This is the second part about my project that deals with the Twitter network of members of the Bundestag. After getting the necessary data, which was explained in part 1, we will now focus on creating a network graph with links between the representatives’ Twitter accounts for exploratory network analysis.

Note that again I will not reproduce full code examples but rather focus on some excerpts. If you want to have a look at the full code, please refer to the, especially [friends\_network.R](https://github.com/WZBSocialScienceCenter/mdb-twitter-network/blob/master/friends_network.R) given below. Also, this post only represents a starting point for exploratory network analysis and suggests some packages and techniques for that purpose. It is not an in-depth article about network analysis.

Friends\_network.R

|  |
| --- |
|  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | library(dplyr) |
|  | library(tidyr) |
|  | library(ggplot2) |
|  | library(igraph) |
|  | library(visNetwork) |
|  |  |
|  | # choose the dataset |
|  |  |
|  | #source\_date <- '20181205' |
|  | #source\_date\_title <- 'December 05, 2018' |
|  | source\_date <- '20190702' |
|  | source\_date\_title <- 'July 02, 2019' |
|  |  |
|  | # ---- load and prepare data about deputies and their twitter handles ---- |
|  |  |
|  | dep\_twitter\_full <- read.csv(sprintf('data/deputies\_twitter\_%s.csv', source\_date), stringsAsFactors = FALSE) |
|  | head(dep\_twitter\_full) |
|  |  |
|  | dep\_twitter <- filter(dep\_twitter\_full, !is.na(twitter\_name)) %>% # dismiss rows without Twitter handle |
|  | select(twitter\_name, personal.first\_name, personal.last\_name, personal.gender, personal.birthyear, |
|  | personal.location.state, personal.location.city, party) # these may be variables of interest |
|  |  |
|  | # ---- prepare colors for parties ---- |
|  |  |
|  | unique(dep\_twitter$party) |
|  |  |
|  | party\_colors <- c( # HTML codes for colors to later add a transparency value |
|  | 'SPD' = '#CC0000', |
|  | 'CDU' = '#000000', |
|  | 'DIE GRÜNEN' = '#33D633', |
|  | 'DIE LINKE' = '#800080', |
|  | 'FDP' = '#EEEE00', |
|  | 'AfD' = '#0000ED', |
|  | 'CSU' = '#ADD8E6' |
|  | ) |
|  |  |
|  | party\_colors\_semitransp <- paste0(party\_colors, '40') # add transparency as hex code (25% transparency) |
|  | names(party\_colors\_semitransp) <- names(party\_colors) |
|  |  |
|  | # ---- load and prepare deputies' Twitter connections data ---- |
|  |  |
|  | friends\_full <- readRDS(sprintf('data/deputies\_twitter\_friends\_full\_%s.RDS', source\_date)) |
|  |  |
|  | # in this dataset, "user" is the Twitter handle of the deputy and "screen\_name" and further variables |
|  | # refer to the data of the deputy's Twitter "friend" (i.e. the account she/he follows) |
|  | friends <- select(friends\_full, user, fetch\_friends\_timestamp, fetch\_friendsdata\_timestamp, |
|  | created\_at, screen\_name, name, location, description, |
|  | protected, followers\_count, friends\_count, statuses\_count, account\_created\_at, verified, |
|  | account\_lang) # these may be variables of interest |
|  | head(friends) |
|  |  |
|  | # a few NAs for "screen\_name"; remove those observations |
|  |  |
|  | friends[is.na(friends$screen\_name),] |
|  |  |
|  | friends <- filter(friends, !is.na(screen\_name)) |
|  |  |
|  | # retain only the connections between deputies, not to other Twitter accounts |
|  |  |
|  | dep\_accounts <- unique(friends$user) # Twitter handles of deputies |
|  |  |
|  | dep\_friends <- filter(friends, screen\_name %in% dep\_accounts) # only retain "friends" that are deputies |
|  | head(dep\_friends) |
|  |  |
|  | stopifnot(sum(!(dep\_friends$user %in% dep\_twitter$twitter\_name)) == 0) |
|  |  |
|  | # ---- followings / followers share between parties ---- |
|  |  |
|  | # deputy Twitter handles and their party |
|  | dep\_accounts\_parties <- select(dep\_twitter, twitter\_name, party) |
|  |  |
|  | # make two joins to create a data frame with edges defined by "from\_account", "from\_party" |
