Statistical modeling of visual cortical neurons

Cristian Bargiacchi, Christian Mancini

maggio 2024

Preliminary steps

```
library(igraph)
library(ergm)
library(intergraph)
library(ggplot2)
```

We first need to load our data using `igraph`. Data can be found here

```
neurons_g <- read_graph("Data/mouse_visual.cortex_2.graphml","graphml")
Y = as_adjacency_matrix(neurons_g, sparse = F)
diag(Y) = NA
```

For starting the modeling we first have to convert an igraph object to a network one.

The conversion retains the order of the nodes but we also have to pass the attributes.

```
neurons = network(Y, directed = T)
neurons %v% "type1" = vertex_attr(neurons_g,"type1",V(neurons_g))
neurons %v% "type2" = vertex_attr(neurons_g, "type2",V(neurons_g))
```

Now we are good to go!

Homogeneous Simple Random Graph

Let's start with the simplest model.

Assumptions:

- The probability of forming a tie is the **same** for every pair.

```
srg_homo = ergm(neurons ~ edges)
```

```
summary(srg_homo)
```

```
Call:
ergm(formula = neurons ~ edges)

Maximum Likelihood Results:

      Estimate Std. Error MCMC % z value Pr(>|z|)
edges -5.16921    0.06854      0  -75.42   <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 52444  on 37830  degrees of freedom
 Residual Deviance:  2642  on 37829  degrees of freedom

AIC: 2644  BIC: 2652  (Smaller is better. MC Std. Err. = 0)
```

This model corresponds to a logistic regression, so we can interpret the result as odds:

> The odds of observing a relation between two randomly selected nodes is about 99.43% lower than that of not observing it.

Non-Homogeneous Simple Random Graph

Assumptions:

- The same probability of forming a tie is relaxed.
- Takes in consideration sender and receiver effect

> Since some node do not have in or out degree, those parameter will be set to `-inf`, so the model can't be used, but it can be estimated.

```
srg_no_homo = ergm(neurons ~ edges + sender + receiver,
                   control = control)
```

Dyad independence model

Assumptions:

- Dyads are independents and follows a *Multinomial* distribution
- We take in consideration the reciprocity parameter γ_{ij} (*mutual*)

Classic p1 model

Assumptions:

- the reciprocity parameter is $\gamma = \gamma_{ij}$
- μ_{ij} depends additively from on the sender and receiver effect of node i and j involved.

```
p1_classic = ergm(neurons ~ edges + sender + receiver + mutual,
                  control = control.ergm(seed = 1))
```

Sender and reciver independency assumption

We now construct 3 p1 model to check for *reciprocity* with the following assumptions:

1. Sender effect independent
2. Receiver effect independent
3. Sender and receiver effect independent (equals to non-homogeneous SRG)

   For the same reasons explained in the non-homogeneous SRG, these models can't be estimated. The coefficients are set to `-inf`.

```
p1_sender_ind = ergm(neurons ~ edges + receiver + mutual,
                     control = control.ergm(seed = 1))
```

```
p1_receiver_ind = ergm(neurons ~ edges + sender + mutual,
                       control = control.ergm(seed = 1))
```

```
p1_mutual_only = ergm(neurons ~ edges + mutual,
                      control = control.ergm(seed = 1))
```

At this point we can just relay on the SRG. The next step is to include nodal attributes and exploit the background knowledge that we have of this network. We want to exploit the starts that are cleary visible in the network. Let's do one step at a time.

Nodal attributes

In this part of the analysis we include nodal attributes to explore *homophily* and *main* effects.

- **Main effect**: Nodes of a specific type have more chance to form ties
- **Homophily effect** Nodes of a specific type have more chance to form ties between nodes of the same type

  As saw in the descriptive analysis we expect to observe dissortative mixing.

