Ministerio de Desarrollo Productivo Argentina

Secretaría de Economía del Conocimiento

Argentina programa 4.0

TRABAJO PRÁCTICO INTEGRADOR

Argentina Programa 4.0

Curso: Programador Junior en Machine Learning – Módulo III **Duración** del curso: 6 semanas

Instructores:

- VILLAR, Santiago – FCEQyN, UNaM
- ZARSKE, Arnold – FCEQyN, UNaM

Estudiante: Arturo Marin Bosquet

Correo electrónico: Juan.Perez@gmail.com

Fecha de entrega: 28/10/2023

Introducción

El presente informe detalla el proceso de desarrollo de una red neuronal para analizar un conjunto de datos. Se aborda la creación de un modelo de aprendizaje automático, la identificación del desbalance de clases, la aplicación de técnicas para abordar este desbalance y el análisis de la relación entre las variables mediante matrices de correlación. El estudio se enfoca en el contexto de un conjunto de datos desbalanceado y aborda técnicas para mitigar este desbalance.

Caso de estudio

En este caso de estudio, asumimos que trabajamos como consultores de IA y hemos sido contratados por una importante empresa bancaria, donde es de vital importancia que su sistema sea capaces de detectar de manera efectiva las transacciones fraudulentas con tarjetas de crédito para garantizar que los clientes no sean cobrados por compras que no realizaron.

Se centra en un conjunto de datos con desbalance de clases, específicamente un problema de clasificación binaria. Se utiliza una red neuronal para predecir la variable de salida (clase) en función de múltiples características. El objetivo principal es implementar una solución que ofrezca predicciones precisas, a pesar del desbalance en las clases de salida.

Desarrollo del proyecto

Para la realización de este proyecto se usó como IDE, visual studio code y con ayuda de foros, PDFs, libros y más para poder desarrollar este proyecto.

Lo primero que hacemos es importar toda la biblioteca que debemos usar

```
#librerias
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow import keras
from keras.layers import Dense
from keras.models import Sequential
```

Luego se carga el archivo CSV de forma local

```
# Ruta local del archivo CSV en la carpeta "Integrador"
local_file_path = r'C:\Users\Arturo Marin\Desktop\INTEGRADOR\creditcard.csv'

# Carga los datos desde el archivo local
data = pd.read_csv(local_file_path)
```

Desarrollo de la Red Neuronal:

Se definió la siguiente estructura de la red neuronal

Carga de datos y procesamiento:

```
# Carga los datos desde el archivo local
data = pd.read_csv(local_file_path)

# separacion de caracteristicas (x) y etiquetas (y)
x = data.drop('Class', axis=1)
y = data['Class']

# division de los datos en conjuntos de entrenamiento y prueba
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# escalar las caracteristicas para normalizar

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Construcción del modelo neuronal:

```python
model = Sequential ([
    Dense(64, activation='relu', input_shape=(x_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

# compilacion del modelo

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Entrenamiento del modelo:

```python
# entrenamiento del modelo
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
```

```
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate
Epoch 1/10
7121/7121 [==============================] - 13s 2ms/step - loss: 0.0060 - accuracy: 0.9991 - val_loss: 0.0027 - val_accuracy: 0.9994
Epoch 2/10
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0029 - accuracy: 0.9994 - val_loss: 0.0030 - val_accuracy: 0.9994
Epoch 3/10
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0026 - accuracy: 0.9994 - val_loss: 0.0029 - val_accuracy: 0.9994
Epoch 4/10
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0024 - accuracy: 0.9995 - val_loss: 0.0029 - val_accuracy: 0.9994
Epoch 5/10
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0022 - accuracy: 0.9995 - val_loss: 0.0029 - val_accuracy: 0.9995
Epoch 6/10
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0021 - accuracy: 0.9995 - val_loss: 0.0031 - val_accuracy: 0.9995
Epoch 7/10
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0021 - accuracy: 0.9996 - val_loss: 0.0032 - val_accuracy: 0.9995
Epoch 8/10
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0019 - accuracy: 0.9996 - val_loss: 0.0032 - val_accuracy: 0.9994
Epoch 9/10
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0018 - accuracy: 0.9996 - val_loss: 0.0032 - val_accuracy: 0.9995
Epoch 10/10
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0018 - accuracy: 0.9996 - val_loss: 0.0034 - val_accuracy: 0.9995
1781/1781 [==============================] - 2s 967us/step
```

