Demo Code

This R Markdown document illustrates how to use R code to implement our method proposed in the paper. The code below can be used to conduct the simulation study reported in the main text. Below we simulate and analyze a single data set. For the final simulation we used multiple cores to analyze 100 synthetic datasets for each scenario. The four cases of the simulation study can be specified by changing type and utility in the "Data simulation" section below.

Those R packages need to be installed before running the demo code.

```
library(fields)
```

```
## Warning: package 'spam' was built under R version 3.4.4
```

```
## Warning: package 'dotCall64' was built under R version 3.4.4
```

```
## Warning: package 'maps' was built under R version 3.4.4
```

```
library(MCMCpack)
```

```
## Warning: package 'MCMCpack' was built under R version 3.4.4
```

Set working directory where you save all the code files:

```
path <- "~/Demo_code"
setwd(path)
source("functions.R")
source("MCMC_DPM.R")
```

Data simulation

Set simulation design and optimization details:

```
type <- 2  # 1 for simulating a single group, 2 for simulating mixture groups
source("SimDesign.R")   # Set simulation parameters

m       <- 20000  # Number of MC samples per candiate alpha
n2      <- 200    # Number of steps in the sequential optimization
indX_policy <- c(1)  # Indicator of the elements in X that will be a feature in the policy
q       <- length(indX_policy)+2    # Number of policy features
utility <- "red"   # Choose utility function: "red"/"ave"
```

Generate a fake dataset and also return a list of sufficient statistics X'X, X'Y and Y'Y for each patient:

```
set.seed(806)
source("Sim_One_Dataset.R")
```

Model fitting

Fit the DPM model:

```
fit <- MCMC_DPM(base,nt,XX1,XY1,YY1,XX2,XY2,YY2,iters=5000,burn=3000,update=1000)
```

```
## [1] "Done with 1000 of 5000"
## [1] "Done with 2000 of 5000"
## [1] "Done with 3000 of 5000"
## [1] "Done with 4000 of 5000"
## [1] "Done with 5000 of 5000"
```

Select a subset of the MCMC posterior samples for policy search:

```
iters <- dim(fit$b0)[1]
samps <- sample(1:iters,100,replace=FALSE)

B0  <- NULL
B1  <- NULL
B2  <- NULL
PR  <- NULL
for(draw in samps){
  B0 <- rbind(B0,fit$b0[draw,,])
  B1 <- rbind(B1,fit$b1[draw,,])
  B2 <- rbind(B2,fit$b2[draw,,])
  PR <- c(PR,fit$pg[draw,])
}
keep <- PR>0.01
B0   <- B0[keep,]
B1   <- B1[keep,]
B2   <- B2[keep,]
PR   <- PR[keep]

params_fit <- list(
  b0      = B0,
  b1      = B1,
  b2      = B2,
  tchols0 = t(chol(apply(fit$s0,2:3,mean))),
  s1      = mean(fit$s1),
  s2      = mean(fit$s2),
  prob    = PR/sum(PR))
```

Policy search

Optimize the feature weights that minimize the opposite number of the value (equivalent to maximizing the value):

```
source("optimization.R")
```

```
## [1] "Estimate values for a grid of alphas:"
## [1] "Done with 10 of 96"
## [1] "Done with 20 of 96"
## [1] "Done with 30 of 96"
## [1] "Done with 40 of 96"
## [1] "Done with 50 of 96"
## [1] "Done with 60 of 96"
## [1] "Done with 70 of 96"
## [1] "Done with 80 of 96"
## [1] "Done with 90 of 96"
## [1] "Sequential optimization:"
## [1] "Done with 10 of 200"
## [1] "Done with 20 of 200"
## [1] "Done with 30 of 200"
## [1] "Done with 40 of 200"
## [1] "Done with 50 of 200"
## [1] "Done with 60 of 200"
## [1] "Done with 70 of 200"
```

```
## [1] "Done with 80 of 200"
## [1] "Done with 90 of 200"
## [1] "Done with 100 of 200"
## [1] "Done with 110 of 200"
## [1] "Done with 120 of 200"
## [1] "Done with 130 of 200"
## [1] "Done with 140 of 200"
## [1] "Done with 150 of 200"
## [1] "Done with 160 of 200"
## [1] "Done with 170 of 200"
## [1] "Done with 180 of 200"
## [1] "Done with 190 of 200"
## [1] "Done with 200 of 200"
```

Store the output

```
# Estimate the threshold corresponding to the estimated optimal alpha
thresh  <- get.thresh(alpha_est,params_fit,m=50000,policy=riskscore,
                      indX_policy=indX_policy)
# Compute the value corresponding to the estimated optimal alpha using fitted model
v1      <- get.value(alpha_est,params_fit,m=1000000,thresh=thresh,policy=riskscore,
                     utility=utility,indX_policy=indX_policy)
# Compute the value corresponding to the estimated optimal alpha using oracle model
v2      <- get.value(alpha_est,params_or,m=1000000,thresh=thresh,policy=riskscore,
                     utility=utility,indX_policy=indX_policy)

cat("The estimated optimal feature weights alpha: ",alpha_est)
```

```
## The estimated optimal feature weights alpha:  0.385451 -0.408901 0.8271805
```

```
cat("The threshold: ",thresh)
```

```
## The threshold:  -0.2436131
```

```
cat("The value, standard error of the value, cost and standard error of the cost
    corresponding to the estimated optimal alpha using fitted model:\n",c(-v1[1],v1[2:4]))
```

```
## The value, standard error of the value, cost and standard error of the cost
##     corresponding to the estimated optimal alpha using fitted model:
##  -0.8431184 0.00182606 6.007857 0.002143215
```

```
cat("The value, standard error of the value, cost and standard error of the cost
    corresponding to the estimated optimal alpha using oracle model:\n",c(-v2[1],v2[2:4]))
```

```
## The value, standard error of the value, cost and standard error of the cost
##     corresponding to the estimated optimal alpha using oracle model:
##  -0.8512397 0.001871988 6.015173 0.002126593
```