# Tobi's Twitter Network

Authors: Tobias Hölzer; Christopher Lohse

Date: November 2020

Abstract:

This is a project about Tobi's Twitter Network. The question we asked ourselves is: "Which people (accounts) are influencing Tobis Feed the most?". To answer this question we are using methods of the Social Network Analysis. Our first approach is to collect all friends (people which Tobi is following) and check which of his friends has overlapping friends. The second approach is to just create a network with all friends and friends of friends (2nd-grade friends) from Tobi and calculate the centrality betweenness of each node.

## 1. Crawling

To crawling Twitter we are using the tweepy library. To use this library it is necessary to get an API-key from Twitter Developer. Our API-keys is stored in `config.py`. A template is included in the repository.

In [34]:
```
import string # for printable comparison
import time # time thread-timeout
import datetime # get current time
import config # for apikeys
import tweepy # for crawling Twitter
```

First, we must authenticate us with our API-keys:

In [35]:
```
# Authenticate Tweepy, a Python Library for crawling Twitter via
the Twitter API
# Get Apikey from here: https://developer.twitter.com/en
auth = tweepy.AppAuthHandler(
    config.APIKEY,
    config.SECRET_APIKEY)
api = tweepy.API(auth)
```

## Get data from Twitter API

Now we want to access Tobis Twitter-ID (Should be `3490529422`):

In [36]:
```
# Get my ID
myName = 'tobiashoelzer'
myID = api.get_user(myName).id # get_user returns a huge User
Object with name, id, etc.

myID
```

Out[36]:
```
3490529422
```

Wow! An ID! How awesome! Now we crawl Tobi's friends, their IDs and Names:

In [37]:
```
# Get my follows (friends)
myFriendsIDs = api.friends_ids(myID) # Returns a list with the IDs
of max. 100 friends
myFriendsNames = {} # Dict where each friend ID is mapped to his
name

# F is for Friends who do stuff together. U is for You and me. N is
for Anywhere and anytime at all. Down here in the deep blue sea! -
so 'f_id' means 'friend_id'
for f_id in myFriendsIDs:
    friend = api.get_user(f_id)
    f_name = ''.join(s for s in friend.name if s in
string.printable) # Cleanup non-printable chars
    myFriendsNames[f_id] = f_name

# Name of the latest added friend of Tobi
myFriendsNames[myFriendsIDs[0]]
```

Out[37]:
```
'Diana Ivanova'
```

After getting a list with the IDs of Tobi's friends, we can go deeper! We crawl all friends of friends from Tobi and store them in a "dict". This may take some time since Twitter's API doesn't allow bigger API-calls like this, so there is at least one 15 min timeout after getting a RateLimitError. Take a cup of tea or do some sport while this executes.

In [38]:

```
# Get 2. Grade Follows (Friends)
secondGradeFriends = {}
print(f'Start crawling {len(myFriendsIDs)} Friends...')
for f_id in myFriendsIDs:
    try:
        # Get friends of current friend (of Tobi) and stores them
in secondGradeFriends dict
        f_friends = api.friends_ids(f_id)
        secondGradeFriends[f_id] = f_friends
    except tweepy.RateLimitError:
        # Prevents crashing if the 300 API-Call Limit from Twitters
API caused an exeption
        # Tries to continue in 15 Minutes again.
        sleep_time = 15 * 60 + 1
        now = datetime.datetime.now()
        tend = now + datetime.timedelta(0, sleep_time)
        print(f'Crawled already {len(secondGradeFriends)}!')
        print(f'Current Time: {now.strftime("%H:%M:%S")}')
        print(f'Sleep for 15 Minutes (until {tend.strftime("%H:%M:
%S")}) to avoid RateLimitErrors. ')
        time.sleep(sleep_time)
        f_friends = api.friends_ids(f_id)
        secondGradeFriends[f_id] = f_friends
```

```
Start crawling 40 Friends...
Crawled already 14!
Current Time: 14:30:17
Sleep for 15 Minutes (until 14:45:18) to avoid RateLimitErrors.
Crawled already 29!
Current Time: 14:45:29
Sleep for 15 Minutes (until 15:00:30) to avoid RateLimitErrors.
```

## Transforming data

After waiting for Twitter to hand over the data we are now able to create an edge list and a node list. The first approach is counting all overlapping friends of Tobi's friends. E.g. Tobi has three friends: Ole, Christopher and Philipp. Ole and Christopher are both following (befriended with) Barack Obama and Elon Musk. So there would be an edge `Ole <--2--> Christopher`. Ole and Philipp are both following Michael Reeves, Alexandria Ocasio-Cortez and Greta Thunberg. This edge would be `Ole <--3--> Philipp`. All edges are after the counting saved to a file called `my-edgy-friends.edges` and all nodes are saved to `my-nody-friends.nodes`:

