Social Media Analysis

Antonio Cavuoto[1], Riccardo Sieve[1]
[1] Reti e Sistemi Informatici – Università degli Studi di Torino
antonio.cavuoto@edu.unito.it, riccardo.sieve@edu.unito.it

Network Science project for the course of Complex Network Analysis and Visualization developed to study and analyze the social media during the Covid Pandemic and how it was perceived.

Throughout the Covid pandemic[1] in 2020, lots of news and comments have been published in both sides, the one that supports the mainstream media, and the one that does not support mainstream media; we studied a self-made social network using a web scraper.

1 Introduction

We wanted to build our own dataset made by posts and comments. Posts are taken from a list of the following public facebook pages:

- ilgrandeinganno1
- BuffonateDiStato
- Il-Sofista-764004616975397
- NoiconTrump
- stefanomontanari.net
- ScoglioStefano
- ilFattoQuotidiano
- Repubblica
- leretico2020
- quotidianolaverita
- ilGiornale
- ilsole24ore
- lastampa.it

To guarantee plurality, we used pages of mainstream media and others more focused on conspiracy theories as well.

Our first problem was that Facebook doesn't allow the public Graph API[1] to get comments and posts from a single page, unless we get a special authorization token from the admin of that page.

Even though we could have get some pre-made dataset from the internet, we wanted some "actual" data, in order to get a baseline to compare with the academic work we studied during the course.

Therefore, the only way we came up to, was to build a web scraper for Facebook, taking in consideration that nowadays all major social media are fully rendered in Javascript.

We needed to get rid of all the Javascript. The breakthrough was the following: it exists a specific frontend[2] made with PHP, in order to guarantee backward compatibility with old devices (like the Nokia C3 which we used as initial user agent).

So at that point, we pulled out of the hat our old knowledge of python web scraping.

2 Getting the Data

To emulate human interaction in the login form, we had to toggle sleeps — so that the system could have been tricked that it was someone making different attempts — as well as providing the login form with the correct `POST` requirements:

- lsd
- jazoest
- m_ts
- li
- try_number
- unrecognized_tries

[1] https://developers.facebook.com/docs/graph-api/

[2] https://mbasic.facebook.com/

- login

This parameters are normally hidden in the code. We managed to come up with a solution only after some trial and error exploring the struture of our requests.

2.1 Setting up the Scraper

In order to retrieve the data we needed to deal with three different tasks

1. Login: getting inside the system to retrieve the data
2. Scraper: getting the data for the various comments
3. Parallelism: parallelize the code to make it faster and more efficient

All trying not to get banned, which happened later when we added parallelism.

Login to login into the machine we had to refer — as previously mentioned — to the php frontend and execute a basic authentication. Since python would raise error due to the presence of a certificate for the TLS support we had to suppress the warning with `urllib`.

```python
def init_process(cred):
  global s
  s = requests.session()

  urllib3
    .disable_warnings(
      InsecureRequestWarning)
  base_url = 'https://mbasic.facebook.com'
  login_form = "/login/device-based/regular/login/" \
    "?refsrc=https%3A%2F%2F" \
    "mbasic.facebook.com" \
    "%2F%3Fref%3Ddbl&lwv=100&ref=dbl "

  r1 = s.get(
    "https://mbasic.facebook.com/login", verify=False)
  page1 = BeautifulSoup(r1.text, "lxml")
  lsd = page1.find('input', {'name': 'lsd'})['value']
  jazoest = page1.find('input', {'name': 'jazoest'})['value']
  mts = page1.find('input', {'name': 'm_ts'})['value']
  li = page1.find('input', {'name': 'li'})['value']
  try_number = page1.find('input', {'name': 'try_number'})['value']
  unrecognized_tries = page1.find('input', {'name': 'unrecognized_tries'})['value']
  data = {'lsd': lsd, 'jazoest': jazoest, 'm_ts': mts, 'li': li, 'try_number': try_number,
      'unrecognized_tries': unrecognized_tries, 'email': cred["email"], 'pass': cred["pass"],
      'login': 'Accedi'}
  r1 = s.post(base_url + login_form, data=data, verify=False)
  c = BeautifulSoup(r1.text, "lxml")
  form = c.find('a')
  action = form.get('href')
  r1 = s.get(base_url + action, data=data, verify=False)

  try:
    r1.raise_for_status()
  except requests.exceptions.HTTPError:
    print('oops bad status code {} on request!'.format(r1.status_code))
  else:
    pass
```

The code is pretty much straightforward as it simply finds the element in the page and filles them with the proper value in order to get access. Probably some of the parameters are used in order to manage CORS[3].

Scraper after the login inside Facebook we had to retrieve the proper data that we had to put together for our network in order to make a study on it. The code for this part is a bit long, as it involves many parts

```python
def f(task):
  final_result1 = list()
  global s
  session = s
  base_url = 'https://mbasic.facebook.com'
  regex = re.compile('^[0-9]{15,17}$')
  twochars_regex1 = re.compile('^[a-z]{2}$')
```

[3]https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

```
onechar_regex = re.compile('^[a-z]$')
regex_post_replies =
↪   re.compile('^see_next_[0-9]{15,17}')
regex_comment_replies =
↪   re.compile('^comment_replies_more_')
regex_string_until_question_mark = re
  .compile('([^?]+)')
second_regex_for_profile = re
  .compile('^(/profile.php\?id=[0-9]*)')
urllib3
  .disable_warnings(
    InsecureRequestWarning)
index = 0
while index >= 0:
  r = session.get(base_url +
  ↪   task["post_url"] + "&p=" +
  ↪   str(index), verify=False)
  post = BeautifulSoup(r.text, "lxml")

  for EachComment in post.find_all("div",
  ↪  {"id": regex, "class":
  ↪   twochars_regex1}):
    h = hashlib.new('ripemd160')

    child = EachComment
      .findChild(recursive=False)
    children = child
      .findChildren(recursive=False)

    nickname_a =
    ↪   children[0].find('a').text
    nickname = re.search(
      regex_string_until_question_mark,
      children[0].find('a')
      .get('href'))
      .group(0)

    if "profile.php" in nickname:
      nickname = re.search(
        second_regex_for_profile,
        ↪   children[0]
        .find('a')
        .get('href')).group(0)
    post_text = children[1].text
    h.update(nickname_a.encode("utf-8") +
    ↪   nickname.encode("utf-8"))
    c = Comment(h.hexdigest(), nickname_a,
    ↪   nickname, post_text,
    ↪   task["post_id"])
    final_result1.append(c)

    comment_replies = child.find("div",
    ↪  {"id": regex_comment_replies})
    if comment_replies is not None:
      comment_reply_link = comment_replies
        .find("a").get('href')

      r = session.get(base_url +
      ↪   comment_reply_link,
      ↪   verify=False)

      result = BeautifulSoup(r.text,
      ↪   "lxml")
      while True:

        previous_replies1 =
        ↪   result.find("span",
        ↪   text="Visualizza le risposte
        ↪   precedenti")

        if previous_replies1 is not None:
          previous_replies2 =
          ↪   previous_replies1
            .find_parent('a')
            .get('href')
          r = session.get(base_url +
          ↪   previous_replies2,
          ↪   verify=False)
          result = BeautifulSoup(r.text,
          ↪   "lxml")
        else:
          break

      other_replies = result.find("span",
      ↪   text="Visualizza le risposte
      ↪   successive")
      while other_replies is not None:
        all_page_replies =
        ↪   result.find_all("div", {"id":
        ↪   regex, "class":
        ↪   onechar_regex})
        if len(all_page_replies) == 0:
          break
        for reply in all_page_replies:
          try:
            h = hashlib.new('ripemd160')
            children = reply.findChild(
                recursive=False)
              .findChildren(
                recursive=False)
            nickname_a =
            ↪   children[0].find('a').text
            nickname = re.search(

              ↪   regex_string_until_question_mark,
              children[0].find('a')
                .get('href'))
                .group(0)
            if "profile.php" in nickname:
              nickname = re.search(
                second_regex_for_profile,
                children[0].find('a')
                  .get('href'))
                  .group(0)
```

```
       post_text = children[1].text
       h.update(nickname_a
        .encode("utf-8") + nickname
         .encode("utf-8"))

       c = Comment(h.hexdigest(),
       ↪  nickname_a, nickname,
       ↪  post_text,
       ↪  task["post_id"])
       final_result1.append(c)
       other_replies2 = other_replies
        .find_parent('a')
        .get('href')
       r = session.get(base_url +
       ↪  other_replies2,
       ↪  verify=False)
       result = BeautifulSoup(r.text,
       ↪  "lxml")
       other_replies =
       ↪  result.find("span",
       ↪  text="Visualizza le
       ↪  risposte successive")

      except AttributeError:
       pass

  other_comments = post.find_all("div",
  ↪  {"id": regex_post_replies})
  if len(other_comments) == 0:
   index = -1
  else:
   index += 10
 return final_result1
```

The first thing that is done is the creation of regular expressions to perform “ad hoc” queries on the various page of the popular social network. We managed to extract the html tree with the Python package `BeautifulSoup`[4] combined with a `lxml` parser. The tricky part is what follows: we first get the comment (to which we have to give a proper id) and build the adjacency list of siblings comments and child replies — for each comment we need to find all the replies to the various comment, and then move onto the other comments of the same post; the approach is quite similar to a depth-first search of a graph.

Parallelism to achieve good results in a reasonable amount of time, we had to resolve to parallelism, using a pool of processes in order to bypass Python GIL restrictions. So once we create the task list, the following steps are quite straightforward. Our test system is a Ryzen 5 3600 @4.00 GHz and 16GB of RAM. The data retrieval process is approximately 20 minutes long

```
if __name__ == '__main__':
 final_result = list()
 credentials =
 ↪  json_to_obj("credentials.json")
 file = open("all_comments2.json", "w",
 ↪  encoding="utf-8")
 tmp = json_to_obj("all_posts2.json")
 posts = list()
 for i in range(len(tmp)):
  x = dict(tmp[i])
  if "watch" in x["post_url"]:
   continue
  else:
   x["post_url"] = x["post_url"].replace(
    "https://facebook.com", "")
   posts.append(x)
 with multiprocessing.Pool(
  initializer=init_process,
  ↪  initargs=(credentials,)) as pool:
  # x = pool.map(f, posts)
  r = list(tqdm.tqdm(pool.imap(f, posts),
  ↪  total=len(posts)))
  for elem1 in r:
   for elem in elem1:
    final_result.append(elem)
  s = json.dumps(final_result,
  ↪  default=dumper)
  file.write(s)
  file.close()
```

3 Studies of the Data

Having collected the data, we had to build our network based on the separate `json` file that we previously got; to do so, we decided to use the library `NetworkX`[5].

We converted every post and every user into a node; moreover we hashed all the names and the respective ids in order to preserve privacy — to do so, we decided to use the non-NSA `RIPEMD160`[6] algorithm.

We mapped every comment of a user to a specific post as an edge between those two nodes. As

[4]https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[5]https://networkx.org/

[6]https://en.wikipedia.org/wiki/RIPEMD

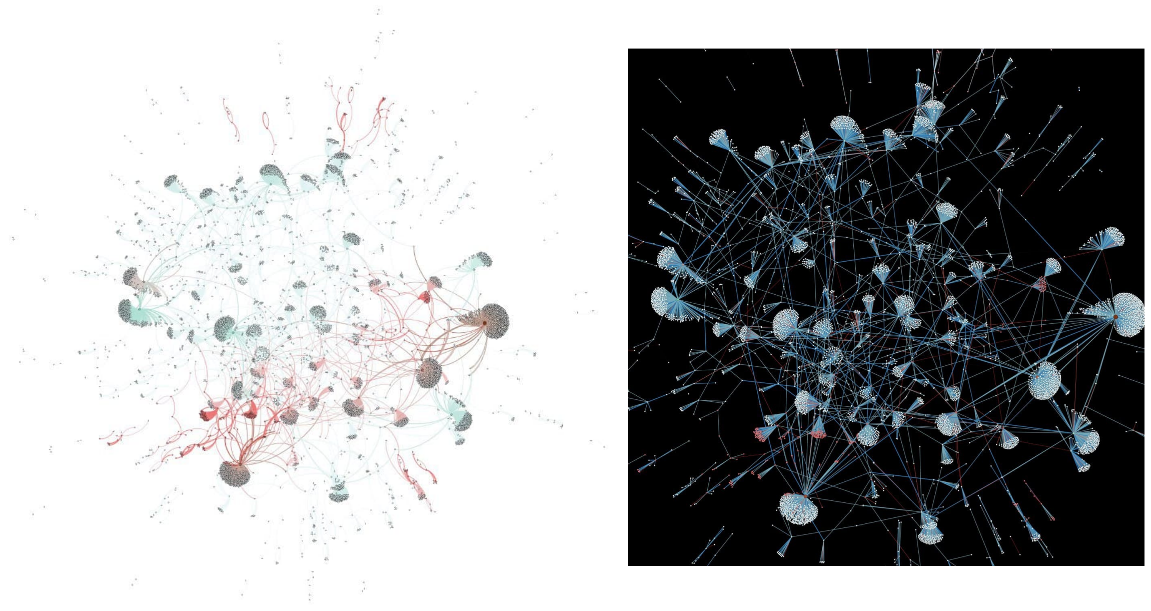


Figure 1: *Complete Network obtained with Gephi*

a result, we exported the adjacency list to use as input for the baseline R script that we saw during the course.

But that was not enough. We wanted more originality. Thus, we decided to do a sentiment analysis on our network in order to be able to assing to every edge, a specific weight representing the polarity of that comment respect to the post (positive, negative).

To implement the sentiment analysis perspective, we had many options, among which:

- Training an AI Natural Language Neural Network from scratch
- Using a frequency based word tag using the text we got from our comments
- Using some kind of cutting edge tools such as Google's BERT[7]

To speed-up the process, we decided to resort to a python library called `Polyglot`[8] to automate NLP[9] tasks like the one we are facing.

We took all our comments and make it compute the average polarity of the whole, to obtain a "sentiment range" score — between -10 and 10.

Given the polarity parameter as weight for our edges between users and post, we build a "sentiment snapshot" network from the period before the Lockdown[10] until the end of October 2020.

4 Results

We obtained our complete network using the `Force Atlas 2` layout algorithm with Gephi[11]; for better usability and readability of the complete topology, we exported both a light and dark version of it.

Some of the results that we were able to get are the following:

Structural properties

Nodes: 7221
Edges: 7425
Components: 3
Is Connected: FALSE
Average Degree: 2.06
Density: 0.000284834053953899

[7] https://en.wikipedia.org/wiki/BERT_(language_model)
[8] https://polyglot.readthedocs.io/en/latest/
[9] https://en.wikipedia.org/wiki/Natural_language_processing
[10] https://en.wikipedia.org/wiki/Lockdown
[11] https://gephi.org/

Assortativity coefficient: -0.303365009731658

It is plausible for our network not to be connected, as long as there are posts without any comments. Moreover, given an average degree of 2.06, we might now of the possibility of users that have commented multiple posts from our dataset — again this is not hard to be plausible. The median degree value will be much more indicative of what's going on here. Another option is that we could find some kind of "governative task forces" with the job of influencing social media with a particular narrative, combining the degree, the sentiment weights and the betweenness. We can further show that the value found for the density is comparable to the Facebook graph shown during the course. The reference here is Table 1.1 of [ns1][12].

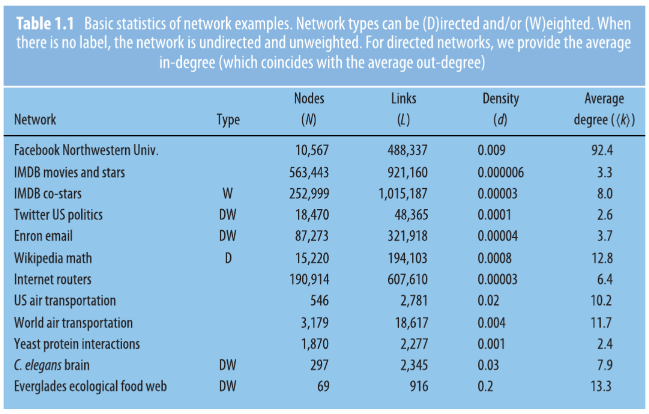


Table 1.1 Basic statistics of network examples. Network types can be (D)irected and/or (W)eighted. When there is no label, the network is undirected and unweighted. For directed networks, we provide the average in-degree (which coincides with the average out-degree)

Network	Type	Nodes (N)	Links (L)	Density (d)	Average degree ($\langle k \rangle$)
Facebook Northwestern Univ.		10,567	488,337	0.009	92.4
IMDB movies and stars		563,443	921,160	0.000006	3.3
IMDB co-stars	W	252,999	1,015,187	0.00003	8.0
Twitter US politics	DW	18,470	48,365	0.0001	2.6
Enron email	DW	87,273	321,918	0.00004	3.7
Wikipedia math	D	15,220	194,103	0.0008	12.8
Internet routers		190,914	607,610	0.00003	6.4
US air transportation		546	2,781	0.02	10.2
World air transportation		3,179	18,617	0.004	11.7
Yeast protein interactions		1,870	2,277	0.001	2.4
C. elegans brain	DW	297	2,345	0.03	7.9
Everglades ecological food web	DW	69	916	0.2	13.3

Figure 2: *Table 1.1*

Our assortativity coefficient is coherent with the fact that our network is disassortative — high degree nodes tend to attach to lower degree nodes — however, we have reasons to expected that, provided the friendship links, we would get uniformed result with a typical social assortative network.

Small World measures

Local transitivity: 0
Global transitivity: 0 (as well)
Average distance: Inf.
Average Path Length: 6.93

A value of 0 for our local transitivity is quite interesting; we know by construction that the majority of the connections are between a single user and a single post — degree 1 — and, for those cases, we cannot compute the local transitivity.

Having an infinity value for the average distance, we can say that the graph is not connected, which is what we expected.

An average path length of 6.93 allows us to confirm the general rule of "six degree of separation".

It is to be noted that, to achieve this effect, a network of size n has an average distance of $O(\log n)$; in our case we had $\log(7221) \sim 8.88$, and our Average path length is 6.93 which is slightly less.

Centralities We can now observe some of the graphs that compare various aspect of degree and betweenness centrality. First, we need to make one observation: we can expect that the nodes with more betweenness will be the posts with many comments (high degree). Furthermore, if we were able to collect also the user friendship connections, we will be able to be much more precise (and the previous transitivity measures will probably be different from zero).

We are also confident that including friendships in our network, we will get a connect graph. As per the **closeness centrality** we get from R the following warning message "closeness centrality is not well-defined for disconnected graphs".

As a general comment, it's quite useless looking for communities since, by construction, our network is only made by comment-to-post relationships.

Rich Club Observation Regarding the rich club phenomenon, we obtained the following graph

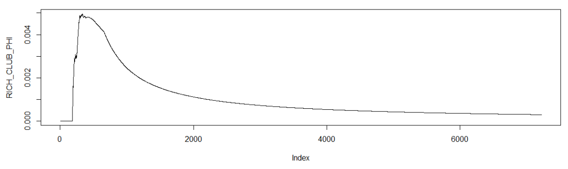


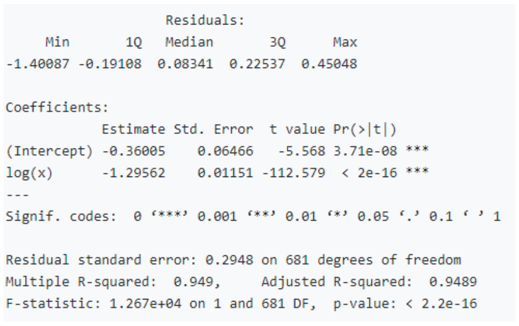
Figure 3: *Rich club graph*

The graph shows that we are not int front of a Rich Club. In fact, to confirm so, we need Phi to be a decreasing monotone function; if we exclude

[12]Filippo Menczer, Santo Fortunato , and Clayton A. Davis, A First Course in Network Science, Cambridge University Press

the first group of vertices, we can confirm the Rich Club. This is also reasonable due to how is formed our network.

Is this a Scale-Free network? To answer this question, we made a linear regression model to fit the formula $\log(y) \sim \log(x)$ where y is our degree distribution and x is an index variable — from 1 to MAX_DEGREE — that got us the following results



```
                     Residuals:
     Min       1Q   Median       3Q      Max
-1.40087 -0.19108  0.08341  0.22537  0.45048

Coefficients:
            Estimate Std. Error  t value Pr(>|t|)
(Intercept) -0.36005    0.06466   -5.568 3.71e-08 ***
log(x)      -1.29562    0.01151 -112.579  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2948 on 681 degrees of freedom
Multiple R-squared:  0.949,     Adjusted R-squared:  0.9489
F-statistic: 1.267e+04 on 1 and 681 DF,  p-value: < 2.2e-16
```

If we look at the Multiple R-squared value, we discovered that our model fit the formula for the 94.9%. So it approximate the power law. Here's our degree distribution in log scale:

and this is the final power law graph with the portion fitted by our model in figure 6 (in red):

Betweenness We plotted the graph taking in consideration the various betweenness centralities and the corresponding values for the degrees

Nodes with the highest betweenness are also the nodes with higher degree. This is coherent with the fact that that type of nodes are our posts. Thus the nodes with the lowest betweennes are users.

4.1 Simulate an Epidemic Outbreak

The last result that we want to explore is how a pathogen would behave inside our network. As previously mentioned, we used our disassortative network which is not connected. Given these pre-conditions we expect that, on our system, a pathogen would attack isolated nodes or bunch of nodes without infecting the whole network — resulting in minor casualties and limited damage.

To simulate this process we referred to the SIR model, a simple mathematical model of epidemics, that allows us to track the complete process of infection and relative risk based on three different function of time:

1. $S(t)$: number of susceptible in the time t; this value refers to people that are not yet infected but that might be in due time — we can expect this to be people with previous pathologies, elders and so on
2. $I(t)$: number of infected/infectious in the time t; this value refers to people that actually are infected and are yet to recover
3. $R(t)$: number of people that recovered from the pathogen and are now immune

But first we had to setup our baseline: we used the parameters for our model based on the work of Giordano et Al [2] about the recent Covid19 epidemic model:

- tmax = 100
- τ = 0.01
- γ = 0.456
- ρ = 0.034

This parameters will simulate an R_0 of 2.38, a common agreed start value at day one.

We applied this model to our network, and started a simulation: we started by marking each node as S since, for new pathogens, we don't have any immune system that can protect us — much like with Covid19

In due time, we might experience two different outcomes:

I. Every node gets infected
II. Every infected recovers, and the pathogen will get only a small portion of the network

where, given our pre-conditions, we expect the latter.

The model for the simulation given in figure 4 shows what we actually expect, which is a restricted area of interest — only a small group of people actually gets infected and this is due to the network being not connected, combined with the other structural parameters.

To further explore this results, we wanted to know how the model varies as the main parameters change (respecively τ and γ): we mapped the

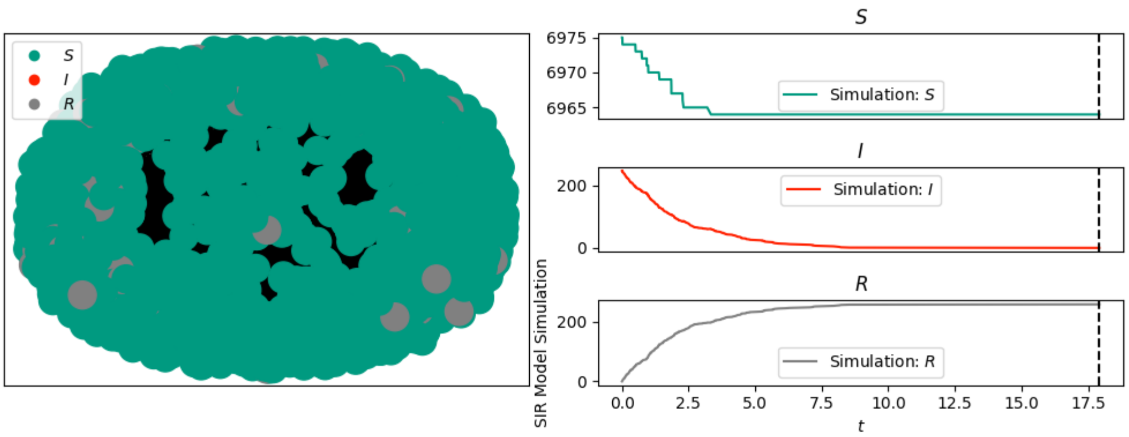


Figure 4: *The SIR simulation on our network*

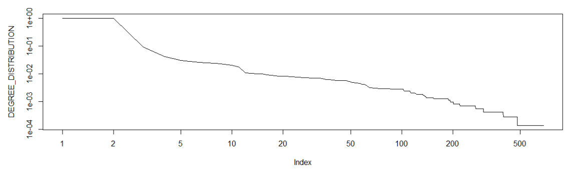


Figure 5: *Degree distribution*

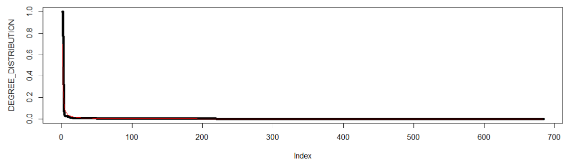


Figure 6: *Degree distribution with Power Law*

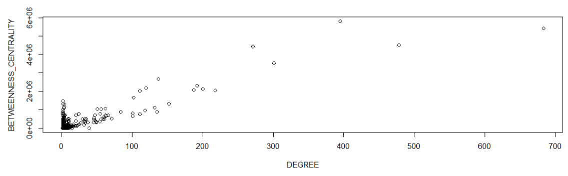


Figure 7: *Betweenness centralities*

various case with different degree for both the infected and the recovered.

We can see that the distribution of the infected follows the distribution of a kind of negative exponential function in which we get our peak of infected right after the beginning of the simulation and showing it fading over time.

This is quite reasonable, if we infect hubs, the pathogen will spread way faster. However, once we start the first recoveries, the epidemics is already almost exhausted. This is also clear if we look at the S variation in time: we register a linear decreasing trend. Due to the nature of our network, the curve describes “quick developments” independently if we talk about infected or recovered.

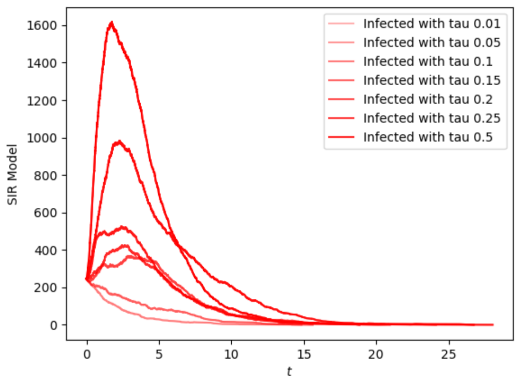


Figure 8: *Infected distribution*

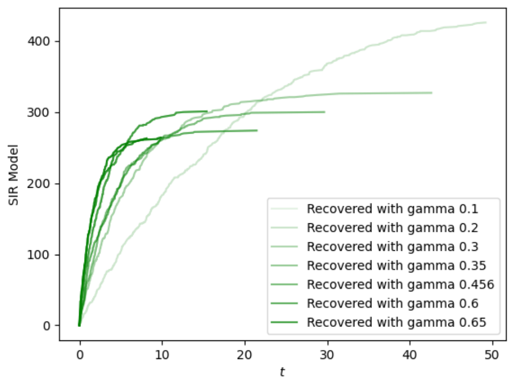


Figure 9: *Recovered distribution*

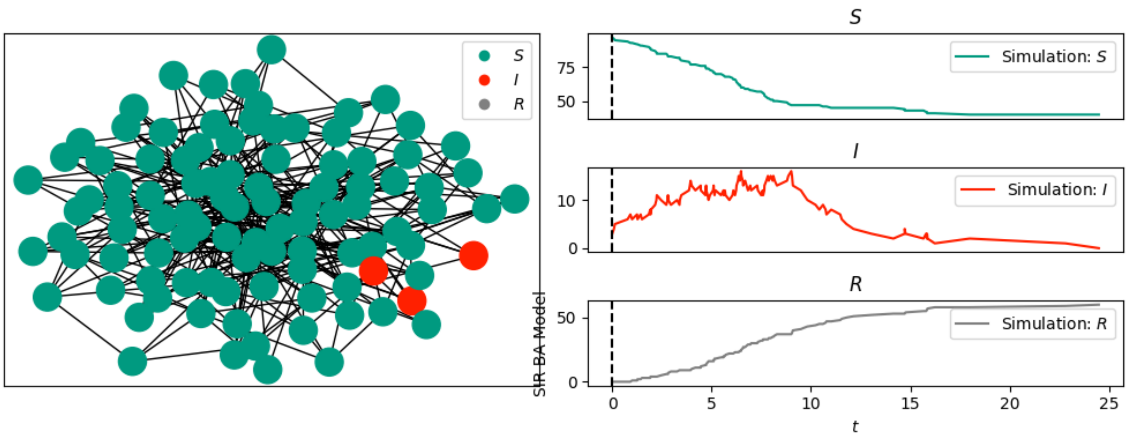


Figure 10: *The SIR simulation on the Barbási-Albert network*

For the recovered people, we see a logarithmic curve with various parameters and an exponential one with more aggressive gammas; apart from a distinct different behaviour for a specific parameter, the other parameters behave in a very similar manner; this is quite expected due to our assumptions: we expect to have a peak of recovered right after the start of the infection. Moreover we expect our curve to be on a similar manner as the one for the infected (with opposite trend).

The last result we want to achieve is to test our simulation against the `Barabási-Albert` model, a random scale-free network model. We created the following random network with this model

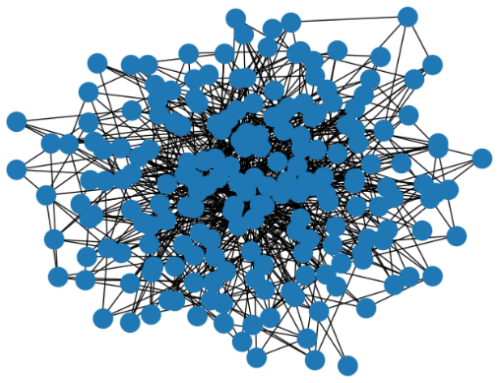


Figure 11: *Barbási-Albert random network*

in which the parameters are $N = 100$ and $m = 4$, and we applied the previous simulation on this network to check if the outcome is similar to the one we found for our network. We expect a longer lifespan of the infection due to the higher connectivity.

5 Discussion

The data that we analyzed are quite promising and reflect what we planned for the social media analysis, the scraper itself is far from being perfect, since `css` classes are randomized at runtime — we don't have any guarantees to find one. Moreover, we tried as much as possible to act as plausible human being with our script (adding sleeps, avoid crawling too deeply, etc.) to avoid being banned from Facebook — eventually we got banned after having collected data — and, as a result, we were not able to further advance with the analysis. However, we showed that it is possible to do.

As regards our network science work, we can make a final synthesis:

- Our Facebook Web scraper is capable of getting the comments for a given post. Furthermore, we can combine this features with a list of public posts of some specific public pages in order to mine the data we need.
- With some addition in python we are able to build a complete sentiment snapshot of our network combining the `networkX` library with the polarity measure given by `polyglot`.
- However, our network makes no claims about completeness or accuracy, since Facebook has a bunch of countermeasures at application

level in order to make our scraper life harder (i.e class randomization in javascript & co)

- The current results are based on a social network without friendship links nor speculations about it. We only represent a link in the form of comment-to-post relashionship (done by a specific user respect to a specific post of a public page), thus the scraper is not guaranteed to produce an exact representation of what's going on in Facebook (Thank you useless Graph API).
- We are able to iterate the collection process in order to create multiple snapshots and then collect also temporal data, which are useful to model the timed evolution of the system.
- Given a specific labeling rule we can assert that Facebook algorithms are probably made to enphasize the separation of people to the extent of influencing the creation of opposing thought groups. This is quite evident given the colored network image we created with Gephi. Further investigation must be conducted in order to establish if this effect in amplified by user choices (likes, comments) or if there is a particular imprint from the algorithm (i.e social engineering).
- As regards our NetSci final network, we showed the main characteristics. This network is comparable to the one we saw in [ns1] from Facebook Northwestern Univ., therefore, despite all the possible inaccuracies we managed to get a result not dissimilar from the expectations.
- Our network is quite disassortative, and approximately is also a scale free network with a small rich club effect. However these result are mainly due to the structural properties that we have imposed by-construction so they must not be used to infer the real Facebook behaviour, but only for the purposes of our AVRC exam.
- Due to the presence of a large number of hubs and users with very little degree (< 2) we can assert that there aren't much chances for an epidemics to have an easy life in our context. If we then add to this the fact that our network is non connected, it's easy to imagine that even with a random walk approach the probability of infection is highest for the hub nodes which therefore, after the transition from I to R, prevent the contagion of all other users connected to them.

We conclude our work saying that we did not expanded further our analysis for the sake of simplicity and, more important, for time reasons. We spent too much time in our "preprocessing" phase (i.e building our support software as the scraper).

However, we walked along an interesting road from which in the future many types of investigations will be possible.

References

[1] V. J. Munster, M. Koopmans, N. van Doremalen, D. van Riel, and E. de Wit, "A novel coronavirus emerging in china — key questions for impact assessment," *New England Journal of Medicine*, vol. 382, no. 8, pp. 692–694, 2020, PMID: 31978293. DOI: 10.1056/NEJMp2000929. eprint: https://doi.org/10.1056/NEJMp2000929. [Online]. Available: https://doi.org/10.1056/NEJMp2000929.

[2] G. Giordano, F. Blanchini, R. Bruno, P. Colaneri, A. Di Filippo, A. Di Matteo, and M. Colaneri, "Modelling the covid-19 epidemic and implementation of population-wide interventions in italy," *Nature Medicine*, vol. 26, 2020, ISSN: 1546-170X. DOI: 10.1038/s41591-020-0883-7. [Online]. Available: https://doi.org/10.1038/s41591-020-0883-7.