-to-K انجام شده است. به این صورت که هر داده‌ای که دارای این ویژگیِ بالاتر از آستانه باشد متعلق به کلاس پنجم است و اگر کمتر از این آستانه باشد متعلق به سایر کلاس‌هاست. در نقطه تصمیم‌گیری بعدی یک آستانه گذاری برای ویژگی BP گذاشته شده است. که طبق آن اگر مقدار این ویژگی کمتر از آستانه باشد آن داده متعلق به کلاس اول یا دوم و اگر بیشتر از آستانه باشد متعلق به کلاس سوم یا چهارم است.

برای تست روی داده‌های ارزیابی نیز به صورت ۴ داریم که برابر مقدار بالایی نیست. برای حل این مشکل و اینکه مدل بتواند بصورت دقیق‌تری کلاس‌ها را جداسازی کند و مطابق خواسته سوال، پارامترهای طراحی درخت را تغییر می‌دهیم.

```
clf.score(X_test_d, y_test_d)
```

```
[Text(0.6, 0.8333333333333334, 'x[4] <= 14.829\nentropy = 1.948\nsamples = 170\nvalue = [19, 14, 12, 47, 78]'),
 Text(0.4, 0.5, 'x[2] <= 1.5\nentropy = 1.762\nsamples = 92\nvalue = [19, 14, 12, 47, 0]'),
 Text(0.2, 0.16666666666666666, 'entropy = 0.983\nsamples = 33\nvalue = [19, 14, 0, 0, 0]'),
 Text(0.6, 0.16666666666666666, 'entropy = 0.729\nsamples = 59\nvalue = [0, 0, 12, 47, 0]'),
 Text(0.8, 0.5, 'entropy = 0.0\nsamples = 78\nvalue = [0, 0, 0, 0, 78]')]
```



شکل ۲۷: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

```
2

3 0.8
```

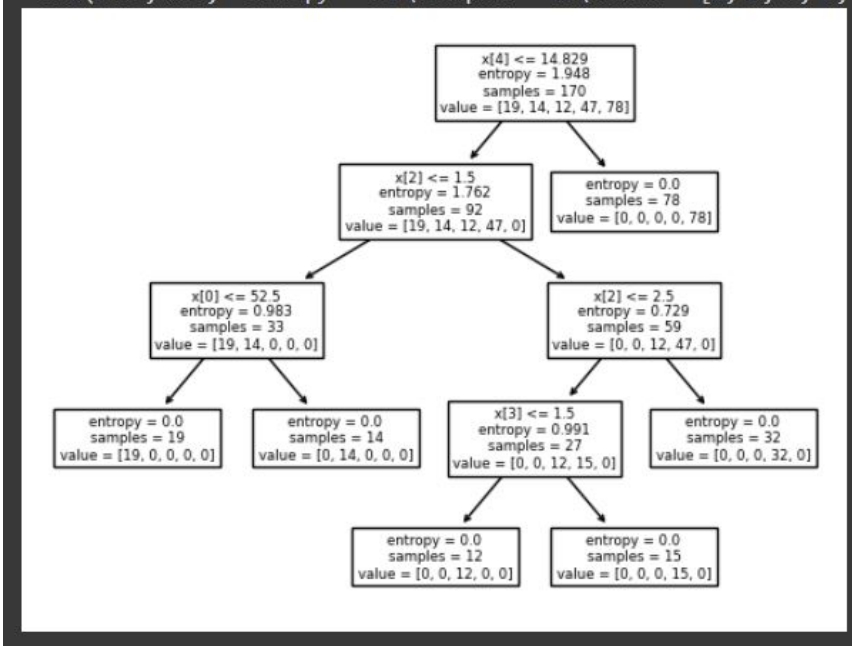
Code 34: import dataset and library

تغییر پارامتر را بصورت انتخاب مقدار ccp-alpha برابر ۱.۰ انخاب می‌دهیم. درخت طراحی شده پس از انتخاب این پارامتر بصورت شکل ۲۸ درمی‌آید. با این طراحی با تعداد ۵ نقطه تصمیم‌گیری و روی چهار ویژگی مختلف، مدل توانسته است مطابق تصویر در تمامی لیف‌نود ها، تنها یک کلاس حضور داشته باشد که نشان‌دهندهٔ تفکیک شدن همهٔ کلاس‌ها از یکدیگر است.

