Dokumentasi R-Cluzzy Possibilistic FCM.

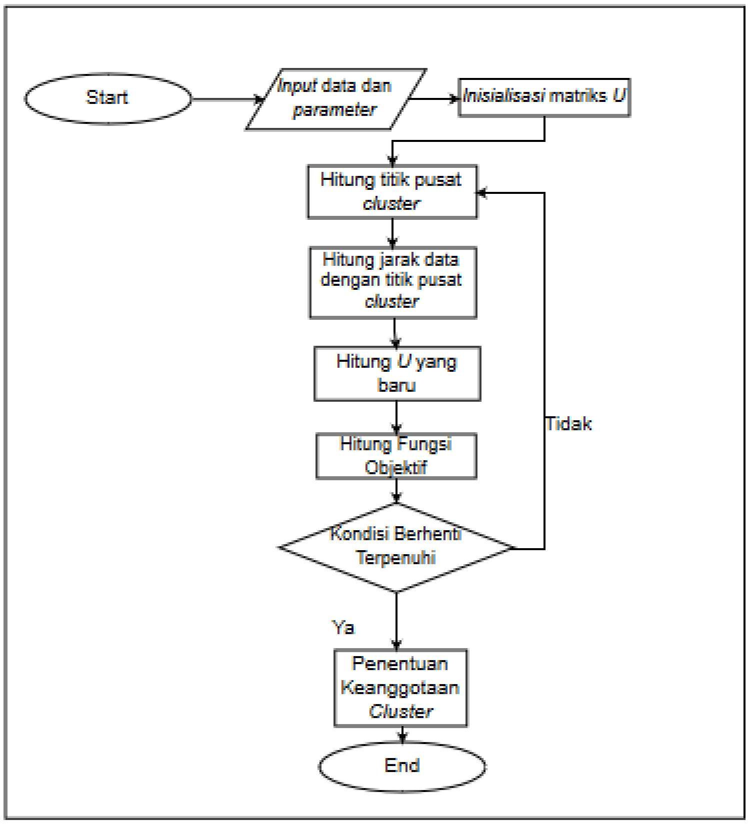
System Requirement :

- Processor: Intel® Core™ i3-4030U CPU @ 1.90GHz, 1900 Mhz, 2 Core(s), 4 Logical Processors.
- RAM: 4GB DDR3L
- Kebutuhan Harddisk:
    - +- 300MB untuk instalasi .NET Framework
    - +- 100MB untuk instalasi aplikasi R
    - +- 20MB untuk instalasi aplikasi *Rcluzzy*

Software yang digunakan :

- R. 3.4.2 atau
- R. 3.4.3 (copy folder rlapack.dll dari folder C:\Program Files\R\R-3.4.2\bin\i386 ke C:\Program Files\R\R-3.4.2\library\stats\libs\i386 dan rlapack.dll dari folder C:\Program Files\R\R-3.4.2\bin\x64 ke folder C:\Program Files\R\R-3.4.2\library\stats\libs\x64)
- Tidak bisa untuk R.3.4.3 ke atas.
- Library R “mapdata” , “maps”, “ggmap”, “rgdal”, “rgeos”, “factominer” dan “factoextra”.

