FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Departamento de Computación

Organización de Datos

Trabajo Práctico de Análisis de Datos

Primer cuatrimestre de 2022

Grupo N°19

Fecha de entrega: 11/08/2022

Padrón	Apellido/Nombre	Correo Electrónico
104667	Francisco Monopoli	fmonopoli@fi.uba.ar
97641	Braian Villalba	bvillalba@fi.uba.ar
107992	Mateo Lardiez	mlardiez@fi.uba.ar
93216	German Samoluk	gsamoluk@fi.uba.ar

Repositorio : https://github.com/BraianV/Org-Datos-Grupo-19

Parte 1 - Conjunto de Datos

Dicho código se encuentra en la notebook Reducción de set de datos, nuestra semilla por ser el grupo 19 corresponde al número 904.
Obs: el set de datos original era tan pesado que se utilizó una notebook con mucha capacidad de ram para poder delimitar el set de datos con sample_without_replacement, había otras alternativas que se podían haber utilizado que venían de la mano de algoritmos streams para poder levantar parcialmente el set de datos original sin la necesidad de que todo el dataframe quede en memoria.

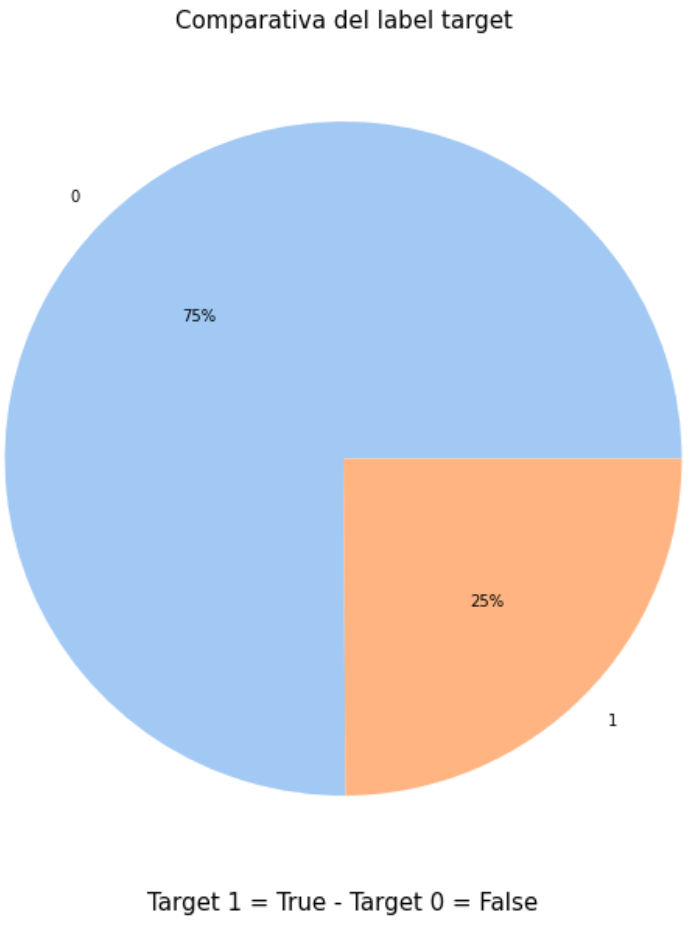
Parte 2 - Ciencia de Datos

Exploración, preprocesamiento y transformación de datos

A. Visualización de los datos

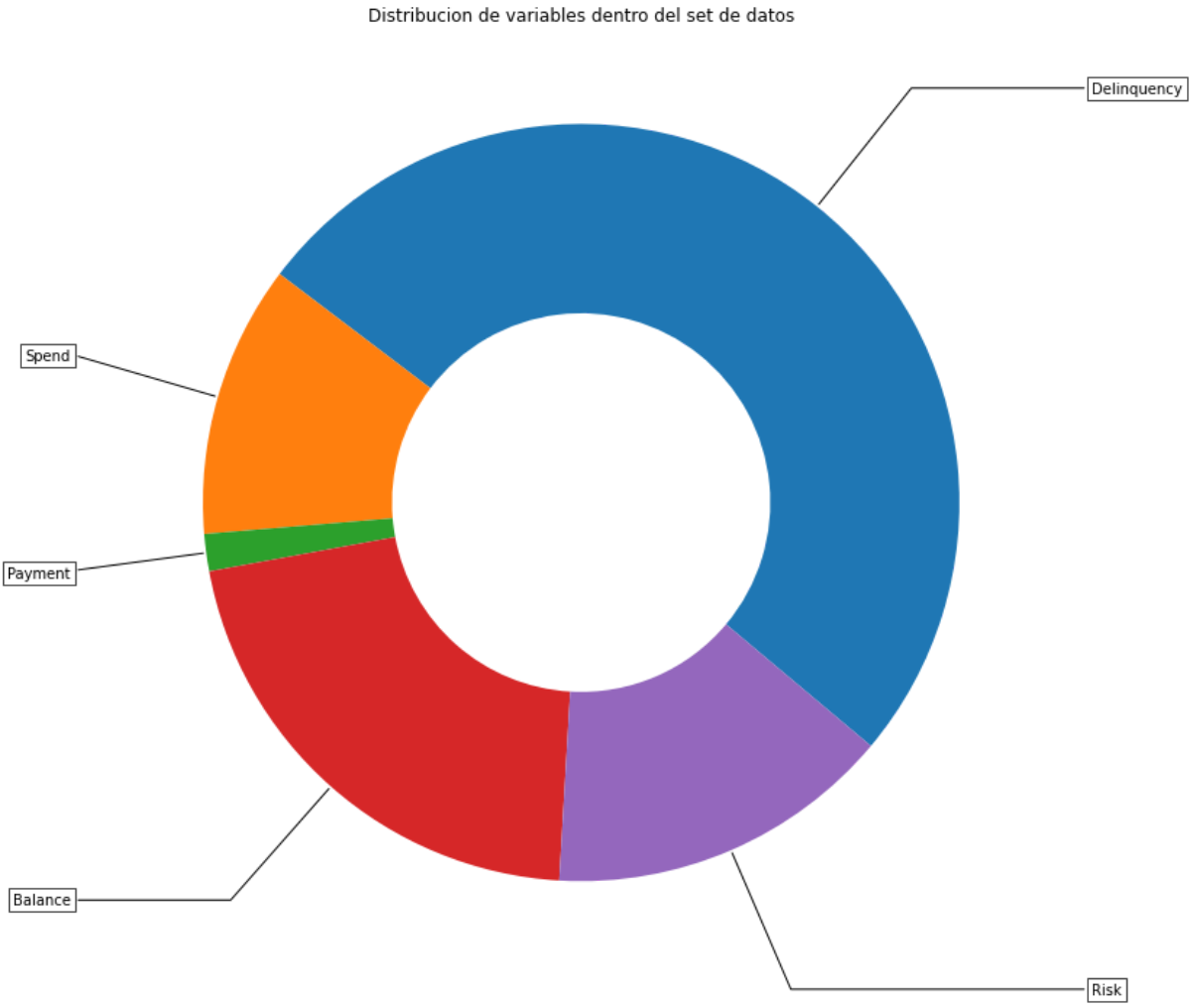
Todas las visualizaciones están cargadas(con su código fuente correspondiente) en la notebooks Visualizaciones Parte 1 y Visualizaciones Parte 2 con un pequeña conclusión de lo que se puede observar a primera vista.

