Stan Is The Plan Part 2

Breck Baldwin

NYC PyData 2019

Outline

- Turn one_putt.stan into one_putt_predict.stan
- Go over major parts of a Bayesian model
- Try some different likelihoods
- Add some data.
- Add a lot of data.

Stan/one_putt_predict.stan

Externally supplied data

Parameter we are trying to learn

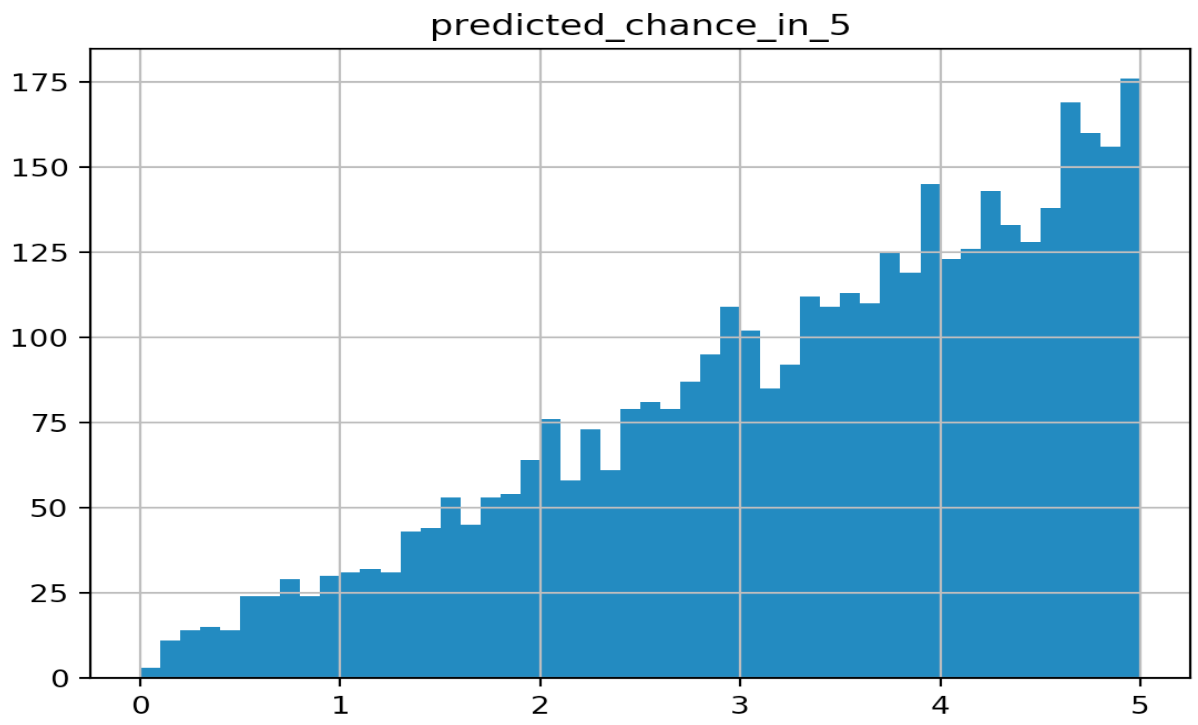
Prior

Likelihood

Prediction

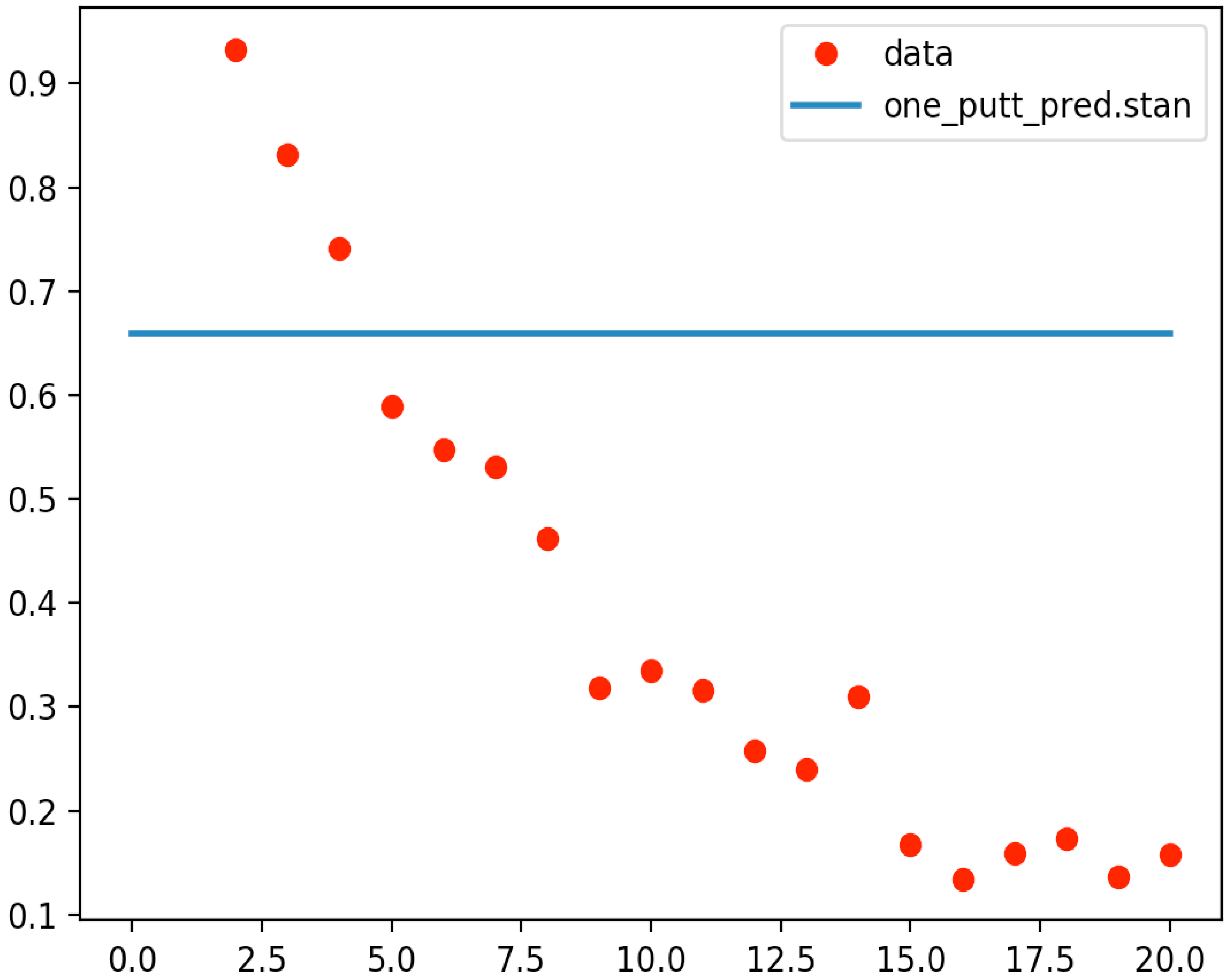
```
data {
   real distance_of_putt;
}

parameters {
  real<lower=0, upper=1> chance_in_1;
}

model {
  chance_in_1 ~ uniform(0,1);
  1 ~ bernoulli(chance_in_1);
}

generated quantities {
  real pred_ch_in_5 = chance_in_1 * 5;
}
```

```
$ python puttBet.py stan/one_putt_predict.stan 5
```



