BDA Project Work

Shaun McNaughton & Paul Sasieta

Introduction

Sports prediction is a branch of statistics that has been growing in the last decade. This growth is related not only to the large monetary amounts involved in betting but also to the access to information that could mean a competitive advantage for certain teams. Due to the diversity that exists between sports, the projects will be focused to study doubles tournmanets of tennis. More particularly, we will focus on Wimbledon 2019. The idea of the project is to use rating of each individual player together with set difference in previous matches of the tournament to estimate the outcome of future matches. The primary aim is to describe two models that enable the prediction of tennis events, but also to predict who wins the final. To establish a consistent approach, the methodology followed can be divided into different steps: data collection, model application and convergence and performance analysis.

As you probably already know, tennis is a racket sport that can be played individually (singles) or between two teams of two players each (doubles). A tennis match is composed of points, games, and sets. A set consists of a number of games (a minimum of six), which in turn each consist of points. A set is won by the first side to win 6 games, with a margin of at least 2 games over the other side (e.g. 6–3 or 7–5). If the set is tied at six games each, a tie-break is usually played to decide the set. A match is won when a player or a doubles team has won the majority of the prescribed number of sets. Matches employ either a best-of-three or best-of-five set format.

In professional tennis, the four Grand Slam tournaments are particularly popular: the Australian Open played on hard courts, the French Open played on red clay courts, Wimbledon played on grass courts, and the US Open also played in hard courts. These Grand Slams are organized as single-elimination tournaments, with competitors being eliminated after a single loss and Men's singles and doubles matches following the best-of-five format. The brackets are seeded according to a recognised ranking system, in order to keep the best players in the field from facing each other until as late in the tournament as possible.

We will focus on the championship of Wimbledon, the oldest and most prestigious tennis tournament in the world. The championship has five main events: gentlemen's singles, ladies' singles, gentlemen's doubles, ladies' doubles and mixed doubles. We will be using data from gentlemen's doubles Wimbledon 2019 tournament.

Aims

The aim of this is to predict the outcome of the finals men's doubles match using results from the tournament and a team rating score, a representation of the skill level of the doubles team.

While we are primarily interested in the outcome of a match (win/lose), knowing if a player/team completely outclasses another player/team is critical in understanding the variability of our prediction. A better player/team is not only expected to win against a weaker player/team, but is expected to win by a larger margin (3-0).

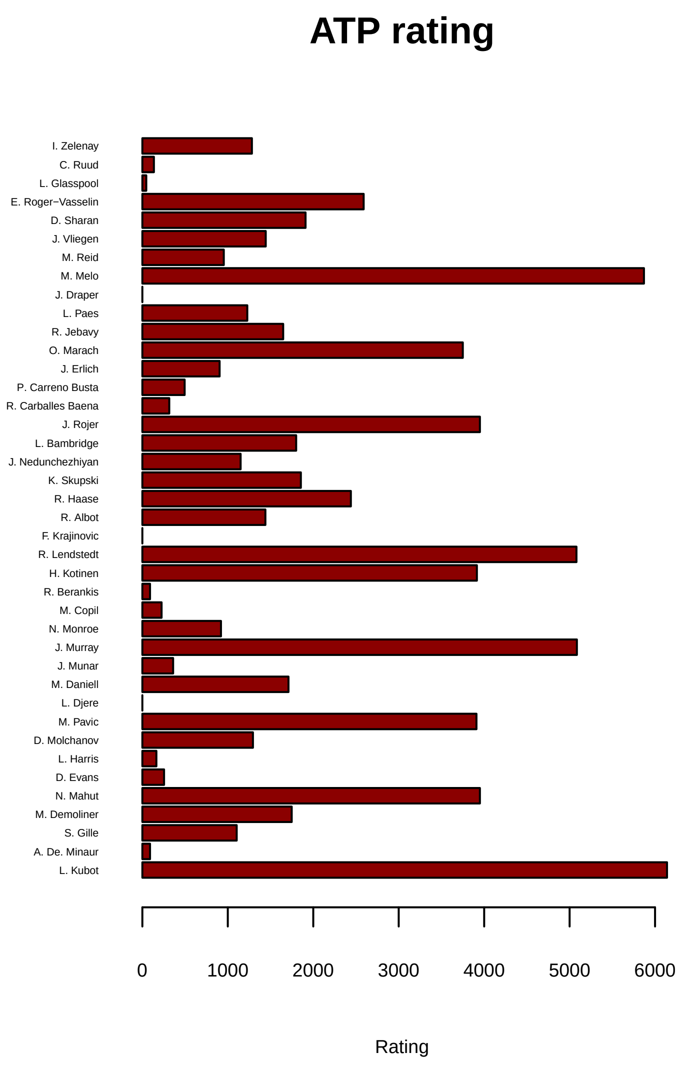
Winning in close match (3-2), typically has implications for the prediction of following match, as these matches can be long and can also be physically and mentally draining. This leads to a time effect where one close match can have a negative follow on effects on subsequent matches. While our model does not take this into account, it could be extended on if we wish to construct a full tournament prediction.

Datasets

The player rating dataset comes from the Association of Tennis Professional (ATP) website on players ratings. Specifically we look at the men's doubles player rating in the week prior to Wimbledon 2019.

The doubles player rating is derived from the amount of tournament points accumulated from a 12 month rolling window. Winning an associated ATP event will grant tournament points to the player, with the winners taking the largest share of the tournament points and the amount decreases the lower placement the team reaches. The total amount of tournament points won are determined by the ATP and the four Grand Slam Tournaments are heavily weighted in the points system.

The tournament match result data comes from Wimbledon's website. This details who is on which team, which team won and by how much. As Wimbledon is run in an elimination round format, teams who lose on the first round only play once, where the teams that reach the final play 5 games. This has the added consequence that the teams that play fewer games rely on the initial team rating more than the teams that win.



Transformation of the data

The models we are fitting are based on Andrew Gelman's model to estimate the world cup. It uses the signed square root of a point differential. Hence, the first step is to transform the data and derive the set difference of each match.

$$sqrt\_dif[i] = 2 * (step(dif[i]) - .5)\sqrt{fabs(dif[i])}$$

where fabs() is the absolute value function and step() the 0-1 step function. The square root models that when the game is less close, it becomes more impredictable.

