Problem Set 2, Question 2

Statistics 506, Fall 2017

Solution

The solution here has more detailed discussion of what we are trying to accomplish than your needs to have. Your solution should clearly separate parts a, b, and c and producec a reasonable looking plot.

Data Prep

First we read in the data an prepare our workspace.

```
## load packages
library(tidyverse)
library(stringr)

## read data
social = read_delim(
  'https://jbhender.github.io/Stats506/Stats506_F17_ps2_interactions.csv',
  delim=',')
focal = read_delim(
  'https://jbhender.github.io/Stats506/Stats506_F17_ps2_focal_names.csv',
  delim=',', col_names = "Name")
comm = read_delim(
  'https://jbhender.github.io/Stats506/Stats506_F17_ps2_all_names.csv',
  delim=',', col_names = "Name")
```

Part a

Reformat the data as a data-frame with one row per interaction type for each focal individual in the first column. The reformatted data should have columns for all individuals counting the frequencies of each interaction type with each focal individual. You will need to so some data cleaning to match misspelled names to the name list provided.

Our basic approach here will be to first split the delimited column `toward` into indvidual rows. The first approach below does this using `tidyr::separate` followed by `gather`. This can be done more tidily using `tidyr::separate_rows` but I'll leave this here as an example of figuring out how to use limited tools to accomplish what you want.

Separating `toward`

We first need to split the names from `toward` into a form more conducive to comparing names to the provided lists.

A longer approach

To begin find the maximum number of names in `toward` and separate these into columns. You can then gather the columns to create a data frame with one row per interaction.

```
# max length of 'toward' list
to_split = sapply(str_split(social$toward, ','), length)
#sum(to_split) # want this many rows at end
max_sep = max(to_split)

# separate toward into columns
social_sep = social %>%
  separate(toward,paste0("ID",1:max_sep),sep=",",
           extra="merge",fill="right")
#head(social_sep)

# gather those columns and discard position
individual = social_sep %>%
  gather(IDx,toward,ID1:ID13, na.rm=TRUE) %>%
  select(-IDx)
#head(individual)
```

A tidy approach

We could do the above more compactly using `separate_rows`:

```
ind2 = social %>%
  separate_rows(toward, sep=",")
all.equal(individual, ind2)
```

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

Fix typos in names

Next, we can check the names in `focal` and `toward` against the provided lists.

```
# Focal names are all present
sum(individual$focal %in% focal$Name) == nrow(individual)
```

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

```
# These require cleaning. There are too many to do by hand.
sum(individual$toward %in% comm$Name) == nrow(individual)
```

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

```
individual = individual %>%
  mutate(Matched=individual$toward %in% comm$Name)
table(individual$Matched)
```

```
##
## FALSE  TRUE
## 16110  7851
```

```
head(individual %>% filter(!Matched))
```

```
## # A tibble: 6 x 4
##      focal behavior_cat    toward Matched
##      <chr>        <chr>     <chr>   <lgl>
## 1     Tadd        groom   NIcolas   FALSE
## 2    Zahra         mate      Aham   FALSE
## 3    Elisa        share      Nlan   FALSE
```

```
## 4    Elisa   aggression         ?   FALSE
## 5  Kenyota         mate LaquiNton   FALSE
## 6 Marijane     approach    CoLene   FALSE
```

```
tail(individual$toward)
```

```
## [1] " Fareed"  " Manny"   " Daved"   " Autumn"  " Kade"    " Diandra"
```

