1 JOBSHEET 7 CLUSTERING

Nama : Elis Nurhidayati

NIM : 2241720035

Kelas : TI-3C

Link Google Colab :

https://colab.research.google.com/drive/1rzUyHDtkqr33ezlmMWl26apxxkNE3M0x?usp=sharing

1.1 Praktikum 1

KMeans adalah satu metode unsupervised learning pada machine learning. Metode ini menentukan jumlah cluster sesuai dengan jumlah k yang dipilih. Pada modul jobsheet ini, kita akan langsung mempraktikkan pembuatan model KMeans dengan menggunakan python. Untuk modul pertama ini, kita akan menggunakan contoh kasus yang sederhana, yaitu dengan menggunakan dataset iris. Sedangkan untuk modul kedua, kita akan melakukan clustering dengan lebih advance, yaitu reduksi warna dengan data gambar

[1]:
```
from google.colab import drive
drive.mount('/content/drive')

path = "/content/drive/MyDrive/Elis-3C/SMT 5/ML/Jobsheet 7/"
```

```
Mounted at /content/drive
```

[18]:
```
# Persiapan data
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

df = pd.read_csv(f'{path}/dataset/Iris.csv')
df.head()
```

[18]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa

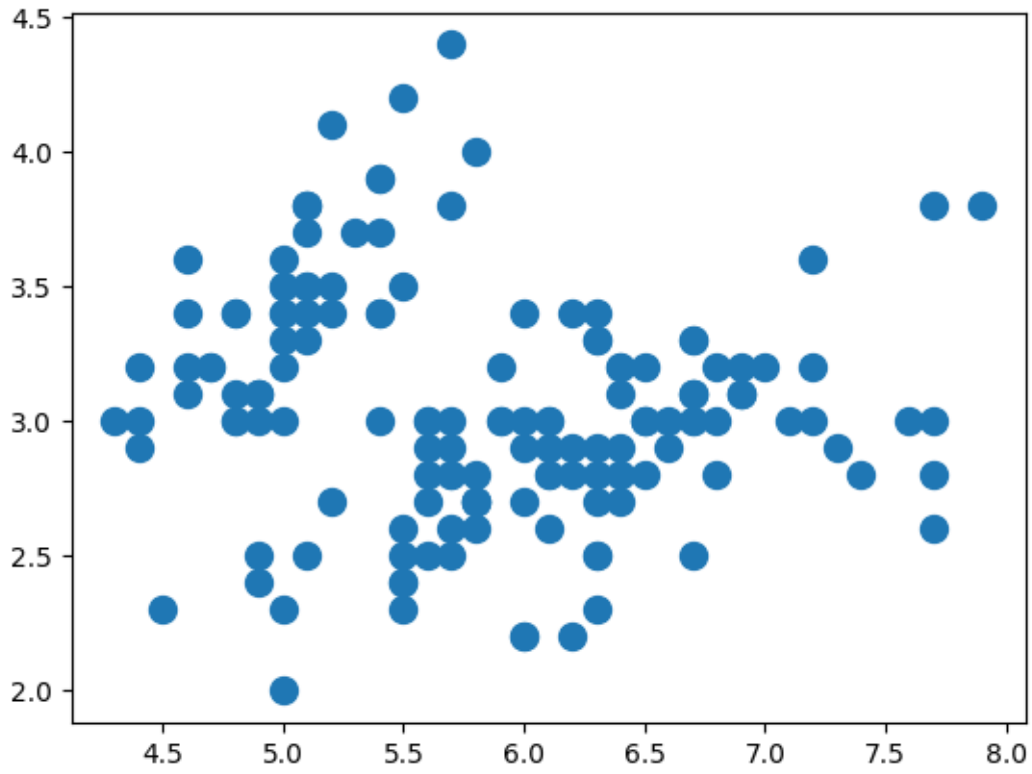
```
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

[20]:
```
# Seleksi Fitur

X = df.iloc[:, 1:-1]
y = df.iloc[:, -1]
print(X.head())
```

```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
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
```

[21]:
```
# Plot Data
# Karena data 4 dimensi, maka akan kita coba
# plot cluster berdasarkan Sepal Length dan Sepal Width  saja

plt.scatter(X.iloc[:, 0], X.iloc[:, 1], s = 100)
```

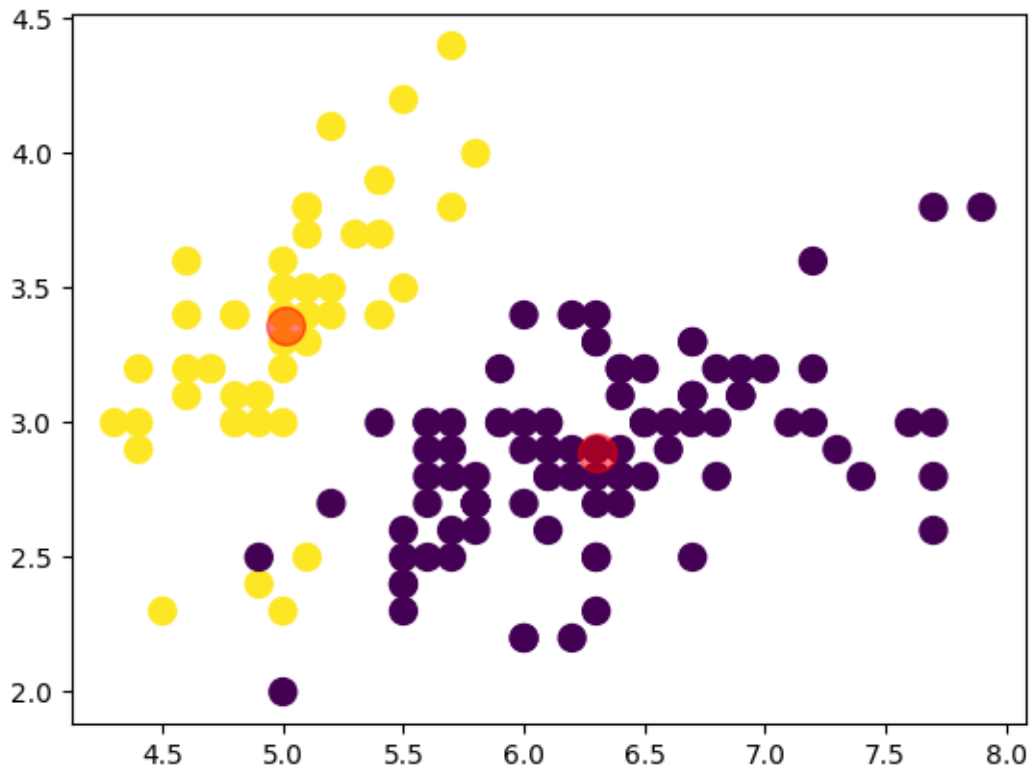
[21]:
```
<matplotlib.collections.PathCollection at 0x7a6542273bb0>
```



[22]:
```
# Buat Model KMeans
# Kali ini kita coba menggunakan k=2 - anggap saja kita tidak tahu jumlah label␣
↪ada 3 :)

from sklearn.cluster import KMeans

# Inisiasi obyek KMeans
cl_kmeans = KMeans(n_clusters=2)

# Fit dan predict model
y_kmeans = cl_kmeans.fit_predict(X)
```

[24]:
```
# Plot hasi cluster berdasarkan Sepal Length dan Sepal Width
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], s = 100, c=y_kmeans)

