Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Puebla

Inteligencia Artificial para la ciencia de datos TC3006C

Reto
Etapa 2.

Fernando Jiménez Pereyra	A01734609
Daniel Flores Rodríguez	A01734184
Alejandro López Hernández	A01733984
Daniel Munive Meneses	A01734205

10 de Septiembre 2022

Índice

1. Business Understanding

El enfoque del proyecto que se va a desarrollar va sobre el proceso de la realización de préstamos bancarios que se dan mediante el uso de tarjetas de crédito, y la importancia de predecir quién podría llegar a cometer un incumplimietno crediticio es algo muy importante para los prestamistas a la hora de poder optimizar sus decisiones sobre a quién es buena idea dar un préstamo, esto por consecuencia también beneficia al cliente al tener una mejor experiencia con el banco y al mismo tiempo mantener un estado financiero.

En este caso se trabajará con American Express que es una empresa de pagos globalmente integrada y el emisor de tarjetas de pago más grande del mundo brinda a los clientes acceso a productos, conocimientos y experiencias que enriquecen vidas y construyen el éxito comercial.

Estudiando el caso de American Express -Default Predition nos dimos cuenta que no podíamos conocer los factores involucrados, ya que los datos se encuentran anonimizados, pero sin embargo entendimos que para generar un modelo capaz de predecir los impagos debíamos de juntar los set de datos de los comportamientos de los clientes, con el de las aprobaciones de crédito, y con esto ayudar a crear una mejor experiencia del para los titulares de tarjetas al facilitar la aprobación de una tarjeta de crédito

2. Data Understanding

Primeramente para poder realizar un buen eso de los datos que teníamos se realizó una investigación y comprensión de la Base de datos que nos fue entregada, lo que se encontró fue que el conjunto de datos contiene características de perfil agregadas para cada cliente en cada fecha de estado de cuenta y las funciones se anonimizan y normalizan, y se clasifican en las siguientes categorías generales:

- D_* = Variables de delincuencia
- S_* = Variables de gasto
- P_* = Variables de pago
- B_* = Variables de balance
- R_* = Variables de riesgo

Donde las siguientes variables son categóricas

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

Donde además la variable independiente era guardada únicamente con un valor binario de 1 o 0 para saber con esto si es que este había pagado o no,

3. Data preparation

3.1. Extracción

Se realizó una extracción total de dos sets de datos presentes en el reto "American Express -Default Predition", data.csv que contienen el comportamiento de los clientes segmentados en categorías, los cuales realizaron el rol de variables dependientes, y train labels.csv que contienen los resultados del cumplimiento de los pagos de los clientes.

A partir de estos dos conjuntos de datos se creó un único set de datos conjuntos, tomando el identificador de los usuarios para unir los datos.

3.2. Transformación

3.2.1. Limpieza

Decidimos que para manejar las columnas con valores faltantes, aquellas que poseyeran 20% o más de valores faltantes las eliminamos, porque consideramos que el imputar datos con tantos valores faltantes afectaría el comportamiento real de las variables, y para aquellas que poseyeran 5% o menos de valores faltantes eran una cantidad insignificante de valores faltantes y por lo tantos eliminar las filas era una opción óptima. Finalmente para las columnas con un porcentaje de valores faltantes intermedio imputamos los valores faltantes con la media.

Una vez que habíamos limpiado el set de datos de los datos faltantes, realizamos un subgrupo del 15% para el entrenamiento, guardandolo como train_merged_clean_small_data.csv, y train_merged_clean_big_data.csv para los datos restantes.

Una vez teniendo un dataset pequeño y uno grande, se optó por hacer una limpieza de datos a partir del dataset pequeño empleando OneHot Encoding, PCA y agrupación de usuarios por fila; comenzando por la lectura del dataset pequeño

```
trainSmallSet = pd.read_csv('../../data/train_merged_clean_small_data.csv')
trainSmallSet
```

	Unnamed: 0	Unnamed: 0.1	customer_ID	S_2
0	23	9358	8cff42b70b8c710e494ad76a6a2b81ce8fed1c43a0c207...	2017-09-02
1	24	13414	ca4f526810d21c72d7a355a21946d7c46277b046cd8fb7...	2017-09-11
2	24	7907	770148eb1c05ed2bc015cb46633eef6a9ced0444a2cebf...	2017-06-29
3	39	11466	ac80171e4c62c8acd8f08d9e4c6b1787b9671910048156	2017-

