trabalho_bayes

November 21, 2019

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pystan
import pybaseball
import arviz as az
```

[2]:
```
player_names = ["Peter Alonso","Keston Hiura","Fernando Tatis Jr.","Harold␣
↪Ramirez","Jose Trevino","Yordan Alvarez","Vladimir Guerrero Jr.","Steve␣
↪Wilkerson"]
```

[3]:
```
batter_data = pybaseball.batting_stats(2019)
```

[4]:
```
batter_data = batter_data.loc[batter_data['Name'].isin(player_names)].
↪set_index("Name")
batter_data
```

[4]:

Name	Season	Team	Age	G	AB	PA	H	\
Peter Alonso	2019.0	Mets	24.0	161.0	597.0	693.0	155.0	
Yordan Alvarez	2019.0	Astros	22.0	87.0	313.0	369.0	98.0	
Fernando Tatis Jr.	2019.0	Padres	20.0	84.0	334.0	372.0	106.0	
Keston Hiura	2019.0	Brewers	22.0	84.0	314.0	348.0	95.0	
Vladimir Guerrero Jr.	2019.0	Blue Jays	20.0	123.0	464.0	514.0	126.0	
Harold Ramirez	2019.0	Marlins	24.0	119.0	421.0	446.0	116.0	
Jose Trevino	2019.0	Rangers	26.0	40.0	120.0	126.0	31.0	
Steve Wilkerson	2019.0	Orioles	27.0	119.0	329.0	361.0	74.0	

Name	1B	2B	3B	...	wSL/C (pi)	wXX/C (pi)	\
				...			
Peter Alonso	70.0	30.0	2.0	...	0.25	NaN	
Yordan Alvarez	45.0	26.0	0.0	...	3.49	NaN	
Fernando Tatis Jr.	65.0	13.0	6.0	...	1.95	4.87	
Keston Hiura	51.0	23.0	2.0	...	2.56	NaN	
Vladimir Guerrero Jr.	83.0	26.0	2.0	...	-1.39	NaN	
Harold Ramirez	82.0	20.0	3.0	...	-0.85	NaN	
Jose Trevino	20.0	9.0	0.0	...	-2.29	NaN	
Steve Wilkerson	44.0	18.0	2.0	...	-0.51	NaN	

Name	O-Swing% (pi)	Z-Swing% (pi)	Swing% (pi)	\
Peter Alonso	0.339	0.652	0.468	
Yordan Alvarez	0.311	0.591	0.434	
Fernando Tatis Jr.	0.304	0.701	0.477	
Keston Hiura	0.318	0.729	0.515	
Vladimir Guerrero Jr.	0.301	0.681	0.467	
Harold Ramirez	0.402	0.715	0.540	
Jose Trevino	0.413	0.554	0.486	
Steve Wilkerson	0.344	0.572	0.454	

Name	O-Contact% (pi)	Z-Contact% (pi)	Contact% (pi)	\
Peter Alonso	0.599	0.825	0.729	
Yordan Alvarez	0.623	0.853	0.761	
Fernando Tatis Jr.	0.445	0.800	0.673	
Keston Hiura	0.423	0.769	0.657	
Vladimir Guerrero Jr.	0.592	0.875	0.772	
Harold Ramirez	0.610	0.875	0.765	
Jose Trevino	0.716	0.848	0.794	
Steve Wilkerson	0.637	0.819	0.748	

Name	Zone% (pi)	Pace (pi)
Peter Alonso	0.414	25.1
Yordan Alvarez	0.439	24.4
Fernando Tatis Jr.	0.436	25.0
Keston Hiura	0.479	24.2
Vladimir Guerrero Jr.	0.437	25.3
Harold Ramirez	0.442	23.3
Jose Trevino	0.520	23.4
Steve Wilkerson	0.484	24.2

[8 rows x 286 columns]

[5]:
```
binomial_data = {}
for player in batter_data.index:
    hits = int(batter_data.loc[player]["H"])
    ab_minus_hits = int(batter_data.loc[player]["AB"]) - hits
    ab_outcomes = [0]*ab_minus_hits + [1]*hits
    binomial_data.update({player:ab_outcomes})
at_bat_totals = {}
for player in batter_data.index:
    at_bat_count = int(batter_data.loc[player]["AB"])
    at_bat_totals.update({player:at_bat_count})
hit_totals = {}
```