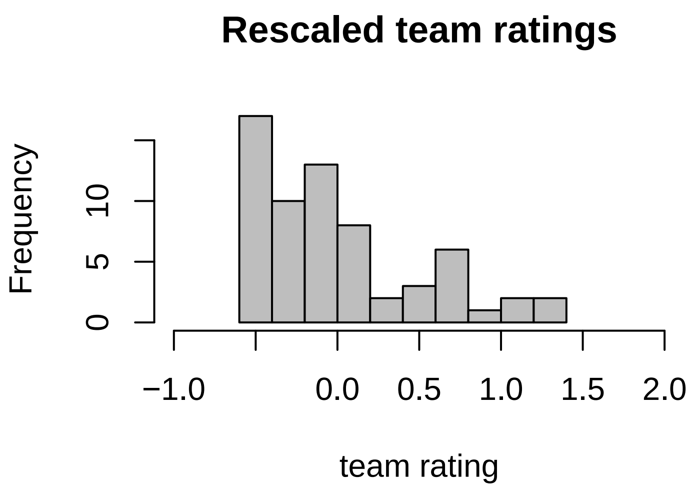
Our prior consists of individual player ratings, we take the sum of the player's individual ratings to derive a team rating.

$$rating(team) = rating(player1) + rating(player2)$$

Team ratings represent values from 0 until 12,010 and are then scaled into a $N(0, 0.5)$ distibution, to get an indicative measure of team skills. It is this scaled rating that becomes the prior.

$$team\_ranks = \frac{team\_ranks - m}{2\sigma}$$

where m is the mean and σ is the standard deviation of the team rank.



We are now ready to fit our models.

Model 1

We first model the skill of each team, simply doing

$$a_i \sim N(b * team\_rank[i]\ ,\ \sigma_a)$$

where b and σ_a play parameter role in our model. Remark that we have chosen $sigma_a$ to be independent of the team. Also, the lower the $sigma_a$ value is the more representative the team rating is.

On the other hand, if team 1 and team 2 are playing against each other on match number i, we have the respective values of $sqrt\_dif$ representing set difference on that match. This set diferrence is tightly related to skill diference between the teams, so we used the following model:

$$sqrt\_dif[i] \sim t_7(a[team1] - a[team2], \sigma_y)$$

where σ_y plays the role of a parameter in the model. The Stan code is the following.

```
parameters {
  real b;
  real<lower=0> sigma_a;
  real<lower=0> sigma_y;
  vector[nteams] a;
}
model {
  a ~ normal(b*prior_score, sigma_a);
  for (i in 1:ngames)
    sqrt_dif[i] ~ student_t(df, a[team1[i]]-a[team2[i]], sigma_y);
}
```

In conclusion, the model has three real parameters b,σ_a and σ_y and a vector parameter a of length 64. The idea now is to predict the winner of the final match of the tournament using the obtained information. The team final was played between teams number 13 and 3. We will estimate the skill difference between these two teams sampling from their respective normal distribution modeling skill and computing the difference. Then, we will reconvert the value of those differences undoing the transformations. This is computed in the generated quantities block.

```
generated quantities {
  real team1rank;
  real team2rank;
  real rankdif;
  vector[64] log_lik;

  team1rank = normal_rng(b*prior_score[13],sigma_a);
  team2rank = normal_rng(b*prior_score[3],sigma_a);
  rankdif = team1rank - team2rank;
}
```

We now fit this model to our data.

```r
# Model Data
nteams = length(W2019teamrank$Team) #Number of teams
ngames = 62                         #Number of games
prior_score = ranks                 #Team ratings
team1 = W2019teamresults$Team1      #Team1 index
team2 = W2019teamresults$Team2      #Team2 index
score1 = W2019teamresults$SetT1     #Team1 number of sets
score2 = W2019teamresults$SetT2     #Team2 number of sets
df = 7
```

```
data <- list(nteams=nteams, ngames = ngames , team1 = team1 , score1 = score1 ,
             team2=team2 , score2=score2 , prior_score=prior_score,df=df)
```

```
fit_bern1 <- stan(file='codeM1.stan', data=data,iter=5000)
fit_bern1a <- stan(file='codeM1A.stan',data=data,iter=5000)
fit_bern1b <- stan(file='codeM1B.stan',data=data,iter=5000)
```

The obtained results by stan are $b = 0.26$, $\sigma_a = 0.88$, $\sigma_y = 0.83$, $a[3] = 0.20$, $a[13] = 0.23$ and $rankdif = -0.21$. Therefore, we assume that the signed squared root of set difference of the finals match satisfy

$$sqrt\_dif\_final \sim t_7(-0.21, 0.83)$$

Sampling 10,000 draws from this distribution, computing the mean of those samples and then undoing the transformation translates in a expected value of -0.16 set difference in the final match. This means that the final is expected do be a close game and Team 2 is expected to win it. Hence, the point prediction would be a 2-3 loss to Team 1.

We now proceed to do a convergence analysis based on $\hat{R}$. We will be using the latest version of $\hat{R}$, which is an improved version of the traditional Rhat presented in Eq. 11.4 in BDA3.

```
print(fit_bern1,pars=c("b","sigma_a","sigma_y","a[3]","a[13]","rankdif"))
```

```
## Inference for Stan model: codeM1.
## 4 chains, each with iter=5000; warmup=2500; thin=1;
## post-warmup draws per chain=2500, total post-warmup draws=10000.
##
##          mean se_mean   sd  2.5%   25%   50%  75% 97.5% n_eff Rhat
## b        0.28    0.02 0.34 -0.41  0.05  0.28 0.51  0.94   350 1.01
## sigma_a  0.88    0.00 0.09  0.69  0.83  0.90 0.95  1.00   433 1.02
## sigma_y  0.84    0.01 0.10  0.62  0.78  0.86 0.93  0.99   417 1.02
## a[3]     0.23    0.03 0.68 -1.10 -0.23  0.22 0.70  1.57   617 1.00
## a[13]    0.25    0.02 0.69 -1.14 -0.21  0.26 0.71  1.56   759 1.01
## rankdif -0.20    0.01 1.28 -2.73 -1.06 -0.20 0.64  2.33  7562 1.00
##
## Samples were drawn using NUTS(diag_e) at Sun Dec  8 15:24:28 2019.
## 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).
```

As we can see in the table, the values of $\hat{R}$ obtained are very close to 1 so we can conclude that convergence has occurred.

