Deusto
Facultad de Ingeniería
Ingeniaritzako Fakultatea

Machine Learning
Proyecto Final

Detección de Fraudes en Transacciones con Tarjetas de Crédito

Gorka Celaya
Unai Arévalo

Índice:

1. Introducción

Para la realización de este proyecto se explorarán y desarrollarán múltiples modelos de detección de fraudes bancarios en pagos realizados con tarjeta de crédito. Se busca crear varios modelos y evaluar su rendimiento en base a métricas adecuadas y seleccionar el que mejor funcione de cara a una futura implementación en un entorno real.

Como parte del proceso, se va a realizar un análisis exploratorio de datos (EDA) para estudiar los patrones y características presentes en nuestro dataset. Este paso es fundamental para que podamos adquirir un conocimiento profundo de los datos e identificar posibles problemas de cara al desarrollo de los modelos.

Uno de los principales desafíos será el hecho de que el dataset utilizado está muy desbalanceado, ya que se tienen muchos más datos de transacciones legítimas que de fraudulentas. Como es lógico este desbalance dificultará notablemente la búsqueda de modelos efectivos pero tal y como se explicará más adelante se considerarán diversas técnicas y modelos específicos para hacer frente a ello.

2. Objetivos

Cuando se desarrolla un modelo de detección de fraudes como este, el objetivo es detectar el mayor número de fraudes (maximizando el recall) para proteger a nuestros clientes y evitar que se pierda el dinero involucrado, y hacerlo además minimizando los falsos positivos (transacciones legítimas catalogadas como fraude). Esto último es vital para garantizar una buena experiencia de nuestros clientes ya que una cancelación de pago o bloqueo de tarjeta de crédito en casos en los que no corresponde podría provocar que se sientan descontentos y se acaben pasando a la competencia. En resumen, debemos ser capaces de encontrar un balance entre beneficio para la empresa y buena experiencia del cliente.

De esta forma, debemos considerar las métricas adecuadas para que nuestros modelos sean capaces de cumplir nuestros objetivos. En este caso, se tendrán en cuenta el F1-score y el PR-AUC (Área Bajo la Curva de Precisión-Recall).

- F1-Score: Se trata de la media armónica entre precisión y recall. Como buscamos un balance entre ambas, esta se trata de una métrica ideal para este caso.

- PR-AUC: Mide el área bajo la curva que representa la relación entre precisión y recall a diferentes umbrales de decisión. A diferencia de ROC-AUC se enfoca en la clase positiva (fraudes en este caso), lo que la hace más adecuada para evaluar los resultados en casos con datasets desbalanceados. Permite identificar el umbral óptimo para maximizar el recall sin sacrificar demasiada precisión, asegurando así que se reduzcan los falsos positivos mientras detectamos el mayor número posible de fraudes. En cuanto a los ejes:

    - Eje Y: Precisión de las predicciones para un valor de umbral concreto.
    - Eje X: Recall de las predicciones para el mismo umbral.

3. Análisis preliminar

Realizamos un análisis básico del dataset con el fin de familiarizarnos con su estructura y contenido. En esta primera aproximación se busca poder examinar aspectos fundamentales como el número de filas y columnas (así como el significado de cada una), nombres de las variables, tipos de datos o desbalance de la variable objetivo entre otros. El propósito de este apartado es entender cómo son nuestros datos y poder identificar formas de transformarlos para explotar todo su valor. De esta forma, lo enfocamos como un primer paso para planificar las siguientes etapas del proyecto.

En primer lugar, se debe destacar el hecho de que contamos con casi dos millones de filas. Esto claramente va a condicionar cómo debemos proceder a la hora de diseñar los modelos ya que los tiempos de entrenamiento serán bastante elevados. Para hacer frente a este primer problema será necesaria una estrategia clara que se detalla más adelante.

En cuanto a las columnas, tenemos un total de 23. Centrémonos en ver cuáles son y qué información nos da cada una. A continuación se da una breve explicación del contenido de cada columna:

- Unnamed: 0: Simplemente se trata de un ID secuencial, exactamente igual que el que proporciona pandas al crear el DataFrame, no aporta ninguna información.
- trans_date_trans_time: Fecha y hora en la que se realiza la transacción.
- cc_num: Número de la tarjeta de crédito que realiza el pago.
- merchant: Comercio/Comerciante al que se realiza el pago. Por alguna razón, todos los merchant empiezan por "fraud_", esto lo tenemos en cuenta y lo solucionamos más abajo.
- category: Hace referencia a la categoría del gasto realizado con la tarjeta, se tienen valores como "misc_net" (Miscellaneous Network, es decir, pagos de carácter general realizados en línea), o "grocery_pos" (Grocery Point of Sale, es decir, compras en supermercados realizadas en el propio punto de venta).
- amt: Cuantía de la transacción. Al tratarse de transacciones realizadas en EE.UU. la divisa es el dólar.
- first y last: Nombre y apellido de la persona que realiza la transacción.
- gender: Sexo de la persona que hace el pago.
- state, city, zip y street: Estado, ciudad, código postal y calle asociadas al domicilio del titular de la tarjeta.
- lat, long, merch_lat y merch_long: Latitud y longitud asociadas al cliente y al comercio.
- city_pop: Población de la ciudad del titular.
- job y dob: Empleo y fecha de nacimiento del titular.
- trans_num: Identificación única asociada a cada transacción, una especie de clave primaria.
- unix_time: Timestamp asociado a la transacción en formato unix, es decir, se cuentan los segundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 UTC.
- is_fraud: Variable objetivo que indica si una transacción es fraudulenta (1) o no (0).

Vemos que a pesar de tener un número bastante elevado de columnas, en realidad muchas de ellas dan información redundante o poco útil. Más adelante se eliminarán algunas de ellas y se crearán otras más para poder sacar el máximo partido a los datos.

Para obtener mayor familiaridad con el dataset, analizamos las columnas categóricas y numéricas por separado, identificando los datos más relevantes de cada tipo. En las columnas numéricas, exploramos la cantidad de datos, la media, la desviación estándar y los cuartiles, mientras que, en las categóricas, revisamos la cantidad de datos, los valores únicos, el valor más frecuente y su frecuencia. Este análisis nos permitió comprender mejor los valores presentes en los datos.

4. Creación de variables sintéticas

Tras realizar un primer análisis preliminar del dataset, a pesar de contar con información valiosa, creemos que las variables disponibles no son suficientes para capturar la complejidad del problema, razón por la cual hemos decidido crear una serie

de variables sintéticas para facilitar el EDA posterior y mejorar el rendimiento de los modelos que vamos a diseñar. La creación de estas variables es clave para enriquecer nuestro dataset y sacar el máximo partido a los datos.

En nuestro caso, para crear estas nuevas variables vamos a combinar, transformar o extraer información de las variables que ya existen dependiendo del caso, permitiéndonos obtener una mayor interpretabilidad de los datos.

4.1. Año, mes, día y hora

En primer lugar partimos de la columna "trans_date_trans_time". Como sabemos, esta contiene la fecha y hora de la transacción, información que puede resultar muy relevante de cara a la búsqueda de patrones temporales de los fraudes. Sin embargo, para poder sacar partido a esta información necesitamos poder acceder a cantidades como el día o la hora. Para ello, primero nos fijamos en que ahora mismo las fechas son de tipo string y debemos transformarlas a datetime. Tras hacer esa conversión creamos nuevas columnas: hora, día, mes y año.

4.2. Conversión de mes y día a variables categóricas

En relación al anterior punto, creamos columnas para guardar los nombres de los días de la semana y de los meses para facilitar la interpretación de las visualizaciones de cara al EDA del próximo apartado.

```
[ ] # Diccionario para mapear los valores numéricos a los nombres de los días.
    dias_semana = {
        0: 'Lunes',
        1: 'Martes',
        2: 'Miércoles',
        3: 'Jueves',
        4: 'Viernes',
        5: 'Sábado',
        6: 'Domingo'
    }

    df['nombre_dia'] = df['dia'].map(dias_semana)
```

```
    # Diccionario para mapear los valores numéricos a los nombres de los meses.
    meses_del_año = {
        1: 'Enero',
        2: 'Febrero',
        3: 'Marzo',
        4: 'Abril',
        5: 'Mayo',
        6: 'Junio',
        7: 'Julio',
        8: 'Agosto',
        9: 'Septiembre',
        10: 'Octubre',
        11: 'Noviembre',
        12: 'Diciembre'
    }

    df['nombre_mes'] = df['mes'].map(meses_del_año)
```

Figura 1. Conversión de mes y día.

4.3. Edad

La fecha de nacimiento del titular de la tarjeta también puede llegar a ser útil para tratar de encontrar patrones en los rangos de edades en los que se producen más

fraudes. Sin embargo, una vez más hay que convertir esta columna a fechas haciendo uso del correspondiente formato. Tras ello, se resta la fecha actual a la del momento en el que se realiza la transacción.

4.4. Partes del día

Más allá de que ya contamos con la hora en la que se realizan las transacciones, vamos a crear una nueva columna que haga referencia al momento del día, ya que también podría favorecer la búsqueda de patrones. En este caso dividimos según

- Mañana: Desde las 6h hasta las 13h.
- Tarde: Desde las 13h hasta las 21h.
- Noche: Desde las 21h hasta las 6h del día siguiente.

4.5. Fin de semana

De forma similar, creamos una columna que indica si la transacción se ha producido durante el fin de semana o entre semana.

4.6. Compañía a la que pertenece la tarjeta

El número de la tarjeta de crédito que realiza la transacción podría parecer poco relevante, sin embargo, es posible mapear cada uno de ellos con la compañía que le corresponde (VISA o MASTERCARD por ejemplo) a partir de los primeros dígitos de las mismos. Esto nos permite llevar a cabo un análisis para ver qué tarjetas de crédito son más propensas al fraude. Para realizar el mapeo mencionado, se parte de un dataset que hemos encontrado en el que se tiene información al respecto. Sin embargo, no conseguimos mapear todas las tarjetas de esta forma y tuvimos que realizar una búsqueda exhaustiva en diversas bases de datos en línea para completar esta tarea.

4.7. Distancia entre customer y merchant

La latitud y la longitud tanto de la persona que efectúa el pago como de la que lo recibe pueden resultar de interés. Valores inusuales de dichas cantidades pueden ser claros indicadores de fraude. Sin embargo, utilizar directamente la distancia en kilómetros puede ser más fácil de interpretar para tratar de identificar patrones.

Con este fin, se ha hecho uso de la Fórmula de Haversine, la cual permite hallar la distancia entre dos puntos del globo conocidas sus latitudes y longitudes. De esta forma, la distancia ***d*** se calcula según:

$$\mathrm{d} = 2\mathrm{R} \times arctan2(\sqrt{a}, \sqrt{1-a})$$

donde ***R*** es el radio de la tierra en km, ***arctan2*** es la función arcotangente de dos parámetros y

$$a = sin^2(\frac{\Delta lat}{2}) + cos(lat_1) \times cos(lat_2) \times sin^2(\frac{\Delta long}{2})$$

4.8. Historial de fraudes de un cliente

Dado que contamos con la fecha en la que se realizan las transacciones, nos es posible crear un historial de fraudes previos que se hayan cometido con una tarjeta de crédito concreta. Esto puede ser muy útil ya que permitirá a los modelos tener contexto de transacciones realizadas con anterioridad.

Para crear esta columna debemos ordenar y agrupar por número de tarjeta y fecha, de esta forma, si hay fraudes previos a esta nueva columna le corresponderá el valor 1 y si no el 0.

4.9. Historial de fraudes de un comercio

Siguiendo la misma lógica que para el caso anterior, creamos ahora un historial de fraudes previos para los comercios que reciben los pagos. La forma de crear esta columna es idéntica a la anterior.

4.10. Modificación de los merchant

Por alguna razón, todos los merchant del dataset tienen el prefijo 'fraud_'. Por ello, decidimos eliminar ese prefijo.

4.11. Modificación de las categorías

Se ha traducido la columna 'category' ya que sus valores se encuentran en inglés y con palabras abreviadas, con el fin de que la interpretabilidad durante el EDA sea más clara.

4.12. Eliminación de columnas innecesarias

Dado que se han creado varias columnas, optamos por eliminar algunas de las que no vamos a utilizar en el EDA que vamos a realizar. No obstante, es importante recalcar que antes de pasar a entrenar los modelos se hará otro análisis de qué columnas merece la pena mantener y cuáles no. En este caso simplemente eliminamos las columnas que no aportan valor como el identificador, el nombre, el apellido, la dirección y el número de transacción entre otros.

5. EDA

En este apartado se va a realizar un EDA para tratar de detectar patrones y relaciones entre variables. El objetivo es entender aún mejor nuestros datos y ver qué variables pueden resultar de interés de cara al entrenamiento de los modelos.

5.1. Distribución de la variable objetivo

Tras el análisis preliminar hemos visto que existe un desbalance entre los casos fraudulentos y legítimos. A continuación, en la figura 2, vemos cómo de grande es la diferencia.

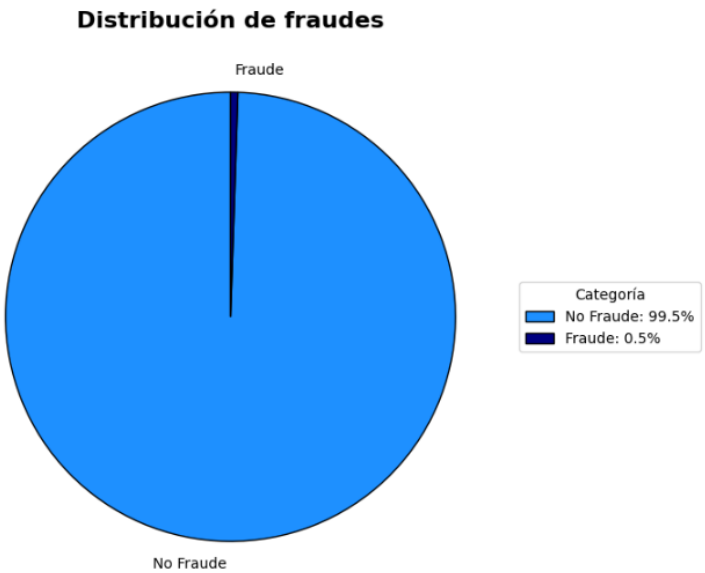

Figura 2. Gráfico de la distribución de fraudes.

Como adelantamos, solo un 0.5% de los datos son transacciones fraudulentas, por lo que más adelante se deberán aplicar diversas técnicas para hacer frente al desbalance y que nuestros modelos sean capaces de detectar los fraudes correctamente.

5.2. Importes de transacciones fraudulentas o no fraudulentas

Queremos visualizar si el importe de las transacciones guarda relación con el hecho de que se produzca un fraude. Para ello, mostramos en la figura 3 la cuantía de las transacciones tanto fraudulentas como legítimas. Se ha usado un histograma para cada caso, donde en el eje Y se muestra el porcentaje dentro de cada bin del total de fraudes o no fraudes según el caso.

Es importante recalcar que se ha limitado el valor de las transacciones a un máximo de 1.400 dólares ya que para el caso de las transacciones legítimas había unos cuantos outliers correspondientes a pagos de hasta cerca de 30.000 dólares que dificultan la interpretabilidad de la figura.

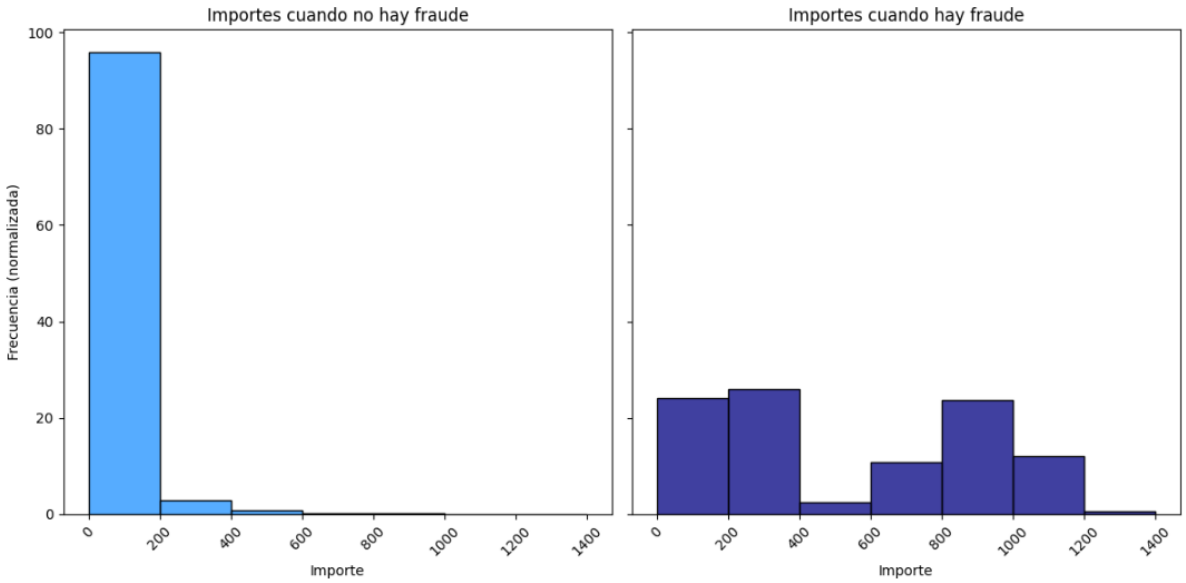

Figura 3. Frecuencia de importes.

