

به نام خدا



دانشگاه تهران



دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

|                |                    |        |
|----------------|--------------------|--------|
| آرمنین قاسمی   | نام و نام خانوادگی | پرسش ۱ |
| 810100198      | شماره دانشجویی     |        |
| امیرحسین شمودی | نام و نام خانوادگی | پرسش ۲ |
| 810100108      | شماره دانشجویی     |        |
| ۱۴۰۳.۱۲.۲۷     | مهلت ارسال پاسخ    |        |

## فهرست

### پرسش ۱. تشخیص تقلب در کارت‌های اعتباری با استفاده از شبکه‌های عصبی چندلایه (MLP)

|         |  |
|---------|--|
| 1.....  | (MLP   |
| 1.....  | 1-۱. پیش‌پردازش و بررسی دادگان                                       |
| 5.....  | 2-۱. چند گام دیگر در پیش‌پردازش (فراتر از دستور کار)                 |
| 15..... | 3-۱. طراحی و پیاده‌سازی یک شبکه MLP ساده                             |
| 23..... | 4-۱. طراحی یک شبکه MLP عمیق تر                                       |
| 30..... | 5-۱. تحلیل ماتریس آشفتگی و معیارهای ارزیابی                          |
| 32..... | 6-۱. جست‌وجوی بهترین هایپرپارامترها شبکه یک لایه مخفی با روش حریصانه |
| 33..... | 7-۱. مقایسه مدل MLP با مدل Logistic Regression                       |
| 34..... | 8-۱. جمع‌بندی  |

### پرسش ۲ - طراحی شبکه عصبی چند لایه در مسئله رگرسیون مقاومت بتن

|         |  |
|---------|--|
| 37..... | 1-۲. آماده‌سازی دادگان و تحلیل آماری   |
| 48..... | 2-۲. پیاده‌سازی مدل شبکه عصبی چند لایه |
| 49..... | 3-۲. بررسی تغییرات تنظیمات مدل         |
| 52..... | 4-۲. جمع‌بندی                          |

### پرسش ۳ - پیاده‌سازی Adaline برای دیتابست Iris

|         |                                    |
|---------|------------------------------------|
| 54..... | 1-۳. مقدمه                         |
| 54..... | 2-۳. آشنایی با Adaline             |
| 54..... | 1. الگوریتم‌های Madaline و Adaline |
| 54..... | 2. تفاوت اصلی MLP و Madaline       |
| 55..... | 3-۳. آماده‌سازی دادگان             |
| 55..... | 1. دانلود دیتابست:                 |

|                 |   |
|-----------------|---|
| 55.....         | 2. انتخاب زیرمجموعه:  |
| 56.....         | 3. نرمال سازی و تقسیم بندی داده ها:                         |
| 56.....         | 4-3. پیاده سازی و آموزش مدل Adaline                         |
| 56.....         | 1. پیاده سازی الگوریتم Adaline                              |
| 57.....         | 2. آموزش مدل با نرخ های یادگیری متفاوت                      |
| 58.....         | 3-3. نمایش و تحلیل نتایج                                    |
| 58.....         | 1. نمایش خط جداکننده:                                       |
| 61.....         | 2. نمودار خطا و دقت:  |
| 62.....         | 3. تحلیل نتایج:   |
| <b>67 .....</b> | <b>پرسش ۴ – آموزش اتوانکودر و طبقه بندی با دیتاست MNIST</b> |
| 67.....         | 2-4 . دانلود و پیش پردازش داده ها                           |
| 67.....         | 3-4 . طراحی و پیاده سازی مدل                                |
| 67.....         | بخش اول : اتوانکودر ها                                      |
| 69.....         | بخش دوم : طبقه بندی با انکودر                               |
| 70.....         | 4-4 . نتایج و تحلیل   |
| 70.....         | 1. نتایج:   |
| 73.....         | 2. تحلیل و مقایسه:  |

## شکل‌ها

- شکل 1 : Info دادگان مربوط به creditcard
- شکل 2 : head دادگان مربوط به creditcard
- شکل 3 : نمودار میله‌ای دادگان مربوط به creditcard
- شکل 4 : نمودار میله‌ای دادگان ترین undersample شده
- شکل 5 : نمودار میله‌ای دادگان تست undersample شده
- شکل 6 : هیستوگرام ویژگی های داده های train با اسکیل min , max
- شکل 7 : هیستوگرام ویژگی های داده های train استاندارد شده
- شکل 8 : هیستوگرام ویژگی های داده های test با اسکیل min , max
- شکل 9 : هیستوگرام ویژگی های داده های test استاندارد شده
- شکل 10 : ماتریس همبستگی داده ها با اسکیل min , max قبل از حذف ویژگی ها
- شکل 11: ماتریس همبستگی داده ها با اسکیل min , max بعد از حذف ویژگی ها
- شکل 12 : ماتریس همبستگی داده های استاندارد قبل از حذف ویژگی ها
- شکل 13: ماتریس همبستگی داده های استاندارد بعد از حذف ویژگی ها
- شکل 14: امتیاز ویژگی ها بر اساس Random forest برای داده های نرمالایز
- شکل 15: امتیاز ویژگی ها بر اساس Random forest برای داده های استاندارد
- شکل 16: ویژگی برتر انتخاب شده بر اساس Random forest برای داده های نرمالایز
- شکل 17: ویژگی برتر انتخاب شده بر اساس Random forest برای داده های استاندارد
- شکل 18: آموزش مدل روی داده های نرمالایز بدون dropout و بدون regularization
- شکل 19: آموزش مدل روی داده های نرمالایز بدون dropout و با regularization
- شکل 20: آموزش مدل روی داده های نرمالایز با dropout و بدون regularization
- شکل 21: آموزش مدل روی داده های نرمالایز با regularization و با dropout
- شکل 22: آموزش مدل روی داده های استاندارد بدون dropout و بدون regularization
- شکل 23: آموزش مدل روی داده های استاندارد بدون dropout و با regularization
- شکل 24: آموزش مدل روی داده های استاندارد با dropout و بدون regularization
- شکل 25: آموزش مدل روی داده های استاندارد با regularization و با dropout
- شکل 26: تست مدل روی داده های نرمالایز بدون dropout و بدون regularization
- شکل 27: تست مدل روی داده های نرمالایز بدون regularization و با dropout

- شکل 28: تست مدل روی داده های نرمالایز با dropout و بدون regularization ..... 19
- شکل 29: تست مدل روی داده های نرمالایز با dropout و با regularization ..... 20
- شکل 30: تست مدل روی داده های استاندارد بدون dropout و بدون regularization ..... 20
- شکل 31: تست مدل روی داده های استاندارد بدون dropout و با regularization ..... 20
- شکل 32: تست مدل روی داده های استاندارد با dropout و بدون regularization ..... 21
- شکل 33: تست مدل روی داده های استاندارد با dropout و با regularization ..... 21
- شکل 34: آموزش مدل عمیق تر روی داده های نرمالایز بدون dropout و بدون regularization ..... 24
- شکل 35: آموزش مدل عمیق تر روی داده های نرمالایز بدون dropout و با regularization ..... 24
- شکل 36: آموزش مدل عمیق تر روی داده های نرمالایز با dropout و بدون regularization ..... 24
- شکل 37: آموزش مدل عمیق تر روی داده های نرمالایز با dropout و با regularization ..... 25
- شکل 38: آموزش مدل عمیق تر روی داده های استاندارد بدون dropout و بدون regularization ..... 25
- شکل 39: آموزش مدل عمیق تر روی داده های استاندارد بدون dropout و با regularization ..... 25
- شکل 40: آموزش مدل عمیق تر روی داده های استاندارد با dropout و بدون regularization ..... 26
- شکل 41: آموزش مدل عمیق تر روی داده های استاندارد با dropout و با regularization ..... 26
- شکل 42: تست مدل عمیق تر روی داده های نرمالایز بدون dropout و بدون regularization ..... 27
- شکل 43: تست مدل عمیق تر روی داده های نرمالایز بدون dropout و با regularization ..... 27
- شکل 44: تست مدل عمیق تر روی داده های نرمالایز با dropout و بدون regularization ..... 27
- شکل 45: تست مدل عمیق تر روی داده های نرمالایز با dropout و با regularization ..... 28
- شکل 46: تست مدل عمیق تر روی داده های استاندارد بدون dropout و بدون regularization ..... 28
- شکل 47: تست مدل عمیق تر روی داده های استاندارد بدون dropout و با regularization ..... 28
- شکل 48: تست مدل عمیق تر روی داده های استاندارد با dropout و بدون regularization ..... 29
- شکل 49: تست مدل عمیق تر روی داده های استاندارد با dropout و با regularization ..... 29
- شکل 50: نتیجه تست مدل Logistic Regression با داده های نرمالایز ..... 33
- شکل 51: نتیجه تست مدل Logistic Regression با داده های استاندارد ..... 34
- شکل 52: نگاهی کلی به دادگان مربوط به مقاومت بتن ..... 37
- شکل 53: info دادگان مربوط به مقاومت بتن ..... 37
- شکل 54: مشخصات آماری دادگان مربوط به مقاومت بتن ..... 38
- شکل 55: هیستوگرام تک تک ویژگی ها ..... 39

|         |  |
|---------|--|
| 41..... | شکل 56: پیدا کردن K بهینه برای ستون FlyAshComponent                                    |
| 41..... | شکل 57: پیدا کردن K بهینه برای ستون BlastFurnaceSlag                                   |
| 41..... | شکل 58: پیدا کردن K بهینه برای ستون SuperplasticizerComponent                          |
| 43..... | شکل 59: هیستوگرام ویژگی های داده های تست بعد از استاندارد سازی و جایگزینی مقادیر گمشده |
| 43..... | شکل 60: هیستوگرام ویژگی های داده های تست بعد از استاندارد سازی و جایگزینی مقادیر گمشده |
| 44..... | شکل 61: ماتریس همبستگی ویژگی ها  |
| 45..... | شکل 62: CementComponent ویژگی Scatter Plot :62   |
| 45..... | شکل 63: WaterComponent ویژگی Scatter Plot :63  |
| 46..... | شکل 64: CoarseAggregateComponent ویژگی Scatter Plot :64                                |
| 46..... | شکل 65: FineAggregateComponent ویژگی Scatter Plot :65                                  |
| 46..... | شکل 66: AgeInDays ویژگی Scatter Plot :66   |
| 47..... | شکل 67: FlyAshComponent ویژگی Scatter Plot :67   |
| 47..... | شکل 68: BlastFurnaceSlag ویژگی Scatter Plot :68  |
| 47..... | شکل 69: SuperplasticizerComponent ویژگی Scatter Plot :69                               |
| 48..... | شکل 70: توابع مربوط به خطاب و دقت در آموزش شبکه عصبی رگرسور                            |
| 49..... | شکل 71: توابع مربوط به خطاب و دقت برای ایپاک های مختلف                                 |
| 50..... | شکل 72: توابع مربوط به خطاب و دقت برای توابع خطابی مختلف                               |
| 51..... | شکل 73: توابع مربوط به خطاب و دقت برای توابع بهینه ساز مختلف                           |
| 55..... | شکل 74: نام کلاس ها و ویژگی های دیتاست   |
| 55..... | شکل 75: بخش از داده های فیلتر شده  |
| 56..... | شکل 76: نرمال سازی و تقسیم داده ها   |
| 58..... | شکل 77: مقادیر خطاب و دقت شبکه در هر epoch با نرخ های یادگیری مختلف                    |
| 59..... | شکل 78: خطوط جداکننده بر روی دادگان آموزش (epoch 10)                                   |
| 60..... | شکل 79: خطوط جداکننده بر روی دادگان تست (epoch 10)                                     |
| 61..... | شکل 80: نمودار خطاب بر حسب epoch طی شده  |
| 62..... | شکل 81: نمودار دقت شبکه بر حسب epoch طی شده  |
| 63..... | شکل 82: خطوط جداکننده بر روی دادگان آموزشی (epoch 50)                                  |

|         |  |
|---------|--|
| 64..... | شکل 83: پیشرفت مدل با نرخ یادگیری 0.001                        |
| 64..... | شکل 84: پیشرفت مدل با نرخ یادگیری 0.005                        |
| 65..... | شکل 85: پیشرفت مدل با نرخ یادگیری 0.02                         |
| 67..... | شکل 86: دانلود و پیش پردازش داده ها                            |
| 68..... | شکل 87: طراحی اتوانکودر با خروجی 8 نورون                       |
| 69..... | شکل 88: طراحی طبقه بند با اتوانکودر                            |
| 70..... | شکل 89: نمودار خطای بازسازی اتوانکودر با 8 نورون خروجی         |
| 71..... | شکل 90: نمودار خطای بازسازی اتوانکودر با 4 نورون خروجی         |
| 72..... | شکل 91: نمودار دقیق مدل با اتوانکودر با 8 نورون خروجی          |
| 73..... | شکل 92: نمودار دقیق مدل با اتوانکودر با 4 نورون خروجی          |
| 76..... | شکل 93: نمودار خطای بازسازی اتوانکودر 3 لایه با 16 نورون خروجی |
| 77..... | شکل 94: نمودار دقیق مدل با اتوانکودر با 16 نورون خروجی         |

## جدول‌ها

- 21 : مقایسه تمام متریک های کیفیت مدل برای تمام مدل ها با یک لایه مخفی.....  
جدول 2 : مقایسه تمام متریک های کیفیت مدل برای تمام مدل ها با دو لایه مخفی .....  
جدول 3 . نتیجه grid search  
جدول 4 : نتیجه تست Logistic Regression  
جدول 5 : نتیجه تست و ارزیابی در شبکه عصبی رگرسور.....  
جدول 6 : نتیجه تست و ارزیابی برای ایپاک های مختلف.....  
جدول 7 : نتیجه تست و ارزیابی برای توابع خطای مختلف ..  
جدول 8 : نتیجه تست و ارزیابی برای توابع بهینه‌ساز مختلف ..  
جدول 9 : تعداد پارامتر های مدل های بررسی شده.....  
جدول 10: دقت طبقه بندی مدل ها در اخرين epoch ..  
جدول 11: دقت طبقه بندی در مدل پیشنهادی ..  
جدول 12: تعداد پارامتر های مدل پیشنهادی ..

# پرسش 1. تشخیص تقلب در کارت‌های اعتباری با استفاده از شبکه‌های عصبی چندلایه (MLP)

## ۱-۱. پیش‌پردازش و بررسی دادگان

برای بررسی خلاصه داده‌ها می‌توانیم پنج داده اول (head) جدول و اطلاعات کلی (info) جدول را ببینیم.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column   Non-Null Count   Dtype  
--- 
 0   Time     284807 non-null   float64
 1   V1       284807 non-null   float64
 2   V2       284807 non-null   float64
 3   V3       284807 non-null   float64
 4   V4       284807 non-null   float64
 5   V5       284807 non-null   float64
 6   V6       284807 non-null   float64
 7   V7       284807 non-null   float64
 8   V8       284807 non-null   float64
 9   V9       284807 non-null   float64
 10  V10      284807 non-null   float64
 11  V11      284807 non-null   float64
 12  V12      284807 non-null   float64
 13  V13      284807 non-null   float64
 14  V14      284807 non-null   float64
 15  V15      284807 non-null   float64
 16  V16      284807 non-null   float64
 17  V17      284807 non-null   float64
 18  V18      284807 non-null   float64
 19  V19      284807 non-null   float64
 ...
 29  Amount    284807 non-null   float64
 30  Class     284807 non-null   int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

شکل 1 دادگان مربوط به creditcard Info

```
Time      V1      V2      V3      V4      V5      V6      V7  \
0 0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1 0.0  1.191857  0.266151  0.166480  0.448154  0.068018 -0.082361 -0.078803
2 1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.880499  0.791461
3 1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4 2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

V8      V9  ...      V21      V22      V23      V24      V25  \
0 0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1 0.085102 -0.255425  ... -0.225775 -0.638672  0.181288 -0.339846  0.167170
2 0.247676 -1.514654  ...  0.247998  0.771679  0.999412 -0.689281 -0.327642
3 0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

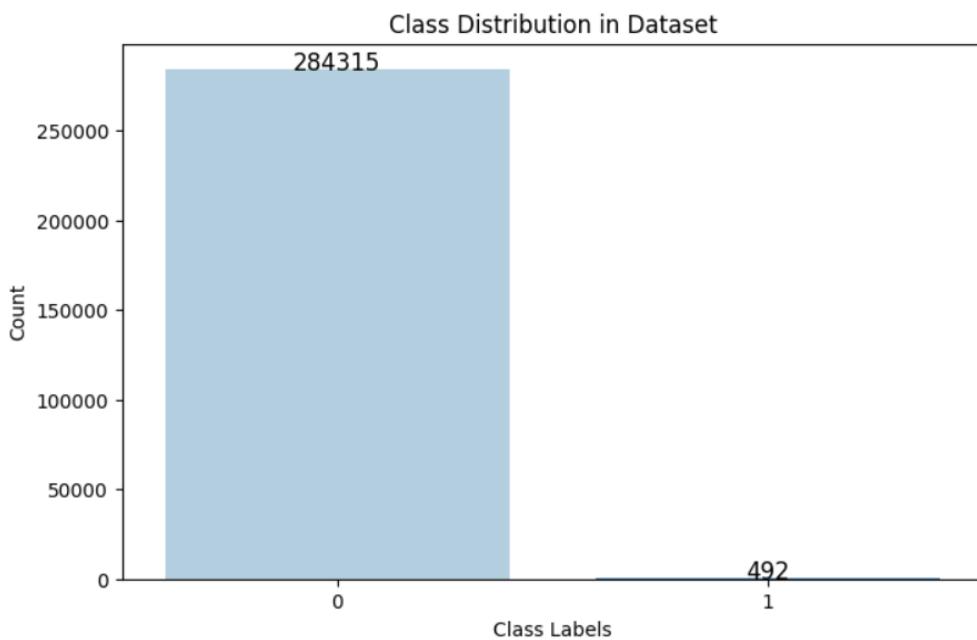
V26      V27      V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.56      0
4  0.502292  0.219422  0.215153   69.99      0
```

شکل 2 دادگان مربوط به creditcard head

با توجه به بررسی این گزارش‌ها می‌توانیم به تعداد ویژگی‌ها (ستون‌ها) و نوع هر یک از آنها پی ببریم. همچنین می‌توانیم ببینیم که جدول ما 284807 سطر دارد و با توجه به همین موضوع (تعداد سطر‌ها و ستون‌ها) حجم به نسبت زیادی از حافظه را اشغال کرده است.

نکته‌ای که در head دادگان توجه را به خود جلب می‌کند این است که هر پنج داده اول جدول دارای class=0 هستند. می‌توان گفت که با توجه به این موضوع احتمالاً تعداد داده‌هایی که داری class=0 هستند زیاد است. در بخش بعدی این مورد را خواهیم دید.

نمودار میله دادگان را رسم می‌کنیم. در این نمودار داده‌های را با توجه به class آنها plot می‌کنیم:



شکل 3 : نمودار میله‌ای دادگان مربوط به creditcard

همانطور که مشاهده می‌کنیم حدسمن درست بود و تعداد داده‌های مربوط به کلاس 0 بسیار بیشتر از تعداد داده‌های کلاس دیگر است. این عدم تعادل (این عدم تعادل وحشتناک!) باعث می‌شود مدل یک کلاس را خوب یادبگیرد و کلاس دیگر را خوب یادنگیرد. واضح است که اگر مدل 284315 داده مربوط به کارت‌های اعتباری بدون تقلب و 492 داده مربوط به کارت‌های اعتباری تقلب شده را ببیند، داده‌های کارت‌های تقلب نشده را بهتر یاد می‌گیرد. این موضوع ممکن است در دقت کلی تاثیر چندانی نداشته باشد در این داده‌های حتی اگر به ویژگی‌ها نگاه نکنیم و برای همه داده‌ها کلاس 0 را پیش‌بینی کنیم دقیق بالای 99% می‌گیریم! با توجه به این موضوع می‌توان گفت که این مورد قدرت تعمیم پذیری مدل را هم کم می‌کند.

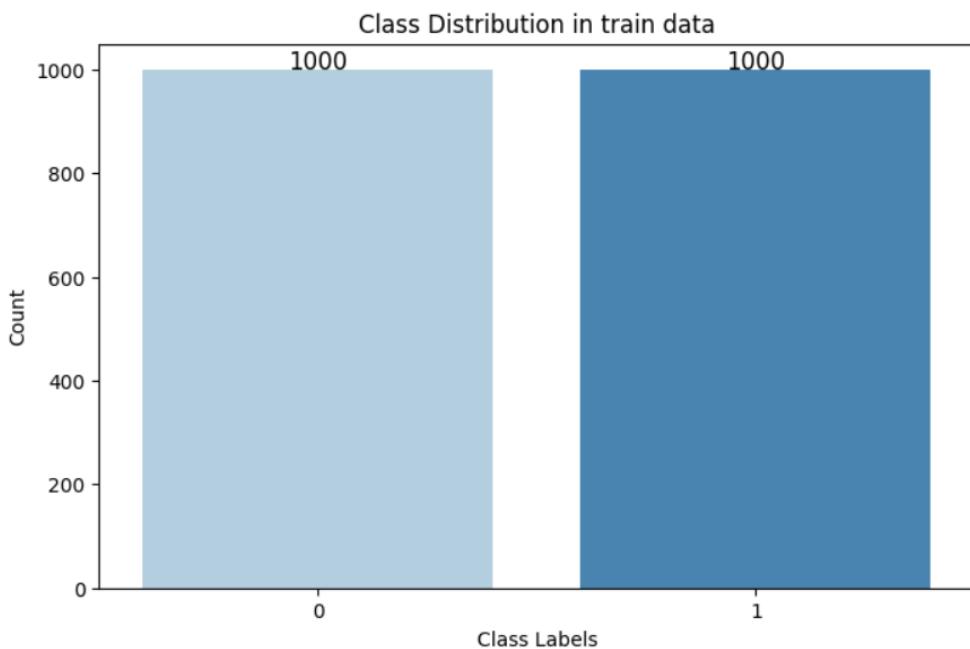
برای حل کردن این مشکل می توانیم undersampling یا oversampling انجام دهیم. یعنی یا تعداد داده های کلاس کمتر را به صورت مصنوعی افزایش دهیم و داده های رندوم با حدود همان مشخصات تولید کنیم. یا داده های کلاس بیشتر را کاهش دهیم و به تعداد داده های کلاس کمتر برسانیم. واضح است که در روش اول داده های فیک به مجموعه دادگان اضافه می شود و در روش دوم داده های زیادی از بین میروند ولی به هر حال مجبور به استفاده از یک از این دو روش ( یا ترکیب آن ها) هستیم.

با توجه به اینکه قرار است یک مدل MLP را روی این داده ها آموزش دهیم، باید تعداد پارامتر های بسیار زیادی را از روی این داده ها تخمین بزنیم (در بخش اول که یک مدل MLP ساده با 64 نرون در لایه مخفی داریم با فرض اینکه هیچ کدام از ویژگی ها را حذف نکنیم باید حدود 2000 پارامتر را تخمین بزنیم!). اگر بخواهیم undersampling انجام دهیم یعنی باید داده های کلاس 0 را هم به حدود 500 داده برسانیم تا تعادل بین کلاس ها حفظ شود. یعنی اگر undersampling کنیم نهایتا 1000 داده داریم و واضح است که نمی توان از روی 1000 داده، 2000 پارامتر را تخمین زد، بنابراین undersampling برای این مسئله مناسب نیست.

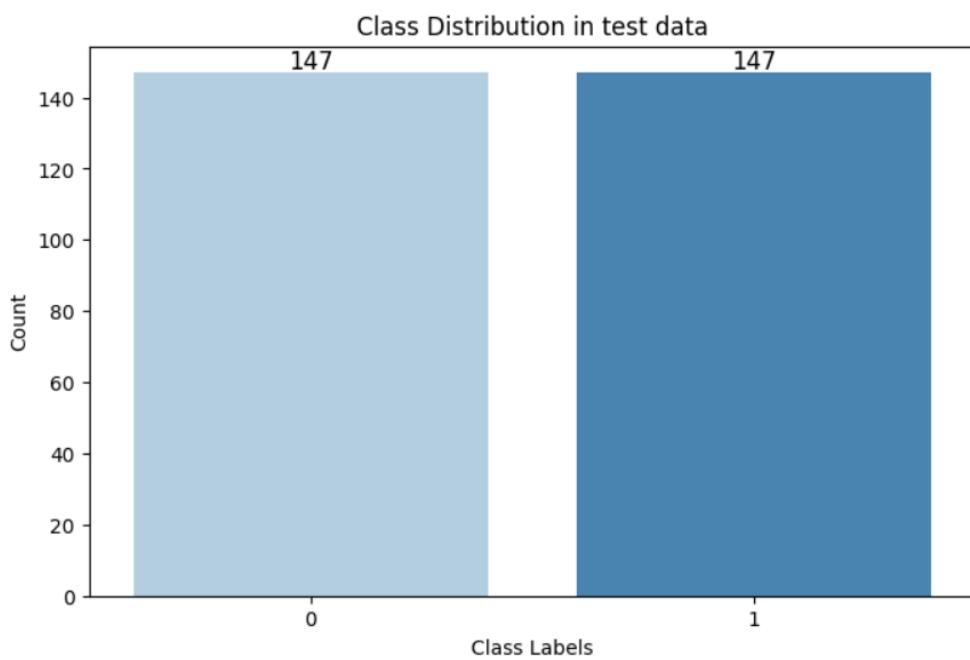
از طرفی اگر بخواهیم oversampling نیز انجام دهیم باید حدود 280000 داده از کلاس 1 را به صورت مصنوعی تولید کنیم در صورتی که فقط حدود 500 داده واقعی از این کلاس داریم. پس این راهکار نیز منطقی به نظر نمیرسد.

برای اینکه بهترین trade off را انجام دهیم، ابتدا داده های کلاس 1 را به مقدار کمی oversample می کنیم و سپس داده های کلاس 0 را undersample می کنیم تا تعداد داده های هر دو کلاس برابر شود. برای این کار ابتدا 30 درصد از داده های کلاس 1 را برای تست بر میداریم و به همان مقدار از داده های کلاس 0 بر میداریم (این کار را انجام میدهیم که داده های مصنوعی وارد مجموعه تست نشوند). در داده های باقی SMOTE مانده تعداد داده های کلاس 1 را با استفاده از روش SMOTE تا 1000 داده افزایش میدهیم (روش SMOTE بر خلاف روش های معمول دیگر داده ها را تکرار نمیکند با استفاده از متدهای KNN و داده های مصنوعی تولید میکند) و سپس تعداد داده های کلاس 0 را تا 1000 داده کاهش میدهیم. دقت کنیم که پس از انجام این موارد احتمالا باز هم تعداد داده های ما برای تخمین 2000 پارامتر کافی نخواهد بود ولی در گام های بعدی با حذف ویژگی های غیرضروری سعی می کنیم تعداد پارامتر ها را کمتر کنیم.

حالا که داده های تست و ترین را داریم آنها را استاندارد و نرمال می کنیم. دقت کنیم که برای نرمال سازی و استاندارد سازی داده ها تست از میانگین، واریانس، مینیمم و ماکزیمم داده های ترین استفاده می کنیم. پس از انجام این موارد دوبار نمودار میله ای را برای کلاس ها رسم می کنیم:



شکل 4 : نمودار میله‌ای دادگان ترین undersample شده



شکل 5 : نمودار میله‌ای دادگان تست undersample شده

همانطور که مشخص در هر دو دسته داده های تست و ترین تعادل داده ها حفظ شده. در نگاه اول ممکن است بنظر برسد نسبت 70/30 داده های ترین به تست حفظ نشده است ولی در واقع این نسبت در داده های واقعی موجود در مجموعه داده ها حفظ شده و صرفا در مجموعه داده های ترین داده های مصنوعی تولید شده اند.

## 2-2. چند گام دیگر در پیش پردازش (فراatter از دستور کار!)

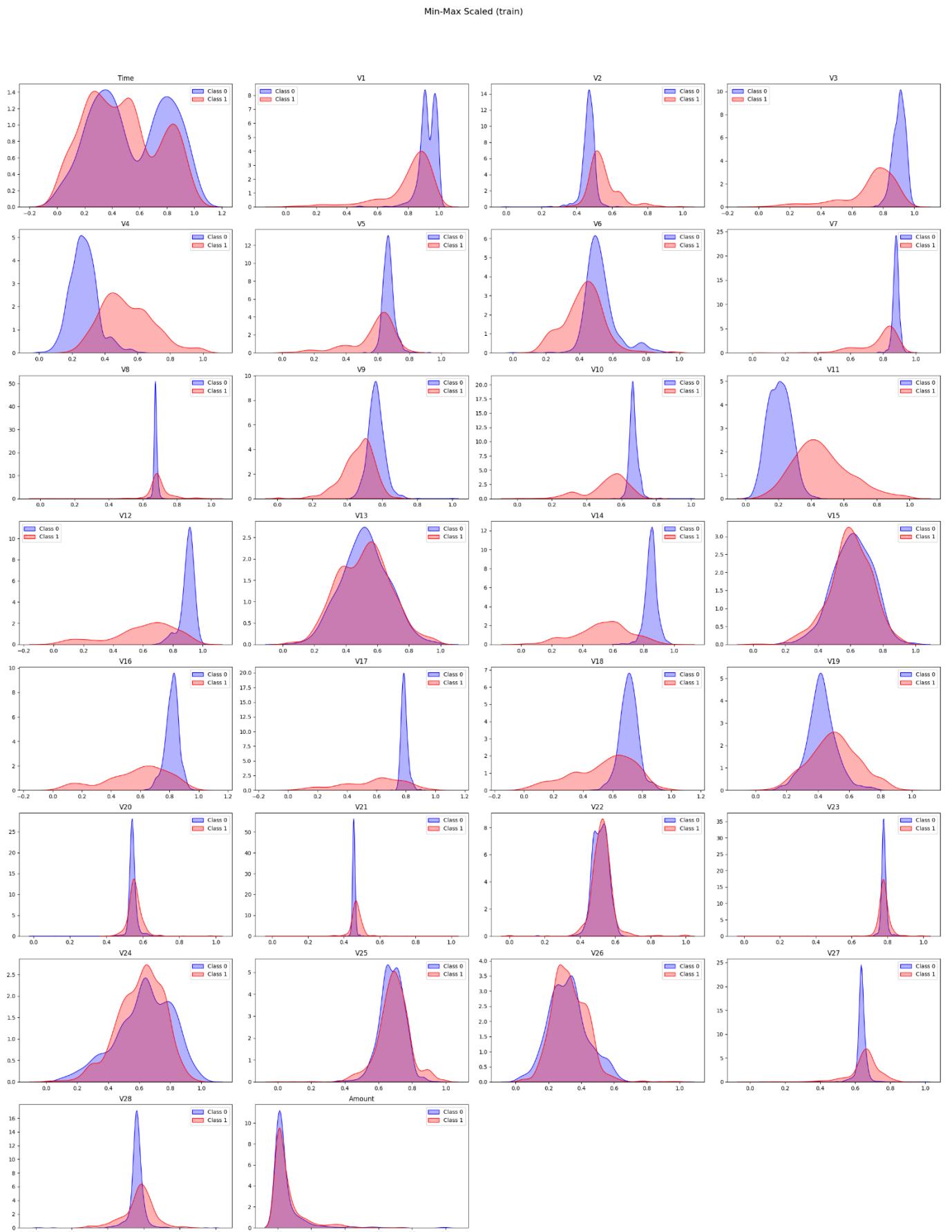
واضح است که تعداد 30 ویژگی برای 2000 داده بسیار زیاد است و ابعاد داده ها را بسیار بالا میبرد. به طوری که حتی ممکن است دچار Curse of dimensionality شویم! برای اجتناب از این قضیه باید یکسری ویژگی ها را حذف یا با هم ترکیب کنیم تا ابعاد را پایین تر بیاوریم. به این منظور در ابتدا هیستوگرام کلاس بر حسب تک تک ویژگی ها را plot میکنیم. این نمودار ها در شکل های 6 و 7 در صفحات بعد مشخص شده اند(اگر روی نمودار ها زوم کنید واضح هستند). با توجه به این نمودار ها میتوان تشخیص داد که کدام ویژگی ها جداسازی خوبی دارند و کدام ویژگی ها جدا سازی خوبی ندارند.

برای مثال در این نمودار ها می توانیم ببینیم که ویژگی Amount عمل جداسازی روی داده ها انجام نمیدهد ولی ویژگی مثل V14 یا V12 جدا سازی بسیار خوبی روی داده ها انجام میدهد. یا برای مثال ویژگی های V16 تا V18 همبستگی بسیار زیادی باهم دارند و عمل جاوی اطلاعات یکسان هستند. به طور کلی باید بررسی کدام ویژگی ها با یکدیگر کمترین همبستگی و با لیل نهایی بیشترین همبستگی را دارند. این ویژگی ها ویژگی های خوب هستند و در ادامه قصد داریم آنها را پیدا کنیم.

