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

یادگیری ماشین
پاسخ مینی پروژه دوم

نام و نام خانوادگی: سیدمحمد حسینی
شمارهٔ دانشجویی: ۹۹۰۱۳۹۹
تاریخ: مهرماه ۱۴۰۲

فهرست مطالب

۱ سوال ۱ ۵
۱.۱ بخش اول ۵
۲.۱ بخش دوم ۶
۳.۱ بخش سوم ۶
۱.۳.۱ نتایج MLP با تابع فعال‌ساز ReLU ۱۱
۲.۳.۱ نتایج MLP با تابع فعال‌ساز ELU ۱۱
۳.۳.۱ نتایج مدل بر مبنای mcculloch-pitts نرون ۱۳

۲ سوال ۲ ۱۵
۱.۲ بخش اول ۱۵
۱.۱.۲ پیش‌پردازش ۱۵
۲.۲ بخش دوم ۱۸
۱.۲.۲ نتایج ۲۱
۳.۲ بخش سوم ۲۲
۱.۳.۲ نتایج ۲۳
۴.۲ بخش چهارم ۲۴
۱.۴.۲ نتایج ۲۵

۳ سوال ۳ ۲۵
۱.۳ بخش اول ۲۷
۲.۳ بخش دوم ۲۸
۳.۳ بخش سوم ۳۵

۴ سوال ۴ ۳۸

فهرست تصاویر

۲ نمودارهای متریک و تابع هزینه حین آموزش ۱۱
۵ نمودارهای متریک و تابع هزینه حین آموزش ۱۲
۸ mcculloch-pitts و مرز تصمیم مربوط به مدل ۱۴
۱۰ نتایج آموزش مدل MLP ۲۱
۱۱ Matrix Confiusion ۲۲
۱۲ تابع هزینه آموزش مدل MLP با بهینه ساز و تابع هزینه جدید ۲۳
۱۳ متریک‌های آموزش مدل MLP با بهینه ساز و تابع هزینه جدید ۲۳
۱۴ ماتریس درهم‌ریختگی روی داده‌های آزمون ۲۴
۱۵ cross-entropy fold ۵ . . . . . ۲۵
۱۶ نمودار توزیع داده ۲۶
۱۷ . . . . . ۲۸
۱۸ Drug over Sex ۲۹
۲۰ Graph Hyper-parameters ۳۲
۲۱ Tree Best ۳۳
۲۳ . . . . . ۳۵
۲۴ نحوه عملکرد و آموزش مدل AdaBoost [۲] ۳۶

فهرست جداول

۱ Data Sample ۲۷
۲ Data Sample ۳۸
۳ Data Test ۴۰

فهرست برنامه‌ها

۱ generation data ۶
۲ Configuration ۸
۳ حلقه آموزش ۹
۴ حلقه آموزش ۱۳
۵ Window Sliding ۱۵
۶ Selection Feature and Extraction Feature ۱۶
۷ Split Test Validatiaon Train ۱۷
۸ One-hot and Normalization ۱۸
۹ Configuration and Model ۱۸
۱۰ Loop Train ۱۹
۱۱ Selection Data ۲۷
۱۲ Tree Decision Train ۲۷
۱۳ search Grid ۳۰
۱۴ alpha best ۳۴
۱۵ Training Forest Random ۳۶
۱۶ alpha best ۳۸

۱ سوال ۱

۱.۱ بخش اول

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

پاسخ:
توابع فعال‌ساز Relu و Sigmoid جزو پرکاربردترین اجزاء در یک مدل یادگیری ماشین هستند که به‌منظور اضافه‌کردن خاصیت غیرخطی به مدل مورداستفاده قرار می‌گیرند. به‌منظور تحلیل اثر این دو تابع فعال‌ساز بهتر است ابتدا خواص هرکدام را به‌صورت مجزا بررسی کنیم.

- تابع فعال‌سازی Sigmoid:
تابع سیگموید مقداری بین ۰ و ۱ خروجی می‌دهد که می‌تواند به‌عنوان احتمال تفسیر شود. این تابع فعال‌سازی اصولاً به‌عنوان فعال‌ساز در آخرین لایه مدل‌های طبقه‌بندی استفاده می‌شود؛ زیرا تفسیر احتمالی مستقیمی از خروجی ارائه می‌دهد. از دیگر خواص تابع سیگموید مشتق‌پذیری آن است که از لحاظ برنامه‌نویسی کار را بسیار ساده می‌کند؛ زیرا همان‌طور که در معادله ۱ مشاهده می‌کنید مشتق تابع سیگموید رابطه‌ای است متشکل از خود تابع سیگموید.

$$\frac{d}{dx}\sigma(x) = \sigma(x) \cdot (1 - \sigma(x)) \tag{۱}$$

- تابع فعال‌سازی ReLU:
تابع ReLU ساده‌ترین تابع فعال‌سازی غیرخطی است که مورداستفاده قرار می‌گیرد، این تابع به‌ازای مقادیر مثبت تابع همانی است و به‌ازای مقادیر منفی عدد ۰ را به‌عنوان خروجی در نظر می‌گیرد که این مسئله به‌سادگی محاسبات هنگام Back propagation بسیار کمک می‌کند. مشتق این تابع به‌ازای مقادیر مثبت برابر ۱ و به‌ازای مقادیر منفی برابر ۰ است.

حال باتوجه‌به توضیحات فوق می‌توانیم یک شبکه که دولایه انتهایی آن متشکل از یک تابع ReLU و Sigmoid است را بررسی کنیم. در این شبکه اطلاعات به‌دست‌آمده از backbone شبکه به لایه یکی مانده به آخر ارسال می‌شود که با انجام یک عملیات خطی این اطلاعات به محیط دیگری تصویر می‌شوند که می‌توانند مقادیر مثبت و منفی را با هر اندازه‌ای داشته باشند، این مسئله به مقادیر موجود در Wight و Bias نورون بستگی دارد. حال خروجی نرون‌ها به لایه فعال‌ساز ارسال می‌شوند که در نتیجه آن مقادیر مثبت تبدیل خطی نگه داشته می‌شوند و مقادیر منفی به عدد ۰ تصویر می‌شوند. باید توجه داشت که فعال‌ساز ReLU به تعبیری همانند یک ماژول Attention عمل می‌کند و در فرایند آموزش شبکه Wight و Bias نرون‌ها را به سمتی متمایل می‌کند که در صورت پیدانکردن Feature معنی‌دار از ورودی خود، یک عدد منفی تولید کند. ایراد فعال‌ساز ReLU این است که اگر لایه‌های پشت‌هم در شبکه، همگی مقدار مثبت تولید کنند، تمام تبدیل‌های خطی موجود در این لایه‌ها می‌توانند با یک تبدیل خطی شبیه‌سازی شوند که این مسئله باعث کاهش پیچیدگی مدل ما خواهد شد؛ بنابراین استفاده

از روش‌های رگولاریزیشن و Batch Normalization در کنار توابع فعال‌سازی که به‌صورت پیوسته رفتار غیرخطی دارند موجب رفع این مشکل خواهد شد. در ادامه این مسئله خروجی لایه یکی مانده به آخر وارد لایه تصمیم‌گیری خواهد شد؛ ورودی این لایه تماماً اعداد مثبت است و تبدیل خطی برای اینکه بتواند کلاس مثبت را از منفی جدا کند باید به‌گونه‌ای تنظیم شود که بتواند ورودی‌های تماماً مثبت را به یک عدد مثبت یا منفی تبدیل کند؛ زیرا تابع سیگموید در نقطه ۰ مقدار ۰.۵ را دارد و اگر تبدیل خطی نتواند اعداد مثبت و منفی را از ورودی‌های تماماً مثبت استخراج کند، تابع سیگموید توانایی ایجاد خروجی مناسب را ندارد.

۲.۱ بخش دوم

سوال:

یک جایگزین برای ReLU تابع ELU می‌باشد . ضمن محاسبه گرادیان آن، حداقل یک مزیت آن را مطرح کنید

پاسخ:

ابتدا مشتق این تابع را محاسبه می‌کنیم:

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

$$\frac{d}{dx}ELU(x) = \begin{cases} 1 & \text{if} x > 0 \\ \alpha e^x & \text{if} x \leq 0 \end{cases} \tag{۲}$$

همانطور که در معادله ۲ مشخص است مشتق این تابع برای مقادیر مثبت با تابع ReLU تفاوتی ندارد اما برای مقادیر منفی برابر e^x می‌باشد. همانطور که در قسمت قبل مطرح شد تابع ReLU می‌تواند موجب این شود که در عمل تعداد لایه‌های شبکه کاهش پیدا کند اما با استفاده از تابع فعال‌ساز ELU و استفاده از تکنیک‌های مناسب، می‌توانیم به این مشکل غلبه کنیم. علاوه‌بر این تابع ELU در مشتق خود ناپیوستگی ندارد و بر خلاف ReLU مقدار آن به ازای مقادیر منفی به آرامی برابر $-\alpha$ می‌شود.[۳]

۳.۱ بخش سوم

در ابتدا با استفاده از توزیع یکنواخت تعداد ۲۰۰۰ داده ایجاد می‌کنیم و برچسب نقاطی که داخل مثلث مورد نظر هستند را به مقدار ۱ و برچسب بقیه نقاط را ۰ می‌کنیم. این عملات توسط کد زیر انجام گرفته است.

```python

def point_in_triangle(point, v1, v2, v3):
    """Check if point (px, py) is inside the triangle with vertices v1, v2, v3."""
    # Unpack vertices
    x1, y1 = v1
    x2, y2 = v2
    x3, y3 = v3
    px, py = point

```

```
    # Vectors
    v0 = (x3 - x1, y3 - y1)
    v1 = (x2 - x1, y2 - y1)
    v2 = (px - x1, py - y1)

    # Dot products
    dot00 = np.dot(v0, v0)
    dot01 = np.dot(v0, v1)
    dot02 = np.dot(v0, v2)
    dot11 = np.dot(v1, v1)
    dot12 = np.dot(v1, v2)

    # Barycentric coordinates
    invDenom = 1 / (dot00 * dot11 - dot01 * dot01)
    u = (dot11 * dot02 - dot01 * dot12) * invDenom
    v = (dot00 * dot12 - dot01 * dot02) * invDenom

    # Check if point is in triangle
    return (u >= 0) and (v >= 0) and (u + v < 1)

# Triangle vertices
v1 = (1, 0)
v2 = (2, 2)
v3 = (3, 0)

# Generate random points
np.random.seed(53)

x_coords = np.random.uniform(0, 4, 2000)
y_coords = np.random.uniform(-1, 3, 2000)
x_train = np.column_stack((x_coords, y_coords))

x_coords = np.random.uniform(0, 4, 500)
y_coords = np.random.uniform(-1, 3, 500)
x_test = np.column_stack((x_coords, y_coords))

# Label points based on whether they are inside the triangle
y_train = np.array([point_in_triangle(pt, v1, v2, v3) for pt in x_train]).astype(int)
y_test = np.array([point_in_triangle(pt, v1, v2, v3) for pt in x_test]).astype(int)
```

Code : ۱ data generation

کد فوق به منظور ایجاد ۲۰۰۰ نقطه و بررسی اینکه هر نقطه درون مختصات مثلث قرار دارد یا خیر نوشته شده است. تابع `point_in_triangle` بررسی می‌کند که آیا یک نقطه داخل مثلثی با رئوس داده شده قرار دارد.

مختصات رئوس مثلث و نقطه مورد نظر به تابع داده می‌شودو سپس نسبت مختصات نقطه به مثلث مشخص می‌شود. نقاط تصادفی برای مجموعه‌های آموزشی و تست تولید و با استفاده از تابع مذکور برچسب‌گذاری می‌شوند. همانطور که در شکل ۱آ نمایش داده شده است، دو مجموعه داده به منظور آموزش و ارزیابی مدل تولید شده است.

