ITESM- Campus Puebla

American Express - Default Prediction

Inteligencia artificial avanzada para la ciencia Datos

Reporte número 2

Integrantes Equipo 1:

Myroslava Sánchez Andrade A01730712
José Antonio Bobadilla García A01734433
Karen Rugerio Armenta A01733228
Alejandro Castro Reus A01731065

Fecha: 09/09/2022

I. Reto de limpieza de conjunto de datos

1 Introducción

Las empresas bancarias tienen un peso muy grande en la economía global. Tan sólo en México, se considera que al menos 41.1 millones de Mexicanos, que representa a un 49.1% del total de adultos de 18 a 70 años, tienen al menos una cuenta bancaria, según el Instituto Nacional de Estadística y Geografía (Inegi, 11 de mayo de 2022).

Los bancos desempeñan un papel importante, ya que se encargan de permitir el flujo de recursos financieros en todo el país, distribuir efectivo, pagar los cheques que se emiten, ofrecer servicio con tarjetas y transferencias bancarias, otorgar préstamos, entre otros.

Saber predecir si una persona va a pagar el crédito que se le otorga es una parte esencial del negocio. Es por ello, que este proyecto trató de predecir, haciendo uso de un modelo de Machine Learning y entrenando un algoritmo con base en un historial de usuarios, si un usuario pagará el crédito de la tarjeta a la que aplicó.

1.1 American express - Default Prediction

Desarrollo

- Business Understanding

El enfoque de este proyecto es en los préstamos bancarios que los bancos otorgan mediante tarjetas de crédito. El Portal de Educación Financiera, define el crédito como un préstamo de dinero que una parte otorga a otra, con el compromiso de que, en el futuro, quien lo recibe devolverá dicho préstamo en forma gradual (mediante el pago de cuotas) o en un solo pago y con un interés adicional que compensa a quien presta, por todo el tiempo que no tuvo ese dinero.

Para poder tener acceso a una tarjeta de crédito con American Express, se debe tener cierta seguridad de que la persona que está aplicando al crédito tiene los recursos para pagar los montos acordados conforme a las tasas de interés.

- Data Understanding

Para entender los datos del reto a solucionar, se procedió a realizar una investigación del reto en la página oficial de Kaggle. Lo que se encontró es que hay tres archivos de tipo CSV, dos

con la información dividida por clientes y estos a su vez divididos por periodos (uno es el train y otro es el test) y otro con los datos de si el cliente pagó o no.

Las variables independientes se nos presentaron divididas en 5 categorías:

- D_* = Variables de Delincuencia
- S_* = Variables de Gasto
- P_* = Variables de Pago
- B_* = Variables de Balance
- R_* = Variables de Riesgo

Las siguientes variables eran categóricas:

'B_30', 'B_38', 'D_114', 'D_116', 'D_117', 'D_120', 'D_126', 'D_63', 'D_64', 'D_66', 'D_68'

La variable independiente era únicamente un valor binario en el que el 1 significa que el cliente no pagó.

Las variables habían sido anonimizadas por lo que no se pudo identificar qué representan. En el contexto de nuestro reto no tiene importancia alguna que supiéramos el significado de cada una de estas, ya que el modelo haría lo necesario para entender los datos y sacar una predicción.

● Data Preparation

Como primer acercamiento a la solución de esta primera fase del reto, se intentó trabajar con todos los datos, haciendo uso de la librería de Python llamada Pandas, sin embargo, no fue posible procesar toda la información, ya que el set de datos tiene un peso de 16gb , con total de poco más de 5 millones de filas. Debido a la magnitud de este dataset se decidió trabajar únicamente un 20%.

El archivo pudo ser leído y procesado del dataset completo en la computadora de un integrante del equipo, que cuenta con un procesador Arm M1 de Apple, Intel Core i5-10300H, 4 procesadores principales y 8 procesadores lógicos. Esto fue un contratiempo ya que se tuvo que organizar de una mejor manera el equipo para avanzar con la limpieza de los datos como primer aproach y posteriormente investigar alternativas que ayudaran a analizar el dataset completo en cada una de las computadoras de los miembros del equipo, teniendo en cuenta el hardware de cada integrante.

