Redis Data Network Analysis

Research Questions

We posed the following research questions that we wanted to answer by creating a network graph of reddit users engaged in discussions about the US Election

- Are there distinct communities
- if so, do they largely correspond to subreddits or are they mixed in topics
- What do communities talk about
- are there central / important users within communities
- are the communities in bubbles or is there much inter community exchange

Code

DB Connection

```
library("RPostgres")
library("DBI")
db_con = dbConnect(
  RPostgres::Postgres(),
  dbname = db, host=host_db,
  port=db_port,
  user=db_user,
  password=db_password
)
```

Getting Data

We want to create a network graph with users as vertices. For edge weights we need a similarity measure between users. The only information we have about users is which posts the posted. About the posts we know the content and subreddit, as well as some meta information like upvote count and number of comments. I decided against using the content of posts for the similarity meeasure as that would already be done in the clustering part of our project.

So I needed a list of users x subreddits and the amount of posts a user had made in a particular subreddit. Fetching the data is done via a simple SQL Query. Since we scraped a lot of data partitioned into multiple tables with different focus, I wrote a function that can take in the table we want data from as well as 2 parameters to limit the amount of data retreived

```
fetch_data = function(
    posts_table,
    min_posts = 2,
    max_users = 100
){
    # we want to select the top x users based on the number of posts they made
    query = paste('
      select p.subreddit, u.username, count(*) as count
      from user_posts u
```

```
      inner join "', posts_table, '" p
          on p.url = u.url
      where u.username != \'[deleted]\'
      and u.username in
          (
              select username from user_posts u_i
              inner join "', posts_table, '" p_i
                on p_i.url = u_i.url
              group by username
              having count(username) >=', min_posts,'
              order by count(username) desc
              limit ', max_users, '
          )
      group by 1,2
      order by 3 desc
    ', sep='')
    data = dbGetQuery(db_con, query)
    return(data)
}
```

```
data=fetch_data("US_Election_Posts_By_Subreddits_Year_260924", min_posts=2, max_users=6)
head(data)
```

```
##              subreddit           username count
## 1           Republican intelligentreviews   195
## 2              economy              mafco    58
## 3           Republican interestingfactoid    58
## 4 Political_Revolution       greenascanbe    55
## 5            democrats             jonfla    42
## 6            democrats       greenascanbe     6
```

Now how to construct a similarity measure from this. I wanted a similarity measure such that

- users have high similarity if they posted a lot in the same forum
- if a user posted in a lot of different forums, then he should be less similar to users a particular forum than a user who posts mainly in that one forum.

I then realized the similarity of this situation to string document similarity

- Documents are similar if they contain similar words
- Terms are less important for similarity if they appear in a lot of documents

It seems more natural to treat users as terms appearing as posts in subreddits treated as documents. Since we want to calculate similarity between users (terms) not between documents (subreddits) this is not quite the same as calculating similarity between text documents. It should still be a good enough similarity measure.

```
tf_idf = function(data, term_col, doc_col, val_col){
    terms =  unique(data[[term_col]])
    documents = unique(data[[doc_col]])
    # 0L makes sure the data type is int
    df = data.frame(matrix(0L, ncol = length(terms), nrow = length(documents)))
    colnames(df) = terms
    row.names(df) = documents
    for(i in 1:nrow(data)) {
        row = data[i,]
        df[row[[doc_col]], row[[term_col]]] = as.integer(row[[val_col]])
```

```
    }

    # apply tf idf
    docs.wordcount = rowSums(df)
    docs.wordcount.matrix = matrix(docs.wordcount, dim(df)[1], dim(df)[2])
    tf = log(df / docs.wordcount.matrix + 1)
    # NaN cannot be summed up later, replace it with 0 instead
    tf[is.na(tf)] = 0
    words.doccount = colSums(df > 0)
    words.doccount.matrix =  matrix(words.doccount, dim(df)[1], dim(df)[2], byrow=TRUE)
    idf = log(dim(df)[1] / words.doccount.matrix)

    df.weighted.matrix = tf * idf
    return(df.weighted.matrix)
}
```

```
df.weighted.matrix = tf_idf(data, "username", "subreddit", "count")
head(df.weighted.matrix[, 1:4] ,5)
```

```
##                       intelligentreviews mafco interestingfactoid greenascanbe
## Republican                         0.191 0.000               0.26       0.0018
## economy                            0.021 0.553               0.00       0.0024
## Political_Revolution               0.022 0.000               0.00       0.1015
## democrats                          0.012 0.047               0.00       0.0165
## progun                             0.198 0.000               0.23       0.0000
```

For getting similarities based on the tf idf matrix we can just calculate pairwiese similarities of column vectors

For this we will need to calculate the similarity between numerical vectors, which can be done for example with cosine similarity.