```
for player in batter_data.index:
    hit_count = int(batter_data.loc[player]["H"])
    hit_totals.update({player:hit_count})
```

$BA_i \sim Beta(81, 219)$
$y_i \sim Bin(AB_i, BA_i)$
$i = 1, 2, ..., 8$

[6]:
```
#https://mc-stan.org/users/documentation/case-studies/rstan_workflow.html
#https://people.duke.edu/~ccc14/sta-663/PyStan.html
#http://varianceexplained.org/statistics/beta_distribution_and_baseball/

model_code = '''
data {
  int<lower=0> N;
  int<lower=0> at_bats[N];
  int<lower=0> hits[N];
  real<lower=0> A;
  real<lower=0> B;
}
parameters {
  real<lower=0,upper=1> AVG[N];
}
model {
  AVG ~ beta(A, B);
  hits ~ binomial(at_bats, AVG);
}
generated quantities {
    vector[N] log_lik;
    vector[N] predicted_hits;
    for (i in 1:N) {
        log_lik[i] = binomial_lpmf(hits[i] | at_bats[i], AVG[i]);
        predicted_hits[i] = binomial_rng(at_bats[i], AVG[i]);
    }
}
'''

model_data = dict(N=8, hits=list(hit_totals.
 ↪values()),at_bats=list(at_bat_totals.values()),A=81,B=219)
stan_model = pystan.StanModel(model_code=model_code)
fit = stan_model.sampling(data=model_data)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL
anon_model_4d40fdfa42b9a644c6433b2a714e9368 NOW.
```

[7]:
```
print(fit)
```