Análisis sobre el target



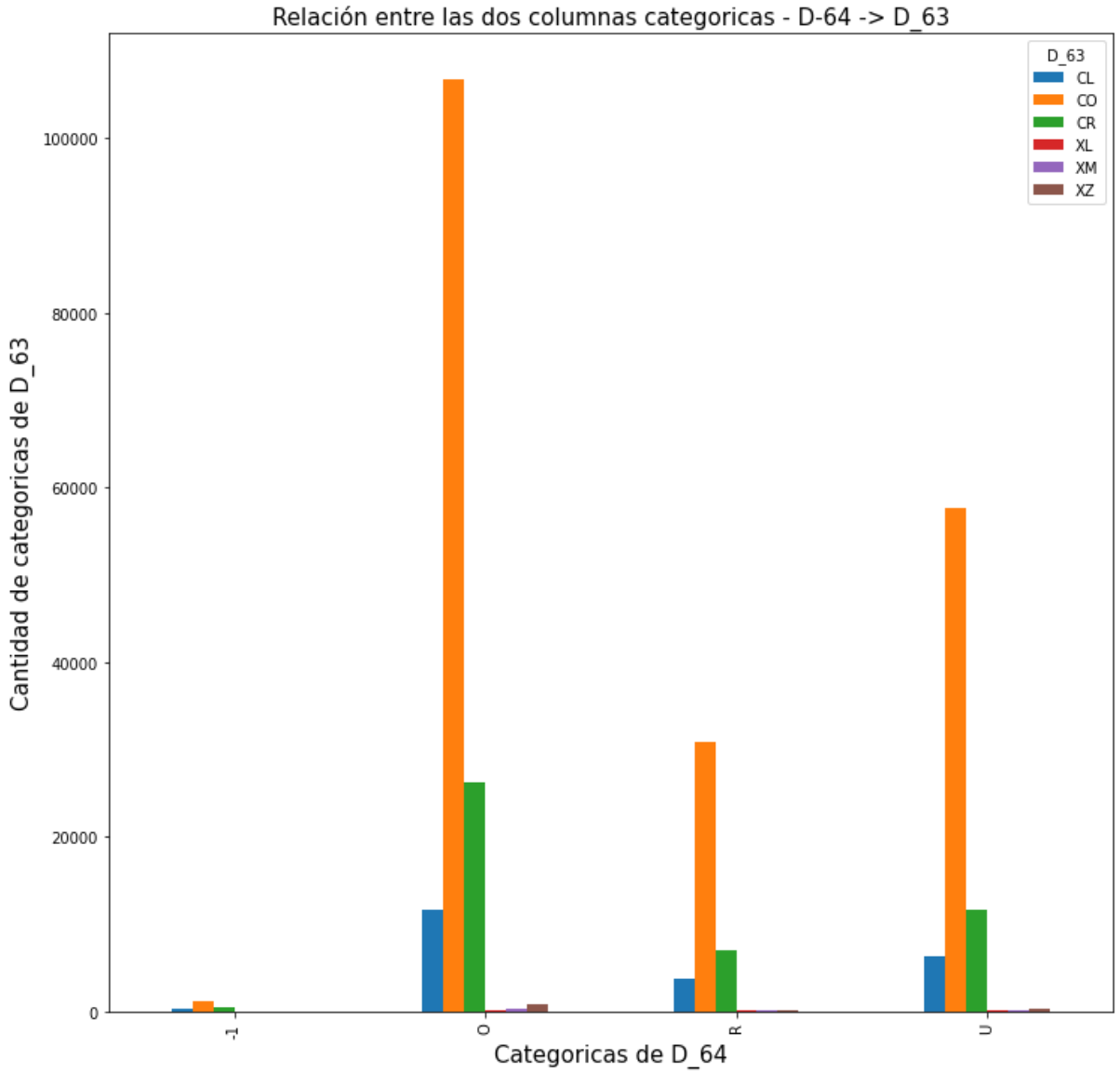
Se puede observar que un 75% de los targets son negativos, probablemente haya que aplicar algún tipo de balanceo a la hora de tomar este label para predecir.

