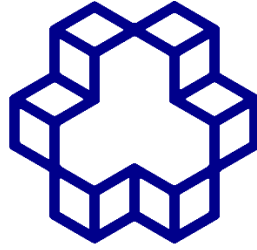


Google drive: <https://drive.google.com/drive/folders/1n8uWALN39KWm7Zrq6tpUICi1efUUQn4D?usp=sharing>

Github: <https://github.com/AlibagheriNejad/ML-AliYari>



دانشگاه صنعتی خواجه نصیرالدین طوسی

گزارش نیمچه پروژه سوم درس یادگیری ماشین

علی باقری نژاد

۴۰۲۰۲۸۵۴

ازمیان سوال ۱ و ۲، سوال دوم برای حل انتخاب شد

سوال ۲.....	۴
طرح مسئله.....	۴
معرفی روش.....	۵
نتیجه گیری.....	۷
نوآوری ها.....	۷
پیاده سازی.....	۸
سوال ۳.....	۹
چالش ها و راه حل.....	۹
معماری شبکه ارائه شده.....	۱۰
پیاده سازی مدل.....	۱۱
عملکرد مدل.....	۱۶
عملکرد در مقابل threshold ها مختلف.....	۱۸
عملکرد مدل بدون ازبین بردن عدم توازن در داده و نویز گیری.....	۱۸

سوال ۲

طرح مسئله

SVM به صورت بهینه ای می تواند مسائل طبقه بندی خطی و غیر خطی را با استفاده از support vector حل نماید. درضمن با استفاده از regularization هم می توان از مسئله overfitting روی داده های آموزش نیز جلوگیری کرد. به دلیلی عملکرد مناسب این الگوریتم روی طبقه بندی دو کلاسه، تلاش های بسیاری انجام شده که بتوان این الگوریتم را برای طبقه بندی چند کلاسه هم استفاده نمود.

به صورت کلی روش های پیشنهاد شده برای حل مسئله MSVM را می توان به ۳ بخش تقسیم نمود:

۱. **روش های اکتشافی:** این روش ها از SVM دو کلاسه به عنوان اساس روش خود استفاده کرده و مسئله

طبقه بندی چند کلاسه را به چندین مسئله دو کلاسه تبدیل می کنند. از جمله این روش ها می توان به روش های one-vs-one یا one-vs-all اشاره کرد.

۲. **روش های اصلاح خطا کد:** این روش از کدهای اصلاح خطا استفاده می کند. در این روش ها، مسئله به

چندین مسئله طبقه بندی دو کلاسه تبدیل می شود که زیرمسائل طبقه بندی دو کلاسه توسط یک ماتریس کدینگ ساخته شده، گروه بندی می شوند.

۳. **روش های تک ماشین:** این روش ها صرفاً با استفاده از یک تابع تبدیل و بهینه کردن آن، مرز بندی های

همه کلاس ها را به صورت هم زمان انجام می دهند.

روش های کنونی از مشکلات متعددی رنج می برند.

- روش هایی که بیس طبقه بندی دو کلاسه دارند، باید تعداد زیادی مسئله بهینه سازی را انجام دهند؛ که این باعث افزایش هزینه و طولانی شدن زمان حل مسئله می شود.
- روش های تک ماشین، بر اساس misclassification های کلاس مشاهده شده با کلاس های دیگر می باشند. جمع کردن خطاهای misclassification که از مشاهدات خطا چندین کلاس بدست نقش

پررنگ تری نسبت به misclassification هایی دارند که تنها از مشاهده یک کلاس بدست می آیند. یعنی تاثیر خطا دسته اول تاثیر بیشتری روی بهینه سازی می گذارد که این ویژگی مناسبی برای طبقه بندی چند کلاسه نیست.

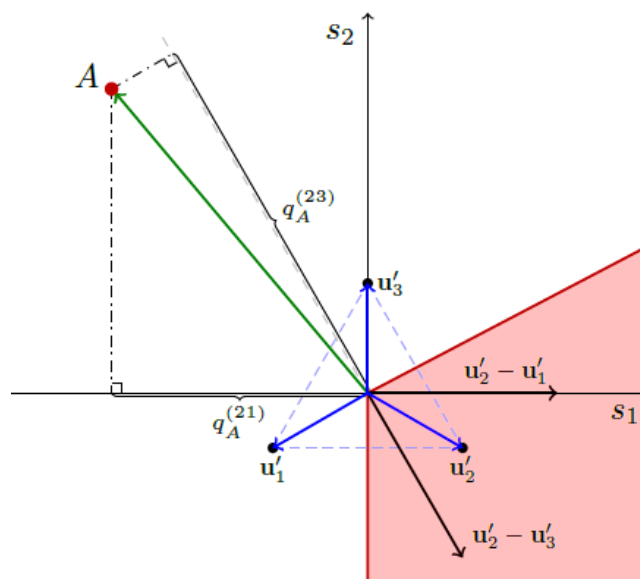
معرفی روش

مدل پیشنهاد شده یک ساختار encoding ساده را برای فرمول بندی کردن MSVM پیشنهاد می دهد که ابعاد مسئله را به یک مسئله طبقه بندی $K-1$ (K تعداد کل کلاس ها می باشد) بعدی کاهش می دهد و بدین صورت تنها یک مسئله بهینه سازی را حل می نماید. با استفاده از یک تابع flexible hinge و یک نرم l_p برای خطا، تابع هزینه GenSVM سه روش MSVM را با هم ترکیب می کند و در عین حال آن روش ها را گسترش می دهد.

