

# Network Analysis: Homework

Jeff Ziegler

Due: August 23, 2018

## 1 Nigeria Data Processing

- a) Process the data: turn this event dataset into a matrix.
- b) Specifically, summarize the interactions across all time periods into an adjacency matrix where:
  1. "1" indicates that  $i$  and  $j$  had a conflictual interaction sometime during the temporal span of the original dataset and zero otherwise.
  2. Make sure all actors that existed at any point during the temporal span are included in the adjacency matrix.

```
1 rm(list=ls())
2 # set working directory to Git location
3 setwd('/Users/jeffziegler/Documents/Git/network2018_hw1/')
4 # load data
5 load("nigeria.rda")
6
7 #####
8 # (1) Nigeria Data Processing
9 #####
10
11 # create variable for row and column length of adjacency matrix to fill
12 rowLength <- length(unique(nigeria$sender))
13 colLength <- length(unique(nigeria$receiver))
14 # create adjacency matrix of sender and receiver, filled w/ zeroes
15 nigeriaAdjMat <- matrix(0, nrow=rowLength, ncol=colLength)
16 # adjust row and column names for sender and receiver
17 rownames(nigeriaAdjMat) <- unique(nigeria$sender)
18 colnames(nigeriaAdjMat) <- unique(nigeria$receiver)
19 # fill in adjacency matrix per year (i)
20 # start by sorting all unique years to iterate over
21 nigeriaAdjMatYearlyList <- lapply(sort(unique(nigeria$year)), function(i){
22   # find just those pairings for (1) a given year
23   currentYear <- nigeria[nigeria$year == i,]
24   # and (2) had a conflict (conflict == 1)
```

```

25 yearlyConflicts <- currentYear[currentYear$conlict == 1,]
26 # now that we know which pairings had conflicts
27 # fill in a "1" based on sender and receiver
28 for(i in 1:nrow(yearlyConflicts)){
29   nigeriaAdjMat[as.character(yearlyConflicts[i,]$sender),
30                 as.character(yearlyConflicts[i,]$receiver)] <- 1
31 }
32 # return the adjacency matrix, which will be placed in a list
33 return(nigeriaAdjMat)
34 })
35 # collapse all the matrices in list into one matrix
36 # since instructions are to "summarize the interactions
37 # across all time periods into a single matrix"
38 nigeriaAdjMatTotalMatrix <- Reduce('+', nigeriaAdjMatYearlyList)

```

## 2 Measurements & Community Detection

- a) Which actor is the most “influential” in the network? Justify your response and the measure you choose to estimate “influence.”
- b) Employ the blockmodel function from the sna package to explore potential group level structure in the data (see slides 61-63 from day 2 for details):
  - Run blockmodel with varying levels of k.
  - Save the node classifications from each run.
  - Now how do we choose k?
    - \* You will do so through an out-of-sample cross-validation exercise (at least 10 folds).
    - \* Report the AUC (ROC) and AUC (PR) statistics from each model.
- c) After having determined the k that gives the best out of sample performance, visualize your results as shown in slide 67 from the day 2 lecture

```

1 #####
2 # (2) Measurements and Community Detection
3 #####
4
5 # (a) We want to discern who the most "influential" actor in the network is
6 # We'll create two measures of "influence":
7 # (1) degree (number of connections)
8 # (2) eigenvector centrality (connections to high-scoring nodes
9 # contribute more to the score of the node)
10
11 # create graph object from igraph package
12 library(igraph)
13 igeriaGraph <- graph_from_adjacency_matrix(nigeriaAdjMatTotalMatrix,
14                                             mode='directed', weighted=TRUE, diag=FALSE)

