Package ‘Homework2’

December 3, 2013

Type Package

Title Advanced Statistics Computing HW1 EM and Newton Appreciation

Version 1.0

Date 2013-12-03

Author Yu Du<ydu@jhsph.edu>

Maintainer Yu Du<ydu@jhsph.edu>

Description Homework 2 in Advanced Statistics Computing Class which includes two algorithm realization.One is Newton method and the other is EM algorithm.

License GPL

R topics documented:

`Homework2-package` *Homework 2*

Description

Advanced Statistics Computing Homework 2 which includes EM algorithm and Newton algorithm.

Details

Package:	Homework2
Type:	Package
Version:	1.0
Date:	2013-12-03
License:	GPL

run mixture() function to estimate parameters for two mixed normal model.

Author(s)

Yu Du

Maintainer: Yu Du<ydu@jhsph.edu>

References

Advanced Statistics Computing Class. Dr. Peng.

`hw2_data` *Random Mixture Normal Data*

Description

This data set gives the randomly generated mixture normal data with given parameters.

Usage

```
data(hw2_data)
```

Format

A data frame with 19600 observations of mixture model.

Source

From Dr. Peng

References

Advanced Statistics Computing Class. Dr. Peng.

Examples

```
data(hw2_data)
```

mixture	*EM and Newton for Mixture Normal Parameters Estimation*

Description

Use EM or Newton algorithm to estimate parameters in the mixture normal model.

Usage

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

Arguments

`y`	y is a vector of the observed data from a mixture normal model.
`method`	method can take EM or newton depending on which algorithm people want to use.
`maxit`	maxit is the max number of iteration and default value is 100 for Newton and 500 for EM.
`tol`	tol is the tolerance to control the convergence.
`param0`	param0 is the initial parameters input for the model and in case people didn't enter any initial value, there is default specified in the program.

Details

Use EM or Newton algorithm to estimate parameters in the mixture normal model.

Value

`mle`	return the mle value for the five parameters in the mixed two normal model.
`stderr`	return the asymptotic standard error for the parameters in the mixed two normal model.

Author(s)

Yu Du

References

Advanced Statistics Computing Class, Dr. Peng.