(آ) مجموعه داده تولید شده

در ادامه یک مدل MLP روی این مجموعه داده آموزش دیده است و نتایج آن به همراه Decision boundary آن نمایش داده شده است. مدل اول MLP با استفاده از توابع ReLU ایجاد شده است که جزییات آن به شرح زیر است:

```
MLP(
(layers): Sequential(
(0): Linear(in_features=2, out_features=8, bias=True)
(1): ReLU()
(2): Linear(in_features=8, out_features=64, bias=True)
(3): ReLU()
(4): Linear(in_features=64, out_features=8, bias=True)
(5): ReLU()
(6): Linear(in_features=8, out_features=1, bias=True)
(7): Sigmoid()
)
)
```

در ادامه مدل فوق با config زیر به میزان ۱۵۰ Epoch آموزش داده شده است.

```
device = "cuda" if torch.cuda.is_available else "cpu"
model = MLP(input_size=2, hidden_size1=8, hidden_size2=64, hidden_size3=8, output_size=1).to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.BCELoss()  # For binary classification
# DataLoader
train_loader = DataLoader(TensorDataset(x_train_tensor, y_train_tensor), batch_size=128, shuffle=True)
test_loader = DataLoader(TensorDataset(x_test_tensor, y_test_tensor), batch_size=512, shuffle=False)
```

Code :۲ Configuration

کد زیر به منظور اجرای حلقه آموزش نوشته شده است. در این کد، چندین متغیر برای نگه‌داری اطلاعات مانند تاریخچه‌ی خطاها و معیارهای متریکی مورد استفاده قرار گرفته است. سپس یک حلقه تکرار برای ایپاک‌های مختلف اجرا می‌شود. در هر ایپاک، داده‌های آموزشی بارگذاری شده و مدل در حالت آموزش قرار می‌گیرد. سپس خطا محاسبه می‌شود و بهینه‌سازی می‌شود. معیارهای متریکی مورد نظر نیز برای داده‌های آموزشی توسط توابعی که مستقلا پیاده‌سازی شده‌اند محاسبه می‌شوند. سپس مدل به حالت ارزیابی در آورده می‌شود و برای داده‌های تست خطا و معیارهای متریکی محاسبه می‌شود. در انتهای آموزش، این اطلاعات برای تحلیل و نمایش خروجی‌های آموزش به دست می‌آید.

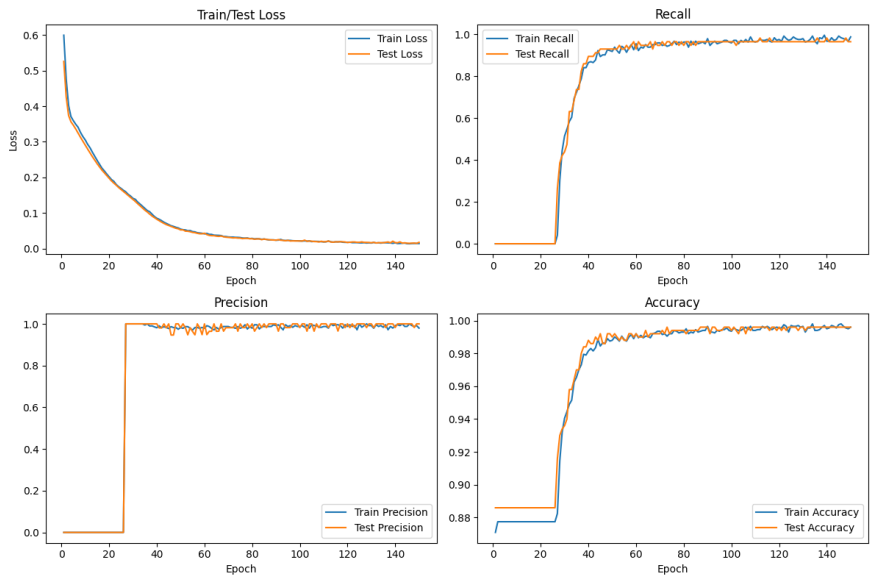
```
num_epochs = 150
train_loss_hist = []
test_loss_hist = []
train_metrics = []
test_metrics = []

for epoch in range(num_epochs):
    loop = tqdm(train_loader)
    model.train()
    train_loss = 0.0
    train_TP, train_FP, train_TN, train_FN = 0, 0, 0, 0

    print("train")
    for inputs, labels in loop:
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = model(inputs)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

        TP, FP, TN, FN = calculate_metrics(outputs, labels)
        train_TP += TP
        train_FP += FP
        train_TN += TN
        train_FN += FN

        loop.set_postfix(
            epoch=epoch+1,
            total_loss=train_loss / len(train_loader),
        )

```

```
    train_metrics.append((train_TP, train_FP, train_TN, train_FN))
    train_loss_hist.append(train_loss / len(train_loader))

    model.eval()
    torch.cuda.empty_cache()
    test_loss = 0.0
    test_TP, test_FP, test_TN, test_FN = 0, 0, 0, 0
    print("Test:")

    with torch.no_grad():
        loop = tqdm(test_loader)
        for inputs, labels in loop:
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)

            test_loss += loss.item()

            TP, FP, TN, FN = calculate_metrics(outputs, labels)
            test_TP += TP
            test_FP += FP
            test_TN += TN
            test_FN += FN

            loop.set_postfix(
                loss=loss.item(),
                total_loss=test_loss / len(test_loader),
            )
    test_metrics.append((test_TP, test_FP, test_TN, test_FN))
    test_loss_hist.append(test_loss / len(test_loader))

```

آموزش حلقه ۳: Code

۱.۳.۱ نتایج MLP با تابع فعال‌ساز ReLU

در شکل ۲ نتایج مربوط به این آموزش نشان داده شده است. همانطور که مشخص است فرآیند آموزش برای مدل MLP به درستی انجام شده و مدل روی دیتاست آموزش over fit نشده است این درحالی است که میزان متریک‌ها برای هر دو دیتاست به ۱ بسیار نزدیک شده است.



شکل ۲: نمودارهای متریک و تابع هزینه حین آموزش

در ؟؟ نتیجه عملکرد مدل روی دیتای ارزیابی قابل ملاحضه است و همانطور که مشخص است تنها ۲ نقطه در راس بالایی مثلث اشتباه طبقه بندی شده‌اند این درحالی است که هیچ کلاسی که متعلق به داخل دایره بوده، به اشتباه به عنوان یک کلاس در خارج از دایره تشخیص داده نشده است، به عبارت دیگر False Negative برابر صفر است و Precision برابر ۱ است.

در انتها در ؟؟ مرز تصمیم مدل مشخص است که با توجه با این نمودار مرز حوالی راس پایین سمت راست مثلث از دقت کافی برخوردار نیست.

(آ) نتایج مدل MLP و مرز تصمیم مربوط به مدل

در شکل ۴آماتریس درهم‌ریختگی به صورت نرمال و غیرنرمال نشان داده شده است که نشان می‌دهد مدل تنها دو نقطه را که مطعلق به داخل مثل بوده است به اشتباه به عنوان کلاس خارج از مثلث تشخیص داده است.

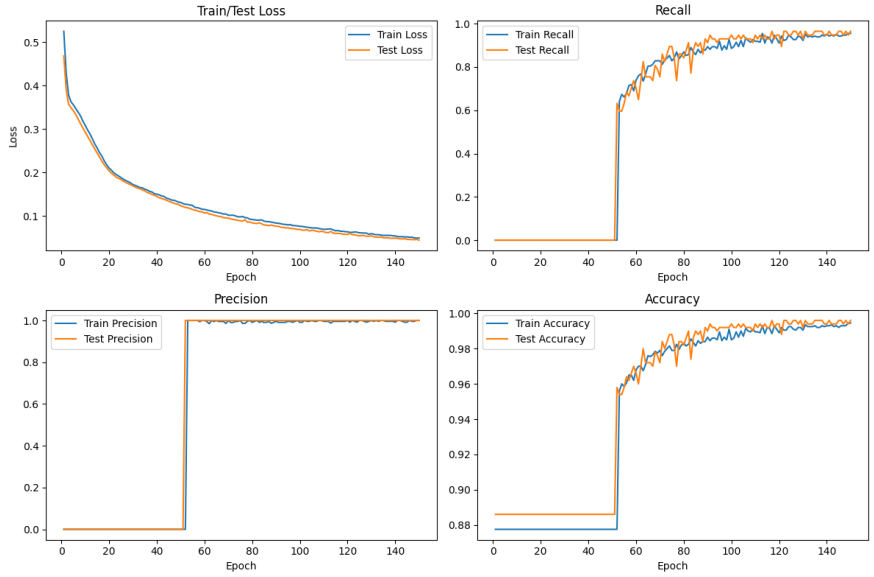
(آ) Matrix Confusion

۲.۳.۱ نتایج MLP با تابع فعال‌ساز ELU

این مدل با استفاده از تابع ELU ساخته شده است و جزییات آن به شرح زیر می‌باشد:

```
MLP(
  (layers): Sequential(
    (0): Linear(in_features=2, out_features=8, bias=True)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=8, out_features=64, bias=True)
    (3): ELU(alpha=1.0)
    (4): Linear(in_features=64, out_features=8, bias=True)
    (5): ELU(alpha=1.0)
    (6): Linear(in_features=8, out_features=1, bias=True)
    (7): Sigmoid()
  )
)
```

در شکل ۵ نتایج مربوط به این آموزش نشان داده شده است. همانطور که مشخص است فرآیند آموزش برای مدل MLP با تابع فعال‌ساز ELUبه درستی انجام شده و مدل روی دیتاست آموزش over fit نشده است این درحالی است که شیب نمودار هزینه همچنان کاهشی می‌باشد.در ادامه میزان متریک‌ها برای هر دو دیتاست به ۱ بسیار نزدیک شده است.



شکل ۵: نمودارهای متریک و تابع هزینه حین آموزش

در ؟؟ نتیجه عملکرد مدل روی دیتای ارزیابی قابل ملاحضه است و همانطور که مشخص است تنها ۲ نقطه در راس بالایی مثلث اشتباه طبقه بندی شده‌اند این درحالی است که هیچ کلاسی که متعلق به داخل دایره بوده، به اشتباه به عنوان یک کلاس در خارج از دایره تشخیص داده نشده است، به عبارت دیگر False Negative برابر صفر است و Precision برابر ۱ است.

در ؟؟ مرز تصمیم مدل مشخص است که با توجه به این نمودار و مقایسه آن با ؟؟، مشخص است که نتایج

بدست آمده از این مدل توانایی تعمیم‌پذیری بیشتری دارند و هر سه راس این مثلث به یک شکل هستند.

(آ) نتایج مدل MLP و مرز تصمیم مربوط به مدل

در شکل ۷آماتریس درهم‌ریختگی به صورت نرمال و غیرنرمال نشان داده شده است که نشان می‌دهد مدل تنها دو نقطه را که مطعلق به داخل مثل بوده است به اشتباه به عنوان کلاس خارج از مثلث تشخیص داده است.

Matrix Confusion (آ)

۳.۳.۱ نتایج مدل بر مبنای mcculloch-pitts نرون

مدل‌های مبتنی بر نرون‌های mcculloch-pitts به ما این امکان را می‌دهند که با استفاده از دانش انسانی، بهترین پاسخ را برای مسئله خود بدست بیاوریم. راه حلی که می‌توانیم نقاط داخل مثلث را شناسایی کنیم حاصل ترکیب AND ۳ گزاره متفاوت است؛ هر یک از این گذاره‌ها نشان‌دهنده این است که نقطه مد نظر ما نسبت به خط گذرنده از هر ضلع مثلث چه وضعیتی دارد. بدین منظور باید سه معادله خط بدست آوریم و با قرار دادن نقاط اطراف خط مشخص کنیم که تغییر علامت چه زمانی رخ می‌دهد و با چه ترکیب منطقی از این تغییر علامت‌ها می‌توانیم مثلث را پیدا کنیم.