## String.join

The String.join (e.g. ','.join) joins a list to a string seperated by the string. E.g. :

```
mySeperatorString = '; '
mySeperatorString.join(['Apple', 'Bee', 'Cat'])

# or

'; '.join(['Apple', 'Bee', 'Cat'])
```

Returns:

```
Apple; Bee; Cat
```

In [39]:

```
# Reform data to fit into an edge list

# Opens edge file with write access and write a head row
f = open("my-edgy-friends.edges", "w") # This time 'f' stands for
'file'
f.write(','.join(['User ID', 'User ID', 'Number of overlapping
friends (weight)']) + '\n')

# Array which stores already calculated combinations
matched = []

# I know, there is for sure a better way to do this, but its late
and I want to go home. :)
# Double iteration of myFriends, dont try to understand
for f_id_i in myFriendsIDs:
    f_friends_i = secondGradeFriends[f_id_i]

    for f_id_j in myFriendsIDs:
        # If i and j are same users or i and j in combination was
already calculated continue with next one
        if f_id_j == f_id_i or f'{f_id_j}-{f_id_i}' in matched:
            continue

        f_friends_j = secondGradeFriends[f_id_j]

        # Number which counts the amount of overlapping friends
        same_friends = 0

        # Iterate through friends of i and j to count overlapping
friends
        for f_f_id_i in f_friends_i:
            for f_f_id_j in f_friends_j:
                if f_f_id_i == f_f_id_j:
                    same_friends += 1

        # If overlapping friends exists write them to the edge list
[id of i, id of h, number of overlapping friends]
        if same_friends > 0:
```

In [40]:
```
# Get Names from friends
f = open("my-nody-friends.nodes", "w")
f.write(','.join(['User ID', 'name']) + '\n')
for f_id in myFriendsIDs:
    f.write(','.join([str(f_id), str(myFriendsNames[f_id])]) +
'\n')

# Close file for os-security
f.close()
```

---

## 2. Social Network Analysis

This part is executable without the 1. part! We are now able to read the data from the edge- and node files and put them in a pandas dataFrame:

In [41]:
```
import pandas as pd # for analysing and changing data
import networkx as nx # for creating edges and nodes
import IPython # for showing html output in cells
from pyecharts.charts import Graph # library for visualising
Network
from pyecharts import options as opts # further visualising options
```

In [42]:
```
# Reading Data into pandas DataFrame adjusting columns
df = pd.read_csv("my-edgy-friends.edges",names = ["node1", "node2",
"value"] )
df.drop(index=0, inplace=True) # Drop old header
df.head(2)
```

Out[42]:

| | node1 | node2 | value |
|---|---|---|---|
| 1 | Diana Ivanova | Berlin4Future #RodungsSTOPPjetzt !! | 2 |
| 2 | Diana Ivanova | Marie von den Benken | 2 |

In [43]:
```
f'The DataFrame has the dimension {df.shape[0]} rows and
{df.shape[1]} columns'
```

Out[43]:
```
'The DataFrame has the dimension 433 rows and 3 columns'
```

## Create the Network

We can now extract these edges (rows) from our dataFrame with the help of this function:

In [44]:
```
def create_edgelist(df: pd.DataFrame) -> [(str, str)]:
    """
    takes:
        a pandas datframe with target and source nodes in columsn
    returns:
        an edgelist in form of [(source, target)]

    """
    # Iterating over the whole datframe and append the nodes to the edgelist
    edgelist = []
    for index, row in df.iterrows():
        edgelist.append([str(row.node1), str(row.node2)])
    return edgelist
```

In [45]:
```
edgelist = create_edgelist(df)
```

## Initiate the Network Graph

The edges from the edgelist can now be added to a new nx.Graph object. The nx.Graph extracts all Nodes automatically from the edges:

In [46]:
```
G = nx.Graph()
G.add_edges_from(edgelist)
G.nodes()
```

Out[46]:
```
NodeView(('Diana Ivanova', 'Berlin4Future #RodungsSTOPPjetzt !!', 'Marie von den Benken', 'Mai Thi Nguyen-Kim', 'Tommi Schmitt', 'Scientists for Future', 'Luisa Neubauer', 'DevelopersForFuture  #WeVsClimateCrisis', 'Rezo', 'Ende Gelnde', 'Karl Lauterbach', 'Fridays For Future Germany', 'Nico Semsrott', 'Fabian Kster', 'Hazel Brugger', 'Volksverpetzer', 'Bill Gates', 'Ole', 'teresa bcker', 'extra3', 'Juju', 'Felix Lobrecht', 'Till Reiners', 'erzaehlmirnix', 'Sven Stueven', 'Der Postillon', 'Tobse', 'Schdegie.', 'L. Duy Pham', 'TheMinnieTheMouse', 'Barack Obama', 'Elon Musk', 'ESL', 'INA HOUT', 'ZDF heute-show', 'Joyce', "Let'sPlayBros"))
```

Our first network is an `undirected` `1-mode` network. The edges are `weighted`.

In [47]:
```
print(f'The Network has {len(G.nodes())} nodes and {len(G.edges())} different edges.')
```

```
The Network has 37 nodes and 433 different edges.
```

With the help of the nx library, it is now possible to calculate the `degree_centrality` for all nodes in our nx.Graph object. We want to calculate the degree degree_centrality because we want to find out who has the most contacts shared with Tobi.

In [48]:
```
# Calculating degree_centrality which returns an object with the value for each node:
# {'nodename'(str): degree_centrality_from_node(float), ...}
centrality = nx.degree_centrality(G)
# Making a Top 5 List regarding degree centrality
centrality_top5 = sorted(centrality, key=centrality.get, reverse=True)[:5]
centrality_top5
```

Out[48]:
```
['Ole', 'extra3', 'Mai Thi Nguyen-Kim', 'Luisa Neubauer', 'Rezo']
```

## Visualize the Network Graph

Now we can visualize the Network Graph. In order to achieve this, we create two helper functions to configure the appearance of the nodes and the edges.

In [199…

```
def configure_nodes_visualisation(G : nx.Graph, multiplikator =
15)->[dict]:
    """
    takes:
        an networkx graph object
    returns:
        a list of dicts suitable for pyecharts network
visualisation nodes
    """
    # Calculating the centrality degree of every node in the
network and make a list out of the Top 5
    centrality = nx.degree_centrality(G)
    centrality_top5 = sorted(centrality, key=centrality.get,
reverse=True)[:5]

    # Create a new viz_nodes list from the Graph nodes with
appearence information for each node
    viz_nodes = []
    for node in G.nodes():
        # Fail-fast if the centrality of the node is to low. This
minimizes the network which is very helpfull for large networks
        #if centrality[node] <= 0.001:
           # continue

        # Different appearence for the Top 5 nodes
        if node in centrality_top5: #make another layout for top5
nodes
            viz_nodes.append({
                "name": node,
                "symbol": "triangle",
                "symbolSize": centrality[node]*multiplikator,
                "categorie": "top5",
                "draggable": True,
                "itemStyle": {"color": "#ff3f76"},
                "label": {"show": True},
                "value": round(centrality[node],2)
            })
        else:
```

In [200…

```
def configure_edges_visualisation(df:pd.DataFrame)->[dict]:
    """
    takes:
        an networkx graph object
    returns:
        a list of dicts suitable for pyecharts network
visualisation edges
    """
    # Create a new links list from the Graph nodes with appearence
information for each link / edge
    links = []
    for index, row in df.iterrows():
        links.append({
            "source": str(row.node1),
            "target": str(row.node2),
            "value": row.value
        })

    return links
```

In [201…

```
nodes = configure_nodes_visualisation(G)
nodes[:2] # Example of the created list
```

Out[201…

```
[{'name': 'Diana Ivanova',
  'symbol': 'rect',
  'symbolSize': 7.916666666666667,
  'categorie': 'ordinary',
  'draggable': True,
  'itemStyle': {'color': '#789704'},
  'label': {'show': False},
  'value': 0.53},
 {'name': 'Berlin4Future #RodungsSTOPPjetzt !!',
  'symbol': 'rect',
  'symbolSize': 11.25,
  'categorie': 'ordinary',
  'draggable': True,
  'itemStyle': {'color': '#789704'},
  'label': {'show': True},
  'value': 0.75}]
```

In [202…

```
links = configure_edges_visualisation(df)
links[:2] # Example of two edge dicts
```