|  | # and "to\_account", "to\_party" |
|  | no\_party\_label <- c('fraktionslos', 'parteilos') |
|  | edges\_parties <- select(dep\_friends, from\_account = user, to\_account = screen\_name) %>% |
|  | left\_join(dep\_accounts\_parties, by = c('from\_account' = 'twitter\_name')) %>% |
|  | rename(from\_party = party) %>% |
|  | left\_join(dep\_accounts\_parties, by = c('to\_account' = 'twitter\_name')) %>% |
|  | rename(to\_party = party) %>% |
|  | filter(!(tolower(from\_party) %in% no\_party\_label | tolower(to\_party) %in% no\_party\_label)) |
|  |  |
|  | head(edges\_parties) |
|  |  |
|  | # count how often each "from\_party" -> "to\_party" edge occurs |
|  | counts\_p2p <- group\_by(edges\_parties, from\_party, to\_party) %>% count() %>% ungroup() |
|  | head(counts\_p2p, 10) |
|  | # count the absolute number of edges per "from\_party"; this is required to calculate the proportions |
|  | counts\_party\_edges <- group\_by(counts\_p2p, from\_party) %>% summarise(n\_edges = sum(n)) |
|  | counts\_party\_edges |
|  | # add a column "prop" for the "from\_party" -> "to\_party" edges proportions |
|  | counts\_p2p <- left\_join(counts\_p2p, counts\_party\_edges, by = 'from\_party') %>% |
|  | mutate(prop = n/n\_edges) %>% select(-n\_edges) |
|  | head(counts\_p2p, 10) |
|  |  |
|  | stopifnot(min(counts\_p2p$prop) > 0) |
|  | stopifnot(max(counts\_p2p$prop) <= 1) |
|  |  |
|  | #interaction(counts\_p2p$from\_party, counts\_p2p$to\_party) |
|  |  |
|  | # create a matrix of "friends" proportions with "from\_party" in rows and "to\_party" in columns |
|  | p2p\_mat <- select(counts\_p2p, -n) %>% spread(to\_party, prop) %>% |
|  | mutate\_all(function(x) { ifelse(is.na(x), 0, x) }) # some edge combinations do not occur -> replace NAs with 0 |
|  | p2p\_mat |
|  |  |
|  | # rows must sum up to 1 |
|  | stopifnot(all(rowSums(as.matrix(p2p\_mat[, 2:ncol(p2p\_mat)])) == 1)) |
|  |  |
|  | # to make a heatmap with ggplot, we can't use the matrix but need the "long format" with |
|  | # "from\_party", "to\_party", "prop" columns |
|  | # convert the matrix back to this format, because we already filled in 0 for edge combinations |
|  | # that did not occur |
|  | counts\_p2p <- gather(p2p\_mat, 'to\_party', 'prop', 2:ncol(p2p\_mat)) %>% arrange(from\_party, to\_party) %>% |
|  | mutate(perc = prop \* 100, # we use percent in the plot |
|  | perc\_label = sprintf('%.1f', perc)) # a label to display the rounded number in the cells |
|  | head(counts\_p2p, 10) |
|  |  |
|  | # make a heatmap using geom\_raster |
|  | p <- ggplot(counts\_p2p, aes(x = to\_party, y = from\_party, fill = perc)) + |
|  | geom\_raster() + |
|  | geom\_text(aes(label = perc\_label), color = 'white') + |
|  | scale\_fill\_viridis\_c(guide = guide\_legend(title = 'Followers / following\nshare in percent')) + |
|  | labs(x = 'party in column is followed by party in row', y = 'party in row follows party in column', |
|  | title = 'Proportion of followings / followers between parties', |
|  | subtitle = paste('In percent as of', source\_date\_title)) + |
|  | theme\_minimal() + |
|  | theme(axis.text.x = element\_text(angle = 45, hjust = 1)) |
|  | p |
|  |  |
|  | ggsave(sprintf('plots/p2p\_follower\_shares\_%s.png', source\_date), p, width = 8, height = 6) |
|  |  |
|  | # TODO: do this on deputy level? |
|  |  |
|  | # ---- create an edge list and vertices for igraph ---- |
|  |  |
|  | # we can re-use the edges\_parties data frame as edge list for igraph |
|  | head(edges\_parties) |
|  |  |
|  | # remove accounts that are not connected to any other deputy account |
|  |  |
|  | accounts\_connected <- unique(c(edges\_parties$from\_account, edges\_parties$to\_account)) |
|  | accounts\_not\_connected <- dep\_twitter$twitter\_name[!(dep\_twitter$twitter\_name %in% accounts\_connected)] |
|  | accounts\_not\_connected |
|  |  |
|  | # these accounts are used as vertices (aka nodes) |
|  | dep\_twitter\_connected <- filter(dep\_twitter, twitter\_name %in% accounts\_connected) |