```
Inference for Stan model: anon_model_4d40fdfa42b9a644c6433b2a714e9368.
4 chains, each with iter=2000; warmup=1000; thin=1;
```

post-warmup draws per chain=1000, total post-warmup draws=4000.

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
AVG[1]	0.26	1.7e-4	0.01	0.24	0.25	0.26	0.27	0.29	7260	1.0
AVG[2]	0.29	2.2e-4	0.02	0.26	0.28	0.29	0.3	0.33	6842	1.0
AVG[3]	0.3	2.2e-4	0.02	0.26	0.28	0.29	0.31	0.33	6528	1.0
AVG[4]	0.29	2.1e-4	0.02	0.25	0.27	0.29	0.3	0.32	7373	1.0
AVG[5]	0.27	2.1e-4	0.02	0.24	0.26	0.27	0.28	0.3	5894	1.0
AVG[6]	0.27	2.0e-4	0.02	0.24	0.26	0.27	0.28	0.31	6362	1.0
AVG[7]	0.27	2.7e-4	0.02	0.23	0.25	0.27	0.28	0.31	5854	1.0
AVG[8]	0.25	2.1e-4	0.02	0.22	0.24	0.25	0.26	0.28	6435	1.0
log_lik[1]	-3.64	0.01	0.49	-5.01	-3.74	-3.45	-3.33	-3.29	1873	1.0
log_lik[2]	-3.62	0.01	0.7	-5.54	-3.9	-3.38	-3.12	-3.03	3387	1.0
log_lik[3]	-3.71	0.01	0.73	-5.63	-4.01	-3.47	-3.18	-3.06	4215	1.0
log_lik[4]	-3.48	0.01	0.58	-5.12	-3.67	-3.26	-3.07	-3.02	2921	1.0
log_lik[5]	-3.49	0.01	0.44	-4.69	-3.58	-3.33	-3.21	-3.18	1371	1.0
log_lik[6]	-3.42	9.4e-3	0.39	-4.54	-3.52	-3.27	-3.17	-3.14	1673	1.0
log_lik[7]	-2.64	5.1e-3	0.22	-3.24	-2.7	-2.56	-2.5	-2.49	1877	1.0
log_lik[8]	-3.6	0.01	0.73	-5.56	-3.88	-3.36	-3.06	-2.95	3949	1.0
predicted_hits[1]	157.05	0.2	13.74	131.0	148.0	157.0	166.0	184.0	4592	1.0
predicted_hits[2]	91.42	0.14	10.18	72.0	84.0	91.0	98.0	111.0	4960	1.0
predicted_hits[3]	98.44	0.15	10.08	79.0	92.0	98.0	105.0	119.0	4683	1.0
predicted_hits[4]	90.08	0.15	9.96	71.0	83.0	90.0	97.0	110.0	4394	1.0
predicted_hits[5]	125.81	0.18	12.17	103.0	118.0	125.0	134.0	151.0	4739	1.0
predicted_hits[6]	115.5	0.18	11.65	93.0	108.0	115.0	123.0	139.0	4335	1.0

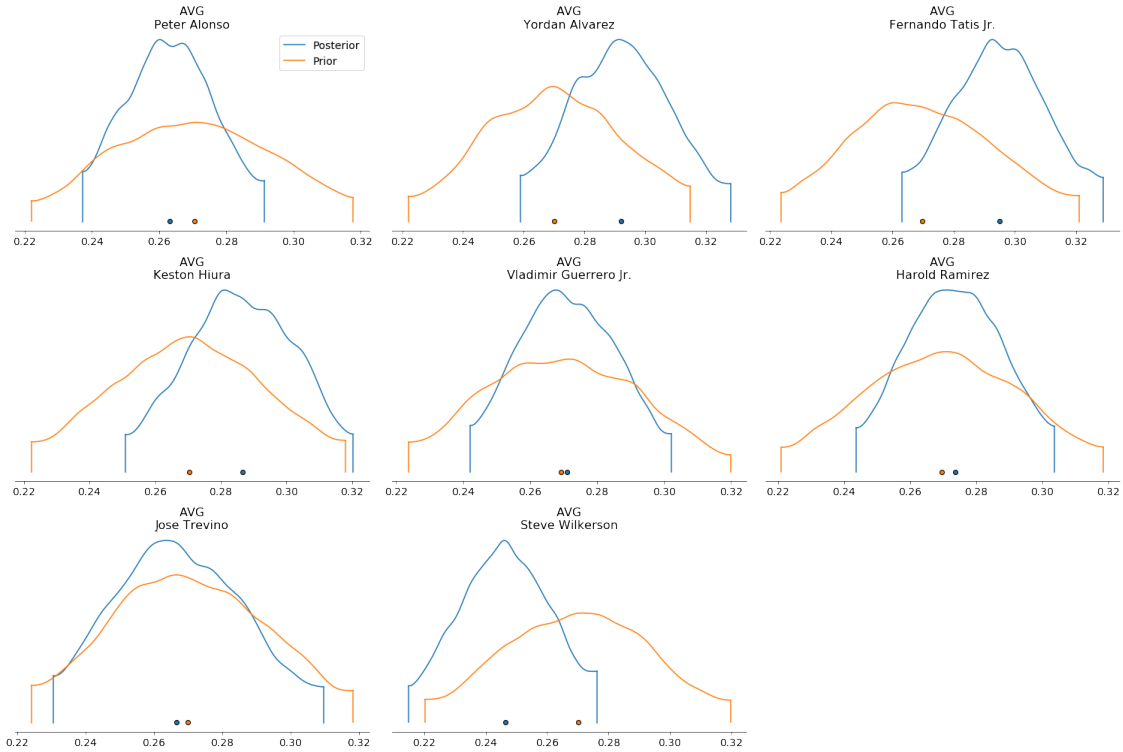
```
predicted_hits[7]  32.09    0.08   5.42   22.0   28.0   32.0   36.0   43.0
4109    1.0
predicted_hits[8]  81.14    0.14   9.27   64.0   75.0   81.0   87.0  100.0
4561    1.0
lp__               -3107    0.05   1.95  -3111  -3108  -3107  -3105  -3104
1748    1.0

