Twitch Users Network Analysis

Suhaib Khan

2025-04-10

Install Packages

```r
list.of.packages <- c("tidyverse", "igraph")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages) > 0) {install.packages(new.packages)}
lapply(list.of.packages, require, character.only=T)
```

```
## Loading required package: tidyverse
```

```
## Warning: package 'tidyverse' was built under R version 4.2.3
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

```
## Warning: package 'tibble' was built under R version 4.2.3
```

```
## Warning: package 'tidyr' was built under R version 4.2.3
```

```
## Warning: package 'readr' was built under R version 4.2.3
```

```
## Warning: package 'purrr' was built under R version 4.2.3
```

```
## Warning: package 'dplyr' was built under R version 4.2.3
```

```
## Warning: package 'stringr' was built under R version 4.2.3
```

```
## Warning: package 'forcats' was built under R version 4.2.3
```

```
## Warning: package 'lubridate' was built under R version 4.2.3
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.4     ✔ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.5.0     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.1
## ✔ purrr     1.0.2
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
## Loading required package: igraph
```

```
## Warning: package 'igraph' was built under R version 4.2.3
```

```
##
## Attaching package: 'igraph'
##
## The following objects are masked from 'package:lubridate':
##
##     %--%, union
##
## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union
##
## The following objects are masked from 'package:purrr':
##
##     compose, simplify
##
## The following object is masked from 'package:tidyr':
##
##     crossing
##
## The following object is masked from 'package:tibble':
##
##     as_data_frame
##
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
##
## The following object is masked from 'package:base':
##
##     union
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
```

Data Description

This dataset represent the social network Twitch, an online live-streaming app mostly dedicated to gaming but other types of streaming content are also broadcasted as well. The Nodes are the Twitch users, while the edges are the mutual follower relationships between them.

This dataset consists of 168,114 Nodes and 6,797,557 edges

The dataset can be found in the Stanford University SNAP database: https://snap.stanford.edu/data/twitch_gamers.html

The goal of this analysis is to find distinct communities in the network, and what may define them.

Loading the Data

```r
edges <- read.csv("large_twitch_edges.csv")
features <- read.csv("large_twitch_features.csv")

head(edges)
```

```
##   numeric_id_1 numeric_id_2
## 1        98343       141493
## 2        98343        58736
## 3        98343       140703
## 4        98343       151401
## 5        98343       157118
## 6        98343       125430
```

```r
head(features)
```

```
##    views mature life_time created_at updated_at numeric_id dead_account
## 1   7879      1       969 2016-02-16 2018-10-12          0            0
## 2    500      0      2699 2011-05-19 2018-10-08          1            0
## 3 382502      1      3149 2010-02-27 2018-10-12          2            0
## 4    386      0      1344 2015-01-26 2018-10-01          3            0
## 5   2486      0      1784 2013-11-22 2018-10-11          4            0
## 6   4987      1      1288 2015-04-03 2018-10-12          5            0
##   language affiliate
## 1       EN         1
## 2       EN         0
## 3       EN         1
## 4       EN         0
## 5       EN         0
## 6       EN         1
```

We can see that, edges shows the direct link between nodes (numeric_ids), this shows the mutual connection between two users.Features show extra data about each user, showing how many views they get, if they stream mature content, their language, and if they are an affiliate streamer (They are able to generate income from streaming).

Data Cleaning

The README said that there are no missing values in this dataset, so cleaning should be minimal.

```r
colnames(edges)
```

```
## [1] "numeric_id_1" "numeric_id_2"
```

```r
# changing naming conventions so it is easier to understand what the relationship actually is between them
colnames(edges) <- c("numeric_id_from","numeric_id_to")

# check the datatype of edges data (they should be numeric)
str(edges)
```

```
## 'data.frame':    6797557 obs. of  2 variables:
##  $ numeric_id_from: int  98343 98343 98343 98343 98343 98343 98343 98343 98343 98343 ...
##  $ numeric_id_to  : int  141493 58736 140703 151401 157118 125430 3635 495 116648 1679 ...
```

Now, lets see if there are any accounts following themselves, if there are, we should remove them

```r
sum(edges$numeric_id_from == edges$numeric_id_to)
```

```
## [1] 0
```

The researchers who compiled this dataset stated that it is an undirected graph, so (X -> Z) is equivalent to (Z -> X)

Let's see where the ids start at, if it starts at 0 we should start it at 1

