Analyze_networks

Preamble

Title: *Script for Objective 1: analyzing seasonal differences in panther networks*
Author: Marie Gilbertson
Date: "08/30/2020"

What this code does:
1. Analyzes differences in wet and dry season panther home range overlap networks

```
#### Clear Environment ####
remove(list=ls())

#### Load libraries ####
library(plyr)
library(dplyr) # group_by, sample_n fxn, ddply
library(igraph)
library(data.table)
library(reshape2)


#### set seed ####
set.seed(8535)

#### load external functions ####
source("../Scripts/bootstrap.node.metrics_clustersamp.R")
```

Set parameters for analysis

```
seasons <- c("Wet_1996", "Dry_1996", "Wet_1997", "Dry_1997", "Wet_1998",
"Dry_1998", "Wet_1999", "Dry_1999", "Wet_2000",
  "Dry_2000", "Wet_2001", "Dry_2001", "Wet_2002", "Dry_2004", "Wet_2005",
"Dry_2005", "Wet_2006", "Dry_2006",
  "Dry_2002", "Wet_2003", "Dry_2003", "Wet_2004")


### choose UDOI cutoff (for edgelist analysis portion)
# options are 0, 0.01, 0.1
co.UDOI <- 0
```

Perform social network analysis (SNA)

Runs as a loop, storing output for each season.

```r
for(j in 1:length(seasons)){

  ### read in edgelist and isolates for the given season
  edgelist.filename <- paste("../Data/UDOI_edgelists/", seasons[j],
"_edgelist.Rdata", sep="")
  edgelist <- get(load(edgelist.filename))

  isolates.filename <- paste("../Data/UDOI_edgelists/", seasons[j],
"_isolates.Rdata", sep="")
  isolates <- get(load(isolates.filename))


  ### subset by current UDOI cutoff
  if(co.UDOI>0){ # if subsetting by a UDOI cutoff, do the following
    edgelist2 <- subset(edgelist, edgelist$UDOI>=co.UDOI)

    orig.inds <- unique(c(edgelist$CATNUMBER.1, edgelist$CATNUMBER.2))
    new.inds <- unique(c(edgelist2$CATNUMBER.1, edgelist2$CATNUMBER.2))

    if(length(orig.inds)!=length(new.inds)){ # if new isolates were created,
update isolates dataframe
      new.isos <- orig.inds[!orig.inds %in% new.inds]
      new.isos <- data.frame(Season = seasons[j],
                             iso.CATNUMBER=new.isos
                             )

      isolates2 <- rbind(isolates, new.isos)
      isolates2 <- na.omit(isolates2) # can get rid of NA's because just now
want to use nrows to determine # of isolates

    }else{ # if no new isolates were created, just rename isolates dataframe
for consistency
      isolates2 <- na.omit(isolates)
    }

  }else{ # if not, subsetting by new UDOI cutoff, just rename dataframes for
consistency
    edgelist2 <- edgelist
    isolates2 <- na.omit(isolates)
  }



  ### create empty dataframe for storing network-level results

  nw.network.analysis.results <- data.frame(net.size = NA,
                                            n.isolates = NA,
                                            dens = NA,
                                            mod.4 = NA,
```

```r
                                              mod.7 = NA,
                                              udoi.co = NA,
                                              season = NA
                                              )


  ### determine total network size (including isolates)
  n.size <- length(unique(isolates2$iso.CATNUMBER)) +
length(unique(c(edgelist2$CATNUMBER.1, edgelist2$CATNUMBER.2)))
  nw.network.analysis.results$net.size <- n.size

  nw.network.analysis.results$n.isolates <-
length(unique(isolates2$iso.CATNUMBER))


  ### create empty dataframe for storing node-level results
  node.network.analysis.results <- data.frame(matrix(nrow = n.size, ncol =
6))
  colnames(node.network.analysis.results) <- c("node.id", "std.deg",
"norm.deg", "std.str",
                                               "udoi.co",  "season")



  ### create igraph network object
  # reorder columns for conversion to igraph network
  edgelist2 <- edgelist2[,c("CATNUMBER.1", "CATNUMBER.2", "UDOI", "Season")]
  g <- graph_from_data_frame(edgelist2, directed = F) %>%
    add_vertices(length(unique(isolates2$iso.CATNUMBER)), name =
paste(isolates2$iso.CATNUMBER))


  ### store node/individual ID's
  node.network.analysis.results$node.id <- vertex_attr(g, "name")


  #### node-level metrics
  ### calculate degree (both standard and normalized)
  std.degree <- as.data.frame(degree(g, normalized = F))

  if(identical(node.network.analysis.results$node.id,
rownames(std.degree))==F){
    print("WARNING: ordering problem for std.degree")
  }else{
    node.network.analysis.results$std.deg <- std.degree$`degree(g, normalized
= F)`

  }
```

```
  norm.degree <- as.data.frame(degree(g, normalized =T))

  if(identical(node.network.analysis.results$node.id,
rownames(norm.degree))==F){
    print("WARNING: ordering problem for norm.degree")
  }else{
    node.network.analysis.results$norm.deg <- norm.degree$`degree(g,
normalized = T)`

  }

  ### calculate strength (standard only)
  # use UDOI as weight
  std.str <- as.data.frame(strength(g, weight = E(g)$UDOI))

  if(identical(node.network.analysis.results$node.id, rownames(std.str))==F){
    print("WARNING: ordering problem for std.str")
  }else{
    node.network.analysis.results$std.str <- std.str$`strength(g, weight =
E(g)$UDOI)`
  }




  #### network-level metrics

  ### calculate network density
  dens <- edge_density(g)
  nw.network.analysis.results$dens <- dens


  ### calculate modularity
  # use different walk lengths to examine sensitivity of modularity to walk
length
  # be sure to use inverse UDOI for "shortest" paths
  mem<-cluster_walktrap(g, weights = 1/E(g)$UDOI, steps = 4)
  memb<-membership(mem)
  nw.network.analysis.results$mod.4 <- modularity(g, membership = memb,
weights = 1/E(g)$UDOI)

  mem<-cluster_walktrap(g, weights = 1/E(g)$UDOI, steps = 7)
  memb<-membership(mem)
  nw.network.analysis.results$mod.7 <- modularity(g, membership = memb,
weights = 1/E(g)$UDOI)
```

```
  ### add tracking data to results files
  node.network.analysis.results$udoi.co <-
nw.network.analysis.results$udoi.co <- co.UDOI
  node.network.analysis.results$season <- nw.network.analysis.results$season
<- seasons[j]


  #### save results

  ndl.results.filename <- paste("../Output/SNA_results/", seasons[j], "_",
co.UDOI, "_UDOI_nodelevel_results.Rdata", sep="")
  save(node.network.analysis.results, file = ndl.results.filename)

  nw.results.filename <- paste("../Output/SNA_results/", seasons[j], "_",
co.UDOI, "_UDOI_networklevel_results.Rdata", sep="")
  save(nw.network.analysis.results, file = nw.results.filename)

}
```

