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

یادگیری ماشین
پاسخ مینی پروژه دوم

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

فهرست مطالب

فهرست تصاویر

فهرست جداول

فهرست برنامه‌ها

۱ سوال ۲

۱.۱ مشکل مورد بررسی

مقاله به چالش مدل‌های مبتنی بر SVM به منظور طبقه‌بندی چندکلاسه می‌پردازد. SVM های سنتی، عملکرد Binary Classification دارند این در حالی که برای مسائل چند کلاسه، نیاز به سازگار کردن مدل یا رویکرد تصمیم‌گیری، به منظور مدیریت تمام کلاس‌ها می‌باشد.

به صورت کلی سه نوع رویکرد برای هایSVM چندکلاسه وجود دارد: روش‌های heuristic که شامل one-vs-one و one-vs-all می‌باشد. روش‌های مذکور one-vs-one و one-vs-all، اگرچه محبوب و پر استفاده هستند، محدودیت‌هایی دارند. روش one-vs-one شامل حل $K(K-1)$ مسئله دودویی است که برای Kهای بزرگ باعث افزایش شدید بار محاسباتی می‌شود. از سوی دیگر، روش one-vs-all باید k مرتبه تمام دیتاست را استفاده کند تا یک مدل طبقه‌بندی ایجاد کند . این روش‌ها که مبتنی بر تبدیل مسئله به زیرمسئله‌های کوچکتر است، می‌تواند مناطق تصمیم‌گیری ایجاد کنند که متعلق به هیچ کلاسی نمی‌باشد.

روش دیگر مبتنی بر اصلاح خطا می‌باشد که حالت کلی‌تر روش قبلی می‌باشد. ایراد مدل‌های مبتنی بر این روش این است که ماتریس اصلاح باید از قبل مشخص باشد که در بسیاری از موارد انجام اینکار بسیار دشوار است. آخرین روش مبتنی بر طراحی و آموزش مدل‌هایی است که مستقیما مسئله چند کلاسه را با استفاده از یک ماشین SVM حل کند.

۲.۱ روش پیشنهادی

در این مقاله یک مدل به اسم Generalized Multiclass Support Vector Machine (GenSVM) پیشنهاد داده شده است که برای مدیریت طبقه‌بندی چندکلاسه بدون تجزیه مسئله به چندین طبقه‌بندی دودویی طراحی شده است. روش GenSVM از بهینه‌سازی iterative majorization، یک تکنیک بهینه‌سازی ریاضی، برای حل مسئله طبقه‌بندی استفاده می‌کند. این روش یک تابع بزرگی ایجاد می‌کند که به حداقل رساندن آن آسان‌تر از تابع هدف اصلی است و تضمین می‌کند که به سمت راه‌حل بهینه همگرا می‌شود. الگوریتم بهینه‌سازی تکراری شامل مراحل تعیین نقطه شروع، به حداقل رساندن تابع بزرگی و تکرار تا همگرایی است. دلیل استفاده از این الگوریتم این است که عموماً بهینه کردن یک تابع دوگان می‌تواند هزینه محاسباتی زیادی داشته باشد.

GenSVM یک طبقه‌بند تک‌ماشینه است که از فضای Simplex استفاده می‌کند. Simplex فضایی است که در آن بردارهای ویژگی‌ها به نقاطی در داخل یک مثلث نگاشت می‌شوند. یکی از مزایای استفاده از فضای Simplex این است که تصمیم‌گیری در این فضا ساده‌تر است و مرزهای تصمیم‌گیری به طور طبیعی شکل می‌گیرند.

این مدل از تابع هزینه Huber Hinge استفاده می‌کند که مشتق‌پذیر می‌باشد. نگاشت از فضای ورودی به فضای Simplex با کمینه‌سازی خطاهای طبقه‌بندی بهینه‌سازی می‌شود که با اندازه‌گیری فاصله یک نمونه از مرزهای تصمیم‌گیری در فضای Simplex محاسبه می‌شود. پیش‌بینی کلاس نیز در همین فضای Simplex انجام می‌شود.

در بهینه‌سازی تابع هزینه این مدل pـ نرم را بهینه می‌کند. مرزهای تصمیم‌گیری این مدل در فضای Simplex به مثلثی مربوط می‌شوند که نقاط درون آن به کلاس‌های مختلف تعلق دارند. تابع زیان GenSVM این امکان

را فراهم می‌کند که مدل‌های تک‌ماشینه قبلی در قالب طبقه‌بندهای نوع سوم پیاده‌سازی شوند.

۳.۱ ارزیابی عملکرد

عملکرد GenSVM با استفاده از آزمایشات گسترده‌ای محاسبه شده است تا با بقیه مدل‌های موجود در زمینه هایSVM چندکلاسه مقایسه شود. برای ارزیابی جامع عملکرد، از مجموعه داده‌های بزرگ و کوچک استفاده شد. استفاده از مجموعه داده‌های بزرگ به منظور بررسی مقیاس‌پذیری و کارایی محاسباتی در شرایط واقعی بود، در حالی که مجموعه داده‌های کوچک به منظور بررسی دقت طبقه‌بندی و قابلیت تعمیم در شرایط مختلف مورد استفاده قرار گرفتند.

برای مقایسه روش‌ها، از شاخص adjusted Rand index (ARI) برای اندازه‌گیری دقت طبقه‌بندی و همچنین از روش‌های رتبه‌بندی برای ارزیابی کارایی نسبی استفاده شد. روش رتبه‌بندی بر اساس زمان آموزش و دقت طبقه‌بندی هر روش انجام شد. در این مطالعه، GenSVM با چندین روش موجود در زمینه هایSVM چندکلاسه مقایسه شد که شامل روش‌های one-vs-all، one-vs-one و سایر روش‌های بهینه‌سازی چندکلاسه می‌شود.

نتایج نشان داد که GenSVM به طور قابل توجهی از نظر دقت طبقه‌بندی و کارایی محاسباتی برتری دارد. GenSVM بهترین نتایج را برای سریع‌ترین روش در جستجوی شبکه‌ای داشت و زمان آموزش متوسط آن به طور قابل توجهی کمتر از سایر روش‌ها بود. علاوه بر این، این روش نشان داد که در مقابل داده‌های مختلف مقاوم و قابل توسعه است و مقیاس‌پذیری و قابلیت اطمینان آن را نشان می‌دهد. این مدل بعد از روش one-vs-one بهترین دقت را کسب کرده است.

۴.۱ مزایا و معایب

مزایا:

- **سرعت**: GenSVM زمان محاسبات را به طور قابل توجهی نسبت به روش‌های سنتی کاهش می‌دهد.
- **دقت**: این روش از نظر دقت طبقه‌بندی از اکثر روش‌های موجود پیشی می‌گیرد.
- **مقیاس‌پذیری**: GenSVM مقاوم و مقیاس‌پذیر است و در مقابل مجموعه داده‌های مختلف عملکرد خوبی دارد.
- **حذف ناحیه ابهام**: خروجی این مدل، فضای ویژگی‌ها را به گونه‌ای افراز می‌کند که هیچ ناحیه‌ای بدون کلاس مشخص وجود نداشته باشد.
- **یک مدل به عنوان خروجی**: در این مدل به جای آموزش چندین SVM برای تشخیص هر کلاس، یک مدل آموزش داده می‌شود که تمامی کلاس‌ها را پیش‌بینی کند.

معایب:

- **پیچیدگی**: پیاده‌سازی بهینه‌سازی تکراری و مدیریت مجموعه‌ای بزرگتر از هاhyperparameter می‌تواند پیچیده باشد.

- **منابع مورد نیاز:** علیرغم بهبودهای کارایی، نیاز اولیه به منابع محاسباتی می‌تواند بالا باشد، به ویژه برای مجموعه داده‌های بسیار بزرگ درحالی که بخواهیم تمام هایhyperparameter موجود را بررسی کنیم.

در نتیجه، روش پیشنهادی GenSVM پیشرفت قابل توجهی در زمینه طبقه‌بندی چندکلاسه با هاSVM ارائه می‌دهد، دقت و کارایی بهتری را به ارمغان می‌آورد، اگرچه با افزایش پیچیدگی و نیازهای منابع همراه است.

