به نام خدا

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

یادگیری ماشین

گزارش تمرین شماره ۲

شیما سادات ناصری

۴۰۱۱۲۸۱۴

دکتر مهدی علیاری شوره دلی

اردیبهشت ۱۴۰۳

فهرست مطالب

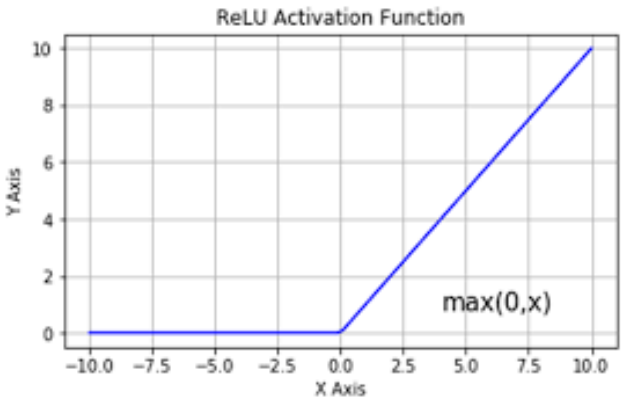
سوال ۱

۱

در یک طبقه بندی دو کلاسه، اگر دو لایه پایین شبکه به ترتیب از فعال کننده های ReLU و Sigmoid استفاده کنند، می‌تواند نحوه رفتار این توابع فعال ساز مشکل ساز شود.

- فعال سازی ReLU (واحد خطی اصلاح شده):

تابع فعال سازی ReLU به صورت f(x)=max(0,x) تعریف شده است. تصویر این تابع به صورت زیر است.



شکل ۱- نمودار **ReLU**

این تابع برای هر ورودی منفی صفر و اگر مثبت باشد خود ورودی را خروجی می دهد.

ReLU معمولاً در لایه‌های پنهان استفاده می‌شود، زیرا به کاهش مشکل گرادیان ناپدید شدن کمک می‌کند، امکان آموزش سریع‌تر را فراهم می‌کند و به شبکه کمک می‌کند الگوهای پیچیده را یاد بگیرد.

- فعال سازی Sigmoid:

تابع فعال سازی Sigmoid به صورت $f(x) = \frac{1}{1+e^{-x}}$ تعریف شده است. ورودی را به مقداری بین ۰ و ۱ تبدیل می‌کند.

تابع Sigmoid اغلب در لایه خروجی برای مسائل طبقه بندی باینری استفاده می شود زیرا بیانی احتمالی از خروجی ها را می دهد.

اگر لایه دوم از آخر تابع فعال سازی ReLU را داشته باشد، خروجی های این لایه غیر منفی (۰ یا مقادیر مثبت) خواهد بود. این خروجی ها سپس به آخرین لایه، که از یک تابع فعال سازی Sigmoid استفاده می‌کند، تغذیه می شود. تابع Sigmoid این مقادیر غیر منفی را می‌گیرد و آنها را در محدوده ای بین ۰ و ۱ محاسبه می‌کند.

مشکلات احتمالی این روش به صورت زیر رخ می‌دهد:

- جریان گرادیان و ناپایداری آموزش:

اگر خروجی فعال سازی ReLU همیشه مثبت یا صفر باشد، تابع Sigmoid فقط ورودی های غیر منفی را دریافت می کند. برای ورودی های مثبت بزرگ، تابع Sigmoid اشباع می شود (مقادیر خروجی نزدیک به ۱). این می تواند منجر به شیب های بسیار کوچک شود و باعث کند شدن قابل توجهی در روند آموزش شود.

در طول انتشار پس زمینه، گرادیان ها می توانند ناپدید شوند(بی تاثیر شوند)، به خصوص اگر ورودی تابع Sigmoid بسیار بزرگ باشد، زیرا مشتق تابع Sigmoid برای ورودی های بزرگ به صفر نزدیک می شود.

- توزیع خروجی:

خروجی های تابع Sigmoid به سمت مقادیر بالاتر بایاس می شوند. اگر بیشتر ورودی های ReLU مثبت باشند، می‌تواند باعث شود که شبکه به سمت پیش بینی یک کلاس بر دیگری سوگیری کند.

از طرفی در طول آموزش اولیه، اگر بیشتر خروجی‌های ReLU مثبت باشند، ممکن است شبکه برای تنظیم وزن‌ها برای نگاشت صحیح ورودی‌ها به احتمالات کلاس مورد نظر مشکل داشته باشد. این می تواند منجر به همگرایی کُند یا گیر افتادن در راه حل های غیربهینه شود.

راه‌حل‌های ممکن برای این مشکلات به صورت زیر خواهند بود:

- نرمال‌سازی دسته‌ای (Batch Normalization):

اعمال نرمال‌سازی دسته‌ای بین لایه‌های ReLU و Sigmoid می‌تواند با نرمال‌سازی ورودی‌های تابع Sigmoid کمک کند، به طوری که همه آن‌ها به مقادیر مثبت متمایل نشوند. این می‌تواند جریان گرادیان‌ها را بهبود بخشد و فرایند آموزش را پایدارتر کند.

- استفاده از تابع فعال‌سازی متفاوت:

اگر از یک تابع فعال‌سازی متفاوت مانند Tanh که مقادیر بین ۱- و ۱ را برمی‌گرداند، برای لایه قبل از آخر استفاده شود، می‌تواند کمک کند تا ورودی‌های تابع Sigmoid متعادل‌تر باشند.

- تغییر لایه خروجی:

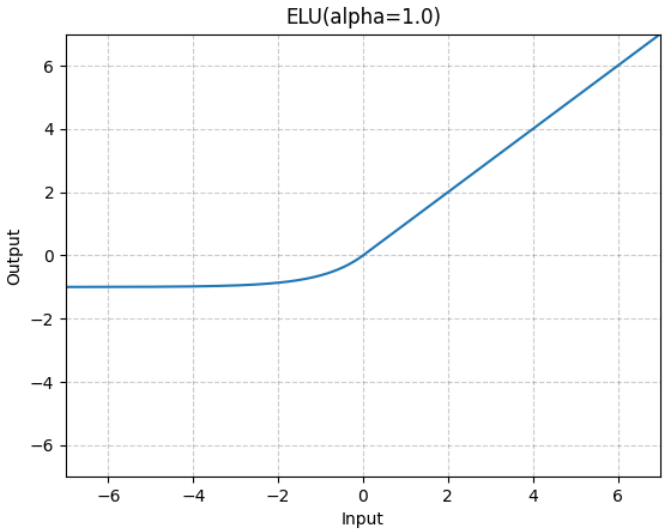
برای یک مسئله طبقه‌بندی دوکلاسه، استفاده از تابع فعال‌سازی Softmax در لایه آخر با دو نرون خروجی به جای یک نرون با فعال‌سازی Sigmoid می‌تواند مؤثرتر باشد. تابع Softmax می‌تواند ورودی‌های نامتوازن را بهتر مدیریت کند و احتمالات کلاس واضح‌تری ارائه دهد.

۲

معادله داده شده در تصویر تابع فعال سازی واحد خطی نمایی (ELU) است که به صورت زیر تعریف می شود:

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

تصویر این نمودار نیز به صورت زیر است:



شکل ۲- نمودار **ELU**

جایی که α یک هایپرپارامتر برای تعیین مقداری که یک ELU برای ورودی های خالص منفی اشباع می‌شود، می‌باشد.

گرادیان ELU

گرادیان این معادله به صورت زیر محاسبه می‌شود:

به ازای $x \geq 0$:

$$\frac{d}{dx} ELU(x) = 1$$

و به ازای سایر مقادیرx:

$$\frac{d}{dx} ELU(x) = \alpha e^x$$

بر این اساس می‌توان گفت که این معادله مزایای زیر را دارد:

- اجتناب از مشکل ReLU های مرده یا (Dying ReLU):

این مورد یکی از معایب اصلی ReLU است که در آن نورون ها می توانند غیرفعال شوند و همیشه برای هر ورودی صفر خروجی داشته باشند. این حالت زمانی اتفاق می‌افتد که ورودی ReLU همیشه منفی است و گرادیان را صفر می‌کند، که می‌تواند از به‌روزرسانی نورون در حین backpropagation جلوگیری کند.

ELU با در نظر گرفتن یک گرادیان کوچک و غیر صفر برای ورودی‌های منفی (αe^x) این مشکل را برطرف می‌کند که می‌تواند به شبکه کمک کند تا حتی اگر ورودی‌ها منفی باشند، به یادگیری و به‌روزرسانی وزن‌ها ادامه دهد.

