Package 'Homework2'

December 4, 2013

Type Package

Title What the package does (short line)

Version 1.0

Date 2013-12-04

Author Jerryppa

Maintainer Who to complain to `<09fudansqy@gmail.com>`

Description It can generate the mld for gaussian mixture using either EM or newton method

License GPL-2

R topics documented:

`Homework2-package` *Generate mle for gaussian mixture What the package does (short line) ~~ package title ~~*

Description

This function generates the mle for gaussian mixture with a set of ovserved data.There are two options.One is Em algorithm and the other is Newton method.Also, for each method,standard error is given. More about what it does (maybe more than one line) ~~ A concise (1-5 lines) description of the package ~~

Details

Package:	Homework2
Type:	Package
Version:	1.0
Date:	2013-12-04
License:	What license is it under?

~~ An overview of how to use the package, including the most important functions ~~

Author(s)

Jerryppa Who wrote it

Maintainer: Who to complain to <yourfault@somewhere.net> ~~ The author and/or maintainer of the package ~~

References

~~ Literature or other references for background information ~~

See Also

~~ Optional links to other man pages, e.g. ~~ ~~ <pkg> ~~

Examples

```
a1<-rbinom(1000,1,0.5)
a2<-vector(length=1000)
for (i in 1:1000){
  if(a1[i]==0){
    a2[i]=rnorm(1,mean=1,sd=1)

  }
  else{a2[i]=rnorm(1,mean=2,sd=2)}

}

mixture(a2,method="EM",maxit=100)
```

---

der *To generate the gradient and Hessian of the likelihood function*

Description

To generate the gradient and Hessian of the likelihood function

Usage

```
der(xt, y, m)
```

Arguments

xt

y

m

Value

it contains several elements.One important element is gradient vector and another one is hessian matrix

Author(s)

Jerryppa

Examples

```
p0=c(0.6,10,20,60,250)
y=c(10,15,20,20,15)
l=length(y)
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (xt, y, m)
{
    gd = matrix(rep(0, 5), nrow = 1, ncol = 5)
    hes = matrix(rep(0, 25), nrow = 5, ncol = 5)
    nor = expression(log(lambda1/sqrt(2 * pi)/sqrt(sigma1) *
        exp((-1) * (y - mu1)^2/(2 * sigma1)) + (1 - lambda1)/sqrt(2 *
        pi)/sqrt(sigma2) * exp((-1) * (y - mu2)^2/(2 * sigma2))))
    gra = deriv3(nor, c("lambda1", "mu1", "mu2", "sigma1", "sigma2"))
    lambda1 = xt[1]
    mu1 = xt[2]
    mu2 = xt[3]
    sigma1 = xt[4]
    sigma2 = xt[5]
    Gra = attr(eval(gra), "gradient")
    Gras = as.matrix(apply(Gra, 2, sum))
    ss = matrix(rep(0, 25), nrow = 5, ncol = 5)
    hes = attr(eval(gra), "hessian")
    Hes = matrix(rep(0, 5 * 5), nrow = 5)
    for (i in 1:m) {
        Hes = Hes + hes[i, , ]
        ss = ss + Gra[i, ] %*% t(Gra[i, ])
    }
    vr = sqrt(diag(solve(ss)))
    im = -Hes
    IM = sqrt(1/m * diag(solve(im %*% t(im))))
    list(gd = Gras, hes = Hes, vr = vr)
  }
```

`mixture` *Generate mle for gaussian mixture*

Description

This function generates the mle for gaussian mixture with a set of ovserved data.There are two options.One is Em algorithm and the other is Newton method.Also, for each method,standard error is given.

Usage

```
mixture(y, method, maxit = NULL, tol = 1e-08, param0 = NULL)
```

Arguments

`y`	The ovserved data
`method`	The method to use
`maxit`	the maximum number of iterations
`tol`	the tolerance of the iterations
`param0`	the starting point of the iterations

Value

A list of 2. One is mle and the other is standard error.