۲ شبیه‌سازی

شبیه‌سازی عملکرد مدل پیشنهادی در مقاله مذکور توسط کتابخانه GenSVM انجام شده است. باید توجه داشت که این کتابخانه بر پایه ورژن‌های قدیمی NumPy و Sikit-Learn می‌باشد و به منظور استفاده از آن باید ورژن‌های مربوطه را تغییر دهیم. با استفاده از دستورات زیر، کتابخانه‌های مورد نیاز را نصب می‌کنیم.

```
!pip install scikit-learn==1.0.2
!pip install gensvm

```

Code : ۱ Intsall Libraries

در ادامه دیتاست iris از طریق کتابخانه sklearn دانلود می‌شود و با استفاده از MaxAbsScaler مقدار ویژگی‌ها را به یک عدد بین -۱ تا ۱ تصویر می‌کنیم تا مدل GenSVM بهتر همگرا شود.

```
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,stratify=y,test_size=0.2,random_state
    =53)
scaler = MaxAbsScaler().fit(X_train)
X_train, X_test = scaler.transform(X_train), scaler.transform(X_test)

```

Code : ۲ Intsall Libraries

در انتها با استفاده از کد زیر چند hyperparameter مربوط به مدل را تغییر می‌دهیم تا بهترین نتیجه حاصل شود.

```
for p in  [1.0, 2.0]:
    for kappa in [-0.9, 0.0]:
        for weight in ["group", "unit"]:
            for kernel in ['linear']:
                for epsilon in [1e-3 ,1e-5]:
                    clf = GenSVM(epsilon=epsilon,
                            kappa=kappa,
                            kernel=kernel,
                            p=p,
                            random_state=53,
                            verbose=True,
                            weights=weight)
                    clf.fit(X_train, y_train)
```

```
                y_pred = clf.predict(X_test)
                report = classification_report(y_test, y_pred)
                print(clf)
                print(report+"\n")
```

Code :۳ Intsall Libraries

در کد فوق p مشخص می‌کند که نرم چندم از تابع لاس باید بهینه شود، مقدار kappa محدوده درجه۲ تابع هزینه را مشخص می‌کند، weight نشان می‌دهد که به منظور رفع imbalance در دیتاست، وزن نمونه‌ها باید چگونه تنظیم شود و epsilon مشخص می‌کند که تا چه میزان باید بهینه‌سازی انجام شود. نتایج بدست آمده از مجموعه آموزش‌های فوق به شرح زیر است.

```
GenSVM(epsilon=0.001, kappa=9.0-, random_state=53, verbose=True,
       weights='group')
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30


GenSVM(epsilon=1e05-, kappa=9.0-, random_state=53, verbose=True,
       weights='group')
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30
```

```
GenSVM(epsilon=0.001, kappa=9.0-, random_state=53, verbose=True)
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30


GenSVM(epsilon=1e05-, kappa=9.0-, random_state=53, verbose=True)
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30


GenSVM(epsilon=0.001, random_state=53, verbose=True, weights='group')
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30
```

```
GenSVM(epsilon=1e05-, random_state=53, verbose=True, weights='group')
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30


GenSVM(epsilon=0.001, random_state=53, verbose=True)
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30


GenSVM(epsilon=1e05-, random_state=53, verbose=True)
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30
```

```
GenSVM(epsilon=0.001, kappa=9.0-, p=2.0, random_state=53, verbose=True,
       weights='group')
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30


GenSVM(epsilon=1e05-, kappa=9.0-, p=2.0, random_state=53, verbose=True,
       weights='group')
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30


GenSVM(epsilon=0.001, kappa=9.0-, p=2.0, random_state=53, verbose=True)
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
```

```
weighted avg       97.0      97.0      97.0        30


GenSVM(epsilon=1e05-, kappa=9.0-, p=2.0, random_state=53, verbose=True)
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30


GenSVM(epsilon=0.001, p=2.0, random_state=53, verbose=True, weights='group')
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       88.0      70.0      78.0        10
           2       75.0      90.0      82.0        10

    accuracy                           87.0        30
   macro avg       88.0      87.0      87.0        30
weighted avg       88.0      87.0      87.0        30


GenSVM(epsilon=1e05-, p=2.0, random_state=53, verbose=True, weights='group')
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
```

```
weighted avg       97.0      97.0      97.0        30


GenSVM(epsilon=0.001, p=2.0, random_state=53, verbose=True)
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       88.0      70.0      78.0        10
           2       75.0      90.0      82.0        10

    accuracy                           87.0        30
   macro avg       88.0      87.0      87.0        30
weighted avg       88.0      87.0      87.0        30


GenSVM(epsilon=1e05-, p=2.0, random_state=53, verbose=True)
              precision    recall  f1-score   support

           0       00.1      00.1      00.1        10
           1       00.1      90.0      95.0        10
           2       91.0      00.1      95.0        10

    accuracy                           97.0        30
   macro avg       97.0      97.0      97.0        30
weighted avg       97.0      97.0      97.0        30
```

همانطور که مشخص است با تغییر hyperparameter ها تغییر محسوسی در نتایج ایجاد نمی‌شود و دلیل آن این است که دیتاست مذکور از پیچیدگی خاصی برخوردار نیست و نتایج به راحتی به درصد مناسبی می‌رسند.

۳ سوال ۳

۱.۳ بخش اول

۱.۱.۳ چالش‌ها

تشخیص تقلب برای تراکنش‌های کارت اعتباری، چالش‌های قابل توجهی را ایجاد می‌کند که موارد اشاره شده در مقاله مورد نظر به شرح زیر می‌باش:

- **داده‌های نامتوازن:** چالش اصلی در تشخیص تقلب، عدم توازن شدید بین تعداد تراکنش‌های تقلبی و غیرتقلبی است. در دیتاست موجود، تراکنش‌های تقلبی تنها ۰.۱۷۲% از کل تراکنش‌ها را تشکیل می‌دهند. این عدم توازن، یادگیری کلاس اقلیت را برای مدل‌های یادگیری ماشین دشوار می‌سازد زیرا مدل‌ها روی تمام داده‌های موجود فرایند آموزش را اعمال می‌کنند که این مسئله موجد تمایل مدل به جانب‌داری از کلاس اکثریت خواهد شد و در نتیجه آن عملکرد طبقه‌بندی تضعیف خواهد شد.
- **Over fitting**: با توجه به تعداد کم تراکنش‌های تقلبی، خطر Over fitting مدل به داده‌های آموزشی بسیار زیاد است. Over fitting زمانی رخ می‌دهد که مدل، نویز و جزئیات داده‌های آموزشی را به حدی یاد می‌گیرد که در داده‌های جدید عملکرد ضعیفی دارد و نمی‌تواند ویژگی‌های مناسب را استخراج کند. این مسئله به ویژه در تشخیص تقلب مشکل‌ساز است زیرا تعمیم‌پذیری به داده‌های جدید حیاتی است.
- **معیارهای ارزیابی**: معیارهای ارزیابی سنتی مانند Accuracy برای دیتاست‌های نامتوازن مناسب نیستند. دقت بالا می‌تواند گمراه‌کننده باشد زیرا در این مسئله که فقط ۰.۱۷۲% داده‌ها تقلبی هستند، مدلی که همه تراکنش‌ها را غیرتقلبی پیش‌بینی کند هنوز دقت بالای ۹۹% خواهد داشت به دلیل تعداد زیاد تراکنش‌های غیرتقلبی. فرمول محاسبه Accuracy به شرح زیر می‌باشد که با تشخیص تمام نمونه‌ها به عنوان سالم، مقدار TN به قدری زیاد می‌شود TP قابل صرف‌نظر است.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (۱)$$

بنابراین، انتخاب معیارهای ارزیابی مناسب برای ارزیابی عملکرد مدل ضروری است.

۲.۱.۳ روش‌های مقابله با این چالش‌ها

برای مقابله با این چالش‌ها، مقاله مذکور از چندین روش استفاده می‌کند که به شرح زیر است.

- **شبکه‌های عصبی Auto-Encoder**: مقاله از شبکه‌های عصبی Auto-Encoder برای تشخیص ناهنجاری استفاده می‌کند. Auto-Encoder ها نوعی از مدل‌های یادگیری بدون نظارت هستند که یاد می‌گیرند داده‌ها را فشرده کنند و درعین‌حال ویژگی‌هایی را از آنها استخراج کنند، سپس داده‌های ورودی را بازسازی کنند. با آموزش بر روی کلاس اکثریت (تراکنش‌های غیرتقلبی)، Auto-Encoder الگوهای نرمال در داده‌ها را یاد می‌گیرد. در طی تست، تراکنش‌هایی که به طور قابل توجهی از این الگوها انحراف دارند به عنوان تقلب احتمالی علامت‌گذاری می‌شوند. این روش کمک می‌کند تا بدون نیاز به دیتاست متوازن، ناهنجاری‌ها شناسایی شوند.
- **معیارهای ارزیابی**: به جای استفاده از Accuracy ، مقاله پیشنهاد می‌دهد از Recall به عنوان معیار اصلی ارزیابی استفاده شود. Recall درصد موارد تقلبی درست شناسایی‌شده از کل موارد تقلبی واقعی را اندازه‌گیری می‌کند و برای دیتاست‌های نامتوازن مناسب‌تر است. فرمول Recall به شرح زیر است.

$$\text{Recall} = \frac{TP}{TP + FN} \quad (۲)$$

- **تنظیم آستانه**: مقاله پیشنهاد می‌دهد از یک آستانه بر روی خطای بازسازی برای طبقه‌بندی تراکنش‌ها به عنوان تقلبی یا غیرتقلبی استفاده شود. با تنظیم یک آستانه مناسب، مدل می‌تواند بهتر بین تراکنش‌های نرمال و غیرنرمال تمایز قائل شود و بدین ترتیب تشخیص خصوصا در معیار Recall را بهبود بخشد.

