

R programs for Lip cancer data set

Approch 1

Details about the variables used are given below.

x: data file of covariates (PAFF)

y: data file of response (no of cancer)

w: adjacent matrix (value 1 for the same cluster, 2 for adjacent clusters, 0 for other clusters)

dt1: data file that contain x, y and cluster ID together

The following R function is run for given values of sigma (for σ_γ^2), and s3 (for ϕ). The estimates of parameters (β s, σ_γ^2 , and ϕ) at the time we can get the given value of σ_γ^2 , are taken as the optimal estimates.

In the function, random effects for clusters are found by using PQL approach.

```
-----  
getwd()          # get working directory  
options(digits=4) # up to 4 decimal places  
x=read.table(file.choose(),header=F, as.is=T,quote="",sep = "\t") # reading data set  
x=as.matrix(x)    # convert data set as a matrix  
y=read.table(file.choose())  
y=as.matrix(y)  
w=read.table(file.choose())  
w=as.matrix(w)  
dt<-as.data.frame(read.table("dt1.txt",header=TRUE))  
  
# loading necessary packages  
library(glmm)  
library(MASS)  
library(nlme)  
  
f<-function(sigma,s3)  
{k=56          # no of clusters(observations)  
#finding random effects for all clusters by using PQL appraoch  
ss1<-glmmPQL(y ~ 0+x, random = ~ 1 |ind,family = poisson, data = dt)  
ss2<-ss1[4]  
gamas<-ss2$coefficients$random$ind
```

```

# creating z variables based on adjacent clusters( summing x values if cluster is adjacent)
z=matrix(nrow=k,ncol=1)
i=1
while (i<=k)
{
z[i]=0
j=1
while (j<=k)
{
if (w[i,j]>0)
{
h=1
}
else
{
h=0
}
if (abs(i-j)==0)
{
h=0
}
z[i]=z[i]+h*x[j]
j=j+1
}
i=i+1
}

zz<-z    # assigning z to zz

#fitting a linear model for random effects using z variables without intercept
lf<-lm(gamas~zz-1)
p<-lf[[1]]
p1<-p[[1]]
pp<-c(p1)    #  $\theta$  vector
zzpp<-zz%*%pp    #predictor of the model for random effects(  $z\theta$  product)

# obtaining errors of model for random effects
e=matrix(nrow=k,ncol=1)
e[1:k,]=0
i=1
while (i<= k)
{
e[i]=gamas[i]-zzpp[i]
i=i+1
}
sigma2=var(e)    # variance of erro

```

```

#Estimation of beta
beta=as.matrix(c(0)) # initial values for beta
for (p in 1:20)
{
beta=as.matrix(beta) # as a matrix

mst=matrix(nrow=k,ncol=1) # creating conditional mean vector
i=1
while (i<= k)
{
mst[i]=exp(x[i]%*%beta+gamas[i])
i=i+1
}
# finding errors
ee=matrix(nrow=k,ncol=1)
ee[1:k,]=0
i=1
while (i<= k)
{
ee[i]=y[i]-mst[i]
i=i+1
}
sigma3=var(ee) # variance of errors

#Creating meu values(unconditional means)
meu=matrix(nrow=k,ncol=1)
i=1
while (i<=k)
{
meu[i]=exp(x[i,]%*%beta+0.5*sigma+zz[i,]%*%pp)
i=i+1
}

sigmam=as.matrix(rep(sigma2,k)) # column matrix with value of sigma
xb=x %*% beta # multipling x and beta
meu=exp(xb+0.5*sigmam+zz%*%pp) #Creating meu vector
# creating the covariance matrix by taking spatial dependence into consideration
xd=matrix(nrow=k, ncol=k)
xd[1:k,]=0
i=1
while (i<=k)
{
j=1
while (j<=k)

