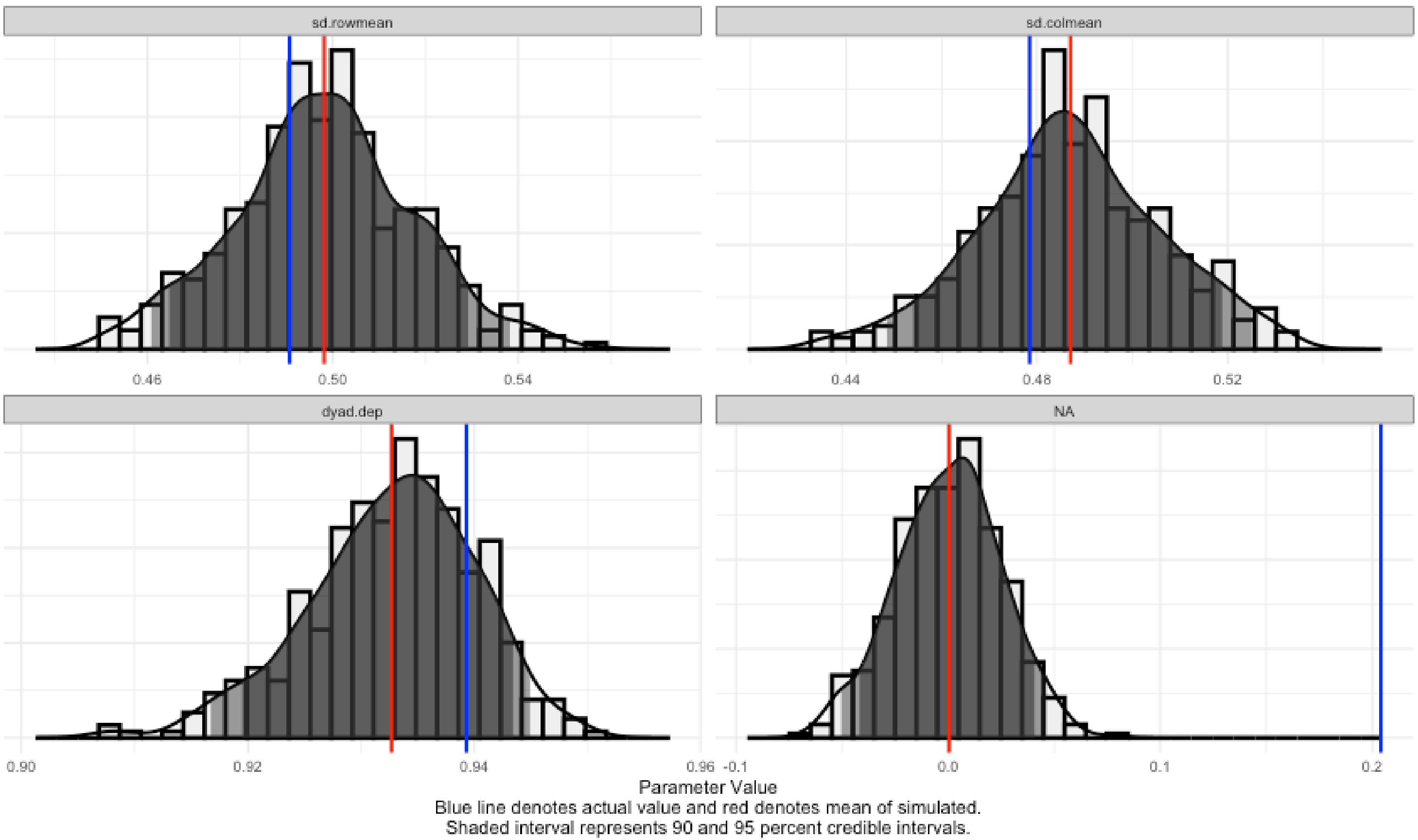
Advanced Network Analysis

Inferential Modeling with Blocks

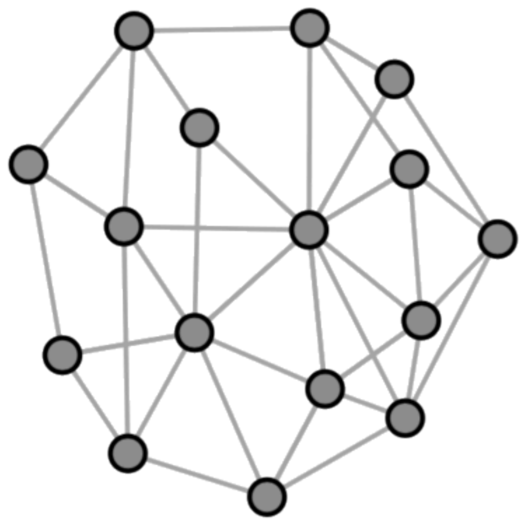
Shahryar Minhas [s7minhas.com]

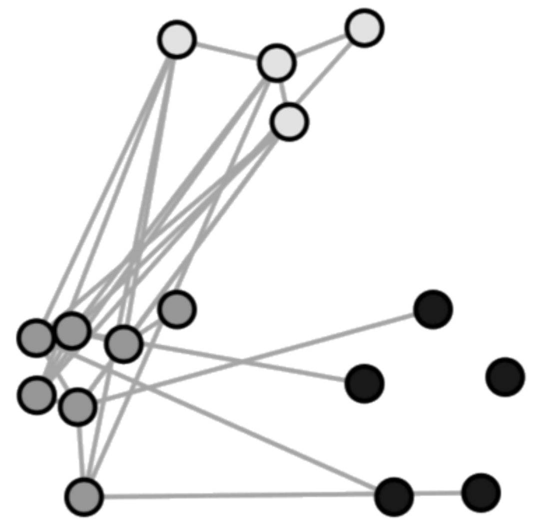
Capturing network features?

```
gofPlot(fitSRM$GOF, symmetric=FALSE)
```



What are we missing?





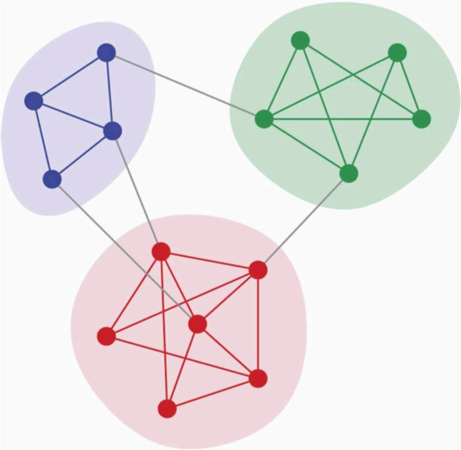
- **Homophily**: "birds of a feather flock together"
- **Stochastic equivalence**: nothing as pithy to say here, but this model focuses on identifying actors with similar roles

Basically, our inferential model is missing something, one way to think about it is that we need to find an expression for γ:

$$y_{ij} \approx \beta^T X_{ij} + a_i + b_j + \gamma(u_i, v_j)$$

One way of getting at this ... community detection

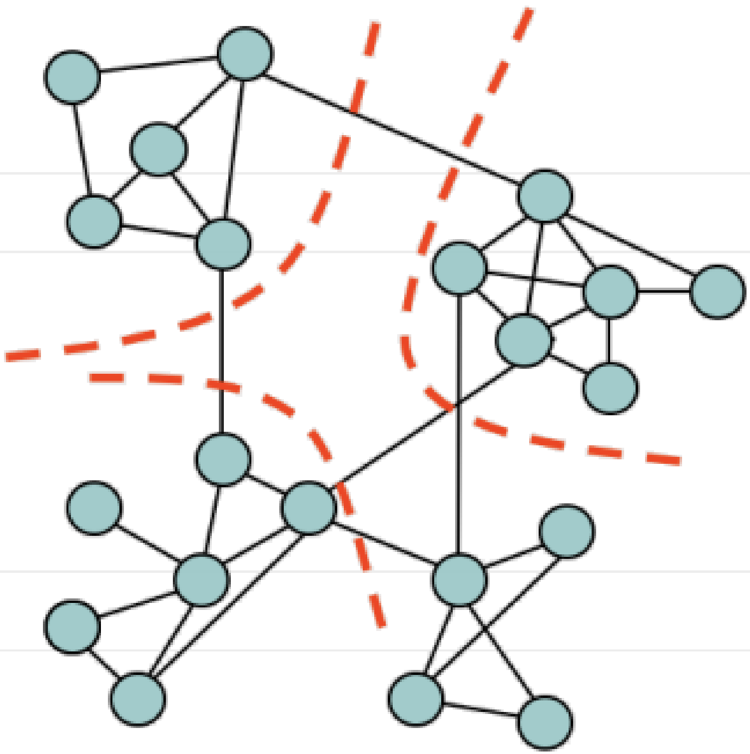
- What makes a community?
  - Cliques: Everyone in the group has connections to one another
  - Compact: The path between nodes in the community are small
  - Differentiation: High frequency of ties among people in the community versus with non-members
- Community structure
  - One way of thinking about this is that vertices often cluster into tight-knit groups with a high density of within-group edges and a lower density of between-group edges