- **افزایش داده‌ها:** مقاله از تکنیک‌های افزایش داده برای بهبود عملکرد مدل استفاده می‌کند. یکی از روش‌های مورد استفاده، ایجاد نمونه‌های مصنوعی از تراکنش‌های تقلبی با استفاده از تکنیک‌هایی مانند *Synthetic Minority Over-sampling Technique* (SMOTE) است. این روش با تولید نمونه‌های جدید از ترکیب خطی نمونه‌های موجود کلاس اقلیت، به تعادل کلاس‌ها کمک کرده و مدل را قادر می‌سازد تا الگوهای مربوط به تقلب را بهتر یاد بگیرد.

با پرداختن به چالش‌ها از طریق این روش‌ها، مقاله یک رویکرد جامع برای توسعه یک مدل مؤثر تشخیص تقلب با استفاده از شبکه‌های عصبی Auto-Encoder ارائه می‌دهد. این رویکرد نه تنها تشخیص تراکنش‌های تقلبی را بهبود می‌بخشد بلکه اطمینان می‌دهد که مدل قابل اعتماد بوده و به خوبی به داده‌های جدید تعمیم می‌یابد.

۲.۳ بخش دوم

۱.۲.۳ معماری مدل

معماری مدل در این مطالعه از دو جزء اصلی تشکیل شده است، Denoising Auto-Encoder و classifier نهایی.

- **Denoising Auto-Encoder**

این Auto-Encoder برای حذف نویز از دیتاست طراحی شده است که برای آموزش آن نویزهای مختلفی مثل گوسین یا Salt و pepper نویز به داده‌اضافه می‌شود و در خروجی انتظار داریم تا دیتای بدون نویز بازسازی شود. معماری این مدل شامل هفت لایه است:

- لایه ورودی با ۲۹ نورون.
- سه لایه پنهان در بخش Encoder که به ترتیب ۲۲، ۱۵ و ۱۰ نورون دارند که دیتای ورودی را فشرده می‌کنند.
- سه لایه پنهان در بخش Decoder که به ترتیب ۱۰، ۱۵ و ۲۲ نورون دارند که دیتای فشرده را رمزگشایی می‌کنند و به ۲۹ نورون در لایه خروجی می‌رساند.
- از تابع هزینه MSE برای بهینه‌سازی بازسازی ویژگی‌های ورودی استفاده می‌شود.

ساختار به شرح زیر است:

لایه	تعداد نورون‌ها
ورودی	۲۹
۱ FCN	۲۲
۲ FCN	۱۵
۳ FCN	۱۰
۴ FCN	۱۵
۵ FCN	۲۲
خروجی	۲۹

- **Classifier** پس از فرآیند حذف نویز، یک شبکه عصبی عمیق FCN برای طبقه‌بندی تراکنش‌ها استفاده می‌شود. این طبقه‌بند شامل شش لایه به شرح زیر است.
  - لایه ورودی با ۲۹ نورون.
  - چهار لایه پنهان با ۲۲، ۱۵، ۱۰ و ۵ نورون به ترتیب.
  - لایه خروجی با ۲ نورون، که به دو کلاس تقلب یا عدم تقلب مربوط می‌شود.
  - مدل از تابع SoftMax همراه با تابع هزینه Cross-Entropy برای طبقه‌بندی نهایی استفاده می‌کند.

ساختار به شرح زیر است:

لایه	تعداد نورون‌ها
ورودی	۲۹
۱ FCN	۲۲
۲ FCN	۱۵
۳ FCN	۱۰
۴ FCN	۵
خروجی	۲

۲.۲.۳ پارامترهای آموزش

پیش‌پردازش داده‌ها

- ویژگی 'زمان' حذف می‌شود.
- ویژگی 'مقدار' نرمال‌سازی می‌شود.
- دیتاست که قبلاً با PCA تبدیل شده است، نیاز به نرمال‌سازی بیشتری ندارد و ۲۸ ویژگی اول دیتاست به عنوان مولفه‌های اصلی درنظر گرفته می‌شوند.

Augmentation

برای مقابله با عدم توازن در دیتاست، داده‌های آموزشی میزان داده‌های تقلبی را تنها در مجموعه‌داده آموزش به صورت مصنوعی زیاد می‌کنیم.

۳.۳ بخش سوم

به منظور شبیه‌سازی مدل‌های معرفی شده در مقاله، ابتدا دیتا را دریافت می‌شود و ستون زمان از آن حذف می‌شود سپس ستون "مقدار" را نرمال شده است. در گام بعد مجموعه داده آموزش از ارزیابی با نسبت ۷.۰ به ۳.۰ جدا شده است و فرایند Over-Sampling روی دیتاست آموزش اجرا می‌شود که کد آن به شرح زیر می‌باشد.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,stratify=y,
  random_state=53)
smote = SMOTE(random_state=53,sampling_strategy=1.0)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

```

Code :۴ Split data and over-sampling

نتیجه اعمال کد فوق به روی مجموعه داده به شرح زیر است.

```
Original dataset shape: [199020    344]
Resampled dataset shape: [199020 199020]
```

۱.۳.۳ Auto-Encoder Denoising

در ادامه مدل Denoising Auto-Encoder پیاده‌سازی شده است که اطلاعات لایه‌های آن متناسب با جزییات اشاره شده در مقاله یکسان می‌باشد. به منظور آموزش این مدل از بهینه‌ساز Adam و تابع هزینه MSE استفاده شده است. به منظور ایجاد داده آموزش در این مدل باید مقداری نویز به داده‌های اصلی اضافه شود که بدین منظور با توجه به جزییات مقاله از نویز گوسین و نویز salt - pepper استفاده شده است که در نتیجه آن تمام داده‌ها شامل نویز گوسین می‌باشند اما تنها ۳۰ درصد از داده‌ها نویز salt - pepper را دارند. به منظرو پیاده‌سازی این دو نوع نویز از توابع زیر استفاده شده است.

```
def add_gaussian_noise(X, mean=0.0, std=1):
    noise = np.random.normal(mean, std, X.shape)
    X_noisy = X + noise
    return X_noisy

def add_salt_and_pepper_noise(X, salt_prob, pepper_prob):
    X_noisy = X.copy()
    # Adding salt noise
    num_salt = int(np.ceil(salt_prob * X.size))
    salt_coords = [np.random.randint(0, i - 1, num_salt) for i in X.shape]
    X_noisy[tuple(salt_coords)] = 1

```

```
    # Adding pepper noise
    num_pepper = int(np.ceil(pepper_prob * X.size))
    pepper_coords = [np.random.randint(0, i - 1, num_pepper) for i in X.shape]
    X_noisy[tuple(pepper_coords)] = 0

    return X_noisy

```

Code :۵ Noise generator

در ادامه دو تابع فوق با استفاده از مجموعه کد زیر به دیتاست اعمال شده است. باید توجه داشت که دیتای تست نیز باید شامل نویز باشد و به عنوان Ground Truth باید خود دیتای اصلی را بدهیم.

```
# Convert training and test data to numpy arrays
X_train_res_np = X_train_res.values
X_test_np = X_test.values

# Add Gaussian noise to the entire training and test sets
X_train_gaussian_noisy = add_gaussian_noise(X_train_res_np)
X_test_gaussian_noisy = add_gaussian_noise(X_test_np)

# Select 30% of the training and test sets for additional salt-and-pepper noise
num_train_samples = X_train_res_np.shape[0]
num_test_samples = X_test_np.shape[0]
train_indices = np.random.choice(num_train_samples, int(0.3 * num_train_samples), replace=False)
test_indices = np.random.choice(num_test_samples, int(0.3 * num_test_samples), replace=False)
X_train_gaussian_snp_noisy = X_train_gaussian_noisy.copy()
X_test_gaussian_snp_noisy = X_test_gaussian_noisy.copy()
X_train_gaussian_snp_noisy[train_indices] = add_salt_and_pepper_noise(X_train_gaussian_snp_noisy[
    train_indices], salt_prob=0.02, pepper_prob=0.02)
X_test_gaussian_snp_noisy[test_indices] = add_salt_and_pepper_noise(X_test_gaussian_snp_noisy[
    test_indices], salt_prob=0.02, pepper_prob=0.02)