۱.۳ با استفاده از ماتریس درهم ریختگی و حداقل سه شاخصهٔ ارزیابی مربوط به وظیفهٔ طبقه بندی، عمل کرد درخت آموزش داده شدهٔ خود را روی بخش آزمون داده ها ارزیابی کنید و نتایج را به صورت دقیق گزارش کنید. تأثیر مقادیر کوچک و بزرگ حداقل دو فراپارامتر را بررسی کنید. تغییر فراپارامترهای مربوط به هرس کردن چه تأثیری روی نتایج دارد و مزیت آن چیست؟

ماتریس در هم ریختگی و سه معیار recall, precision, accuracy برای ارزیابی عملکرد مدل روی بخش داده‌های تست مورد استفاده قرار گرفته‌اند. همانطور که از ماتریس درهم‌ریختگی در شکل ۲۹ مشخص است تنها یک داده به اشتباه تشخیص داده شده است. این داده متعلق به کلاس B بوده و به اشتباه به کلاس A تعلق پیدا کرده است. این تشخیص اشتباه در معیارهای ذکر شده به صورت درصد خود را نشان داده‌اند.

```
[Text(0.625, 0.9, 'x[4] <= 14.829\nentropy = 1.948\nsamples = 170\nvalue = [19, 14, 12, 47, 78]'),
 Text(0.5, 0.7, 'x[2] <= 1.5\nentropy = 1.762\nsamples = 92\nvalue = [19, 14, 12, 47, 0]'),
 Text(0.25, 0.5, 'x[0] <= 52.5\nentropy = 0.983\nsamples = 33\nvalue = [19, 14, 0, 0, 0]'),
 Text(0.125, 0.3, 'entropy = 0.0\nsamples = 19\nvalue = [19, 0, 0, 0, 0]'),
 Text(0.375, 0.3, 'entropy = 0.0\nsamples = 14\nvalue = [0, 14, 0, 0, 0]'),
 Text(0.75, 0.5, 'x[2] <= 2.5\nentropy = 0.729\nsamples = 59\nvalue = [0, 0, 12, 47, 0]'),
 Text(0.625, 0.3, 'x[3] <= 1.5\nentropy = 0.991\nsamples = 27\nvalue = [0, 0, 12, 15, 0]'),
 Text(0.5, 0.1, 'entropy = 0.0\nsamples = 12\nvalue = [0, 0, 12, 0, 0]'),
 Text(0.75, 0.1, 'entropy = 0.0\nsamples = 15\nvalue = [0, 0, 0, 15, 0]'),
 Text(0.875, 0.3, 'entropy = 0.0\nsamples = 32\nvalue = [0, 0, 0, 32, 0]'),
 Text(0.75, 0.7, 'entropy = 0.0\nsamples = 78\nvalue = [0, 0, 0, 0, 78]')]
```