Analyze network metrics

Start by assembling all **network-level** results into one dataset.

```
### assemble all network level results into one dataset
all.network.level <- NULL
for(i in 1:length(seasons)){
  nw.results.filename <- paste("../Output/SNA_results/", seasons[i], "_",
co.UDOI, "_UDOI_networklevel_results.Rdata", sep="")
  temp.nw.results <- get(load(nw.results.filename))

  all.network.level <- rbind(all.network.level, temp.nw.results)
}


# create a column with just season (no year)
all.network.level$season.only <- as.factor(substr(all.network.level$season,
start = 1, stop = 3))
```

Next, analyze those network level metrics with Kruskal-Wallis tests.

```
#make data frame for loop results
KW.test.results <- as.data.frame(matrix(ncol=4, nrow = 3))
colnames(KW.test.results)<- c("metric","p.value","deg.freedom","rank.sum")
KW.test.results$metric <- names(all.network.level[,c(3:5)])



for(i in 1:nrow(KW.test.results)){
```

```
  temp.metric <- names(all.network.level[,c(3:7)])[i]
  KW.test.results[i,2] <-
kruskal.test((formula(paste(temp.metric,"~season.only"))),
data=all.network.level)$p.value
  KW.test.results[i,3] <-
kruskal.test((formula(paste(temp.metric,"~season.only"))),
data=all.network.level)$parameter
  KW.test.results[i,4] <-
kruskal.test((formula(paste(temp.metric,"~season.only"))), data =
all.network.level)$statistic
}
```

View results.

```
KW.test.results

##   metric    p.value deg.freedom rank.sum
## 1   dens 0.15800953           1 1.993173
## 2  mod.4 0.05273094           1 3.752425
## 3  mod.7 0.05273094           1 3.752425
```

Analyze node level metrics with cluster-level bootstrap.

