Proyecto Final de Curso

Renzo Beux *Estudiante de grado de Computación*
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
renzobeux@gmail.com
Agustín Tornaría *Estudiante de grado de Computación*
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
atornaria@fing.edu.uy
Joaquín Menes *Estudiante de grado de Computación*
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
joaquinmenes@fing.edu.uy

Resumen

Este informe abarca un estudio de técnicas de data science y algoritmos de machine learning para grafos utilizando Neo4J Graph Data Science.

I. Introducción

En la actualidad se cuenta cada vez con más datos, con los que, manejados adecuadamente, se pueden utilizar para resolver varios problemas y obtener información muy valiosa. Para ello se involucran numerosas áreas, desde el manejo de distintas bases de datos, análisis y visualización de los mismos, aplicación de técnicas de machine learning entre otras.

Estos datos son modelados de distintas maneras con lo que es necesario ser capaces de manejar los mismos en distintos formatos, procesarlos adecuadamente y adaptar los algoritmos según el contexto.

En los últimos años hubo un crecimiento de la utilización de bases de datos no relacionales[1]. Con el auge de redes sociales, interconexiones entre distintas entidades, entre otros; los datos cada vez tienen más relaciones intrínsecas, esto permite que bases de datos de grafos sean cada vez más utilizadas para el análisis y modelado de los mismos. Existen varias aplicaciones de data science sobre grafos de gran utilidad, como detección de fraudes, sistemas de recomendación a través de predicción de links, segmentación de usuarios, clasificación de nodos, etcétera.

El presente informe abarca un estudio de técnicas de data science y algoritmos de machine learning para bases de datos de grafos en Neo4J. Se enfoca en la utilización de la biblioteca Neo4J Graph Data Science, la cual ofrece un kit de herramientas para facilitar el análisis de datos y la ejecución de algoritmos de data science, permitiendo al usuario ejecutar estos algoritmos como si fueran consultas Cypher.

Se analiza algunas de las facilidades que brinda la biblioteca, así también como las limitaciones y posibles mejoras de la misma. Para lograr el cometido se realizan pruebas sobre 3 datasets. Se empieza probando entrenar un modelo para realizar link prediction sobre el dataset “Cora“. En segunda instancia se utiliza un dataset sintético el cual se le varian la cantidad de nodos generados para poder probar la escalabilidad del algoritmo de link prediction. Por último se realiza un análisis sobre el dataset ”Twitch”, sobre el cual luego se le aplica el algoritmo node classification provisto por la biblioteca para poder clasificar a los nodos.

II. Neo4J Graph Data Science

Neo4J Graph Data Science (GDS a partir de este momento) es una biblioteca de Neo4J que permite al usuario utilizar implementaciones de algoritmos de grafos como si fuesen consultas Cypher, entre ellos algoritmos de centralidad, detección de comunidades, clustering y embeding entre otros.

II-A. Utilización de GDS

En el momento de escritura la ultima versión estable es la 2.1 que es la que se utilizara a lo largo del proyecto. Dado un grafo, para utilizar los algoritmos de GDS se debe proyectar el grafo, lo cual crea una copia en memoria del mismo en donde se corren los algoritmos. Los algoritmos pueden ser ejecutados de diferentes maneras, en modo *Stream* el cual devuelve el resultado de la consulta, *Mutate* el cual modifica el grafo proyectado en memoria y *Write* que escribe los cambios en el grafo de la base de datos.

GDS tiene 3 tipos de funciones, las que se encuentran en producción que son aquellas que ya fueron probadas en cuanto a estabilidad y escalabilidad, luego las funciones beta que están listas para ser probadas en un futuro cercano y por ultimo las que se encuentran en alpha que son funciones experimentales y que pueden ser eliminadas o cambiadas en un futuro.

Es importante tener en cuenta que para usar GDS es necesario instalar la biblioteca. La forma más fácil de hacer esto es utilizando Neo4J Desktop tal como se ilustra en la siguiente imagen:

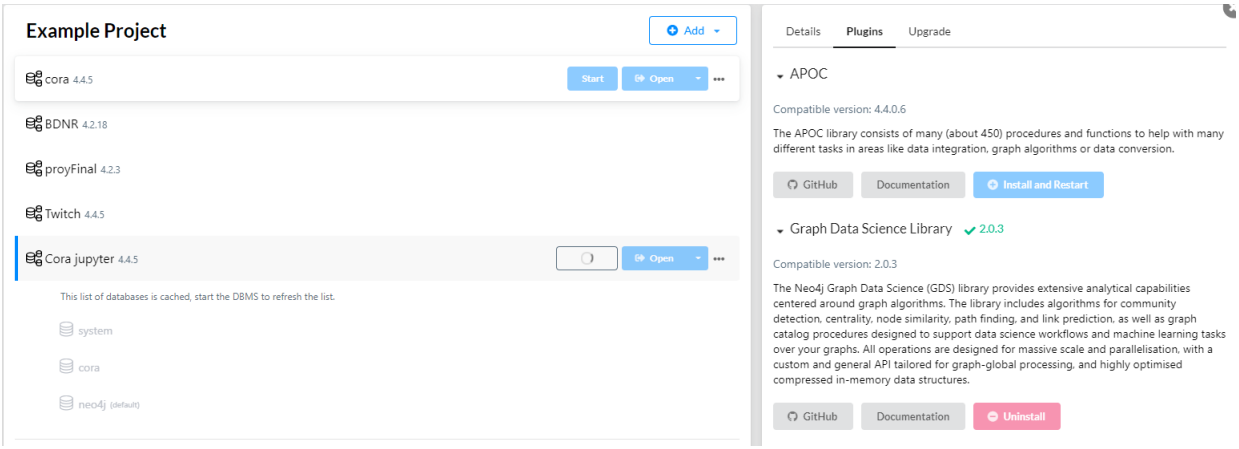

Figura 1: Instalando plugin GDS

II-B. Herramientas para análisis de datos

Para realizar análisis sobre el dataset de Twitch se utilizaron funciones relacionadas a los algoritmos de grafos, estas se dividen en diferentes categorías:

- Centrality
- Community detection
- Similarity
- Topological link prediction

El algoritmo de centralidad utilizado fue “Page Rank”, este se utiliza para determinar la importancia de distintos nodos en una red. Para realizar detección de comunidades se utilizó el algoritmo de “Louvain”, este es utilizado para evaluar cómo se agrupan o dividen los grupos de nodos, así como su tendencia a fortalecerse o dividirse. El algoritmo de similitud utilizado fue “K-Nearest Neighbors” el cual calcula la similitud de pares de nodos en función de sus vecindarios o sus propiedades. El algoritmo de topological link prediction utilizado fue “Adamic Adar”, este ayuda a determinar la cercanía de un par de nodos utilizando la topología del gráfico.

II-C. Pipelines para machine learning

La pipeline es un conjunto de ejecuciones organizadas que permite ejecutar una tarea en concreto. En este caso las pipeline permiten entrenar y predecir usando un modelo en especifico. Estas pipeline se crean usando GDS las cuales luego pueden ser configuradas para permitir un uso especifico. Estas cuentan con 5 configuraciones las cuales permiten realizar la tarea de forma especifica a la situación que se presenta:

- Split
- Node properties
- Features
- Agregar candidatos de modelos
- Configurar el autotunning

Split permite configurar de que manera se particionará los datos que tome la pipeline a la hora de entrenar un modelo. En machine learning es muy común separar los datos en conjuntos, TEST y TRAIN, este proceso lo realiza automáticamente la pipeline siguiendo las preferencias configuradas. Node properties permite agregarle a la pipeline algoritmos de GDS en modo mutate para generar propiedades en los nodos. Un ejemplo típico son los algoritmos de embeding como FastRP y Node2Vec. Feature varia dependiendo si es una pipeline para link prediction o node classification, pero a grandes rasgos permite configurarle a la pipeline cuales van a ser las features que el modelo de machine learning va a usar para aprender. También permite agregarle varios modelos como candidatos, permitiéndole a la pipeline elegir el mejor de ellos en el momento del entrenamiento. Por último permite configurar los parámetros del auto-tunning los cuales en este informe no se utilizan.

Luego que la pipeline esta configurada como se desea esta se puede usar para entrenar el modelo y luego poder realizar predicciones con el mismo.

III. Cora

Para comenzar el estudio se utiliza el DataSet CORA de citaciones entre distintos papers. El mismo es cargado a partir de un código extraído del repositorio de GitHub de Neo4J[5][6]. Este DataSet es un conjunto de documentos perteneciente a temáticas de Machine Learning. Hay 2708 nodos y 5429 relaciones.

El único tipo de nodo presente es :Paper el cual contiene un vector de 1433 enteros, donde cada posición está asociada a una palabra y su valor depende de su ocurrencia en el paper, además tiene una propiedad en la cual se los clasifica entre varios tipos de temáticas como puede ser “Neural_Networks“, “Genetic_Algorithms“ entre otros. Estas temáticas están codificadas usando enteros de la siguiente manera:

```
{
        "Neural_Networks": 0,
        "Rule_Learning": 1,
        "Reinforcement_Learning": 2,
        "Probabilistic_Methods": 3,
        "Theory": 4,
        "Genetic_Algorithms": 5,
        "Case_Based": 6
}
```

Los distintos nodos están relacionados usando la relación :CITES la cual representa que el nodo de origen tiene una cita que hace referencia al nodo de destino.

III-A. Link Prediction

Con este DataSet se ejecuta el algoritmo de “Link Prediction”. En este caso, dado un subconjunto de papers y enlaces de citas, se quiere predecir que papers que se citan entre sí.

Para entrenar el modelo es necesario contar con un conjunto de datos ‘Features‘ (para la pipeline ‘Outer_Train‘) que se usan como entrada del modelo, un conjunto ‘Train‘ con el cual se comparan las predicciones del modelo y se ajusta el modelo y uno de ‘Test‘ para medir los resultados, la forma que se hace esto para este dataset es dividir el conjunto de todas las relaciones en 3, de lo que se encarga automáticamente el split del pipeline.

Luego de cargar en memoria el grafo con etiquetas y relaciones, se crea el pipeline y se configura. Para ello se añade un embedding utilizando FastRp, se configura el split, las features a usar y el modelo a entrenar, en este caso una regresión logística. Por último se entrena el modelo y se usa para predecir. Todo ese proceso usando GDS es bastante simple y fácil de seguir.

III-B. Pruebas

Se prueba con varias cantidades de épocas en el modelo de regresión logística, obteniendo resultados similares al momento de entrenar pero mejores resultados al momento de predecir a medida que las épocas aumentan.