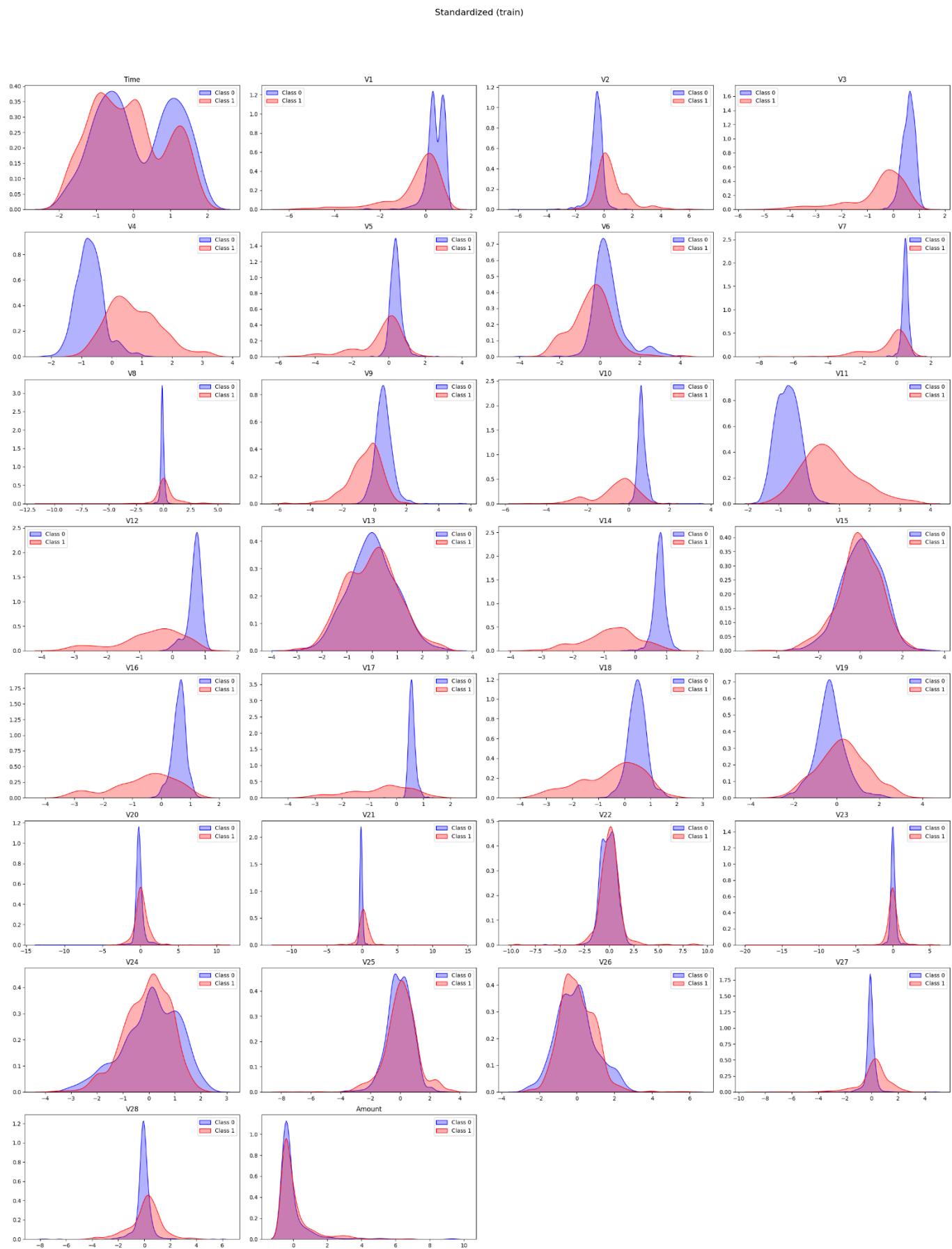
پیش از گذر به مرحله انتخاب بهترین ویژگی ها، میتوانیم از نمودار های شکل های 6 و 7 چند چیز دیگر را تحلیل کنیم! میدانیم که مجموعه داده های آموزشی ما شامل داده های مصنوعی است ولی مجموعه داده های تست از داده های حقیقی تشکیل شده. می خواهیم ببینیم واقعاً چقدر داده های مصنوعی شبیه به داده های حقیقی تولید شده اند. اگر واقعاً داده های مصنوعی بسیار شبیه به داده های حقیقی باشند اصولاً باید هیستوگرام کلاس ها بر حسب ویژگی ها در داده های مصنوعی و حقیقی شبیه هم باشد. بدین منظور هیستوگرام ها را برای داده های تست نیز در شکل های 8 و 9رسم میکنیم تا به صورت حدودی کیفیت تولید داده های مصنوعی را بررسی کنیم.( واضح است که داده های تست در اکثر موقع در دسترس ما نیستند که بتوانیم این گام را انجام دهیم ولی در اینجا این گام را صرفا برای ارزیابی کیفیت داده های مصنوعی انجام دادیم و واضح است که داده های تست به ترین نشت نکرده)

با بررسی این نمودار ها میتوان پی برد این توزیع ها بسیار شبیه هم هستند و کیفیت داده های مصنوعی که در مجموعه داده های train استفاده شده قابل قبول و خوب است.

در گام بعدی تلاش میکنیم ویژگی هایی که حاوی اطلاعات زیادی نیستند را حذف کنیم و ویژگی های بهتر را نگه داریم.

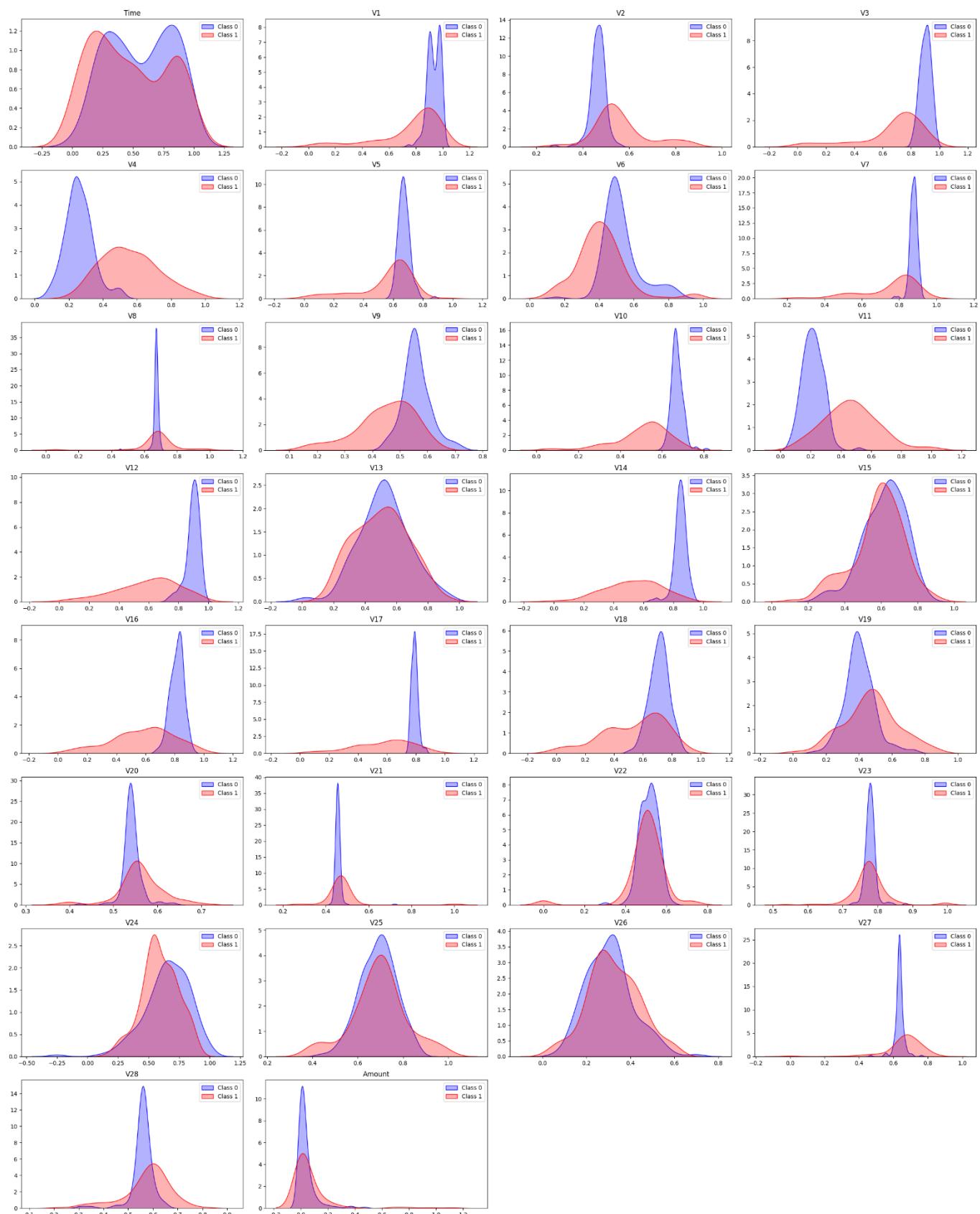


شکل 6 : هیستوگرام ویژگی های داده های train با اسکیل  $\min$ ,  $\max$

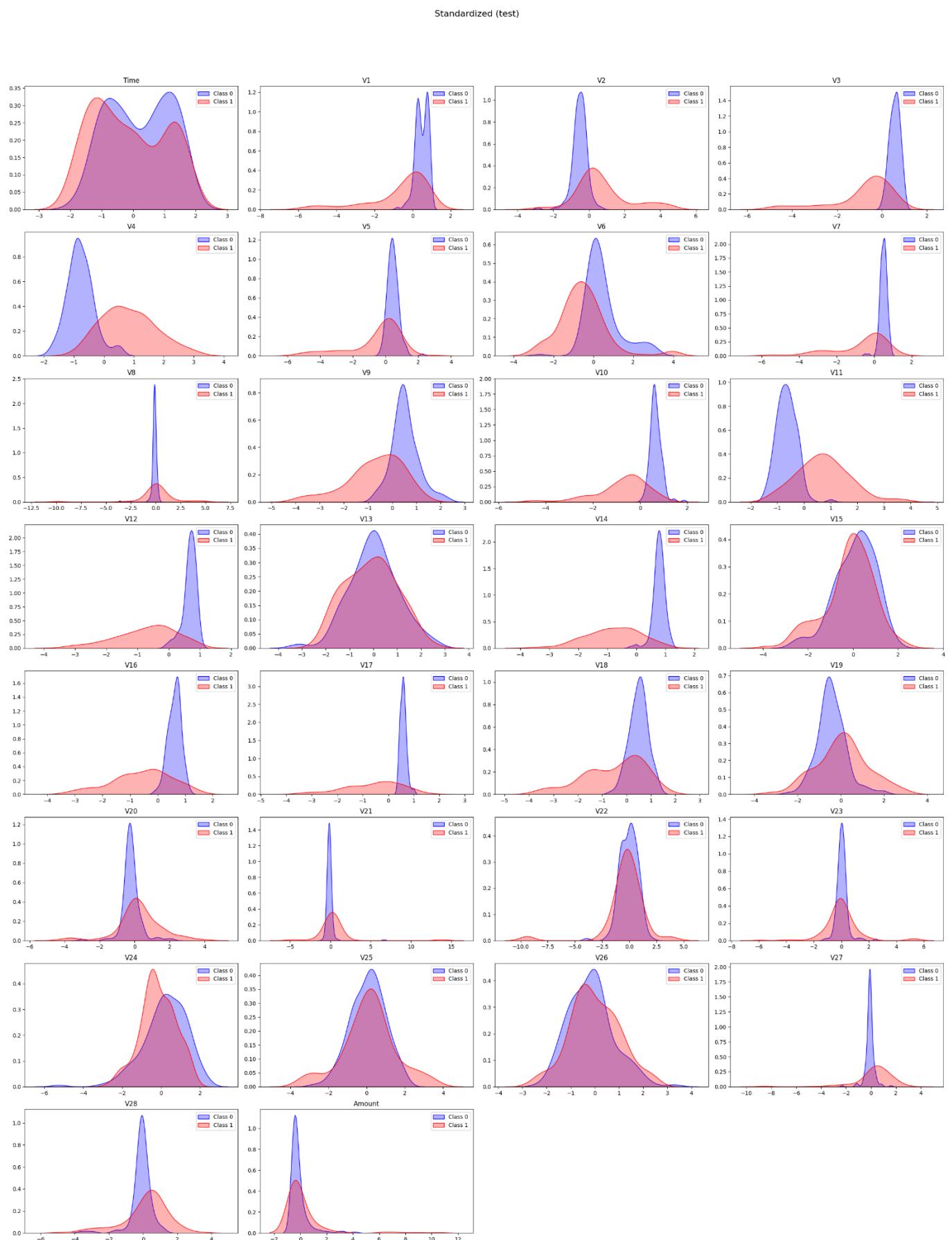


شکل 7 : هیستوگرام ویژگی های داده های train استاندارد شده

Min-Max Scaled (test)



شکل 8 : هیستوگرام ویژگی های داده های **test** با اسکیل **min , max**

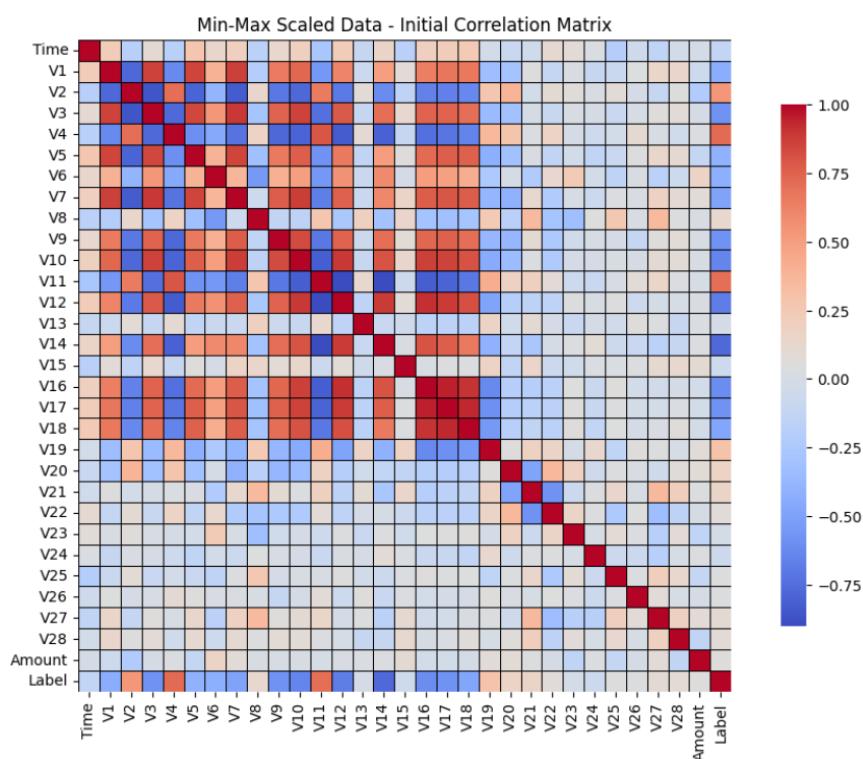


شکل ۹ : هیستوگرام ویژگی های داده های **test** استاندارد شده

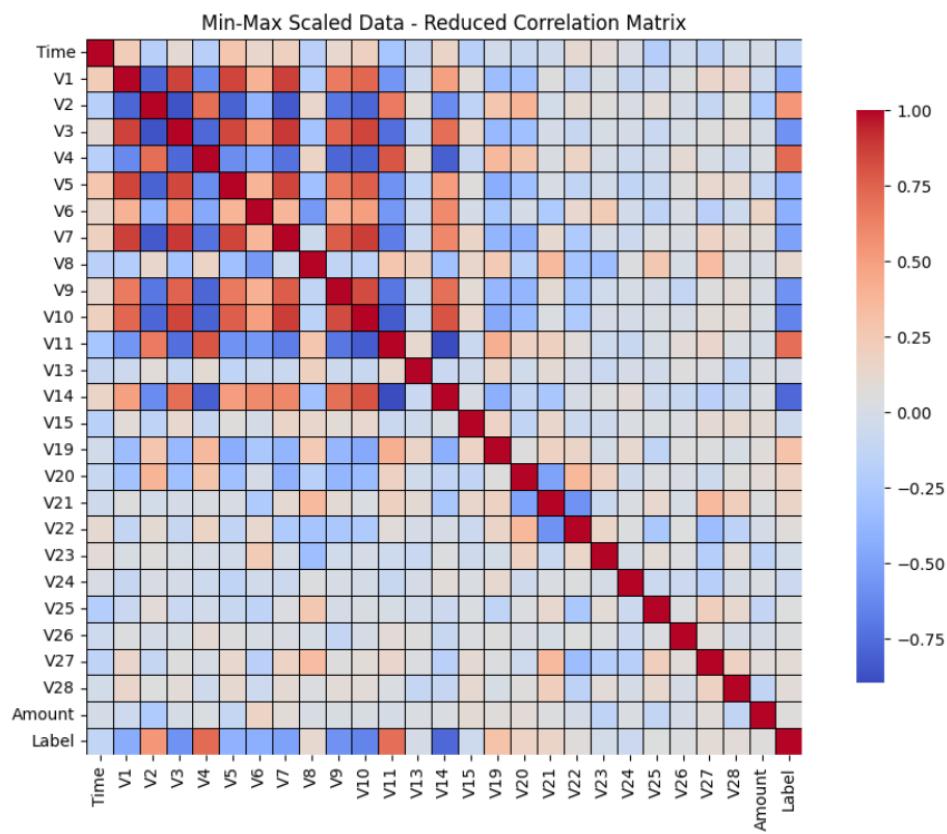
پس از بررسی چشمی که انجام دادیم، می خواهیم در این مرحله ویژگی های خوب را نگهداریم و ویژگی هایی که اطلاعات زیادی ندارند را حذف کنیم.

در ابتدا از Correlation Matrix استفاده میکنیم. ابتدا همبستگی تمام ویژگی ها با یکدیگر و با خروجی را بدست می آوریم و سپس از بین ویژگی هایی که مقدار همبستگی آنها بیشتر از 0.9 است ویژگی که بیشترین همبستگی با خروجی دارد را نگه میداریم.

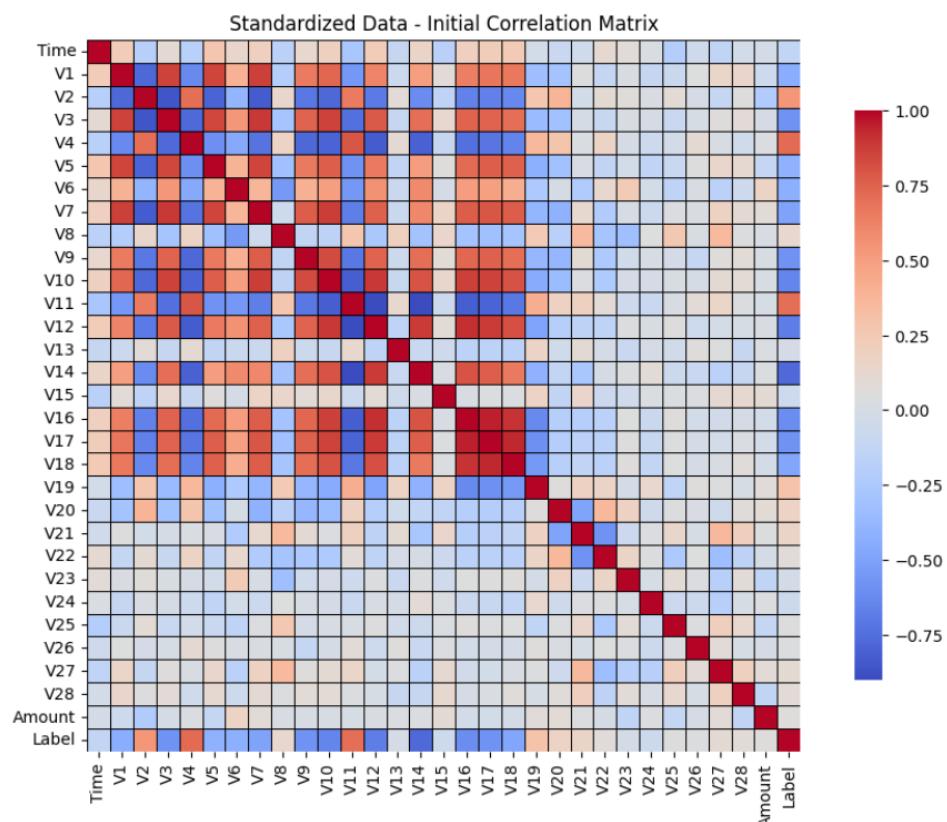
تصاویر مربوطه در شکل های 10 تا 13 آورده شده و همانطور که مشخص است ویژگی های V12,V16,V17,V18 از هر دو مجموعه داده استاندارد و نرمال حذف شده است.



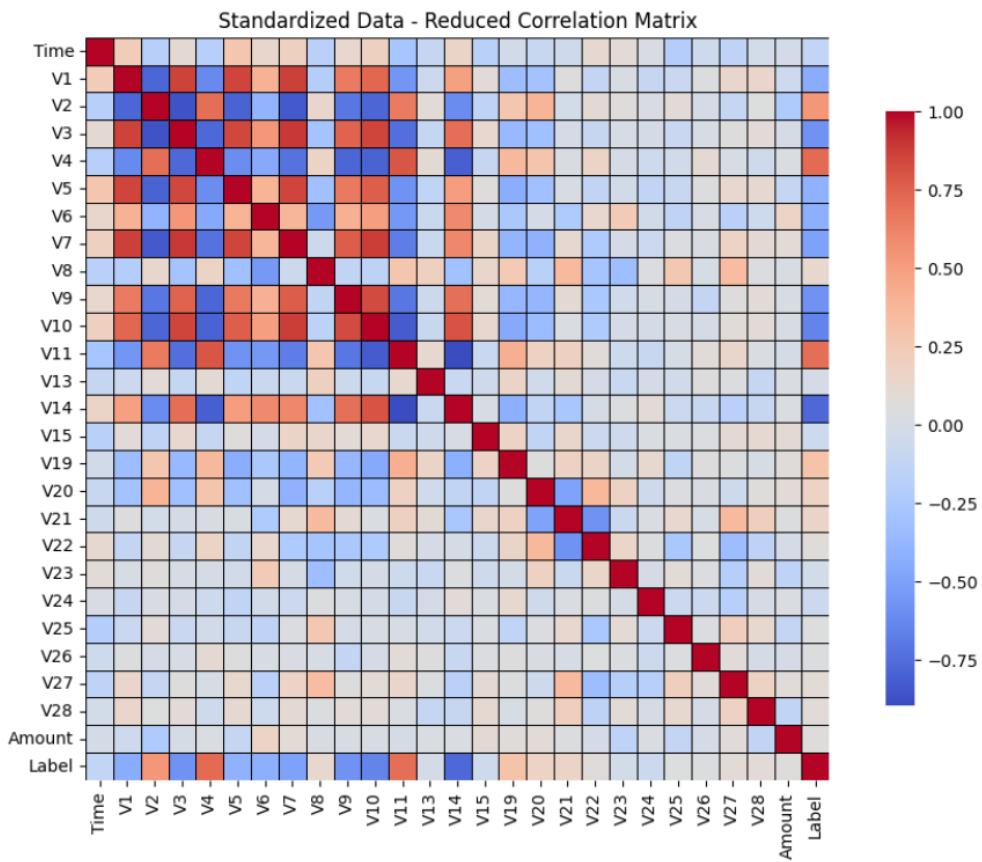
شکل 10 : ماتریس همبستگی داده ها با اسکیل **min , max** قبل از حذف ویژگی ها



شکل 11: ماتریس همبستگی داده ها با اسکیل  $\min$ ,  $\max$  بعد از حذف ویژگی ها



شکل 12: ماتریس همبستگی داده های استاندارد قبل از حذف ویژگی ها

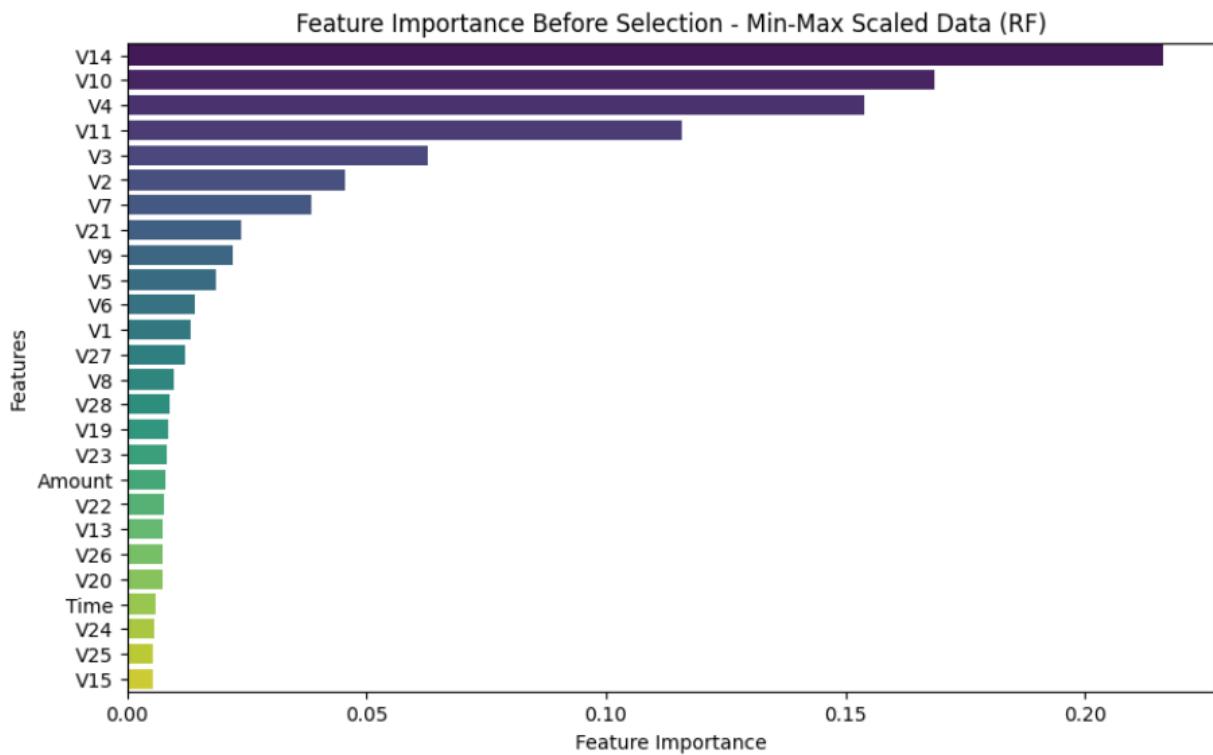


شکل 13: ماتریس همبستگی داده های استاندارد بعد از حذف ویژگی ها

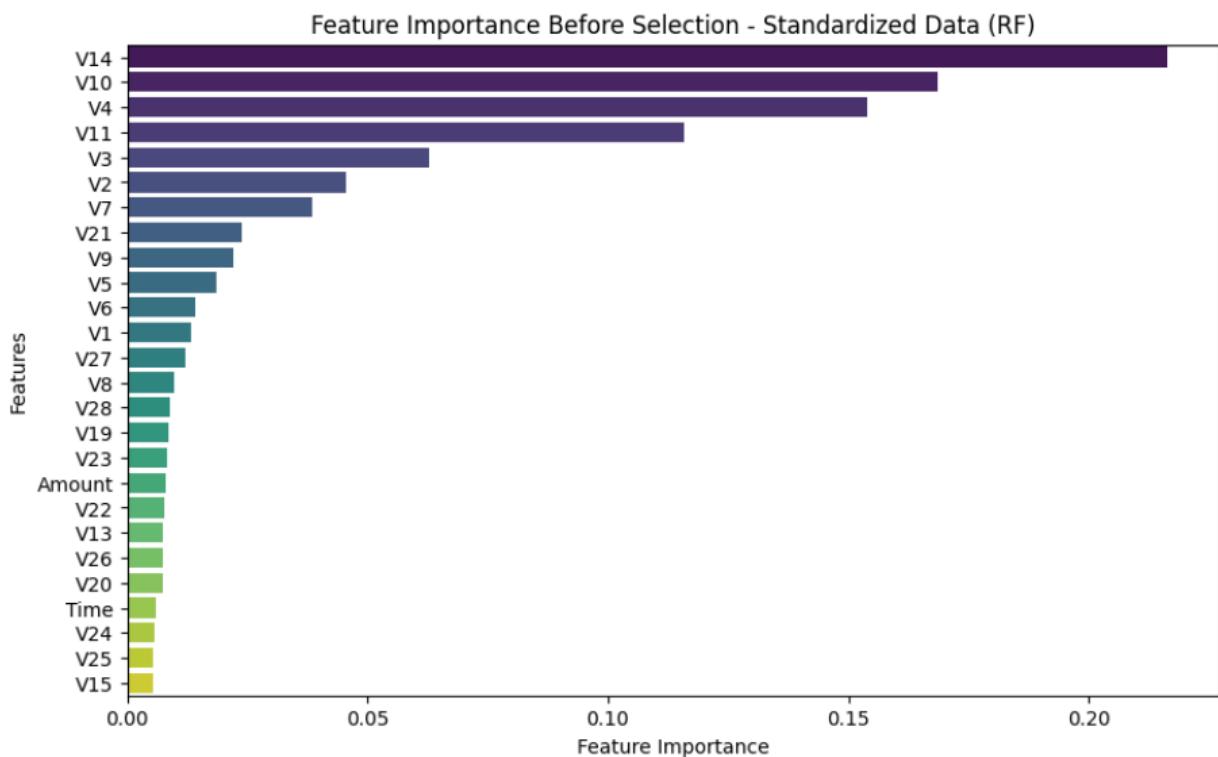
پس این بخش می خواهیم از Random Forest برای انتخاب ویژگی های بهتر استفاده کنیم. در این روش چندین درخت تصمیم را به صورت رندوم آموزش میدهیم و سپس ویژگی هایی که نزدیک تر به ریشه استفاده شدند را نگه میداریم می کنیم. در این روش یک امتیاز به هر ویژگی نسبت میدهیم که نشانگر این است که ویژگی در درخت های بیشتری نزدیک به ریشه انتخاب شده است. حالا می توانیم امتیاز ها را به صورت نمودار میله ای plot کنیم. این نمودار ها در شکل های 14 تا 17 آورده شدند.

با توجه به این نمودار ها 20 ویژگی اول را انتخاب می کنیم و 6 ویژگی دیگر را حذف میکیم. همانطور که مشخص است V26,V20,Time,V24,V25,V15 از لیست ویژگی های ما حذف شدند. و بقیه ویژگی ها باقی مانند.

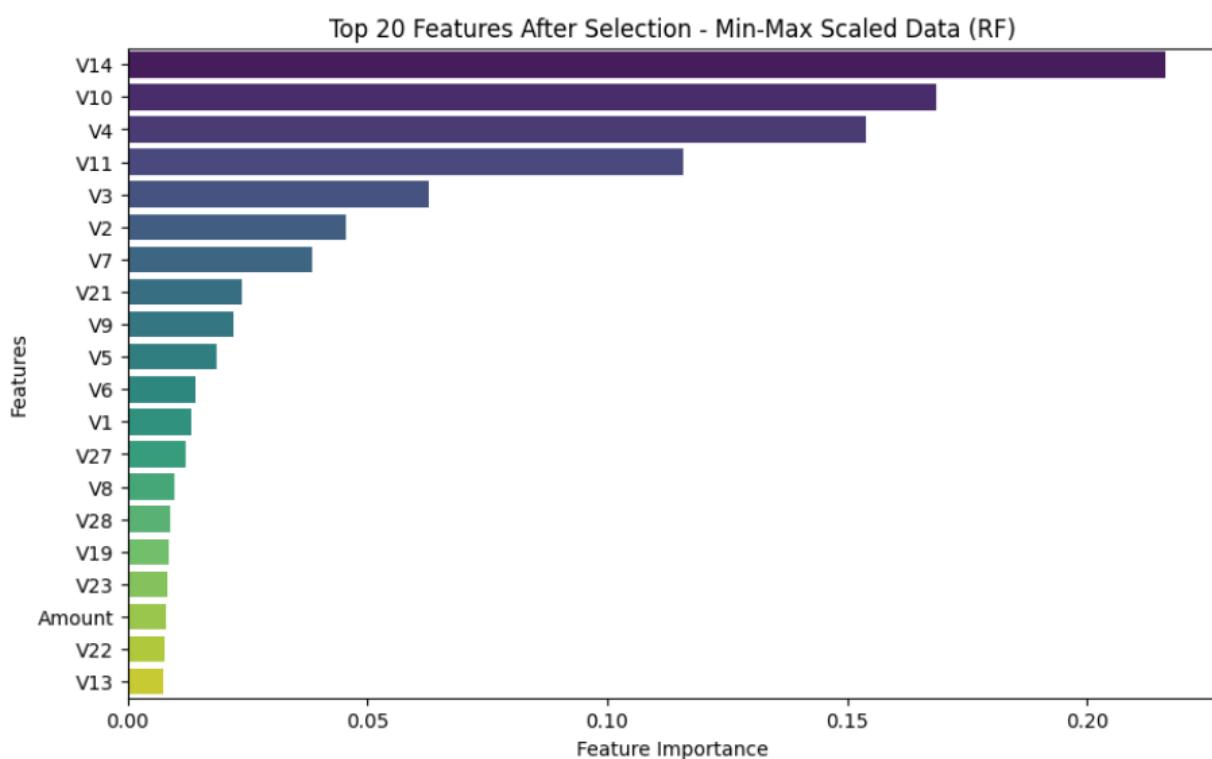
حالا داده ها را پیش پرداش کردیم و ویژگی های غیر ضروری را حذف کردیم. اگه در لایه مخفی اول 64 نرون داشته باشیم با حذف همین 10 ویژگی توانستیم 640 پارامتر را کم کنیم. بنابراین می توانیم مدل MLP خود را به نحو خوبی train کنیم.



