Simulation

2019-03-31

Highlight results

1. Changes: change the monte carlo simulation calculation method; change the parameter values
2. The simulation results is good, the max value is exactly the true value
3. Tried this method on the hcaf dataset. The selected covariates combination is consistent to the results that found in the paper.

More details

Procedures of simulation

Step 1: Specify true α, where $||\alpha|| = 1$.

We could set $\alpha = (sin(\theta_0), cos(\theta_0))$. θ_0 is the true θ to make α equal to the true α. Here we choose the true θ as $\frac{\pi}{3}$, then $\alpha_{true} = (sin(\frac{\pi}{3}), cos(\frac{\pi}{3})) = (\frac{\sqrt{(3)}}{2}, \frac{1}{2}) \approx (0.866, 0.5)$.

Step 2: Specify β, Γ, D for drug group (drg) and placebo group (pbo)

The outcome follows the formula:

$$\text{the outcome is } Y : \mathbf{Y} = \mathbf{X}(\beta + \mathbf{b} + \mathbf{\Gamma}(\alpha' \mathbf{x})) + \epsilon.$$

where

- The baselines come from the same distributions $\mathbf{x} = [x_1, x_2]$, the baseline covariate x_1, x_2, iid $\sim N(0, 1)$
- A combination of baseline covariate w: $w = \alpha^T [x_1, x_2]$, which is $\alpha_1 * x_1 + \alpha_2 * x2$
- X: the independent covariates, $X = [1, t, t^2]$, where $t = 0, 1, 2, ..., 6$

The β:

- $\beta_{drg} = \begin{bmatrix} 0 \\ -5 \\ -3 \end{bmatrix}$, $\beta_{pbo} = \begin{bmatrix} 0 \\ 3 \\ 1 \end{bmatrix}$

The Γ:

- $\Gamma_{drg} = \begin{bmatrix} 0 \\ -2 \\ 1 \end{bmatrix}$, $\Gamma_{pbo} = \begin{bmatrix} 0 \\ 2 \\ -1 \end{bmatrix}$

The ϵ:

- The ϵ_{drg}, $\epsilon_{pbo} \sim N(0, 1)$

The bi:

- The random effect covariance matrix D_{drg}:

```r
set.seed(21)
eign1 = diag(c(1,0.4,0.5))
eign2 = matrix(runif(9,0,1),3,3)
a = eign2 %*% eign1 %*% solve(eign2)
bi_sigma = t(a) %*% (a) # make sure it is positive defined
round(bi_sigma,3)
```

```
##        [,1]   [,2]   [,3]
## [1,]  1.450 -0.110  0.203
## [2,] -0.110  0.165 -0.077
## [3,]  0.203 -0.077  0.229
```

- The random effect covariance matrix D_{pbo}:

```r
set.seed(7)
eign1 = diag(c(1,0.5,2))
eign2 = matrix(runif(9,0,2),3,3)
bi_sigma2 = eign2 %*%  t(eign2) /10 # make sure it is positive defined
round(bi_sigma2,3)
```

```
##       [,1]  [,2]  [,3]
## [1,] 0.439 0.296 0.090
## [2,] 0.296 0.465 0.160
## [3,] 0.090 0.160 0.267
```

Step 3: Simulate n observations (e.g. n = 100/treatment)

For drug group, the baseline covariates are generated from N(0,1):

```r
p = 2
baseline = as.matrix(rnorm(p,0,1),p,1)
x1 = baseline[1]; x2 = baseline[2]
```

Then w is calculated as $\alpha' x$

```r
alpha = c(sin(pi/3),cos(pi/3))
w = t(alpha) %*% baseline
```

The random effect is generated from MVN, whose mean = 0 and sigma equals to the covariance matrix D

```r
bi = mvrnorm(1, c(0,0,0), bi_sigma)
```

Then the outcome is calculated

```r
yi = X%*%(beta_drg+bi+gamma_drg*w[1]) + sigma_drg*rnorm(ni,0,1)
```

The placebo group follows the same generation process.

