TUBES ML

13515055 | Rizky Faramita
13515064 | Tasya
13515140 | Francisco Kenandi Cahyono

Data Handling

In [6]:
```
from sklearn import datasets
iris = datasets.load_iris().data
target = datasets.load_iris().target
```

Public Library & Importing Libraries

In [7]:
```
from sklearn.metrics.pairwise import euclidean_distances,manhattan_distances
import numpy as np

def calculateEucledianDistance(data):
    return euclidean_distances(data)

def calculateManhattanDistance(data):
    return manhattan_distances(data)

def eucledianDistance(a,b):
    dist = 0
    for i in range(len(a)):
        dist += np.power((a[i]-b[i]),2)
    return np.sqrt(dist)

def manhattanDistance(a,b):
    dist = 0
    for i in range(len(a)):
        dist += abs(a[i]-b[i])
    return dist

def checkWithScikit(customLabel,scikitLabel):
    if len(customLabel) == len(scikitLabel):
        same = True
        for i in range(len(customLabel)):
            if (customLabel[i] != scikitLabel[i]):
                print("Oops, it's different.")
                same = False
                break
        if same:
            print("Yes, it's the same!!")
    else:
        print("Oops, it's different.")

def calculatePurity(true_label,predicted_label,ncluster):
    n = len(true_label)
    temp_sum = 0
    for i in range(ncluster):
        n_label = [0,0,0]
        for j in range(len(predicted_label)):
            if predicted_label[j] == i:
                if true_label[j] == 0:
                    n_label[0]+=1
                elif true_label[j] == 1:
                    n_label[1]+=1
                elif true_label[j] == 2:
                    n_label[2]+=1
        temp_sum += max(n_label)

    return (1/n * (temp_sum))
```

K-MEANS

K-Means merupakan salah satu metode *clustering* yang sensitif terhadap nilai inisialisasi. Pada

metode ini, banyaknya kluster yang ingin dibuat sudah diketahui sejak awal.

Pada dasarnya, akan diinisialisasi sentroid awal untuk masing-masing kluster. Kemudian, dihitung jarak dari setiap data ke setiap sentroid yang ada. Data tersebut akan dimasukkan ke dalam suatu klulster yang sentroidnya paling dekat dengan data. Kemudian, akan dihitung kembali sentroid setiap kluster dengan cara menghitung *mean*-nya.

Langkah di atas akan diulang secara terus-menerus sehingga tidak terjadi perubahan sentroid ketika perhitungan *mean*.

Pseudocode:

1. Masukkan banyak kluster yang ingin dibuat dan pilih sentroid awal untuk masing-masing kluster.
2. Ulangi: Masukkan (ulang) setiap data ke dalam kluster terdekat, kemudian *update* sentroid setiap kluster dengan cara menghitung *means*-nya.
3. Berhenti ketika sentroid dari setiap kluster tidak ada yang berubah.

Kompleksitas dari algoritma K-Means adalah O(nkt) dengan t adalah jumlah iterasi, k adalah banyaknya kluster, dan n adalah banyaknya data yang ingin dikluster.

Source Code:

In [37]:
```
# KMeans Function Implementation

# calculate Euclidean distance
def euclDistance(vector1, vector2):
    return sqrt(sum(power(vector2 - vector1, 2)))

# init centroids with random samples
def initCentroids(dataSet, k):
    numSamples, dim = dataSet.shape
    centroids = zeros((k, dim))
    for i in range(k):
        index = int(random.uniform(0, numSamples))
        centroids[i, :] = dataSet[index, :]
    return centroids

# k-means cluster
def kmeans(dataSet, k):
    numSamples = dataSet.shape[0]
    clusterAssment = mat(zeros((numSamples, 2)))
    clusterChanged = True

    ## step 1: init centroids
    centroids = initCentroids(dataSet, k)

    while clusterChanged:
        clusterChanged = False
        ## for each sample
        for i in range(numSamples):
            minDist  = 100000.0
            minIndex = 0
            ## for each centroid
            ## step 2: find the centroid who is closest
            for j in range(k):
                distance = euclDistance(centroids[j, :], dataSet[i, :])
                if distance < minDist:
                    minDist  = distance
                    minIndex = j

            ## step 3: update its cluster
            if clusterAssment[i, 0] != minIndex:
                clusterChanged = True
                clusterAssment[i, :] = minIndex, minDist**2

        ## step 4: update centroids
        for j in range(k):
            pointsInCluster = dataSet[nonzero(clusterAssment[:, 0].A == j)[0]]
            centroids[j, :] = mean(pointsInCluster, axis = 0)

        model = []
        for c in clusterAssment[:,0]:
            model = append(model, c)

    return model

# OUR KMEANS
kmeans_custom = kmeans(iris, 3)
```

```
print("Clustering with custom KMeans")
print(kmeans_custom)
print()

# SCIKIT'S KMEANS
kmeans_sklearn = KMeans(n_clusters=3, init='k-means++', n_init=10, max_iter=300, 
                verbose=0, random_state=None, copy_x=True, n_jobs=1, algorithm='a
print("Clustering with sklearn's KMeans:")
print(kmeans_sklearn)
print()

# Calculate Purity
print("Purity custom KMeans: ", calculatePurity(target, kmeans_custom, 3))
print("Purity scikit's KMeans: ", calculatePurity(target, kmeans_sklearn, 3))
```

```
Clustering with custom KMeans
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 1. 2. 1. 1. 1. 1. 2. 1. 1. 1. 1. 1. 1. 2. 2. 1. 1. 1. 1. 2.
 1. 2. 1. 2. 1. 1. 2. 2. 1. 1. 1. 1. 1. 2. 1. 1. 1. 1. 2. 1. 1. 1. 2. 1.
 1. 1. 2. 1. 1. 2.]

Clustering with sklearn's KMeans:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]

Purity custom KMeans:  0.8866666666666667
Purity scikit's KMeans:  0.8933333333333334
```