نگاشت از فضای ورودی به فضای simplex توسط minimize کردن misclassification error حل می شود. این خطا توسط محاسبه فاصله مختصات از مرز تصمیم گیری در فضای simplex محاسبه می شود.

محاسبه خطا

اگر نقطه x را در فضای ورودی در نظر بگیریم، در فضای simplex خواهیم داشت $s'_i = x'_i W + t'$ که در واقع نگاشت نقطه در فضای simplex می باشد.



تصویر بالا را در نظر بگیرید. نقطه A مربوط به کلاس ۲ می‌باشد. برای محاسبه خطا مربوط به این نقطه، ابتدا

فاصله این نقطه نسبت به محورهای $u_k - u_j$ محاسبه می‌نماییم (uها تعریف خاص خودشان را دارند که در این گزارش

آورده نشده‌اند). $q_i^{(kj)} = (x_i'W + t')(u_k - u_j)$. با استفاده از این فرمول فاصله نقطه محاسبه می‌شود. در این

الگوریتم قصد بر این است که تا جای ممکن q در بیشترین مقدار خود باشد. برای این کار از تابع خطا Huber-hinge

استفاده می‌شود. البته باید این نکته را ذکر کرد که خطا محاسبه شده، وزن دار می‌باشد.

$$h(q) = \begin{cases} 1 - q - \frac{\kappa+1}{2} & \text{if } q \leq -\kappa \\ \frac{1}{2(\kappa+1)}(1 - q)^2 & \text{if } q \in (-\kappa, 1] \\ 0 & \text{if } q > 1, \end{cases}$$

تابع بالا، تابع خطا Huber-hinge را نشان می‌دهد. در این جا اگر فاصله بیشتری از مقدار ۱ نسبت به بردار

محور کلاس واقعی (در این جا کلاس ۲) باشد، تابع هیچ خطایی ندارد. اما هر چه که از این خط فاصله بیگیریم و به

سمت مخالف بردار کلاس ۲ حرکت کنیم، به همان نسبت خطا خواهیم داشت.

نرم p مربوط به هر خطا به صورت زیر محاسبه می‌شود:

$$\left(\sum_{\substack{j=1 \\ j \neq y_i}}^K h^p(q_i^{(y_i j)}) \right)^{1/p}.$$

نرم p به این دلیل تاثیر regularizationی که دارد، به عبارت خطا اضافه می‌شود.

در این الگوریتم، از p و K به عنوان پارامتر استفاده می‌شود.

$$L_{\text{MSVM}}(\mathbf{W}, \mathbf{t}) = \frac{1}{n} \sum_{k=1}^K \sum_{i \in G_k} \rho_i \left(\sum_{j \neq k} h^p(q_i^{(kj)}) \right)^{1/p} + \lambda \text{tr } \mathbf{W}'\mathbf{W}$$

با استفاده از فرمول بالا، مقدار وزن مربوط به هر داده وارد بازی می شود.. در مقاله اثبات می شود که تابع هزینه بالا، یک تابع convex می باشد.

در نهایت با استفاده از $\hat{y}_{n+1} = \arg \min_k \|s'_{n+1} - u'_k\|^2$, for $k = 1, \dots, K$. ، مقادیر لیبیل update می شوند.

بهینه سازی

برای بهینه سازی و کمینه کردن خطا، از الگوریتم $\text{iterative majorization (IM)}$ استفاده می شود.

نتیجه گیری

این مقاله، روش خود را با سه روش دیگر مقایسه کرد و گزارش داده که روش طراحی شده، نسبت به روش های دیگر از لحاظ پیشبینی خروجی، ۵٪ بهبود عملکرد داشته است که GenSVM را به عنوان بهترین مدل در میان مدل های تک ماشین تبدیل می کند. در مقایسه با ۵ مدل دیگر نیز، GenSVM دارای کمترین مدت زمان لازم برای آموزش بود. این درحالی بود که تعداد پارامترهای آموزش داده شده، ۱۸ برابر دیگر مدل ها بود.

برتری ها:

- سرعت محاسباتی بالا در عین داشتن پارامترهای بیشتر
- دارای قابلیت warm start که باعث کاهش زمان آموزش می شود (کاهش زمان آموزش در مقاطع ابتدایی آموزش مدل)
- انعطاف پذیری مدل (انعطاف پذیری تابع هزینه تعریف شده)
- قابلیت Generalization

نوآوری ها

از جمله خلاقیت های بکار گرفته شده در ساختار مدل می توان به موارد زیر اشاره کرد:

- تعریف فضای simplex برای کاهش ابعاد
- استفاده از Huber-hinge و نرم p برای انعطاف پذیر کردن مدل و افزایش generalization

- استفاده از الگوریتم IM که قابلیت warm start را به الگوریتم می دهد

پیاده سازی

مدل GenSVM به صورت یک تابع از پیش آماده در کتابخانه gensvm موجود است و برای استفاده از آن کتابخانه را بر روی محیط colab نصب کردیم. البته برای ادامه دادن محاسبات این مدل، باید یک نسخه قبلی -scikit-learn را هم نصب می کردیم.