```

```

15
16 # (1) degree
17 head(sort(degree(nigeriaGraph), decreasing=T))
18
19 # interestingly, the Police (Nigeria) and the Military (Nigeria)
20 # are two of the top 3 most engaged actors (Fulani Militia is #2)
21
22 # (2) eigenvector centrality
23 head(sort(eigen_centrality(nigeriaGraph, directed = TRUE)$vector,
24           decreasing=T))
25
26 # again, the police and military are not only more involved in conflicts
27 # but they engage w/ other highly conflicted actors
28
29 # (b) Instruction: Run blockmodel with varying levels of k
30 # Tasks/traits for blockmodel function (each run needs to):
31 # [1] Save the node classifications
32 # [2] Conduct out-of-sample CV (10 folds)
33 # [3] Report the AUC (ROC) and AUC (PR) statistics
34
35 # first, recreate matrices so that they are network objects
36 library(network)
37 nigeriaAdjMatNetworkList <- lapply(sort(unique(nigeria$year)), function(i){
38   # find just those pairings for (1) a given year
39   currentYear <- nigeria[nigeria$year == i,]
40   # and (2) had a conflict (conflict == 1)
41   yearlyConflicts <- currentYear[currentYear$conflict == 1,]
42   # now that we know which pairings had conflicts
43   # fill in a "1" based on sender and receiver
44   for(i in 1:nrow(yearlyConflicts)){
45     nigeriaAdjMat[as.character(yearlyConflicts[i,]$sender),
46                  as.character(yearlyConflicts[i,]$receiver)] <- 1
47   }
48   # return the adjacency matrix, which will be placed in a list
49   return(as.network.matrix(nigeriaAdjMat))
50 })
51
52 # create function that will do tasks 1-3
53 # then we can run CV function for varying levels of k
54 # Arguments:
55 # (remember function takes in igraph object)
56 # nFolds = number of folds (default = 10)
57 # nClusters = number of cluster (default = 2)
58 library(sna); library(caret); library(networkDynamic); library(btergm)
59 crossValidateFunc <- function(networkData, nFolds=NULL, nClusters=NULL) {
60   # set seed for reproducibility
61   set.seed(5)
62   # createFolds function from caret package
63   # argument gives a list of the indices in each fold
64   # from the groups that comprise all possible conflicts
65   # return training data

```

```

66 cvFolds <- createFolds(y = unique(nigeria$sender),
67                        k=nFolds, returnTrain = T)
68 # create empty vectors to fill w/ goodness-of-fit stats
69 # from TERGMS (AUC (ROC) and AUC (PR))
70 # ROC and PR curves can be used to compare different model specifications,
71 # also for within-sample goodness-of-fit
72 AUC_ROC <- NULL; AUC_PR <- NULL
73 # iterate over folds
74 for (i in 1:nFolds) {
75   # transform input list into network list
76   networkList <- networkDynamic(network.list=networkData)
77   # remove the necessary observations that are exempt from each fold
78   delete.vertices(networkList, (1:dim(nigeriaAdjMat)[1])[-cvFolds[[i]]])
79   # create clusters from structural equivalence
80   equivNetClusters <- equiv.clust(networkList)
81   # perform blockmodel
82   blockModel <- blockmodel(networkList, equivNetClusters, k=nClusters)
83   # take info that pertains to which block actors are placed in
84   groupMembership <- blockModel$block.membership[blockModel$order.vec]
85   # assign the block group values from the model back in the networkList
86   networkList%v%"member" <- groupMembership
87   # now run the out-of-sample prediction with TERGMS
88   outSampleTERGM <- btergm(as.network.networkDynamic(networkList) ~ edges +
89                          gwsnp(.5, fixed = TRUE) + nodecov("member"))
90   # now, simulate 100 networks from the model w/ rocpr
91   # to condense the performance into a single measure, the area under
92   # the curve (AUC) can be reported for both curves.
93   goodFitStats <- gof(outSampleTERGM, statistics = rocpr, nsim = 100)
94   # for each iteration/fold, remove and store statistics to existing list
95   AUC_ROC <- c(AUC_ROC, goodFitStats$`Tie prediction`$auc.roc)
96   AUC_PR <- c(AUC_PR, goodFitStats$`Tie prediction`$auc.pr)
97 }
98 # return the mean of each statistic pooled over the folds
99 return(list(avgAUC_ROC=mean(AUC_ROC), avgAUC_PR=mean(AUC_PR)))
100 }
101
102 # create empty vectors to fill with each run of clusters
103 # which performs CV for each run
104 AUC_ROCvec <- NULL; AUC_PRvec <- NULL
105 # for each number of clusters
106 for (k in 2:10) {
107   # run cross-validate function w/ 10 fold validation
108   cvNetwork <- crossValidateFunc(nigeriaAdjMatNetworkList, nFolds=10,
109                                nClusters=k)
109   # store AUC ROC and PR stats
110   AUC_ROCvec <- c(AUC_ROCvec, cvNetwork$avgAUC_ROC)
111   AUC_PRvec <- c(AUC_PRvec, cvNetwork$avgAUC_PR)
112 }
113 # show table of goodness-of-fit statistics
114 print(data.frame(k=2:10, AUC_ROC=AUC_ROCvec, AUC_PR=AUC_PRvec))