# Convert noisy training and test data to PyTorch tensors and move to device
X_train_res_tensor = torch.tensor(X_train_res_np, dtype=torch.float32).to(device)
X_train_noisy_tensor = torch.tensor(X_train_gaussian_snp_noisy, dtype=torch.float32).to(device)
X_test_tensor = torch.tensor(X_test_np, dtype=torch.float32).to(device)
X_test_noisy_tensor = torch.tensor(X_test_gaussian_snp_noisy, dtype=torch.float32).to(device)

```

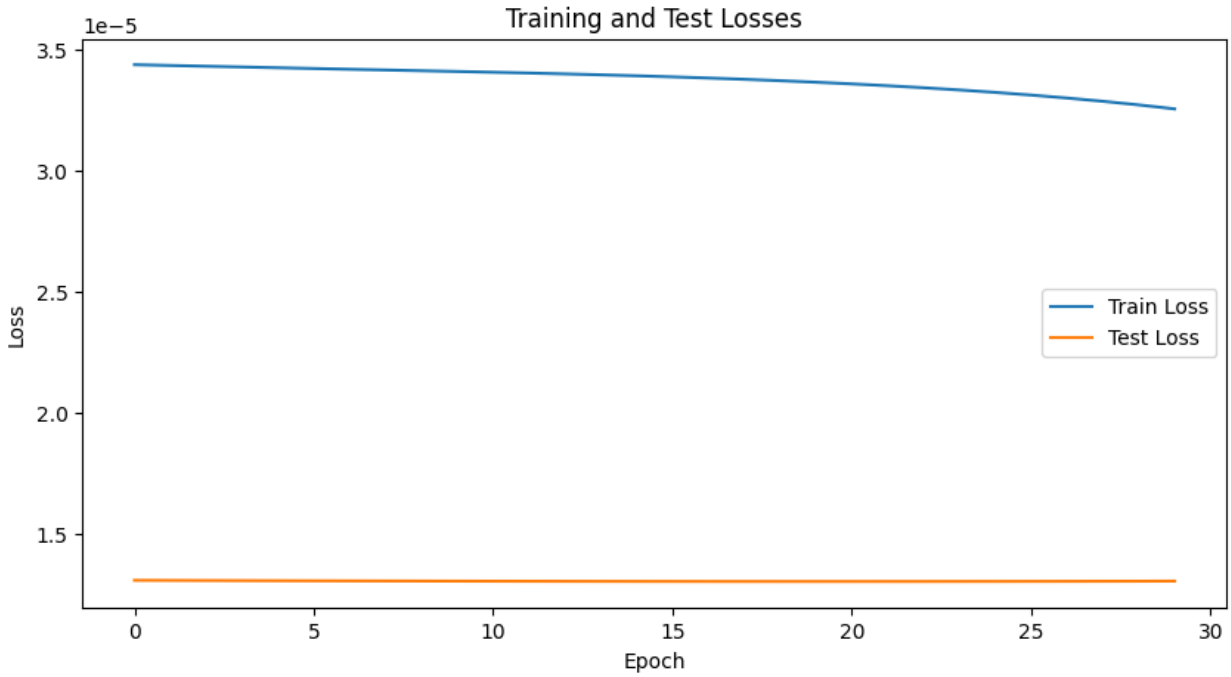
Code :۶ Add noise

در انتها مدل مورد نظر با استفاده از کد زیر آموزش داده شده است که نتیجه تابع هزینه آن در شکل ۱ مشخص است که دامنه این تابع هزینه بیانگر عملکرد خوب مدل در بازسازی سیگنال است.

```
# Training the autoencoder and plotting losses
num_epochs = 30
```

```
batch_size = 1024**2
train_losses = []
test_losses = []

for epoch in range(num_epochs):
    train_loss = 0
    test_loss = 0
    # Training
    autoencoder.train()
    for i in range(0, len(X_train_res_tensor), batch_size):
        batch = X_train_noisy_tensor[i:i+batch_size]
        target = X_train_res_tensor[i:i+batch_size]

        # Forward pass
        outputs = autoencoder(batch)
        loss = criterion(outputs, target)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    train_losses.append(train_loss / len(X_train_res_tensor))

    # Evaluation on test set
    autoencoder.eval()
    with torch.no_grad():
        outputs = autoencoder(X_test_noisy_tensor)
        test_loss = criterion(outputs, X_test_tensor).item()
        test_losses.append(test_loss / len(X_test_tensor))

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_losses[-1]:.4f}, Test Loss: {
    test_losses[-1]:.4f}')

```

Code :۷ Train model



شکل ۱: auto-encoder denoising on set test and train of value Loss

۲.۳.۳ Classifier

مدل طبقه‌بندی با توجه به جزییات مشروح در مقاله طراحی شده است و ساختار مدل متناسب با جدول موجود در مقاله می‌باشد همچنین برای آموزش این مدل از تابع هزینه Cross-Entropy استفاده شده است. گام نخست در آموزش این مدل این است که داده‌های آموزش و ارزیابی را از مدل Denoising عبور دهیم که این مسئله توسط کد زیر انجام شده است.

```
# Denoise the training and test sets using the trained autoencoder
autoencoder.eval()
with torch.no_grad():
    X_train_denoised_tensor = autoencoder(X_train_noisy_tensor)
    X_test_denoised_tensor = autoencoder(X_test_noisy_tensor)

```

Code :۸ Remove noise

حال مدل طبقه‌بندی با استفاده از کد زیر آموزش داده می‌شود.

```
num_epochs_classifier = 2000
batch_size_classifier = 1024**2
train_losses_classifier = []
test_losses_classifier = []
train_recall = []
test_recall = []
train_accuracy = []
```

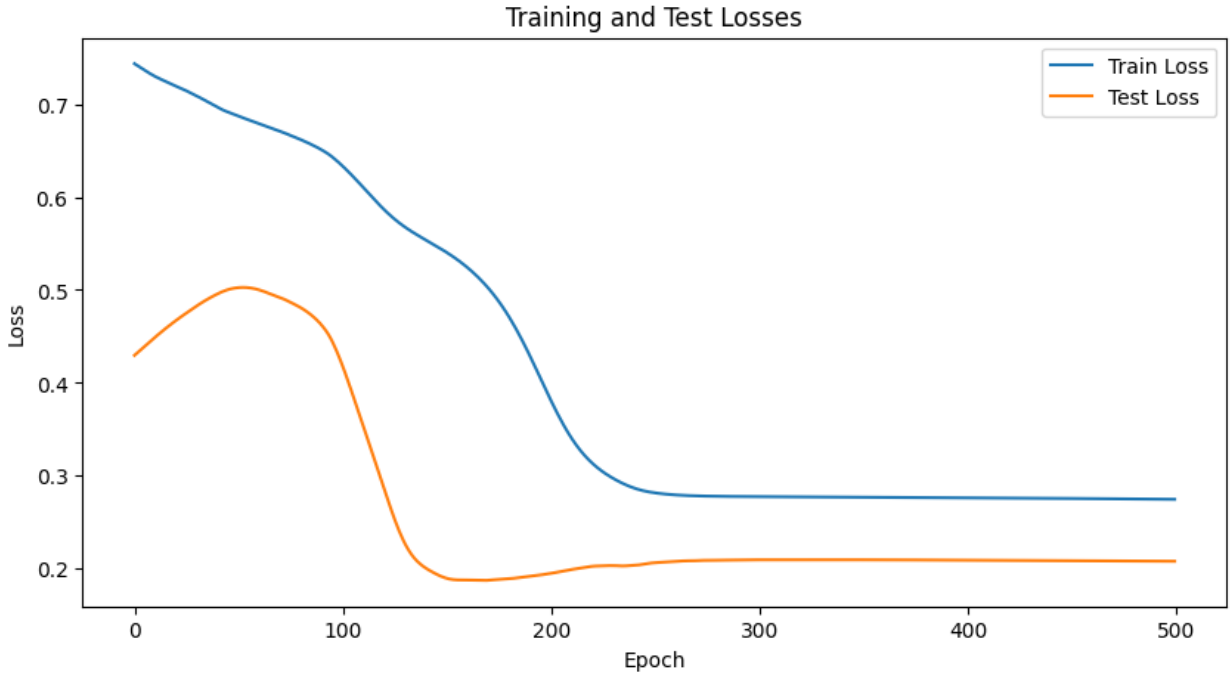
```
test_accuracy = []
c = 0
t = 200
# Convert targets to PyTorch tensors and move to device
y_train_res_tensor = torch.tensor(y_train_res.values, dtype=torch.float32).to(device).unsqueeze
    (1)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32).to(device).unsqueeze(1)

best_test_loss = float('inf')
best_model_state = None

for epoch in range(num_epochs_classifier):
    train_loss = 0
    test_loss = 0
    classifier.train()
    # Training
    for i in range(0, len(X_train_denoised_tensor), batch_size_classifier):
        batch_X = X_train_denoised_tensor[i:i+batch_size_classifier]
        batch_y = y_train_res_tensor[i:i+batch_size_classifier]

        # Forward pass
        outputs = classifier(batch_X)[:, 1]
        loss = criterion(outputs, batch_y.squeeze())

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    # Evaluation on test set
    classifier.eval()
    with torch.no_grad():
        outputs_train = classifier(X_train_denoised_tensor)[:, 1]
        outputs_test = classifier(X_test_denoised_tensor)[:, 1]

        train_loss = criterion(outputs_train, y_train_res_tensor.squeeze()).item()
        test_loss = criterion(outputs_test, y_test_tensor.squeeze()).item()

        test_losses_classifier.append(test_loss)
        train_losses_classifier.append(train_loss)