Model performance

We did not initially specify any prior for the parameters in any of the models. In Stan, not specifiying a prior is equivalent to specifiying an uniform (0,1) prior. In order to study the behaviour of the inference under different priors, we will add weakly informative priors for b,σ_a and σ_y.

So we test a few weakly informative priors to see if it affects the estimates.

Gaussian (0,1) priors for sigma.

```r
# Gaussian (0,1) priors
print(fit_bern1a,pars=c("b","sigma_a","sigma_y","a[3]","a[13]","rankdif"))
```

```
## Inference for Stan model: codeM1A.
## 4 chains, each with iter=5000; warmup=2500; thin=1;
## post-warmup draws per chain=2500, total post-warmup draws=10000.
##
##          mean se_mean   sd  2.5%   25%   50%  75% 97.5% n_eff Rhat
## b        0.30    0.01 0.35 -0.40  0.06  0.30 0.53  0.99  4378 1.00
## sigma_a  0.96    0.01 0.22  0.49  0.82  0.96 1.10  1.37   796 1.01
## sigma_y  0.88    0.01 0.23  0.47  0.72  0.86 1.02  1.38   826 1.01
## a[3]     0.24    0.01 0.69 -1.11 -0.23  0.24 0.68  1.62  6942 1.00
## a[13]    0.30    0.01 0.74 -1.14 -0.18  0.29 0.79  1.78  4983 1.00
## rankdif -0.21    0.01 1.41 -3.02 -1.11 -0.21 0.69  2.66  9457 1.00
##
## Samples were drawn using NUTS(diag_e) at Sun Dec  8 15:27:43 2019.
## 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).
```

Student t (3,0,1) priors for sigma.

```r
# Student (3,0,1) priors
print(fit_bern1b,pars=c("b","sigma_a","sigma_y","a[3]","a[13]","rankdif"))
```

```
## Inference for Stan model: codeM1B.
## 4 chains, each with iter=5000; warmup=2500; thin=1;
## post-warmup draws per chain=2500, total post-warmup draws=10000.
##
##          mean se_mean   sd  2.5%   25%   50%  75% 97.5% n_eff Rhat
## b        0.30    0.01 0.36 -0.40  0.07  0.31 0.53  1.01  4530 1.00
## sigma_a  0.94    0.01 0.25  0.35  0.80  0.95 1.10  1.40   441 1.01
## sigma_y  0.89    0.01 0.24  0.47  0.73  0.87 1.04  1.43   541 1.00
## a[3]     0.23    0.01 0.68 -1.12 -0.23  0.24 0.68  1.58  6322 1.00
## a[13]    0.31    0.01 0.72 -1.08 -0.16  0.29 0.77  1.74  4439 1.00
## rankdif -0.20    0.01 1.38 -3.00 -1.05 -0.20 0.67  2.58  9895 1.00
##
## Samples were drawn using NUTS(diag_e) at Sun Dec  8 15:30:33 2019.
## 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).
```

The point estimates do not change much from modifying the sigma priors. Therefore it does not appear to matter strongly which prior we choose.

Model 2

In the second model we came up with a hypothesis that teams who share the same nationality, have some latent performance bonus that is not captured by team rating alone. As teams can be formed and broken apart fairly regularly. Tournament scores can instead reflect performance from multiple teams that a player might have had in the past. Players who share the same nationality, may perform differently than those who do not share similar cultural habits. To model this, we extracted the data from the Wimbledon's men's doubles teams and coded them as a true/false indicator.

To add this effect to the model this we fit the same model, but now use two parameters, sigma_a1 and sigma_a2. These reflect the spread of our ranking were teams who have same or different nationalities. This goes into the idea that the current team ranking model fails to capture some element of having a shared nationality.

```
model {
  a ~ normal(b*prior_score,sigma_a1*nationality + (1 - nationality)*sigma_a2);
  for (i in 1:ngames)
    sqrt_dif[i] ~ student_t(df, a[team1[i]]-a[team2[i]], sigma_y);
}
```

After fitting the model and checking for convergence. (Note Stan fails to run this model in knitr - so these results are copied from R)

```
fit_bern2 <- stan(file='codeM2.stan',data=data,iter=5000,control = list(adapt_delta = 0.90))
fit_bern2a<- stan(file='codeM2A.stan',data=data,iter=5000,control = list(adapt_delta = 0.90))
```

```
Inference for Stan model: codeM2.
4 chains, each with iter=5000; warmup=2500; thin=1;
post-warmup draws per chain=2500, total post-warmup draws=10000.

          mean se_mean   sd  2.5%   25%   50%  75% 97.5% n_eff Rhat
b         0.36    0.01 0.39 -0.41  0.10  0.36 0.61  1.09  4529 1.00
sigma_a1  1.09    0.01 0.31  0.52  0.89  1.08 1.28  1.74  1033 1.01
sigma_a2  0.89    0.01 0.29  0.29  0.71  0.89 1.07  1.45   596 1.01
sigma_y   0.91    0.01 0.25  0.46  0.74  0.89 1.06  1.43   665 1.01
a[3]      0.21    0.01 0.75 -1.25 -0.29  0.19 0.69  1.71  6582 1.00
a[13]     0.36    0.01 0.79 -1.21 -0.16  0.34 0.88  1.93  5995 1.00
rankdif  -0.26    0.02 1.67 -3.65 -1.30 -0.26 0.80  3.03  9627 1.00

Samples were drawn using NUTS(diag_e) at Sun Dec  8 15:20:54 2019.
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).
```