Source: Newman (2012)

How does it work?

Most community detection techniques are graph partitioning problems:



Why would we detect communities?

- Exploratory: What structures my network
- Confirmatory: Do communities map onto some exogenous variable
- Predictive: Does community membership predict something else

Methods for community detection

There's a lot ... the list below is not comprehensive

- Edge-betweenness (Girvan and Newman 2001)
  - In `igraph` the relevant function is `cluster_edge_betweenness`
- Leading Eigenvector (Newman 2006)
  - `igraph::cluster_leading_eigen`
- Fast-Greedy (Clauset et al. 2004)
  - `igraph::cluster_fast_greedy`
- Multi-Level (Blondel et al. 2008)
  - `igraph::cluster_louvain`
- Walktrap (Pons and Latapy 2005)
  - `igraph::cluster_walktrap`
- Label propagation (Raghavan et al. 2007)
  - `igraph::cluster_label_prop`
- InfoMAP (Rosvall et al. 2009)
  - `igraph::cluster_infomap`

What "unites" these approaches?

- Modularity is a score for the fraction of the edges that fall within the given group minus the expected such fraction if edges were distributed at random (`igraph::modularity`)
- Formally, given a particular division of communities, the modularity of the division is:

$$Q = \frac{1}{2m} \sum_{ij} [Y_{ij} - \frac{k_i k_j}{2m}] I(c_i = c_j)$$

- Y_{ij} is the element of the A adjacency matrix in row i and column j
- k_i is the degree of i, k_j is the degree of j
- m is the number of edges in the network
- c is the group index; c_i is the type (or component) of i, c_j that of j

Basically ...

- Modularity provides us with an assessment of how good the communities we identified are at grouping together actors
- Ideally, we want communities where actors with dense connections are grouped together and actors with sparse are grouped separately

So those algorithms ...

Lets discuss them in the context of Gade et al. 2019:

```
library(igraph)
load('gadeData.rda')
dim(Y)
```

```
## [1] 31 31
```

```
Y[1:5,1:5]
```

```
##       101st 13th AARB AF ANF
## 101st     0    1    0  0   0
## 13th      1    0    0  0   0
## AARB      0    0    0  1   1
## AF        0    0    1  0   1
## ANF       0    0    1  1   0
```

Lets find communities, we'll start with edge-betweenness

- Edge-betweennness (aka Girvan-Newman method) starts by calculating betweenness at the edge level
  - Recall betweenness at the node level focuses on how often a node acts as a bridge along the shortest path between two other nodes
- The idea:
  - Find edges that serve as bridges between communities
  - Remove those edges
  - Group remaining actors into communities
  - Repeat until some maximal level of modularity is reached

Edge-betweenness cont'd

- So how do we find those edges that serve as bridges?
- We can calculate the edge betweenness score for a given edge by: $\sum \frac{\sigma_{ij}(e)}{\sigma_{ij}}$, where
  - σ_{ij} is the total number of shortest paths from node i to j and
  - $\sigma_{ij}(e)$ is the number of those paths that pass through edge e

So how do we calculate?

