

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

یادگیری ماشین پاسخ مینی پروژه دوم

نام و نام خانوادگی: سید محمد حسینی
شماره دانشجویی: ۹۹۰۱۳۹۹
تاریخ: مهرماه ۱۴۰۲

GitHub

Drive Google



فهرست مطالب

۵	۱ سوال ۱
۵	۱.۱ بخش اول
۶	۲.۱ بخش دوم
۶	۳.۱ بخش سوم
۱۱	۱.۳.۱ نتایج MLP با تابع فعال‌ساز ReLU
۱۱	۲.۳.۱ نتایج MLP با تابع فعال‌ساز ELU
۱۳	۳.۳.۱ نتایج مدل بر مبنای mcculloch-pitts نرون
۱۵	۲ سوال ۲
۱۵	۱.۲ بخش اول
۱۵	۱.۱.۲ پیش‌پردازش
۱۸	۲.۲ بخش دوم
۲۱	۱.۲.۲ نتایج
۲۲	۳.۲ بخش سوم
۲۳	۱.۳.۲ نتایج
۲۴	۴.۲ بخش چهارم
۲۵	۱.۴.۲ نتایج
۲۷	۳ سوال ۳
۲۹	۱.۳ بخش اول
۳۰	۲.۳ بخش دوم
۳۷	۳.۳ بخش سوم
۴۰	۴ سوال ۴



فهرست تصاویر

۱۱	نمودارهای متریک و تابع هزینه حین آموزش	۲
۱۲	نمودارهای متریک و تابع هزینه حین آموزش	۵
۱۴	mcculloch-pitts و مرز تصمیم مربوط به مدل	۸
۲۱	نتایج آموزش مدل MLP	۱۰
۲۲	Matrix Confusion	۱۱
۲۳	تابع هزینه آموزش مدل MLP با بهینه ساز و تابع هزینه جدید	۱۲
۲۳	متریک‌های آموزش مدل MLP با بهینه ساز و تابع هزینه جدید	۱۳
۲۴	ماتریس درهم‌ریختگی روی داده‌های آزمون	۱۴
۲۶	matrix confusion cross-validation fold ۵	۱۵
۲۷	metrics cross-validation fold ۵	۱۶
۲۸	نمودار توزیع داده	۱۷
۳۰		۱۸
۳۱	Drug over Sex	۱۹
۳۴	Graph Hyper-parameters	۲۱
۳۵	Tree Best	۲۲
۳۷		۲۴
۳۸	نحوه عملکرد و آموزش مدل AdaBoost [۲]	۲۵



فهرست جداول

۲۹ Data Sample	۱
۴۰ Data Sample	۲
۴۲ Data Test	۳



فهرست برنامه‌ها

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



۱ سوال ۱

۱.۱ بخش اول

سوال:

فرض کنید در یک مسأله طبقه بندی دوکلاسه، دو لایه انتهایی شبکه شما فعال ساز ReLU و سیگموید است. چه اتفاقی می افتد؟

پاسخ:

توابع فعال ساز Relu و Sigmoid جزو پرکاربردترین اجزاء در یک مدل یادگیری ماشین هستند که به منظور اضافه کردن خاصیت غیرخطی به مدل مورد استفاده قرار می گیرند. به منظور تحلیل اثر این دو تابع فعال ساز بهتر است ابتدا خواص هر کدام را به صورت مجزا بررسی کنیم.

• تابع فعال سازی Sigmoid:

تابع سیگموید مقداری بین ۰ و ۱ خروجی می دهد که می تواند به عنوان احتمال تفسیر شود. این تابع فعال سازی اصولاً به عنوان فعال ساز در آخرین لایه مدل های طبقه بندی استفاده می شود؛ زیرا تفسیر احتمالی مستقیمی از خروجی ارائه می دهد. از دیگر خواص تابع سیگموید مشتق پذیری آن است که از لحاظ برنامه نویسی کار را بسیار ساده می کند؛ زیرا همان طور که در معادله ۱ مشاهده می کنید مشتق تابع سیگموید رابطه ای است متشکل از خود تابع سیگموید.

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

• تابع فعال سازی ReLU:

تابع ReLU ساده ترین تابع فعال سازی غیرخطی است که مورد استفاده قرار می گیرد، این تابع به ازای مقادیر مثبت تابع همانی است و به ازای مقادیر منفی عدد ۰ را به عنوان خروجی در نظر می گیرد که این مسئله به سادگی محاسبات هنگام Back propagation بسیار کمک می کند. مشتق این تابع به ازای مقادیر مثبت برابر ۱ و به ازای مقادیر منفی برابر ۰ است.

حال با توجه به توضیحات فوق می توانیم یک شبکه که دو لایه انتهایی آن متشکل از یک تابع ReLU و Sigmoid است را بررسی کنیم. در این شبکه اطلاعات به دست آمده از backbone شبکه به لایه یکی مانده به آخر ارسال می شود که با انجام یک عملیات خطی این اطلاعات به محیط دیگری تصویر می شوند که می توانند مقادیر مثبت و منفی را با هر اندازه ای داشته باشند، این مسئله به مقادیر موجود در Wight و Bias نرون بستگی دارد. حال خروجی نرون ها به لایه فعال ساز ارسال می شوند که در نتیجه آن مقادیر مثبت تبدیل خطی نگه داشته می شوند و مقادیر منفی به عدد ۰ تصویر می شوند. باید توجه داشت که فعال ساز ReLU به تعبیری همانند یک ماژول Attention عمل می کند و در فرایند آموزش شبکه Wight و Bias نرون ها را به سمتی متمایل می کند که در صورت پیدانکردن Feature معنی دار از ورودی خود، یک عدد منفی تولید کند. ایراد فعال ساز ReLU این است که اگر لایه های پشت هم در شبکه، همگی مقدار مثبت تولید کنند، تمام تبدیل های خطی موجود در این لایه ها می توانند با یک تبدیل خطی شبیه سازی شوند که این مسئله باعث کاهش پیچیدگی مدل ما خواهد شد؛ بنابراین استفاده



از روش‌های رگولاریزیشن و Batch Normalization در کنار توابع فعال‌سازی که به صورت 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 به گونه‌ای تنظیم شود که بتواند ورودی‌های تماماً مثبت را به یک عدد مثبت یا منفی تبدیل کند؛ زیرا تابع سیگموئید 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 سیگموئید 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:

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

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

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. علاوه بر این تابع ELU در مشتق خود ناپیوستگی ندارد و بر خلاف ReLU مقدار آن به ازای مقادیر منفی به آرامی برابر α 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 عملیات توسط کد زیر انجام گرفته است.

```

۱
۲ def point_in_triangle(point, v1, v2, v3):
۳     """Check if point (px, py) is inside the triangle with vertices v1, v2, v3."""
۴     # Unpack vertices
۵     x1, y1 = v1
۶     x2, y2 = v2
۷     x3, y3 = v3
۸     px, py = point
۹

```



```
10 # Vectors
11 v0 = (x3 - x1, y3 - y1)
12 v1 = (x2 - x1, y2 - y1)
13 v2 = (px - x1, py - y1)
14
15 # Dot products
16 dot00 = np.dot(v0, v0)
17 dot01 = np.dot(v0, v1)
18 dot02 = np.dot(v0, v2)
19 dot11 = np.dot(v1, v1)
20 dot12 = np.dot(v1, v2)
21
22 # Barycentric coordinates
23 invDenom = 1 / (dot00 * dot11 - dot01 * dot01)
24 u = (dot11 * dot02 - dot01 * dot12) * invDenom
25 v = (dot00 * dot12 - dot01 * dot02) * invDenom
26
27 # Check if point is in triangle
28 return (u >= 0) and (v >= 0) and (u + v < 1)
29
30 # Triangle vertices
31 v1 = (1, 0)
32 v2 = (2, 2)
33 v3 = (3, 0)
34
35 # Generate random points
36 np.random.seed(53)
37
38 x_coords = np.random.uniform(0, 4, 2000)
39 y_coords = np.random.uniform(-1, 3, 2000)
40 x_train = np.column_stack((x_coords, y_coords))
41
42 x_coords = np.random.uniform(0, 4, 500)
43 y_coords = np.random.uniform(-1, 3, 500)
44 x_test = np.column_stack((x_coords, y_coords))
45
46 # Label points based on whether they are inside the triangle
47 y_train = np.array([point_in_triangle(pt, v1, v2, v3) for pt in x_train]).astype(int)
48 y_test = np.array([point_in_triangle(pt, v1, v2, v3) for pt in x_test]).astype(int)
49
50
```

