CVBF in BSCRN

CVBF in BSCRN

CVBFtestrsplit

data set <- 분포가 같은지 확인하고 싶은 2개의 data set

trainsize <- 훈련 데이터 크기

return : CVBF로 계산한 logBF 와 train dataset index 을 반환한다.

알아야 할 subfunction : likvec, HallKernel, bwlik2, ExpectedKernML1, laplace.kernH2c, logpriorused

```
CVBFtestrsplit = function(dataset1, dataset2, trainsize1, trainsize2, seed = NULL, train1_ids = NULL, t
{
  if(is.null(trainsize1))
  {
    stop("Please enter a trainsize for the dataset of class1.")
  }
  if(is.null(trainsize2))
  {
    stop("Please enter a trainsize for the dataset of class2.")
  }
  if(!is.null(seed))
  {
    set.seed(seed)
  }
  if(!is.null(train1_ids))
  {
    train_ids = train1_ids
  } else{
    train_ids = sample(1:length(dataset1), size = trainsize1)
  }
  if(!is.null(train2_ids))
  {
    train_ids2 = train2_ids
  } else{
    train_ids2 = sample(1:length(dataset2), size = trainsize2)
  }

  XT1 = dataset1[train_ids]
  XV1 = dataset1[-train_ids]

  YT1 = dataset2[train_ids2]
  YV1 = dataset2[-train_ids2]


  likvec = function(h) {sum(log(HallKernel(h,datagen2 = XT1, x = XV1)))}
  bwlik2 = optimize(f = function(h){  likvec(h)}, lower = 0, upper = 10, maximum = TRUE)
```

```
  ExpectedKernML1 = laplace.kernH2c(y = XT1, x = XV1, hhat = bwlik2$maximum, c = bwlik2$objective + logp

  likvec = function(h) {sum(log(HallKernel(h,datagen2 = YT1, x = YV1)))}
  bwlikcy = optimize(f = function(h){  likvec(h)}, lower = 0, upper = 10, maximum = TRUE)
  ExpectedKernML2 = laplace.kernH2c(y = YT1, x = YV1, hhat = bwlikcy$maximum, c= bwlikcy$objective + log

  likveccombc = function(h) {sum(log(HallKernel(h,datagen2 = c(XT1,YT1), x = c(XV1,YV1))))}
  bwlikcombc = optimize(f = function(h){  likveccombc(h)}, lower = 0, upper = 10, maximum = TRUE)
  ExpectedKernMLcomb = laplace.kernH2c(y = c(XT1,YT1), x = c(XV1,YV1), hhat = bwlikcombc$maximum, c= bwl

  return(list(logBF = ExpectedKernML1[1] + ExpectedKernML2[1] - ExpectedKernMLcomb[1], train1_ids = trai
}
```

likvec

$\sum \log(Hallkernel(h, TrainData, ValidData))$

Hallkernel

Compute Hall Kernel density estimate, given a training set and a set of values to evaluate on

datagen2 <- Hall 밀도추정치 계산을 위한 훈련데이터셋

h <- Bandwidth parameter

n <- train dataset 크기

sum < - $\sum_j \sqrt{8\pi e}\Phi(1)^{-1} \times \exp(-\log(1 + \frac{|X_{Valid} - X_{Train,j}|}{h}))$

return: $\frac{h}{n}$×sum vector다 validdataset크기만큼 길이를 가진다.

```
likvec = function(h) {sum(log(HallKernel(h,datagen2 = XT1, x = XV1)))}


HallKernel = function(h,datagen2,x)
{
  sum = 0
  for(i in 1:length(datagen2))
  {
    sum = sum + (((8*pi*exp(1))^.5)*pnorm(1))^(-1)*exp(-.5*(log(1+abs(x - datagen2[i])/h)^2))
  }
  return((1/(length(datagen2) * h)) * sum)
}
```

bwlik2

Bandwidth parameter인 h를 최적화 해줌 (0,10)사이에서 우도를 최대화 해주는 h값 찾는 함수

bwlik2$objective : maximize된 likvec

bwlik2$maximum : 최적화된 h 값

```
bwlik2 = optimize(f = function(h){  likvec(h)}, lower = 0, upper = 10, maximum = TRUE)
```

ExpectedKernML1

주변우도 계산하는 함수

laplace.kernH2c

Compute Marginal Likelihoods for CVBF

y <- 훈련데이터

x <- 검증데이터

hhat <- 가능도를 max해주는 Bandwidth parameter

c <- max가능도 + max log 사전 (A constant that is equal to the log likelihood + log prior evaluated at the maximum)

return : 라플라스 근사를 통해 계산한 주변우도

logintegrand.Hall

logpriorused

```
 ExpectedKernML1 = laplace.kernH2c(y = XT1, x = XV1, hhat = bwlik2$maximum, c = bwlik2$objective + logpr


laplace.kernH2c = function(y, x, hhat, c){

  n = length(x)
  out = hessian(f = function(h) {logintegrand.Hall(h, y=y, x=x, hhat = hhat)}, x = hhat, pert = 10^(-7))
  cons = c
  hess= abs(out)
  laplace = 0.5 * log(2 * pi) - 0.5 * log(hess) + cons
  c(laplace,hhat,hess)

}


logintegrand.Hall=function(h, y, x, hhat){

  n = length(x)
  beta = hhat
  Prior = log(2 * beta / sqrt(2 * pi)) - beta^2 / h^2 - 2 * log(h)
  f=loglike.KHall(h, y, x) - Prior
  return(f)

}

loglike.KHall=function(h, y, x){

  n = length(x)
  m = length(y)
  nh = length(h)
  M = t(matrix(y,m,n))
  llike=1:nh

  for(j in 1:nh){

    M1 = (x-M)/ h[j]
    M1 = KHall(M1) / h[j]
    fhat=as.vector(M1 %*% matrix(1,m,1))/m
```

```
    fhat[fhat<10^(-320)]=10^(-320)
    llike[j]=sum(log(fhat))
  }
  return(-llike)
}

KHall=function(x){

  con = sqrt(8 * pi * exp(1)) * pnorm(1)
  K=exp(-0.5 * (log(1 + abs(x)))^2) / con
  return(K)
}


logpriorused <- function(h,hhat)
{
  beta = hhat
  Prior = log(2*beta) - .5*log(pi) - 2*log(h) - (beta^2 / h^2)
  return(Prior)
}
```