Algoritmos de detección de fraude en bases de grafos

Federica Silberberg
Maestría en Ciencia de Datos y Aprendizaje Automático
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
federica.silberberg@gmail.com
Sofía Pérez
Maestría en Ciencia de Datos y Aprendizaje Automático
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
sperezcasulo@gmail.com

I. Introducción

I-A. Motivación

Los eventos de fraude en entidades financieras provocan pérdidas anuales multimillonarias. Debido a esto, la detección y prevención de fraude ha ganado protagonismo en los últimos años. Si bien los métodos tradicionales han tenido buenos resultados hasta ahora, día a día surgen nuevas formas de elusión, ocasionando la necesidad de implementar métodos más sofisticados.

Uno de los puntos débiles al momento de utilizar modelos relacionales en este tipo de casuísticas, son las limitantes a la hora de tener que detectar, visualizar y entender las conexiones entre los datos. En muchos casos las mismas son detectadas fuera de tiempo o incluso pasan desapercibidas.

Las bases de datos de grafos ofrecen la posibilidad de reformular el problema, enfocándose en las conexiones existentes entre los datos. Esto posibilita la creación de nuevos métodos que ofrecen mejores posibilidades en la detección de anillos de fraude, así como un análisis eficiente de los vínculos existentes y en tiempo real.

I-B. Objetivo

El objetivo del presente trabajo consiste en realizar un análisis del proceso de transformación de una base de datos relacional a una no relacional, específicamente de grafos, y con esta última resolver el problema de detección de fraude como un problema de clasificación de nodos.

El estudio se centrará en la detección de un tipo de fraude particular del sistema financiero conocido como “Fraude de tercer tipo” (“3er-party-fraude” en inglés). En este escenario, los actores fraudulentos eligen un cliente y roban su identidad, por ejemplo vía suplantación de identidad (phishing en inglés). Luego realizan transferencias de dinero a una cuenta intermedia, denominada mula. Finalmente, cuando se alcanza cierto balance vacían dicha cuenta.

Un desafío no menor para el estudio de fraude en el entorno financiero es la disponibilidad de fuentes de datos. Esto se debe a la falta de datos públicos sobre clientes y sus respectivas transacciones derivado de la confidencialidad de los mismos. En particular, las entidades financieras no hacen pública la información personal de sus clientes imposibilitando analizar otras casuísticas interesantes, como ser la detección de anillos de fraude basado en características de los clientes, conocido como “Fraude de primer tipo” (“1st-party-fraude” en inglés). Lo detallado anteriormente resume el motivo por el cual, este análisis se basa únicamente en la detección del mencionado “Fraude de tercer tipo”.

A su vez, existe una escasez de datos disponibles en formato de grafos, razón por la cual se requirió realizar una etapa previa de transformación de datos del modelo relacional a un modelo de grafos.

Para la implementación del modelo de clasificación binaria se utilizan diferentes algoritmos definidos en la biblioteca “Graph Data Science” (GDS) [1] de Neo4J.

I-C. Resultados esperados y conclusiones

Tras la realización de este proyecto, se esperaba poder resolver un problema de detección de fraude mediante la utilización de una base de datos de grafos y los algoritmos apropiados de la biblioteca *GDS*. Se pretendía obtener una solución que brindase resultados de igual o mayor exactitud en comparación con los obtenidos con un modelo sobre datos relacionales.

Sin embargo, durante el proceso se encontraron ciertos desafíos a la hora de modelar este problema con grafos, sobre los cuales se profundiza en la sección final del documento dentro de las conclusiones.

I-D. Estructura del documento

El documento se encuentra organizado de la siguiente forma:

En primer lugar, se tiene la sección de trabajos relacionados, en donde se hace un breve resumen del trabajo que se utilizó como referencia, además de las similitudes y diferencias con el presente análisis.

Luego, se presenta la sección de modelado del problema, en donde se profundiza en el análisis realizado para modelar nuestra base de datos de grafo. Dentro de esta sección se ahonda en el pasaje de formato relacional a grafos y la estructura de los datos.

En tercer lugar, se tiene la sección de experimentación, en donde se analizan los algoritmos utilizados para resolver el problema de detección de fraude, así como los resultados y desafíos del modelado.

Por último, se encuentra al final del documento una sección de conclusiones y trabajo futuro donde se mencionan los próximos pasos a seguir en este análisis y se repasan los desafíos encontrados durante el proceso.

II. Trabajos relacionados

El presente trabajo se inspira en un caso de uso publicado por Neo4J [2] con el fin de ejemplificar la utilización de la biblioteca *GDS*. En este caso, se emplea un conjunto de datos basado en la herramienta Paysim [3], el cual es enriquecido luego con características personales de los clientes; por lo que cuenta con una mayor cantidad de información que el conjunto de datos relacionales con el que se trabaja en esta implementación.

Si bien el ejemplo presentado no se adapta en un cien por ciento a esta aplicación, pues no se trabaja con "Fraude de tercer tipo", se utiliza como referencia a la hora de hacer el modelado y diseño de la base de grafos.

A diferencia de este trabajo, en el caso de uso de Neo4J no se requiere hacer una conversión los datos pues estos ya se encuentran en formato de grafos, y se comparte un repositorio con el correspondiente archivo en formato ".dump".

A su vez, en el mismo se cuenta con los identificadores de clientes, por lo que el foco está puesto en realizar análisis de "Fraude de primer tipo". Esto tiene algunas ventajas respecto al análisis de "Fraude de tercer tipo".

Dentro de la biblioteca *GDS* de Neo4j existen tres tipos de estado de algoritmos [4]:

- Algoritmos en calidad de producción, que son aquellos que han sido testeados en base a estabilidad y escalabilidad.
- Algoritmos en estado Beta, que son algoritmos candidatos para producción.
- Algoritmos en estado Alpha, que son experimentales, es decir pueden cambiarse o quitarse de funcionamiento en cualquier momento dado.

