Project 8 Nested Effect Models

Team K - Minghang Li

April 26, 2023

Problem 20: Classical NEMs

Subproblem 1

Construct transitive closure and define Φ

Define a function to make any Φ matrix transitive closed by powering it up until convergence.

```
transitive_closify <- function(phi) {
  old_phi <- phi
  while (TRUE) {
    new_phi <- old_phi %*% phi
    new_phi[new_phi > 0] <- 1
    if (isTRUE(all.equal(new_phi, old_phi))) {
      break
    }
    old_phi <- new_phi
  }
  return(new_phi)
}
```

Construct Φ for Model (a).

```
phi_a <- array(
  dim = c(5, 5),
  dimnames = list(c("S1", "S2", "S3", "S4", "S5"),
                  c("S1", "S2", "S3", "S4", "S5"))
)
#                  S1, S2, S3, S4, S5
phi_a["S1", ] <- c( 1,  0,  1,  1,  0)
phi_a["S2", ] <- c( 0,  1,  0,  0,  1)
phi_a["S3", ] <- c( 0,  0,  1,  1,  1)
phi_a["S4", ] <- c( 0,  0,  0,  1,  1)
phi_a["S5", ] <- c( 0,  0,  0,  0,  1)
```

```
phi_a <- transitive_closify(phi_a)
phi_a
```

```
##    S1 S2 S3 S4 S5
## S1  1  0  1  1  1
## S2  0  1  0  0  1
## S3  0  0  1  1  1
## S4  0  0  0  1  1
## S5  0  0  0  0  1
```

Construct Φ for Model (b).

```
phi_b <- array(
  dim = c(5, 5),
  dimnames = list(c("S1", "S2", "S3", "S4", "S5"),
                  c("S1", "S2", "S3", "S4", "S5"))
)
#                  S1, S2, S3, S4, S5
phi_b["S1", ] <- c( 1,  0,  0,  1,  0)
phi_b["S2", ] <- c( 0,  1,  0,  0,  1)
phi_b["S3", ] <- c( 1,  0,  1,  1,  1)
phi_b["S4", ] <- c( 0,  0,  0,  1,  1)
phi_b["S5", ] <- c( 0,  0,  0,  0,  1)
```

```
phi_b <- transitive_closify(phi_b)
phi_b
```

```
##    S1 S2 S3 S4 S5
## S1  1  0  0  1  1
## S2  0  1  0  0  1
## S3  1  0  1  1  1
## S4  0  0  0  1  1
## S5  0  0  0  0  1
```

Define Θ

Define Θ for Model (a).

```
theta_a <- array(
  dim = c(5, 6),
  dimnames = list(c("S1", "S2", "S3", "S4", "S5"),
                  c("E1", "E2", "E3", "E4", "E5", "E6"))
)
#                    E1, E2, E3, E4, E5, E6
theta_a["S1", ] <- c( 0,  0,  0,  0,  0,  0)
theta_a["S2", ] <- c( 0,  0,  0,  1,  0,  1)
theta_a["S3", ] <- c( 1,  1,  0,  0,  0,  0)
theta_a["S4", ] <- c( 0,  0,  1,  0,  0,  0)
theta_a["S5", ] <- c( 0,  0,  0,  0,  1,  0)
theta_a
```

```
##    E1 E2 E3 E4 E5 E6
## S1  0  0  0  0  0  0
## S2  0  0  0  1  0  1
```

```
## S3  1  1  0  0  0  0
## S4  0  0  1  0  0  0
## S5  0  0  0  0  1  0
```

Define Θ for Model (b).

```
theta_b <- array(
  dim = c(5, 6),
  dimnames = list(c("S1", "S2", "S3", "S4", "S5"),
                  c("E1", "E2", "E3", "E4", "E5", "E6"))
)
#                     E1, E2, E3, E4, E5, E6
theta_b["S1", ] <- c( 1,  1,  0,  0,  0,  0)
theta_b["S2", ] <- c( 0,  0,  0,  1,  0,  1)
theta_b["S3", ] <- c( 0,  0,  0,  0,  0,  0)
theta_b["S4", ] <- c( 0,  0,  1,  0,  0,  0)
theta_b["S5", ] <- c( 0,  0,  0,  0,  1,  0)
theta_b
```

```
##    E1 E2 E3 E4 E5 E6
## S1  1  1  0  0  0  0
## S2  0  0  0  1  0  1
## S3  0  0  0  0  0  0
## S4  0  0  1  0  0  0
## S5  0  0  0  0  1  0
```