در شکل ۵ نتایج مربوط به این آموزش نشان داده شده است. همانطور که مشخص است فرآیند آموزش برای مدل MLP با تابع فعال‌ساز ELUبه درستی انجام شده و مدل روی دیتاست آموزش over fit نشده است این درحالی است که شیب نمودار هزینه همچنان کاهشی می‌باشد.در ادامه میزان متریک‌ها برای هر دو دیتاست به ۱ بسیار نزدیک شده است. روش محاسبه هر یک از این معادلات خط با قرار دادن یک جفت از راس‌های مثلث در معادله زیر می‌باشد:

$$y = w(x - x_1) + y_1, \quad w = \frac{y_1 - y_2}{x_1 - x_2} \tag{۳}$$

با استفاده از رابطه فوق سه جفت مقدار برای هر معادله خط بدست آمد که این مقادیر عبارتند از $(-2,1)$، $(2,1)$ و $(0,1)$ که سه ضلع مثلث را تشکیل می‌دهند. در ادامه یک نرون لازم است تا عملیات منطقی AND را انجام دهد. با جایگذاری نقاط مختلف از صفحه در معادلات خط بدست آمده و مقایسه آنها رابطه منطقی زیر بدست می‌آید:

$$l_1' + l_2' + l_3$$

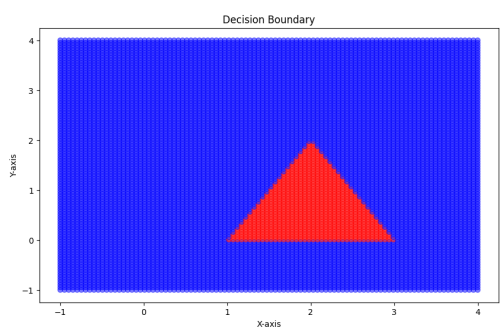
مدل مرتبط با روابط فوق به شکل زیر در پایتون قابل پیاده‌سازی می‌باشد:

```
#define muculloch pitts
class McCulloch_Pitts_neuron():

  def __init__(self , weights ,bias, threshold):
    self.weights = np.array(weights).reshape(-1, 1)     #define weights
    self.threshold = threshold    #define threshold
```

```
        self.bias = np.array(bias)

      def model(self , x):
        #define model with threshold
        return (x.T @ self.weights + self.bias >= self.threshold).astype(int)

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

        z1 = neur1.model(np.array(x))
        z2 = neur2.model(np.array(x))
        z3 = neur3.model(np.array(x))
        z4 = np.squeeze(np.array([1-z1, 1-z2, z3]), axis=-1)
        z4 = neur4.model(z4)

        # 3 bit output
        # return str(z1) + str(z2)
        return z4
```

آموزش حلقه ۴: Code

همانطور که در شکل ؟؟ مشخص است یک مثلث دقیق بدست آمده است و انتظار می‌رود که نتایج ماتریس درهم‌ریختگی آن بهترین وضعیت ممکن باشد که این مسئله در شکل ۹آ قابل مشاهده است.



شکل ۸: mcculloch-pitts و مرز تصمیم مربوط به مدل

Matrix Confusion (آ)

۲ سوال ۲

۱.۲ بخش اول

سوال:

در مورد دیتاست Bearin CWRU و عیوب موجود توضیح دهید.

پاسخ:

دیتاست استفاده شده برای این پروژه، دیتاست Bearing CWRU است. این دیتاست شامل داده‌های ارتعاشی از بلبرینگ‌ها در شرایط مختلف، شامل حالت‌های سالم و معیوب است. در این مینی‌پروژه، ما بر روی داده‌های بخش 12k Drive End Bearing Fault Data تمرکز می‌کنیم. علاوه بر داده‌های استفاده شده در مینی‌پروژه قبلی، ما اکنون دو کلاس جدید OR007@6_1 و B007_1 از بلبرینگ‌های معیوب را اضافه می‌کنیم. این گسترش پیچیدگی و تنوع بیشتری به داده‌ها اضافه می‌کند و امکان تحلیل جامع‌تر و آموزش مدل قوی‌تر را فراهم می‌کند. انواع داده‌های معیوب و توضیحات آن به شرح زیر می‌باشد:

- **سالم:** داده‌هایی که نشان‌دهنده بلبرینگ‌های در عادی خوب هستند.
- **کلاس معیوب ۱ OR007@6_1:** این عیوب در قسمت بیرونی بلبرینگ رخ می‌دهد و می‌تواند ارتعاش و نویز قابل توجهی ایجاد کند که عملکرد کلی ماشین‌آلات را تحت تاثیر قرار می‌دهد.
- **کلاس معیوب ۲ B007_1:** این عیوب در سطح بلبرینگ‌ها رخ می‌دهد و با نامنظمی‌هایی در سطح بلبرینگ‌ها همراه است.
- **کلاس معیوب ۳ (IR007_1):** این عیوب در قسمت داخلی بلبرینگ رخ می‌دهد و باعث ایجاد ناهنجاری در عملکرد کلی بلبرینگ می‌شود.

۱.۱.۲ پیش‌پردازش

کد زیر یک چنجره به طول len_data را روی سری زمانی عبور می‌دهد و داده۷ها را به شکل یک ماتریس $(n, \texttt{len_data})$ ایجاد می‌کند که مقدار n با توجه به تعداد داده موجود در فایل دیتا بستگی دارد. سپس با توجه به متغییر n_samples تعداد نمونه دلخواه را جدا می‌کنیم. به دلیل اینکه در این مسئله تصمیم به آموزش یک شبکه عصبی داریم بیشترین تعداد داده ممکن را از دیتاست استخراج می‌کنیم که برابر ۶۰۰ نمونه پنجره با طول ۲۰۰ است.

```
n_samples = 600
len_data = 200

normal_data_matrix = normal_data[:-(normal_data.shape[0] % len_data)].reshape(-1, len_data)
fault1_data_matrix = fault1_data[:-(fault1_data.shape[0] % len_data)].reshape(-1, len_data)
fault2_data_matrix = fault2_data[:-(fault2_data.shape[0] % len_data)].reshape(-1, len_data)
fault3_data_matrix = fault3_data[:-(fault3_data.shape[0] % len_data)].reshape(-1, len_data)

```

Code :۵ Sliding Window

در ادامه خروجی کد فوق نشان داده شده است.

```
(2419, 200)
(609, 200)
(607, 200)
(612, 200)
```

در ادامه با استفاده از کد زیر featureهای ممکن را از هر پنجره از داده استخراج می‌کنیم و سپس ویژگی‌هایی که مد نظر هست را انتخاب می‌کنیم.

```python
def feature_extraction(data):
    features = {}
    # Standard Deviation
    features['Standard Deviation'] = np.std(data, axis=1)
    # Peak
    features['Peak'] = np.max(np.abs(data), axis=1)
    # Skewness
    features['Skewness'] = scipy.stats.skew(data, axis=1)
    # Kurtosis
    features['Kurtosis'] = scipy.stats.kurtosis(data, axis=1)
    # Crest Factor (peak divided by RMS)
    rms = np.sqrt(np.mean(np.square(data), axis=1))
    features['Crest Factor'] = features['Peak'] / rms
    # Clearance Factor (peak divided by the mean of the square root of the absolute values)
    features['Clearance Factor'] = features['Peak'] / np.mean(np.sqrt(np.abs(data)), axis=1)
    # Peak to Peak
    features['Peak to Peak'] = np.ptp(data, axis=1)
    # Shape Factor (RMS divided by the mean of the absolute values)
    features['Shape Factor'] = rms / np.mean(np.abs(data), axis=1)
    # Impact Factor (peak divided by mean)
    features['Impact Factor'] = features['Peak'] / np.mean(data, axis=1)
    # Square Mean Root (the square root of the mean of the squares)
    features['Square Mean Root'] = rms
    # Mean
    features['Mean'] = np.mean(data, axis=1)
    # Absolute Mean
    features['Absolute Mean'] = np.mean(np.abs(data), axis=1)
    # Root Mean Square
    features['Root Mean Square'] = rms
    # Impulse Factor (peak divided by the absolute mean)
    features['Impulse Factor'] = features['Peak'] / features['Absolute Mean']
    return features
normal_features = pd.DataFrame(feature_extraction(normal_data_matrix))
fault1_features = pd.DataFrame(feature_extraction(fault1_data_matrix))
fault2_features = pd.DataFrame(feature_extraction(fault2_data_matrix))
```

```
fault3_features = pd.DataFrame(feature_extraction(fault3_data_matrix))
normal_features['Label'] = 0
fault1_features['Label'] = 1
fault2_features['Label'] = 2
fault3_features['Label'] = 3

selected_features = ['Standard Deviation', 'Peak', 'Skewness', 'Kurtosis',
                     'Crest Factor', 'Mean', 'Root Mean Square',
                     'Impulse Factor','Square Mean Root','Shape Factor','Label']

main_df = pd.concat([normal_features, fault1_features, fault2_features, fault3_features],
    ignore_index=True)
df = main_df[selected_features]

```

Code :۶ Feature Extraction and Feature Selection

در ادامه تقسیم دیتاست به ۳ زیرمجموعه آموزش، ارزیابی و آزمون انجام شده است که به موجب آن ابتدا به صورت تصادفی shuffle می‌شوند و پس از آن یک بار به صورت عادی و بار بعدی به صورت stratify تقسیم داده انجام می‌شود. با مشخص کردن آرگمان stratify باعث می‌شویم که توزیع داده در زیرمجموعه آزمون و ارزیابی بهم نریزد. تقسیم مجموعه داده با استفاده از آرگمان فوق باعث بهبود نتایج آموزش مدل نسبت به شرایطی که انتخاب هر زیرمجموعه به صورت کاملا تصادفی بوده، شده است.

اضافه کردن مجموعه داده اعتبارسنجی و بررسی تابع هزینه روی این مجموعه داده حین آموزش، موجب آن میشود که از اتفاق افتادن over-fitting جلوگیری شود.

```
df_shuffled = df.sample(frac=1).reset_index(drop=True)

X = df_shuffled.drop('Label', axis=1)
y = df_shuffled['Label']

# X_train_raw, X_test_raw, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state
    =53)
# X_train_raw, X_val_raw, y_train, y_val = train_test_split(X_train_raw, y_train, test_size=0.25,
     random_state=53)
X_train_raw, X_test_raw, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y,
    random_state=53)
X_train_raw, X_val_raw, y_train, y_val = train_test_split(X_train_raw, y_train, test_size=0.25,
    stratify=y_train, random_state=53)

```

Code :۷ Train Validatiaon Test Split

۲.۲ بخش دوم

در ادامه مجموعه داده بدست آمده را نرمال می‌کنیم، باید توجه داشت که مقادیر میانگین و واریانسی که به منظور نرمال کردن مجموعه داده ارزیابی و آزمون در نظر گرفته شده است از مجموعه داده آموزش بدست آمده است. در انتها متناسب با عدد موجود در Label هر داده، یک آرایه به صورت One-hot به آن داده اختصاص داده شده است.

```
X_train, X_train_max, X_train_min = min_max_normalization(X_train_raw)
X_test, _, _ = min_max_normalization(X_test_raw,X_train_max,X_train_min)
X_val, _, _ = min_max_normalization(X_val_raw,X_train_max,X_train_min)

X_train = torch.tensor(X_train.to_numpy(), dtype=torch.float32)
X_val = torch.tensor(X_val.to_numpy(), dtype=torch.float32)
X_test = torch.tensor(X_test.to_numpy(), dtype=torch.float32)

y_train = torch.tensor(y_train.to_numpy(), dtype=torch.long)
y_val = torch.tensor(y_val.to_numpy(), dtype=torch.long)
y_test = torch.tensor(y_test.to_numpy(), dtype=torch.long)

# Perform one-hot encoding
y_train = torch.nn.functional.one_hot(y_train, num_classes=4).float()
y_val = torch.nn.functional.one_hot(y_val, num_classes=4).float()
y_test = torch.nn.functional.one_hot(y_test, num_classes=4).float()

```

Code :۸ Normalization and One-hot

در ادامه یک مدل MLP با سه لایه پنهان ایجاد می‌کنیم که شرح تعداد نرون‌های آن در کد زیر آمده است سپس با مقداردهی اولیه مدل، یک شی در محیط برنامه نویسی ایجاد می‌کنیم. پارامتر بعدی که در آموزش این مدل استفاده شده است تابع هزینه `CrossEntropyLoss` است که یک شی از آن با مقادیر پیش‌فرض ایجاد شده است. تابع بهینه ساز این مدل از الگوریتم Adam پیروی میکند که با گام اموزشی اولیه 0.001 شروع به بهینه کردن مدل می‌کند و در ادامه با استفاده از تابع `lr_scheduler` اگر طی ۱۵ Epoch بهبودی در نتایج تابع هزینه روی مجموعه داده ارزیابی رخ ندهد، میزان گام آموزشی ۱.۰ مقدار قبلی خود خواهد شد.

```
class MLP(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(MLP, self).__init__()
        self.hidden1 = nn.Linear(input_dim, 32)
        self.hidden2 = nn.Linear(32, 64)
        self.output = nn.Linear(64, output_dim)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.relu(self.hidden1(x))
```

```
        x = self.relu(self.hidden2(x))
        x = self.output(x)
        x = self.softmax(x)
        return x

# Initialize the model, loss function, and optimizer
model = MLP(input_dim=X_train.shape[1], output_dim=len(set(y)))
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=15,
    verbose=True)