```r
min(edges$numeric_id_from)
```

```
## [1] 0
```

```r
min(edges$numeric_id_to)
```

```
## [1] 0
```

```r
min(features$numeric_id)
```

```
## [1] 0
```

```r
edges <- edges+1
features$numeric_id <- features$numeric_id + 1

min(edges$numeric_id_from)
```

```
## [1] 1
```

```r
min(edges$numeric_id_to)
```

```
## [1] 1
```

```r
min(features$numeric_id)
```

```
## [1] 1
```

igraph Object

Now, let's create an iGraph object

```r
set.seed(10)
twitch_user_network <- graph_from_data_frame(d = edges, directed = F)

twitch_user_network
```

```
## IGRAPH 21ae831 UN-- 168114 6797557 --
## + attr: name (v/c)
## + edges from 21ae831 (vertex names):
##  [1] 98344 --141494 98344 --58737  98344 --140704 98344 --151402 98344 --157119
##  [6] 98344 --125431 98344 --3636   98344 --496    98344 --116649 98344 --1680
## [11] 98344 --123862 98344 --89632  98344 --113418 98344 --145282 98344 --10409
## [16] 98344 --3182   98344 --40676  98344 --95915  98344 --155128 98344 --124828
## [21] 98344 --16784  98344 --122270 98344 --87517  98344 --106970 98344 --10373
## [26] 98344 --66894  98344 --75404  98344 --143082 98344 --44327  98344 --95698
## [31] 98344 --48851  98344 --59893  98344 --159098 98344 --63151  98344 --90149
## [36] 98344 --78900  98344 --30521  98344 --54231  98344 --90698  141494--36838
## + ... omitted several edges
```

Degree distribution

Let's see how the degrees are distributed in our data (Degrees meaning how many connections a node has)

```r
deg <- degree(twitch_user_network)

summary(deg)
```

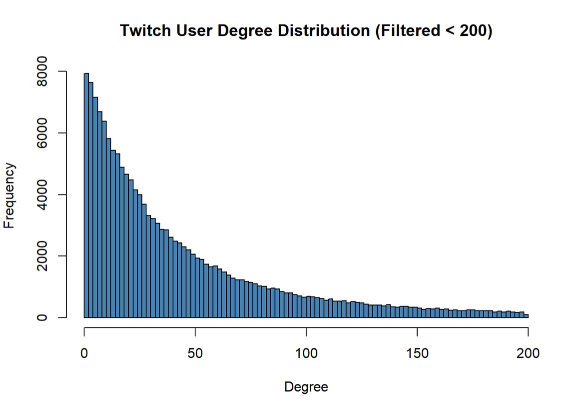
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##     1.00    13.00    32.00    80.87    75.00 35279.00
```

```r
mean(deg)
```

```
## [1] 80.86842
```

We can see that the largest node has 35279 connections, while the overall data has a median of 32 connections. There is an average of 80.87 mutual follower relationship between nodes.

```r
hist(deg[deg < 200],
     breaks = 100,
     main = "Twitch User Degree Distribution (Filtered < 200)",
     xlab = "Degree",
     ylab = "Frequency",
     col = "steelblue")
```



The histogram shows us that many of the node connections are less than 50.

Louvain Cluster

Let's create a Louvain Cluster algorithm to detect communities in our database

```r
set.seed(10)
twitch_comm <- cluster_louvain(twitch_user_network)
length(twitch_comm)
```

```
## [1] 21
```

```r
str(twitch_comm)
```

```
## Class 'communities'  hidden list of 6
##  $ membership : num [1:168114] 1 2 1 1 1 1 1 2 2 2 ...
##  $ memberships: num [1:3, 1:168114] 1 1 1 2 2 2 1 1 1 1 ...
##  $ modularity : num [1:3] 0.421 0.424 0.424
##  $ names      : chr [1:168114] "98344" "141494" "58737" "140704" ...
##  $ vcount     : num 168114
##  $ algorithm  : chr "multi level"
```