Posteriormente, se obtuvieron el nombre de las columnas importantes basándonos en la descripción del dataset en Kaggle; para así aplicar OneHot Encoding para aquellas columnas que manejen valores no numéricos, y así traducirlas de tal forma que se puedan leer dentro del modelo.

```
columns_trainSmallSet  = trainSmallSet [['B_30', 'B_38', 'D_114', 'D_116', 'D_11
labelEncoder = preprocessing.LabelEncoder()
columns_trainSmallSet  = columns_trainSmallSet .apply(labelEncoder.fit_transform
columns_trainSmallSet
```

	B_30	B_38	D_114	D_116	D_117	D_120	D_126	D_63	D_64	D_68
0	0	1	1	0	4	0	2	5	3	6
1	0	1	1	0	5	0	2	4	3	6
2	1	4	0	0	0	0	1	5	1	2
3	1	2	1	0	3	0	2	5	3	3
4	0	2	1	0	4	1	2	3	3	6
...	...	...	...	...	...	...	...	...	...	...

```
mat = preprocessing.OneHotEncoder()
mat.fit(columns_trainSmallSet )
one_hot_labels = mat.transform(columns_trainSmallSet ).toarray()
one_hot_labels
```

```
ray([[1., 0., 0., ..., 0., 0., 1.],
    [1., 0., 0., ..., 0., 0., 1.],
    [0., 1., 0., ..., 0., 0., 0.],
    ...,
    [1., 0., 0., ..., 0., 0., 0.],
    [1., 0., 0., ..., 0., 0., 1.],
    [1., 0., 0., ..., 0., 0., 1.]])
```

```
cat_col_df = pd.DataFrame(one_hot_labels, columns = cat_col_names)
cat_col_df
```

Pyth

	B_30_0	B_30_1	B_30_2	B_38_0	B_38_1	B_38_2	B_38_3	B_38_4	B_38_5	B_38_6
0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
1	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
3	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...
23205	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
23206	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0

Con dichos datos traducidos, se sacó el eje x e y del data frame original; siendo el X el data frame original salvo la columna ‘target’, y el eje y sólo la columna ‘target’.

```
trainSmallSet_x = trainSmallSet.drop(columns=['target'])
trainSmallSet_y = trainSmallSet[['target']]
```

Después, se implementó PCA con tal de rellenar datos vacíos de acuerdo a la misma información del dataset con tal de mantener congruencia en las filas del data set. Haciendo uso de funciones de imputación iterativa y la obtención de 20 componentes o variables importantes para el futuro modelo, se logró hacer una limpieza de datos efectiva

```
trainSmallSet = trainSmallSet.sort_values(by=['customer_ID'])
pca_trainSmallSet_x = trainSmallSet_x.drop(columns=['customer_ID', 'S_2'])
imp = impute.IterativeImputer()
imputed_trainSmallSet_x = imp.fit_transform(pca_trainSmallSet_x)
```

Python

```
imp_trainSmallSet_x = pd.DataFrame(imputed_trainSmallSet_x, columns = list(pca_trainS
```

Python

```
pca = decomposition.PCA(n_components=20)
trainSmallSet_x_imputed_pca_df = pca.fit_transform(imp_trainSmallSet_x)
```

Como último paso de la limpieza, se eliminaron las columnas de customer_ID y fecha de la fila; esto con tal de poder hacer la agrupación de los usuarios por filas únicas y así tener un data set más eficiente para el modelo a emplear. Con el dataset limpio, se envía a un archivo csv que leerá el código del modelo llamado train_user_collapsed_small_data.csv.

```
trainSmallSet_pca = trainSmallSet_y.join(trainSmallSet_x[['customer_ID','S_2']]).join(trainSmallSet_x_imputed_pca_df)
trainSmallSet_pca
```

	target	customer_ID	S_2	column_0	column_1	column_2	column_3	column_4
0	0	8cff42b70b8c710e494ad76a6a2b81ce8fed1c43a0c207...	2017-09-02	8453.217259	-986.438274	-0.237870	-0.202708	-0.112456
1	0	ca4f526810d21c72d7a355a21946d7c46277b046cd8fb7...	2017-09-11	9349.369229	2969.323176	-0.230675	-0.163126	-0.100315
2	1	770148eb1c05ed2bc015cb46633eef6a9ced0444a2cebf...	2017-06-29	8131.302292	-2401.278465	-0.209537	0.088882	-0.097364
3	0	ac80171e4c62c8acd8f08d9e4c6b1787b9671910048156...	2017-	8903.871967	1072.889365	-0.215740	0.024188	-0.091726