The formula is $\text{sim}_{\cos}(a, b) = \frac{a \cdot b}{\|a\| \|b\|}$

In R code where m is a matrix and a and b are row indices this becomes:

```
cosineSim = function(m, a, b){
    v_a = as.numeric(m[a,])
    v_b = as.numeric(m[b,])
    return (v_a %*% v_b) / ((v_a %*% v_a)^(1/2) (v_b %*% v_b)^(1/2))
}
```

We just need to wrap this in a function that constructs a similarity matrix from the pairs of rows in the tf idf matrix

```
get_similarities = function(tf_idf.matrix){
  documents = rownames(tf_idf.matrix)
  similarities = data.frame(matrix(0, ncol = length(documents), nrow = length(documents))) # OL makes s
  colnames(similarities) = documents
  row.names(similarities) = documents
  i = 1

  for(d1 in documents){
      for(d2 in documents){
          i = i+1
          if(d1 == d2){
              similarities[d1, d2] = 0
          } else {
```

```
            similarities[d1, d2] = cosineSim(tf_idf.matrix, d1, d2)
        }
    }

  }

  # remove rows / cols that consist only of 0.
  # They are also not very interesting for our analysis since they mean that an object has no connectio
  #similarities = similarities[rowSums(similarities != 0) > 0, ]
  #similarities = similarities[, colSums(similarities != 0) > 0]
  # normalize to similarities between 0 and 1
  similarities = (similarities-min(similarities)) / (max(similarities)-min(similarities))
  return(similarities)
}
```

Usually we use the TF IDF matrix to get similarities of rows / documents (here: subreddits) where a row is considered with a matrix with a numerical entry for every term (here: users).

In our case we actually want similarity of columns (terms). We can still use the TF/IDF Matrix and just transpose it so the terms (users) become rows

```
similarities = get_similarities(t(df.weighted.matrix))
head(similarities[, 1:4] ,5)
```

```
##                   intelligentreviews mafco interestingfactoid greenascanbe
## intelligentreviews              0.000 0.096             0.7510       0.0227
## mafco                           0.096 0.000             0.0000       0.0516
## interestingfactoid              0.751 0.000             0.0000       0.0037
## greenascanbe                    0.023 0.052             0.0037       0.0000
## jonfla                          0.073 1.000             0.0000       0.2290
```

Out of curiosity we could also achieve a similar goal by treating users as documents and subreddits as terms. This results some similarities being very similar (e.g. mafco-interestingfactoid) and other similarities being quite different (e.g. mafco-jonfla).

```
inversed.df.weighted.matrix = tf_idf(data, "subreddit", "username", "count")
inversed.similarities = get_similarities(inversed.df.weighted.matrix)
head(inversed.similarities[, 1:4] ,5)
```

```
##                   intelligentreviews mafco interestingfactoid greenascanbe
## intelligentreviews             0.0000 0.028             1.000        0.144
## mafco                          0.0279 0.000             0.000        0.024
## interestingfactoid             1.0000 0.000             0.000        0.065
## greenascanbe                   0.1438 0.024             0.065        0.000
## jonfla                         0.0026 0.015             0.000        0.026
```

For the size of the node I wanted to display how important the node is, A good candidate for this would be centrality. My first approach was to use betweenness centrality

```
btw = betweenness(
    g,
    normalized = TRUE,
)
```

A problem was that betweenness is 0 for the vast majority of vertices but we want a distribution of sizes that is somewhat uniform between a minimum and a maximum size An easy remidy was a linear interpolation between desired min and max value

```
max_value = 1
min_value = 0.2
sizes = (max_value-min_value)*sizes + min_value
```

This still had the issue of many nodes with very similar sizes.

Another approach was to use the softmax function

```
softmax = function(x) {
    b = 3
    exp_x = exp(b * x)  # Calculate exponentials of each element
    return(exp_x / sum(exp_x))  # Normalize by dividing by the sum of exponentials
    }
sizes = softmax(btw)
```

I played with the b value a bit such that values didnt get too similar since we want to keep some variance in sizes

This did not feel right either since the b value would have to be tweeked manually for every new usecase. So I decided to move on.

I tried using eigen centrality

```
sizes = eigen_centrality(g)
```

but the results were also not very satisfying.

I finally settled on using a very simple centrality measure: the sum of edge weights (row sums of the adjacency matrix)

```
get_node_sizes = function(g, adj){
    sizes = rowSums(adj)
    sizes = (sizes-min(sizes)) / (max(sizes)-min(sizes)) # normalization
}
```

Now we can create the graph from the adjacency matrix and perform simple clustering (clusters being called communities in igraph), e.g. with the walktrap algorithm

```
library("igraph");
```

```
##
## Attaching package: 'igraph'
```