In the original call to `separate` (or `separate_rows`) I split on `","` which would leave white space at the front of all but the first name. Let's remove that whitespace and also ensure consistent capitalization. Then we can remove any rows labeled `"?"`.

```
# changing cases and trimming whitespace takes care of most issues
# but there are still too many to do by hand
individual = individual %>%
  mutate(toward = str_trim(str_to_title(toward)),
         Matched= toward %in% comm$Name)
table(individual$Matched)
```

```
##
## FALSE  TRUE
##  1468 22493
```

```
head(individual %>% filter(!Matched))
```

```
## # A tibble: 6 x 4
##        focal behavior_cat toward Matched
##        <chr>        <chr>  <chr>   <lgl>
## 1      Zahra         mate   Aham   FALSE
## 2      Elisa        share   Nlan   FALSE
## 3      Elisa   aggression      ?   FALSE
## 4    Francia         mate  Aquel   FALSE
## 5 Shoshannah        carry     Ia   FALSE
## 6    Francia        share      ?   FALSE
```

```
## Remove ? and "" after recording instances.
n_rm_qm = with(individual,sum(toward=="?" | toward=="", na.rm=TRUE))

individual = individual %>%
  filter(!{toward=="?" | toward==""})
table(individual$Matched)
```

```
##
## FALSE  TRUE
##   291 22493
```

```
head(individual %>% filter(!Matched))
```

```
## # A tibble: 6 x 4
##        focal behavior_cat  toward Matched
##        <chr>        <chr>   <chr>   <lgl>
## 1      Zahra         mate    Aham   FALSE
## 2      Elisa        share    Nlan   FALSE
## 3    Francia         mate   Aquel   FALSE
## 4 Shoshannah        carry      Ia   FALSE
## 5   Marijane        groom  Threas   FALSE
## 6      Zahra        groom Cpriano   FALSE
```

We are now down to 291 unmatched names. We can try to match these using string distances in `agrep` or

`adist`. In this case, I am going to assume deletions are more likely than insertions or substitutions. You were not required to make a similar assumption.

```
# Use approximate matching to search for others.
fuzz_match = lapply({individual %>% filter(!Matched)}$toward,
                    agrep, x=comm$Name,
                    value=TRUE, # Value true returns value and not index.
                    fixed=TRUE, ignore.case=TRUE,
                    max.distance=list(insertions=0L,
                                      deletions=1L,    #Match with at most one deletion
                                      substitutions=0L
                                      )
                    )
amatch_length = sapply(fuzz_match,length)
table(amatch_length)
```

```
## amatch_length
##   1   2   3   4   5   7  16  22  28  57
## 245  29   4   7   1   1   1   1   1   1
```

```
to_match = {individual %>% filter(!Matched)}$toward[which(amatch_length>1)]

## Let's replace the unique matches assuming one deletion in the
## original data.
# First the locations
ind_one_del = with(individual, which(!Matched)[which(amatch_length==1)])
# These are names, because we set value=T
names_one_del = unlist(fuzz_match[which(amatch_length==1)])

##  Double check the indexing:
rplcmnt = with(individual, cbind(toward[ind_one_del], names_one_del))
table(apply(rplcmnt, 1, function(x) adist(x[1], x[2], ignore.case=TRUE)))
```

```
##
##   1
## 245
```

```
## Replace
individual$toward[ind_one_del] = names_one_del
individual$Matched[ind_one_del] = TRUE

## Update to_match
to_match = {individual %>% filter(!Matched)}$toward
```

We found 245 matches with at most a single deletion and updated the rows of `individual` accordingly. Can we do anything with the rest? Let's compute string distances, again preferring deletions to insertions and both to substitutions.

```
## For long matches, use adist
# Convert to lower as dist("Abc","bc") < dist("Abc","Bc")
match_dist = adist(str_to_lower(to_match), str_to_lower(comm$Name))
best = apply(match_dist,1,function(x) which(x==min(x)))

# 38 of 46 have a single best match
table(sapply(best,length))
```

```
##
```

```
##  1  2
## 38  8
```

```
## View them to spot check.
#cbind(to_match[which(sapply(best,length)==1)],
#      comm$Name[unlist(best[which(sapply(best,length)==1)])]
#)

# Prdoduce a matrix of all cases with more than one best match
cbind(to_match[which(sapply(best,length)>1)],
      sapply(best[which(sapply(best,length)>1)],
             function(x){
              paste(comm$Name[x],collapse = ", ")
             }
      )
)
```

```
##      [,1]     [,2]
## [1,] "Amon"   "Aron, Almon"
## [2,] "Armn"   "Aron, Armin"
## [3,] "Taitha" "Tabitha, Tamitha"
## [4,] "Armn"   "Aron, Armin"
## [5,] "Ela"    "Lela, Kla"
## [6,] "Shann"  "Shane, Shanyn"
## [7,] "Jamai"  "Jamari, Jamail"
## [8,] "Arris"  "Parris, Farris"
```