Start by assembling all **node-level** results into one dataset.

```
### read in and assemble all node-level results
all.node.level <- NULL
for(i in 1:length(seasons)){
  ndl.results.filename <- paste("../Output/SNA_results/", seasons[i], "_",
co.UDOI, "_UDOI_nodelevel_results.Rdata", sep="")
  temp.node.level <- get(load(ndl.results.filename))

  # add home range data
  hr.results.filename <- paste("../Data/HR_data/", seasons[i], "_HR area
results.Rdata", sep="")
  hr.data <- get(load(hr.results.filename))
  colnames(hr.data)[colnames(hr.data)=="CATNUMBER"] <- "node.id"

  temp.node.level <- plyr::join(temp.node.level, hr.data[,c("node.id",
"terr.km")], by = "node.id", type = "left")

  all.node.level <- rbind(all.node.level, temp.node.level)
}

# create a column with just season (no year)
all.node.level$season.only <- as.factor(substr(all.node.level$season, start =
1, stop = 3))
all.node.level$year.only <- as.factor(substr(all.node.level$season, start =
5, stop = 8))
```

```r
all.node.level$year.only <- format(as.Date(all.node.level$year.only, format =
"%Y"), "%Y")
```

Next, use cluster-level bootstrapping to generate confidence intervals for the relationship between node-level metrics (outcome) and home range size and season.

```r
## bootstrapping by sampling individuals by their number of observations
(cluster size)
nsims <- 1000 # number of simulations per bootstrap
coefs <- c("intercept","terr.km", "season.only")


deg.bootstrap_clust <- bootstrap.node.metrics_clustersamp(nsims = nsims,
                                       metric = "norm.deg",
                                       coefs = coefs,
                                       dataset = all.node.level,
                                       co.UDOI = co.UDOI,
                                       progress = F # don't print progress
bar
)


str.bootstrap_clust <- bootstrap.node.metrics_clustersamp(nsims = nsims,
                                       metric = "std.str",
                                       coefs = coefs,
                                       dataset = all.node.level,
                                       co.UDOI = co.UDOI,
                                       progress = F # don't print progress
bar
)
```

View the results, as well as original coefficient estimates.

```r
# original estimates
base.model_deg <- lm(norm.deg ~ log(terr.km) + season.only,
data=all.node.level)
base.model_str <- lm(std.str ~ log(terr.km) + season.only,
data=all.node.level)

# view estimates and bootstrapped confidence intervals
base.model_deg$coefficients

##    (Intercept)   log(terr.km) season.onlyWet
##    -0.10042653     0.05828047    -0.01607443

deg.bootstrap_clust$quantiles

##          coef lower.quant  upper.quant
## 1   intercept -0.20189162  0.012173431
## 2     terr.km  0.03677219  0.076613998
## 3 season.only -0.02477945 -0.006468056
```

```
base.model_str$coefficients
```

```
##    (Intercept)   log(terr.km) season.onlyWet
##     0.03609749     0.18205761    -0.13742654
```

```
str.bootstrap_clust$quantiles
```

```
##          coef lower.quant upper.quant
## 1   intercept  -0.3920014  0.47803181
## 2     terr.km   0.1064842  0.25635179
## 3 season.only  -0.1959147 -0.07722142
```

