Computação Natural - Tarefa Computacional 1

Alexandre Rosseto Lemos

Matrícula: 2021231580

email: alexandrerossetolemos@gmail.com

PPGI (Mestrado)

Inicialização

Bibliotecas

In [1]:
```
import pandas as pd
import numpy as np
from scipy.optimize import differential_evolution
from numpy import argsort
import random
from numpy.random import randn
from numpy.random import rand

# Datasets
from sklearn.datasets import load_breast_cancer, load_wine, load_iris

pd.options.display.max_columns = 50
```

Funções utilizadas na leitura dos dados

data_to_df

In [2]:
```
def data_to_df(data):
    '''
    Info:
        Essa função lê os dados e transforma eles em um dataframe.
    ----------
    Input:
        data: Dados obtidos através da biblioteca sklearn.datasets.
    ----------
    Output:
        df: Dataframe com os dados
    '''

    # Obtendo as features e target
    feat = data.data
    target = data.target
    target = target.reshape(len(target), 1)

    # Concatenando as informacoes
    info = np.hstack((feat, target))

    # Obtendo os nomes das features e adiciona o nome da coluna target
    feat_name = list(data.feature_names)
    feat_name.append('target')

    # Criando o dataframe
    df = pd.DataFrame(data = info, columns = feat_name)

    display(df.head())
    print('Shape:',df.shape)

    return df
```

Carregando os dados

In [3]:
```
# Obtendo os dados de breast cancer
data_wdbc = load_breast_cancer()
df_wdbd = data_to_df(data_wdbc)
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error	perimeter error	area error
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	1.0950	0.9053	8.589	153.40
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	3.398	74.08
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	0.7456	0.7869	4.585	94.03
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	0.4956	1.1560	3.445	27.23
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	0.7572	0.7813	5.438	94.44

Shape: (569, 31)

In [4]:
```
# Obtendo os dados de iris
data_iris = load_iris()
df_iris = data_to_df(data_iris)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

Shape: (150, 5)

In [5]:
```
# Obtendo os dados de wine
data_wine = load_wine()
df_wine = data_to_df(data_wine)
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	h
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.

Shape: (178, 14)

K-means

Os valores de K para cada dataset será igual à quantidade de classes nas colunas target:

- Wine: k = 3
- Wdbd: k = 2
- Iris: k = 3

In [6]:
```
# Definindo o numero de clusters a serem formados
k_wdbd = 2
k_wine = 3
k_iris = 3
```

In [7]:
```
class KMeans:

    def __init__(self, n_clusters):
        self.n_clusters = n_clusters
        self.max_lim = None
        self.min_lim = None
        self.X = None
        self.centroids = []
        self.clusters = None
        self.sse_baseline = None

    def inicializacao(self):
        '''
        Info:
            Funcao que gera os centroides iniciais aleatoriamente.
        ----------
        Input:
            None
        ----------
        Output:
            None
        '''
        # Obtem os limites maximo e minimo das features
        self.max_lim = self.X.max().values
        self.min_lim = self.X.min().values

        # Inicializa k centroides aleatoriamente entre os limites definidos
        for c in range(self.n_clusters):
            centroid = []
            for i in range(len(self.max_lim)):
                centroid.append(np.random.uniform(self.min_lim[i],self.max_lim[i]))

            # Gerando a lista com todos os  centroides
            self.centroids.append(centroid)

    def clusterizacao(self):
        '''
        Info:
            Funcao que calcula as  amostras pertencentes a cada cluster.
        ----------
        Input:
            None
        ----------
        Output:
            None
        '''
        # Atribuindo o cluster a cada amostra baseado na distancia entre ela e todos os clusters
        # (escolhendo o cluster com a menor distancia)
        clusters = []
        for i in self.X.index:
            sample = self.X.iloc[i].values
            dist = []

            # Calculando a distancia da amostra aos centroides
            for c in range(self.n_clusters):
                dist.append(np.linalg.norm(sample - self.centroids[c]))

            #  Encontrando a menor distancia e o indice do cluster
            min_value = min(dist)
            cluster = (dist.index(min_value) + 1)

            # Criando a lista dos cluster de cada  amostra
            clusters.append(cluster)

        self.clusters = pd.DataFrame(data = clusters, columns = ['clusters'])

    def convergir(self):
        '''
        Info:
            Funcao que recalcula os clusters e os centroides ate que nao se altere o resultado obtido.
        ----------
        Input:
            None
        ----------
        Output:
            None
        '''
        # Recalculando os novos centroides enquanto nao houver convergencia
        convergence = False
        while convergence == False:

            # Juntando os clusters aos dados
            data = pd.concat([self.X, self.clusters], axis = 1)

            new_centroids = []
            for k in range(self.n_clusters):
                cluster = data[data['clusters'] == (k + 1)]

                # Removendo a coluna de cluster
                cluster = cluster.drop(columns = 'clusters')

                # Calculando o centroide para o cluster
                centroid = list(cluster.mean().values)

                # Verifica se o centroide foi calculado (!= null)
                if np.isnan(centroid[0]) == True:

                    # Caso seja nulo, nao altera o valor do centroide para o cluster
                    centroid = self.centroids[k]

                new_centroids.append(centroid)

            # Verifica se os novos centroides sao iguais aos antigos
            convergence = self.centroids == new_centroids

            # Recalculando os clusters
            if convergence == False:

                # Alterando os centroides dos clusters
                self.centroids = new_centroids

                # Calculando os novos clusters utilizando os novos centroides
                self.clusterizacao()

    def calculate_sse_baseline(self):
        '''
        Info:
            Funcao que calcula o SSE para os resultados obtidos apenas com K-Means.
        ----------
        Input:
            None
        ----------
        Output:
            None
        '''
        n_features = self.X.shape[1] # Numero de features
        n_coords = len(self.centroids) # Numero total de coordenadas de todos os centroides

        # Juntando informacoes de dados e clusters
        data = pd.concat([self.X, self.clusters], axis = 1)

        sse = 0
        # Variando os clusters
        for k in data['clusters'].value_counts().index.to_list():

            # Selecionando apenas as amostras do cluster k e removendo a coluna de cluster
            data_cluster = data[data['clusters'] == k].drop(columns = 'clusters')

            # Obtendo as coordenadas do centroide do cluster k
            ind = int((k - 1) * (n_coords/n_clusters))
            centr_coord = self.centroid[ind:ind + n_features]

            # Calculando a distancia ao quadrado das amostras ao centroide do cluster
            cluster_err = 0
            for i in data_cluster.index:
                sample = data_cluster.loc[i].values

                # Calculando a distancia
                dist = (np.linalg.norm(sample - centr_coord) ** 2)

                cluster_err = cluster_err + dist

            # Calculando o SSE
            sse = sse + cluster_err

        self.sse_baseline = sse

    def run(self, X):
        '''
        Info:
            Funcao que executa os passos do algoritmo k-means.
        ----------
        Input:
            X: Dados utilizados no algoritmo
        ----------
        Output:
            None
        '''
        # Armazenando as informacoes dos dados
        self.X = X

        # Gerando os centroides iniciais
        print("Gerando os centroides iniciais")
        self.inicializacao()

        # Gerando os primeiros clusters
        print("Gerando primeiros clusters")
        self.clusterizacao()

        # Convergindo
        print("Convergindo")
        self.convergir()

        # Calculando o SSE
        print("Calculando o SSE")
        self.calculate_sse_baseline()
```

Para o dataset wine

In [8]:
```
# Rodando o algoritmo 5 vezes
kmeans_wine_list = []
kmeans_wine_clusters = []
kmeans_wine_centroids = []
for loop in range(5):
    print(f"Executando loop: {loop+1}")

    # Definindo o modelo
    kmeans_wine = KMeans(n_clusters = k_wine)

    # Obtendo apenas as features do dataframe
    X_wine = df_wine.drop('target', axis = 1)

    # Treinando o modelo
    kmeans_wine.run(X_wine)
    print()

    # Salvando os resultados obtidos
    kmeans_wine_list.append(kmeans_wine.sse_baseline)
    kmeans_wine_clusters.append(kmeans_wine.clusters)
    kmeans_wine_centroids.append(kmeans_wine.centroids)
```

```
Executando loop: 1
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 2
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 3
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 4
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 5
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE
```

Para o dataset breast cancer

In [9]:
```
kmeans_wdbd_list = []
kmeans_wdbd_clusters = []
kmeans_wdbd_centroids = []
for loop in range(5):
    print(f"Executando loop: {loop+1}")
    # Definindo o modelo
    kmeans_wdbd = KMeans(n_clusters = k_wdbd)