# Plot centroid
centers = cl_kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.5)
```

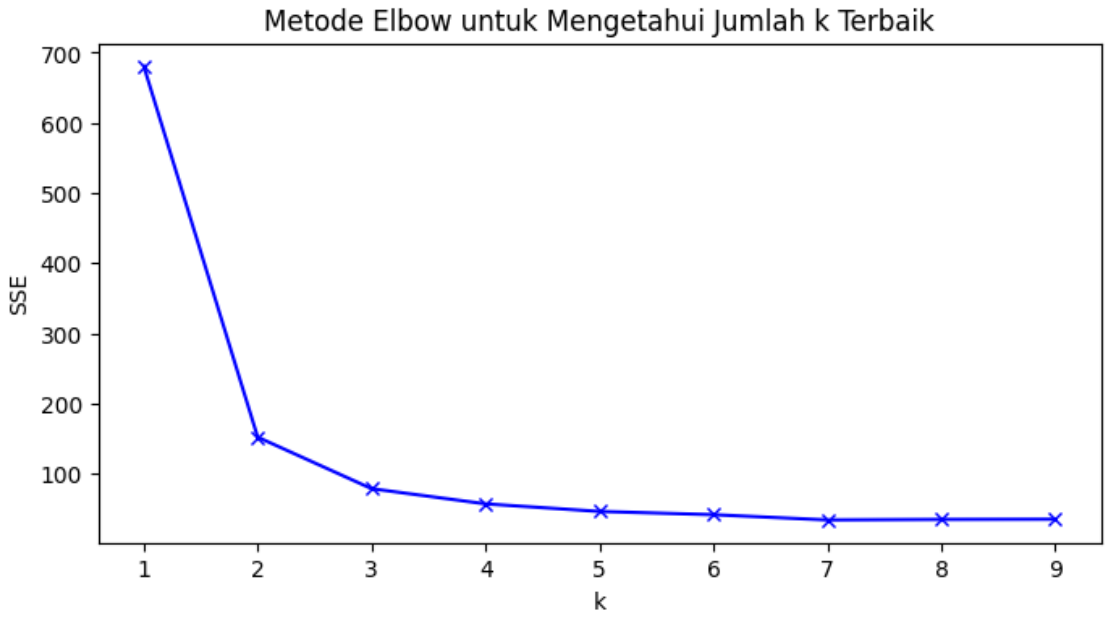
[24]: <matplotlib.collections.PathCollection at 0x7a6539714700>



[25]:
```
# Cek Nilai SSE
print(f'Nilai SSE: {cl_kmeans.inertia_}')
```

```
Nilai SSE: 152.36870647733915
```

[26]:
```
# Implementasi Metode Elbow

# List nilai SSE
sse = []

# Cari k terbaik dari 1-10
K = range(1,10)

# Cek nilai SSE setiap k
for k in K:
 kmeanModel = KMeans(n_clusters=k)
 kmeanModel.fit(X)
 sse.append(kmeanModel.inertia_)


# Plotting the distortions
plt.figure(figsize=(8,4))
plt.plot(K, sse, "bx-")
plt.xlabel("k")
plt.ylabel("SSE")
plt.title("Metode Elbow untuk Mengetahui Jumlah k Terbaik")
plt.show()
```



[28]:
```
# Cek Nilai SSE setiap k
for idx, sse_val in enumerate(sse, start=1):
    print(f'k={idx}; SSE={sse_val}')
```