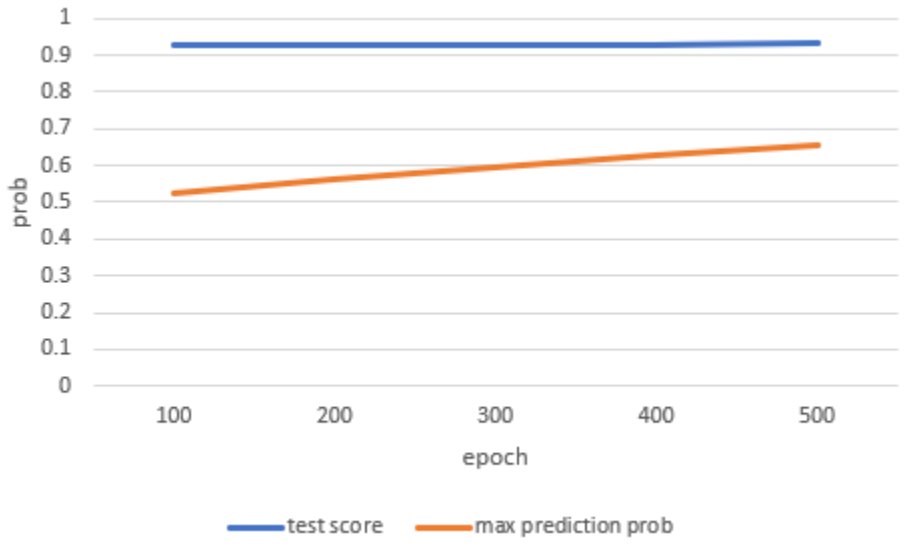

Figura 2: Gráfica de estadísticas de entrenamiento de Cora

Se utiliza AUCPR para medir la performance y al entrenar con 500 epócas se consigue un test score de 0.93.

Además se utiliza este modelo para predecir los 100 links más probables y se modifica el grafo en memoria agregando estas relaciones con una propiedad “probability“ que contiene la probabilidad. Estás relaciones que se agregan representan posibles citas entre papers los cuales tienen una probabilidad del 66 % de ocurrir.

Para corroborar si tiene sentido alguno las relaciones que se agregaron se realiza una query que imprime los subjects de los nodos y se obtiene la siguiente tabla:

	sourceNodeId	targetNodeId	sourceSubject	targetSubject	probability
0	0	751	Neural Networks	Neural Networks	0.656841
1	1	316	Rule Learning	Rule Learning	0.656841
2	31	708	Neural Networks	Neural Networks	0.656841
3	31	736	Neural Networks	Neural Networks	0.656841
4	44	116	Case Based	Case Based	0.656841
5	44	225	Case Based	Case Based	0.656841
6	44	760	Case Based	Case Based	0.656841
7	49	222	Case Based	Case Based	0.656841
8	69	149	Case Based	Case Based	0.656841
9	73	113	Reinforcement Learning	Neural Networks	0.656841
10	73	175	Reinforcement Learning	Neural Networks	0.656841
11	73	235	Reinforcement Learning	Neural Networks	0.656841
12	73	315	Reinforcement Learning	Neural Networks	0.656841
13	73	336	Reinforcement Learning	Neural Networks	0.656841
14	73	349	Reinforcement Learning	Neural Networks	0.656841
15	73	369	Reinforcement Learning	Neural Networks	0.656841
16	73	545	Reinforcement Learning	Neural Networks	0.656841
17	73	599	Reinforcement Learning	Neural Networks	0.656841
18	73	620	Reinforcement Learning	Neural Networks	0.656841
19	73	628	Reinforcement Learning	Neural Networks	0.656841
20	91	458	Neural Networks	Neural Networks	0.656841
21	113	73	Neural Networks	Reinforcement Learning	0.656841
22	113	175	Neural Networks	Neural Networks	0.656841
23	113	235	Neural Networks	Neural Networks	0.656841
24	113	315	Neural Networks	Neural Networks	0.656841
25	113	336	Neural Networks	Neural Networks	0.656841
26	113	349	Neural Networks	Neural Networks	0.656841
27	113	369	Neural Networks	Neural Networks	0.656841
28	113	545	Neural Networks	Neural Networks	0.656841
29	113	599	Neural Networks	Neural Networks	0.656841
30	113	620	Neural Networks	Neural Networks	0.656841

Figura 3: Primeras 30 predicciones de CORA al entrenar con 500 épocas

En la tabla se puede observar como la mayoría de las predicciones de links son entre nodos con la misma temática, lo que es esperable que muchas de las citas entre papers compartan temática. Los nodos que no cumplen esto, es decir que la temática de sus nodos es distinta, tienen temática “Reinforcement Learning“ y “Neural Networks“, pero parece bastante normal que un paper de Reinforcement Learning cite a uno de Neural Networks.

Por último se ilustra algunos de los nodos con las relaciones originales y las nuevas:

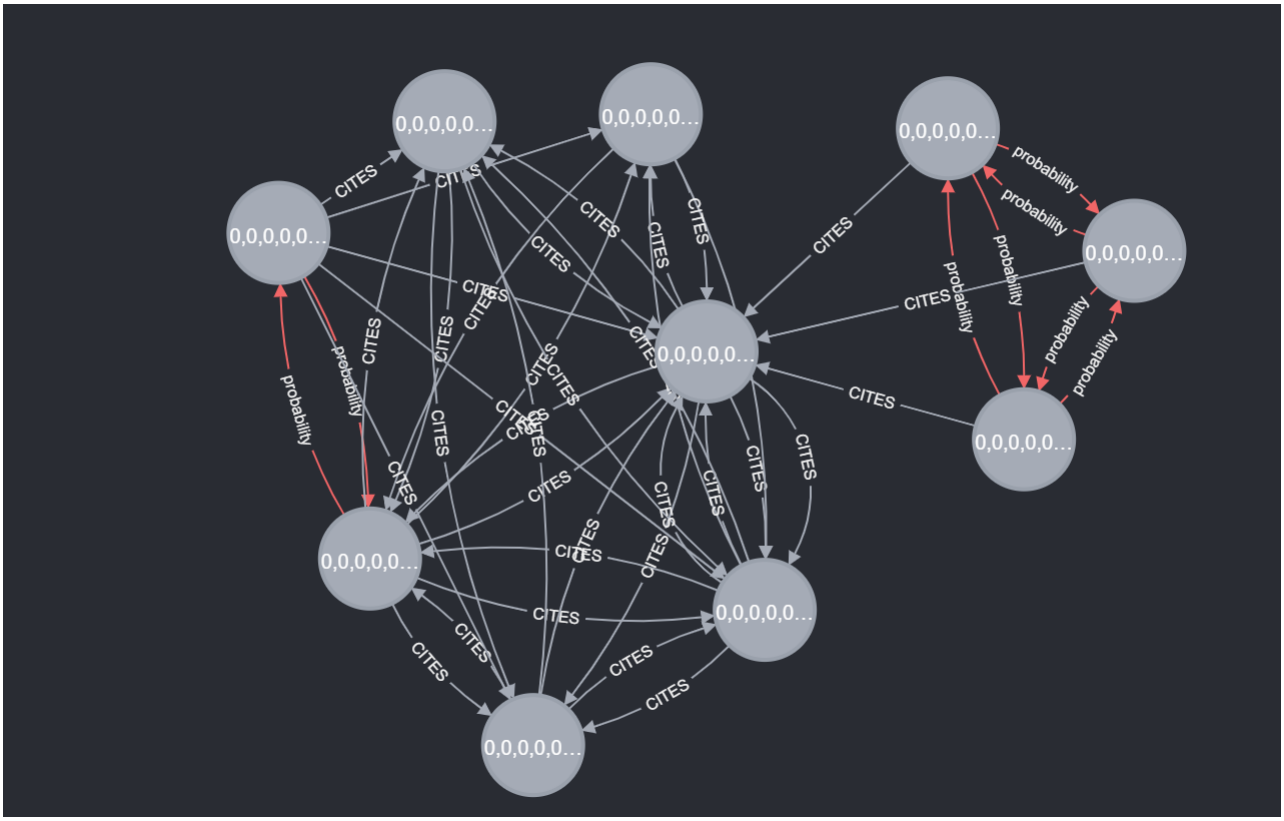

Figura 4: Segmento de grafo resultante luego de aplicar Link Prediction con 500 épocas

Al ser un DataSet muy pequeño, los tiempos de proyección del grafo, entrenamiento y predicción son relativamente bajos incluso para altos valores de iteración del modelo de regresión logística. Si bien es un DataSet fácil de trabajar como para introducir el tema, este no representa un desafío al momento de evaluar la escalabilidad.

Se tiene un Jupyter Notebook "coraLinkPrediction.ipynb"que puede ser accedido desde el repositorio gitlab del proyecto[2] con todos los pasos a seguir para poder llevar a cabo las pruebas realizadas con este DataSet junto con algunas explicaciones de los mismos.

IV. Social

Social es un DataSet sintético que intenta simular una red social, teniendo nodos de usuarios los cuales pertenecen a comunidades y están relacionados entre sí (como si fueran los amigos de Facebook u otra conexión de otra red social). Además las relaciones están ubicadas temporalmente en 3 períodos.

El mismo es generado de forma sintética a partir de un código extraído del repositorio de GitHub de Neo4J[5][6]. Este código permite a partir de parámetros generar grafos de distintas cantidades de nodos. Este dataset sintético abre la posibilidad entonces de hacer prueba de escalabilidad para los algoritmos propuestos.

IV-A. Link Prediction

Se crea una pipeline para realizar Link Prediction sobre 'Social'. Dado el grafo con nodos de usuarios relacionados entre si, pertenecientes a comunidades, se pretende predecir posibles relaciones nuevas entre los usuarios.

Gracias a que el datset cuenta con información temporal surgen dos posibilidades a la hora de obtener los conjuntos 'Features', 'Train' y 'Test'. Dividir el conjunto como fue hecho anteriormente, es decir tomando una fracción de los datos de manera aleatoria para cada uno de estos conjuntos. O dividir el conjunto utilizando la información temporal, usando todos los nodos que ocurren en el primer intervalo de tiempo como 'Feature', el segundo intervalo como 'Train' y el último intervalo como 'Test'.

Se cree que podría ser de gran interés probar con la división temporal pero se encontró una gran limitación en la biblioteca GDS. La pipeline no tiene manera de configurar quienes son estos 3 conjuntos, la misma ejecuta de manera automática la función split y los resultados del mismo son usados como conjuntos de 'Features', 'Test' y 'Train', pudiendo configurar que fracción de los datos son utilizados para cada uno, entre otras configuraciones, pero nunca dar los conjuntos de otra manera.

Por lo tanto se vuelve a utilizar la división aleatoria generada por la función split. Los pasos que se siguen son entonces similares a los realizados anteriormente, cambiando como se configuran algunos parámetros. En este caso se usa nuevamente FastRP para generar el embedding y el modelo que se usa es el de regresión logística con 1000 épocas.

IV-B. Experimentación con cantidad de nodos

Se utiliza el dataset de Social para probar la escalabilidad de algoritmos de machine learning en la cantidad de nodos. Para ello se saca provecho de contar con una base de datos sintética, creando bases de datos con distinta cantidad de nodos, partiendo desde 1000 usuarios y aumentando la cantidad de 1000 en 1000, hasta obtener 10000 usuarios. Sobre estas bases se ejecuta la pipeline creada anteriormente para predicción de links.