Author(s)

Jerryppa

Examples

```
a1<-rbinom(1000,1,0.5)
a2<-vector(length=1000)
for (i in 1:1000){
  if(a1[i]==0){
    a2[i]=rnorm(1,mean=1,sd=1)

  }
  else{a2[i]=rnorm(1,mean=2,sd=2)}

}

mixture(a2,method="EM",maxit=100)

##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (y, method, maxit = NULL, tol = 1e-08, param0 = NULL)
{
    mle = vector(length = 5)
    stderr = vector(length = 5)
    if (is.null(param0) == TRUE) {
        lambda1 = 0.5
        lambda2 = 0.5
        mu1 = 10.5
        sigma1 = 59
        mu2 = 20.5
        sigma2 = 249
    }
    else {
        lambda1 = param0[1]
        mu1 = param0[2]
        mu2 = param0[3]
        sigma1 = param0[4]
        sigma2 = param0[5]
    }
    m <- length(y)
```

```
if (method == ”EM”) {
    it = 0
    if (is.null(maxit) == TRUE) {
        maxit = 500
    }
    T <- matrix(rep(0, m * 2), nrow = m, ncol = 2)
    for (i in 1:maxit) {
        ll = sum(log(lambda1 * dnorm(y, mean = mu1, sd = sqrt(sigma1)) +
            lambda2 * dnorm(y, mean = mu2, sd = sqrt(sigma2))))
        f1 = dnorm(y, mean = mu1, sd = sqrt(sigma1))
        f2 = dnorm(y, mean = mu2, sd = sqrt(sigma2))
        T[, 1] = lambda1 * f1/((lambda1 * f1) + (lambda2 *
            f2))
        T[, 2] = lambda2 * f2/((lambda1 * f1) + (lambda2 *
            f2))
        lambda1 = mean(T[, 1])
        lambda2 = mean(T[, 2])
        mu1 = sum(T[1:m, 1] * y[1:m])/sum(T[1:m, 1])
        mu2 = sum(T[1:m, 2] * y[1:m])/sum(T[1:m, 2])
        sigma1 = sum(T[1:m, 1] * (y[1:m] - mu1)^2)/sum(T[1:m,
            1])
        sigma2 = sum(T[1:m, 2] * (y[1:m] - mu2)^2)/sum(T[1:m,
            2])
        lln = sum(log(lambda1 * dnorm(y, mean = mu1, sd = sqrt(sigma1)) +
            lambda2 * dnorm(y, mean = mu2, sd = sqrt(sigma2))))
        if (abs(lln - ll) <= tol) {
            break
        }
        it = it + 1
    }
    dl = T[, 1]/lambda1 - T[, 2]/lambda2
    dmu1 = T[, 1] * (y - mu1)/sigma1
    dmu2 = T[, 2] * (y - mu2)/sigma2
    ds1 = 0.5 * T[, 1] * ((y - mu1)^2 - sigma1)/sigma1^2
    ds2 = 0.5 * T[, 2] * ((y - mu2)^2 - sigma2)/sigma2^2
    temp = rbind(dl, dmu1, dmu2, ds1, ds2)
    ss = matrix(rep(0, 25), nrow = 5, ncol = 5)
    for (i in 1:m) {
        ss = ss + temp[, i] %*% t(temp[, i])
    }
    va = sqrt(diag(solve(ss)))
    mle = c(lambda1, mu1, mu2, sigma1, sigma2)
    stderr = va
}
else if (method == ”newton”) {
    if (is.null(maxit) == TRUE) {
        maxit = 100
    }
    x = c(lambda1, mu1, mu2, sigma1, sigma2)
    it = 0
    for (i in 1:maxit) {
        ll = sum(log(lambda1 * dnorm(y, mean = mu1, sd = sqrt(sigma1)) +
            lambda2 * dnorm(y, mean = mu2, sd = sqrt(sigma2))))
        gd = der(x, y, m)$gd
        hes = der(x, y, m)$hes
        x = x - solve(hes) %*% gd
        lambda1 = x[1]
```

```
          mu1 = x[2]
          mu2 = x[3]
          sigma1 = x[4]
          sigma2 = x[5]
          lln = sum(log(lambda1 * dnorm(y, mean = mu1, sd = sqrt(sigma1)) +
              lambda2 * dnorm(y, mean = mu2, sd = sqrt(sigma2))))
          if (abs(lln - ll) <= tol) {
              break
          }
          it = it + 1
      }
      lambda1 = x[1]
      mu1 = x[2]
      mu2 = x[3]
      sigma1 = x[4]
      sigma2 = x[5]
      va = der(x, y, m)$vr
      mle = c(lambda1, mu1, mu2, sigma1, sigma2)
      stderr = va
  }
  mle = as.matrix(mle)
  stderr = as.matrix(stderr)
  rownames(mle) = c("lambda", "mu1", "mu2", "sigma1", "sigma2")
  rownames(stderr) = c("lambda", "mu1", "mu2", "sigma1", "sigma2")
  list(mle = mle, stderr = stderr)
}
```

Index