$ python puttBet.py stan/one_putt_predict.stan 5

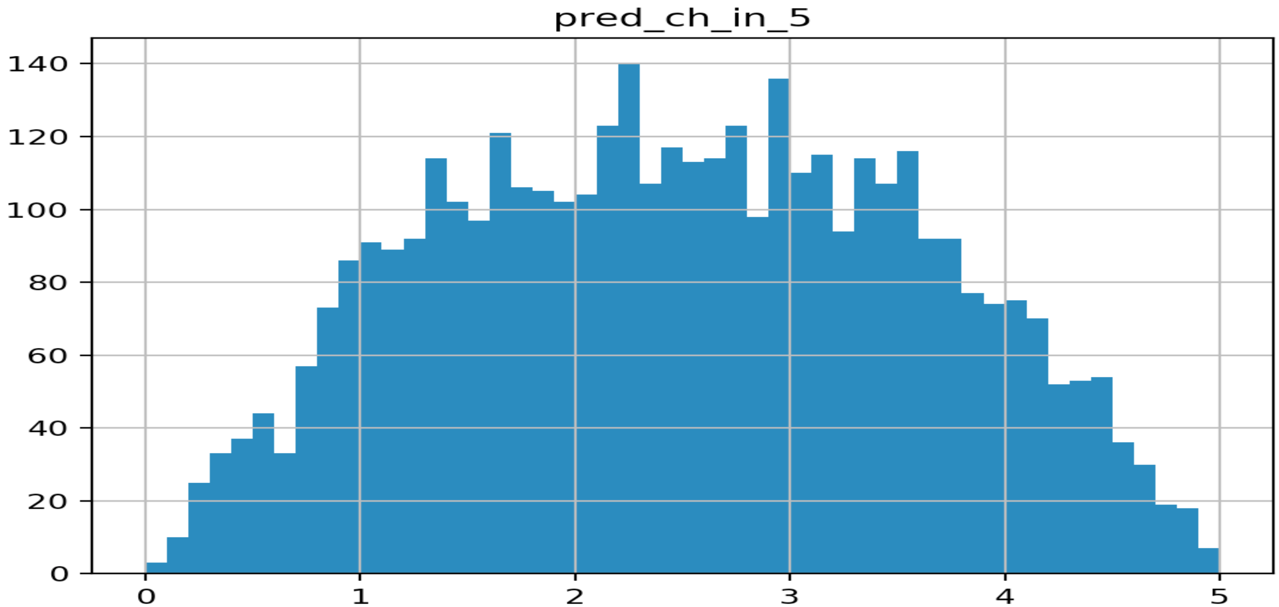
	Mean	MCSE	StdDev	5%	...	95%	N_Eff	N_Eff/s	R_hat
name					...				
lp__	-2.539970	0.025559	0.904508	-4.321050	...	-1.912100	1252.40	6963.88	1.00248
chance_in_1	0.668104	0.006756	0.239877	0.208745	...	0.977099	1260.52	7009.04	1.00323
pred_ch_in_5	3.340520	0.033782	1.199390	1.043720	...	4.885490	1260.52	7009.04	1.00323



data
one_putt_pred.stan
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Add some more data

```
model {
   chance_in_1 ~ uniform(0,1);
   1 ~ bernoulli(chance_in_1);
   0 ~ bernoulli(chance_in_1);
}
$ python puttBet.py stan/two_putt.stan 5
```