The whole code for true data generation:

```r
### data generation
true_generation = function(alpha, p, n, ni, tt, X,
                           beta_drg, gamma_drg, bi_sigma,sigma_drg,
                           beta_pbo, gamma_pbo, bi_sigma2,sigma_pbo){
  # alpha
  set.seed(123)
  alpha = as.matrix(alpha,p,1)
  dat_drg = c()
  for(i in 1:n){
```

```
    drg_temp = NULL
    drg_temp$subj = rep(paste('drg',i,sep=''),ni)
    drg_temp$trt = rep('drg',ni)
    baseline = as.matrix(rnorm(p,0,1),p,1)
    x1 = baseline[1]; x2 = baseline[2]
    w = rep(t(alpha) %*% baseline,ni)
    drg_temp$x1 = rep(x1,ni); drg_temp$x2 = rep(x2,ni)
    drg_temp$w = w
    drg_temp$tt = tt
    bi = mvrnorm(1, c(0,0,0), bi_sigma)
    yi = X%*%(beta_drg+bi+gamma_drg*w[1]) + sigma_drg*rnorm(ni,0,1)
    drg_temp$y = yi
    dat_drg = rbind(dat_drg, as.data.frame(drg_temp))
  }

  dat_pbo = c()
  for(i in 1:n){
    pbo_temp = NULL
    pbo_temp$subj = rep(paste('pbo',i,sep=''),ni)
    pbo_temp$trt = rep('pbo',ni)
    baseline = as.matrix(rnorm(p),p,1)
    x1 = baseline[1]; x2 = baseline[2]
    w = rep(t(alpha) %*% baseline,ni)
    pbo_temp$x1 = rep(x1,ni); pbo_temp$x2 = rep(x2,ni)
    pbo_temp$w = w
    pbo_temp$tt = tt
    bi = mvrnorm(1, c(0,0,0), (bi_sigma2))
    yi = X%*%(beta_pbo+bi+gamma_pbo*w[1]) + sigma_pbo*rnorm(ni,0,1)
    pbo_temp$y = yi
    dat_pbo = rbind(dat_pbo, as.data.frame(pbo_temp))
  }
  print('True data generated')
  return(list(dat_drg = dat_drg, dat_pbo = dat_pbo))
}
```

Step 4: Monte carlo simulation

Since we would like to calculate the integral of

$$\int \frac{[f_1(z_i|w_i) - f_2(z_i|w_i)]^2}{f_1(z_i|w_i) + f_2(z_i|w_i)} dz_i$$

where we assume that $f_1(z_i|w_i)$ and $f_2(z_i|w_i)$ are two MVN for drug group and placebo group separately.

To calculate this, we could firstly generate a large dataset (10000) from a 2 by 2 standard multivariate normal distribution.

```
Xstart = mvrnorm(10000, c(0,0), diag(c(1,1)))
```

Then transform those points to MVN of drug and MVN of placebo separately.

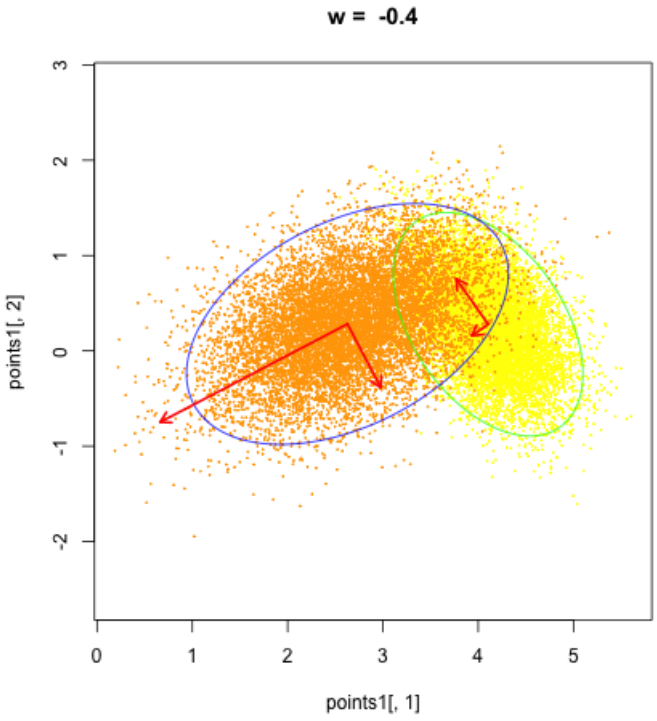
- standard MVN: $X_0 \sim MVN(\mu_0, \sigma_0)$
- MVN_drg: $X_1 \sim MVN(\mu_{drg}, \sigma_{drg})$, $\sigma_{drg} = H\Lambda H^T$, which is the eigenvalue decomposition of σ_{drg}. H is the matrix of eigen vectors and Λ is the matrix whose diagnoal values are σ_{drg}'s eignevalues.
- Transformation: $X_1 = H\Lambda^{\frac{1}{2}}X + \mu_{drg}$