- گرادیان نرم‌تر برای ورودی های منفی:

برای ورودی های منفی، ELU یک مقدار منفی خواهد داشت که با منفی شدن x به آرامی مجانبی به α- (در شکل ۲، بعد از مدتی مجانب به ۱- شده که در آن α برابر ۱ است) می‌شود. این نرمی و گرادیان غیرصفر به یادگیری بهتر و حفظ جریان گرادیان قوی‌تر در طول backpropagation کمک می‌کند.

ماهیت پیوسته و قابل تمایز ELU برای ورودی های منفی، یادگیری پایدارتر و کارآمدتری را به خصوص در شبکه های عمیق فراهم می کند.

- همگرایی بهبود یافته:

درکنار ویژگی هایی که در بالا گفته شد، ELU ها اغلب منجر به همگرایی سریع‌تر و قابل اعتمادتر در طول آموزش در مقایسه با ReLU می‌شوند. گرادیان های غیرصفر برای مقادیر منفی به کاهش مشکلات مرتبط با گرادیان های ناپدید شده کمک می‌کند، بنابراین به روز رسانی بهتر وزن را تسهیل می کند.

۳

برای حل این مسئله ابتدا رئوس مثلث را به صورت زیر در نظر گرفته می‌شود:

$$A(2,2), \qquad B(3,0), \qquad C(1,0)$$

بر این اساس، معادلات خطوط تشکیل دهنده اضلاع مثلث به صورت زیر خواهند بود:

$$(y-2) = \frac{2-0}{2-3}(x-2) \Rightarrow 2x + y = 6$$
$$(y-0) = \frac{0-0}{3-0}(x-3) \Rightarrow y = 0$$
$$(y-0) = \frac{0-2}{0-1}(x-1) \Rightarrow y = -2x - 2$$

هر ضلع مثلث را می توان به عنوان یک مرز خطی در نظر گرفت. برای طراحی شبکه‌ای که نقاط داخل مثلث را طبقه بندی می‌کند، برای بررسی هر شرایط مرزی به سه نورون نیاز داریم:

$$2x + y \leq 6$$
$$y \geq 0$$
$$y < 2x - 2$$

نورون فعال می‌شود و خروجی ۱ می‌دهد؛ اگر همه این شرایط به طور همزمان برآورده شوند.

کد این فرایند به صورت زیر خواهد بود:

```python
class McCulloch_Pitts_neuron():
    def __init__(self, weights, threshold):
        self.weights = weights    # weights
        self.threshold = threshold    # threshold

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

def Area(x, y):
    neur1 = McCulloch_Pitts_neuron([2, -1], 2)  # for y < 2x - 2
    neur2 = McCulloch_Pitts_neuron([-2, -1], -6)  # for y < -2x + 6
    neur3 = McCulloch_Pitts_neuron([0, 1], 0)  # for y > 0

    # get outputs of the neurons
    z1 = neur1.model(np.array([x, y]))
    z2 = neur2.model(np.array([x, y]))
    z3 = neur3.model(np.array([x, y]))

    final_neuron = McCulloch_Pitts_neuron([1, 1, 1], 3)
    z_final = final_neuron.model(np.array([z1, z2, z3]))

    return z_final
```

برای تعیین اینکه آیا یک نقطه در داخل مثلث است یا خیر، باید هر سه شرط را به طور همزمان برآورده کنیم. اگر از یک نورون خروجی (نرون نهایی) استفاده شود که نتایج سه نورون را ترکیب می‌کند، می‌توان این امر را بررسی نمود:

وزن ها: [۱، ۱، ۱]

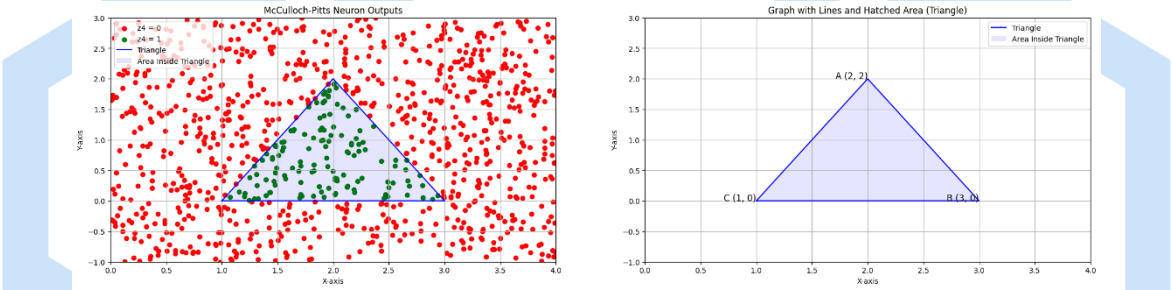
آستانه: ۳

اگر هر سه نورون خروجی ۱ را داشته باشند (که نشان می دهد نقطه داخل مثلث است) از این نورون ۱ و در غیر این صورت ۰ خواهد بود.

نقاط به صورت زیر تولید شده و به شبکه تزریق می‌شوند:

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

red_points = []
green_points = []

for i in range(num_points):
    z_value = Area(x_values[i], y_values[i])
    if z_value == 0:  # z value is 0
        red_points.append((x_values[i], y_values[i]))
    else:  # z value is 1
        green_points.append((x_values[i], y_values[i]))


red_x, red_y = zip(*red_points)
green_x, green_y = zip(*green_points)
```

و در نهایت با استفاده از نقاط بدست آمده‌ی قرمز و سبز که نمایانگر کلاس آن‌هاست، به صورت زیر نمایش داده می‌شوند:



شکل ۳- نمایش خروجی نورون‌ها به همراه خطوط محاسبه شده

افزودن توابع فعال‌سازی مختلف

برای مشاهده تأثیر توابع فعال‌سازی مختلف بر فرآیند تصمیم‌گیری، موارد زیر را بررسی می‌کنیم:

- تابع فعال‌سازی پله‌ای:

این تابع عملا در مدل McCulloch-Pitts استفاده شده است که بر اساس اینکه مجموع وزنی آستانه را برآورده کند یا خیر، ۰ یا ۱ را برمی‌گرداند.

- تابع فعال‌سازی Sigmoid:

استفاده از تابع فعال‌سازی Sigmoid خروجی احتمالاتی را فراهم می‌کند که اگر پیاده‌سازی شود، نیاز به تنظیم آستانه‌ها و تفسیر خروجی به صورت احتمالاتی خواهد داشت.

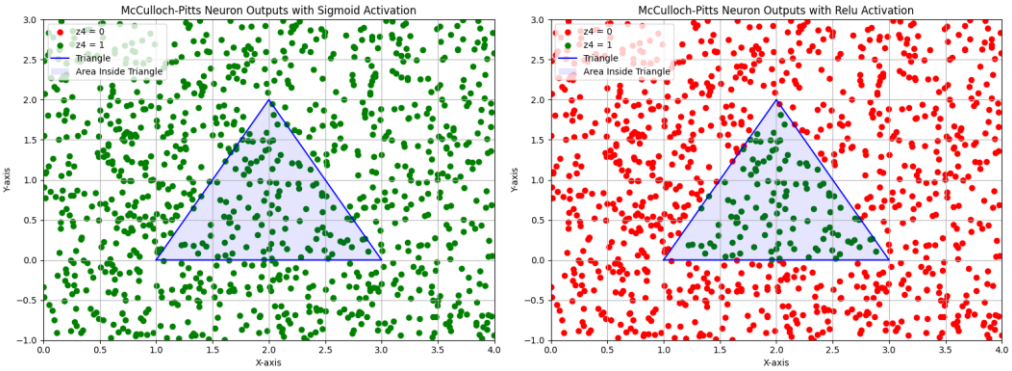
تابع فعال‌سازی ReLU:

معمولاً در لایه‌های مخفی شبکه‌های عصبی استفاده می‌شود و نه به عنوان تابع فعال‌سازی نهایی در طبقه‌بندی باینری.

برای بررسی این موضوع، کد اصلی به صورت کامل‌تر بازنویسی شده‌است:

```
class McCulloch_Pitts_neuron():
    def __init__(self, weights, threshold, activation='step'):
        self.weights = weights    # weights
        self.threshold = threshold    # threshold
        self.activation = activation  # activation function

    def model(self, x):
        net_input = np.dot(self.weights, x)
        if self.activation == 'step':
            return 1 if net_input >= self.threshold else 0
        elif self.activation == 'sigmoid':
            return 1 / (1 + np.exp(-net_input))
        elif self.activation == 'relu':
            return 0 if net_input < self.threshold else 1
        else:
            raise ValueError('Unknown activation function')

def Area(x, y, activation='step'):
    neur1 = McCulloch_Pitts_neuron([2, -1], 2, activation)  # for y < 2x - 2
    neur2 = McCulloch_Pitts_neuron([-2, -1], -6, activation)  # for y < -2x + 6
    neur3 = McCulloch_Pitts_neuron([0, 1], 0, activation)  # for y > 0

    # get outputs of the neurons
    z1 = neur1.model(np.array([x, y]))
    z2 = neur2.model(np.array([x, y]))
    z3 = neur3.model(np.array([x, y]))

    if activation == 'sigmoid':
        z1 = 1 if z1 >= 0.45 else 0
        z2 = 1 if z2 >= 0.45 else 0

    final_neuron_threshold = 2 if activation == 'sigmoid' else 3
    final_neuron = McCulloch_Pitts_neuron([1, 1, 1], final_neuron_threshold, activation)
    z_final = final_neuron.model(np.array([z1, z2, z3]))

    if activation == 'sigmoid':
        return 1 if z_final >= 0.45 else 0
    elif activation == 'relu':
        return z_final
    else:
```

خروجی این دو فعال ساز در این شبکه به صورت زیر می‌باشد:



شکل ۴- نمایش خروجی نورون‌ها با دو تابع فعال ساز **ReLU** (راست) و **Sigmoid** (چپ) به همراه خطوط محاسبه شده

همانطور که گفته شد نیاز است تا آستانه‌ها بر اساس این توابع دوباره تنظیم شوند، که این امر برای تابع ReLU پاسخ‌گو بوده اما برای تابع sigmoid متاسفانه نتوانستم تنظیم متناسب با پاسخ مدنظر سوال بدست آورم. در کل اما می‌توان دید که برای بدست آوردن این دست از نتایج که یک خط تصمیم گیرنده است، وجود یک فعال ساز منقطع کننده (که عموما مشتق پذیر نیستند) کار ساده‌تری نسبت به روش‌های احتمالاتی است، هر چند که این دسته می‌توانند عمومیت[1] بیشتری به شبکه بدهند.

[1] Generalization

سوال ۲

۲

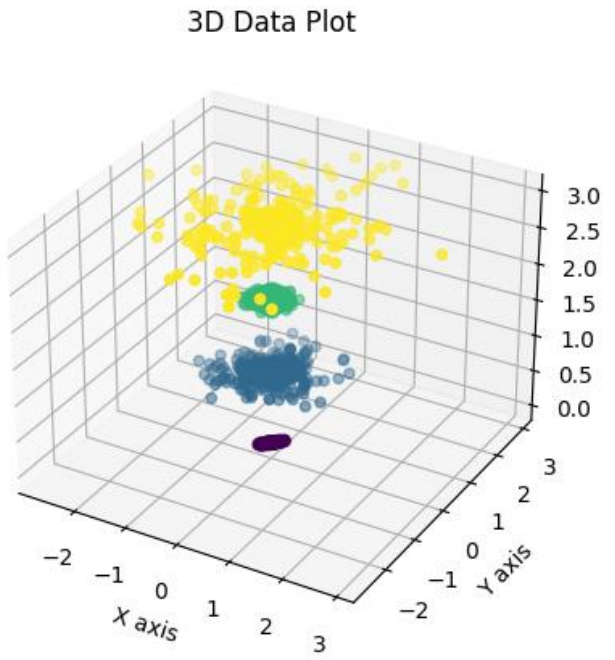
برای استخراج داده‌ها از داده اصلی به صورت زیر عمل شده‌است:

```
import pandas as pd
import scipy.io as scio
import numpy as np

mat = scio.loadmat('99.mat')
variables = scio.whosmat('99.mat')

data1 = mat['X098_DE_time']
data2 = mat['X098_FE_time']

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

normal=[]

for i in range(700):
    a=np.concatenate([df1.values[i*200:200+i*200], df2.values[i*200:200+i*200]])
    a=np.reshape(a,(200,2))
    normal.append(a)


normal=np.squeeze(normal)
normal.shape
```

کلاس‌های فالت به صورت مشابه به صورت زیر بدست آمده‌اند:

```
mat = scio.loadmat('107.mat')
variables = scio.whosmat('107.mat')

data1 = mat['X107_DE_time']
data2 = mat['X107_FE_time']

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

f1=[]

for i in range(600):
    a=np.concatenate([df1.values[i*200:200+i*200], df2.values[i*200:200+i*200]])
    a=np.reshape(a,(200,2))
    f1.append(a)


f1=np.squeeze(f1)
```

در کل، ۷۰۰ داده ۲۰۰ در ۲ از کلاس نرمال و ۶۰۰ داده ۲۰۰ در ۲ از کلاس‌های فالت بدست آمد که با هم concat شدند و به ازای هر اندیس متعلق به آن کلاس در کل داده برای داده‌های نرمال، مقدار ۰ و برای داده‌های فالت مقدار ۱، ۲ و ۳ به عنوان برچسب تعریف شد. نمایش داده‌ها در یک فضای دو بعدی به صورت زیر می‌باشد:



شکل ۵- توزیع داده‌ها (نرمال با زرد و فالت با بنفش)

توزیع داده‌ها نشان‌گر درهم تنیدگی داده‌های فالت با داده‌های نرمال است. همچنین واریانس داده‌های نرمال بیشتر از داده‌های خطاست.

ویژگی‌های زیر استخراج شدند.

```
[ ]  1  std_X= X.std(axis=1)
     2  peak_X= X.max(axis=1)
     3
     4  from scipy.stats import skew, kurtosis
     5  skewness_X= skew(X, axis=1)
     6  kurtosis_X= kurtosis(X, axis=1)
     7
     8  crest_factor_X = np.max(X, axis=1) / np.sqrt(np.mean(X**2, axis=1))
     9  ptp_X= np.ptp(X, axis=1)
    10  mean_X= np.mean(X, axis=1)
    11  rms_X = np.sqrt(np.mean(X**2, axis=1))
    12  abs_mean_X= np.mean(np.abs(X), axis=1)
```

```
[ ]  1  X_new=np.concatenate([std_X, peak_X, skewness_X, kurtosis_X, crest_factor_X, ptp_X, mean_X, rms_X, abs_mean_X],axis=1)
```

```
[ ]  1  X_new.shape
```

```
(2500, 18)
```

ج

در زمینه یادگیری ماشین، مجموعه داده‌ها معمولاً به سه بخش اصلی تقسیم می‌شوند: آموزش، اعتبارسنجی و آزمایش. هر یک از این بخش ها هدف مشخصی را در فرآیند توسعه و ارزیابی مدل انجام می دهند.

۱. مجموعه آموزشی

- مجموعه آموزشی برای آموزش مدل یادگیری ماشین استفاده می شود. مدل از طریق این مجموعه، الگوها و روابط اساسی در داده ها را یاد می گیرد.
- مدل با داده های ورودی همراه با برچسب های مربوطه (در یادگیری نظارت شده) تغذیه می شود.
- پارامترهای داخلی خود را برای به حداقل رساندن خطا در پیش بینی های خود بر اساس برچسب های ارائه شده تنظیم می کند.

۲. مجموعه اعتبارسنجی

- مجموعه اعتبار سنجی برای تنظیم دقیق فراپارامترهای مدل و ارائه یک ارزیابی بی طرفانه از مدل در طول فرآیند آموزش استفاده می شود. این به نظارت بر عملکرد مدل و جلوگیری از نصب بیش از حد کمک می کند.
- پس از هر دوره (گذر کامل از مجموعه آموزشی)، عملکرد مدل بر روی مجموعه اعتبارسنجی ارزیابی می شود.

- مجموعه اعتبار سنجی بازخوردی را در مورد میزان تعمیم مدل به داده های دیده نشده ارائه می دهد.
- برای تصمیم گیری در مورد تنظیم هایپرپارامتر (به عنوان مثال، نرخ یادگیری، اندازه دسته، انتخاب های معماری) استفاده می شود.
- در صورتی که عملکرد مدل در مجموعه اعتبارسنجی شروع به کاهش کند، می توان از آن برای توقف زودهنگام فرآیند آموزش استفاده کرد که نشان دهنده بیش از حد برازش است.
- امکان مقایسه مدل‌های مختلف و انتخاب مدلی که بهترین عملکرد را در داده‌های اعتبارسنجی دارد را می‌دهد.

۳. مجموعه تست

- مجموعه آزمون ارزیابی نهایی عملکرد مدل را پس از مراحل آموزش و اعتبار سنجی ارائه می دهد. برای تخمین میزان عملکرد مدل بر روی داده های کاملاً جدید و دیده نشده استفاده می شود.
- مجموعه تست در طول مراحل آموزش یا اعتبار سنجی استفاده نمی‌شود.
- هنگامی که مدل به طور کامل آموزش داده شد، مدل نهایی بر روی مجموعه تست ارزیابی می شود.
- مجموعه آزمون به ارزیابی توانایی مدل برای تعمیم به داده های جدید کمک می کند.
- معیارهای نهایی عملکرد (به عنوان مثال، دقت، دقت، یادآوری) بر روی این دسته از داده‌ها محاسبه می‌شود و میزان اثربخشی مدل را نشان می‌دهد.

کد این بخش به صورت ساده به شرح زیر است:

```
from sklearn.model_selection import train_test_split

X_train_val, X_test, y_train_val, y_test = train_test_split(data.values, y, test_size=0.2, random_state=14) #shuffling is always true
X_train, X_valid, y_train, y_valid = train_test_split(X_train_val, y_train_val, test_size=0.25, random_state=14)
```

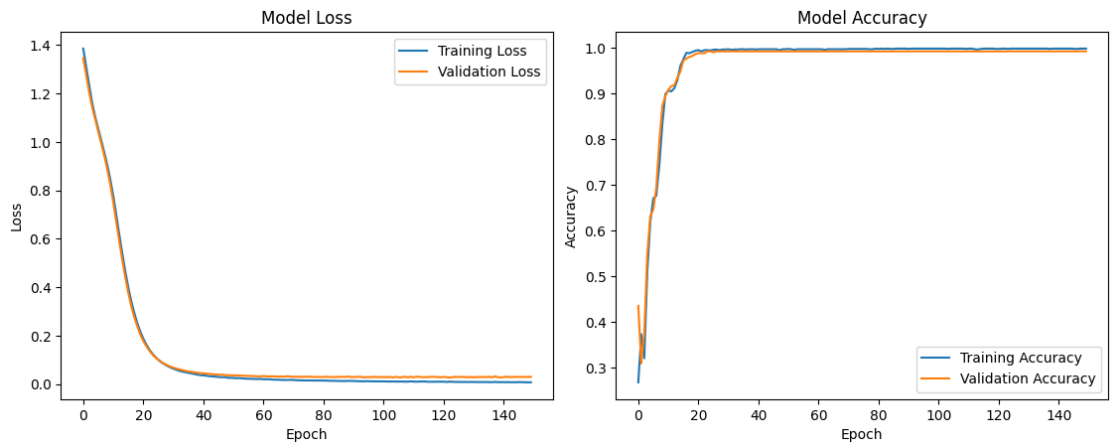
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)
X_valid_normalized = scaler.transform(X_valid)
```

۲

مدل به صورت زیر پیاده سازی شد:

```
from keras.callbacks import Callback
from keras.models import Sequential
from keras.layers import Input, Dense, Activation, Flatten, Dropout

num_classes = 4
input_shape= X_train_normalized[0].shape

# Create the model
model = Sequential()
model.add(Dense(24, activation='relu', input_shape=input_shape))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.summary()

# Define the optimizer and compile the model
import keras
opt = keras.optimizers.Adam(learning_rate=0.0005)
model.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
history= model.fit(X_train_normalized, y_train , batch_size=64, epochs=150, validation_data=(X_valid_normalized, y_valid))
```

نمودار loss و accuracy به صورت زیر می‌باشد:



شکل ۶- نمودار **loss** و **accuracy** شبکه

بعد از تغییر های بسیار برای نرخ آموزش، این مقدار بهترین حالت برای دوری از سَدل هایی است که در این نمودار ها به صورت خیلی کم دیده می‌شوند.

داده‌های تست نیز به صورت زیر بررسی شدند:

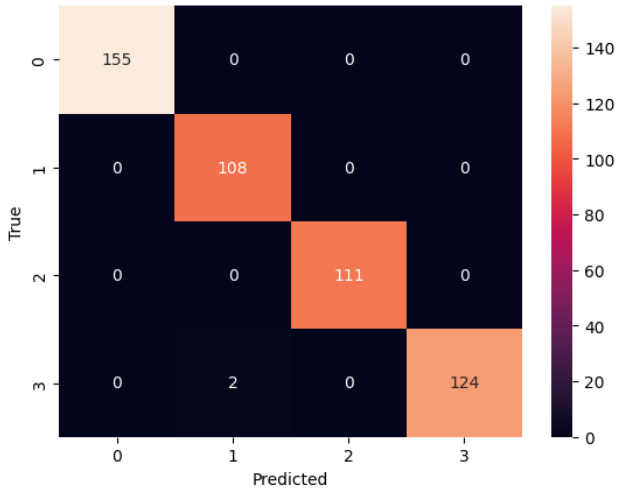
```
loss, accuracy = model.evaluate(X_test_normalized, y_test, batch_size=64)
print('Test loss:', loss)
print('Test accuracy:', accuracy)
```

```
8/8 [==============================] - 0s 5ms/step - loss: 0.0216 -
accuracy: 0.9960
Test loss: 0.021598493680357933
Test accuracy: 0.9959999918937683
```

```python
from keras.utils import to_categorical
from sklearn.metrics import roc_auc_score, recall_score, f1_score, precision_score, classification_report, confusion_matrix

# Assuming 'model' is your trained model and 'X_test', 'y_test' are your test datasets
y_pred = model.predict(X_test_normalized)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(to_categorical(y_test, 4), axis=1)

# Print classification report
print(classification_report(y_test_classes, y_pred_classes, target_names=['Normal', 'Fault 1', 'Fault 2', 'Fault 3']))

# Additional metrics
auc_score = roc_auc_score(to_categorical(y_test_classes, 4), to_categorical(y_pred_classes, 4), multi_class='ovr')
recall = recall_score(y_test_classes, y_pred_classes, average='micro')
f1 = f1_score(y_test_classes, y_pred_classes, average='micro')
precision = precision_score(y_test_classes, y_pred_classes, average='micro')

print('Test AUC:', auc_score)
print('Test Recall:', recall)
print('Test F1-score:', f1)
print('Test Precision:', precision)
```

```
16/16 [==============================] - 0s 4ms/step
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00       155
     Fault 1       0.98      1.00      0.99       108
     Fault 2       1.00      1.00      1.00       111
     Fault 3       1.00      0.98      0.99       126

    accuracy                           1.00       500
   macro avg       1.00      1.00      1.00       500
weighted avg       1.00      1.00      1.00       500

Test AUC: 0.9973781179138322
Test Recall: 0.996
Test F1-score: 0.996
Test Precision: 0.996
```

ماتریس در هم ریختگی نیز به صورت زیر می‌باشد:



شکل ۷- ماتریس درهم ریختگی طبقه بندی دیتاست **CWRU Bearing**

در کل فرایند طبقه بندی به خوبی انجام شده و اورفیت هم ندارد. تنها ۲ داده از کل داده‌های تست به اشتباه تشخیص داده شده‌است.

۳

شبکه به صورت زیر بازنویسی شد:

```
from keras.callbacks import Callback
from keras.models import Sequential
from keras.layers import Input, Dense, Activation, Flatten, Dropout
from keras.optimizers import RMSprop

num_classes = 4
input_shape = X_train_normalized[0].shape

# One-hot encode y_train and y_valid
y_train_categorical = keras.utils.to_categorical(y_train, num_classes)
y_valid_categorical = keras.utils.to_categorical(y_valid, num_classes)
y_test_categorical = keras.utils.to_categorical(y_test, num_classes)

# Create the model
model = Sequential()
model.add(Dense(24, activation='relu', input_shape=input_shape))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.summary()

# Define the optimizer and compile the model with mean_squared_error loss
opt = RMSprop(learning_rate=0.0005)
model.compile(loss='mean_squared_error', optimizer=opt, metrics=['accuracy'])

# Train the model
history = model.fit(X_train_normalized, y_train_categorical, batch_size=64, epochs=150, validation_data=(X_valid_normalized, y_valid_categorical))
```

RMSprop (تکثیر ریشه مربعی میانگین) یک الگوریتم بهینه‌سازی نرخ یادگیری تطبیقی است که توسط جفری هینتون توسعه داده شده است تا مشکلات موجود در روش‌های دیگر بهینه‌سازی را به ویژه در زمینه آموزش شبکه‌های عصبی عمیق، حل کند. RMSprop به گونه‌ای طراحی شده است که حتی وقتی گرادیان‌ها به طور گسترده‌ای در بزرگی متفاوت هستند، نرخ یادگیری ثابتی را حفظ کند و به این ترتیب سرعت و پایداری همگرایی را بهبود بخشد.