|  |  |
|  | # ---- create an visualize igraph network ---- |
|  |  |
|  | g <- graph\_from\_data\_frame(edges\_parties, vertices = dep\_twitter\_connected) |
|  | g |
|  |  |
|  | # graph centrality scores |
|  |  |
|  | degree\_score <- degree(g, mode = 'total') |
|  | betw\_score <- betweenness(g) |
|  | head(degree\_score) |
|  | head(betw\_score) |
|  |  |
|  | stopifnot(all(names(betw\_score) == names(degree\_score))) |
|  | graph\_scores <- data.frame(twitter\_name = names(degree\_score), |
|  | degr\_score = degree\_score, |
|  | betw\_score = betw\_score, |
|  | row.names = NULL, stringsAsFactors = FALSE) |
|  | graph\_scores <- left\_join(dep\_twitter\_connected, graph\_scores, by = 'twitter\_name') %>% |
|  | mutate(full\_name = paste(personal.first\_name, personal.last\_name)) %>% |
|  | select(twitter\_name, full\_name, degr\_score, betw\_score, party) |
|  |  |
|  | graph\_scores %>% arrange(desc(degree\_score)) %>% head(10) |
|  | graph\_scores %>% arrange(desc(betw\_score)) %>% head(10) |
|  |  |
|  | # set the vertice and edge colors according to party membership |
|  |  |
|  | V(g)$color <- party\_colors[V(g)$party] |
|  | E(g)$color <- party\_colors\_semitransp[E(g)$from\_party] |
|  |  |
|  | # ---- create a layout and plot a static image ---- |
|  |  |
|  | # can try out different layout algorithms |
|  | #lay <- layout\_with\_kk(g) # okay |
|  | #lay <- layout\_with\_fr(g) # not optimal |
|  | lay <- layout\_with\_drl(g, options=list(simmer.attraction=0)) # good separation |
|  |  |
|  | #lay <- layout\_nicely(g) # uses fr |
|  |  |
|  | png(sprintf('plots/dep\_igraph\_%s.png', source\_date), width = 2048, height = 2048, pointsize = 30) |
|  | par(mar = c(1, 1, 3, 1)) |
|  |  |
|  | plot(g, layout = lay, |
|  | vertex.size = 2, vertex.label.cex = 0.7, |
|  | vertex.label.color = 'black', vertex.label.family = 'arial', |
|  | vertex.label.dist = 0.5, vertex.frame.color = 'white', |
|  | edge.arrow.size = 0.2, edge.curved = TRUE) |
|  | title(sprintf('Twitter network of members of the German Bundestag\nas of %s', source\_date\_title), cex = 1.2, line = -0.5) |
|  | legend('topright', legend = names(party\_colors), col = party\_colors, |
|  | pch = 15, bty = "n", pt.cex = 1.25, cex = 0.8, |
|  | text.col = "black", horiz = FALSE) |
|  | dev.off() |
|  |  |
|  | # ---- visNetwork interactive plot ---- |
|  |  |
|  | # convert igraph object to visNetwork data |
|  | vis\_nw\_data <- toVisNetworkData(g) |
|  |  |
|  | # add a title to be displayed when mouse is over a node |
|  | vis\_nw\_data$nodes$title <- sprintf('@%s (%s %s)', vis\_nw\_data$nodes$id, |
|  | vis\_nw\_data$nodes$personal.first\_name, |
|  | vis\_nw\_data$nodes$personal.last\_name) |
|  | head(vis\_nw\_data$nodes) |
|  |  |
|  | # strip transparency from edge color because visNetwork can't handle it |
|  | vis\_nw\_data$edges$color <- substr(vis\_nw\_data$edges$color, 0, 7) |
|  | head(vis\_nw\_data$edges) |
|  |  |
|  | # create a data frame for the legend |
|  | vis\_legend\_data <- data.frame(label = names(party\_colors), color = unname(party\_colors), shape = 'square') |
|  |  |
|  | # create the network |
|  | vis\_nw <- visNetwork(nodes = vis\_nw\_data$nodes, edges = vis\_nw\_data$edges, height = '700px', width = '90%') %>% |
|  | visIgraphLayout(layout = 'layout\_with\_drl', options=list(simmer.attraction=0)) %>% # use same layout as above |
|  | visEdges(color = list(opacity = 0.25), arrows = 'to') %>% # and same transparency |
|  | visNodes(labelHighlightBold = TRUE, borderWidth = 1, borderWidthSelected = 12) %>% # set node highlighting |
|  | visLegend(addNodes = vis\_legend\_data, useGroups = FALSE, zoom = FALSE, width = 0.2) %>% # add legend |
|  | visOptions(nodesIdSelection = TRUE, highlightNearest = TRUE, selectedBy = 'party') %>% # further options |
|  | visInteraction(dragNodes = FALSE) # disable dragging of nodes |
|  |  |
|  | visSave(vis\_nw, file = sprintf('dep\_visnetwork\_%s.html', source\_date)) |

