225 homework 3

Jing Liao

Consider the stroke diagnostic problem from Assignment 2. This time, use a Gaussian process classification model to predict stroke.

For categorical outcome variable, we assume the Gaussian process prior over a continous latent function, u(x). We definte the distribution of the response variable interms of this latent function. The outcome variable in our problem is binary, I woould use the following logistic model

$$P(y_i = 1|u(x_i)) = \frac{exp(u(x_i))}{1 + exp(u(x_i))}$$

We need to choose an appropriate covariance function in setting Gaussian process models.

Firstly, I tried Squared exponential kernel, this kernel is most flexible kernel compared to OU and Brownian motion.

$$C_{ij} = \sigma^2 exp[-\sum_{p=1}^{p} \rho_p (X_{ip} - X_{jp})^2]$$

We have 14 predictors in our model,there is one ρ for each predictor,where $\sigma \sim U(0.1, 1)$ and $\rho_i \sim Ga(5, 5)$

```
library(R2jags)
stroke=read.table("/Users/jing/Desktop/2020 winter/Stat 225/hw2/Stroke.csv",sep=',',header=T)
stroke$Gender<-as.numeric(stroke$Gender)-1
attach(stroke)
N = dim(stroke)[1];
p = dim(stroke)[2]
Y=stroke[,16]
X = as.matrix(stroke[,2:(p-1)])
#x=scale(x[,2:14])
#x = cbind(stroke$Gender, x)
tse1 <- Sys.time()
gpse1 <- "model{

   # Likelihood

   for(i in 1:T){
    Y[i] ~ dbern(p[i])
    p[i]<-exp(u[i])/(1+exp(u[i]))
   }
   u~ dmnorm(Mu, Sigma.inv)
   Sigma.inv <- inverse(Sigma)

  # Set up mean and covariance matrix

   for(i in 1:T) {
    Mu[i] <- alpha
    Sigma[i,i] <- pow(sigma, 2)
    for(j in (i+1):T) {
      for (p in 1:14){
```

```
      C1[i,j,p]=-(rho[p] * pow(X[i,p] - X[j,p], 2))
      #C2[i,j,p]=  pow(X[i,p] - X[j,p], 2)
      }
      Sigma[i,j] <- exp(sum(C1[i,j,]))
      Sigma[j,i] <- Sigma[i,j]
    }
  }
   #Priors

  alpha ~ dnorm(0, 0.01)
  sigma ~ dunif(0, 1)
  for (p in 1:14){
  rho[p] ~ dgamma(5, 5)
  }

  }"

dat<- list(Y=Y,T=100,X=X)
jags.param=c('alpha','rho','sigma')
fitse1 <- jags(data=dat, n.chains=4, inits=NULL,parameters=jags.param, n.iter=3000, n.burnin=1000,
               model.file=textConnection(gpse1))
tse2 <- Sys.time()
tse2-tse1
print(jagsfitse1)
```

Since this model is flexible and takes a long time to run. It take about 1.5 hours to run with DIC 137.6.

Then I tried the OU kernel,

$$C_{ij} = \sigma^2 exp[-\sum_{p=1}^{p} \frac{1}{\rho}|X_{ip} - X_{jp}|]$$

where $\sigma \sim U(0.1, 1)$ and $\rho_i \sim U(1, 2)$

```
to1 <- Sys.time()
gpou <- "model{

   # Likelihood

   for(i in 1:T){
    Y[i] ~ dbern(p[i])
    p[i]<-exp(u[i])/(1+exp(u[i]))
   }
   u~ dmnorm(Mu, Sigma.inv)
   Sigma.inv <- inverse(Sigma)

  # Set up mean and covariance matrix

   for(i in 1:T) {
    Mu[i] <- alpha
    Sigma[i,i] <- pow(sigma, 2)
    for(j in (i+1):T) {
      for (p in 1:14){
      C[i,j,p]=(1/rho[p])*abs(X[i,p]-X[j,p])
      }
      Sigma[i,j] <- pow(sigma, 2) * exp(-sum(C[i,j,]))
```

```
      Sigma[j,i] <- Sigma[i,j]
    }

  }
   #Priors

  alpha ~ dnorm(0, 0.01)
  sigma ~ dunif(0.1, 1)
  for (p in 1:14){
  rho[p] ~ dunif(1, 2)
  }



  }"

dat<- list(Y=Y,T=100,X=X)
jags.param=c('alpha','sigma',"p")
fitou <- jags(data=dat, n.chains=4, inits=NULL,parameters=jags.param, n.iter=3000, n.burnin=1000,
              model.file=textConnection(gpou))
to2 <- Sys.time()
to2-to1
print(fitou)
```

The OU kernl Gaussian process takes about 1 hours to run with DIC 135 which it is time consuming.