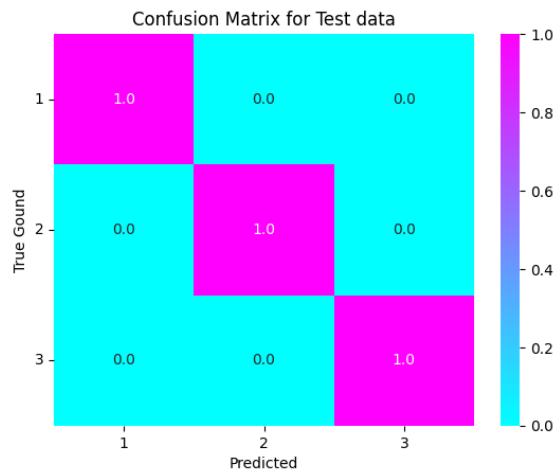
برای مقایسه عملکرد مدل در تمرین با مقاله، از دیتاست IRIS استفاده شد. در قدم اول به نسبت ۱ به ۹، داده تست از داده آموزش جدا شد. قبل از استفاده از داده برای آموزش مدل، داده ها با روش standard scaler، نرمال شدند. سپس با استفاده از GridsearchCV، با استفاده از 10-fold، بهترین مقادیر پارامترهای زیر انتخاب شد:

$p: \{1,2\}, \kappa: \{0,1,5,10\}, C: \{0.1,1,10,100\}$

```
Best parameters found: {'coef': 10, 'kappa': 1, 'p': 2}
Best cross-validation score: 0.99
Accuracy of best parameters on Test data: 100.00
```

تصویر بالا، بهترین پارامترها، عملکرد مدل را برای داده های آموزش (10-fold CV) نمایش می دهد. از طرفی دقت بهترین مدل برای داده های تستی که از قبل جدا شده بودند گزارش شده است.

مدل	دقت گزارش شده
مقاله	٪ ۹۶.۰۰
تمرین	٪ ۹۸.۵۱



تصویر بالا، ماتریس درهم ریختگی را برای داده های تست به تصویر کشیده است. دقت بدست آمده نسبت به مقاله بیشتر می باشد. میتوان این مسئله را بدین صورت تحلیل کرد که از آنجایی که در این جا از 10-fold استفاده کردیم و مقاله از 5-fold و از طرفی در این تمرین، قبل از آموزش داده ها، یک بخش آزمون جدا گذاشته شد و بر ۹۰ درصد باقی مانده، برای K-fold CV استفاده کردیم.

سوال ۳

چالش ها و راه حل

چالش های موجود در بحث تشخیص تقلب، با توجه به مقاله، عبارت اند از:

- رفتار کلاه برداری متغیر (دینامیک): یعنی کلاه برداری ها از روش های مغییری استفاده می کنند که کار خود را قانونی جلوه دهند.
- کمبود دیتاست: دیتاست برای تراکنش های مالی حساب ها به سختی وجود دارد
- عدم تعادل دیتاست: دیتاست ها، عموماً دارای unbalancing می باشند و داده های مربوط به تراکنش های قانونی بسیار بیشتر از کلاه برداری ها می باشد.
- انتخاب ویژگی مناسب
- معیار مناسب برای سنجش عملکرد مدل تشخیص کلاه برداری

- روند انتخاب نمونه

این مقاله برای مقابله با چالش‌ها موجود، از راه‌حل‌های زیر استفاده کرده است:

- ✓ اعمال PCA روی داده‌های برای حذف ویژگی‌های بی‌تاثیر
- ✓ استفاده از AutoEncoder برای از بین بردن نویز در دیتاست
- ✓ بهره‌گیری از over sampling (روش SMOTE) برای متعادل کردن دیتاست
- ✓ استفاده از یک MLP به عنوان classifier
- ✓ در نظر گرفتن معیار recall به همراه معیار accuracy

معماری شبکه ارائه شده

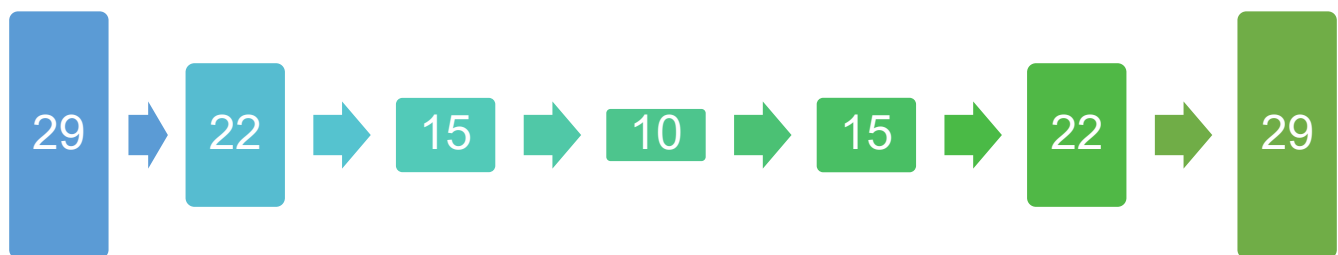
قبل از توضیح معماری شبکه باید بیان شود که این مقاله، قبل از دادن داده‌ها به شبکه برای آموزش، بر روی آن‌ها over sampling اعمال کرد و سپس شبکه‌ها را طراحی کرد.