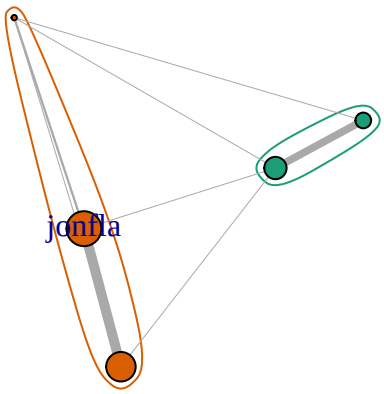
```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
library("RColorBrewer")
make_graph = function (adj){
   # we can use it to plot a graph
    g=graph_from_adjacency_matrix(
        as.matrix(adj),
        mode="undirected",
        diag=FALSE, # not necessary since we set similarity to 0 on the diagonal but cant hurt
        weighted=TRUE,
        );
```

```
    communities = cluster_walktrap(g)
    # alternative would be e.g. cluster_leading_eigen(g)

    sizes = get_node_sizes(g, adj)

    V(g)$size = sizes * 20 # scale to a maximum display size

    # we dont want to show labels for all the vertices since there will be too many
    # display only for top 10 percent (by size)
    threshold = quantile(V(g)$size , 0.9)
    V(g)$label = ifelse(V(g)$size > threshold, V(g)$name, NA)

    # and color the vertices according to their community membership
    V(g)$color = brewer.pal(8, "Dark2")[membership(communities)]

      # Get community membership
    membership_vec = membership(communities)

    # Define specific colors for each community
    num_communities = length(unique(membership_vec))
    community_colors = brewer.pal(8, "Dark2")[1:num_communities]

    graph_layout = layout_with_fr(g)

    plot(
        g,
        layout=graph_layout,
        mark.groups = communities,  # Specify the communities to highlight
        mark.border = community_colors,       # Optional: add a border around the po
        edge.width = E(g)$weight * 5,  # Scale edge width for visibility
        mark.col = NA,                 # No fill color for the community
        mark.expand = 15              # Expand the community outline
    )


    return(list(g, communities))
}
```

```
r = make_graph(similarities)
```



```
g = r[[1]]
communities = r[[2]]
```

This gives us a graph with 2 communities in this case.

We can now start to analyze the clusters. First, lets write the communities to the database for easier selection of posts belonging to communities

```
write_communities_to_db = function(communities){

    membership = data.frame(username=communities$names, community=communities$membership)

    dbWriteTable(
    db_con,
    "user_communities",  # table name
    membership,  # dataframe
    overwrite = TRUE,  # overwrite existing table (default is FALSE)
    append = FALSE  # append to existing table (default is FALSE)
    )
}
```

```
write_communities_to_db(communities)
```

One interesting analysis would be to see if the clusters more or less correspond to singular subreddits or are made up of multiple subreddits. We can visualize this by creating piecharts for each community

```
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:igraph':
##
##     as_data_frame, groups, union
```

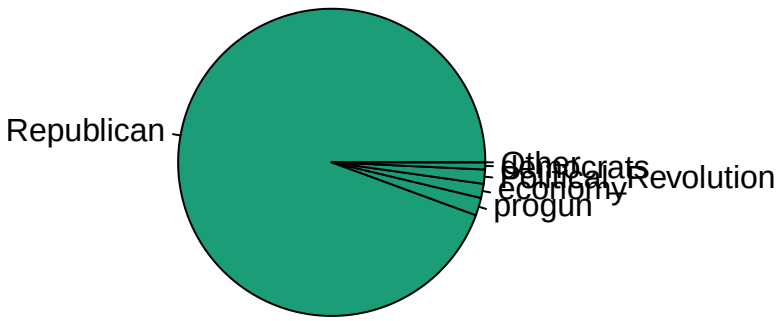
```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

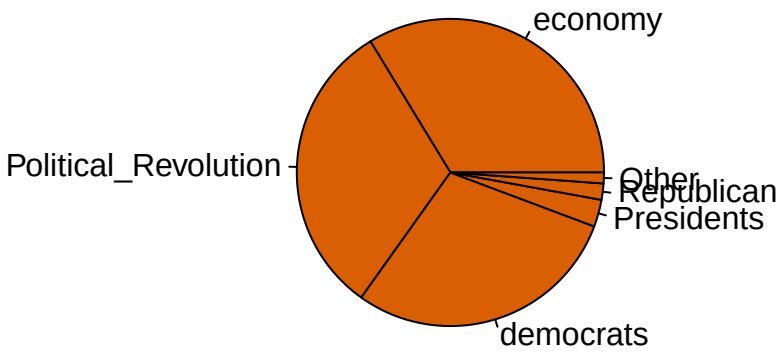
```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
community_piecharts = function(posts_table){
  # now we can add pie charts showing of what proportion of subreddits certain communities are made up
    subreddits_in_communities = dbGetQuery(db_con, paste('
    select c.community, p.subreddit, count(p.subreddit)
    from user_posts u
    inner join "', posts_table, '" p
        on p.url = u.url
    inner join user_communities c
        on c.username = u.username
    group by 1,2
    order by c.community
    ', sep=''))

    communities_names =  unique(subreddits_in_communities$community)
    subreddits_names = unique(subreddits_in_communities$subreddit)
```