And Model 2 with N(0,1) priors for sigma_a1, sigma_a2 and sigma_y.

```
## Inference for Stan model: codeM2.
## 4 chains, each with iter=5000; warmup=2500; thin=1;
## post-warmup draws per chain=2500, total post-warmup draws=10000.
##
##           mean se_mean   sd  2.5%   25%   50%  75% 97.5% n_eff Rhat
## b         0.36    0.01 0.39 -0.41  0.10  0.36 0.61  1.09  4529 1.00
## sigma_a1  1.09    0.01 0.31  0.52  0.89  1.08 1.28  1.74  1033 1.01
## sigma_a2  0.89    0.01 0.29  0.29  0.71  0.89 1.07  1.45   596 1.01
## sigma_y   0.91    0.01 0.25  0.46  0.74  0.89 1.06  1.43   665 1.01
## a[3]      0.21    0.01 0.75 -1.25 -0.29  0.19 0.69  1.71  6582 1.00
## a[13]     0.36    0.01 0.79 -1.21 -0.16  0.34 0.88  1.93  5995 1.00
## rankdif  -0.26    0.02 1.67 -3.65 -1.30 -0.26 0.80  3.03  9627 1.00
##
## Samples were drawn using NUTS(diag_e) at Sun Dec  8 15:20:54 2019.
## 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).

## > print(fit_bern2A,pars=c("b","sigma_a1","sigma_a2","sigma_y","a[3]","a[13]","rankdif"))
## Inference for Stan model: codeM2.
## 4 chains, each with iter=5000; warmup=2500; thin=1;
## post-warmup draws per chain=2500, total post-warmup draws=10000.
##
##           mean se_mean   sd  2.5%   25%   50%  75% 97.5% n_eff Rhat
## b         0.37    0.01 0.36 -0.37  0.13  0.38 0.61  1.07  3716 1.00
## sigma_a1  1.00    0.01 0.28  0.46  0.81  0.99 1.19  1.56  1025 1.01
## sigma_a2  0.80    0.02 0.29  0.12  0.62  0.82 0.99  1.32   256 1.02
## sigma_y   0.94    0.01 0.24  0.51  0.78  0.94 1.10  1.44   634 1.01
## a[3]      0.28    0.02 0.74 -1.18 -0.22  0.27 0.77  1.71  1736 1.00
## a[13]     0.29    0.01 0.73 -1.13 -0.19  0.26 0.78  1.77  6177 1.00
## rankdif  -0.25    0.02 1.50 -3.25 -1.15 -0.24 0.67  2.75  9277 1.00
##
## Samples were drawn using NUTS(diag_e) at Sun Dec  8 16:52:43 2019.
## 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).
```

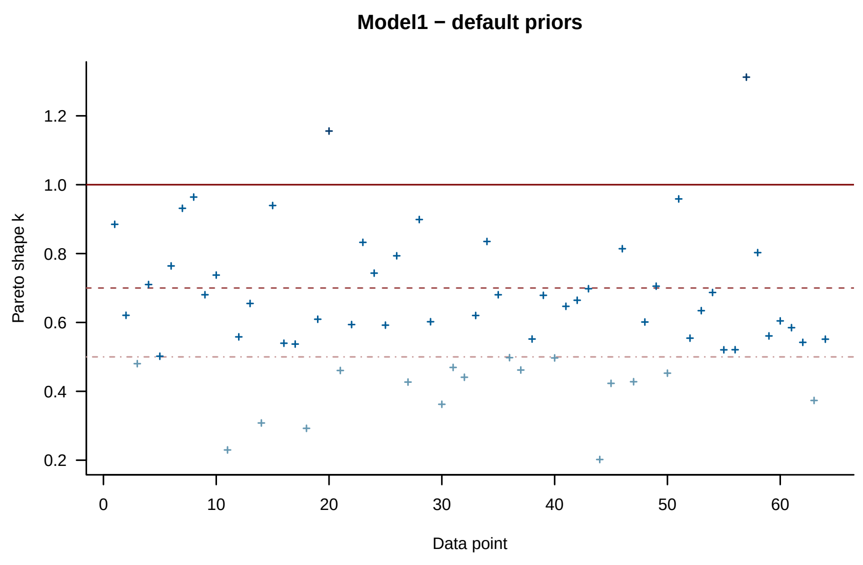
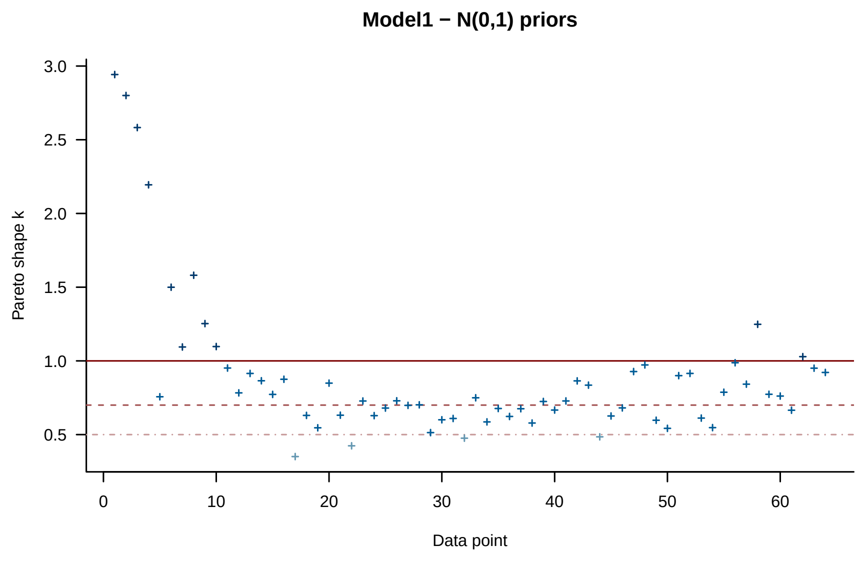
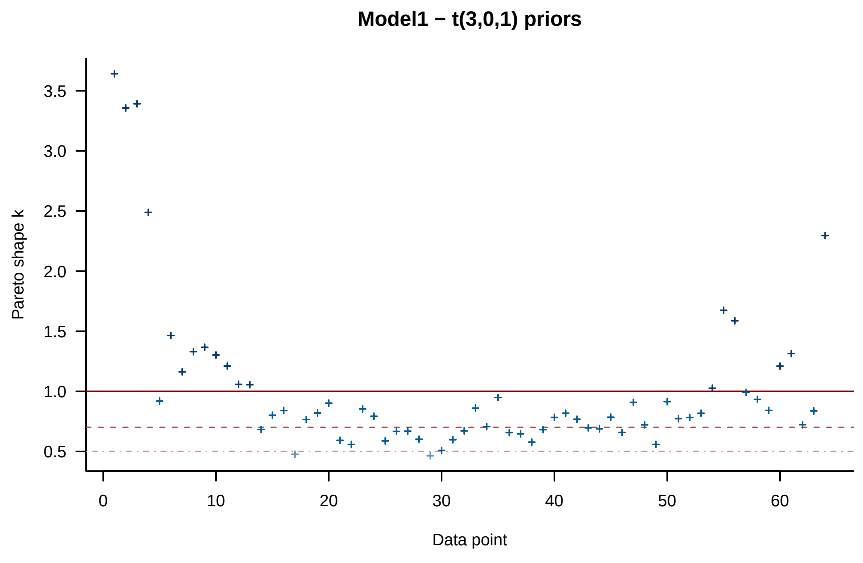
The results show that sigma_a1 and sigma_a2 estimates are similar, but not exactly the same. However, the posterior distribution overlaps heavily so there is not enough evidence to suggest that this model should perform any different to the simpler model

