# Simplified model

Lapo Santi

2023-02-23

```
match_2017_url <- 'https://pkgstore.datahub.io/sports-data/atp-world-tour-tennis-data/match_scores_2017_
df_match <- read.csv(match_2017_url)
edgelist <- data.frame(df_match$winner_name, df_match$loser_name)
reversed_edgelist <- data.frame(df_match$loser_name,df_match$winner_name)

data_clean = data.frame(player1=NA, player2=NA, games=NA, victories=NA)
n=nrow(edgelist)
while(n>0){
  entries = c(which(edgelist[,1] == edgelist[1,1] & edgelist[,2] == edgelist[1,2]), which(edgelist[,1] =
  df_support = data.frame(matches = edgelist[entries,])
  n_games = nrow(df_support)
  n_victories = sum(edgelist[,1] == edgelist[1,1] & edgelist[,2] == edgelist[1,2])
  # if(n_victories == sum(which(df_support[,1] == edgelist[i,1] & df_support[,2] == edgelist[i,2]))){
      data_clean = rbind(data_clean, data.frame(player1 = edgelist[1,1], player2 = edgelist[1,2], games
      # print("ok")
    # }else(print("error"))
  edgelist = edgelist[-entries,]
  n=nrow(edgelist)
}
data_clean = na.omit(data_clean)

#check if the computations are ok
sum(data_clean$games)==3830
```

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

## Data

The data saved in "match_2017_url" consists in 3830 rows reporting the results for 3830 matches; among the various statistics, I extract the winner and the loser name. The problem is how to store and work with such pair-wise comparisons.

### Matrix representation problem

Usually, studies on networks store pairwise comparison data into matrices made of valued entries with different meanings. For example, entry $A_{ij} = 1$ if player $i$ wins against player $j$ and 0 otherwise. However, in this exercise, I am not considering those pairs of players that have never played against each other. In my opinion, this makes a big difference in how I can represent the data. Now, If I want to get rid of the 0s and retain the matrix representation, I should consider just the small number of players such that they all have played against each other at least once. This would guarantee to have a strictly positive-definite matrix, but at a great cost: indeed, it would mean to remove most of the observations, that is, to reduce significantly the sample size. To avoid that, I abandon the classic matrix representation to switch to a dataset representation

| **Winner** | **Loser** |
|---|---|
| Djockovic | Medvedev |
| ⋮ | ⋮ |
| Djockovic | Medvedev |
| Djockovic | Medvedev |
| Medvedev | Djockovic |
| ⋮ | ⋮ |
| Medvedev | Djockovic |
| Medvedev | Djockovic |

Table 1: Raw data

| **Player1** | **Player2** | $n_{games}$ | $n_{victories}$ |
|---|---|---|---|
| Djockovic | Medvedev | 5 | 3 |
| Djockovic | Nadal | 8 | 6 |
| Nadal | Tsonga | 4 | 0 |

Table 2: Final dataset

that hereby I describe. The final result can be found in the code, assigned to the variable "data_clean"

**Constructing the dataset**

Starting from the raw data, If two players have played multiple times, they compare on different rows as we can see in table(1).

To begin with, I select all rows with the first pair of players, regardless of the order in which they apper. Then, I construct these two new variables:

- $n_{games}$, e.g. total number of games between Djockovic and Medvedev = total number of victories of Djockovic vs Medvedev + total number of victories of Medvedev vs Djockovic. Basically one counts the number of times these 2 names appear on the same row, irrespective of their order
- $n_{victories}$, e.g. total number of victories of Djockovic against Medvedev = total number of victories of Djockovic vs Medvedev. Basically one counts the number of times Djockovic and Medvedev appear exactly in this order on the same row.

Then, I retain just the first pair observation, e.g. Djockovic & Medvedev, and I disregards all the others. Then, I repeat the same process for all unique unordered pairs of players. In this way, I obtain a dataset storing, for each pair of players who have played at least one, all the relevant informations in one single entry.

What about the observations in which Medvedev won against Djockovic? These will no longer be an entries of the dataframe.However, this data will still be indirectly available by computing total number of games between Djockovic and Medvedev (available in column $n_{games}$, at the correspective row) - total number of victories of Djockovic vs Medvedev (available in $n_{victories}$,at the correspective row).