```
    community_subreddit = data.frame(matrix(
                                0L,
                                nrow=length(unique(subreddits_in_communities$community)),
                                ncol=length(unique(subreddits_in_communities$subreddit))
    ))
    colnames(community_subreddit) = subreddits_names
    row.names(community_subreddit) = communities_names

    for(i in 1:nrow(subreddits_in_communities)){
        row = subreddits_in_communities[i,]
        community_subreddit[row$community, row$subreddit] = as.integer(row$count)
    }

    for(community in communities_names){
        data = data.frame(
            Count=as.numeric(community_subreddit[community,]),
            Category=colnames(community_subreddit)
        )
        # Select top 5 categories
        top_categories = data %>%
        arrange(desc(Count)) %>%
        slice_head(n = 5)

    # Combine remaining categories into "Other"
        other_count = sum(data$Count[data$Category %in% setdiff(data$Category, top_categories$Category)])
        data_combined = rbind(top_categories, data.frame(Category = "Other", Count = other_count))

        pie(
            data_combined$Count,
            labels=data_combined$Category,
            col=brewer.pal(8, "Dark2")[community]
        )
}
}
```

```
community_piecharts("US_Election_Posts_By_Subreddits_Year_260924")
```





There seems to be one cluster mainly made up of the Republican subreddit. It has also people who post in progun which makes sense for republicans. The other cluster is a little more mixed and seems to be more on the side of democrats.

The titles of the subreddits alone dont give a ton of information. It would now also be interesting to look into what the different communities actually talk about. This could be done for example with a TF / IDF analysis where a document is the concatination of all collected posts in that subreddit.

The TF/IDF approach is has a limitation in this case. The problem is that if there are n unique documents, the Value For document frequency assumes values between 1 and n, so idf also assumes only n unique values at a maximum where $\text{idf}(t) = \log(\frac{n}{\text{df}(t)}) \in [0, \log(n)]$. The problem now is that most words, even words that we are very much interested in like "Trunp" or "democrat", appear in all documents. IDF is therefore not very useful. Instead, we consider a term as less important if it makes up a larger proportion of all occuring terms We still want to remove words that appear very often in all documents though.

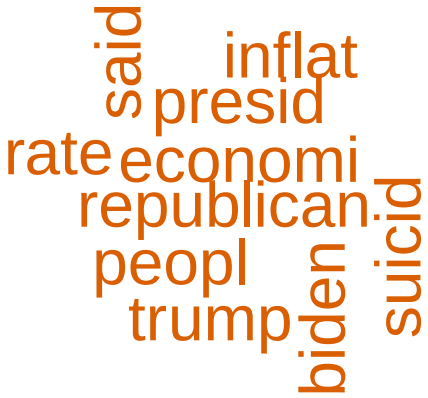
```
library("tm")
```

```
## Loading required package: NLP
```

```
library("SnowballC")
library("wordcloud")
community_wordclouds = function(posts_table, communities){
    posts_in_communities = dbGetQuery(db_con, paste('
      select c.community, string_agg(p.title || p.text, \' \') as text
      from user_posts u
      inner join "', posts_table, '" p
          on p.url = u.url
      inner join user_communities c
          on c.username = u.username
      group by 1
      order by c.community
    ', sep=''))


    communities.corpus = Corpus(VectorSource(posts_in_communities$text))

    communities.corpus = tm_map(communities.corpus, content_transformer(removeNumbers))
    communities.corpus = tm_map(communities.corpus, removePunctuation)
    communities.corpus = tm_map(communities.corpus, tolower)
    communities.corpus = tm_map(communities.corpus, removeWords, stopwords('english'))
    communities.corpus = tm_map(communities.corpus, stripWhitespace)
    communities.corpus = tm_map(communities.corpus, stemDocument)


    communities.dtm = t(as.matrix(TermDocumentMatrix(communities.corpus)))
```

```
    rownames(communities.dtm) = posts_in_communities$community
        # apply tf idf
    docs.wordcount = rowSums(communities.dtm)
    docs.wordcount.matrix = matrix(docs.wordcount, dim(communities.dtm)[1], dim(communities.dtm)[2])
    tf = log(communities.dtm / docs.wordcount.matrix + 1)
    # NaN cannot be summed up later, replace it with 0 instead
    tf[is.na(tf)] = 0

    # modified idf. We dont look at in how many documents does a term occur, but instead how often does
    words.doccount = colSums(communities.dtm)
    words.doccount.matrix =  matrix(words.doccount, dim(communities.dtm)[1], dim(communities.dtm)[2], by
    idf = log(communities.dtm / words.doccount.matrix + 1)


    communities.matrix = tf * idf




    for(i in 1:nrow(communities.matrix)){
        if(sum(communities.matrix[i,]) > 0){
            wordcloud(
                words = colnames(communities.matrix),
                freq = communities.matrix[i,],
                min.freq = 3,
                max.words = 10,
                random.order = F,
                rot.per = 0.35,
                colors = brewer.pal(8, "Dark2")[i],
                scale = c(2,2)
            )
        }
    }
}
```

