Universidad Nacional San Agustin de Arequipa

Facultad de Ingenierias de Produccion y Servicios

Escuela Profesional de Ingenieria de Sistemas

Inteligencia Artificial

Alumno:

Fuentes Paredes Nelson Alejandro

Mayo 2020

1 Importación de Librerias

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

2 Metodos Usados

2.1 Limpiar datos de una columna

Elimina espacios en blanco en los valores de una columna.

[2]:
```
def clear_column(column):
    new_column = []
    for value in column:
        new_column.append(value.replace(' ', ''))
    return new_column
```

2.2 Limpia los datos recibidos

Limpia los datos de un DataFrame.

[3]:
```
def clear_data(data):
    data.columns = clear_column(data.columns)
    data[['']] = clear_column(data[''])
    data = data.set_index('')
    for i in range(len(data)):
        data.iat[i, i] = np.nan
    return data
```

2.3 Obtener Indices del menor valor

Encuentra el menor valor y devuelve un arreglo con sus indices respectivos.

[4]:
```
def getMinIndexes(data):
    minimus = data.idxmin(skipna=True).dropna()
    for i, row in enumerate(minimus):
        minimus[i] = [data[minimus.index[i]][row], row, minimus.index[i]]
    return min(minimus)
```

2.4 Funciones para operar con datos que pueden recivbir datos Nan

Dependiendo de lo que se necesite devuelven el menor, mayor y promedio entre dos datos.

[5]:
```
def minna(a, b):
    if pd.notna(a):
        return min(a, b)
    return b
```

```
def maxna(a, b):
    if pd.notna(a):
        return max(a, b)
    return b

def meanna(a, b):
    if pd.notna(a):
        return (a + b) / 2.0
    return b
```

2.5 Añadir Fila y Columna a un DataFrame

Añade una fila y una fila aplicando en su contenido el resultado de una funcion que se le mande.

[6]:
```
def add_row_and_column_to_data(data, column1, column2, function, label):
    new_column = (data.apply(lambda row: function(row[column1], row[column2]),␣
 ↪axis=1))
    new_row = (data.apply(lambda row: function(row[column1], row[column2]),␣
 ↪axis=0))
    new_row.name = label
    data = data.append(new_row, ignore_index=False)
    del (data[column1])
    del (data[column2])
    data[label] = new_column
    return  data.drop([column1, column2])
```

2.6 Algoritmo para Clutering Jerarquico Aglomerativo

Recibe un conjunto de datos y una función con los cuales aplica el algoritmo aprendido de forma recursiva, retorna todos los niveles que encontro y la data final.

[7]:
```
def cluster(data, function):
    levels = []
    if len(data.columns) > 1:
        minimun = getMinIndexes(data)
        label_minimun = minimun[1] + minimun[2]
        levels.append(minimun)
        data = add_row_and_column_to_data(data, minimun[1], minimun[2],␣
 ↪function, label_minimun)
        new_level, data = cluster(data, function)
        levels = levels + new_level
    return levels, data
```

2.7 Metodos de Clustering Jerarquico

Son los metodos aprendidos que utilizan el metodo cluster enviandole los datos que reciben y una función especifica dependiendo del metodo.

[8]:
```
def single_linkage(data):
    return cluster(data, minna)

def complete_linkage(data):
    return cluster(data, maxna)

def mean_linkage(data):
    return cluster(data, meanna)
```

2.8 Graficacion de Dendograma

Debido a que para crear Dendogramas en Python no se necesita la implementación de algoritmos ya que este los ejecuta segun eleccion del usuario, se decicidio elaborar un metodo propio para graficar el endograma.

[9]:
```
def draw_dendogram(levels, axes, title):
    bottom_level = {}
    for level in levels:
        bottom_level[level[1]] = 0
        bottom_level[level[2]] = 0
    label_levels = {}
    for level in levels:
        a = level[1]
        b = level[2]
        ai = 0
        bi = 0
        if not a in label_levels:
            ai = len(label_levels) + 1
            label_levels[a] = ai
        else:
            ai = label_levels[a]
        if not b in label_levels:
            bi = len(label_levels) + 1
            label_levels[b] = bi
        else:
            bi = label_levels[b]
        label_levels[a + b] = (ai + bi) / 2
    label_levels_1 = []
    for level in label_levels.keys():
        label_levels_1.append([label_levels[level], level])
    label_levels_1 = sorted(label_levels_1)
    for level in label_levels_1:
        axes.plot([level[1]], [0])
```

```
    fig = plt.Figure()
    for level in levels:
        axes.set_title(title)
        axes.plot([level[1]], [0], 'b')
        axes.plot([level[1] + level[2]], [0], 'b')
        axes.plot([level[2]], [0], 'b')
        axes.plot([level[1], level[2]], [level[0]] * 2, 'b')
        axes.plot([level[1], level[1]], [bottom_level[level[1]], level[0]], 'b')
        axes.plot([level[2], level[2]], [bottom_level[level[2]], level[0]], 'b')
        axes.plot([level[1] + level[2]] * 2, [level[0]] * 2, 'b')
        axes.plot([level[1], level[1] + level[2]], [level[0]] * 2, 'b')
        axes.set(ylim=(0, 3))
        bottom_level[level[1] + level[2]] = level[0]
```

3 Ejecucion

3.1 Metodo Main

Recibe un nombre de archivo, esos datos los limpia y grafica los resultados de los tres metodos de clustering

[10]:
```
def main(file):
    data = pd.read_csv(file, sep='\t')
    data = clear_data(data)
    print(data)
    fig, ax = plt.subplots(3, 1, constrained_layout=True, figsize=(8, 10))
    levels, data1 = single_linkage(data)
    draw_dendogram(levels, ax[0], 'Single Linkage')

    levels, data2 = complete_linkage(data)
    draw_dendogram(levels, ax[1], 'Complete Linkage')

    levels, data3 = mean_linkage(data)
    draw_dendogram(levels, ax[2], 'Mean Linkage')
    plt.show()
```

3.2 Ejecucion

[11]:
```
main('cluster.txt')
```

```
      A     B     C     D     E     F   G

A   NaN   NaN   NaN   NaN   NaN   NaN NaN
B  2.15   NaN   NaN   NaN   NaN   NaN NaN
C  0.70  1.53   NaN   NaN   NaN   NaN NaN
D  1.07  1.14  0.43   NaN   NaN   NaN NaN
E  0.85  1.38  0.21  0.29   NaN   NaN NaN
```

```
F  1.16  1.01  0.55  0.22  0.41   NaN NaN
G  1.56  2.83  1.86  2.04  2.02  2.05 NaN
```