In the first two cases above, one match involoves a subsitution and the other a deletion. Below we repeat with the stated preference.

```
## Repeat best matches with higher costs for insertions and subsititutions
match_dist = adist(str_to_lower(to_match), str_to_lower(comm$Name),
                   cost=list(ins=2, deletions=1, sub=4))
best = apply(match_dist,1,function(x) which(x==min(x)))
best_length = sapply(best, length)
table(best_length)
```

```
## best_length
##  1  2
## 43  3
```

```
# Remaining are entirely ambiguous
cbind(to_match[which(sapply(best,length)>1)],
      sapply(best[which(sapply(best,length)>1)],
             function(x){
              paste(comm$Name[x],collapse = ", ")
             }
      )
)
```

```
##      [,1]     [,2]
## [1,] "Taitha" "Tabitha, Tamitha"
## [2,] "Jamai"  "Jamari, Jamail"
## [3,] "Arris"  "Parris, Farris"
```

Now we can replace the matches we found and remove the rest.

```
# First the locations
ind_one_best = with(individual, which(!Matched)[which(best_length==1)])

# Now get the names
names_one_best = comm$Name[unlist(best[best_length==1])]

#  Double check the indexing:
rplcmnt = with(individual, cbind(toward[ind_one_best], names_one_best))
table(apply(rplcmnt, 1, function(x) adist(x[1], x[2], ignore.case=TRUE)))
```

```
##
##  1
## 43
```

```
# Do the replacement and filter others
individual$toward[ind_one_best] = names_one_best
individual$Matched[ind_one_best] = TRUE

individual = individual %>% filter(Matched)
#dim(individual)
```

Computing frequencies

Now that we have a clean data set, we can can compute frequencies and reshape into wide format.

```
freq =
  individual %>%
  group_by(behavior_cat, focal, toward) %>%
  summarize(freq = n()) %>%
  spread(toward,freq, fill=0) %>% ungroup()
```

Part b

For each interaction type, compute pair-wise canberra distances measuring the similarity between pairs of focal animals. See the R help page for dist() for additional details.

I will approach this by writing a function to return a matrix with `focal` as rownames for a given behavior.

```
behavior_distance = function(df, behavior='aggression'){
  df = df %>% filter(behavior_cat == behavior)
  mat = as.matrix(df %>% select(-focal, -behavior_cat))
  rownames(mat) = df$focal

  dist(mat, method='canberra')
}

dist_mats = lapply(unique(freq$behavior_cat), behavior_distance, df=freq)
attr(dist_mats, 'names') = unique(freq$behavior_cat)
```

Part c

Use multidimensional scaling to find a two-dimensional embedding of the pairwise distances. Use the MDS coordinates to produce plots showing the relations among animals for each

interaction type. Present these plots as a single figure faceted by interaction type.

We can apply `cmdscale` to each distance matrix, save the coordinates, and reshape the data for plotting.

