project_presentation

Ruiqi Li, Xingyu Cai

9/8/2020

Preparations

Import libraries

```
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------------------ tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.1
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts --------------------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(bayesrules)
library(rstan)
```

```
## Loading required package: StanHeaders

## 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

##
## Attaching package: 'rstan'

## The following object is masked from 'package:tidyr':
##
##     extract
```

```r
set.seed(84735)
```

Read the CSV

```r
facebook_edges <- read.csv("C:/Users/MancaiNanjolno/Desktop/final_proj/musae_facebook_edges.csv")
facebook_target <- read.csv("C:/Users/MancaiNanjolno/Desktop/final_proj/musae_facebook_target.csv")
```

Turn edges and target into data frames

```r
df_edges <- data.frame(facebook_edges)
df_target <- data.frame(facebook_target)
```

Pick the first 300 cases and assign parameters

```r
#id_1 in edges = starter ids
#id_2 in edges = target ids
#every id's info is in target.csv

#sort the edges by starter_id in the order(id_1)
order_df_edges <- df_edges[order(df_edges$id_1,decreasing=FALSE),]

#filter to the sample of first 300 cases
#starter from 0, set 303 since in this way we can have a sample of size=300
first_300_edges <- filter(order_df_edges,id_1<303)

#get all ids including all starter and their targets into a vector
total_id_300 <- c(first_300_edges$id_1,first_300_edges$id_2)

#get the column of starter ids in edges into a vector
highest_start_id_300 <- c(first_300_edges$id_1)

#get the highest id number in sample (starters+targets)
max_num_of_facebook_id = max(total_id_300)

#get the highest starter id number in sample
max_num_of_edge_start = max(highest_start_id_300)
```

Calculate every cases of starter's probability of matching

```r
# For each case of starter, check all targets that the starter matches
# Establish a temporary dataframe including the details of targets
# Check the starter's pagetype and all targets' pagetypes
# Calculate the each probability of matching
# Append each case of starter's probability of matching into a vector
```

```r
#create an empty vector
l <- c()

for(i in unique(highest_start_id_300)){
  #pick the starter and its information details
  start_point = filter(df_target,id==i)
  #filter all matching cases based on starter
  search_target_dataset = filter(order_df_edges, id_1 == i)
  #establish an empty dataframe
  filling_df <- data.frame(x=numeric(),y = character())

  #fill the targets and their information details into the empty dataframe
  for(j in c(search_target_dataset$id_2)){

    target_point = filter(df_target,id==j)
    filling_df <- rbind(filling_df,target_point)

  }
  #starter's pagetype
  start_point_pagetype = start_point$page_type
  #targets' pagetypes
  target_list_pagetype = filling_df[,"page_type"]

  same_type_num = 0 #counter

  #count how many targets have the same pagetype as the starter
  for(k in target_list_pagetype){
    if(start_point_pagetype == k){
      same_type_num = same_type_num + 1
    }
  }
  #calculate the probability of matching
  prob_for_each_start_edge_point = same_type_num/(length(target_list_pagetype))
  #append the probability to the vector
  l <- c(l,prob_for_each_start_edge_point)
}
```

Hypothesis Testing

Observing sample of 300 of matching results

Test whether the matching rate is bigger than 0.5

Ho: matching probability is less than or equal to 0.5

Ha: matching probability is bigger than 0.5

```r
#Since the data relates to the continuous probability
#Use Beta Prior

#Sample Size and successful trials
```

```r
n = 300

x = 0
for(i in l){
  if(i==1){
    x = x + 1
  }
}

#Calculate alpha and beta parameters
alphabeta <- function(m,v){
  alpha_beta = c()
  alpha = m*m * ((1-m)/v - 1/m)
  beta = alpha * (1/m - 1)
  alpha_beta = c(alpha_beta,alpha)
  alpha_beta = c(alpha_beta,beta)
  return(alpha_beta)
}

#get beta prior's alpha and beta
ab = alphabeta(mean(l),var(l))

alpha = ab[1]
beta = ab[2]

############summarize_beta_binomial(alpha,beta,x,n)#############
```

Approximate the Posterior

```r
# Approximate the Posterior with rstan

#Create Model
beta_binomial_model <- "
    data {
      real<lower=0> alpha;
      real<lower=0> beta;
      int<lower=1> n;
      int<lower=0,upper=n> x;
    }

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

    model{
      x ~ binomial(n,pi);
      pi ~ beta(alpha, beta);
    }
  "

#fit approximate posterior by stan function
```

```r
fb_model <- stan(model_code = beta_binomial_model, data = list(x = x, n = n, alpha = alpha, beta = beta
     chains = 4, iter = 5000*2)
```