    # Obtendo apenas as features do dataframe
    X_wdbd = df_wdbd.drop('target', axis = 1)

    # Treinando o modelo
    kmeans_wdbd.run(X_wdbd)
    print()

    # Salvando os resultados
    kmeans_wdbd_list.append(kmeans_wdbd.sse_baseline)
    kmeans_wdbd_clusters.append(kmeans_wdbd.clusters)
    kmeans_wdbd_centroids.append(kmeans_wdbd.centroids)
```

```
Executando loop: 1
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 2
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 3
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 4
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 5
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE
```

Para o dataset iris

In [10]:
```
kmeans_iris_list = []
kmeans_iris_clusters = []
kmeans_iris_centroids = []
for loop in range(5):
    print(f"Executando loop: {loop+1}")

    # Definindo o modelo
    kmeans_iris = KMeans(n_clusters = k_iris)


    # Obtendo apenas as features do dataframe
    X_iris = df_iris.drop('target', axis = 1)

    # Treinando o modelo
    kmeans_iris.run(X_iris)
    print()

    # Salvando os resultados
    kmeans_iris_list.append(kmeans_iris.sse_baseline)
    kmeans_iris_clusters.append(kmeans_iris.clusters)
    kmeans_iris_centroids.append(kmeans_iris.centroids)
```

```
Executando loop: 1
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 2
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 3
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 4
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE

Executando loop: 5
Gerando os centroides iniciais
Gerando primeiros clusters
Convergindo
Calculando o SSE
```

Resultados

In [11]:
```
# Resultados de cada loop
df_kmeans = pd.DataFrame(data = {'Wine': kmeans_wine_list, 'Breast cancer': kmeans_wdbd_list, 'Iris': kmeans_ir

df_kmeans.index = df_kmeans.index.values + 1
df_kmeans.index.name = 'Loop'

df_kmeans
```

Out[11]:

Loop	Wine	Breast cancer	Iris
1	8.397402e+07	8.828366e+08	2294.927591
2	3.665582e+07	3.595177e+08	1773.377738
3	3.281658e+07	8.828366e+08	1801.526418
4	3.281658e+07	3.595177e+08	2031.319271
5	5.986190e+07	8.828366e+08	2171.476049

In [12]:
```
# Variaveis estatisticas (media, mediana e desvio padrao)
df_stat_kmeans = pd.DataFrame(data = {'Média': df_kmeans.mean().values, 'Mediana': df_kmeans.median().values, '
                              index = ['Wine', 'Breast cancer', 'Iris'])
df_stat_kmeans
```

Out[12]:

	Média	Mediana	Desvio padrão
Wine	4.922498e+07	3.665582e+07	2.245604e+07
Breast cancer	6.735090e+08	8.828366e+08	2.866336e+08
Iris	2.014525e+03	2.031319e+03	2.275202e+02

K-Means com DE

In [31]:
```
def calcula_sse_de(centroids, *args):
    '''
    Info:
        Funcao que calcula o SSE utilizada nos algoritmos de otimizacao.
    ----------
    Input:
        centroids: Array com os centroides obtidos
    ----------
    Output:
        sse: Soma do quadrado dos erros para todos os clusters
    '''

    # Definindo variaveis
    X = args[0] # Amostras
    clusters = args[1] # Coluna com os clusters das amostras
    n_clusters = len(clusters['clusters'].value_counts()) # Numero de clusters
    n_features = X.shape[1] # Numero de features
    n_coords = len(centroids) # Numero total de coordenadas de todos os centroides

    # Juntando informacoes de dados e clusters
    data = pd.concat([X, clusters], axis = 1)
    sse = 0

    # Variando os clusters
    for k in data['clusters'].value_counts().index.to_list():

        # Selecionando apenas as amostras do cluster k e removendo a coluna de cluster
        data_cluster = data[data['clusters'] == k].drop(columns = 'clusters')

        # Obtendo as coordenadas do centroide do cluster k
        ind = int((k - 1) * (n_coords/n_clusters))
        centr_coord = centroids[ind:ind + n_features]

