Análisis de redes sociales, PEC2: Estudio de una red real

Roberto Santamaría Ayuso

Resumen

Para esta práctica he decidido analizar las comunidades de aficionados al motociclismo. Para ello capturé tweets durante la carrera de MotoGP del Gran Premio de Catalunya del día 17 de junio. Los nodos de la red son los usuarios y las relaciones puede ser menciones, retweets y seguimiento entre usuarios. Centré la búsqueda en los hashtags propuestos por la cuenta oficial del campeonato: **#MotoGP** y **#CatalanGP**.

1. Captura de datos

Capturé los tweets usando el módulo de streaming de la librería `tweepy`. Twitter no asegura que las llamadas a su API devuelven todos y cada uno de los tweets que pasen el filtro que especificamos, sino lo más *relevantes* (aunque no explican su concepto de relevancia). En cualquier caso, conseguí recuperar bastantes tweets para formar una red de tamaño interesante. Como hice la captura en tiempo real, volqué los tweets en una base de datos sin ningún preprocesamiento para poder trabajar con ellos más adelante. Usé MongoDB porque el modelo de documentos en JSON encaja exactamente con los objetos que devuelve la API. A continuación muestro el código que usé para la captura.

```
from pymongo import MongoClient
client = MongoClient('localhost', 27017)
db = client['motogp']
tweets = db['tweets']

from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream
import json

consumer_key = 'XXX'
consumer_secret = 'XXX'
access_token = 'XXX'
access_token_secret = 'XXX'

class StdOutListener(StreamListener):
    def __init__(self):
        self._i = 0

    def on_data(self, data):
        tweets.insert_one(json.loads(data))
        self._i +=1
        if self._i % 500 == 0:
            print("Got",self._i,"tweets.")
```

```
        return True
    def on_error(self, status):
        print (status)

l = StdOutListener()
auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
stream = Stream(auth, l)
stream.filter(track=['#MotoGP', '#CatalanGP'])
```

Arranqué la captura 5 minutos antes del inicio de la carrera, a las 13:55, y lo paré 15 minutos después de la finalización, a las 15:00. En total almacené unos 10.000 tweets.

Cada tweet incluye cierta información del usuario, como el nombre y la cantidad de usuarios a los que sigue (*friends)* y que le siguen (*followers*). Hice un primer procesado para quedarme con los datos interesantes con el siguiente código (omitiendo los detalles de conexión a la base de datos). El resultado fueron 4550 usuarios únicos.

```
tweets = db['tweets']
user_details = db['user_details']
user_details.drop()
for tweet in tweets.find():
    user_details.update_one(
        {'user_id': tweet['user']['id_str']},
        {
            '$set':{
                'user_id': ,
                'screen_name': tweet['user']['screen_name'],
                'followers': tweet['user']['followers_count'],
                'friends': tweet['user']['friends_count']},
            '$inc': {'num_tweets': 1},
            '$push': {'tweets': tweet['text']}
        },
        upsert=True)
```

El siguiente paso fue extraer los amigos de cada usuario. El problema aquí fue la limitación de la API de Twitter a 15 llamadas cada 15 minutos. Para poder descargar la información en un tiempo razonable me limité a los usuarios que habían publicado más de un tweet (al menos dentro del subconjunto de tweets que Twitter me hizo llegar). Por suerte la librería de tweetpy tiene facilidades para tratar las limitaciones de la API automáticamente. El resultado fueron 1634 usuarios, cada uno con la lista de usuarios a los que sigue. Tuve que parar y arrancar el proceso varias veces, ya que en total estuvo unas 28 horas en ejecución, así que añadí código para poder arracar de cero sin sobreescribir los datos ya descargados y para controlar algunos posibles errores.

```
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import API, Cursor, error
import json
import time

consumer_key = 'XXXX'
consumer_secret = 'XXXX'
access_token = 'XXXX'
```

```
access_token_secret = 'XXXX'

auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = API(auth)

from pymongo import MongoClient
from pymongo import DESCENDING
client = MongoClient('localhost', 27017)
db = client['motogp']
users_friends = db['users_friends']
tweets = db['tweets']

user_details = db['user_details']
active_users = user_details.find({'num_tweets':{'$gt':1}})
user_ids = [user['user_id'] for user in active_users]

for user in user_ids:
    print('User_id:',user)
    if users_friends.find({'user_id': user}).count() == 0:
        friends = []
        try:
            for page in Cursor(api.friends_ids,
                               user,
                               wait_on_rate_limit=True).pages():
                friends = friends + page
            print("- friends:",len(friends))
            users_friends.replace_one(
                {'user_id': user},
                {'user_id': user, 'friends': friends},
                upsert=True)
        except error.TweepError:
            print("- unauthorized")
        except Exception as e:
            print("- excp type {}, waiting".format(type(e)))
            time.sleep(120)
    else:
        print('- already in DB')
```