**Preparation**

We collected two datasets in part I: First, a dataset that lists all deputies with their Twitter account names (also known as Twitter *handles*) and some additional information, e.g. their party or their electoral district. This data was collected from [Abgeordnetenwatch.de](https://www.abgeordnetenwatch.de/). I will call this dataset dep\_twitter. Second, for each deputy Twitter account we have collected data on their “friends” using the Twitter API. Remember that “friends” is the Twitter terminology for the list of users that someone follows (i.e. the users that appear in the “following” list of a certain Twitter account). I will call this dataset friends. Let’s have a look at a sample from both datasets first:

A random sample from dep\_twitter:

twitter\_name personal.first\_name personal.last\_name party

johannesvogel Johannes Vogel FDP

baerbelbas Bärbel Bas SPD

schickgerhard Gerhard Schick DIE GRÜNEN

gruenebeate Beate Müller-Gemmeke DIE GRÜNEN

houbenreinhard Reinhard Arnold Houben FDP

A sample from friends, where user refers to a deputy Twitter handle from dep\_twitter, screen\_name is a friend’s Twitter handle and name a friend’s name as stated on Twitter.

user screen\_name name

baerbelbas OlafLies Minister Olaf Lies

baerbelbas COntraPipeline COntra-Pipeline

baerbelbas Earthsmell Arturo de la Vega

johannesvogel Peter\_Schaar Peter Schaar

baerbelbas BILD\_Ruhrgebiet BILD Ruhrgebiet

johannesvogel sophiespelsberg Sophie Spelsberg

schickgerhard ulle\_schauws Ulle Schauws

johannesvogel alias\_ccm Christa C. Müller

johannesvogel marcusmeurer95 Marcus Meurer

johannesvogel andreaslandwehr Andreas Landwehr

There are much more variables in both datasets, but for our purpose the selected columns are just fine. We’re interested only in the links between deputy accounts on Twitter, this means we can omit all observations in friends, where screen\_name doesn’t refer to a deputy Twitter handle. Let’s do this now:

library(dplyr)

# a few NAs for "screen\_name"; remove those observations

friends <- filter(friends, ![is.na](http://is.na)(screen\_name))

dep\_accounts <- unique(friends$user) # Twitter handles of deputies

# only retain "friends" that are deputies

dep\_friends <- filter(friends, screen\_name %in% dep\_accounts)

The dataset dep\_friends now only contains the connections between deputies on Twitter. Connections to Twitter accounts that are not accounts of deputy colleagues, which might also be interesting for further analysis, are removed. This reduces the dataset from ~340,000 observations to ~8,500 observations.

**Friends / followers share between parties**

At first, I want to focus on aggregate data at party level: In a set of Twitter accounts associated with party *A*, how many of those follow an account from party *B*?

The first step to answer this question is to create a dataset that doesn’t only include which deputy follows which colleague on Twitter (dep\_friends already contains this information), but also their respective party affiliation. For this, we can make two joins between dep\_friends and dep\_accounts\_parties, the latter simply mapping deputy Twitter handles to their party:

# deputy Twitter handles and their party

dep\_accounts\_parties <- select(dep\_twitter, twitter\_name, party)

# make two joins to create a data frame with edges defined by

# "from\_account", "from\_party" and "to\_account", "to\_party"

edges\_parties <- select(dep\_friends,

from\_account = user, to\_account = screen\_name) %>%

left\_join(dep\_accounts\_parties,

by = c('from\_account' = 'twitter\_name')) %>%

rename(from\_party = party) %>%

left\_join(dep\_accounts\_parties,

by = c('to\_account' = 'twitter\_name')) %>%

rename(to\_party = party)

The new dataset edges\_parties now contains the connections between deputies. These connections are also called the *edges* of a graph. They are specified in the columns from\_account and to\_account, plus the respective party affiliations as seen here in this random sample:

from\_account to\_account from\_party to\_party

christianduerr bstrasser FDP FDP

jenskoeppen kaiwegner CDU CDU

berlinliebich c\_bernstiel DIE LINKE CDU

stefangelbhaar julia\_verlinden DIE GRÜNEN DIE GRÜNEN

gydej nicolabeerfdp FDP FDP

We can now count the connections at party level using group\_by() and count():

# count how often each "from\_party" -> "to\_party" edge occurs

counts\_p2p <- group\_by(edges\_parties, from\_party, to\_party) %>%

count() %>% ungroup()

head(counts\_p2p, 10)