Análisis sobre el tipo de variables dentro del set de datos

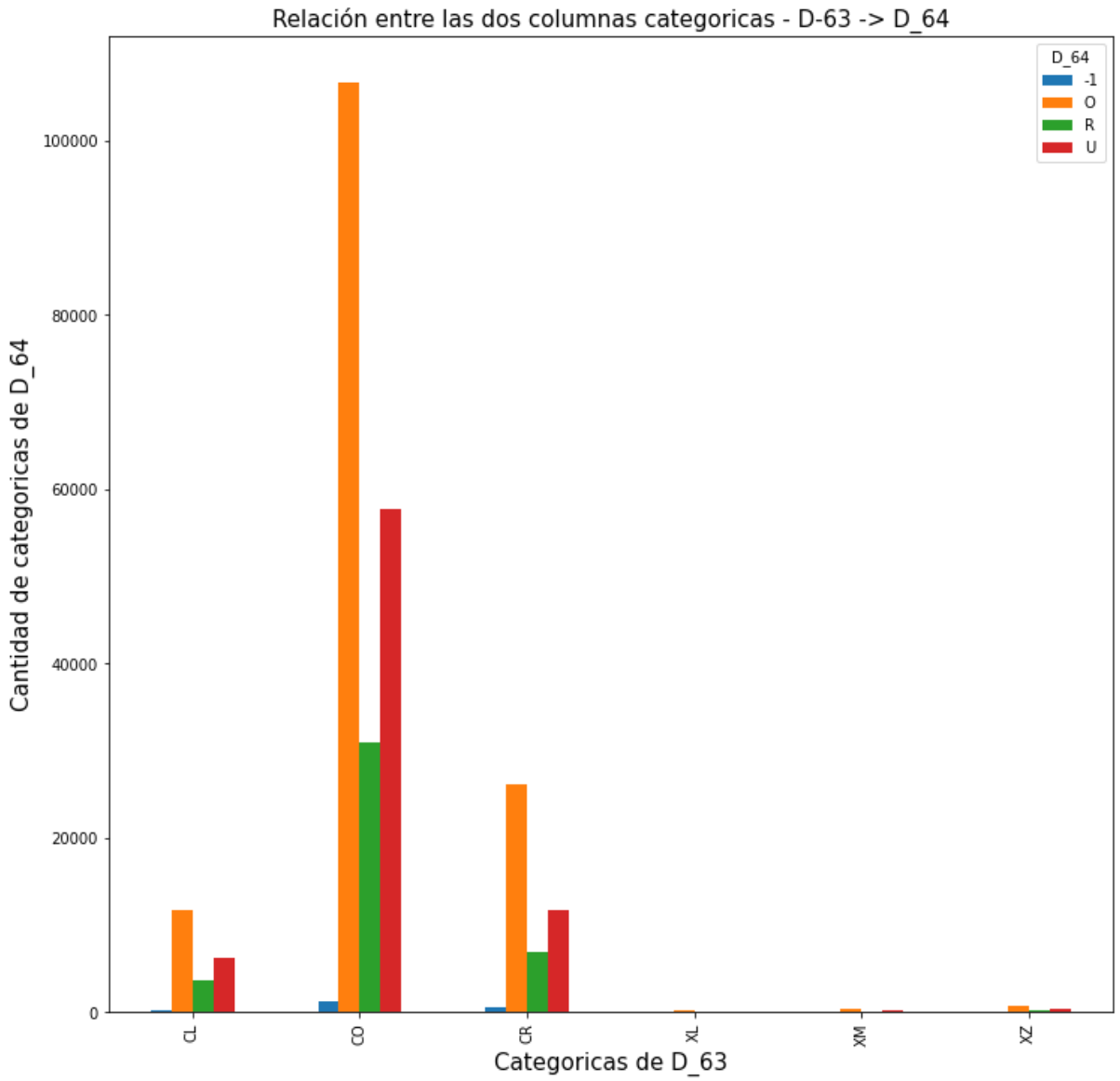


Este plot nos marca la pauta de que las variables de delincuencia tienen mucha injerencia en el set de datos a nivel columna ya que es la que predomina entre todos los tipos de variables, por contraparte, las de payment son el tipo de menor cantidad de features dentro del set de datos (solo 3)