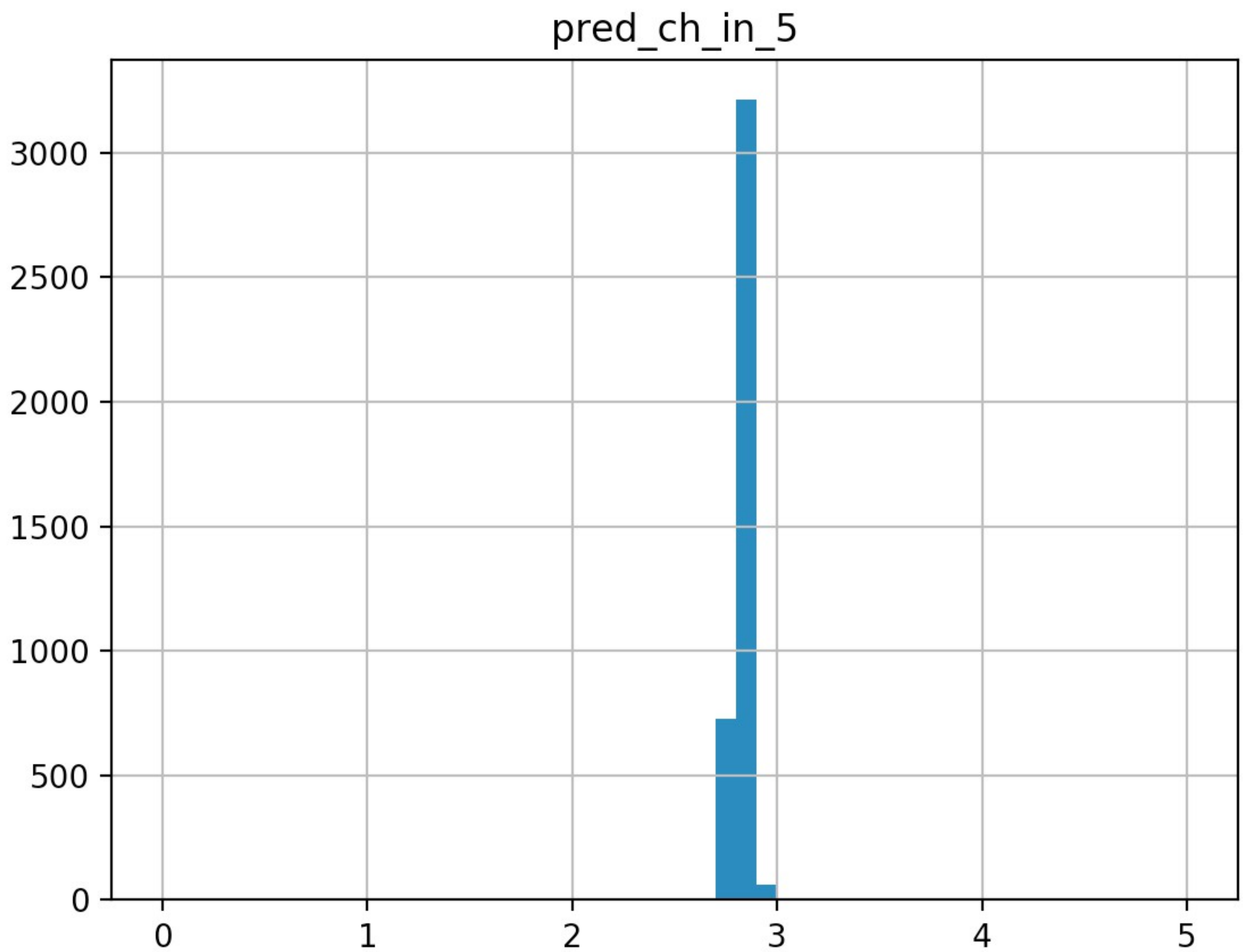
stan/pooled_golf.stan

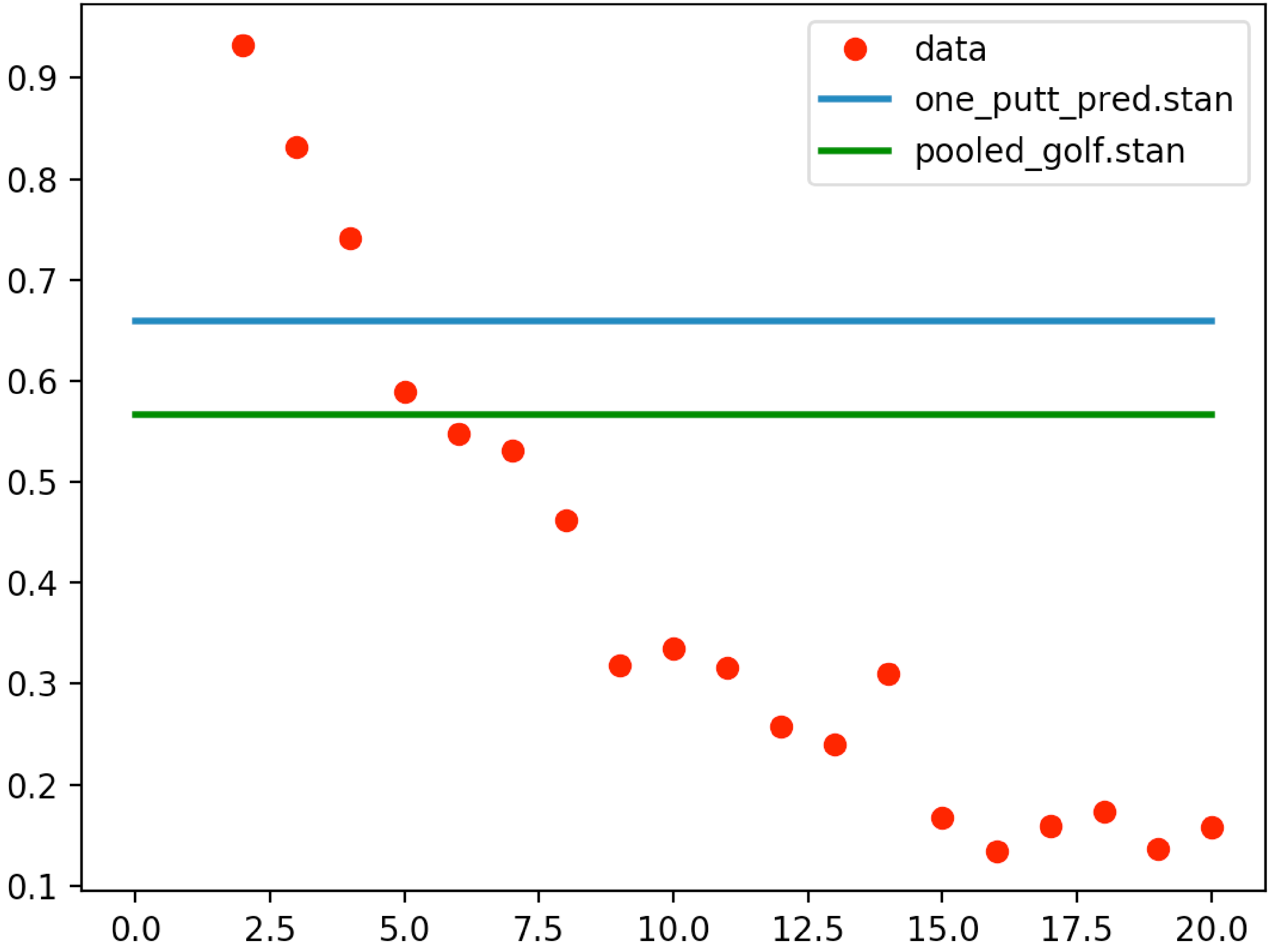
```
data {
  real distance_of_putt;
}

transformed data {
  int J = 19;
  int x[J] = {2,3,4,5,6,7,8,9,10,11,12,13,14,15...};
  int n[J] = {1443,694,455,353,272,256,240,217,200...};
  int y[J] = {1346,577,337,208,149,136,111,69,67,...};
}

parameters {
  real<lower=0,upper=1> chance_in_1;
}

model {
  for (i in 1:J) {
    y[i] ~ binomial(n[i], chance_in_1);
  }
}

generated quantities {
  real pred_ch_in_5 = chance_in_1*5;
}
```

What is the bionomial?

```
functions {

real my_binomial_lpmf(int successes, int attempts,
                      real chance_in_1) {
    real return_probability = 0;
    for (i in 1:successes) {
      return_probability += bernoulli_lpmf(1|chance_in_1);
    }
    for (i in 1:attempts-successes) {
      return_probability += bernoulli_lpmf(0|chance_in_1);
    }
    return return_probability;
  }
}

$ python puttBet.py stan/my_binomial.stan 5
```

```
$ python puttBet.py stan/pooled_golf.stan 5
```




data
one_putt_pred.stan
pooled_golf.stan
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

