Day 1 :
Klastering pada *Iris Dataset* Menggunakan *K – Medoid*

By : Gustian Herlambang

Catatan : *Ini adalah bagian dari saya belajar mengenai klastering menggunakan algoritma k-Medoids pada dataset bunga Iris. Sepenuhnya ini adalah materi yang di publikasikan oleh Tri Nguyen pada platform Medium yang telah saya cantumkan sitasinya pada bagian referensi. Pada materi ini saya telah menambah dan mengubah kodingan yang ada agar lebih mempermudah saya. Terimakasih.*

`K-Medoid` adalah teknik klastering yang jarang gaungnya atau digunakan untuk mengklaster data. Berbeda cerita dengan **`K-Means`** yang tutorialnya sangat banyak dan menjadi *welcoming unsupervised techniques in machine learning* untuk pemula.

- ## *Introduction*

Baik k-Means maupun k-Medoids, keduanya merupakan algoritma yang partisional. Dimana dataset yang ada akan di masukkan kedalam suatu kelompok data. K-Means bertujuan untuk meminimalkan total kesalahan kuadrat (*total squared error*) dari titik pusat pada setiap klaster, titik pusat ini dalam k-Means dinamakan sentroid (*centroid*).

Disisi lain, k-Medoids mencoba untuk meminimalkan jumlah ketidaksamaan antara objek yang telah dilabeli untuk berada pada suatu klaster dan objek yang memang ditunjuk sebagai objek yang representatif dalam suatu klaster. Nah, klaster representatif ini dinamakan dengan *medoids*.

Berbeda dengan algoritma k-Means dimana centroid menjadi pusatnya atau berada di tengah, posisi rata – rata yang mungkin tidak menjadi poin data pada suatu set atau kumpulan, k-Medoids memilih medoids dari kumpulan data (*dataset*).

- ## *Data Preparation*

Pada kesempatan kali ini, penerapan k-Medoids akan menggunakan dataset bunga Iris yang di proses langsung dari library sklearn. Untuk kepentingan demonstrasi algoritma, tidak dilakukan proses pemisahan data menjadi data latih dan data uji. Namun, akan menggunakan satu set data tunggal untuk melakukan pelatihan (*training*) data dan *training* model.

➢ Memanggil *Library*

Library yang akan digunakan yaitu **Numpy**, **Pandas**, **Matplotlib**, dan **Sklearn**. Ketikkan kode berikut ini pada **Google Colaboratory** atau **Jupyter**.

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from mpl_toolkits.mplot3d import Axes3D
5 from sklearn import datasets
6 from sklearn.decomposition import PCA
```

➢ *Load Dataset*

Dataset di *load* menggunakan kodingan berikut :

```
iris = datasets.load_iris()
data = pd.DataFrame(iris.data,columns = iris.feature_names)

target = iris.target_names
labels = iris.target
```

Catatan :

1. Baris 1, untuk melakukan load data iris yang disimpan dalam variable iris.
2. Baris 2, membaca data kolom dan feature_names dari iris yang disimpan dalam variable data.
3. Baris 3, membaca target_names dari dataset iris.
4. Baris 4, membaca label dari dataset iris.

Selanjutnya dibawah baris ke 5, tambahkan kodingan berikut untuk mencetak 5 data awal dari iris dataset.

```
data.head()
```

Kemudian hasilnya akan seperti gambar dibawah :

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

- **Proses Menskalakan Data dengan *MinMaxScaler***

Setelah data diload, maka data di skalakan dengan *minmaxscaler* dimana kodingannya dapat di lihat dibawah ini :

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data = pd.DataFrame(scaler.fit_transform(data),
columns=data.columns)

# menampilkan 5 data awal setelah di skalakan
data.head()
```

Maka hasil setelah data diskalakan adalah sebagai berikut :

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	0.222222	0.625000	0.067797	0.041667
1	0.166667	0.416667	0.067797	0.041667
2	0.111111	0.500000	0.050847	0.041667
3	0.083333	0.458333	0.084746	0.041667
4	0.194444	0.666667	0.067797	0.041667

- ***Principal Component Analysis* (PCA)**

Ada 4 *features* yang mungkin tidak ideal untuk divisualisasi dan penyesuaian model clustering, karena beberapa di antaranya mungkin sangat berkorelasi. Oleh karena itu, pada kesempatan ini akan menggunakan *Principal Component Analysis* (PCA) untuk mengubah data 4 dimensi menjadi data 3 dimensi sambil menjaga signifikansi prediktor tersebut dengan kelas PCA dari library **`sklearn`**.