## from\_party to\_party n

## AfD AfD 180

## AfD CDU 42

## AfD CSU 2

## AfD DIE GRÜNEN 25

## AfD DIE LINKE 27

## AfD FDP 51

## AfD SPD 50

## CDU AfD 5

## CDU CDU 621

## CDU CSU 130

Of course, the size of the factions in the Bundestag differ and the number of Twitter users per faction do too. Hence absolute numbers are not very useful and we will add a column prop with the respective proportions:

# count the absolute number of edges per "from\_party"

# this is required to calculate the proportions

counts\_party\_edges <- group\_by(counts\_p2p, from\_party) %>%

summarise(n\_edges = sum(n))

# add a column "prop" for the "from\_party" -> "to\_party" edges proportions

counts\_p2p <- left\_join(counts\_p2p, counts\_party\_edges,

by = 'from\_party') %>%

mutate(prop = n/n\_edges) %>% select(-n\_edges)

head(counts\_p2p, 3)

## from\_party to\_party n prop

## AfD AfD 180 0.47745358

## AfD CDU 42 0.11140584

## AfD CSU 2 0.00530504

This data can be visualized as a heatmap with ggplot2 and geom\_raster(). This displays the proportions of friends / followers between parties as a matrix, where the intensity of the color in the cells depends on the value in the cell. On the y-axis, i.e. the rows in the matrix, we put the party *A* that follows a party *B* which is listed on the x-axis. We also convert the proportions to percent (new column perc) and display a rounded percentage (perc\_label) inside the cells. The fill color’s scale uses the viridis color map.

library(ggplot2)

ggplot(counts\_p2p, aes(x = to\_party, y = from\_party, fill = perc)) +

geom\_raster() +

geom\_text(aes(label = perc\_label), color = 'white') +

scale\_fill\_viridis\_c(guide = guide\_legend(

title = 'Followers / following\nshare in percent')) +

labs(x = 'party in column is followed by party in row',

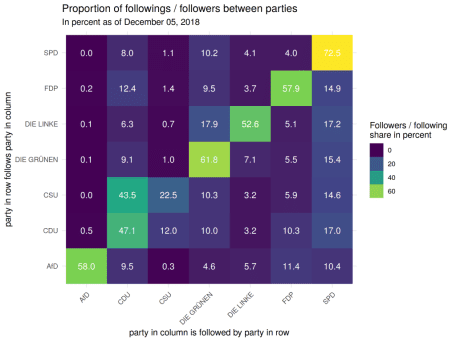
y = 'party in row follows party in column',

title = 'Proportion of followings / followers between parties') +

theme\_minimal() +

theme(axis.text.x = element\_text(angle = 45, hjust = 1))

The following shows a heatmap with data collected on December 5 2018.

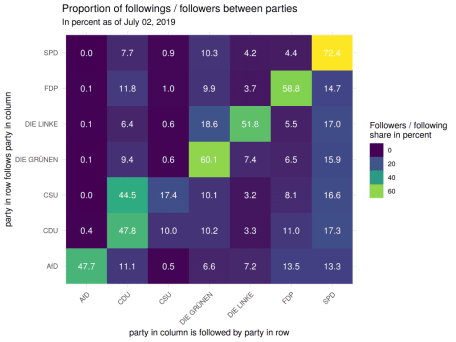
[](https://i0.wp.com/datascience.blog.wzb.eu/wp-content/uploads/10/2019/07/p2p_follower_shares_20181205.png?ssl=1)

We see a strong diagonal, indicating that most parties’ deputies follow party colleagues on Twitter. The only exception here is the CSU, whose deputies twice as often follow CDU colleagues than colleagues from their own party. This doesn’t surprise much though, since both parties form an alliance in the Bundestag and the CSU is the smaller partner.

The SPD has the highest share of intra-party followings. Almost three quarters of their “followings” are towards other SPD colleagues. About 10% of these connections are towards the Green Party and almost 8% towards CDU. At the same time, SPD members are very frequently followed by members of other parties, as you can see at the high values between 10% and 17% across all parties in the SPD column.

The far-right party AfD has the fewest followers from other parties, as we can see in the (due to alphabetic ordering) left-most column. Their members mostly follow FDP and SPD accounts as visible in the bottom row.

This hasn’t changed much in newer data that I collected on July 2 of this year. However, the share of intra-party connections dropped from 58% to 48% for the AfD.

[](https://i2.wp.com/datascience.blog.wzb.eu/wp-content/uploads/10/2019/07/p2p_follower_shares_20190702.png?ssl=1)

Similar things could be done on deputy level, too. However, I will continue with creating and visualizing a graph of the connections at deputy level.