Para la casuística de "Fraude de primer tipo", donde la idea central es detectar anillos de fraude, se utilizan algoritmos de detección de comunidades, de similitud y centralidad. Estos algoritmos o métricas, cuentan con versiones en calidad de producción. Mientras que los algoritmos de *Random Forest* y Regresión logística ofrecidos, que se utilizan dentro de este análisis de clasificación de nodos para la detección de "Fraude de tercer tipo" se encuentran en versión Alfa y Beta respectivamente.

Por lo tanto, si se quisiese realizar una solución de fase de producción, solo sería factible la casuística ejemplificada por Neo4J. La detección de "Fraude de tercer tipo" que es analizada en este trabajo, no sería viable debido a que los algoritmos utilizados no están preparados para la puesta en producción, mientras que los otros sí. Sin embargo, la razón por la cual se basa el presente análisis en "Fraude de tercer tipo", es debido a la ausencia de identificadores personales de clientes en los datos de origen.

En definitiva, en una casuística de la vida real, donde la entidad de interés cuenta con los datos de los clientes (por ejemplo, un banco), se podría realizar una implementación de detección de anillos de fraude. Para esto sería necesaria una gran capacidad de cómputo, así como las licencias de Neo4J correspondientes para poder hacer uso de todas sus funcionalidades.

III. Modelado del problema

Abordar un problema de detección de fraude tiene un mismo objetivo independientemente del formato de datos que utilicemos: detectar si una acción es o no fraudulenta. Sin embargo, el enfoque a tomar en cuanto a la resolución del problema difiere según el modelo de base de datos con el que se cuente: grafos, documental o relacional.

Dado un problema de detección de fraude donde se cuenta con datos transaccionales dentro de un modelo relacional, se busca realizar una clasificación binaria. Para esto, se tiene una variable objetivo, codificada con 1 para el caso en que la transacción fue fraudulenta y 0 en caso contrario. Con este propósito, se utilizan algoritmos como Regresión Logística o Máquinas de soporte de vectores (SVM), entre otros, para detectar transacciones fraudulentas.

Dicha problemática utilizando bases de datos de grafos se ataca de forma diferente según el tipo de fraude a identificar. Con grafos, se necesita encadenar varios algoritmos y cálculos de métricas que computen diferentes tareas para alcanzar el objetivo mencionado.

Cuando la idea central es detectar anillos de fraude ("Fraude de primer tipo"), se identifican en primer lugar grupos de clientes (clusters en inglés) que comparten información de identificación personal (IP, email, teléfono, dirección, etc.) utilizando algoritmos de detección de comunidades [5]. Luego, dentro de cada clase se identifican clientes que sean similares utilizando métricas de similitud [6], y se asignan puntajes de fraude mediante el uso de medidas de centralidad [7]. En base a estos

puntajes, los clientes son etiquetados como posibles actores fraudulentos o no. Para cada etapa se requiere implementar un tipo de algoritmo diferente.

Cuando el objetivo es la detección de "Fraude de tercer tipo", se realiza una primer etapa de cálculo de métricas de centralidad, para posteriormente alimentar los algoritmos predictivos de clasificación binaria. Como se mencionó previamente, Neo4J ofrece dos algoritmos para la clasificación de nodos: Regresión Logística y *Random Forest*.

III-A. Base de datos

Como se mencionó anteriormente, los datos manejados en los sistemas financieros suelen ser confidenciales, imposibilitando la obtención de un conjunto de datos con información real.

Debido a lo anterior, en el presente informe se utiliza un conjunto de datos sintéticos generados en la herramienta *PaySim*. Esta base de datos contiene datos simulados que se basan en la información de una empresa real cuyo negocio consiste en proporcionar un servicio de transferencias entre usuarios a través de sus teléfonos móviles. El conjunto de datos utilizado se extrajo de Kaggle [8] y consiste en un subconjunto del original, representando la cuarta parte del mismo.

Los datos obtenidos están contenidos en un archivo de texto plano en formato *csv*. A modo de ejemplo, en el listado 1 se muestra el contenido del archivo obtenido.

Listado 1 Primeras cinco líneas del archivo *Paysim.csv*

```
step,type,amount,nameOrig,oldbalanceOrg,newbalanceOrig,nameDest,oldbalanceDest,newbalanceDest,isFraud,isFlaggedFraud
1,PAYMENT,9839.64,C1231006815,170136.0,160296.36,M1979787155,0.0,0.0,0,0
1,PAYMENT,1864.28,C1666544295,21249.0,19384.72,M2044282225,0.0,0.0,0,0
1,TRANSFER,181.0,C1305486145,181.0,0.0,C553264065,0.0,0.0,1,0
1,CASH_OUT,181.0,C840083671,181.0,0.0,C38997010,21182.0,0.0,1,0
```

Dentro de los datos transaccionales se cuenta con un identificador del originador del pago (*nameOrig*) y del receptor (*nameDest*) del mismo. Por otro lado, se tiene la información de los estados de cuenta previos y posteriores a cada transacción, para ambos actores involucrados. A su vez, se cuenta con información acerca del tipo de transacción realizada (*type*) y el monto de la misma (*amount*). Por último, se tiene un identificador referente a si la transacción fue o no fraudulenta (*isFraud*), codificado con 1 en caso afirmativo y 0 en caso contrario.

Los tipos de transacciones admitidos son los siguientes:

- ***CASH_IN*:** Refiere a los depósitos de dinero en una cuenta.
- ***CASH_OUT*:** Refiere a las extracciones de dinero de una cuenta.
- ***DEBIT*:** Refiere a montos debitados de una cuenta.
- ***PAYMENT*:** Refiere a los pagos realizados a comercios.
- ***TRANSFER*:** Refiere a transferencias realizadas entre dos cuentas.