En la figura se diferencia claramente como los importes de transacciones fraudulentas tienden a ser mayores. La distribución en estos casos está mucho más dispersa, a diferencia de las transacciones legales donde la inmensa mayoría de los pagos son inferiores a 200 dólares.

También podemos visualizar esto mismo haciendo uso de violin plots para apreciar mejor la distribución de los datos.

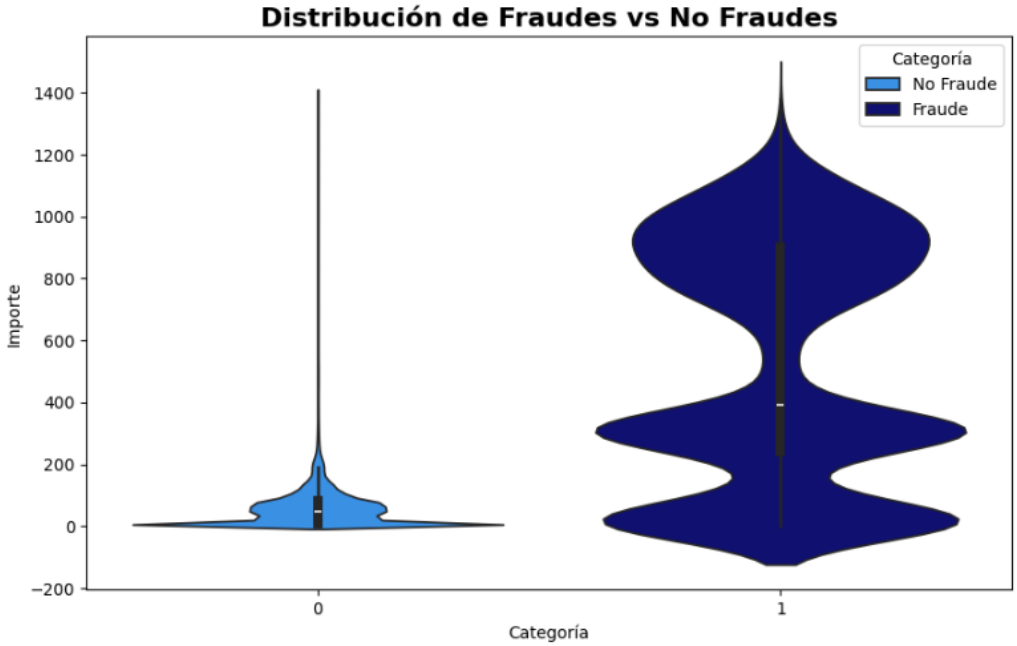

Figura 4. Distribución de importes de transacciones.

Aquí se aprecia mejor la diferencia entre las distribuciones. De hecho, se ve que la gran mayoría de transacciones no fraudulentas son de unos pocos dólares. En las fraudulentas en cambio, se ven tres picos, el primero también para transacciones de pocos dólares, el segundo alrededor de los 400 dólares y el último cerca de los 1.000.

5.3. Importancia de la edad

Vamos a comprobar cómo varía la cantidad de fraudes/transacciones legales en función de la edad de los titulares de las tarjetas. El objetivo principal de este análisis es explorar si la edad del titular está correlacionada con la probabilidad de que una transacción sea fraudulenta. Para ello, se muestra una comparación de la distribución de edades de los titulares de tarjetas de crédito en transacciones fraudulentas frente a transacciones no fraudulentas, utilizando una gráfica de densidad estimada. Dicha densidad no representa un número absoluto de personas, sino la probabilidad relativa de encontrar clientes en cada rango de edad.

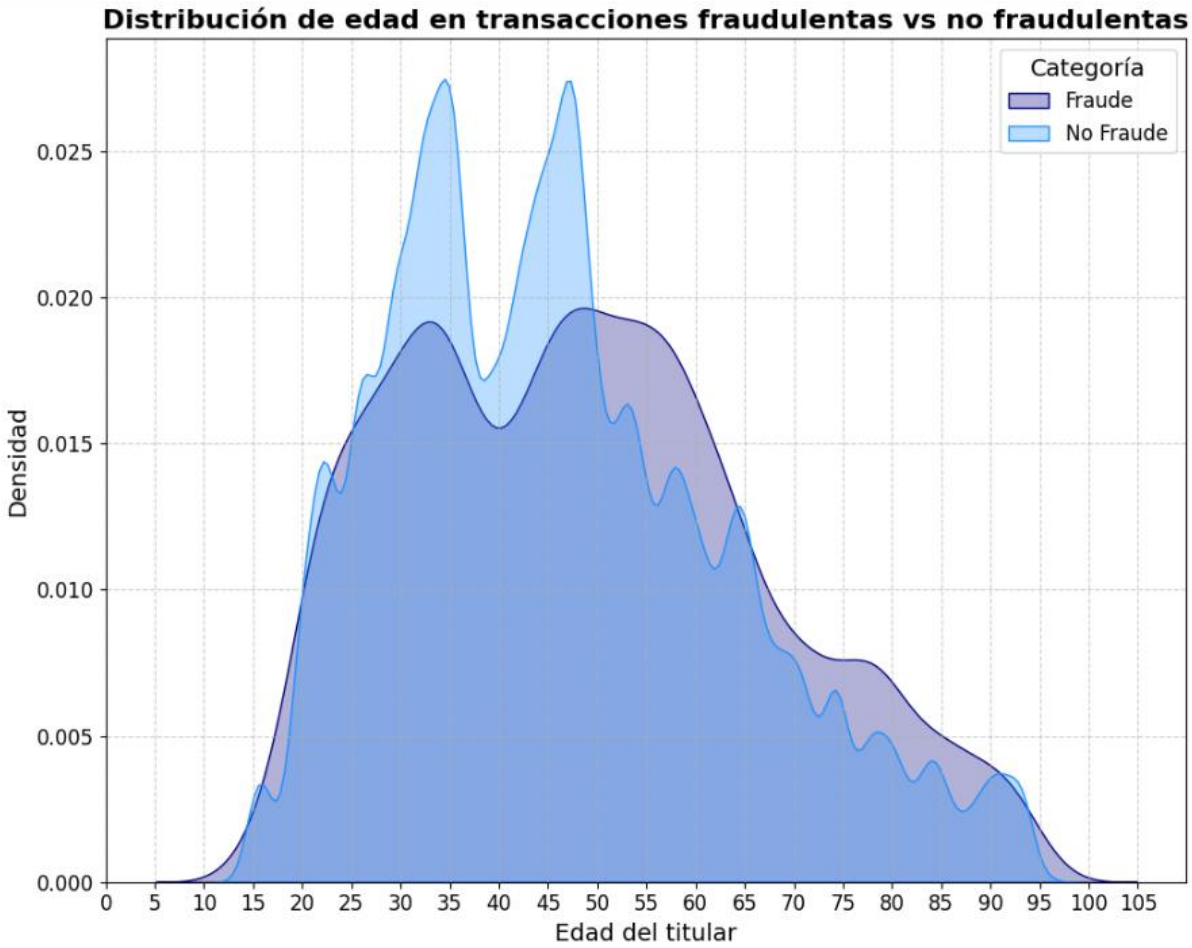

Figura 5. Gráfico de densidad para la edad de los clientes.

De esta gráfica se puede extraer información bastante interesante. Por un lado, la mayoría de transacciones las realizan personas de alrededor de 35 o 45 años, independientemente de si son fraudulentas o no. Para personas de alrededor de 40 años se produce una bajada importante de transacciones.

Además, la probabilidad relativa de fraude es mayor para personas de más de 50 años. Aunque esto no indica el número absoluto de fraudes, sí muestra que entre los casos registrados, los titulares de más de 50 años están representados con mayor frecuencia en transacciones fraudulentas. Esto podría deberse a que:

- 1. Las personas mayores pueden ser más susceptibles a engaños o fraudes digitales debido a una menor familiaridad con las nuevas tecnologías.
- 2. Los atacantes podrían orientar sus estrategias hacia este grupo demográfico por considerarlo más vulnerable.

5.4. Balance de fraudes por categoría

Para ver si la categoría de gasto a la que corresponden las transacciones es de interés, mostramos un bar plot en el que se muestra el porcentaje de fraudes dentro de cada categoría. Además, la línea discontinua que se ve, representa la media de porcentaje de fraude teniendo en cuenta las 14 categorías presentadas.

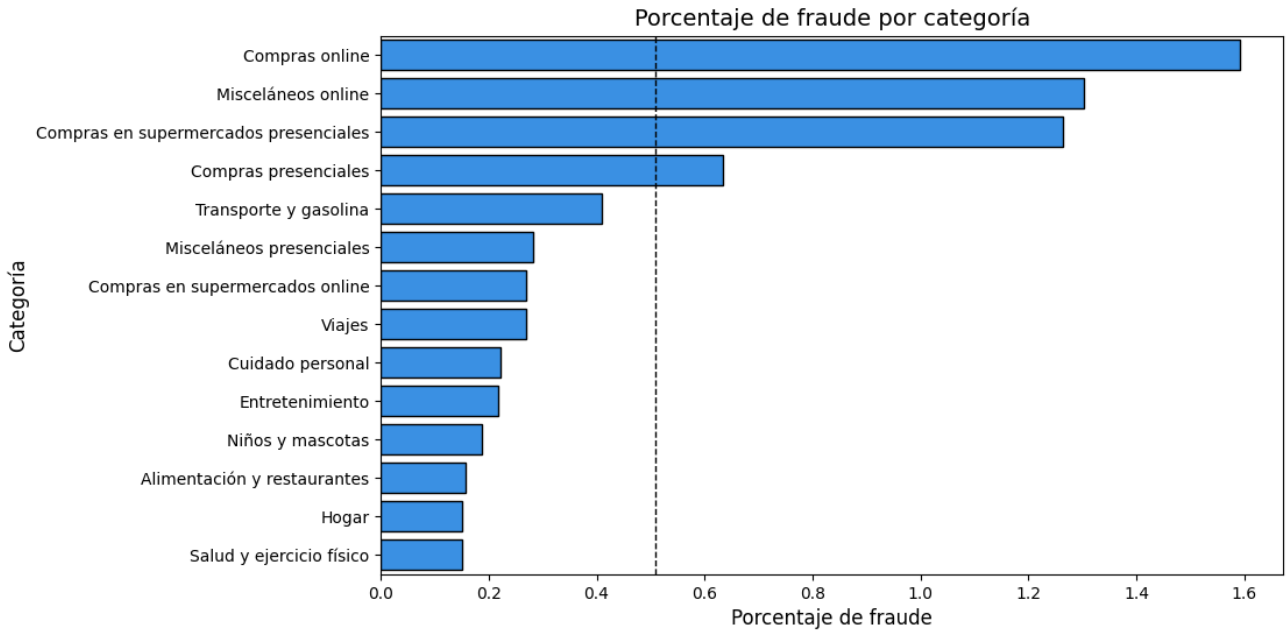

Figura 6. Porcentaje de fraude por categoría.

Se puede ver que las categorías más propensas al fraude por un margen bastante amplio son compras online en general: ropa, electrónica, etc, misceláneos y compras en supermercados de forma presencial. Por otro lado, las categorías menos comunes son salud y ejercicio físico, hogar y alimentación y restaurantes.

Una vez visualizado el porcentaje de fraude de cada categoría, vemos conveniente mostrar el coste promedio por cada una.

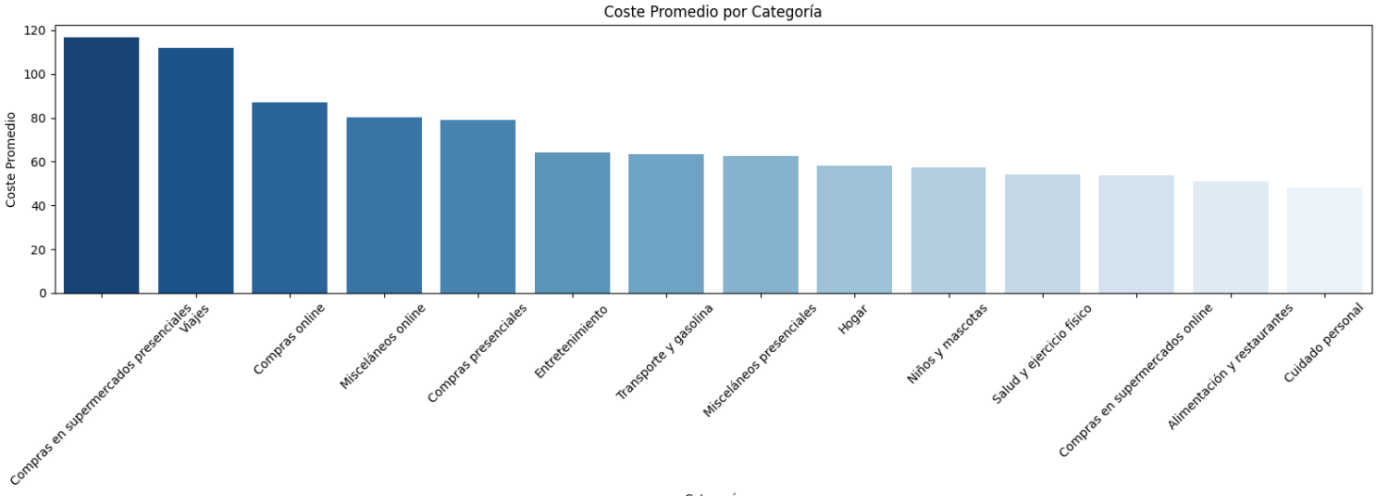

Figura 7. Valor promedio de transacciones por categoría.

Lo interesante de esta gráfica es ver que las categorías con mayor proporción de fraudes son también en las que hay, de media, transacciones de mayor valor.

5.5. Distribución de fraudes en función del tiempo

Comenzamos ahora a fijarnos en datos temporales para tratar de identificar patrones. En primer lugar, vemos el número de fraudes cometidos en las fechas de las que disponemos para tratar de ver si hay algún momento concreto en el que por alguna razón sube el número de los mismos.

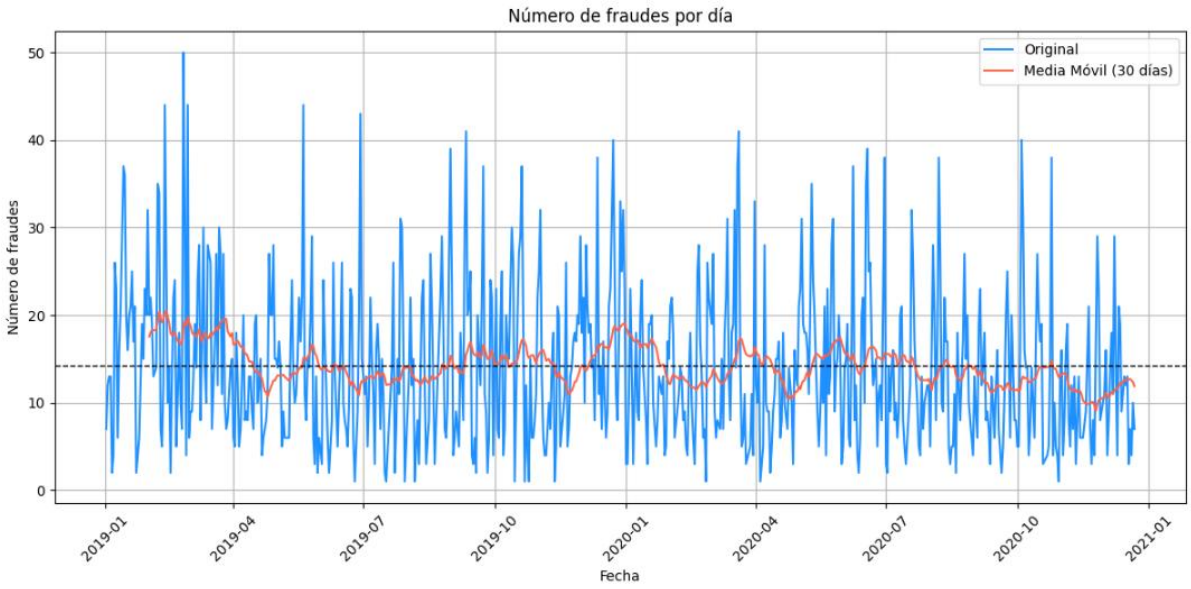

Figura 8. Número de fraudes por día.

Tal y como se puede ver en la figura 8, es complicado sacar conclusiones a partir de la figura dada la cantidad de ruido que hay. Es por ello que se ha decidido incluir una media móvil (color naranja), gracias a la cual vemos la cantidad de fraudes calculada cada 30 días. De nuevo en este caso es complicado sacar conclusiones claras pero sí que destacan dos cuestiones:

- La cantidad de fraudes oscila mucho de un día para otro, por ejemplo, el pico máximo se produjo a principios de 2019 con 50 fraudes pero al día siguiente se produjeron menos de 10. Sin embargo, al promediar cada 30 días vemos que el número de fraudes siempre está entre los 10 y 20.
- Si nos fijamos en la media móvil veremos que los máximos parecen producirse a principios de año, mientras que a finales, la cantidad de fraudes parece decaer ligeramente (conviene fijarse en la línea discontinua mostrada que representa el promedio de fraudes diarios para todo nuestro conjunto de datos).

Veamos ahora qué porcentaje de los fraudes totales se produce cada año.