Aplicación y Explicación de la Matriz de Confusión:

Para evaluar el modelo de una mejor manera, calculamos la matriz de confusión obteniendo los siguientes resultados:

```python
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5) # convertir las probabilidades en etiquetas binarias

# generar la matriz de confusion

conf_matrix = confusion_matrix(y_test, y_pred)

# mostrar la matriz de confusion

print("Matriz de Confusion: ")
print(conf_matrix)
```

Resultados de la matriz de confusión:

```
Matriz de Confusion:
[[56860     4]
 [   27    71]]
```

A partir de dichos resultados podemos decir que el modelo tiene una cantidad de datos de transacciones legítimas y pocas fraudulentas, pero también hubieron falsos negativos**(27)** y falsos positivos**(4).**

Elección y Aplicación de Métricas de Evaluación:

Para tener una mejor evaluación escogimos utilizar la métrica de **Precisión** ya que esta es la que mejor encaja en este escenario dado que al haber pocos datos de fraude hay más margen de error.

```python
from sklearn.metrics import accuracy_score

precision = accuracy_score(y_test, y_pred)

print(f"Precision del modelo: {precision}")
```

Precisión obtenida del modelo:

```
Precision del modelo: 0.9994557775359011
```

Análisis del Desbalance de Clases:

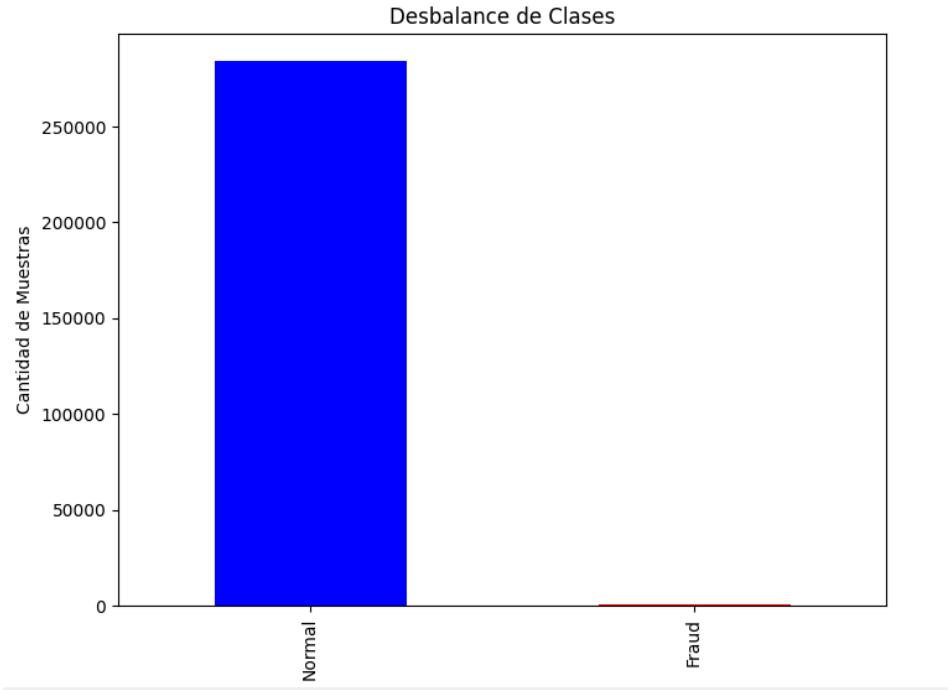
```python
#Visualizar el desbalanceo del Dataset
import matplotlib.pyplot as plt

# Calcular la cantidad de muestras por clase
class_counts = data['Class'].value_counts()

# Mostrar la cantidad de datos por clase
for class_name, count in class_counts.items():
    print(f'Clase: {class_name}, Cantidad de Datos: {count}')

# Crear el gráfico de barras
plt.figure(figsize=(8, 6))
class_counts.plot(kind='bar', color=['blue', 'red'])
plt.title('Desbalance de Clases')
plt.xlabel('Clase')
plt.ylabel('Cantidad de Muestras')
plt.xticks([0, 1], ['Normal', 'Fraud'])
plt.show()

#Que se observan en los datos visualizados?
```



