Package ‘Homework2’

December 4, 2013

Type Package

Title Mixture of two normal distributions

Version 1.0

Date 2013-12-04

Author Yuan He

Maintainer Yuan He <yhe23@jhu.eud>

Description This package lets the users to approximate mixture of two normal distributions using Newton or EM algorithm.

License GPL

R topics documented:

`Homework2-package` *Mixture of two normal distributions*

Description

Approximate mixed normal distribution using Newton or EM algorithm

Details

Package:	Homework2
Type:	Package
Version:	1.0
Date:	2013-12-04
License:	GPL

Author(s)

Yuan He

Maintainer: Yuan He <yhe23@jhu.eud>

References

https://www.cs.duke.edu/courses/spring04/cps196.1/handouts/EM/tomasiEM.pdf

Examples

```
y = c(rnorm(100,10,10),rnorm(19,50,40))
mixture(y,method="EM")
```

`mixture` *Mixture of two normal distributions*

Description

Approximate mixed normal distribution using Newton or EM algorithm

Usage

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

Arguments

`y`	y is a numeric vector
`method`	a vector of character, either "newton" or "EM"
`maxit`	The maximum iteration times
`tol`	The limitation of convergence
`param0`	Initial parameters

Details

You should choose the param0 carefully.

Value

`mle`	a vector consists of Maximum Likelihood Estimate for theta, mu1,mu2,sig1 and sig2
`stderr`	a vector consists of Standard Error for theta, mu1,mu2,sig1 and sig2

Note

Only applied to mixture of two normal distributions.

Author(s)

Yuan He

References

https://www.cs.duke.edu/courses/spring04/cps196.1/handouts/EM/tomasiEM.pdf

See Also

deriv3()

Examples

```
## Generate data
y = c(rnorm(100,3,2),rnorm(3,2,5))

## The function is currently defined as
function (y, method = c("newton", "EM"), maxit = NULL, tol = 1e-08,
    param0 = NULL)
{
    if (method == "newton") {
        if (is.null(maxit))
            maxit = 100
        if (is.null(param0)) {
            lambda = 0.5
            mu1 = 100
            mu2 = 2
            sig1 = 10
            sig2 = 49
        }
        for (i in 1:maxit) {
            ilambda = lambda
            imu1 = mu1
            isig1 = sig1
            imu2 = mu2
            isig2 = sig2
            gr = deriv3(~lambda/sqrt(2 * pi * sig1) * exp(-(y -
                mu1)^2/(2 * sig1)) + (1 - lambda)/sqrt(2 * pi *
                sig2) * exp(-(y - mu2)^2/(2 * sig2)), c("lambda",
                "mu1", "mu2", "sig1", "sig2"), function(lambda,
                mu1, mu2, sig1, sig2) {
            })
            grad = attr((gr(lambda, mu1, mu2, sig1, sig2)), "gradient")
            Grad = as.matrix(apply(grad, 2, sum))
            hes = attr((gr(lambda, mu1, mu2, sig1, sig2)), "hessian")
            Hes = matrix(rep(0, 5 * 5), nrow = 5)
            for (j in 1:n) {
                Hes = hess + hes[j, , ]
            }
            Mins = solve(Hes) %*% Grad
            lambda = lambda - Mins[1, 1]
            mu1 = mu1 - Mins[2, 1]
            mu2 = mu2 - Mins[3, 1]
            sig1 = sig1 - Mins[4, 1]
            sig2 = sig2 - Mins[5, 1]
            tolr = (lambda - ilambda)^2 + (mu1 - imu1)^2 + (mu2 -
                imu2)^2 + (sig1 - isig1)^2 + (sig2 - isig2)^2
            if (tolr < tol)
                break
        }
```

```
  }
  else if (method == "EM") {
      if (is.null(maxit))
          maxit = 500
      if (is.null(param0)) {
          lambda = 0.5
          mu1 = 100
          mu2 = 2
          sig1 = 10
          sig2 = 49
      }
      n = length(y)
      Tmax = matrix(rep(0, 2 * n), ncol = 2)
      for (i in 1:500) {
          ilambda = lambda
          imu = mu
          isig = sig
          f1 = dnorm(y, mu1, sqrt(sig1))
          f2 = dnorm(y, mu2, sqrt(sig2))
          Tmax[, 1] = lambda * f1/(lambda * f1 + (1 - lambda) *
              f2)
          Tmax[, 2] = (1 - lambda) * f2/(lambda * f1 + (1 -
              lambda) * f2)
          lambda = sum(Tmax[, 1])/n
          mu1 = sum(Tmax[, 1] * y)/sum(Tmax[, 1])
          mu2 = sum(Tmax[, 2] * y)/sum(Tmax[, 2])
          sig1 = sum(Tmax[, 1] * (y - mu1)^2)/sum(Tmax[, 1])
          sig2 = sum(Tmax[, 2] * (y - mu2)^2)/sum(Tmax[, 2])
          tolr = (lambda - ilambda)^2 + (mu1 - imu1)^2 + (mu2 -
              imu2)^2 + (sig1 - isig1)^2 + (sig2 - isig2)^2
          if (tolr < 1e-08)
              break
      }
  }
  bp = lambda * dnorm(y, mu1, sqrt(sig1))/(lambda * dnorm(y,
      mu1, sqrt(sig1)) + (1 - lambda) * dnorm(y, mu2, sqrt(sig2)))
  Slambda = bp/lambda - (1 - bp)/(1 - lambda)
  Smu1 = bp * (y - mu1)/sig1^2
  Smu2 = (1 - bp) * (y - mu2)/sig2^2
  Ssig1 = bp/(2 * sig1) * ((y - mu1)^2/sig1 - 1)
  Ssig2 = (1 - bp)/(2 * sig2) * ((y - mu2)^2/sig2 - 1)
  S = rbind(Slambda, Smu1, Smu2, Ssig1, Ssig2)
  SS = matrix(rep(0, 25), ncol = 5)
  for (i in 1:n) {
      Si = (S[, i])
      SS = SS + Si %*% t(Si)
  }
  SE = sqrt(diag(solve(SS)))
  list(mle = c(lambda = lambda, mu1 = mu1, mu2 = mu2, sigma1 = sig1,
      sigma2 = sig2), stderr = c(lambda = SE[1], mu1 = SE[2],
      mu2 = SE[3], sigma1 = SE[4], sigma2 = SE[5]))
}
```

Index