Code : \ 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 می‌شود و 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 ۱۱ 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 جزئیات 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.

```
1 device = "cuda" if torch.cuda.is_available else "cpu"  
2 model = MLP(input_size=2, hidden_size1=8, hidden_size2=64, hidden_size3=8, output_size=1).to(  
   device)  
3 optimizer = optim.Adam(model.parameters(), lr=0.001)  
4 criterion = nn.BCELoss() # For binary classification  
5 # DataLoader  
6 train_loader = DataLoader(TensorDataset(x_train_tensor, y_train_tensor), batch_size=128,  
   shuffle=True)  
7 test_loader = DataLoader(TensorDataset(x_test_tensor, y_test_tensor), batch_size=512, shuffle  
   =False)  
8  
9  
10
```

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 مستقلاً 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.

```
1 num_epochs = 150
2 train_loss_hist = []
3 test_loss_hist = []
4 train_metrics = []
5 test_metrics = []
6
7 for epoch in range(num_epochs):
8     loop = tqdm(train_loader)
9     model.train()
10    train_loss = 0.0
11    train_TP, train_FP, train_TN, train_FN = 0, 0, 0, 0
12
13    print("train")
14    for inputs, labels in loop:
15        inputs = inputs.to(device)
16        labels = labels.to(device)
17
18        outputs = model(inputs)
19        loss = criterion(outputs, labels)
20
21        optimizer.zero_grad()
22        loss.backward()
23        optimizer.step()
24        train_loss += loss.item()
25
26        TP, FP, TN, FN = calculate_metrics(outputs, labels)
27        train_TP += TP
28        train_FP += FP
29        train_TN += TN
30        train_FN += FN
31
32    loop.set_postfix(
33        epoch=epoch+1,
34        total_loss=train_loss / len(train_loader),
35    )
36
```



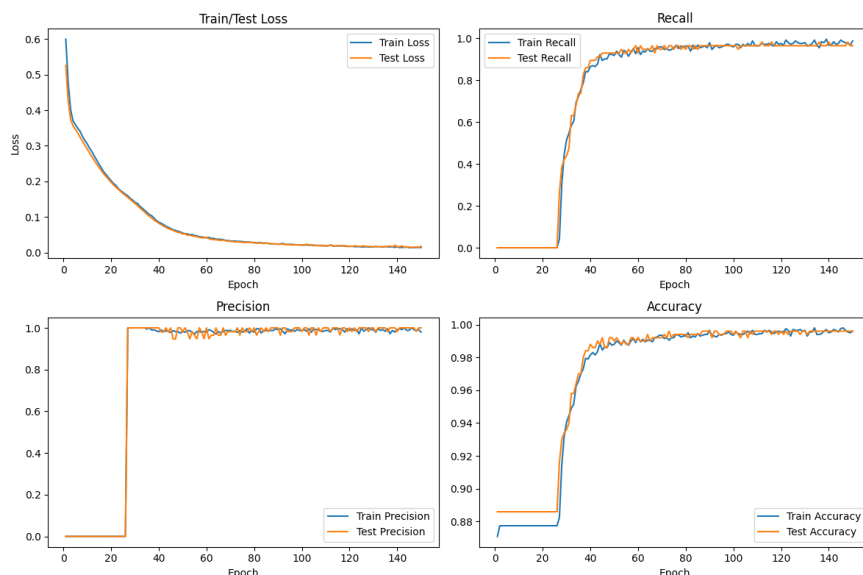
```
۳۷     train_metrics.append((train_TP, train_FP, train_TN, train_FN))
۳۸     train_loss_hist.append(train_loss / len(train_loader))
۳۹
۴۰     model.eval()
۴۱     torch.cuda.empty_cache()
۴۲     test_loss = 0.0
۴۳     test_TP, test_FP, test_TN, test_FN = 0, 0, 0, 0
۴۴     print("Test:")
۴۵
۴۶     with torch.no_grad():
۴۷         loop = tqdm(test_loader)
۴۸         for inputs, labels in loop:
۴۹             inputs = inputs.to(device)
۵۰             labels = labels.to(device)
۵۱
۵۲             outputs = model(inputs)
۵۳             loss = criterion(outputs, labels)
۵۴
۵۵             test_loss += loss.item()
۵۶
۵۷             TP, FP, TN, FN = calculate_metrics(outputs, labels)
۵۸             test_TP += TP
۵۹             test_FP += FP
۶۰             test_TN += TN
۶۱             test_FN += FN
۶۲
۶۳             loop.set_postfix(
۶۴                 loss=loss.item(),
۶۵                 total_loss=test_loss / len(test_loader),
۶۶             )
۶۷     test_metrics.append((test_TP, test_FP, test_TN, test_FN))
۶۸     test_loss_hist.append(test_loss / len(test_loader))
۶۹
```

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

Matrix Confusion (آ)

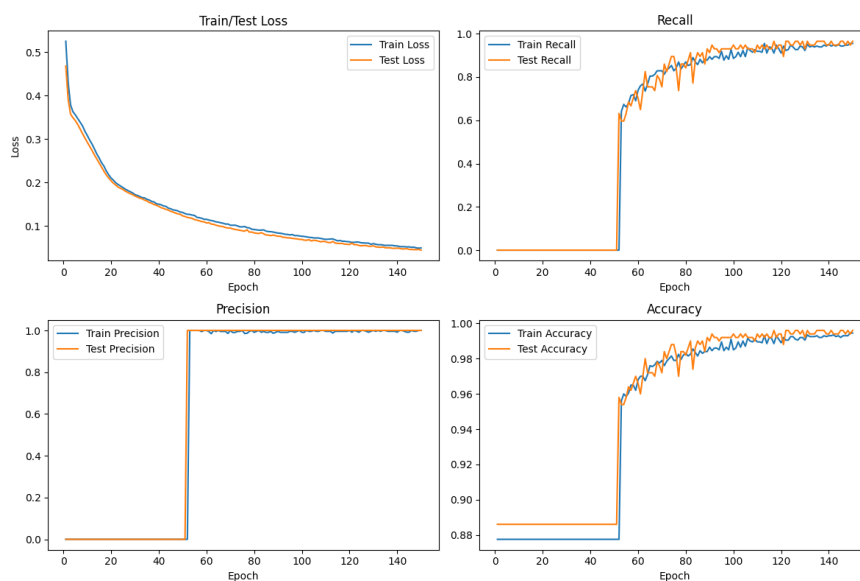
۲.۳.۱ نتایج MLP با تابع فعال‌ساز ELU

این مدل با استفاده از تابع ELU ساخته شده است و جزئیات 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 ELU به 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 ملاحظه 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 ۷ ماتریس درهم‌ریختگی به صورت نرمال و غیرنرمال نشان داده شده است که نشان می‌دهد مدل تنها دو نقطه را که مطعلق به داخل مثلث 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 گزاره‌ها 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 ELU به 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:

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

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:

```

1  #define mculloch pitts
2  class McCulloch_Pitts_neuron():
3
4  def __init__(self , weights ,bias, threshold):
5      self.weights = np.array(weights).reshape(-1, 1)      #define weights
6      self.threshold = threshold      #define threshold

```



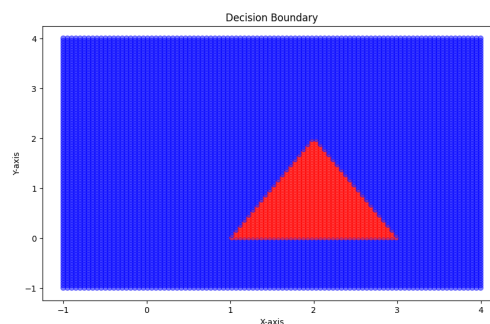
```

۷     self.bias = np.array(bias)
۸
۹     def model(self , x):
۱۰         #define model with threshold
۱۱         return (x.T @ self.weights + self.bias >= self.threshold).astype(int)
۱۲
۱۳     def model(x):
۱۴         neur1 = McCulloch_Pitts_neuron([-2, 1],2, 0)
۱۵         neur2 = McCulloch_Pitts_neuron([2, 1], -6, 0)
۱۶         neur3 = McCulloch_Pitts_neuron([0, 1], 0, 0)
۱۷         neur4 = McCulloch_Pitts_neuron([1, 1, 1], 0, 3)
۱۸
۱۹         z1 = neur1.model(np.array(x))
۲۰         z2 = neur2.model(np.array(x))
۲۱         z3 = neur3.model(np.array(x))
۲۲         z4 = np.squeeze(np.array([1-z1, 1-z2, z3]), axis=-1)
۲۳         z4 = neur4.model(z4)
۲۴
۲۵         # 3 bit output
۲۶         # return str(z1) + str(z2)
۲۷         return z4
۲۸
۲۹

```

آموزش حلقه ۴: Code

همانطور که در شکل؟؟ مشخص است یک مثلث دقیق بدست آمده است و انتظار می رود که نتایج ماتریس درهم ریختگی آن بهترین وضعیت ممکن باشد که این مسئله در شکل ۹ قابل مشاهده است.