Agrupamiento de los usuarios en el dataset final, de tal forma que se tenga un cliente por fila del archivo

```
trainSmallSet = trainSmallSet_pca.groupby('customer_ID', as_index = False).mean()
```

```
trainSmallSet
```

	customer_ID	target	column_0	column_1	column_2	column_3	column_4	column_5	column_6	column_7	...	column_10	column_11
0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	0.0	6397.501907	-10104.584324	-0.230231	-0.121854	-0.124036	0.129043	-0.097399	-0.653849	...	-0.619522	-0.186811
1	00000fd6641609c6ece5454664794f0340ad84dddce9a2...	0.0	6391.218184	-10094.614182	-0.229584	-0.203984	-0.129463	0.100098	-0.141379	-0.671607	...	0.004898	-0.531235
2	00001b22f846c82c51f6e3958ccd81970162bae8b007e8...	0.0	6377.896696	-10073.477481	-0.233251	-0.182213	-0.137842	0.093621	-0.161345	-0.681673	...	0.133387	-0.174952
3	000041bdba6ecadd89a52d11886e8eaaec9325906c9723...	0.0	6376.388602	-10071.084648	-0.230502	0.097643	-0.133321	0.091601	-0.122672	-0.428077	...	0.009457	-0.518589
4	00007889e4fcd2614b6cbe7f8f3d2e5c728eca32d9eb8a...	0.0	6358.291484	-10042.370639	-0.227853	-0.087171	-0.132026	-0.117923	-0.144135	-0.388776	...	-0.692867	0.175252

```
trainSmallSet.to_csv('../../data/train_user_collapsed_small_data.csv')
```

3.3. Carga

Durante cada etapa del proceso en la cual obtenemos como resultado un dataset modificado, lo guardamos con el fin de preservar el resultado del tiempo invertido, por si en un futuro decidimos partir desde algún punto intermedio que abarcara uno de los dataset guardados.

Los datos presentes en train_user_collapsed_small_data.csv fueron guardados en una base de datos relación con el SGBD MySQL en una instancia de Amazon Web Service con el fin de poder consultarlos como mejor se considere.

4. Modeling

Con los datos ya limpios y agrupados en una fila por usuario, corresponde entregar al modelo empleado el resultado de dicha limpieza. Considerando que se tiene una naturaleza de predicción en el reto, se optó por emplear una red neuronal para poder determinar la autorización o negación de las solicitudes de tarjeta de cada uno de los clientes.

Para efectos de la división de datos para entrenamiento, comprobación y pruebas. Se distribuyó de manera aleatoria los datos de la matriz en la siguiente proporción: 75% de los datos se irán al set de entrenamiento; mientras que el otro 25% se irá al set de pruebas