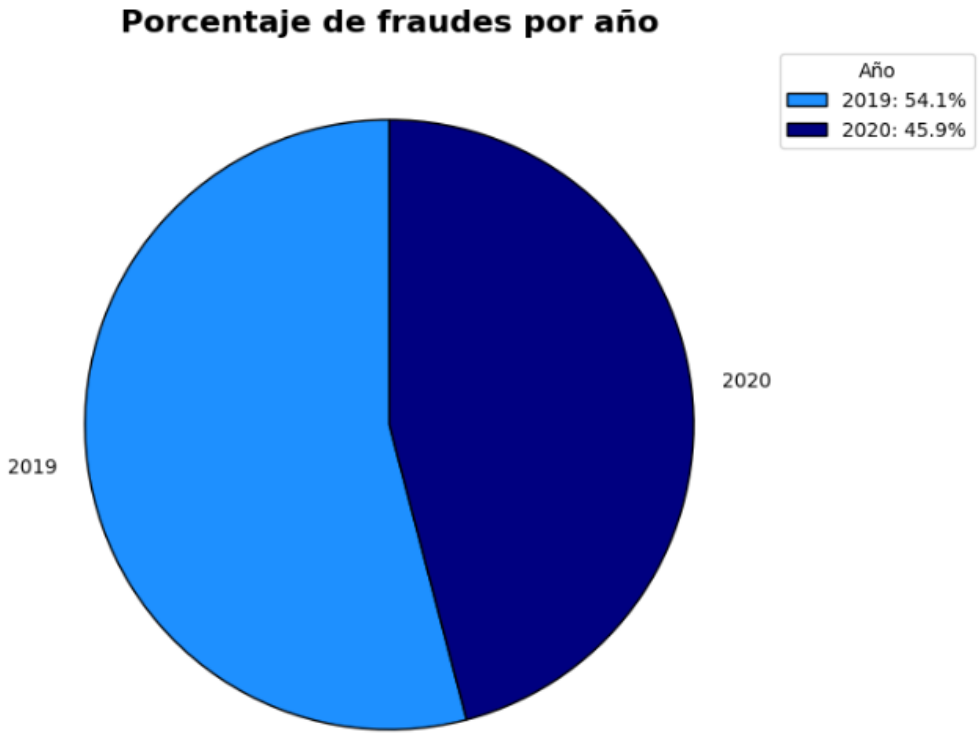

Figura 9. Porcentaje de fraudes por año.

Vemos en la figura que la cantidad de fraudes en 2019 (54.1%) es algo mayor que en 2020 (45.9%).

Para seguir con nuestro análisis en función del tiempo, nos fijamos ahora en el porcentaje de fraudes por cada mes.

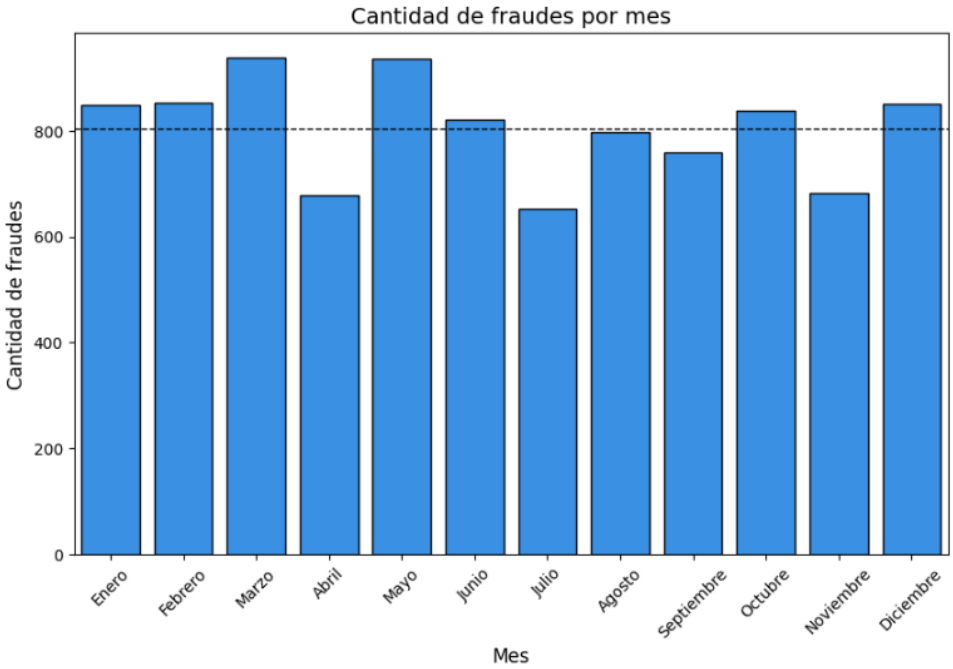

Figura 10. Cantidad de fraudes por mes.

Vemos que

- Los meses con mayor cantidad de fraudes son marzo y mayo. En abril, julio y noviembre en cambio se tienen los mínimos.
- Teniendo en cuenta los datos de 2019 y 2020 que tenemos, se produjeron unos 800 fraudes de media por cada mes.
- Se cometen más fraudes durante la primera mitad del año que durante la segunda.

Veamos si existen diferencias en función del día de la semana. De nuevo hacemos uso de un gráfico de barras para estudiar la cantidad de fraudes.

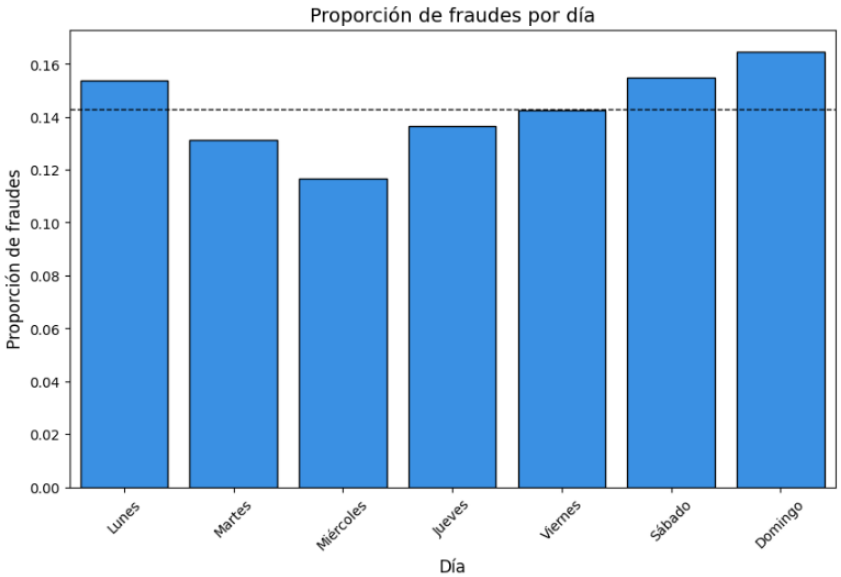

Figura 11. Gráfico con la proporción de fraudes por día.

Vemos que los días con mayor cantidad de fraudes son los sábados, domingos y lunes. El resto de la semana se produce una bajada bastante importante. Parece ser que una cantidad bastante importante de fraudes se produce durante los fines de semana.

Centrémonos ahora en las horas a las que se producen las transacciones. Para ello, de nuevo hacemos uso de un gráfico de barras en el que por cada hora se tiene la frecuencia relativa de cada tipo de transacción.

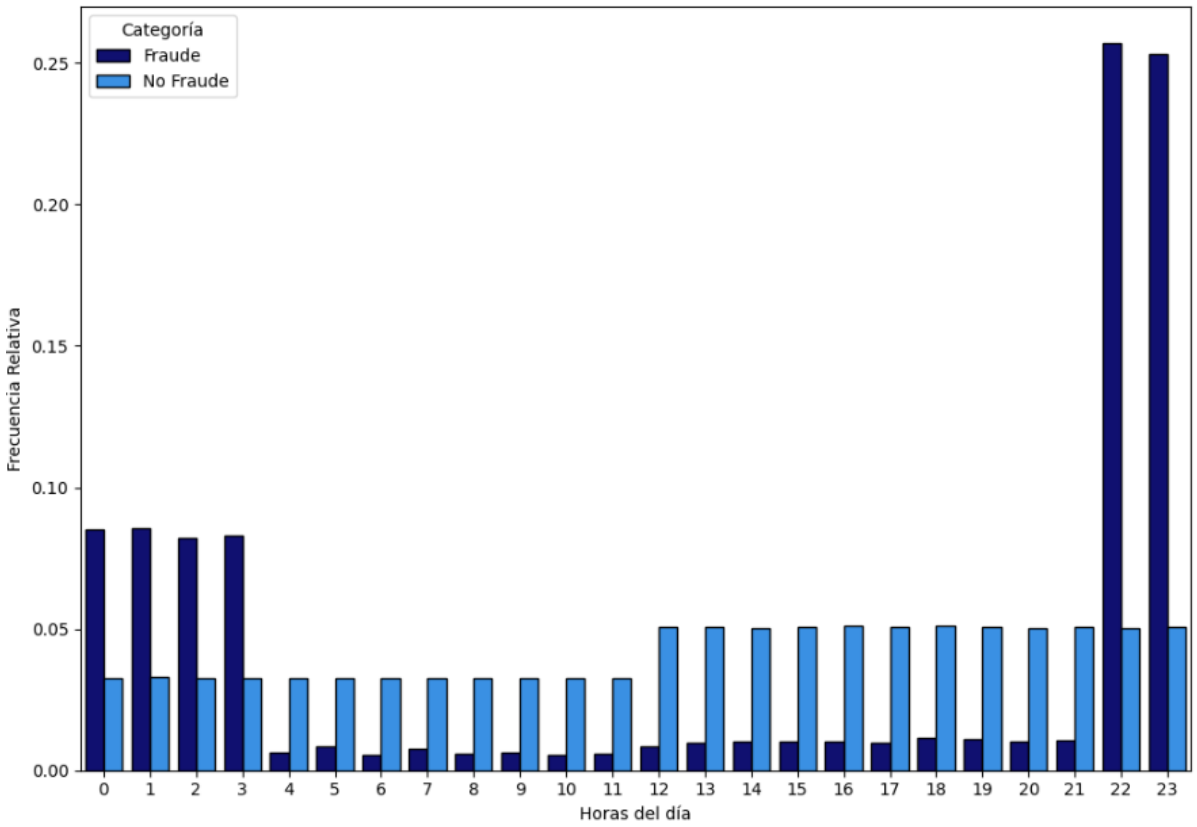

Figura 12. Frecuencia relativa de fraudes/transacciones legítimas por hora.

En la figura 12 se observan patrones muy diferenciados:

- Para las transacciones legítimas se tiene una distribución que solo varía a partir del mediodía, donde se ve cierto aumento en el número de transacciones.
- Para los fraudes, se aprecia que se producen muy pocos durante la mañana y la tarde. Sin embargo, durante la noche se disparan, sobre todo a las 22h y las 23h, en las que se registra la mitad del total de los fraudes.

Por último, visualicemos más detenidamente la proporción de fraudes que se realiza en cada momento del día (mañana, tarde, noche).

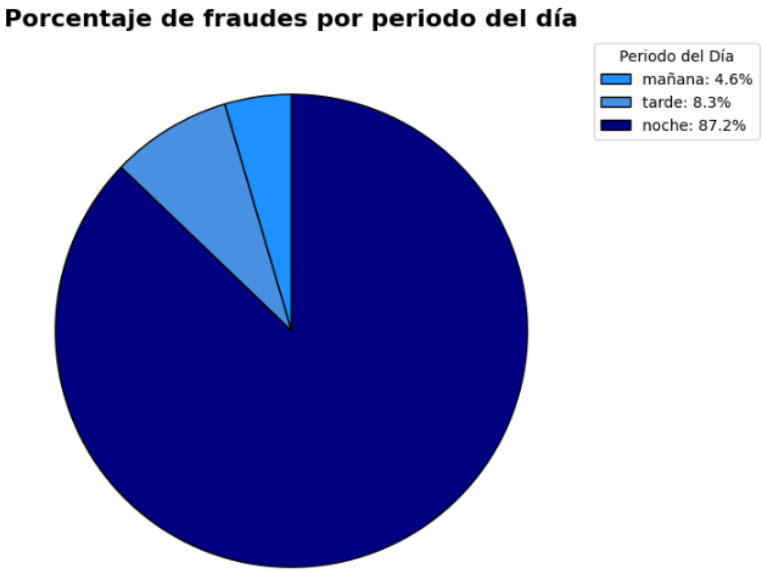

Figura 13. Porcentaje de fraudes por periodo del día.

Volvemos a visualizar como la gran mayoría de fraudes se cometen durante la noche.

5.6. Tarjetas con mayor probabilidad de fraude

Queremos visualizar con qué tarjetas se producen más fraudes. Para ello hacemos uso de un bar plot horizontal, mostrando también el valor promedio del porcentaje de fraudes para todo el dataset.

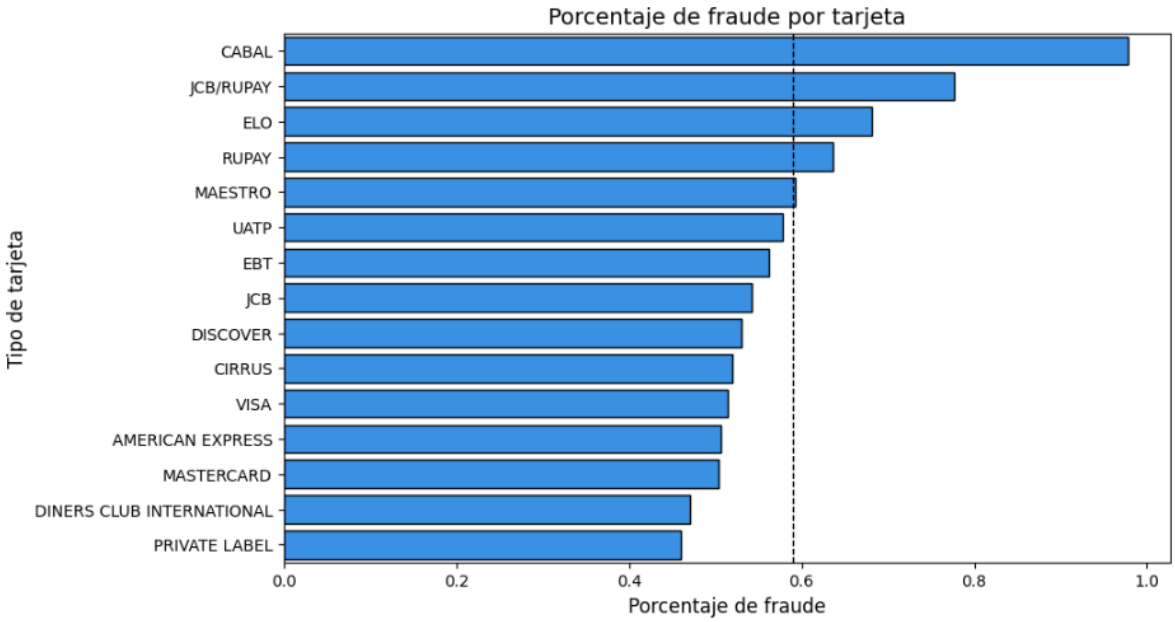

Figura 14. Porcentaje de fraudes por compañía de la tarjeta.

Vemos que:

- CABAL es el tipo de tarjeta que tiene el mayor porcentaje de los fraudes de nuestro dataset.
- JCB/RUPAY, ELO RUPAY y MAESTRO son los demás tipos de tarjeta que también superan la media de porcentaje de fraudes.
- PRIVATE LABEL y DINERS CLUB INTERNATIONAL son los tipos de tarjeta que menos porcentaje de fraude tienen.

5.7. Comercios con más fraudes

Mostramos algunos de los comercios asociados a una mayor cantidad de fraudes.

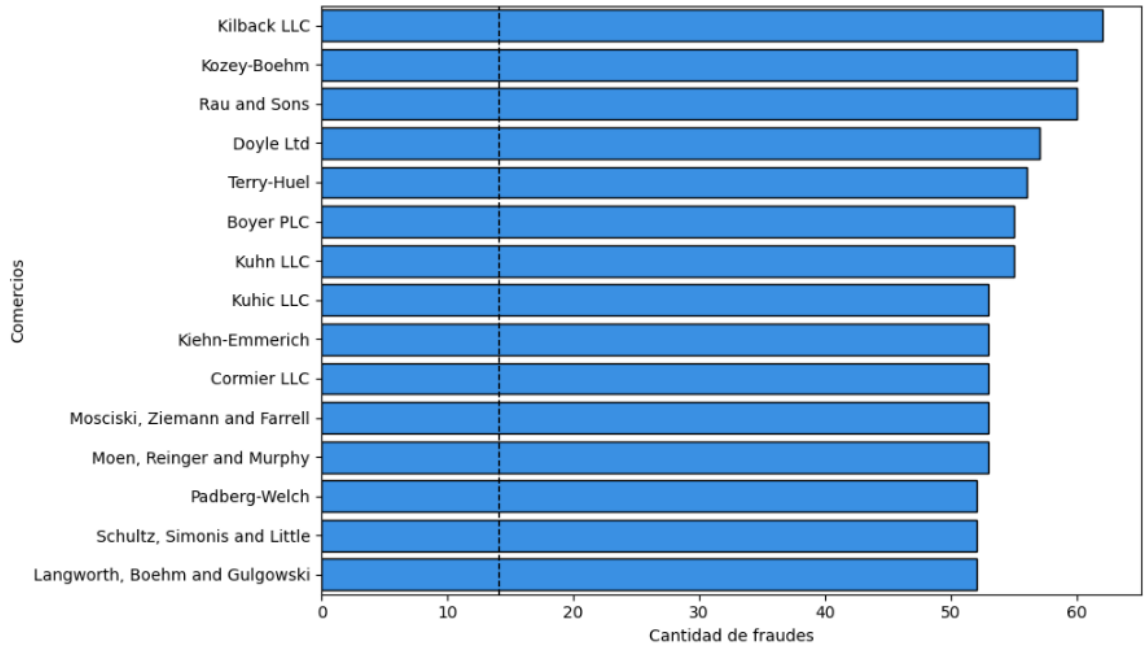

Figura 15. Top 15 de comercios con mayor cantidad de fraudes.