```
k=1; SSE=680.8243999999996
k=2; SSE=152.36870647733915
k=3; SSE=78.94084142614601
k=4; SSE=57.317873214285726
k=5; SSE=46.55057267267267
k=6; SSE=41.91578659147871
k=7; SSE=34.42194766505635
k=8; SSE=35.167314009661844
k=9; SSE=35.50541863354038
```

2 Praktikum 2

Konsep K-Means untuk klasterisasi data

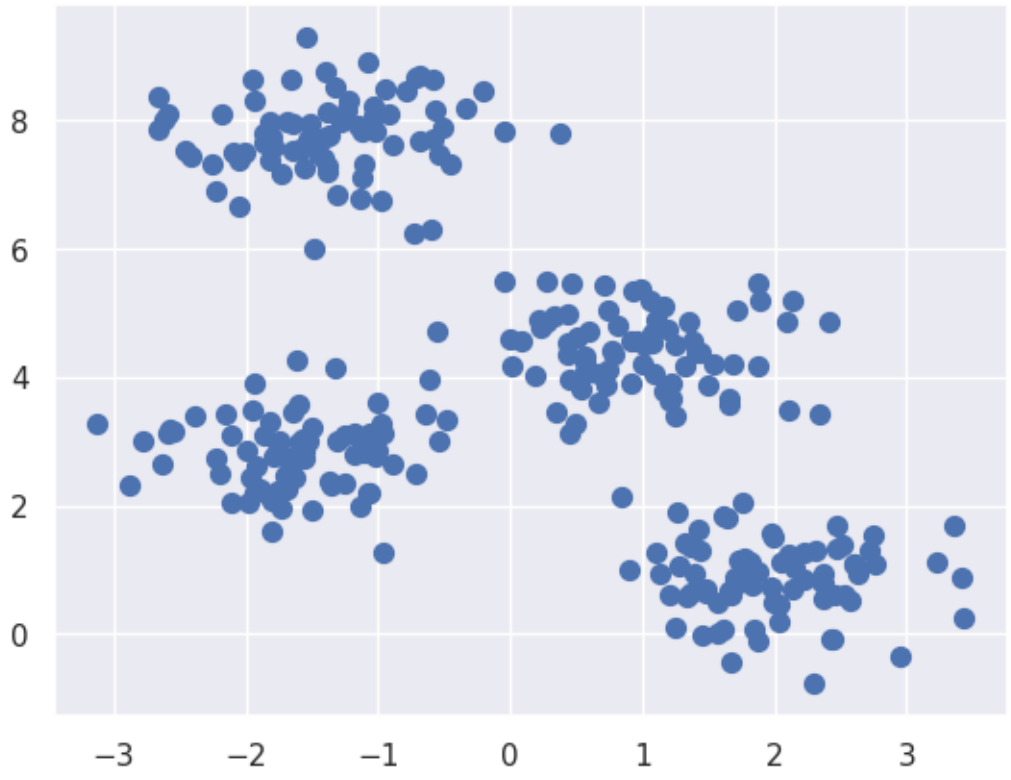
import library

[29]:
```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
```

2.1 Pengantar k-Means

[30]:
```
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50)
```

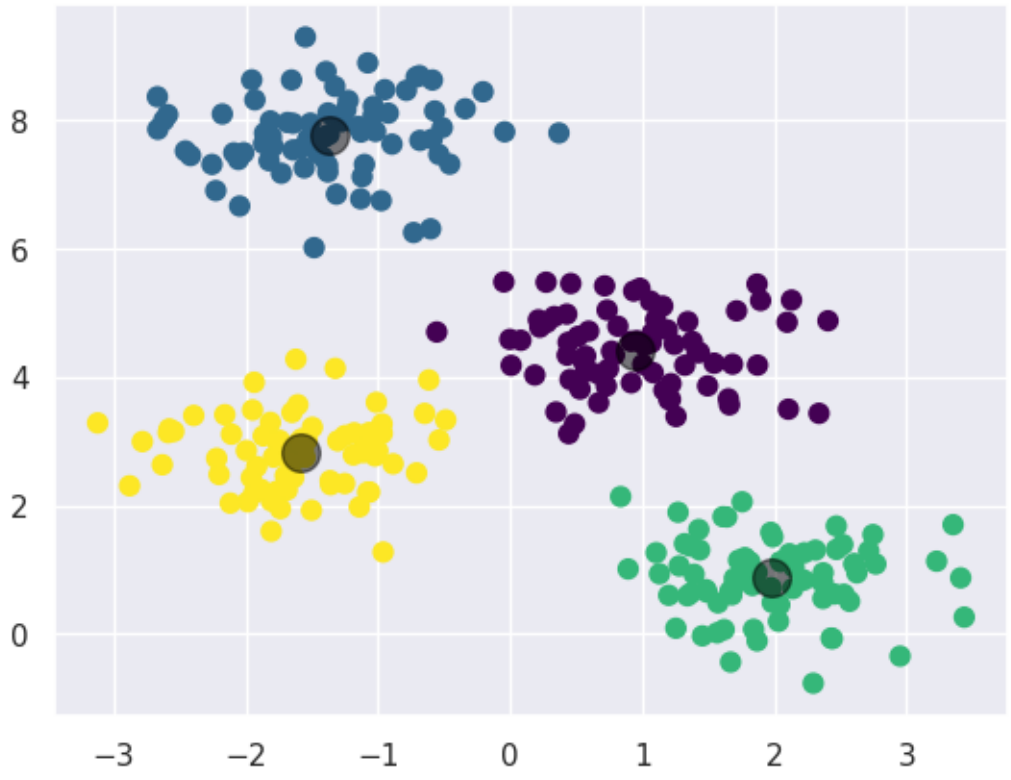
[30]:
```
<matplotlib.collections.PathCollection at 0x7a6540129390>
```



[31]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

[34]:
```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
```

[34]: <matplotlib.collections.PathCollection at 0x7a6539466920>



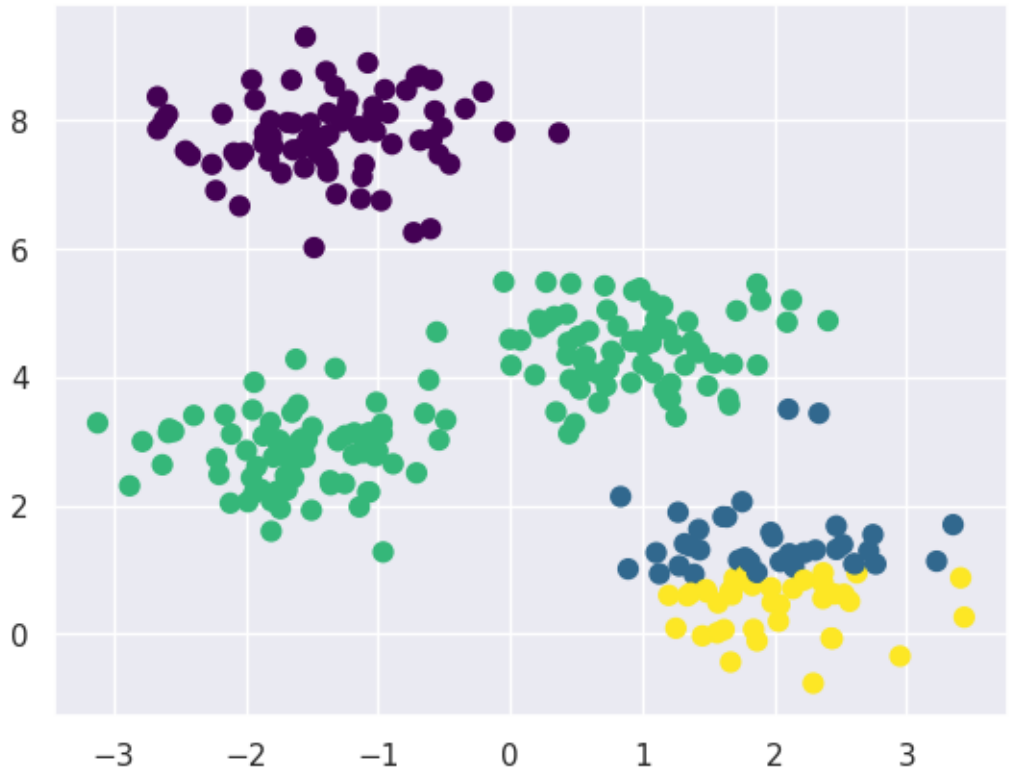
2.2 Algoritma Expectation-Maximization

[36]:
```
from sklearn.metrics import pairwise_distances_argmin

def find_clusters(X, n_clusters, rseed=2):
    # 1. Randomly choose clusters
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]

    while True:
        # 2a. input label center yang baru
        labels = pairwise_distances_argmin(X, centers)