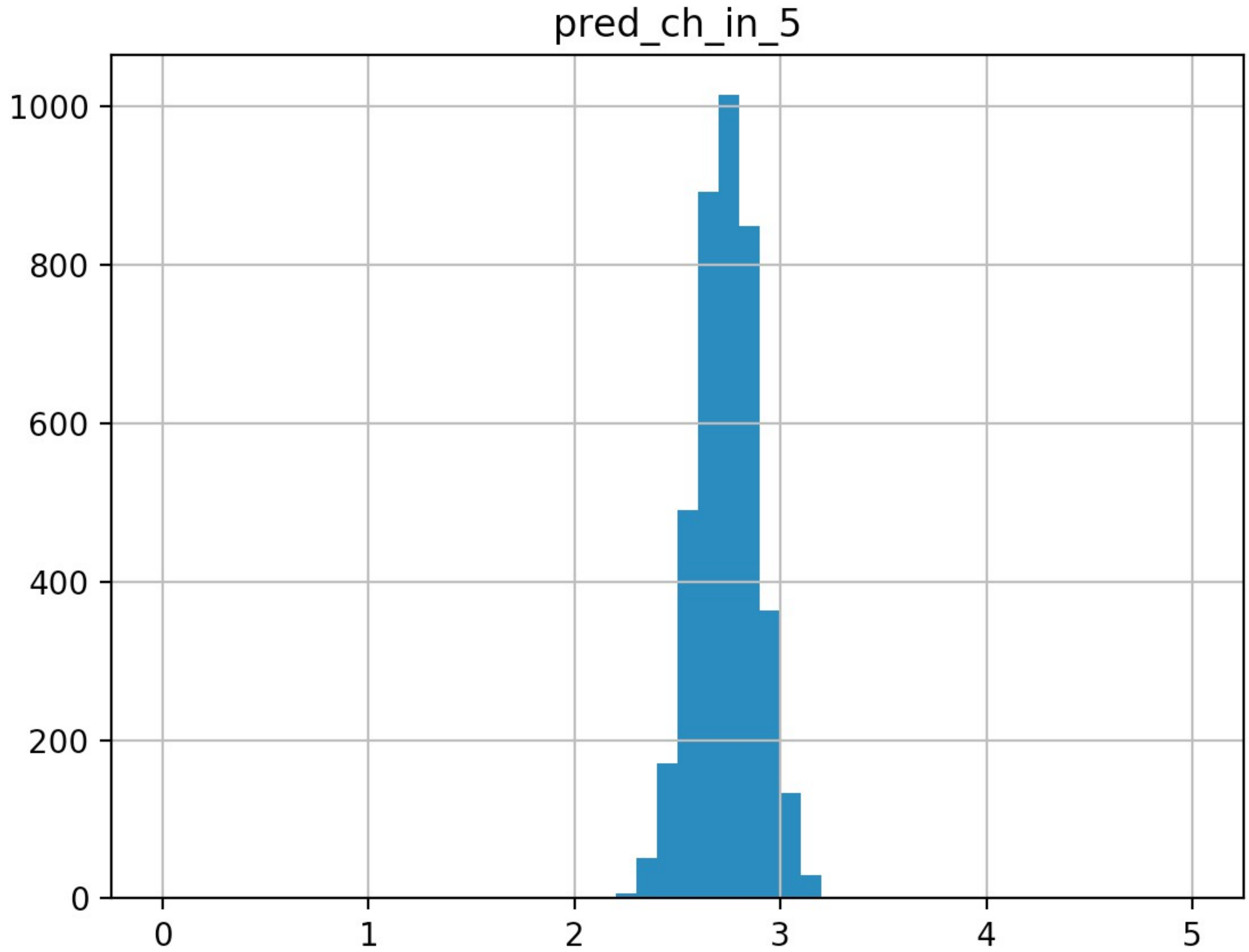
Changes for not pooled: not_pooled_golf.stan

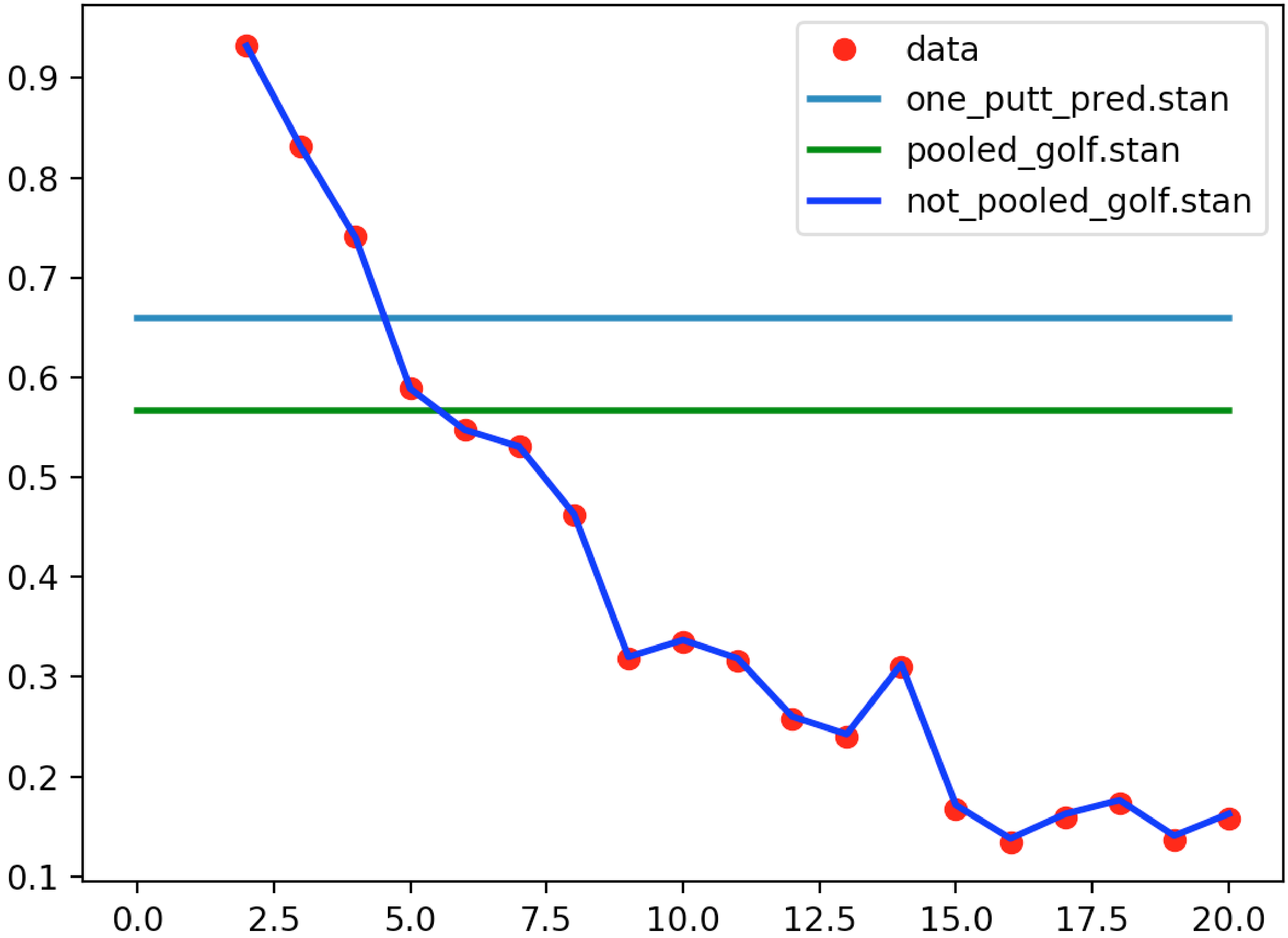
```
parameters {
  real<lower=0,upper=1> chance_in_1_for_dist[J];
}

model {
  for (i in 1:J) {
    y[i] ~ binomial(n[i], chance_in_1_for_dist[i]);
  }
}

generated quantities {
  real pred_ch_in_5 = chance_in_1_for_dist[J]*5;
  for (i in 1:J) {
    if (distance_of_putt < x[i]) {
      pred_ch_in_5 = chance_in_1_for_dist[i]*5;
      break;
    }
  }
}
```

```
$ python puttBet.py stan/not_pooled_golf.stan 5
```

```
                                  Mean      MCSE    StdDev  ...     N_Eff    N_Eff/s     R_hat
name                                                        ...
lp__                      -2935.680000  0.082004  3.129880  ...   1456.75    3062.43  1.000410
chance_in_1_for_dist[1]       0.932163  0.000070  0.006558  ...   8678.76   18244.70  0.999355
chance_in_1_for_dist[2]       0.830304  0.000138  0.014137  ...  10556.90   22192.90  0.999654
chance_in_1_for_dist[3]       0.739621  0.000227  0.021108  ...   8658.28   18201.70  0.999627
chance_in_1_for_dist[4]       0.588373  0.000250  0.026130  ...  10893.30   22900.20  0.999372
chance_in_1_for_dist[5]       0.547546  0.000318  0.029914  ...   8821.41   18544.60  0.999533
chance_in_1_for_dist[6]       0.531649  0.000303  0.030247  ...   9949.41   20915.90  0.999200
chance_in_1_for_dist[7]       0.462658  0.000318  0.031639  ...   9895.18   20801.90  0.999390
...
chance_in_1_for_dist[18]      0.140775  0.000272  0.028243  ...  10756.00   22611.50  0.999154
chance_in_1_for_dist[19]      0.162293  0.000334  0.030506  ...   8325.80   17502.70  0.999948
pred_ch_in_5                  2.737730  0.001592  0.149570  ...   8821.38   18544.50  0.999533
```

```
python puttBet.py stan/not_pooled_golf.stan 5
```