En la gráfica se muestran los 15 comercios involucrados en más fraudes. Vemos que todos ellos tienen un número de fraudes mucho mayor que la media, que es de alrededor de 15.

5.8. Fraudes por ubicación geográfica

Dado que contamos con información sobre la ubicación geográfica de los clientes, creemos que puede resultar interesante crear una visualización en la que podamos ver desde dónde se producen más transacciones/fraudes.

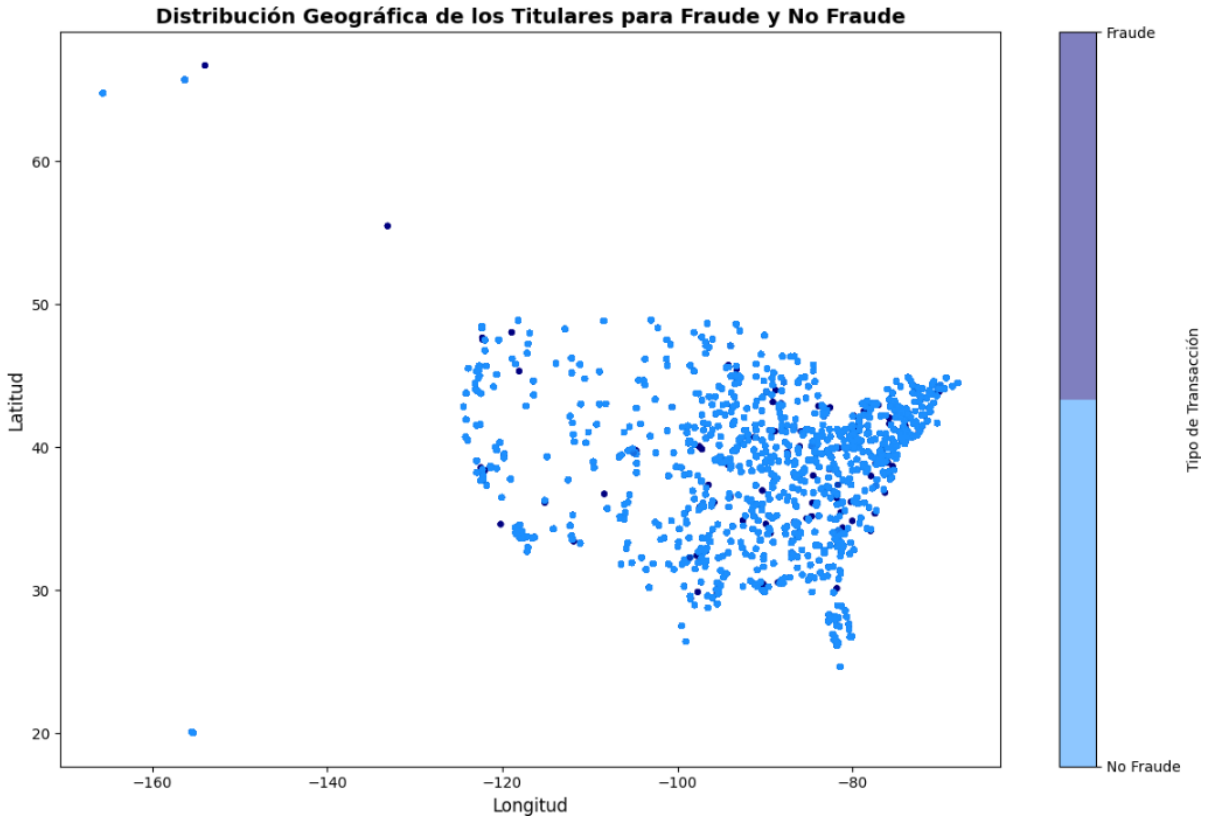

Figura 16. Gráfico con la distribución geográfica de los titulares.

En la figura anterior, se aprecia la distribución geográfica de transacciones en el territorio de EE.UU. Se ve que hay una mayor densidad de transacciones (y también de fraudes) en la mitad este del país.

5.9. Importancia de la distancia

Para ver si la distancia entre cliente y comercio, calculada como nueva variable en el apartado anterior es relevante, visualizamos las distribuciones correspondientes por medio de un 'violin plot'.

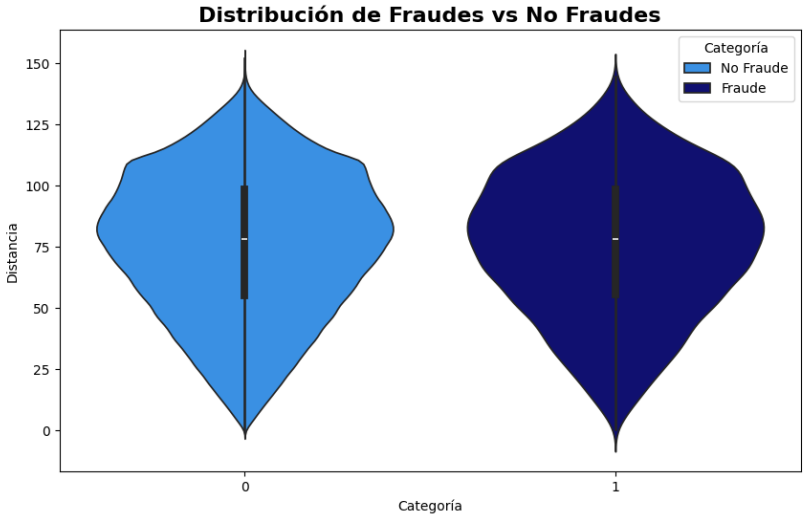

Figura 17. Distribuciones de la distancia cliente-comercio según tipo de transacción.

No parecen existir diferencias entre las distribuciones de distancias para fraudes/no fraudes. Por tanto, es posible que esta variable no resulte de interés de cara al modelo de predicción.

5.10. Historial de fraudes de clientes y comercios

Anteriormente, hemos creado variables que nos indican si a la fecha en la que se realiza una transacción el cliente o el comercio se han visto involucrados en fraudes. Pretendemos ver si estas variables resultan relevantes para detectar fraudes.

Figura 18. Proporción de fraudes de clientes reincidentes.

Vemos que en el 90.8% de los fraudes, están involucrados números de tarjetas que ya habían cometido fraudes anteriormente. Esto resulta de gran interés ya que el hecho de contar con esta variable podría ayudarnos a evitar un gran número de fraudes.

Hacemos lo mismo para la variable asociada a los comercios.

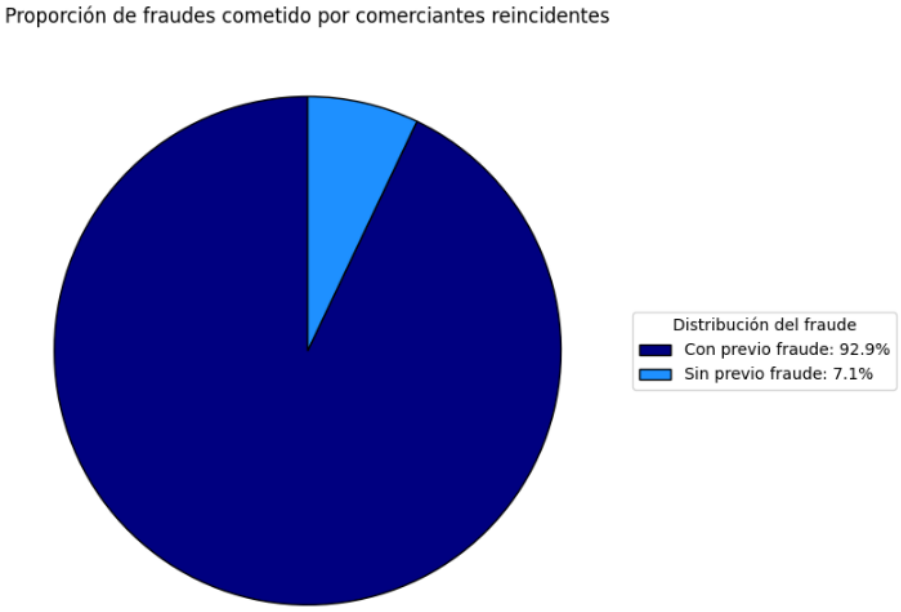

Figura 19. Proporción de fraudes de comercios reincidentes.

En este caso, vemos que en el 92.9% de los casos se tienen comercios que ya se habían visto involucrados anteriormente en transacciones fraudulentas. De esta forma, nos preguntamos finalmente si el porcentaje de fraudes que podríamos detectar aumenta si tenemos en cuenta las dos variables anteriores a la vez.

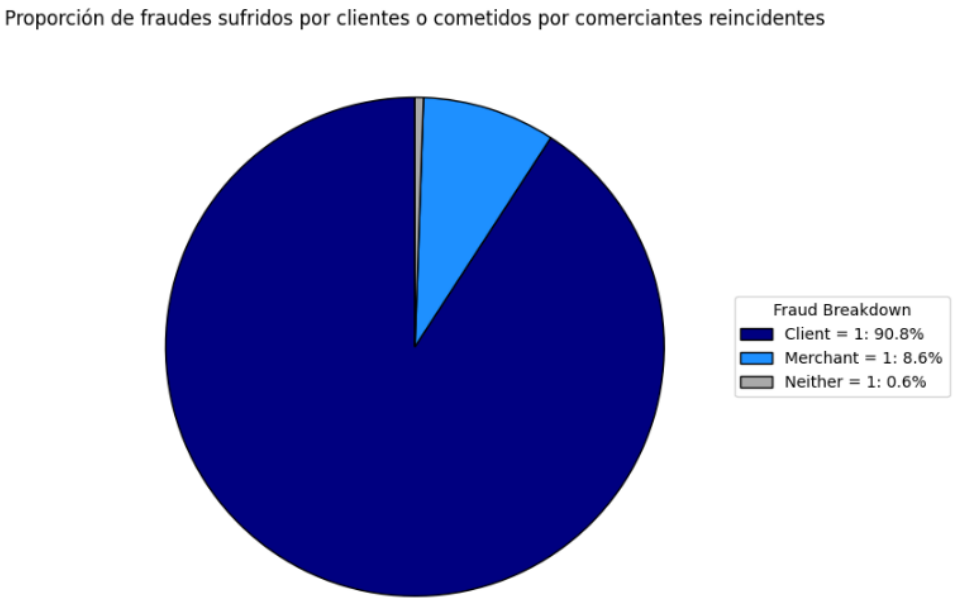

Figura 20. Proporción de fraudes de clientes y comercios reincidentes.

Vemos que de todos los fraudes el 99.4% es cometido o sufrido por un cliente o comerciante que ya se había visto envuelto en otro fraude anteriormente. La conclusión es clara, poniendo el foco en ellos, seremos capaces de detectar muchos más fraudes.

6. Encoding y escalado

Tras haber realizado el EDA y mostrar en gráficas la relación entre las variables del dataset vamos a quedarnos con las más útiles y prepararlas para el entrenamiento de modelos.

6.1. Eliminación de columnas no útiles

Eliminamos las columnas que no son útiles de cara a entrenar modelos.

- cc_num: No nos interesa mantenerla, ya que al contar con el historial de fraudes de cada cliente sería redundante.
- nombre_dia y nombre_mes: Estas son variables que creamos únicamente de cara a las visualizaciones del apartado 5.
- city y city_pop: El número de ciudades era tan elevado y las transacciones estaban tan repartidas que no tendría sentido considerarlo.
- job: De nuevo había demasiados trabajos como para poder extraer información valiosa o patrones significativos.
- año: Al dividir entre train y test en función del momento de la transacción, las del 2020 estarían divididas y no tendría sentido esperar que esta columna nos fuera a servir.
- dist_to_merch: A pesar de pensar que esta podría ser una variable de interés a la hora de crear variables sintéticas, vimos que no había diferencias en las distancia entre cliente y comercio en los casos de fraude y no fraude.

Las columnas que seguimos utilizando son:

1. merchant
2. category
3. amt
4. gender
5. state
6. lat
7. long
8. merch_lat
9. merch_long
10. is_fraud
11. hora
12. dia
13. mes
14. fecha
15. edad
16. periodo_dia
17. tipo_dia
18. card_type
19. previous_fraud_client
20. previous_fraud_merchant

6.2. Encoding de columnas categóricas

Pasamos ahora a codificar las columnas categóricas. Dependiendo de las características de cada una, vamos a utilizar una estrategia u otra. Por ejemplo, para columnas con pocas categorías: 'gender' y 'periodo_día', haremos uso de One Hot Encoding.

Antes de continuar con el encoding, separamos en conjuntos de train y de test para que no se produzca data leakage. Dado que estamos teniendo en cuenta datos históricos de clientes, optamos por dividir los datos en función de la fecha. Para ello, definimos una fecha límite (15/08/2020), tal que datos con fechas anteriores pertenezcan al conjunto de train y el resto al de test. Esto lo hacemos de modo que se reserve alrededor del 20% del total de los datos para test.

```
# Nos aseguramos de que la columna 'fecha' esté en formato datetime.
df['fecha'] = pd.to_datetime(df['fecha'])

# Ordenamos el DataFrame por fecha.
df_sorted = df.sort_values(by='fecha')

# Definir la fecha límite.
fecha_limite = '2020-08-15'

# Dividir el DataFrame en entrenamiento y prueba.
train_df = df_sorted[df_sorted['fecha'] < fecha_limite]
test_df = df_sorted[df_sorted['fecha'] >= fecha_limite]

# Calculamos el total de muestras.
total_samples = df.shape[0]

# Calculamos los tamaños de cada conjunto.
train_size = train_df.shape[0]
test_size = test_df.shape[0]

# Calculamos porcentajes.
train_percentage = (train_size / total_samples) * 100
test_percentage = (test_size / total_samples) * 100

# Imprimimos resultados.
print(f"Conjunto de entrenamiento: {train_size} muestras ({train_percentage:.2f}%)")
print(f"Conjunto de prueba: {test_size} muestras ({test_percentage:.2f}%)")
```

```
Conjunto de entrenamiento: 1452377 muestras (78.41%)
Conjunto de prueba: 400017 muestras (21.59%)
```

Figura 21. División del dataset en train y test.

Ahora que hemos dividido en dos conjuntos de datos, eliminamos la columna de fecha y desordenamos las filas ya que hasta ahora estaban ordenadas por fecha. Para las demás variables categóricas, vamos a realizar un target encoding, ya que tienen muchas categorías.

El target encoding es una técnica de preprocesamiento utilizada para convertir variables categóricas en valores numéricos, basándose en la relación estadística entre cada categoría y la variable objetivo. En lugar de crear múltiples columnas binarias, el target encoding asigna a cada categoría un único valor numérico calculado como la media de la variable target asociada a esa categoría.

```
# Definir las columnas categóricas para aplicar target encoding y el target.
categorical_columns = ['card_type', 'merchant', 'category', 'state']
target_column = 'is_fraud'

for col in categorical_columns:
    # Calcular la media del target por categoría en el conjunto de entrenamiento.
    target_mean = train_df.groupby(col)[target_column].mean()

    # Reemplazar las categorías con la media correspondiente en el conjunto de entrenamiento.
    train_df[col] = train_df[col].map(target_mean)

    # Aplicar la misma transformación al conjunto de prueba.
    test_df[col] = test_df[col].map(target_mean)
```

Figura 22. Target encoding.

6.3. Matriz de correlación

Ahora que nuestros datos son de tipo numérico, podemos mostrar la matriz de correlación.

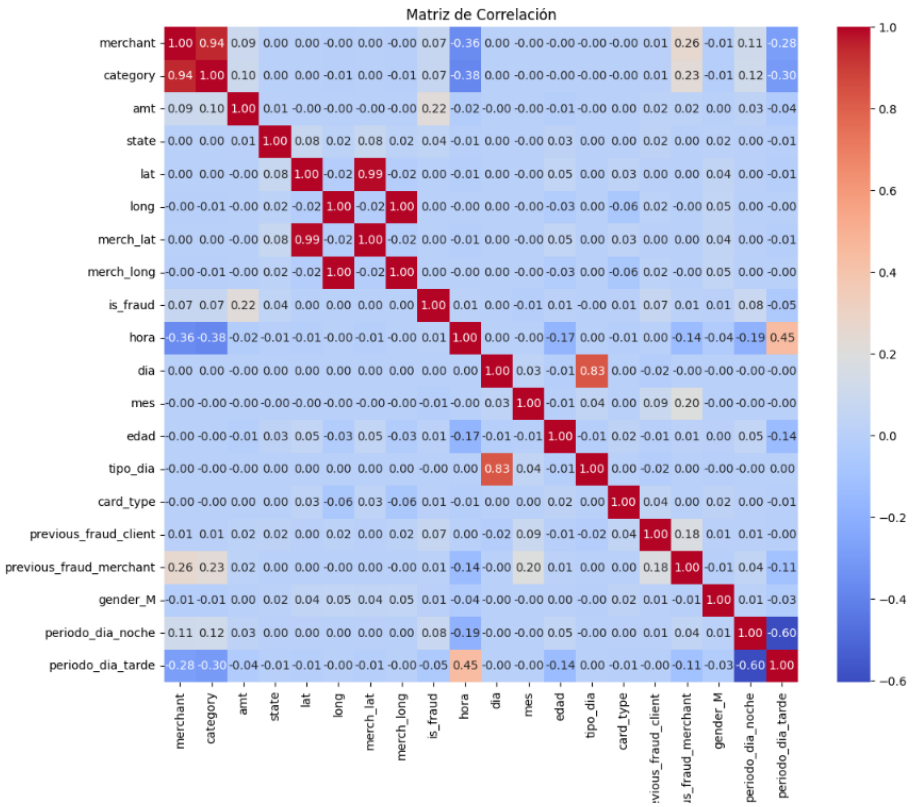

Figura 23. Matriz de correlación.