`igraph::cluster_edge_betweenness`

```
# convert Y to graph object
g = graph_from_adjacency_matrix(Y,
    mode='undirected',
    weighted=NULL,
    diag=FALSE
    )
ebComm = cluster_edge_betweenness(g)
```

Who got assigned where

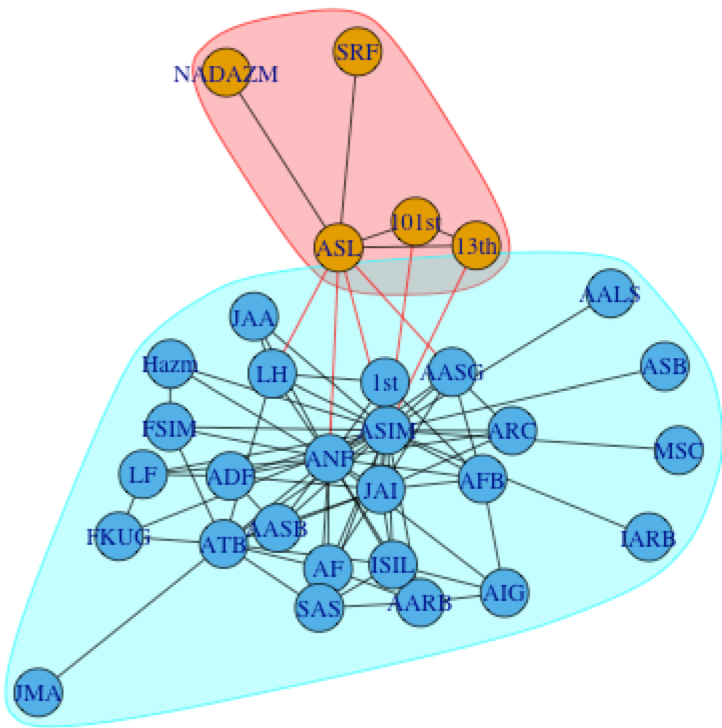
We can use the `membership` function to extract actor assignments

```
membership(ebComm)
```

```
##  101st   13th   AARB     AF    ANF   ASIM   ISIL   AASB    ADF   AASG    A
##      1      1      2      2      2      2      2      2      2      2
##     LF    ATB    JAI    AFB    1st    AIG   FSIM   Hazm    JAA     LH    S
##      2      2      2      2      2      2      2      2      2      2
##   AALS    ASB   FKUG    MSC    ASL NADAZM    SRF    JMA   IARB
##      2      2      2      2      1      1      1      2      2
```

We can also plot

```
plot(ebComm, g)
```



Now, how did this algorithm do?

Lets calculate the modularity score:

```
ebCommScore = modularity(ebComm, g)
ebCommScore
```

```
## [1] 0.09897242
```

How did we do?

- Modularity ranges from $< -1, 1 >$, where 1 indicates stronger community structure
- Values close to 1 indicate strong community structure
- Values close to 0 indicate the community division is not better than random

Lets try another: Leading Eigenvector

- Basically, a principal components analysis for networks
- In each step, the network is split into two parts such that the separation yields a significant increase in modularity
- At each stage, the split is determined by evaluating the results of the principal components analysis
- To run this we can use: `igraph::cluster_leading_eigen`

Newman's Leading Eigenvector

Does it do better?

```
leComm = cluster_leading_eigen(g)
leCommScore = modularity(leComm, g)
leCommScore
```

```
## [1] 0.1980124
```

Practical guidance

- How should we choose between community detection approaches (there are some problems with modularity)?
  - How do you choose topics in a topic model?

How is this used in modeling?

Latent class model/blockmodels (Holland et al. 1983; Nowicki & Snijders 2001; Rohe et al. 2011; Airoldi et al. 2013)

Each node i is a member of an (unknown) latent class:

$$\mathbf{u}_i \in \{1, \ldots, K\},\ i \in \{1, \ldots, n\}$$

The probability of a tie between i and j is:

$$Pr(Y_{ij} = 1 | \mathbf{u}_i, \mathbf{u}_j) = \theta_{\mathbf{u}_i \mathbf{u}_j}$$

- Nodes in the network may have a small or high probability of ties: θ_{kk} may be small or large
- Nodes in the same class are stochastically equivalent

Software packages:

- CRAN: `blockmodels` (Leger 2015)
- CRAN: `sbm` (Chiquet et al 2021)
- CRAN: `mixedMem` (Wang & Erosheva 2015)
- CRAN: `dynsbm` (Matias & Miele 2018)
- CRAN: `NetMix` (Olivella et al 2021)