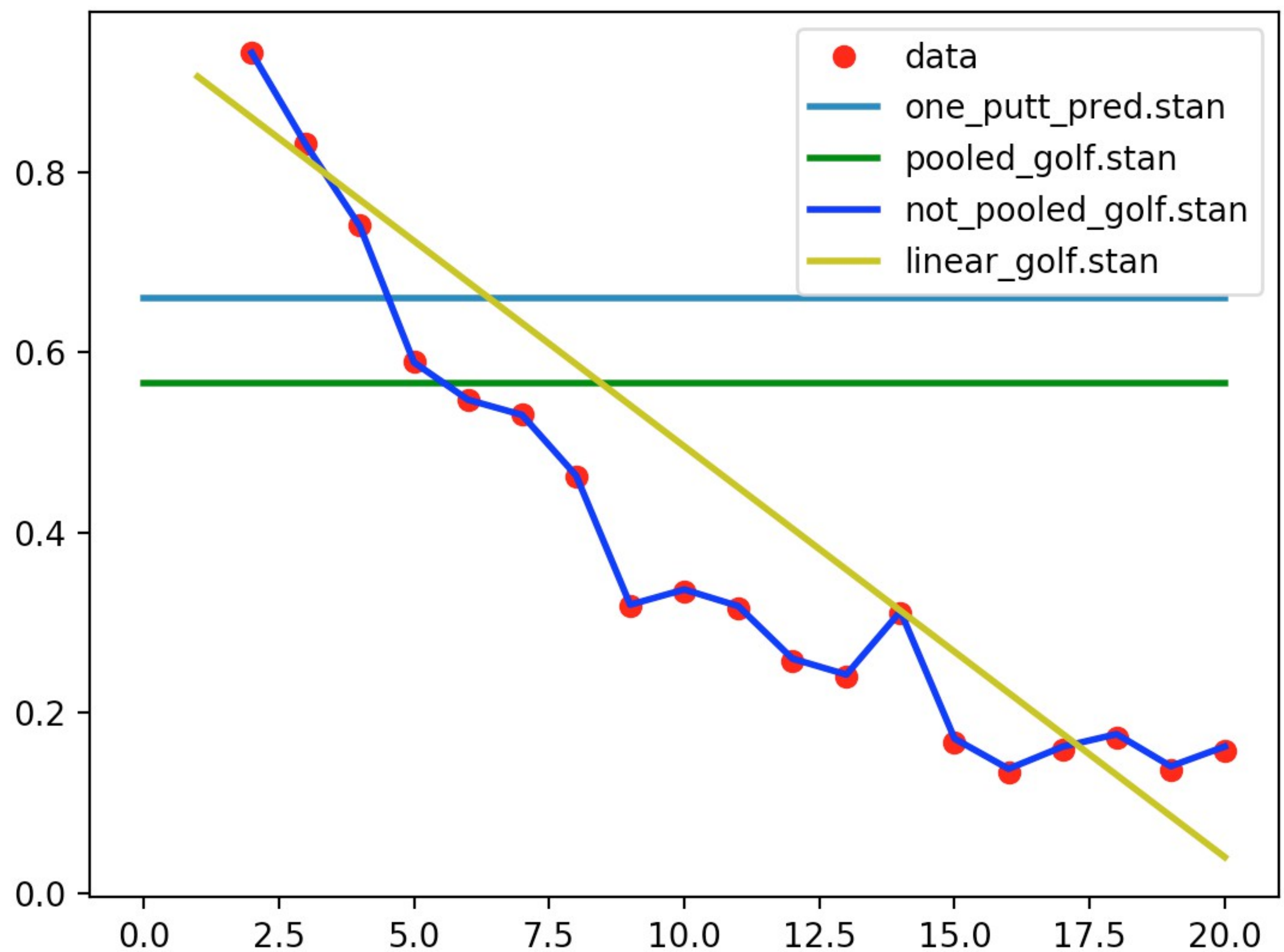


data
one_putt_pred.stan
pooled_golf.stan
not_pooled_golf.stan
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Changes for linear_golf.stan

```
parameters {
  real a_intercept;
  real <lower=-.1,upper=0>b_slope;
}

model {
  for (i in 1:J) {
      real chance_in_1 =  a_intercept + b_slope*x[i];
      y[i] ~ binomial(n[i], chance_in_1);
  }
}

generated quantities {
  real pred_ch_in_5 =
               (a_intercept + b_slope*distance_of_putt)*5;
}
```