        # Calculate recall and accuracy
        outputs_train = outputs_train.cpu().numpy()
```

```
        outputs_test = outputs_test.cpu().numpy()
        y_train_res_np = y_train_res_tensor.cpu().numpy()
        y_test_np = y_test_tensor.cpu().numpy()

        train_recall.append(recall_score(y_train_res_np, outputs_train.round()))
        test_recall.append(recall_score(y_test_np, outputs_test.round()))
        train_accuracy.append(accuracy_score(y_train_res_np, outputs_train.round()))
        test_accuracy.append(accuracy_score(y_test_np, outputs_test.round()))

    # Check if current test loss is the best we've seen so far
    if test_loss < best_test_loss:
        best_test_loss = test_loss
        best_model_state = classifier.state_dict()
        c = 0
    else:
        c = c+1

    if c > t:
        break


    print(f'Epoch [{epoch+1}/{num_epochs_classifier}], Train Loss: {train_losses_classifier
    [-1]:.4f}, Test Loss: {test_loss:.4f}, Train Recall: {train_recall[-1]:.4f}, Test Recall: {
    test_recall[-1]:.4f}, Train Accuracy: {train_accuracy[-1]:.4f}, Test Accuracy: {test_accuracy
    [-1]:.4f}')

# Load the best model state
classifier.load_state_dict(best_model_state)
print(f'Best Test Loss: {best_test_loss:.4f}')

# Return the classifier with the best test loss
classifier

```

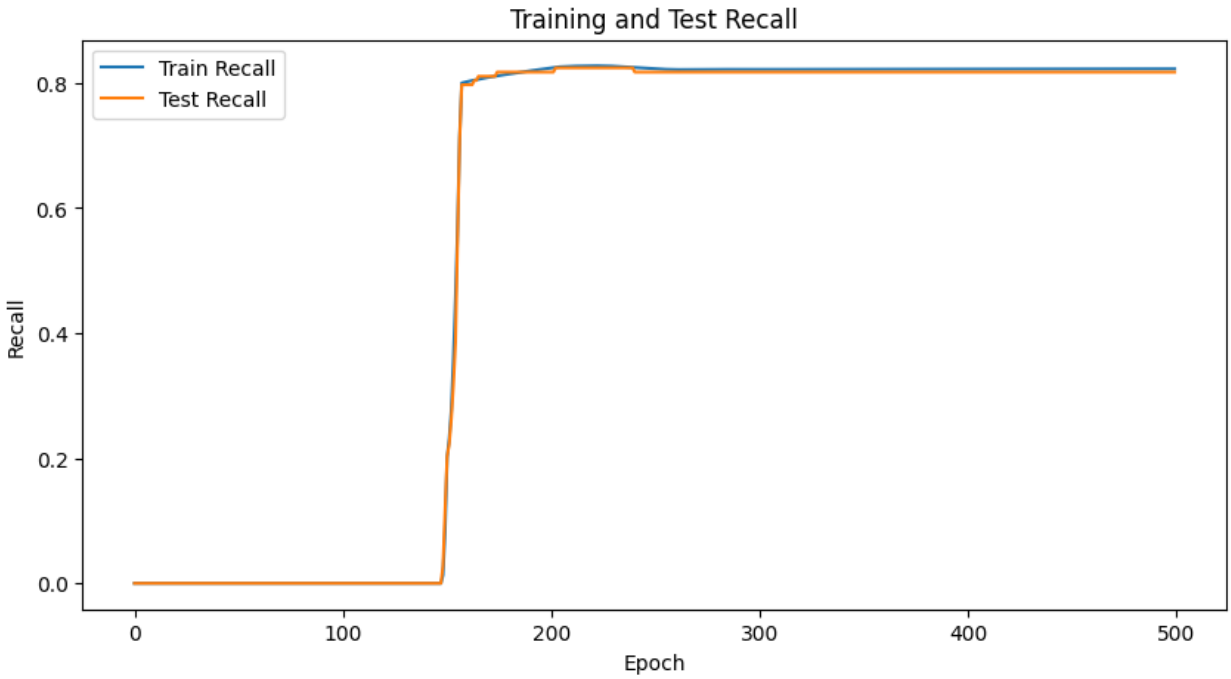
Code :۹ train model

نتایج حاصل شده از آموزش مدل در شکل ۲ ، شکل ۳ و شکل ۴ که به ترتیب نمودار تابه هزینه، Recall و Accuracy را نمایش می‌دهند مشخص است.
همانطور که از نمودار تابع هزینه مشخص است، این معیار تا Epoch ۱۵۰ کاهشی بوده و پس از آن مقدار تابع هزینه مجموعه ارزیابی افزایش پیدا کرده که این نشان‌دهنده overfit شدن مدل است. بهترین مدل با توجه به همین نمودار استخراج شده است. نمودار بعدی که مختص Recall است که در حوالی Epoch ۱۵۰ شروع به یادگرفتن نمونه‌های تقلب کرده است و سریعا مقدار آن برابر ۸۰ درصد شده است. نمودار بعدی مختص به Accuracy است که در فرایند آموزش به عدت Over-sampling در روند مشابه نمودار Recall از خود نشان داده است اما در مجموعه ارزیابی به علت ناترازی شدید در حوالی Epoch ۱۵۰ کمی از دقت آن کاسته

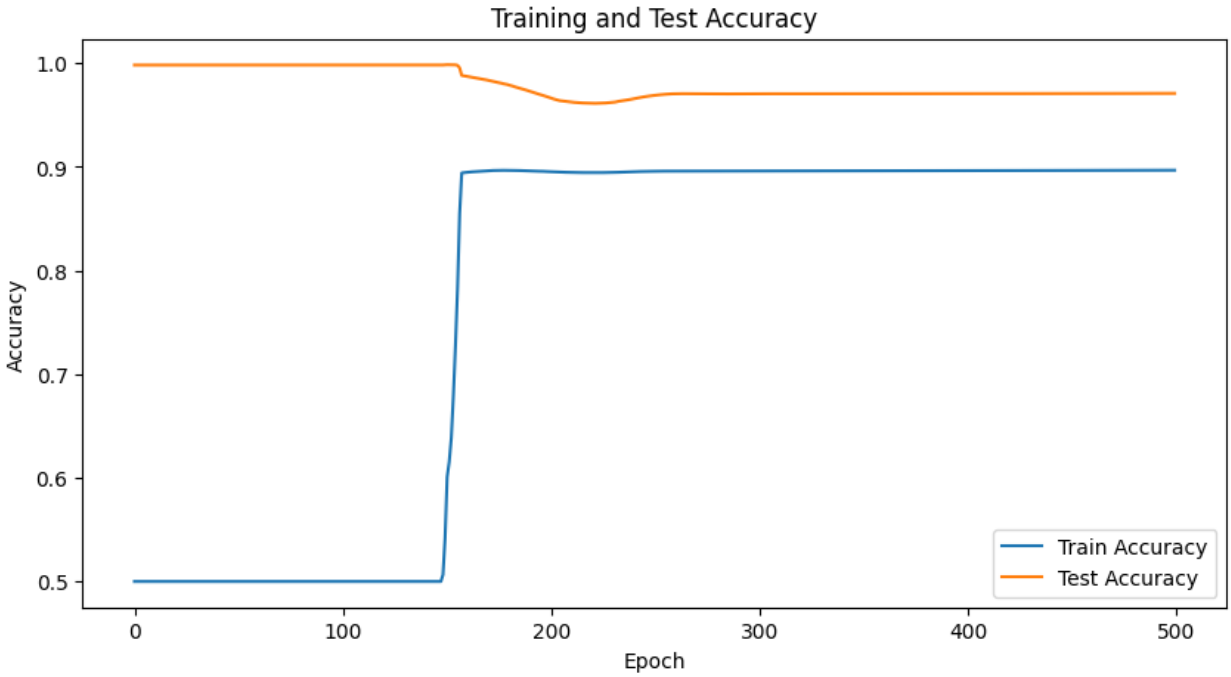
شده است.



شکل ۲: loss Classification



شکل ۳: Recall Classification



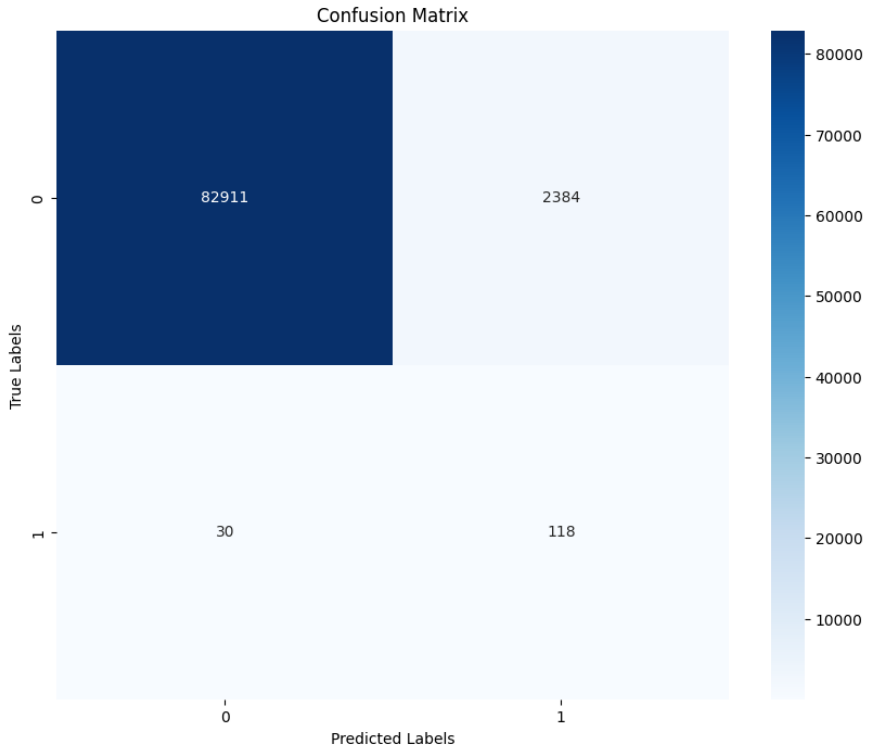
شکل ۴: Accuracy Classification

۴.۳ بخش چهارم

متریک‌های مورد نظر به شرح زیر است.

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

           0       00.1      97.0      99.0     85295
           1       05.0      80.0      09.0       148

    accuracy                           97.0     85443
   macro avg       52.0      88.0      54.0     85443
weighted avg       00.1      97.0      98.0     85443
```