Eliminamos las columnas con alta correlación debido a que solo introducimos datos 'repetidos' a los modelos que aumentan los tiempos de entrenamiento. Las columnas eliminadas son: 'merchant', 'merch_lat', 'merch_long' y 'tipo_dia'.

6.4. Escalado de datos

Por último llegamos al escalado de datos. Por ahora vamos a utilizar un StandardScaler pero a la hora de entrenar más modelos deberemos modificar esto, ya que el escalado depende del modelo a utilizar.

```python
from sklearn.preprocessing import StandardScaler

# Dividimos según variables.
x_train = train_df.drop('is_fraud', axis = 1)
y_train = train_df['is_fraud']

x_test = test_df.drop('is_fraud', axis = 1)
y_test = test_df['is_fraud']

# Configurar el escalador
scaler = StandardScaler()

# Obtener los datos escalados.
X_train_scaled = scaler.fit_transform(x_train)
X_test_scaled = scaler.transform(x_test)
```

Figura 24. Escalado de datos.

7. Modelos

Tras procesar los datos ya estamos listos para empezar a entrenar los modelos. En este apartado vamos a considerar modelos basados en árboles. Dado el gran volumen de datos y los tiempos de entrenamiento que vamos a tener, hemos decidido escoger cuatro modelos que suelen funcionar bien en casos como el nuestro, concretamente Decision Tree, Random Forest, XGBoost y Balanced Random Forest. A la hora de entrenarlos vamos a seguir un enfoque incremental en cuanto a la complejidad de modelos, respondiendo siempre a puntos de mejora que vayamos identificando.

En los siguientes entrenamientos vamos a mostrar diferentes métricas que miden el desempeño de los modelos. Las métricas mostradas y sus cálculos son las siguientes:

- Accuracy: Proporción de predicciones correctas, tanto positivas como negativas, sobre el total de casos.

$$\frac{\text{Verdaderos Positivos} + \text{Verdaderos Negativos}}{\text{Total de Ejemplos}}$$

- Precisión: Fracción de casos positivos predichos que realmente son positivos.

$$\frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Positivos}}$$

- Recall: Proporción de casos positivos reales que el modelo identifica correctamente.

$$\frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Negativos}}$$

- F1 Score: Media armónica de la precisión y la sensibilidad, equilibrando ambas métricas.

$$2 \cdot \frac{\text{Precisión} \cdot \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}}$$

- ROC AUC: Área bajo la curva ROC. Capacidad del modelo para separar clases en todos los posibles umbrales
- MCC: Coeficiente de correlación de Matthews. Evalúa la calidad de la clasificación teniendo en cuenta todas las combinaciones de resultados

$$\mathrm{MCC} = \frac{(\mathrm{VP} \cdot \mathrm{VN}) - (\mathrm{FP} \cdot \mathrm{FN})}{\sqrt{(\mathrm{VP} + \mathrm{FP})(\mathrm{VP} + \mathrm{FN})(\mathrm{VN} + \mathrm{FP})(\mathrm{VN} + \mathrm{FN})}}$$

- PR AUC: Área bajo la curva de precisión-recall. Se calcula usando la relación entre precisión y sensibilidad en diferentes umbrales.

7.1. Modelos Base

En primer lugar, vamos a utilizar una única combinación de hiperparámetros para cada modelo y así ver en qué punto estamos y a partir de ahí tratar de mejorar los resultados. Al finalizar el entrenamiento de cada modelo, vamos a guardarlo en formato pickle. Esto lo hacemos para evitar que al desconectar el entorno de trabajo de colab se pierdan los avances realizados ya que tendríamos una especie de "checkpoint" cada vez que finaliza un entrenamiento. Para casos como el nuestro resulta especialmente útil dados los elevados tiempos de entrenamiento.

Se han creado una serie de funciones para organizar mejor el código y los resultados. A continuación se detalla el funcionamiento de cada una.

- get_metrics: Devuelve un DataFrame con las métricas de interés de los modelos que queramos.
  Entradas:
  - models: Diccionario con nombres de modelos y modelos considerados.
  - x_test e y_test: Datos de test considerados.

  Salidas:
  - DataFrame con valores de precisión, recall y F1 de cada modelo.

```
def get_metrics(models, x_test, y_test):
    results = []

    for name, model in models.items():
        y_pred = model.predict(x_test)

        precision = precision_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        results.append({
```

```
            "Model": name,
            "Precision": precision,
            "Recall": recall,
            "F1-Score": f1
        })

    return pd.DataFrame(results)
```

- plot_cm: Permite visualizar la correspondiente matriz de confusión, que nos permite conocer el número de falsos positivos y falsos negativos.
  - Entradas:
    - model: Modelo entrenado
    - x_test e y_test: Datos de test considerados.
  - Salidas:
    - Visualización de la matriz de confusión para una mejor interpretación.

```
def plot_cm(model, x_test, y_test):
    y_pred = model.predict(x_test)

    # Calcular matriz de confusión.
    mcm = confusion_matrix(y_test, y_pred)
    # Graficar la matriz de confusión.
    plt.figure(figsize=(8, 6))
    sns.heatmap(mcm, annot=True, fmt='d', cmap='Blues', xticklabels=[0, 1],
yticklabels=[0, 1])
    plt.ylabel('Real')
    plt.xlabel('Predicho')
    plt.title(f'Matriz de confusión del modelo: {model.__class__.__name__}')
    plt.show()
```

- plot_pr_curve: Permite mostrar las curvas precision-recall de los modelos que especifiquemos.
  - Entradas:
    - models: Diccionario con nombres y modelos considerados.
    - x_test e y_test: Conjunto de datos de test.
  - Salidas:
    - Visualización de las curvas precision-recall.

```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

def plot_pr_curve(models, x_test, y_test):
  for name, model in models.items():
      y_pred = model.predict(x_test)
      y_prob = model.predict_proba(x_test)[:, 1]

      # Calcular métricas.
      model_precision = precision_score(y_test, y_pred)
```

```
        model_recall = recall_score(y_test, y_pred)
        model_f1 = f1_score(y_test, y_pred)

        # Generar curva precisión-recall.
        precision, recall, _ = precision_recall_curve(y_test, y_prob)
        pr_auc = auc(recall, precision)
        plt.plot(recall, precision, label=f'{name}: {round(pr_auc, 2)}')

    # Mostrar resultados.
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title("Curvas Precisión-Recall")
    plt.legend()
    plt.grid()
```

7.1.1. Decision Tree

Este modelo crea una estructura similar a un árbol donde cada nodo representa una pregunta o condición sobre los datos, y cada rama representa un posible resultado o decisión basándose en esa pregunta. El proceso continúa dividiendo los datos en subconjuntos hasta llegar a un nodo final o "hoja", que contiene la predicción o respuesta final. A continuación se citan los hiperparámetros utilizados:

- max_depth: Especifica la profundidad máxima del árbol. Una mayor profundidad permite que el modelo aprenda patrones más complejos, pero también aumenta el riesgo de overfitting.

- min_samples_split: Define el número mínimo de muestras necesarias para dividir un nodo. Si un nodo tiene menos muestras que el valor especificado, no se dividirá y se convertirá en una hoja. Valores mayores ayudan a evitar divisiones innecesarias y a reducir el sobreajuste.

- min_samples_leaf: Determina el número mínimo de muestras que debe tener cada hoja. Esto asegura que las hojas no contengan pocas muestras, lo cual también ayuda a prevenir el sobreajuste.

- max_features: Controla cuántas variables se consideran para buscar la mejor división en cada nodo. La opción 'sqrt' usa la raíz cuadrada del número total de características, mientras que 'log2' utiliza el logaritmo en base 2. Esta restricción reduce la variabilidad del modelo y puede mejorar su generalización.

- criterion: Define la función de criterio usada para medir la calidad de la división en cada nodo. 'gini' y 'entropy' son las dos métricas más comunes. 'gini' intenta minimizar la probabilidad de clasificación incorrecta, mientras que 'entropy' busca minimizar la entropía, premiando divisiones con mayor "pureza" de clase en cada nodo.

Ajustando manualmente estos valores conseguimos un primer modelo con las siguientes métricas:

Accuracy	Precisión	Recall	F1 Score	ROC AUC	MCC	PR AUC
0.9982	0.7359	0.8023	0.7677	0.9322	0.7675	0.7627

Figura 25. Métricas Decision Tree Base.

7.1.2. Random Forest

El RandomForestClassifier es un tipo de modelo de ensamble que consiste en múltiples árboles de decisión entrenados en distintas porciones de los datos. Un Random Forest mejora la precisión y la estabilidad de las predicciones al combinar los resultados de múltiples árboles de decisión, generados aleatoriamente a partir de diferentes subconjuntos de datos y variables. En este proceso, cada árbol es entrenado de manera independiente, y la predicción final del bosque es la votación mayoritaria para clasificación. La aleatorización de datos y características, junto con la combinación de varios árboles, ayuda a reducir el sobreajuste. Los hiperparámetros considerados son los mismos que para el modelo anterior.

Ajustando manualmente estos valores conseguimos un primer modelo con las siguientes métricas:

Accuracy	Precisión	Recall	F1 Score	ROC AUC	MCC	PR AUC
0.9981	0.8907	0.5557	0.6844	0.9787	0.7028	0.7177

Figura 26. Métricas Random Forest Base.

7.1.3. XGBoost

El modelo XGBClassifier es otro algoritmo de boosting muy popular. XGBoost implementa técnicas avanzadas que permiten manejar grandes conjuntos de datos y optimizar la precisión del modelo. Se basa en el principio de construir un modelo fuerte a partir de múltiples modelos débiles (generalmente árboles de decisión) de manera secuencial. A continuación se citan los hiperparámetros utilizados:

- n_estimators: Especifica el número de árboles que se estrenarán en el modelo.
- learning_rate: Controla la contribución de cada árbol al modelo final. Un valor más bajo (como 0.01) puede mejorar la capacidad de generalización del modelo, pero generalmente se necesitarán más árboles (n_estimators) para lograr un rendimiento comparable.

- max_depth: Establece la profundidad máxima de cada árbol.

- subsample: Define la proporción de muestras que se utilizarán para entrenar cada árbol. Un valor menor a 1.0 reduce el riesgo de sobreajuste y mejora la generalización al introducir aleatoriedad en el proceso de entrenamiento.

- colsample_bytree: Controla la proporción de características que se utilizarán al construir cada árbol. Un valor menor a 1.0 puede ayudar a evitar el sobreajuste al agregar variabilidad a la construcción de los árboles.

- gamma: También conocido como "min_split_loss", es un parámetro de regularización que especifica la reducción mínima de la función de pérdida requerida para hacer una división en un nodo. Un valor mayor de gamma significa que se requiere una mayor mejora para realizar una división, lo que puede ayudar a prevenir el sobreajuste.

- reg_alpha: Este es el término de regularización L1. Un valor mayor puede ayudar a hacer que el modelo sea más robusto al reducir la complejidad de los árboles.

- reg_lambda: Este es el término de regularización L2. Similar al reg_alpha, pero se basa en una penalización cuadrática.

Ajustando manualmente estos valores conseguimos un primer modelo con las siguientes métricas:

Accuracy	Precisión	Recall	F1 Score	ROC AUC	MCC	PR AUC
0.9984	0.8398	0.7224	0.7767	0.9853	0.7782	0.798

Figura 27. Métricas XGBoost Base.

7.1.4. Balanced Random Forest

El Balanced Random Forest es una variante del Random Forest diseñada para problemas de clasificación con datos desbalanceados. En lugar de entrenar cada árbol del bosque con una muestra aleatoria del conjunto de datos, este método crea subconjuntos balanceados seleccionando aleatoriamente la misma cantidad de muestras de cada clase. Esto asegura que todas las clases estén representadas equitativamente durante el entrenamiento, reduciendo el sesgo hacia la clase mayoritaria. Este modelo es muy popular para casos como el nuestro, con lo que queremos probar si da buenos resultados. Muchos de los hiperparámetros utilizados son idénticos a los del Random Forest pero se introducen algunos nuevos.

- sampling_strategy: Define cómo se realiza el muestreo para equilibrar las clases. Mejora el rendimiento en datasets desbalanceados al garantizar que todas las clases estén representadas proporcionalmente.

- replacement: Indica si el muestreo de las muestras se hace con reemplazo o sin reemplazo. Aumenta la diversidad de los árboles al evitar que las mismas muestras se repitan.

Ajustando manualmente estos valores conseguimos un primer modelo con las siguientes métricas:

Accuracy	Precisión	Recall	F1 Score	ROC AUC	MCC	PR AUC
0.879	0.0271	0.9263	0.0526	0.9589	0.1469	0.5124

Figura 28. Métricas Balanced Random Forest Base.

Una vez ya hemos entrenado todos los modelos de esta primera tanda, ya podemos hacer uso de las funciones get_metrics y plot_pr_curve para comparar los resultados obtenidos.

En cuanto a los valores de precisión, recall y F1, en la figura 29 vemos que todos los modelos obtienen puntuaciones bastante decentes teniendo en cuenta que esta no es más que una primera aproximación. Sin embargo, el Balanced Random Forest es de momento el peor con diferencia. Sospechamos que esto ocurre porque dicho modelo, al balancear los datos en un caso tan extremo como el nuestro, causa un sesgo hacia la clase de fraudes que hemos aumentado. Esto provoca que nuestro modelo piense que siempre se darán situaciones similares y por eso cuando después le pasamos datos de test de nuevo desbalanceados cataloga como fraudulentas muchas más transacciones que las que debería. No obstante, veremos si somos capaces de mejorar dicho rendimiento en apartados posteriores.

	Model	Precision	Recall	F1-Score
0	Decision Tree	0.735944	0.802342	0.767710
1	Random Forest	0.890728	0.555785	0.684478
2	XGBoost	0.839872	0.722452	0.776749
3	Balanced Random Forest	0.027107	0.926309	0.052673

Figura 29. Métricas modelos base.

En cuanto a las curvas precision-recall y el área bajo las mismas, vemos en la figura 30 que de momento el mejor resultado lo tiene el Decision Tree (0.83), aunque seguido muy de cerca por el XGBoost (0.8)

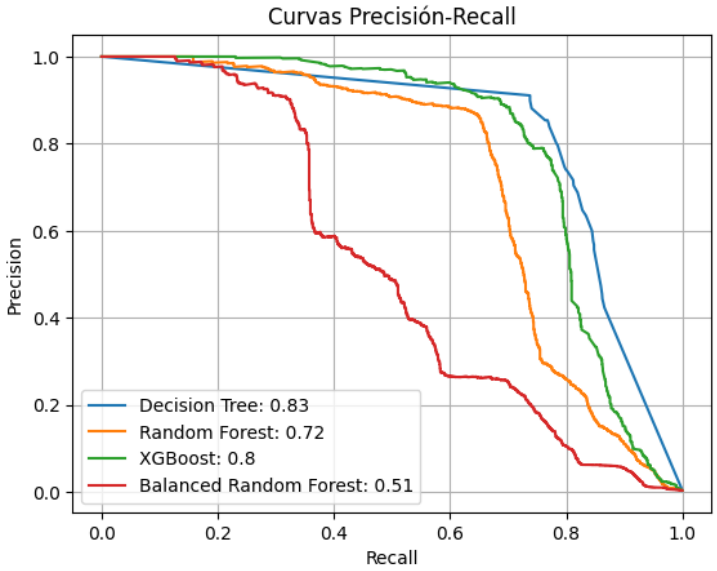

Figura 30. Curvas PR modelos base.

7.2. Modelos con GridSearch

Partiendo de un primer entrenamiento eligiendo manualmente los hiperparámetros de los modelos pasamos a realizar un GridSearch que encuentre la mejor combinación utilizando el F1 Score como medida de referencia y volvemos a entrenar los modelos. Sin embargo, nos hemos visto obligados a modificar únicamente los hiperparámetros más importantes dados los elevados tiempos de entrenamiento.

7.2.1. Decision Tree

Los hiperparámetros que mejores resultados han dado son:

Criterion	Max_depth	Max_features	Min_samples_ leaf	Min_samples_ split
Entropy	10	None	1	10

Figura 31. Hiperparámetros Decision Tree GridSearch.

Y los resultados obtenidos son:

Accuracy	Precisión	Recall	F1 Score	ROC AUC	MCC	PR AUC
0.9986	0.8484	0.7713	0.8080	0.9815	0.8083	0.8119

Figura 32. Métricas Decision Tree GridSearch.

7.2.2. Random Forest

Los hiperparámetros que mejores resultados han dado son:

Bootstrap	Max depth	Max features	Min_samples_ leaf	Min_samples_ split	N_estimators
True	30	None	2	10	100

Figura 33. Hiperparámetros Random Forest GridSearch.