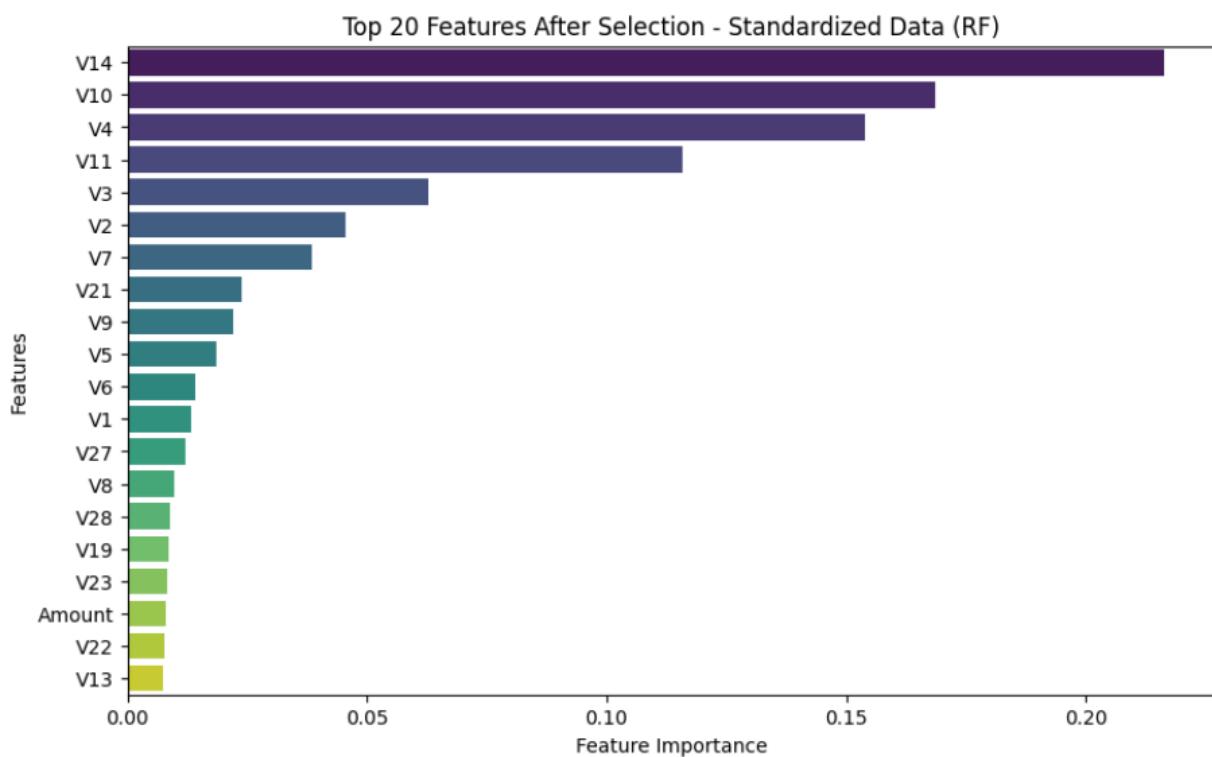
شکل 14: امتیاز ویژگی ها بر اساس **Random forest** برای داده های نرمالایز



شکل 15: امتیاز ویژگی ها بر اساس **Random forest** برای داده های استاندارد



شکل 16: 20 ویژگی برتر انتخاب شده بر اساس **Random forest** برای داده های نرمالایز



شکل 17: 20 ویژگی برتر انتخاب شده بر اساس **Random forest** برای داده های استاندارد

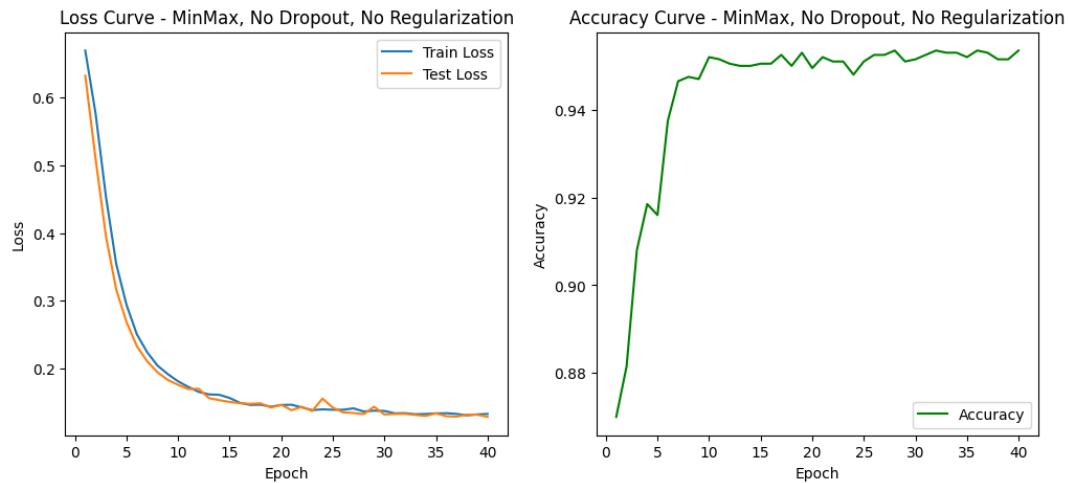
### 3-1. طراحی و پیاده‌سازی یک شبکه MLP ساده

در این بخش می‌خواهیم یک شبکه MLP ساده با یک لایه مخفی را آموزش دهیم و سپس تست کنیم. تنظیماتی مانند ران شدن کد روی GPU، تنظیم batch size و تبدیل داده‌ها به فرم tensor را انجام میدهیم و معماری شبکه با 64 نورون در لایه مخفی، 20 ورودی و یک نورون در لایه خروجی را مطابق با خواسته سوال طراحی می‌کنیم.

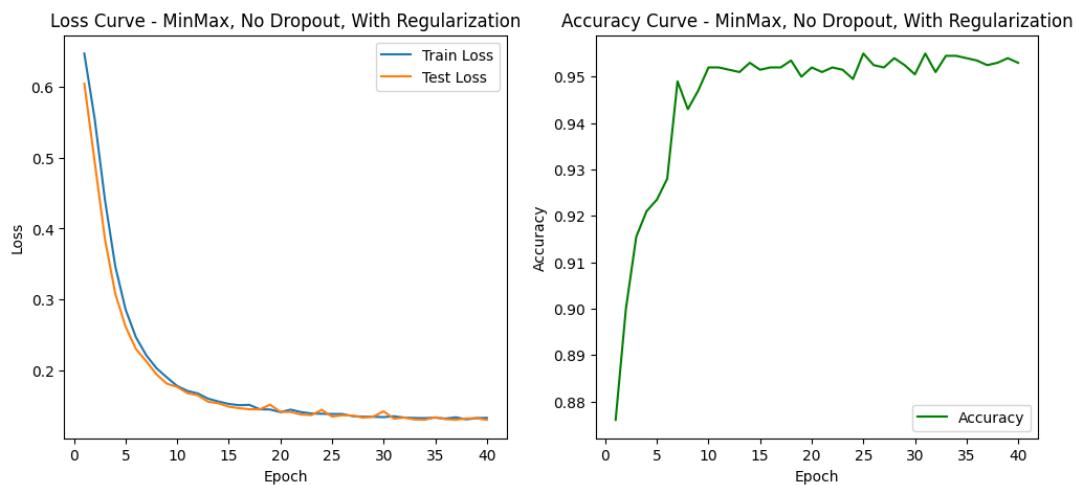
مطابق زیر، لیستی از تنظیمات مورد نیاز برای خواسته سوال را طراحی می‌کنیم و مدل را در تمامی این حالات آموزش می‌دهیم:

```
configs = [
    (train_loader_minmax, False, False, "MinMax, No Dropout, No Regularization"),
    (train_loader_minmax, False, True, "MinMax, No Dropout, With Regularization"),
    (train_loader_minmax, True, False, "MinMax, With Dropout, No Regularization"),
    (train_loader_minmax, True, True, "MinMax, With Dropout, With Regularization"),
    (train_loader_standardized, False, False, "Standardized, No Dropout, No Regularization"),
    (train_loader_standardized, False, True, "Standardized, No Dropout, With Regularization"),
    (train_loader_standardized, True, False, "Standardized, With Dropout, No Regularization"),
    (train_loader_standardized, True, True, "Standardized, With Dropout, With Regularization")
]
```

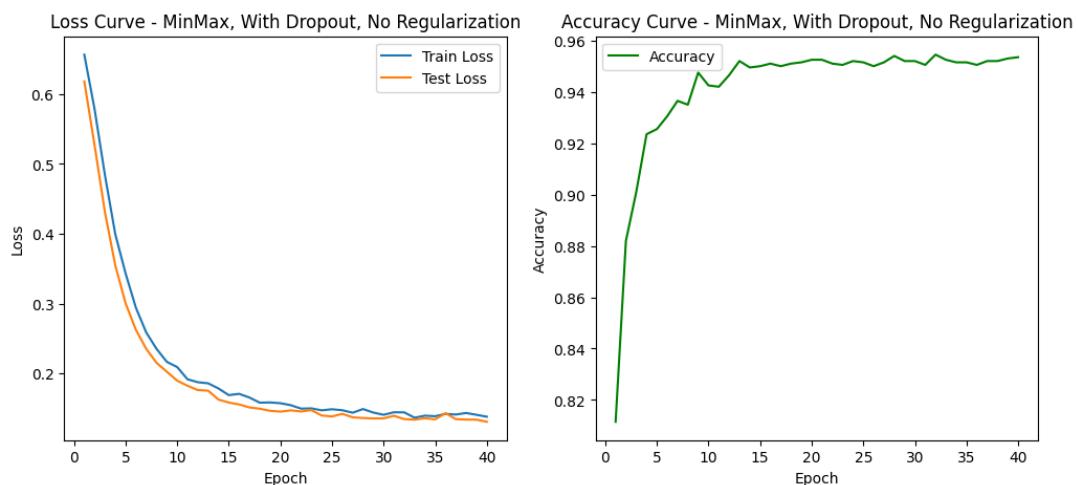
همینطور در حین آموزش بهترین مدل‌ها را که دقیق‌ترین داشتند در یک لیست ذخیره می‌کنیم که بعداً آنها را تست کنیم. نمودارهای ارزیابی مدل‌ها در شکل‌های زیر آورده شده:



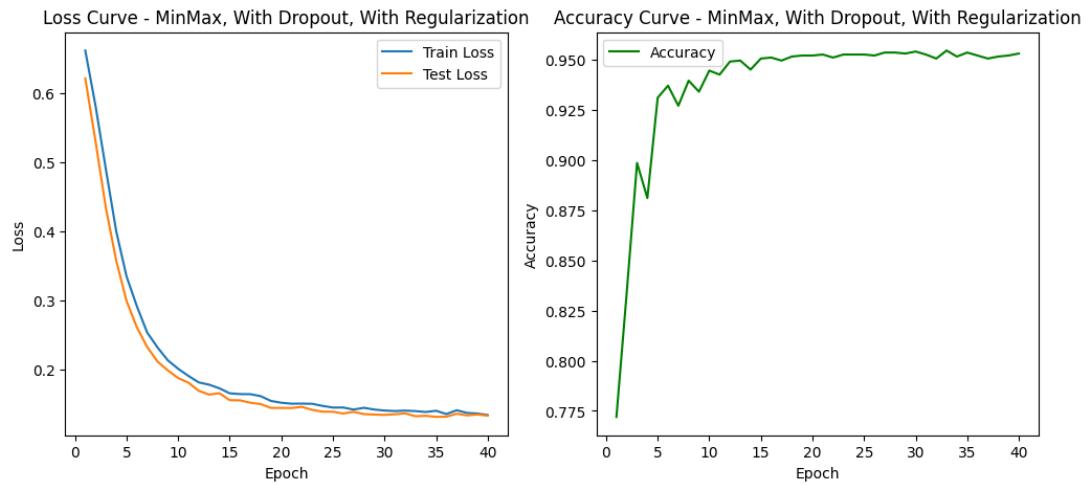
شکل 18: آموزش مدل روی داده های نرمالایز بدون regularization و dropout



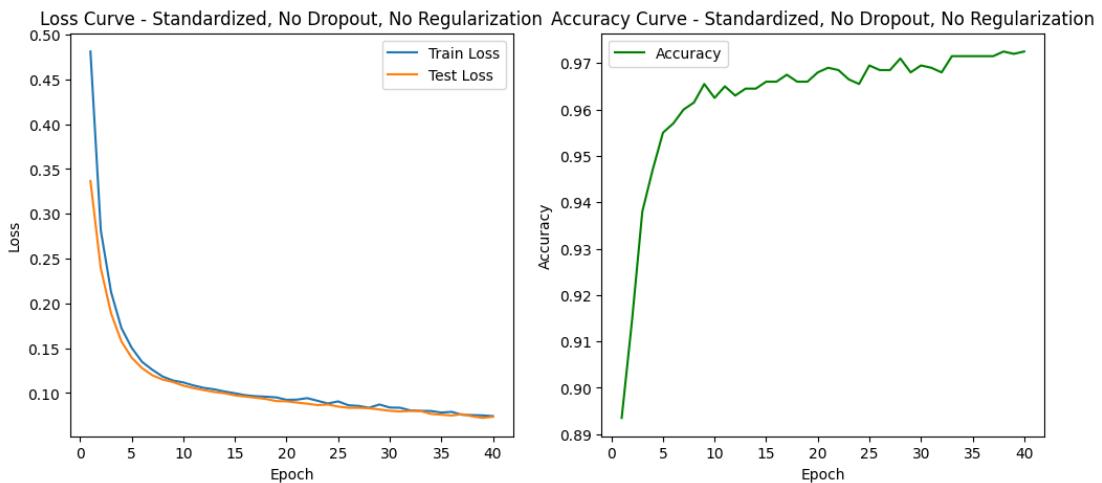
شکل 19: آموزش مدل روی داده های نرمالایز بدون regularization و dropout با regularization



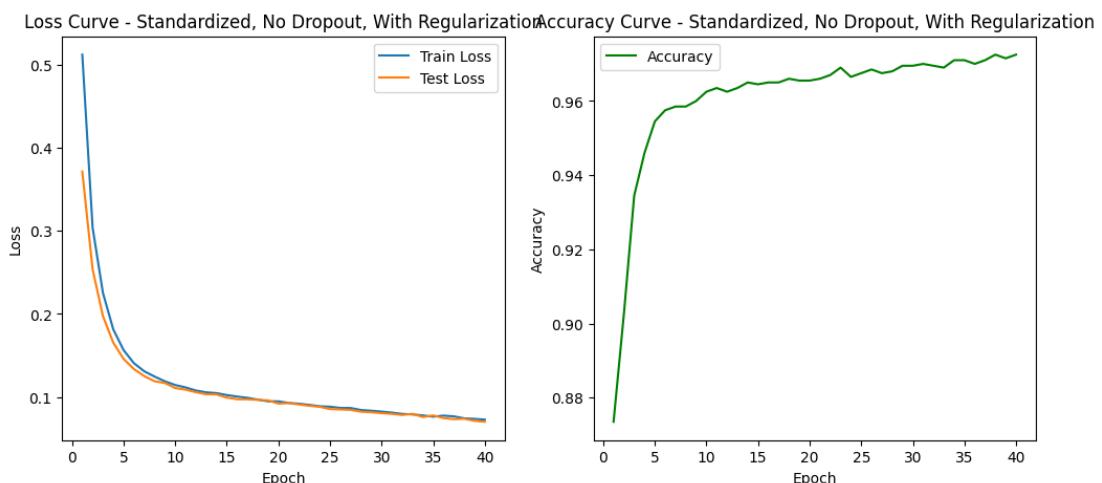
شکل 20: آموزش مدل روی داده های نرمالایز با regularization و dropout



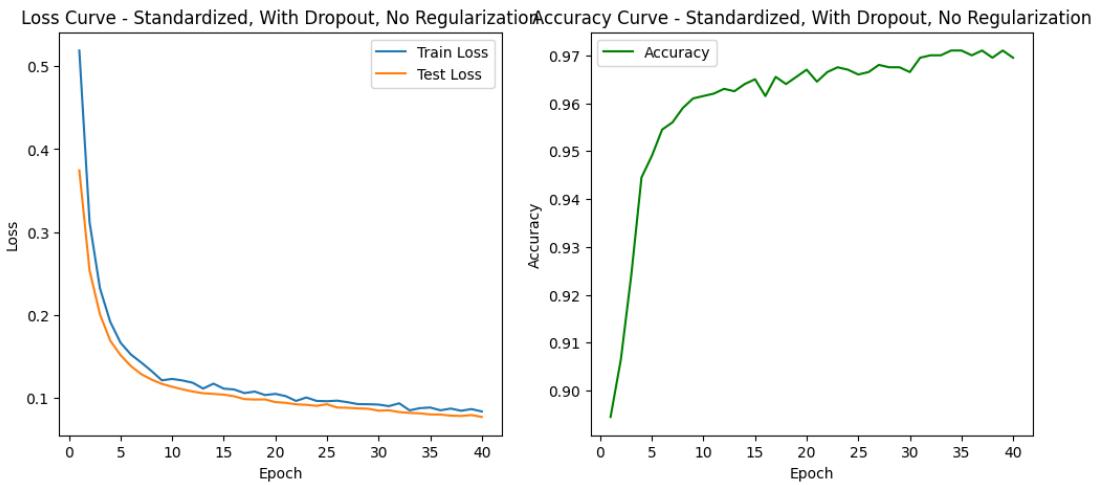
شکل 21: آموزش مدل روی داده های نرمالایز با regularization و dropout



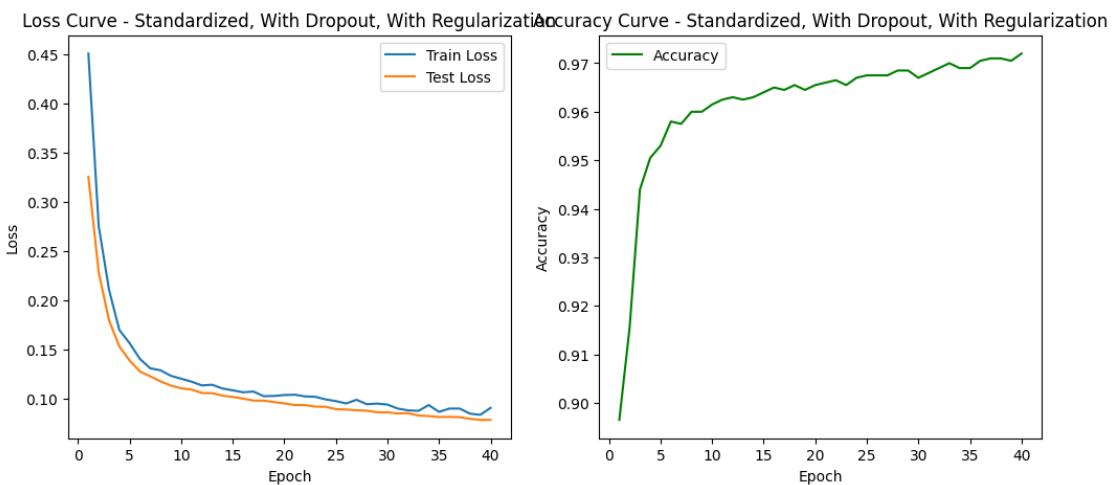
شکل 22: آموزش مدل روی داده های استاندارد بدون regularization و dropout



شکل 23: آموزش مدل روی داده های استاندارد بدون regularization و dropout



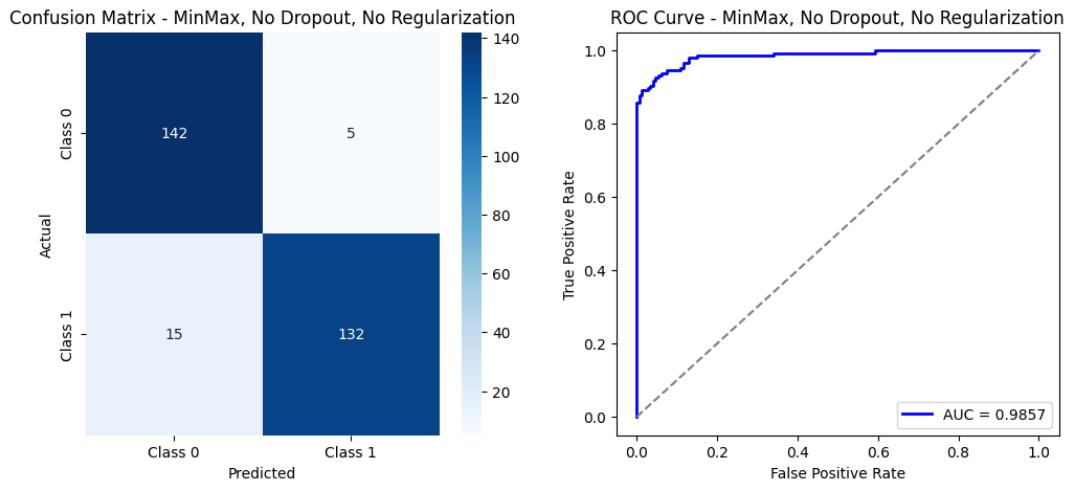
شکل 24: آموزش مدل روی داده های استاندارد با regularization و بدون dropout



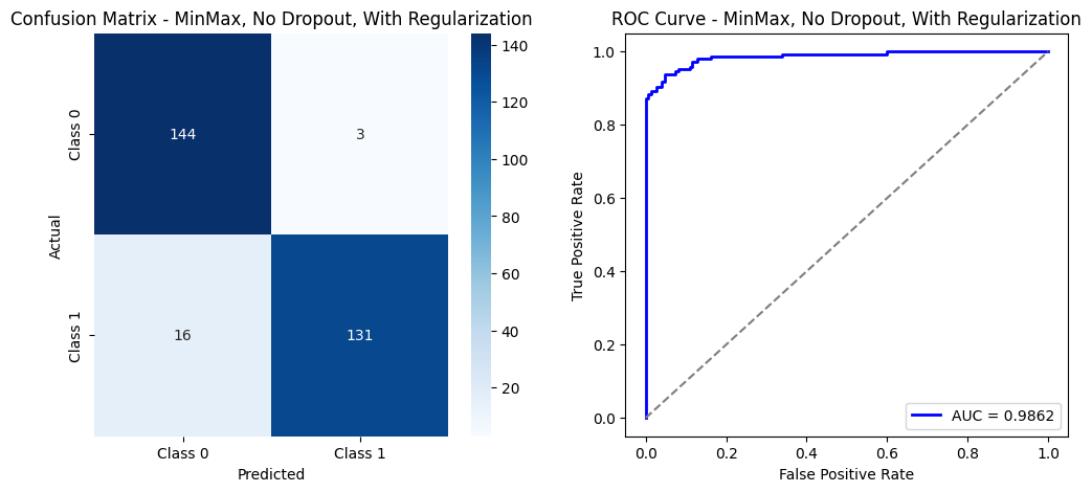
شکل 25: آموزش مدل روی داده های استاندارد با regularization و با dropout

در همین نمودار ها به طور چشمی می توانیم ببینیم که انگار مدل با داده های استاندارد بهتر کار میکند و همچنین regularization و dropout می توانند تأثیرات مختلفی رو مدل بگذارند. مثلا در همین نمودار های بالا مواردی وجود دارند که regularization و dropout باعث بهبود مدل شده اند و مواردی نیز وجود دارند که باعث کاهش کیفیت مدل شده اند. به هر حال در داده های تست بهتر می توان این موارد را تحلیل کرد.

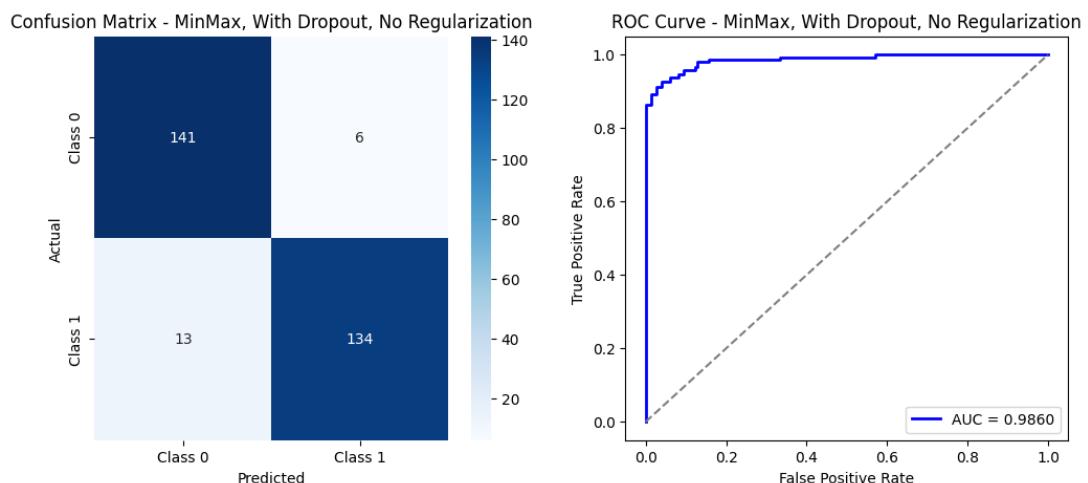
همه مدل های بالا با داده های تست، تست می کنیم و موارد و متريک های خواسته شده را گزارش می دهیم. نهايتا در يك جدول، تمام موارد را در کنار هم ميگذارييم تا باهم مقاييسه کنیم.



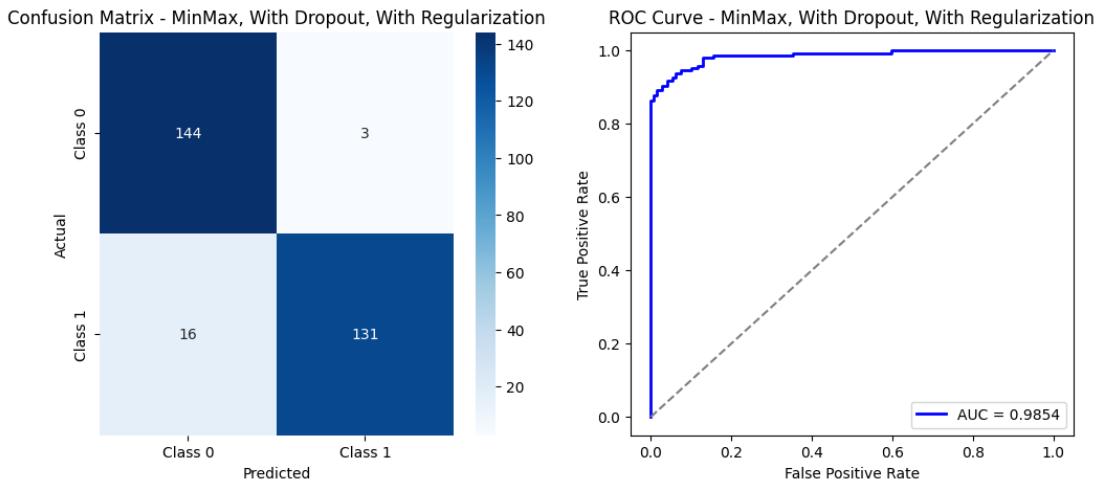
شکل 26: تست مدل روی داده های نرمالایز بدون dropout و بدون regularization



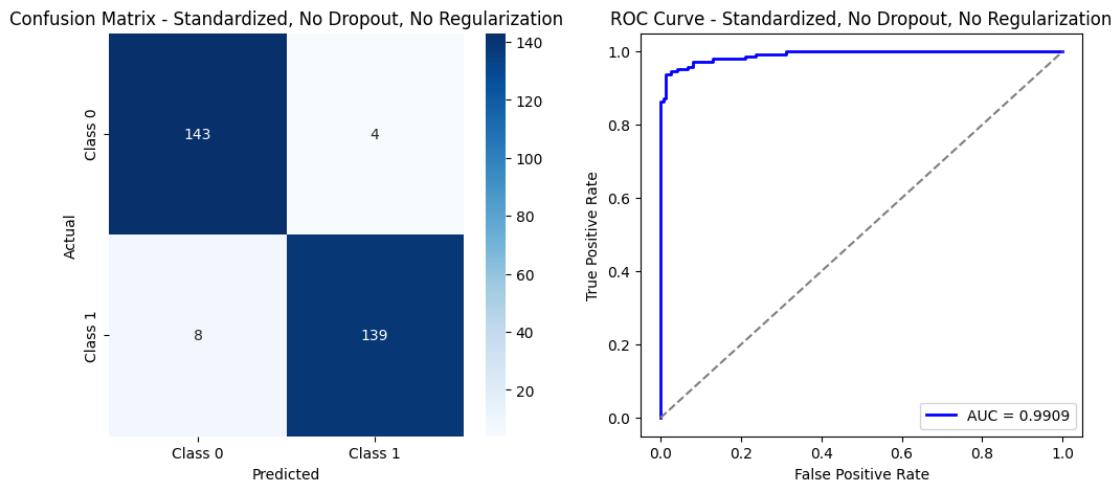
شکل 27: تست مدل روی داده های نرمالایز بدون dropout و با regularization



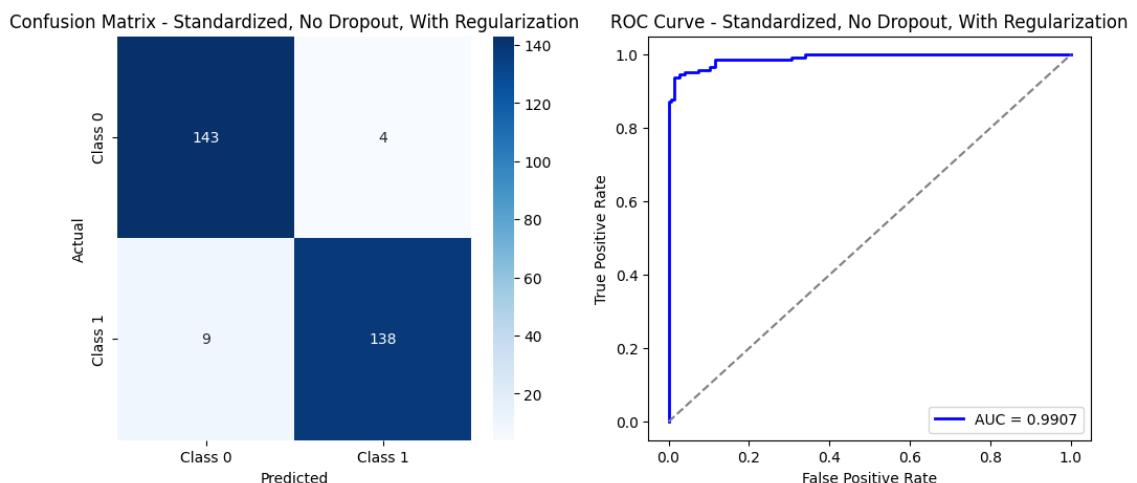
شکل 28: تست مدل روی داده های نرمالایز با dropout و بدون regularization



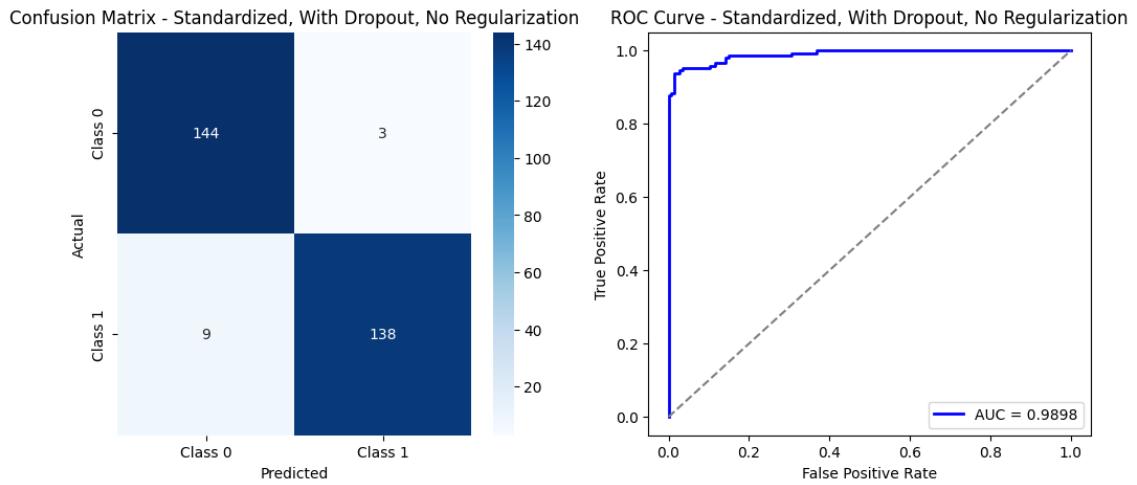
شکل 29: تست مدل روی داده های نرمالایز با dropout و regularization



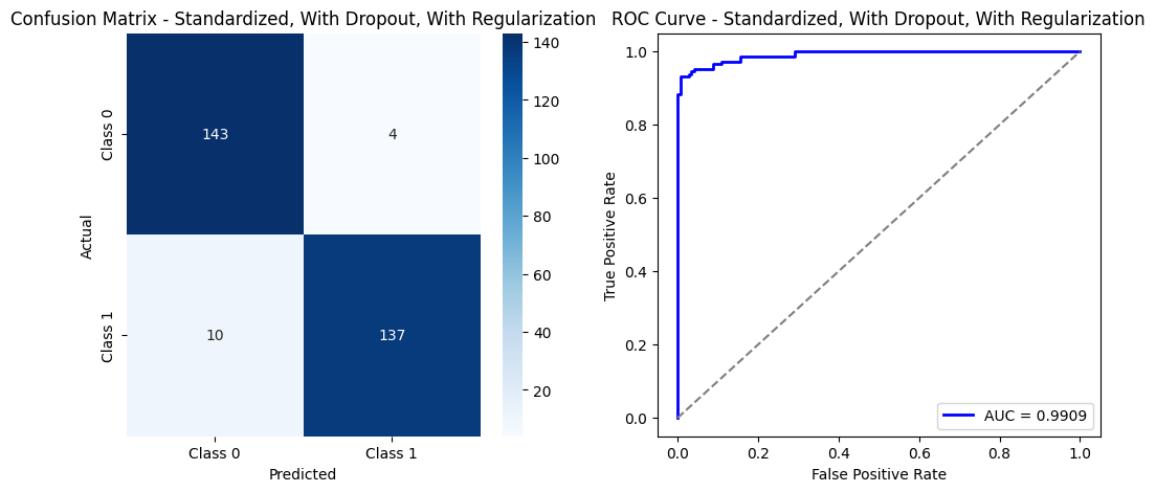
شکل 30: تست مدل روی داده های استاندارد بدون dropout و بدون regularization



شکل 31: تست مدل روی داده های استاندارد بدون dropout و با regularization



شکل 32: تست مدل روی داده های استاندارد با dropout و بدون regularization



شکل 33: تست مدل روی داده های استاندارد با dropout و با regularization

جدول 1 : مقایسه تمام متریک های کیفیت مدل برای تمام مدل ها با یک لایه مخفی

| Model   | Accuracy | Precision | Recall   | F1-Score | AUC Score |
|---|----------|-----------|----------|----------|-----------|
| Standardized, No Dropout, No Regularization   | 0.959184 | 0.972028  | 0.945578 | 0.958621 | 0.990930  |
| Standardized, With Dropout, No Regularization | 0.959184 | 0.978723  | 0.938776 | 0.958333 | 0.989773  |
| Standardized, No Dropout, With Regularization | 0.955782 | 0.971831  | 0.938776 | 0.955017 | 0.990698  |

|  |          |          |          |          |          |
|--|----------|----------|----------|----------|----------|
| <b>Standardized, With Dropout, With Regularization</b> | 0.952381 | 0.971631 | 0.931973 | 0.951389 | 0.990930 |
| <b>MinMax, With Dropout, No Regularization</b>         | 0.935374 | 0.957143 | 0.911565 | 0.933798 | 0.986024 |
| <b>MinMax, No Dropout, With Regularization</b>         | 0.935374 | 0.977612 | 0.891156 | 0.932384 | 0.986163 |
| <b>MinMax, With Dropout, With Regularization</b>       | 0.935374 | 0.977612 | 0.891156 | 0.932384 | 0.985423 |
| <b>MinMax, No Dropout, No Regularization</b>           | 0.931973 | 0.963504 | 0.897959 | 0.929577 | 0.985654 |

اولین نکته ای که در این جدول جلب توجه میکند این است که مدل های شبکه عصبی MLP روی داده های استاندارد به وضوح بهتر عمل میکنند و روی داده های نرمالایز به نسبت داده های استاندارد عملکرد خوبی ندارند.