Para visualizar los resultados se hace una gráfica del tiempo en función de la cantidad de usuarios con los puntos obtenidos, además se realiza una regresión con la que se obtiene una aproximación cuadrática la cuál también se agrega a la gráfica:

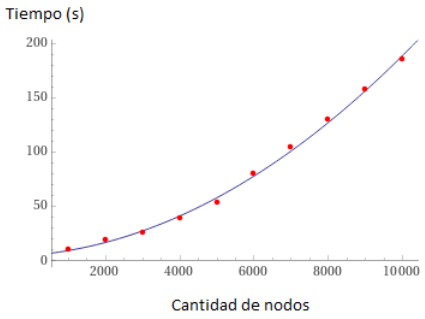

Figura 5: Gráfica del tiempo de entrenamiento en función de la cantidad de usuarios al entrenar con 1000 épocas

Se observa que la cantidad de relaciones crece cuadráticamente con la cantidad de usuarios. Por esta razón se realiza otra gráfica del tiempo en función de la cantidad de relaciones, incluyendo una regresión que aproxima linealmente:

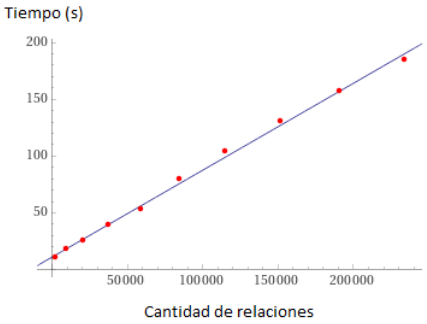

Figura 6: Gráfica del tiempo de entrenamiento en función de la cantidad de relaciones al entrenar con 1000 épocas

Se puede ver como el tiempo necesitado para entrenar el modelo con la misma cantidad de épocas aumenta cuadráticamente en función de la cantidad de usuarios. Pero el tiempo aumenta linelamente en función de la cantidad de relaciones.

Se tiene un Jupyter Notebook “socialLinkPrediction.ipynb”que puede ser accedido desde el repositorio gitlab del proyecto[2] con todos los pasos a seguir para poder llevar a cabo las pruebas realizadas con este DataSet junto con algunas explicaciones de los mismos.

V. Twitch

El dataset de Twitch es un dataset extraído en un intervalo de 3 días de la plataforma Twitch provisto por el repositorio de Neo4J [7] . Se utilizó el dump ’twitch-40.dump’. En este dataset podemos encontrar 4.680.870 nodos y 10.076.938 relaciones. El modelo del grafo es el siguiente:

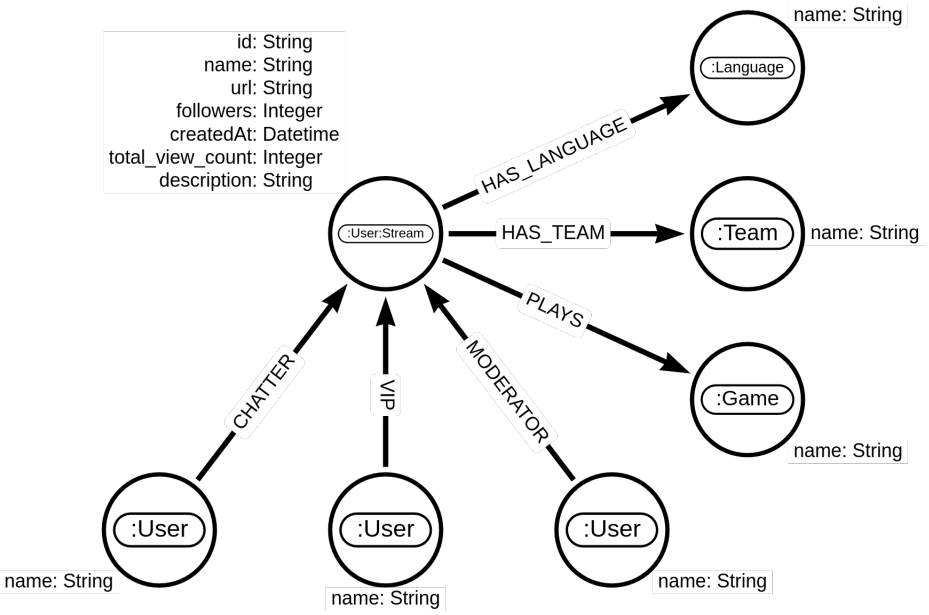

Figura 7: Modelo de la base de datos Twitch

La imagen explica por si sola el modelo del grafo, se cree importante aclarar que la relación CHATTER hace referencia a los usuarios que chatearon, VIP son aquellos usuarios que se subscribieron (pagando) y MODERATOR son aquellos usuarios con el rol de moderador en el canal del streamer.

V-A. Análisis de datos

Para realizar el estudio sobre el dataset de Twitch se comenzó utilizando GDS. Es interesante utilizar distintos algoritmos para analizar el grafo, tales como la centralidad, detección de comunidades, “link prediction” utilizando la topología del grafo, y los k-vecinos más cercanos. Se analizaron distintos grafos, en el primero se proyectan los nodos “Users” y “Stream” con las posibles relaciones entre ellos, “CHATTER”, ”MODERATOR”, y “VIP”.

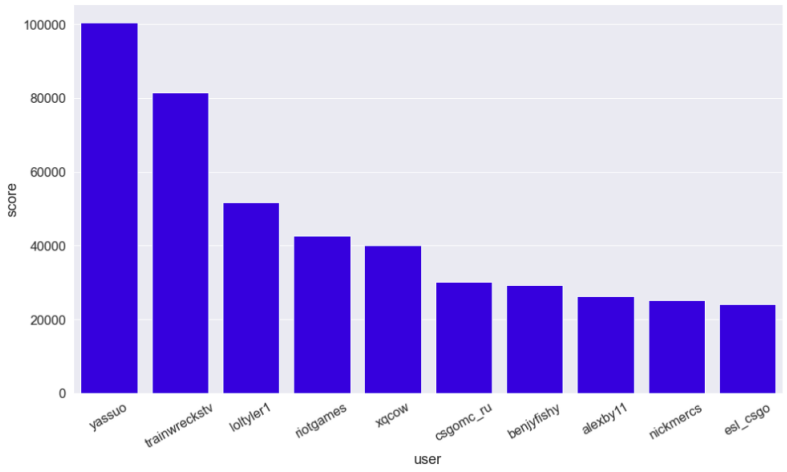

Figura 8: Resultado de aplicar el algoritmo PageRank al primer grafo

En la figura 8 se puede ver que todos los usuarios que aparecen son Streamers, ya que la importancia de cada nodo en este algoritmo esta dada por la cantidad de relaciones entrantes y la importancia de los nodos fuente. Por otro lado es relevante mencionar un análisis realizado en un post en “towardsdatascience”[3] donde además de ver el score, se ven cuantos streamers y cuantos usuarios interactuaron con determinado streamer.