The final dataframe will look like as in table(2)

# Generating synthetic data

In the following section, I simulate players, with different strengths, and a tournament, in which these players compete. The function `generating_synthetic_data` creates the simulated tournament by taking the following arguments:

- `alpha, beta`: the parameters of the beta distribution. The entries of the $p$ matrix, containing the inter-block connection probabilities;

- `K`: the number of blocks in the simulated data;
- `n`: the number of players in the tournament;
- `max_number_games`: max number of times two players can play against each other

The function does the following steps:

1) It simulates the matrix $p$, by using the code explained in the report "Prior". The simulated matrix exhibit the SST property. Therefore, it will be interesting to estimate it first with a simplified model and then with a model suited for the ordered data.
2) It assigns, uniformly at random, to each of the player a label $k \in \{1, \dots, \texttt{K}\}$, standing for the block membership
3) It simulates the tournament, by extracting, uniformly at random, the pair of players that have played against each other. The tournament is stored in the dataframe `data_clean_fake`. Every row of the dataframe contains in the first two entries the players who have played against each other at least once. I attach in this dataframe also the block membership for each pair of players.
4) Still uniformly at random, it samples $n_{ij} \in \{1, \dots, \texttt{max\_number\_games}\}$, for each pair of players.
5) It samples $y_{ij} \sim Binom(n_{ij}, p_{z_i, z_j})$
6) It returns both the actual membership (`z_true`) and the simulated tournament (`matches_results`)

```
generating_synthetic_data = function(alpha, beta, K, n, max_number_games ){
  # alpha: first parameter of the beta(alpha,beta) distribution
  # beta:second parameter of the beta(alpha,beta) distribution
  # K: number of clusters in the generated network
  # n: number of players
  # max_number_games: maximum number of times 2 players are allowed to play against each other


  #P matrix generating the SST P matrix ---
  # the code is explained in the "prior" file
  p_fake = sampling_SST_matrix_beta(K,alpha, beta)
  p_fake[is.na(p_fake)] <- 0
  p_fake = (1 -t(p_fake*upper.tri(p_fake))) * (lower.tri(p_fake, diag = F)*1) + upper.tri(p_fake, diag =

  #Z vector: assigning to players a random block membership ----
  players_fake = data.frame(id = c(1:n), z_true = sample(1:K, n, replace = T) )
  #----

  ## Simulating the tournament: the two first columns contain the pair of players
  #who have playerd against each other ------------

  #check whether we teo identical rows: if so, resample

  while(sum(data_clean_fake$player1 == data_clean_fake$player2)>1){
  data_clean_fake = data.frame(player1 = sample(n_fake,n_fake,replace = F), player2 = sample(n_fake,n_fa
  }


  #including the block membership for player1 and player 2 into the main dataframe
  for(i in 1:nrow(data_clean_fake)){
    data_clean_fake$z_player1[i] = players_fake[which(players_fake$id == data_clean_fake$player1[i]),]$z
    data_clean_fake$z_player2[i] =players_fake[which(players_fake$id == data_clean_fake$player2[i]),]$z_
  }
```

```r
#sampling n_ij for each pair of players
data_clean_fake = cbind(data_clean_fake, n_ij = sample((1:max_number_games),n_fake, replace = T))
#sampling y_ij for each pair of players according to y ~ Binom(n_ij, p_{zi,zj})

data_clean_fake = cbind(data_clean_fake, y_ij = rep(0,n_fake))
for(i in 1:n_fake){
  data_clean_fake$y_ij[i] = rbinom(1, size = data_clean_fake$n_ij[i], p= p_fake[data_clean_fake$z_play
}

#removing the block membership from the dataframe (this info appears into player fake df)
data_clean_fake = data_clean_fake[,-c(3:4)]
colnames(data_clean_fake) = c("player1", "player2","nij", "yij")
#------------

#The function returns:--------
# players_fake: a Nx2 dataframe: players ID in the first column, players block membership in the seco
# matches_result: sa Nx4 dataframe: player1 ID in the first column,player2 ID in the second column pl
#n_ij in the third column, y_ij in the fourth colomun
# p_true: KxK matrix containing the interblock connection probabilities
#-------------
return(list(z_true = players_fake, matches_results = data_clean_fake, p_true = p_fake))}
```

# Generative model

In this setting we denote the nodes as $1, ..., n$.

- $K \sim Poi(1; K > 0)$
- $\theta_1, ..., \theta_K | K \sim Dir(\gamma)$
- $z_i | \theta_1, ..., \theta_K, K \overset{iid}{\sim} Multi(1; \theta_1, ..., \theta_K)$
- $P = \{p_{i,j} Beta(a, b)\}$
- $y_{i,j} | n_{ij} > 0, z_i, z_j, p_{z_i,z_j} \sim Bin(n_{i,j}, p_{z_i,z_j}), n_i j > 0$

$z_i \in \{1, ..., K\}$

## Specifying the likelihood

$$p(\mathbf{y}|p, \theta, z, \gamma) = \prod_{i<j}^{n-1} \binom{n_{ij}}{y_{ij}} p_{z_i,z_j}^{y_{ij}} (1 - p_{z_i,z_j})^{n_{ij} - y_{ij}} \tag{1}$$

## Specifying the prior

Starting with the prior on $z$

$$p(z_1, \ldots, z_k | \theta_1, \ldots, \theta_k) = \frac{\Gamma(\sum_i n_i + 1)}{\prod_i \Gamma(n_i + 1)} \prod_{i=1}^{K} \theta_i^{n_i} \tag{2}$$

where $n_i$ accounts for the number of nodes/ players allocated to block $i$. On $\theta$ instead, the Dirichlet prior has the following shape:

$$f\left(\theta_1, \ldots, \theta_K; \gamma_1, \ldots, \gamma_K\right) = \frac{1}{\mathrm{B}(\boldsymbol{\gamma})} \prod_{i=1}^{K} \theta_i^{\gamma_i - 1} \tag{3}$$

By marginalizing out $\theta$, and setting the hyperparameter $\boldsymbol{\gamma}/K = 1$ we can write the marginal distribution of $z$ as

$$p(z|K) = \frac{\Gamma(K)}{\Gamma(N+K)} \prod_{i=1}^{K} \Gamma(1 + n_i) \tag{4}$$

Contrary to the literature, here I do not want to integrate out the beta distribution on the parameter $p$

$$p(\mathbf{p}) = \prod_{i<j}^{K} \frac{1}{Beta(a,b)} p_{ij}^{\alpha-1} \cdot (1 - p_{ij})^{\beta-1} \tag{5}$$

Finally, on the parameter $K$:

$$p(K|\lambda = 1) = \frac{1^K e^{-1}}{K!} = \frac{1}{e \cdot K!} \propto \frac{1}{K!} \tag{6}$$

# Estimation

First I set $\alpha = 1$. The posterior will be proportional to

$$\begin{aligned} p(z, \beta, K|\mathbf{y}, z, \gamma) &= \prod_{i<j}^{N} \binom{n_{ij}}{y_{ij}} p_{z_i,z_j}^{y_{ij}} (1 - p_{z_i,z_j})^{n_{ij} - y_{ij}} \cdot \\ &\cdot \prod_{i<j}^{K} \frac{1}{Beta(1,\beta)} p_{ij}^{1-1} \cdot (1 - p_{ij})^{\beta-1} \cdot \prod_{i=1}^{K} \frac{\Gamma(K)}{\Gamma(N+K)} \prod_{i=1}^{K} \Gamma(1 + n_i) \cdot \frac{1}{K!} \\ &\propto \prod_{i<j}^{N} \binom{n_{ij}}{y_{ij}} p_{z_i,z_j}^{y_{ij}} (1 - p_{z_i,z_j})^{n_{ij} - y_{ij}} \cdot \\ &\cdot \prod_{i<j}^{K} \frac{1}{Beta(1,\beta)} \cdot (1 - p_{ij})^{\beta-1} \cdot \prod_{i=1}^{K} \frac{\Gamma(K)}{\Gamma(N+K)} \prod_{i=1}^{K} \Gamma(1 + n_i) \cdot \frac{1}{K!} \end{aligned} \tag{7}$$

### Not integrating out the parameter $p$

Another difference with respect to the most common modelling in the literature is that here I am not integrating out the parameter $p$. This choice is motivated by two reasons:

1) By looking at the next step of the research. Given that the ultimate goal is to impose the stochastic ordering on the parameter $p$, I need a suituable way of estimating it. To do that, I need to sample matrices $p$ and evaluate them in a Metropolis-Hastings way.

2) There is a problem in collapsing the likelihood at the block level, that is, rewriting:

$$\begin{aligned} p(\mathbf{y}|p, \theta, z, \gamma) &= \prod_{i<j}^{N} \binom{n_{ij}}{y_{ij}} p_{z_i,z_j}^{y_{ij}} (1 - p_{z_i,z_j})^{n_{ij} - y_{ij}} \\ &= \prod_{i<j}^{K} \binom{n_{z_i,z_j}}{y_{z_i,z_j}} p_{z_i,z_j}^{y_{z_i,z_j}} (1 - p_{z_i,z_j})^{n_{z_i,z_j} - y_{z_i,z_j}} \end{aligned} \tag{8}$$

where $n_{z_i,z_j}, y_{z_i,z_j}$ denote respectively the number of games played between two given pairs of blocks. The problem is the following: the number of games between the pairs $(i,i)$ for $i \in k = 1,\dots,K$ is equal to the number of victories between the pairs $(i,i)$. When computing the probability $p(y_{z_i,z_j} = n_{z_i,z_j}|\mathbf{p},\mathbf{z}) \approx 0$ for large $n_{z_i,z_j}$ and $p_{ii}$ close to 0.5. Given such conditions, the computer precision is not enough, it just sets the probability value equal to 0. When taking the log of this probability we get $\log(p_{ii}) = -\infty \quad \forall i \in 1,\dots,K$, which breaks the following code. And more in general, to have the same number of games and victories for players in the same block is not very informative.

For these two reasons, I sacrifice some computational efficiency and I do not integrate out $p$.

## Estimation

To estimate the previous posterior, I use a modified version of the Metropolis Hastings algorithm.

1) Initialize all the quantities, namely:

2.1) **FS**(full sweep) is selected

- for each node:
  - reassign it to a new cluster.
  - compute the ratio $r$ of the posterior density at the proposed and current state which will look as follows:

$$\begin{aligned}
\frac{p(\mathbf{z\prime},K,\mathbf{p})}{p(\mathbf{z},K,\mathbf{p})} &= \frac{p(\mathbf{y}|\mathbf{z\prime},\mathbf{p})\cdot p(\mathbf{p}|K)\cdot p(\mathbf{z\prime}|K)\cdot p(K)}{p(\mathbf{y}|\mathbf{z},\mathbf{p})\cdot p(\mathbf{p}|K)\cdot p(\mathbf{z}|K)\cdot p(K)} \\
&= \frac{p(\mathbf{y}|\mathbf{z\prime},\mathbf{p})\cdot p(\mathbf{z\prime}|K)}{p(\mathbf{y}|\mathbf{z},\mathbf{p})\cdot p(\mathbf{z}|K)} \\
&= \frac{\prod_{i<j}^{N}\binom{n_{ij}}{y_{ij}} p_{z_i\prime,z_j\prime}^{y_{ij}}(1-p_{z_i\prime,z_j\prime})^{n_{ij}-y_{ij}}\frac{\Gamma(K)}{\Gamma(N+K)}\prod_{i=1}^{K}\Gamma(1+n_i\prime)}{\prod_{i<j}^{N}\binom{n_{ij}}{y_{ij}} p_{z_i,z_j}^{y_{ij}}(1-p_{z_i,z_j})^{n_{ij}-y_{ij}}\frac{\Gamma(K)}{\Gamma(N+K)}\prod_{i=1}^{K}\Gamma(1+n_i)} \\
&= \frac{\prod_{i<j}^{N}\binom{n_{ij}}{y_{ij}} p_{z_i\prime,z_j\prime}^{y_{ij}}(1-p_{z_i\prime,z_j\prime})^{n_{ij}-y_{ij}}\prod_{i=1}^{K}\Gamma(1+n_i\prime)}{\prod_{i<j}^{N}\binom{n_{ij}}{y_{ij}} p_{z_i,z_j}^{y_{ij}}(1-p_{z_i,z_j})^{n_{ij}-y_{ij}}\prod_{i=1}^{K}\Gamma(1+n_i)}
\end{aligned} \tag{9}$$

  - compare it with $\log(u) \quad u \sim Uniform(0,1)$
  - accept or reject the proposed modification

2.2) **MK** is selected:

- An insertion or a removal step of an empty cluster is proposed with probability 0.5.
- If the insertion attempt is selected:
  - If $K = K_{max}$, the new state is set equal to the current state: $K^{(s+1)} = K_{max}$
  - If $K < K_{max}$, we accept $K^{(s+1)} = K+1$ as the new state with acceptance probability $\min[1,r]$

where $r$ is equal to:

$$
\begin{aligned}
\frac{p(\mathbf{z}, K\prime, \mathbf{p})}{p(\mathbf{z}, K, \mathbf{p})} &= \frac{p(\mathbf{y}|\mathbf{z}, \mathbf{p}) \cdot p(\mathbf{p}|K\prime) \cdot p(\mathbf{z}|K\prime) \cdot p(K\prime)}{p(\mathbf{y}|\mathbf{z}, \mathbf{p}) \cdot p(\mathbf{p}|K) \cdot p(\mathbf{z}|K) \cdot p(K)} \\
&= \frac{p(\mathbf{p}|K\prime) \cdot p(\mathbf{z}|K\prime) \cdot p(K\prime)}{p(\mathbf{p}|K) \cdot p(\mathbf{z}|K) \cdot p(K)} \\
&= \frac{\prod_{i<j}^{K\prime} \cdot (1 - p_{ij}\prime)^{\beta-1} \cdot \frac{\Gamma(K\prime)}{\Gamma(N+K\prime)} \prod_{i=1}^{K\prime} \Gamma(1+n_i) \cdot \frac{1}{K\prime!}}{\prod_{i<j}^{K} \cdot (1 - p_{ij})^{\beta-1} \cdot \frac{\Gamma(K)}{\Gamma(N+K)} \prod_{i=1}^{K} \Gamma(1+n_i) \cdot \frac{1}{K!}} \\
&= \frac{\prod_{i<j}^{K\prime} \cdot (1 - p_{ij}\prime)^{\beta-1} \cdot \frac{\Gamma(K\prime)}{\Gamma(N+K\prime)} \cdot \frac{1}{K\prime!}}{\prod_{i<j}^{K} (1 - p_{ij})^{\beta-1} \cdot \frac{\Gamma(K)}{\Gamma(N+K)} \cdot \frac{1}{K!}}
\end{aligned} \tag{10}
$$

where $K\prime = K + 1$. We simplified the terms involving $n_i$ because adding or removing a cluster does not affect the size of the blocks (newly added clusters have size zero).We can simplify the two terms on the right of the ratio by rewriting:

$$
\frac{\frac{\Gamma(K+1)}{\Gamma(N+(K+1))} \cdot K!}{\frac{\Gamma(K)}{\Gamma(N+K)} \cdot (K+1)!} = \frac{K}{(N+K) \cdot (K+1)} \tag{11}
$$

Putting all together back we have:

$$
\frac{\prod_{i<j}^{K+1} (1 - p_{ij}\prime)^{\beta-1}}{\prod_{i<j}^{K} (1 - p_{ij})^{\beta-1}} \times \frac{K}{(N+K)(K+1)} \tag{12}
$$

## Modifying the $P$ matrix when a new cluster is inserted

2.3) **PE** is selected:

- $p_{ij}$ are sampled according to the following symmetric proposal distribution:

$$
p_{ij} \sim N(p_{ij}^{(s-1)}, \sigma^2) I(0 \leq p_{ij} \leq 1) \tag{13}
$$

- the ratio of the posterior density looks as follows:

$$
\begin{aligned}
\frac{p(\mathbf{z}, K, \mathbf{p}\prime)}{p(\mathbf{z}, K, \mathbf{p})} &= \frac{p(\mathbf{y}|\mathbf{z}, \mathbf{p}\prime) \cdot p(\mathbf{p}\prime|K) \cdot p(\mathbf{z}|K) \cdot p(K)}{p(\mathbf{y}|z, \mathbf{p}) \cdot p(\mathbf{p}|K) \cdot p(\mathbf{z}|K) \cdot p(K)} \\
&= \frac{p(\mathbf{y}|z, \mathbf{p}\prime) \cdot p(\mathbf{p}\prime|K)}{p(\mathbf{y}|z, \mathbf{p}) \cdot p(\mathbf{p}|K)} \\
&= \frac{\prod_{i<j}^{N} \binom{n_{ij}}{y_{ij}} p\prime_{z_i,z_j}^{y_{ij}} (1 - p\prime_{z_i,z_j})^{n_{ij}-y_{ij}} \cdot \prod_{i<j}^{K} (1 - p\prime_{ij})^{\beta-1}}{\prod_{i<j}^{N} \binom{n_{ij}}{y_{ij}} p_{z_i,z_j}^{y_{ij}} (1 - p_{z_i,z_j})^{n_{ij}-y_{ij}} \cdot \prod_{i<j}^{K} (1 - p_{ij})^{\beta-1}}
\end{aligned} \tag{14}
$$

where the allocation vector $\mathbf{z}$ do not change. - - If the delete attempt is selected: - If $K = 1$, the new state is set equal to the current state: $K^{(s+1)} = 1$ - If $K > 1$, we always accept $K^{(s+1)} = K - 1$

```
#vector containing the player list and their block assignement
players_list = data.frame( players = unique(c(data_clean$player1,data_clean$player2)))
n= as.numeric(length(players_list))
K =4

#dataframe containg the pairs and the block assignment
```

```
data_clean = cbind(data_clean, z_player1 = rep(NA, nrow(data_clean)), z_player2 =rep(NA, nrow(data_clean
players_list = cbind(players_list, z_vec = sample(1:K, nrow(players_list), replace = T))
for(i in 1:nrow(players_list)){
  data_clean$z_player1[which(data_clean$player1 == players_list$players[i])] <- players_list$z_vec[i]
  data_clean$z_player2[which(data_clean$player2 == players_list$players[i])] <- players_list$z_vec[i]
}
```

```
get_n_z_ij = function(dataset, K){
  n_z_ij = matrix(NA, nrow=K,ncol=K)
for(i in 1:K){
  for(j in 1:K){
  n_z_ij[i,j] = sum(dataset[which(dataset$z_player1 == i & dataset$z_player2 == j| dataset$z_player1 ==
  }
}
  return(n_z_ij)
}
```

```
get_y_z_ij = function(dataset, K){
  y_z_ij = matrix(NA, nrow=K,ncol=K)
for(i in 1:K){
  for(j in 1:K){
  y_z_ij[i,j] = sum(dataset[which(dataset$z_player1 == i & dataset$z_player2 == j),]$victories,
                    (dataset[which(dataset$z_player2 == i & dataset$z_player1 == j),]$games -dataset[whi
  }
}
  diag(y_z_ij) = diag(y_z_ij)*0.5
  return(y_z_ij)
}

get_n= function(players_list){return(as.vector(table(players_list)))}
n_vec = get_n(players_list$z_vec)
```

```
#it transforms a scalar label k into an indicator vector of length K
#indicator = [K x 1] with entries equal to 0, except indicator[k] == 1
indicator = function(z_i,K){
  #z_i must be a scalar, the label of the block node i belongs
  #K is the number of labels
  indicator <- rep(0, K)
  indicator[z_i] <- 1
  return(indicator)
}
```

```
# N = number of players
# data_clean = dataframe containing the pairs of players(column1,2),the number of games and victories (
# players_list = dataframe containing the list of all players (column1) and their block membership (col

N= nrow(players_list)
#hyperparameters
```

```r
gamma_0=1
beta_0=1

#Fixing K max
K_max=5

set.seed(1605)

#random initialization of K and Z
K_current<-sample.int(n=K_max, size=1)
Z_current<-sample.int(n=K_current, size=N, replace=TRUE)

# set number of steps to discard
burn_in_level<-50000
# Number of steps to retain + to discard
S = 200000+burn_in_level
```

```r
# Initialize structures for chain output

# keeping track of the number of clusters
K_seq=numeric(S)
K_seq[1]<-K_current

#keeping track of the cluster assignment
Z_seq=matrix(nrow=S, ncol=N)
Z_seq[1,]= as.vector(players_list$z_vec)

#keeping track of the matrix p
p_seq=array(0,dim = c(K_max,K_max,S))
p_seq[,,1]= rbeta(K_max*K_max,1,1)

labels_available<-seq(1:K_current)




# we take track of the sequence of the loglik which is just
log_lik_seq<-numeric(S)
```