Kodingannya adalah sebagai berikut :

```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
```

```
principalComponents = pca.fit_transform(data)
PCAdf = pd.DataFrame(data = principalComponents , columns =
['principal component 1', 'principal component 2','principal
component 3'])

datapoints = PCAdf.values
m, f = datapoints.shape
k = 3

datapoints
```

Dari kodingan tersebut, **Nguyen** selaku *publisher* mendapatkan data reduksi dimensional sebagai berikut :

	principal component 1	principal component 2	principal component 3
0	1.823288	-0.104530	0.501130
1	-1.295968	-0.350770	0.461567
2	-0.219921	-0.041491	0.512261
3	0.724683	0.274496	0.272093
4	-0.271082	0.608131	0.374939

Lain halnya dengan saya data yang direduksi secara dimensional yang sudah saya coba adalah sebagai berikut :

	principal component 1	principal component 2	principal component 3
0	-0.630703	0.107578	-0.018719
1	-0.622905	-0.104260	-0.049142
2	-0.669520	-0.051417	0.019644
3	-0.654153	-0.102885	0.023219
4	-0.648788	0.133488	0.015116

Dengan nilai *array* (diambil 2 kolom) sebagai berikut :

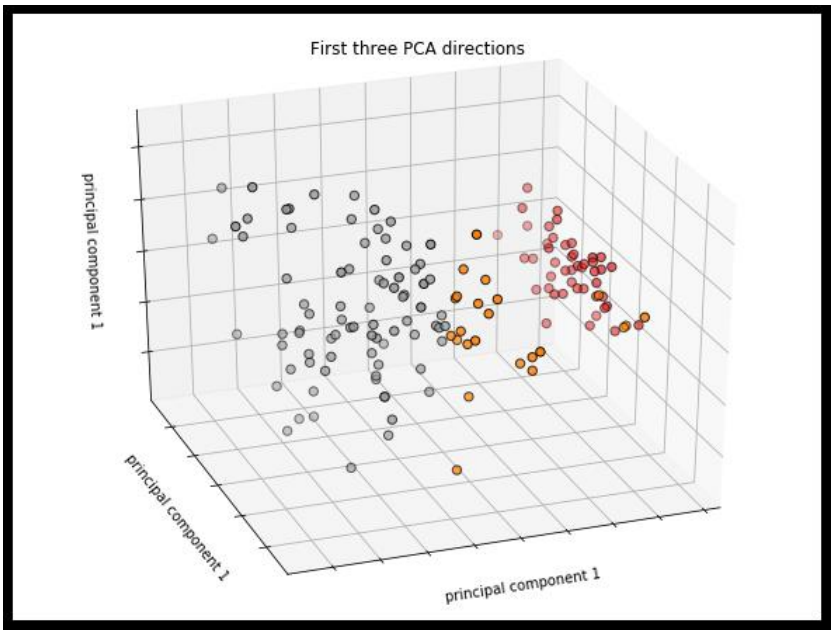
```
array([[-6.30702931e-01,  1.07577910e-01, -1.87190977e-02],
       [-6.22904943e-01, -1.04259833e-01, -4.91420253e-02],
```

Kemudian data yang sudah direduksi dimensinya divisualisasikan dengan **`matplotlib`**, berikut kodenya :

```
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
X_reduced = datapoints
```

```
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2],
c=labels,
           cmap=plt.cm.Set1, edgecolor='k', s=40)
ax.set_title("First three PCA directions")
ax.set_xlabel("principal component 1")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("principal component 1")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("principal component 1")
ax.w_zaxis.set_ticklabels([])
plt.show()
```

Dan hasilnya adalah sebagai berikut :



- ***Implementation***

Pada tahap implementasi ini akan dijelaskan 4 tahapan yaitu : 1) *Medoid Initialization*, 2) *Computing the Distances*, 3) *Cluster Assignment*, 4) *Swap Test* dan yang terakhir 5) *Putting All Together*. Baiklah, pertama – tama adalah mencari fungsi ketidaksamaan. Banyak pilihan yang dapat dilakukan namun pada kesempatan ini saya belajar dari *publisher* nya menggunakan ***Common Minkowski Distance Function***. Kemudian algoritma yang akan digunakan pada implementasi ini adalah *Partitioning Around Medoids* (PAM). Menurut **Kaufman and Rousseeuw 1990**, algoritmanya dapat diringkas menjadi :

1. **Inisialisasi**, yaitu secara acak memilih *k* dari *m* data poin yang sebagai medoids.
2. ***Assignment* (penugasan)**, kaitkan setiap titik data ke medoid terdekat berdasarkan jarak Minkowski.