Model comparison and performance

The question of which model is better naturally arises. We will be using the statistical approach of PSIS-LOO eldp values and $\hat{k}$ to determine wich of the models performs better. The loo function is being used to obtain that information.

For the first model, the obtained values are the following.

```r
par(mfrow=(c(3,1)))
ll_m1_mat <- extract_log_lik(fit_bern1, parameter_name = "log_lik")
fit_m1_loo <- loo(ll_m1_mat)
plot(fit_m1_loo,main="Model1 - default priors")
ll_m1a_mat <- extract_log_lik(fit_bern1a, parameter_name = "log_lik")
fit_m1a_loo <- loo(ll_m1a_mat)
plot(fit_m1a_loo,main="Model1 - N(0,1) priors")
ll_m1b_mat <- extract_log_lik(fit_bern1b, parameter_name = "log_lik")
fit_m1b_loo <- loo(ll_m1b_mat)
plot(fit_m1b_loo,main="Model1 - t(3,0,1) priors")
```


Model1 − default priors
Pareto shape k
1.2
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50
60
Data point

Model1 − N(0,1) priors
Pareto shape k
3.0
2.5
2.0
1.5
1.0
0.5
0
10
20
30
40
50
60
Data point

Model1 − t(3,0,1) priors
Pareto shape k
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
10
20
30
40
50
60
Data point

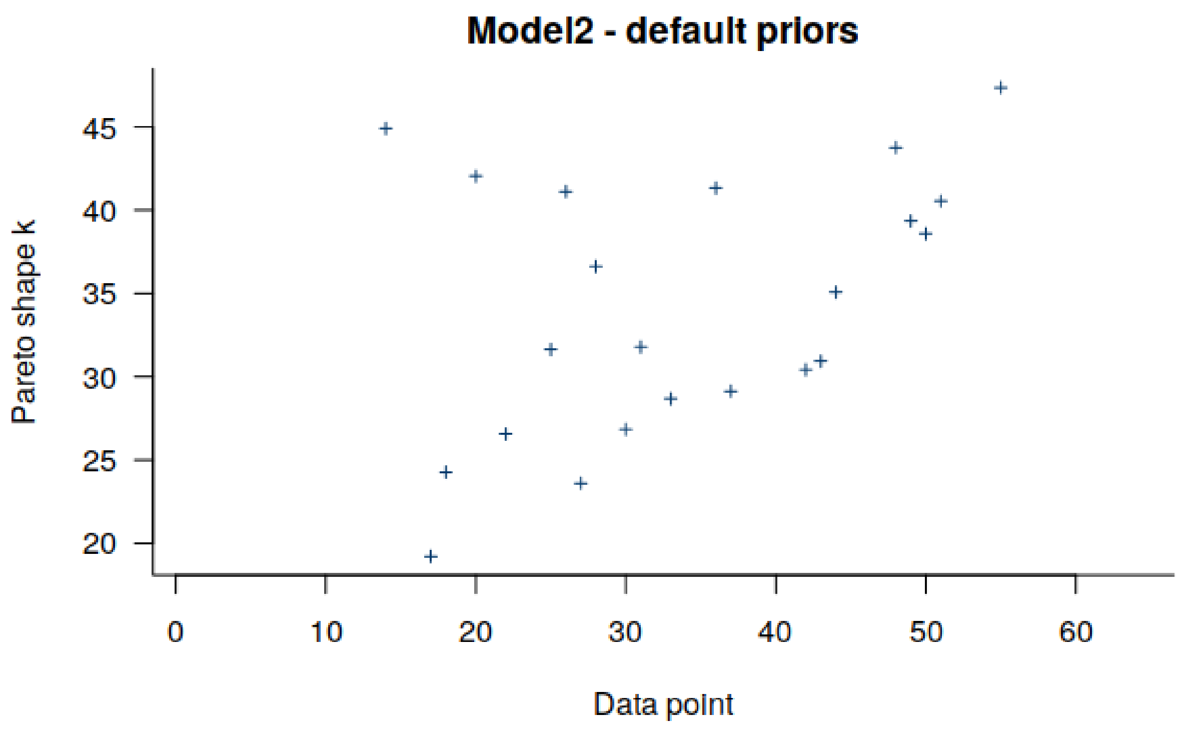
As we can see, most of the values of $\hat{k}$ for Model 1 are above 0.7 which translates on the values of the PSIS-LOO estimates not being reliable at all.

```r
Model1Loo <- data.frame("Model1 Default"=fit_m1_loo$p_loo,"Model1 N(0,1)"=fit_m1a_loo$p_loo,
                   "Model1 T(3,0,1)"=fit_m1b_loo$p_loo)
kable(Model1Loo)
```

Model1.Default	Model1.N.0.1.	Model1.T.3.0.1.
38.40985	79.1337	104.9666

Given a choice between the priors, it would appear that the default prior also provides a better predictive model than the other priors. So while the priors do not appear to have much impact, it does seem to suggest that the prior does play a substantial element in the actual predictive power.

```r
ll_m2_mat <- extract_log_lik(fit_bern2, parameter_name = "log_lik")
fit_m2_loo <- loo(ll_m2_mat)
#print(fit_m2_loo)
plot(fit_m2_loo,main="Model2 - default priors")
```