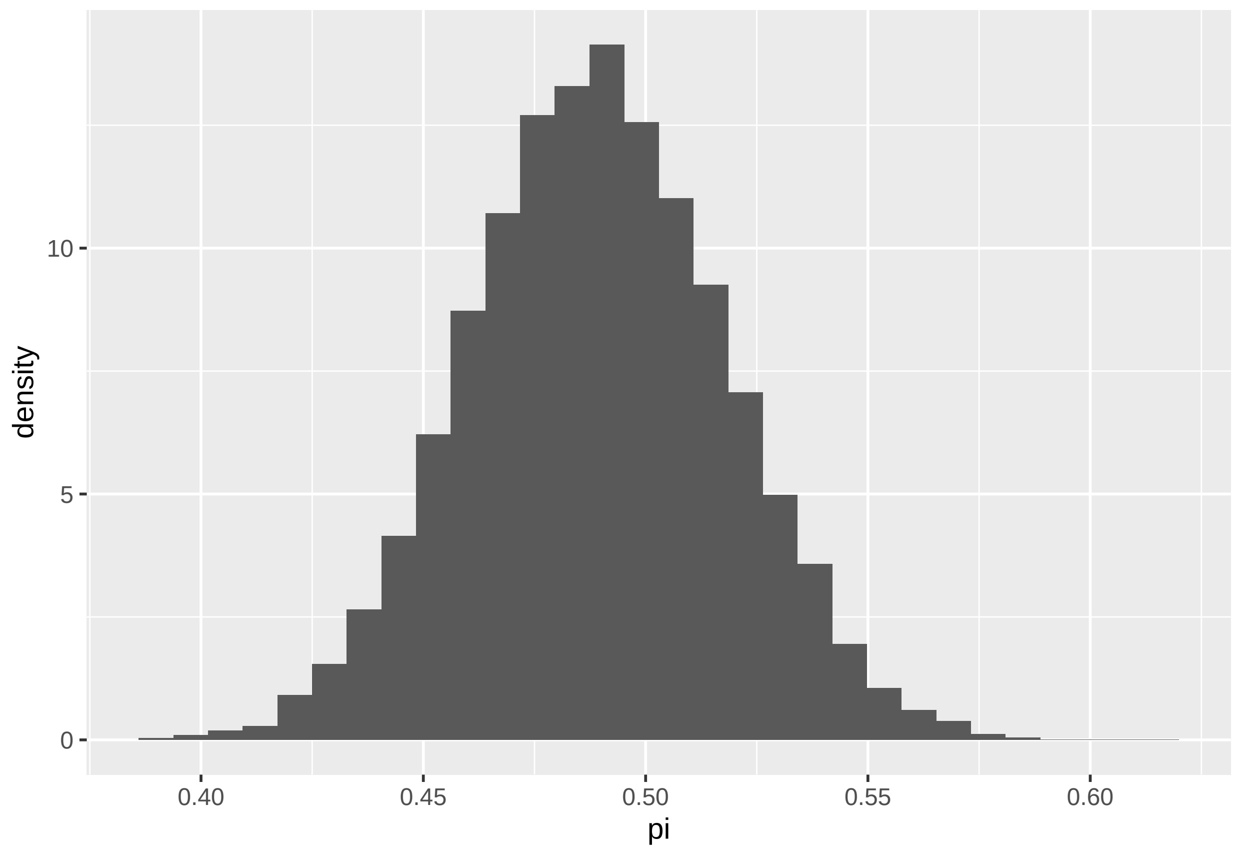
```
##
## SAMPLING FOR MODEL '2fb7dab4c784df43e953a84fc620e6a5' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.059 seconds (Warm-up)
## Chain 1:                0.074 seconds (Sampling)
## Chain 1:                0.133 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '2fb7dab4c784df43e953a84fc620e6a5' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 2: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 2: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 2: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 2: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 2: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 2: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 2: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 2: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 2: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 2: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 2: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.066 seconds (Warm-up)
## Chain 2:                0.063 seconds (Sampling)
## Chain 2:                0.129 seconds (Total)
## Chain 2:
```

```
##
## SAMPLING FOR MODEL '2fb7dab4c784df43e953a84fc620e6a5' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 3: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 3: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 3: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 3: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 3: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 3: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 3: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 3: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 3: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 3: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 3: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.052 seconds (Warm-up)
## Chain 3:                0.056 seconds (Sampling)
## Chain 3:                0.108 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '2fb7dab4c784df43e953a84fc620e6a5' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 4: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 4: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 4: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 4: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 4: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 4: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 4: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 4: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 4: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 4: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 4: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.055 seconds (Warm-up)
## Chain 4:                0.055 seconds (Sampling)
## Chain 4:                0.11 seconds (Total)
## Chain 4:
```

```r
#get approximated information into dataframe and check pi
bb_app <- as.array(fb_model, pars="pi") %>%
  reshape2::melt() %>%
  rename(pi = value)

#Posterior Approximation Histogram
ggplot(bb_app)+
  aes(x=pi) +
  geom_histogram(aes(y = ..density..))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
#Calculate and list out the mean, variance, and 95% Credible Interval
bb_app %>%
  summarize(mean(pi), var(pi), quantile(pi, c(0.025, 0.975)))
```

```
##    mean(pi)       var(pi) quantile(pi, c(0.025, 0.975))
## 1 0.4890442 0.0008247326                     0.4331614
## 2 0.4890442 0.0008247326                     0.5451336
```

