



Ministerio de
Desarrollo Productivo
Argentina

Secretaría de Economía del
Conocimiento



Argentina
programa
4.0

GUÍA PARA EL 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: Fabian Villada

Correo electrónico: fabianvillada6@gmail.com

Fecha de entrega: 26/10/2023

Introducción

Utilizamos una metodología de aprendizaje automático y redes neuronales con oversampling para abordar el problema de clasificación de transacciones fraudulentas en un conjunto de datos de tarjetas de crédito.

El enfoque busca mejorar la capacidad del modelo para detectar transacciones fraudulentas al equilibrar las clases y entrenar una red neuronal para aprender patrones y características distintivas de esta clase de transacciones.

Caso de estudio

Detección de fraudes con tarjetas de crédito

Transacciones anónimas con tarjeta de crédito, etiquetadas como fraudulentas o auténticas

Es importante que las empresas de tarjetas de crédito sean capaces de reconocer las transacciones fraudulentas con tarjeta de crédito para que no se carguen a los clientes artículos que no compraron. El conjunto de datos contiene transacciones realizadas con tarjetas de crédito en septiembre de 2013 por titulares de tarjetas europeos. Este conjunto de datos presenta transacciones que ocurrieron en dos días, donde tenemos 492 fraudes de 284.807 transacciones. El conjunto de datos está muy desequilibrado, la clase positiva (fraudes) representa el 0,172% de todas las transacciones.

Sólo contiene variables de entrada numéricas que son el resultado de una transformación PCA (El PCA es una técnica estadística para reducir la dimensionalidad de un conjunto de datos.).

Objetivo: Dado que estamos tratando con un conjunto de datos altamente desequilibrado, donde las transacciones fraudulentas representan una pequeña fracción de todas las transacciones, y se busca minimizar los falsos negativos, es decir, la capacidad para atrapar la mayoría de los fraudes, sin dejar muchos sin detectar.

Desarrollo del proyecto

Para la realización de este proyecto se hizo uso de una Red Neuronal.

Desarrollo de la Red Neuronal:

Se definió la siguiente estructura de la red neuronal

```
#creacion de la red neuronal y entrenamiento

model = Sequential()
model.add(Dense(units=128, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
```

La red neuronal descrita en el código consiste en un modelo secuencial construido utilizando la librería Keras.

Descripción de cada capa de la red:

1) *Capa Dense(units=128, activation='relu', input_dim=X_train.shape[1]):*

Esta es la primera capa de la red. Es una capa densamente conectada o totalmente conectada, lo que significa que cada neurona en esta capa se conecta a todas las neuronas de la capa anterior. La capa consta de 128 neuronas y Es una capa densamente conectada o totalmente conectada, lo que significa que cada neurona en esta capa se conecta a todas las neuronas de la capa anterior.

input_dim=X_train.shape[1] indica que el número de neuronas de entrada en esta capa es igual a la dimensión del conjunto de datos de entrenamiento X_train.

2) *Capa Dense(units=64, activation='relu'):*

Esta es la segunda capa de la red. También es una capa densamente conectada pero de 64 neuronas. Utiliza la función de activación ReLU.

3) *Capa Dense(units=1, activation='sigmoid'):*

Esta es la capa de salida de la red. Es una capa densamente conectada de una sola neurona que representa la salida del modelo.

Utiliza la función de activación sigmoide (sigmoid), que es comúnmente utilizada en problemas de clasificación binaria para obtener una salida entre 0 y 1, que puede interpretarse como la probabilidad de pertenecer a una clase.

Dividir el conjunto de datos en características (X) y etiquetas (y)

En el análisis de machine learning, es común dividir un conjunto de datos en características (X) y etiquetas (y) antes de aplicar algoritmos de aprendizaje automático. Esta división se realiza para separar las variables que se utilizarán como entradas para el modelo (características) de la variable objetivo que se pretende predecir (etiquetas). En este caso la variable a predecir es la variable Class correspondiente al dataSet "creditcard.csv".

```
# Divide los datos en características (X) y etiquetas (y)
data = pd.read_csv(output)
X = data.drop('Class', axis=1)
y = data['Class']
```

Particionamiento del conjunto de datos en entrenamiento y prueba

En esta técnica, el conjunto de datos se divide en dos partes: un **conjunto de entrenamiento** y un **conjunto de prueba**. El modelo se entrena con el conjunto de entrenamiento y se evalúa con el conjunto de prueba. Esta técnica es simple y rápida de implementar, pero puede ser sensible a la distribución de los datos y a la selección aleatoria de las muestras.

```
# Divide 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)
```