**Creating and visualizing the graph of Twitter friends with igraph**

I will construct a graph of the deputy Twitter friends connections with the package igraph. The graph should display the connections between the individual deputies and also indicate their party membership. There are several ways to construct such a graph. I will use the graph\_from\_data\_frame() function and pass it a data frame of connections (called *edge list*) and a data frame that describes the nodes (*vertices*) with their attributes. In our case, the latter means all deputy Twitter accounts with the respective party affiliation.

We essentially already defined the edge list in the edges\_parties dataset that we created for the heatmap. It specifies the edges with the from\_account and to\_account columns. Additionally, it contains the respective party memberships in the columns from\_party and to\_party. We also already have the data frame that describes the nodes: dep\_twitter. However, if we directly use this dataset to create the graph, we will also include a few stray accounts that don’t connect to any of the other accounts. This is because a few deputies don’t follow any of their colleagues. We will create a nodes dataset without these accounts first:

accounts\_connected <- unique(c(edges\_parties$from\_account, edges\_parties$to\_account))

accounts\_not\_connected <- dep\_twitter$twitter\_name[!(dep\_twitter$twitter\_name %in% accounts\_connected)]

# these accounts are used as vertices (aka nodes):

dep\_twitter\_connected <- filter(dep\_twitter, twitter\_name %in% accounts\_connected)

With this, we can create an igraph graph object now:

library(igraph)

g <- graph\_from\_data\_frame(edges\_parties,

vertices = dep\_twitter\_connected)

g

## IGRAPH 17b614f DN-- 359 8416 --

## + attr: name (v/c), personal.first\_name (v/c), personal.last\_name

## | (v/c), personal.gender (v/c), personal.birthyear (v/n),

## | personal.location.state (v/c), personal.location.city (v/c),

## | party (v/c), from\_party (e/c), to\_party (e/c)

## + edges from 17b614f (vertex names):

## [1] martinschulz->katarinabarley martinschulz->oezoguz

## [3] martinschulz->kahrs martinschulz->schneidercar

## [5] martinschulz->sigmargabriel martinschulz->fbrantner

## [7] fabiodemasi ->victorperli fabiodemasi ->f\_schaeffler

## [9] fabiodemasi ->lgbeutin fabiodemasi ->pascalmeiser

## + ... omitted several edges

The output from the igraph object seems cryptic at first, because it is very condensed: 359 8416 refers to the number of vertices and edges respectively. Then follows a list of attributes after “+ attr”. After each attribute is a specifier in parentheses that denotes the scope of the attribute and its type. So for example name (v/c) means that “name” is a **v**ertex attribute of type **c**haracter, personal.birthyear (v/n) is a **v**ertex attribute of type **n**umeric and from\_party (e/c) is an **e**dge attribute of type **c**haracter. In the “+ edges” section a sample of edges is displayed.

With this igraph object we can calculate several [graph centrality measures](https://en.wikipedia.org/wiki/Centrality) which allows us to identify the most important nodes in a graph. Let’s have a look at two measures: First, the *degree* which is the number of incoming and outgoing edges of a node. Second, the *betweenness* that roughly speaking quantifies the number of shortest paths that pass through a node. Let’s calculate both measures (we use the *total* degree per node, counting both incoming and outcoming edges):

degree\_score <- degree(g, mode = 'total')

betw\_score <- betweenness(g)

head(degree\_score, 3)

## martinschulz fabiodemasi anked

## 6 13 107

head(betw\_score, 3)

## martinschulz fabiodemasi anked

## 0.0000 0.0000 235.4485

We can combine this with the deputy data and order per score (see full script on GitHub) to get a top ten. The first table is ordered by degree, the second by betweenness score.