Out[202…

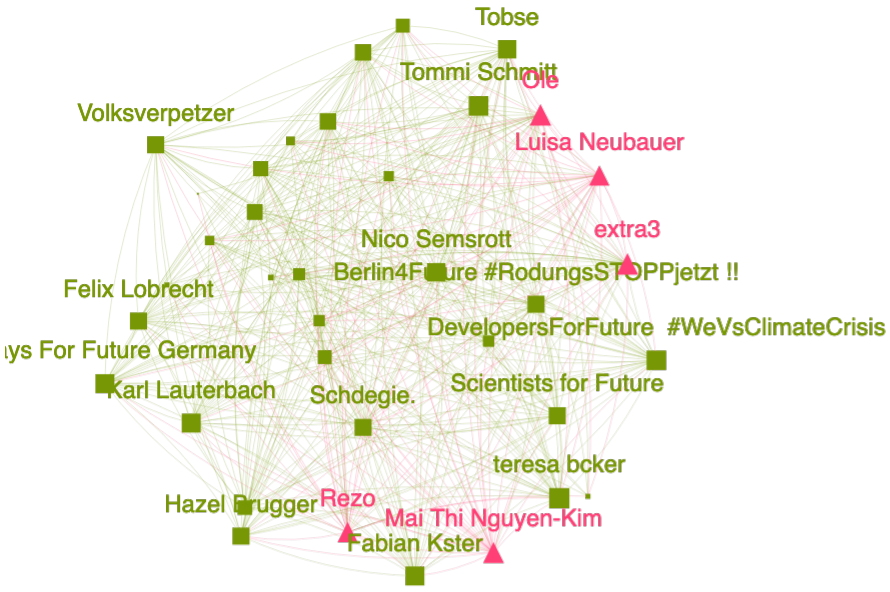
```
[{'source': 'Diana Ivanova',
  'target': 'Berlin4Future #RodungsSTOPPjetzt !!',
  'value': '2'},
 {'source': 'Diana Ivanova', 'target': 'Marie von den Benken', 'value': '2'}]
```

And with these configured links and nodes we can render a Graph Figure:

In [216…

```
# Create a figure object and add nodes and edges (called links)
into the figure format layout, set labels for tooltip
fig = Graph()
fig.add(
    "Tobis Twitter Network",
    nodes,
    links,
    edge_length=300,
    tooltip_opts = "{a} <br> {b} : {c}",
    repulsion=0,
    gravity= 0.5,
    linestyle_opts = opts.LineStyleOpts(width=0.1, curve=0.1,
color="source")
)
fig.height = "900px"
fig.width = "900px"
fig.render_notebook()
```

Out[216…



To see the grafic in big click here

## Analysis

But what can the Visualization tell us? Each link represents overlapping friends between two friends of Tobi. Therefore each link represents the homogeneity of the interests of those two friends. So, the centrallity_degree of a node says how homogeneous the interests of the friend are compared to the rest of Tobi's friends. Therefore this Network shows us which accounts from Tobis Twitter friendslist have the most homogeneous interests compared to Tobi. So that can tell us which friend shares the relative most content which Tobi agrees on, but not which friend influences Tobi the most.

---

# 3. Alternative Approach

To answer the question which friend of Tobi influences his Twitter Feed the most we must try another approach. This approach is to use a simple directed network. So here each edge represents a directed relationship between a 1st-grade friend of Tobi and a 2nd-grade friend of Tobi. E.g. Tobi has two friends (follows two accounts): Günni and Peter. Günni has two more friends, Sarah and Eli. Peter has also a friend: Erik. So there would be the edges: `Günni --> Sarah`, `Günni --> Eli` and `Peter --> Erik`. The edge list is saved to the file `my-edgy-friend-network.edges` but with just the account IDs since it would take to long to crawl Twitter to get the clear names of 20 000 people:

In [78]:
```
f = open("my-edgy-friend-network.edges", "w")
f.write(','.join(['User ID', 'User ID']) + '\n')

for f_id in secondGradeFriends:
    for f_f_id in secondGradeFriends[f_id]:
        f.write(','.join([str(f_id), str(f_f_id)]) + '\n')

f.close()
```

*This can be executed with the saved edge-files from the repository without running the above code.*

Then we can import this data into a dataFrame, create an edgelist, give it into the networkx library to create a nx.Graph object and calculate the centrality for each node:

In [79]:
```
df_alt = pd.read_csv("my-edgy-friend-network.edges", names =
["node1", "node2", "value"])
df_alt.drop(index=0, inplace=True)
df_alt = df_alt.fillna(1)
```

In [80]:
```
edgelist_alt = create_edgelist(df_alt)
```

In [81]:
```
G_alt = nx.Graph()
G_alt.add_edges_from(edgelist_alt)
```

The second network is a `directed` Singlemode network this time, which is `not weighted`.