نکته دیگر اختلاف میزان Accuracy در داده های تست و ترین است. در داده های ترین دیدیم که میزان Accuracy حتی تا حدود 97 % هم بالا رفت ولی در داده های ترین از حدود 96 بیشتر نمیشود.

درباره تحلیل confusion matrix ها می توان گفت که مدل همواره کلاس 0 را بهتر از کلاس 1 پیشبینی میکند. علت آن نیز این است که داده های کلاس 0 همگی داده های واقعی هستند ولی تقریبا 50 % داده های کلاس 1 مصنوعی هستند و واضح است که مدلی که با داده های مصنوعی آموزش دیده نمی تواند به خوبی داده های واقعی را پیشبینی کند.

آن طور که بنظر میرسد تاثیر dropout و regularization تا مقداری تصادفی است. برای مثال در داده های نرمالایز شده regularization و dropout آنچنان تاثیری روی کیفیت مدل نگذاشته اند و در داده های استاندارد نیز اگر فقط یکی از آنها اعمال شده باشد مدل عملکرد بهتری از قبل دارد ولی اگر هر دو آنها اعمال شده باشند آنچنان تاثیری روی کیفیت مدل ندارند.

در این مثال با توجه به اینکه تعادل کلاس ها را در ابتدا برابر کردیم می توان گفت که Accuracy (تعداد تمام نمونه هایی که درست پیشبینی شده اند تقسیم بر تعداد تمام نمونه ها) معیار مناسبی برای سنجش دقت مدل است ولی اگر تعادل کلاس ها برابر نباشد این معیار مناسب نیست زیرا تاثیر داده های یک کلاس روی این معیار بسیار بالا میرود. در این موارد می توانیم از recall یا Precision استفاده کنیم. به طور عمومی این دو معیار برای هر دو کلاس تعریف می شوند ولی در این سوال ما آنها را برای کلاس مثبت (همان کلاس 1) محاسبه کردیم ایم. Precision کلاس 1 نشان میدهد از بین نمونه هایی که به عنوان کلاس 1 پیشبینی کردیم واقعا چند درصد آنها متعلق به کلاس 1 بودند. در واقع به صورت عمودی روی confusion matrix نسبت تناسب میبینیم. همچنین Recall کلاس 1 نشان میدهد که از بین داده هایی که واقع متعلق به کلاس 1 بودند، چند درصد به درستی در کلاس 1 طبقه بندی شده اند. همانطور که در ماتریس ها میبینیم ما نسبت به Recall بالاتر است. یعنی اگر داده ای را به عنوان کلاس 1 پیشبینی کردیم، به Precision

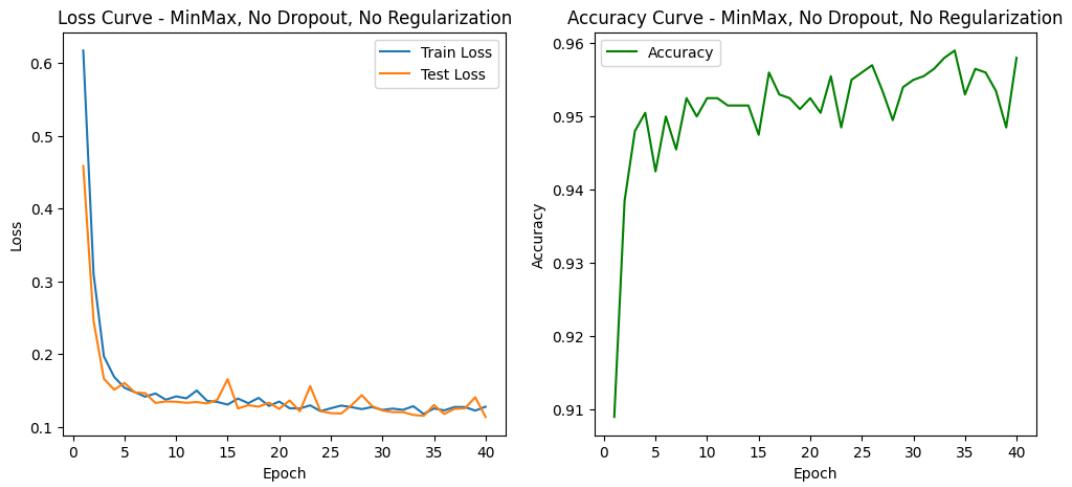
احتمال زیادی آن را درست پیش‌بینی کردیم ولی اگر یک داده از کلاس ۱ به مدل دهیم، احتمال آن که کلاس درست را پیش‌بینی کند کمتر است.

F1 Score نیز میانگین هارمونیک Precision و Recall است که وقتی داده‌ها نا متوازن باشند گزینه بهتری از Accuracy برای سنجش مدل است. همانطور که مشخص است چون تعادل داده‌ها در این مثال وجود داشت، F1 و Accuracy بسیار نزدیک به هم هستند.

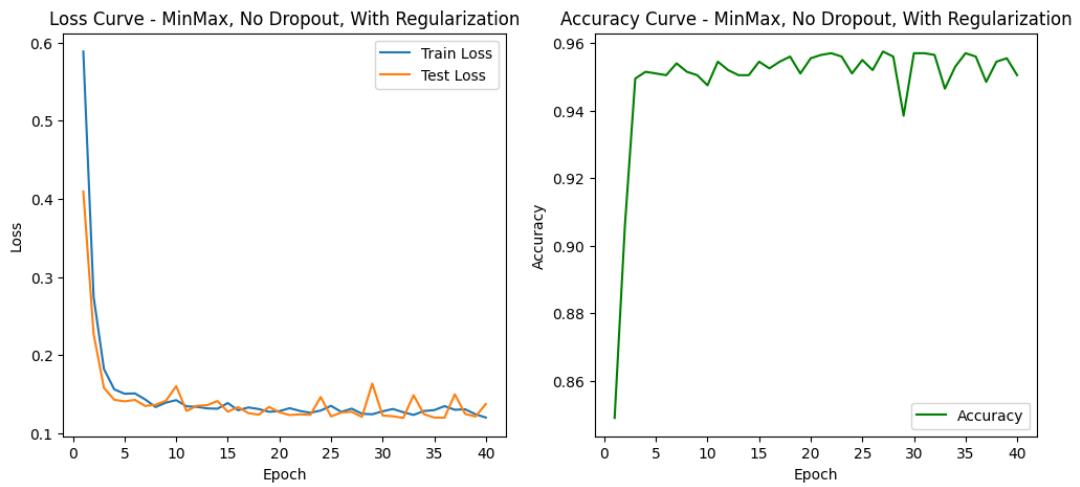
منحنی ROC نیز نرخ داده‌هایی از کلاس ۱ که درست پیش‌بینی شدند را در مقابل نرخ داده‌هایی از کلاس ۱ نشان میدهد که اشتباه پیش‌بینی شدند. برای یک طبقه بند رندوم ROC یک خط مورب است و برای یک طبقه بند ایده آل ROC به صورت یک زاویه ۹۰ درجه است. AUC نیز مساحت زیر آن است. در واقع این معیار قدرت تمایز مدل بین کلاس‌ها را نشان میدهد و در مقابل داده‌های نامتعادل نیز معیار بهتری از Accuracy است.

#### 4-۱. طراحی یک شبکه عصبی تر

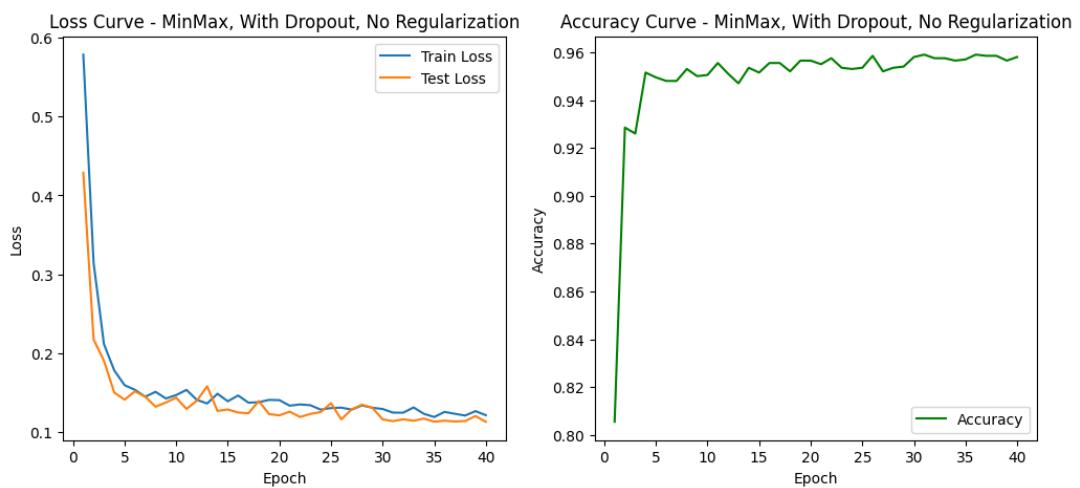
با توجه به بخش قبل و صورت سوال مدل را طراحی، ترین و تست می‌کنیم. نتایج آموزش و ارزیابی مدل مانند بخش قبل در زیر آورده شده.



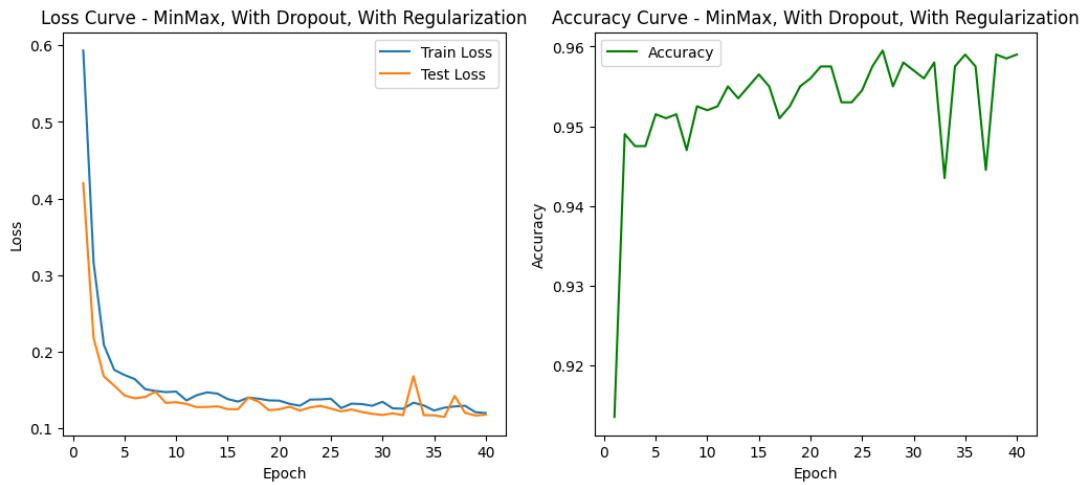
شکل 34: آموزش مدل عمیق تر روی داده های نرمالایز بدون regularization و dropout و بدون regularization و dropout



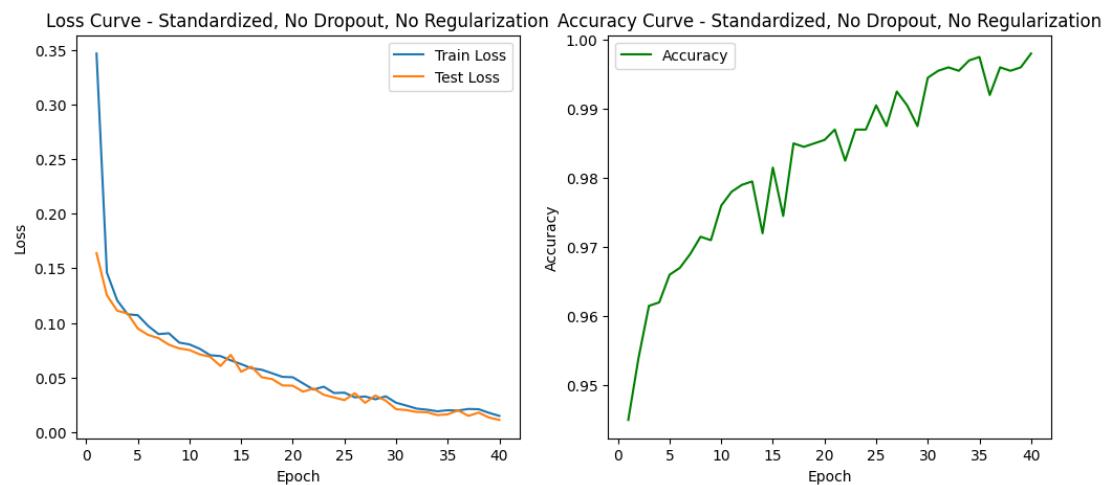
شکل 35: آموزش مدل عمیق تر روی داده های نرمالایز بدون regularization و dropout و با regularization و dropout



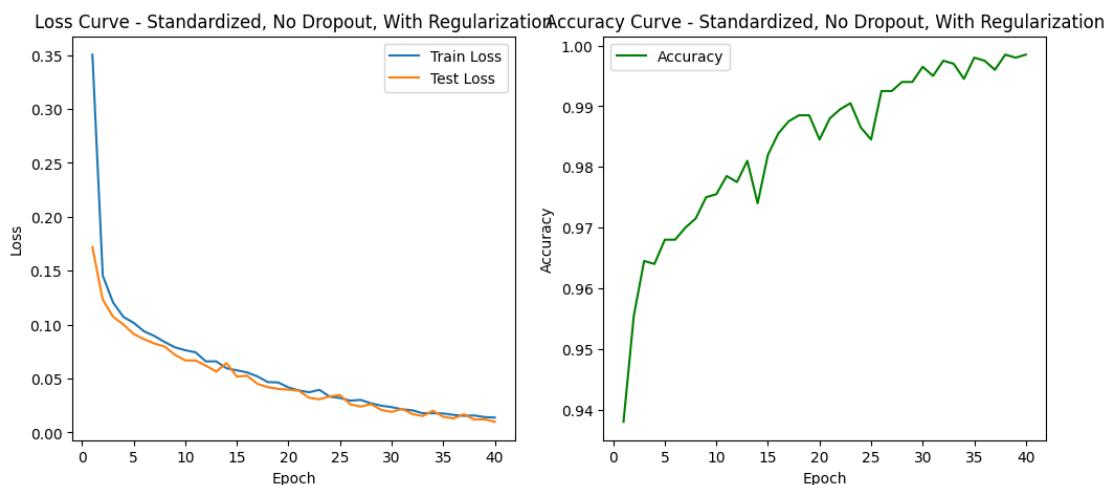
شکل 36: آموزش مدل عمیق تر روی داده های نرمالایز با regularization و dropout و بدون regularization و dropout



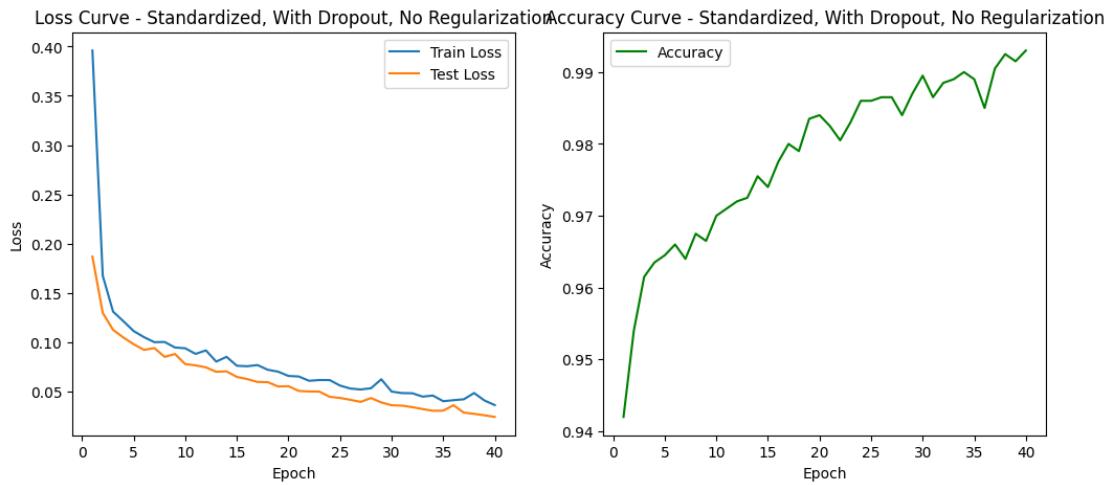
شکل 37: آموزش مدل عمیق تر روی داده های نرمالایز با regularization و با dropout



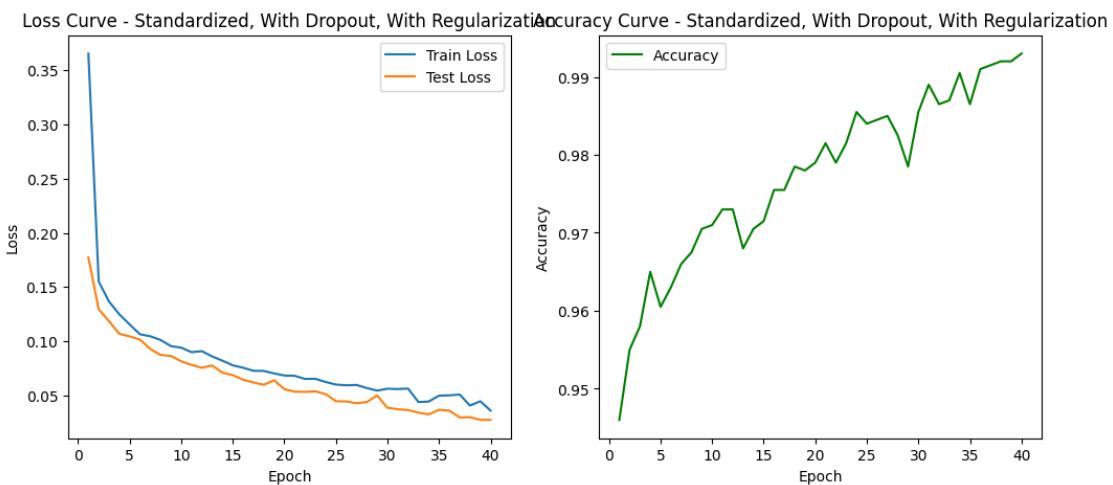
شکل 38: آموزش مدل عمیق تر روی داده های استاندارد بدون regularization و بدون dropout



شکل 39: آموزش مدل عمیق تر روی داده های استاندارد بدون regularization و با dropout



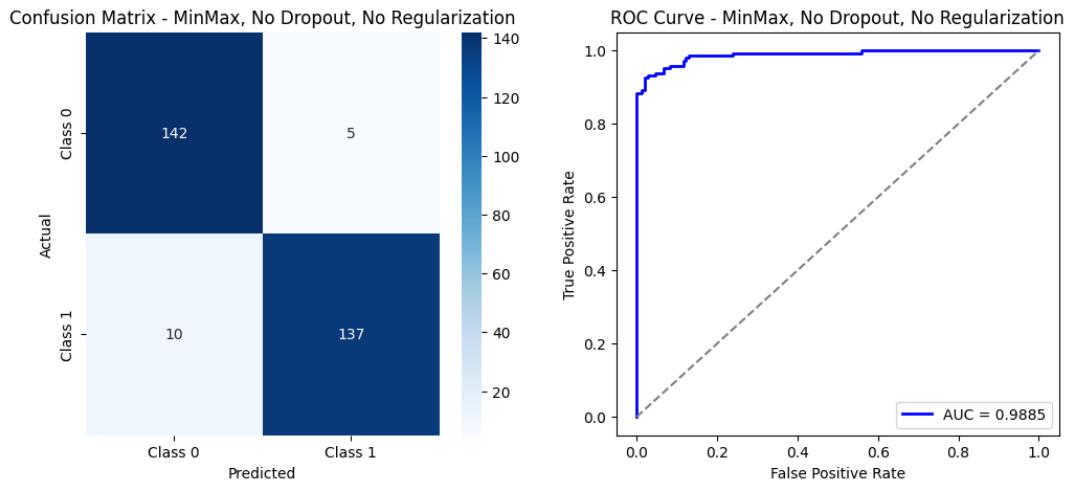
شکل 40: آموزش مدل عمیق تر روی داده های استاندارد با **dropout** و بدون **regularization**



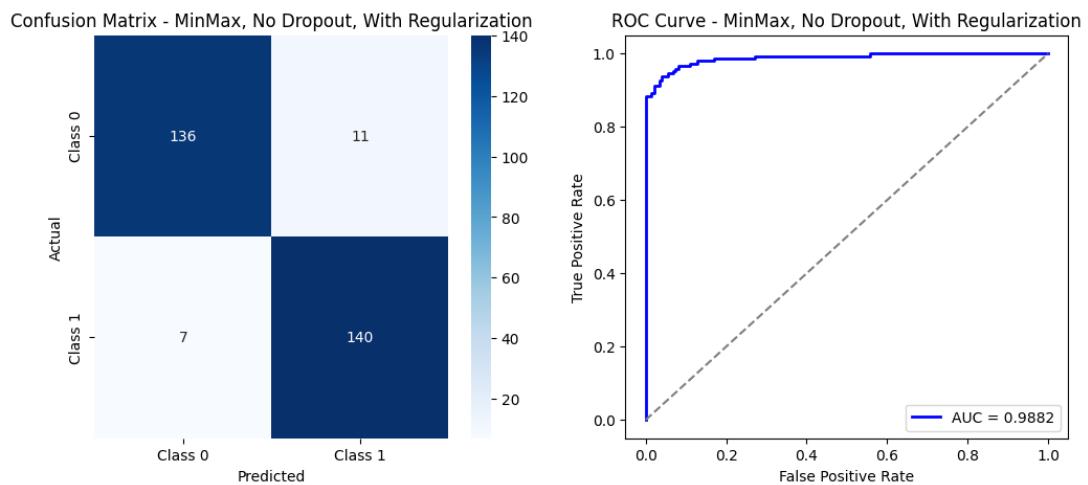
شکل 41: آموزش مدل عمیق تر روی داده های استاندارد با **dropout** و با **regularization**

با توجه به این نمودار ها می توان گفت با افزایش لایه دقت مدل روی داده های ترین بسیار افزایش یافته و این مورد هم می تواند نشانه این باشد که مدل روی داده های تست نیز عملکرد خوبی خواهد داشت و هم می تواند نشانه این باشد که مدل overfit شده است و روی داده های تست عملکرد خوبی نخواهد داشت.

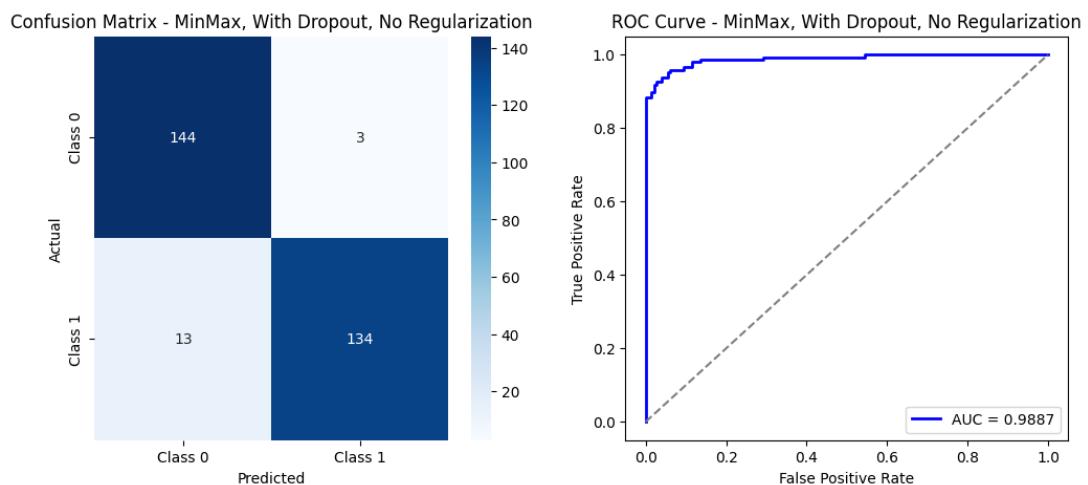
مانند قبل، مدل را تست میکنیم و نتایج را بدست می آوریم:



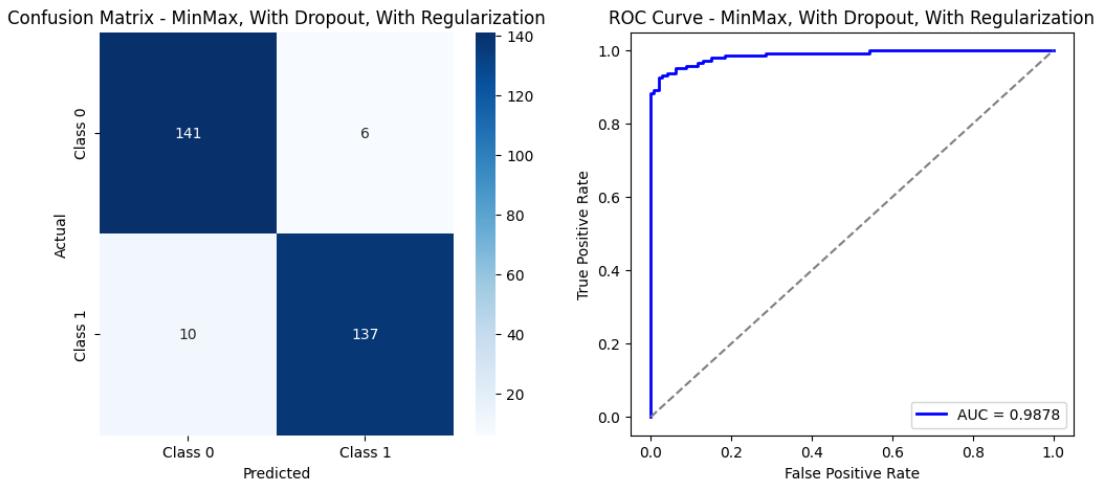
شکل 42: تست مدل عمیق تر روی داده های نرمالایز بدون regularization و بدون dropout



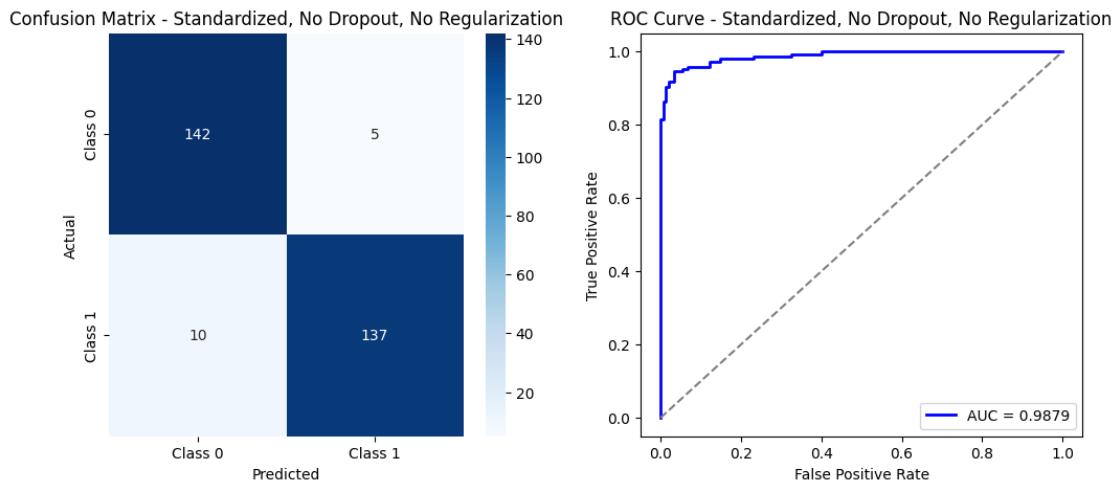
شکل 43: تست مدل عمیق تر روی داده های نرمالایز بدون regularization و با dropout



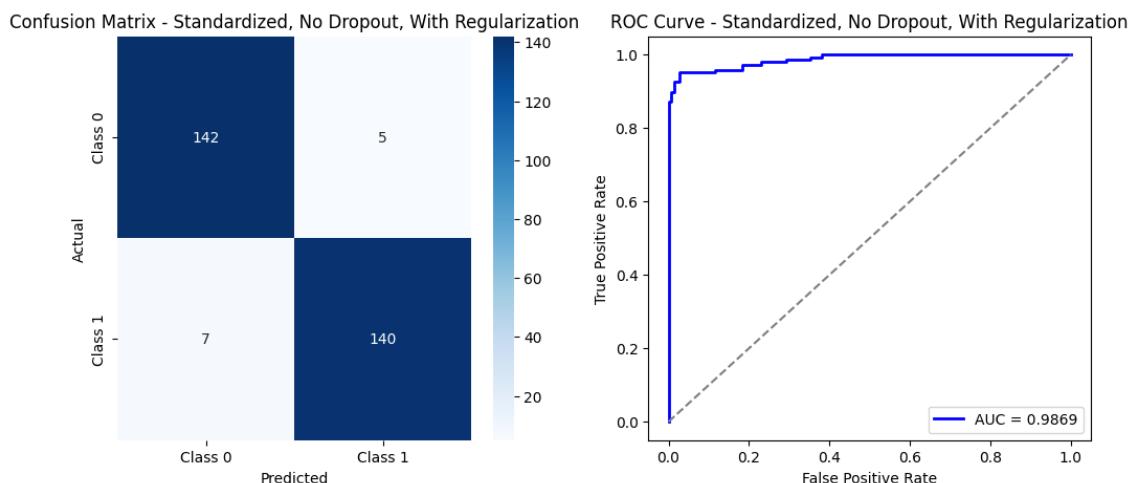
شکل 44: تست مدل عمیق تر روی داده های نرمالایز با regularization و بدون dropout



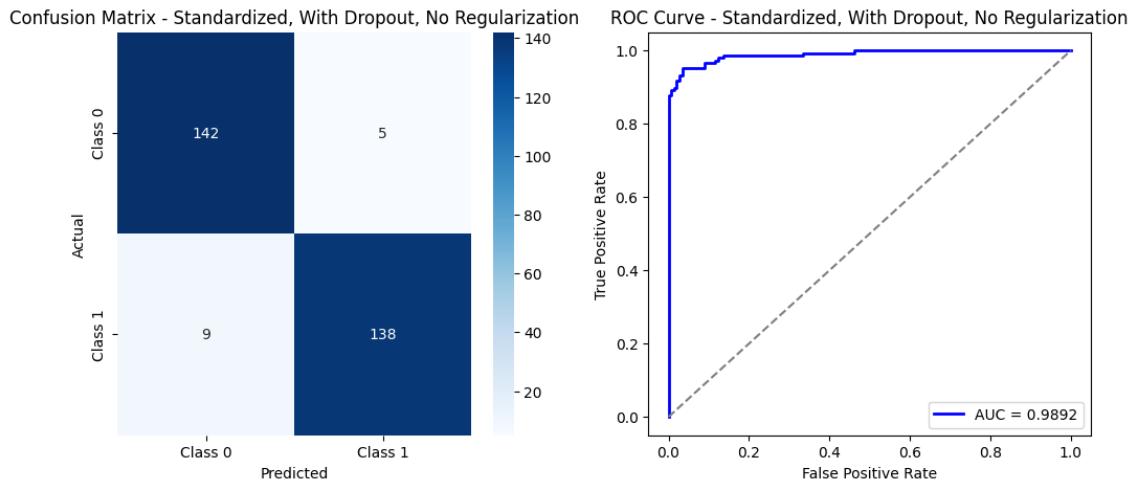
شکل 45: تست مدل عمیق تر روی داده های نرمالایز با dropout و با regularization



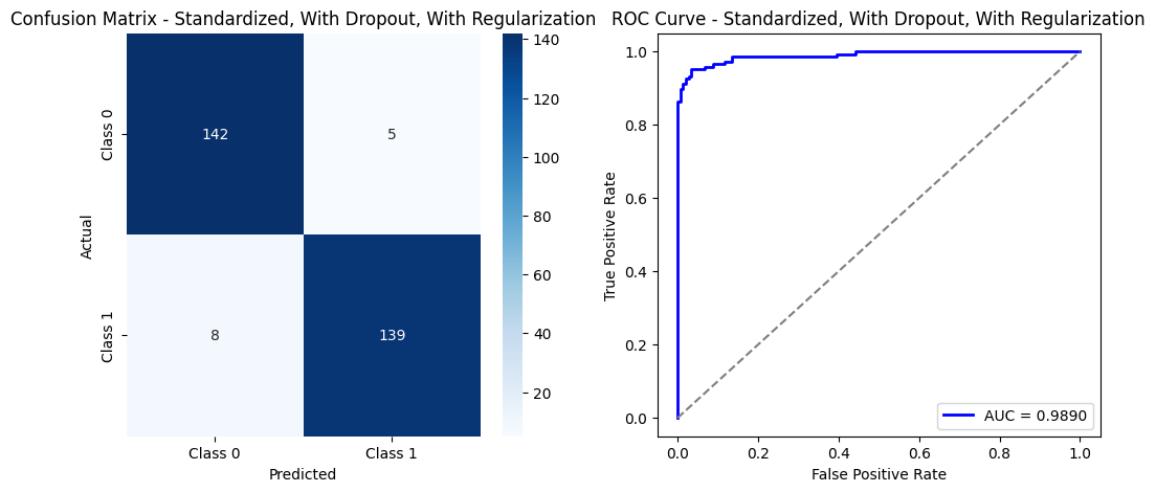
شکل 46: تست مدل عمیق تر روی داده های استاندارد بدون dropout و بدون regularization



شکل 47: تست مدل عمیق تر روی داده های استاندارد بدون dropout و با regularization



شکل 48: تست مدل عمیق تر روی داده های استاندارد با dropout و بدون regularization



شکل 49: تست مدل عمیق تر روی داده های استاندارد با dropout و با regularization

جدول 2 : مقایسه تمام متريک های کيفيت مدل برای تمام مدل ها با دو لایه مخفی

| Model  | Accuracy | Precision | Recall   | F1-Score | AUC Score |
|--|----------|-----------|----------|----------|-----------|
| <b>Standardized, No Dropout, With Regularization</b>   | 0.959184 | 0.965517  | 0.952381 | 0.958904 | 0.986857  |
| <b>Standardized, With Dropout, With Regularization</b> | 0.955782 | 0.965278  | 0.945578 | 0.955326 | 0.989032  |

|  |          |          |          |          |          |
|--|----------|----------|----------|----------|----------|
| <b>Standardized, With Dropout, No Regularization</b> | 0.952381 | 0.965035 | 0.938776 | 0.951724 | 0.989171 |
| <b>MinMax, No Dropout, No Regularization</b>         | 0.948980 | 0.964789 | 0.931973 | 0.948097 | 0.988523 |
| <b>Standardized, No Dropout, No Regularization</b>   | 0.948980 | 0.964789 | 0.931973 | 0.948097 | 0.987875 |
| <b>MinMax, With Dropout, With Regularization</b>     | 0.945578 | 0.958042 | 0.931973 | 0.944828 | 0.987783 |
| <b>MinMax, With Dropout, No Regularization</b>       | 0.945578 | 0.978102 | 0.911565 | 0.943662 | 0.988662 |
| <b>MinMax, No Dropout, With Regularization</b>       | 0.938776 | 0.927152 | 0.952381 | 0.939597 | 0.988153 |

همانطور که دیده می شود نسبت کیفیت مدل نسبت به حالت قبلی آنچنان بیشتر نشده و حتی در برخی موارد کمتر شده. این مورد نشان میدهد که مدل overfit شده است و روی داده های تست به خوبی عمل نمیکند.