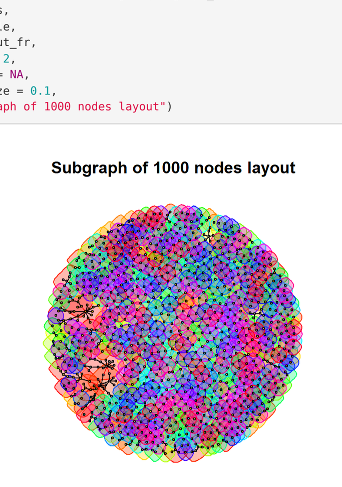
What we can understand from this is, that the Algorithm detected 19 communities within our full dataset, since this is a large set of data, a preliminary should should be to use the algorithm on a smaller sample of the data and see what the network may look like. Note that since the sample is so small relative to the dataset it probably won't tell us much about the 168,114 nodes we actually have.

```r
set.seed(10)
# First lets find the largest connected component in our graph
components_info <- components(twitch_user_network)
components_info$no # We see that the entire graph is connected, so no worries there
```

```
## [1] 1
```

```r
sample_nodes <- sample(V(twitch_user_network), 1000)
subgraph_sample <- induced_subgraph(twitch_user_network, sample_nodes)
subgraph_comms <- cluster_louvain(subgraph_sample,resolution=1)
deg_subgraph <- degree(subgraph_sample)

layout_fr <- layout.fruchterman.reingold(subgraph_sample)
plot(subgraph_comms,
     subgraph_sample,
     layout = layout_fr,
     vertex.size = 2,
     vertex.label = NA,
     edge.arrow.size = 0.1,
     main = "Subgraph of 1000 nodes layout")
```



Interestingly, the graph does not show a lot between the overall connection between the 1000 sampled users, but we can see a tight-knit portion in the bottom left. These users may be a group of streamers who are affiliates, or speak a niche language.

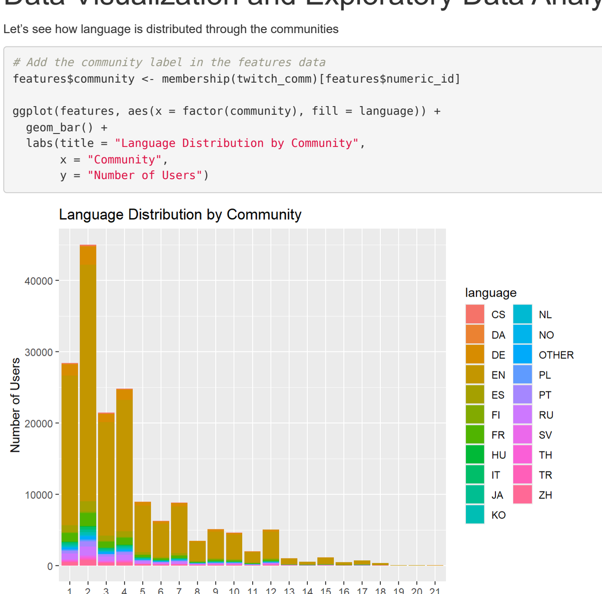
Overall, such a small sample of the dataset will not tell us anything meaningful about the connections of the full datatset.

Data Visualization and Exploratory Data Analysis

Let's see how language is distributed through the communities

```r
# Add the community label in the features data
features$community <- membership(twitch_comm)[features$numeric_id]

ggplot(features, aes(x = factor(community), fill = language)) +
  geom_bar() +
  labs(title = "Language Distribution by Community",
       x = "Community",
       y = "Number of Users")
```



We see that for each community, English is the most dominant language, and there are no distinct communities who share a unique language.

Let's now see the proportion of affiliates/ non-affiliates between each group. Remember that "affiliate" means that the user gets paid for streaming.

```r
library(dplyr)
features$affiliate <- as.factor(features$affiliate)

# Proportion
affiliate_prop <- features %>%
  count(community, affiliate) %>%
  group_by(community) %>%
  mutate(prop = n / sum(n))

ggplot(affiliate_prop, aes(x = factor(community), y = prop, fill = affiliate)) +
  geom_bar(stat = "identity") +
  labs(title = "Affiliate Proportion by Community",
       x = "Community", y = "Proportion")
```

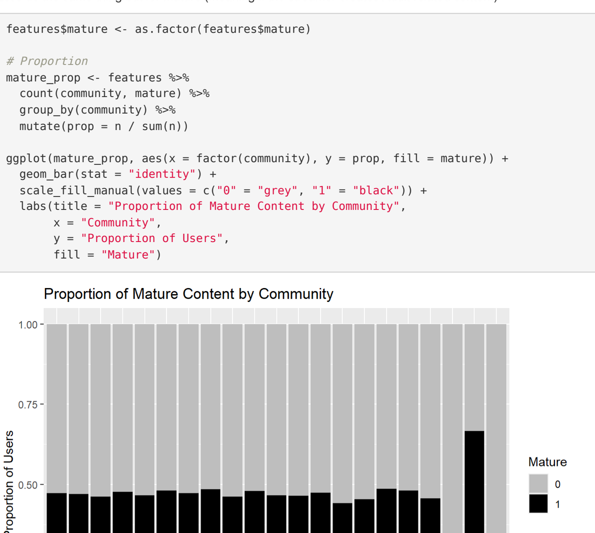
Every group has nearly an even number of affiliates and non-affiliates, except for group 20 who are all non-affiliates.