Since Brownian montin is still simpler than the OU process,the computational cost of inverting C is O(n)

$$C_{ij} = \sum_{p=1}^{p} \sigma_p^2 min(X_{ip}, X_{jp})$$

where $\sigma_p \sim U(1, 2)$

```
gpbm1 <- "model{

   # Likelihood

   for(i in 1:T){
    Y[i] ~ dbern(p[i])
    p[i]<-exp(u[i])/(1+exp(u[i]))
   }
   u~ dmnorm(Mu, Sigma.inv)
   Sigma.inv <- inverse(Sigma)

  # Set up mean and covariance matrix

   for(i in 1:T) {
    Mu[i] <- alpha
    for (p in 1:14){
    C[i,i,k]=sigma[p]*X[i,k]
    }
    Sigma[i,i] <- sum(C[i,i,])
    for(j in (i+1):T) {
      for (p in 1:14){
```

Kernel	DIC	Running time(minutes)
Squared exponential	137.6	150
OU process	135	120
Brownian motion	53	25

Table 1: Comparison of three kernel

```
      C[i,j,p]=sigma[p]*min(X[i,p],X[j,p])
      }
      Sigma[i,j] <- sum(C[i,j,])
      Sigma[j,i] <- Sigma[i,j]
    }
  }
   #Priors

  alpha ~ dnorm(0, 0.01)
  for (p in 1:14){
  sigma[p] ~ dunif(0.1, 2)
  }

 }"
dat<- list(Y=Yr,T=100,X=Xr)
jags.param=c('alpha','sigma')
fitbm1 <- jags(data=dat, n.chains=4, inits=NULL,parameters=jags.param, n.iter=3000, n.burnin=1000,
               model.file=textConnection(gpbm1))
```

The Brownian montin Gaussian process takes about 25 minutes to run with DIC 53, which is much more computationally convenient. Hence I choose Brownian montion as kernel function with hyperparameter σ_p $U(1,2)$

The comparsion of 3 kernel function is shown as below:

Since the value of $sigma_p$ are very close to each other, I would simplyfy the model that all the predictor share same priors $\sigma \sim U(1,2)$ in 5-fold cross validation for computaionlly convenient.

```
library(R2jags)
```

```
## Loading required package: rjags
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.0
```

```
## Loaded modules: basemod,bugs
```

```
##
## Attaching package: 'R2jags'
```

```
## The following object is masked from 'package:coda':
##
##     traceplot
```

```
stroke=read.table("/Users/jing/Desktop/2020 winter/Stat 225/hw2/Stroke.csv",sep=',',header=T)
stroke$Gender<-as.numeric(stroke$Gender)-1
attach(stroke)

strokecv<-stroke[sample(nrow(stroke)),]
#Create 5 equally size folds
```

```
folds <- cut(seq(1,nrow(strokecv)),breaks=5,labels=FALSE)

auc2<- matrix(nrow=5,ncol=1)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
tp1 <- Sys.time()
for (l in 1:5) {
  testIndexes <- which(folds==l,arr.ind=TRUE)
  testData <- stroke[testIndexes, ]
  trainData <- stroke[-testIndexes, ]
  Xr<-trainData[,2:15]
  Yr<-trainData[,16]
  Xt<-testData[,2:15]
  Yt<-testData[,16]
gpbm2 <- "model{

   # Likelihood

   for(i in 1:T){
    Y[i] ~ dbern(p[i])
    p[i]<-exp(u[i])/(1+exp(u[i]))
   }
   u~ dmnorm(Mu, Sigma.inv)
   Sigma.inv <- inverse(Sigma)

  # Set up mean and covariance matrix

   for(i in 1:T) {
    Mu[i] <- alpha
    Sigma[i,i] <-sum(X[i,])*pow(sigma, 2)
    for(j in (i+1):T) {
      for (p in 1:14){
      C[i,j,p]=min(X[i,p],X[j,p])
      }
      Sigma[i,j] <- pow(sigma, 2) * sum(C[i,j,])
      Sigma[j,i] <- Sigma[i,j]
    }
  }
   #Priors

  alpha ~ dnorm(0, 0.01)
  sigma ~ dunif(1, 2)
```

```
  }"

dat<- list(Y=Yr,T=80,X=Xr)
jags.param=c('alpha','sigma')
fitbm2<- jags(data=dat, n.chains=4, inits=NULL,parameters=jags.param, n.iter=3000, n.burnin=1000,
              model.file=textConnection(gpbm2))
print(fitbm2)
alpha = fitbm2$BUGSoutput$sims.list$alpha
sigma=fitbm2$BUGSoutput$sims.list$sigma
#check the posterior distribution
par(mfrow = c(1,2))
hist(alpha,30)
hist(sigma,30)

# prediction
T_new = 20
Mu = rep(mean(alpha), 80)
Mu_new = rep(mean(alpha), 20)
Sigma=matrix(rep(0,80*80),nrow=80,ncol=80)
#Sigma_new = mean(sigma)^2 * exp( -mean(rho) * outer(t, t_new, '-')^2 )

#pred_mean = Mu_new + t(Sigma_new)%*%solve(Sigma, y - Mu)

C_function = function(x1,x2,k) {
  Sigma <- matrix(rep(0, length(x1)*length(x2)), nrow=length(x1))
  for (i in 1:nrow(Sigma)) {
    for (j in 1:ncol(Sigma)) {
      Sigma[i,j] <- mean((sigma)^2) *min(x1[i],x2[j])
    }
  }
  return(Sigma)
}

for (p in 1:14){
  Sigma=Sigma+C_function(Xr[,p],Xr[,p],p)
}

Sigma_new=matrix(rep(0,20*80),nrow=20,ncol=80)
for (p in 1:14){
  Sigma_new=Sigma_new+C_function(Xt[,p],Xr[,p],p)
}




for (i in 1:80){
  if (Yr[i]==1){Yr[i]=Yr[i]-0.0001}
  if (Yr[i]==0){Yr[i]=Yr[i]+0.0001}
}

# compute the posterior predictive distribution
pred_mean = Mu_new + Sigma_new%*%solve(Sigma, log(Yr/(1-Yr)) - Mu)
```

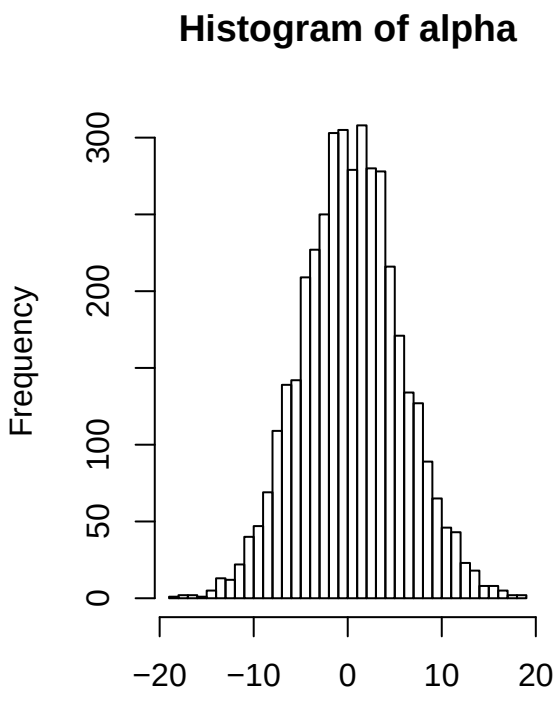
```
p=exp(pred_mean)/(1+exp(pred_mean))

auc2[l]<-auc(Yt, p)
#par(mfrow = c(1,1))
#print(roc(Yt, p,plot = T,levels=c("0", "1"), direction="<"))

}
```