        # Calculando a distancia ao quadrado das amostras ao centroide do cluster
        cluster_err = 0
        for i in data_cluster.index:
            sample = data_cluster.loc[i].values

            try:
                # Calculando a distancia
                dist = (np.linalg.norm(sample - centr_coord) ** 2)

                cluster_err = cluster_err + dist
            except:
                pass

        # Calculando o SSE
        sse = sse + cluster_err

    return sse
```

Para o dataset wine

In [32]:
```
# Definindo os limites de busca para os elementos (coordenadas) do vetor  de busca
bounds_wine = []
for i in range(len(kmeans_wine.max_lim)):
    bounds_wine.append((kmeans_wine.min_lim[i], kmeans_wine.max_lim[i]))

bounds_wine = bounds_wine * k_wine


# Rodando o algoritmo 5 vezes
list_result_wine = []
for loop in range(5):
    print(f"Executando loop {loop + 1}")

    # Selecionando aleatoriamente um conjunto de clusters e de centroides para utilizar no algoritmo
    ind = random.choice(list(range(5)))

    result_wine = differential_evolution(func = calcula_sse_de, bounds = bounds_wine,
                                        args = (kmeans_wine.X, kmeans_wine_clusters[ind]),
                                        strategy = 'rand1bin', x0 = np.array(kmeans_wine_centroids[ind]).ravel


    results_wine = (result_wine.x, result_wine.fun)

    # Salvando o resultado
    list_result_wine.append(result_wine.fun)
```

```
Executando loop 1
Executando loop 2
Executando loop 3
Executando loop 4
Executando loop 5
```

Para o dataset breast cancer

In [33]:
```
# Definindo os limites de busca para os elementos (coordenadas) do vetor  de busca
bounds_wdbd = []
for i in range(len(kmeans_wdbd.max_lim)):
    bounds_wdbd.append((kmeans_wdbd.min_lim[i], kmeans_wdbd.max_lim[i]))

bounds_wdbd = bounds_wdbd * k_wdbd

# Executando o algoritmo 5 vezes
list_result_wdbd = []
for loop in range(5):
    print(f"Executando loop {loop + 1}")

    # Selecionando aleatoriamente um conjunto de clusters e de centroides para utilizar no algoritmo
    ind = random.choice(list(range(5)))

    result_wdbd = differential_evolution(func = calcula_sse_de, bounds = bounds_wdbd,
                                        args = (kmeans_wdbd.X, random.choice(kmeans_wdbd_clusters)),
                                        strategy = 'rand1bin', x0 = np.array(kmeans_wdbd_centroids[ind]).ravel


    results_wdbd = (result_wdbd.x, result_wdbd.fun)

    # Salvando o resultado
    list_result_wdbd.append(result_wdbd.fun)
```

```
Executando loop 1
Executando loop 2
Executando loop 3
Executando loop 4
Executando loop 5
```

Para o dataset iris

In [34]:
```
# Definindo os limites de busca para os elementos (coordenadas) do vetor  de busca
bounds_iris = []
for i in range(len(kmeans_iris.max_lim)):
    bounds_iris.append((kmeans_iris.min_lim[i], kmeans_iris.max_lim[i]))

bounds_iris = bounds_iris * k_iris

# Executando o algoritmo 5 vezes
list_result_iris = []
for loop in range(5):
    print(f"Executando loop {loop + 1}")

    # Selecionando aleatoriamente um conjunto de clusters e de centroides para utilizar no algoritmo
    ind = random.choice(list(range(5)))

    result_iris = differential_evolution(func = calcula_sse_de, bounds = bounds_iris,
                                        args = (kmeans_iris.X, random.choice(kmeans_iris_clusters)),
                                        strategy = 'rand1bin', x0 = np.array(kmeans_iris_centroids[ind]).ravel


    results_iris = (result_iris.x, result_iris.fun)

    # Salvando o resultado
    list_result_iris.append(result_iris.fun)
```

```
Executando loop 1
Executando loop 2
Executando loop 3
Executando loop 4
Executando loop 5
```

Resultados

In [35]:
```
# Resultados de cada loop
df_kmeans_de = pd.DataFrame(data = {'Wine': list_result_wine, 'Breast cancer': list_result_wdbd, 'Iris': list_r
                           )