In [82]:
```
print(f'The Network has {len(G_alt.nodes())} nodes and
{len(G_alt.edges())} different edges.')
```

```
The Network has 18852 nodes and 23906 different edges.
```

Since the Network is quite big it is doesn't make much sense to visualize it, but we can have a further look in order to extract meaning from it.
Since we want to find out who influences Tobi the most we want to find who knows the most influential people in Tobi's network. This means we want to find out who has the most influential friends in the second-degree friend network. To achieve this we calculate again the `betweenness degree scores` for all persons in Tobi's network.

In [172…
```
centrality_alt = nx.degree_centrality(G_alt)
```

Now we have the Top 50 Nodes and can crawl Twitter to get the clear names:

In [174…

```
centrality_alt_top50_ids = sorted(centrality_alt,
key=centrality_alt.get, reverse=True)[:50]
top_50 = []
for topID in centrality_alt_top50_ids:
    friend = api.get_user(topID)
    f_name = ''.join(s for s in friend.name if s in
string.printable) # Cleanup non-printable chars
    top_50.append(f_name)

top_50
```

Out[174…

```
['Barack Obama',
 'extra3',
 'teresa bcker',
 'Berlin4Future #RodungsSTOPPjetzt !!',
 'Luisa Neubauer',
 'Mai Thi Nguyen-Kim',
 'Ole',
 'Ende Gelnde',
 'Scientists for Future',
 'Fridays For Future Germany',
 'Tommi Schmitt',
 'Sven Stueven',
 'DevelopersForFuture  #WeVsClimateCrisis',
 'Volksverpetzer',
 'Juju',
 'Schdegie.',
 'Rezo',
 'ESL',
 'Karl Lauterbach',
 'Bill Gates',
 'Fabian Kster',
 'Marie von den Benken',
 'Diana Ivanova',
 'INA HOUT',
 'Der Postillon',
 'TheMinnieTheMouse',
 'Nico Semsrott',
 'Elon Musk',
 'Tobse',
 'Felix Lobrecht',
 'Hazel Brugger',
 'Till Reiners',
 'erzaehlmirnix',
 'ZDF heute-show',
 'Jan  MASKE AUF  HNDE WASCHEN  Bhmermann ',
 'Kevin Khnert ',
 'Christian Drosten',
 'Greta Thunberg',
 'Tilo Jung',
 'Sascha Lobo',
 'Ralph Ruthe',
 'Marina Weisband',
 'Igor Levit',
 'tagesschau',
 'Stefan Rahmstorf ',
 'Dunja Hayali ',
 'Micky Beisenherz',
 'Margarete Stokowski',
```

```
'ZEIT ONLINE',
```

**So these are the 50 accounts which influence Tobi's Twitter contacts the most.**

## Second Visualisation

So lets have a deeper look at this 50 people by visualizing their Subnetwork:

In [175…

```
G_top50 = G_alt.subgraph(centrality_alt_top50_ids)
```

Build a DataFrame with the top 50 nodes

In [186…

```
source_top50 = []
target_top50 = []
for edge in G_top50.edges():
    target_top50.append(edge[0])
    source_top50.append(edge[1])
df_top_50 = pd.DataFrame()
df_top_50["node1"] = source_top50
df_top_50["node2"] = target_top50
df_top_50["value"] = [pd.NA]*len(df_top_50)
```

Prepare the Subnetwork for Visualisation

In [210…

```
nodes_top50 = configure_nodes_visualisation(G_top50,
multiplikator=15)
links_top50 = configure_edges_visualisation(df_top_50)
```

In [219…

```
# Create a figure object and add nodes and edges (called links)
into the figure format layout, set labels for tooltip
fig = Graph()
fig.add(
    "Tobis Twitter Network Top 50",
    nodes_top50,
    links_top50,
     edge_length=300,
    tooltip_opts = "{a} <br> {b} : {c}",
    repulsion=0,
    gravity= 0.5,
    linestyle_opts = opts.LineStyleOpts(width=0.1, curve=0.1,
color="source")
)
fig.height = "900px"
fig.width = "900px"
fig.render_notebook()
```

Out[219…

To see the grafic in big click here

As you can see the Top 50 Second Degree Network is highly connected, which means that very much of the top 50 nodes share the same followers. Nevertheless, the top 5 nodes have much smaller values for the centrality than the top 5 nodes in the first network.

---

So we have analysed two different networks based on Tob's Twitter Account. One small Network consisting only of the first degree followed People of Tobi and another big Network consisting of Tobis second Degree Network. We hope that our analysis could show how you can analyse one's Twitter Network and the above code should work with every Twitter account.