Análisis sobre las variables categóricas

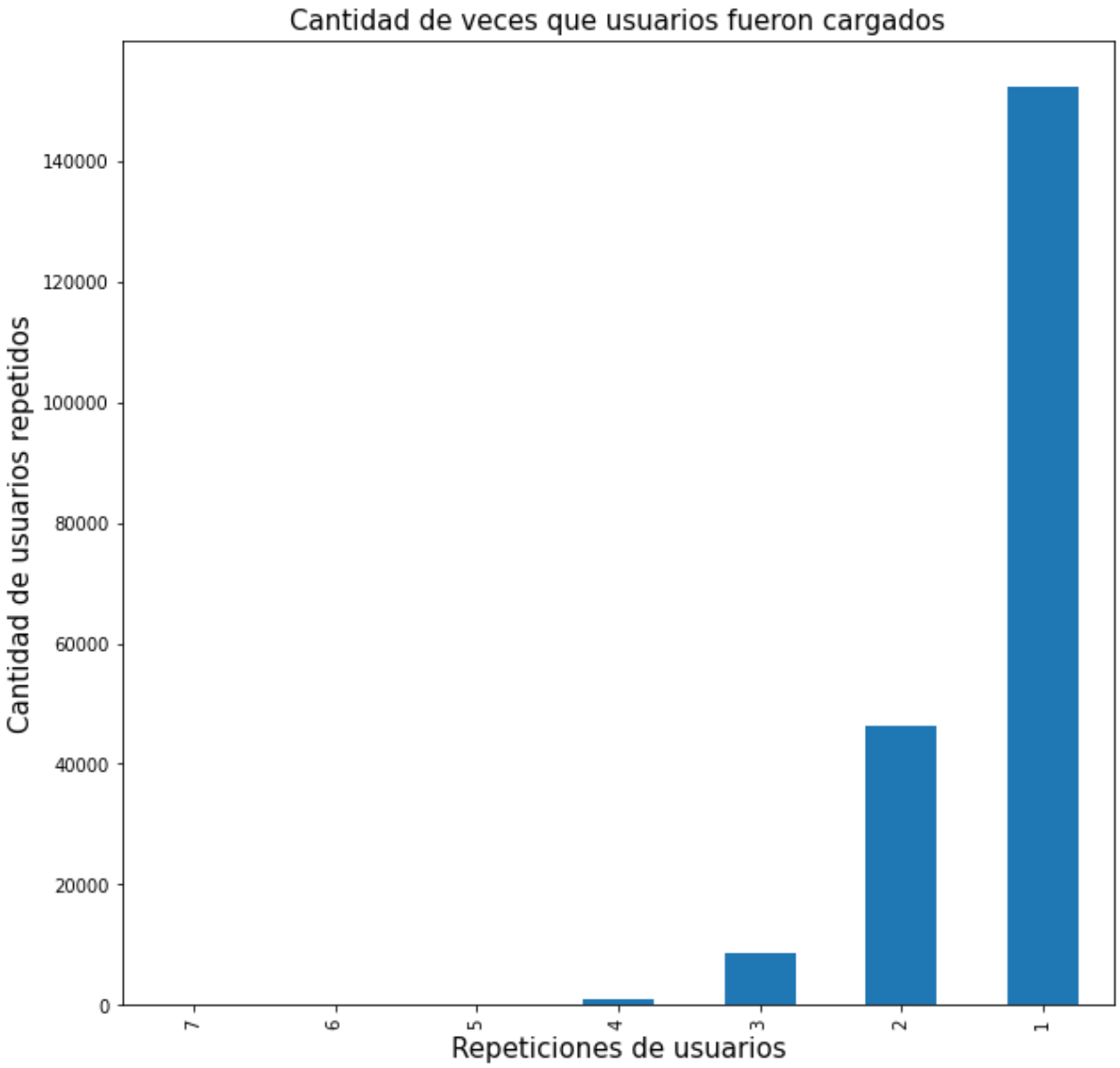


Se toma a D_64 como columna de agrupación más grande, pero es simplemente porque tienen menos categorías que D_63, como se puede observar, CO en D_63 es totalmente dominante en todas las categorías.

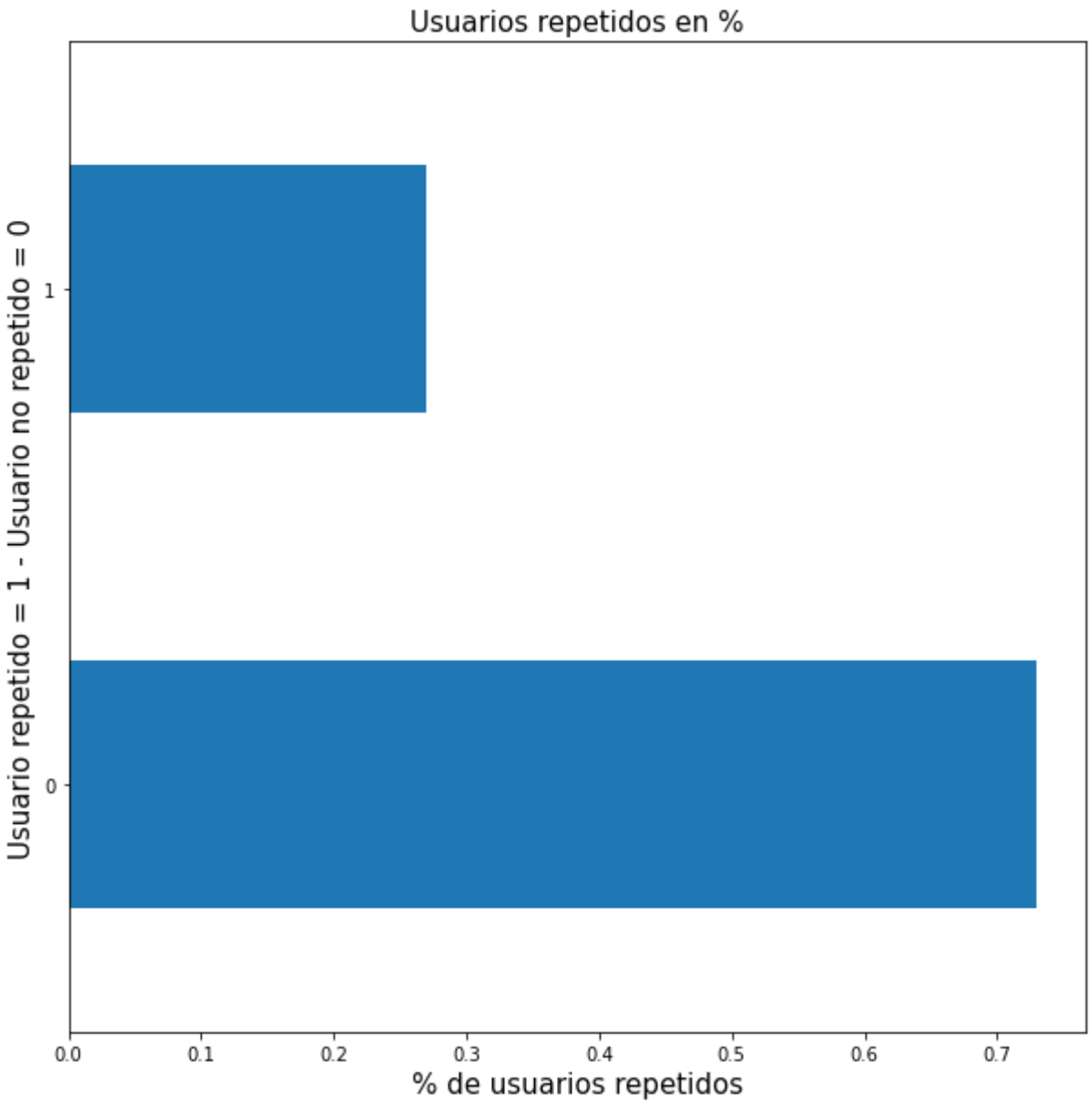


En este caso, invertimos el orden de agrupamiento y podemos observar que dentro de CO (que es la que tiene más importancia en cantidad), el tipo de variable O se lleva las de ganar en muchas de las categorías de D_63