Block Models: Core Concepts

- Partition network nodes into K blocks (communities or roles)
- Fundamental assumption: Nodes in same block have similar connection patterns
- Edge probability ($P(Y_{ij} = 1)$) depends solely on block memberships of nodes
- Key methods:
    - Mixture models: Network as mixture of connectivity patterns
    - Probabilistic clustering: Assign nodes to blocks based on connectivity
    - Maximum likelihood estimation: Optimize block assignments and probabilities

Block Model Setup

Say that we have a network with N nodes, and we want to position each of the N nodes into K blocks:

- Block membership: $z_i \in 1, \ldots, K$ for each node i
- Block probability matrix: θ (dimensions $K \times K$)
  - $\theta_a b$ = probability of edge between blocks a and b
- Edge probability: $P(Y_i j | z_i, z_j) = \theta_{z_i, z_j}$
- Likelihood function: $L(z, \theta | \text{network}) = \prod_{i,j} \theta_{z_i, z_j}^{Y_{ij}} (1 - \theta_{z_i, z_j})^{(1 - Y_{ij})}$ Where $Y_i j$ is 1 if edge exists between i and j, 0 otherwise

Likelihood Function in Stochastic Block Models

The likelihood function represents the probability of observing the network given block assignments (z) and edge probabilities (θ):

$$L(z, \theta | \text{network}) = \prod_{i,j} \theta_{z_i,z_j}^{Y_{ij}} (1 - \theta_{z_i,z_j})^{(1-Y_{ij})}$$

- Product over all node pairs i and j
- θ_{z_i,z_j}: edge probability between blocks z_i and z_j
- Y_{ij}: 1 if edge exists between i and j, 0 otherwise

Interpretation:

- For each edge: contribute θ_{z_i,z_j}
- For each non-edge: contribute $(1 - \theta_{z_i,z_j})$
- Assumes edge independence given block structure

Goal: Find z and θ that maximize this likelihood

Interpreting the Likelihood Function in Block Models

$L(z, \theta | \text{network}) = \prod_{i,j} \theta_{z_i,z_j}^{Y_{ij}} (1 - \theta_{z_i,z_j})^{(1-Y_{ij})}$

Simplified interpretation:

1. The likelihood function measures how well our model fits the observed network
2. It asks: Given our current guess about:
   - which nodes belong to which blocks (z)
   - how likely connections are between blocks (θ)

   How probable is the network we actually observe?
3. Higher likelihood = better fit:
   - Nodes in same block connect as expected
   - Nodes in different blocks connect as expected
4. We aim to find block assignments (z) and connection probabilities (θ) that make our observed network as likely as possible
5. This helps us uncover the underlying block structure of the network

Estimation Process: High Level

1. Start with a guess:
    - Randomly put nodes into groups
    - Make an initial guess about how likely nodes in different groups are to connect
2. Improve node groupings:
    - For each node, figure out which group it fits best in
    - We do this by looking at its connections and the current group setup
    - Move nodes to groups where they fit better
3. Update connection probabilities:
    - Look at how nodes in different groups are actually connected
    - Update our estimates of how likely nodes in different groups are to connect
4. Repeat steps 2 and 3:
    - Keep improving group assignments and connection probabilities
    - Stop when things aren't changing much anymore, or after a set number of tries

Estimation Process: Detail

1. Initialization:
    - Randomly assign nodes to K blocks
    - Initialize θ matrix (e.g., with random values or based on overall edge density)
2. Expectation step:
    - For each node i:
        - Calculate $P(z_i = k|\text{network}, \theta, z_{-i})$ for all k
        - z_{-i} denotes block assignments of all nodes except i
    - Use Bayes' rule: $P(z_i = k|\text{network}, \theta, z_{-i}) \propto P(\text{network}|z_i = k, \theta, z_{-i}) \cdot P(z_i = k)$
3. Maximization step:
    - Update θ based on current block assignments: $\theta_{ab} = \frac{\text{number of edges between blocks } a \text{ and } b}{\text{possible edges between } a \text{ and } b}$
4. Iterate steps 2-3 until convergence
    - Convergence when change in log-likelihood falls below threshold
    - Or after predetermined number of iterations

Block Model Estimation Procedures

Goal: Find optimal block assignments (\$z\$) and block-to-block edge probabilities (\$\theta\$)