```

Code :۹ Model and Configuration

در انتها مدل با استفاده از کد زیر آموزش داده می‌شود؛ باید توجه داشته که کد زیر با بررسی تغییرات تابع هزینه دستاست ارزیابی، میزان بهبود آن را بررسی می‌کند و اگر به ازای ۳۰ Epoch بهبود محسوس رخ ندهد فرایند آموزش متوقف می‌شود.

```
# Training loop with early stopping
num_epochs = 500
patience = 30
best_val_loss = float('inf')
epochs_without_improvement = 0
Best_model = None

# Metrics storage
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

for epoch in range(num_epochs):
    model.train()
    train_loss = 0.0
    train_correct = 0
    total_train = 0

    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

        _, preds = torch.max(outputs, 1)
```

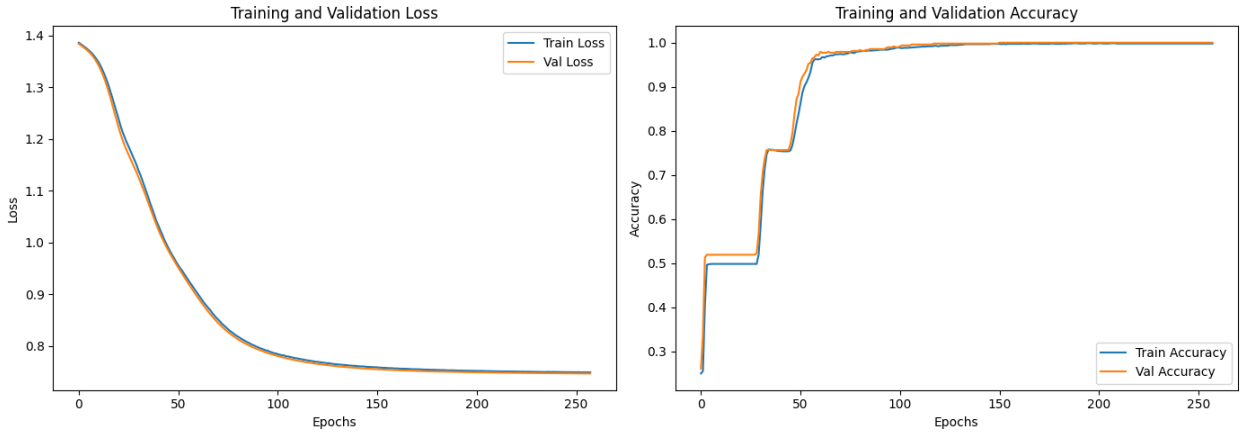
```
        _, targets = torch.max(labels,1)
        train_correct += (preds == targets).sum().item()
        total_train += labels.size(0)

    train_loss /= len(train_loader)
    train_accuracy = train_correct / total_train
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)

    model.eval()
    val_loss = 0.0
    val_correct = 0
    total_val = 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()

            _, preds = torch.max(outputs, 1)
            _, targets = torch.max(labels,1)
            val_correct += (preds == targets).sum().item()
            total_val += labels.size(0)

    val_loss /= len(val_loader)
    val_accuracy = val_correct / total_val
    val_losses.append(val_loss)
    val_accuracies.append(val_accuracy)

    print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f},
     Train Accuracy: {train_accuracy:.4f}, Val Accuracy: {val_accuracy:.4f}')

    scheduler.step(val_loss)

    if  best_val_loss - val_loss > 1e-3:
        best_val_loss = val_loss
        torch.save(model.state_dict(), 'best_model.pth')
        epochs_without_improvement = 0
    else:
        epochs_without_improvement += 1

    if epochs_without_improvement >= patience:
        print(f'Early stopping triggered after {epoch+1} epochs.')
        break
```

۷۲

Code :۱۰ Train Loop

۱.۲.۲ نتایج

پس از آموزش مدل با configuration که در بالا مطرح شد نتایج بدست آمده از تابع هزینه و متریک Accuracy در شکل ۱۰ نشان داده شده است. در این آموزش، با گذشتن ۲۵۸ Epoch بهبودی در میزان تابع هزینه دیتاست ارزیابی مشاهده نشده است و فرایند آموزش stop early شده است. همانطور که مشخص است میزان accuracy این مجموعه داده آموزش و ارزیابی در طی فرایند آموزش به حدود یک نزدیک شده است و همانطور که از این نمودار مشخص است، بهینه سازی این مدل در ۲ نقطه به extremum local رسیده است و برای حدود چند Epoch بهبودی در نتایج این متریک مشاهده نمی‌شود.



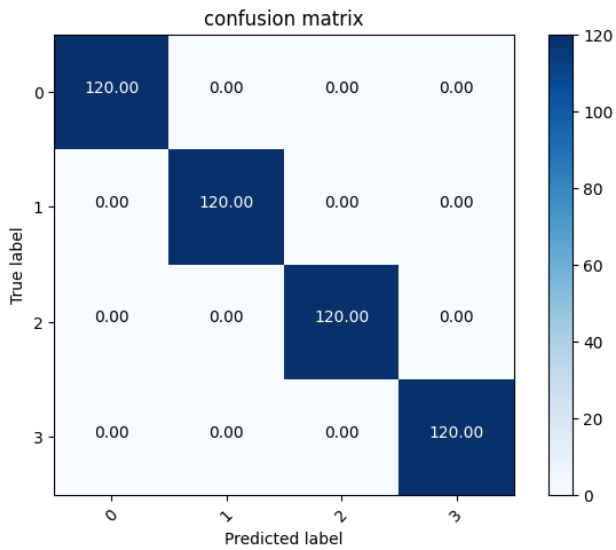
شکل ۱۰: نتایج آموزش مدل MLP

به منظور تحلیل بهتر نتایج بدست آمده از آموزش این مدل متریک‌ها و معیارهای دیگری علاوه بر Accuracy مورد نیاز است زیرا که متریک Accuracy با در نظر گرفتن کلاس‌های نرمالی که به درستی تشخیص داده شده‌اند در فرمول خود، باعث کاهش اثر نمونه‌هایی که به اشتباه تشخیص داده‌شده‌اند می‌شود بنابراین متریک‌های دیگری مورد نیاز است که در ادامه نتایج مدل را برحسب Recall و Precision و Score F-۱ و ماتریس درهم‌ریختگی روی مجموعه داده آزمون بررسی می‌شود.

نتایج آموزش مدل روی داده آزمون در شکل ۱۱ و شکل ۱.۲.۲ نشان دهنده عملکرد بسیار خوب این مدل می‌باشد که تمام نمونه‌ها را به درستی تشخیص داده است.

	precision	recall	f1-score	support
0	00.1	00.1	00.1	359
1	00.1	00.1	00.1	375
2	00.1	00.1	00.1	348

```
           3       00.1      00.1      00.1       358

    accuracy                          00.1      1440
   macro avg       00.1      00.1      00.1      1440
weighted avg       00.1      00.1      00.1      1440
```



شکل ۱۱: Matrix Confiusion

۳.۲ بخش سوم

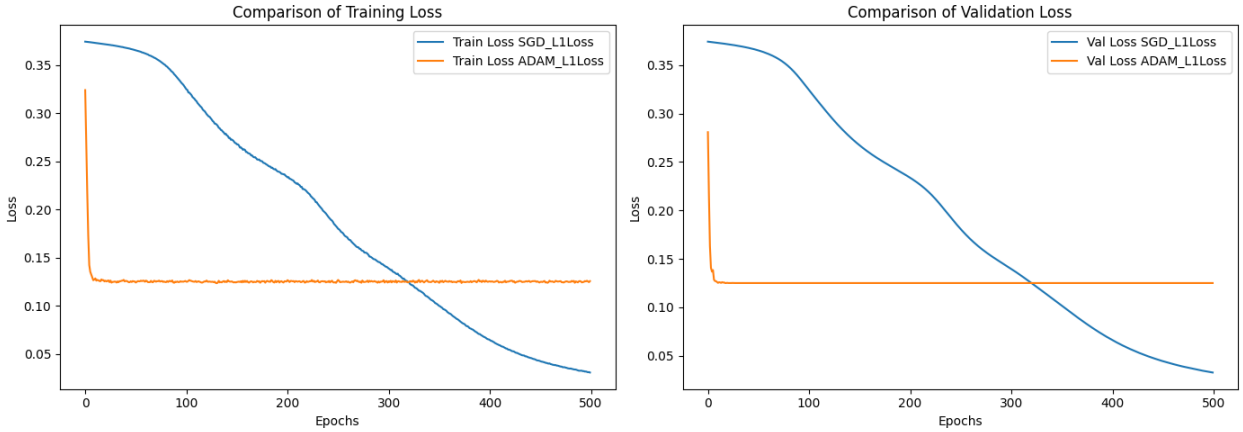
در ادامه با استفاده از کد زیر ۲ مدل جدید با هایconfiguratin جدید ایجاد می‌کنیم که در مدل اول تنها تابع هزینه به L1Loss تبدیل شده و در مدل دوم علاوه بر آن بهینه‌ساز به الگوریتم SGD تغییر یافته است.

```
# Model 1
model = MLP(input_dim=X_train.shape[1], output_dim=len(set(y)))
criterion = nn.L1Loss()
optimizer = optim.Adam(model.parameters(), lr=0.1)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=15,
    verbose=True)

# model 2
model = MLP(input_dim=X_train.shape[1], output_dim=len(set(y)))
criterion = nn.L1Loss()
optimizer = optim.SGD(model.parameters(), lr=0.1)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=15,
    verbose=True)

```

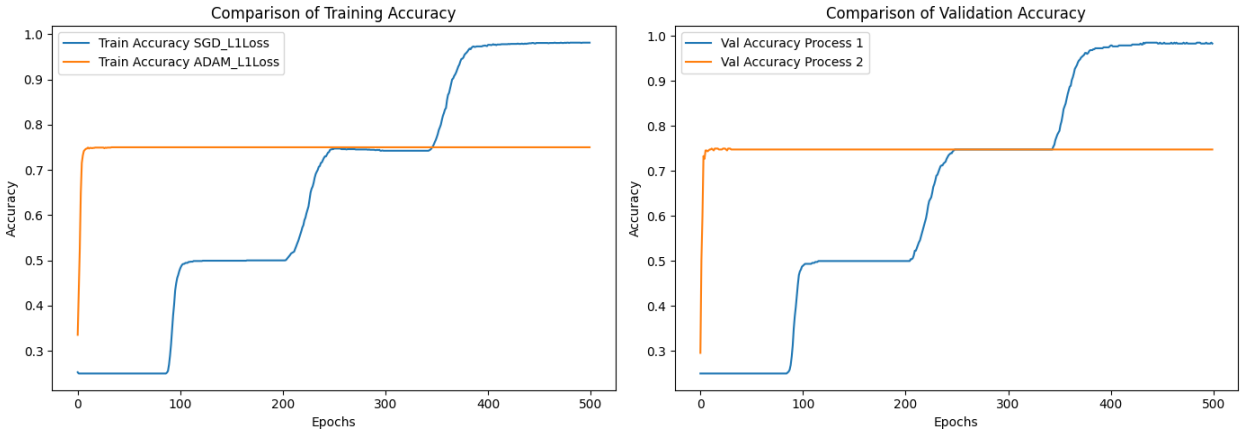
۱.۳.۲ نتایج

با توجه به نمودارهای موجود در شکل ۱۳ و شکل ۱۲ مشخص می‌شود که تغییر تابع هزینه به تنهایی می‌تواند به شدت روی عملکرد مدل تاثیر بگذارد؛ این مسئله با بررسی نمودارهای نارنجی که مربوط به مدل با تابع هزینه `L1Loss` و بهینه ساز `Adam` است مشخص می‌شود. در آموزش این مدل مشخص می‌شود که یک نقطه بهینه محلی پیدا شده است که مدل نتوانسته است از آن خارج شود و مقدار Accuracy برابر ۷۵% شده است که نشان دهنده افت شدید این متریک نسبت به شرایطی است که تابع هزینه از نوع `CrossEntropyLoss` است.