Let's do the same thing but for mature (meaning if the streamer creates Adult related content)

```r
features$mature <- as.factor(features$mature)

# Proportion
mature_prop <- features %>%
  count(community, mature) %>%
  group_by(community) %>%
  mutate(prop = n / sum(n))

ggplot(mature_prop, aes(x = factor(community), y = prop, fill = mature)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("0" = "grey", "1" = "black")) +
  labs(title = "Proportion of Mature Content by Community",
       x = "Community",
       y = "Proportion of Users",
       fill = "Mature")
```

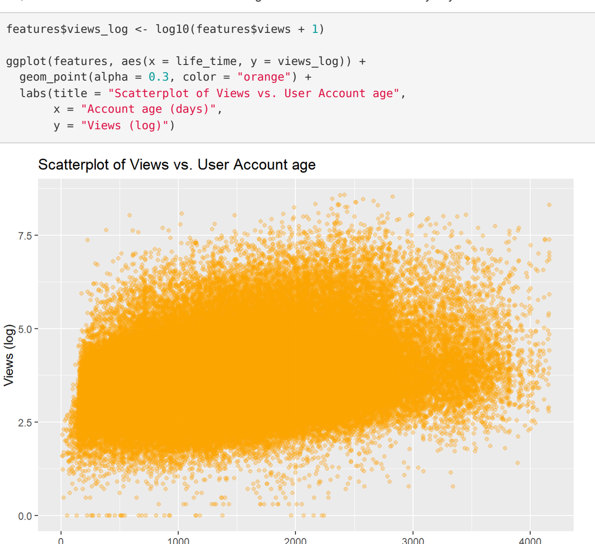


No community is indicated by whether they appeal to mature audiences or not

Now, lets see if the number of views a streamer gets is correlated with how many days their account has been created for (life_time).

```r
features$views_log <- log10(features$views + 1)

ggplot(features, aes(x = life_time, y = views_log)) +
  geom_point(alpha = 0.3, color = "orange") +
  labs(title = "Scatterplot of Views vs. User Account age",
       x = "Account age (days)",
       y = "Views (log)")
```



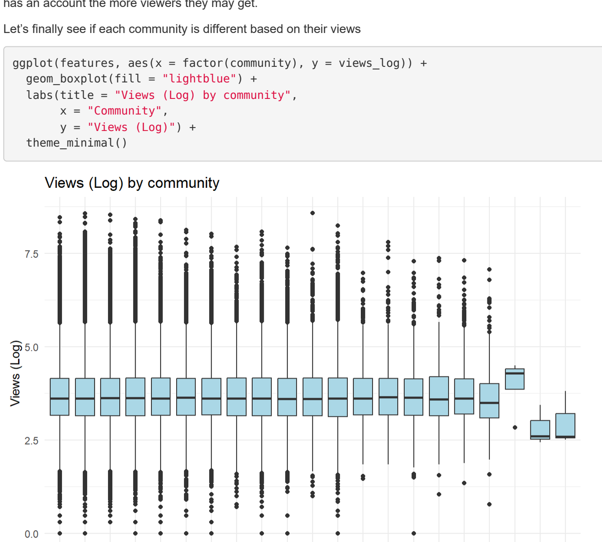

```r
cor(features$views_log, features$life_time)
```

```
## [1] 0.2371278
```

We used a log-scale so this plot would not be dominated by large-viewed streamers. Nonetheless, we can see slight correlation. The longer a user has an account the more viewers they may get.

Let's finally see if each community is different based on their views

```r
ggplot(features, aes(x = factor(community), y = views_log)) +
  geom_boxplot(fill = "lightblue") +
  labs(title = "Views (Log) by community",
       x = "Community",
       y = "Views (Log)") +
  theme_minimal()
```



Once again, each community is not seperable by the views. This is mainly due the the clustering not being based on features. Twitch is a social media streaming community and an individual may not be mutually close with anybody based on if they create mature content, are affiliates, or get more or less views. Two users can be in the same community even if they speak different languages, maybe they create similar content such as gaming.