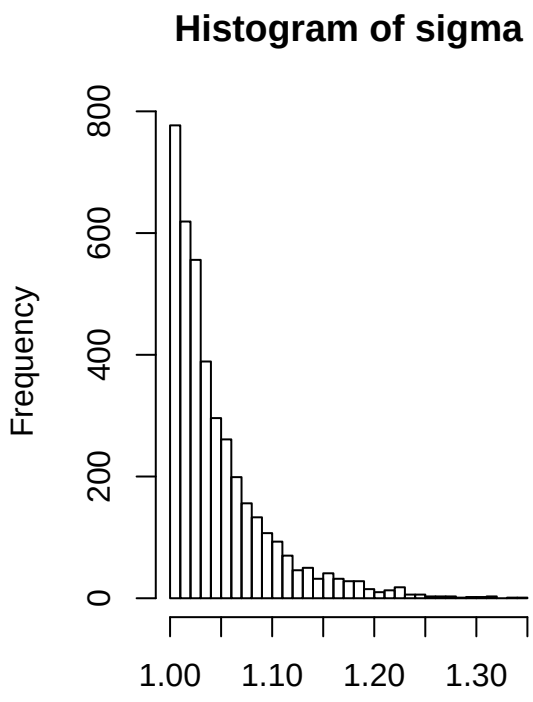
```
## module glm loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 80
##    Unobserved stochastic nodes: 3
##    Total graph size: 43390
##
## Initializing model
##
## Inference for Bugs model at "4", fit using jags,
##  4 chains, each with 3000 iterations (first 1000 discarded), n.thin = 2
##  n.sims = 4000 iterations saved
##          mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
## alpha      0.396   5.314 -9.854 -3.173  0.388  3.832 11.239 1.021   120
## sigma      1.048   0.049  1.001  1.013  1.031  1.065  1.185 1.040    93
## deviance  54.829   7.846 39.896 49.307 54.793 59.647 71.146 1.625     8
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 18.0 and DIC = 72.8
## DIC is an estimate of expected predictive error (lower deviance is better).
```

