Aprendizaje Automático
Algoritmo CAIM

El algoritmo CAIM desarrollado por Lukasz A. Kurgan y Krzysztof J. Cios, sirve para discretizar variables continuas para posteriormente ser procesadas por algoritmos de aprendizaje automático. La finalidad es encontrar la mejor combinación de rangos en los cuales clasificar los datos de la variable a discretizar, sin perder información relevante para el entrenamiento y que los datos sean lo suficientemente representativo del problema a resolver. Este algoritmo es de discretización supervisada, ya que se necesita saber las clases a las que pertenece cada ejemplo y busca maximizar la interdependencia entre la clase y el atributo y generar el menor numero posible de intervalos discretos, ademas de no necesitar que el usuario defina un número especifico de intervalos.

El algoritmo CAIM requiere un conjunto de M ejemplos, donde cada ejemplo pertenece a solo una clase S, ademas cada uno de los valores continuos que tiene la variable en el conjunto de ejemplos se denomina como F. Existe un esquema de discretización D en F con n numero de intervalos:

$$D : \{[d_0, d_1], (d_1, d_2], \ldots, (d_{n-1}, d_n]\}$$

D0 es el valor mínimo y dn es el valor máximo del atributo en F y los valores están acomodados de manera ascendente. Estos valores de intervalos son posteriormente utilizados para crear la matriz quanta:

Class	Interval					Class Total
	$[d_0, d_1]$	…	$(d_{r-1}, d_r]$	…	$(d_{n-1}, d_n]$	
C_1	q_{11}	…	q_{1r}	…	q_{1n}	M_{1+}
:	:	…	:	…	:	:
C_i	q_{i1}	…	q_{ir}	…	q_{in}	M_{i+}
:	:	…	:	…	:	:
C_S	q_{S1}	…	q_{Sr}	…	q_{Sn}	M_{S+}
Interval Total	M_{+1}	…	M_{+r}	…	M_{+n}	M

Para el calculo del valor se utiliza la siguiente formula:

$$CAIM(C, D \mid F) = \frac{\sum_{r=1}^{n} \frac{\max_r^2}{M_{+r}}}{n}$$

Donde n es el número de intervalos, r itera a través de todos los intervalos, maxr es el valor máximo entre todos los valores qir (valor máximo de la columna r) M+r es el total de valores continuos del atributo F que están dentro del intervalo (dr-1, dr].

Pseudocodigo:

Paso 1:
1.1 Encontrar el valor máximo dn y mínimo d0 de Fi
1.2 Formar un conjunto de todos los valores distintos de Fi, colocados en orden ascendente, crear una variable B que contendrá la siguiente información: d0, todos los valores intermedios sin repetir, dn.
1.3 Crear una variable D que contendrá el esquema de discretización e inicializara con D:{[d0, dn]} y GlobalCAIM=0
Paso2:
2.1 Inicializar una variable K=1
2.2 Tentativamente agregue algún limite de B que no se encuentre en D y calcular su correspondiente CAIM
2.3 Después de todas las posibles adiciones han sido probadas aceptar la de mayor valor de CAIM
2.4 si (CAIM>GlobalCAIM o K<S) entonces actualizar D con el aceptado en el paso 2.3 y establecer GlobalCAIM=CAIM, si no terminar
2.5 establecer K=K+1 y repetir a partir de 2.2

Al probar con la base de datos de Iris encuentra que los mejores intervalos para discretizar son los siguientes:

SepalLengthCm
El mejor intervalo de discretización es:
[[4.3, 4.35], [4.35, 5.8], [5.8, 7.9]]

SepalWidthCm
El mejor intervalo de discretización es:
[[2.0, 2.1], [2.1, 4.0], [4.0, 4.4]]

PetalLengthCm
El mejor intervalo de discretización es:
[[1.0, 1.05], [1.05, 1.9], [1.9, 6.9]]

PetalWidthCm
El mejor intervalo de discretización es:
[[0.1, 0.15000000000000002], [0.15000000000000002, 0.4], [0.4, 2.5]]

Y al utilizar la base de datos discretizada para clasificación con el algoritmo Naive Bayes en WEKA, se obtiene una precisión de 78.66%, el cual al compararlo con el algoritmo interno de WEKA para discretizar resulta ser de menor precisión, ya que con el de WEKA se obtiene una precisión de 96%, por lo cual habría que mejorar el algoritmo para obtener mejores resultados al obtenido.