K-MEDOIDS

K-Medoids pada dasarnya mirip dengan K-Means. Letak perbedaan K-Medoids dengan K-Means terletak pada *update* sentroid. Jika pada K-Means cara meng-*update* sentroid tiap kluster adalah dengan menghitung *means* dari kluster tersebut, namun ada K-Medoids akan diambil satu data untuk menggantikan salah satu sentroid yang ada. Dengan cara ini, K-Medoids lebih rigid terhadap pencilan (*outliers*) dibandingkan K-Means.

Karena pemilihan data *random* memiliki banyak kemungkinan, biasanya K-Medoids akan berhenti setelah beberapa data yang dipilih menghasilkan kluster dengan *error* yang lebih besar.

Pseudocode:

1. Masukkan banyak kluster yang ingin dibuat dan pilih sentroid awal untuk masing-masing kluster.
2. Ulangi: Masukkan (ulang) setiap data ke dalam kluster terdekat, kemudian *update* sentroid setiap kluster dengan cara memilih salah satu data *random* untuk menggantikan sentroid yang sudah ada.

3. Berhenti ketika *error* yang dihasilkan minimal atau banyaknya data *random* yang ditemukan untuk menggantikan sentroid lainnya tidak menghasilkan *error* yang lebih kecil melebihi *threshold*.

Source Code:

In [45]:
```
# KMedoids Function Implementation
from numpy import *

# calculate Manhattan distance
def manhatDistance(vector1, vector2):
    return sum(fabs(vector2 - vector1))

# init centroids with random samples
def initCentroids(dataSet, k):
    numSamples, dim = dataSet.shape
    centroids = zeros((k, dim))
    for i in range(k):
        index = int(random.uniform(0, numSamples))
        centroids[i, :] = dataSet[index, :]
    return centroids

def updateCentroids(dataSet, k, centroids, clusterAssment):
    clust = int(random.uniform(0,k))
    numSamples = dataSet.shape[0]
    idx = int(random.uniform(0, numSamples))
    while clusterAssment[idx, 0] != clust:
        idx = int(random.uniform(0, numSamples))
    centroids[clust,:] = dataSet[idx, :]
    return centroids

# k-medoids cluster
def kmedoids(dataSet, k):
    numSamples = dataSet.shape[0]
    clusterAssment = mat(zeros((numSamples, 2)))
    clusterChanged = True

    ## step 1: init centroids
    centroids = initCentroids(dataSet, k)
    error = 0

    while clusterChanged:
        clusterChanged = False
        ## for each sample
        for i in range(numSamples):
            minDist  = 100000.0
            minIndex = 0
            ## for each centroid
            ## step 2: find the centroid who is closest
            for j in range(k):
                distance = manhatDistance(centroids[j, :], dataSet[i, :])
                if distance < minDist:
                    minDist  = distance
                    minIndex = j

            ## step 3: update its cluster
            if clusterAssment[i, 0] != minIndex:
                clusterChanged = True
                clusterAssment[i, :] = minIndex, minDist

        ## step 4: update centroids
        new_error = sum(clusterAssment[:, 1])
```

```
        if new_error < error:
            clusterChanged = False
            error = new_error
        else:
            centroids = updateCentroids(dataSet,k,centroids,clusterAssment)

        model = []
        for c in clusterAssment[:,0]:
            model = append(model, c)
    return model

# OUR KMEDOIDS
kmedoids_custom = kmedoids(iris, 3)
print("Clustering with custom KMedoids:")
print(kmedoids_custom)
print()

# SCIKIT'S KMEDOIDS
from pyclustering.cluster import kmedoids
kmedoids_sklearn_init = kmedoids.kmedoids(iris, [0,50,100],ccore='True')
kmedoids_sklearn_init.process()
kmedoids_sklearn_model = kmedoids_sklearn_init.get_clusters()
i = 0
kmedoids_sklearn = np.arange(150)
for c in kmedoids_sklearn_model:
    for cj in c:
        kmedoids_sklearn[cj] = i
    i += 1
print("Clustering with scikit's KMedoids:")
print(kmedoids_sklearn)
print()

# Calculate Purity
print("Purity custom KMedoids: ", calculatePurity(target, kmedoids_custom, 3))
print("Purity scikit's KMedoids: ", calculatePurity(target, kmedoids_sklearn, 3))
```

```
Clustering with custom KMedoids:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 2. 1. 2. 2. 2. 2. 1. 2. 2. 2. 2. 1. 2. 1. 1. 2. 2. 2. 2. 1.
 2. 1. 2. 1. 2. 2. 1. 1. 2. 2. 2. 2. 2. 1. 1. 2. 2. 2. 1. 2. 2. 2. 1. 2.
 2. 2. 1. 2. 2. 1.]

Clustering with scikit's KMedoids:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 1 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]

Purity custom KMedoids:  0.8933333333333334
Purity scikit's KMedoids:  0.9
```

