|  |  |
| --- | --- |
| نام و نام خانوادگی | **محمد پویا افشاری – علیرضا اسمعیل زاده** |
| شماره دانشجویی | **810198351-810198577** |
| تاریخ ارسال گزارش | **1402.09.17** |

|  |  |  |
| --- | --- | --- |
|  | **به نام خدا**  **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر** |  |
| **درس شبکه‌های عصبی و یادگیری عمیق**  **تمرین امتیازی شماره 1** | | |

**فهرست**

[**پاسخ 1** – 3](#_Toc156239925)

[۱-۱ توضیحات مدل 3](#_Toc156239926)

[۲-۱ توضیحات مجموعه دادگان و پیش پردازش آنها 4](#_Toc156239927)

[۲-۱-۱. گرفتن دادگان 4](#_Toc156239928)

[۲-۱-۲. ترسیم نمودار های دادگان 6](#_Toc156239929)

[۲-۱-۳. پیش پردازش دادگان 8](#_Toc156239930)

[۲-۳ ساخت مدل 12](#_Toc156239931)

[۲-۴ ارزیابی و تحلیل نتایج 14](#_Toc156239932)

[**پاسخ** ۳ - تشخیص تقلب 16](#_Toc156239933)

[۳-۱. 16](#_Toc156239934)

[۳-۱-۱. کتابخانه 16](#_Toc156239935)

[۳-۱-۲. نمودار هیستوگرام کلاس‌ها 16](#_Toc156239936)

[۳-۱-۳. چرا نمی توانیم این کلاس ها را اموزش بدهیم؟ 17](#_Toc156239937)

[۳-۲. پياده سازي مدل مقاله 17](#_Toc156239938)

[۳-۲-۱. پيش پردازش 17](#_Toc156239939)

[۳-۲-۲. آموزش با داده هاي unbalanced 18](#_Toc156239940)

[۳-۳. نمونه برداري – به كمك Adaptive Synthetic Sampling (ADASYN) 21](#_Toc156239941)

[۳-۳-۳. توزيح دهيد نمونه برداري بايد قبل از تقسيم داده ها به اموزش و تست انجام بشود يا بعد از ان؟ 24](#_Toc156239942)

[۳-۴. آموزش مدل 27](#_Toc156239943)

**شکل‌ها**

نمایش مثالی از کوتاهترین مسیر وابستگی برای یک جمله 4

معماری مدل 4

تعداد جملات تست و آموزش 6

تعداد کلاس ها در داده های آموزش و تست 7

تعداد نمونه در هر کلاس در کل دادگان 8

نمودار دقت و خطا 14

ماتریس درهم ریختگی 15

توزیع کلاس ها 17

نمودار دقت و خطا 20

ماتریس درهم ریختگی 21

نمودارتوزیع کلاس ها 26

نمودار دقت و خطا 29

ماتریس درهم ریختگی 30

# **پاسخ 1** –

## ۱-۱ توضیحات مدل

مدل مورد بررسی یک شبکه عصبی کانولوشنال بازگشتی دو جهته (BRCNN) است که برای طبقه‌بندی رابطه بر اساس کوتاه‌ترین مسیر وابستگی (SDP) بین موجودیت‌های یک جمله طراحی شده است. این معماری شامل شبکه‌های عصبی بازگشتی دو کاناله است که مجهز به واحدهای حافظه کوتاه‌مدت (LSTM)، لایه‌های کانولوشن و لایه‌های تجمع حداکثری هستند.

ورودی مدل از یک جمله و درخت وابستگی متناظر آن، با تمرکز بر کوتاهترین مسیر وابستگی (SDP) مشتق شده است. دو کانال مجزا به کار گرفته شده است، یکی به کلمات و دیگری به روابط وابستگی اختصاص داده شده است.

واحدهای LSTM نقشی محوری در نمایش کلمات و روابط وابستگی دارند. هر کلمه و رابطه وابستگی با استفاده از جداول جاسازی شده در یک بردار با ارزش واقعی کدگذاری می شود.

در طول آموزش، مدل هم SDP و هم معکوس آن را در نظر می گیرد. طبقه‌بندی‌کننده‌های softmax ریزدانه، طبقه‌بندی کلاس (2K + 1) را برای هر جهت تسهیل می‌کنند. خروجی های دو RCNN به هم متصل شده و به یک طبقه بندی کننده softmax دانه درشت برای طبقه بندی کلاس (K + 1) تغذیه می شوند.

SDP که برای گرفتن روابط موجودیت حیاتی است، بر اساس موقعیت موجودیت ها در درخت وابستگی تعیین می شود.

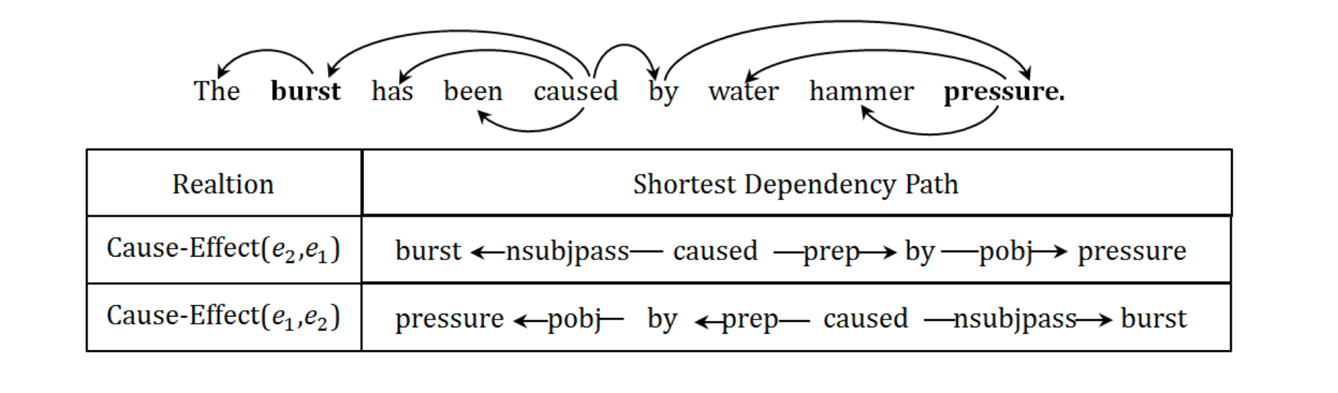
واحدهای LSTM برای مدل‌سازی داده‌های متوالی استفاده می‌شوند که در مدیریت وابستگی‌های بلندمدت مهارت دارند. مکانیسم‌های دروازه‌ای تطبیقی، از جمله دروازه ورودی، دروازه فراموشی، دروازه خروجی و سلول حافظه، به اثربخشی مدل کمک می‌کنند.

لایه‌های کانولوشن ویژگی‌های محلی را از بازنمایی‌های پنهان کلمات همسایه و روابط وابستگی متناظر آن‌ها می‌گیرند.

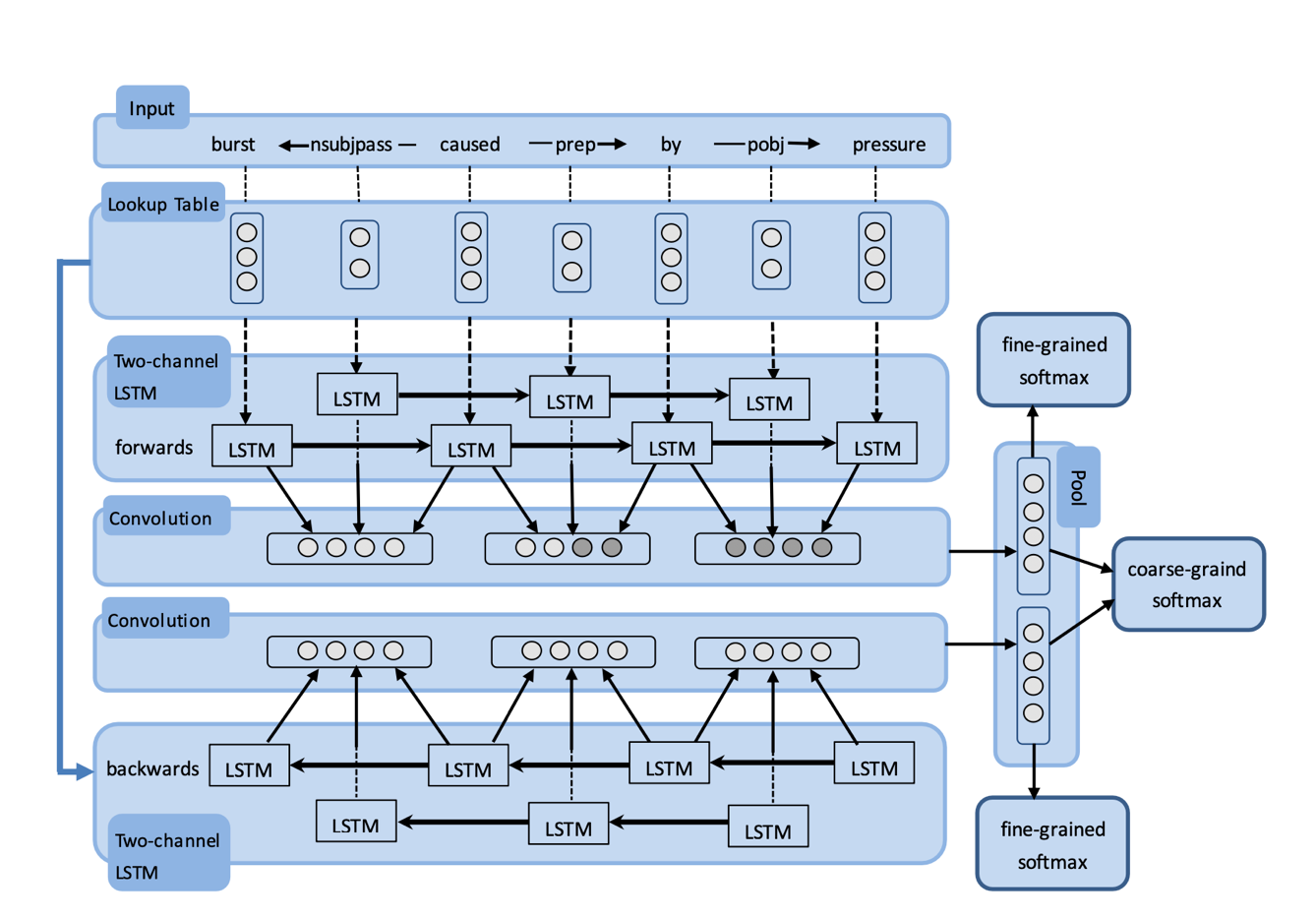
max pooling layer اطلاعات را از ویژگی های محلی استخراج شده از SDP یا معکوس آن جمع می کند. لایه‌های Softmax برای طبقه‌بندی استفاده می‌شوند و توزیع‌های احتمال را برای کلاس‌های مختلف ارائه می‌کنند.