Since we only have categorical attributes we will use:

- `nodefactor()` to include main effect
- `nodematch()` to include homophily effect

Let's explore "*type1*" nodal attribute.

```
main_homo_type_one = ergm(neurons ~ edges + nodefactor("type1") + nodematch("type1"),
            control = control.ergm(seed = 1))
```

```r
summary(main_homo_type_one)
```

```
Call:
ergm(formula = neurons ~ edges + nodefactor("type1") + nodematch("type1"),
    control = control.ergm(seed = 1))

Maximum Likelihood Results:

                                                Estimate Std. Error MCMC % z value Pr(>|z|)
edges                                           -7.24825    0.37136      0 -19.518   <1e-04 ***
nodefactor.type1.Characterized pyramidal neuron  4.12940    0.33609      0  12.287   <1e-04 ***
nodefactor.type1.Dendritic fragment             -0.01284    0.16781      0  -0.077    0.939
nodematch.type1                                 -4.10482    0.51257      0  -8.008   <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 52444  on 37830  degrees of freedom
 Residual Deviance:  1850  on 37826  degrees of freedom

AIC: 1858  BIC: 1892  (Smaller is better. MC Std. Err. = 0)
```

As we expect, the *Dissortative mixing* is captured. The probability of forming ties with nodes of the same type is 98.35% lower, with respect to probability of forming ties with different one. Moreover, "*Characterized pyramidal neuron*" have more chances to form ties (which is true because they start the synapses). More precisely, the odds is 62.14 times higher respect to a "*Cell body in EM volume*".

Now we use *"type2"* attribute, but only as main effect, and we remove `mutual` since it is estimated as `-inf`.

We exclude homophily, since all the coefficents will result non significants.

```r
main_type_two = ergm(neurons ~ edges + nodefactor("type2"),
            control = control.ergm(seed = 1))
```

```r
summary(main_type_two)
```

```
Call:
ergm(formula = neurons ~ edges + nodefactor("type2"), control = control.ergm(seed = 1))

Maximum Likelihood Results:

                                               Estimate Std. Error MCMC % z value Pr(>|z|)
edges                                           -0.9258     0.1354      0  -6.836   <1e-04 ***
nodefactor.type2.Postsynaptic excitatory target -2.9271     0.1270      0 -23.047   <1e-04 ***
nodefactor.type2.Postsynaptic inhibitory target -2.6742     0.1349      0 -19.824   <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 52444  on 37830  degrees of freedom
 Residual Deviance:  2016  on 37827  degrees of freedom

AIC: 2022  BIC: 2048  (Smaller is better. MC Std. Err. = 0)
```

We basically got the same conclusions. Since the reference is "*NA*" which can only be "*Characterized pyramidal neuron*", the odds of form a tie are smaller if a node is "*excitatory*" or "*inhibitory*".

Let's now put all together:

```
main_homo = ergm(neurons ~ edges + nodefactor("type1") + nodefactor("type2")
                 + nodematch("type1"),
             control = control.ergm(seed = 1))
```

```
summary(main_homo)
```

```
Call:
ergm(formula = neurons ~ edges + nodefactor("type1") + nodefactor("type2") +
    nodematch("type1"), control = control.ergm(seed = 1))

Maximum Likelihood Results:

                                                Estimate Std. Error MCMC % z value Pr(>|z|)
edges                                           -2.72926    0.91004      0  -2.999 0.002708 **
nodefactor.type1.Characterized pyramidal neuron  1.87771    0.53008      0   3.542 0.000397 ***
nodefactor.type1.Dendritic fragment             -0.06639    0.16889      0  -0.393 0.694258
nodefactor.type2.Postsynaptic excitatory target -2.37042    0.42591      0  -5.566  < 1e-04 ***
nodefactor.type2.Postsynaptic inhibitory target -2.10745    0.42811      0  -4.923  < 1e-04 ***
nodematch.type1                                 -4.12024    0.51293      0  -8.033  < 1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 52444  on 37830  degrees of freedom
 Residual Deviance:  1827  on 37824  degrees of freedom

AIC: 1839  BIC: 1890  (Smaller is better. MC Std. Err. = 0)
```

We can see that the main effect of being a *Dendritic fragment* is not significant, so the tendency to form more ties than by chance depends only of being a *Characterized pyramidal neuron* or not. This is a good thing, since it reflect our descriptive analysis.

We can now compare BIC for these models including nodal attributes.

Model	BIC
main_Homo_Type_One	1891.98
main_Type_Two	2047.53
main_Homo	1890.25

According to `BIC` the full model is better.

Nodal attributes (Binary)

Let's repeat the above analysis but using new attributes which are the binarization of the real ones.