3. ***Update***, setiap medoid *j* dan setiap data poin *i* yang terhubung dengan *j*, tukar *j* dan *i* dan hitung total *cost of the configuration* (yang mana, ketidaksamaan rata-rata *i* ke semua titik data yang terkait dengan *j*). Pilih medoid *j* dengan biaya konfigurasi terendah. Ulangi antara langkah 2 dan 3 hingga tidak ada perubahan pada *assignment* lagi.

Dalam koding nantinya, akan lebih mudah untuk menggunakan konvensi "matriks data" yaitu dalam matriks data X, setiap baris $\hat{x}$ adalah salah satu dari m pengamatan dan setiap kolom x adalah salah satu fitur f .

$$X \equiv \begin{bmatrix} x_{0,0} \cdots x_{0,f-1} \\ \vdots \\ x_{m-1,0} \cdots x_{m,f-1} \end{bmatrix}$$

➢ *Medoid Initialization*

Untuk memulai algoritma, kita harus punya perkiraan awal. Mari memilih *k* secara acak dari data. Saya memilih 3 karena sesuai dengan label data iris yang punya 3 label juga. Selanjunya, akan dibuat fungsi **`init_medoids(X,k)`**, sehingga secara acak memilih *k* dari pengamatan yang diberikan untuk dijadikan medoid. Ini harus mengembalikan array *NumPy* dengan ukuran $k \times d$, di mana d adalah jumlah kolom X.

```
def init_medoids(X, k):
    from numpy.random import choice
    from numpy.random import seed

    seed(1)
    samples = choice(len(X), size=k, replace=False)
    return X[samples, :]

medoids_initial = init_medoids(datapoints, 3)

print(medoids_initial)
```

Setelah menginisialisasi 3 *medoids* secara acak, maka akan didapatkan matriks *array* :

```
[[-0.60037159  0.38024069 -0.08516953]
 [-0.15863457 -0.28913985  0.0524159 ]
 [ 0.21396272  0.059963   -0.11409813]]
```

➢ *Computing Distances*

Menerapkan fungsi yang menghitung matriks jarak, $S = (s\_ij)$ sehingga $s\_ij = d$ ^ p_ij adalah jarak urutan Minkowski p dari titik x_x ke medoid μ_i. Ini harus mengembalikan matriks **NumPy** S dengan bentuk (m, k). Urutan p ^ th Minkowski Jarak antara dua titik, x dan μ diberikan oleh :

$$d_{ij} = \left(\sum_{a=1}^{f} | x_{ia} - \mu_{ja} |^p \right)^{\frac{1}{p}}, \quad 0 < i < (m-1), \quad 0 < j < (k-1)$$

Jadi, untuk elemen matriks S adalah :

$$s_{ij} = d_{ij}^p = \sum_{a=1}^{f} | x_{ia} - \mu_{ja} |^p$$

Jarak Minkowski biasanya digunakan dengan p menjadi 1 atau 2, yang masing-masing sesuai dengan jarak *Manhattan* dan jarak *Euclidean*. Pada pembelajaran ini akan menggunakan jarak *Euclidean* dalam perhitungannya. Ketik kodingan berikut :

```
def compute_d_p(X, medoids, p):
    m = len(X)
    medoids_shape = medoids.shape
    # kalau sebuah array berdimensi 1 tersedia,
    # itu akan di bentuk ulang ke array single row berdimensi 2
    if len(medoids_shape) == 1:
        medoids = medoids.reshape((1,len(medoids)))
    k = len(medoids)

    S = np.empty((m, k))

    for i in range(m):
        d_i = np.linalg.norm(X[i, :] - medoids, ord=p, axis=1)
        S[i, :] = d_i**p

    return S

S = compute_d_p(datapoints, medoids_initial, 2) #ubah points ke
datapoints

print(S)
```

Maka akan menghasilkan array matriks berikut :

```
[[0.07968064 0.3852937  0.7248244 ]
 [0.23654649 0.26004161 0.73153592]
```

dst

➢ Cluster Assignment

Sekarang kita membangun fungsi yang bekerja pada matriks jarak S untuk menetapkan "label kluster" 0, 1, 2 ke setiap titik menggunakan jarak minimum untuk menemukan medoid "paling mirip".