En este punto ya tenía toda la información necesaria para construir la red. Por un lado guardé las relaciones de menciones, retweets y seguimiento en la propia base de datos (aunque estos datos ya se pueden construir desde cero sin consultar a la API).

```
# solo incluimos relaciones de usuarios con mas de un tweet
# los tweets tienen detalles del usuario, asi que evitamos
# una segunda consulta a la API
active_users = user_details.find({'num_tweets':{'$gt':1}})
user_ids = [user['user_id'] for user in active_users]

# user_a menciona a user_b
mentions.drop()
for tweet in tweets.find({},
```

```
                        {'user.id_str':1,
                        'entities.user_mentions':1}):
    for mention in tweet['entities']['user_mentions']:
        if tweet['user']['id_str'] in user_ids and \
                mention['id_str'] in user_ids:
            mentions.insert_one({
                'user_a': tweet['user']['id_str'],
                'user_b': mention['id_str']
            })

# user_a retweetea a user_b
retweets.drop()
for tweet in tweets.find({'retweeted_status': {'$exists': True}}):
    if tweet['user']['id_str'] in user_ids and
            tweet['retweeted_status']['user']['id_str'] in user_ids:
        retweets.insert_one({
            'user_a': tweet['user']['id_str'],
            'user_b': tweet['retweeted_status']['user']['id_str']
        })

# user_a sigue a user_b
follows.drop()
for user_a in users_friends.find():
    for user_b in user_a['friends']:
        if str(user_b) in user_ids:
            follows.insert_one({
                'user_a': user_a['user_id'],
                'user_b': str(user_b)
            })
```

El siguiente paso fue construir el grafo a partir de estas relaciones. Decidí usar un grafo dirigido ya que las relaciones en Twitter no tienen por qué ser recíprocas. Además decidí dar peso a las aristas siguiendo esta convención:

- peso 1 para los retweets,
- peso 2 para los amigos,
- peso 4 para las menciones.

Con estos pesos intento captar la importancia que el usuario que inicia el tweet da al segundo usuario: hacer un retweet es muy fácil y puede ser por algo puntual; el hecho de seguir a alguien implica un interés continuado, y una mención implica acordarse de alguien expresamente. Sólo he includio relaciones entre nodos cuando ambos nodos han publicado alguno de los tweets que capturé. Esta decisón ha sido puramente práctica para evitar consultas adicionales a la API con el consiguiente tiempo de descarga. Para completar la red añadí atributos a los nodos a partir de los detalles de los usuarios.

```
import networkx as nx
G = nx.DiGraph()
connected_users = set()

for r in retweets.find():
    connected_users.add(r['user_a'])
    connected_users.add(r['user_b'])
    G.add_edge(r['user_a'], r['user_b'],
```

```
                rel='retweets', weight=1.0)

for f in follows.find():
    connected_users.add(f['user_a'])
    connected_users.add(f['user_b'])
    G.add_edge(f['user_a'], f['user_b'],
                rel='follows', weight=2.0)

for m in mentions.find():
    connected_users.add(m['user_a'])
    connected_users.add(m['user_b'])
    G.add_edge(m['user_a'], m['user_b'],
                rel='mentions', weight=3.0)

for u in user_details.find({'num_tweets':{'$gt':1}}):
    if u['user_id'] in connected_users:
        tweets = "|".join(u['tweets'])
        G.add_node(u['user_id'],
                   screen_name=u['screen_name'],
                   tweets=tweets,
                   num_tweets = u['num_tweets'],
                   followers = u['followers'],
                   friends = u['friends'])

nx.write_gexf(G, 'motogp.gexf')
```

El grafo tiene 1558 nodos y 9986 aristas.

2. Datos básicos de la red

En una primera clasificación, se puede identificar que la red es:

- valorada, ya que he asignado pesos a las aristas;
- dirigida, porque he tenido en cuenta que las relaciones en Twitter no son recíprocas;
- unicapa, ya que aunque he identificado tres tipos de relaciones diferentes, he tratado los tipos como un gradiente de intensidad y solo hay un tipo de arista;
- las relaciones de retweets y menciones forman una red bipartita, con los usuarios relacionados sólo con tweets, mientras que las de seguimiento forman una red unipartita, con relaciones entre usuarios. He construido el grafo directamente como una red unipartita, aplanando los retweets y menciones.