As reference category we choose "*Characterized pyramidal neuron*" (for *type1*) and "*Postsynaptic excitatory target*" (for *type2*).

Generate the new attributes:

```
type1.new = rep(0, 195)
type1.new[vertex_attr(neurons_g,"type1") == "Characterized pyramidal neuron"] = 1
type1.new
neurons %v% "type1.new" = type1.new

type2.new = rep(0, 195)
type2.new[vertex_attr(neurons_g,"type2") == "Postsynaptic excitatory target"] = 1
type2.new
neurons %v% "type2.new" = type2.new
```

Estimate all the models again:

```
main_homo_type_one_binary = ergm(neurons ~ edges + nodefactor("type1.new")
                                 + nodematch("type1.new"),
            control = control.ergm(seed = 1))

main_type_two_binary = ergm(neurons ~ edges + nodefactor("type2.new"),
            control = control.ergm(seed = 1))

main_homo_binary = ergm(neurons ~ edges + nodefactor("type1.new")
                        + nodefactor("type2.new") + nodematch("type1.new"),
            control = control.ergm(seed = 1))
```

Model	BIC
main_homo_type_one_binary	1853.40
main_type_two_binary	2502.18
main_homo_binary	1858.36

According to BIC the new model with just "*type1.new*" is the best for now with a score of 1853.40.

Let's explore the summary of the model

```
summary(main_homo_type_one_binary)
```

```
Call:
ergm(formula = neurons ~ edges + nodefactor("type1.new") + nodematch("type1.new"),
    control = control.ergm(seed = 1))

Maximum Likelihood Results:

                      Estimate Std. Error MCMC % z value Pr(>|z|)
edges                  -5.5203     0.3742      0 -14.752   <1e-04 ***
nodefactor.type1.new.1  2.4305     0.3674      0   6.616   <1e-04 ***
nodematch.type1.new    -3.2727     0.3742      0  -8.745   <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 52444  on 37830  degrees of freedom
 Residual Deviance:  1822  on 37827  degrees of freedom

AIC: 1828  BIC: 1853  (Smaller is better. MC Std. Err. = 0)
```

As we can see every parameter is significant. The interpretation is the same as before.

Markov Model

We move in the direction of including stars. Given the best model until now, we add parameters.

We add the `instar(2)` and `triangles` parameter. Unfortunately, `ostar(2)` can't be used due to model degeneracy.

```
markov = ergm(neurons ~ edges + nodefactor("type1.new") +
              nodematch("type1.new") + istar(2) + triangles,
             control = control.ergm(seed = 1))
```

```
summary(markov)
```

```
Call:
ergm(formula = neurons ~ edges + nodefactor("type1.new") + nodematch("type1.new") +
    istar(2) + triangles, control = control.ergm(seed = 1))

Monte Carlo Maximum Likelihood Results:

                      Estimate Std. Error MCMC % z value Pr(>|z|)
edges                  -3.6724     0.5329      0  -6.892   <1e-04 ***
nodefactor.type1.new.1  3.1929     0.4647      0   6.871   <1e-04 ***
nodematch.type1.new    -3.1103     0.5011      0  -6.206   <1e-04 ***
istar2                 -3.1225     0.3093      0 -10.094   <1e-04 ***
triangle                0.9264     0.5341      0   1.734   0.0828 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 52444  on 37830  degrees of freedom
 Residual Deviance:  1528  on 37825  degrees of freedom

AIC: 1538  BIC: 1580  (Smaller is better. MC Std. Err. = 2.633)
```

Let's tweak this model a bit. We know that no triangles are present in the network, so we remove it and see if `BIC` improves. We also take in consideration `nodefactor("type2.new")`.

```
markov_no_triangles = ergm(neurons ~ edges + nodefactor("type1.new")
                           + nodematch("type1.new") + nodefactor("type2.new") + istar(2),
             control = control.ergm(seed = 1))
```

```
summary(markov_no_triangles)
```

```
Call:
ergm(formula = neurons ~ edges + nodefactor("type1.new") + nodematch("type1.new") +
    nodefactor("type2.new") + istar(2), control = control.ergm(seed = 1))

Monte Carlo Maximum Likelihood Results:

                      Estimate Std. Error MCMC % z value Pr(>|z|)
edges                  -2.7284     0.5615      0  -4.859   <1e-04 ***
nodefactor.type1.new.1  3.0275     0.3956      0   7.653   <1e-04 ***
nodematch.type1.new    -2.7376     0.3813      0  -7.179   <1e-04 ***
```

```
nodefactor.type2.new.1  -1.0756     0.2690      0  -3.998   <1e-04 ***
istar2                  -3.3052     0.3501      0  -9.441   <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 52444  on 37830  degrees of freedom
 Residual Deviance:  1508  on 37825  degrees of freedom

AIC: 1518  BIC: 1561  (Smaller is better. MC Std. Err. = 2.815)
```