و ماتریس درهم‌ریختگی در شکل ۵ نشان داده شده است. با توجه به گزارشات و ماتریس درهم‌ریختگی عملکرد مدل در پیدا کردن نمونه‌های تقلبی با توجه به متریک Recall قبول است اما اکر مترین Precision و F۱-Score را بررسی کنیم در مورد کلاس ۱ نتایج خیلی بد است که این مسئله نشان می‌دهد مدل تعداد زیادی از نمونه‌های نرمال را به عنوان تقلب تشخیص داده است اما به لحاظ عملی شاید این مسئله مشکلی اساسی نباشید زیرا عدم تشخیص یک مورد تقلب می‌تواند تبعات سنگینی داشته باشد اما تشخیص یک مورد نرمال به عنوان تقلب نهایتا باعث به تاخیر افتادن گردش مالی خواهد شد که این مسئله خطر کمتری دارد.



شکل ۵: matrix Confusion

به منظور بررسی عملکرد متریک Accuracy در مسائلی که imbalance هستند باید به شکل ۵رجوع کرد که درایه‌های آن در فرمول معادله ۱ قرار می‌گیرد. با توجه به فرمول ،Accuracy مقادیر روی قطر اصلی ماتریس درهم‌ریختگی در صورت و مخرج قرار دارند و باقی مقادیر تنها در مخرج قرار دارند اما با توجه با این فرمول و اعداد موجوذ ذر ماتریس، مشخصا مقدار TN به قدری زیاد است که تمام خانه‌های دیگر ماتریس در مقابل آن قابل صرف‌نظر می‌باشد که در نتیجه آن اگر تمام نمونه‌های تقلب هم به عنوان نرمال شناسایی شود، نتیجه محسوسی در این متریک مشاهده نخواهد شد، بنابراین در این مدل مسائل Accuracy اصلا متریک خوبی نیست و اگر مدل تمام نمونه‌ها را نرمال تشخیص دهد باز عدد خیلی بالایی را نتیجه می‌دهد.

باتوجه به تحلیل مقاله، متریک Recall در کنار Accuracy می‌تواند نشان‌دهنده عملکرد مدل شود زیرا Recall تنها روی سطر دوم شکل ۵ اجرا می‌شود و به خانه اول این ماتریس کاری ندارد و در نتیجه آن اگر مدل تمام نمونه‌ها را نرمال تشخیص دهد میزان این متریک ۰ می‌شود. در کنار این متریک Precision که روی ستون دوم ماتریس عمل می‌کند و f۱-score که توازنی بین Recall و Precision برقرار می‌کند نیز کمک کننده می‌باشد.

۵.۳ بخش پنجم

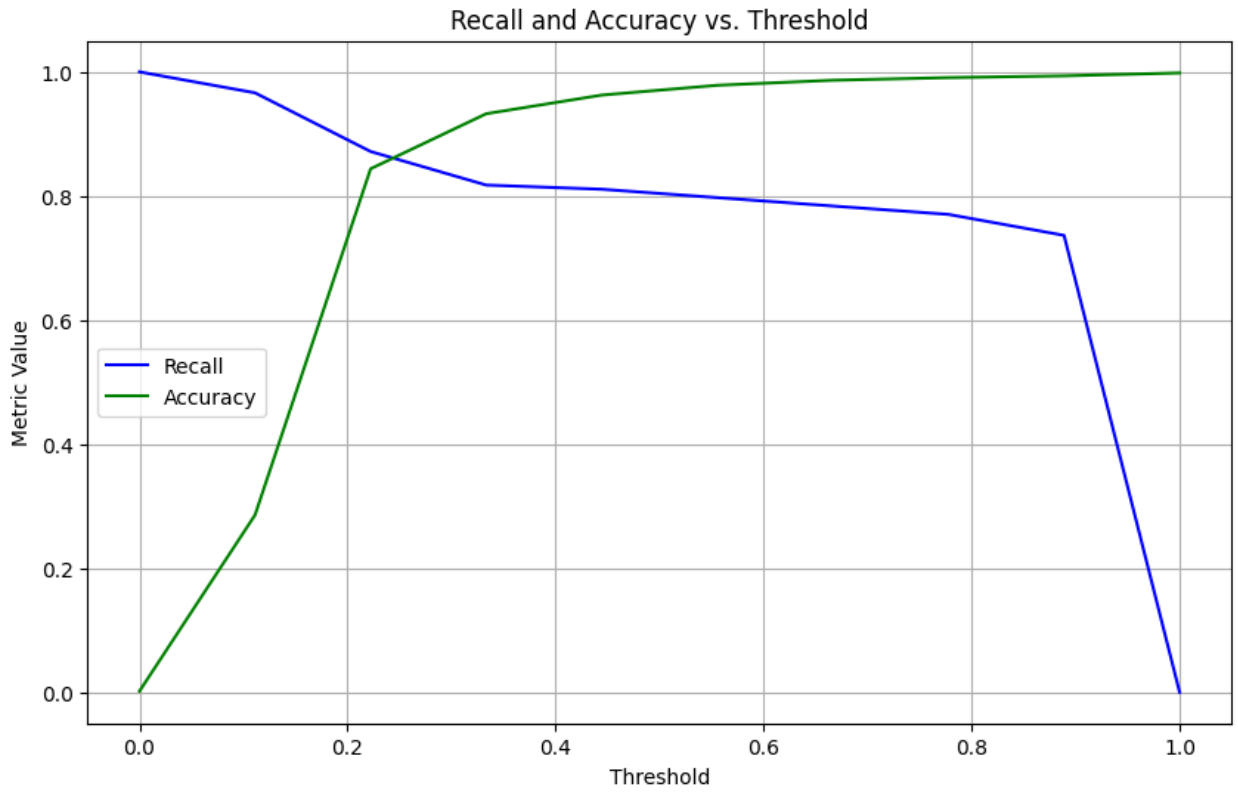
با استفاده از تابع زیر مقادیر به ازای هایthreshold متفاوت حساب شده است.

```
۱
۲
```

```python
def calculate_metrics(outputs, labels, thresholds):
    recalls = []
    accuracies = []

    for threshold in thresholds:
        predictions = (outputs > threshold).astype(int)
        recall = recall_score(labels, predictions)
        accuracy = accuracy_score(labels, predictions)
        recalls.append(recall)
        accuracies.append(accuracy)

    return recalls, accuracies

```

Code :۱۰ Thresholding

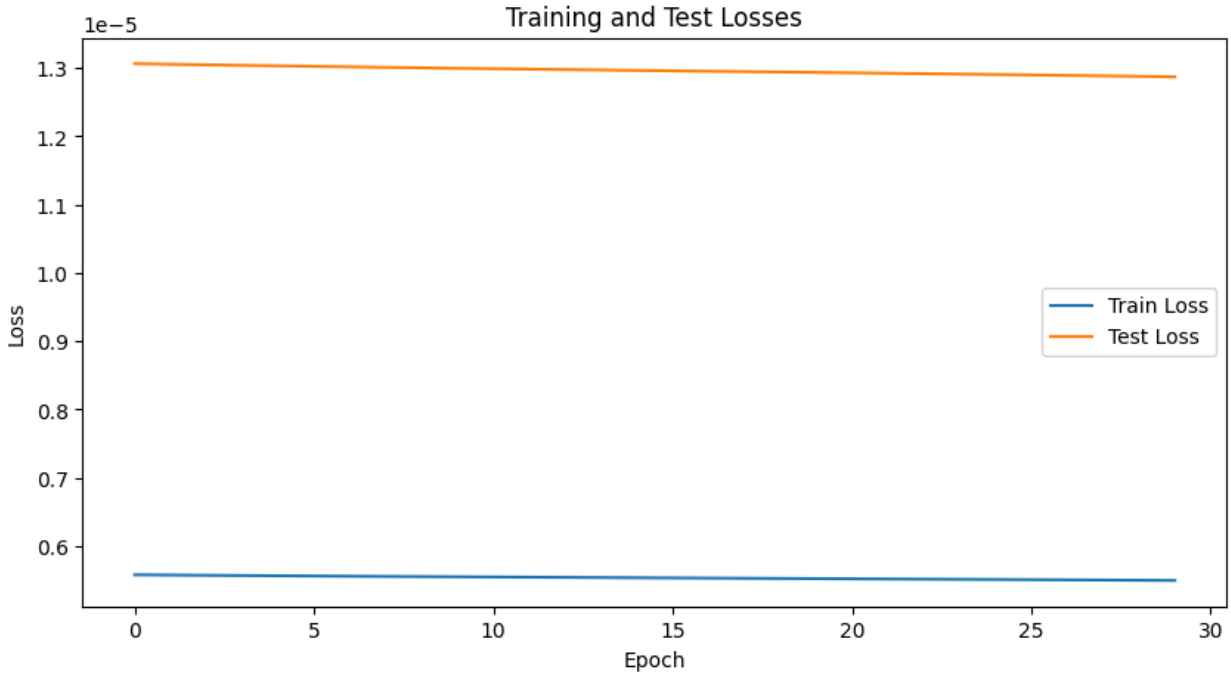
که نتیجه اجرای کد فوق در شکل ۶ قابل مشاهده می‌باشد.