Artinya, untuk setiap titik, ditunjukkan dengan indeks baris *i*, jika *s_ij* adalah jarak minimum untuk titik *i*, maka indeks *j* adalah label *cluster i*. Dengan kata lain, fungsi tersebut harus mengembalikan larik satu dimensi, *y*, dengan panjang *m* sehingga :

$$y_i = \underset{j \in \{0,\ldots,k-1\}}{\operatorname{argmin}} \; s_{ij}$$

Label untuk titik data dari iterasi pertama menggunakan kode berikut ini :

```
def assign_labels(S):
    return np.argmin(S, axis=1)

labels = assign_labels(S)

print(labels)
```

Menghasilkan array berikut :

```
[0 0 0 0 0 0 0 0 1 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 0 0 1 0 0 0 1 0 0 0 0 2 2 2 1 2 2 2 1 2 1 1 2 1 2 1 2 2 1 2 1 2 2 2 2
 2 2 2 2 2 1 1 1 1 2 2 2 2 2 1 1 2 1 1 1 2 2 2 1 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 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

➢ Swap Test

Dalam langkah ini, untuk setiap medoid *j* dan setiap titik data *i* yang terkait dengan *j* ditukar dengan *j* dan *i* dan hitung total biaya konfigurasi (yaitu rata-rata perbedaan *i* ke semua titik data, ∑*s_ij*) sebagai Langkah 3. Pilih medoid *j* dengan biaya konfigurasi terendah.

Artinya, untuk setiap cluster, telusuri jika ada titik dalam cluster yang menurunkan koefisien ketidaksamaan rata-rata. Pilih titik yang paling banyak menurunkan koefisien ini sebagai medoid baru untuk cluster ini.

Kodenya adalah :

```
def update_medoids(X, medoids, p):
```

```
    S = compute_d_p(datapoints, medoids, p)
    labels = assign_labels(S)

    out_medoids = medoids

    for i in set(labels):
        # change points to datapoints
        avg_dissimilarity = np.sum(compute_d_p(datapoints,
medoids[i], p))
        #change points to datapoints
        cluster_points = datapoints[labels == i]

        for datap in cluster_points:
            new_medoid = datap # ubah datap ke datapoints
            new_dissimilarity= np.sum(compute_d_p(datapoints,
datap, p)) #ubah points ke datapoints

            if new_dissimilarity < avg_dissimilarity :
                avg_dissimilarity = new_dissimilarity

                out_medoids[i] = datap

    return out_medoids
```

Terakhir, kita juga membutuhkan fungsi yang memeriksa apakah medoid telah "bergerak/dipindahkan", mengingat dua contoh nilai medoid. Ini berfungsi untuk memeriksa apakah medoid tidak lagi bergerak dan iterasi harus dihentikan. Menggunakan kodingan berikut :

```
def has_converged(old_medoids, medoids):
    return set([tuple(x) for x in old_medoids]) == set([tuple(x) for
x in medoids])
```

- **Putting All Together**

Pada tahap ini ternyata saya sudah menggabungkan tahapan k-Medoid dari step diatas. Ini kodingan yang sudah saya tambahkan tampilan matriks *array* :

```
def kmedoids(X, k, p, starting_medoids=None, max_steps=np.inf):
    if starting_medoids is None:
        medoids = init_medoids(X, k)
    else:
        medoids = starting_medoids

    converged = False
    labels = np.zeros(len(X))
    i = 1
    while (not converged) and (i <= max_steps):
```

```
        old_medoids = medoids.copy()

        S = compute_d_p(X, medoids, p)

        labels = assign_labels(S)

        medoids = update_medoids(X, medoids, p)

        converged = has_converged(old_medoids, medoids)
        i += 1
    return (medoids,labels)

results = kmedoids(datapoints, 3, 2)
final_medoids = results[0]
data['clusters'] = results[1]

print(final_medoids)
```

Dengan menghasilkan array dari **`final_medoids`** sebagai berikut :