```

```

{
if (w[i,j] == 0)
{
s=0
delta=0
h=0
}
else if (w[i,j] ==1)
{
s=1
delta=1
h=1
}
else
{
s=s3
delta=0
h=1
}
xd[i,j]=exp((x[i,]+h*x[j,])%%beta+(zz[i,]+h*zz[j,])%%pp+sigma)*(exp(s*sigma)-
1)+delta*sigma3
j=j+1
}
i=i+1
}

covf=xd          # Final covarince matrix
#creating derivatives of meu
trx=t(x)         # transpose of x
difmeu=matrix(nrow=1,ncol=k) #creating 2*k matrix
for (i in 1:k)
{
difmeu[,i]=trx[,i]*meu[i] # derivatives of meu
}
#inverse of covef
library(MASS)    # package for invers of matrix
incovf=ginv(covf,tol = sqrt(.Machine$double.eps)) #inverse of covef
# creating terms for convergence equation
indf=ginv(difmeu%%incovf%%t(difmeu))
ydf=difmeu%%incovf%%(y-meu)
beta=beta+indf%%ydf # iterative equation
beta=as.numeric(beta)
xb=x %% beta
cat("beta",beta,"\b\n")
}
cat(beta, file="beta.txt", sep ="\t","\n",append=TRUE) # writing all beta values to a file

```

```

xb=x %*% beta # multipling x and beta
uvec=y*y      # squared responses
sigma=0.1     # initial value for sigma

```

```

#Marginal GQL for sigma#

```

```

beta=as.matrix(beta) # as matrix
mst=matrix(nrow=k,ncol=1)
i=1
while (i<= k)
{
mst[i]=exp(x[i]%*%beta+gamas[i])
i=i+1
}
# finding errors
ee=matrix(nrow=k,ncol=1)
ee[1:k,]=0
i=1
while (i<= k)
{
ee[i]=y[i]-mst[i]
i=i+1
}
sigma3=var(ee) # variance of errors

```

```

#Repeat from this step onwards

```

```

for (j in 1:25)
{
if (sigma > 0) # to keep process running by keeping sigma as positive
{
sigmam=rep(sigma,k) # column vector of sigma2
sigma3=rep(sigma3,k) # column vector of sigma3
lamda=exp(2*xb+2*sigmam+2*zz%*%pp)+sigma3 # mean of squared responses
#creating cov of capital U
covsq=matrix(nrow=k,ncol=k) # creating k*k matrix to store covariances
covsq[1:k,]=0
i=1
while (i<=k)
{
j=1
while (j<=k)
{
if (w[i,j]==0)
{
s=0
delta=0

```

```

h=0
} else if (w[i,j] ==1)
{
s=1
delta=1
h=1
}
else
{
s=s3
delta=0
h=1
}
covsq[i,j]=exp(2*(x[i,]+h*x[j,])%%beta+2*(zz[i,]+h*zz[j,])%%pp+4*sigma)*(exp(4*s*sigma)
)-1)+delta*(exp(2*x[i,]%%beta+2*zz[i,]%%pp+2*sigma)*sigma3+sigma3*sigma3)
j=j+1
}
i=i+1
}
# Final covariance matrix of U(squared ys)
bsigma=covsq # covariance matrix of squared responses
inbsig=ginv(bsigma,tol = sqrt(.Machine$double.eps)) #inverse of bsigma
diflamda=2*exp(2*xb+2*sigmam+2*zz%%pp) # derivatives of lamda
#Convergence equation for sigma
inbdf=ginv(t(diflamda)%%inbsig%%diflamda)
udf=t(diflamda)%%inbsig%%(uvec-lamda)
prd=inbdf%%udf
sigma=sigma+prd # iterative equation for sigma1
sigma=as.numeric(sigma)
if (sigma>0 && sigma<50) # to keep process running by avoiding large values
{
sigma=as.numeric(sigma)
cat("sigma",sigma,"\b\n")
}
else{
sigma=0.2
cat("sigma",sigma,"\b\n")
}
}
}
cat(sigma,file="sigma.txt",sep ="t", "\n", append=TRUE)
}
}

```