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

Matrix Confusion (\bar{I})



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 مینی پروژه، ما بر روی داده‌های بخش 12k Drive End Bearing Fault Data تمرکز می‌کنیم. علاوه بر داده‌های استفاده شده در مینی پروژه 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 داده‌ها را به شکل یک ماتریس $(n, \text{len_data})$ ایجاد می‌کند که مقدار n با توجه به تعداد داده موجود در فایل دیتا بستگی دارد. سپس با توجه به متغیر 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.

```

۱ n_samples = 600
۲ len_data = 200
۳
۴ normal_data_matrix = normal_data[:-(normal_data.shape[0] % len_data)].reshape(-1, len_data)
۵ fault1_data_matrix = fault1_data[:-(fault1_data.shape[0] % len_data)].reshape(-1, len_data)
۶ fault2_data_matrix = fault2_data[:-(fault2_data.shape[0] % len_data)].reshape(-1, len_data)
۷ fault3_data_matrix = fault3_data[:-(fault3_data.shape[0] % len_data)].reshape(-1, len_data)
۸

```

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 feature های ممکن را از هر پنجره از داده استخراج می‌کنیم و سپس ویژگی‌هایی که مد نظر هست را انتخاب می‌کنیم.

```
1 def feature_extraction(data):
2     features = {}
3     # Standard Deviation
4     features['Standard Deviation'] = np.std(data, axis=1)
5     # Peak
6     features['Peak'] = np.max(np.abs(data), axis=1)
7     # Skewness
8     features['Skewness'] = scipy.stats.skew(data, axis=1)
9     # Kurtosis
10    features['Kurtosis'] = scipy.stats.kurtosis(data, axis=1)
11    # Crest Factor (peak divided by RMS)
12    rms = np.sqrt(np.mean(np.square(data), axis=1))
13    features['Crest Factor'] = features['Peak'] / rms
14    # Clearance Factor (peak divided by the mean of the square root of the absolute values)
15    features['Clearance Factor'] = features['Peak'] / np.mean(np.sqrt(np.abs(data)), axis=1)
16    # Peak to Peak
17    features['Peak to Peak'] = np.ptp(data, axis=1)
18    # Shape Factor (RMS divided by the mean of the absolute values)
19    features['Shape Factor'] = rms / np.mean(np.abs(data), axis=1)
20    # Impact Factor (peak divided by mean)
21    features['Impact Factor'] = features['Peak'] / np.mean(data, axis=1)
22    # Square Mean Root (the square root of the mean of the squares)
23    features['Square Mean Root'] = rms
24    # Mean
25    features['Mean'] = np.mean(data, axis=1)
26    # Absolute Mean
27    features['Absolute Mean'] = np.mean(np.abs(data), axis=1)
28    # Root Mean Square
29    features['Root Mean Square'] = rms
30    # Impulse Factor (peak divided by the absolute mean)
31    features['Impulse Factor'] = features['Peak'] / features['Absolute Mean']
32    return features
33 normal_features = pd.DataFrame(feature_extraction(normal_data_matrix))
34 fault1_features = pd.DataFrame(feature_extraction(fault1_data_matrix))
35 fault2_features = pd.DataFrame(feature_extraction(fault2_data_matrix))
```



```

۳۶ fault3_features = pd.DataFrame(feature_extraction(fault3_data_matrix))
۳۷ normal_features['Label'] = 0
۳۸ fault1_features['Label'] = 1
۳۹ fault2_features['Label'] = 2
۴۰ fault3_features['Label'] = 3
۴۱
۴۲ selected_features = ['Standard Deviation', 'Peak', 'Skewness', 'Kurtosis',
۴۳                       'Crest Factor', 'Mean', 'Root Mean Square',
۴۴                       'Impulse Factor', 'Square Mean Root', 'Shape Factor', 'Label']
۴۵
۴۶ main_df = pd.concat([normal_features, fault1_features, fault2_features, fault3_features],
                        ignore_index=True)
۴۷ df = main_df[selected_features]
۴۸

```

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 کاملاً 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.

```

۱ df_shuffled = df.sample(frac=1).reset_index(drop=True)
۲
۳ X = df_shuffled.drop('Label', axis=1)
۴ y = df_shuffled['Label']
۵
۶ # X_train_raw, X_test_raw, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state
   =53)
۷ # 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)
۸ X_train_raw, X_test_raw, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y,
   random_state=53)
۹ 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)
۱۰

```

Code :۷ Train Validation Test Split



۲.۲ بخش دوم

در ادامه مجموعه داده بدست آمده را نرمال می‌کنیم، باید توجه داشت که مقادیر میانگین و واریانس 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.

```

۱ X_train, X_train_max, X_train_min = min_max_normalization(X_train_raw)
۲ X_test, _, _ = min_max_normalization(X_test_raw, X_train_max, X_train_min)
۳ X_val, _, _ = min_max_normalization(X_val_raw, X_train_max, X_train_min)
۴
۵ X_train = torch.tensor(X_train.to_numpy(), dtype=torch.float32)
۶ X_val = torch.tensor(X_val.to_numpy(), dtype=torch.float32)
۷ X_test = torch.tensor(X_test.to_numpy(), dtype=torch.float32)
۸
۹ y_train = torch.tensor(y_train.to_numpy(), dtype=torch.long)
۱۰ y_val = torch.tensor(y_val.to_numpy(), dtype=torch.long)
۱۱ y_test = torch.tensor(y_test.to_numpy(), dtype=torch.long)
۱۲
۱۳ # Perform one-hot encoding
۱۴ y_train = torch.nn.functional.one_hot(y_train, num_classes=4).float()
۱۵ y_val = torch.nn.functional.one_hot(y_val, num_classes=4).float()
۱۶ y_test = torch.nn.functional.one_hot(y_test, num_classes=4).float()
۱۷

```

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 پیش فرض ایجاد شده است. تابع بهینه ساز این مدل از الگوریتم Adam پیروی میکند که با گام آموزشی 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 ۱۰۰ مقدار قبلی خود خواهد شد.

```

۱ class MLP(nn.Module):
۲     def __init__(self, input_dim, output_dim):
۳         super(MLP, self).__init__()
۴         self.hidden1 = nn.Linear(input_dim, 32)
۵         self.hidden2 = nn.Linear(32, 64)
۶         self.output = nn.Linear(64, output_dim)
۷         self.relu = nn.ReLU()
۸         self.softmax = nn.Softmax(dim=1)
۹
۱۰     def forward(self, x):
۱۱         x = self.relu(self.hidden1(x))

```



```
۱۲     x = self.relu(self.hidden2(x))
۱۳     x = self.output(x)
۱۴     x = self.softmax(x)
۱۵     return x
۱۶
۱۷ # Initialize the model, loss function, and optimizer
۱۸ model = MLP(input_dim=X_train.shape[1], output_dim=len(set(y)))
۱۹ criterion = nn.CrossEntropyLoss()
۲۰ optimizer = optim.Adam(model.parameters(), lr=0.001)
۲۱ scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=15,
        verbose=True)
۲۲
```

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 Epoch ۳۰ بهبود محسوس رخ ندهد فرایند آموزش متوقف می‌شود.

```
۱ # Training loop with early stopping
۲ num_epochs = 500
۳ patience = 30
۴ best_val_loss = float('inf')
۵ epochs_without_improvement = 0
۶ Best_model = None
۷
۸ # Metrics storage
۹ train_losses = []
۱۰ val_losses = []
۱۱ train_accuracies = []
۱۲ val_accuracies = []
۱۳
۱۴ for epoch in range(num_epochs):
۱۵     model.train()
۱۶     train_loss = 0.0
۱۷     train_correct = 0
۱۸     total_train = 0
۱۹
۲۰     for inputs, labels in train_loader:
۲۱         optimizer.zero_grad()
۲۲         outputs = model(inputs)
۲۳         loss = criterion(outputs, labels)
۲۴         loss.backward()
۲۵         optimizer.step()
۲۶         train_loss += loss.item()
۲۷
۲۸     _, preds = torch.max(outputs, 1)
```