- Los datos de train y test fueron extraídos en formato CSV desde Kaggle. Dado el tamaño del dataset y la naturaleza de carga completa de este formato, se tomó la decisión de guardar parte de la información de entrenamiento en formato *parquet*, particionando adicionalmente a través de compresión *snappy.* Posteriormente los

datos fueron cargados en un script de python como dataframe de dask que nos permite dividir los datos para procesarlos de manera paralela.

```
# Get data
train_data = dd.read_parquet('train_data/snappy-parquet', engine='pyarrow')
train_labels = dd.read_csv('train_labels.csv')
# train_data = train_data.compute()
```

```
new_train_data.to_parquet("train_data/filtered-columns", engine="pyarrow", compression="snappy")
```

- Tras verificar el contenido y estructura de las columnas se pudo verificar que ya estaban estandarizadas y se procedió a realizar el cálculo de las correlaciones, de esta forma se pueden encontrar las variables independientes que mejor explican la variables dependientes.

```
correlation = train_data.corr().compute()
correlation
```

	P_2	D_39	B_1	B_2	R_1	S_3	D_41	B_3	D_42	D_43	...	D_136	D_137	D_138	D_139	D_140	
P_2	1.000000	-0.191551	-0.368003	0.545083	-0.464548	-0.334633	-0.317395	-0.454154	-0.468156	-0.299000	...	-0.054096	-0.000098	-0.005665	-0.184728	-0.138198	-0.1
D_39	-0.191551	1.000000	0.183356	-0.194832	0.194509	0.052350	0.440503	0.173106	0.071271	0.046419	...	-0.018367	-0.014772	0.018504	0.032060	0.022463	0.0
B_1	-0.368003	0.183356	1.000000	-0.631075	0.229492	0.155834	0.185095	0.729890	-0.054717	0.071996	...	-0.033516	0.004573	-0.014813	0.103833	0.054776	0.1
B_2	0.545083	-0.194832	-0.631075	1.000000	-0.303907	-0.235561	-0.257901	-0.713539	-0.090105	-0.142679	...	0.013797	0.014416	0.023164	-0.154482	-0.094415	-0.1
R_1	-0.464548	0.194509	0.229492	-0.303907	1.000000	0.212158	0.275890	0.272562	0.175894	0.187207	...	-0.001821	0.001757	0.034417	0.082094	0.066854	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
D_141	-0.180141	0.032016	0.103099	-0.152619	0.080836	0.046756	0.049332	0.131708	0.023146	0.069027	...	0.083115	-0.014723	-0.102947	0.998168	0.324814	1.0
D_142	0.106109	0.008826	0.002896	0.019708	-0.008253	-0.076683	0.002999	0.008902	-0.058363	-0.050937	...	-0.038171	-0.003379	0.026285	-0.000369	-0.038292	0.7
D_143	-0.184773	0.032051	0.103864	-0.154545	0.082147	0.049118	0.049851	0.132831	0.025598	0.071392	...	0.085240	-0.014724	-0.104664	0.999498	0.325544	0.9
D_144	0.006733	0.009343	0.036570	-0.039635	0.004660	-0.019125	0.010392	0.041107	-0.056738	-0.008098	...	0.011069	-0.008651	-0.039596	0.569267	-0.038592	0.5
D_145	-0.165209	0.024202	0.078757	-0.124659	0.073060	0.045380	0.042079	0.107736	0.035377	0.066420	...	0.074369	-0.013157	-0.080764	0.645738	0.214737	0.6

Por cada conjunto de columnas cuya correlación es superior a 0.9, se conservó una sola columna de acuerdo a la justificación incorporada en el código.

```
for column in correlation.columns:
    local = correlation[column][correlation[column].abs() > 0.9]
    local = local[local.index != column]
    if local.count() > 0:
        print(local)
        print()
```

```
B_11    0.995574
B_37    0.992915
Name: B_1, dtype: float64

B_33    0.912814
Name: B_2, dtype: float64

S_7    0.904635
Name: S_3, dtype: float64

B_23    0.995051
Name: B_7, dtype: float64

B_1     0.995574
B_37    0.987941
Name: B_11, dtype: float64

S_3    0.904635
Name: S_7, dtype: float64
```

```
# B_1 -> B_11, B_37, elegimos porque tenía mayor correlación y no tenía nulos
# B_33 -> B_2, elegimos porque la distribución era ligeramente más compacta
# S_7 -> S_3, porque tiene menos valores extremos
# B_23 -> B_7, menos valores extremos
# D_58 -> D_74, D_75, pues tiene una distribución ligeramente más uniforme
# B_14 -> B_15, no tiene tantos valores faltantes
# D_62 -> D_77, no tiene tantos valores faltantes
# S_24 -> S_22, no tiene tantos valores faltantes y tiene una distribución ligeramente más compacta
# D_103 -> D_104, la distribución es más compacta
# D_118 -> D_119, misma distribución
# D_139 -> D_141, D_143, mejor correlación con ambas
drop_columns = ['B_11', 'B_37', 'B_2', 'S_3', 'B_7', 'D_74', 'D_75', 'B_15', 'D_77', 'S_22', 'D_104', 'D_119', 'D_141', 'D_143']
```