Por su parte, a modo meramente ilustrativo se reconstruye el modelo relacional del cual proviene el conjunto de datos original. El mismo contiene cuatro tablas:

- ***PAYSIM_TRANSACTION_TYPE*:** Contiene los diferentes tipos de transacciones posibles.
- ***PAYSIM_CLIENTS*:** Contiene la información de los clientes que pertenecen al negocio.
- ***PAYSIM_TRANSACTIONS*:** Contiene las transacciones históricas realizadas.
- ***PAYSIM_FRAUD*:** Contiene información histórica respecto a las transacciones fraudulentas identificadas.

La figura 1 presenta el esquema relacional de la base de datos *Paysim*.

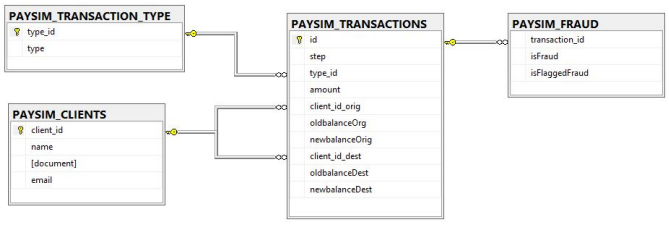

Figura 1: Esquema relacional de la base de datos *Paysim*

Analizando el contenido del archivo se observa que del total de transacciones realizadas, solo el 0.13 % representan casos fraudulentos. A su vez, estos solo corresponden a dos tipos de transacciones: *CASH_OUT* y *TRANSFER*.

Por otra parte, se observa que las operaciones siguen la siguiente lógica:

- Los destinos de las transacciones pueden ser clientes (*Clients*) o comerciantes (*Merchants*).
- Los orígenes de las transacciones sólo pueden ser clientes.
- Los comerciantes sólo reciben transacciones de pago (*Payments*)
- Entre los clientes se efectúan todas las demás operaciones.

Como se puede apreciar del análisis anterior, el conjunto de datos obtenido no contiene identificadores personales de los clientes. Por lo tanto, se vuelve complejo realizar una detección de anillos de fraude para este escenario. Por lo que se utilizarán los mismos para realizar una detección de "Fraude de tercer tipo", aplicando algoritmos de clasificación de nodos.

III-B. Pasaje del modelo relacional a grafos

De forma intuitiva, se podría pensar como un diseño adecuado definir a los actores de origen y destino como nodos del grafo, y las transacciones realizadas como relaciones entre los mismos.

Sin embargo, en este caso se opta por representar a todos estos como nodos del grafo, es decir, tanto las entidades de origen y destino como los diferentes tipos de transacciones entre ellos. Las relaciones entre los nodos se definen con el objetivo de dar dirección y sentido a las transacciones entre entidades. Este diseño posibilita realizar consultas sobre las transacciones de forma más eficiente, habilitando la posibilidad de implementar aplicaciones a tiempo real sobre el grafo. A continuación, se describen los diferentes componentes del modelo de grafos planteado.

1. **NODOS**
   Se definen siete tipos de nodos diferentes que comprenden a los actores de origen y destino, y los tipos de transacciones posibles. A su vez estos nodos se mapean en dos clases globales:
   - Entidad (*ENTITY*): Comprende a los clientes y comerciantes.
   - Transacción (*TRANSACTION*): Comprende todas las operaciones posibles que se realizan entre las entidades.

   En el cuadro I presentado a continuación, se muestra en detalle la definición de los nodos para el diseño propuesto y sus respectivas propiedades.

Cuadro I: Nodos del grafo

Nodos		
Etiqueta	*Tipo*	*Propiedades*
Entity	Client	Id (Integer) [Unique key]
	Merchant	Id (Integer) [Unique key]
Transaction	Cash-IN	Id (Integer) [Unique key]
	Cash-OUT	Amount (Float)
	Payment	Fraud (Integer)
	Transfer	Step (Integer)
	Debit	Type_id (Integer)

2. **RELACIONES**
   Se definen tres tipos de relaciones. Las primeras dos le dan sentido y dirección a las interacciones entre entidades. La última, se calcula posteriormente con el objetivo de enriquecer la entrada de los algoritmos *GDS* aplicados. Los tipos de relaciones definidos son:
   - *PERFORMS*: Se trata de una relación direccional, conecta el nodo entidad de origen con su respectiva transacción.
   - *TO*: Se trata de una relación direccional, que vincula la transacción con su correspondiente entidad destino.
   - *HAS_COMMON_ENTITY*: Se trata de una relación unidireccional, pretende ilustrar las entidades comunes entre transacciones y tiene asociada la propiedad de cantidad de entidades (cantidad de saltos) para ponderación.

A continuación se presenta el proceso de generación del grafo y su respectivo esquema.

1. En primer lugar, se implementa una función de ingesta en *Python* cuyo objetivo es recorrer el archivo de datos (*Paysim.csv*) y crear los tres tipos de nodos involucrados en cada transacción (origen, destino y acción) con sus respectivas propiedades

y relaciones. Su implementación se encuentra detallada en el *notebook Jupyter* publicado en el repositorio de *gitlab* del proyecto [9].
La figura 2 muestra el esquema inicial obtenido tras la implementación de la función de ingesta.