Determine the corresponding expected effect patterns (F)

```
F_a <- phi_a %*% theta_a
F_a
```

```
##    E1 E2 E3 E4 E5 E6
## S1  1  1  1  0  1  0
## S2  0  0  0  1  1  1
## S3  1  1  1  0  1  0
## S4  0  0  1  0  1  0
## S5  0  0  0  0  1  0
```

```
F_b <- phi_b %*% theta_b
F_b
```

```
##    E1 E2 E3 E4 E5 E6
## S1  1  1  1  0  1  0
## S2  0  0  0  1  1  1
## S3  1  1  1  0  1  0
## S4  0  0  1  0  1  0
## S5  0  0  0  0  1  0
```

Subproblem 2

If we assume no noise (no false positives and false negatives)… then the D matrix is simply the F matrix transpose.

```
D_a <- t(F_a)
D_a
```

```
##    S1 S2 S3 S4 S5
## E1  1  0  1  0  0
## E2  1  0  1  0  0
## E3  1  0  1  1  0
## E4  0  1  0  0  0
## E5  1  1  1  1  1
## E6  0  1  0  0  0
```

```
D_b <- t(F_b)
D_b
```

```
##    S1 S2 S3 S4 S5
## E1  1  0  1  0  0
## E2  1  0  1  0  0
## E3  1  0  1  1  0
## E4  0  1  0  0  0
## E5  1  1  1  1  1
## E6  0  1  0  0  0
```

Given the discrete data D_a and D_b (sorry for the different notation from the exercise pdf) it's not possible to tell apart the two models because they are identical.

```
all.equal(D_a, D_b)
```

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

Subproblem 3

Calculate the marginal log-likelihood ratio (network score) given the data by setting the false positive rate to be 5% and the false negative rate to be 1%.

```
network_score_a <- scoreAdj(D_a,
                            adj = phi_a,
                            method = "disc",
                            fpfn = c(0.05, 0.01)
                            )$score
network_score_a
```

```
## [1] 60.304
```

```
network_score_b <- scoreAdj(D_b,
                            adj = phi_b,
                            method = "disc",
                            fpfn = c(0.05, 0.01)
                            )$score
network_score_b
```

```
## [1] 60.304
```

Problem 21: Hidden Markov NEMs

Subproblem 1

Compute the transition probabilities from $G_t = u$ to $G_{t+1} \in \{v_1, v_2\}$ for different smoothness parameter $\lambda \in \{0.1, \dots, 0.9\}$.

By definition, the probability of transition from network u to network v is calculated by:

$$\begin{aligned} T_{uv} &= P(\Phi_{t+1} = v | \Phi_t = u) \\ &= \frac{1}{C_u}(1-\lambda)^{s_{uv}} \cdot \lambda \end{aligned}$$

The distance s_{uv} is defined as

$$s_{uv} = ||u - v||_1 := \sum_i \sum_{i'} |u_{ii'} - v_{ii'}|$$

The normalizing constant C_u is defined as

$$C_u = \sum_w (1-\lambda)^{s_{uw}} \cdot \lambda$$

where w is all possible networks given the S genes at hand.

So the basic implementation idea would be:

1. Represent u, v_1 and v_2 using adjacency matrix
2. Compute s_{uv_1} and s_{uv_2} by "diff"ing the pairs of matrices respectively
3. Generate all the networks w using `mnem`, compute all the s_{uw}.
4. Compute transition probability for each λ