جدای از این مورد ممکن است اصلا داده ها به صورت خطی جداپذیر باشند و اضافه کردن لایه صرفا وقت تلف کردن باشد!

## ۵-۱. تحلیل ماتریس آشفتگی و معیار های ارزیابی

اکثر مواردی که در این بخش بررسی میکنیم در انتهای بخش ۳-۱ بررسی و تعریف شد ولی مجددا آنها را تعریف و بررسی میکنیم.

معیار های Recall ، Precision ، Accuracy و F1-score معیار هایی هستند که تماماً از روی confusion matrix بدست می آیند.

Accuracy برابر است با تعداد تمام نمونه هایی که درست پیشبینی شده اند تقسیم بر تعداد تمام نمونه ها. اگر داده ها نامتوازن باشند ممکن است مدل درصد زیادی از داده های کلاس بزرگتر را درست پیشبینی

کند و درصد زیادی از داده های کلاس کوچکتر را اشتباه پیشبینی کند (زیرا با توجه به تعداد داده های بالا مدل نسبت به کلاس بزرگتر بایاس شده) در این حالت Accuracy مقدار بسیار بالایی دارد زیرا فقط به تعداد نمونه هایی که درست پیشبینی شده اند توجه میکند و تعداد نمونه هایی که درست پیشبینی شده اند نیز در این مثال بسیار بالاست ولی مدل در داده های مربوط به کلاس کوچکتر اصلاً عملکرد خوبی ندارد و Accuracy این را به ما نشان نمیدهد.

Precision کلاس 1 نشان میدهد از بین نمونه هایی که به عنوان کلاس 1 پیشبینی کردیم واقعاً چند درصد آنها متعلق به کلاس 1 بودند. در واقع به صورت عمودی روی confusion matrix نسبت تناسب میبندیم. علت اینکه Precision در تشخیص تقلب مهم است این است که هزینه پیشبینی مشتب کاذب بالاست. اگر مدل یک نمونه غیرتقلبی را به عنوان نمونه تقلبی پیشبینی کند، ممکن است باعث انسداد حساب، اخذ جریمه و ... شود و این هزینه بسیار زیادی را متحمل میکند. بنابراین ترجیح ما این است که درصد بسیار بالایی از نمونه هایی که متقلب تشخیص می دهیم واقعاً متقلب باشند حتی اگر درصد متقلب هایی که آنها را تشخیص نمیدهیم بالا رود (Recall) کاهش یابد.

Recall کلاس 1 نشان میدهد که از بین داده هایی که واقع متعلق به کلاس 1 بودند، چند درصد به درستی در کلاس 1 طبقه بندی شده اند. در واقع به صورت افقی روی confusion matrix نسبت تناسب میبندیم. در همان مسئله تقلب، میتوان اینگونه تعبیر کرد که چند درصد از متقلب های واقعی را تشخیص میدهیم. واضح است که ممکن است در مسئله ای برای ما هزینه تشخیص ندادن متقلب واقعی بیشتر از هزینه اشتباه تشخیص دادن افراد غیر متقلب باشد. در این حالت بدیهی است که سعی میکنیم Recall را افزایش دهیم حتی به قیمت کاهش Precision.

F1 Score میانگین هارمونیک Recall و Precision است که وقتی داده ها نا متوازن باشند گزینه بهتری از Accuracy برای سنجش مدل است. رابطه محاسبه F1 Score به این شکل است:

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

اگر توازن داده های کلاس ها نا متوازن باشد، ممکن است Precision کلاس ها مقادیر بزرگی باشند ولی Recall کلاس کوچکتر کاهش میابد و نشان میدهد که داده های آن کلاس به درستی طبقه بندی نمی شوند. در این حالت صورت کسر کوچک می شود ولی مخرج کسر همچنان مقدار بزرگی دارد و باعث می شود کل مقدار کسر کوچک شود. همچنین بر عکس این موضوع نیز صادق است. ممکن است یک مدل روی یک خروجی بایاس داشته باشد و همواره آن کلاس را پیشبینی کند. فرض کنید یک مدل همواره

کلاس 0 را پیشبینی میکند. در این حالت Recall کلاس 0 مقدار بزرگی دارد ولی Precision آن کم است. همین امر باعث می شود که F1 Score آن نیز پایین باشد.

ROC نموداری است که با تغییر آستانه تصمیم گیری، نشان میدهد مدل چقدر بین دو کلاس تمایز قائل میشود. محور X میزان پیشبینی های مثبت کاذب و محور Y میزان پیشبینی ها مثبت صحیح است. هر چقدر این نمودار به شبیه به اضلاع مربع باشد یعنی مدل عملکرد خوبی دارد و هرچقدر نمودار شبیه به قطر مربع باشد یعنی رفتار مدل بیشتر شبیه به رفتار یک طبقه بند تصادفی است. AUC نیز مساحت زیر این نمودار است که بین 0 و 1 است و هرچقدر به 1 نزدیک تر باشد یعنی مدل عملکرد بهتری داشته است.

درباره تحلیل *confusion matrix* ها می توان گفت که مدل همواره کلاس 0 را بهتر از کلاس 1 پیشبینی میکند. علت آن نیز این است که داده های کلاس 0 همگی داده های واقعی هستند ولی تقریبا 50% داده های کلاس 1 مصنوعی هستند و واضح است که مدلی که با داده های مصنوعی آموزش دیده نمی تواند به خوبی داده های واقعی را پیشبینی کند.

بین Precision و Recall همواره یک trade off وجود دارد و با افزایش یکی معمولاً دیگری افت میکند. Precision بالا یعنی فقط مواردی را مثبت تشخیص بدهیم که واقعاً مثبت هستند، حتی اگه بعضی از موارد مثبت واقعی را از دست بدھیم. Recall بالا یعنی سعی کنیم همهی موارد مثبت واقعی را پیدا کنیم، حتی اگه بعضی از موارد منفی را هم اشتباهها مثبت در نظر بگیریم. ذات تعریف این دو معیار باعث می شود همواره یک trade off بین این دو وجود داشته باشد.

## 6- جستجوی بهترین هایپرپارامتر ها شبکه یک لایه مخفی با روش حریصانه

برای جستجوی بهترین هایپر پارامتر ها، یک راه استفاده از کلاس GridSearchCV از کتابخانه Sklearn است. اما بخاطر اینکه برای تنظیم dropout نمی توانیم از کلاس MLPClassifier همین کتابخانه استفاده کنیم و باید از کتابخانه های دیگر استفاده کنیم، grid search را به صورت دستی انجام میدهیم. به این صورت که تمام مدل ها با مشخصات گفته شده را train می کنیم و سپس با توجه به ملاک های ارزیابی، بهترین آنها را انتخاب می کنیم.

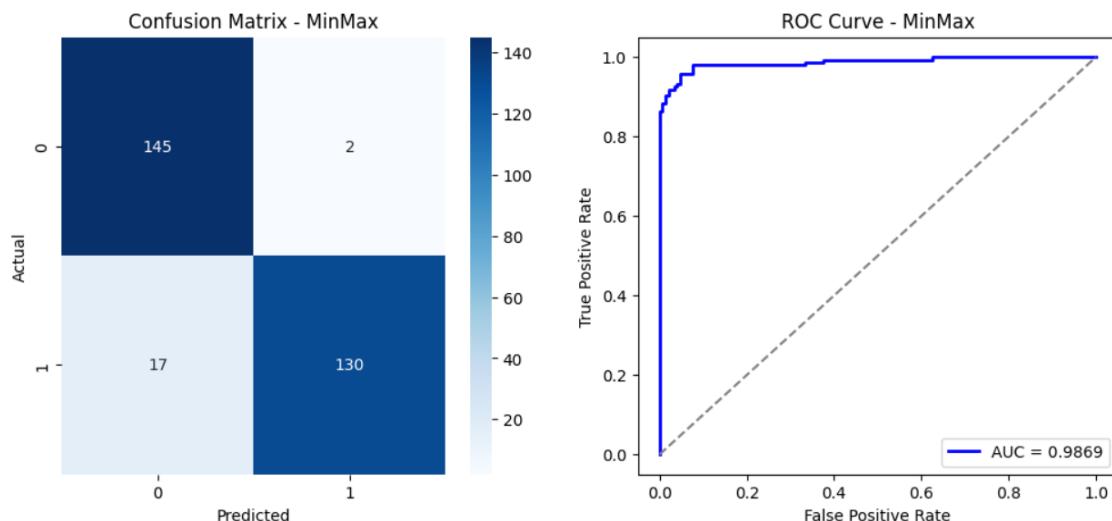
با توجه به جدول صفحه بعد که نتیجه grid search را نشان میدهد، مدلی که دارای 256 نورون در لایه مخفی است، دارای Dropout Rate = 0.2 L2 Regularization = 0.0001 است و دارای batch size = 16 ترین شود، بهترین کیفیت را بین مدل های بررسی شده خواهد داشت.

### جدول 3 . نتیجه grid search

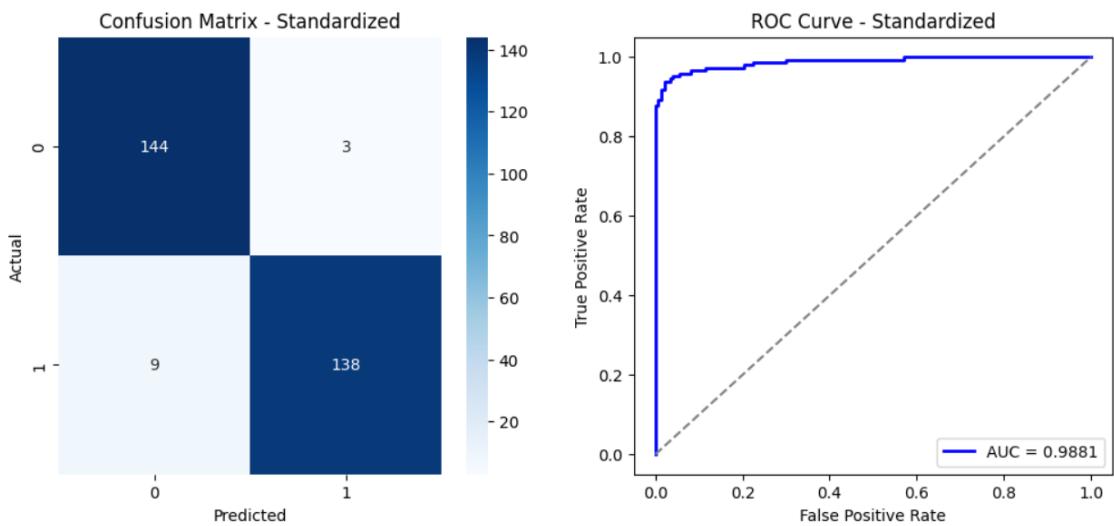
| Dataset      | Hidden Size | Dropout Rate | L2 Regularization | Batch Size | Accuracy | Precision | Recall   | F1-Score | AUC Score |
|--------------|-------------|--------------|-------------------|------------|----------|-----------|----------|----------|-----------|
| Standardized | 256         | 0.2          | 0.0001            | 16         | 0.959184 | 0.965517  | 0.952381 | 0.958904 | 0.988940  |
| Standardized | 128         | 0.2          | 0.0001            | 32         | 0.959184 | 0.972028  | 0.945578 | 0.958621 | 0.990606  |
| Standardized | 128         | 0.3          | 0.0010            | 16         | 0.959184 | 0.972028  | 0.945578 | 0.958621 | 0.990745  |
| Standardized | 128         | 0.3          | 0.0001            | 64         | 0.959184 | 0.978723  | 0.938776 | 0.958333 | 0.990837  |
| ...          | ...         | ...          | ...               | ...        | ...      | ...       | ...      | ...      | ...       |
| MinMax       | 64          | 0.3          | 0.0010            | 32         | 0.931973 | 0.970370  | 0.891156 | 0.929078 | 0.986302  |
| MinMax       | 64          | 0.2          | 0.0001            | 32         | 0.931973 | 0.970370  | 0.891156 | 0.929078 | 0.985839  |
| MinMax       | 256         | 0.4          | 0.0001            | 64         | 0.931973 | 0.970370  | 0.891156 | 0.929078 | 0.986071  |
| MinMax       | 64          | 0.4          | 0.0001            | 32         | 0.928571 | 0.963235  | 0.891156 | 0.925795 | 0.985191  |

### 7-1 مقایسه مدل MLP با مدل Logistic Regression

نتیجه آموزش و تست مدل Logistic Regression به شکل زیر است:



شکل 50 . نتیجه تست مدل Logistic Regression با داده های نرمالایز



شکل 51: نتیجه تست مدل Logistic Regression با داده های استاندارد

جدول 4: نتیجه تست Logistic Regression

| Dataset      | Accuracy | Precision | Recall   | F1-Score | AUC Score |
|--------------|----------|-----------|----------|----------|-----------|
| Standardized | 0.959184 | 0.978723  | 0.938776 | 0.958333 | 0.988061  |
| MinMax       | 0.935374 | 0.984848  | 0.884354 | 0.931900 | 0.986904  |

با توجه به نتایج بالا در این مثال خاص مدل Logistic Regression تفاوت چندانی با یک شبکه MLP ندارد و حتی سریع‌تر از یک شبکه MLP آموزش داده می‌شود.

برتری شبکه MLP زمانی محسوس می‌شود که تعداد داده‌ها زیاد و مرز جدابذیری آنها غیرخطی باشد. اگر تعداد داده‌ها کم باشد و داده‌ها به صورت خطی تفکیک پذیر باشند مدل Logistic Regression حتی می‌تواند بهتر عمل کند.

## 8-1. جمع‌بندی

با توجه به نتایج بدست آمده، اگر F1 Score را به عنوان معیار درنظر بگیریم، مدل‌های MLP تک لایه با 256 نورون (که مشخصات آن در سطر اول جدول 3 وجود دارد) و دو لایه (سطر اول جدول 2) روی داده‌ای استاندارد با مقدار  $F1\ Score = 0.958904$  بهترین عملکرد را داشته‌اند.

در این مثال افزودن لایه‌های بیشتر تاثیر زیادی کیفیت مدل نداشت. علت این امر می‌تواند overfit شدن مدل یا خطی بودن مرز تفکیک داده‌ها باشد. (برای اینکه مرز تفکیک داده‌ها را ببینیم می‌توانیم از scatter plot استفاده کنیم). ولی به طور کلی افزودن لایه‌های بیشتر باعث می‌شود مدل بتواند مرزها پیچیده‌تر را پیدا کند.

بهینه سازی هایپرپارامتر ها روی دقت، سرعت آموزش مدل و قدرت تعمیم پذیری آن تاثیرگذار هستند. به طور کلی بعضی هایپر پارامتر ها مربوط به معماری شبکه (مثلاً تعداد نورون های لایه مخفی) و برخی هایپر پارامتر ها مربوط به آموزش شبکه (مثلاً نرخ regularization) هستند. همچنین بهینه سازی این پارامتر ها می تواند از Underfitting و Overfitting نیز جلوگیری کند. در این مثال بهینه سازی هایپرپارامتر ها باعث شد مدل تا حدودی عملکر بہتری داشته باشد.

میزان خطای مدل حدود 5% بود و خب بنظر خطای قابل قبولی بود. با توجه به اینکه اکثر اشتباهات مدل در کلاس 1 رخ میداد بنظر میرسد دلیل این خطا وجود داده های مصنوعی در کلاس 1 است. راه حل نیز این است که از کلاس 1 داده واقعی بیشتری داشته باشیم.

اگر دقت را افزایش دهیم، مدل محافظه کارتر خواهد شد و فقط مواردی را تقلیبی تشخیص می دهد که بسیار مطمئن است، که می تواند منجر به کاهش بازخوانی شود (چندین مورد تقلبی واقعی را از دست می دهیم). اگر بازخوانی را افزایش دهیم، مدل تلاش می کند همه موارد تقلب را شناسایی کند، حتی به قیمت افزایش مثبت های کاذب (False Positives)، که باعث کاهش دقت می شود. حالا باید ببینیم که هزینه کدام مورد برای ما بیشتر است. اینکه یک تقلب واقعی را از دست بدھیم یا اینکه یک نمونه درست را تقلب تشخیص دهیم.

به نظر من در این مثال مدل پیچیده تر فقط محاسبات را افزایش داد. درست است که مدل دولایه مقدار کمی بهتر عمل می کرد ولی با توجه به افزایش تعداد لایه ها و تعداد نورون ها این افزایش دقت زیاد به چشم نمی آمد و ارزش نداشت. ولی در داده هایی با مرز های پیچیده تر و تعداد بیشتر قطعاً مدل با لایه های بیشتر عملکرد بہتری خواهد داشت.

برای بهتر کردن عملکرد مدل می توان چند کار را انجام داد. اولین مورد این است که داده ها را بهتر پیش پردازش کنیم. داده های پرت و نویز ها را حذف کنیم و مقادیر گمشده را حذف کنیم. سپس می توانیم در grid search پارامتر های دیگری مثل learning rate و نوع بهینه ساز را نیز جستجو کنیم و بهترین پارامتر ها را پیدا کنیم.

بنظر من مهم ترین چالش عدم تعادل کلاس ها است. چون معمولاً نمونه های متقلب تعداد بسیار کمتری نسبت به نمونه های درست دارند و از همین نمونه های کم نیز تعداد بسیار کمی تشخیص داده می شوند و داده های آنها در اختیار ما قرار میگیرد (که اگر تعداد زیادی از آنها تشخیص داده میشندند نیازی به آموزش مدل برای تشخیص تقلب نبود!).

اگر بخواهیم مدل را دوباره طراحی کنم گرید سرچ را روی پارامتر های بیشتری انجام میدهم و همینطور برای بهینه سازی تعداد نورون ها مدل دو لایه با تعداد کم نورون را امتحان میکنم. مثلا مدل دو لایه ای که در لایه مخفی اول 32 نورون و در لایه مخفی دوم 8 نورون دارد. همچنین پیش پردازش داده ها را با دقیق بیشتر انجام میدهم و احتمالا با استفاده از معیار های بیشتری تعداد ویژگی های بیشتر را حذف می کنم.

## پرسش ۲ - طراحی شبکه عصبی چند لایه در مسئله رگرسیون مقاومت

بتن

### ۱-۲. آماده سازی دادگان و تحلیل آماری

برای بررسی کلی دادگان می توانیم در ابتدا یک نگاه کلی به دیتافریم داشته باشیم، سپس با استفاده از متدهای info کلیات دادگان مانند تعداد سطرها و ستونها و نوع دادگان هر ویژگی را بدست آوریم و نهایتاً با استفاده از متدهای describe مشخصات آماری دادگان مانند میانگین، واریانس و چارک های مختلف را ببینیم.

| CementComponent | BlastFurnaceSlag | FlyAshComponent | WaterComponent | SuperplasticizerComponent | CoarseAggregateComponent | FineAggregateComponent | AgeInDays | Strength  |
|-----------------|------------------|-----------------|----------------|---------------------------|--------------------------|------------------------|-----------|-----------|
| 0               | 540.0            | 0.0             | 0.0            | 162.0                     | 2.5                      | 1040.0                 | 676.0     | 28 79.99  |
| 1               | 540.0            | 0.0             | 0.0            | 162.0                     | 2.5                      | 1055.0                 | 676.0     | 28 61.89  |
| 2               | 332.5            | 142.5           | 0.0            | 228.0                     | 0.0                      | 932.0                  | 594.0     | 270 40.27 |
| 3               | 332.5            | 142.5           | 0.0            | 228.0                     | 0.0                      | 932.0                  | 594.0     | 365 41.05 |
| 4               | 198.6            | 132.4           | 0.0            | 192.0                     | 0.0                      | 978.4                  | 825.5     | 360 44.30 |
| ...             | ...              | ...             | ...            | ...                       | ...                      | ...                    | ...       | ...       |
| 1025            | 276.4            | 116.0           | 90.3           | 179.6                     | 8.9                      | 870.1                  | 768.3     | 28 44.28  |
| 1026            | 322.2            | 0.0             | 115.6          | 196.0                     | 10.4                     | 817.9                  | 813.4     | 28 31.18  |
| 1027            | 148.5            | 139.4           | 108.6          | 192.7                     | 6.1                      | 892.4                  | 780.0     | 28 23.70  |
| 1028            | 159.1            | 186.7           | 0.0            | 175.6                     | 11.3                     | 989.6                  | 788.9     | 28 32.77  |
| 1029            | 260.9            | 100.5           | 78.3           | 200.6                     | 8.6                      | 864.5                  | 761.5     | 28 32.40  |

شکل 52: نگاهی کلی به دادگان مربوط به مقاومت بتن

```
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   CementComponent  1030 non-null   float64
 1   BlastFurnaceSlag 1030 non-null   float64
 2   FlyAshComponent   1030 non-null   float64
 3   WaterComponent    1030 non-null   float64
 4   SuperplasticizerComponent  1030 non-null   float64
 5   CoarseAggregateComponent  1030 non-null   float64
 6   FineAggregateComponent  1030 non-null   float64
 7   AgeInDays         1030 non-null   int64  
 8   Strength          1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.6 KB
```

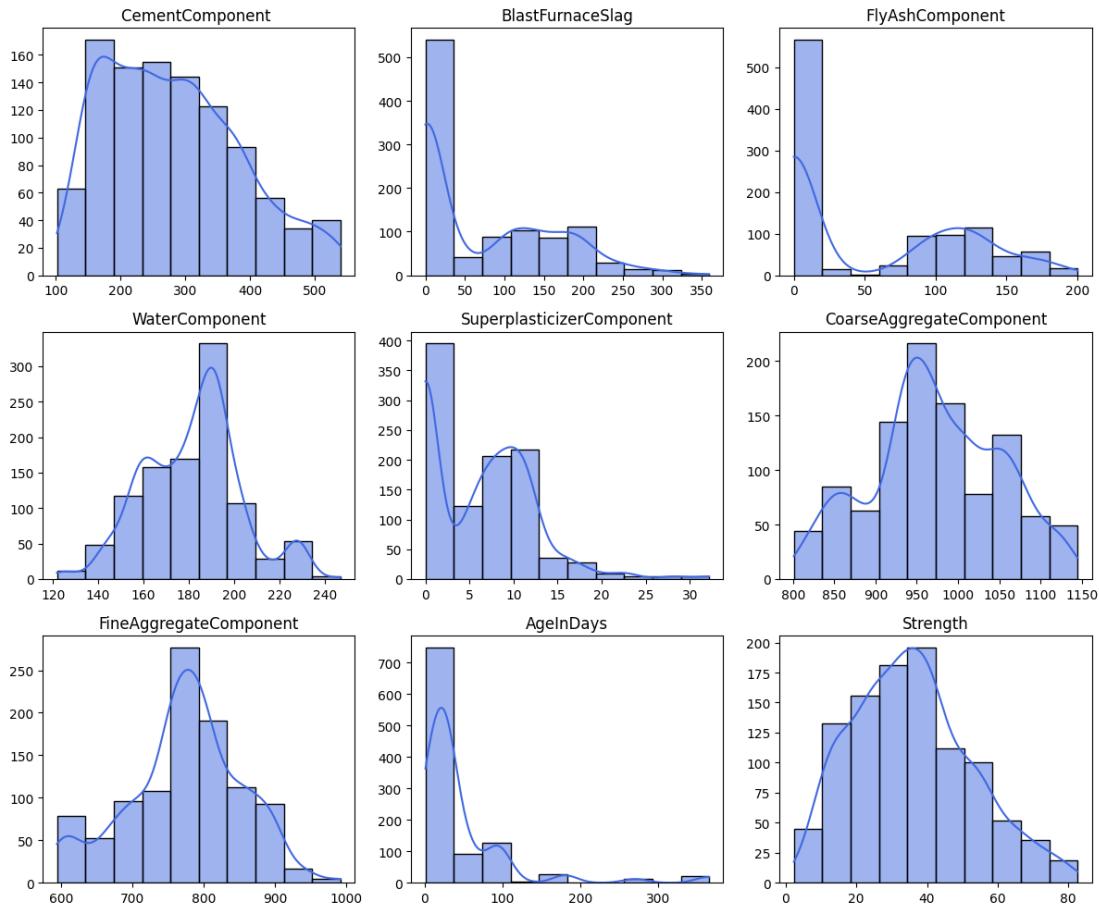
شکل 53: دادگان مربوط به مقاومت بتن info

|       | CementComponent | BlastFurnaceSlag | FlyAshComponent | WaterComponent | SuperplasticizerComponent | CoarseAggregateComponent | FineAggregateComponent | AgeInDays   | Strength    |
|-------|-----------------|------------------|-----------------|----------------|---------------------------|--------------------------|------------------------|-------------|-------------|
| count | 1030.000000     | 1030.000000      | 1030.000000     | 1030.000000    | 1030.000000               | 1030.000000              | 1030.000000            | 1030.000000 | 1030.000000 |
| mean  | 281.167864      | 73.895825        | 54.188350       | 181.567282     | 6.204660                  | 972.918932               | 773.580485             | 45.662136   | 35.817961   |
| std   | 104.506364      | 86.279342        | 63.997004       | 21.354219      | 5.973841                  | 77.753954                | 80.175980              | 63.169912   | 16.705742   |
| min   | 102.000000      | 0.000000         | 0.000000        | 121.800000     | 0.000000                  | 801.000000               | 594.000000             | 1.000000    | 2.330000    |
| 25%   | 192.375000      | 0.000000         | 0.000000        | 164.900000     | 0.000000                  | 932.000000               | 730.950000             | 7.000000    | 23.710000   |
| 50%   | 272.900000      | 22.000000        | 0.000000        | 185.000000     | 6.400000                  | 968.000000               | 779.500000             | 28.000000   | 34.445000   |
| 75%   | 350.000000      | 142.950000       | 118.300000      | 192.000000     | 10.200000                 | 1029.400000              | 824.000000             | 56.000000   | 46.135000   |
| max   | 540.000000      | 359.400000       | 200.100000      | 247.000000     | 32.200000                 | 1145.000000              | 992.600000             | 365.000000  | 82.600000   |

شکل 54: مشخصات آماری دادگان مربوط به مقاومت بتن

نکته قابل توجه در این مشخصات آماری، ستون های BlastFurnaceSlag و FlyAshComponent و SuperplasticizerComponent هستند. در این ستون ها چارک 25% پایین صفر است و حتی در ستون FlyAshComponent چارک 50% درصد نیز صفر است! این مورد نشان میدهد که این ویژگی ها تعداد زیادی داده صفر دارند. در صورتی که معنی آنها به ترتیب « جزء خاکستر بادی » ، « سرباره کوره بلند » و « جزء فوق روان کننده » است. بنظر نمی رسد که مقادیر صفر برای این ستون ها منطقی باشد. آن هم برای این تعداد زیاد از داده. بنابر این نتیجه میگیریم که در این مجموعه دادگان مقادیر گمشده داریم و باید فکری به حال آنها بکنیم. در گام های بعدی این موارد را لحاظ خواهیم کرد.

هیستوگرام داده ها را رسم می کنیم.



شکل ۵۵: هیستوگرام تک تک ویژگی ها

داده های گمشده در هیستوگرام نیز خود را نشان میدهند! نکته دیگری که در هیستوگرام ویژگی ها قابل توجه است همبستگی برخی ویژگی ها با target است. برای مثال بنظر میرسد ویژگی CementComponent همبستگی بسیار خوبی با خروجی داشته باشد و به عنوان یک ویژگی مناسب به کار رود.

پیش از بررسی همبستگی ویژگی ها و رسم Scatter Plot ابتدا داده های تست و ترین را از هم جدا میکنیم و سپس روی داده های تست تکنیک هایی اجرا میکنیم که دادگان گمشده را جایگزین کنیم (مقادیر گمشده باید درون مجموعه داده تست وجود داشته باشند چون در دنیای واقعی مدل ممکن است با داده های ناقص مواجه شود).

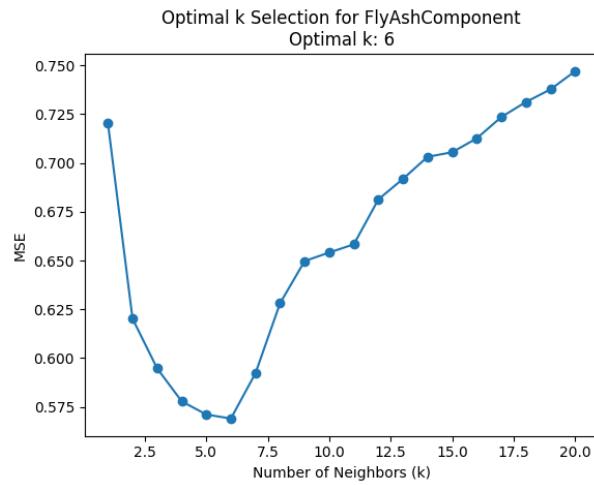
برای هندل کردن مقادیر گمشده در داده ها به ترتیب گام های زیر را طی می کنیم:

1- ابتدا 20% دادگان را برای تست و 80% را برای آموزش مدل جدا میکنیم.

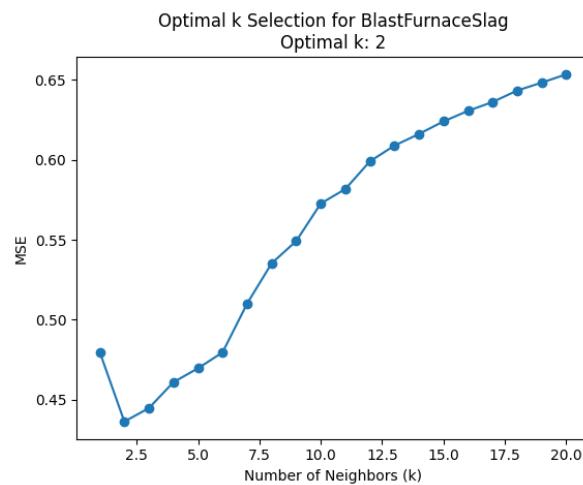
2- مجموعه داده های آموزشی را بدون در نظر گرفتن مقادیر گمشده استاندارد سازی میکنیم(با توجه به اینکه در تمامی موارد سوال قبلی دیدیم که مدل روی داده های استاندارد بهتر از داده های نرمال عمل میکند، دیگر داده ها را نرمال نمی کنیم).

3- مجموعه داده های تست را نیز با میانگین و واریانس داده های ترین استاندارد سازی میکنیم.