And then we can get the large data points sampled from the two MVN distributions. The purity can be calculated.

```
### monta
monta_carlo_pdf = function(Xstart, mu1, D1, mu2, D2){

  # transformation
  d1 = eigen(D1)
  d1 = d1$vectors %*% diag(sqrt(d1$values))

  d2 = eigen(D2)
  d2 = d2$vectors %*% diag(sqrt(d2$values))

  points1 = Xstart %*% t(d1)

  points1[,1] = points1[,1] + mu1[1]; points1[,2] = points1[,2] + mu1[2]

  points2 = Xstart %*% t(d2)
  points2[,1] = points2[,1] + mu2[1]; points2[,2] = points2[,2] + mu1[2]

  f1 = dmvnorm(points1, mu = mu1, sigma=D1)
  f2 = dmvnorm(points2, mu = mu2, sigma=D2)

  f3 = (f1 + f2) # deal with the 0s, f1 + f2 cannot be 0
  f3 = ifelse(f3 == 0, 1e-4,f3)

  purity = c((f1 - f2)^2 / f3)
  return(purity)
}
```

To be more straightforward, we could draw the plot of beta_1 vs. beta_2 conditional on w value:

```
##
## Attaching package: 'grid'

## The following object is masked from 'package:mixtools':
##
##     depth
```



I also made a gif plot, please click the link below:

Link click here

Step 5: Specify $\theta \sim [0, \pi]$, e.g. seq(0,π,100). Calculate αs from θ

Step 6: Fit model using αs and compute purity

The functions

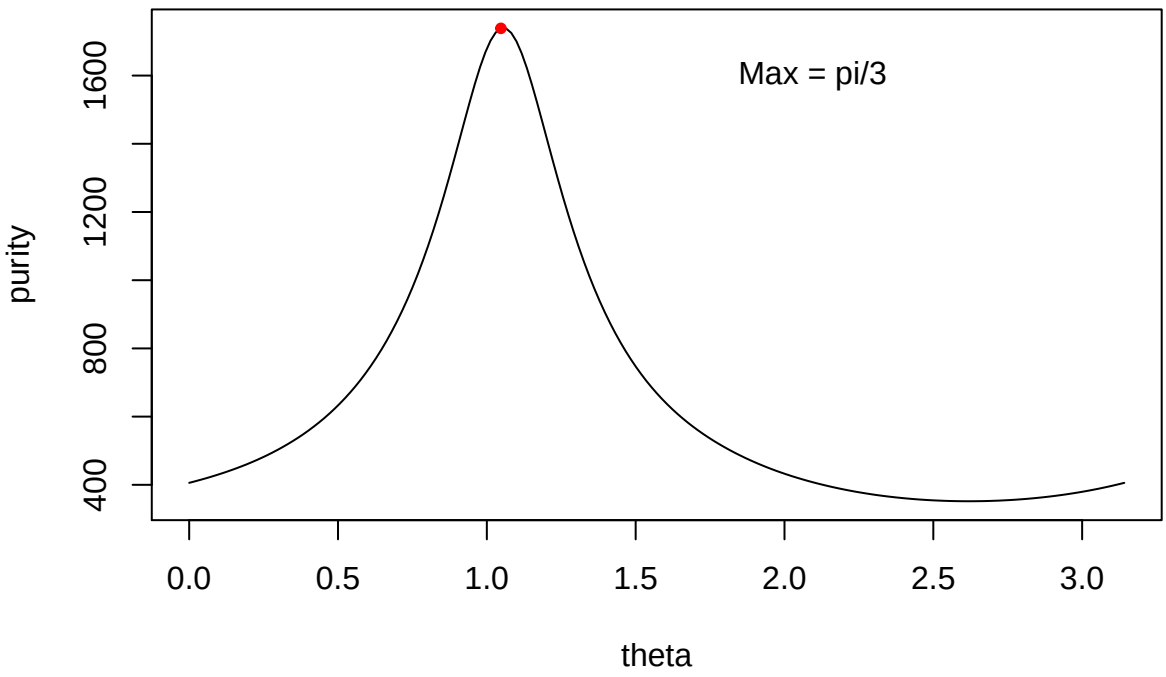
```
### purity calculation
purity_calculation = function(dat, beta1, beta2, gamma1, gamma2,
                              D1, D2, p=2){
  unique_dat = unique(dat[,c('subj','w')])
  purity = c()
  Mu1 = c(); Mu2 = c()
  for(i in 1:dim(unique_dat)[1]){
    #if(i %% 10 ==0) print(i)
    mu1 = beta1 + gamma1 * unique_dat$w[i]
    mu2 = beta2 + gamma2 * unique_dat$w[i]
    Mu1 = c(Mu1, mu1); Mu2 = c(Mu2, mu2)
    res = mean(monta_carlo_pdf(Xstart, mu1, D1, mu2, D2))
    purity = c(purity, res)
  }
  return(list(purity = purity, Mu1 = Mu1, Mu2 = Mu2))
}
purity_function = function(A, data = dat){
  alpha_est = matrix(A,2,1)
  dat_est = dat
```

```
  w_est = cbind(dat$x1,dat$x2) %*% alpha_est
  dat_est$w = w_est
  dat_pbo_est = dat_est[dat_est$trt == 'pbo', ]
  dat_drg_est = dat_est[dat_est$trt == 'drg', ]

  fit_drg_est = lmer(y ~ tt + I(tt^2) + w + w * tt +
                       w * I(tt^2) + (tt+I(tt^2)|subj),
                     data = dat_drg_est, REML = FALSE)
  fit_drg_est; beta_drg; gamma_drg
  fit_pbo_est = lmer(y ~ tt + I(tt^2) + w + w * tt +
                       w * I(tt^2) + (tt+I(tt^2)|subj),
                     data = dat_pbo_est, REML = FALSE)
  fit_pbo_est; beta_pbo; gamma_pbo

  beta1 = as.matrix(fixef(fit_drg_est))[2:3]
  gamma1 = as.matrix(fixef(fit_drg_est))[5:6]
  D1 = as.matrix(VarCorr(fit_drg_est)$subj)[2:3, 2:3]

  beta2 = as.matrix(fixef(fit_pbo_est))[2:3]
  gamma2 = as.matrix(fixef(fit_pbo_est))[5:6]
  D2 = as.matrix(VarCorr(fit_pbo_est)$subj)[2:3, 2:3]

  b = purity_calculation(dat_est, beta1, beta2, gamma1, gamma2, D1, D2, p=2)

  return(list(purity = b$purity, Mu1 = b$Mu1, Mu2 = b$Mu2,
              beta1 = beta1, beta2 = beta2,
              gamma1 = gamma1, gamma2 = gamma2))
}
```