Análisis sobre usuarios repetidos

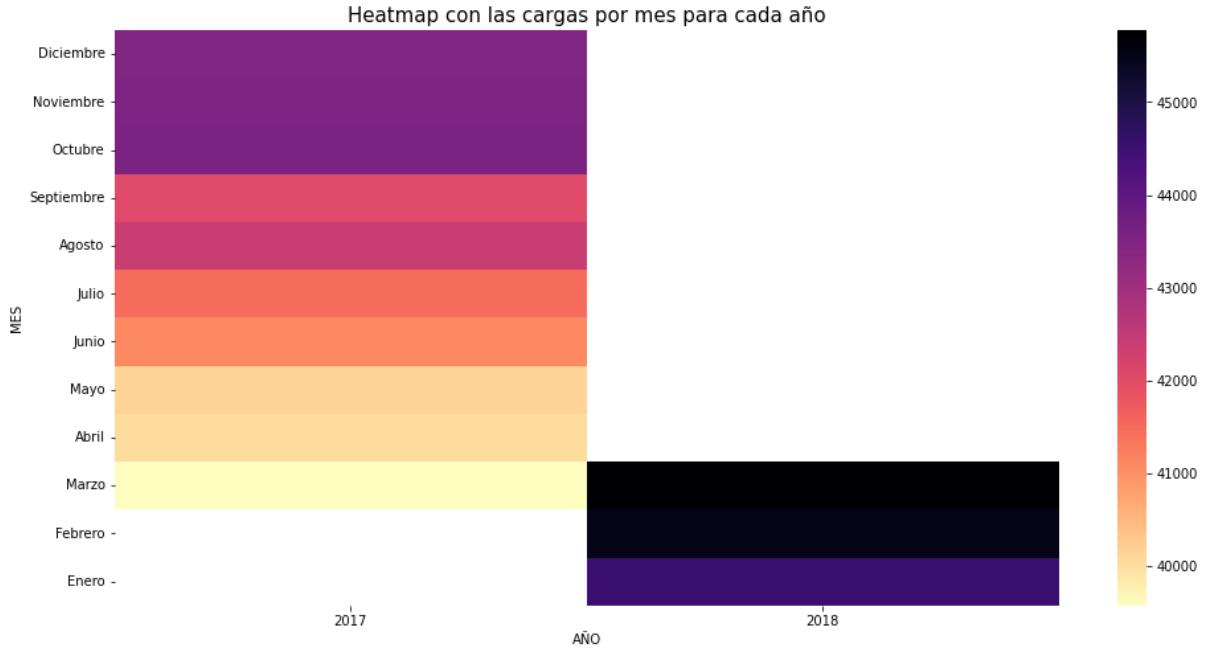


Podemos notar que muchos usuarios fueron cargados una sola vez en el set de datos, con lo cual, es su único ingreso dentro de los datos.



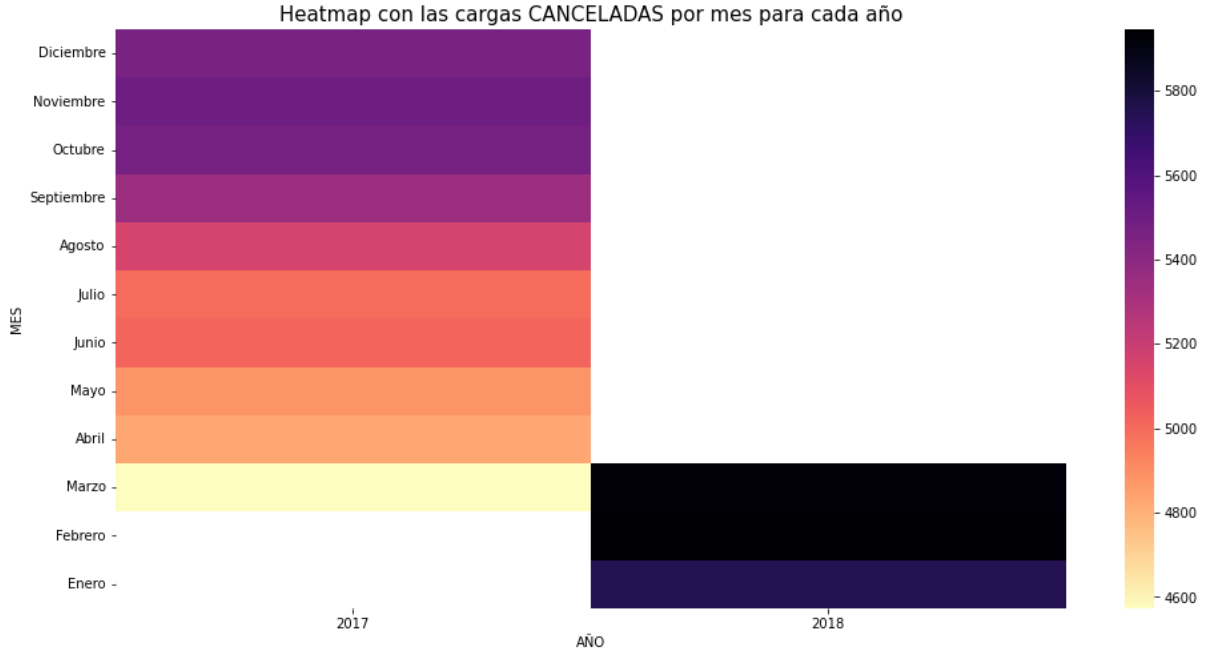
Acá podemos destacar que más del 20% de los customer_id están cargados más de una vez dentro de nuestros datos, posible feature en este caso.