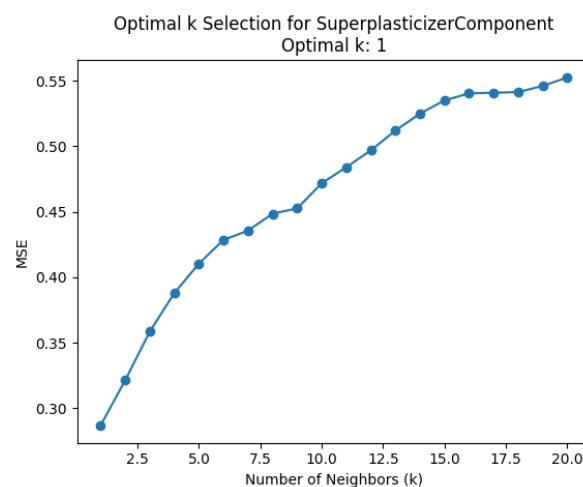
4- ستون هایی از داده های ترین که شامل مقادیر گمشده هستند را جدا میکنیم و هر یک از ستون ها را به عنوان یک ستون target در نظر میگیریم و بقیه ستون ها را به عنوان ویژگی ها درنظر میگیریم. روی داده هایی که از دست نرفته اند مدل K Neighbors Regressor را آموزش میدهیم و با تست کردن K های مختلف و استفاده از روش زانو K بهینه رو پیدا میکنیم، نمودار های مربوط به این بخش در زیر آورده شده:



شکل 56: پیدا کردن K بهینه برای ستون FlyAshComponent



شکل 57: پیدا کردن K بهینه برای ستون BlastFurnaceSlag



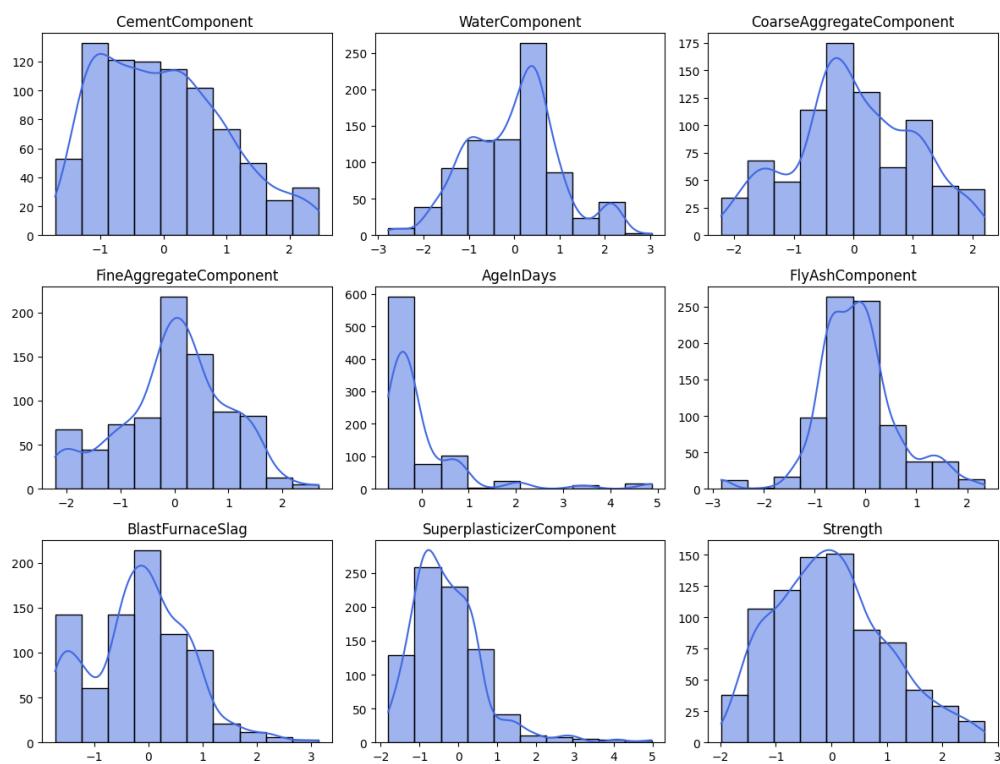
شکل 58: پیدا کردن K بهینه برای ستون SuperplasticizerComponent

5- روی هر ستون با استفاده از K Neighbors Regressor آموزش میدهیم و مقادیر گمشده داده های ترین را با این روش جایگزین میکنیم.

6- داده های تست را به همین مدل که روی داده های ترین آموزش داده ایم میدهیم و خروجی را جایگزین مقدار گمشده میکنیم. در واقع یعنی اگر به مقدار گمشده ای در داده های تست بخوردیم، K همسایه نزدیک آن را در داده های ترین پیدا میکنیم و میانگین آن ها را به عنوان جایگزین در داده تست قرار میدهیم.

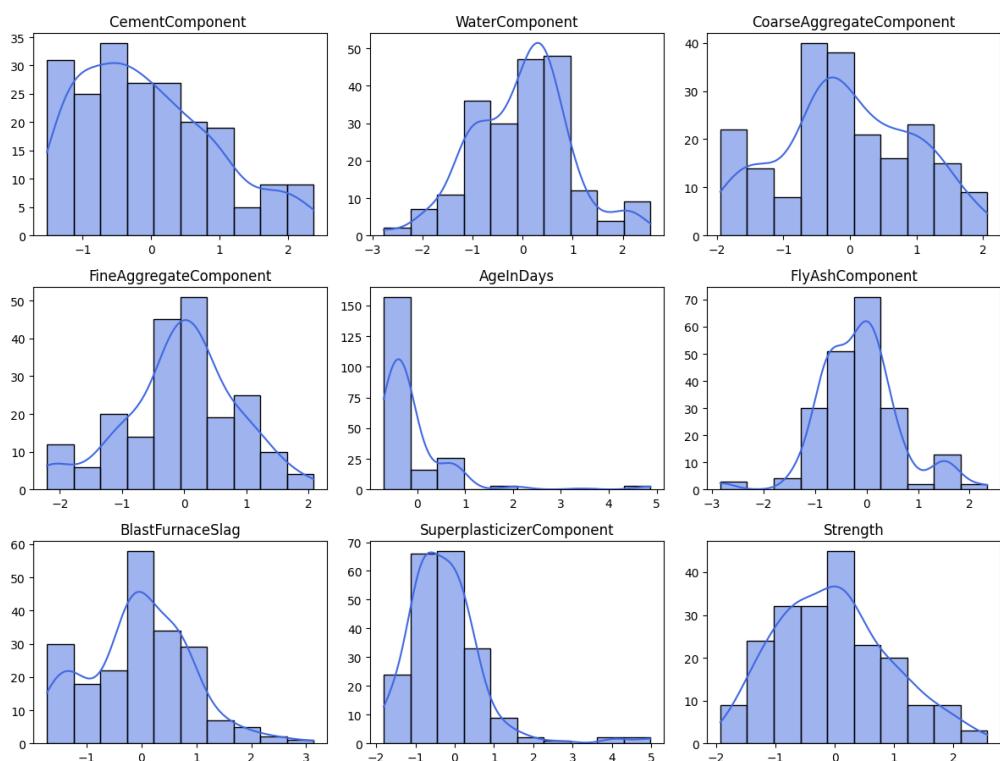
حالا داده های تست و ترین را بدون مقادیر گمشده ، به صورت استاندارد سازی شده در اختیار داریم و می توانیم گام های بعدی را روی آنها اعمال کنیم. هیستوگرام این داده ها را رسم میکنیم:

Train Data Histograms



شکل ۵۹: هیستوگرام ویژگی های داده های تست بعد از استاندارد سازی و جایگزینی مقادیر گمشده

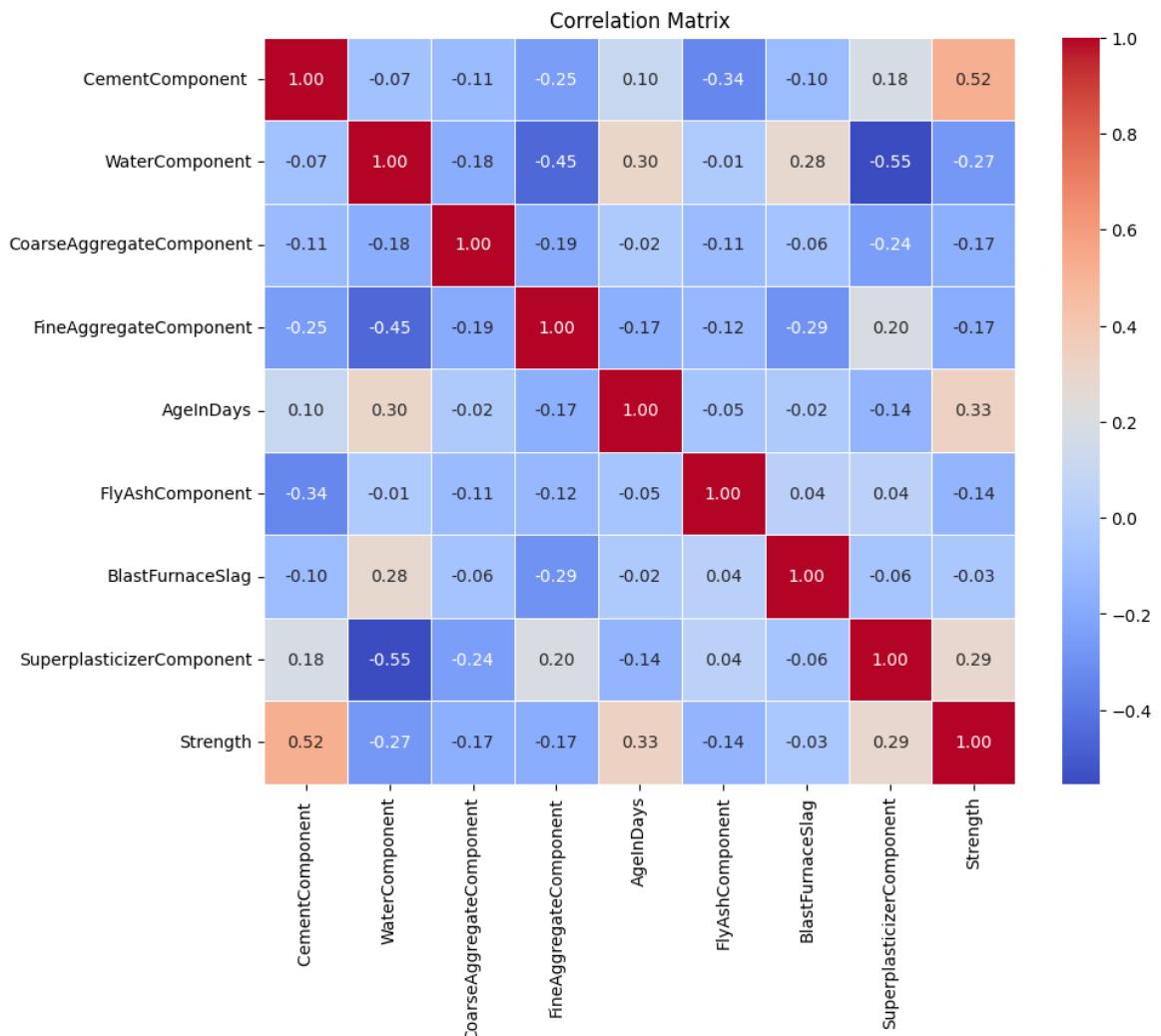
Test Data Histograms



شکل ۶۰: هیستوگرام ویژگی های داده های تست بعد از استاندارد سازی و جایگزینی مقادیر گمشده

همانطور که دیده می شود روی داده های ترین دیگر داده از دست رفته نداریم و داده ها هیستوگرامی تشکیل داده اند که نسبتا با خروجی نیز همبستگی خوبی دارند.

ماتریس همبستگی ویژگی ها را بدست می آوریم

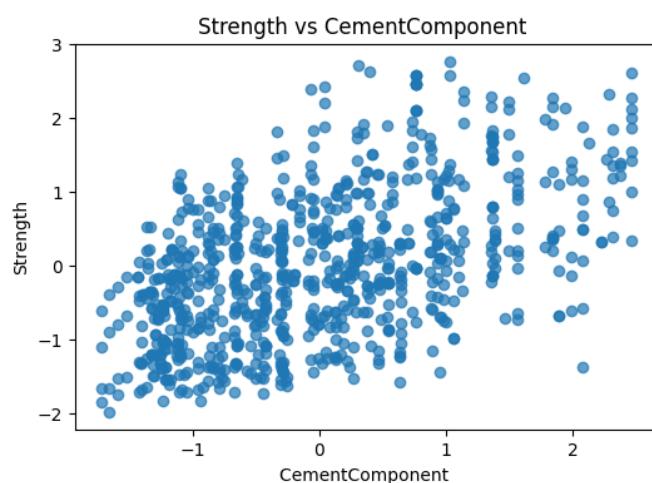


شکل 61: ماتریس همبستگی ویژگی ها

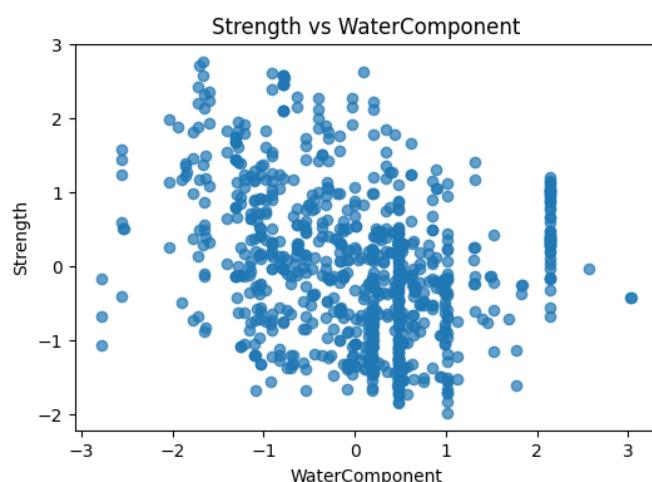
با توجه به این ماتریس هیچ ویژگی همبستگی قابل توجهی با مقاومت بتن ندارد و هیچ دو ویژگی نیز همبستگی قابل توجهی با یکدیگر ندارند. اما اگر بخواهیم از روی همین ماتریس نظر دهیم، میتوانیم بگوییم CementComponent بیشترین همبستگی را با Strength دارد ولی باز این همبستگی آنچنان زیاد نیست. همچنین ویژگی های WaterComponent و SuperplasticizerComponent نیز باهم همبستگی بالایی دارند ام همچنان آنقدر زیاد نیست که بخواهیم یکی از ویژگی ها را حذف کنیم. معمولا ما تمایل داریم ویژگی ها کمترین همبستگی را با یکدیگر و بیشترین همبستگی را با target داشته باشند. زیرا همبستگی بالای دو ویژگی با هم یعنی این دو ویژگی اطلاعات مشترک زیادی دارند

و این خوب نیست. ولی از طرفی همبستگی زیاد یک ویژگی با متغیر هدف، یعنی می توان تا حدودی متغیر هدف را با این ویژگی پیش‌بینی کرد. ولی خب هیچکدام از همبستگی های ماتریس بالا آنچنان قابل توجه نیستند. معمولا همبستگی بالا را بالای 0.9 و همبستگی پایین را کمتر از 0.1 در نظر می‌گیریم. در این ماتری اگر بخواهیم ویژگی ای را حذف کنیم اولین کاندید BlastFurnaceSlag است که فقط 0.03 با متغیر هدف همبستگی دارد. ولی با توجه به اینکه فعلاً محدودیتی روی تعداد ویژگی ها نداریم این ویژگی را نگه میداریم.

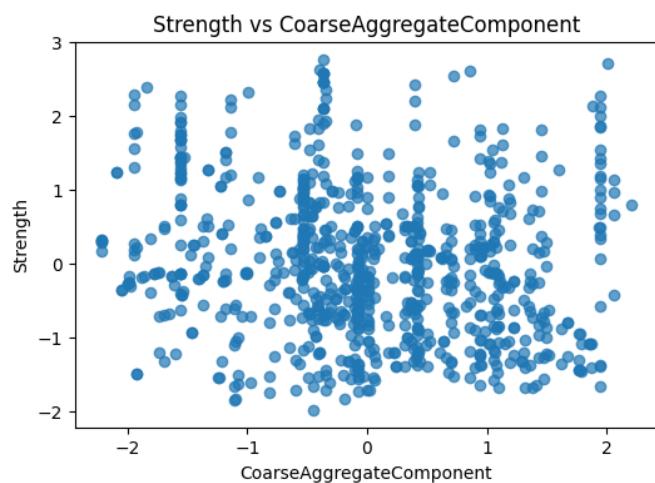
حالا در این بخش می خواهیم Scatter Plot متغیر هدف بر اساس تک تک ویژگی ها را بدست آوریم.



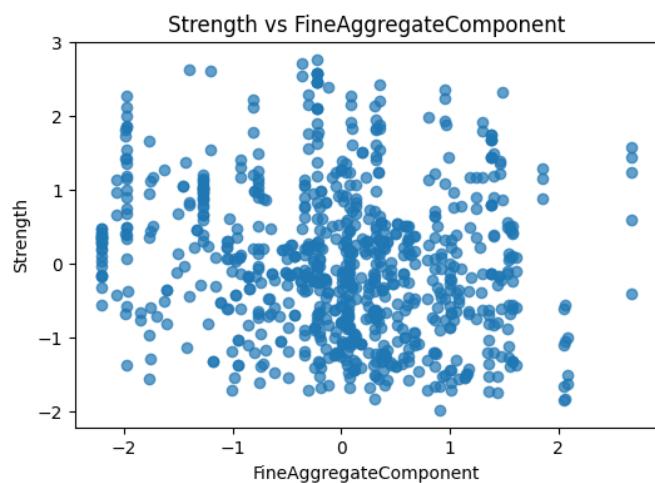
شکل 62: CementComponent ویژگی Scatter Plot



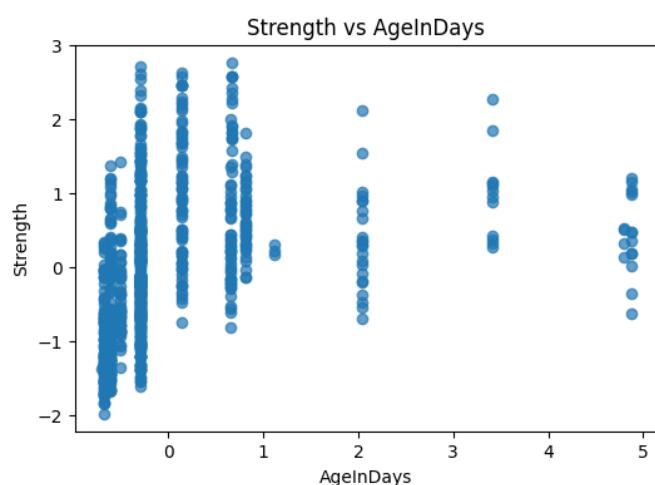
شکل 63: WaterComponent ویژگی Scatter Plot



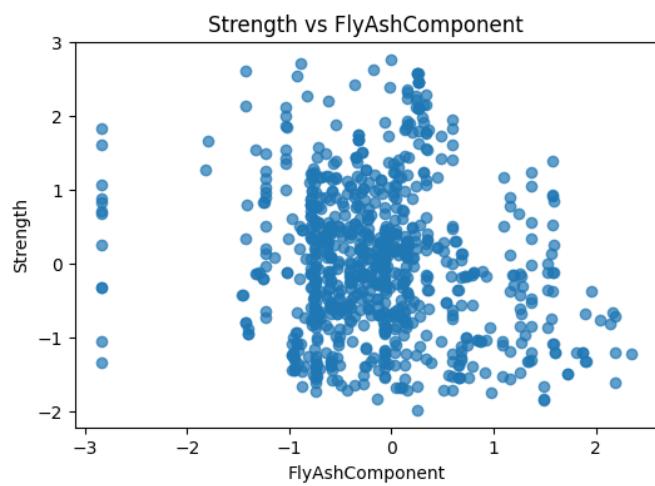
شکل 64: CoarseAggregateComponent ویژگی Scatter Plot



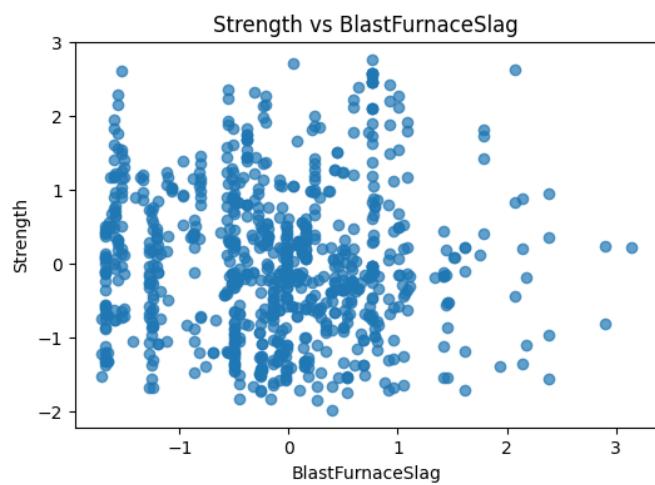
شکل 65: FineAggregateComponent ویژگی Scatter Plot



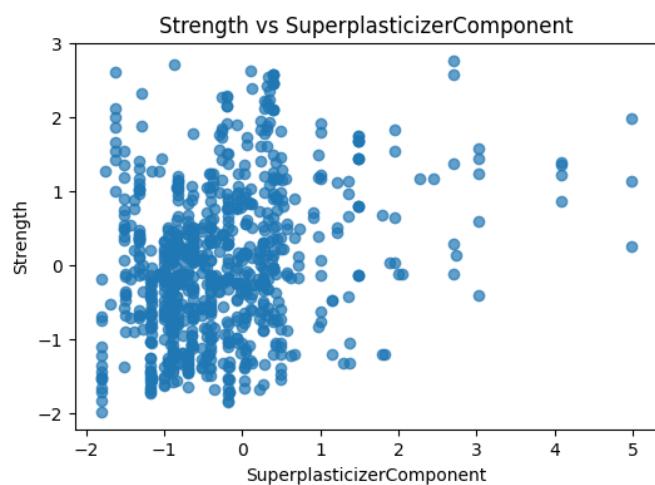
شکل 66: AgeInDays ویژگی Scatter Plot



شکل 67: FlyAshComponent ویژگی Scatter Plot



شکل 68: BlastFurnaceSlag ویژگی Scatter Plot

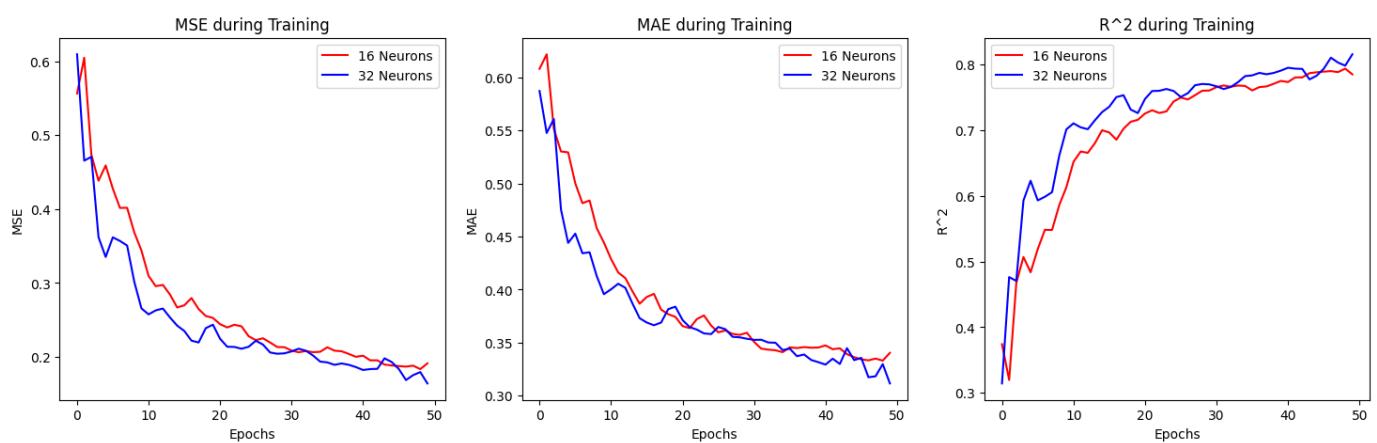


شکل 69: SuperplasticizerComponent ویژگی Scatter Plot

برای تحلیل Scatter Plot باید اینگونه در نظر بگیریم که فرض کنید می خواستیم فقط با استفاده از یک ویژگی متغیر هدف یا همان strength را پیش‌بینی کنیم. در این صورت نمودار Scatter متغیر هدف بر حسب آن ویژگی، باید یکتابع باشد که یک نویز روی آن سوار شده و ما باید آن تابع را تخمین بزنیم. اگر آن تابع خطی باشد می توانیم از رگرسیون خطی نیز استفاده کنیم. حالا در بین این Scatter Plot ها، نمودار مربوط به ویژگی CementComponent از همه خوش فرم تر است. انگار با افزایش CementComponent ، مقدار strength نیز به صورت خطی و مناسب با آن افزایش می یابد. در بقیه نمودارها نیز می توان الگوهایی پیدا کرد، مثلا نمودار مربوط به AgeInDays می تواند الگوی یک سهمی درجه دو برعکس باشد. ولی به صورت کلی هیچکدام به اندازه CementComponent محسوس نیستند. به هر حال در این مرحله نیز هیچ ویژگی ای نیست که به طور قابل توجهی ویژگی بد یا بسیار خوب محسوب شود و با توجه به این موضوع در این مرحله نیز مایل به حذف ویژگی ها نیستیم!

## ۲-۲. پیاده‌سازی مدل شبکه عصبی چند لایه

با توجه به خواسته سوال، معماری خواسته شده را طراحی می کنیم، آموزش میدهیم و نهایتاً ارزیابی می‌کنیم. نمودارهای مربوط به آموزش مدل است، شامل MSE ، MAE و  $R^2$  در هر ایپاک آورده شده است. همچنین نهایتاً نتیجه ارزیابی مدل‌ها روی داده‌های تست، در یک جدول آورده شده.



شکل 70: توابع مربوط به خطأ و دقت در آموزش شبکه عصبی رگرسور

جدول 5 : نتیجه تست و ارزیابی در شبکه عصبی رگرسور

| Neurons    | MSE      | MAE      | $R^2$    |
|------------|----------|----------|----------|
| 16 Neurons | 0.191261 | 0.340414 | 0.784896 |

|            |          |          |          |
|------------|----------|----------|----------|
| 32 Neurons | 0.164104 | 0.311596 | 0.815438 |
|------------|----------|----------|----------|

با توجه به هزینه ها و با توجه به معیار  $R^2$  مدل با 32 نورون عملکرد بهتری داشته است. فرمول معیار

$$R^2 = 1 - \frac{\text{(Sum of Squares of Residuals)}}{\text{(Total Sum of Squares)}}$$

به این شکل است:

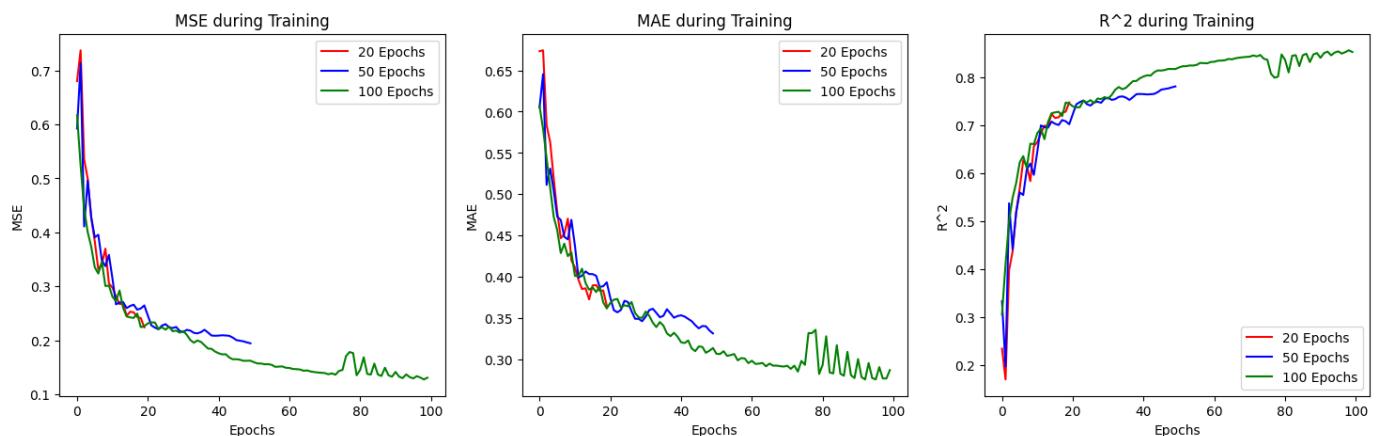
با توجه به این رابطه اگر  $R^2 = 1$  باشد مدل به شکل ایده آل عمل کرده و هرچقدر این مقدار به 1 نزدیک تر باشد، یعنی مدل عملکرد بهتری داشته است.

### 3-۲. بررسی تغییرات تنظیمات مدل

تعداد نورون ها را روی 32 فیکس میکنیم و موارد خواسته شده در سوال را بررسی می کنیم.

- تاثیر تعداد ایپاک ها:

مدل را با تعداد ایپاک های 20 و 50 و 100 ترین میکنیم و نتایج را مانند قبل گزارش میکنیم.



شکل 71: توابع مربوط به خطأ و دقت برای ایپاک های مختلف

جدول 6 : نتیجه تست و ارزیابی برای ایپاک های مختلف

| Epochs    | MSE      | MAE      | $R^2$    |
|-----------|----------|----------|----------|
| 20 Epochs | 0.224061 | 0.365096 | 0.748007 |
| 50 Epochs | 0.194441 | 0.331605 | 0.781320 |

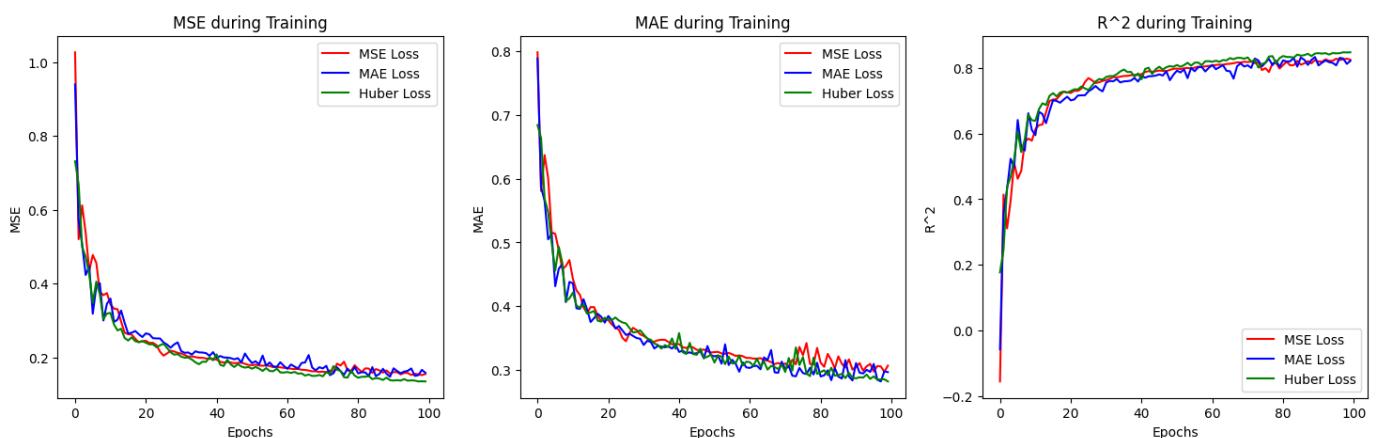
|                   |          |          |          |
|-------------------|----------|----------|----------|
| <b>100 Epochs</b> | 0.130718 | 0.287195 | 0.852986 |
|-------------------|----------|----------|----------|

با توجه به این نمونه ها افزایش تعداد ایپاک ها در این مثال تاثیر مثبت داشته است، ولی نمی توانیم این حرف را به صورت کلی بزنیم. با توجه به نمودار ها مثلا اگر به تعداد 78 ایپاک طی میکردیم احتمالا وضعیت مدل ما بدتر از حالتی میشد که 50 ایپاک طی کردیم.

البته تعداد ایپاک ها به بهینه ساز نیز بستگی دارد. برخی بهینه ساز ها (مانند SGD) به ما این تضمین را می دهند که در هر ایپاک که طی می شود اوضاع بدتر نمی شود. در این نوع بهینه ساز ها افزایش تعداد ایپاک ها می تواند مفید باشد.

## 2- مقایسه توابع هزینه

مدل را با توابع هزینه ذکر شده در صورت سوال آموزش میدهیم و نمودار های مربوطه را رسم میکنیم.



شکل 72: توابع مربوط به خطا و دقت برای توابع خطای مختلف

جدول 7 : . نتیجه تست و ارزیابی برای توابع خطای مختلف

| Loss Function     | MSE      | MAE      | R <sup>2</sup> |
|-------------------|----------|----------|----------------|
| <b>MSE Loss</b>   | 0.155398 | 0.306636 | 0.825230       |
| <b>MAE Loss</b>   | 0.158602 | 0.296401 | 0.821627       |
| <b>Huber Loss</b> | 0.135335 | 0.281806 | 0.847794       |

با توجه به نتایج بالا تابع خطای huber از دو تابع دیگر عملکرد بهتری داشته است. این تابع در خطاهای کوچک مانند MES و در خطاهای بزرگ مانند MAE عمل می‌کند. رابطه این توابع به این شکل است:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \leq \delta \\ \delta\left(|a| - \frac{1}{2}\delta\right) & \text{if } |a| > \delta \end{cases} \quad a = y_i - \hat{y}_i$$

در روابط بالا یک مقدار قابل تنظیم است.

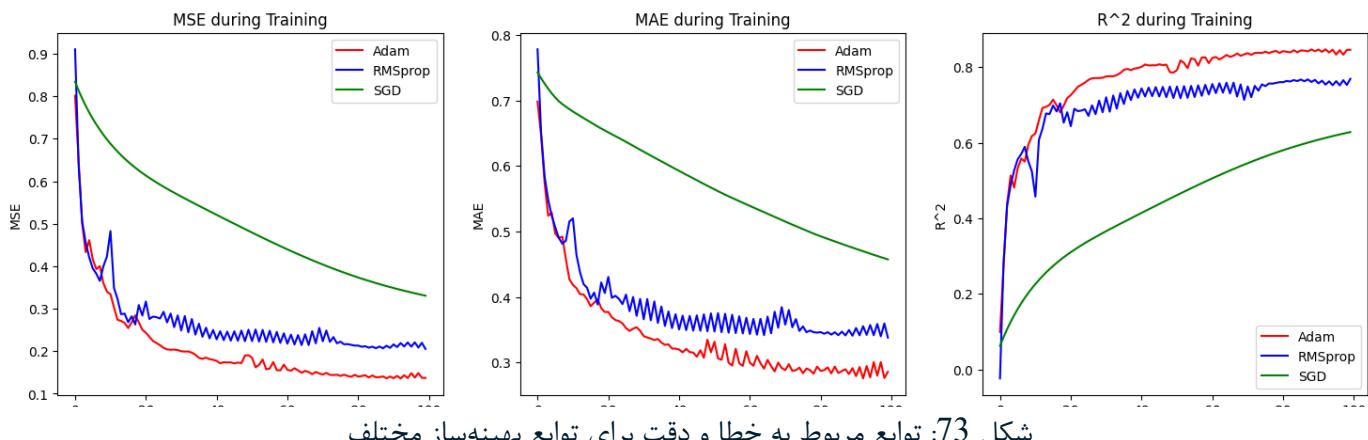
توضیح درباره هر تابع:

MAE : با توجه به اینکه این تابع قدر مطلق اختلاف مقادیر را جمع میکند، به خطاهای بزرگ و مقادیر پرت حساس نیست. همچنین گرادیان آن ثابت است و تغییرات ناگهانی ندارد.