RMSprop با حفظ میانگین متحرک مربعات گرادیان‌ها برای هر پارامتر عمل می‌کند که به نرمال‌سازی گرادیان‌ها و کاهش نوسانات کمک می‌کند. این مکانیزم باعث اطمینان از یک فرآیند همگرایی پایدارتر و کارآمدتر می‌شود.

RMSprop میانگین متحرک مربعات گرادیان‌ها را حفظ می‌کند که به این صورت به‌روزرسانی می‌شود:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

که در آن:

$E[g^2]_t$ میانگین متحرک مربعات گرادیان‌ها در زمان t است.

γ (ضریب کاهش) معمولاً برابر با ۰.۹ است.

g_t گرادیان در زمان t است.

پارامترها با نرمال‌سازی گرادیان فعلی با استفاده از میانگین متحرک مربعات گرادیان‌ها به‌روزرسانی می‌شوند:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

که در آن:

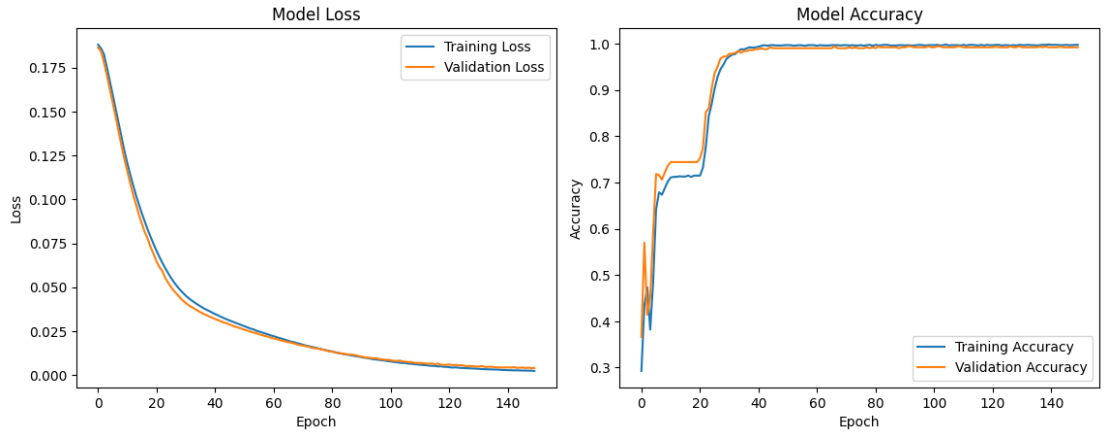
θ پارامتر در زمان t است.

η نرخ یادگیری است.

ϵ یک ثابت کوچک که به منظور جلوگیری از تقسیم بر صفر اضافه می‌شود.

RMSprop به صورت مفهومی شبیه به بهینه‌ساز Adam است که از میانگین‌های متحرک گرادیان‌ها و مربعات آن‌ها استفاده می‌کند. Adam را می‌توان به عنوان توسعه‌ای از RMSprop با یک ترم تصحیح سوگیری اضافی دید.

نتایج شبکه به صورت زیر است.



شکل ۸- نمودار **loss** و **accuracy** شبکه با بهینه ساز و تابع ضرر جدید

مشاهده می‌شود که مدل کمی در نقطه سَدل درگیر بوده اما توانسته از آن گذر کند.

داده‌های تست نیز به صورت زیر بررسی شدند:

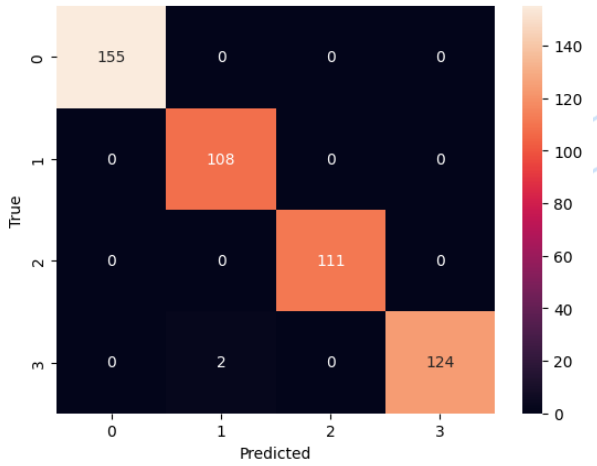
```
loss, accuracy = model.evaluate(X_test_normalized, y_test, batch_size=64)
print('Test loss:', loss)
print('Test accuracy:', accuracy)
```

```
8/8 [==============================] - 0s 5ms/step - loss: 0.0032 -
accuracy: 0.9960
Test loss: 0.003183630295097828
Test accuracy: 0.9959999918937683
```

```python
from keras.utils import to_categorical
from sklearn.metrics import roc_auc_score, recall_score, f1_score, precision_score, classification_report, confusion_matrix

# Assuming 'model' is your trained model and 'X_test', 'y_test' are your test datasets
y_pred = model.predict(X_test_normalized)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(to_categorical(y_test, 4), axis=1)

# Print classification report
print(classification_report(y_test_classes, y_pred_classes, target_names=['Normal', 'Fault 1', 'Fault 2', 'Fault 3']))

# Additional metrics
auc_score = roc_auc_score(to_categorical(y_test_classes, 4), to_categorical(y_pred_classes, 4), multi_class='ovr')
recall = recall_score(y_test_classes, y_pred_classes, average='micro')
f1 = f1_score(y_test_classes, y_pred_classes, average='micro')
precision = precision_score(y_test_classes, y_pred_classes, average='micro')

print('Test AUC:', auc_score)
print('Test Recall:', recall)
print('Test F1-score:', f1)
print('Test Precision:', precision)
```

```
16/16 [==============================] - 0s 5ms/step
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00       155
     Fault 1       0.98      1.00      0.99       108
     Fault 2       1.00      1.00      1.00       111
     Fault 3       1.00      0.98      0.99       126

    accuracy                           1.00       500
   macro avg       1.00      1.00      1.00       500
weighted avg       1.00      1.00      1.00       500

Test AUC: 0.9973781179138322
Test Recall: 0.996
Test F1-score: 0.996
Test Precision: 0.996
```

ماتریس در هم ریختگی نیز به صورت زیر می‌باشد:



شکل ۹- ماتریس درهم ریختگی طبقه بندی دیتاست **CWRU Bearing** در حالت دوم

نتایج مشابه حالت قبلی است اما با توجه به رفتار در حین آموزش، روش اول گزینه بهتری برای آموزش است.

۴

- اعتبار سنجی متقاطع[1] K-Fold:

K-Fold یک تکنیک نمونه‌گیری مجدد است که برای ارزیابی مدل‌های یادگیری ماشین بر روی یک نمونه داده محدود استفاده می‌شود. این روش دارای یک پارامتر واحد است، k، که به تعداد گروه هایی که یک نمونه داده معین باید به آنها تقسیم شود، اشاره دارد. این تضمین می کند که هر نمونه در مجموعه داده فرصت استفاده در هر دو مجموعه آموزشی و اعتبار سنجی را دارد.

مزایای K-Fold Cross-validation:

۱. سوگیری را کاهش می دهد زیرا هر مشاهده ای هم برای آموزش و هم برای اعتبار سنجی استفاده می شود.
۲. واریانس را کاهش می‌دهد زیرا تقسیم‌های آموزشی/ اعتبارسنجی در هر فولد متفاوت است.
۳. تخمین دقیق تری از عملکرد مدل ارائه می دهد.

- اعتبار سنجی متقاطع K-Fold طبقه بندی شده:

اعتبار سنجی متقاطع K-Fold گونه ای از اعتبارسنجی متقاطع K-Fold است که در آن چین ها با حفظ درصد نمونه ها برای هر کلاس ساخته می شوند. این به ویژه در مواردی که مجموعه داده های نامتعادل هستند، که در آن برخی از کلاس ها کمتر ارائه می شوند، مفید است.