در طول مرحله آموزش، دو طبقه‌بندی‌کننده softmax ریز دانه، طبقه‌بندی کلاس (2K + 1) را برای هر جهت انجام می‌دهند. لایه های ادغام دو RCNN به هم پیوسته اند و یک لایه خروجی softmax دانه درشت برای طبقه بندی کلاس (K + 1) استفاده می شود.

این مدل برای وظایف طبقه‌بندی رابطه، با تأکید خاص بر گرفتن اطلاعات دوطرفه در امتداد SDP در درخت وابستگی یک جمله، طراحی شده است. برای هرگونه سوال یا توضیح خاص، در بحث بیشتر شرکت کنید.



شکل 1نمایش مثالی از کوتاهترین مسیر وابستگی برای یک جمله



شکل 2 معماری مدل

## 

## ۲-۱ توضیحات مجموعه دادگان و پیش پردازش آنها

### ۲-۱-۱. گرفتن دادگان

from google.colab import drive

drive.mount('/content/drive')

اتصال به گوگل کلود برای استفاده از داده های آپلود شده بر روی کلود

with open("/content/drive/MyDrive/SemEval2010\_task8\_all\_data/SemEval2010\_task8\_training/TRAIN\_FILE.TXT") as f:

train\_file = f.readlines()

with open("/content/drive/MyDrive/SemEval2010\_task8\_all\_data/SemEval2010\_task8\_testing\_keys/TEST\_FILE\_FULL.TXT") as f:

test\_file = f.readlines()

در فایل TRAIN\_FILE.TXT دادگان آموزش قرار دارد و که تعداد جملات آن ۸۰۰۰ است و تعداد جملات داده های تست ۲۷۱۷ است که در فایل TEST\_FILE\_FULL.TXT قرار دارند با استفاده از فایل Bidirectional\_Dataset.ipynb که در صورت تمرین قرار داده شده بود این فایل ها را که در کلود قرار گرفته اند را می خوانیم.

def prepare\_dataset(raw):

sentences, relations = [], []

to\_replace = [("\"", ""), ("\n", ""), ("<", " <"), (">", "> ")]

last\_was\_sentence = False

for line in raw:

sl = line.split("\t")

if last\_was\_sentence:

relations.append(sl[0].split("(")[0].replace("\n", ""))

last\_was\_sentence = False

if sl[0].isdigit():

sent = sl[1]

for rp in to\_replace:

sent = sent.replace(rp[0], rp[1])

sentences.append(sent)

last\_was\_sentence = True

print("Found {} sentences".format(len(sentences)))

return sentences, relations

def Labeler(relations):

Label=[]

for i in relations:

if i=='Entity-Destination':

Label.append(0)

if i=='Entity-Origin':

Label.append(1)

if i=='Component-Whole':

Label.append(2)

if i=='Member-Collection':

Label.append(3)

if i=='Other':

Label.append(4)

if i=='Message-Topic':

Label.append(5)

if i=='Content-Container':

Label.append(6)

if i=='Instrument-Agency':

Label.append(7)

if i=='Product-Producer':

Label.append(8)

if i=='Cause-Effect':

Label.append(9)

return Label

تابعی که در فایل Bidirectional\_Dataset.ipynb برای آماده سازی داده ها است را به پروژه اضافه می کنیم.

train\_sentences, train\_relations = prepare\_dataset(train\_file)

test\_sentences, test\_relations = prepare\_dataset(test\_file)

train\_label = Labeler(train\_relations)

test\_label = Labeler(test\_relations)

Found 8000 sentences

Found 2717 sentences

با استفاده از این دو تابع Labler و prepare\_dataset فایل های آموزش و تست را به آرایه از جملات و روابط آن ها تبدیل می کنیم و سپس با استفاده از تابع Labler این روابط را به اعداد ۰ تا ۹ نگاشت می دهیم. تعداد جملات خوانده شده هم مانند مقاله ۸۰۰۰ و ۲۷۱۷ است.

### ۲-۱-۲. ترسیم نمودار های دادگان

import matplotlib.pyplot as plt

# Plotting the distribution of sentence lengths

plt.figure(figsize=(10, 6))

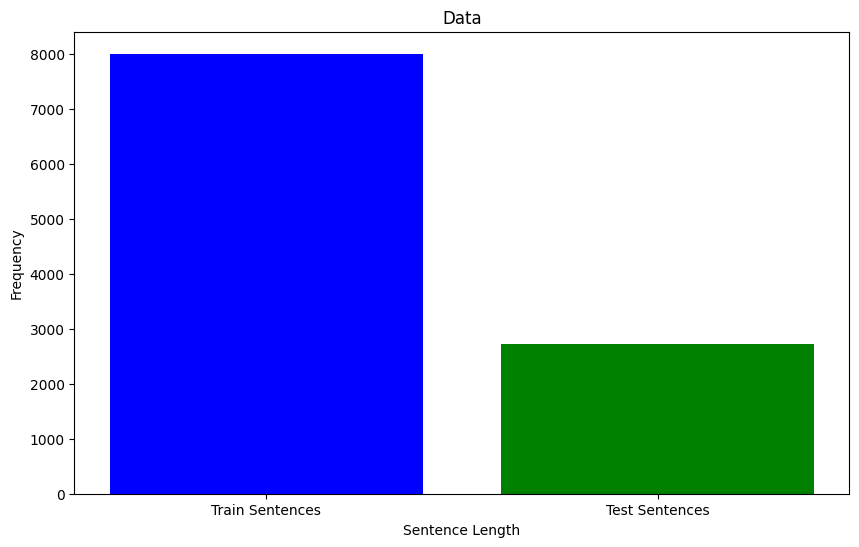
plt.bar(['Train Sentences', 'Test Sentences'], [len(train\_sentences), len(test\_sentences)], color=['blue', 'green'])

plt.title('Data')

plt.xlabel('Sentence Length')

plt.ylabel('Frequency')

plt.show()



شکل 3 تعداد جملات تست و آموزش

تعداد جملات تست و آموزش را که در کد قبلی نمایش دادیم را در اینجا ترسیم می کنیم.

import matplotlib.pyplot as plt

from collections import Counter

# Count occurrences of each integer in train\_label and test\_label

train\_counts = Counter(train\_label)

test\_counts = Counter(test\_label)

# Get unique integers and their counts

train\_values, train\_counts = zip(\*train\_counts.items())

test\_values, test\_counts = zip(\*test\_counts.items())

# Create bar plots for train\_label and test\_label

plt.bar(train\_values, train\_counts, color='blue', label='Train')

plt.bar(test\_values, test\_counts, color='green', label='Test', alpha=0.7) # Use alpha to make bars semi-transparent

# Set labels and title

plt.xlabel('class')

plt.ylabel('Count')

plt.title('Count of class in Train and Test')

# Add legend

plt.legend()

# Show the plot

plt.show()



شکل 4 تعداد کلاس ها در داده های آموزش و تست

در اینجا تعداد کلاس ها در داده های آموزش و تست را نمایش می دهیم برای شمارش کلاس ها از تابع Counter استفاده میکنیم. بیشتر کلاس ها از نوع other (سایر) و کمترین تعداد نیز برای کلاس Instrument Agency است.

all\_label = train\_label + test\_label

counts = Counter(all\_label)

# Get unique integers and their counts

label\_values, label\_counts = zip(\*counts.items())

# Create bar plots for all\_label

plt.bar(label\_values, label\_counts, color='blue', label='Data')

# Set labels and title

plt.xlabel('class')

plt.ylabel('Count')

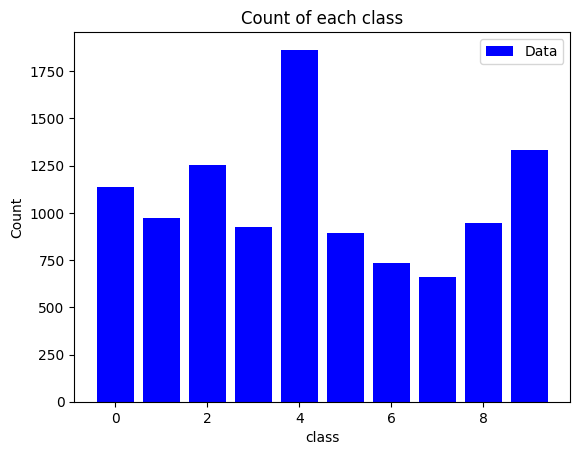
plt.title('Count of each class')

# Add legend

plt.legend()

# Show the plot

plt.show()



شکل 5 تعداد نمونه در هر کلاس در کل دادگان

تعداد نمونه در هر کلاس را در کل داده ها در نظر میگیریم که با مشابه قسمت قبل با تفاوت تجمیع داده ها

print(test\_sentences[0])

The most common <e1> audits </e1> were about <e2> waste </e2> and recycling.