در ادامه با تغییر بهینه‌ساز در فرایند آموزش به `SGD` نشان داده می‌شود که دوباره مدل به نتیجه مطلوب می‌رسد اما باید این نکته را در نظر داشته که بابت رسیدن به این نتیجه نیاز است که ۵۰۰ Epoch طی شود که خیلی بیشتر از تعداد Epoch طی شده در فرایند آموزش نشان داده‌شده در زیرقسمت ۲.۲ می‌باشد. نکته حائز اهمیت در نمودارهای شکل ۱۲ این است که آموزشی که با نمودار آبی نشان داده شده است همچنان می‌تواند آموزش داده شود و داری شیب نزولی هم در نمودار آموزش و هم در نمودار ارزیابی است.

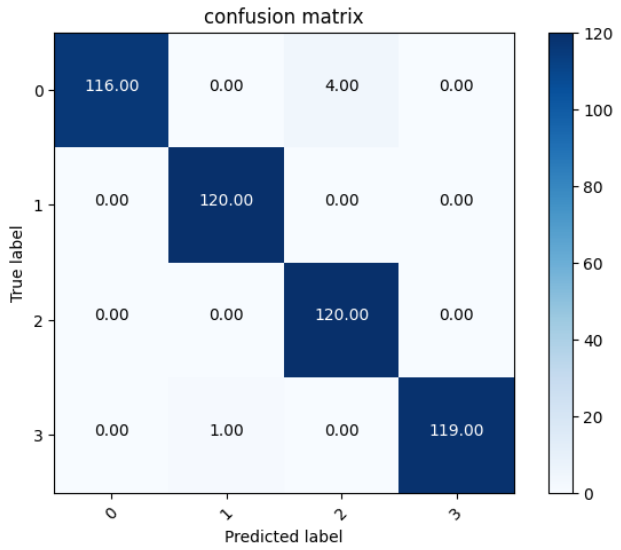


شکل ۱۲: تابع هزینه آموزش مدل MLP با بهینه ساز و تابع هزینه جدید



شکل ۱۳: متریک‌های آموزش مدل MLP با بهینه ساز و تابع هزینه جدید

در ادامه به منظور مقایسه عملکرد مدلی که تابع هزینه و الگوریتم بهینه‌سازی آن عوض شده است با مدل قبلی، ماتریس درهم‌ریختگی را روی مجموعه داده تست محاسبه می‌کنیم که در شکل ۱۴ نمایش آن قابل ملاحظه است. همانطور که از این ماتریس مشخص است، مدلی که تابع هزینه و بهینه ساز آن تغییر کرده است در ۴ نمونه از کلاس سالم را به اشتباه متعلق به کلاس ۲ تشخیص داده است و ۱ نمونه از کلاس ۳ را متعلق به کلاس ۱ تشخیص داده است که در مجموع ۵ خطا دارد؛ این درحالی است که مدل بدست آمده در زیرقسمت ۲.۲ هیچ خطایی روی محموعه داده آزمون نداشته است، بنابراین تغییر تابع هزینه و یا بهینه ساز می‌تواند موجب کاهش عملکرد مدل در این مسئله بشود.



شکل ۱۴: ماتریس درهم‌ریختگی روی داده‌های آزمون

۴.۲ بخش چهارم

Cross-validation K-fold یک روش آماری قوی برای ارزیابی عملکرد و تعمیم‌پذیری یک مدل یادگیری ماشین است. در این تکنیک، مجموعه داده به k بخش مساوی تقسیم می‌شود. مدل بر روی $k-1$ بخش آموزش داده می‌شود و بر روی بخش باقی‌مانده ارزیابی می‌شود. این فرایند k بار تکرار می‌شود، هر بار با استفاده از یک بخش متفاوت برای تست. سپس نتایج میانگین گرفته می‌شود تا یک اندازه‌گیری جامع از عملکرد مدل ارائه شود. این روش تضمین می‌کند که هر داده هم برای آموزش و هم برای ارزیابی استفاده می‌شود و باعث کاهش bias و واریانس در مقایسه با تقسیمات ساده آموزش و ارزیابی می‌شود. Cross-validation K-fold به‌ویژه در مجموعه داده‌های کوچک که حفظ حداکثر داده‌ها برای آموزش اهمیت دارد، مفید است.

Cross-Validation K-Fold Stratified یک نوع تغییر یافته از Cross-validation K-fold است که هدف آن حفظ توزیع متغیر هدف در تمام بخش‌هاست. در مسائل دسته‌بندی، این روش تضمین می‌کند که هر بخش تقریباً همان نسبت کلاس‌ها را دارد که در کل مجموعه داده وجود دارد. این طبقه‌بندی برای مجموعه داده‌های imbalance که برخی کلاس‌ها ممکن است تعداد کمی داشته باشند، بسیار مهم است. با حفظ توزیع کلاس‌ها، Cross-Validation K-Fold Stratified یک تخمین دقیق‌تر از عملکرد مدل، به‌ویژه

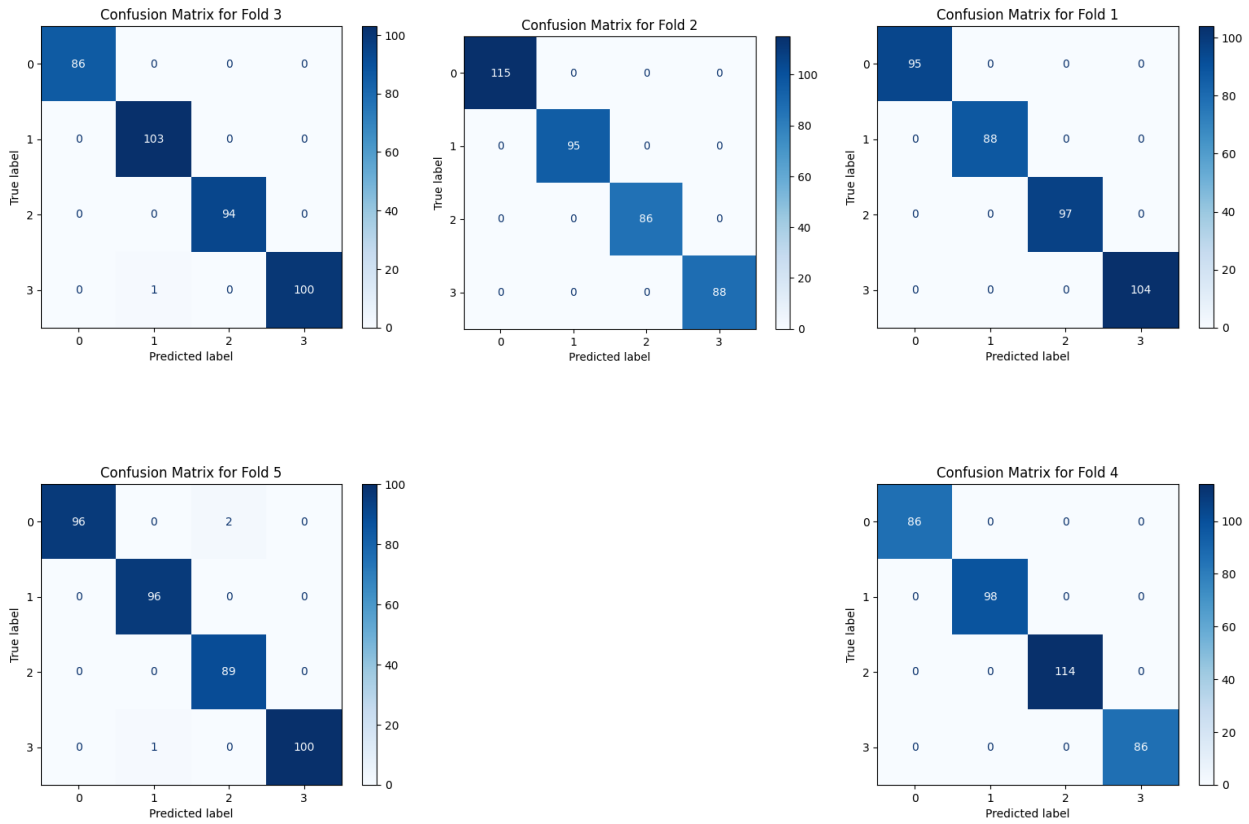
برای کلاس‌های اقلیت ارائه می‌دهد. این روش به جلوگیری از نتایج گمراه‌کننده کمک می‌کند و آن را به انتخابی بهتر برای آموزش مدل‌های یادگیری ماشین تبدیل می‌کند.

با استفاده از کد زیر می‌توانیم مجموعه داده را به k - fold تقسیم کنیم. نکته حائز اهمیت این است که مجموعه داده test به دلیل اینکه قابلیت مقایسه با آموزش‌های قبلی حفظ شود در تمام آموزش‌ها در طی حل سوال ۲ یکسان بوده است.

```
kf = KFold(n_splits=5)
kf_results = []

for fold, (train_index, val_index) in enumerate(kf.split(X_tensor)):
    print(f"Fold {fold+1}")
    X_train, X_val = X_tensor[train_index], X_tensor[val_index]
    y_train, y_val = y_tensor[train_index], y_tensor[val_index]

    train_data = torch.utils.data.TensorDataset(torch.tensor(X_train, dtype=torch.float32), torch
    .tensor(y_train, dtype=torch.float))
    val_data = torch.utils.data.TensorDataset(torch.tensor(X_val, dtype=torch.float32), torch.
    tensor(y_val, dtype=torch.float))

    train_loader = torch.utils.data.DataLoader(train_data, batch_size=512, shuffle=True)
    val_loader = torch.utils.data.DataLoader(val_data, batch_size=2000, shuffle=False)

    model = MLP(input_dim=X_train.shape[1], output_dim=len(set(y))).to(device)
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience
    =15, verbose=True)