	streamer	score	relationships_from_streamers	relationships_from_users
0	yassuo	100479.020062	16	22632
1	trainwreckstv	81438.905166	101	83934
2	loltyler1	51774.700701	9	44564
3	riotgames	42755.563305	128	316735
4	xqcow	40166.719242	105	280443
5	csgomc_ru	30045.479648	39	221012
6	benjyfishy	29315.003129	57	65097
7	alexby11	26306.094834	4	16573
8	nickmercs	25194.171607	18	85486
9	esl_csgo	24080.771476	61	225852

Figura 9: Relaciones desde streamers y usuarios a un streamer

	other_streamers
0	m0e_tv
1	benjyfishy
2	sanchovies
3	macaiyla
4	odablock
5	alfie
6	trainwreckstv
7	m0xyy
8	nemob
9	dogdog
10	sonecarox
11	kalleymae
12	loltyler1
13	sweezy
14	baboabe
15	taiwanstriker

Figura 10: Streamers que interctuan con el streamer “yassuo”

Fácilmente se puede visualizar que no solo importa la cantidad de relaciones entrantes sino también la importancia de estos nodos, ya que al analizar la figura 10 se ve que entre los streamers que interactuan con el streamer “yassuo” están “trainwreckstv”, “loltyler1” y “benjyfishy”, que forman parte del top 10, por esto es que en la figura 9 por más que “trainwreckstv” tenga más relaciones entrantes, “yassuo” está por encima de él.

En el segundo grafo se proyectan los nodos “Stream”, “Game” y “User” con la relación “PLAYS”, “CHATTER”, ”MODERATOR”, y “VIP”. En este grafo se aplicará el algoritmo de “pageRank”(Figura 11) que mide la importancia de cada nodo dentro del grafo, y se comparará con una “Cypher query”(Figura 12).

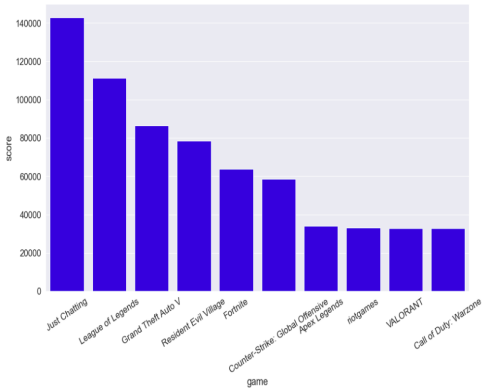

Figura 11: Grafo que muestra los juegos ordenados según importancia del nodo

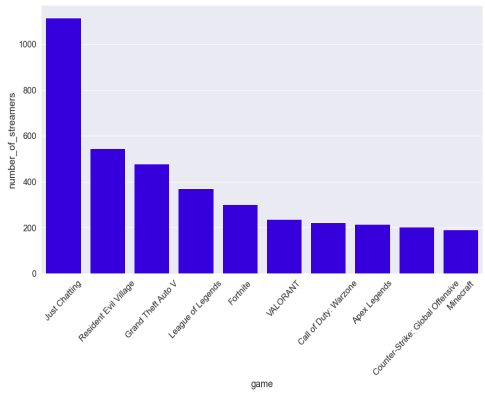

Figura 12: Grafo que muestra los juegos ordenados según la cantidad de streamers

Como se puede ver hay diferencias entre los grafos, por lo tanto, se intuye que existen más factores que la cantidad de streamers que juegan para definir la importancia de juego en este grafo. Por ejemplo, en la gráfica de la Figura 11 se tiene a “League of Legends” en la posición 2 y “GTA V” en la posición 3, y en la gráfica de la Figura 12 se tiene a “League of Legends” en la posición 4 y “GTA V” en la posición 3. Utilizando los datos de la parte anterior, se puede ver que dado el top 10 de streamers, tres de ellos juega al “League of Legends”(top 1, 3 y 4) y uno a “GTA V”(top 5); este es uno de los factores por el cual el nodo que representa al “League of Legends” es más importante que el nodo que representa a “GTA V”.

En el tercero se proyectan los nodos “Stream” con la propiedad “followers” y “total_views_count” y se utiliza el algoritmo “K-Nearest Neighbors” para calcular la similitud entre pares de nodos. Se utilizaron los 10 vecinos más cercanos. Este algoritmo se aplica sobre el nuevo grafo generado y se filtra para comparar los streamers más relevantes obtenidos en el primer grafo.