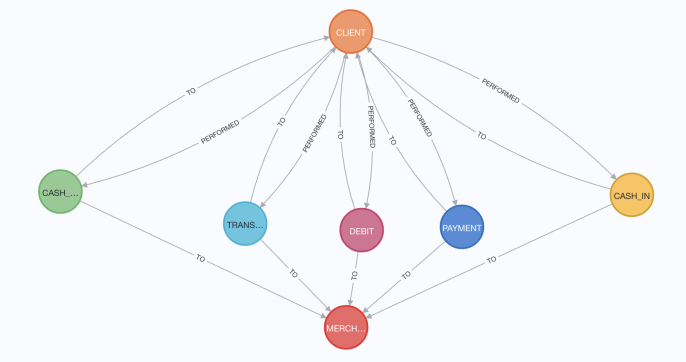

Figura 2: Esquema inicial de la base de grafos diseñada

2. En segunda instancia, se procede a generar y asignar las etiquetas genéricas de entidad (ENTITY) y transacción (TRANSACTION), mencionadas anteriormente. En la figura 3 se muestra el efecto de la mencionada incorporación sobre el esquema de la base, y se observa la aparición de nuevas conexiones entre los nodos.

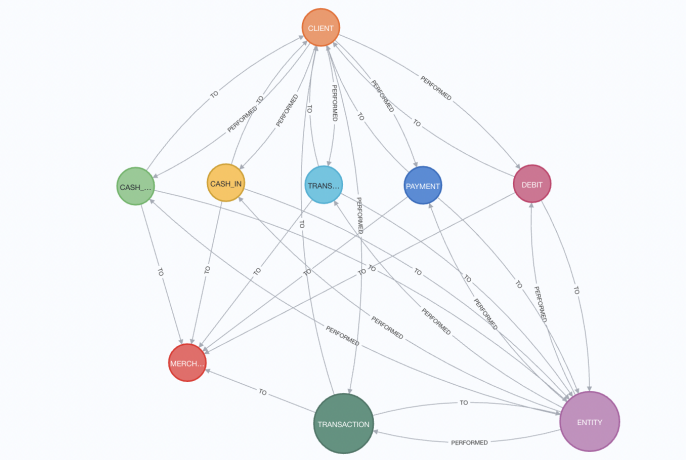

Figura 3: Esquema de la base de grafos diseñada con la incorporación de las etiquetas globales

3. Los algoritmos de *GDS* y el modelo de clasificación son aplicados sobre los nodos de transacción (*TRANSACTION*). Por esta razón, se crea un nuevo tipo de relación que a vincula todas las transacciones, con el fin de conectarlas y heredar información de los nodos de entidad origen y destino. Siguiendo este fin se crea la relación *HAS_COMMON_ENTITY* que une los nodos *TRANSACTION* que tienen un cliente en común, con una propiedad de ponderación basada en la cantidad de clientes comunes. La figura 4 ilustra la nueva relación definida.

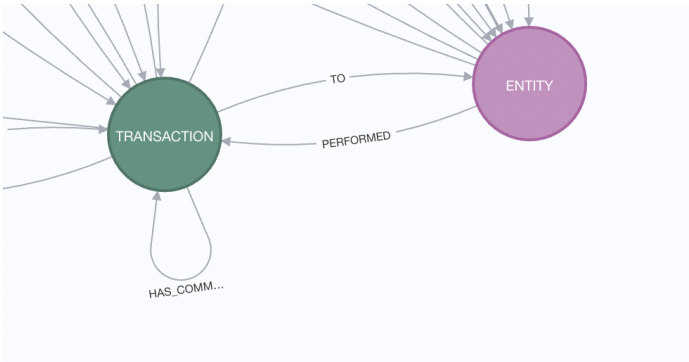

Figura 4: Incorporación de relación *HAS_COMMON_ENTITY* al esquema de la base de grafos

4. Finalmente se incorporan los nodos de predicción al esquema. Para esto, se repiten los pasos anteriores creando nuevos nodos y relaciones de predicción, para diferenciarlos del resto, se los nombra con el sufijo "*_PREDICT*". De esta forma, se obtiene un esquema totalmente independiente que se utiliza luego para realizar las predicciones correspondientes. La figura 5 presentada a continuación muestra el esquema final obtenido. Se destaca que no existe ninguna conexión entre los dos grafos generados: entrenamiento y predicción.

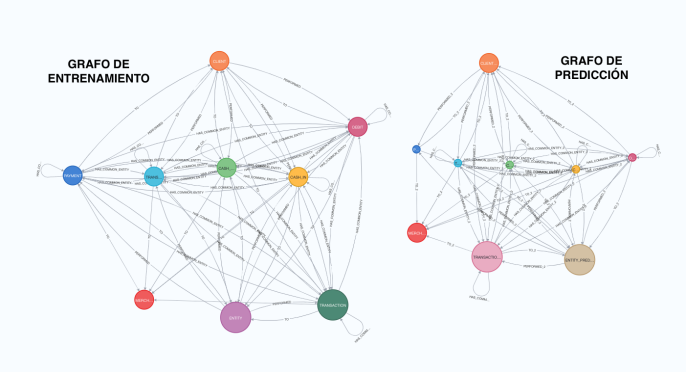

Figura 5: Esquema final de la base de grafos

III-C. Estimación de memoria

Como se mencionó anteriormente, cada transacción involucra tres nodos diferentes: entidad de origen, entidad de destino y acción.

Luego de observar que el conjunto de datos relacional contiene aproximadamente seis millones de tuplas y tras realizar algunas pruebas de rendimiento, se concluye que el costo computacional asociado a la carga de los datos en su totalidad supera ampliamente el poder de cómputo disponible para este proyecto. Por esta razón se opta por reducir la muestra a un diez por ciento, se amplia sobre este punto en la siguiente sección.

IV. EXPERIMENTACIÓN

A lo largo de esta sección, se detallan los pasos realizados para la definición e implementación del modelo y se analizan los algoritmos utilizados. A su vez, se profundiza sobre los desafíos encontrados durante el modelado del problema. Finalmente se presentan los resultados obtenidos tras aplicar el modelo final.