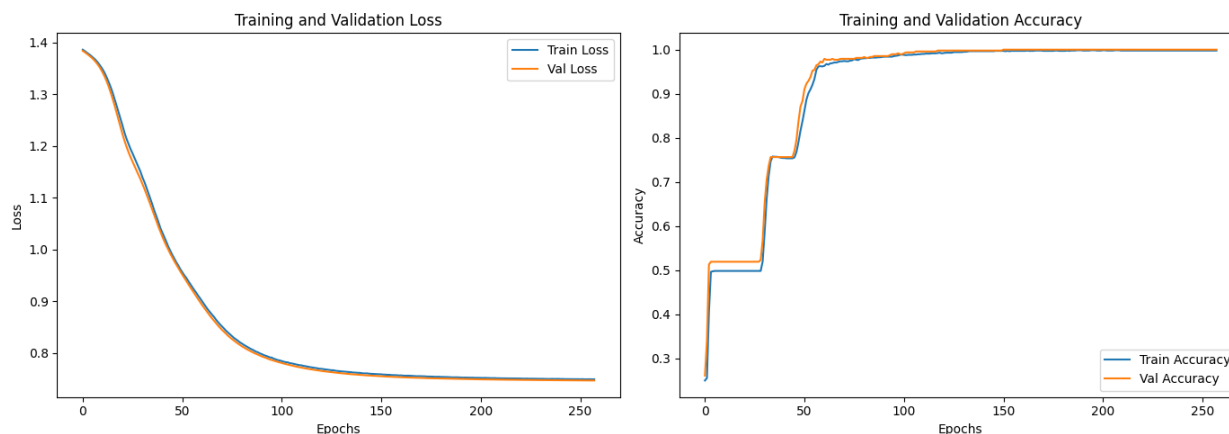
```
۲۹     _, targets = torch.max(labels,1)
۳۰     train_correct += (preds == targets).sum().item()
۳۱     total_train += labels.size(0)
۳۲
۳۳     train_loss /= len(train_loader)
۳۴     train_accuracy = train_correct / total_train
۳۵     train_losses.append(train_loss)
۳۶     train_accuracies.append(train_accuracy)
۳۷
۳۸     model.eval()
۳۹     val_loss = 0.0
۴۰     val_correct = 0
۴۱     total_val = 0
۴۲     with torch.no_grad():
۴۳         for inputs, labels in val_loader:
۴۴             outputs = model(inputs)
۴۵             loss = criterion(outputs, labels)
۴۶             val_loss += loss.item()
۴۷
۴۸             _, preds = torch.max(outputs, 1)
۴۹             _, targets = torch.max(labels,1)
۵۰             val_correct += (preds == targets).sum().item()
۵۱             total_val += labels.size(0)
۵۲
۵۳     val_loss /= len(val_loader)
۵۴     val_accuracy = val_correct / total_val
۵۵     val_losses.append(val_loss)
۵۶     val_accuracies.append(val_accuracy)
۵۷
۵۸     print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f},
۵۹           Train Accuracy: {train_accuracy:.4f}, Val Accuracy: {val_accuracy:.4f}')
۶۰
۶۱     scheduler.step(val_loss)
۶۲
۶۳     if best_val_loss - val_loss > 1e-3:
۶۴         best_val_loss = val_loss
۶۵         torch.save(model.state_dict(), 'best_model.pth')
۶۶         epochs_without_improvement = 0
۶۷     else:
۶۸         epochs_without_improvement += 1
۶۹
۷۰     if epochs_without_improvement >= patience:
۷۱         print(f'Early stopping triggered after {epoch+1} epochs.')
۷۲         break
```



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 Epoch ۲۵۸ 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 نمی شود.



شکل ۱۰: نتایج آموزش مدل MLP

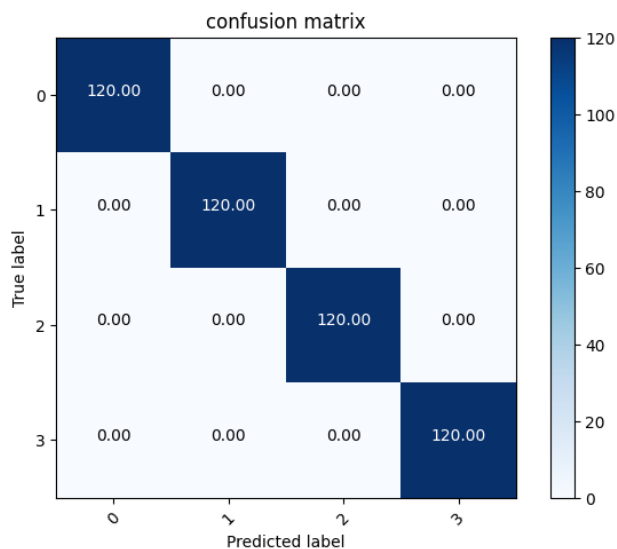
به منظور تحلیل بهتر نتایج بدست آمده از آموزش این مدل متریک ها و معیارهای دیگری علاوه بر Accuracy مورد نیاز است زیرا که متریک Accuracy با در نظر گرفتن کلاس های نرمالی که به درستی تشخیص داده شده اند در فرمول خود، باعث کاهش اثر نمونه هایی که به اشتباه تشخیص داده شده اند می شود بنابراین متریک های دیگری مورد نیاز است که در ادامه نتایج مدل را برحسب Recall و Precision و F1-Score و ماتریس درهم ریختگی روی مجموعه داده آزمون بررسی می شود.

نتایج آموزش مدل روی داده آزمون در شکل ۱۱ و شکل ۱.۲.۲ نشان دهنده عملکرد بسیار خوب این مدل می باشد که تمام نمونه ها 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	00.1	1440
weighted avg	00.1	00.1	00.1	00.1	1440



شکل ۱۱: Matrix Confusion

۳.۲ بخش سوم

در ادامه با استفاده از کد زیر ۲ مدل جدید با های configuratin 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.

```

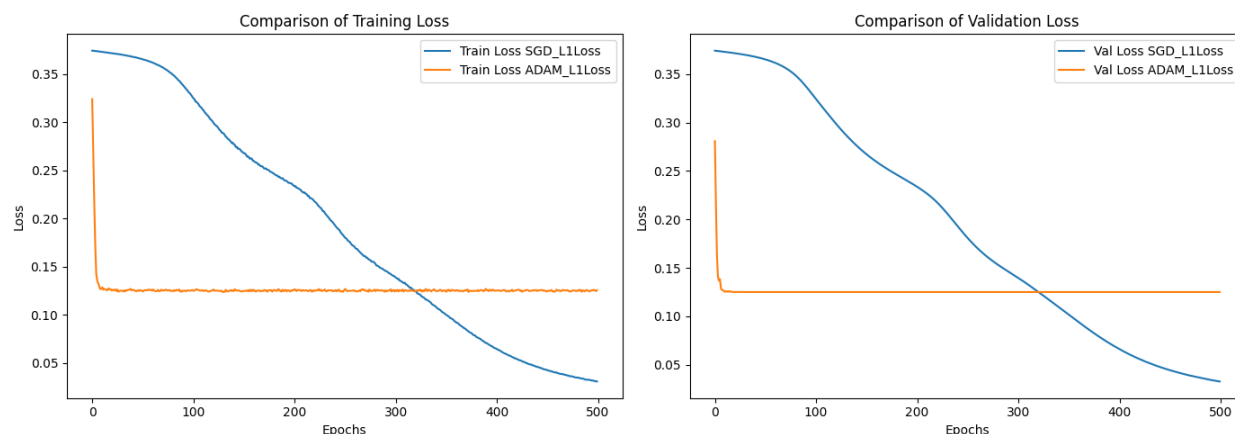
۱ # Model 1
۲ model = MLP(input_dim=X_train.shape[1], output_dim=len(set(y)))
۳ criterion = nn.L1Loss()
۴ optimizer = optim.Adam(model.parameters(), lr=0.1)
۵ scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=15,
    verbose=True)
۶
۷ # model 2
۸ model = MLP(input_dim=X_train.shape[1], output_dim=len(set(y)))
۹ criterion = nn.L1Loss()
۱۰ optimizer = optim.SGD(model.parameters(), lr=0.1)
۱۱ scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=15,
    verbose=True)
۱۲