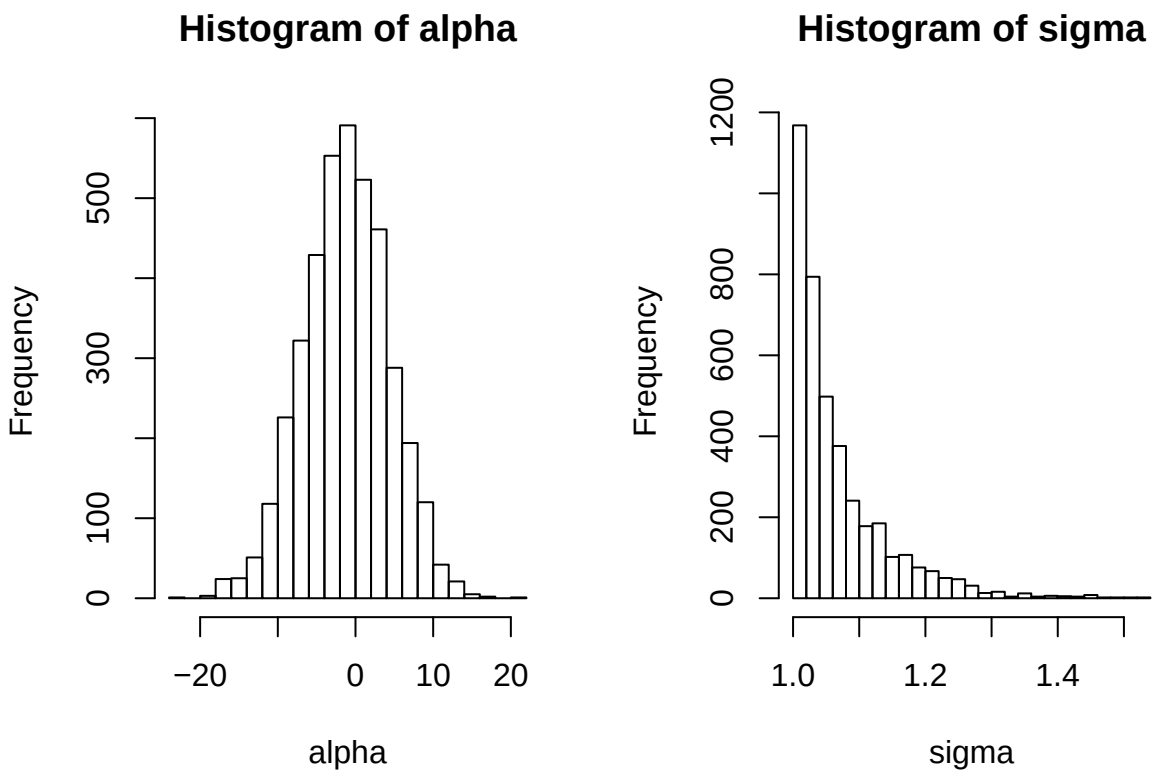
```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

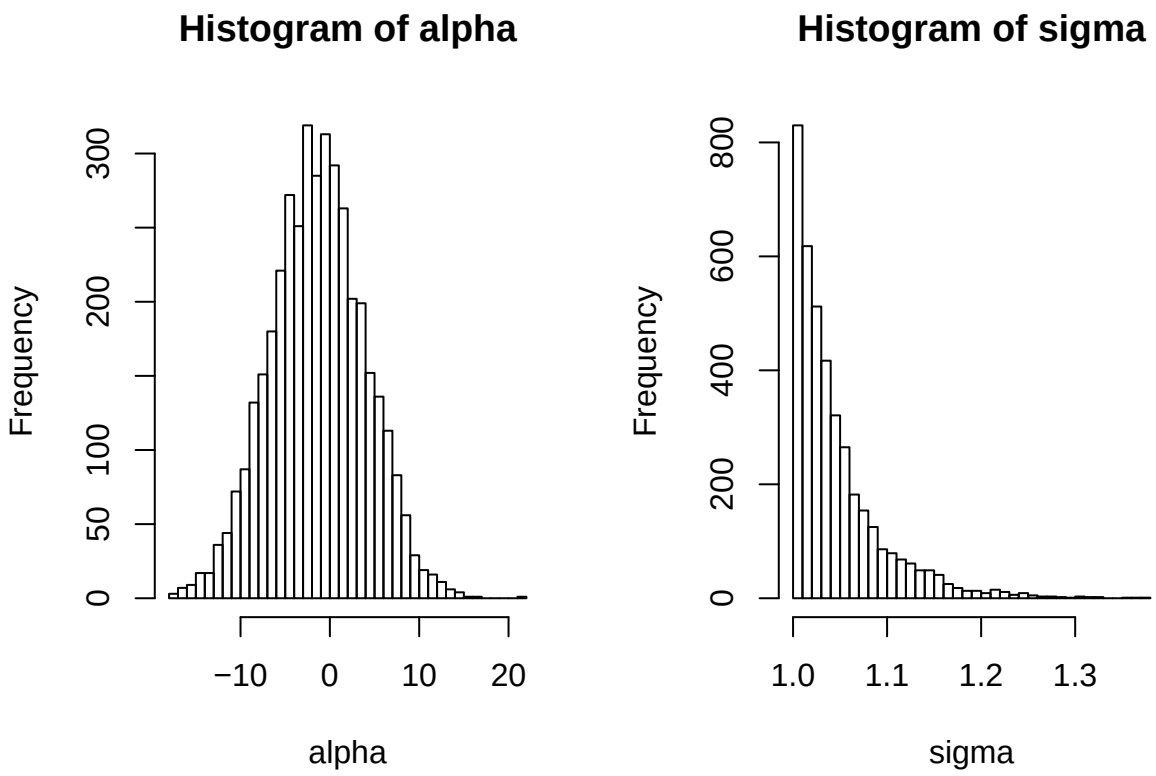




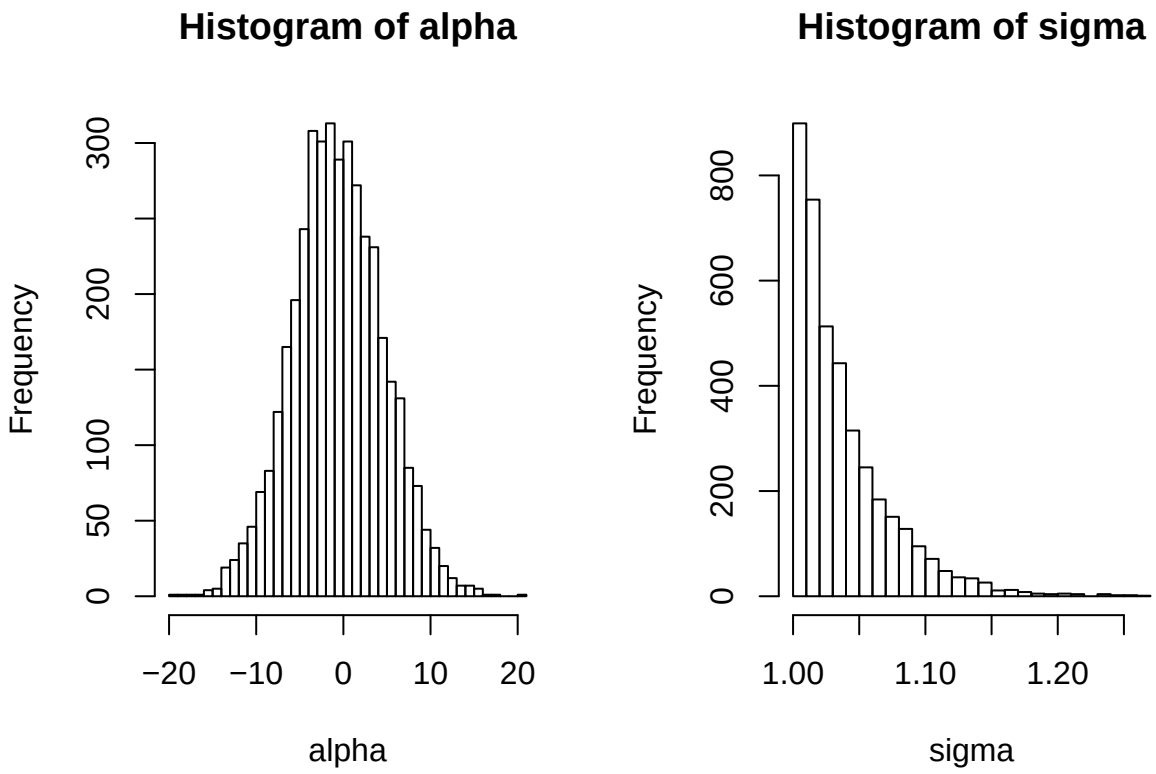

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 80
##    Unobserved stochastic nodes: 3
##    Total graph size: 43438
##
## Initializing model
##
## Inference for Bugs model at "5", fit using jags,
##  4 chains, each with 3000 iterations (first 1000 discarded), n.thin = 2
##  n.sims = 4000 iterations saved
##          mu.vect sd.vect    2.5%    25%    50%    75%  97.5%  Rhat n.eff
## alpha     -1.217   5.518 -12.112 -4.915 -1.147  2.526  9.377 1.025   110
## sigma      1.068   0.075   1.001  1.017  1.042  1.093  1.267 1.242    19
## deviance  56.811   7.845  41.197 51.567 57.202 62.335 71.055 1.976     6
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 14.0 and DIC = 70.8
## DIC is an estimate of expected predictive error (lower deviance is better).

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