The result of basic Markov models are summarized in the table below.

Model	BIC
markov	1580.34
markov_no_triangles	1560.93

Markov with alternating k_stars

Since we want to exploit the stars in the graph, a solution is using the "*alternating k_stars*". We are not interest in "*alternating k-paths*" since they are not observed in the network.

> Maybe what follows is a little cheating or inaccurate, but the only way to let the estimation end is using a *Stochastic-Approximation*.

```
k_star = ergm(neurons ~ edges + nodefactor("type1.new") + nodefactor("type2.new")
              + nodematch("type1.new") + gwidegree(decay = 1, fixed = TRUE)
              + gwodegree(decay = 1, fixed = TRUE) ,
             control = control.ergm(seed = 1, main.method = "Stochastic-Approximation"))
```

```
               edges nodefactor.type1.new.1 nodefactor.type2.new.1    nodematch.type1.new
           -9.075946              -3.359711              -1.543923              -4.045549
       gwideg.fixed.1         gwodeg.fixed.1
           16.247638             -10.634592
Starting burnin of 16384 steps
Phase 1: 200 steps (interval = 1024)
```

```
summary(k_star)
```

```
Call:
ergm(formula = neurons ~ edges + nodefactor("type1.new") + nodefactor("type2.new") +
    nodematch("type1.new") + gwidegree(decay = 1, fixed = TRUE) +
    gwodegree(decay = 1, fixed = TRUE), control = control.ergm(seed = 1,
    main.method = "Stochastic-Approximation"))

Stochastic Approximation Maximum Likelihood Results:

                      Estimate Std. Error MCMC % z value Pr(>|z|)
edges                  -6.5542     0.9519      0  -6.885  < 1e-04 ***
nodefactor.type1.new.1 -1.0905     0.3134      2  -3.480 0.000502 ***
```

```
nodefactor.type2.new.1  -0.9781     0.3554      0  -2.752 0.005923 **
nodematch.type1.new     -2.4491     0.4043      1  -6.058  < 1e-04 ***
gwideg.fixed.1           8.3180     1.2782      1   6.508  < 1e-04 ***
gwodeg.fixed.1          -7.1807     0.6050      1 -11.869  < 1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 52444  on 37830  degrees of freedom
 Residual Deviance:  1219  on 37824  degrees of freedom

AIC: 1231  BIC: 1282  (Smaller is better. MC Std. Err. = 3.638)
```

This is the best model we can get.

Social Circuits

We also propose a social circuit model, but it has worst fit with respect to the previus models. we quote it anyway for completeness.

```
social = ergm(neurons ~ edges + nodefactor("type1.new") + nodefactor("type2.new")
            + nodematch("type1.new") + gwdsp(decay = 1, fixed = T),
            control = control.ergm(seed=1, main.method = "Stochastic-Approximation" ))
```

```
               edges nodefactor.type1.new.1 nodefactor.type2.new.1    nodematch.type1.new
          -5.3562245              3.1484838             -0.1834090             -2.4840012
     gwdsp.OTP.fixed.1
          -0.6633733
Starting burnin of 16384 steps
Phase 1: 200 steps (interval = 1024)
```

```
summary(social)
```

```
Call:
ergm(formula = neurons ~ edges + nodefactor("type1.new") + nodefactor("type2.new") +
    nodematch("type1.new") + gwdsp(decay = 1, fixed = T), control = control.ergm(seed = 1,
    main.method = "Stochastic-Approximation"))

Stochastic Approximation Maximum Likelihood Results:

                     Estimate Std. Error MCMC % z value Pr(>|z|)
edges                 -5.7570     0.5766      1  -9.985  < 1e-04 ***
nodefactor.type1.new.1   3.7323     0.5607      1   6.656  < 1e-04 ***
nodefactor.type2.new.1  -0.4263     0.1573      1  -2.711  0.00671 **
nodematch.type1.new     -2.3133     0.5639      1  -4.102  < 1e-04 ***
gwdsp.OTP.fixed.1       -0.3226     0.1122      0  -2.876  0.00403 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 52444  on 37830  degrees of freedom
 Residual Deviance:  1543  on 37825  degrees of freedom

AIC: 1553  BIC: 1596  (Smaller is better. MC Std. Err. = 2.045)
```