IV-A. Particionamiento y sub-muestreo

Un punto importante a tener en cuenta antes de entrenar cualquier modelo, es la división de los datos en subconjuntos de entrenamiento y validación. Una técnica muy común aplicada a los datos relacionales es el particionamiento aleatorio. Sin embargo, a la hora de trabajar con grafos esto no es del todo posible, pues podría implicar una fuga de información (o *data leakage*). Un ejemplo de esto, es cuando un nodo de entrenamiento presenta una relación hacia un nodo de validación, haciendo que de forma indirecta compartan información. El particionamiento aleatorio por sí solo no tiene dichos recaudos, por lo cual se debe ser muy cuidadoso al aplicarlo sobre un problema basado en grafos. Dentro de la biblioteca *GDS* de Neo4J, se presenta la solución *splitConfig* con el objetivo de mitigar el problema mencionado anteriormente. La misma es aplicada dentro del flujo de entrenamiento y consiste en las siguientes acciones:

1. El grafo de entrada es dividido en dos partes, entrenamiento y testeo, de acuerdo a los porcentajes especificados por el usuario.
2. El grafo de entrenamiento es a su vez dividido en un número configurable de carpetas (*folds*) para luego realizar la técnica de validación cruzada.

Tras la aplicación de esta funcionalidad se observa que no es posible distinguir la forma en la cual se realizó la división, es decir, no es posible obtener cuáles nodos fueron efectivamente elegidos para entrenamiento y cuales para validación. Este punto genera una gran limitante a la hora de analizar los resultados, por lo que se tuvo que recurrir finalmente a una combinación de técnicas para la definición de los grafos de entrenamiento y validación.

Por otra parte, como se mencionó anteriormente, el costo computacional asociado a la migración completa de los datos a un modelo de grafos superó los recursos computacionales disponibles, por esta razón se procedió a realizar una reducción de los datos (*downsample*). Para este punto, se tuvo la precaución de mantener las proporciones de casos fraudulentos, así como los tipos de operaciones existentes, con el fin de mantener la información lo más fiel a la realidad posible.

El cuadro II presentado a continuación detalla los diferentes ratios evaluados y los respectivos tiempos de carga obtenidos.

Cuadro II: Análisis de ratios de downsample

Ratio	Cantidad de transacciones	Tiempo de carga
0.5 %	31812	22 minutos
1 %	63624	37 minutos
5 %	318120	184 minutos

A modo de resumen, se realizaron los siguientes pasos para llevar acabo el particionamiento y sub-muestreo de los datos:

1. Se realiza un *downsample* de los datos con proporción del 5 % del conjunto de datos original, manteniendo las proporciones entre transacciones fraudulentas y tipos de operaciones.
2. Se realiza un particionamiento aleatorio sobre el subconjunto de datos obtenido, seleccionando el 85 % de los datos para entrenamiento y el 15 % restante para predicción.
3. Se realiza la carga de los datos de entrenamiento generando el grafo correspondiente.
4. Se entrena el modelo utilizando la funcionalidad *SplitConfig* provista por Neo4J.
5. Se realiza la carga de los datos de predicción generando el grafo correspondiente.
6. Se valida la no existencia de conexiones entre los grafos de entrenamiento y predicción generados.

IV-B. Algoritmos GDS

Como primer paso en la realización del proceso de clasificación de nodos, se genera una proyección en memoria del grafo original, lo que se denomina *in-memory graph* [10]. Dicha proyección es temporal y existirá mientras la base de datos esté activa.

En segunda instancia, se ejecutan ciertos algoritmos de centralidad y reducción de dimensionalidad, proporcionados por la biblioteca *GDS*, con el fin de obtener nuevas métricas que serán insumo del modelo final.

La medida de centralidad indica la importancia de los diferentes nodos dentro del grafo. Todos los algoritmos de centralidad aplicados se encuentran dentro de la categoría *aptos para producción*, principal razón por la cual fueron escogidos.

Por otro lado, se utiliza un algoritmo de reducción de la dimensionalidad, también en calidad de producción.

A continuación, se presentan los algoritmos seleccionados y se resumen sus principales características.

- **Algoritmo de centralidad del vector propio (Eigenvector Centrality)** [11] [12]: Este algoritmo desarrollado en 1986 por Phillip Bonacich, mide la influencia transitiva de los nodos dentro del grafo. El mismo evalúa las relaciones de los nodos con un esquema de puntuaciones. Los nodos de alta puntuación representan a aquellos que tienen mayores conexiones y por lo tanto se asume que poseen un mayor nivel de relevancia. A su vez, las relaciones que surgen en nodos de puntuación alta contribuyen más a la puntuación de un nodo que las conexiones de los nodos de puntuación baja. Es decir, un puntaje alto de esta métrica, implica que el nodo está conectado a muchos nodos con puntajes altos. Se considera que esta medida es relevante para el objetivo planteado puesto a que el foco está puesto en las relaciones entre nodos y en base a esto cuantificar la influencia de los mismos dentro del grafo.
- **Algoritmo de *Page Rank*** [13] [14]: Este algoritmo creado por *Google* es una variante del algoritmo anterior. En particular, mide la importancia de cada nodo dentro del grafo en función del número de relaciones entrantes y la importancia de los nodos de origen correspondientes. En este caso, se hace énfasis en las relaciones bajo el supuesto de que un nodo es tan importante como los nodos a los que se vincula. A su vez, mejora la asignación haciendo una normalización de los puntajes en base al grado de cada nodo. De esta forma, no necesariamente asigna altos puntajes a todos los nodos conectados a un nodo importante.
- **Algoritmo *Hyperlink-Induced Topic Search (HITS)*** [15] [16]: La búsqueda de temas inducida por hipervínculos (HITS), es un algoritmo que analiza y clasifica a los nodos en base a dos puntajes: un puntaje central (*hub*) y uno de autoridad (*auth*). El primero, estima el valor de sus relaciones con otros nodos, y el segundo, la importancia del nodo en el grafo. En otras palabras, un nodo tendrá un puntaje *hub* alto si apunta a muchos nodos importantes. Mientras que un nodo tendrá un puntaje *auth* alto si muchos nodos apuntan a él. Nuevamente se pone foco en las relaciones existentes entre nodos y por eso es considerado de interés en la casuística a resolver.
- **Algoritmo *Fast Random Projection (FastRP)*** [17]: Se trata de un algoritmo de incrustación de nodos en la familia de algoritmos de proyección aleatoria. La idea central es utilizar representaciones vectoriales para reducir la dimensionalidad y conservar mayor parte de la información de distancia. En el caso de los grafos, lo importante es preservar la similitud entre los nodos y los nodos vecinos. Esto implica que a dos nodos vecinos similares se les debe asignar vectores de incrustación similares. Por el contrario, a dos nodos que no son similares no se les debe asignar vectores de incrustación parecidos.