Análisis sobre las fechas de cargas

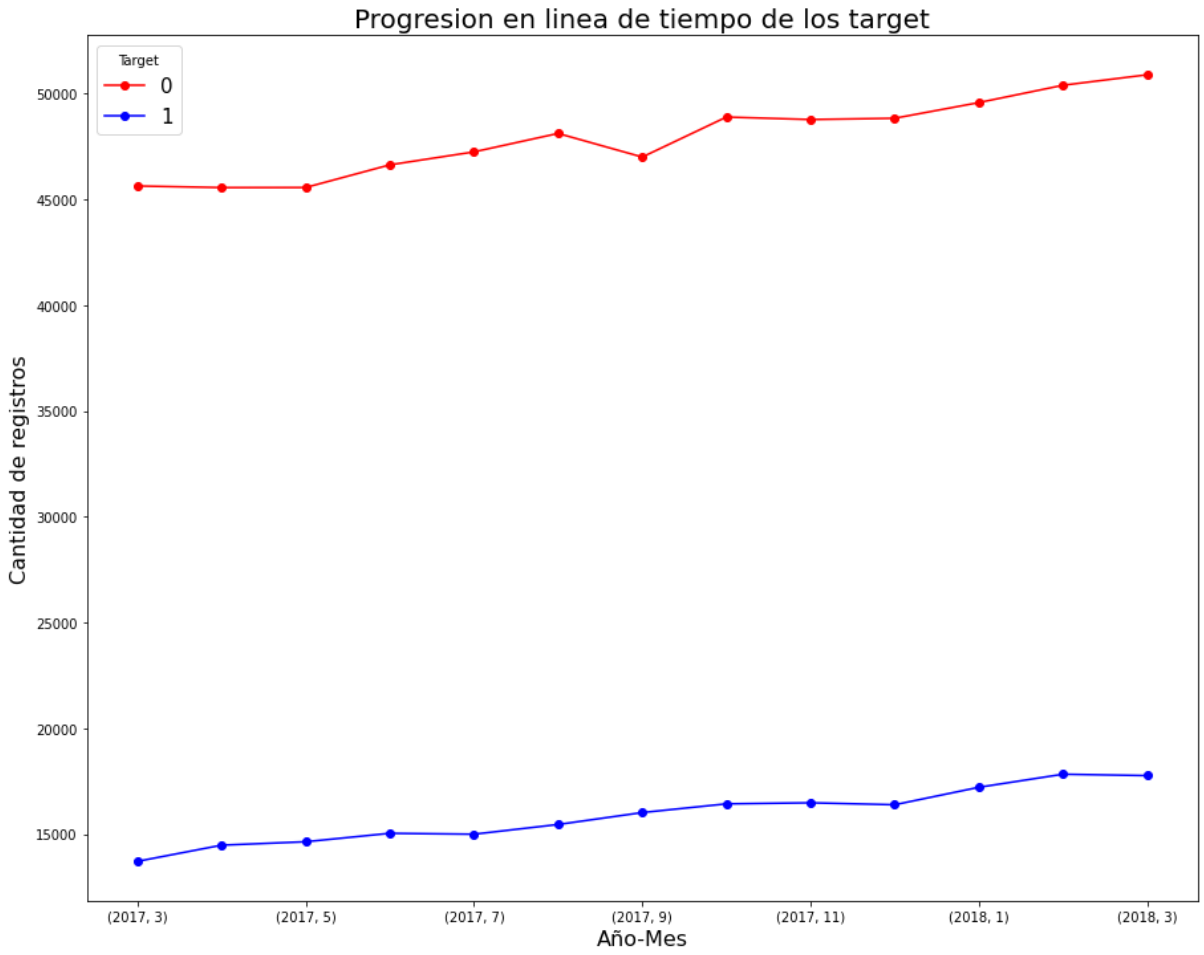


El set de datos marca una tendencia en el mes de Marzo y Febrero del 2018 donde hubo una sobrecarga de datos cargados sobre esa fecha. Hay una especie de incrementalidad desde enero del 2017 hasta marzo del 2018 en cuanto a la cantidad de registros.

Análisis sobre las fechas de cargas donde fueron canceladas (target = 1)



El plot nos muestra que no hay ningún tipo de estacionalidad marcada en algún mes o año particular, mantiene la misma proporción que los datos sin filtrar por el label igual a 1.



Acá se puede observar mejor la progresión de los registros dependiendo del target de cada uno y se puede observar que ambos targets tienen una tendencia de crecimiento correlacionado.

Por último, se realizaron múltiples boxplots sobre cada columna numérica para poder tener una aproximación de mediana, cuartiles, outliers entre otros. Estos plots se encuentran en Visualización - Parte 2.

B. Ingeniería de características

Se realiza una limpieza de columnas correlacionadas, con un umbral de aquellas columnas que superen el 70% de correlación, son dropeadas del dataframe original, ya que infieren una cierta redundancia en las columnas.

En cuanto a los outliers, se comenzó analizando el z-score de cada feature individualmente, removiendo las filas que cumplieran con tener una desviación de más de | z-score | >3 en cualquier columna, lo cual resultó en un dataframe extremadamente reducido (más del 80% removido). Se procedió a realizar un análisis multivariado para contemplar este problema (ya que al tener tantas features, se volvió necesario considerar los outliers a lo largo de todos los features). Se aplicó un algoritmo de Isolation Forest con una tolerancia a la contaminación de 0.05, es decir que se removieron las filas que

excedieran 5% de desviación. De esta forma, aproximadamente 15000 registros o aproximadamente 10% del dataset fue eliminado.