```

۱.۴.۲ نتایج

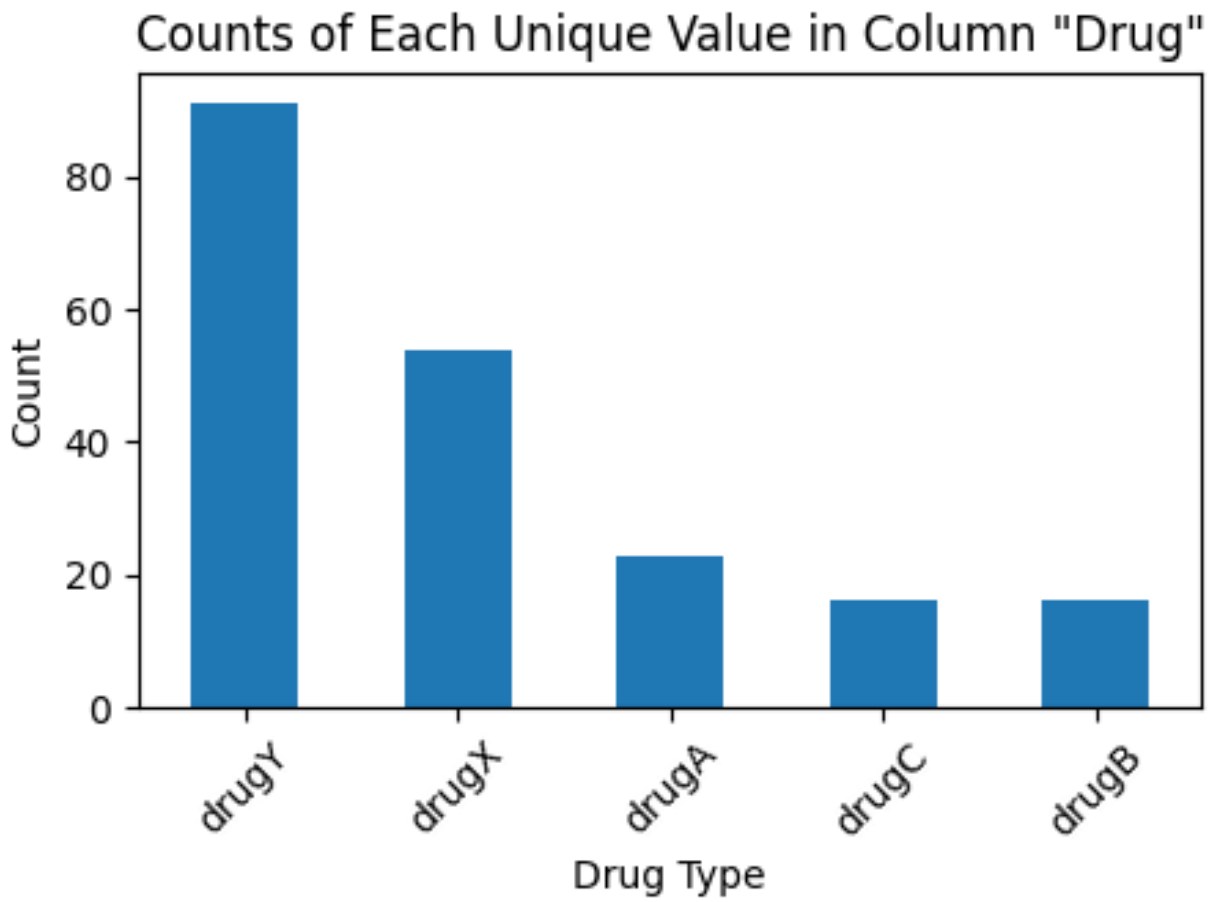
در شکل ۱۵ نمودارهای ماتریس درهم‌ریختگی مشخص شده است، با استفاده از این نمودارها می‌توان متریک‌های Recall و Precision را محاسبه کرد. با توجه به اینکه این نمودارها روی مجموعه داده آزمون حساب شده است، و این مجموعه داده در طول آموزش‌ها یکسان بوده است، می‌توان نتیجه گرفت که در قسمت‌های قبل نتایج صورت تصادفی کسب نشده‌اند و مدل به درستی روی دیتاست آموزش دیده است. اگر پیچیدگی دیتاست بیشتر بود با انجام اینکار ممکن بود نتایج در یک fold به خصوص بهتر از بقیه آموزش‌ها شود.



شکل ۱۵: cross-validation fold ۵

۳ سوال ۳

در ادامه حل این سوال مجموعه داده دارو درنظر گرفته شده است و الگوریتم‌ها به روی این مجموعه داده اجرا شده‌اند. در ادامه تحلیل آماری مجموعه داده دارو انجام شده است. همانطور که در شکل ۱۶ مشخص شده است این دیتاست تا حدی imbalance است و فراوانی داروی x و y بیشتر از باقی داروها می‌باشد اعداد دقیق مربوط به فراوانی این دیتاست در شکل ۳ نشان‌داده‌شده است.



شکل ۱۶: نمودار توزیع داده

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

جدول ۱ نشان‌دهنده چند سطر از مجموعه داده دارو می‌باشد که مقادیر توصیفی ستون‌های Sex، BP و Cholesterol به شرح زیر می‌باشد:

```
Values of Sex column =  ['F' 'M']
Values of BP column =  ['HIGH' 'LOW' 'NORMAL']
Values of Cholesterol column ['HIGH' 'NORMAL']
```

Drug	Na_to_K	Cholesterol	BP	Sex	Age	
drugY	۳۵۵.۲۵	HIGH	HIGH	F	۲۳	۰
drugC	۰۹۳.۱۳	HIGH	LOW	M	۴۷	۱
drugC	۱۱۴.۱۰	HIGH	LOW	M	۴۷	۲
drugX	۷۹۸.۷	HIGH	NORMAL	F	۲۸	۳
drugY	۰۴۳.۱۸	HIGH	LOW	F	۶۱	۴

جدول ۱: Data Sample

۱.۳ بخش اول

سوال:

توضیح دهید که از چه روشی برای انتخاب بخشی از داده استفاده کرده‌اید. آیا روش بهتری برای این کار می‌شناسید؟

سوال:

به منظور پاسخ داده به این سوال ابتدا باید کدی که به وسیله آن دیتاست تقسیم شده است مورد بررسی قرار بگیرد. کد زیر به منظور تقسیم دیتاست موجود به دو زیرمجموعه آموزش و ارزیابی است که در نتیجه آن ۴۰% از این دیتاست با توزیع یکنواخت به مجموعه ارزیابی تخصیص داده می‌شود.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=53)

```

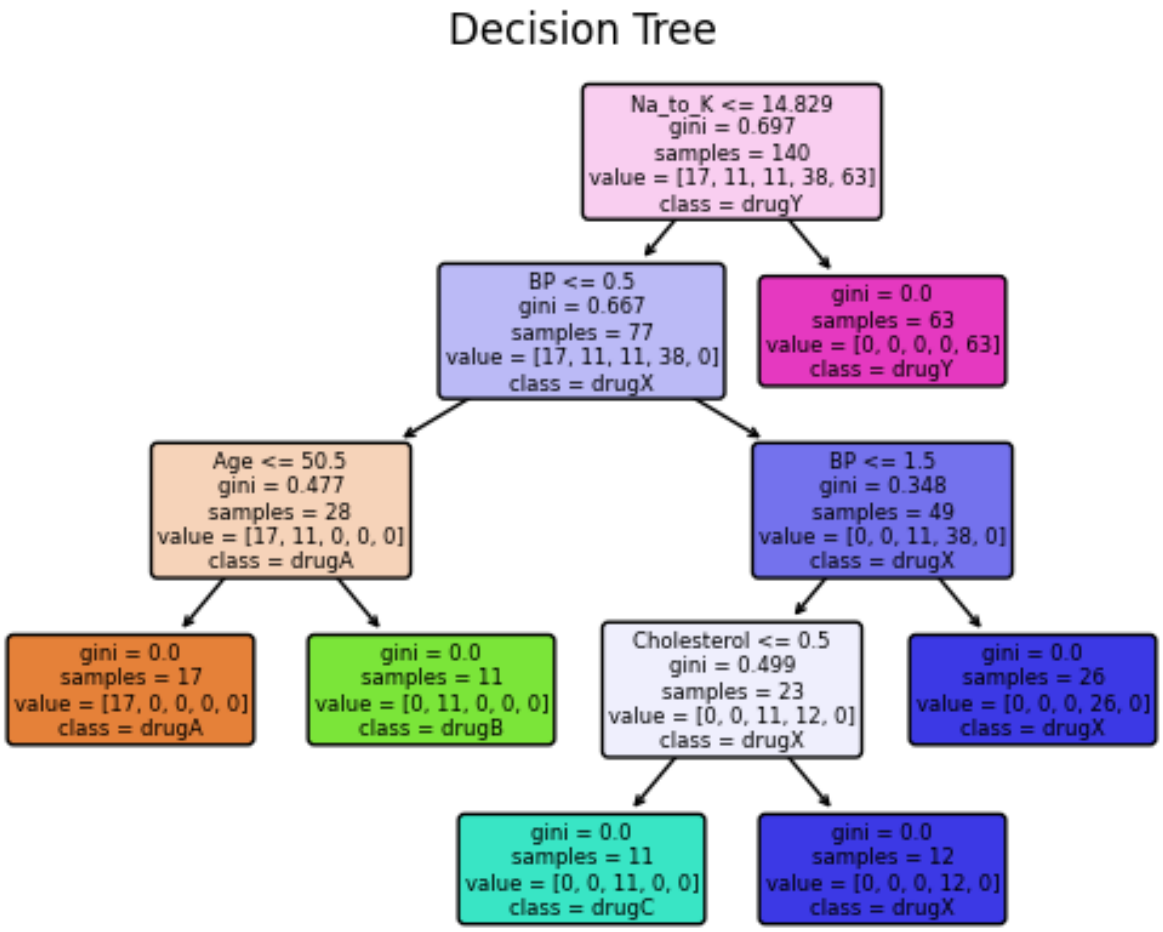
Code :۱۱ Data Selection

نکته حائز اهمیت این است که این دیتاست از تفکیک پذیری بالایی برخوردار است و نیازی نیست که از روش‌های پیشرفته‌تر به منظور تقسیم آن استفاده کنیم اما با توجه به imbalance بودن این دیتاست می‌توان از روش stratify به منظور حفظ توزیع داده‌ها و یا روش K-Fold-cross-validation استفاده کرد تا بهترین نتیجه ممکن را بدست آورد و از عملکرد مدل مطمئن شد.

در ادامه با استفاده از کد زیر و Configuration پیش‌فرض کتابخانه sklearn یک درخت تصمیم ایجاد شده است و روی دیتاست آموزش، آموزش دیده است. نتایج بدست آمده از این آموزش در شکل ۱۷ نشان داده شده است.

```
# Create a Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=53)

# Train the classifier on the training data
clf.fit(X_train, y_train)

plot_tree(clf, filled=True, feature_names=X.columns, class_names=clf.classes_, rounded=True)
plt.title('Decision Tree')
plt.show()
```

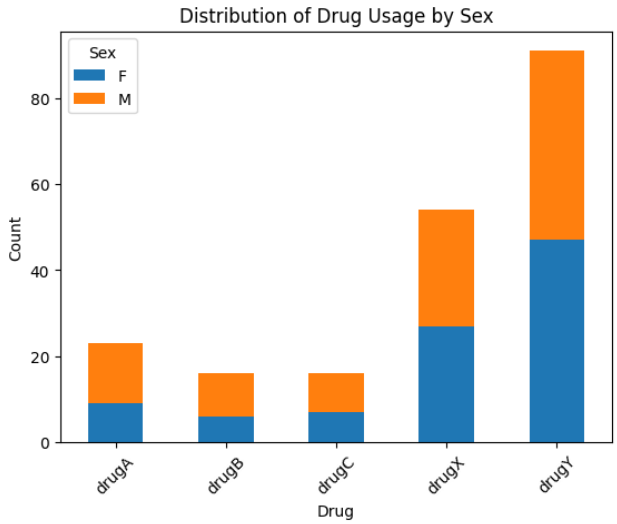
Code :۱۲ Train Decision Tree



شکل ۱۷

۲.۳ بخش دوم

با توجه به شکل ۱۷ می‌توان مشخص کرد که از داده‌های جنسیت به منظور تعیین کلاس هر نمونه استفاده نشده است. به منظور تحلیل این مسئله لازم است تا پراکندگی جنسیت، نسبت به هر نوع دارو رسم شود. این نمودار در شکل ۱۸ رسم شده و است و همانطور که مشخص است میزان مردان نسبت به زنان همواره از entropy بالایی برخوردار بوده که به همین دلیل نیز جنسیت در نوع دارو تاثیر چندانی نگذاشته است، به بیان دیگر این بیماری با جنسیت وابستگی پایینی دارد.



شکل ۱۸: Drug over Sex

در ادامه متریک‌های موردنیاز به منظور بررسی نتایج مدل محاسبه شده است. همانطور که مشخص است برای دو زیرمجموعه آموزش و ارزیابی تمامی متریک‌های و ماتریس درهم‌ریختگی در شکل ۱۹آ مقداری برابر ۱ دارند که این به معنی عملکرد بی‌عیب مدل ما است.

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

       drugA       00.1      00.1      00.1        17
       drugB       00.1      00.1      00.1        11
       drugC       00.1      00.1      00.1        11
       drugX       00.1      00.1      00.1        38
       drugY       00.1      00.1      00.1        63

    accuracy                           00.1       140
   macro avg       00.1      00.1      00.1       140
weighted avg       00.1      00.1      00.1       140

test metrics:

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

```
       drugA      00.1      00.1      00.1         6
       drugB      00.1      00.1      00.1         5
       drugC      00.1      00.1      00.1         5
       drugX      00.1      00.1      00.1        16
       drugY      00.1      00.1      00.1        28

    accuracy                         00.1        60
   macro avg      00.1      00.1      00.1        60
weighted avg      00.1      00.1      00.1        60
```

Matrix Confusion (آ)

حال با استفاده از GridSearchCV هایپرپارامترهای متفاوت را با استفاده از متد ۵-fold-corss-validation بررسی می‌کنیم تا بهترین ساختار مدل را پیدا کنیم. در این جستجو مقادیر max_depth، min_samples_split و min_samples_leaf را در یک بازه متناسب با دیتاستی که در اختیار داریم تغییر می‌دهیم. کد زیر به منظور آموزش مدل‌های مختلف استفاده شده است و در انتها بهترین هایپرپارامترها را خروجی می‌دهد.