IV-C. Implementación de los modelos

La implementación de cada modelo se realiza mediante una serie de etapas dentro de un flujo de trabajo (*pipeline*). Cada una de ellas se describen a continuación:

1. En primer lugar, se especifican las variables con las cuales se entrenará cada modelo. Se analizan dos posibles opciones, la primera consiste en definir explícitamente las variables de entrenamiento, y la segunda utiliza el algoritmo *FastRP* sobre las mismas con el fin de reducir su dimensionalidad.
2. La siguiente etapa consiste en dividir el grafo utilizando la funcionalidad *SplitConfig*. Se implementó un ratio de 80 % para entrenamiento y 20 % validación, y se definió una cantidad de cinco carpetas (*folds*) para validación cruzada.
3. Como próximo paso, se configura un modelo de Regresión Logística con cierto rango de hiper parámetros, con el fin de elegir el mejor en términos de las métricas mencionadas en el siguiente punto:
   - Regresión Logística (Logistic Regression)
     - Batch size: [100, 500]
     - Epochs: [100, 500]
     - Penalty: [0.0625, 0.5, 2.0]
     - Patience: [1, 5]
     - Tolerance: [0.001, 0.01]
4. En cuanto a la elección de métricas de rendimiento, se optará por utilizar la medida de sensibilidad (*recall*). La misma refiere a la proporción de casos positivos que fueron correctamente identificadas por el algoritmo. Esta elección se debe a que se está trabajando con un conjunto de datos sumamente desbalanceado por lo que la medida de exactitud (*accuracy*) no reflejará cien por ciento el rendimiento del algoritmo. A su vez el principal interés es poder predecir correctamente la clase positiva que es la minoritaria en este caso (fraude).

Tras la implementación del mencionado flujo, se detectaron algunos problemas que contradicen la documentación publicada por Neo4J y que se enumeran a continuación:

1. No es posible aplicar el algortimo *Random Forest*, tras su ejecución, la misma nunca finaliza y se debe forzar la detención del código. Se atribuye este error a que el algoritmo se encuentra en fase Alpha.

2. Dentro del algoritmo Regresión Logística no es posible crear un rango de valores para los parámetros *batch size*, *epochs*, *penalty*, *patience* y *tolerance*. La documentación oficial de Neo4J indica que esto es soportado, sin embargo se obtiene un error indicando que el argumento no admite rangos de valores, tal cual se muestra en el Listado 2 debajo.

Listado 2 Error tras implementar rangos de valores en parámetros de la Regresión Logística

```
java.lang.IllegalArgumentException: The value of 'penalty' must be of type 'Double' but was 'HashMap'.
```

   Esta limitante provoca que cada cambio de parámetro implique la incorporación de un nuevo algoritmo, volviendo sumamente tedioso el tuneo fino de hiper parámetros.

3. La biblioteca *GDS* contiene una función de calibración (*Auto-Tunning*) que permitiría ajustar automáticamente los hiper parámetros de los modelos, sin embargo la misma no es reconocida por la última versión de *GDS*, tal como se muestra en el Listado 3. Nuevamente se atribuye este error a que el algoritmo se encuentra en fase Alpha.

Listado 3 Error tras implementar la función de calibración *Auto-Tunning*

```
Neo.ClientError.Procedure.ProcedureNotFound:
There is no procedure with the name 'gds.alpha.pipeline.nodeClassification.configureAutoTuning'
registered for this database instance.
```

A modo de resumen, el cuadro III presentado a continuación, resume las principales características de los dos modelos seleccionados para el estudio.

Cuadro III: Resumen de los modelos candidatos

	Modelo 1	**Modelo 2**
Variables	amount, centrality, hit_auth, hit_hub, pagerank, step, type_id	vector embebido FastRP
Algoritmo	***Regresión Logística:*** *maxEpochs: 500* *minEpochs: 1* *penalty: 0.0625* *patience: 5* *batchSize: 500* *tolerance: 0.001*	
Métrica	Recall	

IV-D. Análisis de resultados

En esta sección se presentan los resultados obtenidos para los dos modelos candidatos, analizando también sus respectivas métricas de rendimiento.

El Cuadro IV presentado a continuación resume los resultados de entrenamiento de cada modelo:

Cuadro IV: Métricas de rendimiento del grafo de entrenamiento

Modelo	**train recall avg**	**val accuracy**	**test precision 0**	**test precision 1**	**test recall 0**	**test recall 1**
1	0.18399999999264	0.99845978	0.99845978	0.0	0.9999999999998	0.0
2	0.0	0.99845978	0.99845978	0.0	0.9999999999998	0.0

Como se puede apreciar, si bien ambos modelos logran valores de exactitud muy buenos, no logran una buena predicción de la clase minoritaria (fraude), de hecho en el segundo modelo no se logra predecir ningún caso de fraude.