به صورت کلی دو شبکه عصبی در این مقاله ارائه شده است:

۱. ساختار Auto Encoder

۲. ساختار MLP

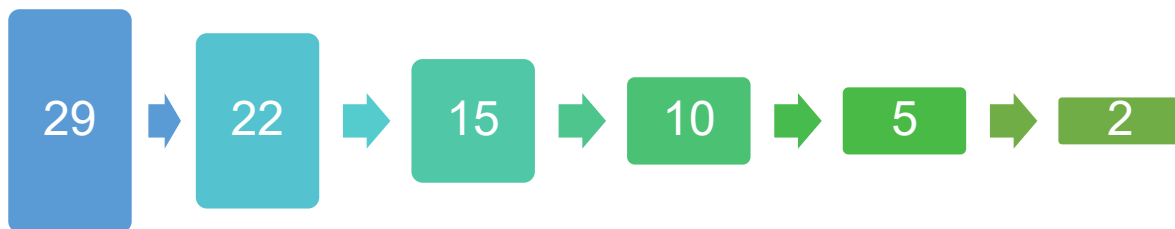
Auto Encoder برای از بین بردن نویز در داده استفاده شده است.



شکل بالا ساختار استفاده شده برای Auto Encoder را به نمایش گذاشته است. هدف این است که با کاهش ابعاد، ویژگی های بی ربط و به درد نخور حذف شده و با دوباره ساختن داده ورودی توسط ویژگی های latent، داده هایی بدون نویز بازتولید شوند.

در این ساختار، از MSE به عنوان تابع هزینه استفاده شده است. دلیل استفاده از این تابع این است که اختلاف بین خروجی و ورودی را به کمترین مقدار ممکن برساند. یعنی خروجی ای که ایجاد می شود، شبیه ورودی شبکه باشد.

ساختار بعدی مربوط به classifier می باشد که هدف آن تشخیص تراکنش قانونی از تراکنش کلاه برداری می باشد.



شکل بالا شماییک ساختار MLP طراحی شده برای طبقه بندی را به نمایش می گذارد. به عنوان تابع هزینه، در این ساختار، از تابع SoftMax استفاده شده است. این تابع هزینه برای طبقه بندی چند کلاسه مناسب می باشد.

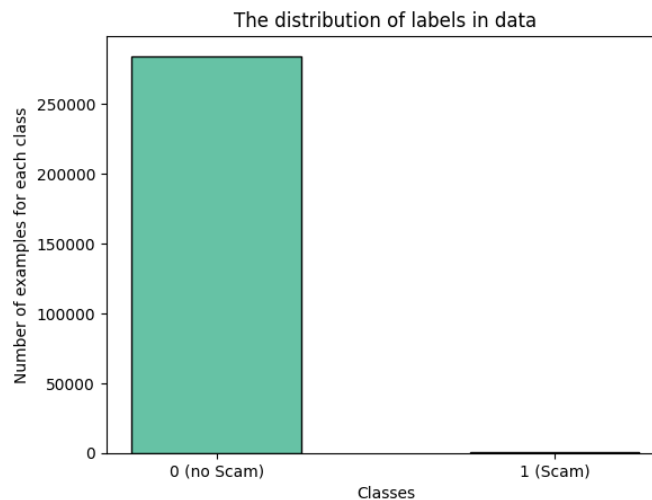
پیاده سازی مدل

دیتاست را از Kaggle دانلود کرده و با استفاده از کتابخانه panda آن را می خوانیم. سپس ستون Time را از داده حذف می کنیم چراکه مقاله هم از این داده استفاده ای نکرده است.

```
Shape of X: (284807, 29)
Data distribution of Class:
Class
0      284315
1         492
```

در شکل بالا تعداد داده ها، ابعاد داده ورودی و تعداد داده های هر کلاس به خوبی نمایش داده شده است.

شکل زیر به خوبی پخش داده برای هر کلاس را به نمایش گذاشته است.



با توجه به شکل بالا، به بالانس کردن داده بسیار نیاز داریم.

مقاله برای نمایش عملکرد مدل خود، یکبار بدون Smote و AE طبقه بندی را انجام داد و بار دیگر با این ابزار ها مدل خود را ایجاد کرد و سپس آن ها را با هم مقایسه نمود. در این تمرین، مدل بدون استفاده از SMOTE و AE را آموزش نمی دهیم و صرفاً مدل را با استفاده از این دو ابزار ایجاد می کنیم.

قدم بعد، استفاده از SMOTE برای بالانس کردن داده است. برای استفاده از این روش، از کتابخانه imblearn استفاده می شود.

```
Shape of RESAMPLED X: (568630, 29)
Data distribution of RESAMPLED Class:
Class
0      284315
1      284315
```

بعد از استفاده از SMOTE، تعداد داده ها به شکل بالا در خواهد آمد. در شکل زیر هم پخش داده جدید را

ملاحظه می کنید.