1. Maximum Likelihood Estimation (MLE):

   - Objective: Maximize the likelihood function
   - $L(z, \theta | A) = \prod_{i<j} \theta_{z_i,z_j}^{A_{ij}} (1 - \theta_{z_i,z_j})^{(1-A_{ij})}$
   - A is the adjacency matrix, $A_{ij} = 1$ if edge exists, 0 otherwise
   - And as always with MLE we typically use the log-likelihood: $\log L(z, \theta | A) = \sum_{i<j} [A_{ij} \log \theta_{z_i,z_j} + (1 - A_{ij}) \log(1 - \theta_{z_i,z_j})]$

2. Posterior Maximization (Bayesian approach):

   - Objective: Maximize the posterior probability
   - $P(z, \theta | A) \propto L(z, \theta | A) \cdot P(z) \cdot P(\theta)$
   - $P(z)$ is prior on block assignments (e.g., uniform or Dirichlet)
   - $P(\theta)$ is prior on edge probabilities (e.g., Beta distribution)

Block Model Estimation Procedures Cont'd

1. Variational Inference:
    - Objective: Minimize Kullback-Leibler divergence
    - $KL(Q||P) = E_Q[\log Q(z, \theta)] - E_Q[\log P(z, \theta, A)]$
    - $Q(z, \theta)$ is variational approximation to true posterior $P(z, \theta | A)$
2. Spectral Clustering Approach:
    - Objective: Maximize modularity or minimize graph cut
    - For k blocks: $\max_z \operatorname{tr}(Z^T B Z)$, where B is modularity matrix
    - Z is $n \times k$ indicator matrix of block assignments

Block Model Estimation Challenges

Challenges:

- Optimization landscape is non-convex
- Multiple local optima may exist
- Solutions may depend on initialization

Approaches to address challenges:

- Multiple random initializations
- Deterministic annealing
- Spectral initialization before likelihood optimization

Model Selection and Evaluation

Choosing number of blocks K:

- Bayesian Information Criterion (BIC): BIC = -2 $log(L)$ + m log(n) where L is likelihood, m is number of parameters, n is sample size
- Integrated Completed Likelihood (ICL)
- Cross-validation on held-out edges

Evaluating model fit:

- Posterior predictive checks
- Compare observed network statistics to those from model-generated networks

Interpreting results:

- Examine block sizes and inter-block connection probabilities
- Visualize network with nodes colored by block assignments

LCM for community detection

Newman (2006): Nouns

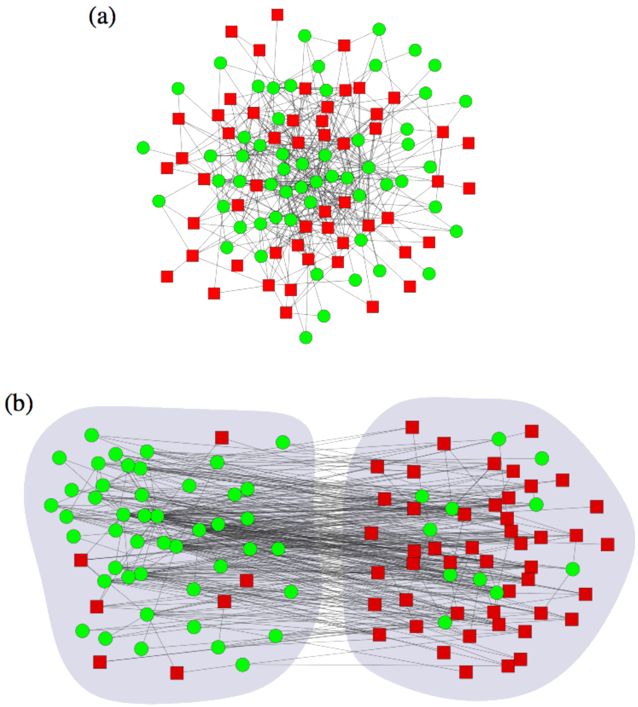


FIG. 7. (Color online) (a) The network of commonly occurring English adjectives (circles) and nouns (squares) described in the text. (b) The same network redrawn with the nodes grouped so as to minimize the modularity of the grouping. The network is now revealed to be approximately bipartite, with one group consisting almost entirely of adjectives and the other of nouns.

