Propuesta de implementación del algoritmo CAIM

Ángel García Báez

2024-13-14

Contents

1 Arquitectura de la función para calcular el CAIM

```
#### Función para identificar cual es la variable con las clases y separar ####
# los atributos de la variable de clases #
# datos = Matriz de datos
# VarClas = Nombre exacto de la variable que contiene las clases
detector = function(datos,VarClas){
  # Guardar los indices originales y los nombres
  indices = 1:ncol(datos)
  nombres = names(datos)
  # Identificar del conjunto el indice de la variable
  idClass = which(names(datos)==VarClas,nombres)
  # Regresar el conjunto de indices de los atributos F_i
  atributos = setdiff(indices,idClass)
  atributos1 = atributos
  # Identificar variables que sean valores numericos
  for(i in 1:length(atributos) ){
    # Si la variable es NO numerica, sacala del conjunto
    if((class(datos[,atributos[i]])== "numeric" | class(datos[,atributos[i]])== "integer")){
      # Todo bien
    }else{
      # Quita esa variable de los atributos
      atributos1 = setdiff(atributos1,atributos[i])
    }
  }
  # junto con la posición donde esta la clase
  resultado = c(Atributos=atributos1,Clase = idClass)
  return(resultado)
}


#### Función CAIMp para calcular el CAIM a partir de la matriz Cuanta ####
# Inter = Valores discretizados por los intervalos propuestos
# Clase = La cantidad de clases que hay en variable de clase
CAIMp = function(Inter,clase){
  #Sacar la matriz Quanta base con sus marginales
  quanta = addmargins(table(Clase = clase,Intervalo = Inter))
  # Dimensiones de la cuanta
  DQ = dim(quanta)
  # Sacar el critero de CAIM con un for
  SumaC = 0
  for(i in 1:(DQ[2]-1)){ # La cantidad de intervalos -1 para evitar el marginal
    # Extraer el maximo de la columa i-esima
    SumaC = SumaC + max(quanta[-DQ[1],i])^2 /quanta[DQ[1],i]
  }
  CAIMr = SumaC/(DQ[2]-1) #Le quito la columna del marginal
  # Devolver el valor del CAIM para este esquema
  return(CAIMr)
}

#### FUNCIÓN CAIM COMPLETA PARA DISCRETIZAR ####
CAIM = function(datos,VarClas){
  # Primero aplica el detector para identificar a la variable clase
  # de los atributos y quitar las variables NO numéricas
```

```
detecta = detector(datos,VarClas)
# Separa la clase
ClaseS = detecta[length(detecta)]
# Separa los atributos considerados numéricos
atributos = setdiff(detecta,ClaseS)
# Ya tengo separativos los atributos y la variable clase

#### Inicializar el algoritmo para que itere sobre cada variable ####
# Genero los objetos con los resultados que van a ser guardados para la salida
resultados = list()
# Aplicar esto para el atributo i-esimo
for(i in atributos){ # Moverse sobre todos los indices de atributos validos
  #### PASO 1 ####
  # Paso 1.1 Encuentra el valor máximo (dn) y el valor mínimo(d0) del atributo
  d0 = min(datos[,i],na.rm = T )
  dn = max(datos[,i],na.rm = T)
  # Paso 1.2 Forma un conjunto de todos los valores distintos de F_i en orden
  # ascendente, inicializa todos los posibles limites de los intervalos B con
  # minimo, maximo y todos los puntos medios de todas las parejas adyacentes
  unicos = sort(unique(datos[,i]))
  B = c()