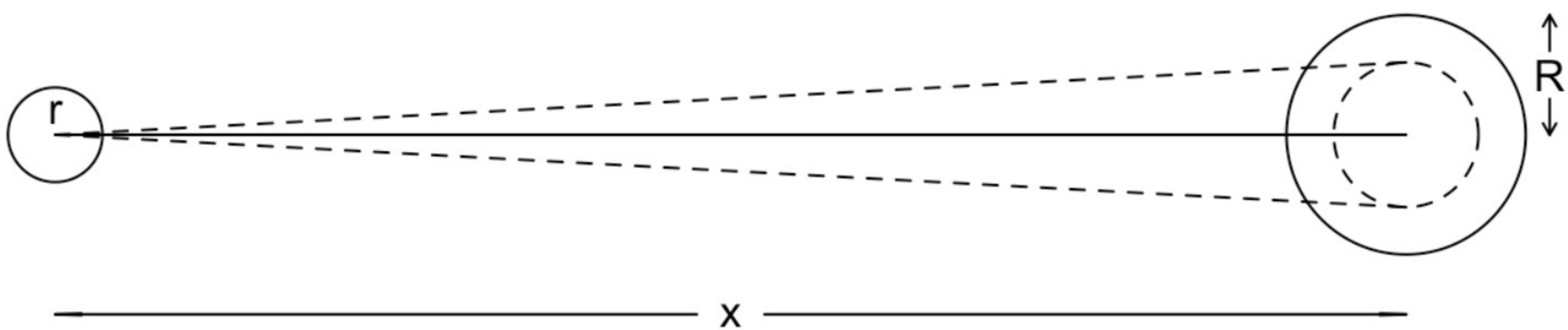
name	Mean	MCSE	StdDev	5%	50%	95%	N_Eff	N_Eff/s	R_hat
lp__	-3061.360000	0.026869	1.005330	-3063.380000	-3061.050000	-3060.370000	1399.98	4256.86	1.00086
a_intercept	0.951225	0.000241	0.008740	0.936986	0.951358	0.965452	1316.54	4003.16	1.00136
b_slope	-0.045555	0.000017	0.000610	-0.046541	-0.045580	-0.044531	1224.13	3722.17	1.00222
pred_ch_in_5	3.845020	0.000879	0.033947	3.790300	3.845330	3.900940	1490.89	4533.30	1.00095



data
one_putt_pred.stan
pooled_golf.stan
not_pooled_golf.stan
linear_golf.stan
0.0
0.2
0.4
0.6
0.8
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

More on the Mechanistic Model

- Andrew Gelman Case Study:
  - https://mc-stan.org/users/documentation/case-studies/golf.html

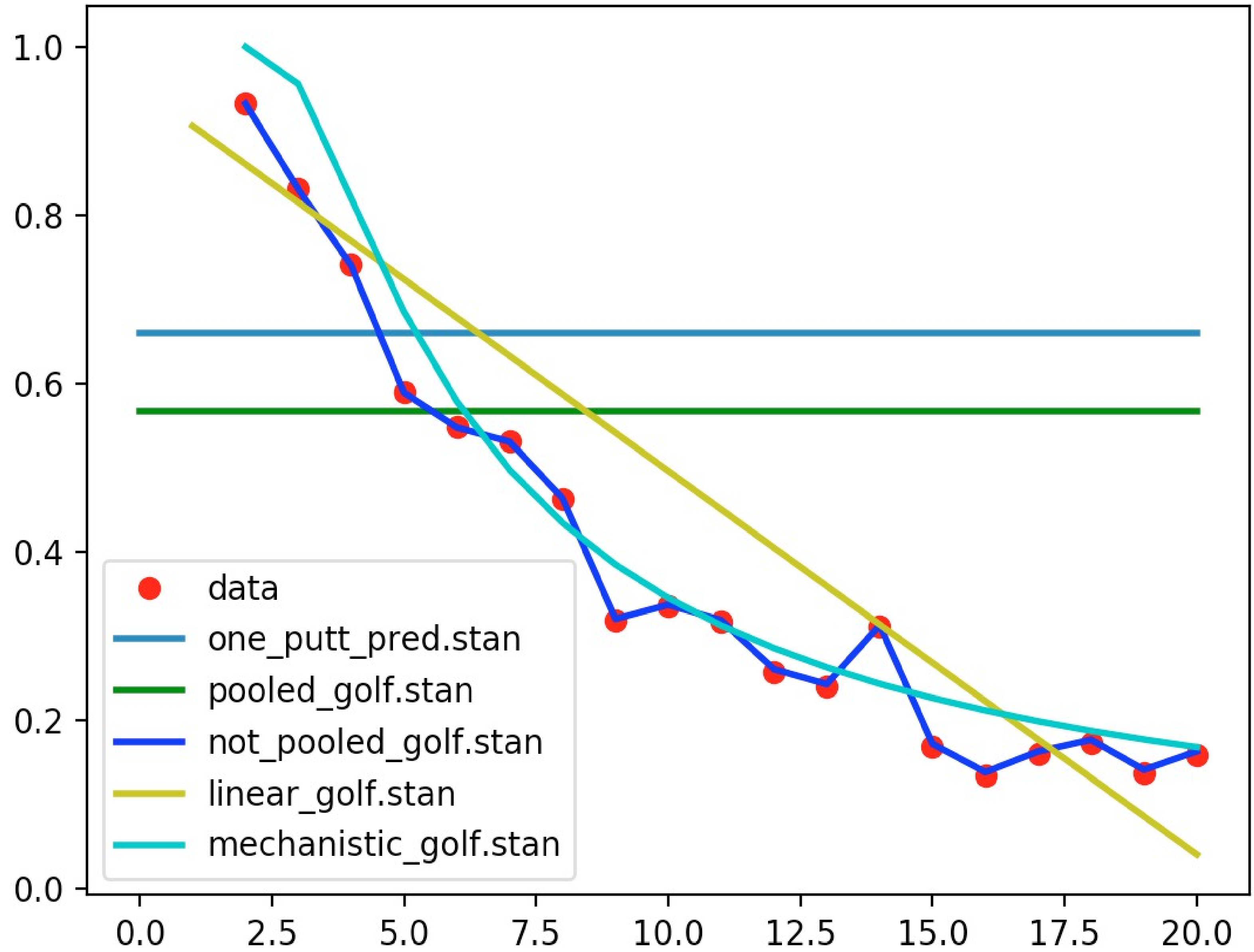


Changes for mechanistic_golf.stan

```
transformed data {
...
  real r = (1.68/2)/12;
  real R = (4.25/2)/12;
  real threshold_angle[J];
  for (i in 1:J)
    threshold_angle[i] = asin((R-r)/x[i]);
}

parameters {
  real<lower=0> sigma;
}

model {
  for (i in 1:J) {
    real prob = 2*Phi(threshold_angle[i]/sigma) - 1;
    y[i] ~ binomial(n[i], prob);
  }
}

generated quantities {
  real sigma_degrees = (180/pi())*sigma;
  real pred_ch_in_5 =
       (2*Phi(threshold_angle_for_distance/sigma) - 1) * 5;
}
```