Primary Data Analysis

Since the data is extremely large, we have to use our lovain variable to look at the network in different levels to get more meaningful information. We should dig deeper into the network and spot smaller, finer communities instead of the big ones.

```r
# view lovain details
str(twitch_comm)
```

```
## Class 'communities'  hidden list of 6
##  $ membership : num [1:168114] 1 2 1 1 1 1 1 2 2 2 ...
##  $ memberships: num [1:3, 1:168114] 1 1 1 2 2 2 1 1 1 1 ...
##  $ modularity : num [1:3] 0.421 0.424 0.424
##  $ names      : chr [1:168114] "98344" "141494" "58737" "140704" ...
##  $ vcount     : num 168114
##  $ algorithm  : chr "multi level"
```

We see that there are three levels in this data.

```r
set.seed(10)
len_com <- function(x) {length(unique(x))}

# Check how many communities at each level
num_of_communities <- apply(twitch_comm$memberships, 1, len_com)
num_of_communities
```

```
## [1] 306  23  21
```

```r
twitch_comm$modularity
```

```
## [1] 0.4205406 0.4240003 0.4240004
```

We see that level 3 has the highest modularity, but less communities in total. Let's focus on level 3 for now.

We will do community detection within the set of communities which are already detected, with the following algorithm.

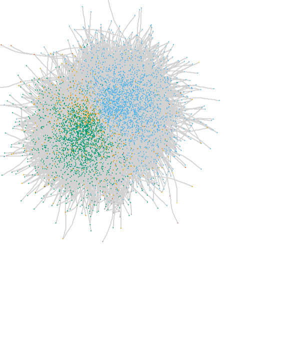
```r
create_subcommunity <- function(graph, initial_communities, community_number){
# Arguments:
# graph: igraph object
# initial_communities: the original community memberships
# community_number: the community number of interest (i.e., the
# community that you want to divide further).
# here we will create a subgraph of just the community of interest
in_community <- which(initial_communities == community_number)
subgraph1 <- induced_subgraph(graph = graph,
vids = in_community)
# We now perform a community detection algorithm (using fast and greedy)
# on the subgraph
comm1 <- cluster_fast_greedy(graph = subgraph1)
# grabbing the community membership of each
# person in the subgraph
mems_subgraph1 <- membership(comm1)
# Now we grab the ids of those in the subgraph, so we can map them back
# onto the original, full network
ids_map <- as.numeric(vertex_attr(subgraph1, "name"))
mems_new <- initial_communities # just copying the original communities
# Here, we begin to relabel the communities so we can put
# them back onto the original set of communities on
# the full network. We want to make sure that
# these new community ids are unique, so we take the max
# original community number and add that to the community
# ids on the subgraph.
mems_subgraph1_relabel <- mems_subgraph1 + max(initial_communities)
# Here we put the new communities onto a vector of community
# membership corresponding to the whole network.
mems_new[ids_map] <- mems_subgraph1_relabel
# Note we just change those in the subgraph of interest.
# We can then relabel all communities, if desired, to take out the old
# community number and put in order from low to high:
num_comms_new <- length(unique(mems_new))
mems_updated <- as.numeric(as.character(factor(mems_new,
labels = 1:num_comms_new)))
# here we output the subgraph, the membership and the updated
# vector of community membership:
return(list(subgraph = subgraph1,
mems_subgraph = mems_subgraph1,
membership_updated = mems_updated))
}
```

Let's run this code on a subcommunity, and also visualize

```r
subcommunity_one <- create_subcommunity(graph = twitch_user_network,
                    initial_communities = twitch_comm$memberships[3,],
                    community_number = 9)


subnet <- subcommunity_one$subgraph
mems_subnet <- subcommunity_one$mems_subgraph
mems_update <- subcommunity_one$membership_updated

plot(subnet, vertex.label=NA, vertex.size=.7,layout=layout.fruchterman.reingold(subnet),edge.color="light gray",
     edge.curved=.2, vertex.frame.color=NA, vertex.color=mems_subnet)
```



Interpretation

We can see that there are two distinct communities which is very interesting when looking at this network. This shows the interconnectiveness and the community of a smaller subgroup of our datatset. Note that this is one subcommunity out of many. In each of the two communities, the nodes are densley connected to eachother, than to other communities.

```r
modularity(twitch_user_network, mems_update)
```

```
## [1] 0.3995984
```

We also notice the modularity score of this subcommunity is relatively high, indicating a strong community structure within Twitch. These communities could be even further broken down in terms of specific demographics.