Si analizamos los datos podemos ver que existe un desbalance entre las clases, por lo que tendremos que balancearlas.

```python
# Calcular la distribución de clases actual
counter = Counter(y)
print("Distribución de clases antes del submuestreo:", counter)

# Identificar la cantidad mínima de muestras entre las clases
clase_menor = min(counter, key=counter.get)
cantidad_menor = counter[clase_menor]

# Realizar el submuestreo para igualar la cantidad de muestras de la clase minoritaria
clases = list(counter.keys())
for clase in clases:
    indices_clase = np.where(y == clase)[0]
    indices_muestra_a_descartar = indices_clase[cantidad_menor:]
    x_submuestreo = np.delete(x, indices_muestra_a_descartar, axis=0)
    y_submuestreo = np.delete(y, indices_muestra_a_descartar)

# Verificar la distribución de clases después del submuestreo
counter_submuestreo = Counter(y_submuestreo)
print("Distribución de clases después del submuestreo:", counter_submuestreo)
```

En este caso decidimos utilizar el método **undersampling** para balancear las clases, y luego si volvemos a entrenar el modelo con nuestro dataset balanceado, al volver a evaluar las métricas anteriores obtenemos que las clases ya se encuentran balanceadas.

```
Distribución de clases antes del submuestreo: Counter({0: 284315, 1: 492})
Distribución de clases después del submuestreo: Counter({0: 284315, 1: 492})
```

Mapa de Calor y Matriz de Correlaciones:

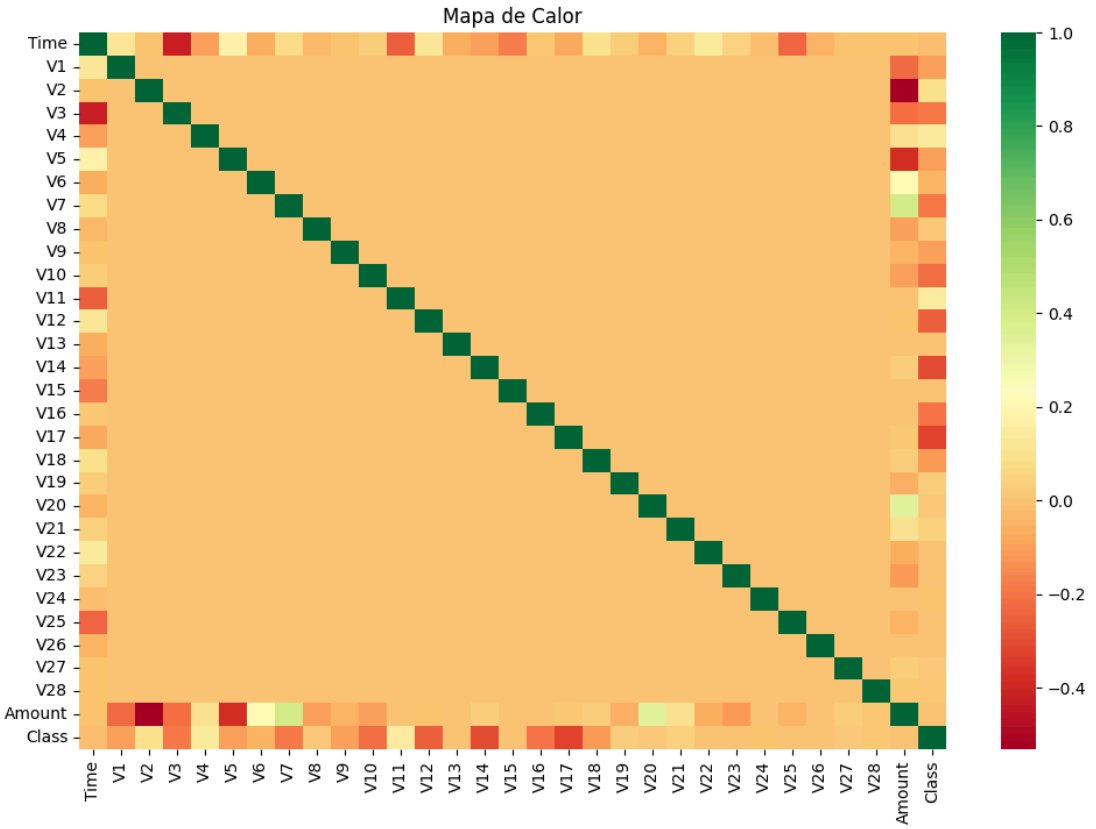
Para obtener el mapa de calor y la matriz de correlaciones se ejecutó el siguiente