Implementation in R is in the following code blocks.

```
S_genes <- c("S1", "S2", "S3", "S4")
```

```
# Initialization of u
u <- array(
  dim = c(4, 4),
  dimnames = list(S_genes, S_genes)
)
#              S1, S2, S3, S4
```

```
u["S1", ] <- c( 1,  1,  1,  0)
u["S2", ] <- c( 0,  1,  1,  1)
u["S3", ] <- c( 0,  0,  1,  1)
u["S4", ] <- c( 0,  0,  0,  1)
u
```

```
##    S1 S2 S3 S4
## S1  1  1  1  0
## S2  0  1  1  1
## S3  0  0  1  1
## S4  0  0  0  1
```

```
# Initialization of v1
v1 <- array(
  dim = c(4, 4),
  dimnames = list(S_genes, S_genes)
)
#               S1, S2, S3, S4
v1["S1", ] <- c( 1,  1,  1,  0)
v1["S2", ] <- c( 0,  1,  1,  1)
v1["S3", ] <- c( 0,  0,  1,  0)
v1["S4", ] <- c( 0,  0,  0,  1)
v1
```

```
##    S1 S2 S3 S4
## S1  1  1  1  0
## S2  0  1  1  1
## S3  0  0  1  0
## S4  0  0  0  1
```

```
# Initialization of v2
v2 <- array(
  dim = c(4, 4),
  dimnames = list(S_genes, S_genes)
)
#               S1, S2, S3, S4
v2["S1", ] <- c( 1,  0,  0,  0)
v2["S2", ] <- c( 1,  1,  1,  0)
v2["S3", ] <- c( 1,  0,  1,  0)
v2["S4", ] <- c( 1,  0,  0,  1)
v2
```

```
##    S1 S2 S3 S4
## S1  1  0  0  0
## S2  1  1  1  0
## S3  1  0  1  0
## S4  1  0  0  1
```

```
# compute s_uv1 and s_uv2
s_uv1 <- sum(u ≠ v1)
s_uv2 <- sum(u ≠ v2)
```

```r
# generate all the possible networks
all_networks <- mnem:::enumerate.models(S_genes, trans.close = FALSE)
```

```
## Generated 4096 unique models ( out of 4096 )
```

```r
# compute s_uw for all networks
num_cores <- detectCores()
registerDoParallel(num_cores)
start <- Sys.time()
s_uw <- foreach (i=1:length(all_networks), .combine = c) %dopar% {
  sum(u ≠ all_networks[[i]])
}
end <- Sys.time()
end - start
stopImplicitCluster()
```

```r
compute_C <- function(lambda, s_uw) {
  res <- foreach (i=1:length(s_uw), .combine = c) %dopar% {
    (1 - lambda)^s_uw[i] * lambda
  }
  return(sum(res))
}
```

```r
# Compute transitive probability
registerDoParallel(num_cores)
start <- Sys.time()
trans_prob <- foreach (lambda=seq(0.1, 0.9, by=0.1), .combine = rbind) %dopar% {
  C_u <- compute_C(lambda = lambda, s_uw = s_uw)
  res1 <- (1 - lambda)^s_uv1 * lambda / C_u
  res2 <- (1 - lambda)^s_uv2 * lambda / C_u
  return(c(res1, res2))
}
end <- Sys.time()
end - start
```

```
## Time difference of 0.8714516 secs
```

```r
stopImplicitCluster()
colnames(trans_prob) <- c("v1", "v2")
rownames(trans_prob) <- sprintf("%.1f", seq(0.1, 0.9, 0.1))
trans_prob
```