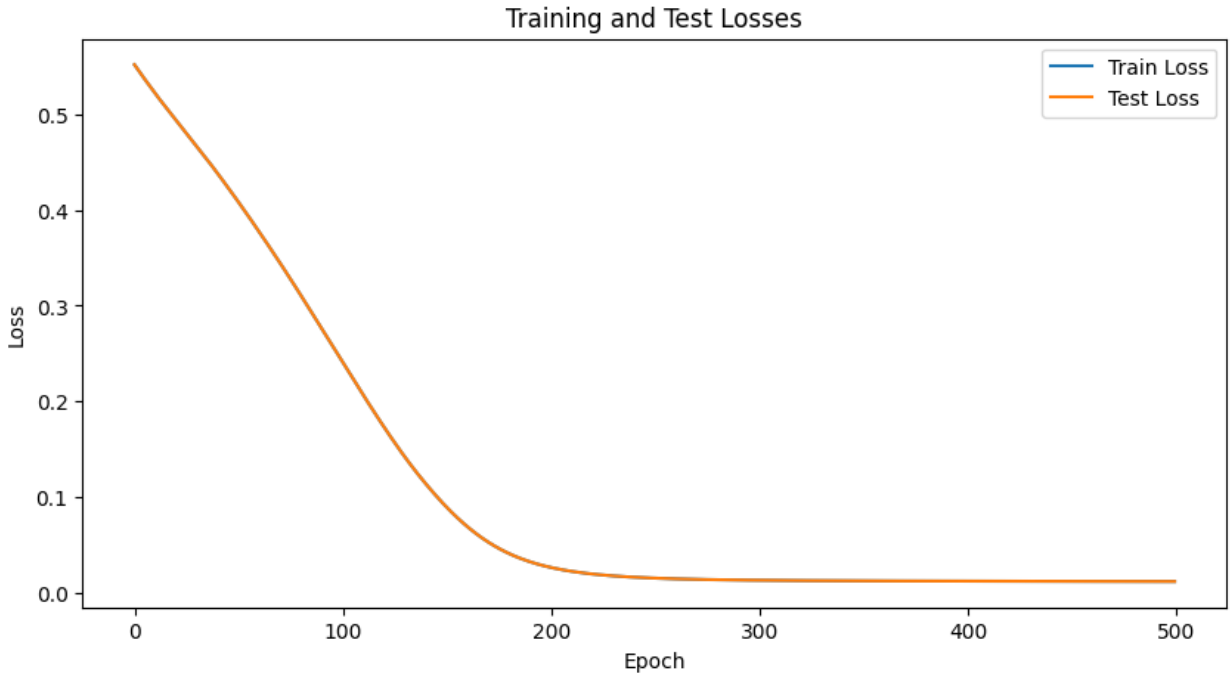
شکل ۶: recall and accuracy on effect Threshold

۶.۳ بخش ششم

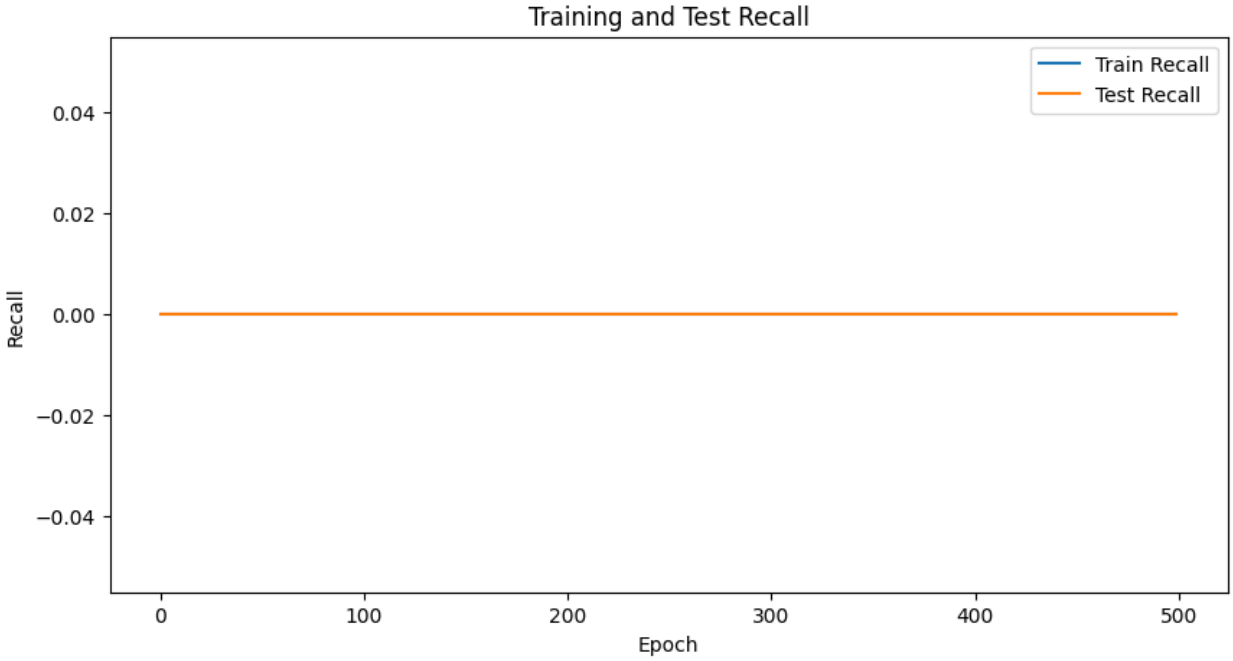
تمام قسمت‌های قبلی را روی دیتاست موجود اجرا می‌کنیم با این تفاوت که بلاک Over-sampling اجرا نخواهد شد. نمودارها و نتایج این آزمایش در ادامه گزارش شده است.



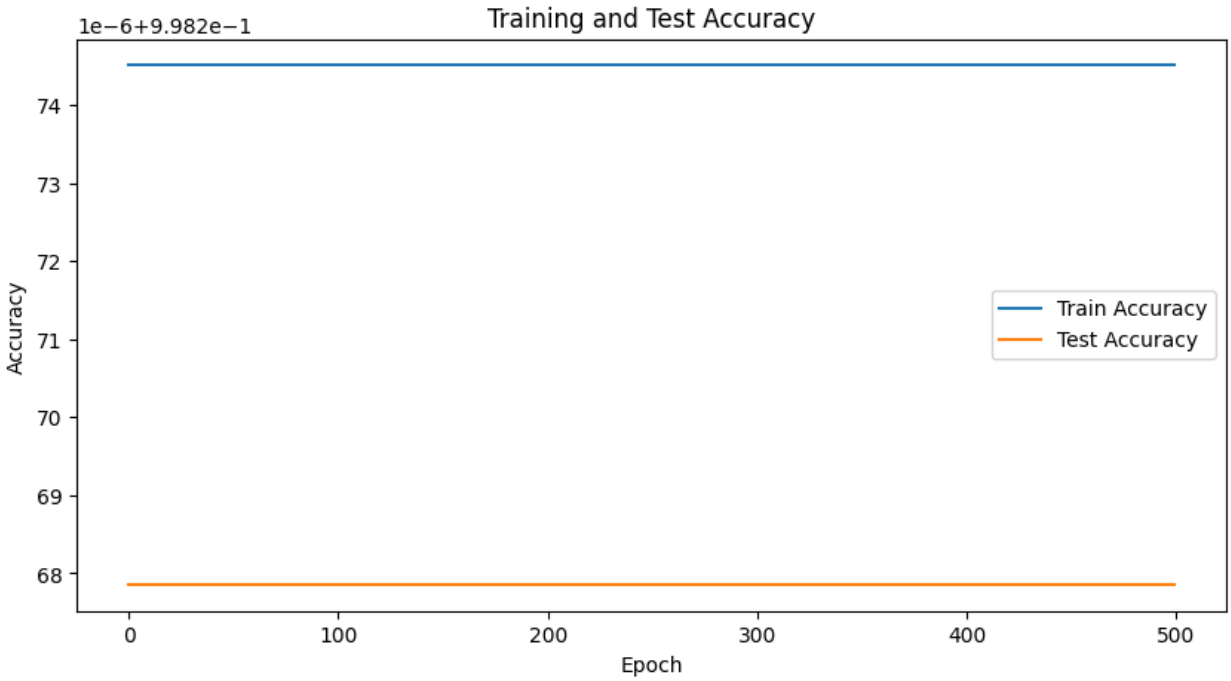
شکل ۷: auto-encoder denoising on set test and train of value Loss