```
community_wordclouds("US_Election_Posts_By_Subreddits_Year_260924", communities)
```



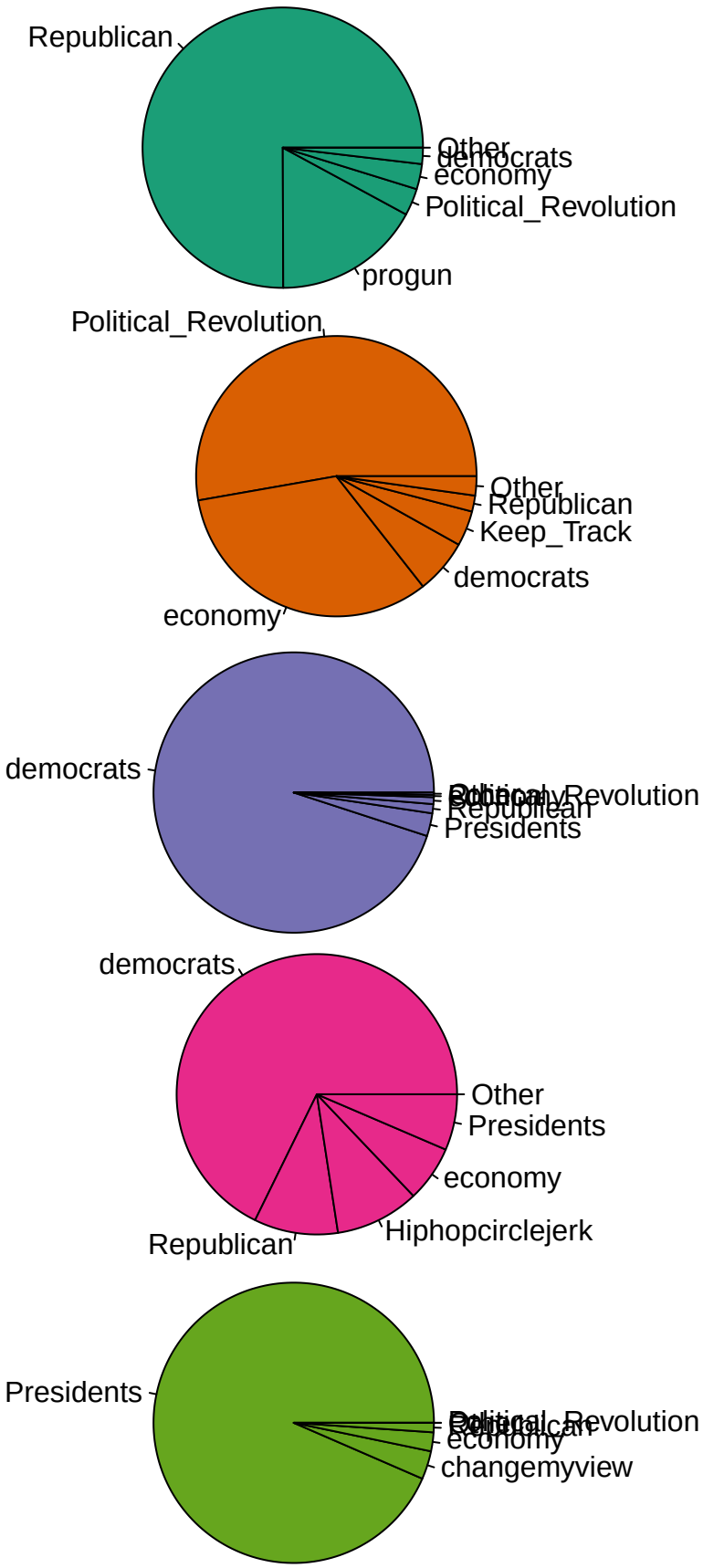
We find that both community share some important terms like “trump” or “biden”. Interestingly the “republican” community seems to talk more about democrats while the “democrat” community talks more about republicans.

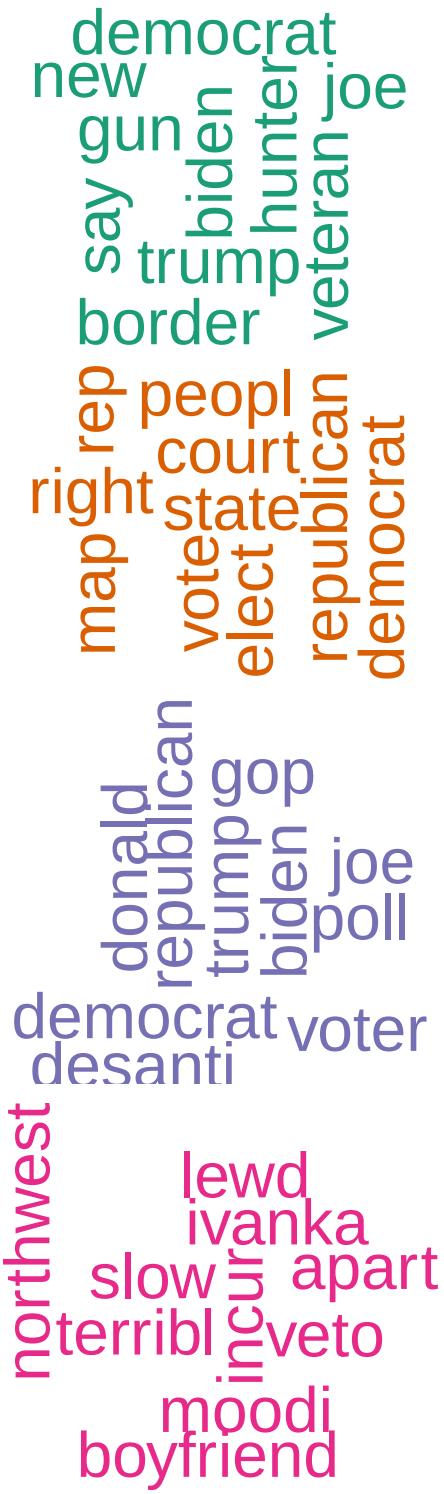
Finally we can put it all together in one reusable function

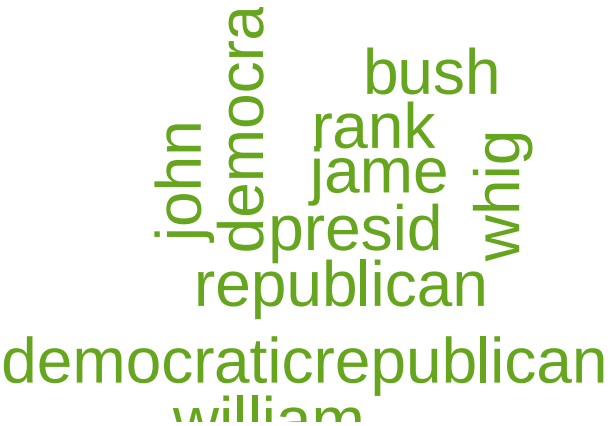
```
data_analysis_and_plot = function(
    posts_table,
    min_posts = 2,
    max_users = 100,
    write_to_file = FALSE
){
    data=fetch_data(posts_table, min_posts, max_users)
    df.weighted.matrix = tf_idf(data, "username", "subreddit", "count")
    similarities = get_similarities(t(df.weighted.matrix))
    r = make_graph(similarities)
    g = r[[1]]
    communities = r[[2]]
    write_communities_to_db(communities)
    community_piecharts(posts_table)
    community_wordclouds(posts_table, communities)
}
```

In our first dataset we scraped all posts from the top 10 subreddits surrounding the us election. Unsurprisingly, this resulted in communities that largely correspond to subreddits and do not overlap a lot. It is interesting to see that the Republican and the progun subreddits are highly associated, which could be expected.