Recap of models

To recap the performances of models the best ones are `markov_no_triangles` and `k_star` (which is however estimated with a stochastic approach).

The table below recap the goodness of fit of the best models compared to the homogeneous SRG.

Model	BIC
srg_homo	2652.18
markov_no_triangles	1560.93
k_star	1282.49

Simulazione

Now that we have chosen the two best models, we can test them in a simulation to see if they can model the network. The reference will be the performance of the homogeneous simple random graph.

```
nsim=100
sim_srg = simulate(srg_homo, nsim = nsim, verbose = TRUE, seed = 1)
sim_markov_no_triangles = simulate(k_star, nsim = nsim, verbose = TRUE, seed = 1)
sim_k_star = simulate(k_star, nsim = nsim, verbose = TRUE, seed = 1)
```

```
fnc = function(xx){
  ig = asIgraph(xx)
  tr = transitivity(ig)
  ideg = sd(degree(ig, mode = "in"))
  odeg = sd(degree(ig, mode = "out"))
  return(c(tr, ideg, odeg))
}

null.distr.srg = matrix(,nsim,3)
null.distr.markov_no_triangles = matrix(,nsim,3)
null.distr.k_star = matrix(,nsim,3)
for(b in 1:nsim){
  null.distr.srg[b,]  = fnc(sim_srg[[b]])
  null.distr.markov_no_triangles[b,]  = fnc(sim_markov_no_triangles[[b]])
  null.distr.k_star[b,]  = fnc(sim_k_star[[b]])
}
```

SRG simulation

```
dev.new()
par(mfrow = c(3,1))
hist(unlist(null.distr.srg[,1]), xlab = "transitivity");
abline(v = transitivity(neurons_g), col = "red")
hist(unlist(null.distr.srg[,2]), xlim = c(0.25,1.25),xlab = "in-degree");
abline(v = sd(degree(neurons_g, mode = "in")), col = "red")
hist(unlist(null.distr.srg[,3]), xlim = c(1,5.25),xlab = "out-degree");
abline(v = sd(degree(neurons_g, mode = "out")), col = "red")
```

Markov without triangles simulation

```
dev.new()
par(mfrow = c(3,1))
hist(unlist(null.distr.markov_no_triangles[,1]), xlab = "transitivity");
abline(v = transitivity(neurons_g), col = "red")
hist(unlist(null.distr.markov_no_triangles[,2]), xlab = "in-degree");
abline(v = sd(degree(neurons_g, mode = "in")), col = "red")
hist(unlist(null.distr.markov_no_triangles[,3]),
     xlim = c(4,5.25),xlab = "out-degree");
abline(v = sd(degree(neurons_g, mode = "out")), col = "red")
```

Alternating k-stars simulation

```
dev.new()
par(mfrow = c(3,1))
hist(unlist(null.distr.k_star[,1]), xlab = "transitivity");
abline(v = transitivity(neurons_g), col = "red")
hist(unlist(null.distr.k_star[,2]), xlab = "in-degree");
abline(v = sd(degree(neurons_g, mode = "in")), col = "red")
hist(unlist(null.distr.k_star[,3]), xlim = c(4,5.25),xlab = "out-degree");
abline(v = sd(degree(neurons_g, mode = "out")), col = "red")
```

Conclusions from model simulations

We can see a marked improvement over the SRG. We are able to model transitivity and in degree. Unfortunately, we still cannot model the outgoing degree (the stars) with the estimated models. We can however be satisfied with the closeness of the estimate. As far as the choice of model is concerned, having similar performance, we prefer the Markov model without triangles because it was estimated using the MCMC procedure.