Y los resultados obtenidos son:

Accuracy	Precisión	Recall	F1 Score	ROC AUC	MCC	PR AUC
0.9989	0.8836	0.8057	0.8429	0.9843	0.8432	0.8761

Figura 34. Métricas Random Forest GridSearch.

7.2.3. XGBoost

Los hiperparámetros que mejores resultados han dado son:

Colsample by_tree	Gamma	Learning rate	Max_depth	Min_child weight	N_estimat ors	Subsample
0.8	0	0.1	10	1	50	0.8

Figura 35. Hiperparámetros XGBoost GridSearch.

Y los resultados obtenidos son:

Accuracy	Precisión	Recall	F1 Score	ROC AUC	MCC	PR AUC
0.9988	0.897	0.7679	0.8274	0.9954	0.8293	0.8801

Figura 36. Métricas XGBoost GridSearch.

7.2.4. Balanced Random Forest

Los hiperparámetros que mejores resultados han dado son:

Bootstrap	Max_depth	Max features	Min_samples leaf	Min_samples split	N_estimators	Sampling strategy
True	20	None	1	2	100	0.5

Figura 37. Hiperparámetros Balanced Random Forest GridSearch.

Y los resultados obtenidos son:

Accuracy	Precisión	Recall	F1 Score	ROC AUC	MCC	PR AUC
0.9796	0.147	0.9621	0.2551	0.9929	0.3719	0.7444

Figura 38. Métricas Balanced Random Forest GridSearch.

Analicemos ahora los resultados de los modelos de este apartado. Atendiendo a los valores de precisión, recall y F1 de la figura 39, de nuevo vemos que el peor es el Balanced Random Forest, como los datos que utiliza para el train están muy sesgados, clasifica como fraude muchas más transacciones que las que debería, por eso su recall es tan alto pero su precisión tan baja. Por el contrario, el resto de modelos han mejorado de manera muy notable, tanto así que todos ellos superan la barrera de 0.8 de F1. Destaca sobre todo el Random Forest, cuyo valor de F1 incluso se acerca a 0.85. Dicho modelo además es capaz de detectar más del 80% de los fraudes y lo hace con una precisión de casi el 90%.

	Model	Precision	Recall	F1-Score
0	Decision Tree	0.848485	0.771350	0.808081
1	Random Forest	0.883686	0.805785	0.842939
2	XGBoost	0.897023	0.767906	0.827458
3	Balanced Random Forest	0.147053	0.962121	0.255113

Figura 39. Métricas modelos GridSearch.

En cuanto a las curvas de precision-recall ocurre algo similar, aunque en este caso Random Forest y XGBoost empatan con valores de 0.88.

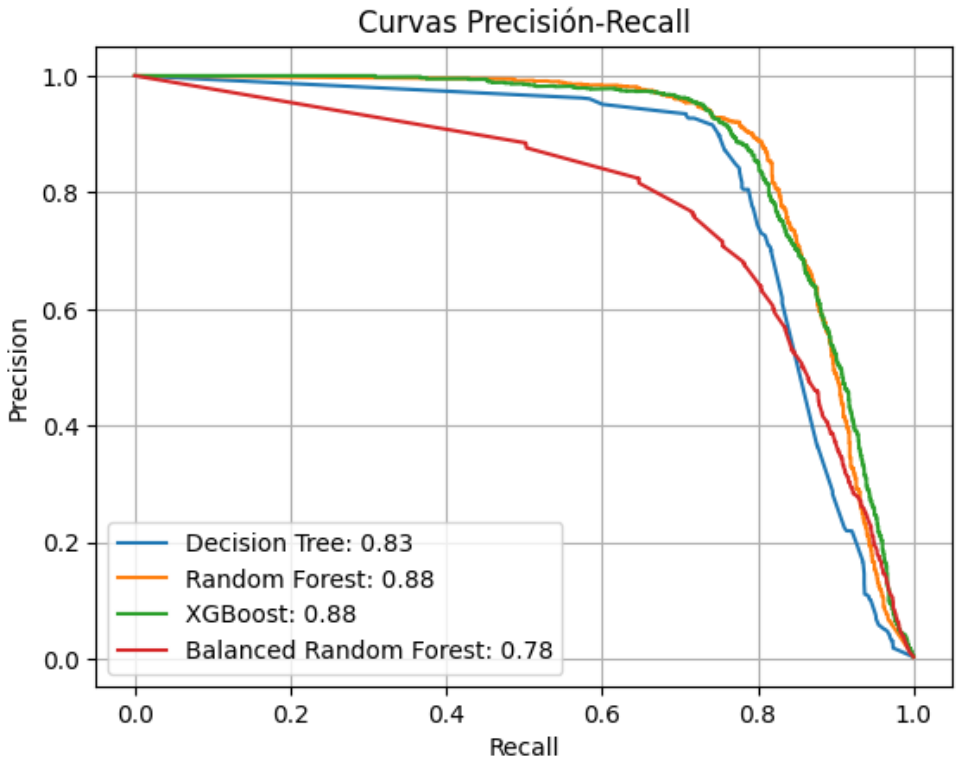

Figura 40. Curvas PR modelos GridSearch.

7.3. Pruebas con SMOTE

Ahora que ya hemos podido obtener resultados bastante sólidos tras los GridSearch, vamos a tratar de utilizar un método de balanceo de datos llamado SMOTE para ver si conseguimos refinar un poco más nuestros modelos. SMOTE se encarga de balancear la clase minoritaria de fraudes creando puntos de datos sintéticos a partir de los que ya existen. A continuación, vamos a ver cuál es su impacto en algunos de los modelos que estamos considerando.

7.3.1. Random Forest

Los hiperparámetros utilizados son los mismos que mejores resultados han dado en el apartado anterior:

Bootstrap	Max_depth	Max_features	Min_samples leaf	Min_samples split	N_estimators
True	30	None	2	10	100

Figura 41. Hiperparámetros Random Forest SMOTE.

Y los resultados obtenidos son:

Accuracy	Precisión	Recall	F1 Score	ROC AUC	MCC	PR AUC
0.9796	0.147	0.9621	0.2551	0.9929	0.3719	0.7444

Figura 42. Métricas Random Forest SMOTE.

7.3.2. XGBoost

Los hiperparámetros utilizados son los mismos que mejores resultados han dado en el apartado anterior:

Colsample by_tree	Gamma	Learning rate	Max_depth	Min_child weight	N_estimators	Subsample
0.8	0	0.1	10	1	50	0.8

Figura 43. Hiperparámetros XGBoost SMOTE.

Y los resultados obtenidos son:

Accuracy	Precisión	Recall	F1 Score	ROC AUC	MCC	PR AUC
0.9988	0.8924	0.7768	0.8306	0.9948	0.832	0.8833

Figura 44. Métricas XGBoost SMOTE.

Analizamos los resultados obtenidos y vemos como solo el XGBoost ha conseguido mejorar su F1 Score a 0,83. Añadir ruido no ha conseguido que el Random Forest prediga mejor sino que aumentan los falsos positivos.

	Model	Precision	Recall	F1-Score
0	Random Forest	0.882353	0.805785	0.842333
1	XGBoost	0.892405	0.776860	0.830633

Figura 45. Métricas modelos SMOTE.

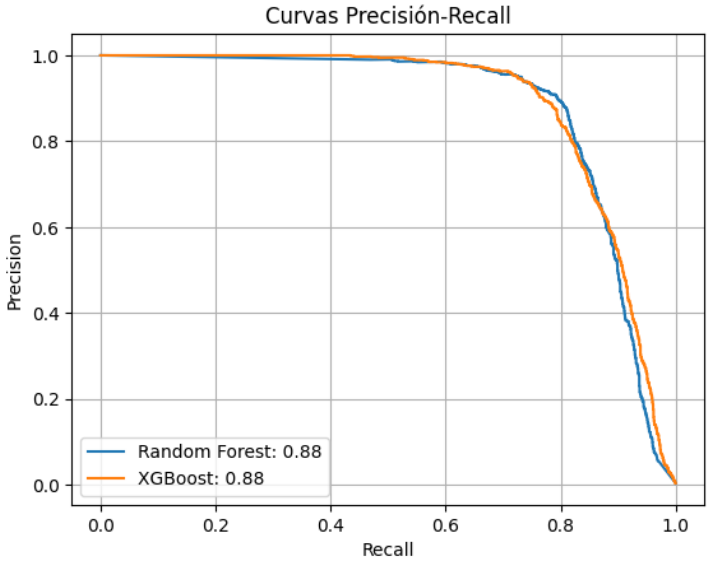

Figura 46. Curvas PR modelos GridSearch.

7.4. Unión de modelos

Como último intento de mejorar los resultados que ya tenemos con los modelos basados en árboles, vamos a tratar de combinar los dos mejores modelos obtenidos con GridSearch utilizando VotingClassifier. Se combinan las predicciones de los modelos utilizando probabilidades ponderadas y se le da un ligero mayor peso al Random Forest que al XGBoost.

Parámetros utilizados:

Estimators	Voting	Weights
[('xgb', model_xgb_gs), ('rf', model_rf_gs)]	Soft	[1, 1.5]

Figura 47. Parámetros Voting Classifier.

Resultados obtenidos:

Accuracy	Precisión	Recall	F1 Score	ROC AUC	MCC	PR AUC
0.9989	0.9015	0.7947	0.8448	0.9953	0.8459	0.8875

Figura 48. Métricas Voting Classifier.

En la figura 49 se muestra una comparativa de los resultados obtenidos con los modelos de los que partíamos y el VotingClassifier. Vemos que el F1 Score ha mejorado de 0,8429 que tenía el RandomForest obtenido con el Grid Search hasta un valor de 0,8448, que se debe a su vez al aumento en la precisión de dicho modelo llegando a un valor superior a 0.9.

	Model	Precision	Recall	F1-Score
0	Random Forest	0.883686	0.805785	0.842939
1	XGBoost	0.897023	0.767906	0.827458
2	Voting Classifier	0.901563	0.794766	0.844802

Figura 49. Métricas modelos Voting Classifier.

En cuanto a las curvas PR, también se aprecia una ligera mejora en la figura 50.

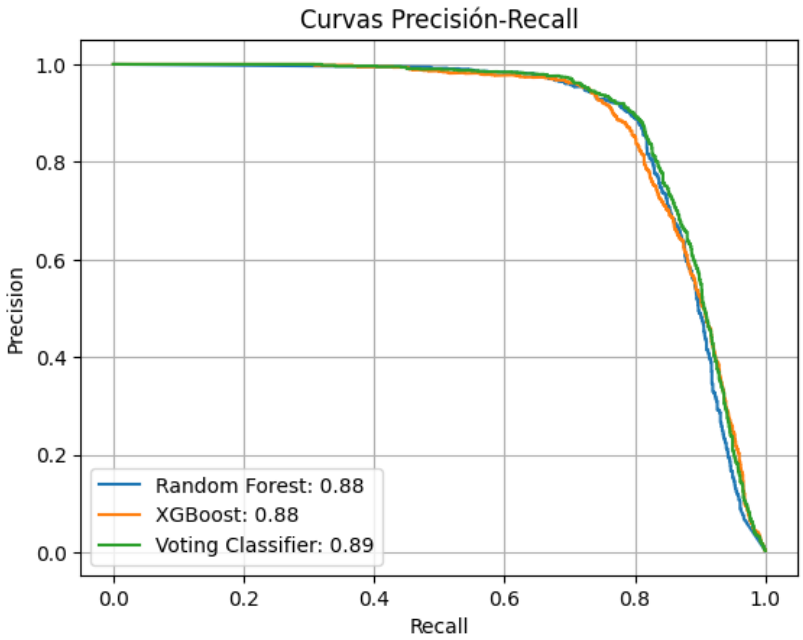

Figura 50. Curvas PR Voting Classifier.

8. Redes neuronales

Una vez exploradas diferentes posibilidades con los modelos de sklearn e imbalanced learn que hemos considerado más adecuados, vamos a proceder a utilizar una serie de redes neuronales para ver si somos capaces de mejorar los resultados obtenidos.

8.1. Redes neuronales básicas

Con el fin de sacar el máximo partido a las redes neuronales, vamos a utilizar MinMaxScaler, para garantizar que los valores de entrada estén entre 0 y 1 y facilitar así el aprendizaje de las redes. Una vez se han escalado los datos y dado que estamos usando el F1-score como una de las métricas de referencia, tenemos que

crear una función que nos permita calcular dicho valor ya que Keras no lo considera como métrica base.

```
from tensorflow.keras import backend as K
from tensorflow.keras.saving import register_keras_serializable

@register_keras_serializable(package='Custom', name='f1_score_custom')

def f1_score_custom(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.round(y_pred)
    y_pred = tf.cast(y_pred, tf.float32)

    tp = tf.reduce_sum(y_true * y_pred)
    fp = tf.reduce_sum((1 - y_true) * y_pred)
    fn = tf.reduce_sum(y_true * (1 - y_pred))

    precision = tp / (tp + fp + tf.keras.backend.epsilon())
    recall = tp / (tp + fn + tf.keras.backend.epsilon())
    f1 = 2 * (precision * recall) / (precision + recall +
tf.keras.backend.epsilon())
    return f1
```

También calculamos una función sencilla que nos permita obtener las métricas de rendimiento sobre el conjunto de test una vez finalizado el entrenamiento de las redes. La función nos permite extraer los resultados de precisión, recall y F1-score para cada una de las redes entrenadas y almacenar dichos valores en un DataFrame para mostrar el resultado.

Además, también calcula y muestra las curvas precisión-recall calculadas sobre el conjunto de test para cada modelo. De esta forma, esta función nos permite comparar los modelos entrenados y ver qué medidas se podrían tomar para tratar de mejorar los resultados.

```
def evaluate_models(models, X_test, y_test):
  # Crear un DataFrame vacío para los resultados.
  results = pd.DataFrame(columns=['Model', 'Precision', 'Recall', 'F1 Score'])

  # Lista para almacenar las curvas de precisión-recall.
  pr_curves = {}

  # Evaluar cada modelo en la lista.
  for model_name, model in models.items():
    # Hacer predicciones en el conjunto de test.
    y_pred_prob = model.predict(X_test)
```

```
    if y_pred_prob.shape[1] == 1:
        y_pred_prob = y_pred_prob[:, 0]
    y_pred = (y_pred_prob >= 0.5).astype(int)

    # Calcular precisión, recall y F1.
    precision = precision_score(y_test, y_pred, average='binary')
    recall = recall_score(y_test, y_pred, average='binary')
    f1 = f1_score(y_test, y_pred, average='binary')

    # Crear un DataFrame para los resultados del modelo actual.
    model_results = pd.DataFrame({
        'Model': [model_name],
        'Precision': [precision],
        'Recall': [recall],
        'F1 Score': [f1]
    })

    # Concatenar los resultados con el DataFrame principal.
    results = pd.concat([results, model_results], ignore_index=True)

    # Calcular y guardar la curva de precision-recall.
    precision_curve, recall_curve, _ = precision_recall_curve(y_test, y_pred_prob)
    pr_curves[model_name] = (precision_curve, recall_curve)

    pr_auc = auc(recall_curve, precision_curve)

    # Graficar la curva de precision-recall.
    plt.plot(recall_curve, precision_curve, label=f'{model_name}: {round(pr_auc,
2)}')
  # Mostrar la gráfica con las curvas de precision-recall.
  plt.xlabel('Recall')
  plt.ylabel('Precision')
  plt.title('Curvas Precision-Recall')
  plt.legend()
  plt.show()

  return results, pr_curves
```

Al igual que en el apartado anterior de modelos basados en árboles, hemos planificado cuidadosamente la estrategia a seguir con los entrenamientos y vamos a seguir un enfoque incremental en la dificultad de los mismos para poder saber de dónde partimos y hasta qué punto somos capaces de mejorar nuestras predicciones. Para ello, en primer lugar vamos a considerar tres arquitecturas sencillas, sin hacer oversampling ni undersampling y sin utilizar regularización.

La primera de las redes que consideramos cuenta únicamente con dos capas intermedias de 128 y 64 neuronas respectivamente. Para ellas se utiliza la función de activación ReLU, puesto que es una función que típicamente se utiliza para problemas como el nuestro y además es computacionalmente más eficiente que la tangente

hiperbólica. La segunda cuenta con tres capas ocultas de 256, 64 y 32 neuronas y la última es algo más compleja con 5 capas de 512, 256, 128, 64 y 32 neuronas, para tratar de hallar patrones más complejos.

Una vez entrenados los modelos, visualizamos cuáles han sido los resultados. En la figura 51 se observan valores bastantes similares, en los tres casos el área bajo la curva precision-recall es de alrededor de 0.7

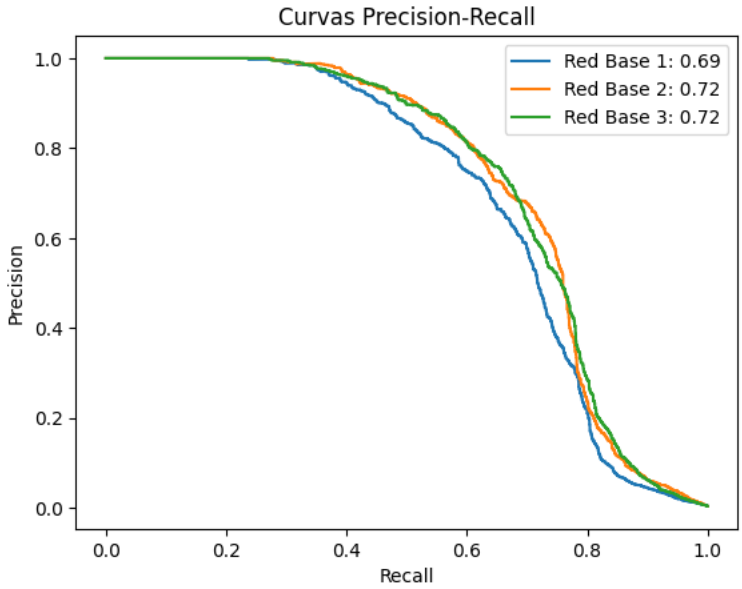

Figura 51. Curvas PR para redes base.