Correlations between precipitation and node-level metrics

NOTE: in manuscript, only calculated these correlations for UDOI cutoff = 0.
Start by loading and preparing the data.

```
### load precipitation data
precip <- get(load("../Data/avg precip_by season.Rdata"))
head(precip)
```

```
##   season_year total.avg.precip.mm
## 1    Dry_1996            417.3583
## 2    Dry_1997            822.5667
## 3    Dry_1998            407.0250
## 4    Dry_1999            373.9167
## 5    Dry_2000            166.6000
## 6    Dry_2001            416.0833
```

```
# extract descriptors of normalized degree per Season_Year
med.deg <- ddply(all.node.level, .(season), function(x) summary(x$norm.deg))
sd.deg <- ddply(all.node.level, .(season), function(x) sd(x$norm.deg))
# merge with precip
precip.deg <- left_join(precip, med.deg, by = c("season_year" = "season"))
precip.deg <- left_join(precip.deg, sd.deg, by = c("season_year" = "season"))
precip.deg$total.avg.precip.cm <- precip.deg$total.avg.precip.mm/10


# extract descriptors of strength per Season_Year
med.str <- ddply(all.node.level, .(season), function(x) summary(x$std.str))
sd.str <- ddply(all.node.level, .(season), function(x) sd(x$std.str))
# merge with precip
precip.str <- left_join(precip, med.str, by = c("season_year" = "season"))
precip.str <- left_join(precip.str, sd.str, by = c("season_year" = "season"))
precip.str$total.avg.precip.cm <- precip.str$total.avg.precip.mm/10


# add "season.only" to both degree and strength datasets
precip.deg$season.only <- substr(precip.deg$season_year, 1, 3)
precip.str$season.only <- substr(precip.str$season_year, 1, 3)
```

Next, perform Spearman correlations.

```r
## degree correlations ##
c.d.d <- cor.test(x =
precip.deg$total.avg.precip.cm[precip.deg$season.only=="Dry"], y =
precip.deg$Median[precip.deg$season.only=="Dry"], method = "spearman",
alternative = "two.sided")
c.d.w <- cor.test(x =
precip.deg$total.avg.precip.cm[precip.deg$season.only=="Wet"], y =
precip.deg$Median[precip.deg$season.only=="Wet"], method = "spearman",
alternative = "two.sided")

## strength correlations ##
c.s.d <- cor.test(x =
precip.str$total.avg.precip.cm[precip.str$season.only=="Dry"], y =
precip.str$Median[precip.str$season.only=="Dry"], method = "spearman",
alternative = "two.sided")
c.s.w <- cor.test(x =
precip.str$total.avg.precip.cm[precip.str$season.only=="Wet"], y =
precip.str$Median[precip.str$season.only=="Wet"], method = "spearman",
alternative = "two.sided")
```

View correlation results

```r
## degree correlations ##
c.d.d # dry season

##
##  Spearman's rank correlation rho
##
## data:  precip.deg$total.avg.precip.cm[precip.deg$season.only == "Dry"] and
precip.deg$Median[precip.deg$season.only == "Dry"]
## S = 200, p-value = 0.7966
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##        rho
## 0.09090909

c.d.w # wet season

##
##  Spearman's rank correlation rho
##
## data:  precip.deg$total.avg.precip.cm[precip.deg$season.only == "Wet"] and
precip.deg$Median[precip.deg$season.only == "Wet"]
## S = 310, p-value = 0.2139
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##        rho
## -0.4090909
```

```r
## strength correlations ##
c.s.d # dry season
```

```
##
##  Spearman's rank correlation rho
##
## data:  precip.str$total.avg.precip.cm[precip.str$season.only == "Dry"] and
precip.str$Median[precip.str$season.only == "Dry"]
## S = 140, p-value = 0.2732
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.3636364
```

```r
c.s.w # wet season
```

```
##
##  Spearman's rank correlation rho
##
## data:  precip.str$total.avg.precip.cm[precip.str$season.only == "Wet"] and
precip.str$Median[precip.str$season.only == "Wet"]
## S = 296, p-value = 0.2994
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##        rho
## -0.3454545
```

For reproducibility:

```r
#### view session info
sessionInfo()
```

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS  10.16
##
## Matrix products: default
## BLAS:
/Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK:
/Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] reshape2_1.4.4    data.table_1.12.8 igraph_1.2.5      dplyr_1.0.4
## [5] plyr_1.8.6
```

```
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.6       knitr_1.28       magrittr_2.0.1
tidyselect_1.1.0
##  [5] R6_2.5.0         rlang_0.4.10     stringr_1.4.0     tools_3.6.3
##  [9] xfun_0.24        DBI_1.1.1        htmltools_0.5.1.1 ellipsis_0.3.1
## [13] yaml_2.2.1       digest_0.6.27    tibble_3.0.6      lifecycle_1.0.0
## [17] crayon_1.4.1     purrr_0.3.4      vctrs_0.3.6       glue_1.4.2
## [21] evaluate_0.14    rmarkdown_2.9    stringi_1.5.3     compiler_3.6.3
## [25] pillar_1.4.7     generics_0.1.0   pkgconfig_2.0.3
```