Mediante la reducción de columnas aplicadas por ser correlacionadas hubo una reducción en la dimensionalidad de los datos, aunque se podría aplicar otros algoritmos más sofisticados como DSV o PCA.

Para la creación de nuevos datos se tendrá en cuenta el modelo a utilizar, ya que cada nueva feature podría tener un impacto diferente en cada modelo utilizado (XGB, random forest o redes neuronales).

En cuanto al balanceo de los datos, se tomó en consideración para la red neuronal ya que funciona mucho mejor con labels entre 0 y 1, para el modelo de XGB se descompuso la fecha de cada row como día, mes o incluso, el año como un nuevo dato. Para Random Forest Classifier se aplicó la misma descomposición.

En cada train_test_split del dataframe se estratifico el label target para que mantenga la distribución de 70/30 que se encuentra dentro del dataframe para poder obtener una distribución equitativa tanto en el set de entrenamiento como en el set testeo.

Para finalizar la depuración, filtramos aquellas rows con Nans dentro de las columnas categóricas ya que era un número despreciable en comparación con el set de datos original.

C. Descripción de los datos

Para comenzar a describir los datos con los que estuvimos trabajando podemos ver que hay una gran cantidad de columnas con porcentajes de nulos muy grandes (mayor a 70%) las cuales suponemos que no aportan nada al ser o no null. Pensamos en encodear las columnas y hacer un 1 si tenía valor o 0 si no, pero al no tener ninguna forma de saber si esto era así o no, decidimos borrar las columnas.

De las 11 columnas que estaban dadas como 'categorical' solamente dos de ellas eran de tipo ‘object’, el resto estaban encodeadas. Asumimos esto debido a que (sacando las que tienen sólo 0 y 1) eran todas variables que tenían valores numéricos flotantes y que los más probable era que representaba un valor encodeado de alguna forma. Contando a las binarias, podría ser que representen lo dicho más arriba. Que había valores o no. Al no tener la forma de encodeo, ni que palabras u objetos que eran antes, es imposible ‘desencodearlas'. No era necesario ya que igual había que encodear las de tipo 'string' para utilizarlas en los modelos, pero si se quisiese saber el valor original, tendríamos que tener la forma de encodeo y las variables utilizadas.

De las variables de Delinquency encontramos que gran parte de las filas tenían gran cantidad de nulos en los features de Delinquency, considerando que posiblemente estén relacionadas a defaults anteriores del cliente, podrán tener cierto peso en el resultado final.

Una de las cosas más notorias que vimos en el conjunto de datos, es que sacando la primera columna que aparece después de Unnamed :0 e ID, había una columna de fechas. El resto de las columnas asumimos que eran valores de cantidad de veces que hubo 'pagos' o 'delincuencia' ya que cualquier aproximación hacia la posible información brindada, no cerraba por ningún lado.

Para el manejo de outlier el principal problema de intentar trabajar con los outliers es que dependiendo de la forma de querer resolverlo, podrías casi vaciar la cantidad de datos ingresados. El método que nosotros utilizamos al final fue el de 'Isolation forest' ya que nos pareció que la cantidad de datos final para trabajar con los modelos era suficientemente grande como para que no varíe los posibles resultados.

Utilizando tanto Z-Score como el método de usar 'topes de outliers' con la frecuencia de aparición de datos, vimos que se eliminaban una gran cantidad de datos. Esto debido a que como los valores no tienen una 'relación notoria', borraban datos a partir de los valores por columna entonces por ahí borraba 2500 filas por columna y si lo multiplicas por la cantidad de columnas que había, sacando las de ID, eliminaban más de la mitad de los datos.

Por último, las métricas que se usarán para medir la efectividad de los modelos son los que hemos utilizado en los trabajos prácticos anteriores

- Precisión
- Recall
- F1 Score
- Área bajo la curva (AUC)

Modelos:

SVM

Se implementó SVM en primera instancia ejecutando SVC de sklearn y LinearSVC. Se esperaba que tanto SVC con kernel 'linear' puedan converger con cierta facilidad, sin embargo no lo logramos a pesar de aplicar StandardScaler (restar la media y dividir por la varianza) y remover conjuntos de features (por ejemplo los de Delinquency).

Se procedió a aplicar grid search para encontrar hiperparametros adecuados, se utilizó un conjunto reducido de filas (20000, un 10%) para lograr encontrar con rapidez los valores adecuados para converger rápidamente. Se encontró que kernel rbf/gaussiano con tolerancia 1e-7 y C=1 resultó apropiado así como kernel polinómico con mismos valores de tolerancia y grado de polinomio 3.

Sin embargo, al probar sobre el conjunto completo, el algoritmo no logra converger en ningún caso correctamente y demora bastante en realizar el entrenamiento. Una posibilidad es que los puntos no sean fácilmente separables y por lo tanto no se puedan formar los conjuntos, otra es que falte algún tipo de preprocesamiento como aplicar una transformación de aproximación de kernel y luego entrenar el modelo. Se probó reducir los features mediante aproximación de kernel a 30 features, pero el tiempo que demora sigue siendo prohibitivo.

Al obtener un accuracy de 74% en train y 65% en test, y por los problemas anteriormente expuestos, se consideró inadecuado para hacer ensambles, en particular el voting classifier.

Random Forest

Para aplicar este modelo primero se utilizó el dataframe ya procesado. Luego se buscó featurizar el dataframe para que devuelva las mejores predicciones posibles. Por ejemplo para la columna que contenía fechas, se la dividió en año, mes y día. Todo esto se hizo en la función featurizer_rf. El paso siguiente fue buscar los mejores hiperparametros de predicción. Al encontrar esos hiperparametros se corrió el random forest classifier y se analizaron sus métricas. Se probó la efectividad también con una confusión matrix. La precisión de este modelo dio un 74,2% mientras que su accuracy fue de 87,1%

XGBoost

Para este modelo se buscó la mejor featurización de datos, como se hizo en RandomForest Classifier. Esto se hizo en la función featurizer_xgb. Luego se buscaron los mejores hiperparametros para la predicción y se corrió el Extreme Gradient Boost con esos parámetros. Se analizó la efectividad con la matriz de confusión y con las métricas dichas anteriormente. La accuracy para el set de testing de XGBoost fue de un 81,2%, mientras que para el set de training fue de un 83,6%. Por lo tanto se ve que no hay overfitting.