The posterior mean is 0.48927

The posterior variance is 0.0008

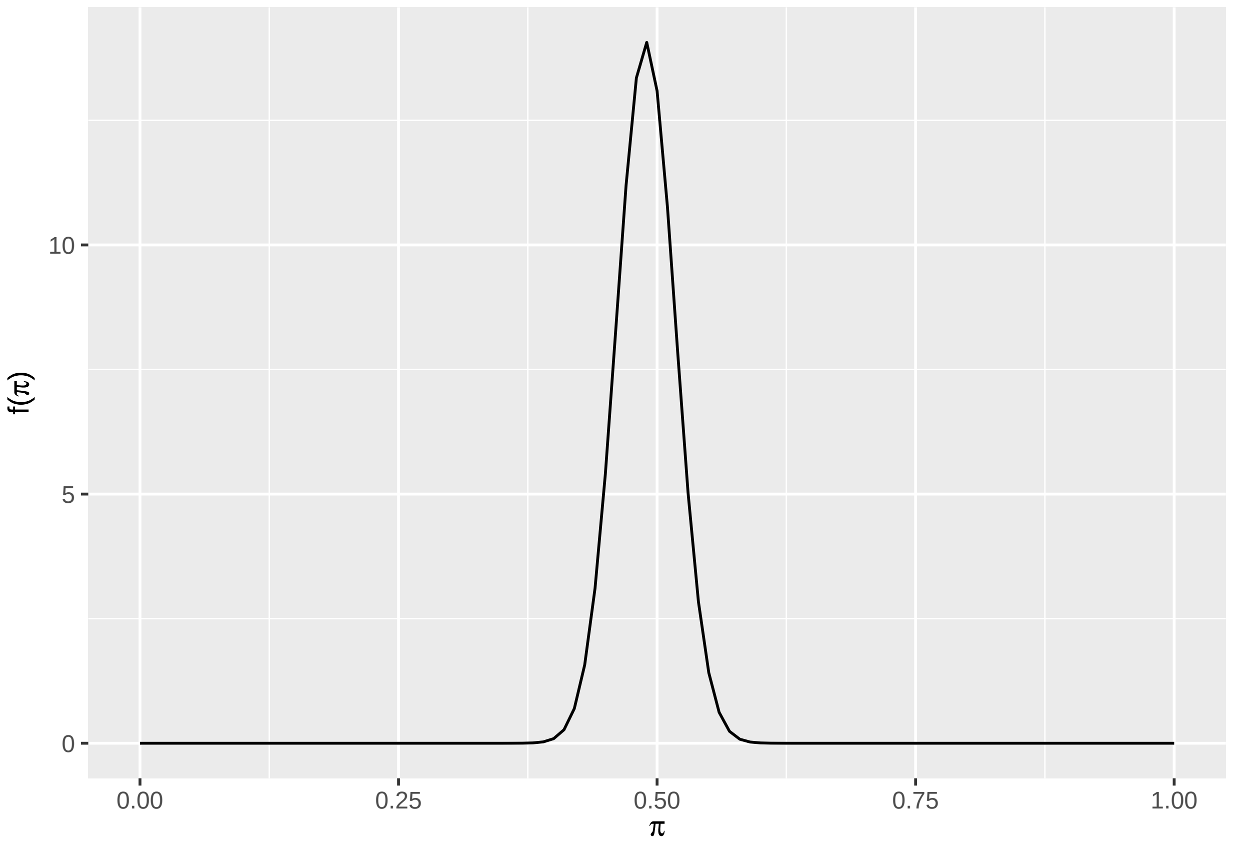
The posterior credible interval of 95% is (0.43,0.55)

Calculate Bayes Factor

```
#prior:alpha, beta
#posterior:palpha, pbeta
#x=146,n=300

#get posterior alpha and beta
pab = alphabeta(0.48927,0.0008)
palpha = pab[1]
pbeta = pab[2]

#plot posterior
plot_beta(palpha,pbeta)
```