```
param_grid = {
    'max_depth': [2, 3, 4, 5, 10, 15 ],
    'min_samples_split':  [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 7, 10]
}
scoring = {
    'precision': make_scorer(precision_score, average='macro'),
    'recall': make_scorer(recall_score, average='macro')
}
# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid,
                           scoring=scoring, refit='precision', return_train_score=True)
grid_search.fit(X_train, y_train)

```

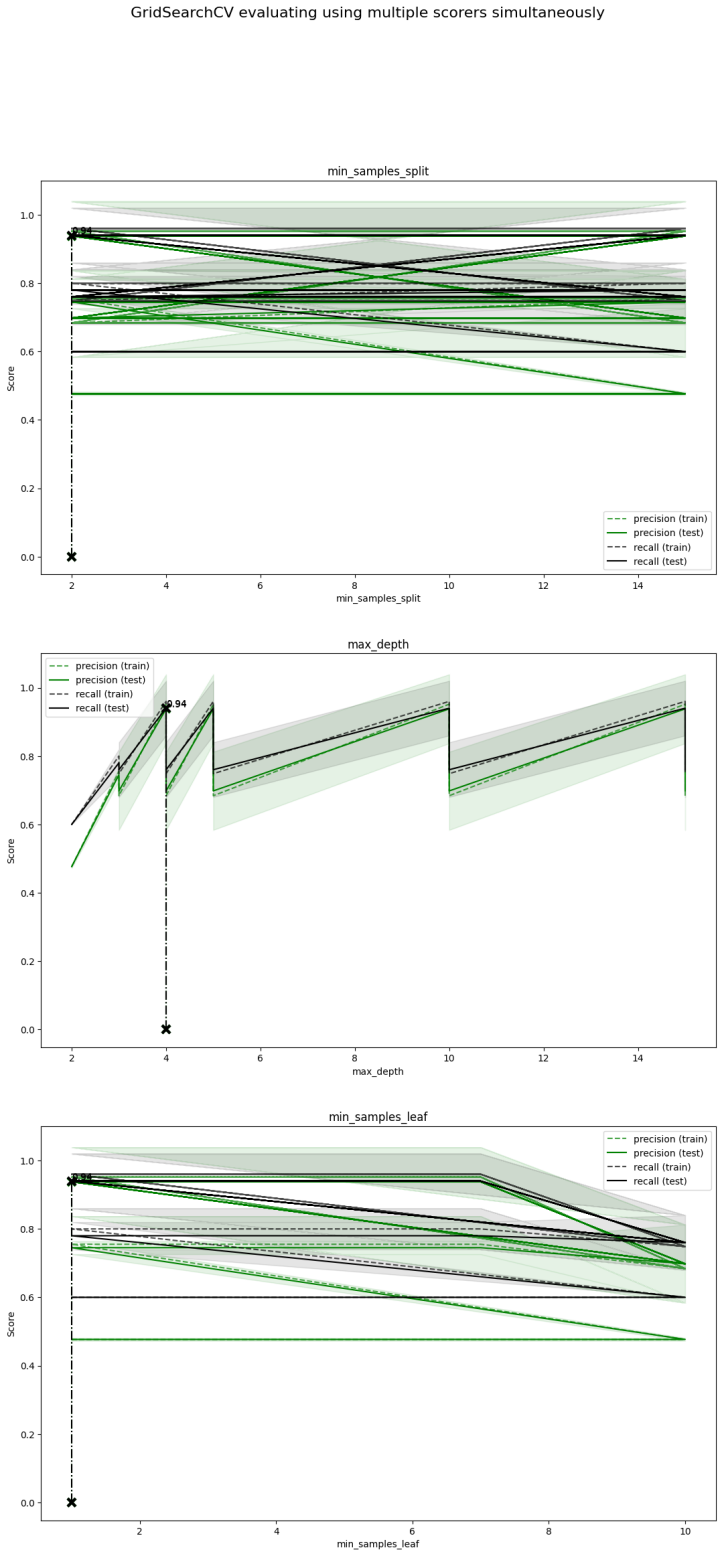
Code :۱۳ Grid search

نتایج بدست آمده از این گروه آموزش‌ها نشان می‌دهد که مدل بهینه هایپرپارامترهای زیر را دارد.

```
Best Parameters: {'max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

نمودارهای موجود در شکل ۲۰ به ازای تغییرات هر هایپرپارامتر کشیده شده است. همانطور که مشخص است، متغییر min_samples_split با افزایش مقدار، گاهی موجب بهبود و گاهی موجب افت عملکرد مدل می‌شود که این بهبود یا افت به مقدار دو متغییر دیگر بستگی دارد بنابراین نمیتوان به صورت کلی نظر قطعی درمورد این متغییر داد. متغییر max_depth یک رفتار تکرار شونده را نشان می‌دهد که به غیر از مقادیر خاص،

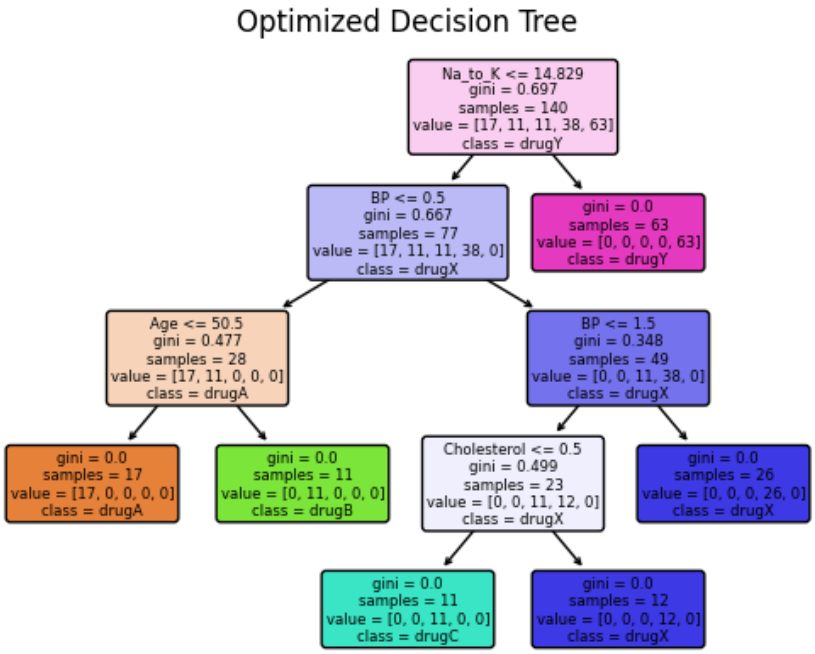
نمودار آن افزایشی می‌باشد بنابراین افزایش مقدار این متغییر می‌تواند به بهبود مدل کمک کند. در انتها متغییر `min_samples_leaf` و نمودار مربوط به آن نشان می‌دهد که با افزایش مقدار این متغییر عملکرد مدل در این مسئله به خصوص کاهش پیدا می‌کند بنابراین کمترین مقدار ممکن را برای این متغییر درنظر می‌گیریم.



شکل ۲۰: Graph Hyper-parameters

حال به سراغ بررسی نتایج بدست آمده از بهترین مدل می‌رویم، همانطور که در شکل ۲۱ مشخص است،

تفاوتی با درخت موجود در شکل ۱۷ نداشته است، از طرف دیگر با توجه به متریک‌ها و ماتریس درهم‌ریختگی در شکل ۲۲آ همچنان عملکرد مدل در بهترین حالت خود می‌باشد و تمام نمونه‌ها را به درستی تشخیص داده است



شکل ۲۱: Tree Best

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

       drugA       00.1      00.1      00.1        17
       drugB       00.1      00.1      00.1        11
       drugC       00.1      00.1      00.1        11
       drugX       00.1      00.1      00.1        38
       drugY       00.1      00.1      00.1        63

    accuracy                           00.1       140
   macro avg       00.1      00.1      00.1       140
weighted avg       00.1      00.1      00.1       140

test metrics:
```

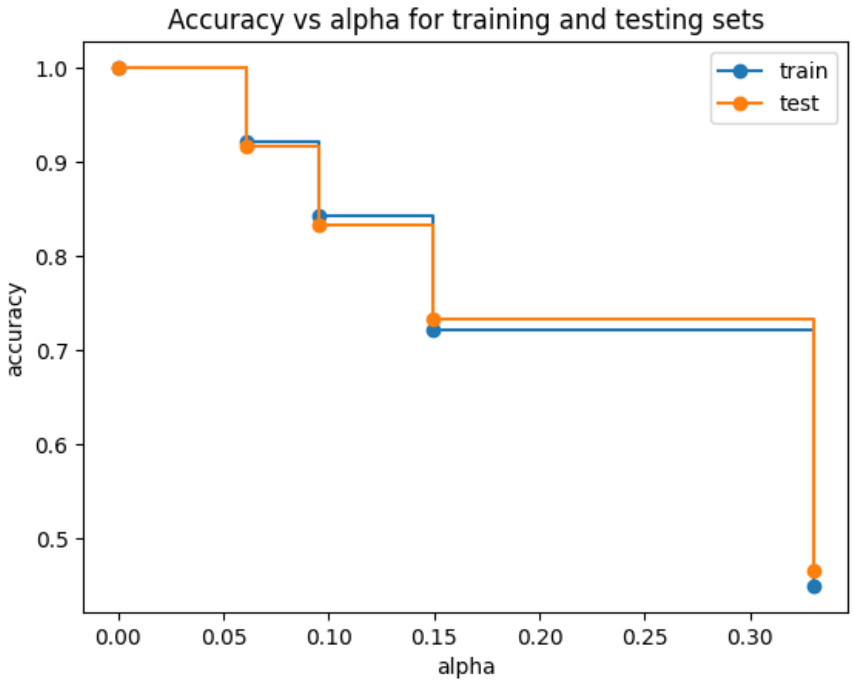
```
Classification Report:
              precision    recall  f1-score   support

       drugA       00.1      00.1      00.1         6
       drugB       00.1      00.1      00.1         5
       drugC       00.1      00.1      00.1         5
       drugX       00.1      00.1      00.1        16
       drugY       00.1      00.1      00.1        28

    accuracy                           00.1        60
   macro avg       00.1      00.1      00.1        60
weighted avg       00.1      00.1      00.1        60
```

Matrix Confusion (آ)

در انتها به سراغ post-pruning می‌رویم و با استفاده از کد زیر چند α متفاوت را تست می‌کنیم که نتیجه آن در شکل ۲۳ قابل مشاهده است. در نتیجه post-pruning روی درخت به وجود آمده توست مجموعه داده دارو تاثیر منفی خواهد داشت.