مزایای اعتبار سنجی متقاطع K-Fold طبقه بندی شده:

۱. اطمینان حاصل می کند که هر فولد نماینده توزیع کلی کلاس است.
۲. تخمین بهتری از عملکرد مدل برای مجموعه داده های نامتعادل ارائه می دهد.
۳. خطر ارزیابی مغرضانه را کاهش می دهد.

[1] Cross-validation

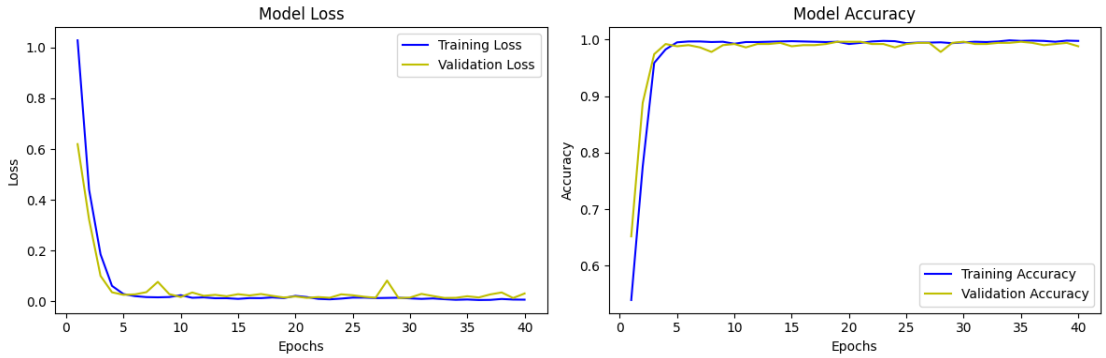
انتخاب روش

با توجه به اینکه اعتبارسنجی متقاطع طبقه بندی شده K-Fold نتایج دقیق و قابل اعتمادتری را برای مجموعه داده‌هایی با توزیع کلاس نامتعادل ارائه می دهد، اغلب ترجیح داده می‌شود. به همین دلیل، برای اجرای آموزش و ارزیابی مدل Stratified K-Fold Cross-validation انتخاب می‌شود.

این روش به صورت زیر پیاده‌سازی شده است:

```
# Perform K-fold cross-validation
skf = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=14)
for train_index, test_index in skf.split(X_new, y):
    x_train, x_test = X_new[train_index], X_new[test_index]
    y_train, y_test = y_categorical[train_index], y_categorical[test_index]

    # Define the input shape and number of classes
    input_shape = x_train.shape[1]
    num_classes = y_categorical.shape[1]

    # Create the model
    model = Sequential()
    model.add(Dense(24, activation='relu', input_shape=(input_shape,)))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    model.summary()

    # Define the optimizer and compile the model
    import keras
    opt = keras.optimizers.Adam(learning_rate=0.005)
    model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

    # Define the callback for storing loss and accuracy history
    class LossAccuracyCallback(Callback):
        def on_train_begin(self, logs={}):
            self.losses = []
            self.accs = []
            self.valid_losses = []
            self.valid_accs = []

        def on_epoch_end(self, epoch, logs={}):
            self.losses.append(logs.get('loss'))
            self.accs.append(logs.get('accuracy'))
            self.valid_losses.append(logs.get('val_loss'))
            self.valid_accs.append(logs.get('val_accuracy'))

    callback = LossAccuracyCallback()

    # Train the model with callback
    history = model.fit(x_train, y_train, batch_size=64, epochs=10, validation_data=(x_test, y_test), callbacks=[callback])
```

نتایج به صورت زیر می‌باشد:



شکل ۱۰- نمودار **loss** و **accuracy** شبکه با **SK-Fold**

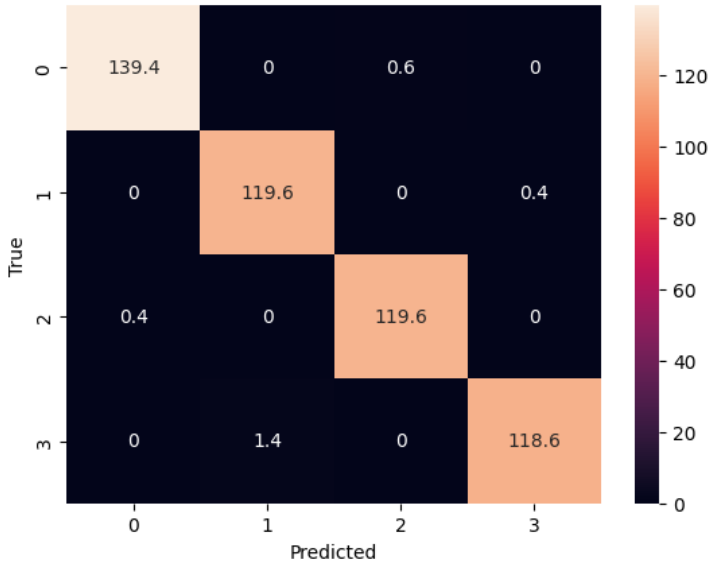
نمودار نشان دهنده روند مطمئن تری از آموزش نسبت به دو روش قبلی است. در این جا به صورت یکنوا و بدون برخورد با حالت‌های سدل، مدل آموزش دیده است.

داده‌های تست نیز به صورت زیر بررسی شدند:

```
# Print the average results across all folds
print("Average loss:", np.mean(losses))
print("Average accuracy:", np.mean(accuracies))
print("Average AUC score:", np.mean(auc_scores))
print("Average recall score:", np.mean(recall_scores))
print("Average F1 score:", np.mean(f1_scores))
print("Average precision score:", np.mean(precision_scores))
print("\nAverage confusion matrix:")

import seaborn as sns
sns.heatmap(np.mean(confusion_matrices, axis=0), annot=True, fmt='g')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Average loss: 0.016832860372960567
Average accuracy: 0.9944000005722046
Average AUC score: 0.9998114035087718
Average recall score: 0.9943452380952381
Average F1 score: 0.9943154688332368
Average precision score: 0.9943366396111424
```

ماتریس درهم‌ریختگی نیز به صورت زیر می‌باشد:



شکل ۱۱- ماتریس درهم‌ریختگی با روش **SK-Fold**

در کل می‌توان گفت که این روش گزینه بسیار بهتری نسبت به دو روش قبلی است، چرا که علاوه بر عمومیت داشتن بیشتر در مدل آموزش دیده، سرعت پردازش برای رسیدن به یک نتیجه مشابه بسیار کمتر بوده است.

سوال ۳

۱

در این سوال از داده‌های دارو موجود در Kaggle استفاده شد، چرا که دیتاست طبقه بندی پوشش جنگلی با ارور ۴۰۳ همراه بود. دیتا به صورت زیر است:

جدول ۱- داده‌های دارو

	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
...	...	...	...	...	...	...
195	56	F	LOW	HIGH	11.567	drugC
196	16	M	LOW	HIGH	12.006	drugC
197	52	M	NORMAL	HIGH	9.894	drugX
198	23	M	NORMAL	NORMAL	14.020	drugX
199	40	F	LOW	NORMAL	11.349	drugX

200 rows × 6 columns

هر ستون دارای ویژگی‌های گسسته یا پیوسته از نمونه‌هاست که برای استفاده داده‌ها آماده می‌شوند. بر اساس نام ستون‌ها، ویژگی‌های طبقه‌بندی و متغیر هدف را به صورت زیر انکد می‌کنیم:

```python
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Identify categorical columns
categorical_features = data.select_dtypes(include=['object']).columns.tolist()

# Separate features and target (assuming 'Drug' is the target column)
target_column = 'Drug'
if target_column in categorical_features:
    categorical_features.remove(target_column)

# Encode target variable
le = LabelEncoder()
y = le.fit_transform(data[target_column])

# One-hot encode categorical features
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), categorical_features)],
    remainder='passthrough')

# Extract features
X = data.drop(columns=[target_column])
X = preprocessor.fit_transform(X)
```

داده‌های آموزش و تست هم به صورت زیر جدا می‌شوند.