  # Sacar las parejas
  for(pi in 2:length(unicos)){
    # Sacar las mitades
    B[pi-1] = (unicos[pi]+unicos[pi-1])/2
  }
  ### Paso 1.3 Define el esquema de discretización inicial como D: {[d0,dn]}
  # y define el GlobalCaim = 0#
  D = c(d0,dn) # Esquema de discretización base
  GlobalCaim = 0 #Global Caim Base

  #### PASO 2 ####
  # Paso 2.1 Inicializa k = 1
  k = 1
  # Inicializo otras variables para hacer funcionar el while
  maxCaim = 0
  while((maxCaim > GlobalCaim) | (k<length(unique(datos[,ClaseS])))){
    # 2.2 Tentativamente agrega un intervalo interno de B,
    # el cual no se encuentre en D y calculale el correspondiente CAIM
    # Indices de CAIM para elegir un esquema
    ICAIM = c()
    # Ciclo para probar con todas las posibles mitades como candidatos
    # a ser parte del esquema
    for(posi in 1:length(B)){
      # Crea el esquema i-esimo
      Di = sort(c(D,B[posi]))
      # Discretiza con el esquema i-esimo
      Inter = cut(datos[,i],
                  breaks = Di,
                  include.lowest = TRUE,
                  right = TRUE)
      # Calcula el CAIM
      ICAIM[posi] = CAIMp(Inter,datos[,ClaseS])
```

```
      }
      # 2.3 Una vez probadas todas las adiciones provisionales,
      # aceptar la que tenga el valor más alto de de CAIM
      # Extraigo el indice máximo y de caim y su posición
      maxCaim = max(ICAIM) # Maximo Caim en la iteración i-esima
      maxCaimPos = which.max(ICAIM) # Posición del maximo Caim
      #2.4 Si (CAIM > GlobalCAIM O k<S) Actualiza D con la propuesta
      #de intervalo aceptada en el paso 2.3 y configura GlobalCAIM = CAIM,
      #si no termina
      # Actualizar el esquema
      D = sort(c(D,B[maxCaimPos])) # Actualización del esquema
      #Quito el valor seleccionado de la bolsa de mitades B
      B = B[-maxCaimPos]
      # Actualiza el valor del global
      GlobalCaim = maxCaim
      k = k+1
    }
    # Reporta y guarda el resultado de la iteración
    # Crear el resultado como un vector
    vr = c(D,GlobalCaim,k,i)
    resultados = append(resultados,list(vr))
  }
  # Ya estando fuera de la función, puedo operar los resultados
  # para reportar las salidas necesarias
  ### Colapsar en una matriz la lista de resultados
  resultados = do.call(rbind, resultados)
  # Separar los esquemas
  esquemas = resultados[,1:(ncol(resultados)-3)]
  # Caim por variable
  Caim =cbind.data.frame(Variables = names(datos[,atributos]),
                         Caim = resultados[,(ncol(resultados)-2)])
  # Usar los intervalos para discretizar por variable
  datosR = datos
  # Discretizar solo las variables que se les obtuvo el esquema
  for(i in 1:length(atributos)){
    # Aplicar la discretización para el atributo i-esimo dentro
    # los permitidos
    datosR[,atributos[i]] = cut(datosR[,atributos[i]],
                                breaks = esquemas[i,],
                                include.lowest = TRUE,
                                right = TRUE)
  }
  #Teniendo todo, se reporta en un solo objeto todo lo depurado
  resultadoF = list(Discretizados = datosR,
                    CAIM = Caim,
                    Esquemas = esquemas)
  return(resultadoF)
}
```