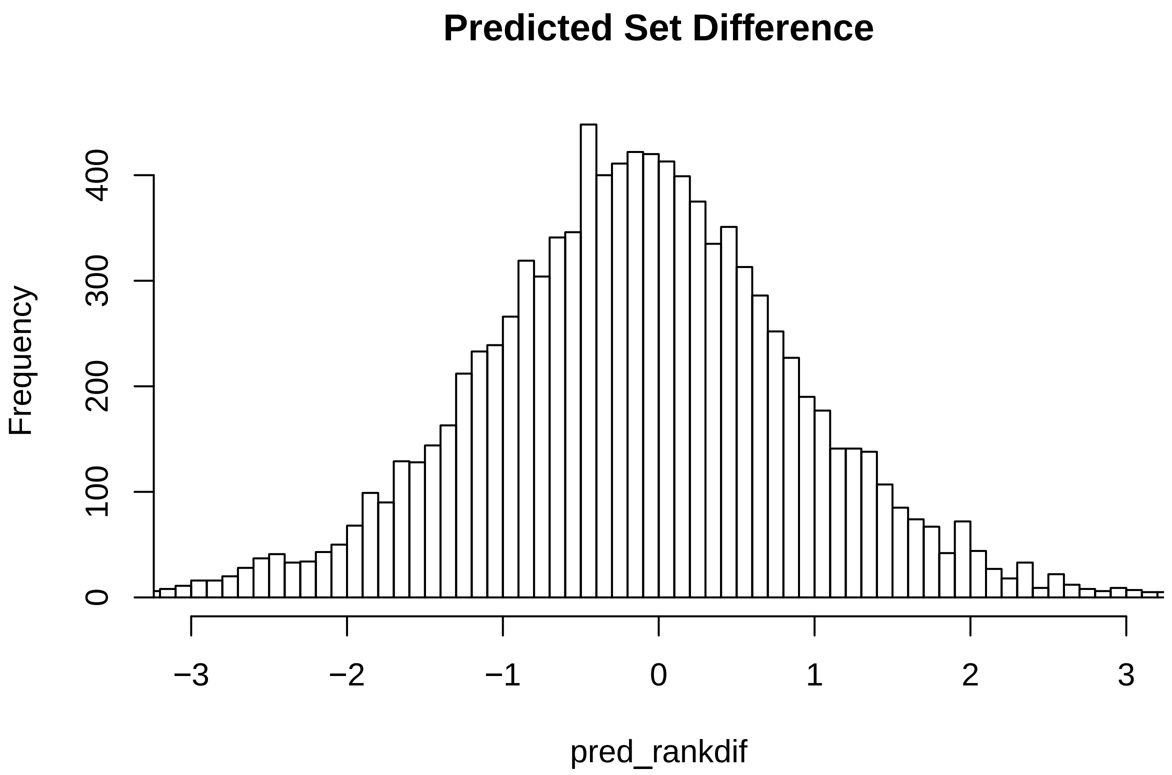
For Model 2, $\hat{k}$ are through the roof, which would indicate that there is something wrong with the derivation.

Results

Based on the first model with uniform priors, we look at the prediction of the final match and the posterior distribution of that prediction.

```r
# Finals prediction code #
# Model 1 #
pred_fit <- extract(fit_bern1)
rankdif_fit <- get_posterior_mean(fit_bern1,par=c("rankdif"))[1]
sigma_y_fit <- get_posterior_mean(fit_bern1,par=c("sigma_y"))[1]
# a[13] - a[3] #
pred_draw <- rt(10000,7)

pred_rankdif <- rankdif_fit + sqrt(sigma_y_fit)*pred_draw
sqrt_dif_find_ptpred <- ((abs(mean(pred_rankdif)) * 2)^2)* sign(mean(pred_rankdif))
sqrt_dif_find <- ((abs(pred_rankdif) * 2)^2)* sign(pred_rankdif)
hist(pred_rankdif,xlim=c(-3,3),breaks=100,main="Predicted Set Difference")
```



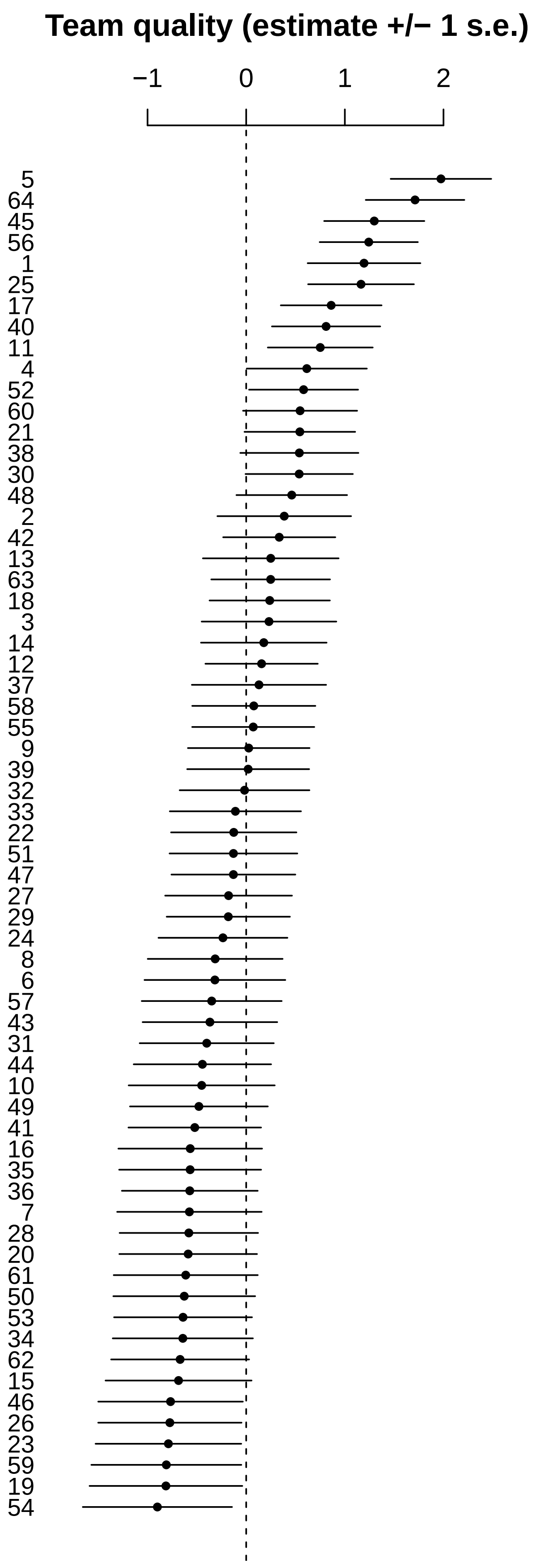
```r
print(fit_bern1,pars=c("rankdif"))
```

```
## Inference for Stan model: codeM1.
## 4 chains, each with iter=5000; warmup=2500; thin=1;
## post-warmup draws per chain=2500, total post-warmup draws=10000.
##
##          mean se_mean   sd  2.5%   25%  50%  75% 97.5% n_eff Rhat
## rankdif -0.2    0.01 1.28 -2.73 -1.06 -0.2 0.64  2.33  7562    1
##
## Samples were drawn using NUTS(diag_e) at Sun Dec  8 15:24:28 2019.
## 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).
```