نمایش جمله نمونه از داده های تست

### ۲-۱-۳. پیش پردازش دادگان

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Create a tokenizer

tokenizer = Tokenizer()

# Fit the tokenizer on the training sentences

tokenizer.fit\_on\_texts(train\_sentences)

# Fit the tokenizer on the testing sentences

tokenizer.fit\_on\_texts(test\_sentences)

# Convert training and test sentences to sequences of tokens

train\_sequences = tokenizer.texts\_to\_sequences(train\_sentences)

test\_sequences = tokenizer.texts\_to\_sequences(test\_sentences)

# Get the word index mapping

word\_index = tokenizer.word\_index

# Pad sequences to the same length

max\_length = max(max(len(seq) for seq in train\_sequences), max(len(seq) for seq in test\_sequences))

padded\_train\_sequences = pad\_sequences(train\_sequences, maxlen=max\_length, padding='post')

padded\_test\_sequences = pad\_sequences(test\_sequences, maxlen=max\_length, padding='post')

# Print the results

print("Word Index:")

print(word\_index)

print("Word Index Size:")

print(len(word\_index))

print("\nTraining Sequences:")

print(train\_sequences)

print("\nPadded Training Sequences:")

print(padded\_train\_sequences)

print("\nTest Sequences:")

print(test\_sequences)

print("\nPadded Test Sequences:")

print(padded\_test\_sequences)

print()

Word Index:

{'e1': 1, 'e2': 2, 'the': 3, 'of': 4, 'a': 5, 'and': 6, 'in': 7, 'to': 8, 'is': 9, 'was': 10, 'from': 11, 'by': 12, 'with': 13, 'on': 14, 'that': 15, 'into': 16, …}

Word Index Size:

22900

Training Sequences:

[[3, 101, 19, 515, 419, 22, 47, 1000, 1491, 7, 18, 7631, 1, 4146, 1, 4, 2270, 2, 2271, 2], [3, 1, 436, 1, 10, 1492, 2491, 6, 3117, 16, 3, 2, 2492, 2, 12, 853, 4, 5, 1908], …], …]

Padded Training Sequences:

[[ 3 101 19 ... 0 0 0]

[ 3 1 436 ... 0 0 0]

[ 3 1 240 ... 0 0 0]

...

[ 8 790 20 ... 0 0 0]

[ 3 1436 6 ... 0 0 0]

[ 3 1 1948 ... 0 0 0]]

Test Sequences:

[[3, 68, 260, 1, 19827, 1, 31, 53, 2, 1568, 2, 6, 7353], [3, 1, 87, 1, 9997, 236, 2, 3900, 2], [3, 250, 1, 1757, 1, 3695, 3, 10666, 13, 5, 2, 827, 2], …]

Padded Test Sequences:

[[ 3 68 260 ... 0 0 0]

[ 3 1 87 ... 0 0 0]

[ 3 250 1 ... 0 0 0]

...

[ 3 1 1939 ... 0 0 0]

[22897 3 488 ... 0 0 0]

[ 5 161 307 ... 0 0 0]]

با استفاده از کتابخانه Tokenizer بر روی جملات آموزش و تست با کمک تابع fit\_on\_texts شاخص منحصر به فرد برای هر داده ایجاد می کنیم که تعداد آن برابر Word Index و برابر ۲۲۹۰۰ کلمه است سپس با استفاده از تابع texts\_to\_sequences جملات آموزش و تست را به دنباله ای از شاخص ها تبدیل می کنیم که به هر کلمه یک عدد منحصر به فرد نسبت می دهد به دلیل آنکه طول جملات یکسان نیست نیاز است تا بر روی دنباله ایجاد شده با استفاده از تابع pad\_sequences طول دنباله ها را به ۸۹ میرسانیم که طولانی ترین جمله در داده های آموزش و تست است.

# Count lengths of padded sequences

train\_lengths = Counter(len(seq) for seq in padded\_train\_sequences)

test\_lengths = Counter(len(seq) for seq in padded\_test\_sequences)

print(“Count of Lengths in Training Sequences:”)

print(train\_lengths)

print(“Count of Lengths in Test Sequences:”)

print(test\_lengths)

Count of Lengths in Training Sequences:

Counter({89: 8000})

Count of Lengths in Test Sequences:

Counter({89: 2717})

برای اطمینان از عملکرد تابع pad\_sequences طول دنباله ها را شمارش می کنیم و مشخص است همه آن ها برابر ۸۹ هستند.

num\_classes = max(max(train\_label), max(test\_label)) + 1

print(num\_classes)

10

تعداد کلاس ها برابر ۱۰ است چون در تابع Labler ما خروجی ۰ تا ۹ داشتیم ولی قبل از کد گذاری یک طرفه از تعداد آن اطمینان کسب می کنیم.

from tensorflow.keras.utils import to\_categorical

# Convert train and test labels to one-hot encoding

class\_number = 10

one\_hot\_train\_labels = to\_categorical(train\_label, num\_classes=class\_number)

one\_hot\_test\_labels = to\_categorical(test\_label, num\_classes=class\_number)

# Print the results

print("Original Train Labels:")

print(train\_label)

print("\nOne-Hot Encoded Train Labels:")

print(one\_hot\_train\_labels)

print("\nOriginal Test Labels:")

print(test\_label)

print("\nOne-Hot Encoded Test Labels:")

print(one\_hot\_test\_labels)

Original Train Labels:

[2, 4, 7, 4, 3, 4, 9, 0, 6, 0, 3, 4, 5, 9, 7, 5, 7, 8, 2, 3, 1, 3, 9, 4, 3, 4, 9, 5, 5, 2, 5, 9, 8, 0, 2, 1, 4, 2, 9, 7, 9, 6, 3, 1, 9, 7, 7, 9, 4, 4, 9, 2, 9, 9,…]

One-Hot Encoded Train Labels:

[[0. 0. 1. ... 0. 0. 0.]

[0. 0. 0. ... 0. 0. 0.]

[0. 0. 0. ... 1. 0. 0.]

...

[1. 0. 0. ... 0. 0. 0.]

[0. 0. 0. ... 0. 0. 0.]

[0. 0. 0. ... 0. 1. 0.]]

Original Test Labels:

[5, 8, 7, 0, 9, 2, 8, 3, 2, 5, 0, 4, 0, 8, 1, 1, 0, 4, 3, 8, 5, 6, 8, 4, 1, 8, 9, 4, 4, 1, 9, 5, 2, 8, 2, 2, 3, 6, 3, 8, 9, 2, 9, 0, 1, 6, 4, 0, 5, 4, 0, 4, 3, 4, 9,…]

One-Hot Encoded Test Labels:

[[0. 0. 0. ... 0. 0. 0.]

[0. 0. 0. ... 0. 1. 0.]

[0. 0. 0. ... 1. 0. 0.]

...

[0. 0. 1. ... 0. 0. 0.]

[0. 0. 0. ... 0. 1. 0.]

[1. 0. 0. ... 0. 0. 0.]]

با استفاده از تابع to\_categorical کد گذاری یک طرفه یا همان one hot را انجام می دهیم و همان طور که در خروجی مشخص است کد‌ گذاری درست انجام شده است.

## ۲-۳ ساخت مدل

from keras.models import Sequential

from keras.layers import Embedding, Bidirectional, LSTM, Conv1D, MaxPooling1D, Flatten, Dense

# Model Definition

model = Sequential()

# Embedding Layer

model.add(Embedding(input\_dim=(len(word\_index)+1), output\_dim=50, input\_length=max\_length))

# Bidirectional LSTM

model.add(Bidirectional(LSTM(50, return\_sequences=True)))

# Convolutional Layers

model.add(Conv1D(200, kernel\_size=3, activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Flatten())

# Output Layer

# Assuming you have n\_classes for the number of relation classes

model.add(Dense(class\_number, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Print the model summary

model.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding (Embedding) (None, 89, 50) 1145050

bidirectional (Bidirection (None, 89, 100) 40400

al)

conv1d (Conv1D) (None, 87, 200) 60200

max\_pooling1d (MaxPooling1 (None, 43, 200) 0

D)

flatten (Flatten) (None, 8600) 0

dense (Dense) (None, 10) 86010

=================================================================

Total params: 1331660 (5.08 MB)

Trainable params: 1331660 (5.08 MB)

Non-trainable params: 0 (0.00 Byte)

طبق مقاله در قسمت Hyperparameter Settings مقدار embedding برابر 50-dimensional هستند و به صورت تصادفی مقداردهی اولیه می شوند. لایه های مخفی در هر کانال به اندازه تعداد embedding (200 یا 50) بودند. لایه convolution 200 بعدی بود. به دلیل برابر بودن ابعاد لایه های مخفی و embedding مقدار ۵۰ برای آن انتخاب شده است. شبکه دارای یک Bidirectional LSTM است که خروجی آن به یک لایه convolution سپس به یک MaxPooling متصل میشود و در نهایت به یک لایه Dense اضافه میکنیم تا خروجی برابر تعداد کلاس ها یا همان ۱۰ بدهد.

train\_history = model.fit(padded\_train\_sequences, one\_hot\_train\_labels, epochs=20, batch\_size=32, validation\_split=0.1)

Epoch 1/20

225/225 [==============================] - 34s 150ms/step - loss: 0.2611 - accuracy: 0.9369 - val\_loss: 1.2720 - val\_accuracy: 0.6706

Epoch 2/20

225/225 [==============================] - 33s 146ms/step - loss: 0.0335 - accuracy: 0.9908 - val\_loss: 1.7789 - val\_accuracy: 0.6956

Epoch 3/20

225/225 [==============================] - 32s 142ms/step - loss: 0.0034 - accuracy: 0.9996 - val\_loss: 2.2718 - val\_accuracy: 0.6950

Epoch 4/20

225/225 [==============================] - 33s 146ms/step - loss: 0.0060 - accuracy: 0.9996 - val\_loss: 2.0003 - val\_accuracy: 0.6919

Epoch 5/20

225/225 [==============================] - 31s 136ms/step - loss: 0.0022 - accuracy: 0.9996 - val\_loss: 2.0879 - val\_accuracy: 0.6888

Epoch 6/20

225/225 [==============================] - 31s 137ms/step - loss: 3.2250e-04 - accuracy: 1.0000 - val\_loss: 2.2725 - val\_accuracy: 0.6894

Epoch 7/20

225/225 [==============================] - 30s 132ms/step - loss: 1.7616e-04 - accuracy: 1.0000 - val\_loss: 2.3090 - val\_accuracy: 0.6919

Epoch 8/20

225/225 [==============================] - 30s 133ms/step - loss: 1.0509e-04 - accuracy: 1.0000 - val\_loss: 2.4220 - val\_accuracy: 0.6919

Epoch 9/20

225/225 [==============================] - 31s 136ms/step - loss: 5.7961e-05 - accuracy: 1.0000 - val\_loss: 2.4882 - val\_accuracy: 0.6919

Epoch 10/20

225/225 [==============================] - 32s 140ms/step - loss: 4.2573e-05 - accuracy: 1.0000 - val\_loss: 2.5516 - val\_accuracy: 0.6950

Epoch 11/20

225/225 [==============================] - 32s 142ms/step - loss: 3.3330e-05 - accuracy: 1.0000 - val\_loss: 2.6152 - val\_accuracy: 0.6938

Epoch 12/20

225/225 [==============================] - 32s 142ms/step - loss: 2.6515e-05 - accuracy: 1.0000 - val\_loss: 2.6719 - val\_accuracy: 0.6931

Epoch 13/20

...