Echando un vistazo algo más detallado a las métricas extraídas vemos que en todos los casos se obtienen valores más altos de precisión que de recall. Por ejemplo en el caso de la red más sencilla se obtiene un valor de precisión superior a 0.8, aunque su recall es bastante bajo.

Model	Precision	Recall	F1 Score
Red Base 1	0.8348	0.5185	0.6397
Red Base 2	0.777	0.6239	0.6921
Red Base 3	0.7684	0.6398	0.6982

Figura 52. Métricas para redes base.

8.2. Ajuste de Impacto de Clase Minoritaria

Para este siguiente enfoque, vamos a tratar de mejorar los resultados obtenidos, asignando un peso mayor a la clase menos representada. Pretendemos equilibrar el impacto de cada clase, lo cual podría ayudar al modelo a evitar sesgos con un dataset tan desbalanceado como el nuestro. En cuanto a la arquitectura de las redes, no se modifica nada. De esta forma, volvemos a entrenar las redes pero esta vez añadiendo valores diferentes al parámetro ‘class_weight’ .

```
from sklearn.utils import class_weight

class_weights = class_weight.compute_class_weight('balanced',
classes=np.unique(y_train), y=y_train)
class_weight_dict = {i: class_weights[i] for i in
range(len(class_weights))}

print(class_weight_dict)
model1_weight = Sequential([
    Dense(128, activation='relu',
input_shape=(x_train_scaled.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
model1_weight.compile(optimizer='adam', loss='binary_crossentropy',
metrics=[f1_score_custom])
model1_weight.fit(x_train_scaled, y_train, epochs=20, batch_size=256,
verbose=1, class_weight=class_weight_dict)
```

En general se tienen resultados muy parecidos, excepto para la primera red, para la que se ha obtenido un rendimiento notablemente peor.

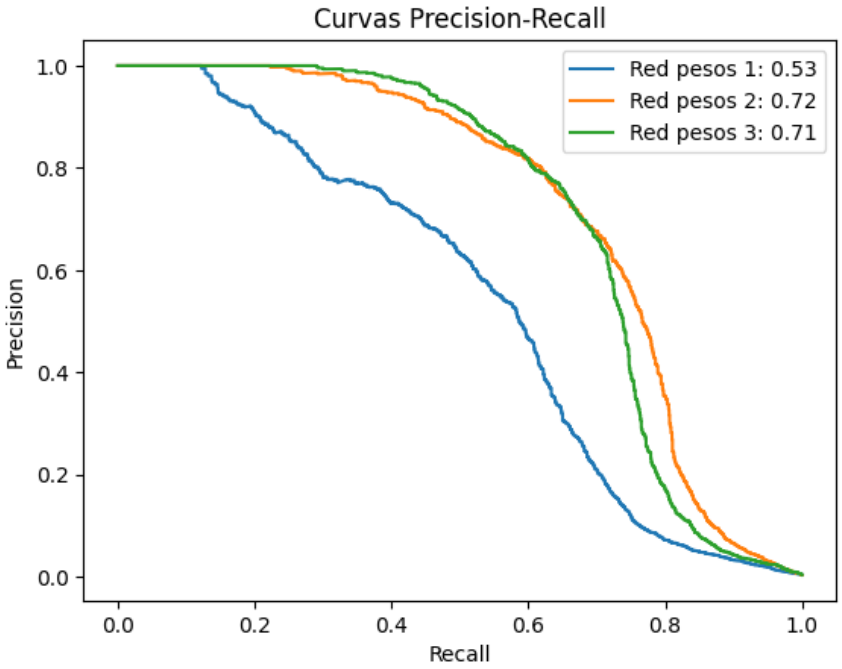

Figura 53. Curvas PR para redes con pesos.

Fijándonos en los valores de precisión y recall, nos damos cuenta de que el problema en la primera red viene de que a pesar un excelente recall cercano a 0,9, la precisión obtenida es muy mala. En definitiva, el ajuste de los pesos por clase no nos ha servido y vamos a tratar de dar con otra alternativa.

Model	Precision	Recall	F1 Score
Red pesos 1	0.0226	0.9352	0.0441
Red pesos 2	0.7681	0.6342	0.6948
Red pesos 3	0.7873	0.6246	0.6966

Figura 54. Métricas para redes con pesos.

8.3. Remuestreo de datos

Tal y como se ha hecho para los modelos basados en árboles, vamos a estudiar el impacto de balancear los datos mediante diversas técnicas. En primer lugar vamos a hacer uso del submuestreo.

8.3.1. Random Undersampling

Como siempre, modificamos nuestros datos y después procedemos a entrenar las redes de nuevo. Hemos optado por no reducir demasiado las instancias de la clase mayoritaria ya que si no los resultados empeoraban mucho. Hemos decidido quedarnos con un valor de 0.01, que indica que se tienen 100 veces más casos de transacciones legítimas que de fraudes, cuando inicialmente la proporción era de 200 a uno.

Se obtienen resultados parecidos a los del enfoque inicial, aunque en el caso de la tercera red se llega a un valor un poco mejor que los anteriores.

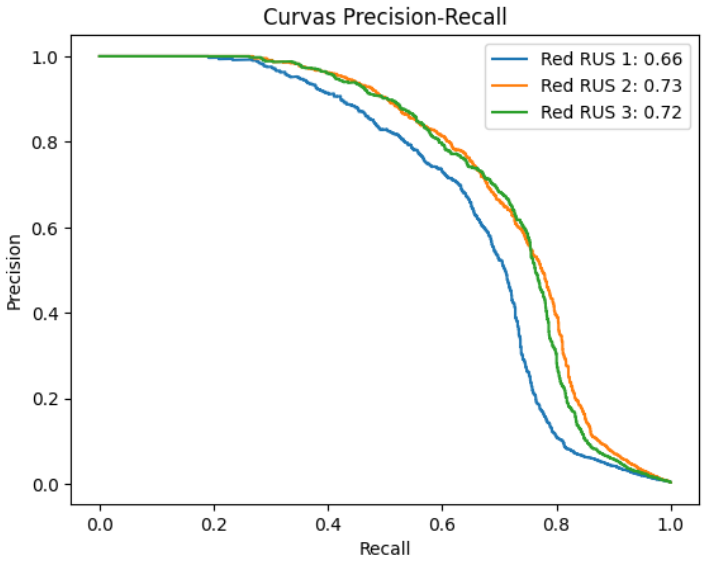

Figura 55. Curvas PR usando RUS.

Pasando a analizar la precisión y el recall, se tienen resultados muy parecidos a los del apartado anterior.

Model	Precision	Recall	F1 Score
Red RUS 1	0.7812	0.5509	0.6462
Red RUS 2	0.6983	0.679	0.6885
Red RUS 3	0.765	0.6322	0.6923

Figura 56. Métricas usando RUS.

8.3.2. Random Oversampling

Habiendo probado a reducir la clase mayoritaria, ahora vamos a probar a aumentar los casos de fraude. De nuevo, se ha probado con distintas proporciones y la que mejores resultados ha dado es pasar a un 1%.

Se aprecia una mejoría con respecto a los casos anteriores, especialmente en el modelo más complejo, donde ya se está cerca del valor de 0.8 en el área bajo la curva precision-recall.

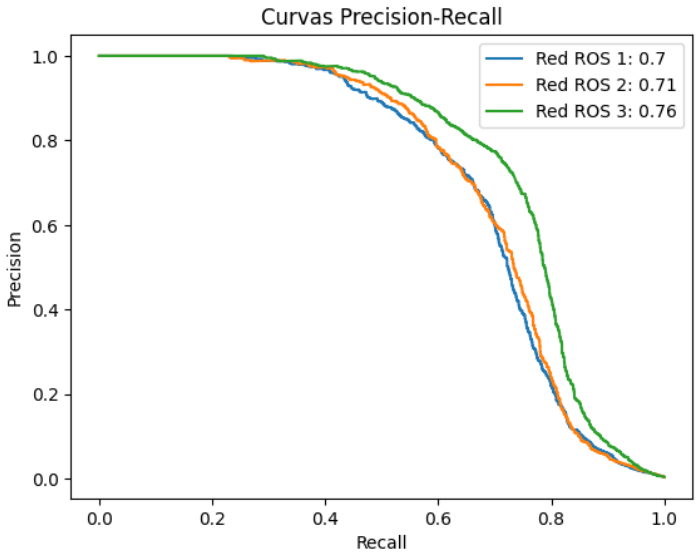

Figura 57. Curvas PR con ROS.

También al observar en mayor profundidad las métricas, se observa para la última red que se tiene un modelo bastante equilibrado.

Model	Precision	Recall	F1 Score
Red ROS 1	0.6719	0.6728	0.6724
Red ROS 2	0.6266	0.69	0.6568
Red ROS 3	0.7635	0.7073	0.7343

Figura 58. Métricas con ROS.

8.3.3. SMOTE

Debido a los buenos resultados en el caso del Random Oversampling, hemos decidido probar con SMOTE, una técnica de sobremuestreo más compleja que la ya mencionada. En este caso, en lugar de tomar simplemente muestras repetidas de la clase minoritaria, SMOTE genera nuevas instancias sintéticas.

Esta técnica utiliza la interpolación entre muestras reales de la clase minoritaria, seleccionando pares de puntos cercanos en el espacio de características y creando nuevos puntos en línea recta entre ellos. El resultado es un conjunto de datos más balanceado, con una distribución más realista que el duplicado aleatorio simple, lo que puede ayudar a nuestras redes neuronales a generalizar mejor.

En vista de los resultados mostrados en la figura 59, parece que los resultados obtenidos con SMOTE no mejoran los que ya teníamos anteriormente.

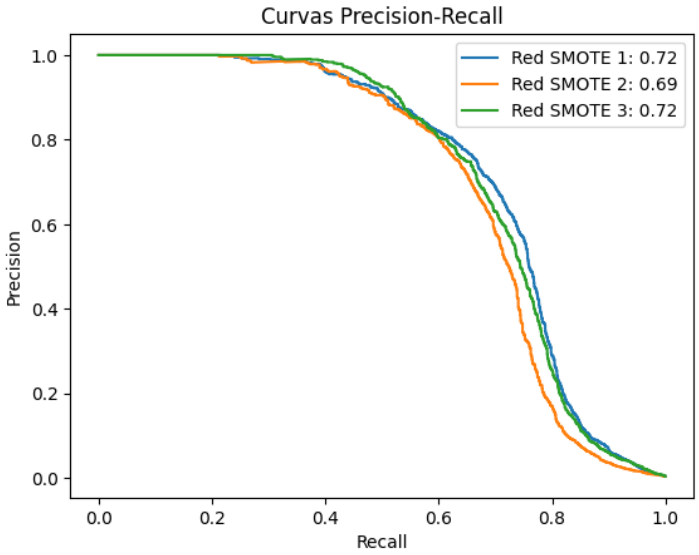

Figura 59. Curvas PR con SMOTE.

Los resultados obtenidos no reflejan ninguna mejora con respecto a los obtenidos mediante ROS.

Model	Precision	Recall	F1 Score
Red SMOTE 1	0.7273	0.6742	0.6997
Red SMOTE 2	0.7041	0.6556	0.679
Red SMOTE 3	0.6396	0.6955	0.6664

Figura 60. Métricas con SMOTE.

8.4. Métodos de regularización

Dado que los mejores resultados los hemos obtenido con la red número 3 al utilizar Random Oversampling, vamos a ver si somos capaces de mejorar aún más el modelo utilizando métodos de regularización y probando otras proporciones al sobremuestrear.

Después de cada capa densa, se ha añadido una capa de Dropout con una tasa de 0.3. Esto significa que durante el entrenamiento, el 30% de las neuronas en cada capa se "desconectarán" aleatoriamente, buscando reducir el sobreajuste.

También se han añadido algunas capas de BatchNormalization para estabilizar el aprendizaje. Esto ayuda a que las activaciones de las capas no se desvíen demasiado durante el entrenamiento, estableciendo un límite superior.

En primer lugar, se prueba a igualar la proporción de fraudes y transacciones legítimas. Después también se van a probar las proporciones de 0.75, 0.5, 0.1, 0.05 y 0.01.

En vista de la gráfica y los valores de precisión y recall, queda claro que no se han podido mejorar los resultados previos.

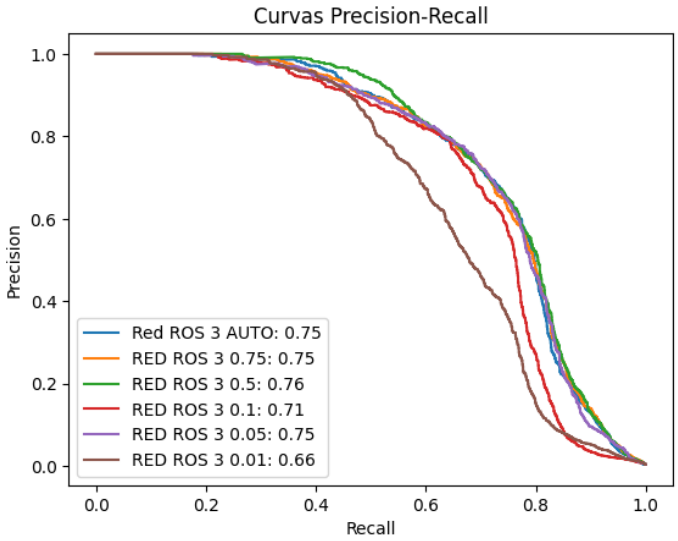

Figura 61. Curvas PR con ROS y métodos de regularización.

Model	Precision	Recall	F1 Score
Red ROS 3 AUTO	0.106	0.910	0.1899
Red ROS 3 0.75	0.1403	0.8987	0.2427
Red ROS 3 0.5	0.1051	0.9166	0.1887
Red ROS 3 0.1	0.2735	0.8002	0.4077
Red ROS 3 0.05	0.5555	0.7782	0.6483
Red ROS 3 0.01	0.672	0.6012	0.6346

Figura 62. Métricas con ROS y métodos de regularización.

Se muestra a continuación la matriz de confusión asociada a la red número 3 en la primera implementación de Random Oversampling que es la que mejores resultados nos ha dado.

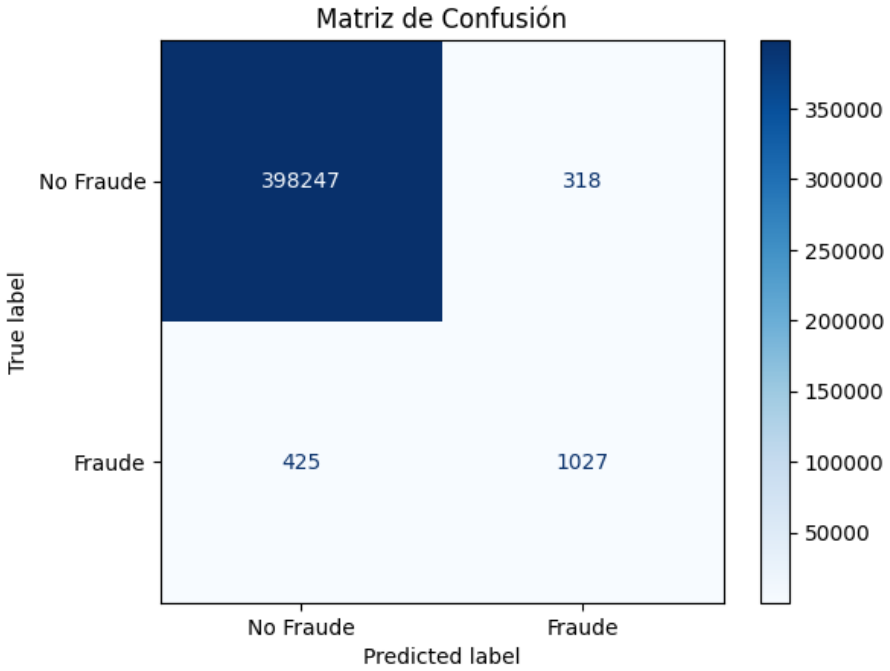

Figura 63. Matriz de confusión red ROS 3.

Como conclusión de esta parte dedicada a redes neuronales, hemos visto que a pesar de explorar diversas combinaciones de arquitecturas de redes y técnicas de remuestreo y regularización no se han podido alcanzar resultados como los obtenidos con XGBoost y modelos similares.

9. Selección de Modelo y Balance Final

Una vez finalizada toda la ronda de entrenamiento de modelos, nos centramos en ver con cuál se obtienen las mejores métricas. Para ello, nos quedamos con la mejor versión de cada uno de los cuatro modelos basados en árboles del apartado 7 y la mejor red neuronal del apartado 8.

Para obtener las métricas de cada uno, en primer lugar nos ocupamos de nombrar de forma diferente los datos escalados con StandardScaler (para modelos de tipo árbol) y con MinMaxScaler (para las redes neuronales).

Una vez hecho eso, obtenemos las curvas precision-recall de cada uno de los modelos considerados, así como algunas métricas de interés como el F1, calculado sobre los datos de test. De nuevo para ello deberemos tener en cuenta de si se trata de redes neuronales o no dado que los datos utilizados son diferentes.