شکل ۸: training of value Loss

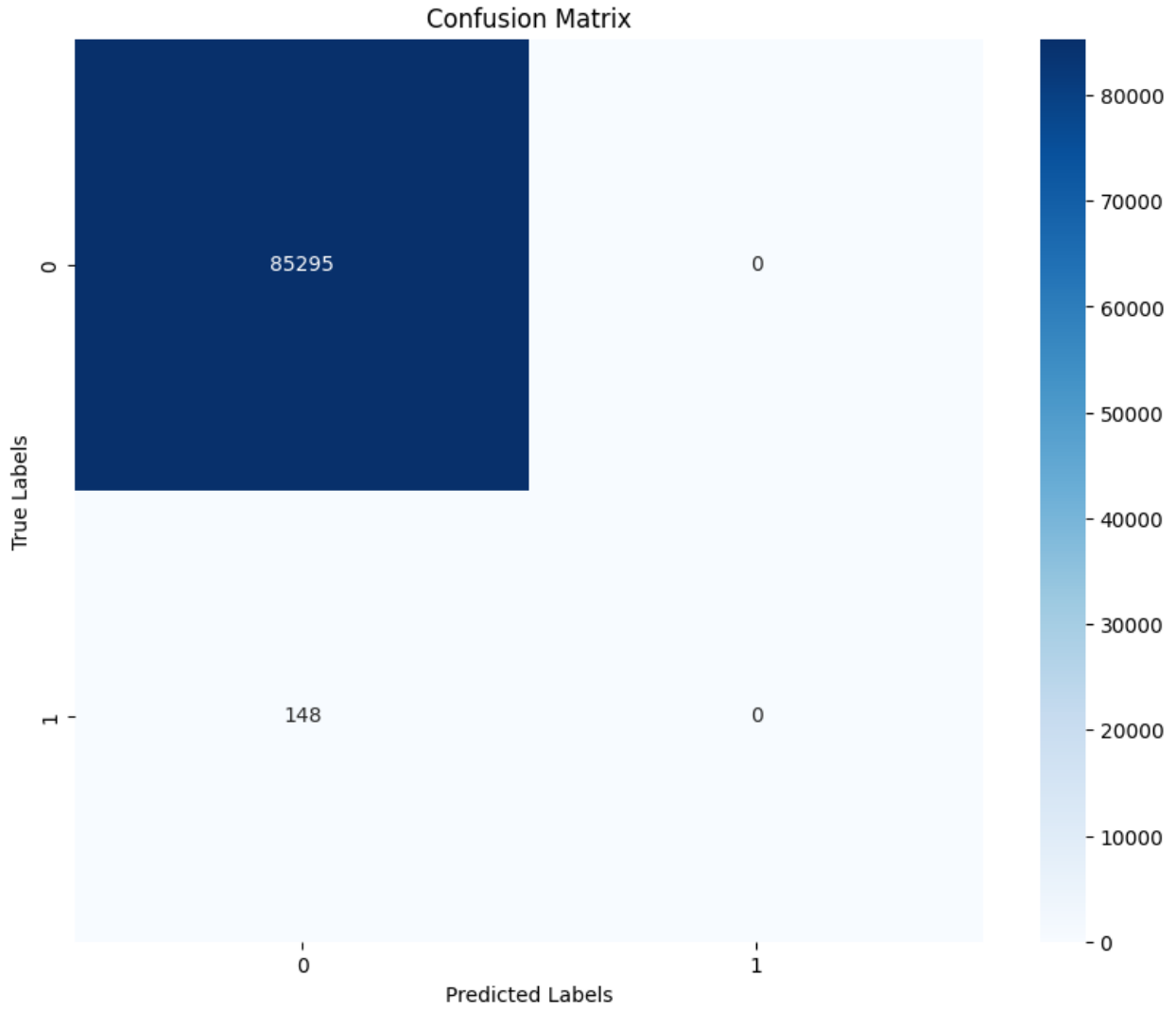


شکل ۹: training of Recall

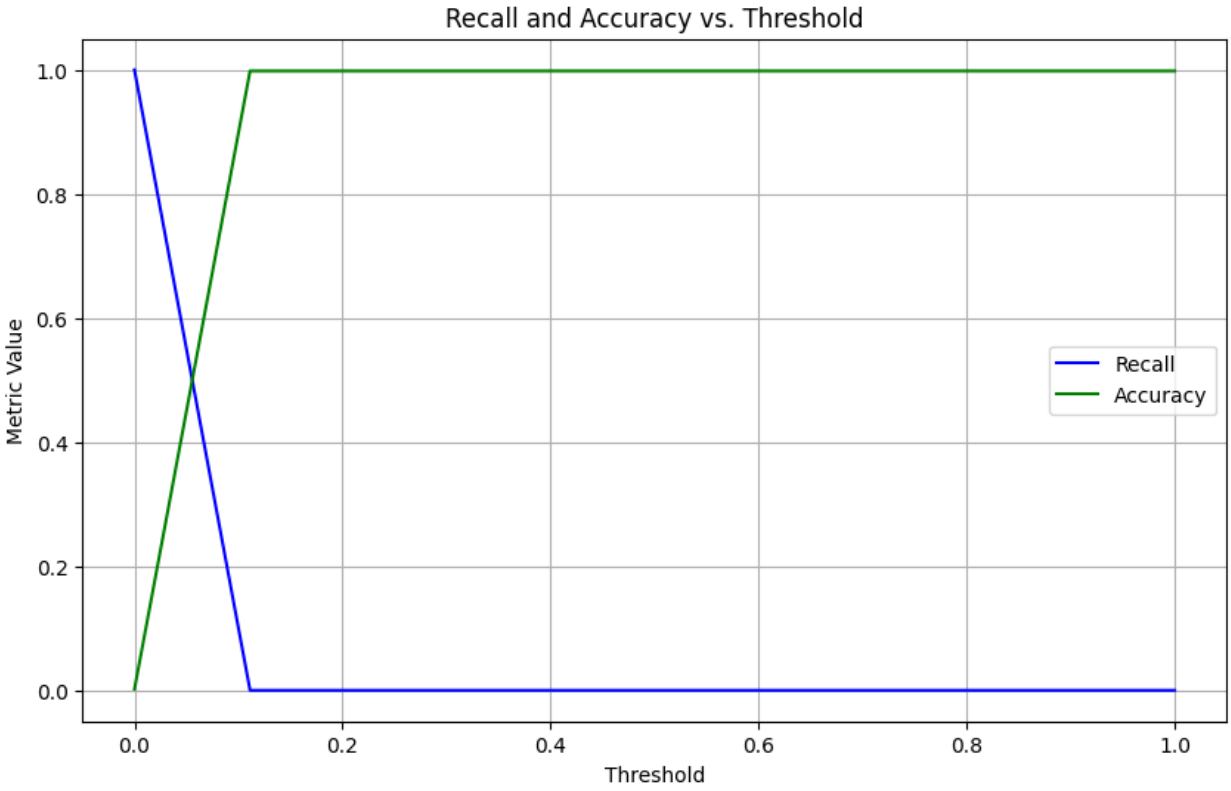


شکل ۱۰: training of Accuracy

	precision	recall	f1-score	support
0	00.1	00.1	00.1	85295
1	00.0	00.0	00.0	148
accuracy			00.1	85443
macro avg	50.0	50.0	50.0	85443
weighted avg	00.1	00.1	00.1	85443



شکل ۱۱: matrix Confusion



شکل ۱۲: accuracy and recall on effect Threshold

همانطور که انتظار می‌رفت نتایج بسیار افت کرده است و اما متریک Accuracy برابر ۱ شده است. اگر به شکل ۱۱ توجه کنیم، مشخص است که مدل تمام نمونه‌ها را به عنوان تقلب در نظر می‌گیرد که نتیجه آن این است که Recall برابر ۰ خواهد شد اما accuracy به مقدار ۱ بسیار نزدیک می‌شود.

مراجع

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

[۲] Towards Data Science. "Ensemble learning: Random forest, adaboost, gradient boosting — step by step explained." Towards Data Science، ۲۰۲۰. Accessed: ۲۰-۰۵-۲۰۲۴.

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