White & Murphy (2016): Mixed membership stochastic block model

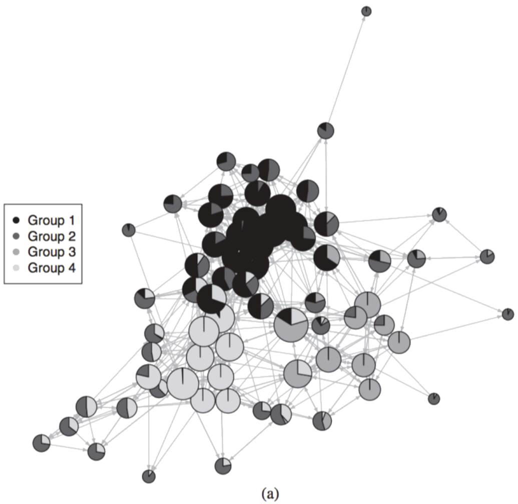


Fig. 7. Visualization of the 4 group MMESBM fitted to the Lazega Lawyers friendship

Apply LCM to trade

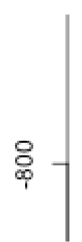
- Below we show how to implement a basic stochastic blockmodel using the `blockmodels` package (see vignette and CRAN function list for more details):

```
library(blockmodels)

# make sure there are no NAs
diag(Y) = 0

# gaussian stochastic blockmodel
set.seed(6886)
sbm = BM_gaussian('SBM', Y)

# to estimate model run
sbm$estimate()
```



Attributes of estimated object

The output from the `blockmodels` package is different than what most other packages do:

```
sbm$show()
```

```
## blockmodels object
##     model: gaussian
##     membership: SBM
##     network: 30 x 30 scalar network
##     maximum of ICL: 5 groups
##     Most usefull fields and methods:
##         The following fields are indexed by the number of groups:
##             $ICL : vector of ICL
##             $PL : vector of pseudo log liklihood
##             $memberships : list of memberships founds by estimation
##                            each membership is represented object
##             $model_parameters : models parameters founds by estimation
##         Estimation methods:
##             $estimate(reinitalization_effort=1) : to run again estimation
##                                                   higher reinitalization e
##         Plotting methods:
##             $plot_obs_pred(Q) : to plot the obeserved and predicted networ
##             $plot_parameters(Q) : to plot the model_parameters for Q group
```

Extracting membership vector

```
mem = sbm$memberships[[5]]$Z
memCat = apply(mem, 1, function(x){which(max(x)==x)})
memCat
```

```
##  [1] 3 5 4 3 5 2 2 3 3 4 5 5 3 4 1 5 4 3 5 3 2 3 5 4 5 2 5 3 4 1
```

Plotting results from stochastic blockmodel

```
# create graph object
diag(Y) = NA
yGraph = igraph::graph.adjacency(Y,
  mode='directed',
  weighted=TRUE,
  diag=FALSE
  )

# add node attributes
V(yGraph)$size = rescale(
  apply(Y, 2, sum, na.rm=TRUE), c(10, 16) )

# Colors
library(RColorBrewer)
cols = brewer.pal(5, 'Set1')
nodeColors = cols[memCat]
```

How to set layout?

- We want to set the layout so it is at least somewhat reflective of the community assignments
- Here's a helper function that tries to do this

```
commPlotHelper = function(m, graph){
  el <- igraph::as_edgelist(graph, names = FALSE)
  m1 <- m[el[, 1]]
  m2 <- m[el[, 2]]
  res <- m1 != m2
  if (!is.null(names(m1))) {
      names(res) <- paste(names(m1), names(m2), sep = "|")
  }
  return(res) }

weights = commPlotHelper(memCat, yGraph)
set.seed(6886)
commLayout = layout_with_fr(yGraph, weights=weights)
```

Now put the pieces together

```
plot(yGraph,
  layout=commLayout,
  vertex.color=cols[memCat],
  vertex.label.color='white',
  vertex.size=V(yGraph)$size,
  vertex.label.cex =.75,
  edge.color='grey20',
  edge.width=E(yGraph)$weight,
  edge.arrow.size=.2,
  asp=FALSE
  )
```

Now put the pieces together