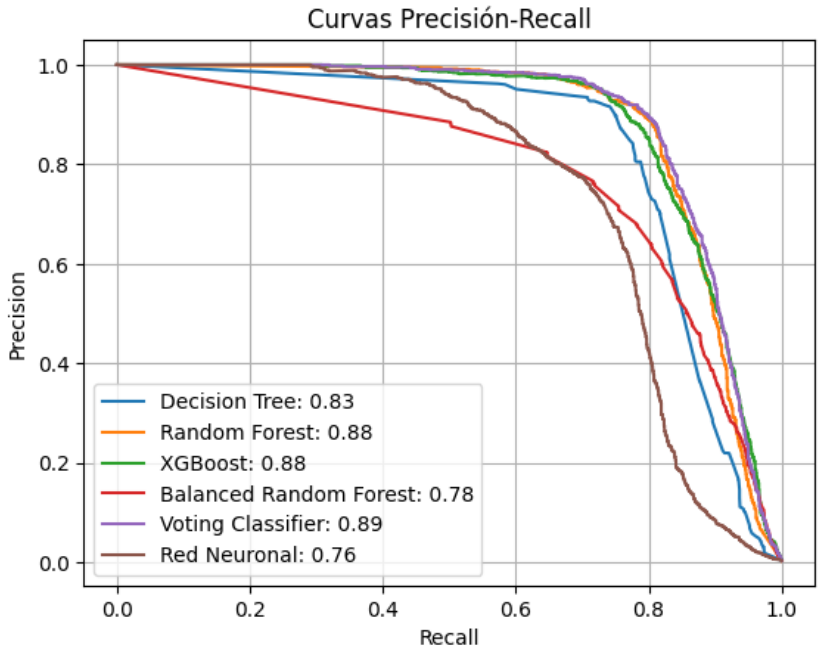

Figura 64. Curvas PR de los mejores modelos.

Como podemos ver en la figura 64, todos los modelos muestran buenos resultados, pero destacan el Random Forest, XGBoost y Voting Classifier, entrenados con los resultados de GridSearch y en el caso de XGBoost añadiendo datos con SMOTE. Por otro lado, se ve que el Balanced Ranfom Forest y la red neuronal (la correspondiente a la tercera arquitectura propuesta y usando ROS) son los que peores resultados dan con valores de 0,78 y 0,76 respectivamente.

Fijándonos ahora en las métricas de precisión, recall y F1, en la tabla de abajo se muestran los resultados de cada modelo ordenados de forma descendente en función del F1. De nuevo, vemos que los que destacan son el Voting Classifier junto con el Random Forest con un valor de más de 0,84 para el F1.

Con el Random Forest se detectan más del 80% de los casos de fraudes y todo ello con un impacto mínimo en la experiencia de los clientes ya que se tiene una precisión del 88%. Sin embargo, con el Voting Classifier se detectan el 79,47% de casos con un impacto todavía menor en la experiencia de los clientes debido a su precisión del 90%.

Por el contrario, destacamos el hecho de que el Balanced Random Forest, a pesar de tener un recall muy alto de prácticamente el 100%, muestra una precisión baja de menos del 15%

Model	**Precision**	**Recall**	**F1 Score**
Voting Classifier	0.9015	0.7947	0.8448
Random Forest	0.8833	0.803	0.8412
XGBoost	0.8891	0.7789	0.8303
Decision Tree	0.8484	0.7713	0.8080
Red neuronal	0.7635	0.7073	0.7343
Balanced Random Forest	0.147	0.9607	0.255

Figura 65. Métricas de los mejores modelos.

Ahora que hemos identificado los dos mejores modelos y lo cercano que están el uno del otro, evaluemos sus impactos potenciales en la práctica. Para empezar, examinemos de nuevo su matriz de confusión. De las 400,000 transacciones del conjunto de prueba, se observa en la figura 66 que el Random Forest identifica correctamente 1,170 casos de fraude, mientras que no logra detectar 282. Además, el número de falsos positivos es muy bajo, con solo 154 casos.

En resumen, este modelo tendría un impacto mínimo en la experiencia de los clientes, ya que los errores ocurren en aproximadamente 100 transacciones de cada 400,000, lo que representa una tasa de error extremadamente baja.

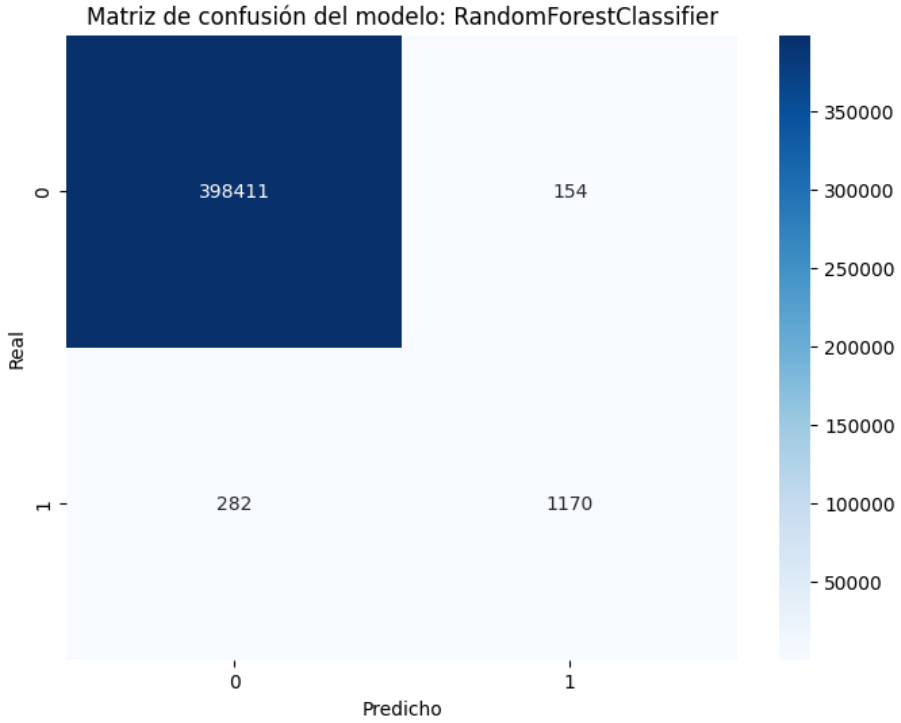

Figura 66. Matriz de confusión del Random Forest.

Por otro lado, el Voting Classifier detecta correctamente 1154 casos de fraude mientras que no logra detectar 298 de ellos. Lo que mejora respecto al anterior modelo es el número de falsos positivos bajando a 126 haciendo que la tasa de error sea aún más baja.

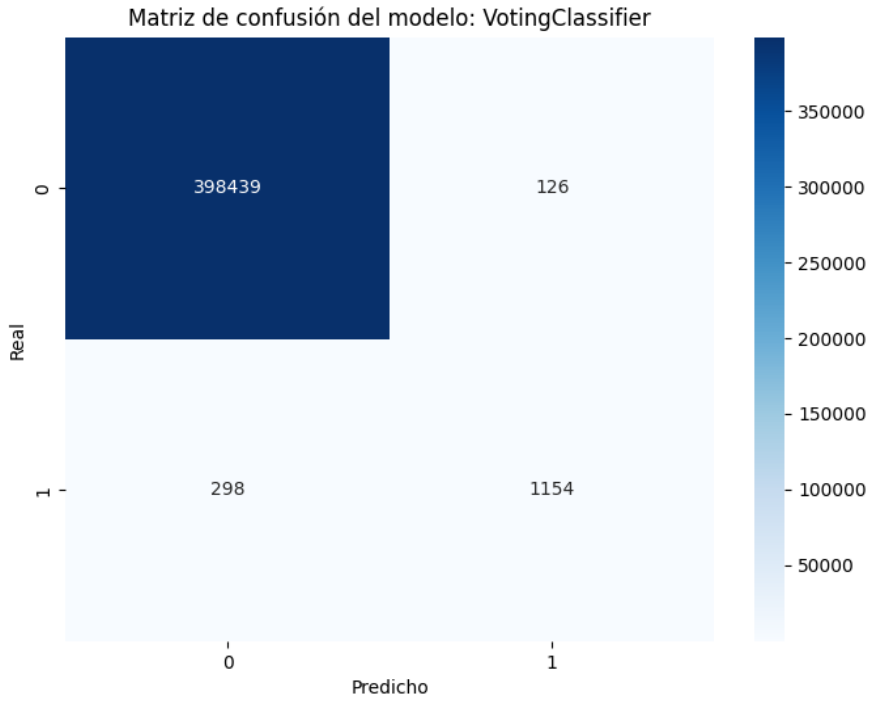

Figura 67. Matriz de confusión Voting Classifier.

En un caso del mundo real, se le presentarían los dos modelos al cliente y este elegiría el que mejor satisfaga sus necesidades. No obstante, vamos a tratar de aterrizar los resultados al mundo real y evaluar qué parte del dinero involucrado en fraudes en el conjunto de datos de test somos capaces de recuperar con uno u otro modelo.

Realizando la suma de la columna ‘amt’ considerando el dinero de las transacciones fraudulentas obtenemos 786.608,23$. Si calculamos la cantidad de dinero resultante de transacciones que hemos identificado correctamente como fraudulentas con el Random Forest obtenemos 731.912,90$, lo cual supone el 93.05% del total . Para el Voting Classifier en cambio, obtenemos la suma de 718.782,16$, es decir, el 91.38% de la suma total.

En primer lugar, esto quiere decir que nuestros modelos revelan resultados muy prometedores ya que no solo identifican alrededor del 80% de los fraudes que se cometen, sino que además se encargan de especializarse en la detección de fraudes con sumas más elevadas de dinero, permitiendo prevenir el robo de más de 700K$ en alrededor de 4 meses, que es la ventana de tiempo considerada para los datos de prueba. En otras palabras, los fraudes no detectados representan una cantidad "residual" en términos de impacto económico.

En vista de ello, creemos estar en posición de tomar una decisión acerca de cuál de los dos modelos seleccionados podría ser más adecuado. Bajo nuestro punto de vista, elegimos el Random Forest ya que preferimos detectar la mayor cantidad de fraudes posible para proteger a nuestros clientes y sus ahorros, brindándoles una buena experiencia con nuestros servicios.

10. Conclusiones

En este proyecto se ha llevado a cabo un proceso completo de Machine Learning, que abarca desde el análisis preliminar hasta el entrenamiento de diversos modelos, pasando por la creación de columnas sintéticas y un análisis exploratorio de datos (EDA) para identificar variables clave. El caso de uso se ha centrado en la detección de fraudes en tarjetas de crédito, uno de los problemas más comunes en el ámbito del ML. Para ello, hemos utilizado un conjunto de datos que simula transacciones realizadas en Estados Unidos durante los años 2019 y 2020. Este dataset contiene aproximadamente 2 millones de registros y 23 columnas.

Al comenzar un proyecto como este, es fundamental establecer objetivos claros. En nuestro caso, nos propusimos entrenar un modelo capaz de identificar la mayor cantidad de fraudes posible, minimizando al mismo tiempo el impacto en la experiencia de los clientes. Esto es crucial, ya que si se bloquean transacciones legítimas, los clientes podrían optar por la competencia debido a la mala experiencia. Para lograrlo, es esencial elegir las métricas adecuadas para evaluar el rendimiento de los modelos. En nuestro caso, nos centramos en el F1 score y el área bajo la curva precision-recall. Además, utilizamos la visualización de estas curvas y las correspondientes matrices de confusión para obtener una mejor comprensión de los resultados.

Sin embargo, antes de comenzar a trabajar con los modelos, fue esencial realizar un análisis exhaustivo de los datos disponibles para comprenderlos mejor y tratar de identificar variables relevantes para nuestras predicciones. En este proceso, nos dimos cuenta de que, dada la gran cantidad de datos y el severo desbalance (solo un 0.5% de las transacciones son fraudulentas), era fundamental asegurar la calidad de los datos utilizados para entrenar los modelos. Por eso, dedicamos una parte significativa del proyecto a estudiar las variables a fondo mediante visualizaciones y a crear nuevas variables de interés, como el historial de fraudes previos de los clientes.

Nuestro enfoque destaca la importancia de partir de buenos datos para obtener resultados de calidad, lo cual es crucial en casos con datasets tan complejos como el nuestro donde los tiempos de entrenamiento son tan elevados.

Una vez que los datos estuvieron listos, decidimos dividir los conjuntos de entrenamiento y prueba según la fecha, con el objetivo de simular de manera realista los resultados que se obtendrían en un entorno de producción, donde nuestros modelos solo tendrían acceso a información de transacciones pasadas. Esta división se realizó asignando un 80% de los datos para el entrenamiento y el 20% restante para la prueba. Esta estrategia es clave, ya que permite obtener resultados lo más cercanos posibles a la realidad, asegurando que el modelo se evalúe en condiciones que reflejan cómo operará una vez implementado en el mundo real.

Como se ha mencionado anteriormente, éramos conscientes de los largos tiempos de entrenamiento que requerirían nuestros modelos. Por ello, antes de comenzar a trabajar, diseñamos una estrategia clara, adoptando un enfoque incremental en cuanto a la complejidad de los modelos. Esto nos permitió sofisticarlos de acuerdo con las necesidades y puntos de mejora que surgieran a lo largo del proceso pero teniendo definida de antemano una estrategia clara a seguir. Asimismo, se identificaron algunos modelos que podrían ser efectivos para este tipo de problemas y se agruparon en dos bloques. Por un lado, seleccionamos modelos basados en árboles, que han demostrado ser muy efectivos en situaciones similares, concretamente Decision Tree, Random Forest, Balanced Random Forest y XGBoost. Por otro lado, consideramos la posibilidad de diseñar una serie de redes neuronales con el objetivo de explorar si podrían mejorar los resultados obtenidos por los modelos basados en árboles.

Para los modelos basados en árboles, comenzamos con unas combinaciones estándar de hiperparámetros para establecer una línea de referencia de cuál era nuestro punto de partida. Posteriormente, implementamos diversos GridSearch con validación cruzada para tratar de mejorar el rendimiento obtenido. Para ello, tuvimos presente en todo momento que debíamos escoger los grids de parámetros con cuidado para tratar de seleccionar valores adecuados sin necesidad de demasiadas combinaciones, pues esto elevaría en exceso los tiempos de entrenamiento. Una vez hecho esto, tratamos de aplicar SMOTE (creación de puntos de datos sintéticos de la clase minoritaria para balancear un poco más los datos) en algunos de los modelos. Por último, también exploramos la posibilidad de combinar dos modelos: uno con un alto recall, para maximizar la detección de fraudes, y otro con alta precisión, para reducir los falsos positivos. La esperanza era que esta combinación pudiera mejorar los resultados generales. Los resultados de esta combinación fueron muy positivos, llegando a superar el 90% de precisión.

A continuación, pasamos al bloque de redes neuronales, donde decidimos probar tres arquitecturas diferentes, siendo la primera la más sencilla y la tercera la más compleja. Inicialmente, todas las redes consistían únicamente de capas densas con la función de activación ReLU.

Luego, intentamos mejorar el rendimiento asignando pesos diferentes a cada clase mediante la opción class_weight, un método comúnmente utilizado para tratar con datos desbalanceados. Sin embargo, los resultados no fueron del todo satisfactorios. A continuación, probamos varias técnicas de manejo del desbalance para cada una de las tres redes: Random Under Sampling (RUS), Random Over Sampling (ROS) y SMOTE. Finalmente, dado que los mejores resultados se obtuvieron con la tercera red utilizando ROS, decidimos quedarnos con esta arquitectura. A continuación, incorporamos métodos de regularización, concretamente Batch Normalization y Dropout, para evitar el sobreajuste y también experimentamos creando más datos de la clase minoritaria utilizando ROS en diferentes proporciones.

Para concluir, tomamos las mejores versiones de los cuatro modelos basados en árboles y de las redes neuronales, y realizamos una comparación entre todos ellos en base a los valores de F1 y al área bajo la curva precision-recall, tal como se había adelantado. De esta forma, pudimos observar que los mejores resultados se obtenían con los modelos Voting Classifier y Random Forest, utilizando los hiperparámetros obtenidos mediante GridSearch. Comparando sus matrices de confusión y su impacto en la recuperación de dinero involucrado en fraudes decidimos escoger el Random Forest con el fin de detectar la mayor cantidad de fraudes, impactando lo mínimo posible en la experiencia de nuestros usuarios. Este modelo mostró un valor de F1 de 0,84 y fue capaz de detectar el 80% de los fraudes con una precisión cercana al 90%. Además, un aspecto clave es que la suma del dinero correspondiente a los casos que clasificamos correctamente como fraude representa más del 93% del total de dinero defraudado en el conjunto de test. Esto resalta no solo la efectividad del modelo para detectar fraudes, sino también su capacidad para minimizar las pérdidas económicas asociadas con los fraudes no detectados. Dicho de otra forma, aunque haya bastantes fraudes que pasan desapercibidos, éstos se pueden considerar "residuales" ya que suponen menos del 7% del total del dinero de transacciones fraudulentas.

En resumen, creemos que hemos sido capaces de cumplir los objetivos que nos planteamos al inicio del proyecto. Para ello hemos tenido que establecer una estrategia clara que nos ha permitido hacer frente al problema del tamaño y desbalance de nuestro dataset. El modelo que proponemos permite beneficiar no sólo a nuestra compañía (ya que optimiza la detección de fraudes y minimiza las pérdidas económicas) sino también a nuestros clientes, ya que nuestro modelo no provocará cambios en su experiencia.