Para saber si es conexa hay que extraer los componentes. NetworkX indica que tiene 2 componentes, pero uno de ellos tiene un sólo nodo. Resulta que el usuario *SmoAdhikari* sólo tiene conexión consigo mismo, así que decidí eliminarlo. El resultado es una red conexa.

```
cs = [c for c in nx.connected_components(nx.Graph(G))]
print([len(c) for c in cs])
> [1557, 1]

print(cs[1])
>{'74511565'}
```

```
G.edges('74511565')
> OutEdgeDataView([('74511565', '74511565')])

G.nodes(data=True)['74511565']['screen_name']
> 'SmoAdhikari'

G.remove_nodes_from(cs[1])
```

A partir de este punto exporté para seguir el análisis en Gephi. Usé el formato GEXF porque permite aristas dirigidas y conserva todos los atributos.

```
nx.write_gexf(G, 'motogp.gexf')
```

Ya en Gephi pude calcular otros parámetros de interés:

- diámetro: 11
- distancia media entre nodos: 4.29
- densidad: 0.004

3. Análisis de los datos básicos

Una densidad de 0.004 nos indica que no nos encontraremos muchas relaciones entre usuarios: probablemente haya unos pocos nodos principales con muchas conexiones a nodos secundarios. La distancia media soporta esta decisión: si faltan muchas conexiones de todas las posibles, un camino entre dos nodos al azar tendrá que pasar a la fuerza por varios de estos nodos principales. El diámetro es más del doble que la distancia media: esto puede ser un indicativo de que la dirección de las relaciones está muy descompensada, con ciertos nodos acumulando la mayoría de las direcciones entrantes. Se puede confirmar esta situación calculando las métricas sin considerar la dirección de las aristas: el diámetro es 7 y la distancia media, 2.30.

4. Gráficas

El primer intento (1) consistió en aplicar una distribución Force Atlas, colorear los nodos según la clase de modularidad y modificar el tamaño de los nodos según su grado, sin modificar los parámetros por defecto.

Figura 1: Clase de modularidad y distribución automática

Se puede intuir que el nodo más grande será el canal oficial de MotoGP. Las clases se han quedado muy juntas, así que será necesario investigar qué relaciones hay dentro de cada una: ¿serán los seguidores de un equipo? ¿De un piloto en particular? ¿Será gente siguiendo la carrera comentada por un medio?

La segunda visualización (2) muestra las comunidades separadas y el nombre de los usuarios con mayor grado. Vemos que las comunidades pequeñas son ajenas al resto, y da la impresión de que las comunidades grandes pueden estar relacionadas con equipos o patrocinadores.

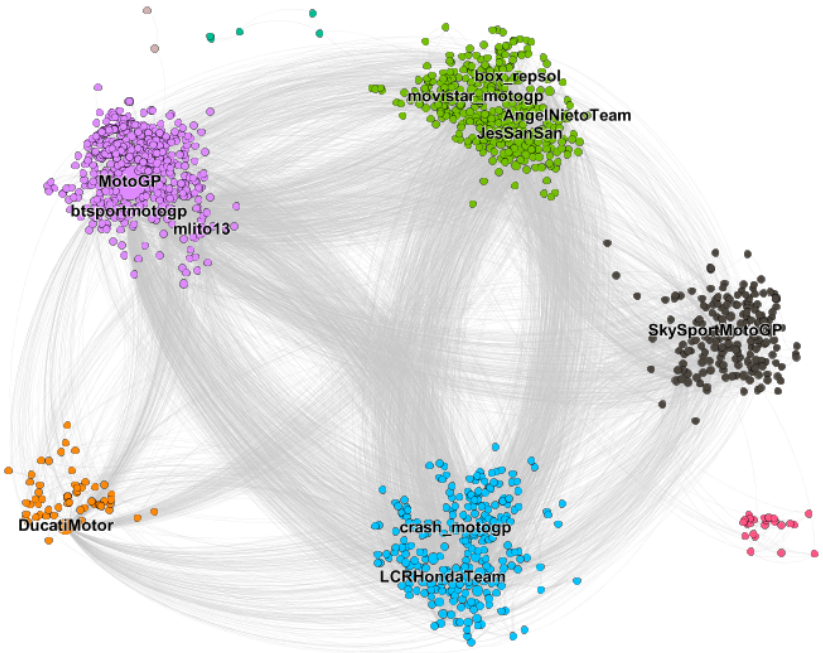

Figura 2: Separación de comunidades y nodos de mayor grado

Esta última visualización (3) muestra la misma información que la anterior con un aspecto más llamativo. Seguiré analizando las visualizaciones en el último apartado, ya que los siguientes apartados extraen datos que se puede usar para hacer un gráfico más informativo.