        # 2b. update center dari titik baru
        new_centers = np.array([X[labels == i].mean(0)
                                for i in range(n_clusters)])

        # 2c. cek konvergensi
        if np.all(centers == new_centers):
            break
```

```
        centers = new_centers

    return centers, labels

centers, labels = find_clusters(X, 4)
plt.scatter(X[:, 0], X[:, 1], c=labels,s=50, cmap='viridis');
```

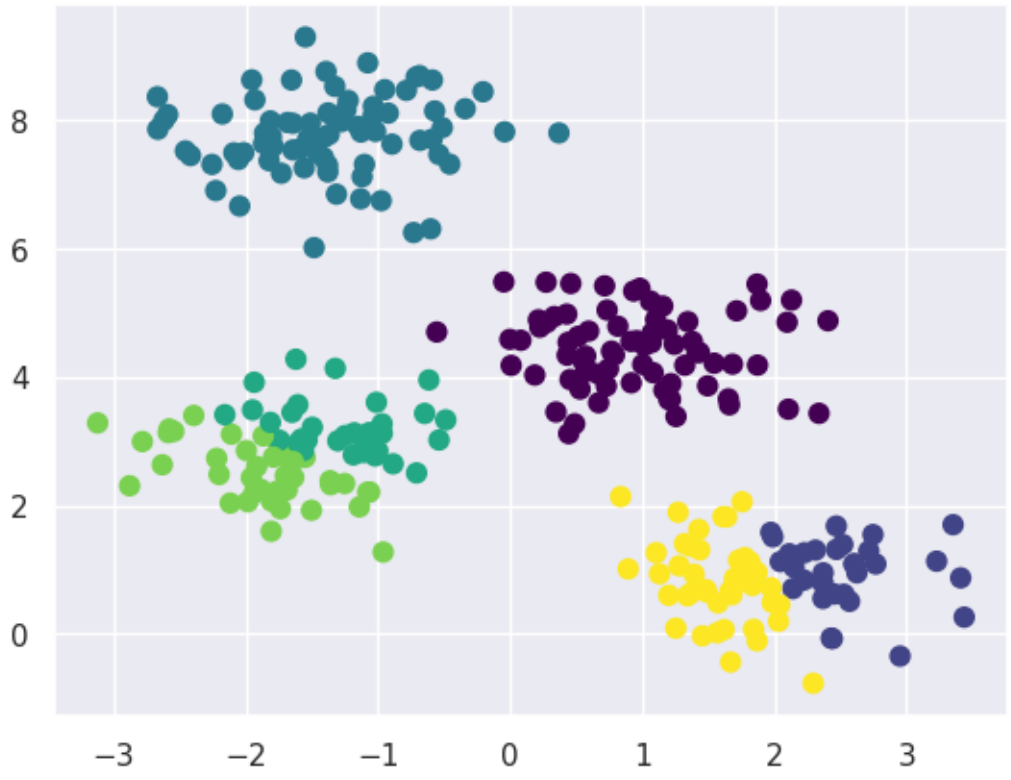
2.2.1 Perubahan random

[37]:
```
centers, labels = find_clusters(X, 4, rseed=0)
plt.scatter(X[:, 0], X[:, 1], c=labels,s=50, cmap='viridis');
```



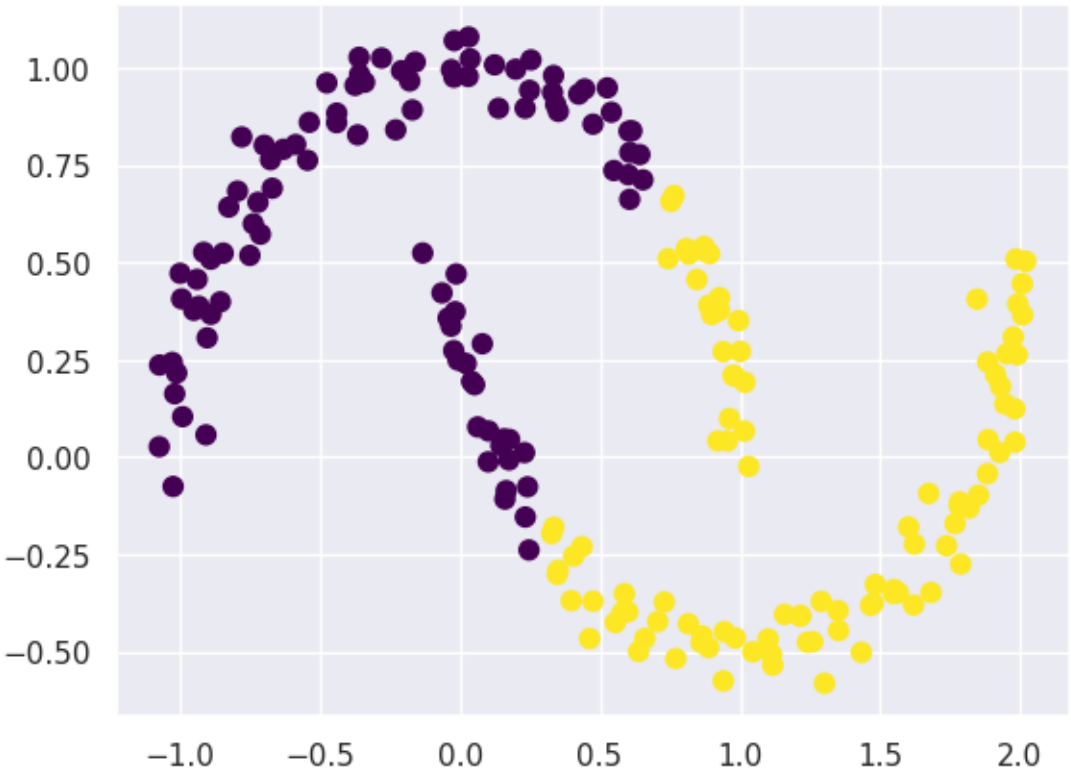
2.2.2 Optimalisasi Jumlah Klaster

[38]:
```
labels = KMeans(6, random_state=0).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis');
```



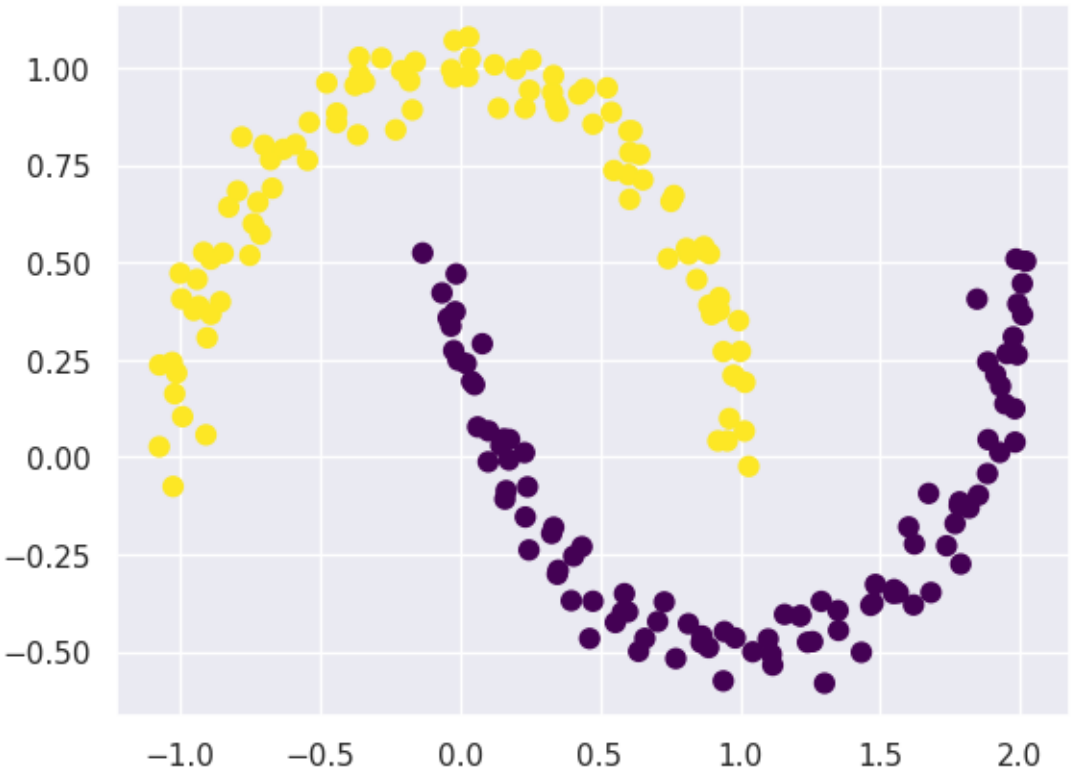
2.2.3 Batas Klaster yang Tidak Selalu Linier

[42]:
```
# K-Means Clustering
from sklearn.datasets import make_moons
X, y = make_moons(200, noise=.05, random_state=0)
labels = KMeans(2, random_state=0).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels,s=50, cmap='viridis');
```



[47]:
```
# Spectral Clustering
from sklearn.cluster import SpectralClustering

model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors',␣
 ↪assign_labels='kmeans')
labels = model.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis');
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/manifold/_spectral_embedding.py:329: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
```



2.3 Contoh Kasus 1: Karakter Angka

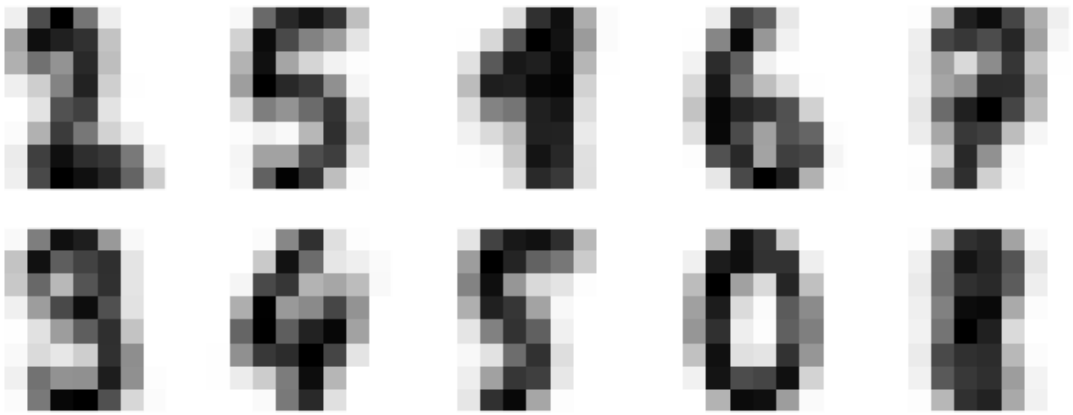
[48]:
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

[48]:
```
(1797, 64)
```

[50]:
```
# Terapkan K-Means
kmeans = KMeans(n_clusters=10, random_state=0)
clusters = kmeans.fit_predict(digits.data)
kmeans.cluster_centers_.shape
```

[50]:
```
(10, 64)
```

[53]:
```
fig, ax = plt.subplots(2, 5, figsize=(8, 3))
centers = kmeans.cluster_centers_.reshape(10, 8, 8)
for axi, center in zip(ax.flat, centers):
    axi.set(xticks=[], yticks=[])
    axi.imshow(center, interpolation='nearest', cmap=plt.cm.binary)
```



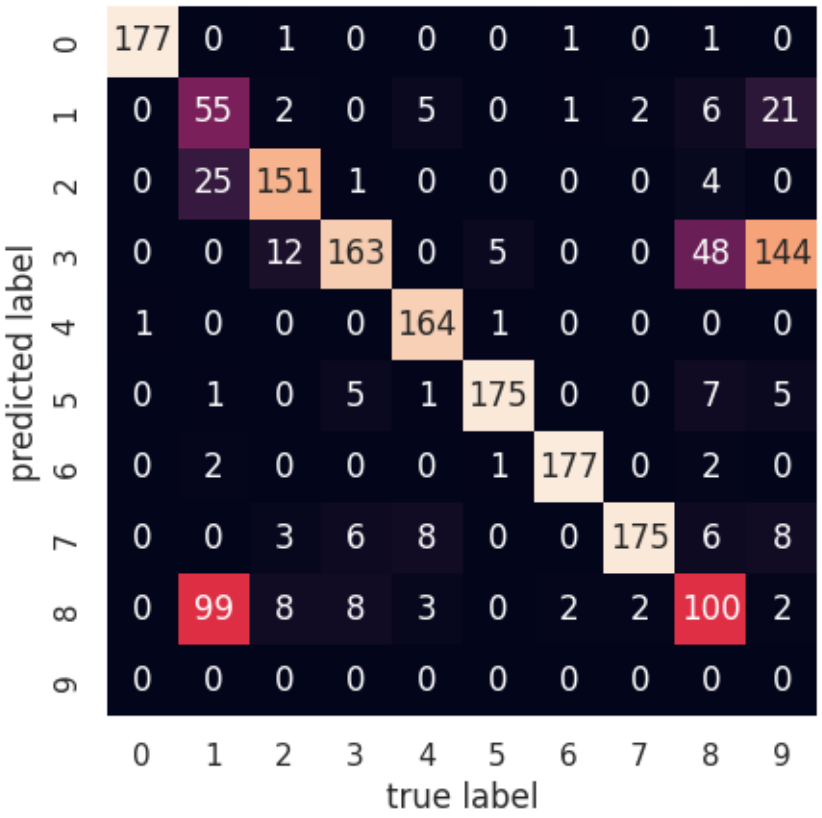
[54]:
```
from scipy.stats import mode

