به نام خدا


دانشگاه صنعتی خواجه نصیرالدین

دانشکده برق


# یادگیری ماشین

## گزارش تمرین شماره ۳

**شیما سادات ناصری**

**۴۰۱۱۲۸۱۴**

**دکتر مهدی علیاری شوره دلی**

خرداد ۱۴۰۳

# فهرست مطالب

عنوان شماره صفحه


سوال ۲ ..... ۳

سوال ۳ ..... ۸

۱ ..... ۸

۲ ..... ۹

۳ ..... ۹

بررسی معیارهای ارزیابی مدل در مسائل با داده‌های نامتوازن ..... ۱۳

۴ ..... ۱۴

۵ ..... ۱۵

مراجع ..... ۱۷

## سوال ۲

روش GenSVM یک رویکرد جدید برای پرداختن به مشکلات طبقه بندی با بیش از دو کلاس را نشان می دهد. در حالی که ماشین‌های بردار پشتیبان سنتی (SVM) در تمایز بین دو دسته برتری دارند، زمانی که با دسته‌های متعدد مواجه می‌شوند، کارایی کمتری دارند. GenSVM یک راه حل جدید برای رسیدگی موثر به این مشکل چند کلاسه ارائه می دهد.

بخش‌های مهم این روش و نوآوری‌های آن به صورت زیر است:

- رمزگذاری Simplex:

موقعیتی را در نظر بگیرید که در آن سه نوع گل مختلف برای طبقه بندی دارید. به جای استفاده از یک سیستم مختصات منظم، GenSVM از یک **simplex** برای نمایش این کلاس ها استفاده می کند.

"سیمپلکس" نسخه پیشرفته تر مثلث یا چهار وجهی است. برای سه طبقه، شبیه قرار دادن هر نوع گل در گوشه های یک مثلث متساوی الاضلاع است. این روش تضمین می کند که کلاس ها به طور مساوی فاصله دارند و به راحتی قابل تشخیص هستند.

- تابع ضرر

GenSVM از یک تابع hinge استفاده می کند که در SVM ها رایج است. طبقه‌بندی‌های اشتباه را جریمه می‌کند، اما با کمی انعطاف‌پذیری، به آن اجازه می‌دهد تا بر اساس مجموعه داده‌های مختلف تنظیم شود.

در فرایند آموزش، مدل مقادیر بهینه ماتریس وزن W و بردار بایاس t را می آموزد. برای هر نقطه داده، مدل موقعیت خود را در فضای سیمپلکس محاسبه می کند.

در صورتی که مدل پیش بینی نادرستی انجام دهد، تابع ضرر جریمه قابل توجهی را اعمال می کند. این شامل هم از دست دادن لولا و هم یک اصطلاح منظم سازی برای جلوگیری از برازش بیش از حد می شود، که فرآیند برازش مدل بسیار نزدیک به داده های آموزشی است.

- بهینه سازی

بهینه سازی فرآیند شناسایی مقادیر بهینه برای پارامترهای مدل ما برای به حداقل رساندن خطاها است. GenSVM این فرآیند را با پرداختن مستقیم به شکل اولیه مسئله بهینه سازی (یعنی کار با مشکل

اصلی به جای نسخه پیچیده تر) ساده می کند. از یک الگوریتم عمده سازی تکراری[1] استفاده می کند که شامل ایجاد مکرر یک نسخه ساده شده از مسئله، حل آن و استفاده از راه حل برای بهبود نسخه بعدی است. این شبیه به بالا رفتن از یک تپه با برداشتن گام های کوچک و قابل کنترل به جای تلاش برای پرش مستقیم به قله است.

پس از آموزش، مدل می تواند نقاط داده جدید را با نگاشت آنها به فضای سیمپلکس و پیدا کردن نزدیکترین کلاس پیش بینی کند.

پیاده سازی مدل در هر بخش به صورت زیر است.

- بخش رمز گذاری:

```python
def _simplex_coordinates(self, K):
    U = np.zeros((K, K - 1))
    for k in range(K):
        if k > 0:
            U[k, :k] = -1 / np.sqrt(k * (k + 1))
        if k < K - 1:
            U[k, k] = k / np.sqrt((k + 1) * (k + 2)) # Avoid division by zero and invalid values
    return U
```

با استفاده از این کد، مختصات سیمپلکس مورد استفاده برای رمزگذاری کلاس ها در فضای بعدی K-1 ساخته می‌شود.