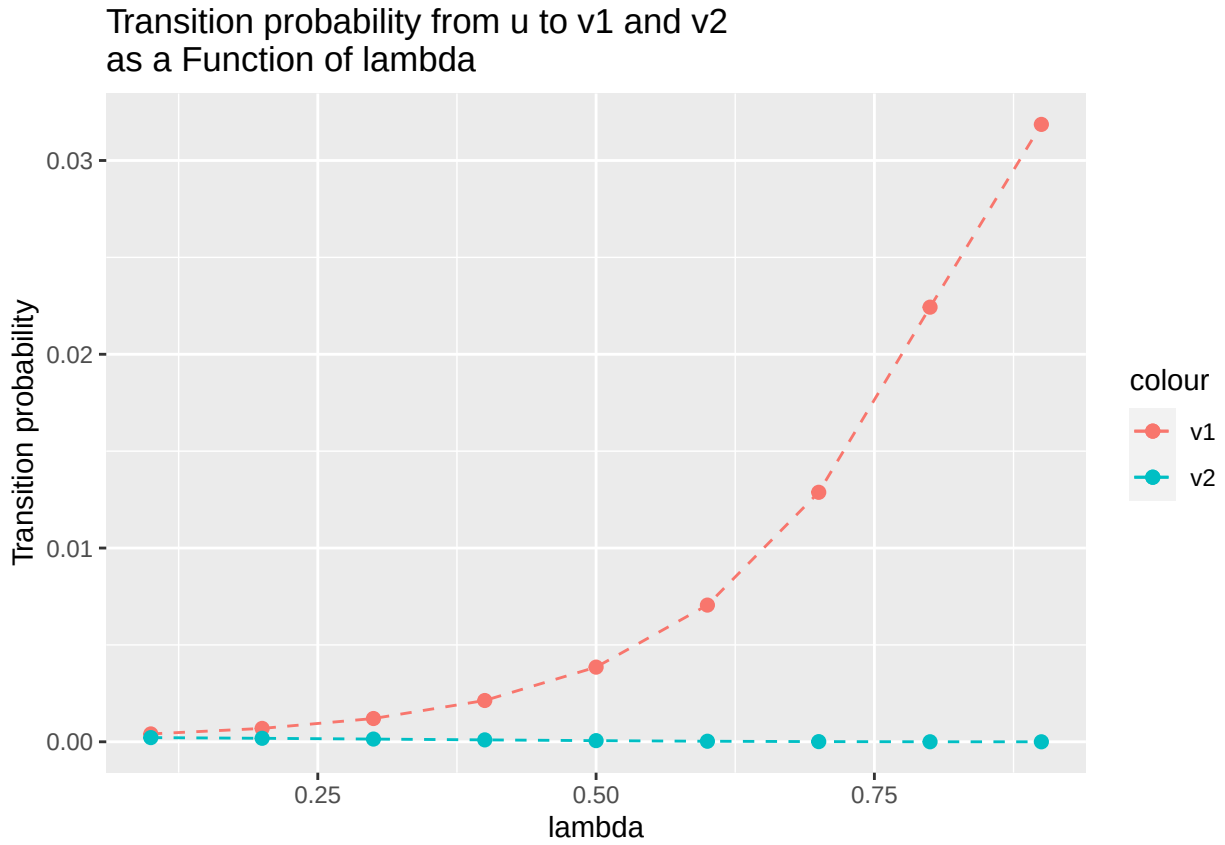
```
##               v1           v2
## 0.1 0.0004066299 2.160998e-04
## 0.2 0.0006915442 1.812842e-04
## 0.3 0.0012014646 1.413511e-04
## 0.4 0.0021316282 9.945325e-05
## 0.5 0.0038536733 6.021365e-05
## 0.6 0.0070554312 2.889905e-05
## 0.7 0.0128765947 9.387038e-06
## 0.8 0.0224313310 1.435605e-06
## 0.9 0.0318630818 3.186308e-08
```

Subproblem 2

Plot the transition probabilites as a function of λ for v_1 and v_2.

```
df <- data.frame(lambda = row.names(trans_prob), trans_prob, row.names = NULL)
df$lambda <- as.numeric(df$lambda)
```

```
ggplot(data = df, aes(x = lambda, group = 1)) +
  geom_line(aes(y = v1, color = "v1"), linetype = "dashed") +
  geom_point(aes(y = v1, color = "v1"), size = 2) +
  geom_line(aes(y = v2, color = "v2"), linetype = "dashed") +
  geom_point(aes(y = v2, color = "v2"), size = 2) +
  xlab("lambda") +
  ylab("Transition probability") +
  ggtitle("Transition probability from u to v1 and v2\nas a Function of lambda")
```



We can see that the transition probabilities of v_1 and v_2 converge when λ is small and differ greatly when λ is large. Dissimilar networks get penalized and result in lower transition probability as λ increase. For similar networks like v_1, the probability to transit into them increases as we increase λ.

Problem 22: Mixture NEMs

Subproblem 1

Subproblem 2

(a) Compute expected effect pattern $(\rho^T \phi_k \theta_k)^T$

(b)

Subproblem 3