MSE : این تابع توان دوم اختلاف مقادیر را باهم جمع میکند و به همین دلیل به خطاهای بزرگ و مقادیر پرت حساس تر است. گرادیان آن به صورت نرم تغییر میکند و در الگوریتم های بهینه سازی به بهبود همگرایی کمک میکند.

Hunber : با توجه به فرمول این تابع ترکیبی از MSE است و برای مواردی مناسب است که داده های پرت وجود دارند ولی نمی خواهیم تاثیر آنها به کلی نادیده گرفته شود.  
از این به بعد در دفعات بعدی از تابع Hunber استفاده خواهیم کرد.

### -3- مقایسه توابع بهینه ساز



## جدول 8 : نتیجه تست و ارزیابی برای توابع بهینه‌ساز مختلف

| Optimizer | MSE      | MAE      | R^2      |
|-----------|----------|----------|----------|
| Adam      | 0.137165 | 0.285921 | 0.845736 |
| RMSprop   | 0.205464 | 0.338023 | 0.768923 |
| SGD       | 0.330719 | 0.457457 | 0.628054 |

در ابتدا باید learning rate مربوط به هر تابع بهینه ساز را در بهترین حالت خودش تنظیم کنیم و سپس بهینه سازی را انجام دهیم.

با توجه به موارد بالا، بهینه ساز Adam بهترین عملکرد را بین تمام بهینه ساز ها دارد. اما نکته جالب توجه درباره بهینه ساز SGD است. این بهینه ساز پیوستگی و نرمی بیشتری از دو بهینه ساز دیگر دارد و همواره یکنوا است. ولی به طور کلی میتوان گفت سرعت همگرایی آن نسبت به دو بهینه ساز دیگر بسیار پایین است.

## 4-۲. جمع‌بندی

با توجه به بررسی هایی که انجام دادیم ویژگی CementComponent یا همان «جزء سیمانی» مهم ترین ویژگی در این دیتا است. بسیار منطقی است که هرچقدر سیمان بتن بیشتر باشد، مقاومت آن نیز بیشتر خواهد بود ولی از روی دادگان می توان موارد زیر را نیز استدلال کرد:

- 1- ویژگی CementComponent در ماتریس همبستگی، بیشترین همبستگی را با متغیر هدف دارد.
- 2- در Scatter Plot واضح است که با افزایش CementComponent مقدار مقاومت بتن نیز افزایش می یابد.

بنابراین این ویژگی مهم ترین ویژگی در این دیتا است.

بهترین تنظیمات برای این مدل بهینه ساز Adam ، تابع خطای Hunber و طی کردن 100 ایپاک بود که حدودا به  $R^2 = 85\%$  رسید.

نمی توان گفت همواره افزایش تعداد ایپاک باعث افزایش دقت مدل میشود. این موضوع خیلی مرتبط به بهینه ساز و تابع خطای است. فرض کنید که learning rate کنید که تابع بهینه ساز عدد بزرگی باشد، در این

حالات احتمال اینکه پس از هر ایپاک مقدار خطای زیاد تر از قبل شود کم نیست. درست است که روند کلی رو به کاهش خطای است ولی نوسانات روی روند کلی بسیار افزایش می یابد و ممکن است افزایش تعداد ایپاک باعث افزایش خطای شود. جدای از این افزایش تعداد ایپاک ها می تواند مدل را overfit کند و مدل نتواند داده های تست را به درستی پیش‌بینی کند. بنا بر این در حالت کلی نمی توان گفت افزایش تعداد ایپاک ها باعث کاهش خطای می‌شود ولی در این مثال افزایش تعداد ایپاک ها باعث کاهش خطای شد.

تابع هزینه Huber نسبت به بقیه توابع هزینه دقیق‌تری داشت. علت آن نیز این است که این تابع ترکیبی از MSE و MAE است و می تواند مقادیر خطای بزرگ و همچنین داده های پرت را به درستی هندل کند.

از بین بهینه ساز های موجود، بهینه ساز Adam از بقیه بهینه ساز ها عملکرد بهتری داشت و زودتر همگرا شد.

بنظر من یکی از چالشهای مهم در مدل سازی رگرسیون با شبکه های عصبی، این بود که ویژگی خوبی برای استناد وجود ندارد. با توجه به پیوسته بودن متغیر هدف احتمال اینکه یک ویژگی با متغیر هدف همبستگی بالای 0.9 داشته باشد بسیار بایین است و این موضوع کار را سخت می‌کند. جدای از این مورد همین ویژگی ها نیز با یکدیگر همبستگی دارند که کار را خراب می‌کند.

یک مسئله دیگر تنظیم هایپرپارامتر های شبکه است که برای پیدا کردن مقادیر بهینه آنها باید یک جستجوی وسیع روی فضای جستجو انجام داد.

## پرسش ۳ - پیاده سازی Adaline برای دیتاست IRIS

### ۱-۳. مقدمه

در این سوال مدل Adaline را به عنوان یکی از ساختار های پایه در شبکه های عصبی (به ویژه مسائل دسته بندی) پیاده سازی کرده و Train میکنیم و به تحلیل نتایج بدست آمده می پردازیم.

در این تمرین بر روی زیر مجموعه ای از دیتاست IRIS که شامل دو کلاس است کار میکنیم (همچنین از ۲ ویژگی از بردار feature ها برای جداسازی کلاس ها استفاده مینماییم).

### ۲-۱. آشنایی با Adaline

#### ۱. الگوریتم های Adaline و Madaline

یک شبکه با یک لایه مخفی و تقریبا مشابه شبکه Perceptron میباشد (جمع وزن دار ورودی Adaline ها و بایاس را محاسبه کرده و از تابع sign به عنوان activation function برای تولید خروجی ۱ یا -۱ استفاده میکند). برای یادگیری از قاعده Delta استفاده میشود که بر پایه روش Gradient descent میباشد. این شبکه میتواند یک Hyper-plane در فضای  $n$  بعدی مسئله پیدا کند که دو کلاس را از هم جدا کند (در صورتی که کلاس ها جدا پذیر باشند).

شبکه Madaline از تعدادی نورون Adaline تشکیل شده که خروجیشان با یک نورون M&P AND OR، ترکیب میشود. این شبکه توانایی جداسازی نمونه هایی از یک کلاس که داخل یک چند ضلعی هستند را دارد.

#### ۲. تفاوت اصلی MLP و Madaline

یکی از تفاوت های اصلی این دو شبکه نحوه یادگیری آنهاست. MLP از قاعده Gradient descent برای یادگیری استفاده میکند ولی روش سیستماتیکی برای آموزش شبکه Madaline نداریم.

همچنین Madaline توانایی حل مسائل Regression را ندارد (چون تنها خروجی بایپولار ایجاد میکند). در کل به دلیل اینکه در ساختار MLP آزادی بیشتری داریم (تعداد لایه ها و توابع فعال سازی مختلف و

...) میتوان مسائل با پیچیدگی بالاتر را نیز مانند دسته بندی کلاس ها با کلاستر های non-convex حل نمود.

### 3-3. آماده سازی دادگان

#### 1. دانلود دیتاست:

ابتدا دیتاست IRIS را دانلود کرده و feature ها را لود میکنیم. سپس برای شناخت کلاس ها و ویژگی ها لیست نام آنها را چاپ میکنیم.

```
iris = load_iris()

features = iris.data
feature_names = iris.feature_names

labels = iris.target
label_names = iris.target_names

print("Feature Names: ", feature_names)
print("Class Names: ", label_names)

Feature Names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Class Names: ['setosa' 'versicolor' 'virginica']
```

شکل 74: نام کلاس ها و ویژگی های دیتاست

در این بخش میخواهیم دو کلاس Setora و Versicolar را نگه داریم و کلاس Virginica را از دیتاست حذف کنیم. همچنین میخواهیم از ویژگی های Petal length و Petal width استفاده نماییم پس دو ویژگی دیگر را لازم نداریم و حذف شان میکنیم. با توجه به شکل 74 ما باید دو کلاس اول را نگه داریم و همچنین در فضای ویژگی ها نیز فیچر های 3 و 4 مورد نیاز ما هست و 1 و 2 را حذف میکنیم.

```
Filtered features (Petal Length & Width):
[[1.4 0.2]
 [1.3 0.2]
 [1.5 0.2]
 [1.4 0.2]
 [1.7 0.4]]
Filtered labels (Setosa: 0, Versicolor: 1):
[0 0 0 0 0]
```

شکل 75: بخش از داده های فیلتر شده

### 3. نرمال سازی و تقسیم بندی داده ها:

حال باید ویژگی ها را بین بازه 0 تا 1 نرمال سازی کنیم. همچنین داده هارا به دو بخش train و test با نسبت 70 به 30 تقسیم نماییم.

```
min_values = filtered_features.min(axis=0)
print("Minimum values:")
print(min_values)
max_values = filtered_features.max(axis=0)
print("Maximum values:")
print(max_values)

normalized_features = (filtered_features - min_values) / (max_values - min_values)
print("Total dataset size:", normalized_features.shape)

X_train, X_test, y_train, y_test = train_test_split(normalized_features, filtered_labels, test_size=0.3, stratify=filtered_labels, random_state=20)

print("Training data size:", X_train.shape)
print("Test data size:", X_test.shape)

Minimum values:
[1.  0.1]
Maximum values:
[5.1 1.8]
Total dataset size: (100, 2)
Training data size: (70, 2)
Test data size: (30, 2)
```

شکل 76: نرمال سازی و تقسیم داده ها

مطابق شکل 6/ از روش Min-Max برای نرمال سازی داده ها استفاده نموده ایم (با بدست اوردن بیشینه و کمینه هر ویژگی).

همچنین سایز دیتاست 100 است که با استفاده ازتابع attribute stratify train\_test\_split با روی لیبل ها (که باعث میشود نسبت اعضای کلاس ها در دو دسته تمرین و تست مشابه باشد تا مدل در تمرین دچار بایاس نشود) داده هارا به دو دسته تمرین و تست تقسیم نموده ایم.

### 4-3. پیاده سازی و آموزش مدل Adaline

#### 1. پیاده سازی الگوریتم Adaline:

در این بخش تابعی پیاده سازی میکنیم که به عنوان ورودی بردار های X و y (به ترتیب بردار ویژگی ها و لیبل های داده های تمرین) و همچنین نرخ یادگیری و تعداد epoch را میگیرد و به عنوان خروجی ماتریس های وزن و بایاس (در هر epoch) و همچنین دقت مدل و خطای آن در هر epoch را بر میگرداند.

نحوه پیاده سازی ما به گونه ای است که با هر سایز ورودی و خروجی ماتریس های وزن و بایاس با سایز متناسب تعریف شوند و تمرین داده شوند. برای همین ماتریس وزن W با سایز output\_dim \* input\_dim و بردار بایاس B با سایز (1 \* output\_dim) تعریف میشوند.

قاعده یادگیری در مدل Adaline (به صورت برداری) مطابق روابط زیر است که برای هر داده تمرینی اعمال میشوند:

$$\begin{cases} W_{new} = W_{old} + lr \cdot (t(p) - net) * X(p) \\ B_{new} = B_{old} + lr \cdot (t(p) - net) \end{cases} \quad p: sample\ number\ in\ training\ set$$

با توجه به درنظر گرفتن ورودی به صورت بردار سطیری و خروجی به صورت ستونی روابط برداری بالا با سایز هایی که برای ماتریس های وزن و بایاس در نظر گرفته ایم همخوانی دارند.

خروجی شبکه نیز از رابطه زیر تعیین میشود:

$$net = W \cdot X^T + b$$

رابطه بالا نیز نتیجه میدهد که net برداری ستونی با اندازه output\_dim است. حال برای محاسبه خطای شبکه Adaline از رابطه زیر استفاده میکنیم که خطای مربعات است که بر روی همه نمونه های ورودی میانگین گرفته شده است:

$$total\ error = \frac{1}{N} \sum_{p=1}^N \frac{1}{2} (t(p) - net(x(p), w, b))^2 \quad N: Size\ of\ training\ set$$

خروجی شبکه Adaline به صورت bipolar میباشد (1 یا -1) و با اعمال تابع sign بروی net خروجی آن بدست می آید. برای محاسبه دقت باید لیبل تارگت آن نمونه را با خروجی حاصل از نورون مقایسه کنیم و در صورت برابر بودن به تعداد پیشینی های درست شبکه در آن epoch یکی اضافه نماییم تا در نهایت دقت شبکه در epoch طی شده مشخص شود.

همچنین برای عملکرد مشابه تئوری های مطرح شده در کلاس لیبل های خروجی را که به صورت باینری هستند (0 یا 1) به بایپولار تغییر داده تا با خروجی بایپولار شبکه مطابقت داشته باشند.

با هر epoch یی که طی میشود این مقادیر خطا و دقت محاسبه شده و همراه ماتریس های وزن و بایاس در آن epoch ذخیره شده و در نهایت پس از طی شدن همه epoch ها به عنوان خروجی تابع باز گردانده میشوند.

## 2. آموزش مدل با نرخ های یادگیری متفاوت

حال مدل را با توجه به داده هایمان (سایز ورودی 2 و سایز خروجی 1) تنظیم کرده و با 3 نرخ یادگیری 0.001 و 0.005 و 0.02 آموزش میدهیم.

مقادیر اولیه ماتریس وزن ها و بایاس ها را نیز در داخل تابع بخش قبل با مقادیر تصادفی کوچکی در شروع کار مقداردهی میکنیم.

```

Training with learning rate: 0.001
Epoch 1/10, Error: 0.4896, Accuracy: 55.7143
→ Epoch 2/10, Error: 0.4754, Accuracy: 51.4286
Epoch 3/10, Error: 0.4622, Accuracy: 51.4286
Epoch 4/10, Error: 0.4499, Accuracy: 57.1429
Epoch 5/10, Error: 0.4383, Accuracy: 58.5714
Epoch 6/10, Error: 0.4273, Accuracy: 65.7143
Epoch 7/10, Error: 0.4169, Accuracy: 71.4286
Epoch 8/10, Error: 0.4069, Accuracy: 80.0000
Epoch 9/10, Error: 0.3974, Accuracy: 87.1429
Epoch 10/10, Error: 0.3883, Accuracy: 91.4286
Training with learning rate: 0.005
Epoch 1/10, Error: 0.4653, Accuracy: 77.1429
Epoch 2/10, Error: 0.4089, Accuracy: 78.5714
Epoch 3/10, Error: 0.3644, Accuracy: 92.8571
Epoch 4/10, Error: 0.3267, Accuracy: 95.7143
Epoch 5/10, Error: 0.2938, Accuracy: 98.5714
Epoch 6/10, Error: 0.2646, Accuracy: 100.0000
Epoch 7/10, Error: 0.2387, Accuracy: 100.0000
Epoch 8/10, Error: 0.2156, Accuracy: 100.0000
Epoch 9/10, Error: 0.1950, Accuracy: 100.0000
Epoch 10/10, Error: 0.1766, Accuracy: 100.0000
Training with learning rate: 0.02
Epoch 1/10, Error: 0.3960, Accuracy: 87.1429
Epoch 2/10, Error: 0.2571, Accuracy: 98.5714
Epoch 3/10, Error: 0.1721, Accuracy: 100.0000
Epoch 4/10, Error: 0.1183, Accuracy: 100.0000
Epoch 5/10, Error: 0.0843, Accuracy: 100.0000
Epoch 6/10, Error: 0.0628, Accuracy: 100.0000
Epoch 7/10, Error: 0.0492, Accuracy: 100.0000
Epoch 8/10, Error: 0.0406, Accuracy: 100.0000
Epoch 9/10, Error: 0.0352, Accuracy: 100.0000
Epoch 10/10, Error: 0.0317, Accuracy: 100.0000

```

شکل 77: مقادیر خطا و دقت شبکه در هر epoch با نرخ های یادگیری مختلف

### 5-3. نمایش و تحلیل نتایج

#### 1. نمایش خط جداگانه:

حال که مدل را بر روی داده های train به ازای 3 نرخ یادگیری متفاوت آموخته داده ایم میخواهیم عملکرد و کارایی هر کدام را بررسی نماییم. برای این کار در این بخش خطوط تصمیم گیری هر مدل (که همان مقادیر وزن و بایاس آن را تعیین میکنند) را در دو نمودار که برروی یکی داده های آموزشی و بر دیگری داده های تست رسم شده اند ترسیم میکنیم تا مشخص شود هر مدل در جداسازی کلاس ها (برروی داده های آموزشی و تست) تا چه میزان موفقیت آمیز عمل می نمایید.

برای برازش خط جدا کننده از رابطه خط زیر استفاده میکنیم که از بردار وزن و بایاس شبکه حاصل میشود:

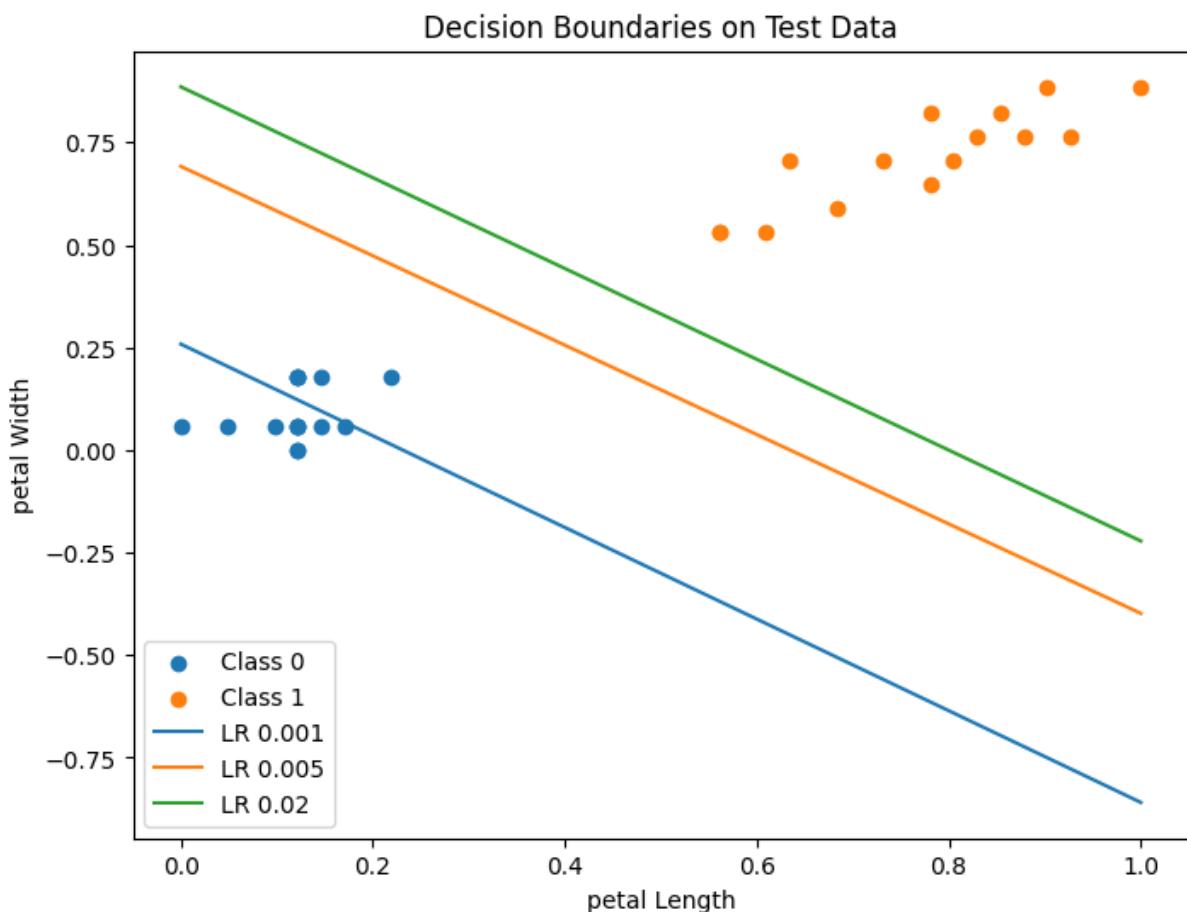
$$W[0,0].X[0] + W[0,1].X[1] + b = 0$$

$$\text{line equation} \rightarrow Y = -\frac{W[0,0].X + b}{W[0,1]}$$

برای این کار از مقادیر نهایی تمرین داده شده هر شبکه (مقادیر حاصل شده در epoch آخر) استفاده میکنیم.



شکل 78: خطوط جدا کننده بر روی دادگان آموزش (epoch) 10



شکل 79: خطوط جداکننده بر روی دادگان تست (epoch 10)

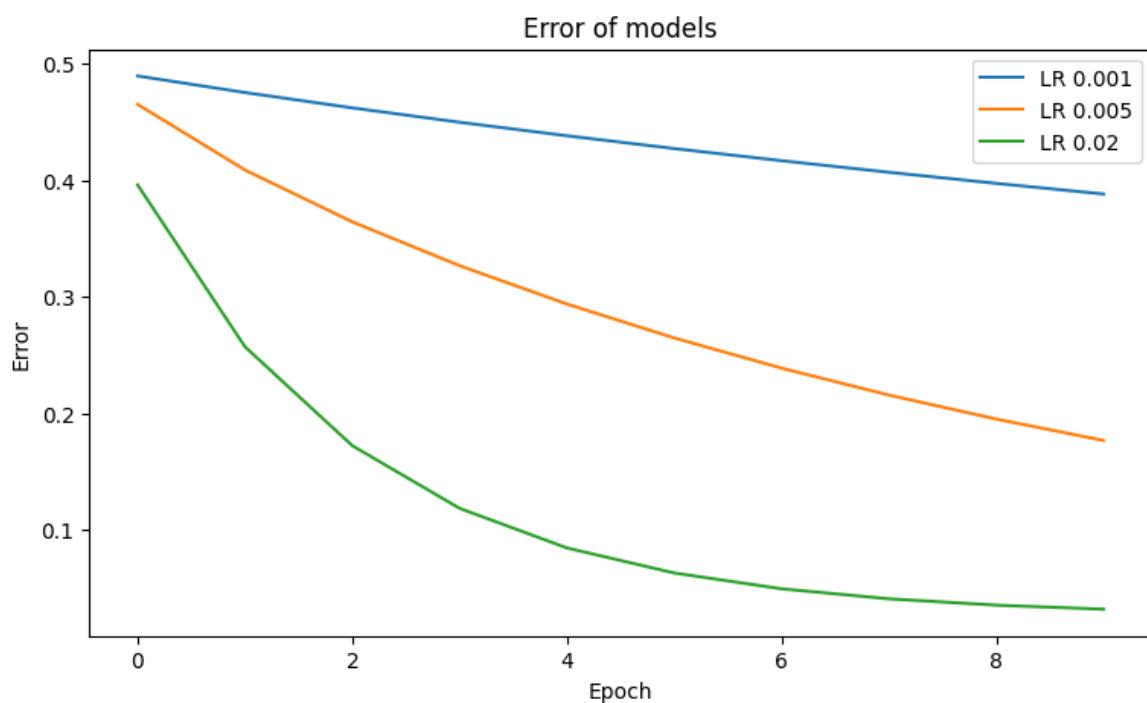
تحلیل نتایج:

در شکل 78 مشاهده میکنیم که خطوط جداکننده در مدل ها با نرخ های یادگیری 0.02 و 0.005 به خوبی نمونه های هر دو کلاس را جدا کرده و در فضای بین آن ها قرار گرفته است اما مدل با نرخ یادگیری 0.001 نتوانسته است خطی را که کلاس ها را از هم تفکیک کند پیدا کند. این امر در تصویر 3-4 هم مشاهده میشود. در آنجا دیدیم که دو مدل اول با گذشت چند epoch اولیه به دقت 100 درصد بروی داده های تمرینی رسیدند که به این معناست که کلاس همه نمونه های تمرینی را فرا گرفته و میتواند به درستی طبقه بندی شان کند ولی در مورد مدل سوم به این صورت نبود و دقت در نهایت به حدود 91 درصد میرسید.

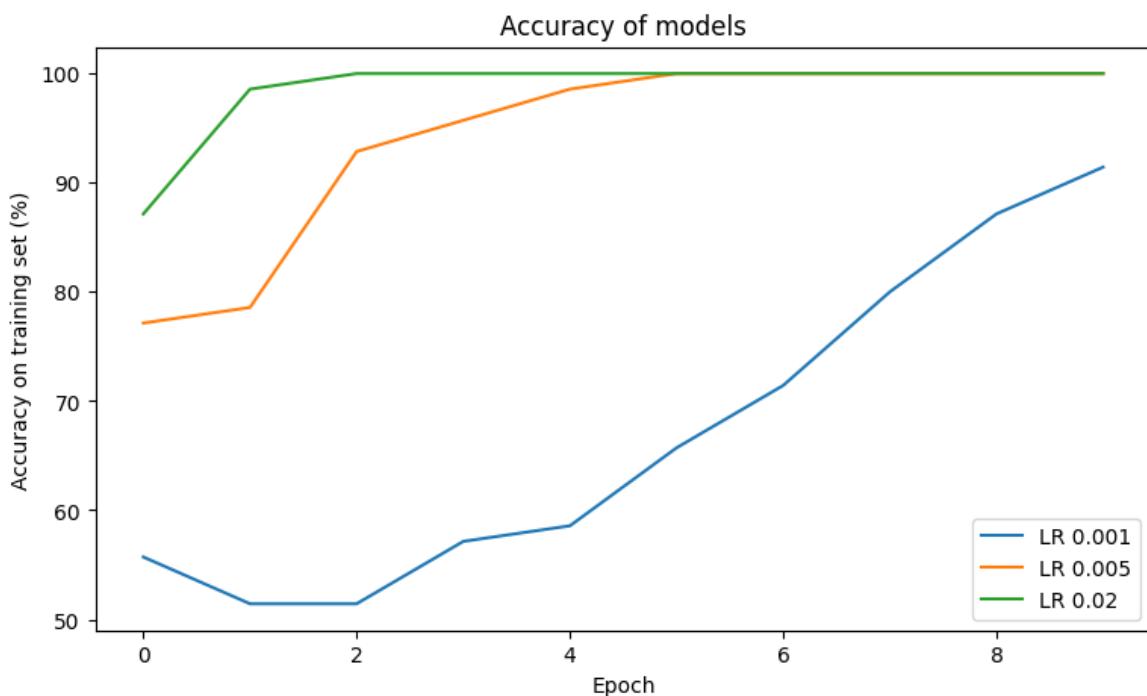
همچنین شکل 79 نیز نشان میدهد دو مدل ابتدایی علاوه بر اینکه دادگان آموزشی را یاد گرفته اند بروی دادگان تست نیز عملکرد عالی دارند و نمونه های دو کلاس را به صورت کامل از هم تفکیک کرده اند. ولی مدل با نرخ یادگیری 0.001 همانطور که انتظار میرفت بروی دادگان تست نیز عملکرد مطلوبی ندارد.

## 2. نمودار خطأ و دقت:

حال با ترسیم مقادیر خطأ و دقت هر شبکه در طی epoch های مختلف در کدام شبکه ها و همچنین سرعت یادگیری شان بدست می آوریم.



شکل 80: نمودار خطأ بر حسب epoch طی شده



## شکل 81: نمودار دقت شبکه بر حسب epoch طی شده

### تحلیل نتایج:

در شکل 80 که خطای شبکه هارا ترسیم کرده است به وضوح مشخص است با اینکه در ابتدا هر سه مدل تقریبا از یک نقطه شروع کرده و خطای مشابهی دارند ولی دو مدل با نرخ یادگیری 0.02 و 0.005 سرعت یادگیری به مراتب بالاتری نسبت به مدل با نرخ یادگیری 0.001 از خود نشان داده اند و نمودار خطایشان با طی شدن هر epoch با شبیه تندتری در حال کاهش است.

مدل 0.001 نیز در حال یادگیری است و مرتبا خطایش کاهش میابد ولی در epoch های محدودی که طی کرده ایم به عملکرد دو مدل ابتدایی نمیتواند برسد.

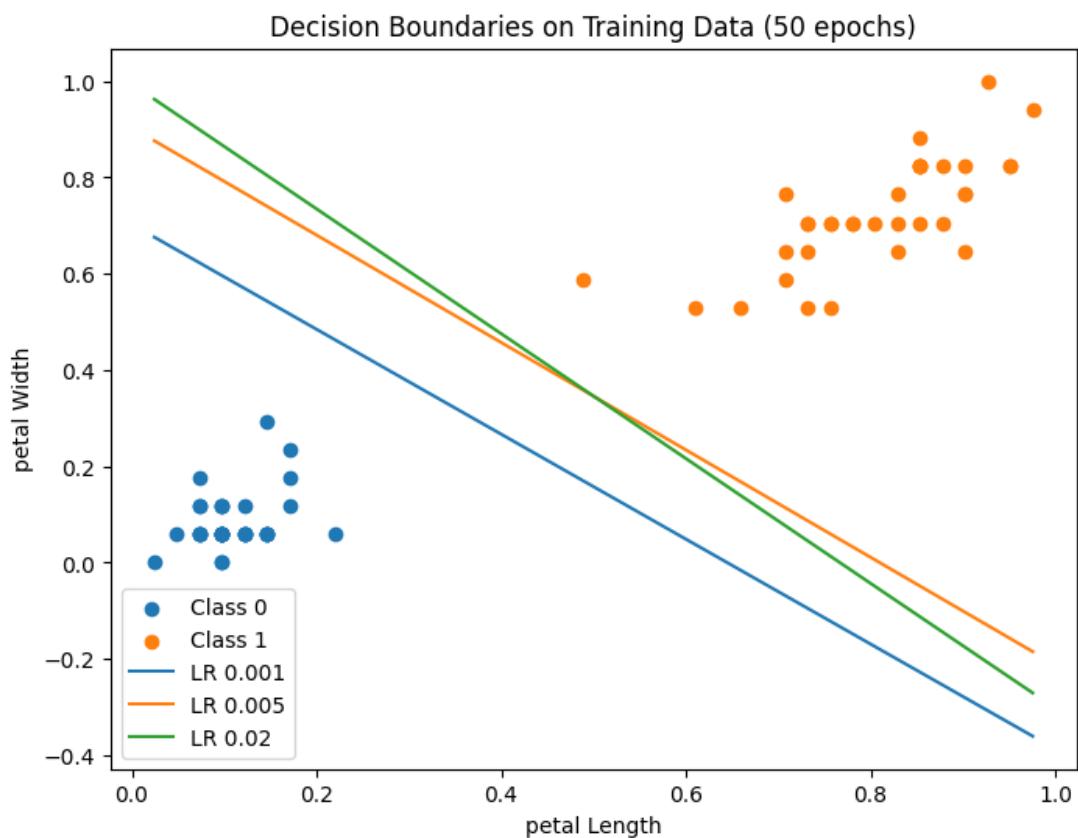
این موضوع در شکل 81 نیز مشخص است. دو مدل ابتدایی به سرعت دادگان را یاد گرفته و خط جدا سازشان در فضای بین دو کلاس قرار گرفته و دقت به 100 درصد میرسد ولی شبکه سوم به سرعت کمی در حال یادگیری است و با هر epoch دقت آن مقدار کمی افزایش میابد (خط جداساز هر بار مقدار خیلی کوچکی در صفحه حرکت میکند).

### 3. تحلیل نتایج:

#### همگرایی مدل ها:

همانطور که در بخش های قبل هم گفته شد مدل ها با نرخ یادگیری 0.02 و 0.005 همگرا شدند و به دقت 100 درصد هم بروی دادگان آموزشی و هم دادگان تست رسیدند ولی مدل با نرخ یادگیری 0.001 به کندی در حال آموزش بود و در 10 epoch طی شده در نهایت همگرا نشد.

علت اصلی این امر learning rate کوچک بود که سرعت یادگیری مدل را به شدت کاهش داده بود. برای نشان دادن این موضوع تعداد epoch های آموزش را به 50 رسانده تا ببینیم عملکرد مدل سوم چگونه میشود.