Examples

```
z=numeric(0)
y=numeric(0)
for(i in 1:1000){
        z[i]=rbinom(1,1,0.2)
        if(z[i]==1){
                y[i]=rnorm(1,10,1)
        }else{
                y[i]=rnorm(1,3,1)
        }
```

```
}
mixture(y,"EM")


function (y, method, maxit = NULL, tol = 1e-08, param0 = NULL)
{
    y = unlist(y)
    n = length(y)
    method = match.arg(method, c("EM", "newton"))
    if (method == "EM") {
        e = numeric(0)
        tlam = numeric(0)
        tmu1 = numeric(0)
        tmu2 = numeric(0)
        tsigma1 = numeric(0)
        tsigma2 = numeric(0)
        if (is.null(param0)) {
            tlam[1] = 0.1
            tmu1[1] = 3
            tmu2[1] = 2
            tsigma1[1] = 4
            tsigma2[1] = 5
        }
        else {
            tlam[1] = param0[1]
            tmu1[1] = param0[2]
            tmu2[1] = param0[3]
            tsigma1[1] = param0[4]
            tsigma2[1] = param0[5]
        }
        if (!is.numeric(maxit)) {
            maxit = 500
        }
        for (j in 1:maxit) {
            e = tlam[j] * dnorm(y, tmu1[j], tsigma1[j]^(1/2))/(tlam[j] *
                dnorm(y, tmu1[j], tsigma1[j]^(1/2)) + (1 - tlam[j]) *
                dnorm(y, tmu2[j], tsigma2[j]^(1/2)))
            tlam[j + 1] = mean(e)
            tmu1[j + 1] = sum(y * e)/sum(e)
            tmu2[j + 1] = sum(y * (1 - e))/sum(1 - e)
            tsigma1[j + 1] = sum(e * ((y - tmu1[j + 1])^2))/sum(e)
            tsigma2[j + 1] = sum((1 - e) * ((y - tmu2[j + 1])^2))/sum(1 -
                e)
            if (sqrt((tlam[j + 1] - tlam[j])^2 + (tmu1[j + 1] -
                tmu1[j])^2 + (tmu2[j + 1] - tmu2[j])^2 + (tsigma1[j +
                1] - tsigma1[j])^2 + (tsigma2[j + 1] - tsigma2[j])^2) <=
                tol) {
                break
            }
        }
        lambda = tlam[j + 1]
        mu1 = tmu1[j + 1]
        mu2 = tmu2[j + 1]
        sigma1 = tsigma1[j + 1]
        sigma2 = tsigma2[j + 1]
        l = expression(log(lambda * exp(-(y - mu1)^2/(2 * sigma1))/sqrt(2 *
            pi * sigma1) + (1 - lambda) * exp(-(y - mu2)^2/(2 *
```

```
            sigma2))/sqrt(2 * pi * sigma2)))
        de = deriv3(l, c("lambda", "mu1", "mu2", "sigma1", "sigma2"))
        lambda = lambda
        mu1 = mu1
        mu2 = mu2
        sigma1 = sigma1
        sigma2 = sigma2
        grad = attr(eval(de), "gradient")
        score = matrix(0, 5, 5)
        for (i in 1:n) {
            score = score + grad[i, ] %*% t(grad[i, ])
        }
        cov = solve(score)
        se = c(sqrt(diag(cov)))
        return(list(mle = c(lambda = lambda, mu1 = mu1, mu2 = mu2,
            sigma1 = sigma1, sigma2 = sigma2), stderr = c(lamda = se[1],
            mu1 = se[2], mu2 = se[3], sigma1 = se[4], sigma2 = se[5])))
    }
    if (method == "newton") {
        if (is.null(param0)) {
            temp = mixture(y, "EM")[[1]]
            add = runif(4, -0.5, 0.5)
            add = as.vector(cbind(runif(1, -0.05, 0.05), t(add)))
            theta = temp + add
        }
        else {
            theta = param0
        }
        if (!is.numeric(maxit)) {
            maxit = 100
        }
        l = expression(log(lambda * exp(-(y - mu1)^2/(2 * sigma1))/sqrt(2 *
            pi * sigma1) + (1 - lambda) * exp(-(y - mu2)^2/(2 *
            sigma2))/sqrt(2 * pi * sigma2)))
        de = deriv3(l, c("lambda", "mu1", "mu2", "sigma1", "sigma2"))
        for (i in 1:maxit) {
            lambda = theta[1]
            mu1 = theta[2]
            mu2 = theta[3]
            sigma1 = theta[4]
            sigma2 = theta[5]
            grad = attr(eval(de), "gradient")
            grad = matrix(apply(grad, 2, sum), 5, byrow = T)
            hes = attr(eval(de), "hessian")
            hes = data.frame(hes)
            hes = matrix(apply(hes, 2, sum), 5, byrow = T)
            theta = theta - solve(hes) %*% grad
            if (sqrt(sum((solve(hes) %*% grad)^2)) <= tol) {
                break
            }
        }
        lambda = theta[1]
        mu1 = theta[2]
        mu2 = theta[3]
        sigma1 = theta[4]
        sigma2 = theta[5]
        grad = attr(eval(de), "gradient")
```

```
        score = matrix(0, 5, 5)
        for (i in 1:n) {
            score = score + grad[i, ] %*% t(grad[i, ])
        }
        cov = solve(score)
        se = c(sqrt(diag(cov)))
        return(list(mle = c(lamda = theta[1], mu1 = theta[2],
            mu2 = theta[3], sigma1 = theta[4], sigma2 = theta[5]),
            stderr = c(lamda = se[1], mu1 = se[2], mu2 = se[3],
                sigma1 = se[4], sigma2 = se[5])))
  }
}
```

Index