$ python puttBet.py stan/mechanistic_golf.stan 4

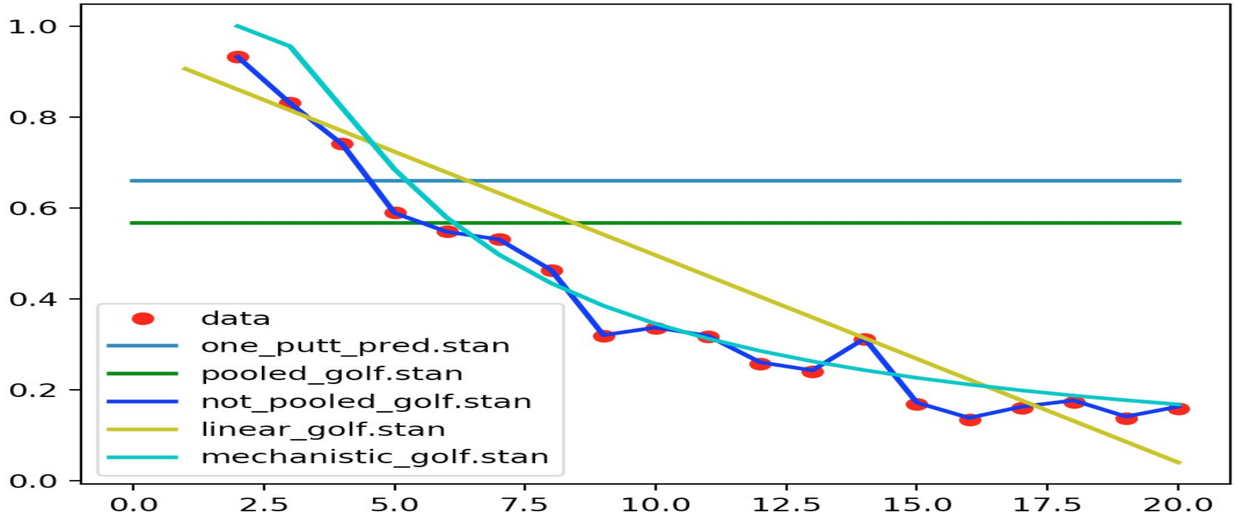
```
                                    Mean          MCSE        StdDev  ...       N_Eff    N_Eff/s     R_hat
name                                                                  ...
lp__                         -2926.760000  1.708170e-02  6.843830e-01  ...  1605.22000  6509.1500  1.002320
sigma                            0.026662  1.016150e-05  3.926940e-04  ...  1493.47000  6056.0100  1.001460
sigma_degrees                    1.527630  5.822050e-04  2.249980e-02  ...  1493.50000  6056.1300  1.001460
pred_ch_in_5                     3.423840  9.231030e-04  3.563680e-02  ...  1490.38000  6043.4600  1.001490
threshold_angle_for_distance     0.026774  9.705930e-17  1.943130e-16  ...     4.00803    16.2525  0.998999
```



1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
data
one_putt_pred.stan
pooled_golf.stan
not_pooled_golf.stan
linear_golf.stan
mechanistic_golf.stan

Second Bayesian Point

- What is the most robust model?



- Custom models allow mechanistic models
- Mechanistic models are human interpretable
- Mechanistic models more likely robust

General Mucking About with Stan

- Execution Environment
  - Hello World
- Messing about with distributions
- Overview of diagnostics

Stan's execution environment

- Metropolis-Hastings
  - Generate proposal values for all params
  - Get max probability from likelihood across all statements accumulated in target
  - Accumulate previous values or proposal values based on probabilistic accept target vs prev_target
  - Start over with target = 0, exp(target)=1

HMC/NUTS

- 4x1000 warmups to scope out the posterior and step size
- 4x1000 draws
  - Proposals are developed sensitive to combined gradients of parameters via leapfrog exploration.
  - Cannot deal with discontinuity in parameters
  - Discrete data is fine
  - Most proposals are accepted
  - Proposals end up drawing from posterior

$ python hello_stan.py

```
import os
from cmdstanpy import cmdstan_path, CmdStanModel
import fileinput
import sys

stan_program =
CmdStanModel(stan_file='stan/hello_world.stan')
stan_program.compile()

fit = stan_program.sample(csv_basename='./output',

data={'count_data':4,'continuous_data':2.1},
                   chains=1,sampling_iters=4)

console_output = open('output-1.txt');
print(console_output.read());
print(fit.summary())
```

```
functions {
  void helloWorld() {
    print("Functions{} hello world!");
  }
}
data {
  int count_data;
  real continuous_data;
}
transformed data {
  int tran_count = 1;
  print("transformed data{} hello have access to count_data=",count_data,
        ", continuous_data=",continuous_data);
  print("transformed data{} created new variable with value,”,
         “tran_count=",tran_count);
  helloWorld();
}
parameters {
  real estimate_me;
}
transformed parameters {
  real modified_estimate_me = estimate_me/count_data;
  print("transformed parameters {} Hello,
modified_estimate_me=",modified_estimate_me);
  print("transformed parameters {} is called once per leapfrog step, as is
parameters{}");
}
model {
  print("model{} Hello every leap frog step");
  print("model{} initial target()=",target(),", exp(target()=",exp(target()),
        ", estimate_me=",estimate_me);
  estimate_me ~ normal(count_data,continuous_data);
  print("model{} after increment target()=",target(),", exp(target()=",exp(target()),
        ", estimate_me=",estimate_me);
}
generated quantities {
  real prediction = estimate_me * 5; //will be accumulated in fit object
  print("generated quantities {} Hello run once per sample");
}
```

Hello World CmdStan

- Compile from cmdstan dir.
  - `cmdstan-2.21.0$ make ~/git/StanIsThePlanDist/stan/hello_world`
- Change directory to executable
  - `cmdstan-2.21.0$ cd ~/git/StanIsThePlanDist/stan/`
- Run with json data
  - `stan$ ./hello_world sample data file=hello_world.json`
- Run with R data
  - `stan$ ./hello_world sample data file=hello_world.RData`

Stan Output-CmdStan

- CmdStan is the core
- Plusses
  - Robust
  - Best way to really get the feel of Stan execution
- Minuses
  - Accumulation is on output.csv
  - One chain at a time

Interface Languages

- CmdStanPy, CmdStanR, ScalaStan....
  - Light weight
  - Easy to keep with current Stan version
- Rstan
  - In memory access to Stan functions
  - Lags 6 mos behind
- PyStan
  - Similar to RStan