- بخش آموزش (تابع ضرر و بهینه سازی):

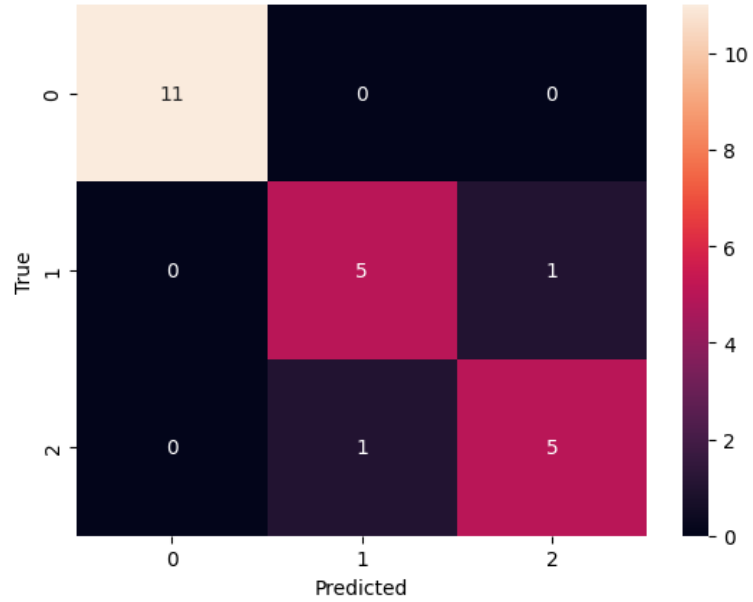
```python
def fit(self, X, y, rho):
    n_samples, n_features = X.shape
    classes = np.unique(y)
    n_classes = len(classes)
    self.classes_ = classes

    # Initialize weights and bias
    self.W = np.random.randn(n_features, n_classes - 1) * 0.01
    self.t = np.random.randn(n_classes - 1) * 0.01

    # Define the simplex encoding
    U = self._simplex_coordinates(n_classes)

    def loss(params):
        W = params[:n_features * (n_classes - 1)].reshape(n_features, n_classes - 1)
        t = params[n_features * (n_classes - 1):]
        loss = 0.0
        for i in range(n_samples):
            xi = X[i]
            yi = y[i]
            si = np.dot(xi, W) + t
            q = np.array([si @ (U[yi] - U[j]) for j in range(n_classes) if j != yi])
            hinge_loss = np.sum(np.maximum(0, 1 - q)**self.p)
            loss += rho[i] * hinge_loss
        loss /= n_samples
        loss += self.lambd * np.sum(W**2)
        return loss

    initial_params = np.hstack([self.W.ravel(), self.t])
    result = minimize(loss, initial_params, method='L-BFGS-B', tol=self.epsilon, options={'maxiter': self.max_iter})
    self.W = result.x[:n_features * (n_classes - 1)].reshape(n_features, n_classes - 1)
    self.t = result.x[n_features * (n_classes - 1):]
```

[1] Iterative Majorization Algorithm

روش برازش پارامترهای مدل را آغاز می کند و بهینه سازی را انجام می دهد. سپس برنامه با استفاده از تابع simplex_coordinates_ مختصات سیمپلکس را محاسبه می‌کند.

همانطور که در مقاله توضیح داده شده است، تابع ضرر به عنوان یک افت لولا با یک اصطلاح منظم تعریف می شود. تابع Minimize از scipy.optimize برای به حداقل رساندن تلفات و شناسایی مقادیر بهینه W و t استفاده می‌شود.