```
data_analysis_and_plot("US_Election_Posts_By_Subreddits_Year_260924")
```


Republican
Other
democrats
economy
Political_Revolution
progun
Political_Revolution
Other
Republican
Keep_Track
democrats
economy
democrats
Presidents
democrats
Other
Presidents
economy
Hiphopcirclejerk
Republican
Presidents
economy
changemyview


democrat
new
joe
gun
hunter
biden
say
veteran
trump
border
rep
peopl
court
republican
right
state
democrat
map
vote
elect
donald
republican
gop
trump
joe
biden
poll
democrat
voter
desanti
northwest
lewd
ivanka
slow
incur
apart
terribl
veto
moodi
boyfriend

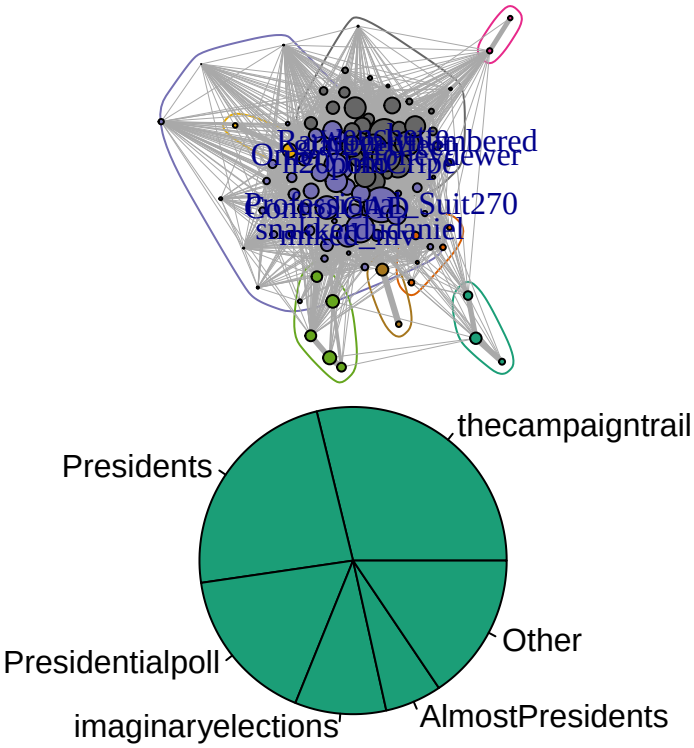


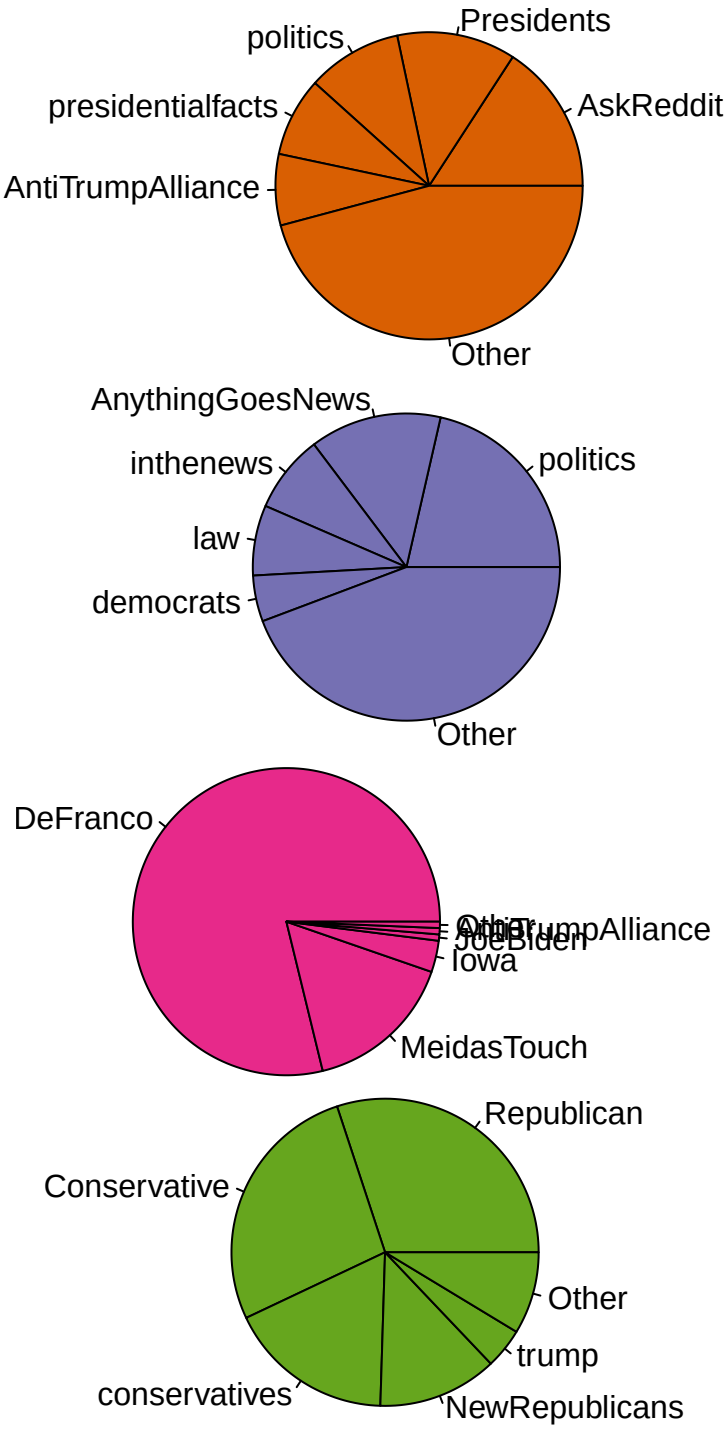
One limitation for the datanalysis that I found is that when looking at only the top 10 subreddits, we get a lot of different users but not many posts by any given user.

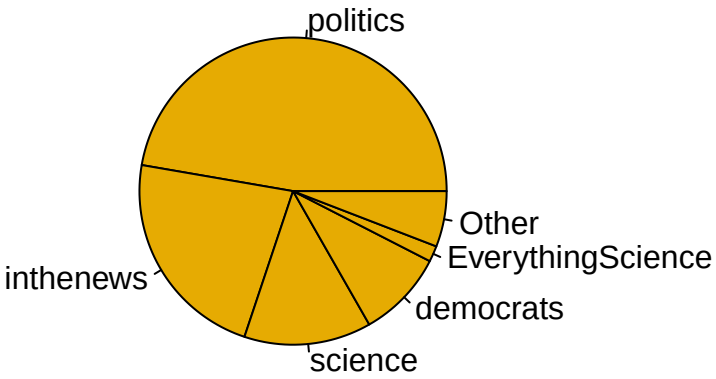
This is why we extended the scraping to scrape all posts by users that we already had identified, but filtered for the US Election. This gives us more communities and also more diverse communities.

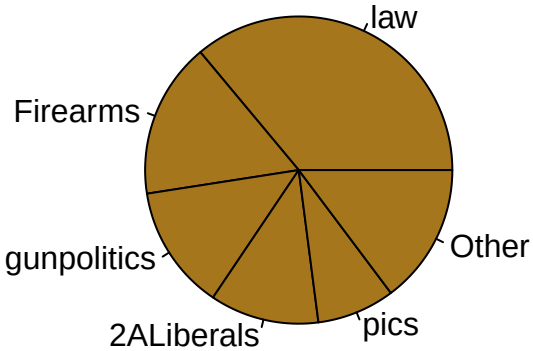
There is still a Republican / Conservatives Community but no democrat dominant community anymore. This could hint at that republicans tend to stay inside their bubble on reddit while democrats tend to post a lot in other forums as well

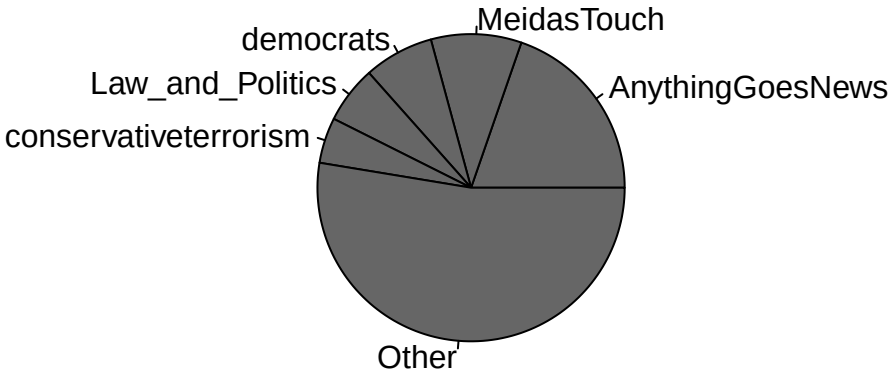
```
data_analysis_and_plot("US_election_posts_by_users")
```




politics
Presidents
presidentialfacts
AskReddit
AntiTrumpAlliance
Other
AnythingGoesNews
inthenews
politics
law
democrats
Other
DeFranco
JoeBiden
Iowa
MeidasTouch
Republican
Conservative
Other
trump
conservatives
NewRepublicans


politics
Other
EverythingScience
democrats
science
inthenews

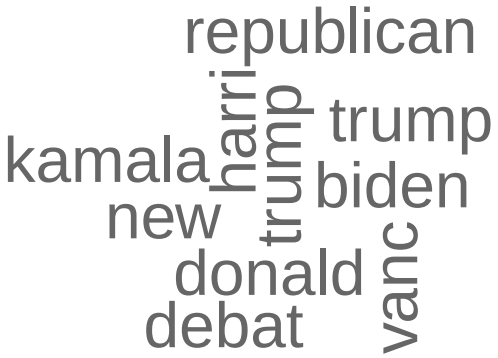

law
Firearms
Other
pics
2ALiberals
gunpolitics


MeidasTouch
democrats
AnythingGoesNews
Law_and_Politics