Epoch 19/20

225/225 [==============================] - 30s 132ms/step - loss: 7.2637e-06 - accuracy: 1.0000 - val\_loss: 3.0259 - val\_accuracy: 0.6944

Epoch 20/20

225/225 [==============================] - 32s 141ms/step - loss: 6.1608e-06 - accuracy: 1.0000 - val\_loss: 3.0733 - val\_accuracy: 0.6938

طبق مقاله مدل را در ۲۰ دور و با ۸۰۰ داده برای validation که برابر ۱۰ درصد می شود آموزش می دهیم.

# Plotting accuracy

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(train\_history.history['accuracy'], label='Training Accuracy')

plt.plot(train\_history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

# Plotting loss

plt.subplot(1, 2, 2)

plt.plot(train\_history.history['loss'], label='Training Loss')

plt.plot(train\_history.history['val\_loss'], label='Validation Loss')

plt.title('Training and Validation Loss')

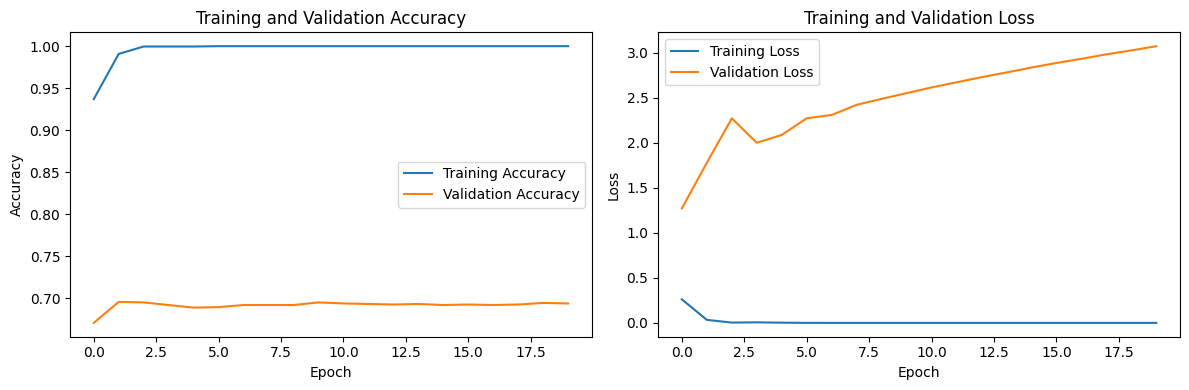
plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.tight\_layout()

plt.show()



شکل 6 نمودار دقت و خطا

نمایش نمودار دقت و خطا هنگام آموزش مدل همان طور که مشخص است دقت مدل بر روی داده آموزشی برابر ۱۰۰ و برای داده ارزیابی برابر ۷۰ درصد است که نتایج قابل قبولی است.

## ۲-۴ ارزیابی و تحلیل نتایج

y\_pred = model.predict(padded\_test\_sequences)

85/85 [==============================] - 4s 41ms/step

اجرای تابع پیشبینی برای تحلیل نتایج

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

import numpy as np

conf\_matrix\_plt\_labels = [f'Class {i}' for i in range(10)]

# Calculate the confusion matrix

conf\_matrix = confusion\_matrix(test\_label, np.argmax(y\_pred, axis=1))

# Plot the confusion matrix using seaborn heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=conf\_matrix\_plt\_labels,

yticklabels=conf\_matrix\_plt\_labels, # Replace with your class labels

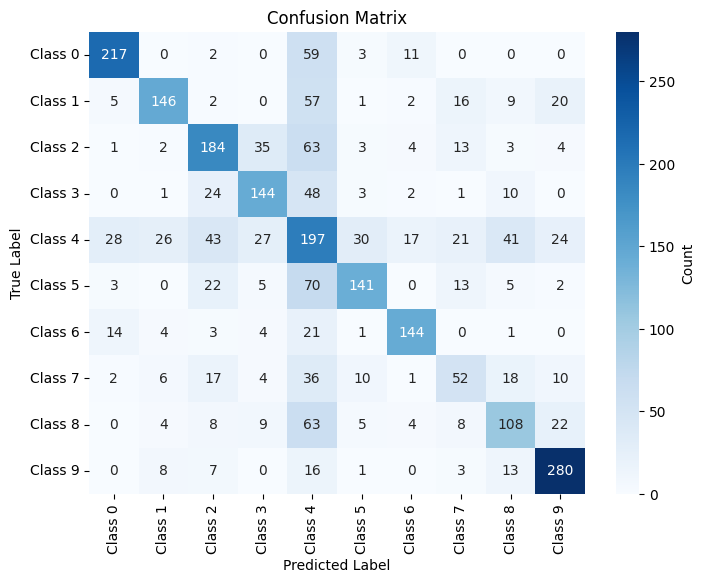
cbar\_kws={'label': 'Count'})

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()



شکل 7 ماتریس درهم ریختگی

ماتریس درهم ریختگی را ترسیم می کنیم برای کلاس ۴ که (سایر) است نتایج کمی نامعتبر است ولی برای کلاس های دیگر عملکرد مدل قابل قبول است مخصوصا کلاس های ۰ و ۹

from sklearn.metrics import classification\_report

classification\_rep = classification\_report(test\_label, np.argmax(y\_pred, axis=1), target\_names=conf\_matrix\_plt\_labels)

print("Classification Report:")

print(classification\_rep)

Classification Report:

precision recall f1-score support

Class 0 0.80 0.74 0.77 292

Class 1 0.74 0.57 0.64 258

Class 2 0.59 0.59 0.59 312

Class 3 0.63 0.62 0.62 233

Class 4 0.31 0.43 0.36 454

Class 5 0.71 0.54 0.61 261

Class 6 0.78 0.75 0.76 192

Class 7 0.41 0.33 0.37 156

Class 8 0.52 0.47 0.49 231

Class 9 0.77 0.85 0.81 328

accuracy 0.59 2717

macro avg 0.63 0.59 0.60 2717

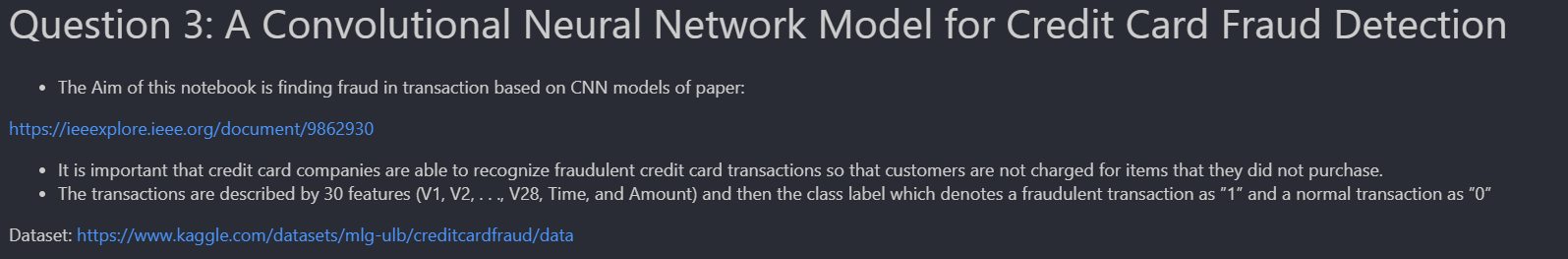
weighted avg 0.62 0.59 0.60 2717

در اینجا f1-score ،recall ،precision را برای هر کلاس محاسبه می کنیم که کلاس ۰ و ۶ بیشترین امتیاز f1 را دارند و کلاس ۴ و ۷ کمترین مقدار f1 که نشاندهنده عملکرد بد مدل در آنجا است.

# **پاسخ** ۳ - تشخیص تقلب

## ۳-۱.

### ۳-۱-۱. کتابخانه



import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import cv2

from sklearn.preprocessing import RobustScaler

import os

import seaborn as sns

import shutil

from google.colab import drive

from google.colab import files

### ۳-۱-۲. نمودار هیستوگرام کلاس‌ها

uploaded = files.upload()

!mkdir -p ~/.kaggle

!mv kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d mlg-ulb/creditcardfraud

!unzip -q creditcardfraud.zip

df = pd.read\_csv("./creditcard.csv")

df

df['Class'].value\_counts()