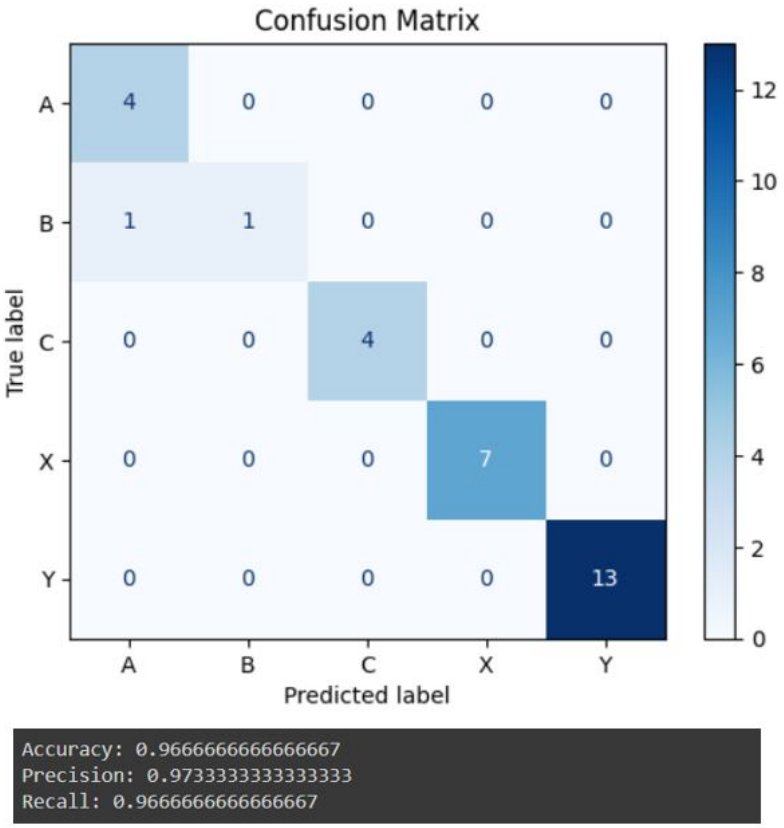


شکل ۲۸: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

y_pred_classes = np.argmax(y_pred_2_2, axis=1)  # Convert predicted probabilities to class labels
cm = confusion_matrix(y_test_d, y_pred)
target_names = ['A', 'B', 'C', 'X', 'Y']

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=target_names)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
```

```
plt.show()


from sklearn.metrics import accuracy_score, precision_score, recall_score

accuracy = accuracy_score(y_test_d, y_pred)
precision = precision_score(y_test_d, y_pred, average='weighted')
recall = recall_score(y_test_d, y_pred, average='weighted')
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

Code 35: import dataset and library



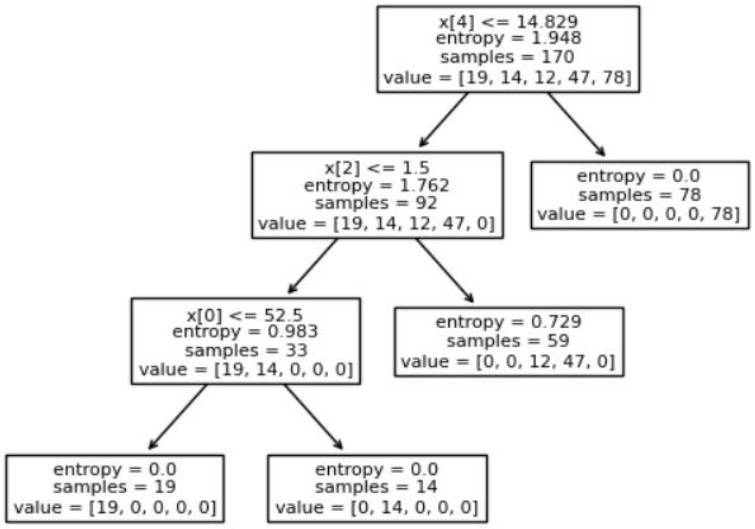
شکل ۲۹: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

اثر تغییر یکی از پارامترها که ccp-alpha بود مورد بررسی قرار گرفت. این پارامتر پیچیدگی درخت تصمیم را از طریق هرس حداقل Cost-Complexity کنترل می کند. مقدار کوچکتر ccp-alpha به این معنی است که جریمه اضافه کردن گره به درخت کمتر است.

بنابراین، الگوریتم به احتمال زیاد گره‌ها را اضافه می‌کند و درخت بزرگ‌تری را رشد می‌دهد، که به طور بالقوه منجر به مدل پیچیده‌تری می‌شود که می‌تواند بهتر با داده‌های آموزشی تناسب داشته باشد. با این حال، این افزایش پیچیدگی همچنین ممکن است منجر به تطبیق بیش از حد داده‌های آموزشی و کاهش عملکرد در داده‌های دیده نشده شود.

به عنوان پارامتر دوم max-depth را انتخاب می‌کنیم. این پارامتر تعیین‌کنندهٔ عمق درخت تولید شده است. هر چقدر مقدار این پارامتر بیشتر باشد، درخت می‌تواند تعداد نود های تصمیم گیری بیشتری داشته باشد. معمولا از این پارامتر برای جلوگیری از اورفیت زدن استفاده می‌شود که درخت خیلی روی دیتاهای ارزیابی تطبیق بیش از اندازه نیابد. در این سوال چون ما با عمق ۴ توانستیم به عملکرد بسیار مطلوبی از مدل برسیم، این پارامتر را برابر ۳ قرار میدهیم. البته این کار صرفا برای مشاهدهٔ نتیجه این تغییر پارامتر صورت می‌گیرد و ما در قسمت قبل به نتیجه دلخواه مدل خود رسیده‌ایم.