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=14)
```

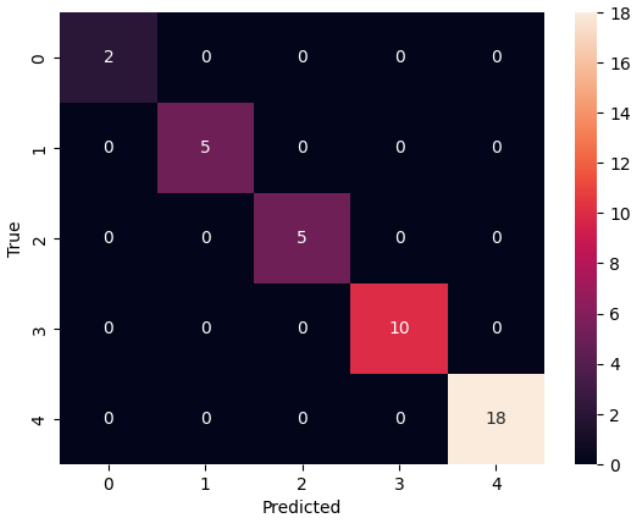
به صورت زیر نیز توسط درخت تصمیم، طبقه بندی انجام می‌گیرد:

```python
clf = DecisionTreeClassifier(random_state=14)
clf.fit(X_train, y_train)
```

نتایج به صورت زیر است:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         5
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00        10
           4       1.00      1.00      1.00        18

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

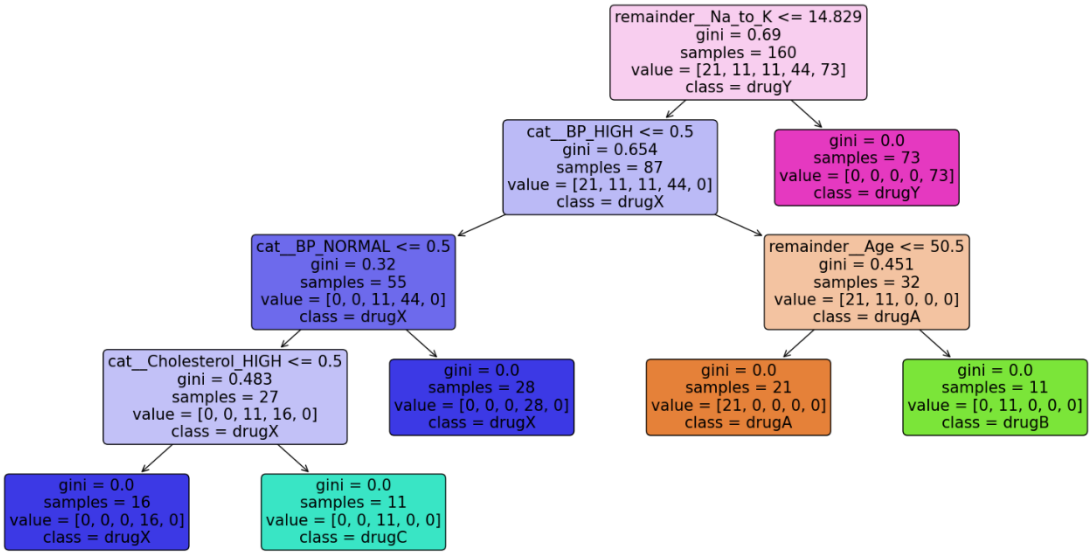
ماتریس درهم‌ریختگی به صورت زیر است:



شکل ۱۲- ماتریس درهم ریختگی درخت تصمیم

اگر داده ها نسبتا ساده و به خوبی از هم جدا شده باشند، درخت تصمیم ممکن است به راحتی به دقت کامل دست یابد. از طرفی، دقت کامل در مجموعه تست گاهی اوقات می تواند نشان دهنده بیش از حد برازش باشد، به خصوص اگر مجموعه تست کوچک باشد یا معرف داده های دیده نشده نباشد. این تطبیق بیش از حد زمانی اتفاق می‌افتد که مدل داده‌های آموزشی را به خوبی یاد می‌گیرد، از جمله نویز و جزئیاتی که به داده‌های جدید تعمیم نمی‌یابند.

نمایی از درخت استفاده شده در این روش به صورت زیر است:



شکل ۱۳- درخت تصمیم برای طبقه بندی داده‌های دارو

درخت با تقسیم داده‌ها بر اساس نسبت Na_to_K شروع می‌کند و از آستانه ۱۴.۸۲۹ استفاده می‌کند. این تقسیم داده‌ها را به دو گروه جدا می‌کند:

- اگر Na_to_K کمتر یا برابر با ۱۴.۸۲۹ باشد، درخت به گره فرزند چپ حرکت می‌کند.
- اگر Na_to_K بیشتر از ۱۴.۸۲۹ باشد، به گره فرزند راست حرکت می‌کند.
- گره فرزند راست گره ریشه: این گره دارای ناخالصی جینی صفر است، به این معنی که یک گره خالص است و همه ۷۳ نمونه به عنوان داروی Y طبقه‌بندی شده‌اند. نیازی به تقسیم بیشتر نیست زیرا تمام نمونه‌ها از یک کلاس هستند.
- گره فرزند چپ گره ریشه: برای نمونه‌هایی با Na_to_K کمتر یا برابر با ۱۴.۸۲۹، تقسیم بعدی بر اساس فشار خون بالا (BP_HIGH) انجام می‌شود. آستانه استفاده شده ۰.۵ است:
- اگر BP_HIGH کمتر یا برابر با ۰.۵ باشد، درخت به گره فرزند چپ حرکت می‌کند.
- اگر BP_HIGH بیشتر از ۰.۵ باشد، به گره فرزند راست حرکت می‌کند.
- گره فرزند چپ گره BP_HIGH: این گره بر اساس فشار خون نرمال (BP_NORMAL) و با استفاده از آستانه ۰.۵ بیشتر تقسیم می‌شود:
  - اگر BP_NORMAL کمتر یا برابر با ۰.۵ باشد، درخت به گره فرزند چپ حرکت می‌کند.
  - اگر BP_NORMAL بیشتر از ۰.۵ باشد، به گره فرزند راست حرکت می‌کند.
- گره فرزند راست گره BP_HIGH: این گره بر اساس سن و با استفاده از آستانه ۵۰.۵ تقسیم می‌شود:
  - اگر سن کمتر یا برابر با ۵۰.۵ باشد، درخت به گره فرزند چپ حرکت می‌کند.
  - اگر سن بیشتر از ۵۰.۵ باشد، به گره فرزند راست حرکت می‌کند.

گره‌ها بر اساس ویژگی‌هایی مانند Cholesterol_HIGH به تقسیم ادامه می‌دهند که منجر به چندین گره خالص (ناخالصی جینی صفر) می‌شود که هر گره فقط شامل نمونه‌های یک کلاس است.

۲

در کنار این، مقدار عمق و تعداد جداسازی مقادیر پیوسته نیز بررسی شد که نتایج به صورت زیر می‌باشد:

```
Depth Results: {5: 1.0, 10: 1.0, 15: 1.0, 20: 1.0, 25: 1.0}
Split Results: {2: 1.0, 10: 1.0, 20: 1.0, 50: 0.75, 100: 0.7}
```

درخت تصمیم کاملاً در اعماق مختلف عمل می کند، که نشان دهنده بیش از حد برازش بالقوه است. در کنار این، مقادیر کمتر min_samples_split (۲، ۱۰، ۲۰) منجر به دقت بالا می‌شود، که نشان می‌دهد درخت تصمیم می‌تواند قوانین بسیار خاصی ایجاد کند که داده‌های آزمایش را کاملاً طبقه‌بندی کند.

با افزایش مقدار min_samples_split (۵۰، ۱۰۰)، دقت کاهش می یابد. این نشان می دهد که درخت تصمیم قادر به ایجاد قوانین خاص نیست و منجر به کاهش عملکرد می شود. با این حال، مقادیر بالاتر به طور کلی با کاهش پیچیدگی و تعمیم بیشتر درخت، به کاهش بیش از حد برازش کمک می کند.

۳

- Random Forest

روش Random Forest با ساخت چندین درخت تصمیم‌گیری در مرحله آموزش و ترکیب خروجی‌های آن‌ها، مشکل بیش‌برازش و واریانس بالا را کاهش می‌دهد. این روش از دو اصل مهم استفاده می‌کند:

- Bootstrap Aggregating (Bagging): چندین زیرمجموعه از داده‌های آموزشی با نمونه‌گیری با جایگزینی ایجاد می‌شود و هر زیرمجموعه برای آموزش یک درخت تصمیم‌گیری استفاده می‌شود.
- Random Feature Selection: در هر تقسیم در درخت تصمیم‌گیری، یک زیرمجموعه تصادفی از ویژگی‌ها انتخاب می‌شود و بهترین تقسیم از این زیرمجموعه انتخاب می‌شود. این امر تصادفی بودن را افزایش داده و اطمینان حاصل می‌کند که درخت‌ها کمتر با یکدیگر همبسته باشند.

پیش‌بینی نهایی در Random Forest با ترکیب پیش‌بینی‌های همه درخت‌های منفرد (معمولاً با رأی‌گیری اکثریت برای دسته‌بندی) انجام می‌شود. این ترکیب واریانس مدل را کاهش داده و منجر به بهبود دقت و استحکام می‌شود.

- AdaBoost

روش AdaBoost که مخفف Adaptive Boosting است، یک روش گروهی قوی دیگر است که چندین مدل ضعیف را برای ایجاد یک دسته‌بندی‌کننده قوی ترکیب می‌کند. برخلاف Random Forest، AdaBoost مدل‌های ضعیف را به صورت متوالی آموزش می‌دهد و وزن نمونه‌های آموزشی را بر اساس خطاهای مدل‌های قبلی تنظیم می‌کند. نمونه‌هایی که به اشتباه دسته‌بندی شده‌اند، وزن بیشتری می‌گیرند و در نتیجه در مراحل بعدی تأثیر بیشتری خواهند داشت. هر مدل ضعیف بر اساس دقت خود یک وزن

دریافت می‌کند و پیش‌بینی نهایی به صورت ترکیب وزنی از پیش‌بینی‌های همه مدل‌های ضعیف است. این فرآیند باعث کاهش سوگیری مدل نهایی می‌شود و با تمرکز بر نمونه‌های سخت‌دسته‌بندی، دقت را بهبود می‌بخشد.

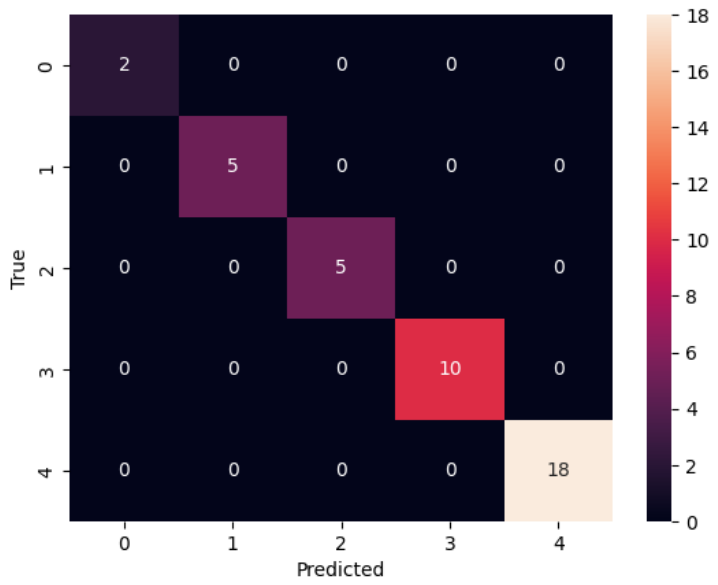
در اینجا از روش اول استفاده شد که به صورت زیر پیاده سازی شده است:

```
rf_clf = RandomForestClassifier(n_estimators=100, random_state=14)
rf_clf.fit(X_train, y_train)
```

نتایج زیر بدست آمده‌اند:

```
Random Forest Accuracy: 1.0
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         5
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00        10
           4       1.00      1.00      1.00        18

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

ماتریس درهم‌ریختگی زیر به صورت زیر است:



شکل ۱۴- ماتریس درهم‌ریختگی روش **RF**

با وجود اینکه نتیجه بدست آمده دیگر ۱۰۰ درصد است اما نگاهی نیز به GridSearchCV که در صورت سوال پیشنهاد شده خواهیم داشت. این مدل به صورت زیر پیاده می‌شود:

```python
param_grid = {
    'max_depth': [5, 10, 15, 20, 25, None],
    'min_samples_split': [2, 10, 20, 50, 100],
    'min_samples_leaf': [1, 2, 5, 10],
    'max_features': [None, 'sqrt', 'log2']
}

# Initialize the Decision Tree classifier
dt_clf = DecisionTreeClassifier(random_state=14)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=dt_clf, param_grid=param_grid, cv=5, n_jobs=-1, verbose=1)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print(f'Best Parameters: {best_params}')
```

این روش عملا یک بررسی در یک بازه معلوم از پارامترها برای پیدا کردن بهترین نتیجه از پارامترهاست.

نتایج به صورت زیر است:

```
Fitting 5 folds for each of 360 candidates, totalling 1800 fits
Best Parameters: {'max_depth': 5, 'max_features': None,
'min_samples_leaf': 1, 'min_samples_split': 2}
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         5
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00        10
           4       1.00      1.00      1.00        18

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

ماتریس درهم ریختگی نیز مشخصا قطری است و نیاز به نمایش نیست.

مشاهده می‌شود که عمق ۵ و جداسازی ۲ که در بخش قبلی بررسی شدند، به عنوان بهترین مقادیر برگردانده شده‌اند.

سوال ۴

ابتدا داده وارد کد شد و تارگت از سایر بخش ها جدا شد. در نهایت هم ۲۰ درصد از داده‌ها به عنوان داده تست جدا شدند.

```
from sklearn.model_selection import train_test_split

# Define features and target variable
X = data.drop('target', axis=1)
y = data['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=14)
```

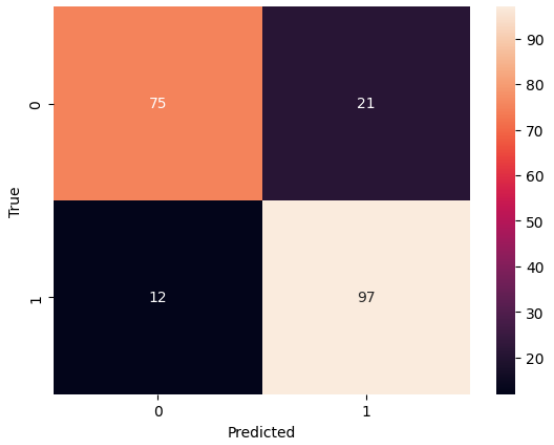
از آن جایی که فرض بر این است که داده به صورت گوسی توزیع شده، از فرم گوسی بیز به صورت زیر می‌توان استفاده کرد:

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

نتایج مدل بر روی داده تست به صورت زیر می‌باشد:

	precision	recall	f1-score	support
0	0.86	0.78	0.82	96
1	0.82	0.89	0.85	109
accuracy			0.84	205
macro avg	0.84	0.84	0.84	205
weighted avg	0.84	0.84	0.84	205

ماتریس درهم ریختگی نیز به صورت زیر می‌باشد:



شکل ۱۵- ماتریس درهم‌ریختگی

طبقه بندی کننده Gaussian Naive Bayes در مجموعه آزمایشی به دقت کلی ۸۰ درصد رسیده است. طبقه بندی کننده در کلاس ۱ در مقایسه با کلاس ۰ بهتر عمل کرد، همانطور که با فراخوان بالاتر و امتیاز F1 برای کلاس ۱ نشان داده شد. مقایسه خروجی‌های واقعی و پیش‌بینی‌شده برای نمونه‌های تصادفی نشان می‌دهد که طبقه‌بندی‌کننده پیش‌بینی‌های درستی را برای اکثر نقاط داده انتخاب‌شده، با این حال نیز چند طبقه‌بندی اشتباه هم انجام داده است.

توضیح حالت های ماکرو و میکرو

در معیارهای طبقه‌بندی، حالت‌های کلان و خرد به روش‌های میانگین‌گیری مختلف در چندین کلاس اشاره دارند:

- میانگین macro:

متریک را برای هر کلاس به طور مستقل محاسبه می کند و سپس میانگین را می گیرد و با همه طبقات بدون توجه به فراوانی آنها به طور مساوی رفتار می کند. برای مثال، دقت میانگین کلان میانگین مقادیر دقت برای هر کلاس است.

- میانگین گیری micro:

مشارکت همه کلاس ها را برای محاسبه متریک جمع می کند و تعداد کل مثبت های درست، منفی های کاذب و مثبت های کاذب را در همه کلاس ها در نظر می گیرد. برای مثال، دقت متوسط میکرو با جمع بندی مثبت های واقعی، منفی های کاذب و مثبت های کاذب در همه کلاس ها و سپس محاسبه دقت از روی این مقادیر جمع شده محاسبه می شود.

مراجع