نتایج این پیاده سازی پس از جدا کردن داده آموزش و تست از هم به صورت زیر است:




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 11 |
| 1 | 0.83 | 0.83 | 0.83 | 6 |
| 2 | 0.83 | 0.83 | 0.83 | 6 |
| accuracy | | | 0.91 | 23 |
| macro avg | 0.89 | 0.89 | 0.89 | 23 |
| weighted avg | 0.91 | 0.91 | 0.91 | 23 |

```
Test Accuracy: 0.9130434782608695
Test Recall: 0.9130434782608695
Test F1-score: 0.9130434782608695
Test Precision: 0.9130434782608695
```

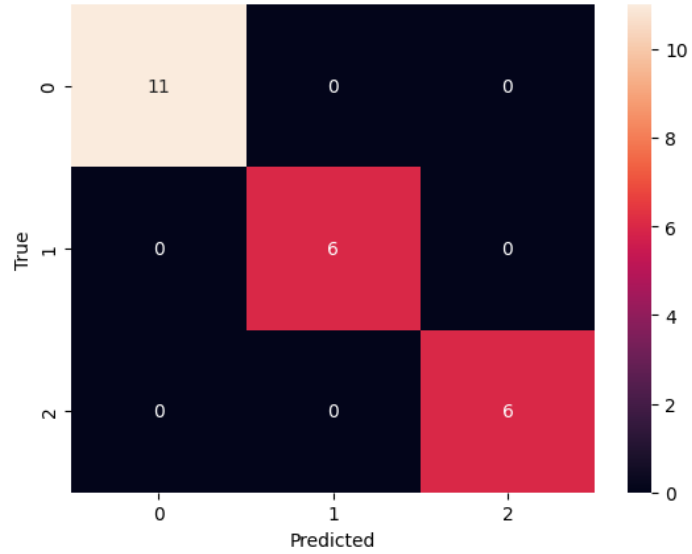
با مقایسه نتایج بدست آمده با نتیجه مقاله، مشاهده می‌شود نیاز است تا براساس مقاله از gridsearch استفاده کنیم. این متد را با کمک gridsearch اما با بازه‌هایی آزادانه‌تر بررسی می‌کنیم تا بهترین پارامتر را پیدا کنیم. کد به صورت زیر است:

```python
from sklearn.metrics import accuracy_score

# Define parameters
rho = np.ones(X_train.shape[0])
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, accuracy_score

# Define a custom scorer
accuracy_scorer = make_scorer(accuracy_score)

# Define the parameter grid
param_grid = {
    'C': [0.1, 1.0, 10.0],
    'p': [1.0, 2.0],
    'kappa': [0.1, 0.5, 1.0],
    'lambd': [0.01, 0.1, 1.0],
    'epsilon': [1e-6, 1e-4, 1e-2]
}

# Initialize the grid search with cross-validation
grid_search = GridSearchCV(GenSVM(), param_grid, scoring=accuracy_scorer, cv=5 , verbose=2)

# Fit the grid search on the training data
grid_search.fit(X_train, y_train, rho=rho)

# Get the best parameters and the best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best parameters found: ", best_params)
print("Best cross-validation accuracy: {:.2f}".format(best_score))

# Predict and evaluate on the test set using the best parameters
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

بهترین پارامتر بدست آمده به صورت زیر است:

```
Best parameters found:  {'C': 0.1, 'epsilon': 1e-06, 'kappa': 0.1, 'lambd': 0.01, 'p': 2.0}
```

نتایج روی دیتای تست به صورت زیر است:




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 11 |
| 1 | 1.00 | 1.00 | 1.00 | 6 |
| 2 | 1.00 | 1.00 | 1.00 | 6 |
| accuracy | | | 1.00 | 23 |
| macro avg | 1.00 | 1.00 | 1.00 | 23 |
| weighted avg | 1.00 | 1.00 | 1.00 | 23 |