The results

1. Simulation

As the plot, the max purity value in the simulation is exactly the one simulated with the true α



The max value is:

```
data[data$y == max(data$y),]$x/180*pi
```

```
## [1] 1.047198
```

which is approximately equal to $\frac{\pi}{3}$

```
pi/3
```

```
## [1] 1.047198
```

2. Find the max value for the purity calculation function

Since the simulation generated discrete points, the max value of those discrete points may not represent the true max value. Therefore, we would like to calculate the max for the function:

$$purity = f(\alpha)$$

For now there are two methods to solve this:

- Gradient descent
- Genetic algorithm

Gradient descent

The code for gradient descent:

```
# function: purity calculation
f = function(x){
  theta_value = x #/180*pi
  A = c(sin(theta_value),cos(theta_value))
  return(sum(purity_function(A)$purity))
```

```
}

ff = function(x){-f(x)} # make it to calculate the min

# derivate
prime1 = function(x,f){
  return((f(x+eps) - f(x))/eps)
}

next_x = 1
for(i in 1:1){ # set to 1000
  # save the plot
  #png(paste(i,'.png',sep=''))
  #plot(data$x /180 * pi, -data$y, type = 'l', cex = 1.1, ylab = 'purity',
  #     xlab = 'theta', main = paste('iteration = ',i))
  #points(data[data$y == max(data$y),]$x/180 * pi,
  #       -data[data$y == max(data$y),]$y,
  #       col = 'red', pch = 20)
  current_x = next_x
  next_x = current_x - 0.01 * prime1(current_x,ff)
  step = next_x - current_x
  #points(next_x, -f(next_x),pch = 4, col ='blue', cex = 0.8)
  #dev.off()
  #print(f(next_x))
  #print(paste('point',next_x))
  if(abs(step) <= eps){
    break
  }
}
```

For gradient descent, it doesn't converge. Maybe because the function is periodic I plot the points trajectory generated from the gradient descent (a gif file). There is the link to check it:

Link click here: points trajectory for gradient descent 1

Link click here: points trajectory for gradient descent 2

Link click here: points trajectory for gradient descent 3

But the **optim** function works very well:

```
optim(1, ff, method = 'Brent', lower = 0, upper = 3.15)
```

```
## $par
## [1] 1.058778
##
## $value
## [1] -81.15896
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
##
```

```
## $message
## NULL
```

The estimated value is around $\frac{\pi}{3} \approx 1.047$

Genetic algorithm

The gentic algorith converges very fast. It can find the correct max value at the first times.