```

	k	AUC_ROC	AUC_PR
1	2	0.5321138	0.07721483
2	3	0.5245793	0.07547592
3	4	0.5420858	0.07672680
4	5	0.5394230	0.08071837
5	6	0.5424247	0.07655145
6	7	0.5628951	0.08848244
7	8	0.5366850	0.08067494
8	9	0.5243585	0.07983234
9	10	0.5489216	0.08615140

```

1 # (c) since we want these values to be higher
2 # we'll do 7 clusters
3 # re-create the network object
4 networkList <- networkDynamic(network.list=nigeriaAdjMatNetworkList)
5 # run the block model w/ 7 clusters
6 bestKblockModel <- blockmodel(networkList,
7                               equiv.clust(networkList),
8                               k=7)
9 # re-assign the groupings from the block model into the network object
10 bestGrouping <- bestKblockModel$block.membership[bestKblockModel$order.vec]
11 networkList%v%"member" <- bestGrouping
12 # now for the plotting of actors' interactions by group
13 # create colour paletter
14 library(RColorBrewer)
15 networkList %v% "col" <- brewer.pal(7, "Dark2")[networkList %v% "member"]
16 # generate plot (see figure 1 below)
17 pdf("figure1.pdf")
18 plot(networkList, label = network.vertex.names(networkList), label.cex=0.5,
19       mode="circle", vertex.cex=log(degree(networkList))+1,
20       label.col="black", vertex.col="col", vertex.border="col", edge.col="black",
21       "dev.off()

```

### 3 ERGMs

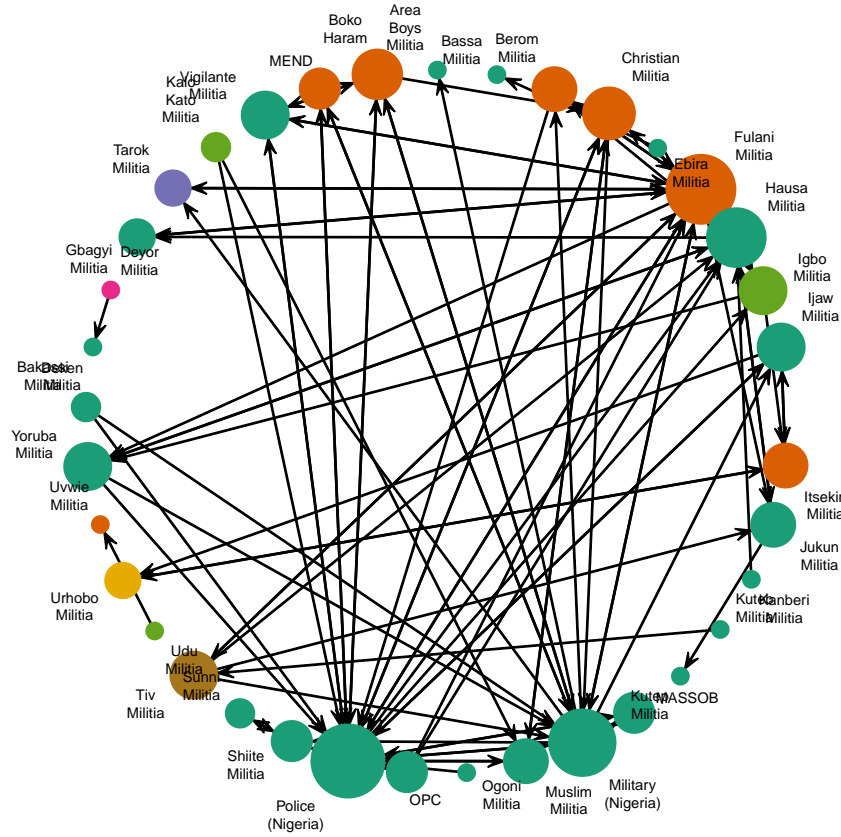
- Run a cross-sectional ERGM on the Nigerian conflict network, develop at least one or two network level hypotheses.
- Briefly discuss the results.
- Make sure to show that you checked for convergence.

```

1 #####
2 # (3) ERGMs
3 #####
4
5 # (a) Run a cross-sectional ERGM on the Nigerian conflict network