```

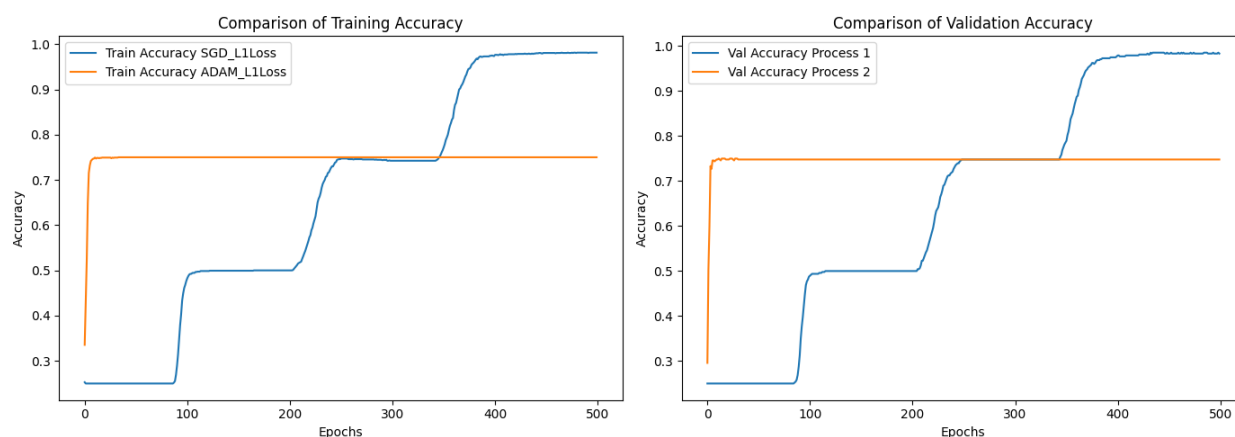
۱.۳.۲ نتایج

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 Epoch ۵۰۰ 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.

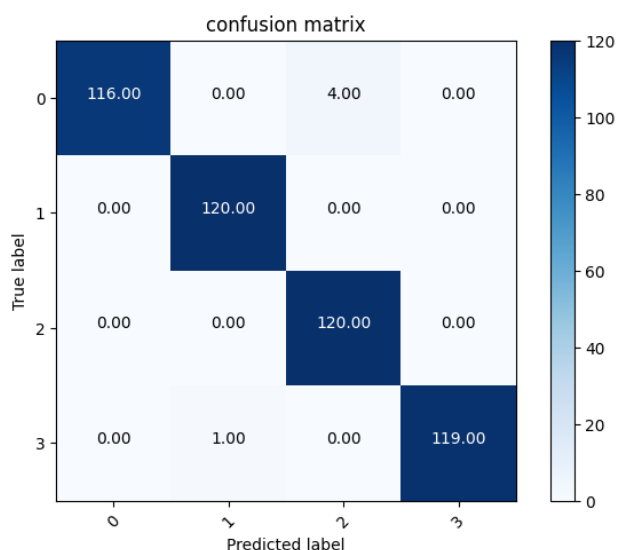


شکل ۱۲: تابع هزینه آموزش مدل MLP با بهینه ساز و تابع هزینه جدید



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 بهینه‌سازی 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 مجموعه داده آزمون نداشته است، بنابراین تغییر تابع هزینه و یا بهینه‌سازی 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



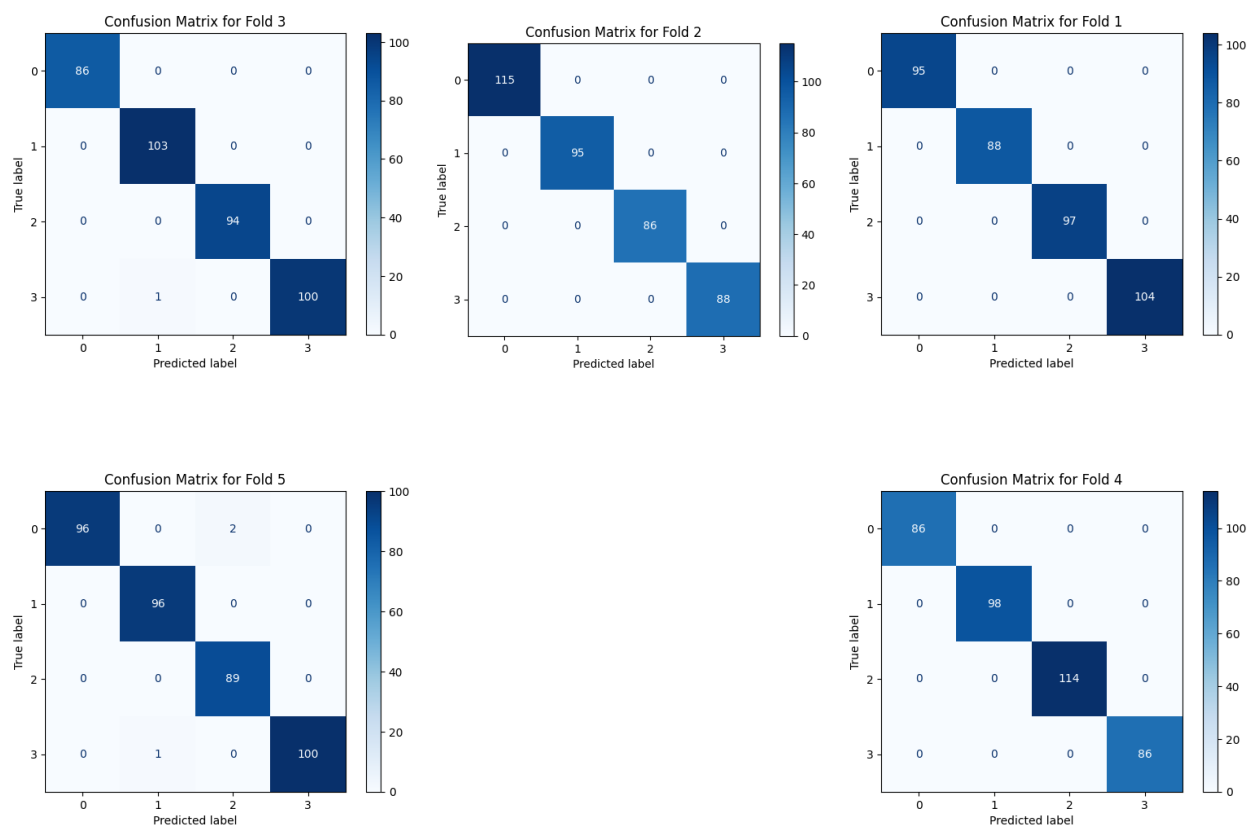
برای کلاس‌های اقلیت ارائه می‌دهد. این روش به جلوگیری از نتایج گمراه‌کننده کمک می‌کند و آن را به انتخابی بهتر برای آموزش مدل‌های یادگیری ماشین تبدیل می‌کند.

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.

```
۱ kf = KFold(n_splits=5)
۲ kf_results = []
۳
۴ for fold, (train_index, val_index) in enumerate(kf.split(X_tensor)):
۵     print(f"Fold {fold+1}")
۶     X_train, X_val = X_tensor[train_index], X_tensor[val_index]
۷     y_train, y_val = y_tensor[train_index], y_tensor[val_index]
۸
۹     train_data = torch.utils.data.TensorDataset(torch.tensor(X_train, dtype=torch.float32), torch
10         .tensor(y_train, dtype=torch.float))
11     val_data = torch.utils.data.TensorDataset(torch.tensor(X_val, dtype=torch.float32), torch.
12         tensor(y_val, dtype=torch.float))
13
14     train_loader = torch.utils.data.DataLoader(train_data, batch_size=512, shuffle=True)
15     val_loader = torch.utils.data.DataLoader(val_data, batch_size=2000, shuffle=False)
16
17     model = MLP(input_dim=X_train.shape[1], output_dim=len(set(y))).to(device)
18     optimizer = optim.Adam(model.parameters(), lr=0.001)
19     scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience
20         =15, verbose=True)
```