df_kmeans_de.index = df_kmeans_de.index.values + 1
df_kmeans_de.index.name = 'Loop'
```

In [36]:
```
# Variaveis estatisticas (media, mediana e desvio padrao)
df_stat_kmeans_de = pd.DataFrame(data = {'Média': df_kmeans_de.mean().values, 'Mediana': df_kmeans_de.median().
                                 index = ['Wine', 'Breast cancer', 'Iris'])
df_stat_kmeans_de
```

Out[36]:

	Média	Mediana	Desvio padrão
Wine	2.583982e+06	2.633555e+06	117557.090514
Breast cancer	7.794310e+07	7.794310e+07	0.090763
Iris	1.062185e+02	7.885567e+01	37.470079

K-Means com ES

- Serão analisados dois tipos de ES, o (mu, lambda)-ES e o (mu + lambda)-ES

In [13]:
```
def calcula_sse_es(centroids, *args):
    '''
    Info:
        Funcao que calcula o SSE utilizada nos algoritmos de otimizacao.
    ----------
    Input:
        centroids: Array com os centroides obtidos
    ----------
    Output:
        sse: Soma do quadrado dos erros para todos os clusters
    '''
    # Definindo variaveis
    X = args[0][0] # Amostras
    clusters = args[0][1] # Coluna com os clusters das amostras
    n_clusters = len(clusters['clusters'].value_counts()) # Numero de clusters
    n_features = X.shape[1] # Numero de features
    n_coords = len(centroids) # Numero total de coordenadas de todos os centroides

    # Juntando informacoes de dados e clusters
    data = pd.concat([X, clusters], axis = 1)

    sse = 0
    # Variando os clusters
    for k in data['clusters'].value_counts().index.to_list():

        # Selecionando apenas as amostras do cluster k e removendo a coluna de cluster
        data_cluster = data[data['clusters'] == k].drop(columns = 'clusters')

        # Obtendo as coordenadas do centroide do cluster k
        ind = int((k - 1) * (n_coords/n_clusters))
        centr_coord = centroids[ind:ind + n_features]

        # Calculando a distancia ao quadrado das amostras ao centroide do cluster
        cluster_err = 0
        for i in data_cluster.index:
            sample = data_cluster.loc[i].values

            try:
                # Calculando a distancia
                dist = (np.linalg.norm(sample - centr_coord) ** 2)

                cluster_err = cluster_err + dist
            except:
                pass

        # Calculando o SSE
        sse = sse + cluster_err

    return sse
```

In [22]:
```
def in_bounds(point, bounds):
    '''
    Info:
        Funcao que checa se um dado ponto esta dentro do limite de busca.
    ----------
    Input:
        point: Ponto analisado
        bounds: Limites de busca
    ----------
    Output:
        True ou False dependendo do resultado encontrado
    '''
    # Enumerate all dimensions of the point
    for d in range(len(bounds)):

        # Check if out of bounds for this dimension
        if point[d] < bounds[d, 0] or point[d] > bounds[d, 1]:
            return False
    return True

def es_comma(objective, population, bounds, n_iter, step_size, mu, lam, *args):
    '''
    Info:
        Funcao que realiza o algoritmo (mu, lambda)-ES.
    ----------
    Input:
        objective: Funcao objetivo
        population: Populacao inicial
        bounds: Limites de busca
        n_iter: Numero de iteracoes do algoritmo
        step_size:
        mu: Numero de pais selecionados
        lam: Numero de filhos gerados pelos pais
    ----------
    Output:
        best: Parametros do melhor resultado
        best_eval: Melhor resultado
    '''
    best, best_eval = None, 1e+10

    # Calculando o numero de filhos
    n_children = int(lam / mu)

    # Realizando a busca
    for epoch in range(n_iter):

        # Avaliando a fitness para a populacao
        scores = [objective(c, *args) for c in population]

        # Ordenando os scores
        ranks = argsort(argsort(scores))

        # Selecionando os indexes para o melhor mu
        selected = [i for i,_ in enumerate(ranks) if ranks[i] < mu]

        # Criando os filhos
        children = list()
        for i in selected:

            # Verificando se os pais sao a melhor solucao
            if scores[i] < best_eval:
                best, best_eval = population[i], scores[i]

            for _ in range(n_children):
                child = None
                while child is None or not in_bounds(child, bounds):
                    child = population[i] + randn(len(bounds)) * step_size
                children.append(child)

        # Substituindo a populacao pelos filhos
        population = children

    return [best, best_eval]
```

Para o dataset wine

In [27]:
```
# Definindo os limites de busca para os elementos (coordenadas) do vetor  de busca
bounds_wine = []
for i in range(len(kmeans_wine.max_lim)):
    bounds_wine.append((kmeans_wine.min_lim[i], kmeans_wine.max_lim[i]))

bounds_wine = np.array(bounds_wine)

# Definindo variaveis
n_iter = 100 # Total de iteracoes
step_size = 0.15 # Step size maximo
mu = 20 # Numero de pais selecionados
lam = 60 # Tamanho da populacao

