Time-varying Bradley-Terry model using Barycentric Interpolation

Erik Strumbelj, Blaz Krese

Model

The model is Bradley-Terry but team strength is allowed to vary over time. Latent team strength λ is determined for each of k teams at m different nodes in time. The latent strengths for times between nodes (θ) are interpolated using Barycentric rational interpolation. The k-th team is taken as the reference - its latent strength θ is assumed to be 0 always. The model also includes an additive term for home team advantage:

$$
\begin{aligned}
y_i|\lambda, \Delta_{\text{hta}}, home, away, t &\sim \text{Bernoulli}\left(\frac{1}{1+\exp(-(\theta_{home(i),t(i)} - \theta_{away(i),t(i)} + \Delta_{\text{hta}}))}\right) \\
\theta_{i,t} &= \frac{\sum_{l=1}^{m}(-1)^l \lambda_{i,l}/(t-t_l)}{\sum_{l=1}^{m}(-1)^l/(t-t_l))}, \forall i \neq k \\
\theta_{k,\cdot} &= 0 \\
\lambda_i &\sim N(0,2), \forall i \neq k \\
\Delta_{\text{hta}} &\sim N(0,1),
\end{aligned}
$$

where vectors $home(i)/away(i)$ and $t(i)$ are the indices of the home and away teams and the time of the i-th game, respectively. The prior on λ is based on the competitive balance of the NBA - the worst team still has at least a 5-10% chance of beating the best team, which roughly coresponds to 2-3 difference in latent strength).

Example: NBA basketball

```r
pct_train   <- 0.50
pct_test    <- 0.50 # the two 'pct' combined can't exceed 1.0
n_teams     <- 3    # between 2 and 30
nodes       <- 5
iter_sample <- 1000
iter_warmup <- 200
n_chains    <- 1
seed        <- 1

df <- readRDS("NBA_dataset.rds")
print(head(df))
```

```
##            Date  Season Time   matchup Abbr.x Abbr.y    Prob.x    Prob.y y
## 7893 2013-11-01 2013-14    0 CLE @ CHA    CHA    CLE 0.3981997 0.6018003 1
## 7894 2013-11-01 2013-14    0 NOP @ ORL    ORL    NOP 0.3827751 0.6172249 1
## 7895 2013-11-01 2013-14    0 PHI @ WAS    WAS    PHI 0.8205434 0.1794566 0
## 7896 2013-11-01 2013-14    0 TOR @ ATL    ATL    TOR 0.5965083 0.4034917 1
## 7897 2013-11-01 2013-14    0 MIL @ BOS    BOS    MIL 0.5605012 0.4394988 0
## 7898 2013-11-01 2013-14    0 DAL @ HOU    HOU    DAL 0.7194330 0.2805670 1
##      Object_a Object_b
```

```
## 7893        4         6
## 7894       22        19
## 7895       30        23
## 7896        1        28
## 7897        3        17
## 7898       11         7
```

The dataset consists of 5904 regular season NBA games between 2013-11-01 and 2018-04-11. The columns include the outcome (y), an indexing of the teams (Object), the relative time in days (Time), and home and away win probabilities (Prob) derived from bookmaker odds.

We'll take only the first 3 teams use 0.5 of the observations for training and 0.5 for testing. The number of interpolation nodes is set to 5. The computation time will increase substantially if we use all 30 teams and in particular if we increase the number of nodes.

```r
library(rstan)
```

```
## Warning: package 'rstan' was built under R version 4.0.3

## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)

## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
```

```r
# Train/Test split
df      <- df[df$Object_a <= n_teams & df$Object_b <= n_teams,]
n       <- nrow(df)
n_train <- floor(pct_train * n)
n_test  <- floor(pct_test  * n)
idx     <- sample(1:nrow(df), n_train + n_test, rep = F)
idx_te  <- idx[1:n_test]
idx_tr  <- idx[-c(1:n_test)]

x_train <- df[idx_tr, colnames(df) != "y"]
y_train <- df[idx_tr, colnames(df) == "y"]
x_test  <- df[idx_te, colnames(df) != "y"]
y_test  <- df[idx_te, colnames(df) == "y"]

# Derived data
objs      <- sort(unique(unlist(x_train[c("Object_a", "Object_b")])))
tt        <- sort(unique(rbind(x_train, x_test)$Time))
tt_train  <- sort(unique(x_train$Time))
idx_train <- sapply(x_train$Time, function(x) which(x == tt_train))
idx_test  <- sapply(x_test$Time, function(x) which(x == tt))
t_k       <- seq(min(tt_train), max(tt_train),
                 (max(tt_train) - min(tt_train)) / (nodes + 1)) + pi/10
t_k       <- t_k[c(-1, -length(t_k))]
weights  <- (-1)^(1:length(t_k))

# Prepare data for Stan
stan_data <- list(N_teams = length(objs), n_nodes = nodes,
```

```r
                  t_k = as.array(t_k), w = as.array(weights),
                  N_tt = length(tt), tt = tt, N_tt_train = length(tt_train),
                  tt_train = tt_train, N_train = nrow(x_train),
                  N_test = nrow(x_test),
                  i_train = cbind(x_train$Object_a, x_train$Object_b),
                  i_test = cbind(x_test$Object_a, x_test$Object_b),
                  idx_train = as.array(idx_train),
                  idx_test = as.array(idx_test), y = y_train)

# Compile and sample
sm  <- stan_model(file = "barycentric.stan")
res <- sampling(sm, stan_data, iter = iter_sample,
                warmup = iter_warmup, chains = n_chains, seed = seed,
                control = list(max_treedepth = 20, adapt_delta = 0.8))
```

```
##
## SAMPLING FOR MODEL 'barycentric' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 201 / 1000 [ 20%]  (Sampling)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Sampling)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Sampling)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.862 seconds (Warm-up)
## Chain 1:                1.501 seconds (Sampling)
## Chain 1:                2.363 seconds (Total)
## Chain 1:
```

```r
print(res, pars = c("hta"))
```

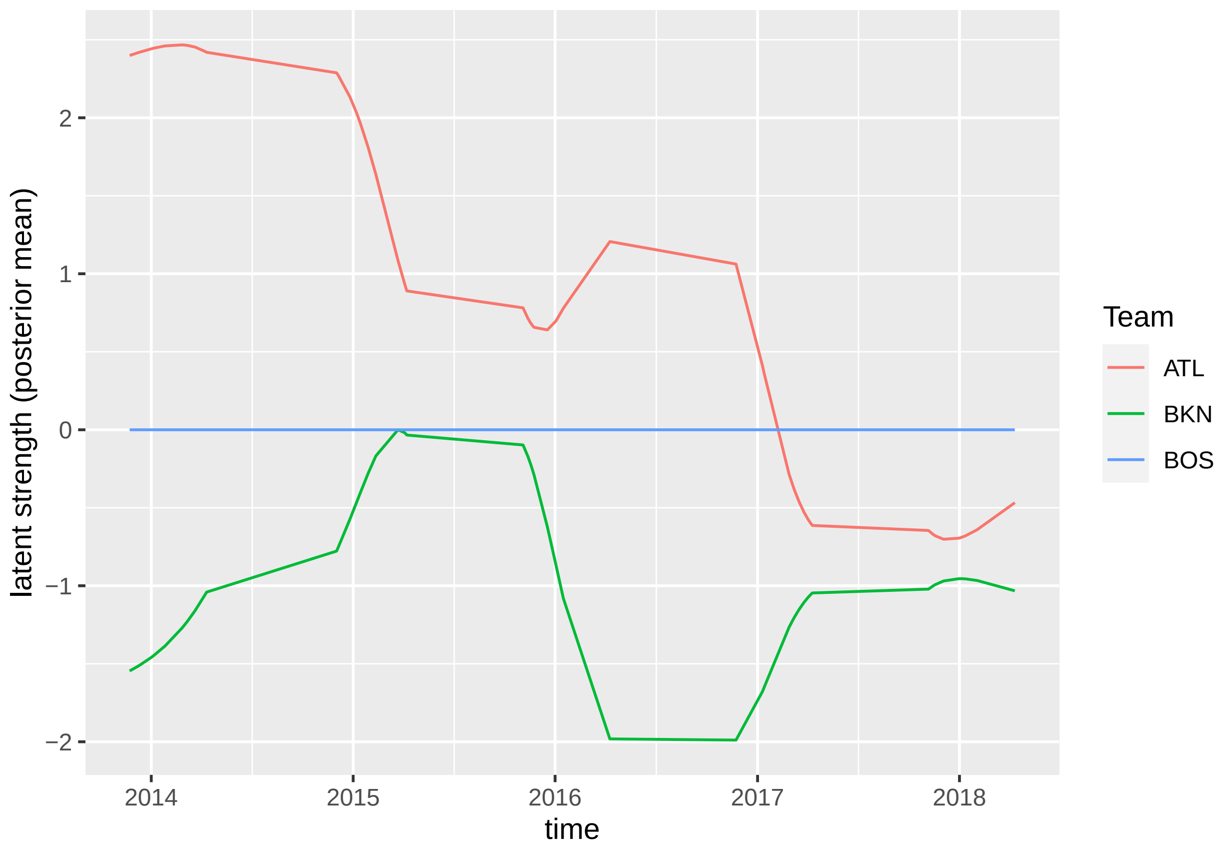
```
## Inference for Stan model: barycentric.
## 1 chains, each with iter=1000; warmup=200; thin=1;
## post-warmup draws per chain=800, total post-warmup draws=800.
##
##      mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## hta -0.66    0.01 0.45 -1.64 -0.96 -0.64 -0.35  0.14   987    1
##
## Samples were drawn using NUTS(diag_e) at Thu Nov 26 00:34:00 2020.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```r
samples <- extract(res)
```

```r
# visualize over time
library(ggplot2)

th_out <- samples$th
p      <- colMeans(samples$p)
hta    <- samples$hta
dates  <- as.Date(df$Date[match(tt, df$Time)])
x <- NULL
for (i in seq(length(objs))) {
  tmp <- t(th_out[,i,])   # No ordering, because tt is sorted
  x <- rbind(x, data.frame(Team = df$Abbr.x[match(i, df$Object_a)],
                           Time = dates,
                           Theta = rowMeans(tmp)))
}

ggplot(x, aes(x = Time, y = Theta, group = Team, colour = Team)) +
  geom_line() + ylab("latent strength (posterior mean)") +
  xlab("time")
```



Model code

```r
cat(readLines('barycentric.stan'), sep = '\n')
```

```
functions {
```

```
  real calc_th(vector lam, real ttime, vector tk, vector w) {

    int N = num_elements(lam);

    vector[N] a1 = rows_dot_product(w, lam);
    vector[N] a2 = 1 ./ (ttime - tk);
    return sum(rows_dot_product(a1, a2)) / sum(rows_dot_product(w, a2));

  }
}


data {
  int<lower=2> N_teams;     // number of teams
  int<lower=1> N_train;     // number of train games
  int<lower=1> N_test;      // number of test games

  int<lower=1> n_nodes;     // number of nodes
  vector[n_nodes] t_k;      // node times
  vector[n_nodes] w;        // weights

  int<lower=1> N_tt;        // unique sorted concat(train time, test time ) points
  real tt[N_tt];

  int<lower=1> N_tt_train;  // unique sorted train time points
  real tt_train[N_tt_train];

  int i_train[N_train,2];   // team indices
  int i_test[N_test,2];
  int idx_train[N_train];
  int idx_test[N_test];

  int y[N_train];           // bernoulli observations
}

parameters {
  vector[n_nodes] lambda[N_teams - 1];
  real hta;
}

transformed parameters {
  vector[N_tt_train] theta[N_teams];

  theta[N_teams] = rep_vector(0, N_tt_train);
  for (i in 1:(N_teams - 1)) {
    for (j in 1:N_tt_train) {
      theta[i][j] = calc_th(lambda[i], tt_train[j], t_k, w);
    }
  }

}

model {
```

```
  for (i in 1:(N_teams - 1)) {
    lambda[i] ~ normal(0, 2);
  }

  hta ~ normal(0, 1);

  for (i in 1:N_train) {
    y[i] ~ bernoulli(inv_logit(theta[i_train[i, 1]][idx_train[i]] + hta
                         - theta[i_train[i, 2]][idx_train[i]]));
  }

}

generated quantities {

  vector[N_tt] th[N_teams];
  vector[N_test] p;
  vector[N_train] log_lik;

  th[N_teams] = rep_vector(0, N_tt);

  for (i in 1:(N_teams - 1)) {
    for (j in 1:N_tt) {
      th[i][j] = calc_th(lambda[i], tt[j], t_k, w);
    }
  }

  for (i in 1:N_test) {
    p[i] = inv_logit(th[i_test[i, 1]][idx_test[i]] + hta
                - th[i_test[i, 2]][idx_test[i]]);
  }

  for (i in 1:N_train) {
    log_lik[i] = bernoulli_lpmf(y[i] | inv_logit(theta[i_train[i, 1]][idx_train[i]]
                                               + hta
                                               - theta[i_train[i, 2]][idx_train[i]]));
  }

}
```