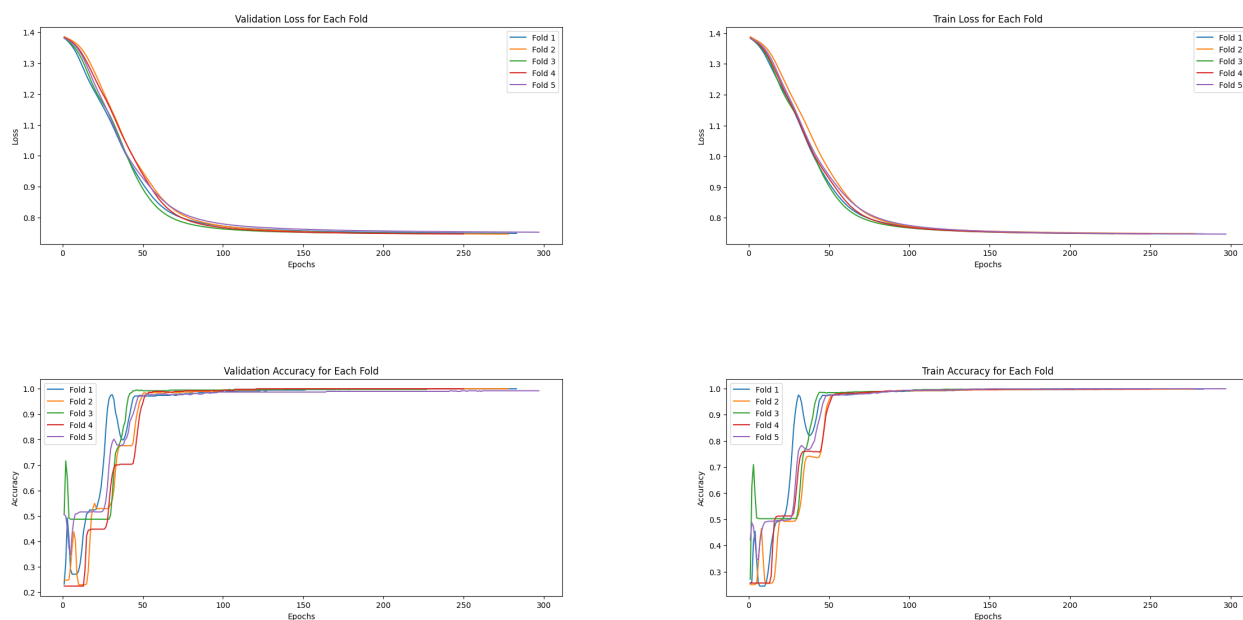
۱.۴.۲ نتایج

در شکل ۱۵ نمودارهای ماتریس درهم‌ریختگی مشخص شده است، با استفاده از این نمودارها می‌توان متریک‌های Precision و Recall 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 matrix confusion

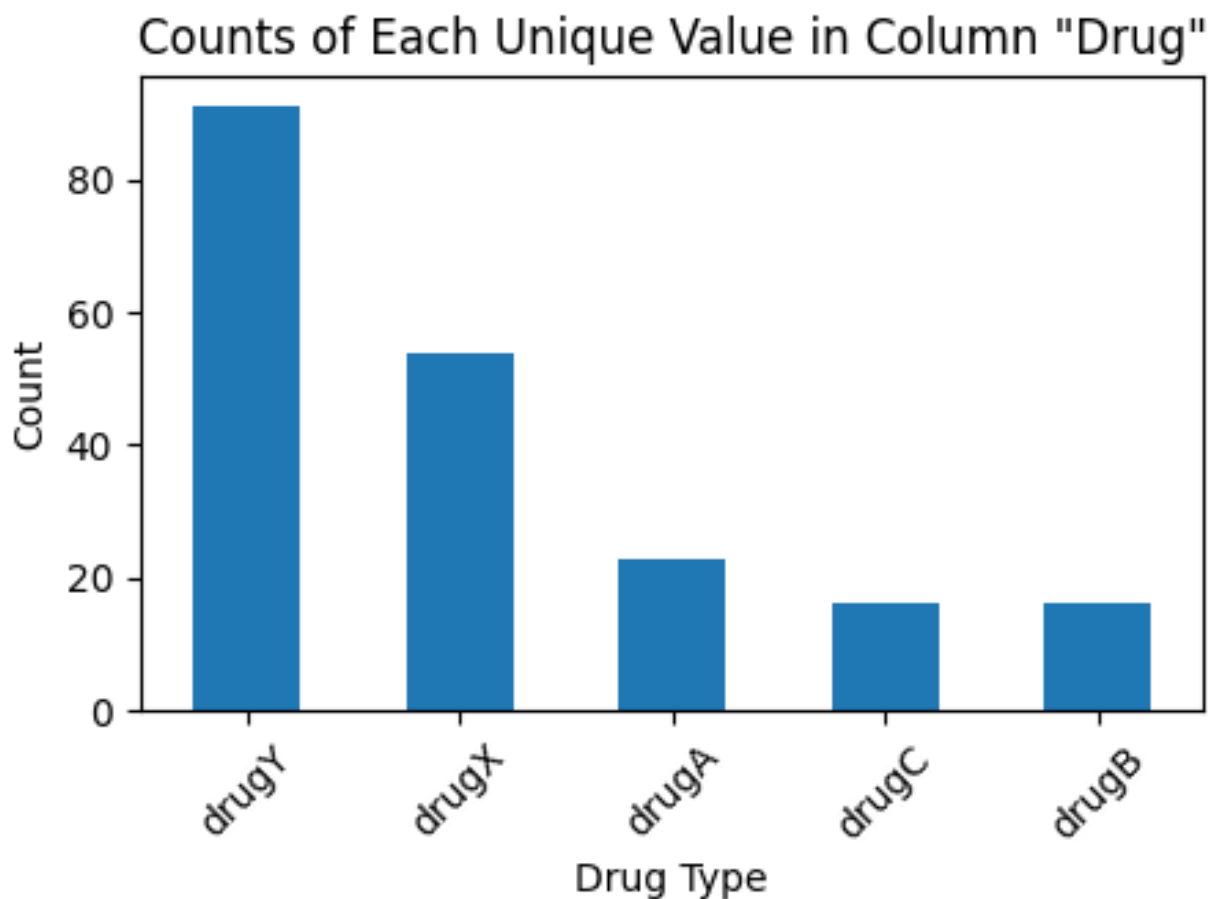
در شکل ۱۶ تصاویر مربوط تابع هزینه و متریک Accuracy طی فرایند آموزش نمایش داده نشان داده شده است که طی fold های متفاوت تفاوت فاحشی با یکدیگر ندارند و با توجه به اینکه حلقه آموزش شرایط early stop را بررسی می‌کرده، همه آموزش‌ها در حدود epoch ۲۶۰ به اتمام رسیده است.



شکل ۱۶: ۵ fold cross-validation metrics

۳ سوال ۳

در ادامه حل این سوال مجموعه داده دارو در نظر 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 ۱۷ 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 نشان داده شده است.



شکل ۱۷: 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 BP Sex 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.

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

۲

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.

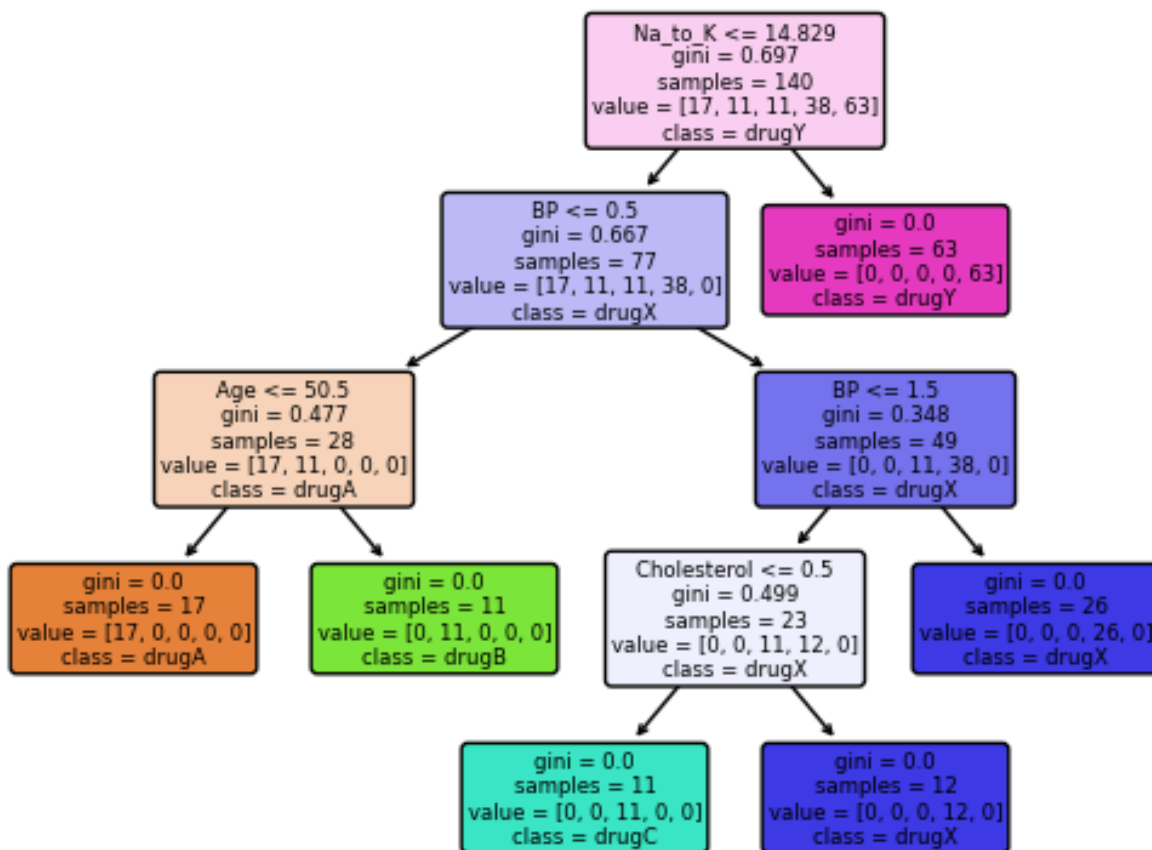
در ادامه با استفاده از کد زیر و Configuration پیش‌فرض کتابخانه sklearn یک درخت تصمیم ایجاد شده است و روی دیتاست آموزش، آموزش دیده است. نتایج بدست آمده از این آموزش در شکل ۱۸ نشان داده شده است.

```
1 # Create a Decision Tree Classifier
2 clf = DecisionTreeClassifier(random_state=53)
3
4 # Train the classifier on the training data
5 clf.fit(X_train, y_train)
6
7 plot_tree(clf, filled=True, feature_names=X.columns, class_names=clf.classes_, rounded=True)
8 plt.title('Decision Tree')
9 plt.show()
```