Algoritme PFCM:



Implementasi coding:

```
pfcm  <-  function(data, ncluster=2, m=2,eta=2,K=1,a=1,b=1,
                   distance = "Euclidean",max.iter=500,
                   error=1e-5) {
  ptm  <-  proc.time()
  iter = 0
  conv <- matrix(0,max.iter,1)
  data  <-  as.matrix(data)

  #generate membership
  uij<-gen_uij(data,ncluster)

  #generate typicality
  tij<-gen_tij(data,ncluster)
  k<-ncluster
  new_uij <- uij
  old_uij <- new_uij+1

  #FCM
  res.fcm<-fcm_try(data,ncluster,w=m)
```

```
  membership<-res.fcm$u
  dista<-res.fcm$d


  #hitung omega dari hasil FCM
  omega<-comp.omega(dista,membership,m,K)


  #Iterasi
  while (max(abs(new_uij-old_uij))>error && iter<max.iter) {
    old_uij <- new_uij

    #Hitung centroid
    v_new  <-  vi.pfcm(data,old_uij,tij,m,eta,a,b)

    #hitung membership baru
    uij  <-  uij.pfcm(old_uij,m,tij,eta,data,v_new,k,distance)

    #hitung jarak
    d<-uij$d

    #hitung typicallity baru
    tij <- tij.pcm(b,d,omega,eta)

    new_uij <- uij$u

    vi <- v_new

    #Hitung fungsi objektif
    for(j in 1:ncluster){
      obj.func <- sum(uij$d[,j]*(a*new_uij[,j]^m + b*tij[,j]^eta))
          + sum(omega[j] * (1-tij[,j])^eta)
    }
    conv[iter,]  <-  obj.func
    iter = iter+1
  }
    #Hasil
  finaldata  <-  determine_cluster(data,new_uij)
  result  <-  list("membership"=new_uij,"dist"=uij$d,"final data"
        = finaldata, "centroid"=vi, "validasi" =
index.fcm(data,new_uij,tij,vi,eta,m,
                    d,e=exp(1),a,b),"jf" = obj.func ,"max.iter" =
        iter,"call"=match.call(), "time" =
proc.time()-ptm)
  return (result)
}

##fungsi hitung centroid

vi.pfcm  <-  function(data,uij,tij,m,eta,a,b) {
             return (t(a*uij^m + b*tij^eta)%*%data/colSums
             (a*uij^m + b*tij^eta))
}

##fungsi hitung membership dan jarak
uij.pfcm  <-  function(uij,m,tij,eta,data,vi,ncluster,distance) {
  u  <-  matrix(0,nrow(data),nrow(vi))
```

```r
  if (distance == "Euclidean"){
    d <- euc(data,vi)
  }else{
    d <- common(uij,m,tij,eta,data,vi,ncluster)
  }
  for (i in 1:nrow(d)) {
    for (j in 1:ncol(d)) {
      if(min(d[i,])==0){
        u[i,] <- rep(0,ncluster)
        u[i,j] <- 1
      }else{
        temp <- (d[i,j]/d[i,])^(1/(m-1))
        u[i,j] <- 1/sum(temp)
      }
    }
  }
  res <- list("d"=d, "u"=u)
  return(res)
}

##menentukan cluster dari data
determine_cluster  <-  function(data,uij,tij){
  clust  <-  matrix(0,nrow(data),1)
  for (i in 1:nrow(data)) {
    clust[i,]  <-  which.max(uij[i,])
  }
  colnames(clust) = "cluster"
  return(clust)
}

##penghitungan index
index.fcm <- function(data,uij,tij,vi,eta,m,d,e,a,b) {
  result<-list()
  result$EXB <- EXB(uij,tij,d,vi,m,eta)
  result$EKwon <- EKwon(data,uij,tij,vi,m,eta,d)
  result$EPE<- ECE(uij,tij,e,a,b)
  return(result)
}

##Xie-Beni
EXB <- function(uij,tij,d,vi,m,eta){
  XB.temp1<-d*(uij^m + tij^eta)
  XB.temp2<-matrix(0,ncol(uij),ncol(uij))
  for(k1 in 1:ncol(uij))
    for(k2 in 1:ncol(uij))
      XB.temp2[k1,k2]<-sum((vi[k1,]-vi[k2,])^2)
  XB.min<-min(XB.temp2[lower.tri(XB.temp2)])
  XB<-sum(XB.temp1)/(XB.min*nrow(uij))
}

#Kwon
EKwon <- function(data,uij,tij,vi,m,eta,d){
  V.bar<-colMeans(data)
  Kwon<-10^10
  try({
    Kwon.temp1<-matrix(0,nrow(uij),ncol(uij))
    for(i in 1:nrow(uij))
```

```
      for(k in 1:ncol(uij))
        Kwon.temp1[i,k]<-d[i,k]*(uij[i,k]^m + tij[i,k]^eta)
      Kwon.temp2<-matrix(0,1,ncol(uij))
      for(k in 1:ncol(uij))
        Kwon.temp2[1,k]<-t(vi[k,]-V.bar)%*%(vi[k,]-V.bar)
      Kwon.temp3<-matrix(0,ncol(uij),ncol(uij))
      for(k1 in 1:ncol(uij))
        for(k2 in 1:ncol(uij))
          Kwon.temp3[k1,k2]<-t(vi[k1,]-vi[k2,])%*%(vi[k1,]-vi[k2,])
      Kwon.min<-min(Kwon.temp3[lower.tri(Kwon.temp3)])
      Kwon<-(sum(Kwon.temp1)+sum(Kwon.temp2)/ncol(uij))/(Kwon.min)
  },silent=T)
}

#Entropy
ECE <- function(uij,tij,e=exp(1),a,b) {
  alpha<-a/(a+b)
  return(alpha*(sum(uij*log(uij,e))/(-nrow(uij)*log(ncol(uij),e))) + (1-
alpha)*(sum(tij*log(tij,e))/(-nrow(tij)*log(ncol(tij),e))))
}

#fungsi menghitung omega
comp.omega<-function(d,u,m,K){
  omega <- numeric(ncol(u))
  for(i in 1:ncol(u))
    omega[i] <- K * ((u[,i]^m) %*% d[,i])/sum(u[,i]^m)
  return(omega)
}

#fungsi menghitung typicality
tij.pcm <- function(b,d,omega,eta){
  t<-matrix(0,nrow(d),ncol(d))
  for (j in 1:ncol(d)){
  for (i in 1:nrow(d)){
      if(min(d[i,]) == 0){
        t[i,] <- rep(0, ncol(d))
        t[i,j] <- 1
      }else{
        t[i,j] <- 1 / (1 + (b*d[i,j]/omega[j])^(1/(eta-1)))
      }
    }
  }
  return(t)
}

#generate uij
gen_uij<-function(data,ncluster){
  u<-matrix(0,nrow(data),ncluster)
  for (i in 1:nrow(data)){
    u<-matrix(runif(nrow(data)*ncluster),nrow(data))
    u<-u/rowSums(u)
  }
  return(u)
}

#generate tij
gen_tij<-function(data,ncluster){
```

```
  t<-matrix(0,nrow(data),ncluster)
  for (i in 1:nrow(data)){
    t<-matrix(runif(nrow(data)*ncluster),nrow(data))
  }
  return(t)
}

#jarak euclid
euc <- function (x,y) {
  dist <- matrix( NA, nrow(x), nrow(y))
  for (i in 1:nrow(x)) {
    for (k in 1:nrow(y)) {
      dist[i,k] <-  t(x[i, ]-y[k, ])%*%(x[i, ]-y[k, ])
    }}
  return (dist)
}

#jarak common mahalanobis
common<-function(uij,m,tij,eta,data,vi,ncluster){
  d.mah <-sum((uij^m) * euc(data,vi))
  cov.m <- matrix(0,ncol(data),ncol(data))
  dist <- matrix( NA, nrow(data), ncluster)
  dif<-array(NA,dim=c(nrow(data),ncol(data),nrow(vi)))

  for (i in 1:nrow(data)){
    for (k in 1:nrow(vi)) {
      dif[i,,k]<-(data[i,]-vi[k,])
      cov.m <- uij[i,k]*dif[i,,k] %*% t(dif[i,,k]) +cov.m
    }}
  cov.m <- cov.m / sum(uij)
  cov.det <- det(cov.m)

if (cov.det > d.mah || cov.det<1/d.mah){
    cov.m <-diag(ncol(data))
  }
  cov.inv <- solve(cov.m)
  cov.ln <- log(det(cov.inv))
  for(i in 1:nrow(data)){
    for(k in 1:ncluster){
      dist[i,k]<-t(dif[i,,k])%*%cov.inv%*%(dif[i,,k])-cov.ln
      if(dist[i,k]<0) dist[i,k]=0
    }
  }
  return(dist)
}

#fungsi FCM
fcm_try<-function( data, cluster, w=2, u=NA, a=NA, distance="Euclidean",
seed=NA,loop.max.fcm=100 ,e.threshold.fcm=1e-5, output="fcm",midt=NA,
madt=NA){

  waktu <- proc.time()
  jf = 0
  loop = 1
  data<-as.matrix(data)
  n.data <- nrow(data)
  m <- ncol(data)
```

```r
  #STEP 1 : INITIALIZING FUZZY PARTITION MATRIX (U)
  for(i in 1:length(u)){
    if(is.na(u[i])){
      u <- matrix( runif(n.data*cluster) , n.data)
      u <- u/rowSums(u)
    } else {
      u <-as.matrix(u)
    }
  }


  # u <- matrix( runif(n.data*cluster) , n.data)
  # u <- u/rowSums(u)
  if(is.na(a)){
    a = matrix( NA, cluster, m )
  }
  else{
    a=as.matrix(a)
  }
  dist <- matrix( NA, n.data, cluster) #jarak data dengan pusat cluster
  dif <- array(NA, dim=c(n.data,m,cluster)) #save different X and A for
mahalanobis
  u1 <-u

  repeat{
    #STEP 2 : COMPUTE cluster CENTER
    u.w <- u^w

    if(is.na(a)==FALSE&&loop==1){
      a<-as.matrix(a)
   }else{
      for (k in 1:cluster) {
        for (j in 1:m) {
          a.num <- sum(u.w[ ,k]*data[ ,j])
          a.denum <- sum(u.w[ ,k])
          a[k,j] <- a.num / a.denum
        }}
    }


    #STEP 3 : COMPUTE ALL DISTANCE BETWEEN DATA AND cluster CENTER

    euc <- function (x,y) {
      dist <- matrix( NA, n.data, cluster)
      for (i in 1:n.data) {
        for (k in 1:cluster) {
          dist[i,k] <-  t(x[i, ]-y[k, ])%*%(x[i, ]-y[k, ])
        }}
      return (dist)
    }
    mahala<-function (x,y) {
      covmat<- cov(x)
      s.covmat<-solve(covmat)
      dist <- matrix( NA, n.data, cluster)
      for (i in 1:n.data) {
        for (k in 1:cluster) {
```

```
          dist[i,k] <-  t(x[i, ]-y[k, ])%*%s.covmat%*%(x[i, ]-y[k, ])
        }}
      return (dist)
    }


    # m.sama <- "Covarians != I sehingga hasil Mahalanobis berbeda
      dengan Euclidean"
    # #value yang menyimpan covarians. sama=T bila cov=I ; sama=""      bila
Cov!=I
    #
    if (distance == "Euclidean") {
      dist <- euc(data,a)

    } else {
      dist <- mahala(data,a)
    }

    #STEP 4 : COMPUTE NEW FUZZY PARTITION MATRIX (u)
    dist.iwi <- dist ^ (-1/(w-1))
    # dist.iwi <- ifelse(dist.iwi==Inf, 0, dist.iwi)
    dist.iwi.cluster <- rowSums(dist.iwi)
    u <- dist.iwi / dist.iwi.cluster
    for (i in 1:n.data)
      for (k in 1:cluster)
        if (dist[i,k]==0) {
          u[i,] <- 0
          u[i,k] <- 1
        }
    u[is.nan(u)] <- 1

    #STEP 5 : STOPPING CRITERIA : e.threshold.fcm and max.loop
    #Menghitung selisih fungsi objektif dengan fungsi objektif
      sebelumnya
    # jf.now <- sum(u.w*dist)
    # e = abs(jf.now - jf)
    #jf <- jf.now
    func <- (u^w)*dist
    jf <- sum(func)
    e <- max(u-u1)
    u1 <- u
    loop=loop+1

    #Mengecek kondisi berhenti
    if(e<e.threshold.fcm || loop>loop.max.fcm) break

  }
  #hasil clustering
  kelompok <- matrix (NA, nrow(data), 1)
  kelompok[,1] <- as.numeric(apply(u, 1, which.max))
  result<-list("u" = u1 , "d" = dist,"jf" =jf, "time" =
      proc.time()-waktu , "loop" = loop ,
                 "cluster"= kelompok , "v" = a)
  return(result)
 }
```