با اعمال max-depth برابر ۳ این درخت تصمیم را داریم:



شکل ۳۰: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

مشاهده می‌شود که با اعمال این پارامتر تعداد نود های تصمیم‌گیرنده کاهش یافته و لیف‌نودهایی با کلاس‌های مشترک دیده می‌شود.

۲.۳ توضیح دهید که روش هایی مانند جنگل تصادفی و AdaBoost چگونه می توانند به بهبود نتایج کمک کنند. سپس، با انتخاب یکی از این روش ها و استفاده از فراپارامترهای مناسب، سعی کنید نتایج پیاده سازی در مراحل قبلی را ارتقاء دهید.

هر دو روش از تکنیک‌های یادگیری جمعی هستند که در موضوع درخت تصمیم برای رسیدن به نتایج بهبودیافته‌تر مورد استفاده قرار می‌گیرند.

Random Forest : در این روش چندین درخت تصمیم در طول آموزش ساخته می‌شود. هر درخت به صورت مستقل و در حالی که فرآیند انتخاب ویژگی به صورت تصادفی است، ساخته می‌شود. هر درخت به صورت مستقل یک پیش‌بینی انجام می‌دهد. پیش‌بینی نهایی در تسک طبقه‌بندی به صورت رای‌گیری بیشینه صورت می‌پذیرد. این تکنیک به دلیل استحکام در برابر اورفیت‌شدن و توانایی در

برابر مدیریت داده‌های بزرگ شناخته می‌شود.

AdaBoost : این تکنیک با ترکیب چند درخت تصمیم ضعیف برای ایجاد یک درخت تصمیم قوی کار می‌کند. این روش با تنظیم وزن‌داده‌های آموزشی بر اساس نحوه عملکرد درخت تصمیم پیشین کار می‌کند. در درخت ابتدایی اولیه به هر داده وزن یکسانی داده می‌شود. در تکرارهای بعدی تمرکز و وزن بیشتری برای داده‌های با طبقه‌بندی اشتباه گذاشته می‌شود و به مدل اجازه می‌دهد از اشتباهات قبلی خود درس بگیرد. پیش‌بینی نهایی با ترکیب طبقه‌بندهای قبلی با اختصاص دادن وزن مخصوص به هر یک شکل می‌گیرد.

از تکینک Random Forest استفاده می‌کنیم. با استفاده از روش GridSearch پارامترهای مختلف را در این روش مورد ارزیابی قرار می‌دهیم.

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'bootstrap': [True, False]
    'max_depth': [6, 10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

rf_classifier = RandomForestClassifier(random_state=4)
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv
    =5, scoring='accuracy')
grid_search.fit(X_train_d, y_train_d)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Use the best model to predict on test data
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_d)

# Evaluate the classifier
accuracy = accuracy_score(y_test_d, y_pred)
print("Accuracy:", accuracy)
```

Code 36: import dataset and library

نتایج نهایی به صورت شکل ۳۱ قابل مشاهده است.

```
Best Hyperparameters: {'bootstrap': True, 'max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Accuracy: 0.9333333333333333
```

شکل ۳۱: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

۴ عنوان سوال چهارم

ابتدا کتابخانه‌های مورد نیاز را فراخوانی می‌کنیم.

```python
from pandas import DataFrame as df
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Code 37: import libraries (Python)

سپس دیتاست بیماری قلب با استفاده از دستور gdown قابلیت دسترسی پیدا می‌کند. سپس با کتابخانه pandas این دیتاست خوانده می‌شود. با دستور head() عنوان ستون‌های دیتاست قابل مشاهده خواهند بود.

```python
!gdown 1qYg0L0-iDvhawJw-v5keUtQc9FpYALxB

data = pd.read_csv("/content/heart.csv")
data.head()
```

Code 38: import libraries (Python)