twitter\_name full\_name degr\_score betw\_score party

1 peteraltmaier Peter Altmaier 263 927.0231 CDU

2 kahrs Johannes Kahrs 251 3482.5345 SPD

3 sigmargabriel Sigmar Gabriel 248 1184.6123 SPD

4 katarinabarley Katarina Barley 224 1135.4914 SPD

5 c\_lindner Christian Lindner 222 1422.2743 FDP

6 hubertus\_heil Hubertus Heil 220 1410.6307 SPD

7 larsklingbeil Lars Klingbeil 216 1129.5727 SPD

8 petertauber Peter Tauber 192 588.7581 CDU

9 sven\_kindler Sven-Christian Kindler 182 917.6740 DIE GRÜNEN

10 berlinliebich Stefan Liebich 179 2478.9492 DIE LINKE

twitter\_name full\_name degr\_score betw\_score party

1 kahrs Johannes Kahrs 251 3482.534 SPD

2 mvabercron Michael von Abercron 149 2543.222 CDU

3 berlinliebich Stefan Liebich 179 2478.949 DIE LINKE

4 f\_schaeffler Frank Schäffler 139 1902.670 FDP

5 c\_lindner Christian Lindner 222 1422.274 FDP

6 hubertus\_heil Hubertus Heil 220 1410.631 SPD

7 ulschzi Ulrike Schielke-Ziesing 45 1353.456 AfD

8 tobiaslindner Tobias Lindner 175 1265.906 DIE GRÜNEN

9 sigmargabriel Sigmar Gabriel 248 1184.612 SPD

10 katarinabarley Katarina Barley 224 1135.491 SPD

We can also visualize our graph with the igraph package. Our graph is quite large and will be better to comprehend when we use different colors for each party. Hence each deputy’s node and outgoing edges should be colored according to her or his party membership. We define a named character vector first with HTML color hex-codes for the nodes and also add a semi-transparent version which we will use for the edges:

party\_colors <- c(

'SPD' = '#CC0000',

'CDU' = '#000000',

'DIE GRÜNEN' = '#33D633',

'DIE LINKE' = '#800080',

'FDP' = '#EEEE00',

'AfD' = '#0000ED',

'CSU' = '#ADD8E6'

)

# add transparency as hex code (25% transparency)

party\_colors\_semitransp <- paste0(party\_colors, '40')

names(party\_colors\_semitransp) <- names(party\_colors)

We can assign a color to nodes and edges, by setting a color attribute for both. The functions V(g) and E(g) give access to the vertex (i.e. node) and edge objects of a graph g. We make the node color dependent on the party attribute of each node. This attribute came from the dataset dep\_twitter\_connected that we passed as vertices argument to graph\_from\_data\_frame() when we constructed our graph. We also passed the edge list edges\_parties there, from which the from\_party attribute of each edge comes. We make the edge color dependent on that attribute:

V(g)$color <- party\_colors[V(g)$party]

E(g)$color <- party\_colors\_semitransp[E(g)$from\_party]

Creating a layout for visualizing a complex graph is not an easy task. You usually want the edges to overlap and cross each other as little as possible. igraph contains several layout generation algorithms for that purpose, which are implemented in functions prefixed by layout\_with\_. I tried out the classic Fruchterman-Reingold algorithm (layout\_with\_fr()), Kamada-Kawai (layout\_with\_kk()) and finally found that Distributed Recursive Layout (Shawn Martin et al., layout\_with\_drl()) provided the best result, as it seems clusters the parties very well (because of the high amount of intra-party connections):

lay <- layout\_with\_drl(g, options=list(simmer.attraction=0))

We’re now ready to visualize the graph with the computed layout. This can be done with the base R plot() function to which we pass the graph object g, the layout lay and several visual adjustments. We also set a title and a legend.

plot(g, layout = lay,

vertex.size = 2, vertex.label.cex = 0.7,

vertex.label.color = 'black', vertex.label.family = 'arial',

vertex.label.dist = 0.5, vertex.frame.color = 'white',

edge.arrow.size = 0.2, edge.curved = TRUE)

title('Twitter network of members of the German Bundestag',

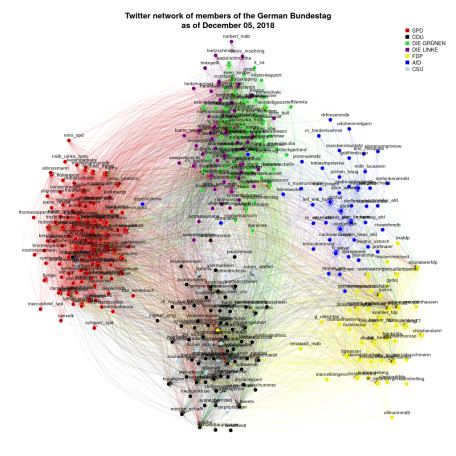
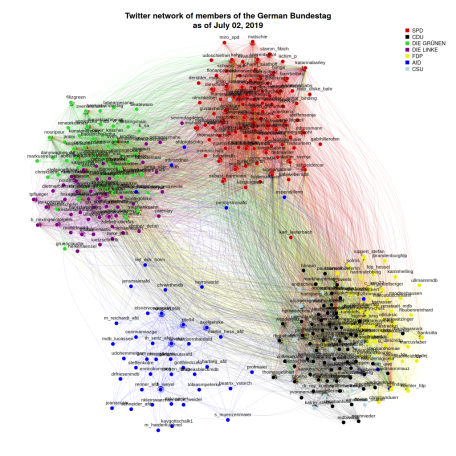
cex = 1.2, line = -0.5)

legend('topright', legend = names(party\_colors), col = party\_colors,

pch = 15, bty = "n", pt.cex = 1.25, cex = 0.8,

text.col = "black", horiz = FALSE)

You can see the plots for the data from December 2018 and July 2019 below. Make sure to click on the thumbnail because a graph of this size can only be visualized properly on a large image.