Escalamiento

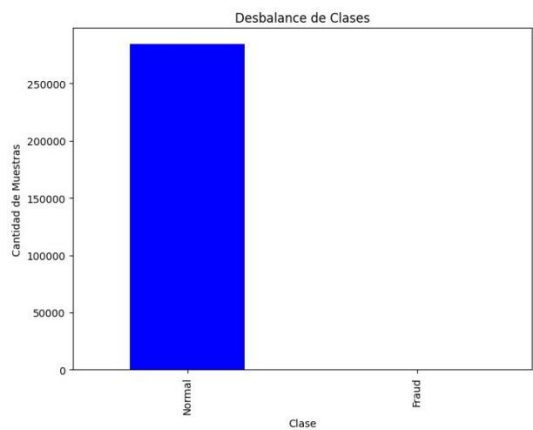
El escalado es una técnica utilizada para llevar todas las características del dataSet a la misma escala. Básicamente da una media = 0 y Varianza = 1 para todas las características.

```
# Escala las características para normalizar
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Sobremuestreo

Las clases desbalanceadas son situaciones en las que tenemos una gran disparidad en la cantidad de ejemplos que pertenecen a diferentes clases, es decir tenemos un número significativamente mayor de ejemplos en una clase en comparación con otra.

En nuestro data set vemos que las transacciones válidas class = 0 son significativamente mayores que las no válidas class = 1



En general, el desequilibrio de clases puede tener un impacto significativo en el proceso de generalización de un modelo de Machine Learning, es decir, en su capacidad para aplicar el conocimiento aprendido a casos nuevos y no vistos anteriormente. Este desequilibrio suele perjudicar el rendimiento de las clases minoritarias, ya que el modelo tiende a tener dificultades para capturar adecuadamente la información y los patrones asociados a estas clases.

Para contrarrestar este inconveniente existen varias técnicas entre ellas: compensación por penalización, submuestreo, sobremuestreo, resampling con smote_tomek etc....

En este caso aplicamos sobremuestreo ya que es la técnica que más reduce los Falsos Negativos, el cual es nuestro objetivo. Que el modelo prediga certeramente la mayor cantidad de casos de fraude.

La técnica de sobremuestreo (oversample) agrega muestras aleatoriamente de la clase minoritaria para mitigar el desbalanceo de datos

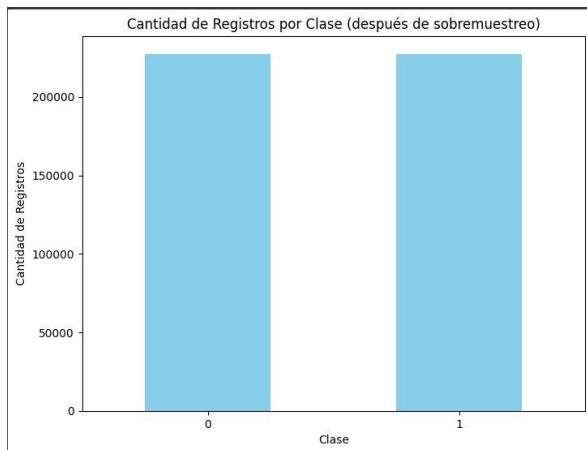
Aplicamos el siguiente código:

```
# Aplicar oversampling utilizando RandomOverSampler

from imblearn.over_sampling import RandomOverSampler

oversampler = RandomOverSampler(random_state=42)
X_train_resampled, y_train_resampled = oversampler.fit_resample(X_train, y_train)
```

Obteniendo el siguiente **set con el cual entrenaremos el modelo** :



Entrenamos y vemos los resultados

En el contexto del entrenamiento de una red neuronal, una época (epoch) se refiere a una pasada completa de todos los ejemplos de entrenamiento a través de la red. **Durante cada época, los pesos de la red se ajustan iterativamente para minimizar una función de pérdida (loss function)** que mide la discrepancia entre las salidas predichas por la red y las salidas reales.

En la siguiente imagen se muestran los resultados del entrenamiento.

Vemos el progreso del entrenamiento de la red neuronal durante 10 épocas. Cada línea representa una época específica, y los números indican la cantidad de muestras procesadas por segundo (steps/second), así como la pérdida (loss) calculada para esa época.