We can also look at the team quality estimate for teams to make sure that they align roughly with the ATP ranking.

```r
teams <- W2019teamrank$Team
teamsRank <- W2019teamrank$TeamRank

# Looking at the team quality estimate
fit_extract <- extract(fit_bern1)
a_sims <- fit_extract$a
a_hat <- colMeans(a_sims)
a_se <- sqrt(colVars(a_sims))
# Appending the variables to a
teams.append <- data.frame(a_hat=a_hat,a_se=a_se,teams,teamsRank)
teams.sort <- teams.append[order(a_hat),]
library ("arm")
# Sorted by estimated team rank #
coefplot (teams.sort$a_hat, teams.sort$a_se, CI=1, varnames=teams.sort$teamsRank,
          main="Team quality (estimate +/- 1 s.e.)\n", cex.var=.9, mar=c(0,4,5.1,2))
```


Team quality (estimate +/− 1 s.e.)
−1
0
1
2
5
64
45
56
1
25
17
40
11
4
52
60
21
38
30
48
2
42
13
63
18
3
14
12
37
58
55
9
39
32
33
22
51
47
27
29
24
8
6
57
43
31
44
10
49
41
16
35
36
7
28
20
61
50
53
34
62
15
46
26
23
59
19
54

Conclusions

To sum up, we have analyzed two different models to predict the outcome of the Wimbledon 2019 couples final match. Even though the obtained $\hat{k}$ values are high in both cases, we conclude that the first model is more appropiate for predictions. We must take into account the small size of the data set, consisting only of 63 matches. Therefore, using a complex model with a huge amount of parameters is infeasible. For example, we thought about using one σ_a per team but convergence was not obtained. Overparamatrization, which refers to the number of parameters exceeding the size of the dataset, is a factor we have been dealing with. We think this is the reason a more simple model rather than the hierarchicality of the second model works better in this particular case. Remark that there is a uncertainty inherent in the problem, the sport would not be entretaining if we could predict it. This is one strong reason to understand why the models are not working extremely good. However, our intuition that we could estimate team abilities by modeling set differentials seems to be correct.

The model does not add a lot of predictive power for most individual games, but it can rank the teams. The key is that each team plays multiple games. So for the purpose of estimating team ability, it's ok if the model does not predict individual games well. We think that the obtained ratings of the teams before the final match are more meaninful than the ATP prior ratings.

About prior sensitivity, remark tha the estimates do not change much from modifying the sigma prior. There are several improvements that could be added to obtain better results. The prior rating of each team is computed just adding the rating of each individual. We believe that there are several factors that influence that prior rating and if atached the predictiong would improve. For example, if some player has some kind of little injury or if it is clearly out of shape are factors our rating system does not consider. The quality of the prior rating of the teams directly and notoriously affect prediction quality. It would also be interesting to try other sports, maybe considering a tournament with more matches. The framework can in principle be adapted to a wide range of sports domains. for example soccer world championship. Another approach to explore in the future is a Knowledge-based system. It is important to understand that each tennis match haves according to a particular environment. Therefore, a better prediction model should include particular features of the match game, such as the importance of the game. One possibility would be to define time-specific parameters, in order to account for periods of variable form of the teams (including injuries, shape, etc.). Moreover, prior information can be included at various level in the model, perhaps in the form of expert opinion about the strength of each team. Future work in this area also includes the use of different metrics for evaluating the quality of the result.

Appendix

Stan Code

Model 1 - Default priors

```
data {
  int nteams;
  int ngames;
  vector[nteams] prior_score;
  int team1[ngames];
  int team2[ngames];
  vector[ngames] score1;
  vector[ngames] score2;
  real df;
}
transformed data {
  vector[ngames] dif;
  vector[ngames] sqrt_dif;
  dif =  score1 - score2;
  for (i in 1:ngames)
    sqrt_dif[i] = 2*(step(dif[i]) - .5)*sqrt(fabs(dif[i]));
}

parameters {
  real b;
  real<lower=0> sigma_a;
  real<lower=0> sigma_y;
  vector[nteams] a;
}
model {
  sigma_a ~ uniform(0,1); // Prior for sigma_a
  sigma_y ~ uniform(0,1); // Prior for sigma_y
  a ~ normal(b*prior_score, sigma_a);
  for (i in 1:ngames)
    sqrt_dif[i] ~ student_t(df, a[team1[i]]-a[team2[i]], sigma_y);
}

generated quantities {
real team1rank;
real team2rank;
real rankdif;
vector[64] log_lik;

// Predicting the Finals consisting of team 13 vs. team 3.
team1rank = normal_rng(b*prior_score[13],sigma_a);
team2rank = normal_rng(b*prior_score[3],sigma_a);
rankdif = team1rank - team2rank;

for (i in 1:64)
log_lik[i] = normal_lpdf(a[i] | prior_score[i],sigma_a);
}
```