	Streamer1	Streamer2	similarity
0	nickmercs	xqcow	0.000021
1	esl_csgo	riotgames	0.000003
2	esl_csgo	loltyler1	0.000002
3	riotgames	xqcow	0.000002
4	nickmercs	riotgames	0.000002
5	nickmercs	esl_csgo	0.000001

	Streamer1	Streamer2	similarity
0	nickmercs	xqcow	1.067444e-05
1	esl_csgo	riotgames	1.472243e-06
2	esl_csgo	loltyler1	1.144487e-06
3	riotgames	xqcow	8.528833e-07
4	nickmercs	riotgames	7.897600e-07
5	nickmercs	esl_csgo	5.148855e-07

Figura 13: Comparación

En la primera figura se aplica solo utilizando la propiedad de “followers” y se puede observar que comparado con la otra figura donde además de utilizar la propiedad “followers” se utiliza también “total_views_count” esta tiene valores mayores de similaridad, es decir la nueva propiedad aleja aún más estos nodos. Otra observación interesante es que aún aumentando la cantidad de vecinos a 50 los dos primeros streamers no logran aparecer en el listado, es decir, los dos streamers más importantes tienen una gran diferencia de “followers” y de “total_views” entre ellos y entre los demás.

Otra posibilidad que brinda la biblioteca GDS es poder analizar las comunidades, en este caso se toma el grafo original y se le agrega una nueva etiqueta a todos aquellos streamers que juegan al juego ”League of Legends”, luego se proyecta el grafo con estos streamers y todas las relaciones posibles entre ellos (“CHATTER” y “MODERATOR”), luego con ese grafo generado se aplica el algoritmo de Louvain y se le escribe a los nodos “Stream“ una propiedad que indica a que comunidad pertenece. Gracias a Neo4j Bloom se pueden visualizar estas comunidades a continuación.

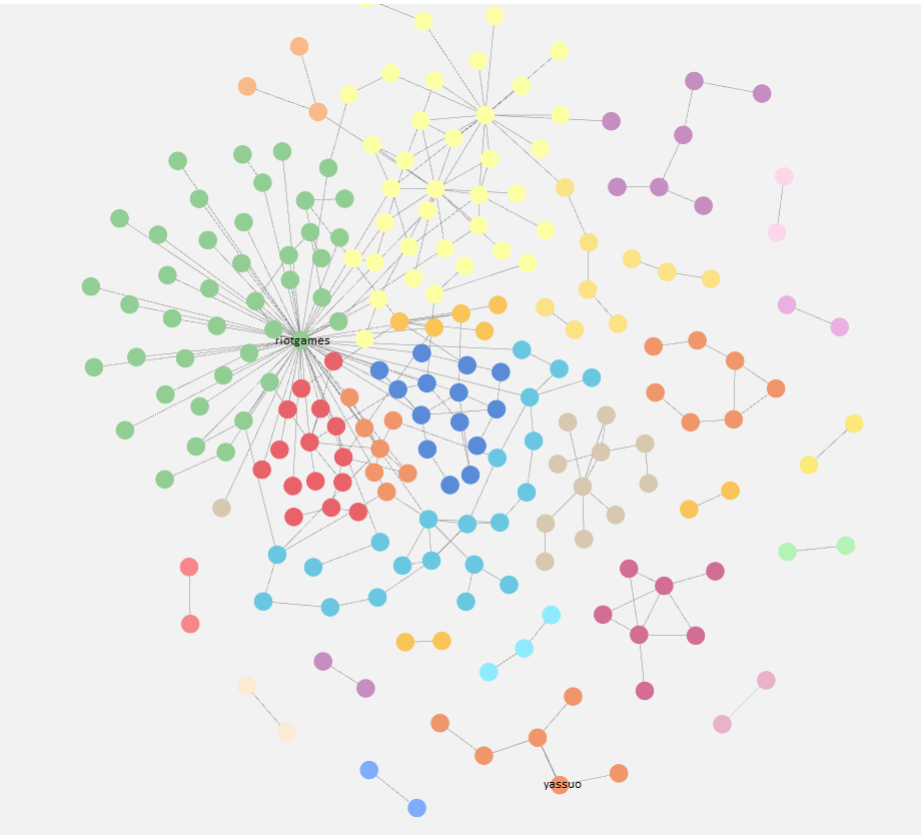

Figura 14: Grafo de comunidades de los streamers que juegan al juego ”League of Legends”

Para la parte de “link prediction” utilizando la topografía del grafo se va a utilizar el algoritmo Adamic Adar, este algoritmo predice enlaces en una red social. Esta predicción se realizará entre dos nodos “Streamer”, “yassuo” y “riotgames”, ya que

son dos streamers relevantes en el grafo, los dos juegan “League of Legends”, no existe una relación que los conecte en este grafo y como se pudo ver en la figura 15 tampoco forman parte de la misma comunidad.

score
3948.660327

Figura 15: Resultado de aplicar Adamic Adar a los nodos “yassuo” y “riotgames”

En este algoritmo un valor de 0 indica que dos nodos no están cerca, mientras que los valores más altos indican que los nodos están más cerca. Por lo tanto la probabilidad de que “yassuo” y “riotgames” interactúen es alta.

Se tiene un Jupyter Notebook “twitchGraphAlgorithms.ipynb”que puede ser accedido desde el repositorio gitlab del proyecto[2] con todos los pasos a seguir para poder llevar a cabo las pruebas realizadas con este DataSet junto con algunas explicaciones de los mismos.

V-B. Node Classification

Luego se procede a intentar predecir el idioma de un streamer, para lo cuál se parte de los pasos presentados en uno de los posts de towardsdatascience [4]. En el post se utiliza GDS para pre-procesar los datos pero luego trabajarlos con SKLearn, osea una biblioteca por fuera de GDS. Se adapta la idea del post, pero utilizando GDS, en especial el pipeline de Node Classification.

Previamente se siguieron una serie de pasos para limpiar el dataset: aquellos datos cuya cantidad de instancias de un idioma fueran muy pequeñas se excluyen, para ello se agrega a los nodos que se quieren ignorar el label :Exclude. También, como cualquier red social hay una inmensidad de bots que perjudican los resultados esperados, a estos también se los excluye.

	user	mod_count
0	nightbot	2384
1	streamelements	1849
2	moobot	1049
3	streamlabs	547
4	ssakdook	190
5	wizebot	165
6	fossabot	134
7	9kmmrbot	38
8	restreambot	29
9	soundalerts	26
10	wzbot	24
11	logiceftbot	23
12	creatisbot	23
13	timeoutwithbits	17
14	papskii_bot	16
15	mikuia	15
16	rewardtts	15
17	louis6321	14
18	teischente	13
19	streamholics	11
20	5crome	11
21	lolrankbot	11
22	crtvly	10
23	songlistbot	10
24	etozhebot	10
25	bu1zer	9

Figura 16: Los moderadores más activos son bots

También se realiza un encoding del idioma ya que al querer proyectar el grafo se tienen problemas con la propiedad ’language’ la cual es una String y este no es soportado por GDS como tipo de propiedad de los nodos.

Entonces se tiene que, al proyectar el grafo, realizarle un encoding al idioma, en donde cada idioma tiene asociado un entero:

	language
0	en
1	es
2	de
3	ru
4	fr
5	ko
6	pt-br
7	it
8	tr
9	zh-tw

Figura 17: Encoding de la propiedad idioma

Se utiliza el algoritmo de Node Similarity con la métrica de Jaccard para relacionar los nodos mediante el link :SHARED_AUDIENCE en donde se guarda la similitud en la propiedad simScore.

Luego de preprocesar los datos se crea y configura una pipeline para realizar Node Classification, intentando predecir el idioma de los streamers. Las configuraciones se realizan de manera similar, ajustando los parámetros para el caso específico. Se utiliza nuevamente FastRp para generar el embedding, esta vez sobre los nodos relacionados mediante :SHARED_AUDIENCE y el modelo que se usa es Random Forest.

Para medir el entrenamiento se utiliza F1 Score con pesos como métrica. Ajustando la pipeline y utilizando 5 arboles de decisión en el modelo de Random Forest se obtiene un test score de 0.88.

Se tiene un Jupyter Notebook “twitchNodeClassification.ipynb”que puede ser accedido desde el repositorio gitlab del proyecto[2] con todos los pasos a seguir para poder llevar a cabo las pruebas realizadas con este DataSet junto con algunas explicaciones de los mismos.

VI. Conclusiones

Se concluye que la herramienta simplifica muchos pasos y permite al usuario ejecutar tareas a través de consultas Cypher, sin necesidad de saber ningún lenguaje de programación ni tener que implementar ningún algoritmo de machine learning desde cero.

Tiene un catalogo amplio de algoritmos comúnmente empleados en grafos para el análisis de datos. Además cuenta con las pipelines para machine learning, las cuales no necesitan demasiada configuración y son faciles de aprender a usar y comenzar entrenando algún modelo.

Al ser una herramienta que simplifica tanto la ejecución de tareas tiene como problemática la limitación de las capacidades de la misma. Entrenar un modelo es fácil, hacerlo funcionar bien no tanto. Es imposible realizar cualquier tarea que se salga un poco del uso planificado de la herramienta, como por ejemplo seleccionar los conjuntos ‘Feature‘, ‘Train‘ y Test‘ sin utilizar automáticamente el split, tal como sucedió con el dataset “social“.

Además las herramientas que provee la biblioteca funcionan como si fueran “cajas negras“, lo cuál por un lado permite utilizarlas de manera sencilla pero se pierde de vista el funcionamiento de estas. Se podría pensar que gracias esto permite entrenar modelos de machine learning sin ningún conocimiento previo, pero esto no es del todo cierto ya que para hacer algo que sirva es necesario tomar decisiones las cuales es imprescindible tener conocimientos.

La documentación de las funcionalidades se encuentra muy completa y ordenada, siendo explicadas con gran claridad y dando ejemplos útiles. Pero aún está en desarrollo y muchas de las funcionalidades que presenta se encuentran en la etapa alpha. Esto es un problema porque mucho de los tutoriales, posts o incluso documentación de hace menos de un año ya se encuentran desactualizados, por lo que dificulta a los usuarios el correcto uso.

Referencias

[1] Matt Asay. *Non-relational’s quiet revolution in databases*. techrepublic. 2021. URL: https://www.techrepublic.com/article/non-relationals-quiet-revolution-in-databases/.

[2] Renzo Beux, Agustín Tornaría y Joaquin Menes. *Repositorio del proyecto*. 2022. URL: https://gitlab.fing.edu.uy/renzo.beux/bdnr04.

[3] Tomaz Bratanic. *Twitchverse: A network analysis of Twitch universe using Neo4j Graph Data Science*. towardsdatascience. 2021. URL: https://towardsdatascience.com/twitchverse-a-network-analysis-of-twitch-universe-using-neo4j-graph-data-science-d7218b4453ff.

[4] Tomaz Bratanic. *Twitchverse: Using FastRP embeddings for a node classification task*. towardsdatascience. 2022. URL: https://towardsdatascience.com/twitchverse-using-fastrp-embeddings-for-a-node-classification-task-bb8d34aa690.

[5] Alicia Frame y Jacob Sznajdman. *Nodes2021 Link Prediction*. Neo4j. 2021. URL: https://github.com/neo-technology/nodes2021-link-prediction.

[6] Neo4j. *9 - Building an ML Pipeline in Neo4j Link Prediction Deep Dive*. Youtube. 2021. URL: https://www.youtube.com/watch?v=qdbhCG-Yn74&t=1s.

[7] Neo4j. *Twitch Network Analysis Graph Example*. Github. 2022. URL: https://github.com/neo4j-graph-examples/twitch.