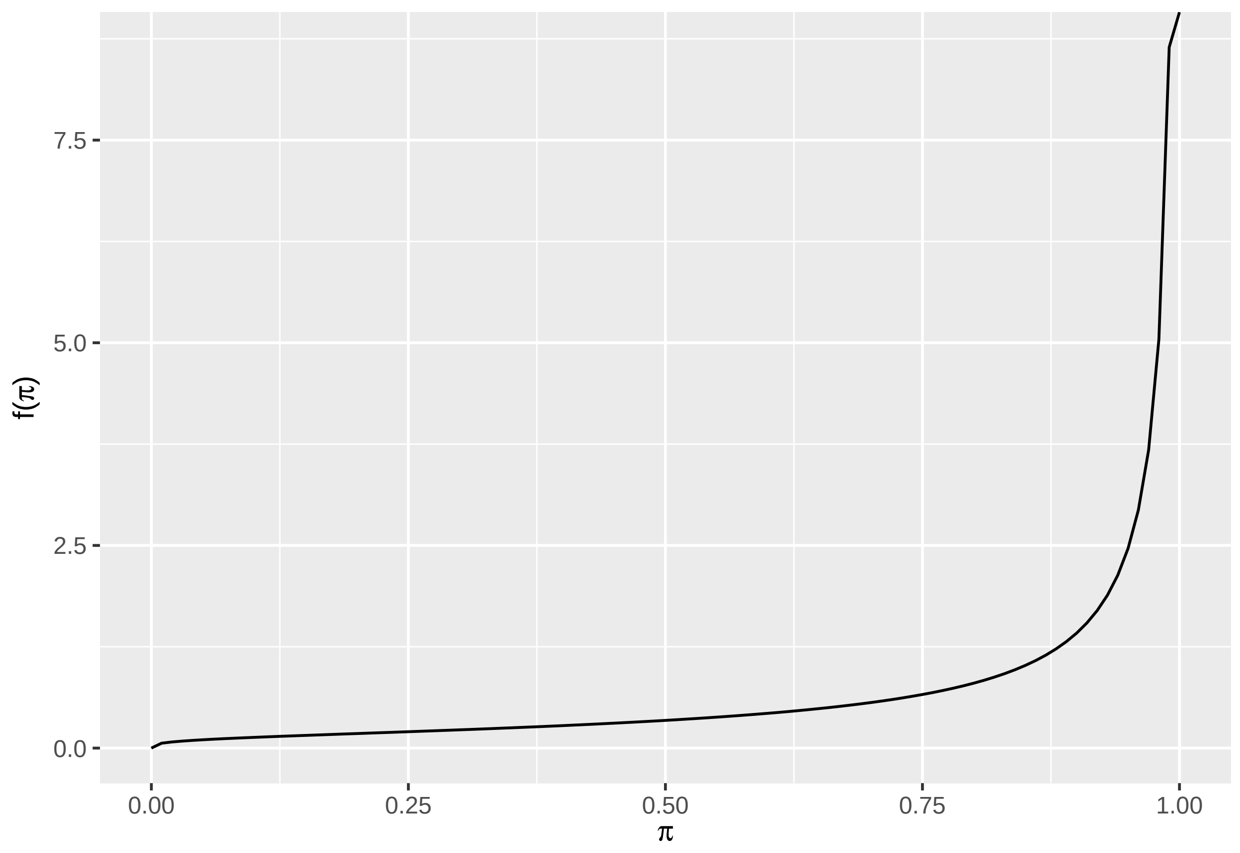
```
#calculate posterior prob for Ha
posterior_prob <- pbeta(0.5,palpha,pbeta,lower.tail=FALSE)

#compare posterior prob for Ha with posterior prob for Ho
```

```r
posterior_odds <- posterior_prob/(1-posterior_prob)

#posterior_odds < 1, Ho more likely in posterior

#calculate prior odds
#what was prob of 0.5 before the analysis
plot_beta(alpha,beta)
```



```r
prior_prob <- pbeta(0.5,alpha,beta,lower.tail=FALSE)
prior_odds <- prior_prob / (1-prior_prob)

##Bayes' Factor
bayes_factor <- posterior_odds/prior_odds
bayes_factor
```

```
## [1] 0.06142222
```

```r
#accept the null hypothesis, since the bayes' factor is less than 1, this is evident that
#null is more credible and more likely to be accepted than alternative
```

Conclusion:

Accept the Null Hypothesis due to the bayes factor is less than 1

Ho Hypothesis is more credible and more likely to be accepted than Ha

The matching probability of starter user's pagetype comparing to target user's pagetypes is less or equal to 0.5