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

شکل ۳۲: بلوک دیاگرام مدل طبقه‌بند خطی

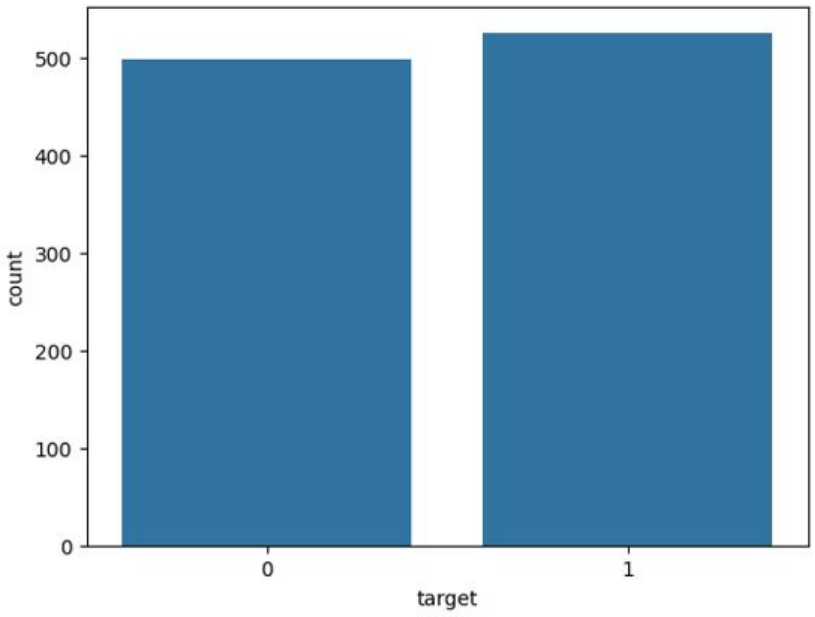
با استفاده از این دستورات می‌توان به میزان داده‌های موجود در هر کلاس پی برد.

```
data.groupby('target').size()
sns.countplot(x='target', data=data)
```

Code 39: import libraries (Python)

خروجی این کد بصورت زیر است. همانطور که قابل مشاهده است میزان داده‌ها در دو کلاس نامتوازن است. بنابراین می‌توان در نظر داشت که ممکن است کار متوازن کردن داده‌ها در نتایج تاثیر داشته باشد.

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



شکل ۳۳: بلوک دیاگرام مدل طبقه‌بند خطی

با استفاده از این دستورها می‌توان اطلاعات ارزشمندی از دیتاست به دست آورد.

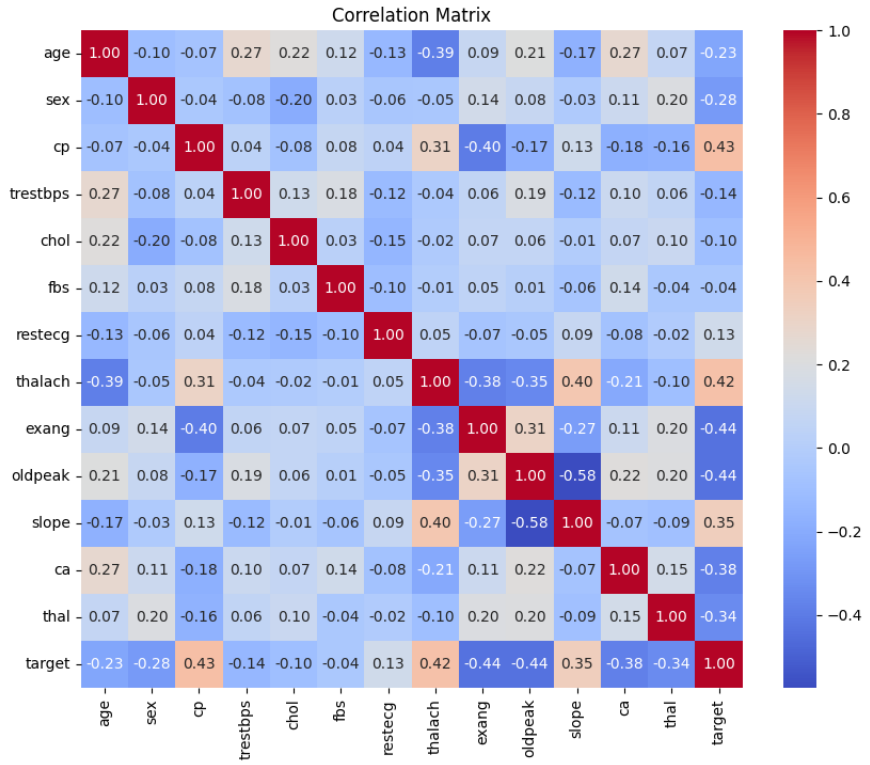
```
sns.pairplot(data,hue='target')

data.hist()

correlation_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
```

```
plt.show()
```

