Universidad Nacional Autónoma de México

Facultad de Estudios Superiores Acatlán

Proyecto Final

Network Analysis with Stack Overflow Tags

Autores:

González Paz Edgar
Leija Estrada Shanelly Abisay

Ciencia de Datos

Materia:

Datos Masivos II

Febrero 2021

Índice

1. Introducción **2**

2. Objetivo **3**

3. Contexto **4**

4. Descripción de la Información **5**

5. Modelado e Implementación **12**

6. Interpretación de Resultados **19**

7. Conclusiones **21**

8. Referencias bibliográficas **22**

1. Introducción

El análisis de redes, es un campo del análisis de datos, en este se aplica en gran manera la teoría de grafos para entender relaciones sociales. En el presente trabajo, veremos como obtener información de una página web a través de la técnica conocida como web scrapping, además de un vistazo a algunas métricas que hacen más amigable el análisis de redes y como aplicar todo lo anterior para obtener un grafo no dirigido que denote relaciones entre ciertos tópicos. Estos tópicos los tomaremos según datos de la vida real, a través de la plataforma Stack Overflow es por eso que es necesario llevar a cabo un trabajo de scrapeo. Con esto daremos paso a conocer la interacción social de un target específico, este target es justo el de los ususarios de la plataforma antes mencionada, por lo que los tópicos que se toman en cuenta serán muy específicos, más adelante veremos en qué se especializa Stack Overflow, para saber a qué público nos enfrentamos y en qué les puede servir dicho análisis.

2. Objetivo

El objetivo de este trabajo es crear un análisis basado en el comportamiento de aquello que las personas en el área de programación y ciencia de datos suelen preguntarse o les interesa discutir, a través de importantes foros, como lo es Stack Overflow. A su vez, lo que buscamos con este análisis es, estar informados sobre los temas top del momento y como se relacionan entre ellos, de forma que podamos ofrecer a los consumidores patrones o áreas que podrían explorar según los temás o herramientas que ya dominan, encaminándolos a la especialización de sus propias habilidades.

3. Contexto

Figura 1: Stack Overflow.

Stack Overflow es una plataforma prinicpalmente para programadores y aficionados, en la que usuarios hacen preguntas y otros de los mismos usuarios les dan respuesta(un sitio de Q&A), fue creada en 2008 por Jeff Atwood y Joel Spolsky.

Ha ganado gran popularidad y, en general, buena reputación, debido a su amplia gama de tópicos y respuestas, además de que los usuarios pueden votar estas respuestas, según consideren si es correcto o funcional lo que proponen. Las respuestas son publicadas por los miembros de una comunidad determinada o por otros usuarios con las mismas experiencias que encontraron solución al problema planteado.

Un estudio en 2013 encontró que el 77 % de los usuarios sólo hacen una pregunta, el 65 % solamente responden a una pregunta, y solo el 8 % de los usuarios responden a más de 5 preguntas. A partir de 2011, el 92 % de las preguntas fueron contestadas en un tiempo medio de 11 minutos. Desde 2013, el software de red Stack Exchange elimina automáticamente las preguntas que cumplen con ciertos criterios, entre ellos el no tener respuestas en una cierta cantidad de tiempo.

A partir de agosto de 2012, 443.000 de los 1,3 millones de usuarios registrados habían respondido al menos una pregunta, y de ellos, unos 6.000 (0,46 % del número total de usuarios) se había ganado una puntuación de reputación superior a 5000.La Reputación se puede ganar más rápido contestando a preguntas relacionadas con las etiquetas con menor densidad de conocimientos, el hacerlo con prontitud (en particular, siendo la primera persona en contestar una pregunta), estar activo durante las horas de menor uso, y contribuyendo a diversas áreas.

Según dichas etiquetas es que haremos el análisis propuesto. Estas etiquetas las taggea directamente el usuario, y puede poner una o varias a su pregunta.