```
clf = DecisionTreeClassifier(random_state=53)
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=53, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)

```

Code :۱۴ best alpha



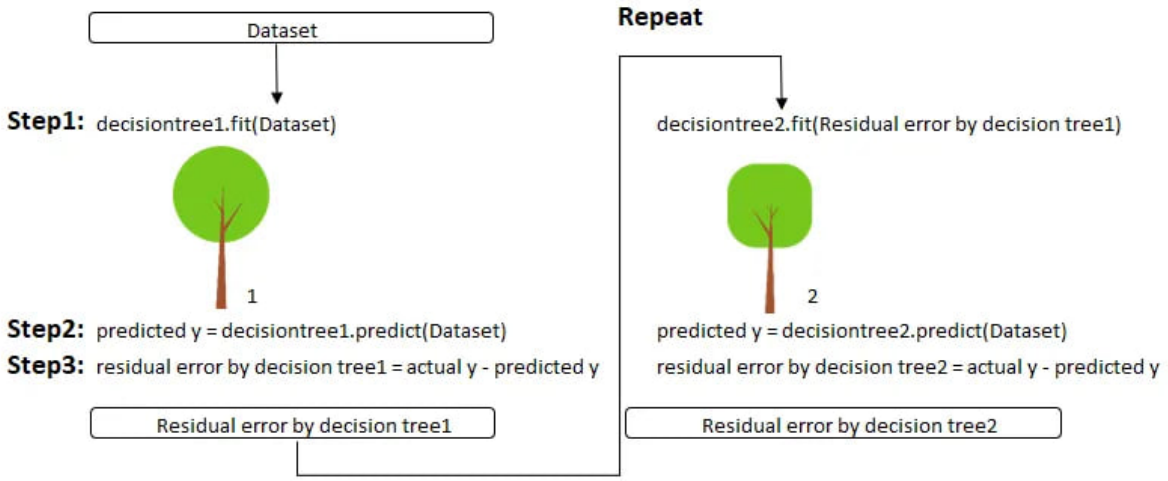
شکل ۲۳

۳.۳ بخش سوم

AdaBoost و RandomForest دو تکنیک یادگیری جمعی هستند که می‌توانند عملکرد درخت تصمیم را به طور قابل توجهی بهبود بخشند. این تکنیک‌ها به شرح زیر عمل می‌کنند.

AdaBoost: AdaBoost یک تکنیک تقویت است که بر بهبود دقت پیش‌بینی‌ها با ترکیب چندین درخت تصمیم تکیه دارد. ایده کلیدی پشت AdaBoost تنظیم وزن داده‌های آموزشی بر اساس عملکرد درخت قبلی است.

طی فرایند آموزش یک مدل AdaBoost ابتدا، وزن تمام نمونه‌ها برابر یکدیگر قرار داده می‌شود. سپس یک درخت تصمیم‌گیری با استفاده از این داده‌هاآموزش داده می‌شود. در ادامه، وزن نمونه‌هایی که نادرست طبقه‌بندی‌شده، افزایش می‌یابد تا درخت بعدی بر روی آن‌ها تمرکز کند. این فرایند برای تعداد مشخصی از تکرارها یا تا رسیدن به دقت مطلوب تکرار می‌شود. مدل نهایی یک جمع وزن‌دار از مجموعه درخت‌ها است، که در آن درخت‌های دقیق‌تر وزن بیشتری دارند. فرایند آموزش یک مدل AdaBoost در شکل ۲۴ آورده شده است.



شکل ۲۴: نحوه عملکرد و آموزش مدل AdaBoost [۲]

RandomForest: RandomForest یک است که با یک مجموعه از درختان تصمیم، یک جنگل تصمیم می‌سازد که هر کدام از درختان این جنگل بر روی یک زیرمجموعه تصادفی از داده‌ها و ویژگی‌ها آموزش داده می‌شوند.

در این روش، ابتدا زیرمجموعه‌های تصادفی از داده‌های آموزشی انتخاب می‌شوند. سپس یک درخت تصمیم‌گیری بر روی هر زیرمجموعه آموزش داده می‌شود. هر درخت به طور مستقل یک پیش‌بینی انجام می‌دهد و پیش‌بینی نهایی با رأی اکثریت بین تمامی درختان انجام می‌شود. این روش مؤثر است زیرا درختان را با آموزش آن‌ها بر روی نمونه‌ها و زیرمجموعه‌های ویژگی‌های مختلف، از هم جدا می‌کند. تجمیع پیش‌بینی‌ها از چندین درخت به طور کلی به عملکرد بهتری نسبت به هر درخت منفرد منجر می‌شود.

هر دو روش AdaBoost و RandomForest عملکرد درخت تصمیم‌گیری را بهبود می‌بخشند. در حالی که AdaBoost بر بهبود متوالی و تنظیم وزن‌ها تمرکز دارد، RandomForest مجموعه‌ای متنوع از درختان را می‌سازد و نتایج آن‌ها را میانگین می‌گیرد. این دو تکنیک می‌توانند مدلی بهتر ایجاد کنند. [۱]

حال با استفاده از کد زیر یک مدل RandomForest را آموزش می‌دهیم.

```
# Define the parameter grid for the Random Forest
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [4, 6, 8, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize the Random Forest classifier
rf_clf = RandomForestClassifier(random_state=53)

```

```python
# Initialize GridSearchCV
grid_search_rf = GridSearchCV(estimator=rf_clf, param_grid=param_grid_rf, cv=5, n_jobs=-1, )

# Fit GridSearchCV
grid_search_rf.fit(X_train, y_train)
```

Code :۱۵ Random Forest Training

نتایج بدست آمده از مدل فوق به شرح زیر می‌باشد که تمام متریک‌ها برابر ۱ شده‌اند و از روی ماتریس درهم‌ریختگی شکل ۲۵آ مشخص است که تمام نمونه‌ها به درستی تشخیص داده‌شده‌اند.

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

       drugA       00.1      00.1      00.1        17
       drugB       00.1      00.1      00.1        11
       drugC       00.1      00.1      00.1        11
       drugX       00.1      00.1      00.1        38
       drugY       00.1      00.1      00.1        63

    accuracy                           00.1       140
   macro avg       00.1      00.1      00.1       140
weighted avg       00.1      00.1      00.1       140

test metrics:

Classification Report:
              precision    recall  f1-score   support

       drugA       00.1      00.1      00.1         6
       drugB       00.1      00.1      00.1         5
       drugC       00.1      00.1      00.1         5
       drugX       00.1      00.1      00.1        16
       drugY       00.1      00.1      00.1        28

    accuracy                           00.1        60
   macro avg       00.1      00.1      00.1        60
weighted avg       00.1      00.1      00.1        60
```

Matrix Confusion (آ)

۴ سوال ۴

روش Bayse با تکیه بر توضیح احتمالاتی مسئله و با فرض اینکه رگرسورها از یکدیگر مستقل هستند، از قانون Bayse استفاده می‌کند تا یک مدل طبقه‌بندی را ایجاد کند. با توجه به اینکه الگوریتم به توزیع احتمالاتی وابسته است نرمال کردن دیتاست تاثیری در نتیجه نخواهد داشت بنابراین پیش‌پردازش خاصی لازم نیست. همانطور که در ؟؟ مشخص است این دیتاست ۱۳ ویژگی را اندازه‌گیری کرده است تا به وسیله آنها پیشبینی انجام شود.

Target	Thal	CA	Slope	Oldpeak	Exang	Thalach	Restecg	FBS	Chol	Trestbps	CP
۰	۳	۲	۲	۰.۱	۰	۱۶۸	۱	۰	۲۱۲	۱۲۵	۰
۰	۳	۰	۰	۱.۳	۱	۱۵۵	۰	۱	۲۰۳	۱۴۰	۰
۰	۳	۰	۰	۶.۲	۱	۱۲۵	۱	۰	۱۷۴	۱۴۵	۰
۰	۳	۱	۲	۰.۰	۰	۱۶۱	۱	۰	۲۰۳	۱۴۸	۰
۰	۲	۳	۱	۹.۱	۰	۱۰۶	۱	۱	۲۹۴	۱۳۸	۰

جدول ۲: Data Sample

در ادامه برای آموزش مدل مبتنی بر Bayse مجموعه داده موجود با استفاده از کد زیر به دو قسمت آموزش و ارزیابی تقسیم می‌شود و پس از آن مدل به روی دیتاست مذکور تنظیم می‌شود.

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=53)

# Initialize and train the Gaussian Naive Bayes classifier
model = GaussianNB()X_train, y_train)

# Train model
model.fit(X_train, y_train)
```

Code :۱۶ best alpha

نتایج بدست آمده از مدل فوق روی مجموعه داده ارزیابی به شرح زیر می‌باشد. با توجه به متریک Recall می‌توانیم به این نتیجه برسیم که مدل در پیدا کردن نمونه‌های کلاس ۰، نسبت به نمونه‌های کلاس ۱ ضعیف‌تر عمل کرده است و با اینکه تعداد نمونه‌ها در هر دو کلاس یکسان بوده اما دقت مدل در پیدا کردن کلاس ۰، ۱۰% ضعیف‌تر بوده اما از سوی دیگر اگر مدل درمورد یک نمونه از کلاس ۰ پیشبینی کرده، احتمال اینکه این پیشبینی درست باشد بیشتر است و این مسئله از متریک Precision قابل نتیجه می‌باشد. به صورت کلی دقت مدل در پیدا کردن و میزان درست بودن پیشبینی‌هایش از رابطه‌ای بین Precision و Recall نتیجه می‌شود که به آن Score f-۱ می‌گویند. Score f-۱ برای هردو کلاس بالای ۸۰% است و اختلاف کمی دارد که نشان

می‌دهد مدل علاوه‌براینکه از دقت مناسبی برخوردار است، به کلاس خاصی متمایل نشده است. این نتایج در ماتریس درهم‌ریختگی موجود در شکل ۲۶آ قابل ملاحضه است.

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

     Class 0       86.0      77.0      81.0       102
     Class 1       80.0      87.0      83.0       103

    accuracy                           82.0       205
   macro avg       83.0      82.0      82.0       205
weighted avg       83.0      82.0      82.0       205
```

Matrix Confusion (آ)

در کتابخانه Scikit-learn، اصطلاحات micro و macro به روش‌های مختلف میانگین‌گیری معیارها مانند Precision Recall، و f-۱ Score در مسائل طبقه‌بندی چاشاره دارند. روش micro متریک‌ها را به صورت کلی در شرایطی هر پیش‌بینی ارزش یکسانی دارد محاسبه می‌کند، به عبارت دیگر این روش تمام true positives، true negatives false negative و false positives را برای محاسبه متریک موردنظر حساب می‌کند که می‌تواند تحت تأثیر imbalance بودن کلاس‌ها قرار گیرد اما عملکرد کلی جامعی را ارائه می‌دهد. در مقابل، میانگین‌گیری macro معیارها را برای هر کلاس به طور مستقل محاسبه و سپس میانگین می‌گیرد، و به هر کلاس بدون توجه به فراوانی آن وزن مساوی می‌دهد، که آن را نسبت به imbalance کلاس‌ها کمتر حساس می‌کند. انتخاب بین میانگین‌گیری micro و macro بستگی به این دارد که آیا می‌خواهید بر عملکرد کلی تأکید کنید یا عملکرد متعادل در تمام کلاس‌ها مدنظر می‌باشد.

در انتها به ارزیابی مدل روی ۵ نمونه از داده‌ها می‌پردازیم. با استفاده از کد زیر ۵ نمونه به صورت تصادفی از مجموعه داده استخراج می‌شود و سپس مدل روی این داده پیشبینی انجام می‌دهد و در گام آخر پیشبینی به همراه کلاس واقعی نمایش داده می‌شود که این نتایج در جدول ۳ قابل مشاهده می‌باشد. این مدل تنها در یک مورد از ۵ مورد اشتباه کرده است که کلاس را ۱ تشخیص داده اما این درحالی است که کلاس واقعی ۰ بوده است.

```
test_data = data.sample(n=5, random_state=53)
pred = model.predict(test_data.drop("target", axis='columns'))
test_data['predictions'] = pred
test_data.head()
```

Predictions	Target	Thal	CA	Slope	Oldpeak	Exang	Thalach	Restecg	FBS	Chol	T
۱	۱	۲	۱	۲	۰.۰	۰	۱۵۹	۰	۰	۳۰۳	
۱	۰	۳	۱	۱	۴.۰	۰	۱۵۰	۰	۰	۲۲۹	
۱	۱	۳	۰	۱	۴.۰	۰	۱۴۷	۰	۰	۲۵۸	
۰	۰	۳	۱	۲	۰.۰	۰	۱۹۵	۰	۰	۲۸۳	
۰	۰	۲	۱	۱	۲.۱	۱	۱۴۴	۰	۱	۲۴۹	

جدول ۳: Data Test

مراجع

[۱] GeeksforGeeks. Differences between Random forest and adaboost. GeeksforGeeks، *GeeksforGeeks*. ۲۰۲۳. Accessed: ۲۰-۰۵-۲۰۲۴.

[۲] Towards Data Science. Basic Learning: ensemble Random forest، adaboost- gradient boosting — step by step explained. Towards *Data Science*، ۲۰۲۰. Accessed: ۲۰-۰۵-۲۰۲۴.

[۳] Avinash Machine Learning Cheatsheet: Activation Functions، n.d. Accessed: May ۱۸، ۲۰۲۴.