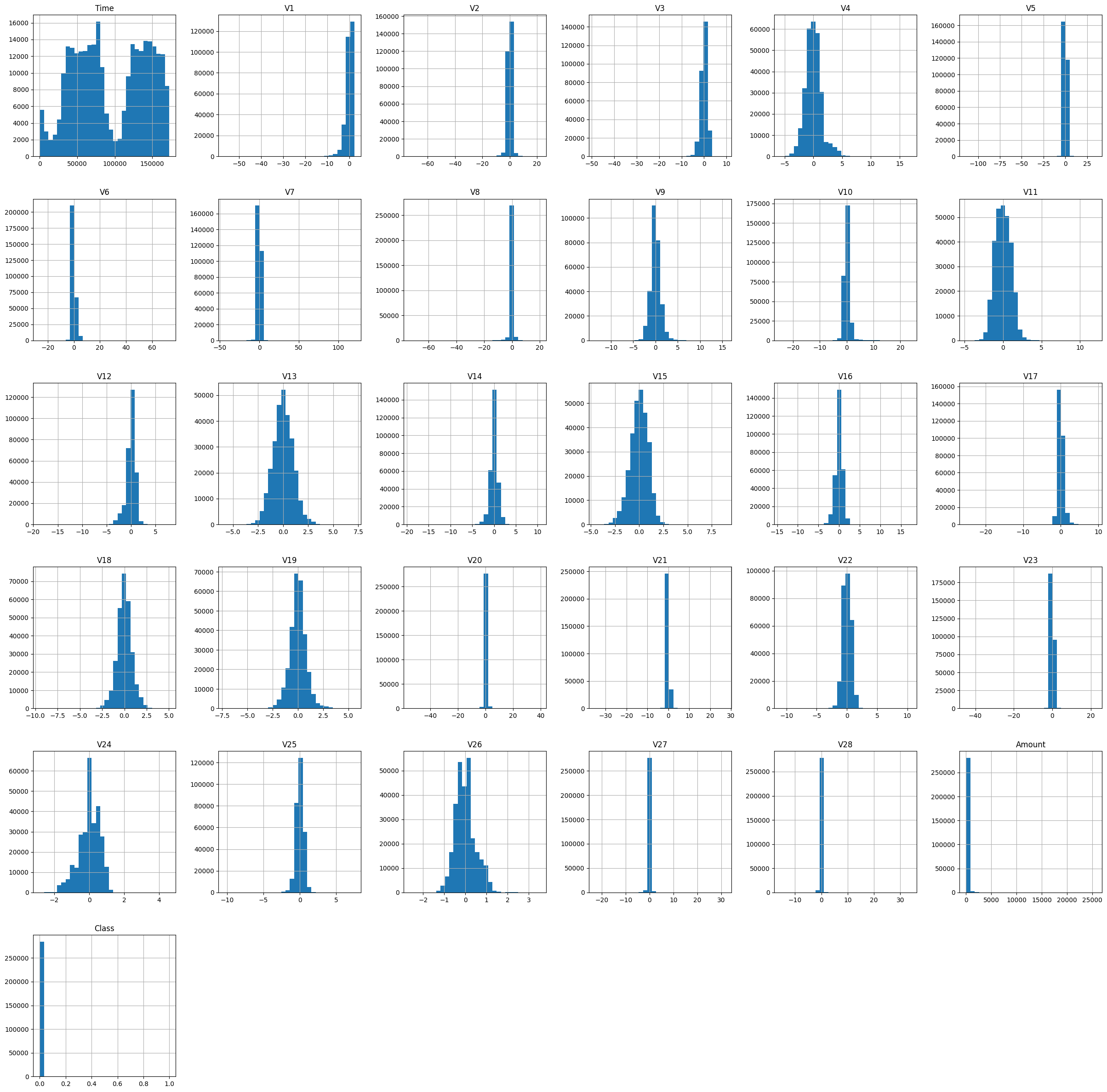
0 284315

1 492

Name: Class, dtype: int64

خب در اینجا متوجه نابرابری توزیع کلاس ها می‌شویم

df.hist(bins=30,figsize=(30,30))



شکل 8 توزیع کلاس ها

df.describe()

### ۳-۱-۳. چرا نمی توانیم این کلاس ها را اموزش بدهیم؟

دیتاست داده شده از تراکنش های حاصله دو روز کاری 492 تا entry fraud و 284.807 تا تراکنش عادی دارد. خب این دیتاست بسیار unbalanced هست برای کلاس fraud و 0.172% کل تراکنش را شامل می‌شود. براي همين هنگام اموزش خيلي روي كلاس negative fragility دارد.

چون 99% تراكنش ها شيادي حساب نمي‌شود الگوريتم ها با احتمال زياد تري هميشه non-fraud براورد مي‌كنند. با اين حالت با درصد بالايي دقت 99% بدست مي‌آيد در داده train. ما دنبال اين كار نيستيم و دنبال generalize بودن هستيم بنابر اين نياز داريم كه new labeling داشته باشيم كه توزيع درست براورد كنيم.

## ۳-۲. پياده سازي مدل مقاله

### ۳-۲-۱. پيش پردازش

new\_df = df.copy()

time = new\_df['Time']

new\_df['Amount'] = RobustScaler().fit\_transform(new\_df['Amount'].to\_numpy().reshape(-1,1))

new\_df['Time'] = (time - time.min()) / (time.max() - time.min())

### ۳-۲-۲. آموزش با داده هاي unbalanced

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, BatchNormalization, MaxPool1D, Dropout, Flatten, Dense

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.losses import BinaryCrossentropy

from tensorflow.keras.metrics import Accuracy

X = df.drop('Class', axis=1)

y = df['Class']

X\_unbalanced = X.values

y\_unbalanced = y.values

X\_unbalanced\_reshaped = X\_unbalanced.reshape(X\_unbalanced.shape[0], X\_unbalanced.shape[1], 1)

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X\_unbalanced\_reshaped, y\_unbalanced, test\_size=0.4, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

print("Training set shape:", X\_train.shape)

print("Validation set shape:", X\_val.shape)

print("Testing set shape:", X\_test.shape)

Training set shape: (170884, 30, 1)

Validation set shape: (56961, 30, 1)

Testing set shape: (56962, 30, 1)

model = Sequential()

model.add(Conv1D(filters=32, kernel\_size=2, activation='relu', input\_shape=(X\_train.shape[1], 1)))

model.add(BatchNormalization())

model.add(MaxPool1D(pool\_size=2))

model.add(Dropout(rate=0.2))

model.add(Conv1D(filters=64, kernel\_size=2, activation='relu'))

model.add(BatchNormalization())

model.add(MaxPool1D(pool\_size=2))

model.add(Dropout(rate=0.5))

model.add(Flatten())

model.add(Dense(units=64, activation='relu'))

model.add(Dropout(rate=0.5))

model.add(Dense(units=64, activation='relu'))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer=Adam(learning\_rate=0.0001),

              loss=BinaryCrossentropy(),

              metrics=[Accuracy()])

model.summary()

Model: "sequential\_4"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv1d\_7 (Conv1D) (None, 29, 32) 96

batch\_normalization\_6 (Bat (None, 29, 32) 128

chNormalization)

max\_pooling1d\_6 (MaxPoolin (None, 14, 32) 0

g1D)

dropout\_9 (Dropout) (None, 14, 32) 0

conv1d\_8 (Conv1D) (None, 13, 64) 4160

batch\_normalization\_7 (Bat (None, 13, 64) 256

chNormalization)

max\_pooling1d\_7 (MaxPoolin (None, 6, 64) 0

g1D)

dropout\_10 (Dropout) (None, 6, 64) 0

flatten\_3 (Flatten) (None, 384) 0

...

Total params: 33505 (130.88 KB)

Trainable params: 33313 (130.13 KB)

Non-trainable params: 192 (768.00 Byte)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

history = model.fit(X\_train, y\_train, epochs=6, batch\_size=32, validation\_data=(X\_val, y\_val))

Epoch 1/6

5341/5341 [==============================] - 45s 8ms/step - loss: 0.0183 - accuracy: 0.0000e+00 - val\_loss: 0.0064 - val\_accuracy: 0.0000e+00

Epoch 2/6

5341/5341 [==============================] - 38s 7ms/step - loss: 0.0078 - accuracy: 0.0000e+00 - val\_loss: 0.0058 - val\_accuracy: 1.7556e-05

Epoch 3/6

5341/5341 [==============================] - 38s 7ms/step - loss: 0.0066 - accuracy: 5.8519e-06 - val\_loss: 0.0048 - val\_accuracy: 0.0000e+00

Epoch 4/6

5341/5341 [==============================] - 38s 7ms/step - loss: 0.0059 - accuracy: 0.0000e+00 - val\_loss: 0.0044 - val\_accuracy: 0.0000e+00

Epoch 5/6

5341/5341 [==============================] - 38s 7ms/step - loss: 0.0052 - accuracy: 0.0000e+00 - val\_loss: 0.0039 - val\_accuracy: 0.0000e+00

Epoch 6/6

5341/5341 [==============================] - 38s 7ms/step - loss: 0.0053 - accuracy: 0.0000e+00 - val\_loss: 0.0043 - val\_accuracy: 0.0000e+00

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

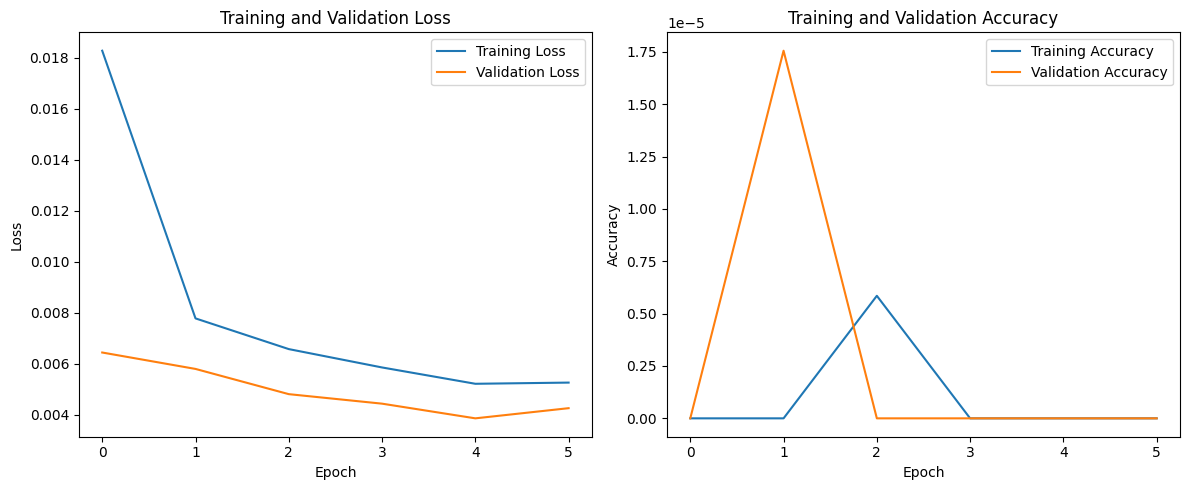
plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()



شکل 9 نمودار دقت و خطا

from sklearn.metrics import confusion\_matrix, classification\_report

import seaborn as sns

import matplotlib.pyplot as plt

*# Obtain model predictions for the test data*

y\_pred = model.predict(X\_test)

y\_pred\_binary = (y\_pred > 0.5).astype(int)  *# Convert to binary predictions (0 or 1)*