Samples were drawn using NUTS at Thu Nov 21 20:41:52 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).
```

[8]:
```
prior_model_code = '''
data {
  int<lower=0> N;
  real<lower=0> A;
  real<lower=0> B;
}

parameters {
  real<lower=0,upper=1> AVG[N];
}

model {
  AVG ~ beta(A, B);
  }
'''

prior_model_data = dict(N=8,A=81,B=219)
stan_model_prior = pystan.StanModel(model_code=prior_model_code)
prior_fit = stan_model_prior.sampling(data=prior_model_data)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL
anon_model_390c11193c42041ad1d6bfb9bf37ebfd NOW.
```

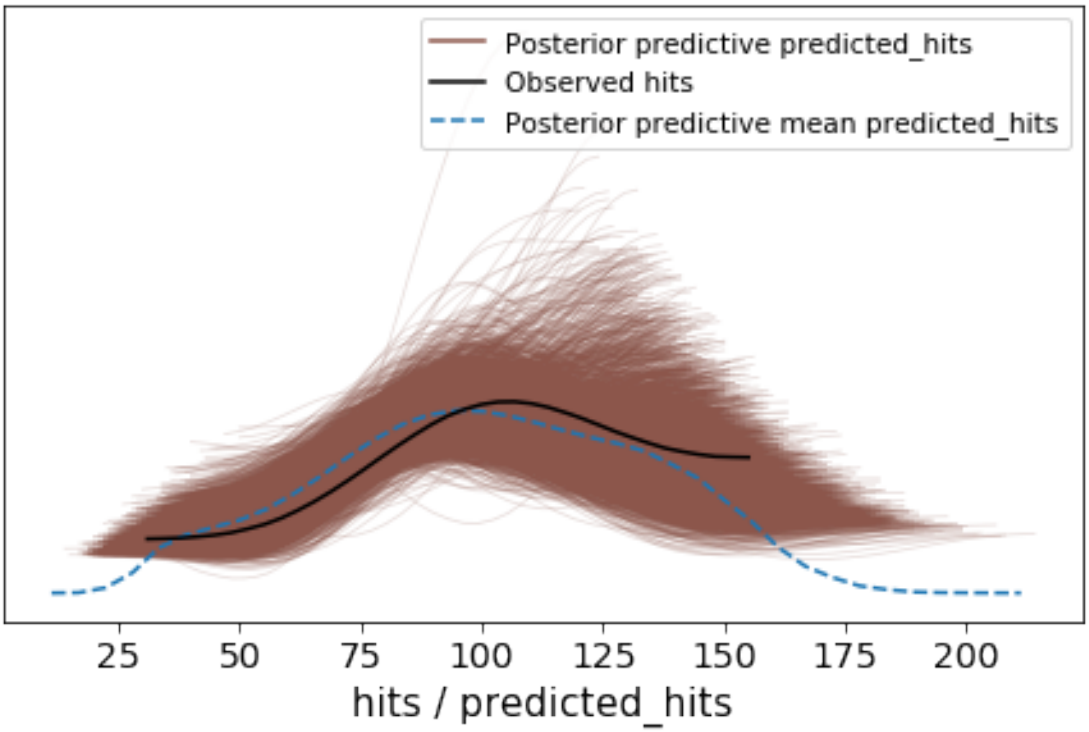
[9]:
```
stan_data = az.from_pystan(posterior=fit,
                           prior=prior_fit,
                           observed_data="hits",
                           posterior_predictive="predicted_hits",
                           log_likelihood="log_lik",
                           posterior_model=stan_model,
                           coords={"player":list(hit_totals.keys())},
                           dims={'AVG': ['player'], 'hits': ['player'],␣
 ↪'log_lik': ['player'], 'predicted_hits': ['player']})
```

[10]:
```
density_plots = az.plot_density([stan_data.posterior,stan_data.
 ↪prior],data_labels=["Posterior","Prior"])
```



[11]:
```
az.plot_ppc(stan_data, data_pairs = {"hits" :␣
↪"predicted_hits"},flatten=["player"])
```

[11]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x0000025AE365B588>],
      dtype=object)
```


Posterior predictive predicted_hits
Observed hits
Posterior predictive mean predicted_hits
25
50
75
100
125
150
175
200
hits / predicted_hits