Para lograr un análisis más exhaustivo, se aplican ambos modelos al grafo de predicción creado en la etapa inicial y se procede a analizar los resultados obtenidos. En particular se estudian sus matrices de confusión en los cuadros V y VI respectivamente. También en las figuras 6 y 7 se visualizan las correspondientes curvas *lift* obtenidas para cada uno de ellos.

El primer modelo posee una sensibilidad (*recall*) alta, entorno al 92 %. Es decir, el modelo es capaz de detectar muy bien las transacciones fraudulentas. A su vez, se destaca que el modelo logra un muy buen *lift*. Esto implica que, si se ordena la base según la probabilidad de fraude obtenida por el modelo, en los primeros dos deciles se logra una detección de fraude de al menos tres veces mayor. Como contra parte, se obtiene un nivel de exactitud demasiado bajo, del 41 % para ser exactos, lo cual se traduce en la generación de muchos falsos positivos. Si bien es de suma importancia la predicción de la clase fraudulenta (minoritaria), es evidente que tampoco resulta de utilidad tener un modelo que genere continuas falsas alertas por casos no fraudulentos. El equilibrio necesario entre las métricas de exactitud y sensibilidad no fue alcanzado en este caso, haciendo que el modelo no sea elegible para un ambiente productivo.

Cuadro V: Matriz de confusión Modelo 1

	True Negative	**True Positive**
Predicted Negative	17274	5
Predicted Positive	24334	54

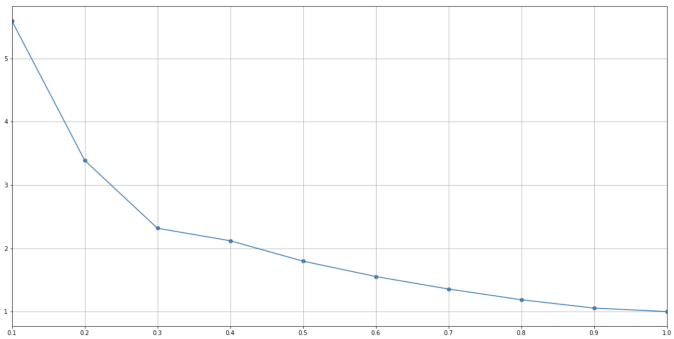

Figura 6: Curva lift Modelo 1

En cuanto al segundo modelo estudiado, si bien posee muy buena exactitud (99 %), su sensibilidad (*recall*) es nula, es decir no logra predecir en absoluto la clase minoritaria.

Como fue advertido durante el entrenamiento, en este caso la reducción de dimensionalidad introducida por la implementación del algoritmo *FastRP* no contribuye al rendimiento del modelo.

Cuadro VI: Matriz de confusión Modelo 2

	True Negative	**True Positive**
Predicted Negative	41608	59
Predicted Positive	0	0

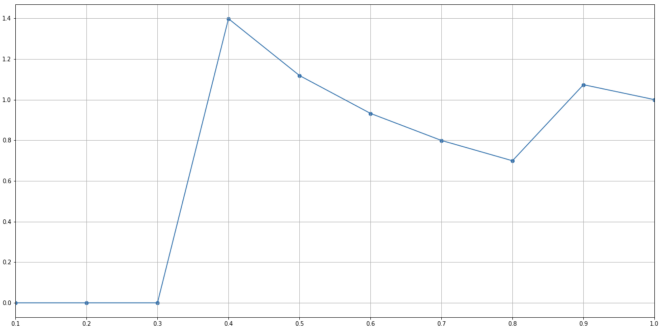

Figura 7: Curva lift Modelo 2

Se concluye que los modelos diseñados no muestran buenos resultados. Su bajo rendimiento se atribuye entre otras cosas a los siguientes motivos:

1. La imposibilidad de trabajar con el conjunto completo de datos debido al alto costo computacional que implica la conversión al modelo de grafos.
2. El estado Alpha y Beta de los algoritmos utilizados, lo cual deriva en la falta de herramientas para realizar una calibración fina de los algoritmos.

V. Conclusiones y trabajo futuro

En primer lugar, se destaca que en el ámbito de la biblioteca *GDS* de Neo4J existen varios algoritmos en estado Alfa y Beta, lo cual dificulta el desarrollo de la solución de detección de "Fraude de tercer tipo" en un ambiente productivo, como se menciona en el desarrollo del presente trabajo. A su vez, en estos casos ocurre también que la documentación no siempre condice con lo observado en la implementación, creando aún más barreras a la hora de desarrollar la solución.

Sin embargo, para el caso de detección de fraude a través de anillos ("Fraude de primer tipo") sería posible, a priori, poner un modelo en producción dado que se utilizan algoritmos que son aptos para ello. Particularmente, si se quisiera desarrollar este caso de uso en la vida real, los requisitos correspondientes refieren a contar con la capacidad de cómputo y las licencias de Neo4J necesarias para poder hacer uso de todas sus funcionalidades. Lo anterior representa un punto no menor y debe tenerse en cuenta, ya que si se cuenta con un volumen de datos muy grande la solución podría volverse costosa.

La falta de licencias correspondientes de Neo4J resultó también en un inconveniente en el desarrollo de este trabajo. Entre otras cosas, este punto imposibilita el guardado de los modelos, forzando a tener que generar y entrenar los modelos cada vez que se reinicie la base de datos. Por otra parte, se tiene la imposibilidad de tener más de tres modelos creados en simultáneo, lo cual genera una limitante a la hora de realizar pruebas entre modelos. La fase de pruebas implicó un continuo descarte y creación de modelos para no sobrepasar dicho límite, lo cual se vuelve tedioso a la hora de experimentar.

Por su parte, es notorio que aún hacen falta desarrollar varias funcionalidades dentro de *GDS* para mejorar el desarrollo de las soluciones. Particularmente, el ajuste fino de hiper parámetros requirió realizar varias pruebas manuales combinando los diferentes parámetros del algoritmo, enlenteciendo sumamente el proceso. Esto presenta una desventaja frente a las implementaciones sobre datos relaciones, donde se cuenta con una gran variedad de bibliotecas para realizar ajustes automáticos, lo cual resulta beneficioso en términos de tiempo y costo.