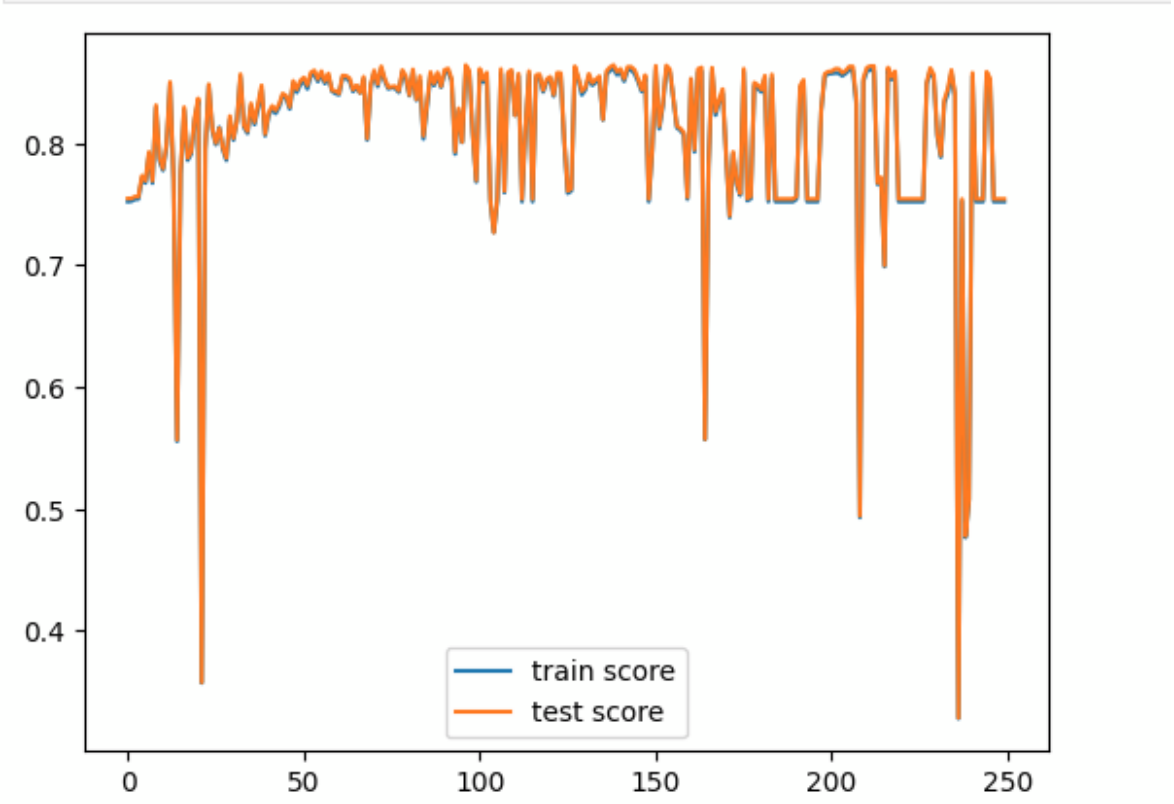
La red neuronal está bajo el marco del modelo MLP (MultiLayer Perceptron); y cuenta con 2 capas internas de 80 y 65 nodos respectivamente. Del mismo modo, se empleó una función de activación basada en el algoritmo ReLU (Rectified Linear Unit); mientras que para el algoritmo solucionador de distribución de pesos se emplea el algoritmo Adam. Del mismo modo, se optó por basar el entrenamiento supervisado en épocas; siendo el límite 250 para hacer correr el modelo con un learning rate de 0.0015.

```
clf = MLPClassifier(hidden_layer_sizes=(80, 65), max_iter=250, activation = 'relu', solver = 'adam', random_state=1, learning_rate_init= 0.001)
```

Como contraste al modelo de clasificación, se hizo uso de un segundo modelo con enfoque a la regresión lineal. Del mismo modo, funciona con el modelo MLP con características similares: 2 capas internas de 80 y 65 nodos, función de activación ReLU, algoritmo de distribución de pesos Adam, learning rate de 0.001 y 250 épocas para correr el modelo

5. Evaluation

Para evaluar la fiabilidad del modelo se realizó un test de confianza, el cual corrió con un resultado cercano a 85% de fiabilidad. En adición a ello, se hizo una graficación comparativa del comportamiento de este porcentaje a lo largo de las épocas del modelo tanto en train como en test.



Por otro lado, el segundo modelo tuvo un desempeño sumamente erróneo, lanzando valores incluso fuera del rango de 0 o 1. Teniendo un score de -38.1799. El motivo principal de emplear este modelo es para reafirmar que el problema en cuestión no puede desenvolverse en un marco de regresión y sí en uno de clasificación; porque la regresión se considera para variables continuas y no paras variables discretas, las cuales son las que corresponden a la salida esperada: el usuario pagaría o no pagaría su tarjeta (0 o 1, sin valores intermedios).