Code 40: import libraries (Python)

خط اول شبکه‌ای از نمودارهای پراکنده ایجاد می‌کند که در آن هر نمودار نشان‌دهنده رابطه بین جفت‌های متغیر در مجموعه داده است و نقاط بر اساس مقادیر در ستون 'هدف' رنگ می‌شوند. این تجسم به ویژه برای بررسی روابط بین چندین متغیر و نحوه ارتباط آنها با متغیر هدف مفید است. خط دوم یک هیستوگرام جداگانه برای هر ستون عددی ایجاد می‌کند واین امکان را می‌دهد که توزیع مقادیر را برای هر متغیر به صورت بصری بررسی کرد. قسمت سوم این کد یک نمایش بصری از همبستگی‌های بین متغیرهای مختلف در مجموعه داده ایجاد می‌کند و به شناسایی الگوها و روابط بین متغیرها کمک می‌کند.

ماتریس همبستگی به صورت زیر قابل مشاهده است.



شکل ۳۴: بلوک دیاگرام مدل طبقه‌بند خطی

همانطور که مشاهده می‌شود ارتباط بین ویژگی thalach و slope نسبت به سایر ویژگی‌ها بیشتر است و می‌توان از این ویژگی در ادامه برای کارهای پیش‌پردازشی بهره برد.

یکی از کارهایی که به طور معمول در پیش‌پردازش استفاده می‌شود، شافل‌کردن داده‌هاست. ممکن است ترتیب چیده شدن دیتاها پشت یکدیگر از قبل مناسب نباشد و مدل را دچار اشتباه کند. شافل کردن باعث می‌شود مدل به دیتاهای مربوط به یک کلاس واحد وابسته نشود.

```
from sklearn.utils import shuffle
```

```
shuffled_data = shuffle(data, random_state=4)
print("shuffled_data shape = ",shuffled_data.shape)

shuffled_data shape =  (1025, 14)
```

Code 41: import libraries (Python)

سپس نیاز است که مجموعه‌داده را به دو بخش ویژگی و تارگت تقسیم کنیم. تارگت در ستونی با عنوان target قابل دسترسی است.

```
X=np.array(shuffled_data.loc[:,data.columns!='target'])
y=np.array(shuffled_data.loc[:,data.columns=='target'])
print(X.shape, y.shape)

(1025, 13) (1025, 1)
```

Code 42: import libraries (Python)

با توجه به نامتوازن بودن تعداد داده‌های متعلق به هر کلاس، می‌توان این دو کلاس را با روش Random Undersampling متوازن کرد. به این صورت که از کلاس با داده‌های بیشتر بصورت رندوم حذف داده صورت بگیرد تا تعداد داده‌های هر کلاس برابر شود. البته با توجه به اینکه این میزان تفاوت داده‌های هر کلاس چشم‌گیر نیست، احتمالا اثر چندانی در نتیجه نخواهد داست. البته کد طوری در نظر گرفته شده است که بتوان هم بدون توازن و هم پس از توازن داده کار طبقه‌بندی را انجام داد.

```
from imblearn.under_sampling import RandomUnderSampler

# rus = RandomUnderSampler(random_state=4)
rus = RandomUnderSampler(sampling_strategy={0: 499, 1: 499}, random_state=4)

# Resample the dataset
UnderSample = True
if UnderSample :
  X_resampled, y_resampled = rus.fit_resample(X, y)
  print("Class Distribution after Random Undersampling:")
  print(pd.Series(y_resampled).value_counts())
else :
  X_resampled = X
  y_resampled = y
```

Code 43: import libraries (Python)

خروجی پس از توازن داده‌ها بصورت دوش Random Undersampling بصورت زیر است.

```
Class Distribution after Random Undersampling:
0    499
1    499
Name: count, dtype: int64
```