```
[[-0.50609386  0.02794708  0.02628302]
 [-0.00826092 -0.0866611   0.05357911]
 [ 0.03302937 -0.04297085  0.01560933]]
```

Pada kodingan **`data['clusters'] = results[1],`** menunjukkan hasil sebagai berikut :

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	clusters
0	0.222222	0.625000	0.067797	0.041667	0
1	0.166667	0.416667	0.067797	0.041667	0
2	0.111111	0.500000	0.050847	0.041667	0
3	0.083333	0.458333	0.084746	0.041667	0
4	0.194444	0.666667	0.067797	0.041667	0
...	...	...	...	...	...
145	0.666667	0.416667	0.711864	0.916667	2
146	0.555556	0.208333	0.677966	0.750000	2
147	0.611111	0.416667	0.711864	0.791667	2
148	0.527778	0.583333	0.745763	0.916667	2
149	0.444444	0.416667	0.694915	0.708333	2

150 rows × 5 columns

➢ Perhitungan

Setelah saya mempraktekkan kodingan diatas dan melihat versi full kodingan dari GitHub Nguyen, dapat diambil kesimpulan bahwa pada tahapan ini adalah proses perhitungan antara **`final_medoids`** dengan kecocokan label pada dataset Iris,

dimana menghasilkan akurasi sebesar 94,7%. Diantaranya 142 cocok dari 150 dataset Iris yang ada. Berikut ini adalah kodingan terakhirnya :

```
# Proses Perhitungan
def mark_matches(a, b, exact=False):
    """
    Diberikan dua larik Numpy dengan label {0, 1}, mengembalikan boolean yang baru,
     array yang menunjukkan di lokasi mana array input memiliki
     label yang sama (yaitu, entri yang sesuai adalah True).

     Fungsi ini dapat mempertimbangkan kecocokan "tidak tepat". Artinya, jika `tepat`
     False, maka fungsinya akan menganggap label {0, 1} mungkin
     dianggap sama hingga penukaran label. Fitur ini
     memungkinkan

       a == [0, 0, 1, 1, 0, 1, 1]
       b == [1, 1, 0, 0, 1, 0, 0]

     untuk dianggap setara. (Artinya, gunakan `exact = False` saat Anda
     hanya peduli tentang pelabelan "relatif".)
    """
    assert a.shape == b.shape
    a_int = a.astype(dtype=int)
    b_int = b.astype(dtype=int)
    all_axes = tuple(range(len(a.shape)))
    assert ((a_int == 0) | (a_int == 1) | (a_int == 2)).all()
    assert ((b_int == 0) | (b_int == 1) | (b_int == 2)).all()

    exact_matches = (a_int == b_int)
    if exact:
        return exact_matches

    assert exact == False
    num_exact_matches = np.sum(exact_matches)
    if (2*num_exact_matches) >= np.prod (a.shape):
        return exact_matches
    return exact_matches == False # Invert

def count_matches(a, b, exact=False):
    """
    Diberikan dua set label {0, 1}, mengembalikan jumlah ketidakcocokan.

     Fungsi ini dapat mempertimbangkan kecocokan "tidak tepat". Artinya, jika `tepat`
     False, maka fungsinya akan menganggap label {0, 1} mungkin
```

```
     dianggap serupa hingga menukar label. Fitur ini
     memungkinkan

       a == [0, 0, 1, 1, 0, 1, 1]
       b == [1, 1, 0, 0, 1, 0, 0]

     untuk dianggap setara. (Artinya, gunakan `exact = False` saat
Anda
     hanya peduli tentang pelabelan "relatif".)
    """
    matches = mark_matches(a, b, exact=exact)
    return np.sum(matches)

n_matches = count_matches(labels, data['clusters'])
print(n_matches,
      "matches out of",
      len(data), "data points",
      "(~ {:.1f}%)".format(100.0 * n_matches / len(labels)))
```

Referensi :

Nguyen,T. 2019. K-Medoids Clustering on Iris Data Set. https://towardsdatascience.com/k-medoids-clustering-on-iris-data-set-1931bf781e05. 18 Agustus 2020.