Adicionalmente, a pesar del tiempo dedicado y las exhaustivas pruebas realizadas durante el ajuste de los hiper parámetros, no se logra alcanzar resultados medianamente satisfactorios en los modelos estudiados. Las dificultades encontradas se atribuyen a la falta de transparencia en el funcionamiento de los algoritmos predictivos de Neo4J. En particular, el algoritmo de Regresión Logística es implementado como una especie de "caja negra", donde no se tiene conocimiento de lo que está ocurriendo dentro.

A modo de ejemplo, no parece ser posible la obtención de los coeficientes beta de la Regresión Logística, imposibilitando el análisis de la importancia de las variables e impidiendo entender dónde invertir tiempo para mejorar los resultados. Por su parte, la evolución de los valores de la función de pérdida (*Loss*) sólo es accesible mediante la consola de *log*, lo cual resulta muy poco intuitivo y práctico.

Como un posible trabajo futuro, en caso de en algún momento poder contar con datos sobre los identificadores personales de los clientes, resulta interesante poder probar otros tipos algoritmos y realizar una comparación con los resultados obtenidos en este trabajo. No solamente en cuanto a las métricas de rendimiento de los modelos de predicción de fraude, sino también en cuanto al desempeño computacional. En el ámbito de las bases de datos de grafos, hay mucho para analizar en cuanto al desempeño computacional de los algoritmos, lo cual es importante optimizar por el costo de cómputo que conllevan este tipo de bases de datos.

Esto último es una de las razones por la cual los grafos vienen creciendo en su uso, ya que para soportar un gran volumen de datos es necesario acompañarlo con capacidad de cómputo, lo cual se mejora cada día más con los avances tecnológicos. Esta también representa una razón por la que para algunas aplicaciones se hace inviable el uso de este tipo de formato de datos y se opta por modelos relacionales.

En cuanto a la implementación de modelos de clasificación binaria sobre bases de grafos, en el proceso de este trabajo se identificó que aún hay muchas cosas por agregar y mejorar en la biblioteca *GDS*, generando que utilizar datos en modelos relacionales sea un proceso más eficiente en este caso.

Dentro de las dificultades encontradas se destaca la ausencia de una comunidad consolidada y la falta de otras implementaciones publicadas. A pesar de esto, las funcionalidades de visualización y la información que aportan las métricas de centralidad resultaron interesantes y útiles para el caso de uso estudiado. Como otro posible trabajo futuro, se propone la implementación de estos algoritmos para enriquecer un análisis de clasificación binaria tradicional, combinando el uso de grafos para potenciar la información provista por los datos del modelo relacional.

REFERENCIAS

[1] Neo4j, "The neo4j graph data science library manual v2.0." https://neo4j.com/docs/graph-data-science/current/, 2022.

[2] Neo4j, "Fraud detection using neo4j platform and paysim dataset." https://github.com/neo4j-graph-examples/fraud-detection/blob/main/documentation/fraud-detection.adocfraud-detection-using-neo4j-platform-and-paysim-dataset.

[3] D. Voutila, "Simulating mobile money fraud." https://www.sisu.io/posts/paysim/, Febrero 2020.

[4] Neo4j, "Neo4j graph algorithms." https://neo4j.com/docs/graph-data-science/current/algorithms/, 2022.

[5] Neo4j, "Comunity detection algorithms." https://neo4j.com/docs/graph-data-science/current/algorithms/similarity/, 2017.

[6] Neo4j, "Similarity algorithms." https://neo4j.com/docs/graph-data-science/current/algorithms/similarity/, 2022.

[7] Neo4j, "Hits algorithm." https://neo4j.com/docs/graph-data-science/current/algorithms/centrality/, 2022.

[8] E. Lopez-Rojas, "Synthetic financial datasets for fraud detection." https://www.kaggle.com/datasets/ealaxi/paysim1, 2017.

[9] S. Pérez and F. Silberberg, "Repositorio del proyeto del curso bases de datos no relaciones." https://gitlab.fing.edu.uy/maria.sofia.perez/proyecto_bdnr_2022, 2022.

[10] Neo4J, "Graph projections." https://neo4j.com/docs/graph-data-science/current/management-ops/graph-catalog-ops/, 2022.

[11] Neo4j, "Eigenvector centrality algorithm." https://neo4j.com/docs/graph-data-science/current/algorithms/eigenvector-centrality/, 2022.

[12] G. Roffo and S. Melzi, "Feature selection via eigenvector centrality." https://www.researchgate.net/profile/Giorgio-Roffo/publication/305918391_Feature_Selection_via_Eigenvector_Centrality/links/586523d708ae329d6204556a/Feature-Selection-via-Eigenvector-Centrality.pdf, 2016.

[13] Neo4j, "Page rank algorithm." https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/, 2022.

[14] M. Needham and A. E. Hodler, "A comprehensive guide to graph algorithms in neo4j." https://go.neo4j.com/rs/710-RRC-335/images/Comprehensive-Guide-to-Graph-Algorithms-in-Neo4j-ebook-EN-US.pdf, 2021.

[15] Neo4j, "Hits algorithm." https://neo4j.com/docs/graph-data-science/current/algorithms/hits/, 2022.

[16] T. Chou, "Hits algorithm: Link analysis explanation and python implementation from scratch." https://towardsdatascience.com/hits-algorithm-link-analysis-explanation-and-python-implementation-61f0762fd7cf, 2021.

[17] Neo4j, "Fast random projection algorithm." https://neo4j.com/docs/graph-data-science/current/machine-learning/node-embeddings/fastrp/, 2022.