شکل ۳۵: بلوک دیاگرام مدل طبقه‌بند خطی

تقسیم داده‌ها به دو بخش آموزش و آزمون بصورت زیر قابل انجام است. با توجه به تعداد حدود ۵۰۰ داده، مناسب است که مقدار ۲۰ درصد از داده‌ها را به مجموعه‌داده تست اختصاص دهیم.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
    test_size=0.2, random_state=4)
print('train:', X_train.shape, y_train.shape, '\ntest: ', X_test.shape, y_test
    .shape)

train: (820, 13) (820, 1)
test:  (205, 13) (205, 1)
```

Code 44: import libraries (Python)

با توجه به اطلاعات مربوط به ویژگی‌های داده در شکل ۳۲ و با استفاده از دستور data.describe() می‌توان مشاهده کرد که بازه مقادیر ویژگی‌های مختلف تفاوت زیادی دارد و این موضوع می‌تواند چالش آفرین باشد و مناسب‌تر است که بازه ویژگی‌های مجموعه‌داده را با هم متناسب کرد. از روش MinMaxScaler بازه ویژگی‌های دیتاست را بین ۰ و ۱ می‌آوریم. البته باید توجه داشت که این کار صرفا با اسفاده از داده‌های آموزش باشد و از اطلاعات داده‌های تست استفاده نشود.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
# print(X_train[:5])
X_train_t = scaler.transform(X_train)
X_test_t = scaler.transform(X_test)
# print(X_train_t[:5])
```

Code 45: import libraries (Python)

در مرحلهٔ بعد به دو صورت کار طبقه‌بندی صورت می‌گیرد. با دستور آماده و استفاده از کتابخانه sklearn و همچنین به صورت کدزنی کلاس این طبقه‌بند. و فراخوانی و آموزش روی داده‌های آموزش آورده شده است.

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

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

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


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

    def _predict(self, x):
        posteriors = []

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

        # return class with the highest posterior
```

```
        return self._classes[np.argmax(posteriors)]

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


from sklearn.naive_bayes import GaussianNB
MyNB = NaiveBayes()
SKNB = GaussianNB()

MyNB.fit(X_train, y_train.ravel())
SKNB.fit(X_train, y_train.ravel())
pred = MyNB.predict(X_test)
pred2 = SKNB.predict(X_test)
```

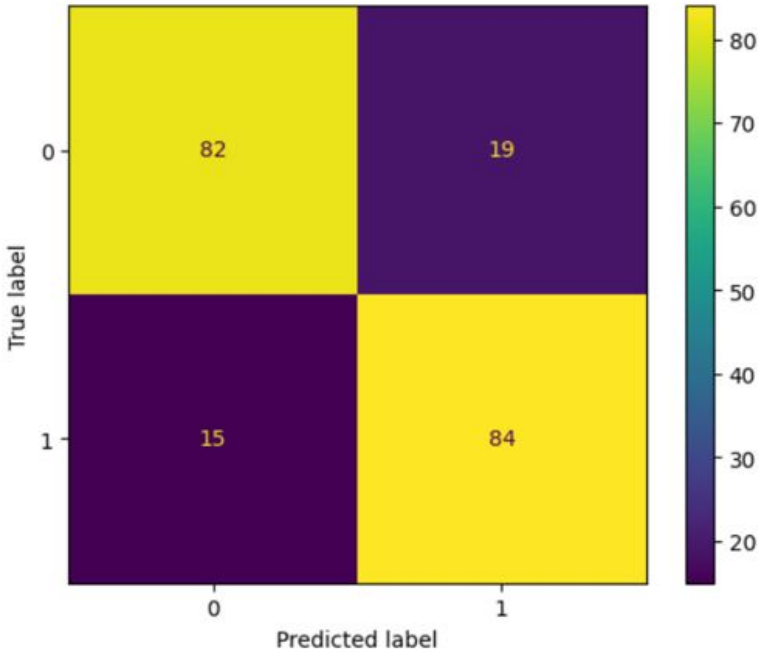
Code 46: import libraries (Python)

مطابق این کد پس از فراخوانی Bayes به دو روش گفته شده، با دستور fit این مدل را بر روی مجموعه‌دادهٔ آموزش ترین می‌کنیم. و پس از ان روی مجموعه تست کار ارزیابی را انجام می‌دهیم.

سپس طبق خواسته مسئله ماتریس درهم‌ریختگی و سایر متریک‌ها را مور ارزیابی قرار می‌دهیم. مشاهده می‌شود که تعداد ۲۰۵ دادهٔ تست ارزیابی شده‌اند و مطابق معیارها مدل در تشخیص کلاس ۰ کمی عملکرد بهتری داشته است. در ماتریس درهم‌ریختگی نیز به ترتیب تعداد ۱۵ و ۱۹ دادهٔ متعلق به کلاس ۱ و ۰ به اشتباه به کلاس دیگری نسبت داده شده است و سایر داده‌ها به درستی طبقه‌بندی شده‌اند.