[](https://i0.wp.com/datascience.blog.wzb.eu/wp-content/uploads/10/2019/07/dep_igraph_20181205.png?ssl=1)[](https://i2.wp.com/datascience.blog.wzb.eu/wp-content/uploads/10/2019/07/dep_igraph_20190702.png?ssl=1)

**Making an interactive network visualization with visNetwork**

Such a static image is fine for smaller graphs but we see that it gets quite crowded and hard to grasp in our scenario. One solution is to generate an interactive graph which allows us to zoom in and out, select specific deputies or parties and display additional information when hovering over certain nodes. The R package visNetwork can be used for that purpose. Switching from igraph to visNetwork is straight forward, as we can convert our igraph object to a visNetwork object via toVisNetworkData():

library(visNetwork)

vis\_nw\_data <- toVisNetworkData(g)

vis\_nw\_data contains two data frames: vis\_nw\_data$nodes and vis\_nw\_data$edges. They’re essentially a tabular form of V(g) and E(g) which means their columns (like color or party) represent the attributes from the igraph nodes and edges.

Setting a title column for the nodes data frame will show this title in the interactive plot when you move the pointer over the node. Here, we set the title to a string of the format “@twitterhandle (Firstname Lastname)”:

vis\_nw\_data$nodes$title <- sprintf('@%s (%s %s)', vis\_nw\_data$nodes$id,

vis\_nw\_data$nodes$personal.first\_name,

vis\_nw\_data$nodes$personal.last\_name)

We strip the transparency channel from the color hex-codes for the edges (the last two characters), because visNetwork can’t display it properly:

vis\_nw\_data$edges$color <- substr(vis\_nw\_data$edges$color, 0, 7)

Finally, we create a data frame that defines the legend:

vis\_legend\_data <- data.frame(label = names(party\_colors),

color = unname(party\_colors),

shape = 'square')

Constructing the interactive network graph works by passing the nodes and edges data frames to visNetwork() and then adjusting the appearance and behavior by concatenating other vis\*() functions with %>% pipe operators. I’ve added comments below that describe the effect of each line:

visNetwork(nodes = vis\_nw\_data$nodes, edges = vis\_nw\_data$edges,

height = '700px', width = '90%') %>%

# use same layout as before

visIgraphLayout(layout = 'layout\_with\_drl',

options=list(simmer.attraction=0)) %>%

# and same transparency

visEdges(color = list(opacity = 0.25), arrows = 'to') %>%

# set node highlighting

visNodes(labelHighlightBold = TRUE, borderWidth = 1,

borderWidthSelected = 12) %>%

# add legend

visLegend(addNodes = vis\_legend\_data, useGroups = FALSE,

zoom = FALSE, width = 0.2) %>%

# show drop down menus and highlight nearest edges

visOptions(nodesIdSelection = TRUE, highlightNearest = TRUE,

selectedBy = 'party') %>%

# disable dragging of nodes

visInteraction(dragNodes = FALSE)

If you run that directly in RStudio, it will show up in the viewer pane. You can also assign the result of the above code to an object like vis\_nw and then store it to an HTML file, which you can share and open in a web browser:

visSave(vis\_nw, file = 'dep\_visnetwork.html')

**Conclusion**

We’ve seen that the first obstacle for creating and analyzing the Twitter network of members of the Bundestag is getting the data. This can be done with a combination of web scraping and querying the Twitter API as I’ve shown in part 1.

After some data preparation, we can already calculate some descriptive aggregate statistics like the friends / followers shares per party. Generating a graph with the igraph package opens the door for numerous network analysis tools. The nodes (vertices) of such a graph are deputy Twitter accounts and links (edges) between them represent who follows who on Twitter. Each node and edge can have additional attributes (meta data) such as name, weight or color. Several functions from igraph allow to compute centrality measures that help to identify important nodes in a network. A static plot of the graph can be generated using one of the layout algorithms that ship with igraph. Interactive plots, which can be created with the package visNetwork, give better insight when dealing with large graphs.

Of course there are a lot more things worth looking at. For example, we could also take into account the full friends network of deputies, i.e. not only concentrate on the links between deputies but also links to other Twitter accounts that are not members of the Bundestag. We’ve also not taken into account many variables from the Abgeordnetenwatch.de data. The tweets of the deputies were also not considered. We would still have to collect them (using the Twitter API), but at least we already have the Twitter handles of the deputies.