حال از این بخش دیتا جدید برای ادامه مسیر استفاده می کنیم.

مرحله بعد، تقسیم داده ها به بخش های آموزش و آزمون تقسیم می کنیم. همانند مقاله، نسبت تقسیم داده ها

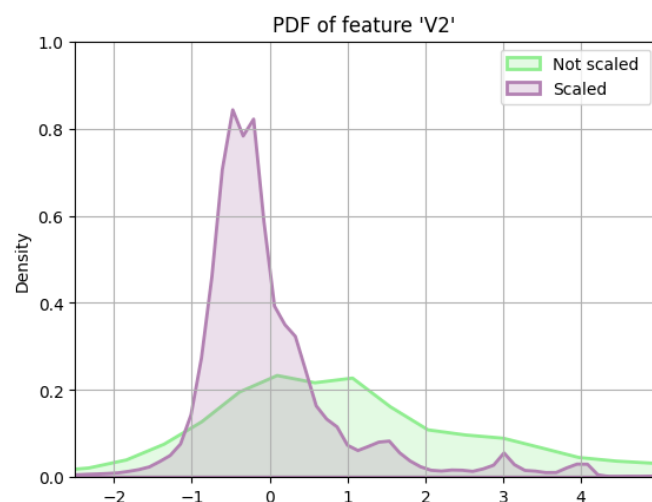
۸۰٪ به ۲۰٪ مربوط به داده های آزمون و ۸۰٪ مربوط به داده های آموزش.

```
Shape of train input data: (454904, 29)
Shape of test input data: (113726, 29)
```

بعد از تقسیم، تعداد داده های هر بخش را ملاحظه می نمایید.

مرحله بعد از تقسیم داده، نرمالایز کردن آن می باشد. برای نرمالایز کردن از تابع `StandardScaler` استفاده

می شود.



شکل بالا، PDF داده V2 را نمایش می دهد. رنگ سبز برای پخش داده قبل از اعمال scaler و رنگ بنفش

نمایانگر پخش داده پس از آن می باشد.

قدم بعدی، اعمال AE به داده ها می باشد. برای این کار از کتابخانه tensorflow استفاده می نماییم.

Model: "Autoencoder"		
Layer (type)	Output Shape	Param #
EC_1 (Dense)	(None, 22)	660
EC_2 (Dense)	(None, 15)	345
Latent (Dense)	(None, 10)	160
DC_1 (Dense)	(None, 15)	165
DC_2 (Dense)	(None, 22)	352
OUT (Dense)	(None, 29)	667
Total params: 2349 (9.18 KB)		
Trainable params: 2349 (9.18 KB)		
Non-trainable params: 0 (0.00 Byte)		

ساختار AE ایجاد شده را در تصویر بالا مشاهده می نمایید. حال باید این شبکه را بر روی داده های ورودی

آموزش دهیم. از تابع ReLU به عنوان تابع فعالسازی تمامی لایه ها استفاده شد.

برای تابع هزینه، همانند مقاله، از MSE بهره گرفته شد. برای بهینه سازی از الگوریتم Adam استفاده کردیم.

در صورت سوال خواسته شده بود که مدل با بهترین وزن ها (کمترین خطا اعتبارسنجی) ذخیره شود. برای این

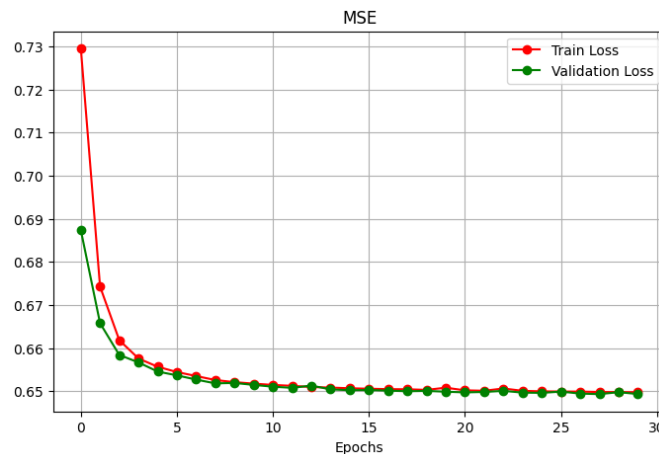
کار می توان از callbackی به نام earlystopping استفاده کرد. این callback را به هنگام آموزش مدل استفاده

می کنیم که هم بهترین مدل را بازگرداند و هم از آموزش بیش از حد مدل خودداری کند. از طرفی با استفاده از

ModelCheckpoint، بهترین وزن ها را هم ذخیره می نماییم.

```
x_train_scaled
x_train_scaled
epochs = 30
batch_size = 128
validation_data = (x_test_scaled, x_test_scaled)
callbacks = [checkpoint, early_stopping]
```

بالا، پارامترهای مربوط به آموزش AE نمایش داده شده است.