```
from sklearn.metrics import classification_report
print(classification_report(y_test,pred))

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test,pred)
names = list(data.groupby('target').groups.keys())
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=names)
disp.plot()
plt.show()
```

Code 47: import libraries (Python)



	precision	recall	f1-score	support
0	0.86	0.77	0.81	99
1	0.80	0.89	0.84	106
accuracy			0.83	205
macro avg	0.83	0.83	0.83	205
weighted avg	0.83	0.83	0.83	205

شکل ۳۶: بلوک دیاگرام مدل طبقه‌بند خطی

دو حالت Micro و Macro به روش‌های مختلف میانگین‌گیری معیارهای عملکرد برای مسائل طبقه‌بندی چند طبقه اشاره دارند. این روش‌های میانگین‌گیری برای جمع‌آوری نمرات دقت، فراخوانی و F۱ در چندین کلاس استفاده می‌شود. حالت Micro معیارهارو بصورت گلوبال محاسبه می‌کند. این کار با شمارش تعداد ،TP ،FN FP بر روی همهٔ کلاس‌ها انجام می‌پذیرد. در این حالت با همه نمونه ها به طور مساوی رفتار می‌شود، بنابراین به هر نمونه، صرف نظر از کلاس آن، وزن یکسانی داده می‌شود. در حالت حالت Macro میانگین کلان معیارها را برای هر کلاس به طور مستقل محاسبه می‌شود و سپس میانگین این معیارها گرفته می‌شود. این حالت با همه کلاس‌ها به یک اندازه رفتار می‌کند، بنابراین به هر کلاس، صرف نظر از تعداد نمونه‌ها در هر کلاس، وزن یکسانی می‌دهد.

حالت Micro زمانی مفید است که عملکرد کلی طبقه‌بندی کننده در همه موارد مورد نیاز باشد که ارزیابی شود . این به شدت تحت تأثیر عملکرد در کلاس های مختلف است. حالت Macro زمانی مفید است که عملکرد طبقه بندی کننده در هر کلاس به طور مستقل ارزیابی شود و با هر کلاس به طور مساوی رفتار شود. وقتی کلاس‌ها نامتعادل هستند، می‌تواند مناسب‌تر باشد.

در انتها پنج داده به صورت تصادفی از مجموعه‌دادهٔ آزمون انتخاب کرده و خروجی پیش‌بینی‌شده را با خروجی واقعی مقایسه می‌کنیم. در این قسمت با دستور randint تعداد ۵ داده تولید می‌کنیم که بین ۰ تا طول بردار تعداد داده‌های تست باشد. از random-seed استفاده

کرده‌ایم که هر بار که اجرا می‌کنیم، همان داده‌های تولیدی اولیه را بدهد و هر بار مجددا رندوم تولید نکند. خروجی نیز به صورت زیر قابل مشاهده است که تعداد ۴ تا از پیش‌بینی‌ها درست و تنها یکی از آنها متفاوت با لیبل خود دیتاست.

```
np.random.seed(4)
random_data = np.random.randint(0, len(X_test), 5)
# print(random_data)
five_data = X_test[random_data]
# five_data.shape

pred_t_five_data = MyNB.predict(five_data)
true_lable = y_test[random_data].T
print('pred:         ',pred_t_five_data)
print('true lable: ', true_lable)
```

Code 48: import libraries (Python)

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

شکل ۳۷: بلوک دیاگرام مدل طبقه‌بند خطی

منابع

[۱] صفحهٔ درس یادگیری ماشین.

[2] Steven X. Ding, “Data-driven Design of Fault Diagnosis and Fault-tolerant Control System”, Springer, 2014.

[3] S. Theodoridis and K. Koutroumbas, ”Pattern recognition”, Fourth Edition, Academic Press, 2009.