2 Discretización para el dataset de Iris

```
# Cargar el codigo fuente del CAIM
library(flextable)
source("CAIM MAIN.r")
#### Base de datos 1 IRIS ####
datos1 = iris
R1 = CAIM(datos1,"Species")
Resumen1 = cbind.data.frame(R1$CAIM,R1$Esquemas)
autofit(theme_box(flextable(Resumen1)))
```

Variables	Caim	1	2	3	4
Sepal.Length	26.63627	4.3	5.55	6.25	7.9
Sepal.Width	17.38251	2.0	2.95	3.05	4.4
Petal.Length	45.55892	1.0	2.45	4.75	6.9
Petal.Width	46.16157	0.1	0.80	1.75	2.5

3 Discretización para el dataset de pinguinos Palmer

```
#### Base de datos 2 Pinguinos Palmer ####
library(palmerpenguins)
datos2 = as.data.frame(penguins)
R2 = CAIM(datos2,"Species")
Resumen2 = cbind.data.frame(R2$CAIM,R2$Esquemas)
autofit(theme_box(flextable(Resumen2)))
```

Variables	Caim	1	2	3	4
bill_length_mm	19.35567	32.1	45.85	46.75	59.6
bill_depth_mm	20.01479	13.1	14.25	17.75	21.5
flipper_length_mm	21.65400	172.0	186.50	219.50	231.0
body_mass_g	17.52661	2,700.0	3,537.50	4,562.50	6,300.0

4 Discretización para el dataset de mujeres PIMA

```
#### Base de datos 3 PIMA ####
library(pdp)
library(DataCombine)
datos3 = pima
R3 = CAIM(datos3,"diabetes")
Resumen3 = cbind.data.frame(R3$CAIM,R3$Esquemas)
autofit(theme_box(flextable(Resumen3)))
```

Variables	Caim	1	2	3
pregnant	178.18365	0.000	6.5000	17.00
glucose	215.29733	44.000	143.5000	199.00
pressure	159.96169	24.000	69.0000	122.00
triceps	124.33870	7.000	23.5000	99.00
insulin	98.37759	14.000	143.0000	846.00
mass	173.28023	18.200	40.8500	67.10
pedigree	168.73337	0.078	1.1075	2.42
age	173.34613	21.000	30.5000	81.00

5 Discretización para el dataset de ataques al corazón

```
#### Base de datos 4 Heart Diases ####
library(kmed)
datos4 = heart
datos4$class =  ifelse(datos4$class==0,0,1)
R4 = CAIM(datos4,"class")
Resumen4 = cbind.data.frame(R4$CAIM,R4$Esquemas)
autofit(theme_box(flextable(Resumen4)))
```

Variables	Caim	1	2	3
age	61.77181	29	54.50	77.0
trestbps	50.40263	94	143.00	200.0
chol	50.96984	126	273.50	564.0
thalach	75.66511	71	147.50	202.0
oldpeak	72.23809	0	0.85	6.2
ca	82.22547	0	0.50	3.0

6 Discretización para el dataset de vinos

```
#### Base de datos 5 Vinos ####
library(HDclassif)
data(wine)
nombres <- c("Clase","Alcohol", "Malic acid", "Ash", "Alcalinity of ash",
             "Magnesium", "Total phenols", "Flavanoids", "Nonflavanoid phenols",
             "Proanthocyanins", "Color intensity", "Hue",
             "OD280/OD315 of diluted wines", "Proline")
datos5 = wine
names(datos5) = nombres
datos5$Clase = as.factor(datos5$Clase)
R5 = CAIM(datos5,"Clase")
Resumen5 = cbind.data.frame(R5$CAIM,R5$Esquemas)
autofit(theme_box(flextable(Resumen5)))
```

Variables	**Caim**	**1**	**2**	**3**	**4**
Alcohol	31.97545	11.03	12.780	12.975	14.83
Malic acid	26.36663	0.74	1.550	2.455	5.80
Ash	16.45138	1.36	2.245	2.405	3.23
Alcalinity of ash	19.99623	10.60	17.900	22.900	30.00
Magnesium	21.66290	70.00	88.500	133.000	162.00
Total phenols	27.49416	0.98	1.840	2.335	3.88
Flavanoids	41.30902	0.34	1.235	2.310	5.08
Nonflavanoid phenols	17.31576	0.13	0.345	0.395	0.66
Proanthocyanins	21.95374	0.41	1.305	1.655	3.58
Color intensity	36.33333	1.28	3.460	7.550	13.00
Hue	24.84595	0.48	0.785	1.005	1.71
OD280/OD315 of diluted wines	30.10134	1.27	2.115	3.305	4.00
Proline	34.75020	278.00	512.500	755.000	1,680.00