labels = np.zeros_like(clusters)
for i in range(10):
    mask = (clusters == i)
    labels[mask] = mode(digits.target[mask])[0]
```

[55]:
```
from sklearn.metrics import accuracy_score
accuracy_score(digits.target, labels)
```

[55]:
```
0.7440178074568725
```

[56]:
```
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(digits.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=digits.target_names,
            yticklabels=digits.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

[56]:
```
Text(109.44999999999997, 0.5, 'predicted label')
```



[57]:
```
from sklearn.manifold import TSNE


tsne = TSNE(n_components=2, init='random', random_state=0)
digits_proj = tsne.fit_transform(digits.data)

# hitung klaster
kmeans = KMeans(n_clusters=10, random_state=0)
clusters = kmeans.fit_predict(digits_proj)

# permutasi label
labels = np.zeros_like(clusters)
for i in range(10):
    mask = (clusters == i)
    labels[mask] = mode(digits.target[mask])[0]

# hitung akurasi
accuracy_score(digits.target, labels)
```

[57]: 0.9415692821368948

2.4 Studi Kasus 2: Kompresi Citra

[58]:
```
from sklearn.datasets import load_sample_image
flower = load_sample_image("flower.jpg")
ax = plt.axes(xticks=[], yticks=[])
ax.imshow(flower)
```

[58]:
```
<matplotlib.image.AxesImage at 0x7a6539224e20>
```

[59]:
```
flower.shape
```

[59]:
```
(427, 640, 3)
```

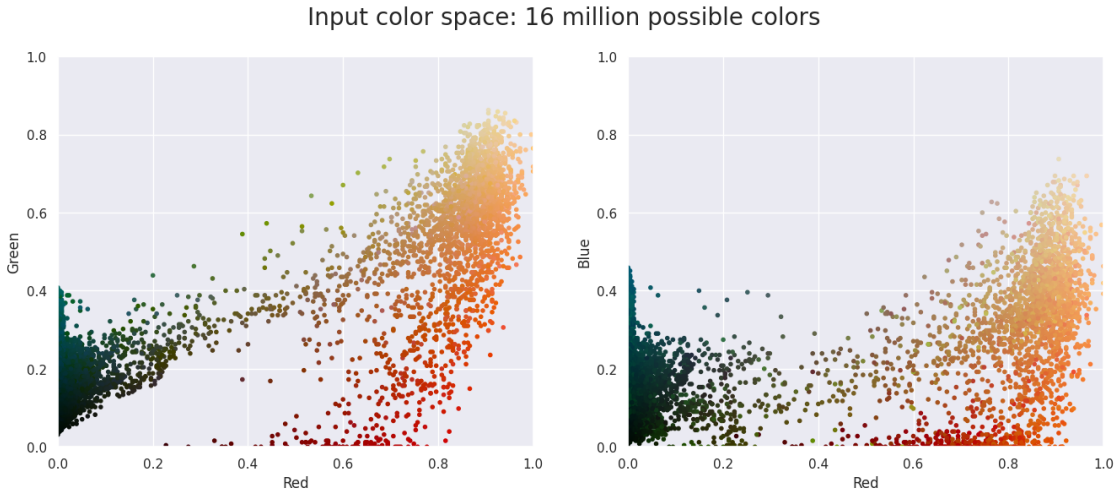
[60]:
```
data = flower / 255.0
data = data.reshape(427 * 640, 3)
data.shape
```

[60]:
```
(273280, 3)
```

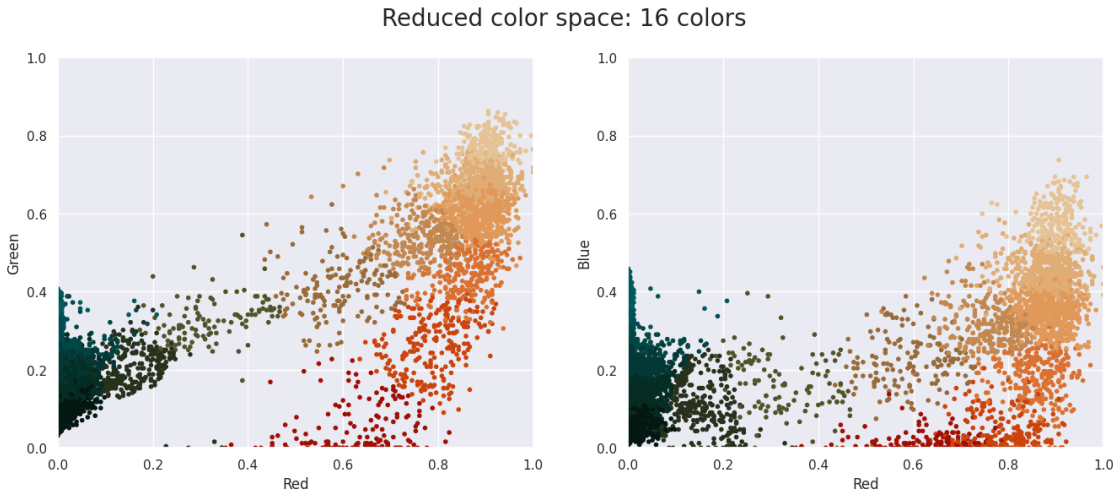
[61]:
```
def plot_pixels(data, title, colors=None, N=10000):
    if colors is None:
        colors = data

    # choose a random subset
    rng = np.random.RandomState(0)
    i = rng.permutation(data.shape[0])[:N]
```

```
    colors = colors[i]
    R, G, B = data[i].T

    fig, ax = plt.subplots(1, 2, figsize=(16, 6))
    ax[0].scatter(R, G, color=colors, marker='.')
    ax[0].set(xlabel='Red', ylabel='Green', xlim=(0, 1), ylim=(0, 1))

    ax[1].scatter(R, B, color=colors, marker='.')
    ax[1].set(xlabel='Red', ylabel='Blue', xlim=(0, 1), ylim=(0, 1))

    fig.suptitle(title, size=20)
```

[62]:
```
plot_pixels(data, title='Input color space: 16 million possible colors')
```



[63]:
```
import warnings; warnings.simplefilter('ignore')  # Fix NumPy issues.

from sklearn.cluster import MiniBatchKMeans
kmeans = MiniBatchKMeans(16)
kmeans.fit(data)
new_colors = kmeans.cluster_centers_[kmeans.predict(data)]

plot_pixels(data, colors=new_colors,title="Reduced color space: 16 colors")
```



[64]:
```
flower_recolored = new_colors.reshape(flower.shape)

fig, ax = plt.subplots(1, 2, figsize=(16, 6),
                       subplot_kw=dict(xticks=[], yticks=[]))
fig.subplots_adjust(wspace=0.05)
ax[0].imshow(flower)
ax[0].set_title('Original Image', size=16)
ax[1].imshow(flower_recolored)
ax[1].set_title('16-color Image', size=16);
```

3 Praktikum 3

3.1 Self-optimizing Map (SOM)

Contoh penerapan Self-Organizing Map (SOM) untuk klaterisasi menggunakan library SOMPY. Sebelum memulai, pastikan Anda telah menginstal library SOMPY. JIka dalam library tidak dite-

mukan SOMPY bisa menggunakan minisom. Dalam Praktikum ini akan menggunakan library MiniSom untuk implementasi SOM dengan ukuran yang lebih kecil.

[65]:
```
pip install minisom
```

```
Requirement already satisfied: minisom in /usr/local/lib/python3.10/dist-
packages (2.3.3)
```

[66]:
```
from minisom import MiniSom
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

3.1.1 Gunakan dataset Iris sebagai contoh

[67]:
```
# Load dataset
iris = datasets.load_iris()
data = iris.data
```

3.1.2 Normalisasi dataset agar nilainya berada dalam rentang yang seragam

[68]:
```
# Normalisasi data
data = data / data.max(axis=0)
```

3.1.3 Menentukan ukuran peta, misalkan (10x10) dan inisialisasi SOM

[69]:
```
# Inisialisasi SOM
map_size = (10, 10)
som = MiniSom(map_size[0], map_size[1], data.shape[1], sigma=0.5,␣
 ↪learning_rate=0.5)
```

3.1.4 Inisialisasi bobot SOM secara acak.

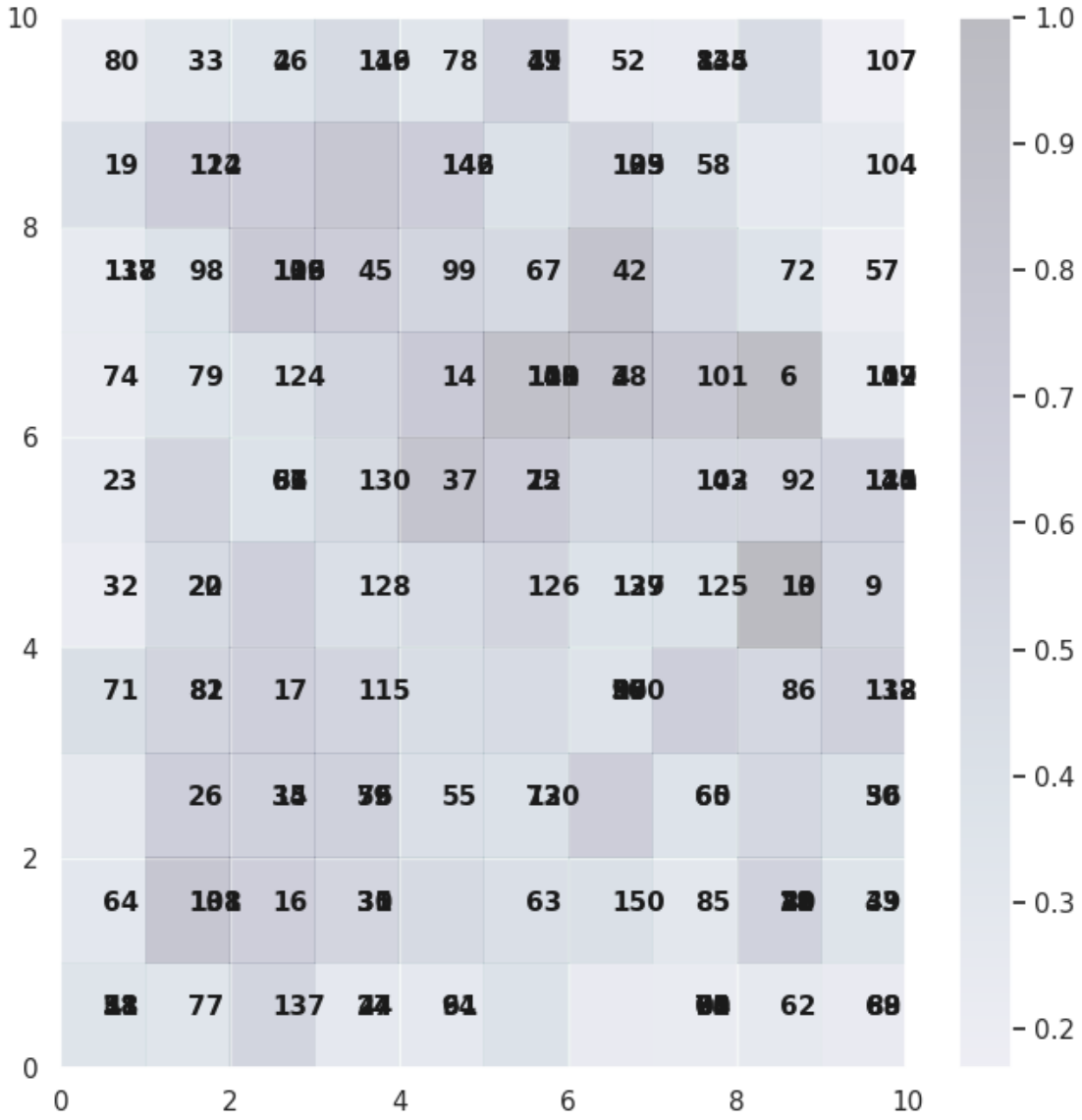
[70]:
```
# Inisialisasi bobot secara acak
som.random_weights_init(data)
```

3.1.5 Melatih SOM dengan data menggunakan metode train_random.

[71]:
```
# Pelatihan SOM
num_epochs = 100
som.train_random(data, num_epochs)
```

3.1.6 Menggunakan peta hasil pelatihan untuk visualisasi. Dalam praktikum ini, akan menampilkan peta dengan warna dan menandai pemenang untuk setiap sampel.

[72]:
```
# Visualisasi hasil SOM
plt.figure(figsize=(8, 8))
for i, x in enumerate(data):
    w = som.winner(x)  # Pemenang untuk sampel x
    plt.text(w[0]+.5, w[1]+.5, str(i+1), color='k', fontdict={'weight': 'bold',␣
 ↪'size': 11})
plt.pcolor(som.distance_map().T, cmap='bone_r', alpha=.2)
plt.colorbar()