```
Test Accuracy: 1.0
Test Recall: 1.0
Test F1-score: 1.0
Test Precision: 1.0
```

نتیجه بدست آمده نشان می‌دهد که برای دیتاست ایریس، مقادیر بدست آمده کاملا مناسب است. از طرف دیگر زمان آموزش برای این دیتاست کوتاه بود و توانسته به صورت کامل دیتای تست را که در فرایند آموزش نبوده را طبقه بندی کند.

# سوال ۳

## ۱

- داده‌های نامتوازن

یکی از بزرگترین چالش‌ها در تشخیص تقلب، نامتوازن بودن شدید داده‌ها است، به‌طوری که تراکنش‌های تقلبی درصد بسیار کمی از کل تراکنش‌ها را تشکیل می‌دهند. این نامتوازن بودن باعث می‌شود که مدل‌های طبقه‌بندی سنتی نتوانند به طور مؤثر تقلب را تشخیص دهند، زیرا تمایل به تشخیص کلاس اکثریت دارند. برای رفع این موضوع ۲ راهکار عمده وجود دارد، یا باید از کلاس بزرگ‌تر داده‌ها برای آموزش حذف شود یا باید داده‌های کلاس اقلیت مشابه سازی و زیاد شوند تا توازن برقرار شود. در این مقاله، برای مقابله با نامتوازن بودن کلاس‌ها، از تکنیک بازنمونه‌گیری با استفاده از SMOTE برای تولید نمونه‌های مصنوعی از کلاس اقلیت استفاده می‌شود و به این ترتیب داده‌ها را متوازن می‌کند.

- نویز در داده‌ها

داده‌های واقعی اغلب حاوی نویز هستند که می‌تواند منجر به خطا در طبقه‌بندی شود. در این مقاله از خودرمزگذار رفع نویز[1] برای مقابله با نویز در داده‌ها استفاده شده‌است. DAE می‌تواند یاد بگیرد که نویز را حذف کند و داده‌های ورودی تمیز را بازسازی کند، بنابراین ویژگی‌های مورد استفاده برای طبقه‌بندی را بهبود می‌بخشد.

- رفتار تقلبی دینامیک

رفتارهای تقلبی با گذشت زمان تغییر می‌کنند، که این موضوع مدل‌های استاتیک را برای هماهنگی با تغییرات دشوار می‌کند.

- انتخاب ویژگی‌ها

شناسایی ویژگی‌های مرتبط برای تشخیص تقلب نیز چالش‌برانگیز و بسیار مهم است.

- معیارهای عملکرد

دقت تنها معیار مناسبی برای ارزیابی مدل‌های تشخیص تقلب نیست به دلیل نامتوازن بودن داده‌ها. برای این موضوع، به جای استفاده accuracy، عملکرد مدل با استفاده از نرخ recall (detection rate)

[1] Denoising Autoencoder (DAE)

ارزیابی می‌شود تا اطمینان حاصل شود که کلاس اقلیت (تراکنش‌های تقلبی) به طور مؤثر تشخیص داده می‌شود.

## ۲

معماری شبکه توضیح داده شده در مقاله شامل دو بخش اصلی است:

۱. خودرمزگذار رفع نویز (DAE):

- لایه ورودی: داده‌های ورودی با نویز را دریافت می‌کند.
- لایه‌های رمزگذار: چندین لایه کاملاً متصل که ابعاد داده‌های ورودی را کاهش می‌دهند.
- لایه گلوگاه: کوچک‌ترین لایه در معماری که ویژگی‌های رمزگذاری شده را نمایش می‌دهد.
- لایه‌های رمزگشا: چندین لایه کاملاً متصل که داده‌ها را از ویژگی‌های رمزگذاری شده بازسازی می‌کنند.
- لایه خروجی: خروجی تمیز شده را تولید می‌کند.
- معماری شامل یک سری لایه‌های کاملاً متصل با ساختار:

Input (29) → FC (22) → FC (15) → FC (10) → FC (15) → FC (22) → Output (29)

۲. مدل طبقه‌بندی:

- لایه ورودی: داده‌های تمیز شده از خودرمزگذار را می‌گیرد.
- لایه‌های پنهان: چندین لایه کاملاً متصل.
- لایه خروجی: از تابع SoftMax برای خروجی احتمالات برای دو کلاس (تقلبی و غیرتقلبی) استفاده می‌کند.
- معماری شامل لایه‌هایی با ساختار:

Input (29) → FC (22) → FC (15) → FC (10) → FC (5) → Output (2)

## ۳

مدل به صورت زیر پیاده سازی شد:

۱. اضافه کردن نویز و اعمال SMOTE:

```python
from imblearn.over_sampling import SMOTE
# Oversampling using SMOTE
smote = SMOTE(sampling_strategy=1.0, random_state=14)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

```python
import numpy as np

# Adding Gaussian noise
noise_factor = 0.2
X_train_noisy = X_train_res + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train_res.shape)
```

لازم به ذکر است که پیش از این داده های تست با نسبت ۰.۲ به کل داده‌ها جدا شدند و در ادامه در فرایند آموزش نیز ۰.۲ داده های آموزش برای validation استفاده خواهند شد.