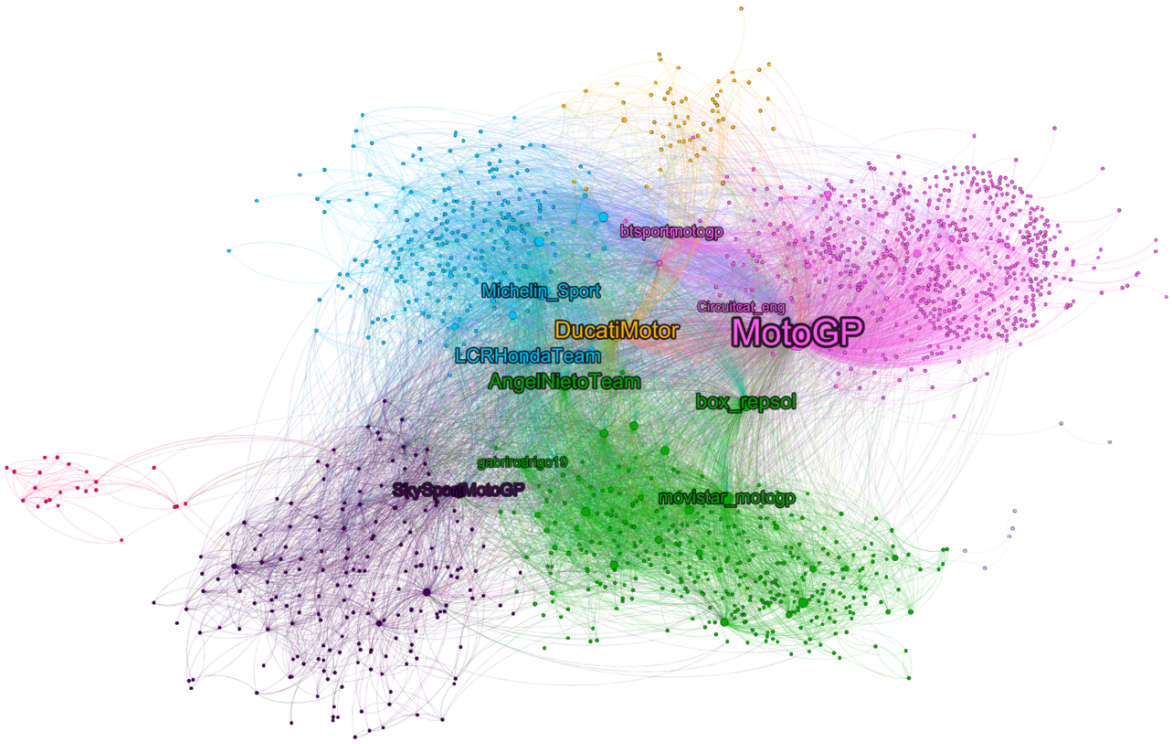

Figura 3: Misma información, cambiando la presentación

5. Prestigio

Es muy habitual que los medios evalúen el prestigio en las plataformas sociales mirando únicamente los números de *followers* y *likes*. Al aplicar este criterio a esta red (1), vemos que los principales nodos son, precisamente, medios de comunicación (el menos conocido puede ser TRANS7, un canal de deportes indonesio). También aparece el canal oficial del campeonato: será una tónica continua, ya que decidí capturar unos hashtags promovidos por este canal.

screen name	followers
el_pais	6681677
TRANS7	5455465
marca	4866853
MotoGP	2484790
mundodeportivo	2433442
SkySport	2344426
SPNSportsIndia	2066585
Eurosport_FR	1454791
LaVanguardia	982957
elpais_deportes	665427

Cuadro 1: Top 10 según número de seguidores

Sin embargo, al atender al PageRank de los nodos (2), el resultado cambia: detrás del canal oficial aparecen cuentas de equipos como DucatiMotor, box_repsol, LCRHondaTeam o AngelNietoTeam. También aparecen medios, pero en este caso especializados: movistar_motogp (emite en exclusiva en España) y SkySportMotoGP.

screen name	pagerank
MotoGP	0.1
DucatiMotor	0.04
AngelNietoTeam	0.03
box_repsol	0.03
LCRHondaTeam	0.03
Michelin_Sport	0.03
movistar_motogp	0.03
SkySportMotoGP	0.02
btsportmotogp	0.02
Circuitcat_eng	0.02

Cuadro 2: Top 10 según PageRank

La valoración la cuenta oficial es mucho mayor que el resto. Las visualizaciones también mostraban que tenía un grado mucho más alto de lo normal. ¿Es realmente esta cuenta la más importante de la red? Claro, pero debido a la propia construcción de la red. Si el filtro de captura hubiera sido un usuario, por ejemplo **@26_DaniPedrosa**, habrían aparecido otros pilotos pero es de esperar que el principal fuera el que ha servido para generar la red. Por tanto, antes de seguir análisis voy a borrer el usuario **@MotoGP**. Además, hay muchos nodos conectados únicamente con este nodo y quedarán aislados, así que los borro también. La red pasa de 1557 a 1333 nodos, y de 9985 a 8660 aristas. La figura 4 muestra la comunidad en torno a @MotoGP antes de después del borrado.