Code : ۱۲ Train Decision Tree

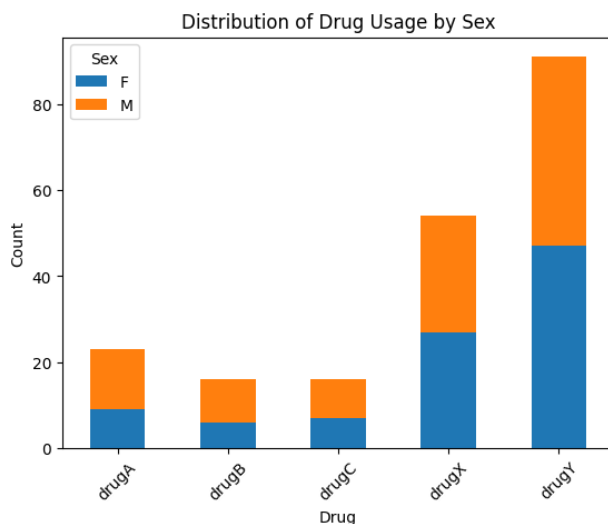
Decision Tree



شکل ۱۸

۲.۳ بخش دوم

با توجه به شکل ۱۸ 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 ۱۹ رسم شده 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 ۱۹: 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 ۲۰ مقدار 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-cross-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.

```

۱ param_grid = {
۲     'max_depth': [2, 3, 4, 5, 10, 15 ],
۳     'min_samples_split': [2, 5, 10, 15],
۴     'min_samples_leaf': [1, 2, 4, 7, 10]
۵ }
۶ scoring = {
۷     'precision': make_scorer(precision_score, average='macro'),
۸     'recall': make_scorer(recall_score, average='macro')
۹ }
۱۰ # Perform grid search with cross-validation
۱۱ grid_search = GridSearchCV(estimator=clf, param_grid=param_grid,
۱۲                             scoring=scoring, refit='precision', return_train_score=True)
۱۳ grid_search.fit(X_train, y_train)
۱۴

```

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}

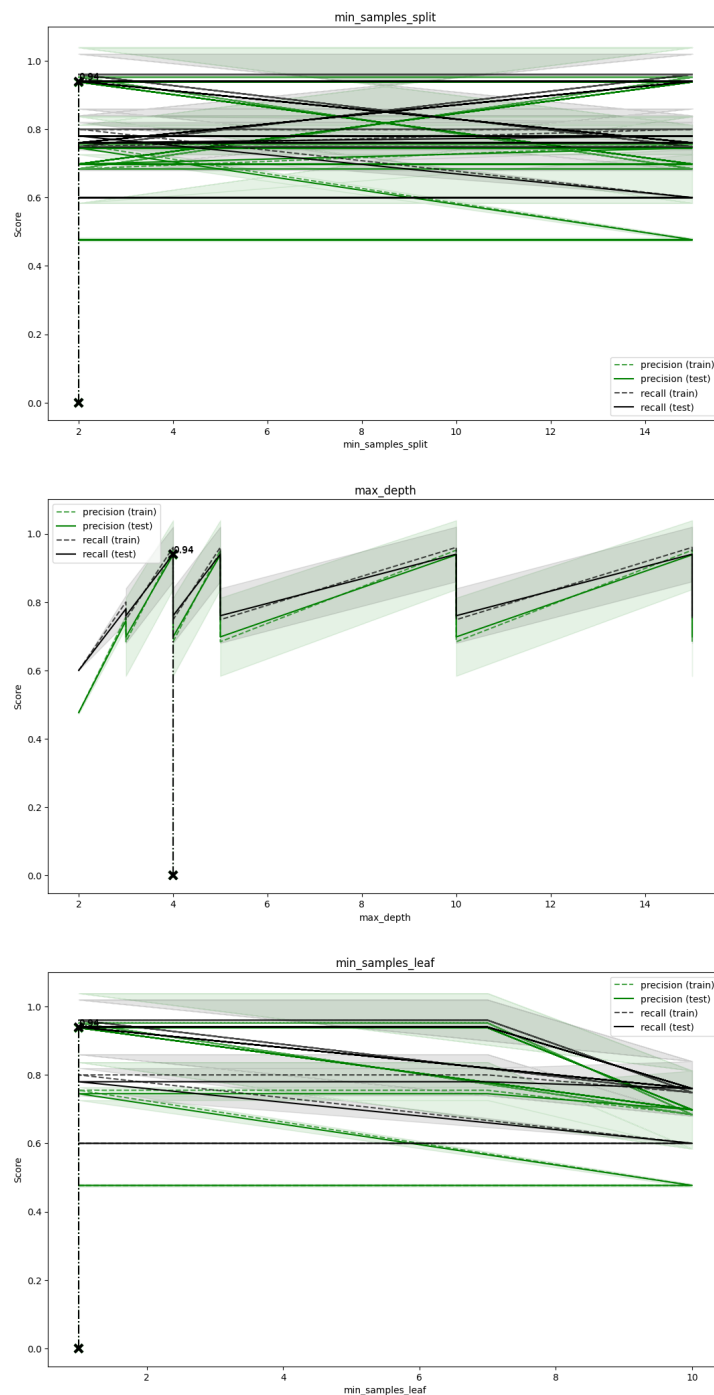
نمودارهای موجود در شکل ۲۱ به ازای تغییرات هر هایپرپارامتر کشیده شده است. همانطور که مشخص است، متغیر min_samples_split با افزایش مقدار، گاهی موجب بهبود و گاهی موجب افت عملکرد مدل می‌شود که این بهبود یا افت به مقدار دو متغیر دیگر بستگی دارد بنابراین نمیتوان به صورت کلی نظر قطعی درمورد این متغیر داد. متغیر max_depth یک رفتار تکرار شونده را نشان می‌دهد که به غیر از مقادیر خاص،



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



GridSearchCV evaluating using multiple scorers simultaneously

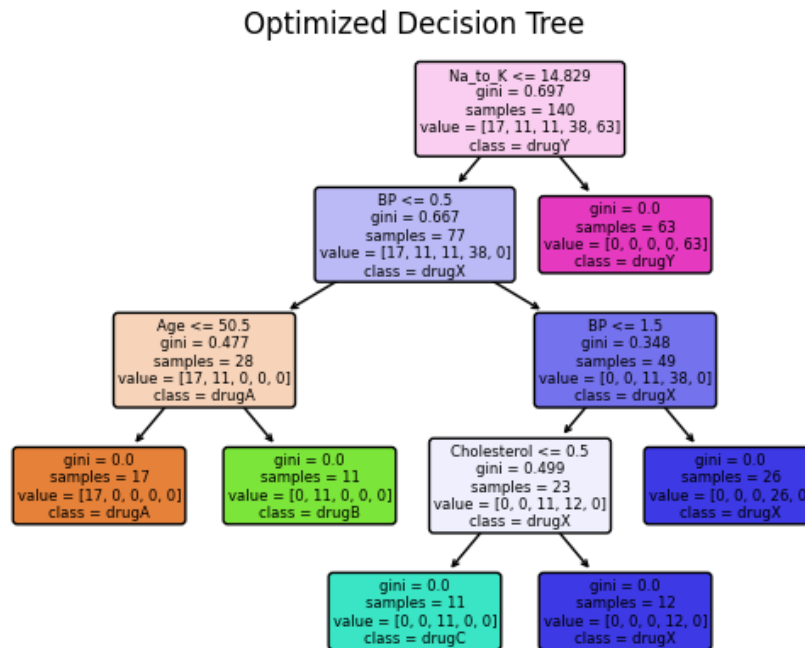


شکل ۲۱: Graph Hyper-parameters

حال به سراغ بررسی نتایج بدست آمده از بهترین مدل می‌رویم، همانطور که در شکل ۲۲ مشخص است،