شکل بالا، نمودار خطا (MSE) مربوط به مدل، حین آموزش را نمایش می دهد.

بعد از denoise کردن داده ها، باید طبقه بندی را انجام دهیم.

Model: "CLASSIFIER"

Layer (type)	Output Shape	Param #
Dense_1 (Dense)	(None, 22)	660
Dense_2 (Dense)	(None, 15)	345
Dense_3 (Dense)	(None, 10)	160
Dense_4 (Dense)	(None, 5)	55
Classifier (Dense)	(None, 2)	12

=====
Total params: 1232 (4.81 KB)
Trainable params: 1232 (4.81 KB)
Non-trainable params: 0 (0.00 Byte)

ساختار شبکه به صورت بالا می باشد. برای تمامی لایه ها به جز لایه آخر، از تابع فعالسازی ReLU استفاده

شد. همانند مقاله، در لایه آخر از SoftMax استفاده کردیم.

برای تابع هزینه از SparseCategoricalCrossentropy استفاده کردیم (به دلیل این که از دو نورون برای

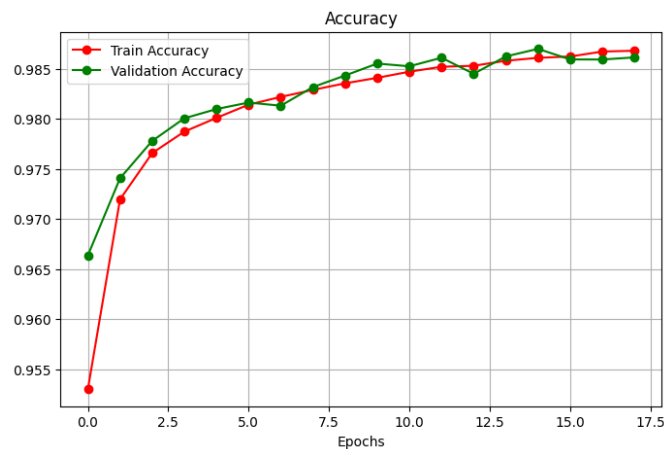
لایه آخر استفاده شده بود). از تابع بهینه سازی Adam نیز به عنوان optimizer برای این MLP استفاده می کنیم.

نکته مهم این است که قبل از آموزش دادن مدل، باید داده های آموزش و اعتبارسنجی خود را به AE داده تا

عمل denoise روی آن ها انجام شود و سپس مدل MLP را آموزش می دهیم.

به دلیل استفاده از `earlystopping`، با اینکه epoch ۳۰ برای آموزش در نظر گرفته شده بود، به دلیل

پیشترت نداشت. با استفاده از این `callack`، بهترین مدل بر اساس دقت اعتبارسنجی، باز گرداننده شد.

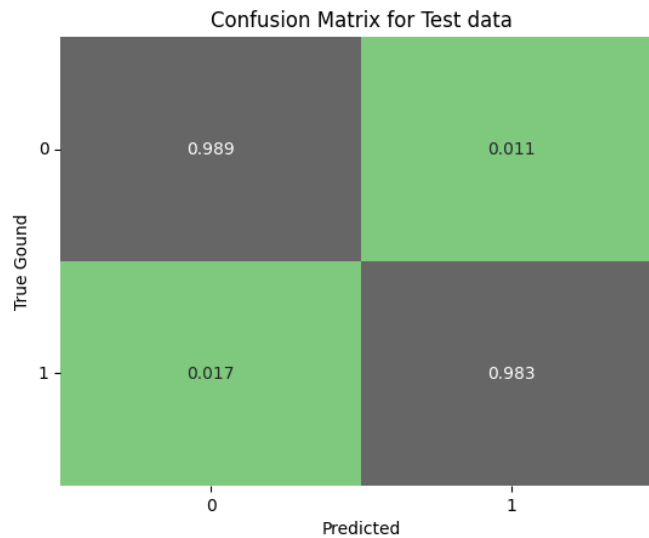


تصویر بالا، دقت و `recall` محاسبه شده روی داده اعتبارسنجی را نمایش می دهد.

عملکرد مدل

	precision	recall	f1-score	support
no Scam	0.98	0.99	0.99	56820
Scam	0.99	0.98	0.99	56906
accuracy			0.99	113726
macro avg	0.99	0.99	0.99	113726
weighted avg	0.99	0.99	0.99	113726

در تصویر بالا، تمامی معیار های `accuracy` و `recall` و `precision` و `f1-score` گزارش شده است.

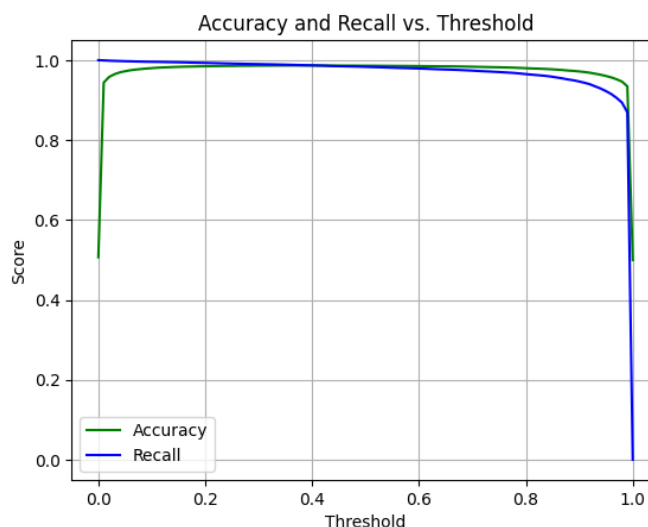


این تصویر نیز مربوط به ماتریس درهم ریختگی می باشد.

دقت به تنهایی نمی تواند معیار خوبی برای ارزیابی مدل ، به هنگام وجود عدم تعادل در داده ها باشد. به این دلیل که این معیار کلاس داده مورد بررسی را در نظر نمی گیرد و صرفا با درست یا اشتباه تخمین زدن لیبل کار می کند. مثلا از ۱۰۰ میان داده که ۹۸ تای آنها نرمال و ۲ تای آن ها عیب باشند، مدل همه را نرمال در نظر بگیرد، دقت ۹۸ درصد خواهیم داشت و این درحالی است که هیچ یک از داده های عیب تشخیص داده نشده است.

برای حل این مشکل می توان از معیار f1-score استفاده کرد. زیرا این معیار، با ترکیب دو معیار recall و precision، برای هر کلاس، هم داده های درست تخمین زده شده را در نظر می گیرد، هم داده های اشتباه لیبل زده شده. یعنی برای هر کلاس داده های True positive و False positive و False negative را در محاسبات خود دخیل می کند. مثلا برای مسئله ای که بیان شد، f1-score برای کلاس نرمال، برابر ۰.۹۹ و برای کلاس عیب برابر ۰ خواهد بود که به خوبی نشان می دهد که مدل در مقابل عمل طبقه بندی چگونه عمل کرده است.

عملکرد در مقابل threshold ها مختلف

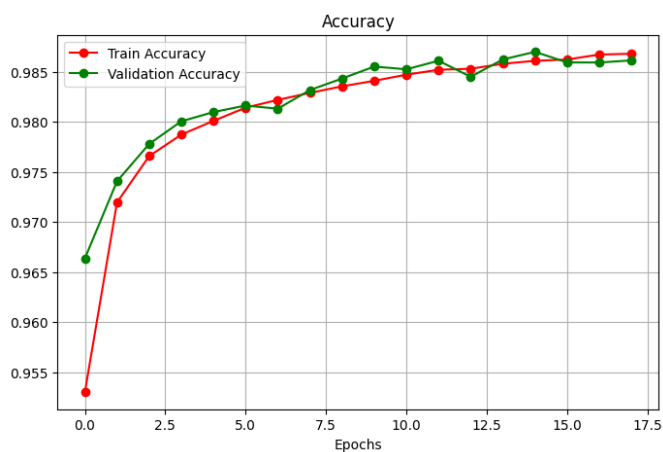
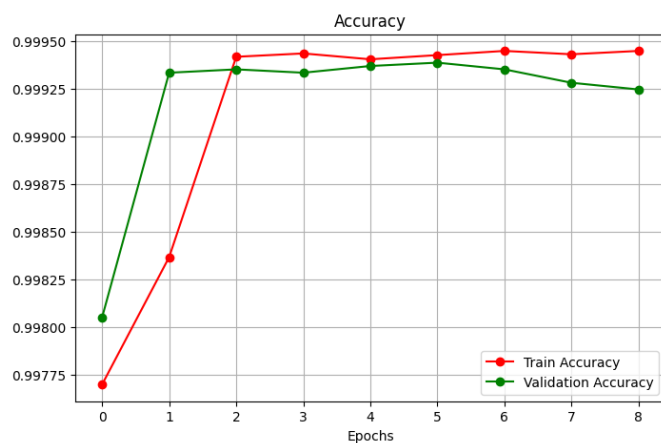


نمودار accuracy و recall براساس آستانه در نظر گرفته شده برای مدل آموزش داده شده روی داده آزمون.

عملکرد مدل بدون از بین بردن عدم توازن در داده و نویز گیری

نکته: در تمامی تصاویر این بخش، تصاویر سمت راست مربوط به مدل متعادل و denoise شده (مدل قبلی)

می باشد و تصویر سمت چپ مربوط به مدل متعادل و denoise نشده (مدل جدید) می باشند.



نمودار دقت، دقت بالاتری را نسب به مدل حالت قبل نشان می دهد، اما همان طور که بیان شد، دقت نمیتواند