Figura 4: Comunidad en torno a @MotoGP antes de borrar el nodo (izquierda) y después.

El nuevo PageRank arroja un ranking similar (3). De nuevo aparecen principalmente cuentas de equipos y de medios especializados.

screen_name	pagerank
DucatiMotor	0.04
box_repsol	0.03
AngelNietoTeam	0.03
LCRHondaTeam	0.03
Michelin_Sport	0.03
movistar_motogp	0.02
SkySportMotoGP	0.02
mundodeportivo	0.02
btsportmotogp	0.02
MCNSport	0.02

Cuadro 3: Top 10 según PageRank (sin @MotoGP)

También voy a evaluar la importancia en base a la centralidad de intermediación (4): es aquí donde desaparecen los equipos y los medios principales. Los usuarios mlito13, JesSanSan, giroveloce, bgmotogp y Ride_Scotland aparecen como los más relevantes, pero no aparecían en lo algo del ranking de PageRank. El primero se identifica como *social media manager* de Ibiza y se dedicó a retwittear a los principales pilotos. JesSanSan es un bloguero sevillano centrado en el motociclismo que relató la carrera por Twitter. Giroveloce y Ride_Scotland están a caballo entre portal y revista de motociclismo, sin llegar a convertirse en medios de gran presencia; el primero en Italia y el segundo en Escocia. Bgmotogp retransmitió la carrera al igual que mlito13 y JesSanSan.

screen name	betweeness
mlito13	389928
JesSanSan	216715
giroveloce	94212
bgmotogp	84543
Ride_Scotland	51180
Swinxy	47640
Dovi04Indonesia	42376
crash_motogp	42053
Evil_Herself_	40461
Off_Bikes	36533

Cuadro 4: Top 10 según centralidad de intermediación

¿Qué se puede decir a partir de estos números? Por un lado, que fijarse únicamente en el número de followers para comprobar el impacto de un usuario en un determinado evento no es suficiente. Las cuentas con una alta centralidad de intermediación han atraído a muchos usuarios a la red - si no hubieran intervenido, puede que muchos usuarios no hubieran entrado a comentar. Incluso las cuentas oficiales de equipos y patrocinadores tienen menos impacto que algunos bloggers y portales de aficionados. Si nos ponemos en el papel de estos últimos, podríamos usar esta información para promocionarnos ante posibles patrocinadores - aunque convencerlos de que el betweennes importa más que los followers será una tarea difícil.

También es reseñable que los dos primeros usuarios de la tabla 4 sean españoles. Quizás Twitter haya considerado *relevantes* aquellos tweets generados en España, ya que la captura se inició también desde España. Sería interesante hacer dos capturas simultáneas de un evento, una desde España y otra desde Italia, por poner dos ejemplos de países con gran afición a este deporte.

A modo de resumen he intentado plasmar este contraste en la siguiente figura (5). Durante este evento las cuentas con más usuarios no eran las más relevantes. Si mlito13 o JesSanSan no hubieran intervenido durante la carrera, los mensajes no habrían llegado a tantos usuarios como la retransmisión de El País.