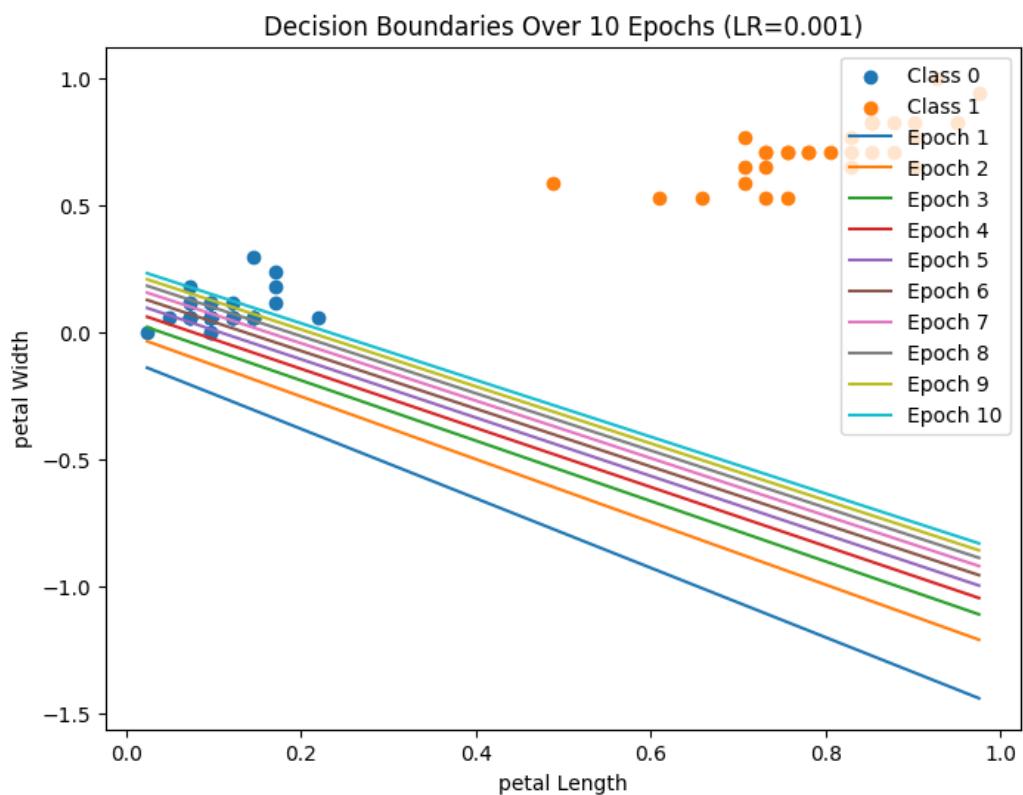


شکل 82: خطوط جداکننده بر روی دادگان آموزشی (epoch 50)

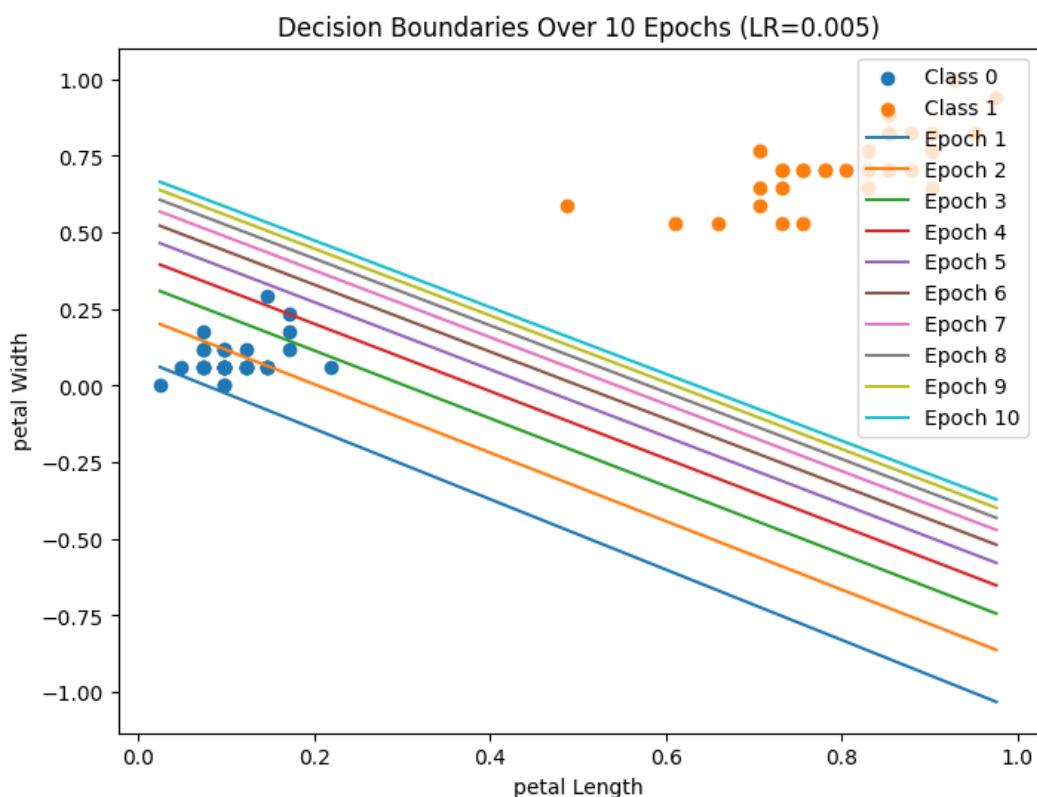
همانطور که میبینیم مدل سوم نیز در نهایت همگرا شده و دو کلاس را از هم تفکیک کرده است.

همچنین خطوط دو مدل اول نیز بیشتر از قبل به هم نزدیک شده و هر دو به یک مقدار بھینه ای نزدیک میشوند.

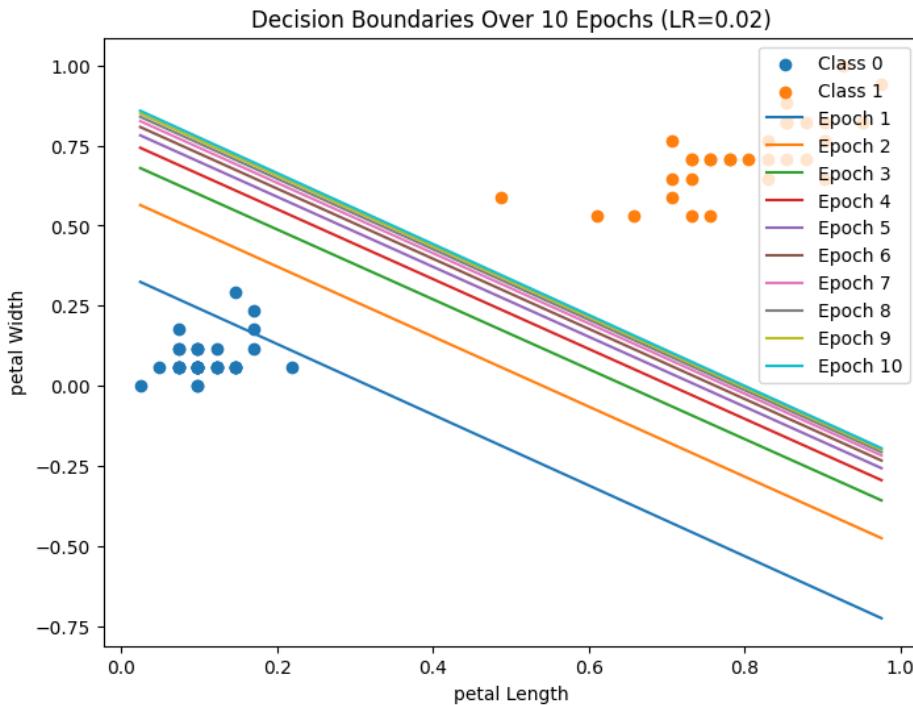
برای نشان دادن بهتر تاثیر نرخ یادگیری بر سرعت پیشرفت مدل تابعی نوشته ایم که داده های وزن و بایاس هر مدل (که در هر epoch 10 ذخیره شده اند) را بر روی یک صفحه رسم کند.



شکل 83: پیشرفت مدل با نرخ یادگیری 0.001



شکل 84: پیشرفت مدل با نرخ یادگیری 0.005



شکل 85: پیشرفت مدل با نرخ یادگیری 0.02

در شکل 85 میبینیم که مدل در epoch های ابتدایی با گام های بزرگی به سمت جواب بهینه حرکت کرده و هرچه نزدیک تر به آن میشویم تغییرات پارامتر ها نیز کمتر میشود.

از طرفی در شکل 83 میبینیم که تغییرات پارامتر ها بسیار کوچک است و خط به ارامی در صفحه به سمت فضای بین دو کلاس حرکت میکند.

تأثیر جداسازی خطی بر عملکرد Adaline:

از آنجایی که Adaline یک مدل خطی است تنها در صورتی امکان یافتن پاسخ مناسب برای حل مسئله Classification را دارد که مرز کلاس ها غیر خطی نباشد زیرا قادر به یادگیری آن نخواهد بود.

در این حالت خطای مدل در حین آموزش ممکن است در epoch های متوالی کاهش نیابد چون به صورت کلی صفحه ای وجود ندارد که بتواند کلاس هارا به طور کامل از هم جدا کند پس Adaline هم نمیتواند آن را یاد بگیرد.

تأثیر نرخ یادگیری:

در بخش های قبل تاثیر نرخ یادگیری بر عملکرد مدل را به تفصیل بررسی کردیم. دیدیم که نرخ یادگیری کوچک سرعت یادگیری را به شدت کاهش میدهد (همچنین میتواند باعث شود در local optimum هایی گیر کنیم و نتوانیم از آنها خارج شویم) از طرفی نرخ یادگیری هر مقداری هم نمیتواند بزرگ شود چون این میتواند خود باعث واگرایی مدل بشود.

از طرفی در این سوال به نظر میرسد نرخ یادگیری  $0.005$  بهتر از  $0.02$  عمل میکند. چون با دقت به نمودار خط های جدا کننده میبینیم که در اولی خط جداکننده با نزدیک ترین اعضای هر یک از کلاس ها فاصله قابل قبولی دارد (یعنی margin خط با هر یک از کلاس ها تقریباً مساوی است) ولی در نرخ  $0.02$  خط به طرف داده های کلاس  $1$  که اعضای بیشتری دارد بایاس شده و نزدیک تر به نمونه های آن قرار دارند.

بنابراین انتخاب نرخ یادگیری مناسب علاوه بر اینکه میتواند باعث همگرایی مدل به جواب مناسب در epoch های کم بشود میتواند باعث شود مدل ما robust تر یاد بگیرد و به طرف کلاس خاصی بایاس نشود (البته این مشکل را با روش های دیگری مثل تابع فعال سازی جایگزین sign نیز میتواند حل نمود).

## پرسش ۴ – آموزش اتوانکودر و طبقه بندی با دیتاست MNIST

### ۲-۴. دانلود و پیش پردازش داده ها

ابتدا داده هارا دانلود کرده و یک transformation pipeline برای پیش پردازش داده ها مینویسیم تا به صورت خواسته شده در بیایند.

```
▶ transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Lambda(lambda x: x.view(-1))
])

train_dataset = torchvision.datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = torchvision.datasets.MNIST(root='./data', train=False, transform=transform, download=True)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=64, shuffle=False)

→ 100% |██████████| 9.91M/9.91M [00:11<00:00, 894kB/s]
100% |██████████| 28.9k/28.9k [00:00<00:00, 62.1kB/s]
100% |██████████| 1.65M/1.65M [00:01<00:00, 1.24MB/s]
100% |██████████| 4.54k/4.54k [00:00<00:00, 7.92MB/s]
```

شکل 86: دانلود و پیش پردازش داده ها

در شکل 86 نحوه تعریف این transformation مشخص است (ToTensor) داده ها را بین 0 و 1 نرمال کرده و همچنین بردار ورودی را Reshape کرده و به صورت بردار در می آوریم. برای دیدن نحوه عملکرد یکی از بردار هارا برای نمونه چاپ کرده و همچنین شکل آن را بررسی میکنیم که به صورت برداری در آمده باشد.

همچنین برای سادگی لود کردن دادگان آموزش و تست در ادامه، یک loader برای هر کدام نوشته ایم که داده هارا در بسته های 64 تایی به ما برمیگرداند.

### ۳-۴. طراحی و پیاده سازی مدل

حال میخواهیم دو اتوانکودر با اندازه های مختلف را طراحی کرده و آموزش دهیم.

#### بخش اول : اتوانکودر ها

در این بخش دو اتوانکودر با مشخصات داده شده در صورت مسئله را پیاده سازی میکنیم و آموزش میدهیم. اتوانکودر اول فضای ویژگی ها با سایز 784 را به 8 خروجی و دومی به 4 خروجی کاهش میدهد.

```

class AutoEncoder8(nn.Module):
    def __init__(self):
        super(AutoEncoder8, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(784, 128),
            nn.ReLU(),
            nn.Linear(128, 8),
            nn.ReLU()
        )

        self.decoder = nn.Sequential(
            nn.Linear(8, 128),
            nn.ReLU(),
            nn.Linear(128, 784)
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

```

شکل 87: طراحی اتوانکودر با خروجی 8 نورون

تصویر بالا نحوه طراحی شبکه اتوانکودر را نشان میدهد. این شبکه از یک بخش انکودر و یک بخش دیکودر تشکیل شده است. بخش انکودر ورودی 784 تایی را در دو لایه به ترتیب به سایز 128 و 8 کاهش میدهد. بخش دیکودر بر عکس عمل کرده و سعی میکند از روی بردار ویژگی های کوچک بدست آمده بردار اولیه را باز تولید کند. بخش Forward pass همین کار را کرده و بردار ورودی را ابتدا از انکودر و سپس از دیکودر عبور میدهد تا خروجی با ورودی اولیه مقایسه شود. (در تمرین مدل) اتوانکودر با 4 نورون خروجی هم درست مشابه مدل بالا تعریف میشود.

برای آموزش مدل یک تابع train\_autoencoder مینویسیم که مدل را همراه هایپر پارامترها (نرخ یادگیری و همچنین تعداد epoch ها) میگیرد و با تابع خطای MSE و اپتیمایزر Adam آموزش میدهد. خطای را با مقایسه خروجی مدل و بردار تصویر ورودی محاسبه میکنیم. همچنین در هر epoch که طی میکنیم خطای را محاسبه کرده تا بتوانیم نمودار خطای epoch را داشته باشیم.

نرخ یادگیری را بعد از چندین بار سعی و خطای برابر 0.0005 قرار دادیم (به نظر میرسید برای نرخ های بالاتر مدل حول نقطه بهینه نوسان میکند و خطای کم نمیشود و برای نرخ های کوچک تر هم در local optimum ها گیر میکرد). تعداد epoch های آموزش نیز برای هر دو 50 می باشد.

## بخش دوم : طبقه بندی با انکودر

حال یک مدل برای طبقه بندی تعریف میکنیم که از بخش انکودر مدل های قبل برای استخراج ویژگی feed-ها استفاده میکند (همچنین وزن های بخش encoder را freeze میکند تا در حین آموزش تنها لایه forward چهار تغییر وزن شود و بخش انکودر بدون تغییر باقی بماند).

برای مدل اول از انکودر با خروجی 8 تایی استفاده میکنیم که با یک لایه با تابع فعال سازی ReLU به 4 کاهش و سپس با یک لایه با تابع SoftMax به 10 کلاس خروجی میرسیم. مدل دوم از انکودر 4 نورونی و یک لایه 10 تایی خروجی با تابع SoftMax استفاده میکند.

نکته ای که وجود دارد این است که تابع خطای Cross entropy loss softmax خودش تابع را ببروی خروجی به صورت داخلی اعمال میکند. بنابراین در تعریف مدل دیگر نیازی نیست که بر لایه خروجی این تابع فعالساز را قرار دهیم.

```
class ClassifierFromAutoEncoder(nn.Module):
    def __init__(self, autoencoder, encoder_size):
        super(ClassifierFromAutoEncoder, self).__init__()

        self.encoder = autoencoder.encoder

        # Freeze encoder
        for param in self.encoder.parameters():
            param.requires_grad = False

        if encoder_size == 8:
            self.classifier = nn.Sequential(
                nn.Linear(8, 4),
                nn.ReLU(),
                nn.Linear(4, 10)
            )
        elif encoder_size == 4:
            self.classifier = nn.Sequential(
                nn.Linear(4, 10)
            )

    def forward(self, x):
        features = self.encoder(x)
        features = features.view(features.size(0), -1)
        output = self.classifier(features)
        return output
```

شکل 88: طراحی طبقه بند با اتوانکودر

برای آموزش این شبکه ازتابع خطای Cross Entropy Loss استفاده میکنیم. تعداد epoch ها را 50 قرار داده و نرخ یادگیری را 0.01 تنظیم میکنیم. مدل را برای هر دو انکودری که بالا داشتیم آموزش میدهیم.

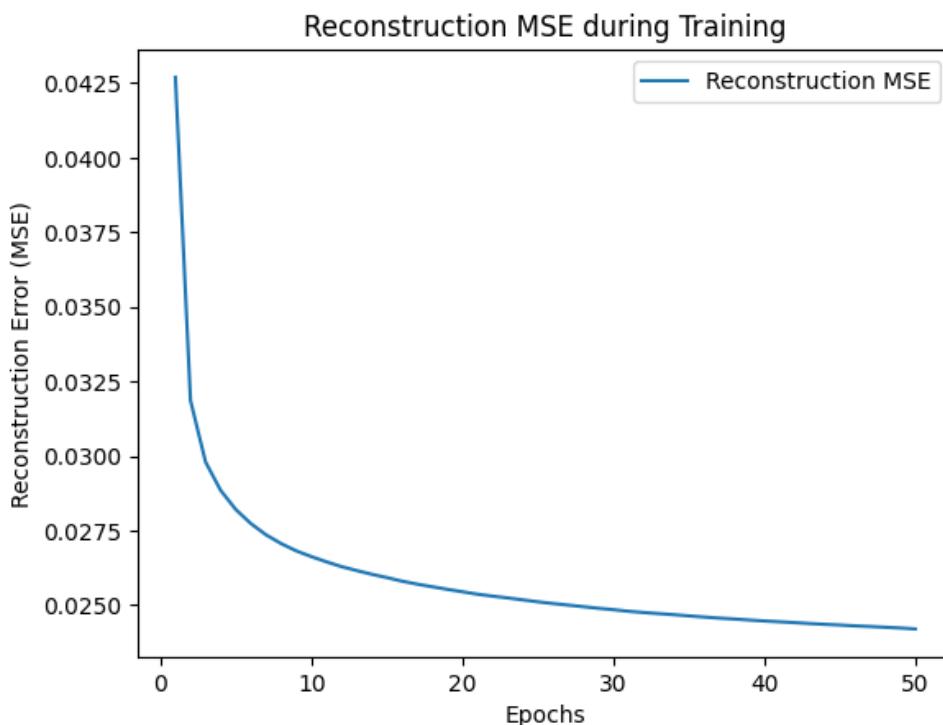
نرخ های یادگیری بالاتری که تست شدند مدل را همگرا به دقت های بالا نمیکردند و برای نرخ های یادگیری پایینتر (برا مثال 0.005) مدل بعد از گذشت چند epoch در یک local optimum که با نقطه بهینه ای که بالاتر پیدا کردیم فاصله نسبتا زیادی هم داشت گیر میکرد و نمیتوانست از آن خارج شود.

همچنین در هر epoch بی که طی میکنیم تعداد پیشビینی های درست مدل بر روی داده های آموزش را برای بدست آوردن دقت مدل در آن epoch میشماریم. همچنین در پایان epoch مدل را در حالت Evaluation قرار داده و داده های تست را به آن داده و خروجی آن را با لیبل ها تطبیق داده تا دقت مدل بر روی داده تست نیز بدست بیاید.

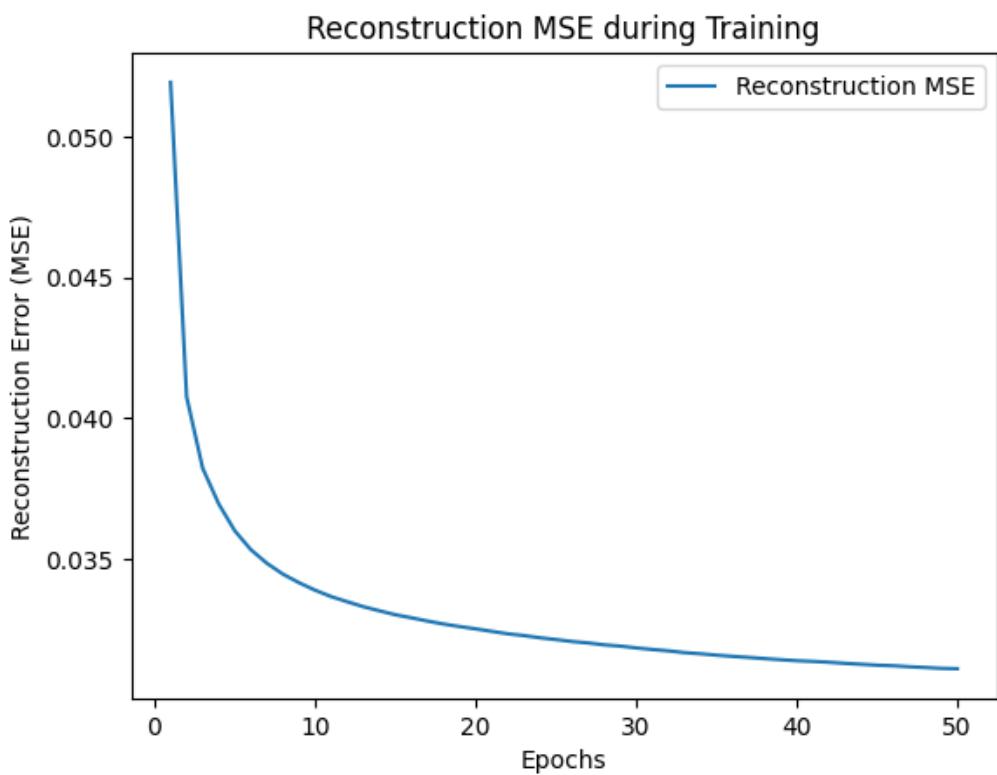
## 4-4 . نتایج و تحلیل

### 1. نتایج:

در این بخش میخواهیم نمودار خطای بازسازی ورودی ها در اتوانکودر را بر حسب طی شده برای هر دو اتوانکودر بخش 4-3 رسم کنیم. این خطاهای بر روی دادگان آموزشی و در حین تمرین مدل بدست آمده اند.



شکل 89: نمودار خطای بازسازی اتوانکودر با 8 نورون خروجی



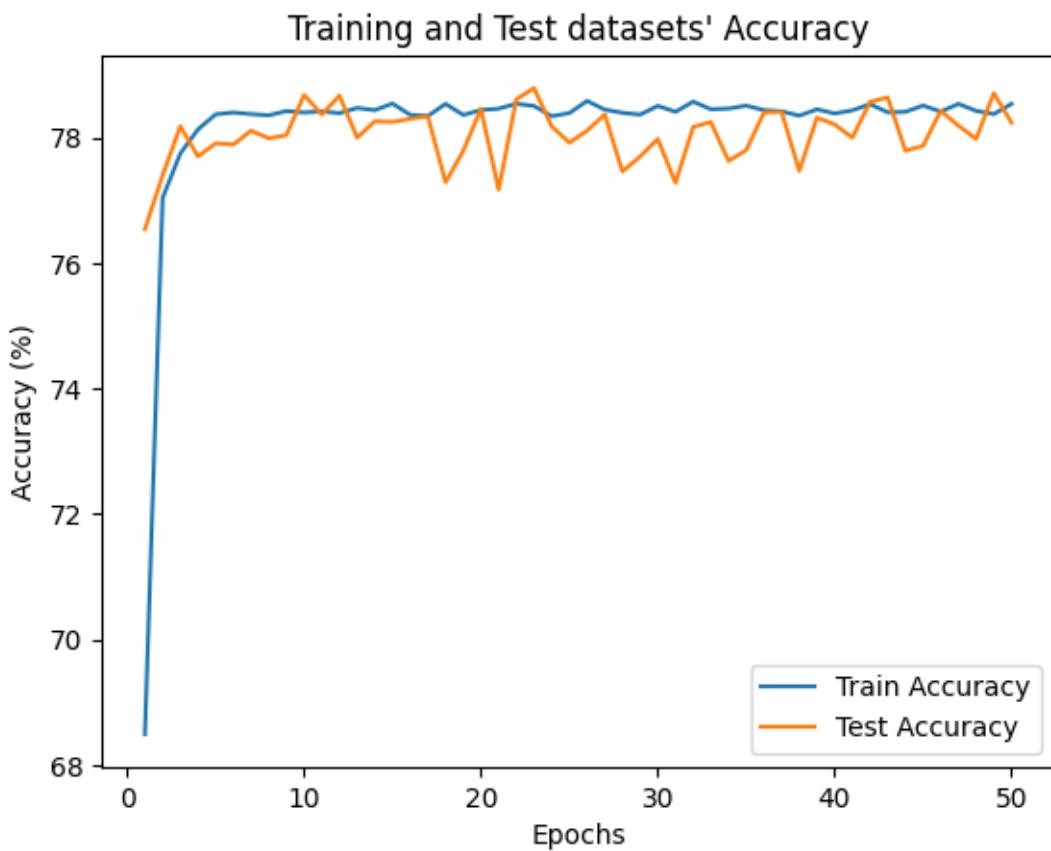
شکل 90: نمودار خطای بازسازی اتوانکودر با 4 نورون خروجی

در شکل 89 میبینیم که بعد از طی شدن 50 epoch 89 خطای بازسازی اتوانکودر با 8 خروجی به 0.0242 رسیده است در صورتی که برای اتوانکودر با 4 خروجی به مقدار 0.0311 میرسد.

حال میخواهیم دقت مدل های طبقه بند نهایی که در بخش 3-4 آموزش داده ایم را رسم نماییم (دقت را در هر epoch هم برای داده های آموزشی و هم داده های تست بدست آورده بودیم. برای هر مدل این دقت هارا بروی یک نمودار رسم میکنیم تا مقایسه ای هم بتوانیم انجام دهیم.)



شکل ۹۱: نمودار دقت مدل با اتوانکوادر با ۸ نورون خروجی



شکل 92: نمودار دقت مدل با اتوانکودر با 4 نورون خروجی

تعداد پارامتر های مدل ها در جدول 9 ذکر شده اند.

جدول 9 : تعداد پارامتر های مدل های بررسی شده

| تعداد پارامتر های مدل            | انکودر با 8 نورون خروجی | انکودر با 4 نورون خروجی |
|----------------------------------|-------------------------|-------------------------|
| اتوانکودر(انکودر + رمزگشایی)     | 203800                  | 202772                  |
| طبقه بندی(انکودر + feed-forward) | 101598                  | 101046                  |

## 2. تحلیل و مقایسه:

### مقایسه خطای بازسازی:

همانطور که در شکل های 86 و 87 مشخص است خطای بازسازی در مدل با 8 نورون کمتر بود و این شبکه اتوانکودر ورودی هارا بهتر بازسازی میکند. علت استفاده از انکودر کاهش ابعاد ورودی بدون از دست دادن اطلاعات ضروری آن است (هدف این است که correlation های بین ورودی هارا از بین برده تا بعد فضای کاهش دهیم).

حال در اینجا انکودر اول بعد فضا را از 784 به 8 رسانده و انکودر دوم به 4. علت این که انکودر دوم خطای بیشتری از خود نشان داد میتواند این امر باشد که در این کاهش بعد بیشتر نسبت به انکودر اول علاوه بر کورلیشن میان ورودی ها برخی اطلاعات اصلی ورودی را نیز از دست داده ایم. برای همین در بازیابی ورودی اولیه نمیتواند به خوبی اولی عمل کند.

در حقیقت ما باید دنبال بیشترین کاهش بعدی هستیم که اطلاعات موجود در ورودی را تا بیشترین حد ممکن حفظ کند یا به عبارتی ورودی از روی خروجی انکودر (با شبکه دیکودر) به خوبی قابل بازتولید باشد.

#### مقایسه دقت طبقه بندی:

دقت طبقه بندی در epoch نهایی برای هر دو مدل در جدول 10 آمده است.

جدول 10: دقت طبقه بندی مدل ها در اخیرین epoch

| دقت طبقه بندی(%) | مدل با فضای نهان 8 | مدل با فضای نهان 4 |
|------------------|--------------------|--------------------|
| دادگان آموزش     | 81.49              | 78.54              |
| دادگان تست       | 82.24              | 78.24              |

همانطور که مبینیم مدل با فضای نهان 8 بر روی هر دو دیتاست آموزش و تست دقت طبقه بندی بالاتری دارد.

این امر میتواند به دلیل عملکرد بهتر این انکودر باشد زیرا در بخش قبل دیدیم که انکودر با خروجی 8 خطای بازسازی کمتری داشت پس فضای نهان آن اطلاعات بیشتری از ورودی را شامل میشود که میتواند باعث این عملکرد بهتر در طبقه بندی شود.

البته باید توجه داشت که این کاهش ابعاد به مقدار زیادی نیز به دقت آسیب نزدیک است و به طور حدودی 4 درصد از دقت مدل کاهش یافته ولی همچنان در یک محدوده عملکردی قرار میگیرد (هر دو به طور آماری تقریبا 2 تصویر از 10 ورودی را اشتباه تشخیص میدهند).

نکته ای که در شکل 92 دیده میشود این است که مدل در همان epoch های اولیه به دقت نهایی میرسد و پس از آن تغییر خاصی در دقت روی دادگان آموزش نداریم (این امر به دلیل شبکه forward تک لایه ساده ای است که این مدل دارد). البته در epoch های بعدی دقت روی دادگان تست چهار نوسان میشود که این امر نیز بخاطر بیش برازش مدل برروی داده های ورودی اتفاق می افتد.

شکل 91 که مربوط به مدل 8 نورونی است نشان میدهد که به دلیل شبکه feed-forward دو لایه این مدل epoch های بیشتری طول میکشد تا این مدل به دقت نهایی خود برسد و به سرعت مدل دیگر عمل نمیکند. در این مدل هم بیش برآذش را میتوانیم ببینیم چون دقت بر دادگان تست در برخی از epoch ها کاهش میابد علی رغم اینکه دقت بر روی دادگان آموزشی اندکی افزایش یافته است.

#### مقایسه تعداد پارامتر ها:

از آنجایی که بخش feed-forward هر دو شبکه بسیار ساده و کوچک است بخش عمدۀ پارامتر های هر دو را انکودر ها تشکیل میدهند.

اتوانکودر با 8 نورون خروجی همانطور که انتظار میرفت پارامترهایش کمی بیشتر از دیگری است (چون لایه آخر آن 8 نورونی است در صورتی که دیگری 4 نورون در لایه خروجی انکودر دارد).

در مدل طبقه بند نیز تعداد پارامترهای مدل با لایه نهان 8 تایی بیشتر از دیگر مدل است.(اگرچه که این اختلاف کم است چون اکثر پارامتر های این مدل ها میان لایه ورودی و اول انکودر که یک fully-connected از 784 به 128 است وجود دارند).

این تعداد پارامتر بیشتر همانطور که در بخش قبل دیدیم باعث عملکرد بهتر مدل نیز شده است.

#### بیش برآذش مدل ها:

بله همانطور که در بخش های قبل هم گفته شد بعد از طی شدن epoch های اولیه دقت مدل روی دادگان آموزشی تغییر آنچنانی نمیکند حال آنکه دقت آن روی دادگان تست نوسانات زیادی دارد. این بدان معناست که مدل در حال یادگیری نایقینی ها و نویزهای روی داده های آموزشی میباشد که اگرچه دقت آموزش را اندکی کاهش میدهد ولی باعث عملکرد ضعیف تر مدل برروی دادگان تست میشود.

برای کاهش آن میتوان از روش هایی مثل اضافه کردن Dropout و یا Regularization به لایه هایمان استفاده نمود.

#### بخش امتیازی: بهبود عملکرد مدل

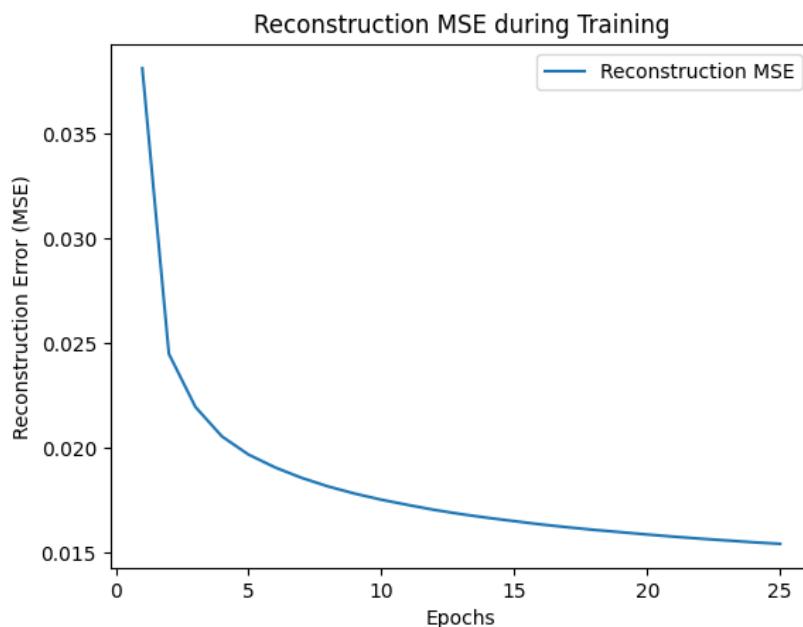
حال با توجه به نکات بالا میخواهیم مدلی طراحی کنیم که ضعف های مدل های قبل را برطرف کرده و عملکرد بهتری داشته باشد.

از بخش auto-encoder شروع میکنیم. هر دو اتوانکودر قبلی 2 لایه (شامل لایه خروجی) داشتند. برای بهبود عملکرد آن و اینکه انکودر بتواند الگوهای پیچیده تر در داده های ورودی را پیدا کند یک انکودر 3 لایه طراحی میکنیم. همچنین لایه خروجی را 16 نورونی قرار میدهیم تا اطلاعات کمتری از دست بدھیم.

انکودر: (ReLU) 16 , (ReLU) 64 , (ReLU) 256 , 784

دیکودر: (Linear) 784 , (ReLU) 256 , (ReLU) 64 , 16

حال با همان هایپرپارامتر های حالت قبل اتوانکودر جدید را آموزش میدهیم.



شکل 93: نمودار خطای بازسازی اتوانکودر 3 لایه با 16 نورون خروجی

همانطور که میبینیم خطای بازسازی به مراتب کاهش یافته است (حدود 0.015 در epoch 25 ام).

حال باید شبکه feed-forward را طراحی کنیم. در اینجا یک شبکه دولایه قرار میدهیم. لایه اول 32 نورون (با تابع ReLU) و لایه دوم 10 نورون (با تابع Softmax).

هدف از این کار این است که بعد فضای نهان را با لایه اول Expand کنیم تا این ویژگی های فشرده حاصل از انکودر باز شده و مدل الگوهای متنوع تری را بتواند یاد بگیرد. لایه دوم نیز وظیفه تصمیم گیری میان 10 کلاس را بر عهده دارد.

همچنین برای بهبود مشکل بیش برآش بعد از لایه اول در اپ اوت با مقدار 0.2 اضافه نموده ایم.

جدول 11: دقت طبقه بنده در مدل پیشنهادی

| داده   | آموزش | تست   |
|--------|-------|-------|
| دقت(%) | 87.7  | 93.08 |

جدول 12: تعداد پارامتر های مدل پیشنهادی

| مدل             | اتوانکودر ۳ لایه | طبقه بندی(انکودر + feed-forward +) |
|-----------------|------------------|------------------------------------|
| تعداد پارامترها | 437664           | 219322                             |



شکل 94: نمودار دقت مدل با اتوانکودر با 16 نورون خروجی

همانطور که میبینیم دقت مدل روی دادگان تست به 93 درصد رسید (حدود 11 درصد بهتر از مدل بخش قبلی).

همینطور در شکل 94 میبینیم که دقت مدل روی دادگان تست دچار نوسان زیادی نشده و روند رو به بهبودی را همواره دارد که نشان میدهد با افزودن Dropout توانسته این مشکل بیش برآذش را رفع نماییم. البته باید توجه داشت که طبق جدول 12 تعداد پارامترهای این مدل نسبت به قبلي ها تقریباً دو برابر است که نقطه ضعف آن محسوب میشود.