Best Practices

https://github.com/stan-dev/stan/wiki/Stan-Best-Practices

- Think Generatively
- Start Simple
- Validate your fit: Necessary, not sufficient!
  - Recover simulated data within 5% to 95%
  - Check rhat
  - N_eff / N < 0.001 (low effective sample size)
  - Check Divergences
  - Check for caterpillars
- Folk Theorem

Recover parameters

```
transformed data {
  int simulated_data[100];
  for(i in 1:100) {
    simulated_data[i] = bernoulli_rng(.7);
  }
}

parameters {
  real<lower=0,upper=1> coin_bias;
}

model {
  for (i in 1:100) {
    simulated_data[i] ~ bernoulli(coin_bias);
    //simulated_data[i] ~ exponential(coin_bias);
  }
}

$ python rv.py stan/recover_simulated_params.stan

           5%        50%        95%
coin_bias  0.578442  0.658552   0.733275
```

Recover parameters

```
transformed data {
  int simulated_data[100];
  for(i in 1:100) {
    simulated_data[i] = bernoulli_rng(.7);
  }
}

parameters {
  real<lower=0,upper=1> coin_bias;
}

model {
  for (i in 1:100) {
    //simulated_data[i] ~ bernoulli(coin_bias);
    simulated_data[i] ~ exponential(coin_bias);
  }
}

$ python rv.py stan/recover_simulated_params.stan

           5%        50%        95%
coin_bias  0.922764  0.980232   0.998481
```

Recover parameters

```
transformed data {
  int simulated_data[100];
  for(i in 1:100) {
    simulated_data[i] = bernoulli_rng(.7);
  }
}

parameters {
  real<lower=0,upper=1> coin_bias;
}

model {
  for (i in 1:100) {
    //simulated_data[i] ~ bernoulli(coin_bias);
    simulated_data[i] ~ exponential(coin_bias);
  }
}

$ python rv.py stan/recover_simulated_params.stan

           5%        50%         95%
coin_bias  0.922764  0.980232    0.998481
```

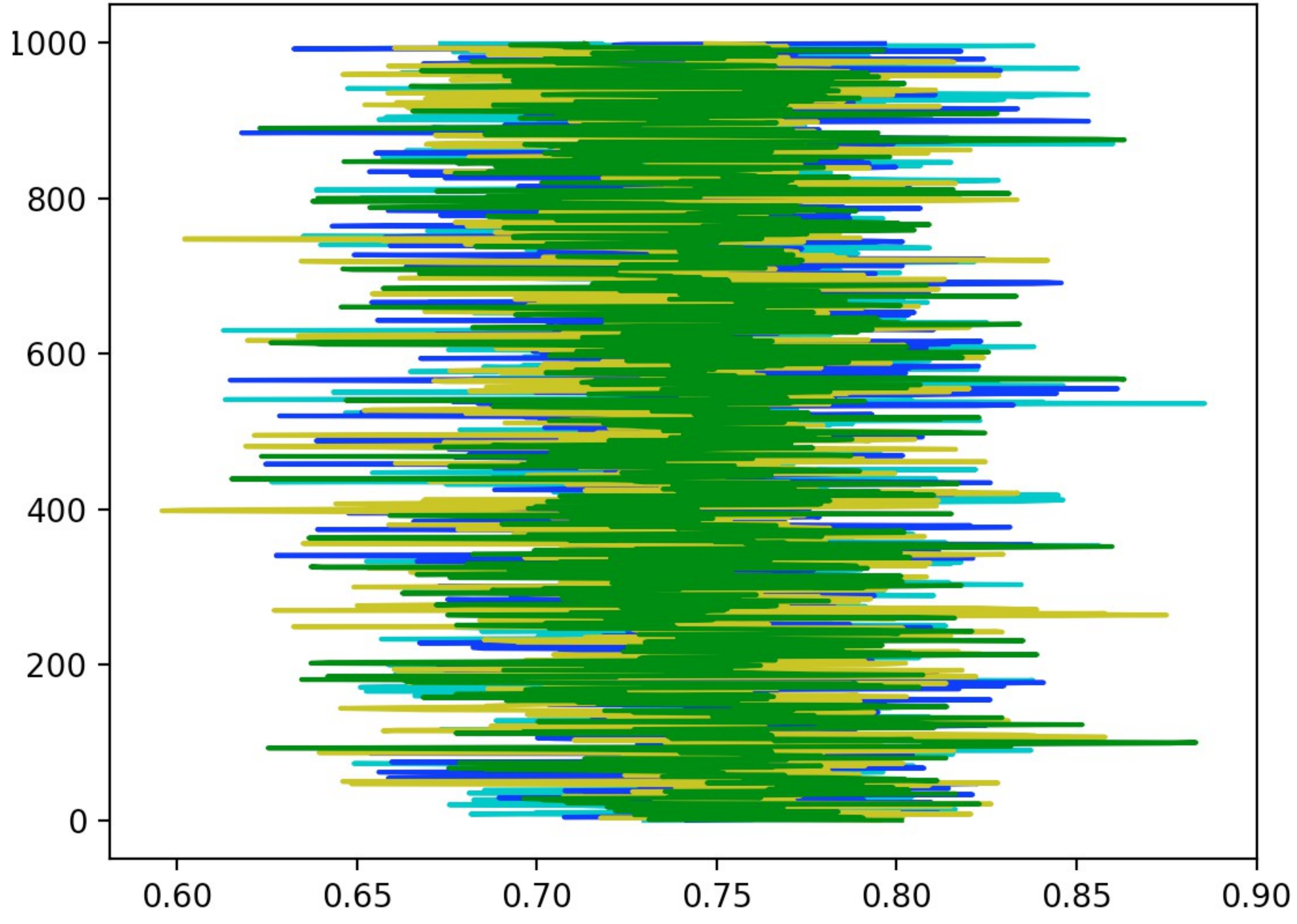
Check rhat and N_eff

```
$ python rv.py stan/recover_simulated_params.stan
```

```
                Mean      MCSE    StdDev         5%        50%        95%     N_Eff  N_Eff/s    R_hat
name
lp__      -73.099500  0.023835  0.742460 -74.680600 -72.807100 -72.542300   970.303  3317.81  1.00162
coin_bias   0.972771  0.000708  0.024974   0.922764   0.980232   0.998481  1245.720  4259.55  1.00201
```

Fuzzy Catepillar

```
$ python rv.py stan/recover_simulated_params.stan coin_bias cat
```



Mucking about with Distributions

- stan/the_answer.stan

Place to get help

- https://discourse.mc-stan.org
- https://mailchi.mp/3544eb9ce55b/stan-this-month-4
- https://www.youtube.com/watch?v=k9sH7x8O0Y8
-