Red Neuronal

Para la red neuronal se comenzó por un modelo solo considerando las variables numéricas y realizando one-hot encoding sobre las categóricas. Se generó un modelo de aproximadamente 150 epochs, y se detectó que convergía demasiado rápido (en una sola epoch). Las funciones de Loss y Accuracy no variaban y en algunas pruebas terminaban quedando en nan durante todo el proceso de entrenamiento.
Se probaron capas de dropout, cambiando funciones de optimización, la función de pérdida y nuevas métricas.
El modelo mejoró en cuanto a convergencia más razonable cuando se trató el tema de nans ya que la red neuronal estaba comportándose de manera extraña.
Los resultados obtenidos fueron mejores en comparación con XGBoost y Random Forest, llegando a una accuracy cercana a 90%.

Voting Classifier

Para este modelo se corrieron los 2 modelos anteriores, RandomForest Classifier y XG Boost. Luego se corre el Voting Classifier pasándole los 2 modelos como parámetro y con el parámetro voting siendo ‘hard’. Se analizó si tenía overfitting a través de matrices de confusión y se comparó la accuracy de entrenamiento: 94,5% con la de test: 87,2%. Con lo cual se puede ver que no hubo un overfitting, ya que el testeo es bastante efectivo.

Ensamble de modelos de tipo cascada:

Este tipo de ensamble se utiliza cuando se quiere minimizar el error que se pueda tener al predecir. Ya que implica entrenar el dataframe con un modelo, revisar cómo anduvo con la predicción y todos los que predice mal, meterlos en otro modelo. Así hasta terminar con todos los modelos que se quiera. El orden de los modelos puede variar el resultado ya que entrenaron distintos datos pero, cómo es un orden que tiene cómo primer modelo una red neuronal y luego el resto, la primera aproximación a los datos mal predecidos, van a ser siempre los mismos.

Para poder hacer este ensamble de modelos, no se utilizan funciones de librerías cómo en otros ensambles ya que hay que hacerlo a mano. Cómo primer paso se utilizó una red neuronal para hacer el primer filtro de los datos, al ser un modelo bastante eficiente, limpia una gran cantidad de datos. Con los datos restantes, al no ser muchos (en comparación al original), utilizamos el modelo de XG Boost. Debido a que el XG Boost arrojó una accuracy de al menos 87%, cuando pasemos al tercer modelo (Random Forest) la cantidad de datos es mínima, por lo que también se minimiza la posibilidad de error de la predicción.

Parte 3 - Conclusiones

Para concluir el trabajo aplicado sobre los datos iniciales, podemos decir que de todos los modelos aplicados o estudiados para intentar predecir el 'target', al ser un conjunto enorme de datos, las aplicaciones de dichos modelos son distintas. También, para trabajar con dichos datos se necesita al menos especificaciones de hardware mínimas que sin estas, estas obligado a trabajar con una entorno virtual, ya que el entrenamiento de los datos consume grandes cantidades de recursos. Por último, debido a que este mismo entrenamiento de modelos requería de al menos 15/20 minutos por modelos, resulta ineficiente tratar de entrenar todo el modelo directamente, sino tomar una porción de los datos para entrenar modelos y ver que se hayan entrenado cómo queríamos.

Cómo conclusiones de modelos, se observa que:

Los modelos de Random Forest y XG Boost obtuvieron un buen rendimiento con poco preprocesamiento de datos. Teniendo cómo resultados en accuracy: 0.873% y 0.878% respectivamente, vemos que utilizando primero un entrenamiento para obtener los mejores parámetros o features, y luego utilizando lo mismos para entrenar de vuelta, se obtienen muy buenos resultados.

Los modelos de SVM no son recomendables con datasets con muy alta dimensionalidad como el utilizado. Esto debido a que con una gran cantidad de datos no logramos que converja por lo que podría no terminar nunca de entrenarse. Al haber dos tipos de SVM (lineal y polinómico) supusimos que el lineal tenía que converger sin importar la cantidad de datos, pero al utilizar matrices según cantidad de datos para hacer productos punto, escala demasiado rápido con grandes cantidades de datos. Polinomicamente es peor, ya que el kernel es una matriz $k(x_i, x_j) = (x_i, x_j)^d$, con $d \neq 1$ lo cual resultaba increíblemente largo su entrenamiento. Por lo cual no conseguimos que converja, sin importar lo aplicado en el apartado de SVM más arriba.

El modelo de red neuronal precisó de bastante tweaking, llegando a una precisión alta cercana al 90%. Aparte de tener una precisión muy alta, es un modelo que entrena mucho más rápido que los otros, entonces en cuestiones de eficiencia, si este modelo se entrena bien con las propiedades y parámetros adecuados. Podemos concluir que es el que utilizaremos para este tipo de datasets, ya que al ser muy grandes y con grandes cantidades de datos, una aproximación rápida y eficiente es muy valorable, ya que otros modelos tardan aproximadamente 20 minutos.

Por último, utilizando un ensamble de modelos de tipo ‘cascada’ podemos minimizar casi al máximo el error al predecir posibles valores. Esto debido a que este tipo de ensambles está armado para eso. Con este set de datos y este ensamble, al final quedan muy pocos datos mal predichos. Específicamente la última vez que lo probamos quedaron solamente 13 elementos de una cantidad inicial de 263965. Así que estos 16 elementos conforman solo el 0.00606140965% de los datos, prácticamente despreciable.