conservativeterrorism
Other


reconstruct
mocraticrepubl
howev
war
poe
adam
brown
polk
johnpresid


someon
state
vote
cmv
elector
delta
think
reddit
art
curious
democrat
will
donald
say
trump
biden
abort
harri
elect
republican
httpspodlinkhttpspodlink
jessica
hardhit
subscrib
unoffici
denson
etp
podcast
discuss
httpspodlinkhttpspodlink
jessica
hardhit
subscrib
unoffici
denson
etp
podcast
discuss


bidenharri
wouldb
video
joe
trump
attempt
assassin
watch
biden
amp
christi
volcan
emot
teeth
studi
dyson
olar
trump
atf
gallup
gun
unpaid
handgun
order
archer
order
atf
gallup
gun
blanch
unpaid
handgun
firearm
harri
trump
biden
donald



Conclusion and Limitations

The Network Graph Analysis was able to answer all our research questions

- Are there distinct communities: yes there are
- do they largely correspond to subreddits or are they mixed in topics: that depends on the partition of data we look at but there are communities that largely correspond to a small number of subreddits
- What do communities talk about: Looking at the wordclouds gives a basic idea of topics but a further analysis would be needed to answer this question to full satisfaction
- are there central / important users within communities: yes there are, and also users that connect different communities
- are the communities in bubbles or is there much inter community exchange: some communities are very isolated, others are closer to each other and could maybe be considered one bigger community.

A common theme was noticeable during my whole analysis: The end results are highly dependant on many small decisions along the way. Getting a Graph that "looks good" and tells us something that we have been looking for (research questions) is at least in part a matter of making biased decision about what metrics to use, which centrality score, which data basis, how to filter, etc. And especially a lot of trial and error.