DBSCAN

DBSCAN adalah algoritma yang termasuk ke dalam Density-based Clustering. Oleh karena itu, definisi kluster adalah daerah padat dalam ruang data yang dipisahkan oleh daerah dengan kepadatan objek yang lebih rendah. Sebuah area dapat disebut sebagai sebuah kluster apabila pada daerah tersebut terdapat data paling sedikit MinPts (didefinisikan pada awal klustering). *High density* adalah apabila di area dengan jari-jari epsilon, terdapat paling sedikit data sebanyak MinPts (epsilon didefinisikan di awal juga).

Sebuah data disebut *core* data apabila di sekeliling data tersebut (dengan jari-jari epsilon) terdapat minimal MinPts banyak data. *Border* data adalah data yang bukan *core* namun data tersebut masih terletak di dekat *core* data. *Outlier* adalah data yang terpisah dari data-data lain atau yang sering disebut juga sebagai pencilan.

Pseudocode:

Algoritma dimulai dengan sembarang data D, kemudian mencari tetangga-tetangga di sekitar D yang dapat dicapai (yaitu masih dicakup pada jari=jari sebesar epsilon). a. Jika D adalah sebuah *core*, iterasi ini akan menghasilkan sebuah kluster. b. Jika D adalah sebuah *border*, tidak akan terbentuk kluster baru dan algoritma ini akan menghitung data lain dari kumpulan data yang ingin dilakukan klustering.

Source Code:

In [9]:
```
# DBSCAN Function Implementation
class customDBSCAN:

    def __init__(self,data,e,minPts,distanceFunction):
        self.data = data
        self.e = e
        self.num_data = len(self.data)
        self.minPts = minPts
        self.labels = []
        self.cores = {}
        for x in range(len(self.data)):
            self.labels.append(-1)
        if distanceFunction == 'eucledian':
            self.distanceMatrix = calculateEucledianDistance(self.data)
        elif distanceFunction == 'manhattan':
            self.distanceMatrix = calculateManhattanDistance(self.data)
    def findCores(self):
        for i in range(self.num_data):
            temp_cluster = []
            for j in range(self.num_data):
                if (self.distanceMatrix[i][j] <= self.e and i != j):
                    temp_cluster.append(j)
            if(len(temp_cluster) >= self.minPts):
                self.cores[i] = temp_cluster

    def dfs_dbscan(self,dict_type_data,current_label):
        if(self.labels[dict_type_data] == -1):
            self.labels[dict_type_data] = current_label
            if dict_type_data in self.cores:
                for i in self.cores[dict_type_data]:
                    self.dfs_dbscan(i,current_label)

    def fit(self):
        label = -1
        self.findCores()
        for k,v in self.cores.items():
            if (self.labels[k] == -1):
                label+=1
            self.dfs_dbscan(k,label)
```