```
Epoch 1/10
14216/14216 [=====] - 31s 2ms/step - loss: 0.0147
Epoch 2/10
14216/14216 [=====] - 29s 2ms/step - loss: 0.0041
Epoch 3/10
14216/14216 [=====] - 29s 2ms/step - loss: 0.0031
Epoch 4/10
14216/14216 [=====] - 28s 2ms/step - loss: 0.0026
Epoch 5/10
14216/14216 [=====] - 28s 2ms/step - loss: 0.0020
Epoch 6/10
14216/14216 [=====] - 29s 2ms/step - loss: 0.0017
Epoch 7/10
14216/14216 [=====] - 28s 2ms/step - loss: 0.0018
Epoch 8/10
14216/14216 [=====] - 28s 2ms/step - loss: 0.0016
Epoch 9/10
14216/14216 [=====] - 28s 2ms/step - loss: 0.0017
Epoch 10/10
14216/14216 [=====] - 29s 2ms/step - loss: 0.0015
1781/1781 [=====] - 2s 1ms/step
```

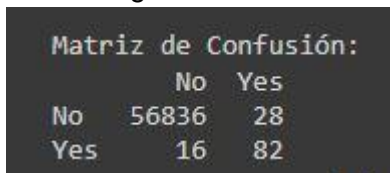
[...]

Matriz de Confusión

La matriz de confusión es una herramienta fundamental en la evaluación de modelos de clasificación en Machine Learning. Proporciona una visión detallada del rendimiento del modelo al mostrar cómo se clasificaron las instancias de prueba en función de las

predicciones del modelo. La matriz de confusión es especialmente útil para comprender el comportamiento del modelo en términos de aciertos y errores en diferentes categorías de clasificación.

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

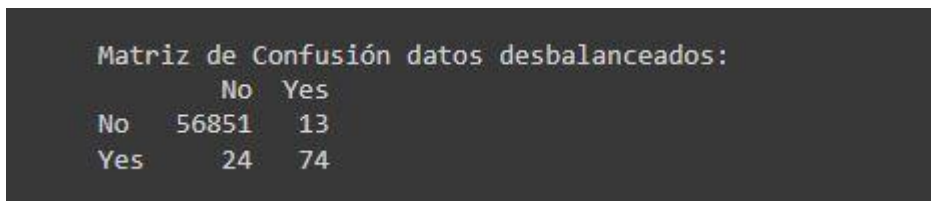


```
Matriz de Confusión:
      No  Yes
No  56836  28
Yes   16  82
```

Donde **las filas son las etiquetas reales** y **las columnas las predicciones**.

El modelo tuvo en la predicción 16 Falsos Negativos, es decir predijo 16 transacciones como válidas que en realidad son fraudulentas.

Para comparar mostramos la matriz de confusión entrenando el modelo con datos desbalanceados.



```
Matriz de Confusión datos desbalanceados:
      No  Yes
No  56851  13
Yes   24  74
```

A partir de dichos resultados y comparando las matrices de confusión, podemos ver que al realizar la predicción usando el modelo entrenado con balanceo de datos con técnica de submuestreo **bajamos considerablemente los Falsos Negativos** de 24 casos a 16 casos.

Elección y Aplicación de Métricas de Evaluación

Para tener una mejor evaluación escogimos utilizar la métrica recall, ya que esta es la que mejor encaja en este escenario dado que tiene en cuenta cómo influyen en el modelo los falsos negativos. También tenemos en cuenta el F1 Score para que la técnica que apliquemos no termine afectando en demasía la precisión.

Se muestran las métricas de las predicciones realizadas con modelo entrenado con datos desbalanceados y modelos entrenados con datos balanceados (submuestreo y sobremuestreo respectivamente), para poder comparar:

Metricas datos desbalanceados				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.85	0.76	0.80	98
accuracy			1.00	56962
macro avg	0.93	0.88	0.90	56962
weighted avg	1.00	1.00	1.00	56962

Metricas Habiendo aplicado submuestreo				
	precision	recall	f1-score	support
0	1.00	0.97	0.99	56864
1	0.05	0.92	0.10	98
accuracy			0.97	56962
macro avg	0.53	0.95	0.54	56962
weighted avg	1.00	0.97	0.98	56962

Metricas Habiendo aplicado sobremuestreo				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.67	0.84	0.75	98
accuracy			1.00	56962
macro avg	0.84	0.92	0.87	56962
weighted avg	1.00	1.00	1.00	56962

ANALISIS:

Establecimos que nuestra métrica representativa es “recall” (sensibilidad) y que así también evaluamos que el valor de f1 score no baje demasiado lo que indicaría que la precisión fue excesivamente afectada.

Observando las imágenes vemos que el modelo que mejor se adapta al escenario establecido es el modelo entrenado con datos Sobremuestreados.

Mapa de Calor y Matriz de Correlaciones

En el campo del Machine Learning, los **mapas de calor** son una herramienta visual poderosa para analizar y comprender los datos. Estos mapas proporcionan una representación gráfica de la relación entre variables en un conjunto de datos y nos permiten identificar patrones, tendencias y correlaciones.

La **matriz de correlaciones** es otra herramienta fundamental en el análisis de datos y el Machine Learning. Esta matriz nos permite calcular y visualizar las correlaciones entre pares de variables en un conjunto de datos. En una matriz de correlaciones, cada celda representa el coeficiente de correlación entre dos variables. El coeficiente de correlación es un valor numérico que indica la fuerza y la dirección de la relación entre dos variables. Puede variar entre -1 y 1, donde -1 indica una correlación negativa perfecta, 1 indica una correlación positiva perfecta y 0 indica que no hay correlación entre las variables.

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

```
import seaborn as sns
import matplotlib.pyplot as plt

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

# Configurar el tamaño del mapa de calor
plt.figure(figsize=(25, 13)) # Ajusta el tamaño de la figura

# Crear el mapa de calor de correlación
sns.heatmap(correlation_matrix, annot=True, cmap="RdYlGn", annot_kws={"size": 8}) # Ajusta el tamaño del texto

# Mostrar el mapa de calor
plt.show()
```

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