*# Calculate and plot the confusion matrix heatmap*

cm = confusion\_matrix(y\_test, y\_pred\_binary)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,

            xticklabels=['Legitimate', 'Fraud'],

            yticklabels=['Legitimate', 'Fraud'])

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

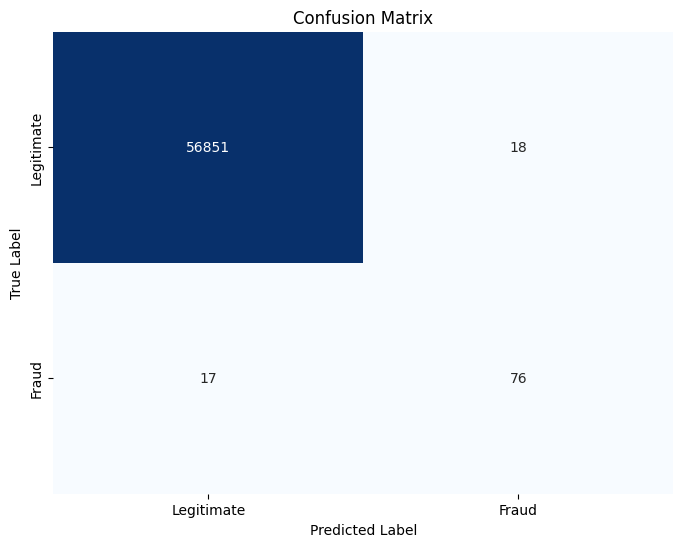
plt.show()

*# Report precision, recall, and F1 score*

report = classification\_report(y\_test, y\_pred\_binary, target\_names=['Legitimate', 'Fraud'])

print("Classification Report:")

print(report)



شکل 10 ماتریس درهم ریختگی

Classification Report:

precision recall f1-score support

Legitimate 1.00 1.00 1.00 56869

Fraud 0.81 0.82 0.81 93

accuracy 1.00 56962

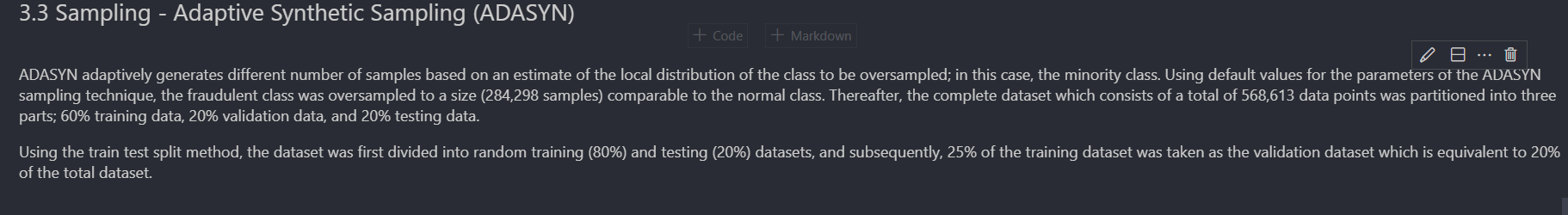
macro avg 0.90 0.91 0.91 56962

weighted avg 1.00 1.00 1.00 56962

خب همانطور كه اشاره كرديم احتمال مي‌رفت كه گزارش نتايج بالا قابل اتكا كردن نباشد.

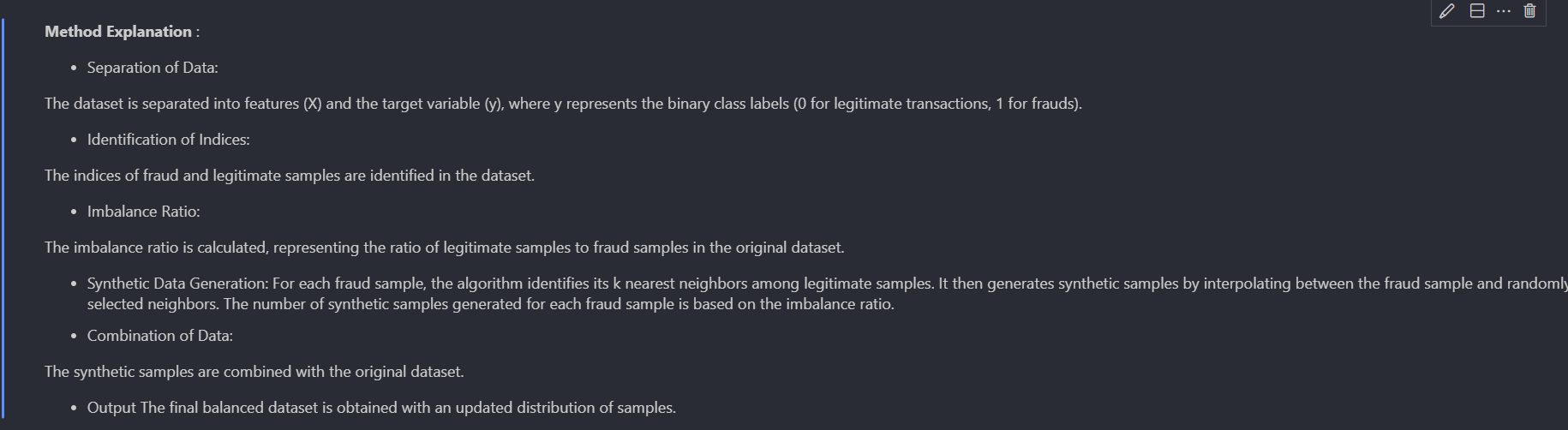
## ۳-۳. نمونه برداري – به كمك Adaptive Synthetic Sampling (ADASYN)

ADASYN



به طور تطبیقی تعداد متفاوتی از نمونه ها را بر اساس تخمینی از توزیع محلی کلاسی که قرار است بیش از حد نمونه برداری شود تولید می کند. در این مورد، طبقه اقلیت. با استفاده از مقادیر پیش‌فرض برای پارامترهای تکنیک نمونه‌گیری ADASYN، کلاس تقلبی به اندازه (284298 نمونه) بیش از حد نمونه‌گیری شد که با کلاس معمولی قابل مقایسه بود. پس از آن، مجموعه داده کامل که از مجموع 568613 نقطه داده تشکیل شده است به سه بخش تقسیم شد. 60 درصد داده های آموزشی، 20 درصد داده های اعتبار سنجی و 20 درصد داده های آزمایشی.

با استفاده از روش train test split دادگان به صورت رندم به 80 20 براي train و test و 25 درصد داده هاي باقي مانده training كه متعاقبا معادل 20 درصد كل ديتاست است برايvalidation در نظر گرفته شد.



توزيح متد:

* جداسازي ديتا

در اين بخش feature (x) و target (y) جداسازي مي‌شودكه y يك باينري دوتايي 0 1 هست.

* مشخص سازي indices

The indices از fraud و legitimate مشخص مي‌شود.

* مشخص كردن Imbalance ratio

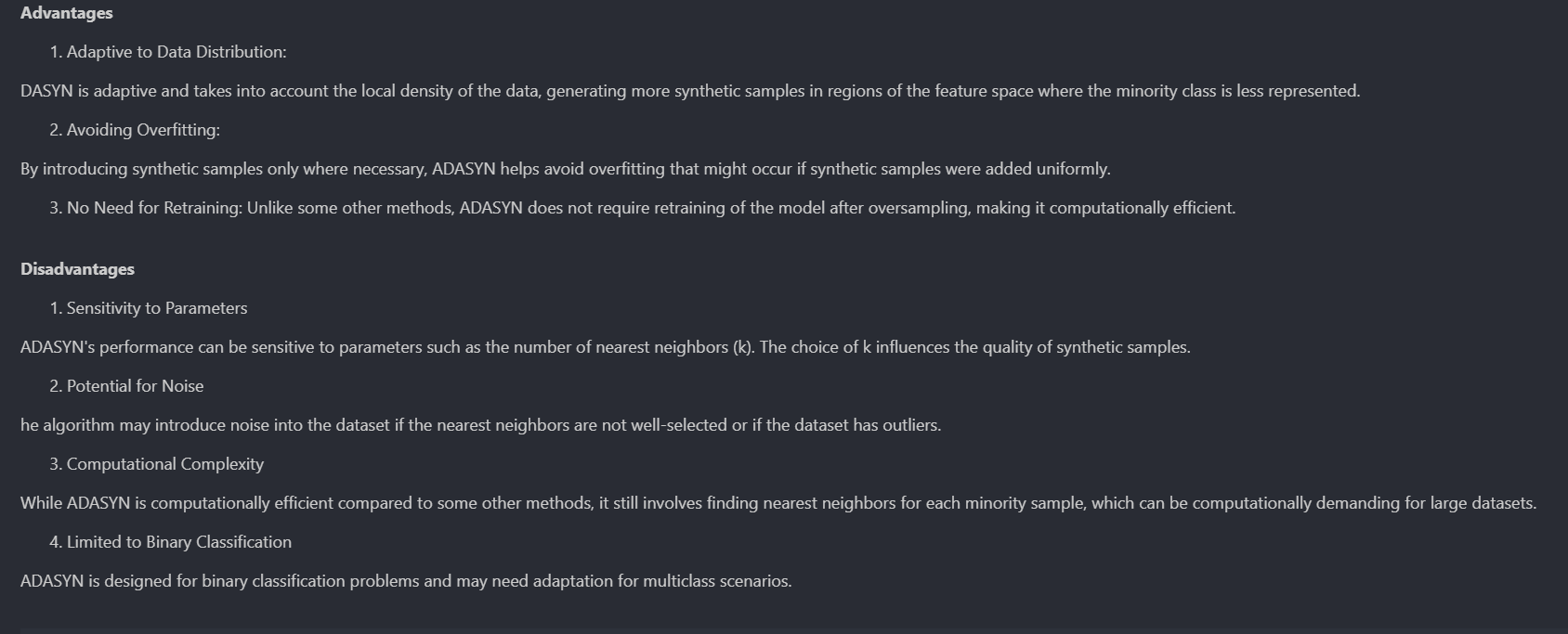
Ratio ناوزني بين دادگان مشخص و حساب مي‌شود.