- Posteriormente, como parte de la limpieza de los datos, se realizó un snippet para calcular los valores nulos contenidos en las columnas, con la finalidad de sustraer las columnas que cuentan con un porcentaje de valores nulos mayor al 30%.

```
max_val = train_data['customer_ID'].count().compute()
count_non_null = train_data.count().compute()
count_non_null
```

```
customer_ID    5531451
S_2            5531451
P_2            5485466
D_39           5531451
B_1            5531451
                ...
D_141          5429903
D_142           944408
D_143          5429903
D_144          5490724
D_145          5429903
Length: 190, dtype: int64
```

```
drop_columns += count_non_null[count_non_null < max_val * 0.7].index.to_list()
```

● Modelado

Una vez finalizada la transformación y limpieza de los datos, se hizo uso del dataset para entrenar el modelo. Como primer acercamiento se realizó una investigación sobre qué modelos de Machine Learning son apropiados para darle una solución al reto. Finalmente se tomó la decisión de entrenar al modelo mediante el algoritmo de Multi Layer Perceptron (MLP) debido a su alta efectividad al tratar problemas de clasificación, justo como el problema que tenemos.

Al integrar el modelo se utilizó el train test con el 20% de los datos totales del dataset y las siguientes configuraciones de la red neuronal:

- 2 capas profundas de 60 nodos cada una.
- Epochs: 300.
- Función de activación: ReLu.
- Algoritmo de optimización de pesos: Adam.

Así mismo, se decidió realizar un segundo modelo. Con base a la investigación, se llegó a la conclusión de usar el modelo Multi-layer Perceptron classifier (MLP-classifier), el cual se encarga de realizar un algoritmo de backpropagation como parte del aprendizaje.

El modelo fue realizado haciendo uso de la librería de scikit-learn y se realizó con las mismas configuraciones, para poder realizar una comparación justa de modelos:

- 2 capas profundas de 60 nodos cada una.
- Epochs: 300
- Función de activación: ReLu.
- Algoritmo de optimización de pesos: Adam.

- Evaluation

Para evaluar el primer modelo se realizó un accuracy test, obteniendo como resultado un porcentaje de 86% de efectividad y como segundo paso para garantizar una alta efectividad se decidió realizar una matriz de confusión para detectar la cantidad de falsos positivos obtenidos en el modelo.

Como evaluación del segundo modelo se tomó la decisión de realizar de igual manera el accuracy test, sin embargo con este modelo, el porcentaje de precisión del modelo fue de tan solo 44% de efectividad.

Tras realizar ambos modelos y analizar su efectividad, se llegó al acuerdo de utilizar el modelo MLP para la implementación de la solución del reto, debido a la alta efectividad observada en el entrenamiento y testeo del modelo.

- Deployment

Para una mejor utilización del modelo se realizó una página web como interfaz gráfica, en esta, las y los encargados de aprobar los créditos, suben un archivo con formato .csv con la información requerida de los clientes y el sistema se encarga de procesarlos en Python haciendo uso del algoritmo para realizar la predicción.

En esta sección se hizo uso de tecnologías como HTML, CSS y JS para el frontend, logrando desarrollar una SPA - Single Page Application, la cual se dividió en 3 secciones:

- Página para subir el archivo CSV.
- Página en donde se procesa dicho archivo.
- Página en donde muestra los resultados de las predicciones.

Para el Backend se hizo uso de Express.js que realiza una conexión con el script de python. Como primer paso, procesa el archivo .csv y lo convierte en un dataframe, posteriormente se carga el dataframe al modelo MPL ya entrenado. Finalmente, realiza las predicciones y regresa los resultados al frontend, de una manera fácil de interpretar.

- Conclusiones

El modelo entrenado cuenta con un alto nivel de efectividad, resultado de un correcto proceso de extracción, limpieza y transformación de datos. Así mismo, es importante mencionar que las variables del modelo tienen alto valor explicativo.

De igual manera sería importante poder asignar diferentes parámetros al modelo de MPL-classifier, ya que al ser un algoritmo diferente, podría mejorar haciendo un cambio de hiperparámetros.