```

Figure 1: Network plot by groups generated from block model with 7 clusters.



```

6 # H1: First order (edges: Some actors (like the gov't) are
7 # engaged in conflict a lot, basically intercepts per actor)
8 # H2: Second order (mutual: Reciprocity should be high)
9 # wanted to test triangle in case allies exist in the network (police,
10 # military), transitivity might be high?)
11 # but so few existed MCMC wouldn't get out of one region
12 library(statnet)
13 ERGMmodel <- ergm(as.network.matrix(nigeriaAdjMatTotalMatrix) ~ edges +
mutual)

```

```

14
15 # (b) results
16 # reciprocity is high (when one actor is attacked, they retaliate)
17 summary(ERGMmodel)
18
19 # (c) check for convergence
20 mcmc.diagnostics(ERGMmodel)

```

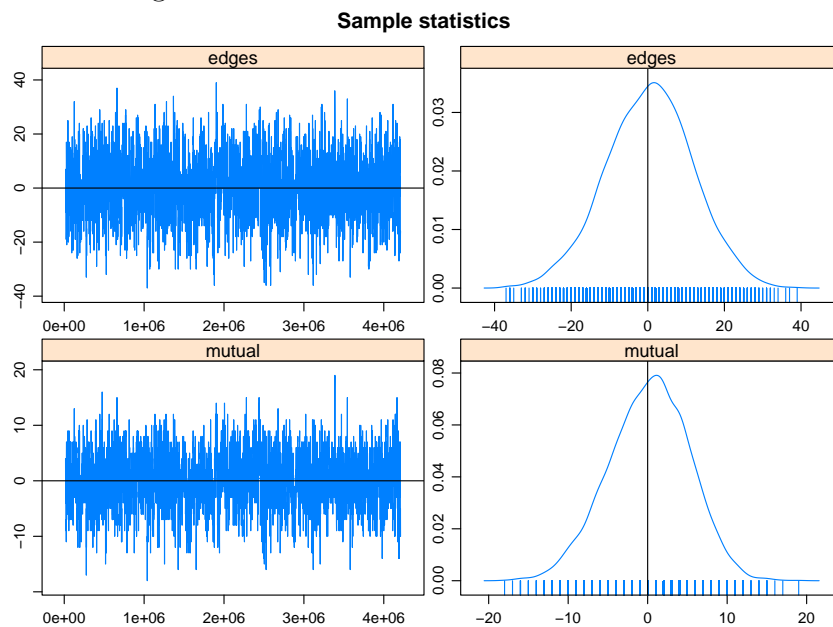
Monte Carlo MLE Results:

	Estimate	Std. Error	MCMC %	z value	Pr(> z )
edges	-3.4543	0.1653	0	-20.9	<1e-04 ***
mutual	3.8490	0.3738	0	10.3	<1e-04 ***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Figure 2: MCMC chain and estimate distribution.



## 4 Find your own data

- Locate data that relates to your field of interest.
- Transform the data, or a subset of it into a matrix, and plot (similar to step 1 in Section 1).
- Include descriptive features in your network graph (similar to step 2, but choose your own measurements).

- d) Run a model, it can be any network model from the course but justify your choices!
- e) Discuss the results in a brief write up. Present for 3-5 minutes to the class.

```

1                                     "Czech and Slovak
  Federal Republic (Czech Republic and Slovak Republic)" = "Czechslovakia",
2                                     "USSR (Russian
  Federation)" = "Russian Federation"))
3
4 disputesRaw$plaintiff <- gsub("\\ v. .*", "", disputesRaw$case); disputesRaw$
  plaintiff <- gsub(".* - ", "", disputesRaw$plaintiff)
5 # need to create a whole vector to fill for adjacency matrix
6 disputesRaw$violation <- 1
7
8 #####
9 # Begin network analysis
10 #####
11
12 # (a) Starting here after I cleaned the data
13 # (sorry I can't share the raw data with you)
14
15 # (b)
16 # create variable for row and column length of adjacency matrix to fill
17 # 251 unique plaintiffs
18 # 52 violating countries
19
20 # so we'll just look at violations that occur within treaties
21 # actors both are countries
22 # undirected
23 # get all possible countries
24 countries <- sort(unique(c(unique(disputesRaw$countryA),
25                               unique(disputesRaw$countryB))))
26 rowLength <- length(countries); colLength <- rowLength
27 # create adjacency matrix of sender and receiver, filled w/ NAs
28 BITadjMat <- matrix(0, nrow=rowLength, ncol=colLength)
29 # adjust row and column names for sender and receiver
30 rownames(BITadjMat) <- countries; colnames(BITadjMat) <- countries
31 # fill in adjacency matrix per year (i)
32 # start by sorting all unique years to iterate over
33 BITadjMatYearlyList <- lapply(sort(unique(disputesRaw$year)), function(i){
34   # find just those pairings for (1) a given year
35   currentYear <- disputesRaw[disputesRaw$year == i,]
36   # and (2) had a violation (violation == 1)
37   yearlyConflicts <- currentYear[currentYear$violation == 1,]
38   # now that we know which pairings had conflicts
39   # fill in a "1" based on sender and receiver
40   for(i in 1:nrow(yearlyConflicts)){
41     BITadjMat[as.character(yearlyConflicts[i,]$countryB),
42               as.character(yearlyConflicts[i,]$countryA)] <- 1
43   }
44   # return the adjacency matrix, which will be placed in a list