* ساخت ديتاي مصنوعي

1. برای هر نمونه تقلب، الگوریتم k نزدیکترین همسایه خود را در بین نمونه های قانونی شناسایی می کند.
2. سپس با درون یابی بین نمونه تقلب و همسایگان به طور تصادفی انتخاب شده، نمونه های مصنوعی تولید می کند.

تعداد نمونه های مصنوعی تولید شده برای هر نمونه تقلب بر اساس نسبت عدم تعادل است.

* تركيب ديتا



مزايا و معايب:

مزايا:

1. سازگار با توزیع داده:

DASYN تطبیقی است و چگالی محلی داده ها را در نظر می گیرد و نمونه های مصنوعی بیشتری را در مناطقی از فضای ویژگی ایجاد می کند که در آن کلاس اقلیت کمتر نمایش داده می شود.

2. اجتناب از Overfit:

ADASYN با معرفی نمونه‌های مصنوعی فقط در صورت لزوم، به جلوگیری از برازش بیش از حد که ممکن است در صورت اضافه شدن یکنواخت نمونه‌های مصنوعی رخ دهد، کمک می‌کند.

3. عدم نیاز به آموزش مجدد:

برخلاف برخی روش‌های دیگر، ADASYN نیازی به آموزش مجدد مدل پس از نمونه‌برداری بیش‌ازحد ندارد و آن را از نظر محاسباتی کارآمد می‌کند.

معايب:

1. حساسیت به پارامترها

عملکرد ADASYN می تواند به پارامترهایی مانند تعداد نزدیکترین همسایگان (k) حساس باشد. انتخاب k بر کیفیت نمونه های مصنوعی تأثیر می گذارد.

2. پتانسیل برای نویز

اگر نزدیکترین همسایگان به خوبی انتخاب نشده باشند یا اگر مجموعه داده دارای مقادیر پرت باشد، الگوریتم ممکن است نویز را به مجموعه داده وارد کند.

3. پیچیدگی محاسباتی

در حالی که ADASYN از نظر محاسباتی در مقایسه با برخی روش‌های دیگر کارآمد است، اما همچنان شامل یافتن نزدیک‌ترین همسایگان برای هر نمونه اقلیت است، که می‌تواند از نظر محاسباتی برای مجموعه‌های داده بزرگ نیاز باشد.

4. محدود به طبقه بندی باینری

ADASYN برای مسائل طبقه بندی باینری طراحی شده است و ممکن است برای سناریوهای چند کلاسه نیاز به تطبیق داشته باشد.

*# Separate the dataset into features (X) and the target variable (y)*

X = df.drop('Class', axis=1)

y = df['Class']

*# Identify the indices of fraud and legitimate samples*

fraud\_indices = np.where(y == 1)[0]

legitimate\_indices = np.where(y == 0)[0]

*# Calculate the imbalance ratio*

imbalance\_ratio = len(legitimate\_indices) / len(fraud\_indices)

*# Initialize variables for the synthetic data*

synthetic\_features = []

synthetic\_labels = []

*# Loop over each fraud sample and generate synthetic samples*

for fraud\_index in fraud\_indices:

*# Find the k nearest neighbors of the fraud sample*

    k\_neighbors = np.argsort(np.linalg.norm(X.values[legitimate\_indices] - X.values[fraud\_index], axis=1))[:5]

*# Calculate the number of synthetic samples to generate for the current fraud sample*

    num\_synthetic\_samples = int(imbalance\_ratio) - 1

*#  Generate synthetic samples*

    for \_ in range(num\_synthetic\_samples):

        random\_neighbor\_index = np.random.choice(k\_neighbors)

        synthetic\_sample = X.values[fraud\_index] + np.random.rand() \* (X.values[random\_neighbor\_index] - X.values[fraud\_index])

        synthetic\_features.append(synthetic\_sample)

        synthetic\_labels.append(1)  *# Label for fraud sample*

*#  Combine the synthetic samples with the original dataset*

X\_balanced = np.vstack((X.values, np.array(synthetic\_features)))

y\_balanced = np.concatenate((y.values, np.array(synthetic\_labels)))

*# Print the dataset distribution before and after balancing*

print("TABLE I. DATASET DISTRIBUTION")

print("Dataset \t Legitimate \t Fraud \t\t Total")

print("Before Balancing \t {} \t\t {} \t\t {}".format(len(legitimate\_indices), len(fraud\_indices), len(y)))

print("After Balancing \t {} \t\t {} \t\t {}".format(len(np.where(y\_balanced == 0)[0]), len(np.where(y\_balanced == 1)[0]), len(y\_balanced)))

TABLE I. DATASET DISTRIBUTION

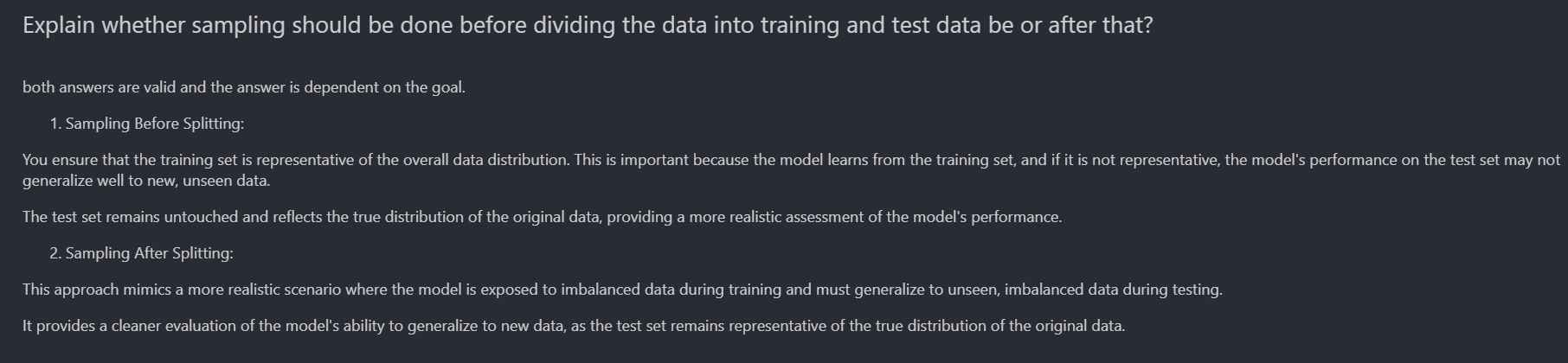
Dataset Legitimate Fraud Total

Before Balancing 284315 492 284807

After Balancing 284315 283884 568199

همان طور كه بالا محاسبه كرديم توزيع داده ها برابر شده است.

### ۳-۳-۳. توزيح دهيد نمونه برداري بايد قبل از تقسيم داده ها به اموزش و تست انجام بشود يا بعد از ان؟



هر دو پاسخ معتبر است و پاسخ به هدف بستگی دارد.

1. نمونه برداری قبل از تقسیم:

شما مطمئن می شوید که مجموعه آموزشی نماینده توزیع کلی داده است. این مهم است زیرا مدل از مجموعه آموزش یاد می گیرد و اگر نماینده نباشد ، عملکرد مدل در مجموعه آزمون ممکن است به خوبی به داده های جدید و غیب تعمیم ندهد.

مجموعه آزمون دست نخورده باقی مانده و توزیع واقعی داده های اصلی را منعکس می کند و ارزیابی واقعی تری از عملکرد مدل ارائه می دهد.

2. نمونه برداری پس از تقسیم:

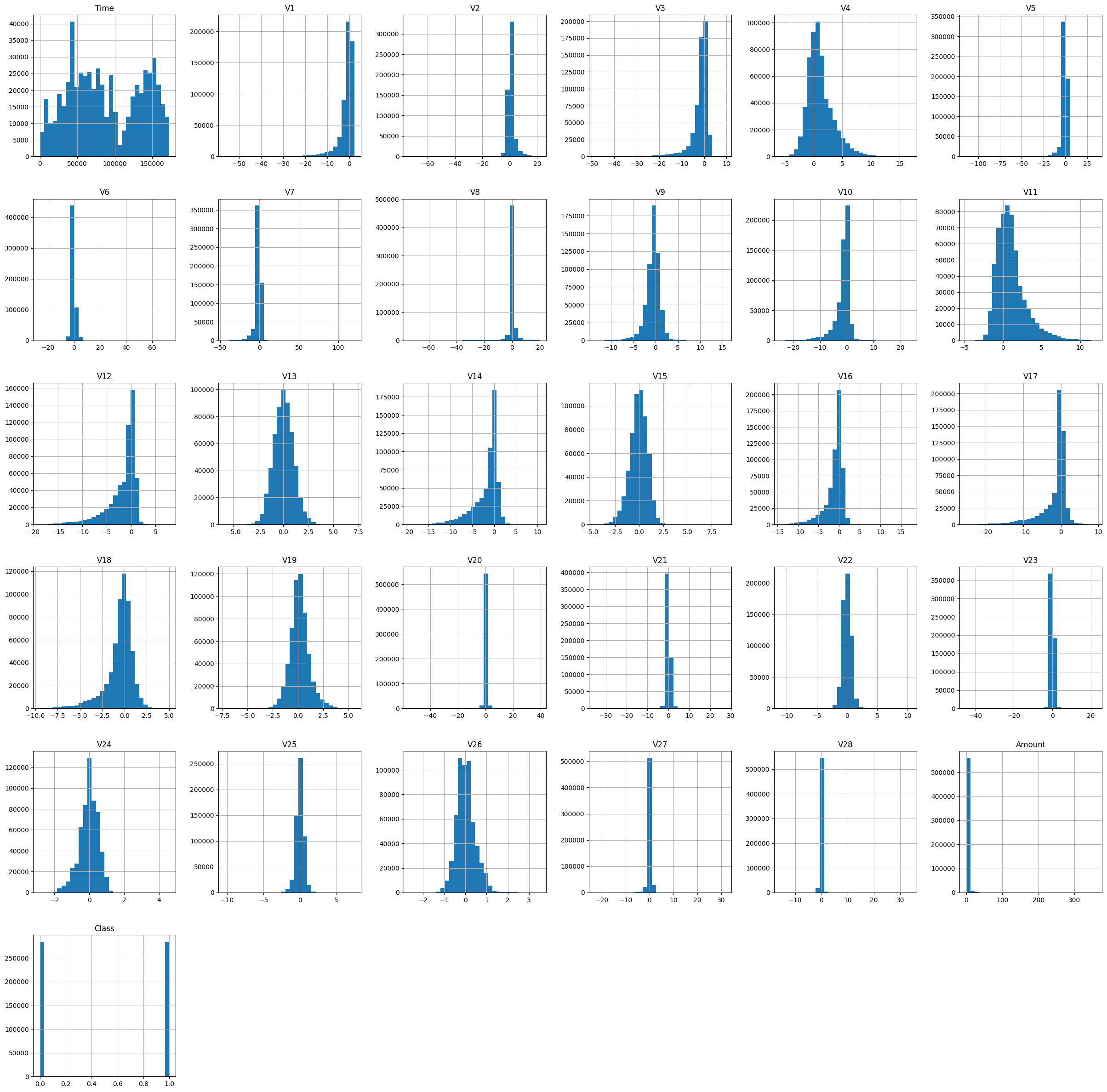
این رویکرد سناریوی واقع بینانه تری را تقلید می کند که در آن مدل در طول آموزش در معرض داده های نامتعادل قرار می گیرد و باید در طول آزمایش به داده های ناشناخته و نابرابر تعمیم یابد.