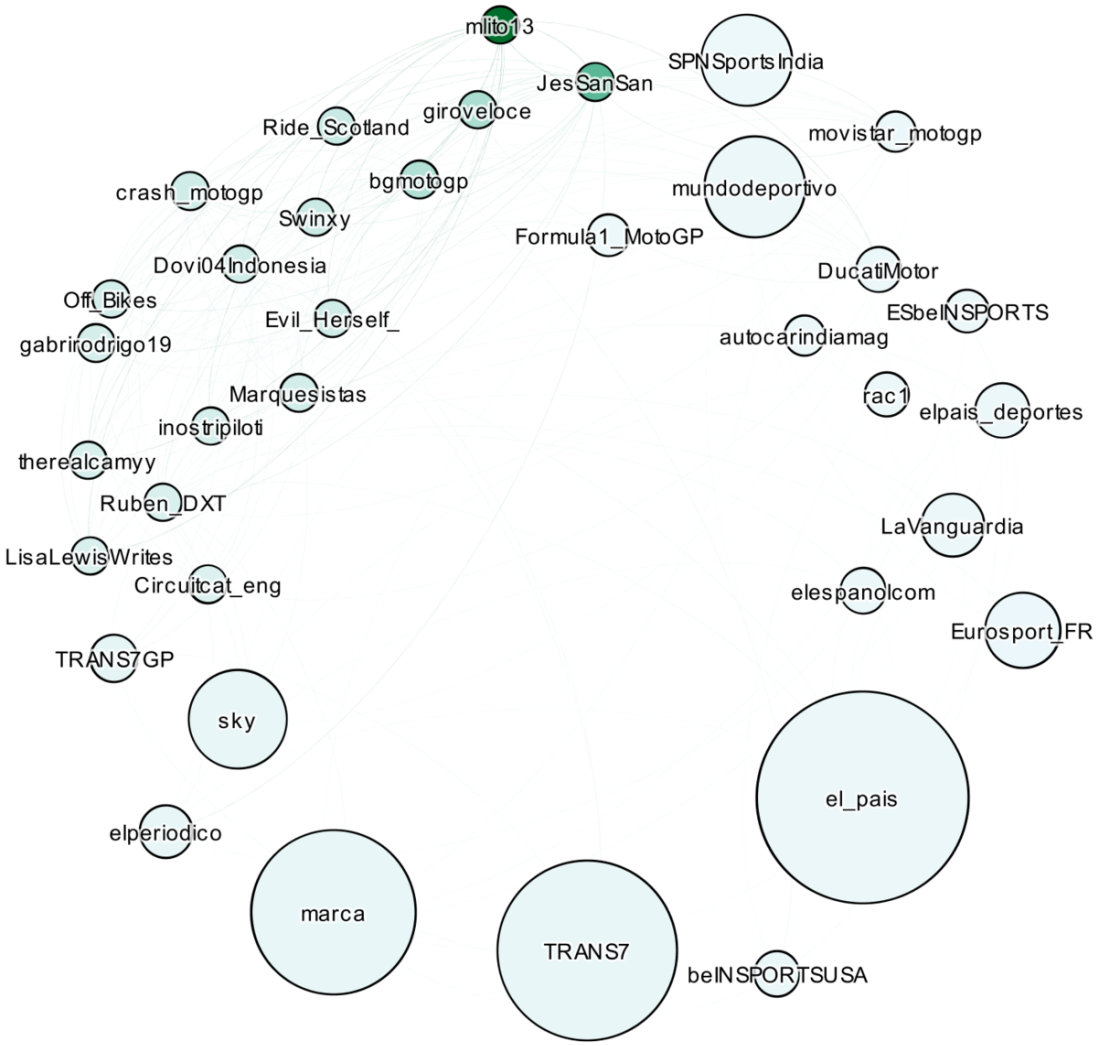

Figura 5: Mayor área implica más followers. Más intensidad de color implica más centralidad de intermediación.

6. Comunidades

Parte de este análisis empezó con la clase de modularidad que usé para colorear los nodos en las primeras visualizaciones. El número de comunidades no tiene una relación directa con la realidad, así que es necesario revisar manualmente por qué cada nodo pertenece a la clase que dice Gephi que pertenece.

Las etiquetas de la figura (6) identifican lo que parecían ser los usuarios más relevantes de cada comunidad.

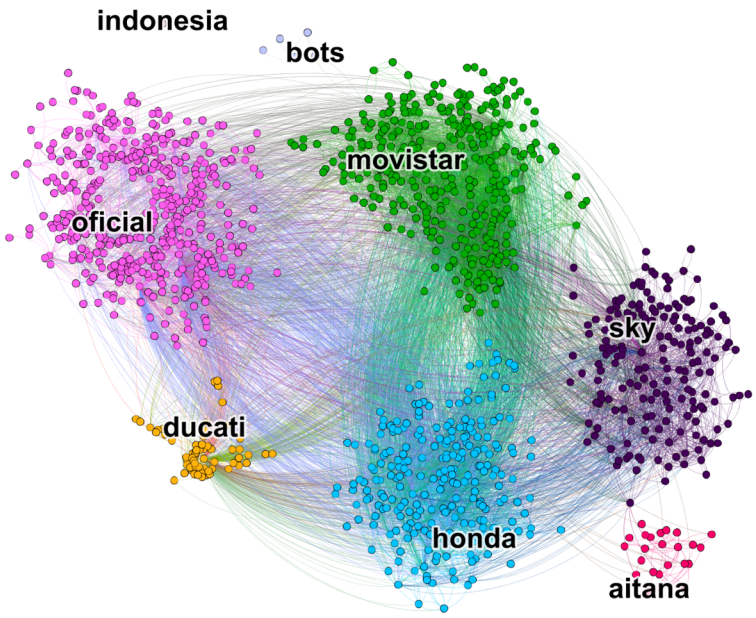

Figura 6: Comunidades en base a la clase de modularidad

Las comunidades **bots** e **indonesia** tienen 5 y 2 nodos respectivamente. La primera está formada por cuentas que se dedican a twittear los *trending topics* del momento y aparecen porque el hashtag *#MotoGP* tuvo mucha relevancia durante la carrera, no porque estén muy relacionados con el resto de usuarios. En indonesia hay dos nodos que, efectivamente, han twitteado en indonesio sobre la carrera.