plt.show()
```



4 Praktikum 4

4.1 Penerapan metode Self-Organizing Map (SOM) untuk segmentasi citra Lenna.

[81]:
```
pip install minisom
```

```
Requirement already satisfied: minisom in /usr/local/lib/python3.10/dist-
packages (2.3.3)
```

[74]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io
```

[75]:
```
# Fungsi untuk menginisialisasi bobot SOM
def initialize_weights(input_shape, output_shape):
    return np.random.rand(output_shape[0], output_shape[1], input_shape[2])
```

[76]:
```
# Fungsi untuk menghitung jarak antara vektor input dan bobot SOM
def calculate_distance(input_vector, weights):
    return np.linalg.norm(input_vector - weights, axis=2)
```

[77]:
```
# Fungsi untuk menemukan indeks unit pemenang (unit dengan bobot terdekat)
def find_winner_unit_in_image(input_vector, weights):
    distances = calculate_distance(input_vector, weights)
    return np.unravel_index(np.argmin(distances), distances.shape)
```

[78]:
```
# Fungsi untuk memperbarui bobot SOM
def update_weights(input_vector, weights, winner, learning_rate,␣
 ↪neighborhood_radius):
    distances = np.linalg.norm(np.indices(weights.shape[:2]).T - np.
 ↪array(winner).reshape(1, -1), axis=2)
    influence = np.exp(-distances / (2 * neighborhood_radius**2))
    weights += learning_rate * influence[:, :, np.newaxis] * (input_vector -␣
 ↪weights)
```

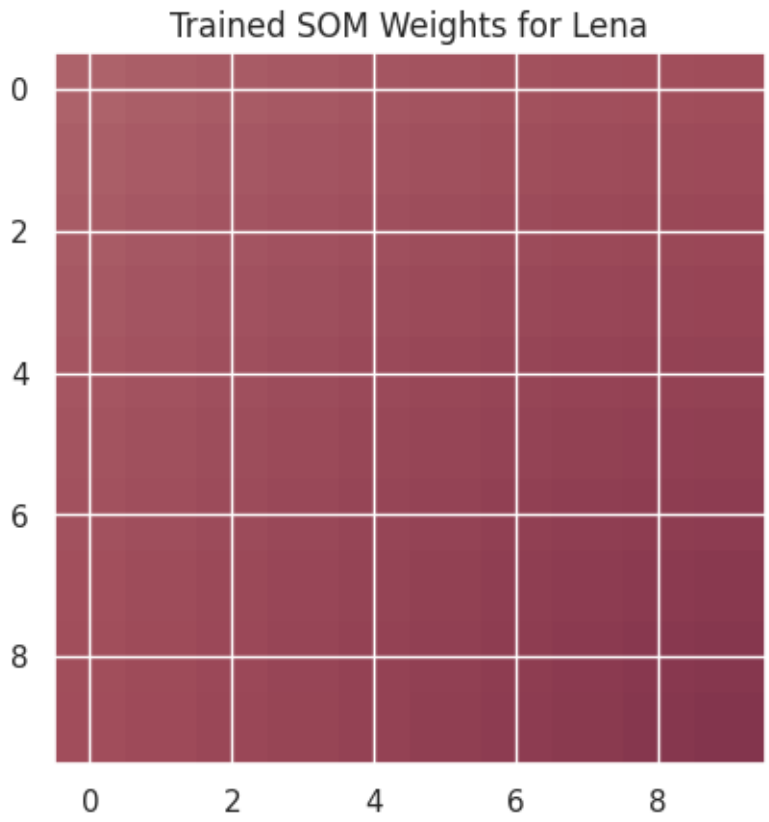
[79]:
```
# Fungsi untuk melatih SOM
def train_som(image, num_epochs, initial_learning_rate,␣
 ↪initial_neighborhood_radius):
    input_shape = image.shape
    som_shape = (10, 10, input_shape[2])  # Ukuran SOM sesuai dengan jumlah␣
 ↪saluran warna
    weights = initialize_weights(input_shape, som_shape)
```

```
    for epoch in range(num_epochs):
        # Update parameter pembelajaran dan radius tetangga
        learning_rate = initial_learning_rate * np.exp(-epoch / num_epochs)
        neighborhood_radius = initial_neighborhood_radius * np.exp(-epoch /␣
↪num_epochs)
        # Pemrosesan SOM
        for i in range(input_shape[0]):
            for j in range(input_shape[1]):
                input_vector = image[i, j, :]
                winner = find_winner_unit_in_image(input_vector, weights)
                update_weights(input_vector, weights, winner, learning_rate,␣
↪neighborhood_radius)

    return weights
```

[87]:
```
# Load citra Lenna (Anda bisa mengganti ini dengan citra lain jika diperlukan)
Lenna_path = "/content/drive/MyDrive/Elis-3C/SMT 5/ML/Jobsheet 7/dataset/Lenna.
↪png"
Lenna = io.imread(Lenna_path) / 255.0  # Normalisasi intensitas piksel menjadi␣
↪rentang [0, 1]
```

[88]:
```
# Latih SOM
num_epochs = 10
initial_learning_rate = 0.1
initial_neighborhood_radius = 5
trained_weights = train_som(Lenna, num_epochs, initial_learning_rate,␣
↪initial_neighborhood_radius)
```

[89]:
```
# Visualisasi bobot SOM
plt.imshow(trained_weights)
plt.title('Trained SOM Weights for Lena')
plt.show()
```


Trained SOM Weights for Lena
0
2
4
6
8
0
2
4
6
8