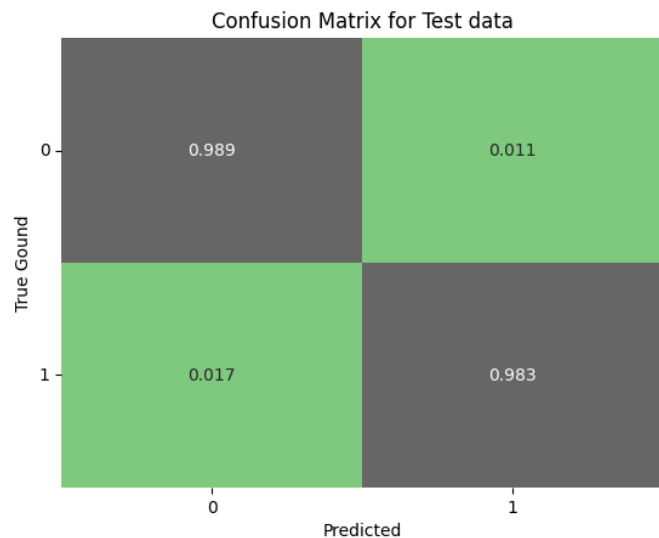
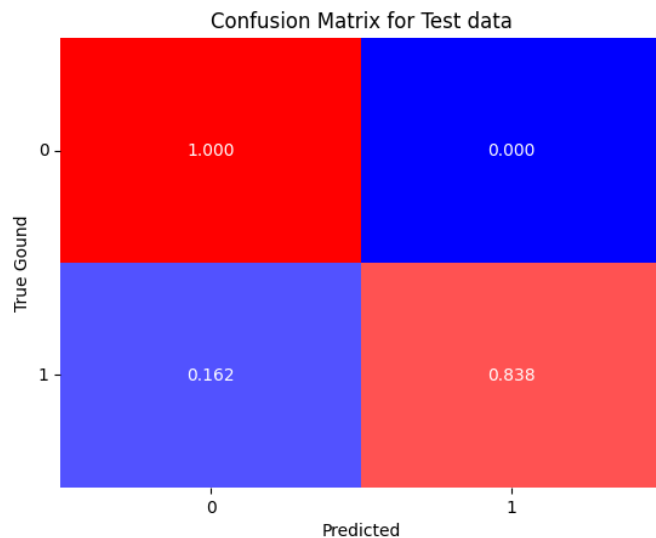
در این حالت معیار مناسبی برای سنجش عملکرد مدل باشد. برای بهتر درک کردن عملکرد مدل، به بیان موارد دیگری

می پردازیم.

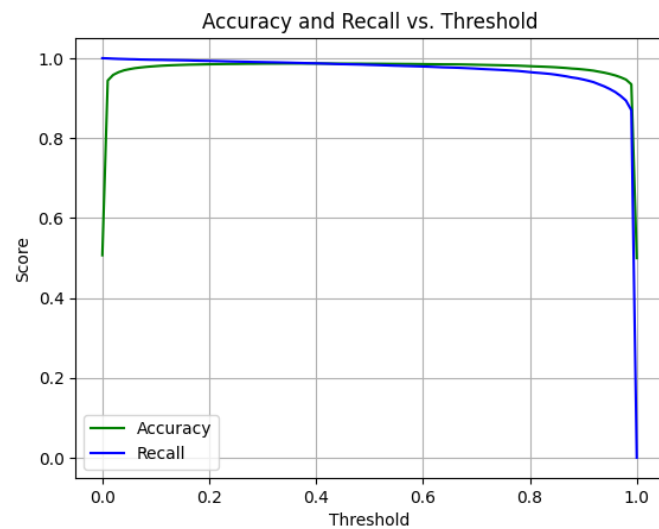
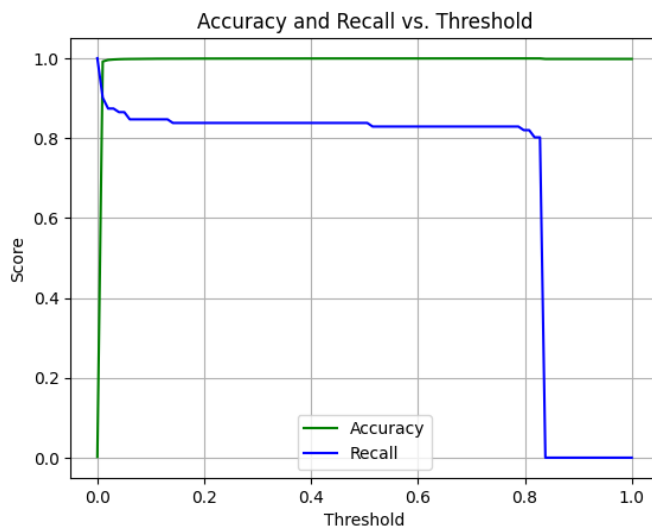
	precision	recall	f1-score	support		precision	recall	f1-score	support
no Scam	1.00	1.00	1.00	56851	no Scam	0.98	0.99	0.99	56820
Scam	0.79	0.84	0.81	111	Scam	0.99	0.98	0.99	56906
accuracy			1.00	56962	accuracy			0.99	113726
macro avg	0.89	0.92	0.91	56962	macro avg	0.99	0.99	0.99	113726
weighted avg	1.00	1.00	1.00	56962	weighted avg	0.99	0.99	0.99	113726

Classification report عملکرد ضعیفی رو کلاس scam نمایش می دهد و عملکرد خیلی خوبی را برای

تشخیص کلاس No scam گزارش می کند.



ماتریس درهم ریختگی هم برای کلاس scam (۱)، عملکرد خوبی را نشان نمی دهد.



تصویر بالا ضعف مدل متوازن و denoise نشده را به خوبی نمایش می دهد. مخصوصاً در recall