Código implementado en python:

```
import numpy as np
import pandas as pd
import copy
import itertools
import time
inicio=time.time()

def duplicado(nums):
        dup = [x for i, x in enumerate(nums) if x in nums[:i]]
        return dup

data=pd.read_csv('iris.csv')
#print(data)

names=data.columns.values
print(names)
print(names[0])

listclas=data[names[-1]].tolist()
lclas=list(set(listclas))
#print(listclas)
print(lclas)

for numero in range(len(data)):
        for nombre in range(len(lclas)):
                if data[names[-1]][numero]==lclas[nombre]:
                        data[names[-1]][numero]=nombre
                        break

#print(data)

for atributo in range(len(names)-1):
        print(names[atributo])

        dataordenada=data.sort_values(names[atributo])
        #print(dataordenada)

        da=list(dataordenada[names[atributo]])
        #print(da)
        dv=list(set(da))
        dv.sort()
```

```
#print("Lista ordenada de valores:")
#print(dv)
clases=list(dataordenada[names[-1]])
clas=list(set(clases)) #crea lista de valores de clase unicos
#print("Clases:")
#print(clas)

#obtiene limites
d0=dv[0]
dn=dv[-1]

b=[]
for i in range(len(dv)-1):
        b.append(dv[i])
        b.append((dv[i]+dv[i+1])/2)
b.append(dv[-1])
#print("B:")
#print(len(b))

b.pop(0)
b.pop()

daux=[d0,dn]
d=[[d0,dn]]
globalcaim=0
#print("D:")
#print(d)

x=[]
x.append(b)
k=1

mejorcaim=[]
globalcaim=0
#for z in range(1):
for z in range(len(b)):
        if k>=len(clas):
                break
        products=x[0]
        for i in range(len(x)-1):
                products = list(itertools.product(products,x[i+1]))
        for i in range(len(products)):
                string=str(products[i])
                string = string.replace("(","").replace(")","").replace(",","").replace(" ",", ")
                tupla=tuple(map(float, string.split(', ')))
                products[i]=tupla
        #print(products)
        for tupla in products:

                if len(duplicado(tupla))==0 and list(tupla)==sorted(tupla):
                        daux=[]
                        daux.append(d0)
```

```
                                for element in tupla:
                                        daux.append(element)
                                daux.append(dn)
                                d=[]
                                for indice in range(len(daux)-1):
                                        d.append([daux[indice],daux[indice+1]])
                                #print("D:")
                                #print(d)

                                quanta=np.zeros((len(clas),len(d)+1))
                                for i in range(len(clas)):
                                        quanta[i][0]=clas[i]

                                for indice in range(len(dv)):
                                        cl=data.iloc[indice][names[-1]]
                                        v=data.iloc[indice][names[atributo]]
                                        for classe in range(len(clas)):
                                                if cl==quanta[classe][0]:
                                                        for intervalo in range(len(d)):
                                                                lims=d[intervalo]
                                                                liminf=lims[0]
                                                                limsup=lims[1]
                                                                if intervalo==0:
                                                                        if liminf<=v and v<=limsup:
                                                                                quanta[classe][intervalo+1]+=1
                                                                                break
                                                                else:
                                                                        if liminf<v and v<=limsup:
                                                                                quanta[classe][intervalo+1]+=1
                                                                                break
                                quanta=np.delete(quanta,0,axis=1)
                                #print("Quanta:")
                                #print(quanta)

                                #sr=np.sum(quanta, axis=1) #suma renglon
                                sc=np.sum(quanta, axis=0) #suma columna
                                maximos=np.argmax(quanta,axis=0) #indice donde se encuentra el valor mas
grande
                                suma=0
                                for i in range(len(d)-1):
                                        if sc[i]!=0:
                                                suma=suma+((quanta[i][maximos[i]]**2)/sc[i])
                                caim=suma/len(d)
                                #print("Caim:")
                                #print(caim)

                                if caim>globalcaim:
                                        globalcaim=copy.deepcopy(caim)
                                        mejorcaim=copy.deepcopy(d)
                        popaux=copy.deepcopy(x[z])
                        #elimina el ultimo elemento de cada tupla
                        for w in range(z+1):
                                x[w].pop()
```

```
                #quita el primer elemento de la ultima tupla y pega esa nueva tupla al final de la lista
                popaux.pop(0)
                x.append(popaux)
                k+=1

        print("El mejor intervalo de discretización es:")
        print(mejorcaim)
        for ejemplo in range(len(data)):
                valor=data[names[atributo]][ejemplo]
                for rango in range(len(mejorcaim)):
                        limites=mejorcaim[rango]
                        inf=limites[0]
                        sup=limites[1]

                        if rango==0:
                                if inf<=valor and valor<=sup:
                                        data[names[atributo]][ejemplo]='clase'+str(rango)
                                        break

                        else:
                                if inf<valor and valor<=sup:
                                        data[names[atributo]][ejemplo]='clase'+str(rango)
                                        break

for numero in range(len(data)):
        for nombre in range(len(lclas)):
                if data[names[-1]][numero]==nombre:
                        data[names[-1]][numero]=lclas[nombre]
                        break


print("La data discretizada es:")
print(data)
print("El tiempo de ejecución es:")
fin=time.time()
print(fin-inicio)
data.to_csv('data.csv')
```

Referencias:
Kurgan, Lukasz & Cios, Krzysztof. (2004). CAIM discretization algorithm. Knowledge and Data Engineering, IEEE Transactions on. 16. 145- 153. 10.1109/TKDE.2004.1269594.