4. Descripción de la Información

La información fue obtenida directamente del sitio de Stack Overflow a través de las pequeterías de Python: bs4 y request.

Se lleva a cabo un proceso de web scrapping de la siguiente manera:

```
from bs4 import BeautifulSoup as bs
import requests
import pandas as pd
import numpy as np
from itertools import *
```

```
def Web_scrapping_Stack_Overflow(url):
    response = requests.get(url)
    soup = bs(response.content, 'html.parser')
    body = soup.find('body')
    question_links = body.select("h3 a.question-hyperlink")
    questions = [i.text for i in question_links]

    tags_divs = body.select("div.summary > div:nth-of-type(3)")
    a_tags_list = [i.select('a') for i in tags_divs]
    tags = []
    for a_group in a_tags_list:
        tags.append([a.text for a in a_group])
    df = pd.DataFrame({"Preguntas": questions, "Tags": tags})
    return df
```

Figura 2: Definición de la función para web scrapping.

En el código anterior definimos una función, que se encarga de acceder a la url que le proporcionamos, posteriormente lee el contenido de la pagína y selecciona en el sitio de Stack Overflow las preguntas y tags de estas preguntas, para posteriormente depositar estos en un dataframe.

En el siguiente fragmento de código podemos ver como se aplica la función anteriormente explicada, como le decimos el número de urls(páginas) que queremos, en este caso le pedimos que consulte 20 páginas, cada página tiene 50 preguntas por lo que scrappeamos al rededor de 100 pregunta, y finalmente lo guardamos en el dataframe.

```
%%time
df = pd.DataFrame(columns=["Preguntas", "Tags"])
paginas = 20
for i in range(paginas):
    url = "https://stackoverflow.com/questions?tab=votes&page="+str(i)
    df = pd.concat([df, Web_scrapping_Stack_Overflow(url)])
```

```
Wall time: 1min 43s
```

Figura 3: Código para web scrapping.

El dataframe se ve de la siguiente manera:

```
df = df.reset_index(drop=True)
df
```

	Preguntas	Tags
0	Why is processing a sorted array faster than p...	[java, c++, performance, optimization, branch-...
1	How do I undo the most recent local commits in...	[git, version-control, git-commit, undo]
2	How do I delete a Git branch locally and remot...	[git, version-control, git-branch, git-push, g...
3	What is the difference between 'git pull' and ...	[git, version-control, git-pull, git-fetch]
4	What does the “yield” keyword do?	[python, iterator, generator, yield, coroutine]
...	...	...
995	How do I diff the same file between two differ...	[git, git-diff]
996	How can I develop for iPhone using a Windows d...	[ios, iphone, windows]
997	How can I merge two commits into one if I alre...	[git, git-merge]
998	Object comparison in JavaScript [duplicate]	[javascript, object, comparison, object-compar...
999	What is the difference between null and undefi...	[javascript, null, undefined]

Figura 4: Dataframe de preguntas y tags.

Como podemos ver, ya tenemos una amplia gama de pregunta, que están asociadas a los tags que el usuario asignó, según los tópicos que el considera que tienen que ver con su pregunta. Para este análisis en específico no nos interesa tanto saber sobre la pregunta específica, nos interesan los tags y sus relaciones entre ellos mismos, por los que crearemos dos dataframes con estos datos.

```
duplas_tags = []
for i in range(df.shape[0]):
    if len(df["Tags"][i]) < 2:
        pass
    else:
        duplas_tags.append(np.array(list(combinations(df["Tags"][i], 2))))
```

```
duplas_tags = np.concatenate(duplas_tags)
```

```
unique_tags = np.unique(duplas_tags, axis = 0)
```

```
conteo_tags = []
for i in range(len(unique_tags)):
    conteo_tags.append(np.insert(unique_tags[i], 2, np.count_nonzero(duplas_tags == unique_tags[i])))
```

Figura 5: Código para crear dataframe de relaciones .