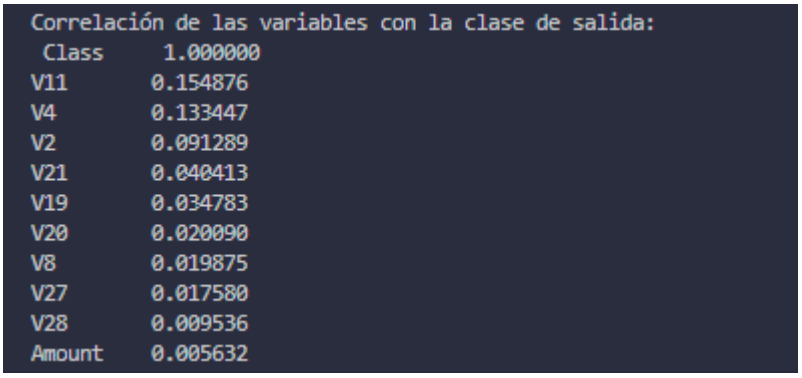
código:

```python
# Calcular la matriz de correlación
correlation_matrix = data.corr()

# Generar un mapa de calor
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, cmap="RdYlGn")
plt.title('Mapa de Calor')
plt.show()
```

A continuación se muestra el mapa de calor y la matriz de correlaciones obtenida de los datos del escenario:





Se puede observar que la variable con mayor correlación con la clase de salida es:

V11 -> 0.154876

Conclusión

El análisis y desarrollo de la red neuronal para el problema de clasificación binaria con desbalance de clases ha demostrado ser un proceso crucial. La identificación y abordaje del desbalance de clases son pasos esenciales para mejorar la capacidad predictiva del modelo. La aplicación de submuestreo permitió equilibrar las clases, aunque puede conllevar a la pérdida de información. Además, el análisis de correlaciones reveló las relaciones entre las variables, destacando las que tienen mayor influencia en la predicción de la variable de salida.

En conclusión se puede decir que todas estas técnicas son importantes para cualquier problema que se nos presente en la vida cotidiana. Esto nos ha ayudado a resolver problemas fácilmente, sino también el manejo efectivo del desbalance de clases para evitar sesgos en los resultados. El presente estudio se sumerge en un análisis detallado del desarrollo de una red neuronal, destacando la importancia de abordar este desbalance, comprender las relaciones entre variables y aplicar estrategias que mejoren la capacidad predictiva del modelo.