6. Deployment

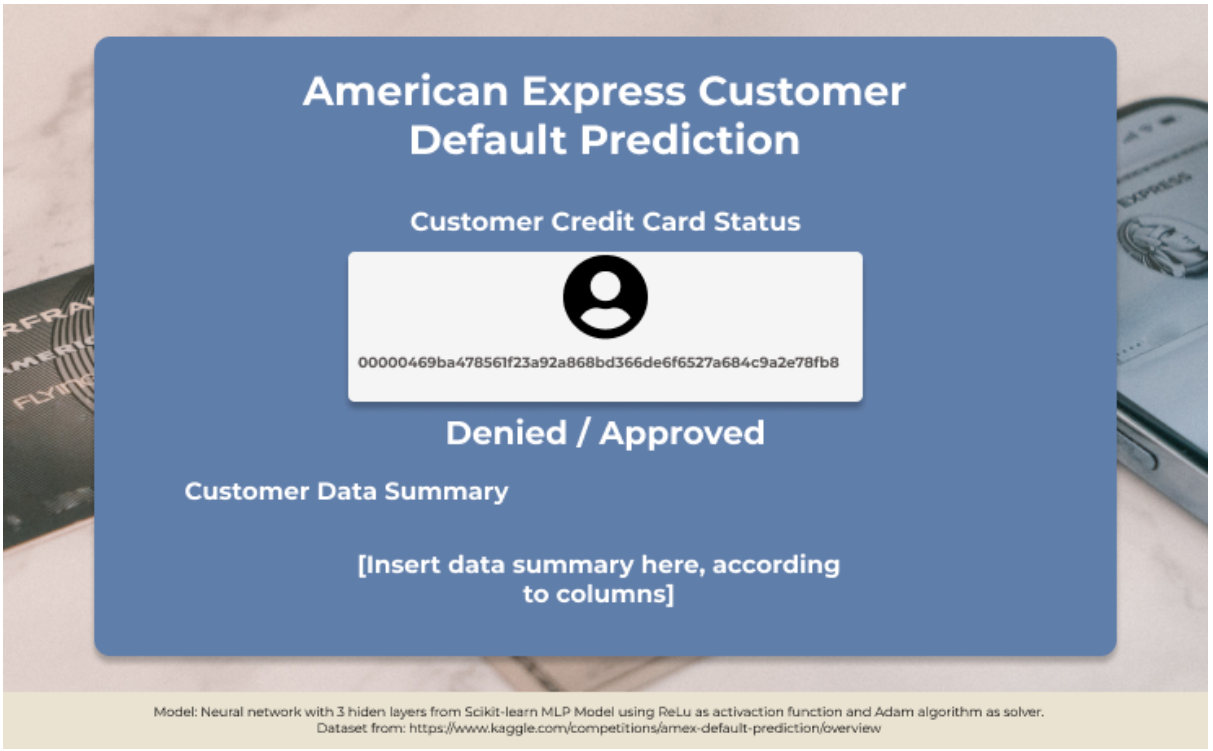
Para una mejor visualización de los resultados obtenidos, así como una mejor comprensión de los resultados del modelo aplicado a la solución del reto, se realiza una visualización gráfica a través del uso de un framework, con el cual se conecta a un servidor (en donde se montará el modelo) y la misma aplicación a realizar.
Para la realización de esta aplicación, se usó el framework llamado SVELTE, esto debido a que es parecido a otros frameworks con los que tenemos experiencia previa trabajando, como lo es React JS.

Al trabajar con este framework, podemos realizar una API con la cual podemos crear peticiones y consultas, y así mostrar el resultado obtenido de haber analizado los datos con el modelo implementado.

A continuación se muestran los prototipos de las pantallas que serán implementadas en dicho framework:

La primera pantalla es la que se mostrará al momento de ingresar a la aplicación, está contendrá textos y algunas descripciones de cómo poder usar, en general, la primera pantalla. Teniendo como componente principal, una caja de texto, donde el usuario podrá ingresar un ID de algún cliente en especifico y así poder predecir la posibilidad de pago de dicho cliente. Además, el usuario podrá tener la posibilidad de 'generar' un id aleatorio, a través de un botón ubicado a un lado de la caja de texto, o bien poder pasar a visualizar los resultados obtenidos por el modelo a través del botón principal.
En la parte de abajo, en la sección del footer, el usuario puede obtener la fuente del origen del dataset usado para la realización del modelo, así como también información sobre el modelo usado para la implementación de la solución de este reto.

Para la segunda pantalla de la aplicación, se podrá visualizar los resultados obtenidos de la predicción del cliente que se especificó en la primera pantalla, por lo tanto, solo se obtendrá y se mostrará el resultado, así como el resumen de datos de acuerdo a las columnas. Aunado a esto, también se podrá visualizar en todo momento información sobre el modelo usado y el dataset usado en la parte inferior del footer.



Cabe mencionar que durante la realización de la aplicación y para poder llevar el control de versiones se hizo uso de GitHub y Visual Studio Code, además de diversas herramientas para poder realizar el diseño de la aplicación, para esto, se hizo uso de la aplicación Figma.
Durante el desarrollo de la aplicación, la forma en la que se visualiza el resultado del diseño web y los resultados de las predicciones, es de manera local, es decir la aplicación simula correr en un servidor, sin embargo, lo que sucede, es que realmente se encuentra corriendo en nuestra computadora, de manera local.
Al tener una aplicación de esta manera, los usuarios o el usuario final no podría acceder. Así que para poder desplegar la aplicación, debemos de montar nuestra aplicación en un servidor, en este caso usaremos AWS (Amazon Web Services), de esta manera, podemos tener nuestra API montada en un servidor y lista para poder ser consultada a través de su link, además de poder acceder de manera remota a la aplicación, de igual manera, a través de un enlace.