El dataframe de relaciones contiene tuplas según las combianciones de tags que hay para una misma pregunta, esta información sale del dataframe que creamos en el paso de arriba, al ser relaciones únicas, también agregamos un columna que denota cuantas veces se repite la misma relación sin importar de que pregunta vengan, simplemente hacemos el conteo.

```
df_tags = pd.DataFrame(conteo_tags, columns = ["Tag_1", "Tag_2", "Count"])
```

```
df_tags
```

	Tag_1	Tag_2	Count
0	.net	.net-framework-version	63
1	.net	algorithm	64
2	.net	alias	69
3	.net	asp.net	64
4	.net	breakpoints	64
...	...	...	...
3420	yaml	newline	14
3421	yield	coroutine	10
3422	youtube	youtube-api	5
3423	youtube	youtube-data-api	6
3424	youtube-api	youtube-data-api	5

Figura 6: Dataframe de relaciones.

Ahora agregamos una nueva columna que contiene las ponderaciones de cada conteo con respecto al conteo total de relaciones.

```
df_tags["Proportion"] = (df_tags["Count"].astype("int64") / df_tags["Count"].astype("int64").sum()) * 100
```

```
df_tags = df_tags[(df_tags.Tag_1 != "null") & (df_tags.Tag_2 != "null")].reset_index(drop=True)
```

```
df_tags
```

	Tag_1	Tag_2	Count	Proportion
0	.net	.net-framework-version	63	0.018375
1	.net	algorithm	64	0.018666
2	.net	alias	69	0.020125
3	.net	asp.net	64	0.018666
4	.net	breakpoints	64	0.018666
...	...	...	...	...
3403	yaml	newline	14	0.004083
3404	yield	coroutine	10	0.002917
3405	youtube	youtube-api	5	0.001458
3406	youtube	youtube-data-api	6	0.001750
3407	youtube-api	youtube-data-api	5	0.001458

Figura 7: Dataframe de relaciones con ponderaciones.

Como siguiente paso, creamos una tabla general de tags, que simplemente contiene un lista de todos los tags únicos junto con el conteo de cuantas veces se repiten.

```
tags_uniq = []
for i in range(df.shape[0]):
    if len(df["Tags"][i]) < 1:
        pass
    else:
        tags_uniq.append(np.array(list(combinations(df["Tags"][i], 1))))
tags_uniq = np.concatenate(tags_uniq)
conteo_tags_uniq  = []
for i in range(len(tags_uniq )):
    conteo_tags_uniq .append(np.insert(tags_uniq [i], 1, np.count_nonzero(tags_uniq  == tags_uniq[i])))
```

```
df_tag = pd.DataFrame(conteo_tags_uniq, columns = ["Tag", "Count"])
df_tag
```

	Tag	Count
0	java	66
1	c++	38
2	performance	19
3	optimization	6
4	branch-prediction	2
...	...	...
3437	comparison	3
3438	object-comparison	1
3439	javascript	220

Figura 8: Dataframe de tags únicos.

Agregamos también la columna de proporción según su conteo.

```
df_tag["Proportion"] = (df_tag["Count"].astype("int64") / df_tag["Count"].astype("int64").sum()) * 100
df_tag
```

	Tag	Count	Proportion
0	java	66	0.052082
1	c++	38	0.029986
2	performance	19	0.014993
3	optimization	6	0.004735
4	branch-prediction	2	0.001578
...	...	...	...
3437	comparison	3	0.002367
3438	object-comparison	1	0.000789
3439	javascript	220	0.173606
3440	null	9	0.007102
3441	undefined	6	0.004735

Figura 9: Dataframe de tags únicos con poderaciones.

Exportamos los dos dataframes a formato .csv

A .csv

```
df_tags.to_csv("df_tags.csv", index=False)
df_tag.to_csv("df_tag.csv", index=False)
```

Figura 10: Exportamos a csv.

Nuestro conjuto de datos está listo para ser introducido al proceso de modelado.

5. Modelado e Implementación

Para la creación del grafo, utilizamos la libería networkx.algorithms.community de python.

Para el modelado se define un grafo que se compone de elementos de nuestros dos dataframes:

Para los nodos:

De df_tag, extraemos el tag y asociamos el tamañano del nodo a la proporción (porcentaje de apariciones en el total de preguntas).

Para las aristas:

De df_tags, extraemos las duplas de tags en todas las preguntas y como peso le asignamos la proporción (porcentaje de apariciones de esta dupla de tags en todas las preguntas)

```
G = nx.Graph()
```

```
for index, row in df_tag.iterrows():
    G.add_node(row["Tag"], nodesize = row["Proportion"])
```

```
for index, row in df_tags.iterrows():
    G.add_edge(row["Tag_1"], row["Tag_2"], weight = row["Proportion"])
```

```
print(nx.info(G))
```

```
Name:
Type: Graph
Number of nodes: 630
Number of edges: 1822
Average degree:   5.7841
```

Figura 11: Definición del grafo.

El número de aristas con las que cuenta nuestro grafo son: 630 y los vértices son 1,822. El average degree denota el promedio de vértices con los que cuenta cada arista, en este caso son 5.78.

Con el Algoritmo de Girvan Newman buscamos encontrar comunidades dentro de nuestro grafo y poder encontrar e interpretar las relaciones entre los tags. Se hace el mapeo al dataframe para asignarle a cada tag a que grupo pertenece.

```
%%time
GN = nxcom.girvan_newman(G)
communities = next(GN)
```

```
new = []
for i in range(len(communities)):
    for item in {x for x in communities[i] if pd.notna(x)}:
        new.append([item, i+1])
```

```
map_group = pd.DataFrame(new).set_index(0).to_dict()[1]
```

```
df_tag["group"] = df_tag.Tag.map(map_group)
```

Figura 12: Aplicación del algoritmo Girvan Newman y mapeo.

El dataframe con etiquetas de grupo se ve de la siguiente forma:

df_tag

	Tag	Count	Proportion	group
0	java	37	0.100334	1
1	c++	21	0.056946	1
2	performance	10	0.027117	1
3	optimization	4	0.010847	1
4	branch-prediction	2	0.005423	1
...	...	...	...	...
1765	sql	9	0.024405	1
1766	sql-server	5	0.013559	1
1767	tsql	3	0.008135	1
1768	date	5	0.013559	1
1769	datetime	8	0.021694	1

Figura 13: Dataframe con etiquetas de grupo.

Obtenemos el número de tags por grupo.

```
df_tag.group.value_counts()
```

```
1     1451
2      289
6       10
3        5
9        4
8        3
4        3
10       2
5        2
7        1
Name: group, dtype: int64
```

```
df_tag_ = df_tag[(df_tag.group == 1) | (df_tag.group == 2)]
```

```
tags_interes = list(df_tag_["Tag"])
```

```
df_tags_ = df_tags[df_tags.Tag_1.isin(tags_interes)]
```

Figura 14: Tags por grupo.

Definimos un nuevo grafo G que contenga el atributo de grupo con la etiqueta obtenida después de ejecutar el Algoritmo de Girvan Newman.

```
G = nx.Graph()
```

```
for index, row in df_tag_.iterrows():
    G.add_node(row["Tag"], group = row["group"], nodesize = row["Proportion"] )
```

```
for index, row in df_tags_.iterrows():
    G.add_edge(row["Tag_1"], row["Tag_2"], weight = row["Proportion"])
```

```
print(nx.info(G))
```

```
Name:
Type: Graph
Number of nodes: 603
Number of edges: 1783
Average degree:   5.9138
```

```
%%time
GN = nxcom.girvan_newman(G)
communities = next(GN)
```

```
Wall time: 3min 25s
```

```
communities
```

```
({'.net',
  '.net-framework-version',
  '2048',
```

Figura 15: Definición del nuevo grafo.

El grafo es el siguiente

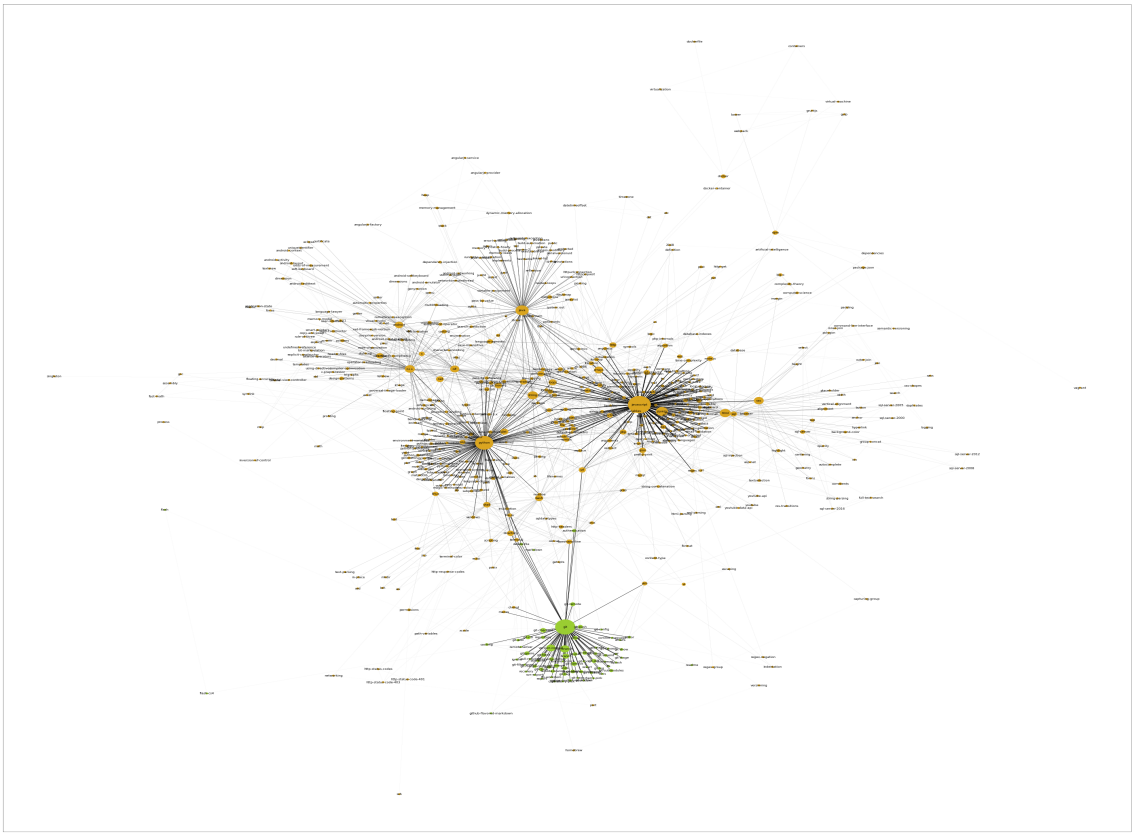

Figura 16: Grafo 1.

Se vuelve a definir un grafo G pero sólo con los elementos de los grupos 1 y 2 (en los cuales hay más elementos).

```
G = nx.Graph()
```

```
for index, row in df_tag_.iterrows():
    G.add_node(row["Tag"], group = row["group"], nodesize = row["Proportion"] )
```

```
for index, row in df_tags_.iterrows():
    G.add_edge(row["Tag_1"], row["Tag_2"], weight = row["Proportion"])
```

```
df_tag_.group.value_counts()
```

```
1    1451
2     289
Name: group, dtype: int64
```

```
nodes = G.nodes()
```

```
color_map = {1:'goldenrod', 2:'yellowgreen', 3:'#52be80', 4:'#28b463', 5:'#bcc2f2', 6:'#2c3e50'
             7:'#eebcbc', 8: '#33c3ee', 9: '#FF5733', 10: '#f39c12'}
```

```
node_color = [color_map[d['group']] for n,d in G.nodes(data=True)]
```

```
node_size = [d['nodesize']*10000 for n,d in G.nodes(data=True)]
```

```
pos = nx.drawing.spring_layout(G)
```

```
weights = [G[u][v]['weight']*5 for u,v in G.edges()]
```

```
pos_attrs = {}
for node, coords in pos.items():
    pos_attrs[node] = (coords[0] + 100, coords[1] + 100)
```

```
plt.figure(figsize=(50,50))
nx.draw_networkx(G, pos, node_color = node_color, node_size = node_size, width = weights, alpha
                 font_size = 8, with_labels = True)
```

Figura 17: Segunda definición del nuevo grafo.

Nuestra visualización está lista.

6. Interpretación de Resultados

La visualización resultante es la siguiente.

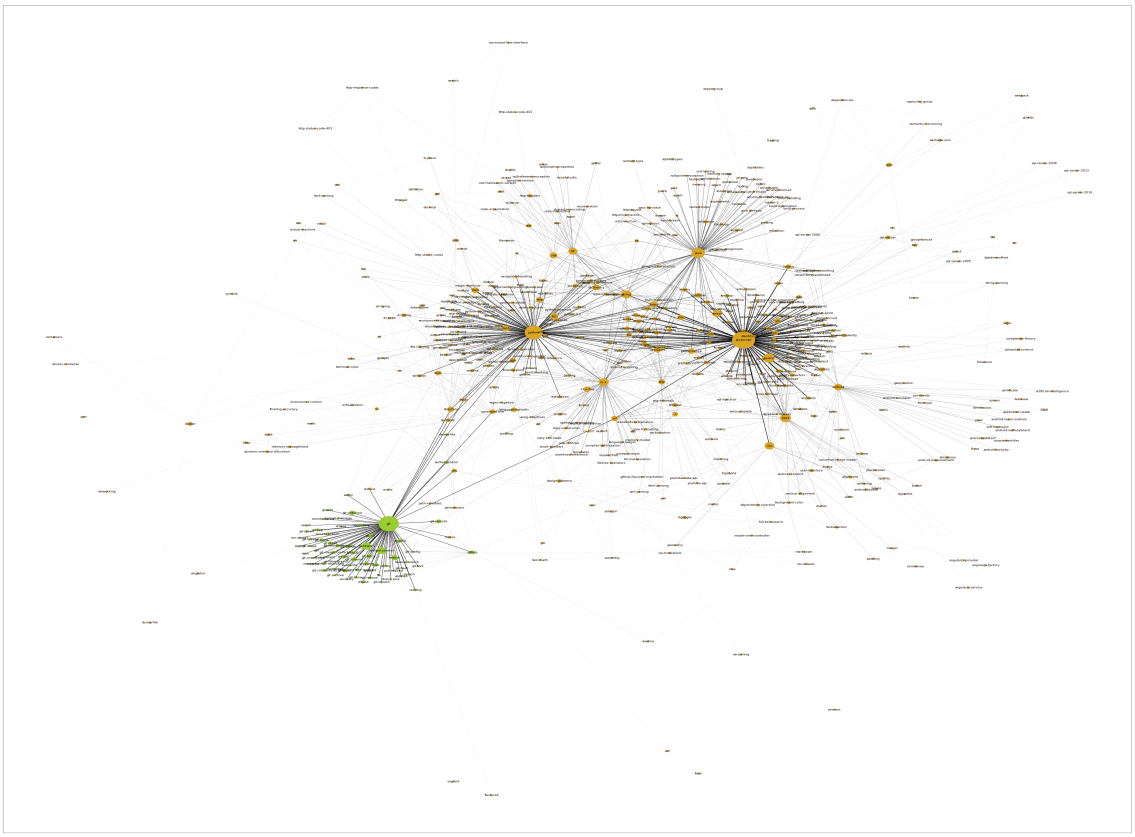

Figura 18: Tag Network.

Por la forma en que se construyo nuestra social network, podemos decir que cada nodo representa un tag y cada vértice una relación entre ellos.

El tamaño de los nodos está dado por la poderación o cantidad de veces que tenemos un tag, es decir, que entre más grande sea el nodo, el tópico aparece en más tags de preguntas. Podemos suponer que los nodos grandes son mucho más importantes que los pequeños.

Por otra parte, el grosor de los vértices va también en función del conteo de veces que se repiten las relaciones entren tags, por lo que si mi conexión tiende a ser de

color más claro, los nodos están poco relacionados, y si es más fuerte y gruesa, los tags suelen aparecer juntos en muchas preguntas.

Los colores que podemos percibir, se relacionan con la comunidad a la que pertenece el tag. En este caso podemos ver que se dividen en dos grandes grupos.

Según lo anterior, podemos ver de que manera se conectan nuestros tópicos. Por ejemplo, en el grupo verde, se tiene como nodo más importante el tag git, este tiene muchas relaciones, y como podemos apreciar, la mayoría de estas se refieren a comandos de git, por lo que podemos suponer que es una comunidad de tags dedicados a dicha plataforma. El segundo grupo es más extenso y tiene distintas vertientes, si lo analizamos, estás se inclinan casi por completo a lenguajes de programación, e incluso, podemos atrevernos a decir que los nodos centrales, son lenguajes de programación en si, como javascript y python que son dos de los principales lenguajes que se ocupan hoy día en el mercado y la inverstigación. Estas comunidades no se pueden desconectar del todo, es por eso que aparceen de un mismo color. Hay muchas características particulares con las que cuenta cada lenguaje, pero también tienen un grupo grande de intersecciones en comandos de manejo de datos, métodos, estructuras de datos, objetos, entre otros. Es por eso que a pesar de ser lenguajes independientes, cuentan con demasiadas relaciones fuertes que no les permiten separarse del todo y se deben de agrupar como lenguajes de programación en general.

7. Conclusiones

Para finalizar, es importante remarcar el cumplimiento de los objetivos planteados, que si recordamos, se refieren más a ser de utilidad para la comunidad consumidora de la plataforma Stack Overflow, como podemos notar, el conocer sus tags nos da una amplia visión de los temas que les son de interés. A partir del grafo antes mostrado, esta comunidad puede identificarse en los tópicos con los que suelen trabajar o que ya son expertos y visualmente, sin necesidad de un gran esfuerzo, identificar qué ramas los pueden ayudar a mejorar sus skills y explotar lo que ya saben o, si así lo desean, explorar aquellas que están más lejanas para saber un poco de todo. Ya dependerá del usuario la utilidad que le de, y si le interesa ser experto en una sola aréa o conocer a grandes rasgos muchas y que camino seguir para cualquiera de las dos opciones.

8. Referencias bibliográficas

(2019). Algoritmo de Girvan Newman. Febrero 2021, de Graph Everywhere Sitio web: https://www.grapheverywhere.com/algoritmo-de-girvan-newman/

Python. (2020). beautifulsoup4 4.9.3. Febrero 2021, de pypi Sitio web: https://pypi.org/project/beautifu

Alex Ronquillo. (2021). Python's Requests Library (Guide). Febrero 2021, de realpython Sitio web: https://realpython.com/python-requests/

Julia Silge. (2020). Tag Network. Febrero 2021, de Kaggle Sitio web: https://www.kaggle.com/stackoverf overflow-tag-network

Universitat Oberta de Catalunya. (2017). Uso de Social Network Analytics (SNA). Febrero 2021, de Universitat Oberta de Catalunya Sitio web: http://dataanalysis.blogs.uoc.edu/2017/08 de-social-network-analytics-sna/