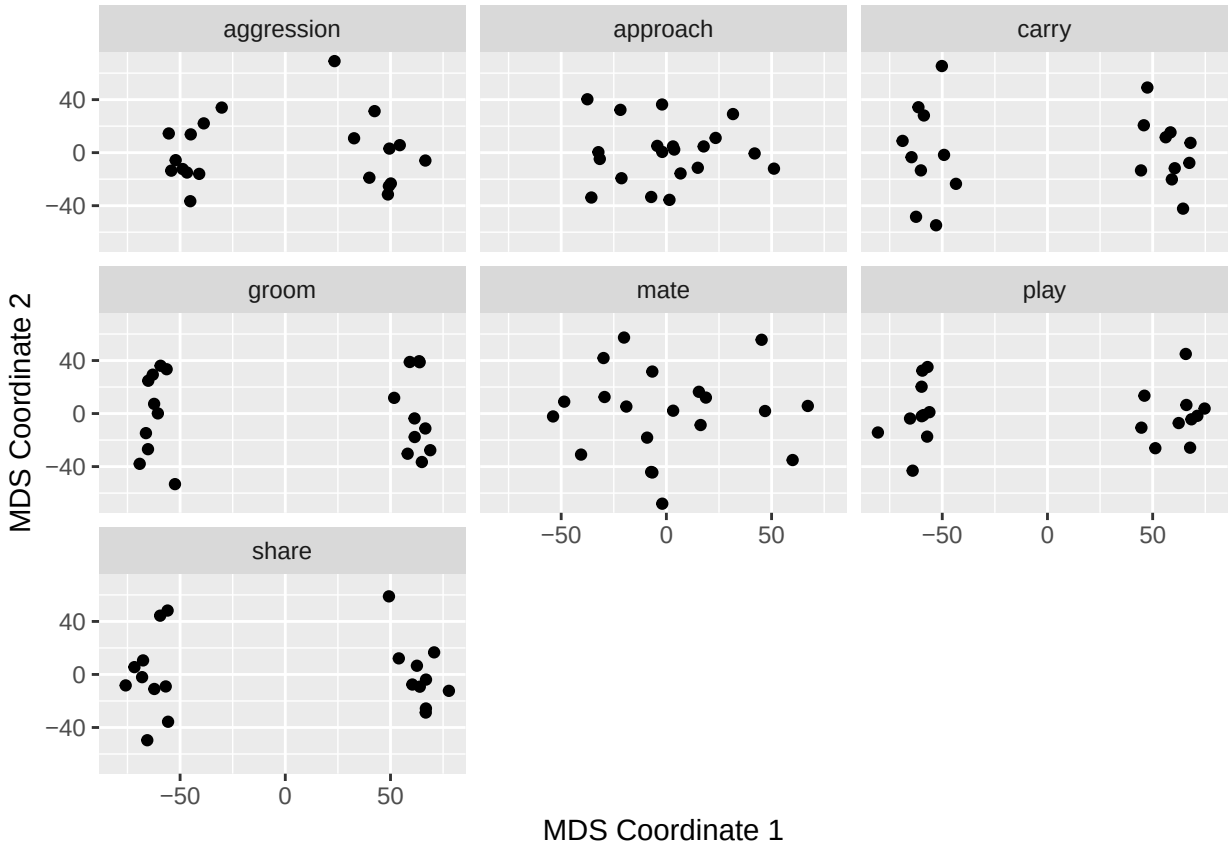
```
dist_MDS = lapply(dist_mats, cmdscale)

## Convert each set to a tibble and join with bind_rows
make_tibble = function(i){
as.tibble(dist_MDS[[i]]) %>%
  mutate(Names=rownames(dist_MDS[[i]]),
         Behavior=names(dist_MDS)[[i]]) %>%
  select(Behavior, Names, V1, V2)
}

mds = bind_rows(lapply(1:length(dist_MDS), make_tibble))
```

Finally, we can do some plotting.

```
mds %>%
  ggplot(aes(x=V1, y=V2)) +
  facet_wrap(~Behavior) +
  geom_point() +
  ylab('MDS Coordinate 2') + xlab('MDS Coordinate 1')
```



Since we distinct groups for some behaviors, we may want to determine whether these are the same accross behvaiors. You were not required to do this, but here is one approach using `kmeans`.

```
# Find clusters and associate with names
km = kmeans(mds %>% filter(Behavior == 'groom') %>% select(V1, V2), 2)
```

```
cluster = km$cluster
attr(cluster, 'names') = {mds %>% filter(Behavior == 'groom')}$Names

mds = mds %>%
  mutate(Cluster = cluster[match(Names, names(cluster))],
         Cluster = sprintf('Cluster %s', Cluster)
         )
```

Now we can color by cluster in the plot.

```
mds %>%
  ggplot(aes(x=V1, y=V2, color=Cluster)) +
  facet_wrap(~Behavior) +
  geom_point(alpha=.5) +
  ylab('MDS Coordinate 2') + xlab('MDS Coordinate 1')
```