```



```

45   return(as.network.matrix(BITadjMat, directed=T))
46 })
47 # collapse and sum across years
48 #BITadjMatTotalMatrix <- Reduce('+', BITadjMatYearlyList)
49
50 # (c-e)
51
52 # show biggest "suers" (origin country of plaintiff)
53 head(sort(rowSums(BITadjMatTotalMatrix), decreasing = T))
54 # show biggest violators (countries that pass potentiall expropriating
    policies)
55 head(sort(colSums(BITadjMatTotalMatrix), decreasing = T))
56
57 # create network object using igraph
58 yGraph = igraph::graph.adjacency(BITadjMatTotalMatrix,
59                                 mode='directed',
60                                 weighted=TRUE,
61                                 diag=FALSE
62 )
63
64 # very little violations in general
65 # check proportion of present edges from all possible edges in the network.
66 ecount(yGraph)/(vcount(yGraph)*(vcount(yGraph)-1)) #for a directed network
67
68 # doesn't seem like passing expropriating policies
69 # leads to other countries to sue, and then pass expropriating policies
70 reciprocity(yGraph)
71
72 # add democratic information
73 polity <- read.csv("/Users/jeffziegler/Dropbox/Research/data/polity.csv", sep=
    "\t", stringsAsFactors = F)
74 countries <- data.frame("country"=countries)
75 polityScores <- merge(countries, polity[c("country", "polity2")], all.x=T)
76 polityAvg <- aggregate(polityScores[,2], list(polityScores$country), mean, na.
    rm=TRUE)
77 names(polityAvg) <- c("country", "polityAvg")
78 polityAvg$demoGroup <- ifelse(polityAvg$polityAvg >=6, "blue",
79                             ifelse(6 < polityAvg$polityAvg | polityAvg$
    polityAvg >=-5, "grey", "red"))
80 # add categories to iGraph object
81 V(yGraph)$demoGroup <- polityAvg$demoGroup
82
83 # convert to network graph object using
84 yNet = intergraph::asNetwork(yGraph)
85 yNet %v% "col" <- brewer.pal(3, "Dark2")[yNet %v% "demoGroup"]
86
87 # run latent distance model
88 y.var<-sd(c(BITadjMatTotalMatrix), na.rm=TRUE)
89 if(!file.exists('bitLDM.rda')){
90   lsEucl = ergmm(yNet ~ euclidean(d=2),
91                 family='normal',

```

```

92         fam.par=list(
93             prior.var=4*sd(c(BITadjMatTotalMatrix), na.rm=TRUE),
94             prior.var.df=2 # certainty of the prior, higher more
               certain
95         ) )
96     save(lsEucl, file='bitLDM.rda') }
97 load('bitLDM.rda')
98
99 # evaluate convergence
100 mcmc.diagnostics( lsEucl )
101 # could probably let the chain run a little longer
102 # visualize results
103 par(mfrow=c(1,1))
104 # jitter the points
105 zPosJitter = zPos+matrix(rnorm(length(zPos),0,.03),ncol=2)
106 pdf("/Users/jeffziegler/Documents/Git/network2018_hw1/figure3.pdf")
107 plot(yGraph,
108     layout=zPosJitter,
109     vertex.color=V(yGraph)$demoGroup,
110     vertex.label.color='black',
111     vertex.size=V(yGraph)$size,
112     vertex.label.cex=.75,
113     edge.color='grey',
114     edge.width=E(yGraph)$weight,
115     edge.arrow.size=.2,
116     asp=FALSE
117 )
118 dev.off()

```