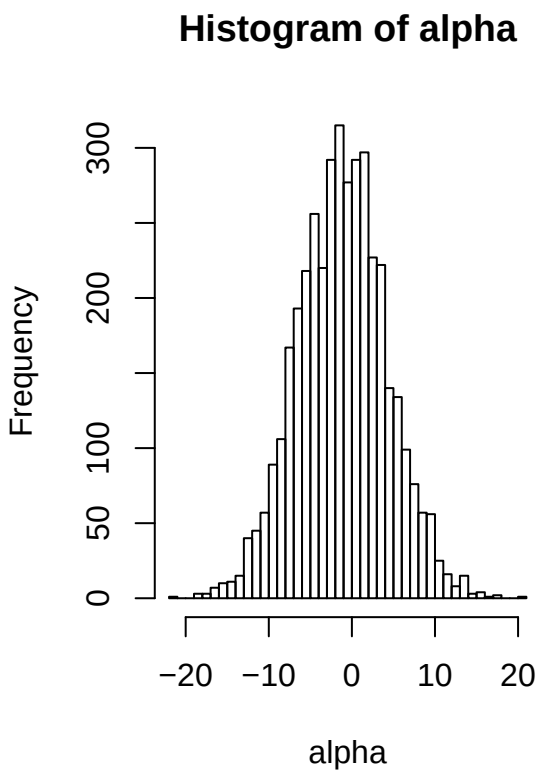

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 80
##    Unobserved stochastic nodes: 3
##    Total graph size: 43353
##
## Initializing model
##
## Inference for Bugs model at "4", fit using jags,
##  4 chains, each with 3000 iterations (first 1000 discarded), n.thin = 2
##  n.sims = 4000 iterations saved
##          mu.vect sd.vect    2.5%    25%    50%    75%  97.5%  Rhat n.eff
## alpha     -1.369   5.271 -11.734 -4.931 -1.360  2.170  8.758 1.011   240
## sigma      1.046   0.049   1.001  1.012  1.031  1.061  1.180 1.062    73
## deviance  58.377   9.089  41.529 51.902 58.396 64.679 76.043 1.635     8
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 23.4 and DIC = 81.8
## DIC is an estimate of expected predictive error (lower deviance is better).

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

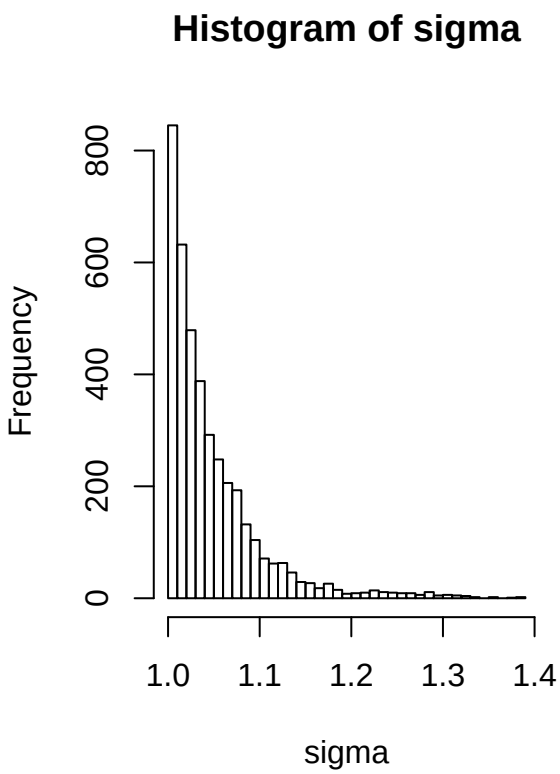