In [16]:
```
# MAIN
distanceFunction = 'eucledian'
epsilon = 1
minPts = 17

# OUR DBSCAN
db = customDBSCAN(iris,epsilon,minPts,distanceFunction)
db.fit()
print("Clustering with custom DBSCAN: ")
print(db.labels)
print()

# SCIKIT'S DBSCAN
from sklearn.cluster import DBSCAN
clustering = DBSCAN(eps=epsilon, min_samples=minPts).fit(iris)
print("Clustering with scikit's DBSCAN: ")
print(clustering.labels_)
print()

# Compare custom DBSCAN with SCIKIT's
print("Compare with scikit:")
checkWithScikit(db.labels,clustering.labels_)
print()

# Calculate Purity
print("Purity custom DBSCAN: ", calculatePurity(target,db.labels,len(set(db.label
print("Purity scikit's DBSCAN: ", calculatePurity(target,clustering.labels_,3))
```

```
Clustering with custom DBSCAN:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Clustering with scikit's DBSCAN:
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1 -1  1
  1  1  1  1  1  1  1  1  1  1  1 -1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1]

Compare with scikit:
Yes, it's the same!!

Purity custom DBSCAN:  0.6666666666666667
Purity scikit's DBSCAN:  0.6666666666666667
```

AGGLOMERATIVE

Agglomerative merupakan salah satu metode *hierarchical clustering*. Pada dasarnya, *hierarchical*

clustering ini ada dua macam, yaitu *top-down* (dari satu kluster besar dipecah menjadi kluster-kluster kecil, disebut juga Divisive Clustering) dan *bottom-up* (dari banyak kluster kecil dijadikan satu kluster besar yang anggotanya adalah semua data yang ada, disebut Agglomerative Clustering). Sama seperti K-Means dan DBSCAN, Agglomerative juga butuh banyaknya kluster secara eksplisit dinyatakan ketika ingin melakukan klustering. Hal ini disebabkan oleh metode Agglomerative akan melakukan iterasi secara terus-menerus hingga membentuk satu kluster besar apabila tidak ada *threshold* banyaknya kluster yang harus dibuat. Dengan adanya batasan banyaknya kluster yang harus dibuat, maka algoritma ini akan berhenti hingga banyak kluster yang dibuat sudah sesuai dengan keinginan pengguna.

Dalam satu iterasi, metode Agglomerative hanya akan menggabungkan satu data ke dalam kluster terdekat dengan dirinya. Cara perhitungan jarak pada algoritma ini ada 4 macam:

1. Single: jarak antarkluster adalah jarak terdekat antara salah satu objek dari kedua kluster tersebut.
2. Complete: jarak antarkluster adalah jarak terjauh antara salah satu objek dari kedua kluster tersebut.
3. Average: jarak antarkluster adalah rata-rata jarak terdekat antara setiap data dari kedua kluster.
4. Average group: jarak antara kedua sentroid (*means*) dari kedua kluster.

Pseudocode:

1. Algoritma dimulai dengan adanya kluster kecil yang banyaknya sesuai dengan banyaknya data (satu kluster hanya berisi satu data).
2. Cari jarak minimal berdasarkan salah satu algoritma jarak yang sudah dijelaskan di atas, gabungkan kluster dengan jarak paling dekat.
3. *Update* matriks jarak dengan menghitung jarak antara kluster baru ke kluster-kluster lainnya.
4. Ulang step 2 dan 3 hingga kluster yang ada sama dengan kluster yang diinginkan atau sampai terbentuk satu kluster besar.

Source Code:

In [17]:
```
#CUSTOM AGGLOMERATIVE
class customAgglomerative:
    def __init__(self,data,n_cluster,linkage='single',distanceFunction='eucledian
        self.data = data
        self.n_cluster = n_cluster

        if linkage == 'single':
            self.linkage = self.singleLinkage
        elif linkage == 'complete':
            self.linkage = self.completeLinkage
        elif linkage == 'average':
            self.linkage = self.averageLinkage
        elif linkage == 'average_group':
            self.linkage = self.averageGroupLinkage

        if distanceFunction == 'eucledian':
            self.distanceFunction = eucledianDistance
        elif distanceFunction == 'manhattan':
            self.distanceFunction = manhattanDistance

        self.distances = {}
        #clusters = list of tuples
        self.clusters = [(idx,)for idx in range(len(self.data))]
        #label of each data's cluster
        self.labels = []
        for i in range(len(self.data)):
            self.labels.append(i)


    def fit(self):

        while len(self.clusters)>self.n_cluster:
            a = 0
            b = 0
            clusterDistance = np.inf

            for i in range(len(self.clusters)):
                for j in range(i+1, len(self.clusters)):
                    temp_clusterDistance = self.getLinkageDistance(self.clusters[
                    if temp_clusterDistance < clusterDistance:
                        a,b,clusterDistance = i,j,temp_clusterDistance

            self.clusters[a] = self.clusters[a] + self.clusters[b]
            del self.clusters[b]

        for i in range(len(self.clusters)):
            for j in self.clusters[i]:
                self.labels[j] = i

        return self.labels


    def getLinkageDistance(self,clust1,clust2):
        dist = self.distances.get((clust1,clust2))

        if dist == None:
```

```
            dist = self.distances.get((clust2,clust1))

        if dist == None:
            self.distances[(clust1,clust2)] = self.linkage(self.data,clust1,clust
            dist = self.distances.get((clust1,clust2))

        return dist


    def singleLinkage(self,nodes,clust1,clust2,distance_function=eucledianDistanc
        dist = np.inf
        for i in clust1:
            for j in clust2:
                dist = min(dist,distance_function(nodes[i],nodes[j]))
        return dist

    def completeLinkage(self,nodes,clust1,clust2,distance_function=eucledianDista
        dist = -np.inf
        for i in clust1:
            for j in clust2:
                dist = max(dist,distance_function(nodes[i],nodes[j]))
        return dist

    def averageLinkage(self,nodes,clust1,clust2,distance_function=eucledianDistan
        dist = 0
        for i in clust1:
            for j in clust2:
                dist += distance_function(nodes[i],nodes[j])
        return dist / (len(clust1)*len(clust2))

    def averageGroupLinkage(self,nodes,clust1,clust2,distance_function=eucledianD
        dist = 0
        mean1 = []
        mean2 = []
        for i in clust1:
            for j in nodes[i]:
                mean1[i]+=j
            mean1[i]= mean1[i]/len(nodes)

        for i in clust2:
            for j in nodes[i]:
                mean2[i]+=j
            mean2[i]= mean2[i]/len(nodes)

        return distance_function(mean1,mean2)
```

In [20]:
```
from sklearn.cluster import AgglomerativeClustering

distanceFunction = 'eucledian'
linkage = 'complete'
n_clusters = 4

scikitClustering = AgglomerativeClustering(linkage = 'complete',n_clusters = n_cl
scikitClustering.fit(iris)

agl = customAgglomerative(iris,n_clusters,linkage,distanceFunction)

print("Clustering with custom Agglomerative:")
print(agl.fit())
print()
print("Clustering with scikit's Agglomerative:")
print(scikitClustering.labels_)
print()

# Compare custom DBSCAN with SCIKIT's
print("Compare with scikit:")
checkWithScikit(agl.labels,scikitClustering.labels_)
print()

# Calculate purity
print("Purity:")
print(calculatePurity(target,agl.labels,n_clusters))
print("Purity custom Agglomerative: ", calculatePurity(target,agl.labels,n_cluste
print("Purity scikit's Agglomerative: ", calculatePurity(target,scikitClustering.
```

```
Clustering with custom Agglomerative:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1,
1, 2, 2, 2, 2, 1, 2, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 3, 1,
1, 3, 2, 3, 1, 3, 1, 1, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 3, 1, 1, 3, 1, 1, 1, 3,
3, 3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Clustering with scikit's Agglomerative:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 2 1 2 1 2 1 2 2 2 2 1 2 1 2 2 1 2 1 2 1 1
 1 1 1 1 1 2 2 2 2 1 2 1 1 1 2 2 2 1 2 2 2 2 2 1 2 2 1 1 3 1 1 3 2 3 1 3 1
 1 1 1 1 1 1 3 3 1 1 1 3 1 1 3 1 1 1 3 3 3 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1
 1 1]

Compare with scikit:
Yes, it's the same!!

Purity:
0.8400000000000001
Purity custom Agglomerative:  0.8400000000000001
Purity scikit's Agglomerative:  0.8400000000000001
```

Implementasi ke Dataset Iris

Implementasi dataset Iris ke fungsi buatan DBSCAN dan Agglomerative menunjukkan hasil clustering yang sama dengan fungsi bawaan dari sklearn. Oleh karena itu, perhitungan kinerja berupa purity juga menunjukkan nilai yang sama. Pada fungsi buatan KMeans dan KMedoids, terdapat perbedaan nilai pada purity yang diakibatkan oleh terpilihnya centroid awal secara random. Namun, berdasarkan pengujian yang dilakukan, perbedaan nilai purity tersebut rata-rata kurang dari 0,5.

Pembagian Tugas

13515055 | Rizky Faramita - KMeans, KMedoids
13515064 | Tasya - Agglomerative
13515140 | Francisco Kenandi Cahyono - DBSCAN