La subred **aitana** es un caso curioso. Uno de los usuarios publicó una foto de la cantante Aitana Ocaña en la carrera (https://twitter.com/SrtaLexus/status/1008317045434437632). Esto provocó una pequeña oleada de retweets y comentarios entre un grupo de usuarios. Estos tweets contenían las etiquetas de la búsqueda, pero los usuarios no tenían apenas conexiones salvo entre ellos y con el primer usuario que mencionó a la cantante.

Las otras cuatro comunidades tienen muchas relaciones entre ellas, y de hecho la primera visualización (1) las muestra prácticamente como una única red. Pero esta distribución de comunidades se calculó *antes* de borrar **@MotoGP** y todos los nodos que estaban conectados únicamente a él. ¿Qué ocurre si recalculamos la clase de modularidad? La red que habló sobre Aitana Ocaña vuelve a aparecer con prácticamente los mismos nodos. Aparecen otras 4 clases de 2 nodos cada uno que han quedado descolgados al desparecer la cuenta oficial. Las clases **sky** y **ducati** repiten, pero las demás se han disgregado. ¿De qué habla cada comunidad?

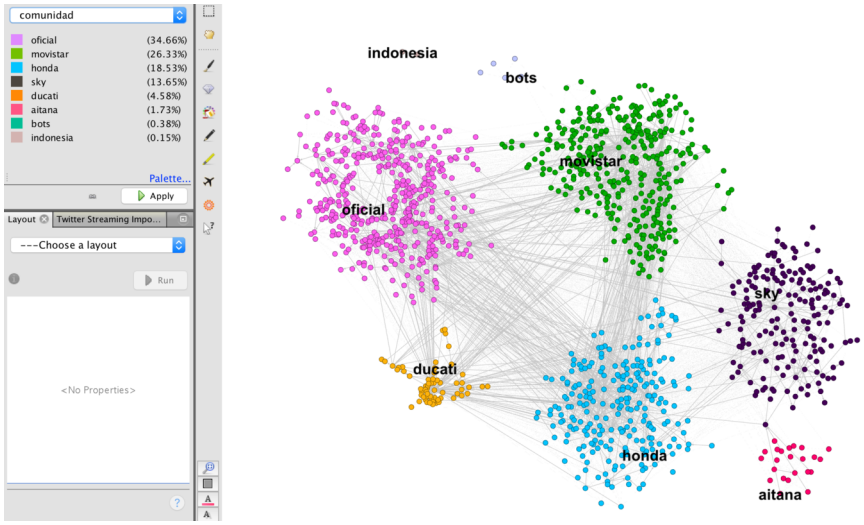

Figura 7: Clases antes de borrar MotoGP

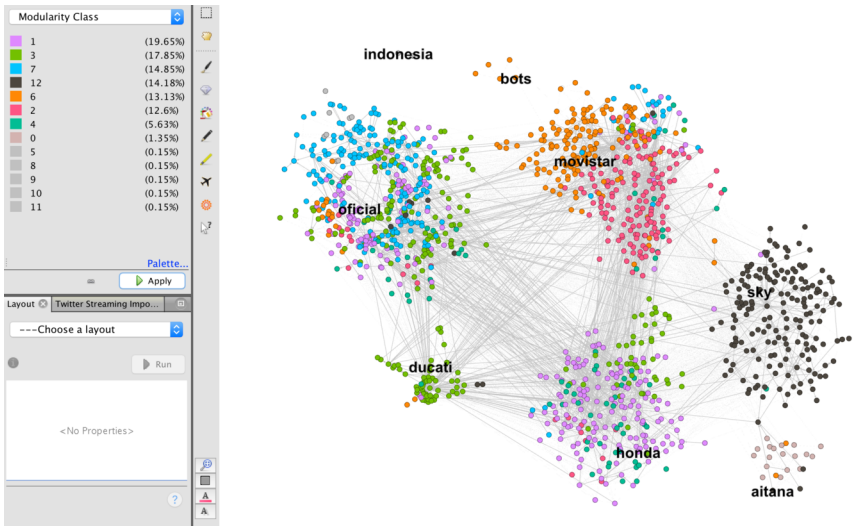

Figura 8: Clases después de borrar MotoGP

La subred que llamé *sky* se ha mantenido sin muchos cambios y está liderada por las cuentas de **Sky** y **giroveloce**, el usuario que apareció con alto índice de intermediación. Sky es un medio británico, pero la cuenta @SkySport es la oficial del canal *en italiano*. La mayoría de los tweets eran en italiano y hablaban de la carrera en general. La comunidad de *movistar* se ha dividido en dos: una cuenta con mucha presencia de **medios** tweeteando en español y catalán sobre el desarrollo de la carrera, y otra liderada por el bloguero **JesSanSan**, en español e inglés.

La subred de *ducati* habla, precisamente, sobre **ducati** y **Jorge Lorenzo**, sobre todo en inglés. Lorenzo ganó la carrera, así que no es de extrañar que aparezca una comunidad hablando precisamente del ganador, esté o no compuesta fundamentalmente por sus fans.

Las comunidades que llamé *honda* y *oficial* se han disgregado en tres. Todas comentan principalmente en inglés: una de ellas liderada por **motorsport** (una revista especializada en motor), otra por el ya conoci-

do **Ride_Scotland** con muchos usuarios británicos, y la última con comentarios centrados en Valentino Rossi y Marc Márquez. Se clasificaron segundo y tercero, y mueven aficiones similares, a pesar de los rifi-rafes de los últimos años - no así los fans de Jorge Lorenzo, que suelen ser siempre muy críticos con cualquier piloto que no sea su ídolo.

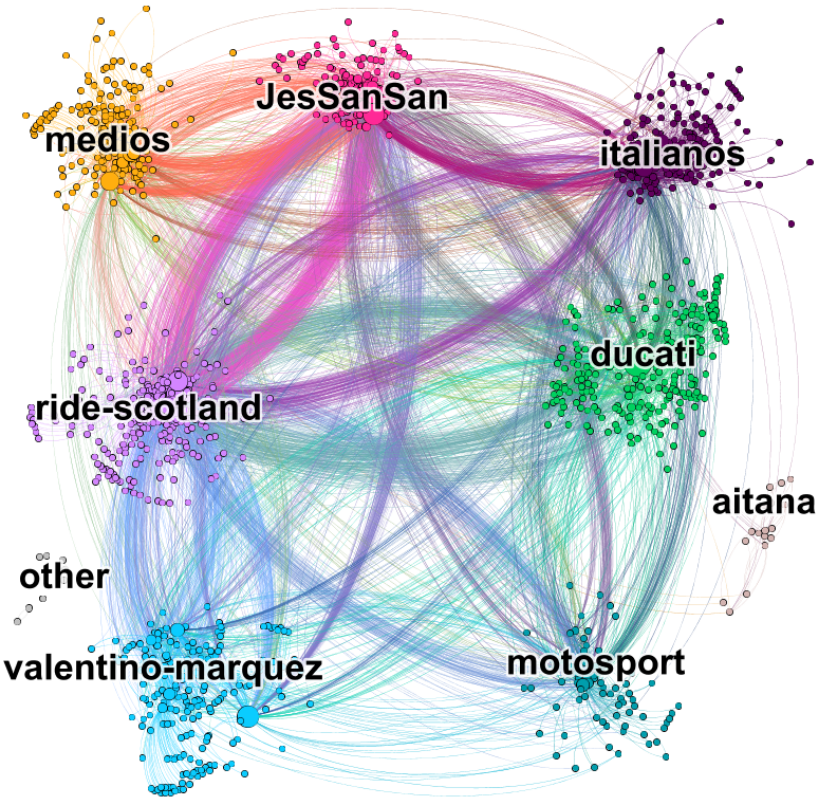

Figura 9: Análisis de comunidades

No todas las comunidades tienen la misma morfología. La tabla 5 presenta las densidades y el número de nodos de las principales comunidades en la red completa y después de seleccionar los k-cores con k=8 y k=12 (en %, ya que tienen diferentes tamaños). Las comunidades de *ducati* y *vale-marquez* apenas tienen conexiones internas: probablemente hayan aparecido durante la carrera y los usuarios no mantendrán conexiones duraderas. Sin embargo, la comunidad formada por *JesSanSan* y, en menor medida, las de *ride-scotland* y *motorsport,* son comunidades con más conexiones internas. Parecen representar aficionados conectados no sólo por un tema sino por un punto de encuentro común (la cuenta de las revistas o del blogger) que forman lazos un poco más permanentes.

comunidad	# nodos	% nodos en K8	% nodos en K12	densidad
ride-scotland	262	40 %	27 %	0.02
ducati	238	15 %	5 %	0.01
vale-marquez	198	13 %	5 %	0.01
italianos	189	40 %	23 %	0.03
medios	175	30 %	12 %	0.02
jessansan	168	68 %	54 %	0.04
motorsport	75	33 %	27 %	0.03

Cuadro 5: Detalles de las comunidades principales

Como he mencionado antes, las comunidades que el algoritmo de clase de modularidad ha identificado son un poco artificiales. He recalculado las clases modificando los parámetros del algoritmo y tendía a aparecer una única clase con el 95% de los nodos y varias clases más de 2-3 nodos. Hay muchas conexiones entre cada comunidad, tal como se aprecia en las visualizaciones. Consecuentemente, habría que ser cauto a la hora de usar las conclusiones de este análisis.