```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 80
##    Unobserved stochastic nodes: 3
##    Total graph size: 43469
##
## Initializing model
##
## Inference for Bugs model at "6", fit using jags,
##  4 chains, each with 3000 iterations (first 1000 discarded), n.thin = 2
##  n.sims = 4000 iterations saved
##          mu.vect sd.vect    2.5%    25%    50%    75%  97.5%  Rhat n.eff
## alpha     -0.630   5.170 -10.780 -4.071 -0.766  2.809  9.481 1.009   290
## sigma      1.038   0.037   1.001  1.011  1.026  1.053  1.135 1.026   160
## deviance  54.683   8.963  37.239 48.407 55.232 61.265 71.030 1.486    10
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 26.4 and DIC = 81.1
## DIC is an estimate of expected predictive error (lower deviance is better).

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```




```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 80
##    Unobserved stochastic nodes: 3
##    Total graph size: 43373
##
## Initializing model
##
## Inference for Bugs model at "4", fit using jags,
##  4 chains, each with 3000 iterations (first 1000 discarded), n.thin = 2
##  n.sims = 4000 iterations saved
##          mu.vect sd.vect    2.5%    25%    50%    75%  97.5%  Rhat n.eff
## alpha     -1.151   5.380 -11.715 -4.816 -1.218  2.314  9.553 1.030    87
## sigma      1.048   0.054   1.001  1.012  1.031  1.065  1.216 1.131    39
## deviance  54.994   9.045  41.547 47.996 53.157 60.949 74.697 1.960     6
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 17.8 and DIC = 72.8
## DIC is an estimate of expected predictive error (lower deviance is better).

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```






```
tp2 <- Sys.time()
tp2-tp1
```

```
## Time difference of 7.857199 mins
```

```
print(mean(auc2))
```

```
## [1] 0.8177529
```

The mean AUC of 5 fold cross validation in Gaussian process is 0.82 which share the similar results with logistic model I built in homework 2. Therefore, the final kenrl I choose is

$$C_{ij} = \sigma^2 \sum_{p=1}^{p} min(X_{ip}, X_{jp})$$

, where $\sigma \sim U(1, 2)$.