Link click here: points trajectory for genetic algorithm

2. True data

After changing the method to do the Monte Carlo simulation and the parameters, it seems that this method works well. We may try to apply this method to the real data.

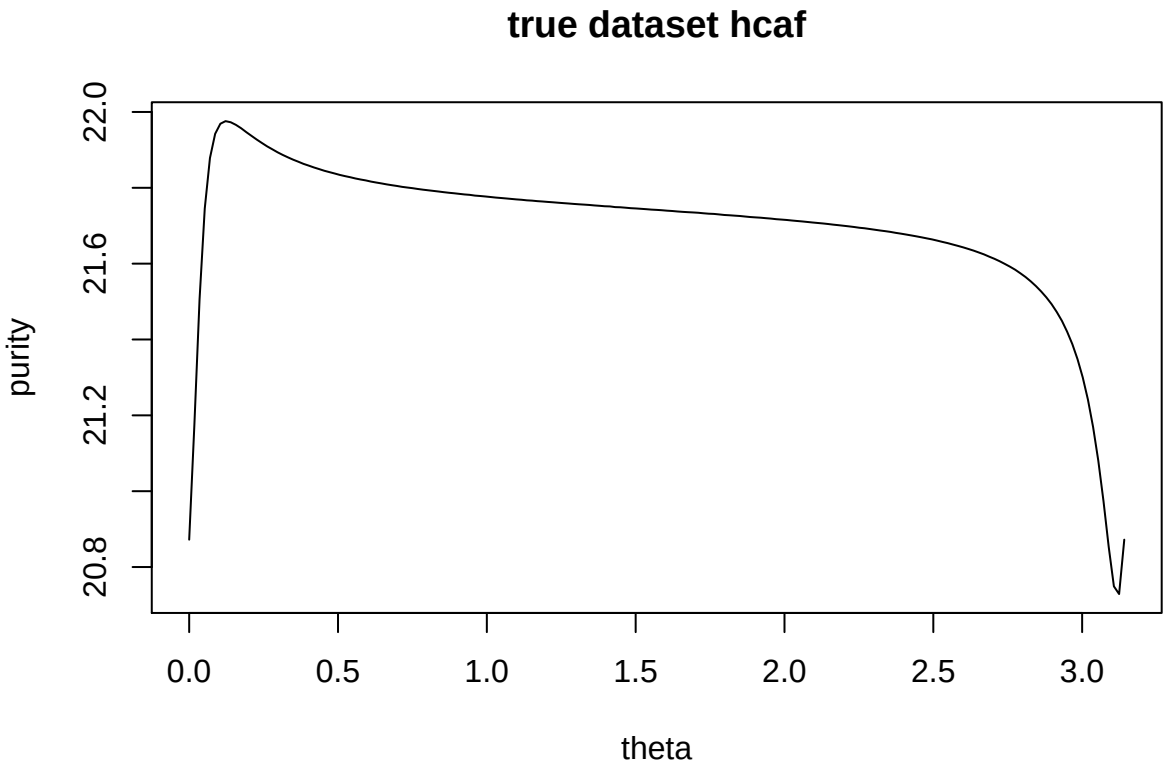
I firstly tried the hcaf dataset

```
##   subj trt  y age BaselineCGI t1 responder
## 1 2497   0 25  29           4  0         0
## 2 2497   0 18  29           4  1         0
## 3 2497   0 11  29           4  2         0
## 4 2497   0  9  29           4  3         0
## 5 2497   0 19  29           4  4         0
## 6 2497   0 15  29           4  5         0
```

The two covariates are "age" and "BaselineCGI". We could try to make combination of these two covariates as a new covariate w:

$$w = sin(\theta) * age + cos(\theta) * BaselineCGI$$

Here are the results:



The max value is

```r
data[data$y == max(data$y),]
```

```
##   x       y
## 8 7 21.9758
```

Which is about consistent to the paper's results. The "age" almost does not have effect on the clusters while "BaselineCGI" can affect the clusters.