# Rodando o algoritmo 5 vezes
list_score_mu_lam_wine = []
for loop in range(5):
    print(f"Executando loop {loop + 1}")

    # Selecionando aleatoriamente um conjunto de clusters e de centroides para utilizar no algoritmo
    ind = random.choice(list(range(5)))

    # Executando o algoritmo
    best, score = es_comma(calcula_sse_es, np.array(kmeans_wine_centroids[ind]),
                           bounds_wine, n_iter, step_size, mu, lam, (kmeans_wine.X, kmeans_wine_clusters[ind])
                           )

    # Salvando o resultado
    list_score_mu_lam_wine.append(score)
```

```
Executando loop 1
Executando loop 2
Executando loop 3
Executando loop 4
Executando loop 5
```

Para o dataset breast cancer

In [26]:

```python
# Definindo os limites de busca para os elementos (coordenadas) do vetor  de busca
bounds_wdbd = []
for i in range(len(kmeans_wdbd.max_lim)):
    bounds_wdbd.append([kmeans_wdbd.min_lim[i], kmeans_wdbd.max_lim[i]])

bounds_wdbd = np.array(bounds_wdbd)

# Definindo variaveis
n_iter = 100 # Total de iteracoes
step_size = 0.15 # Step size maximo
mu = 20 # Numero de pais selecionados
lam = 60 # Tamanho da populacao

# Rodando o algoritmo 5 vezes
list_score_mu_lam_wdbd = []
for loop in range(5):
    print(f"Executando loop {loop + 1}")

    # Selecionando aleatoriamente um conjunto de clusters e de centroides para utilizar no algoritmo
    ind = random.choice(list(range(5)))

    # Executando o algoritmo
    best, score = es_comma(calcula_sse_es, np.array(kmeans_wdbd_centroids[ind]),
                           bounds_wdbd, n_iter, step_size, mu, lam, (kmeans_wdbd.X, kmeans_wdbd_clusters[ind])
                          )

    # Salvando o resultado
    list_score_mu_lam_wdbd.append(score)
```

```
Executando loop 1
Executando loop 2
Executando loop 3
Executando loop 4
Executando loop 5
```

Para o dataset iris

In [24]:

```python
# Definindo os limites de busca para os elementos (coordenadas) do vetor  de busca
bounds_iris = []
for i in range(len(kmeans_iris.max_lim)):
    bounds_iris.append([kmeans_iris.min_lim[i], kmeans_iris.max_lim[i]])

bounds_iris = np.array(bounds_iris)

# Definindo variaveis
n_iter = 100 # Total de iteracoes
step_size = 0.15 # Step size maximo
mu = 20 # Numero de pais selecionados
lam = 60 # Tamanho da populacao

# Rodando o algoritmo 5 vezes
list_score_mu_lam_iris = []
for loop in range(5):
    print(f"Executando loop {loop + 1}")

    # Selecionando aleatoriamente um conjunto de clusters e de centroides para utilizar no algoritmo
    ind = random.choice(list(range(5)))


    # Executando o algoritmo
    best, score = es_comma(calcula_sse_es, np.array(kmeans_iris_centroids[ind]),
                           bounds_iris, n_iter, step_size, mu, lam, (kmeans_iris.X, kmeans_iris_clusters[ind])
                          )

    # Salvando o resultado
    list_score_mu_lam_iris.append(score)
```

```
Executando loop 1
Executando loop 2
Executando loop 3
Executando loop 4
Executando loop 5
```

Resultados

In [28]:

```python
# Resultados de cada loop
df_kmeans_es_mu_lam = pd.DataFrame(data = {'Wine': list_score_mu_lam_wine, 'Breast cancer': list_score_mu_lam_w
                                   )
df_kmeans_es_mu_lam.index = df_kmeans_es_mu_lam.index.values + 1
df_kmeans_es_mu_lam.index.name = 'Loop'
```

In [29]:

```python
# Variáveis estatisticas (média, mediana e desvio padrao)
df_stat_kmeans_es_mu_lam = pd.DataFrame(data = {'Média': df_kmeans_es_mu_lam.mean().values, 'Mediana': df_kmean
                                        index = ['Wine', 'Breast cancer', 'Iris'])
df_stat_kmeans_es_mu_lam
```

Out[29]:

	Média	Mediana	Desvio padrão
Wine	6.070895e+05	5.100956e+05	1.656324e+05
Breast cancer	3.688918e+07	2.855968e+07	1.140564e+07
Iris	3.571542e+01	3.829082e+01	5.758758e+00