این یک ارزیابی تمیزتر از توانایی مدل برای تعمیم داده های جدید را ارائه می دهد ، زیرا مجموعه آزمون نماینده توزیع واقعی داده های اصلی است.

df\_balanced = pd.DataFrame(data=np.column\_stack((X\_balanced, y\_balanced)), columns=df.columns)

df\_balanced

df\_balanced.hist(bins=30,figsize=(30,30))



شکل 11 نمودارتوزیع کلاس ها

در اينجا اخرين كلاس ميبينيم كه برچسب target توازن دارند.

نمونه برداري و تقسيم بندي:

from sklearn.model\_selection import train\_test\_split

X\_balanced\_reshaped = X\_balanced.reshape(X\_balanced.shape[0], X\_balanced.shape[1], 1)

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X\_balanced\_reshaped, y\_balanced, test\_size=0.4, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

print("Training set shape:", X\_train.shape)

print("Validation set shape:", X\_val.shape)

print("Testing set shape:", X\_test.shape)

Training set shape: (340919, 30, 1)

Validation set shape: (113640, 30, 1)

Testing set shape: (113640, 30, 1)

## ۳-۴. آموزش مدل

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, BatchNormalization, MaxPool1D, Dropout, Flatten, Dense

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.losses import BinaryCrossentropy

from tensorflow.keras.metrics import Accuracy

model = Sequential()

model.add(Conv1D(filters=32, kernel\_size=2, activation='relu', input\_shape=(X\_train.shape[1], 1)))

model.add(BatchNormalization())

model.add(MaxPool1D(pool\_size=2))

model.add(Dropout(rate=0.2))

model.add(Conv1D(filters=64, kernel\_size=2, activation='relu'))

model.add(BatchNormalization())

model.add(MaxPool1D(pool\_size=2))

model.add(Dropout(rate=0.5))

model.add(Flatten())

model.add(Dense(units=64, activation='relu'))

model.add(Dropout(rate=0.5))

model.add(Dense(units=64, activation='relu'))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer=Adam(learning\_rate=0.0001),

              loss=BinaryCrossentropy(),

              metrics=[Accuracy()])

model.summary()

Model: "sequential\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv1d\_5 (Conv1D) (None, 29, 32) 96

batch\_normalization\_4 (Bat (None, 29, 32) 128

chNormalization)

max\_pooling1d\_4 (MaxPoolin (None, 14, 32) 0

g1D)

dropout\_6 (Dropout) (None, 14, 32) 0

conv1d\_6 (Conv1D) (None, 13, 64) 4160

batch\_normalization\_5 (Bat (None, 13, 64) 256

chNormalization)

max\_pooling1d\_5 (MaxPoolin (None, 6, 64) 0

g1D)

dropout\_7 (Dropout) (None, 6, 64) 0

flatten\_2 (Flatten) (None, 384) 0

...

Total params: 33505 (130.88 KB)

Trainable params: 33313 (130.13 KB)

Non-trainable params: 192 (768.00 Byte)

history = model.fit(X\_train, y\_train, epochs=6, batch\_size=32, validation\_data=(X\_val, y\_val))

Epoch 1/6

10654/10654 [==============================] - 81s 8ms/step - loss: 0.3358 - accuracy: 0.0755 - val\_loss: 0.2760 - val\_accuracy: 0.1125

Epoch 2/6

10654/10654 [==============================] - 80s 7ms/step - loss: 0.2994 - accuracy: 0.1120 - val\_loss: 0.2601 - val\_accuracy: 0.1262

Epoch 3/6

10654/10654 [==============================] - 78s 7ms/step - loss: 0.2838 - accuracy: 0.1293 - val\_loss: 0.2501 - val\_accuracy: 0.1406

Epoch 4/6

10654/10654 [==============================] - 79s 7ms/step - loss: 0.2754 - accuracy: 0.1300 - val\_loss: 0.2392 - val\_accuracy: 0.1291

Epoch 5/6

10654/10654 [==============================] - 76s 7ms/step - loss: 0.2655 - accuracy: 0.1377 - val\_loss: 0.2376 - val\_accuracy: 0.1198

Epoch 6/6

10654/10654 [==============================] - 80s 7ms/step - loss: 0.2613 - accuracy: 0.1306 - val\_loss: 0.2397 - val\_accuracy: 0.0905

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

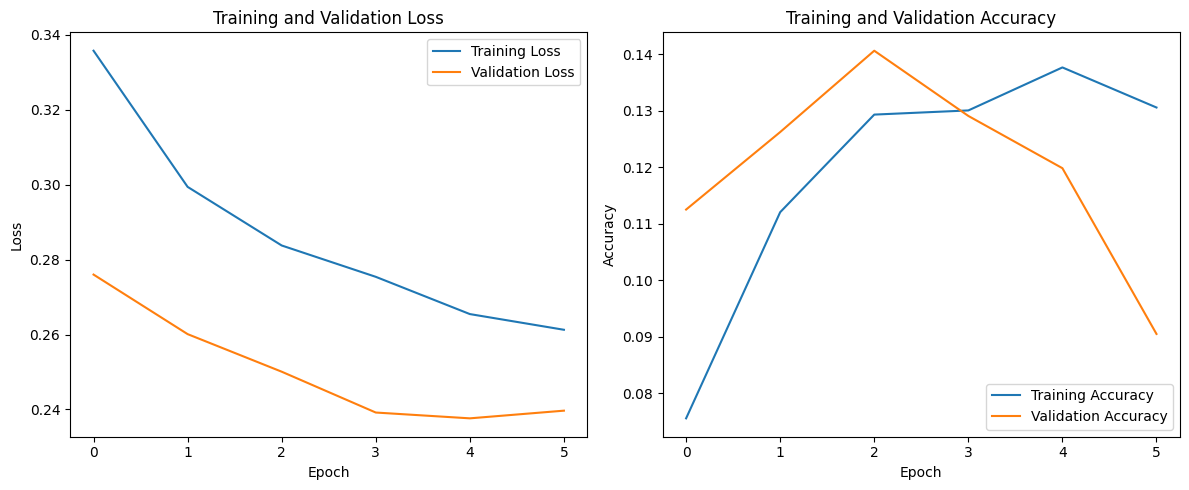
plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()



شکل 12 نمودار دقت و خطا

from sklearn.metrics import confusion\_matrix, classification\_report

import seaborn as sns

import matplotlib.pyplot as plt

*# Obtain model predictions for the test data*

y\_pred = model.predict(X\_test)

y\_pred\_binary = (y\_pred > 0.5).astype(int)  *# Convert to binary predictions (0 or 1)*

*# Calculate and plot the confusion matrix heatmap*

cm = confusion\_matrix(y\_test, y\_pred\_binary)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,

            xticklabels=['Legitimate', 'Fraud'],

            yticklabels=['Legitimate', 'Fraud'])

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

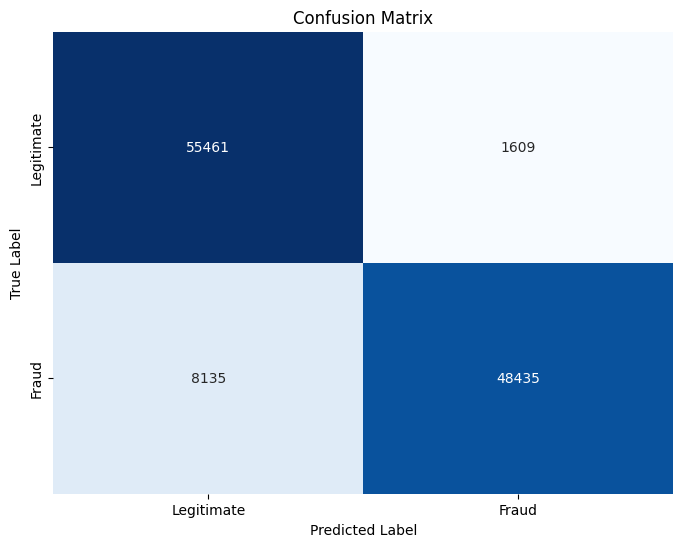
plt.show()

*# Report precision, recall, and F1 score*

report = classification\_report(y\_test, y\_pred\_binary, target\_names=['Legitimate', 'Fraud'])

print("Classification Report:")

print(report)



شکل 13 ماتریس درهم ریختگی

Classification Report:

precision recall f1-score support

Legitimate 0.87 0.97 0.92 57070

Fraud 0.97 0.86 0.91 56570

accuracy 0.91 113640

macro avg 0.92 0.91 0.91 113640

weighted avg 0.92 0.91 0.91 113640

در اين حالت نتايج دريافت شده قابل اتكا بوده و به دقت 91 درصد مي‌رسيم.