Model 1 - N(0,1) priors

```
data {
  int nteams;
  int ngames;
  vector[nteams] prior_score;
  int team1[ngames];
  int team2[ngames];
  vector[ngames] score1;
  vector[ngames] score2;
  real df;
}
transformed data {
  vector[ngames] dif;
  vector[ngames] sqrt_dif;
  dif =  score1 - score2;
  for (i in 1:ngames)
    sqrt_dif[i] = 2*(step(dif[i]) - .5)*sqrt(fabs(dif[i]));
}

parameters {
  real b;
  real<lower=0> sigma_a;
  real<lower=0> sigma_y;
  vector[nteams] a;
}
model {
  sigma_a ~ normal(0,1); // Prior for sigma_a
  sigma_y ~ normal(0,1); // Prior for sigma_y
  a ~ normal(b*prior_score, sigma_a);
  for (i in 1:ngames)
    sqrt_dif[i] ~ student_t(df, a[team1[i]]-a[team2[i]], sigma_y);
}

generated quantities {
real team1rank;
real team2rank;
real rankdif;
vector[64] log_lik;

// Predicting the Finals consisting of team 13 vs. team 3.
team1rank = normal_rng(b*prior_score[13],sigma_a);
team2rank = normal_rng(b*prior_score[3],sigma_a);
rankdif = team1rank - team2rank;

for (i in 1:64)
log_lik[i] = normal_lpdf(a[i] | prior_score[i],sigma_a);
}
```

Model 1 - t(3,0,1) priors

```
data {
  int nteams;
  int ngames;
  vector[nteams] prior_score;
  int team1[ngames];
  int team2[ngames];
  vector[ngames] score1;
  vector[ngames] score2;
  real df;
}
transformed data {
  vector[ngames] dif;
  vector[ngames] sqrt_dif;
  dif =  score1 - score2;
  for (i in 1:ngames)
    sqrt_dif[i] = 2*(step(dif[i]) - .5)*sqrt(fabs(dif[i]));
}

parameters {
  real b;
  real<lower=0> sigma_a;
  real<lower=0> sigma_y;
  vector[nteams] a;
}
model {
  sigma_a ~ student_t(3,0,1); // Prior for sigma_a
  sigma_y ~ student_t(3,0,1); // Prior for sigma_y
  a ~ normal(b*prior_score, sigma_a);
  for (i in 1:ngames)
    sqrt_dif[i] ~ student_t(df, a[team1[i]]-a[team2[i]], sigma_y);
}

generated quantities {
real team1rank;
real team2rank;
real rankdif;
vector[64] log_lik;

// Predicting the Finals consisting of team 13 vs. team 3.
team1rank = normal_rng(b*prior_score[13],sigma_a);
team2rank = normal_rng(b*prior_score[3],sigma_a);
rankdif = team1rank - team2rank;

for (i in 1:64)
log_lik[i] = normal_lpdf(a[i] | prior_score[i],sigma_a);
}
```

Model 2 - default priors

```
data {
  int nteams;
  int ngames;
  vector[nteams] prior_score;
  vector[nteams] nationality;
  int team1[ngames];
  int team2[ngames];
  vector[ngames] score1;
  vector[ngames] score2;
  real df;
}

transformed data {
  vector[ngames] dif;
  vector[ngames] sqrt_dif;
  dif =  score1 - score2;
  for (i in 1:ngames)
    sqrt_dif[i] = 2*(step(dif[i]) - .5)*sqrt(fabs(dif[i]));
}

parameters {
  real b;
  real<lower=0> sigma_a1;
  real<lower=0> sigma_a2;
  real<lower=0> sigma_y;
  vector[nteams] a;
}

model {
  a ~ normal(b*prior_score,sigma_a1*nationality + (1 - nationality)*sigma_a2);
  for (i in 1:ngames)
    sqrt_dif[i] ~ student_t(df, a[team1[i]]-a[team2[i]], sigma_y);
}

generated quantities {
real team1rank;
real team2rank;
real rankdif;
vector[64] log_lik;

// Finals were between two same nationality team
team1rank = normal_rng(b*prior_score[13],sigma_a1);
team2rank = normal_rng(b*prior_score[3],sigma_a1);
rankdif = team1rank - team2rank;

for (i in 1:64)
log_lik[i] = normal_lpdf(a[i] | prior_score[i],sigma_a1*nationality + (1 - nationality)*sigma_a2);
}
```

Model 2 - N(0,1) priors

```
data {
  int nteams;
  int ngames;
  vector[nteams] prior_score;
  vector[nteams] nationality;
  int team1[ngames];
  int team2[ngames];
  vector[ngames] score1;
  vector[ngames] score2;
  real df;
}

transformed data {
  vector[ngames] dif;
  vector[ngames] sqrt_dif;
  dif =  score1 - score2;
  for (i in 1:ngames)
    sqrt_dif[i] = 2*(step(dif[i]) - .5)*sqrt(fabs(dif[i]));
}

parameters {
  real b;
  real<lower=0> sigma_a1;
  real<lower=0> sigma_a2;
  real<lower=0> sigma_y;
  vector[nteams] a;
}

model {
  sigma_a1 ~ normal(0,1); // Prior for sigma_a1
  sigma_a2 ~ normal(0,1); // Prior for sigma_a2
  sigma_y ~ normal(0,1); // Prior for sigma_y
  a ~ normal(b*prior_score,sigma_a1*nationality + (1 - nationality)*sigma_a2);
  for (i in 1:ngames)
    sqrt_dif[i] ~ student_t(df, a[team1[i]]-a[team2[i]], sigma_y);
}

generated quantities {
real team1rank;
real team2rank;
real rankdif;
vector[64] log_lik;

// Finals were between two same nationality team
team1rank = normal_rng(b*prior_score[13],sigma_a1);
team2rank = normal_rng(b*prior_score[3],sigma_a1);
rankdif = team1rank - team2rank;

for (i in 1:64)
log_lik[i] = normal_lpdf(a[i] | prior_score[i],sigma_a1*nationality + (1 - nationality)*sigma_a2);
}
```