۲. DAE

مدل بر اساس مقاله به صورت زیر پیاده سازی شد:

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import tensorflow as tf

# Building the Denoising Autoencoder
def build_autoencoder(input_dim):
    model = Sequential()
    model.add(Input(shape=(input_dim,)))
    model.add(Dense(22, activation='relu'))
    model.add(Dense(15, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(15, activation='relu'))
    model.add(Dense(22, activation='relu'))
    model.add(Dense(input_dim, activation='sigmoid'))
    model.compile(optimizer='adam', loss='mse')
    return model

# Training the Autoencoder
input_dim = X_train_res.shape[1]
autoencoder = build_autoencoder(input_dim)

# Callbacks for early stopping and saving the best model
autoencoder_callbacks = [
    EarlyStopping(monitor='val_loss', patience=3, verbose=1),
    ModelCheckpoint('best_autoencoder.h5', monitor='val_loss', save_best_only=True, verbose=1)
]

history = autoencoder.fit(X_train_noisy, X_train_res,
                          epochs=100,
                          batch_size=256,
                          shuffle=True,
                          validation_split=0.2,
                          callbacks=autoencoder_callbacks)

# Load the best autoencoder model
autoencoder.load_weights('best_autoencoder.h5')

# Denoising the training and test datasets
X_train_denoised = autoencoder.predict(X_train_res)
X_test_denoised = autoencoder.predict(X_test)
```

رمزگذار حذف نویز را آموزش دیده و با حذف آن، مجموعه داده را بهبود می‌بخشد.

### ۳. طبقه بندی داده‌های بدون نویز

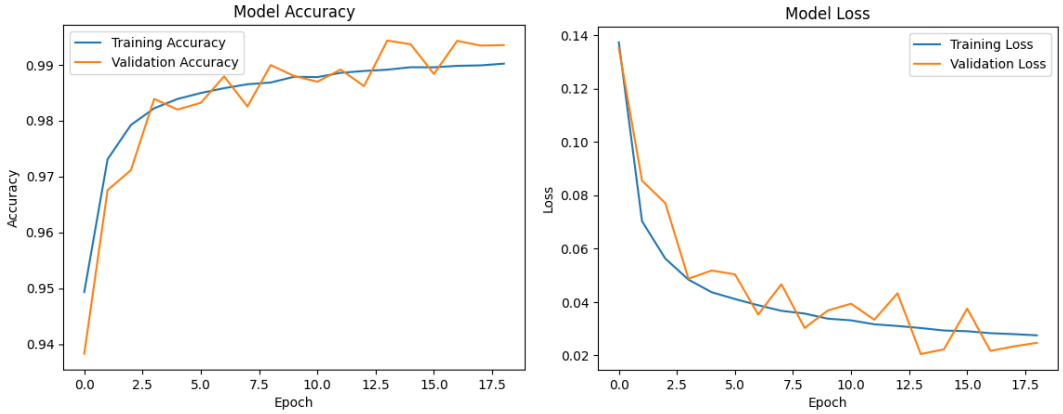
این بخش بر اساس مقاله به صورت زیر پیاده‌سازی شد:

```
def build_classifier(input_dim):
    model = Sequential()
    model.add(Input(shape=(input_dim,)))
    model.add(Dense(22, activation='relu'))
    model.add(Dense(15, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(5, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy', recall_m,precision_m,f1_m])
    return model

classifier = build_classifier(input_dim)

classifier_callbacks = [
    EarlyStopping(monitor='val_loss', patience=5, verbose=1),
    ModelCheckpoint('best_classifier.h5', monitor='val_loss', save_best_only=True, verbose=1)
]

history = classifier.fit(X_train_denoised, y_train_res,
                         epochs=100,
                         batch_size=256,
                         shuffle=True,
                         validation_split=0.2,
                         callbacks=classifier_callbacks)

classifier.load_weights('best_classifier.h5')
```

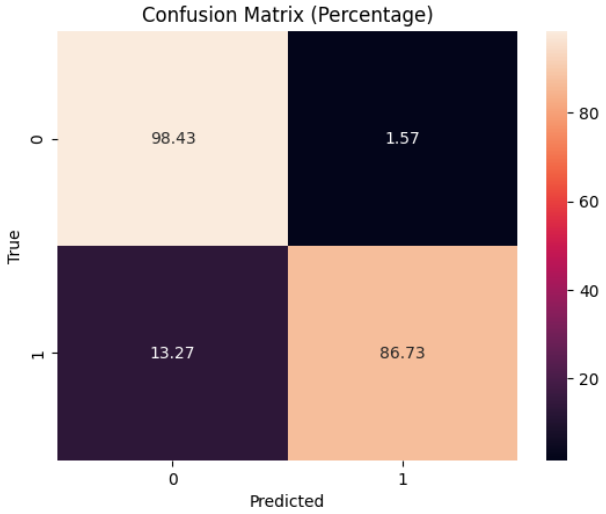
برای آموزش متعادل‌تر از معیارهای زیر در کامپایل استفاده شده‌است:

- recall_m(y_true, y_pred): نسبت مثبت‌های واقعی به مجموع مثبت‌های درست و منفی‌های غلط.
- precision_m(y_true, y_pred): نسبت مثبت های واقعی به مجموع مثبت های درست و مثبت های غلط.
- f1_m(y_true، y_pred): امتیاز F1، میانگین هارمونیک دقت و یادآوری را محاسبه می کند.

استفاده از این موارد، درک بهتر توانایی مدل در طبقه بندی صحیح نمونه های مثبت و منفی را امکان پذیر می‌کند. درنهایت مدل به صورت زیر آموزش دید:




مشاهده می‌شود که مدل اورفیت ندارد و تابع early stopping، آموزش مدل را در زمان مناسبی متوقف کرده‌است. نتیجه تست مدل روی داده‌های تست به صورت زیر است:




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.98 | 0.99 | 56864 |
| 1 | 0.09 | 0.87 | 0.16 | 98 |
| accuracy | | | 0.98 | 56962 |
| macro avg | 0.54 | 0.93 | 0.57 | 56962 |
| weighted avg | 1.00 | 0.98 | 0.99 | 56962 |

```
Test AUC: 0.9258126083860668
Test Recall: 0.9840771040342685
Test F1-score: 0.9840771040342685
Test Precision: 0.9840771040342685
```

همان طور که مشاهده می‌شود، طبقه بندی کننده دقت بالا و عملکرد قوی را از خود نشان می‌دهد.

## بررسی معیارهای ارزیابی مدل در مسائل با داده‌های نامتوازن

استفاده از معیار **دقت (Accuracy)** به تنهایی در مسائل با توزیع نامتوازن برچسب‌ها، معمولاً نمایانگر صحیحی از عملکرد مدل نمی‌باشد و می‌تواند گمراه‌کننده باشد. دلیل این موضوع این است که دقت تنها نسبت پیش‌بینی‌های صحیح به کل پیش‌بینی‌ها را اندازه‌گیری می‌کند، بدون توجه به توزیع کلاس‌ها. در مواردی که داده‌ها نامتوازن هستند، یک مدل می‌تواند با پیش‌بینی غالب کلاس اکثریت، دقت بالایی کسب کند در حالی که نتوانسته است به درستی کلاس اقلیت را پیش‌بینی کند که اغلب از اهمیت بیشتری برخوردار است. برای مثال اگر فرض کنیم یک مسئله دسته‌بندی دودویی داریم که ۹۵٪ نمونه‌ها به کلاس ۰ و فقط ۵٪ به کلاس ۱ تعلق دارند؛ مدلی که همیشه کلاس ۰ را پیش‌بینی می‌کند، دقتی برابر با ۹۵٪ خواهد داشت که به نظر بالا می‌آید، اما این مدل به طور کامل قادر به شناسایی نمونه‌های کلاس ۱ نیست.

برای ارزیابی بهتر مدل‌ها روی داده‌های نامتوازن، معیارهای زیر باید مورد توجه قرار گیرند:

- دقت (Precision):

تعریف: دقت نسبت پیش‌بینی‌های مثبت صحیح به کل پیش‌بینی‌های مثبت (مثبت صحیح + مثبت کاذب) است.

اهمیت: دقت بالا نشان‌دهنده نرخ پایین مثبت‌های کاذب است.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- بازخوانی (Recall یا Sensitivity):

تعریف: بازخوانی نسبت پیش‌بینی‌های مثبت صحیح به کل نمونه‌های واقعی مثبت (مثبت صحیح + منفی کاذب) است.

اهمیت: بازخوانی بالا نشان‌دهنده نرخ پایین منفی‌های کاذب است.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- امتیاز F1 (F1-Score):

تعریف: امتیاز F1 میانگین هارمونیک دقت و بازخوانی است که یک معیار توازن‌بخش بین این دو محسوب می‌شود.

اهمیت: برای موقعیت‌هایی که توازن بین دقت و بازخوانی مهم است مفید می‌باشد.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

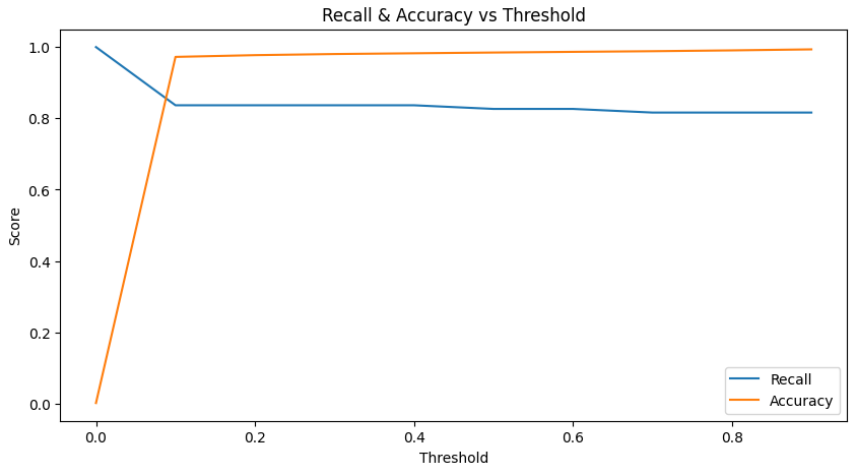
این معیارها می‌توانند در حین آموزش کمک کننده باشند و معیارهایی مانند ماتریس درهم‌ریختگی و مساحت زیر منحنی ROC (ROC-AUC) برای بررسی مدل آموزش دیده می‌توانند بسیار کمک کننده باشند. از سه معیار نام برده شده همانطور که دیده شد در فرایند آموزش استفاده شدند و از معیارهای ماتریس درهم‌ریختگی و AUC نیز در بخش تست مدل روی داده‌های تست استفاده شد.

## ۴

به صورت زیر این مورد بررسی شد:

```
# Lists to store recall and accuracy values
recalls = []
accuracies = []

# Iterate over each threshold
for threshold in thresholds:
    # Oversampling using SMOTE with the current threshold
    smote = SMOTE(sampling_strategy=threshold, random_state=14)
    X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

    # Adding Gaussian noise
    noise_factor = 0.2
    X_train_noisy = X_train_res + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train_res.shape)

    # Training the Autoencoder
    autoencoder = build_autoencoder(input_dim)
    autoencoder.fit(X_train_noisy, X_train_res, epochs=100, batch_size=256, shuffle=True, validation_split=0.2, callbacks=autoencoder_callbacks)
    autoencoder.load_weights('best_autoencoder.h5')

    # Denoising the training dataset
    X_train_denoised = autoencoder.predict(X_train_res)

    # Training the Classifier
    classifier = build_classifier(input_dim)
    classifier.fit(X_train_denoised, y_train_res, epochs=100, batch_size=256, shuffle=True, validation_split=0.2, callbacks=classifier_callbacks)
    classifier.load_weights('best_classifier.h5')

    # Denoising the test dataset
    X_test_denoised = autoencoder.predict(X_test)

    # Evaluating the Classifier
    y_pred = classifier.predict(X_test_denoised)
    y_pred = np.argmax(y_pred, axis=1)

    # Calculating recall and accuracy
    recall = recall_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    # Appending to the lists
    recalls.append(recall)
    accuracies.append(accuracy)
```

خروجی این به صورت زیر است:



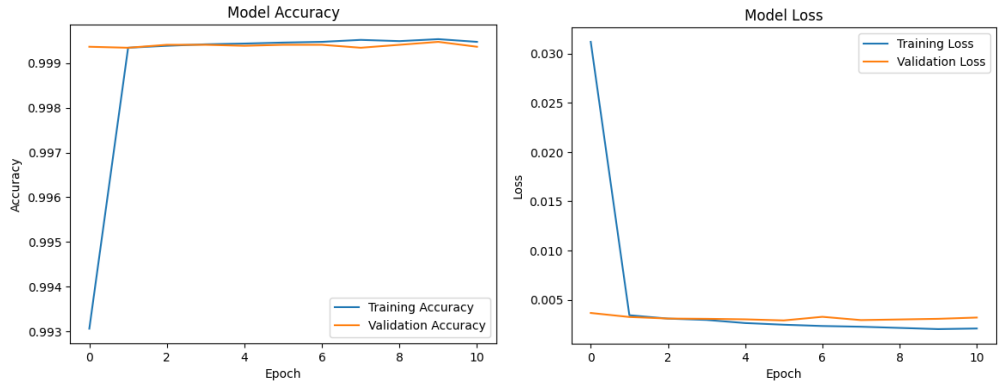

نمودار یادآوری و دقت را برای آستانه هایی از ۰.۱ تا ۰.۹ نشان می دهد. آستانه ۰.۲ ایده آل است. این به طور قابل توجهی هم یادآوری و هم دقت را افزایش داده و نشان می‌دهد مدل در این مرحله به خوبی متعادل و موثر عمل می‌کند.

## ۵

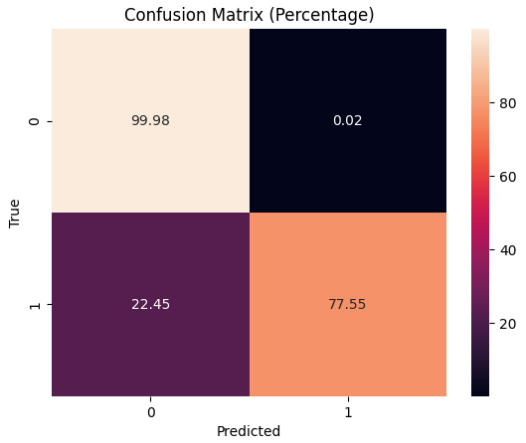
داده‌های نامتوازن را آغشته به نویز کرده و وارد مدل می‌کنیم. از استفاده از معیارهای precision، recall و F1 نیز در فرایند آموزش صرف نظر می‌کنیم:

```
X_train_noisy_norez = X_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train.shape)

def build_classifier(input_dim):
    model = Sequential()
    model.add(Input(shape=(input_dim,)))
    model.add(Dense(22, activation='relu'))
    model.add(Dense(15, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(5, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

input_dim = X_train_noisy_norez.shape[1]

classifier_noisy = build_classifier(input_dim)

classifier_callbacks = [
    EarlyStopping(monitor='val_loss', patience=5, verbose=1),
    ModelCheckpoint('best_classifier_noisy.h5', monitor='val_loss', save_best_only=True, verbose=1)
]

history = classifier_noisy.fit(X_train_noisy_norez, y_train_res,
                               epochs=100,
                               batch_size=128,
                               shuffle=True,
                               validation_split=0.2,
                               callbacks=classifier_callbacks)
```

مدل به صورت زیر آموزش دید:




مشاهده می‌شود که مدل کمی اورفیت دارد ولی تابع early stopping، آموزش مدل را در زمان مناسبی متوقف کرده‌است. نتیجه تست مدل روی داده‌های تست به صورت زیر است:




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56864 |
| 1 | 0.85 | 0.78 | 0.81 | 98 |
| accuracy | | | 1.00 | 56962 |
| macro avg | 0.93 | 0.89 | 0.91 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |

```
Test AUC: 0.8876407942186441
Test Recall: 0.999385555282469
Test F1-score: 0.999385555282469
Test Precision: 0.999385555282469
```

اثر وجود نویز و عدم توازن به خوبی دیده می‌شود. مدل اکنون در تشخیص موارد اکثریت بسیار خوب عمل می‌کند و F1-score بهبود یافته نشان دهنده تعادل بهتر کلی است اما افزایش False Negatives نگران کننده است و نشان می‌دهد مدل موارد بیشتری از اقلیت را از دست می‌دهد. این موضوع در مقدار AUC نیز به خوبی دیده می‌شود.

# مراجع