تفاوتی با درخت موجود در شکل ۱۸ نداشته است، از طرف دیگر با توجه به متریک‌ها و ماتریس درهم‌ریختگی در شکل ۲۳ همچنان عملکرد مدل در بهترین حالت خود می‌باشد و تمام نمونه‌ها را به درستی تشخیص داده است



شکل ۲۲: 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)

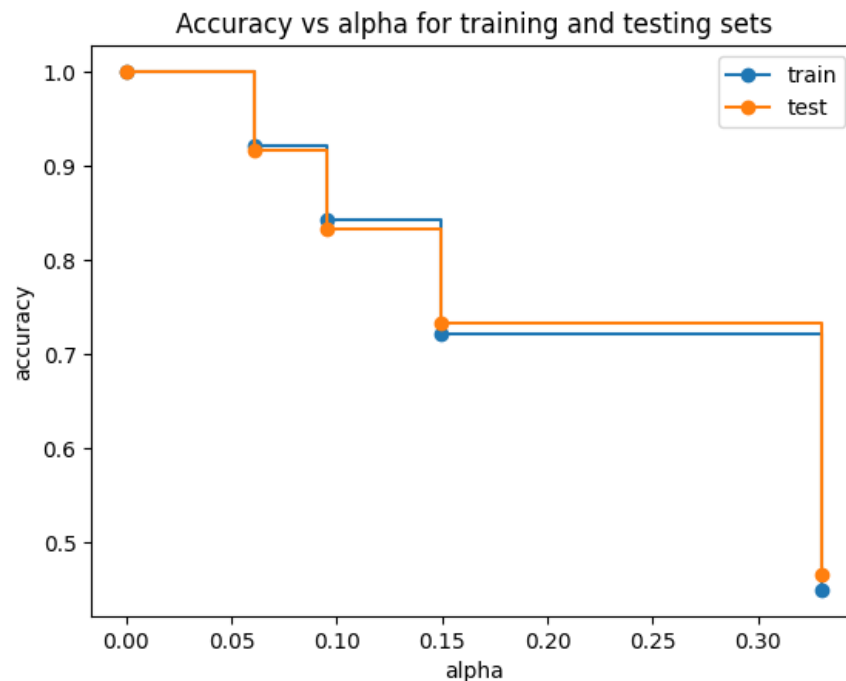
در انتها به سراغ post-pruning می‌رویم و با استفاده از کد زیر چند α متفاوت را تست می‌کنیم که نتیجه آن در شکل ۲۴ قابل مشاهده است. در نتیجه post-pruning روی درخت به وجود آمده توسط مجموعه داده دارو تاثیر منفی خواهد داشت.

```

۱ clf = DecisionTreeClassifier(random_state=53)
۲ path = clf.cost_complexity_pruning_path(X_train, y_train)
۳ ccp_alphas, impurities = path.ccp_alphas, path.impurities
۴ clfs = []
۵ for ccp_alpha in ccp_alphas:
۶     clf = DecisionTreeClassifier(random_state=53, ccp_alpha=ccp_alpha)
۷     clf.fit(X_train, y_train)
۸     clfs.append(clf)
۹

```

Code : ۱۴ best alpha



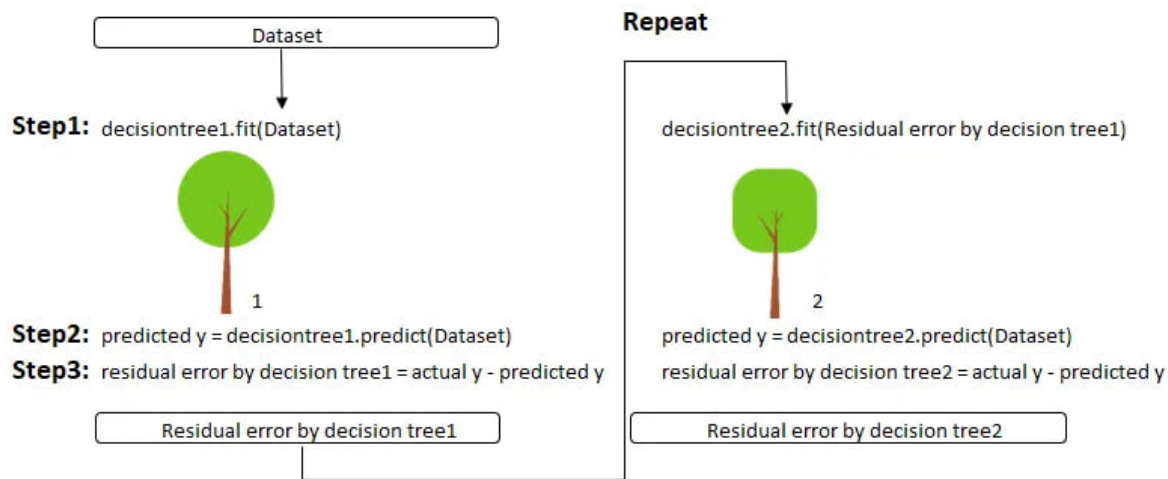
شکل ۲۴

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: 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 داده‌ها آموزش داده می‌شود. در ادامه، وزن نمونه‌هایی که نادرست طبقه‌بندی شده، 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 ۲۵ آورده شده است.



شکل ۲۵: نحوه عملکرد و آموزش مدل AdaBoost [۲]

RandomForest: Random Forest 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.

```

1 # Define the parameter grid for the Random Forest
2 param_grid_rf = {
3     'n_estimators': [100, 200, 300],
4     'max_features': ['auto', 'sqrt'],
5     'max_depth': [4, 6, 8, None],
6     'min_samples_split': [2, 5, 10],
7     'min_samples_leaf': [1, 2, 4],
8     'bootstrap': [True, False]
9 }
10
11 # Initialize the Random Forest classifier
12 rf_clf = RandomForestClassifier(random_state=53)
13

```



```

۱۴ # Initialize GridSearchCV
۱۵ grid_search_rf = GridSearchCV(estimator=rf_clf, param_grid=param_grid_rf, cv=5, n_jobs=-1, )
۱۶
۱۷ # Fit GridSearchCV
۱۸ grid_search_rf.fit(X_train, y_train)

```

Code : ۱۵ Random Forest Training

نتایج بدست آمده از مدل فوق به شرح زیر می باشد که تمام متریک ها برابر ۱ شده اند و از روی ماتریس درهم ریختگی شکل ۲۶ مشخص است که تمام نمونه ها به درستی تشخیص داده شده اند.

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 رگسورها 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.

```

۱ # Split data into training and testing sets
۲ X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=53)
۳
۴ # Initialize and train the Gaussian Naive Bayes classifier
۵ model = GaussianNB()X_train, y_train)
۶
۷ # Train model
۸ 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 f_1 -Score می‌گویند. f_1 -Score برای هر دو 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 ۲۷ قابل ملاحظه 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 (Ā)

در کتابخانه Scikit-learn، اصطلاحات micro و macro به روش‌های مختلف میانگین‌گیری معیارها مانند Precision Recall و f_1 -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، false negatives 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* adaboost. and forest random between Differences *GeeksforGeeks*. Accessed: ۲۰۲۳-۰۵-۲۰.

[۲] *gradi- adaboost, forest, Random learning: ensemble Basic Science. Data Towards* Accessed: ۲۰۲۰-۰۵-۲۰, *Science Data Towards* explained. step by step — boosting ent ۲۰۲۴-۰۵-۲۰.

[۳] *Ac- n.d. Functions. Activation Cheatsheet: Learning Machine Yuan. Avinash* Accessed: ۲۰۲۴-۰۵-۱۸ May cessed: