Proyecto Interciclo

Christian Zhiminaicela

Universidad Politécnica. Salesiana
Cuenca, Ecuador
6zhimi6@gmail.com

***Abstract*— This project is focused on scraping data from the twitter platform, using the python programming language. Also on the application of search algorithms in Neo4j.**

Thanks to the python libraries, we can download the information from the internet and obtain data to be able to manipulate that information.

I. INTRODUCCIÓN

En este documento vamos a encontrar todo acerca del scraping en la red social de twitter, como aplicamos la librería tweepy, que nos permitirá obtener la información de la red social y que métodos podemos usar para dicha implementación, con este proyecto también implementaremos algoritmos de centralidad y mostraremos algunos ejemplos sobre el uso de dichos algoritmos.

II. Scraping in Twitter

- Primero Sacamos los datos de Twitter
- Para realizar este metodo utilizamos la libreria tweepy

Gracias a la librería obtenemos los datos de Twitter, antes de proceder debemos sacar algunos datos de Twitter, como: consumer_key, consumer_secret, acces_token, acces_token_Secret, una vez tenemos esos datos, implemenetamos el siguinete codigo para verificar el acceso a nuestros datos de twitter mediante python.

```
import tweepy
import json, csv, sys

consumer_key = "OwTKTDs0VglLw9WwN3wULDxNb"
consumer_secret = "3zDabqEYJjd2CKHZNcsRaWBJvkY4j9c2NAznirEECN7Z7T5H65"
access_token = "167559772-QAPN52IrQ61jo5bHxkmWYUW5ohVXLyt1XV88yNpo"
access_token_secret = "f0Vs2RHCEoOzg8g9LRnVKqKdP5PTBamALjs21vpyMwv2W"

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)

auth.set_access_token(access_token, access_token_secret)

api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=Tru

#api.me para sacar solo de nuestros propios usuarios
data = api.me()

print(json.dumps(data._json, indent=5))
```

Fig. 1 Código que consume la api de twitter

Después implementamos el siguiente código para buscar los datos de algún usuario en especifico, en este caso nosotros tendremos que scrapiar los datos del candidato a la presidencia Yauku Perez, utilizamos la siguientes líneas de código:

```
data = api.get_user("yakuperezg")
print(json.dumps(data._json, indent=2))

#Numero de usuarios mostrados por la api
data = api.followers(screen_name="yakuperezg")
print(len(data))
```

Fig. 2 Código que scrapea datos de Twitter de Yaku

III. ALGORITMOS

Instalación de paquetes:

Abrimos nuestra terminal y instalamos la libreria py2neo para nuestra conexión a la base de datos.
(pip install py2neo)

A. *Centrality Algoriths- Eigenvector Centrality*

Es un algoritmo que mide la influencia transitiva o la conectividad de los nodos. Las relaciones con los nodos de puntuación alta contribuyen más a la puntuación de un nodo que las conexiones con los nodos de puntuación baja. Una puntuación alta significa que un nodo está conectado a otros nodos que tienen puntuaciones altas.

Puede considerarse como el (puntaje de importancia) de una página web o nodo de red social. Este puntaje de importancia siempre será un número real no negativo y todos los puntajes se sumarán a 1.

Implementacion del algoritmo:

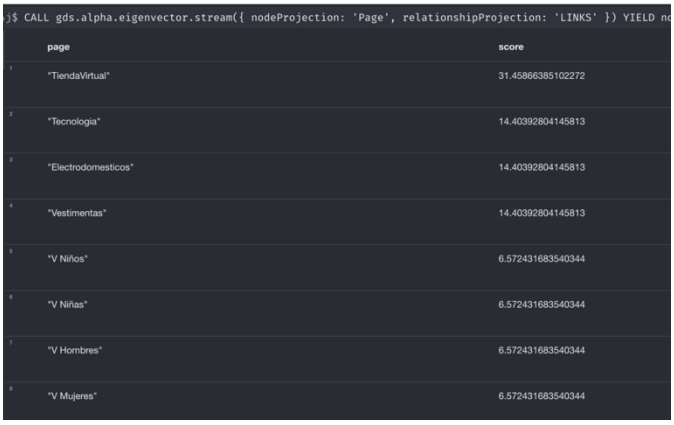
- Primero creamos los nodos:

```
CREATE (tiendaVirtual:Page {name:'TiendaVirtual'}), (tecnologia:Page {name:'Tecnologia'}), (electrodomesticos:Page {name:'Electrodomesticos'}), (vestimentas:Page {name:'Vestimentas'}), (niños:Page {name:'V Niños'}), (niñas:Page {name:'V Niñas'}), (hombres:Page {name:'V Hombres'}), (mujeres:Page {name:'V Mujeres'}), (tiendaVirtual)-[:LINKS]->(tecnologia), (tecnologia)-[:LINKS]->(tiendaVirtual), (electrodomesticos)-[:LINKS]->(tiendaVirtual), (tiendaVirtual)-[:LINKS]->(electrodomesticos), (vestimentas)-[:LINKS]->(tiendaVirtual), (tiendaVirtual)-[:LINKS]->(vestimentas), (vestimentas)-[:LINKS]->(niños), (niños)-[:LINKS]->(tiendaVirtual), (vestimentas)-[:LINKS]->(niñas), (niñas)-[:LINKS]->(tiendaVirtual), (vestimentas)-[:LINKS]->(hombres), (hombres)-[:LINKS]->(tiendaVirtual), (vestimentas)-[:LINKS]->(mujeres), (mujeres)-[:LINKS]->(tiendaVirtual)
```

Fig. 3 Creación de nodos en Neo4j

- Luego procedemos a sacar la centralidad:

  CALL gds.alpha.eigenvector.stream({ nodeProjection: 'Page', relationshipProjection: 'LINKS' }) YIELD nodeId, score RETURN gds.util.asNode(nodeId).name AS page, score ORDER BY score DESC



j$ CALL gds.alpha.eigenvector.stream({ nodeProjection: 'Page', relationshipProjection: 'LINKS' }) YIELD no

	page	score
1	"TiendaVirtual"	31.45866385102272
2	"Tecnologia"	14.40392804145813
3	"Electrodomesticos"	14.40392804145813
4	"Vestimentas"	14.40392804145813
5	"V Niños"	6.572431683540344
6	"V Niñas"	6.572431683540344
7	"V Hombres"	6.572431683540344
8	"V Mujeres"	6.572431683540344

Fig. 4 Resultado de la centralidad

B. *Link Prediction Algorithms- sameCommunity*

Los algoritmos de la misma comunidad es una forma de determinar si dos nodos pertenecen a la misma comunidad. Si dos nodos pertenecen a la misma comunidad, hay una mayor probabilidad de que haya una relación entre ellos en el futuro, si aún no la hay.

Un valor de 0 indica que dos nodos no están en la misma comunidad. Un valor de 1 indica que dos nodos están en la misma comunidad. Si uno de los nodos no tiene una comunidad, esto significa que no pertenece a la misma comunidad que cualquier otro nodo.

Implementacion del algoritmo:

- Primero creamos los nodos:

  CREATE (zhen:Person {name: 'Zhen', community: 1}), (praveena:Person {name: 'Praveena', community: 2}), (michael:Person {name: 'Michael', community: 1}), (arya:Person {name: 'Arya', partition: 5}), (karin:Person {name: 'Karin', partition: 5}), (jennifer:Person {name: 'Jennifer'})

Fig. 5 Creación de nodos

- Lo siguiente indicará que Michael y Zhen pertenecen a la misma comunidad:

  MATCH (p1:Person {name: 'Michael'}) MATCH (p2:Person {name: 'Zhen'}) RETURN gds.alpha.linkprediction.sameCommunity(p1, p2) AS score

```
1 MATCH (p1:Person {name: 'Michael'})
2 MATCH (p2:Person {name: 'Zhen'})
3 RETURN gds.alpha.linkprediction.sameCommunity(p1, p2) AS score
```

neo4j$ MATCH (p1:Person {name: 'Michael'}) MATCH (p2:Person {name: 'Zhen'

Table

	score
1	1.0

Fig. 6 Resultado de la misma Comunidad

C. *Similarity Algoriths – Aproximate Nearest Neighbors(ANN)*

Los algoritmos de similitud calculan la similitud de pares de nodos usando diferentes métricas basadas en vectores. La biblioteca Neo4j GDS incluye los siguientes algoritmos de similitud, agrupados por nivel de calidad

Podemos usar el algoritmo Approximate Nearest Nearest Neighbors para averiguar los elementos k más similares entre sí. La implementación en la biblioteca se basa en el documento de Dong, Charikar. En este caso nos centraremos en la similitud de jaccard en donde se utiliza la siguiente formula: J(A,B) = |A ∩ B| / |A| + |B| - |A ∩ B|

Implementacion del algoritmo:

- Primero creamos los nodos:

  CREATE
  (cuenca:Cuisine {name:'Cuenca'}),
  (sigsig:Cuisine {name:'Sigsig'}),
  (chorde:Cuisine {name:'Chorde'}),
  (gualaceo:Cuisine {name:'Gualaceo'}),
  (quito:Cuisine {name:'Quito'}),
  (christian:Person {name: 'Christian'}),
  (maria:Person {name: 'Maria'}),
  (michael:Person {name: 'Michael'}),
  (juan:Person {name: 'Juan'}),
  (zhimi:Person {name: 'Zhimi'}),

(maria)-[:LIKES]->(chorde),
(maria)-[:LIKES]->(quito),
(christian)-[:LIKES]->(cuenca),
(christian)-[:LIKES]->(chorde),
(michael)-[:LIKES]->(cuenca),
(michael)-[:LIKES]->(sigsig),
(michael)-[:LIKES]->(chorde),
(juan)-[:LIKES]->(gualaceo),
(juan)-[:LIKES]->(sigsig),
(juan)-[:LIKES]->(quito),
(zhimi)-[:LIKES]->(gualaceo),
(zhimi)-[:LIKES]->(sigsig)

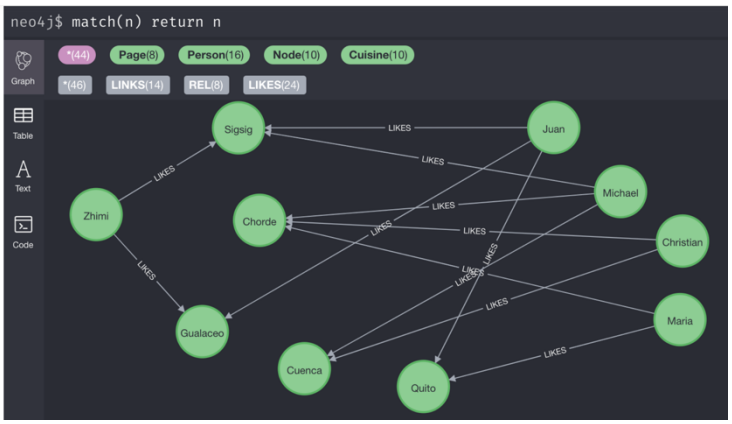


Fig. 7 Creación de los Nodos con sus Relaciones

- Para ver el Resultado en NEO4J, ingresamos el siguiente codigo:

MATCH (p:Person)-[:LIKES]->(cuisine)

WITH {item:id(p), categories: collect(id(cuisine))} AS userData

WITH collect(userData) AS data

CALL gds.alpha.ml.ann.stream({ data: data, algorithm: 'jaccard', similarityCutoff: 0.1, concurrency: 1 })

YIELD item1, item2, similarity

return gds.util.asNode(item1).name AS from,
gds.util.asNode(item2).name AS to,
similarity ORDER BY from

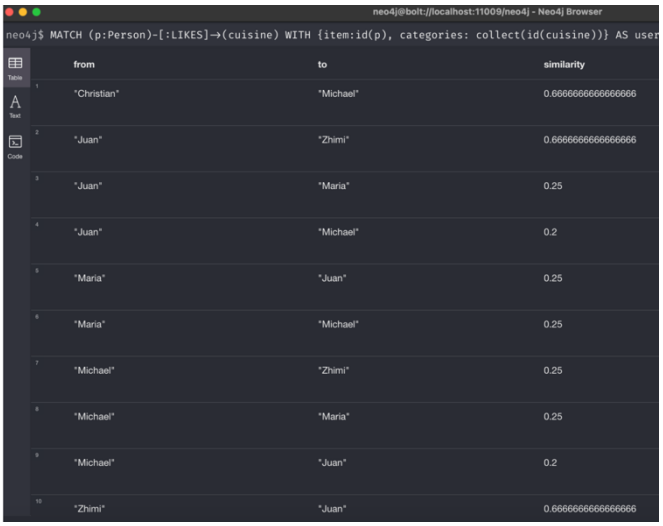


Fig. 8 Resultado de similitud de los Nodos

IV. Conclusiones

Como era de esperaren elalgoritmo de centralidad, la página tiendaVirtual tiene la mayor centralidad de Eigenvector porque tiene enlaces entrantes de todas las demás paginas. También podemos ver que no solo es importante el número de enlaces entrantes, sino también la importancia de las páginas detrás de esos enlaces. El algoritmo de la misma comunidad es implementado para verificar si dos nodos exiten en la misma comunidad, mientras mas cerca el valor del algoritmo este de 1 estan en la misma comunidad, si el algoritmo devuelve el valor de 0, esto quiere decir que no estan en la misma comunidad. El algorimo ANN nos ayuda a obtener la similitud entre nodos y asi poder sugerir a nodos cercanos diferentes gustos o propiedades de un nodo en particular dependiendo de cuanta similitud se tenga entre nodos.

Referencias

[1] "python" obtenido de: https://neo4j.com/developer/python/
[2] "Githud" obtenido de: https://github.com/technige/py2neo/issues/791
[3] "py2neo" obtenido de: https://py2neo.org/v4/database.html
[4] "stackoverflow" obtenido de: https://stackoverflow.com/questions/60023381/securityerror-failed-to-establish-secure-connection-to-eof-occurred-in-violati