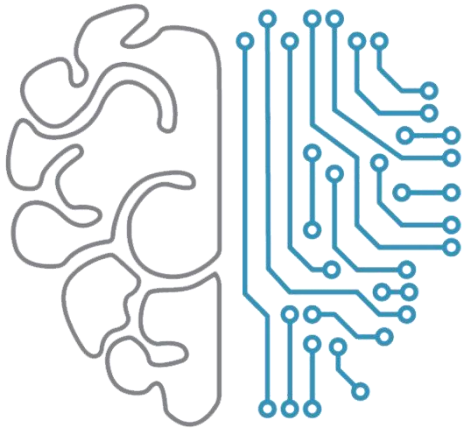
LAPORAN PRAKTIKUM

DASAR PEMROGRAMAN



NAMA : Dino Adianto Silalahi

NIM : 202331193

KELAS : D

DOSEN : DR. DRA. DWINA KUSWARDANI, M.KOM

NO.PC : 28

ASISTEN : 1. VIANA SALSABILA FAIRUZ SYAHLA

2. MUHAMMAD FADEL DESYAPUTRA

3. FAUZAN ARROYYAN

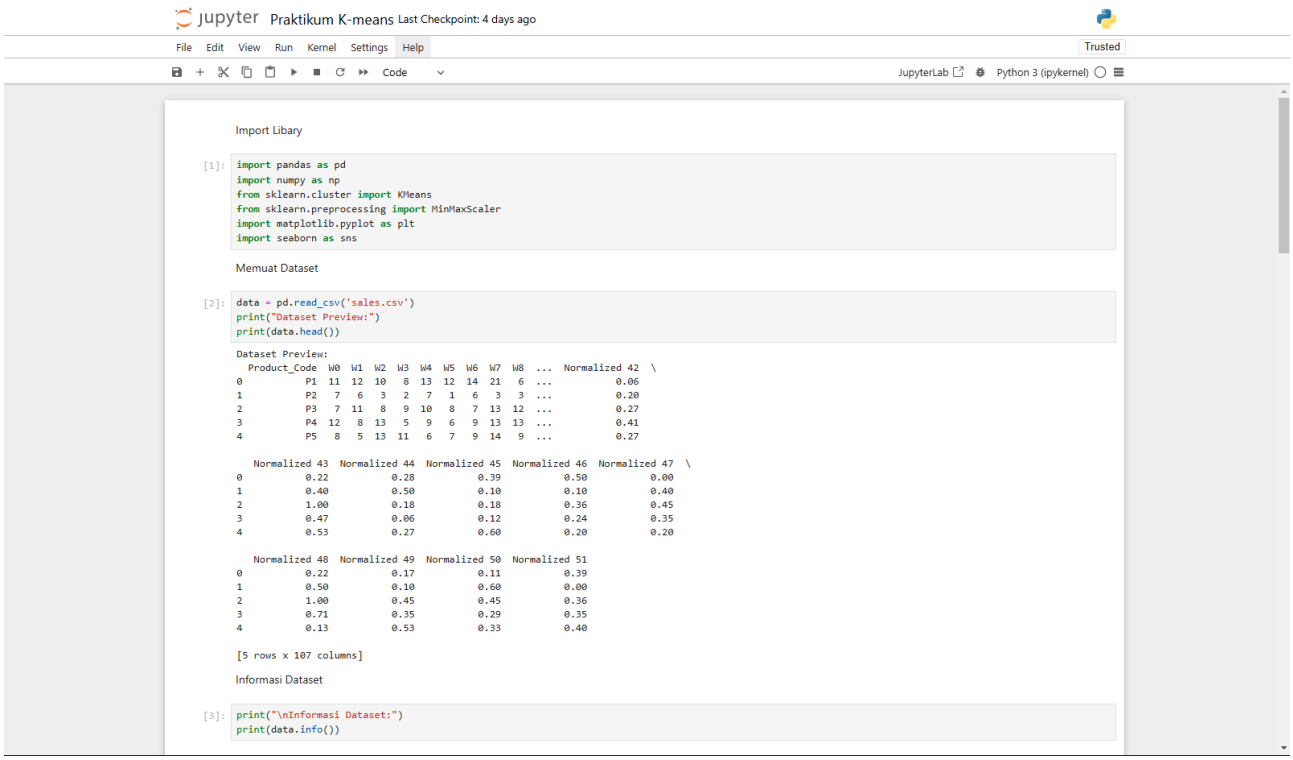
4. CLARENCA SWEETDIVA PEREIRA

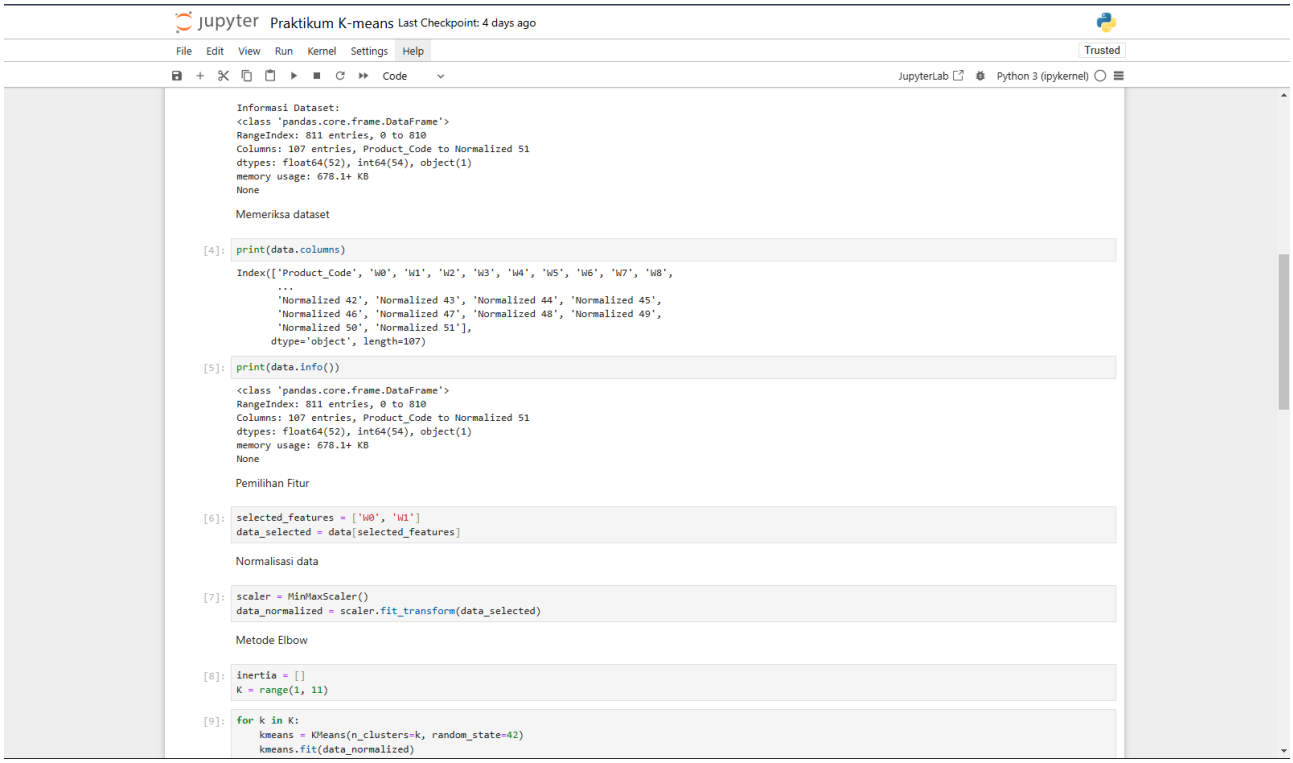
INSTITUT TEKNOLOGI PLN

TEKNIK INFORMATIKA

2024

Screenshot hasil praktkum K-Means





jupyter Praktikum K-means Last Checkpoint: 4 days ago

File Edit View Run Kernel Settings Help Trusted

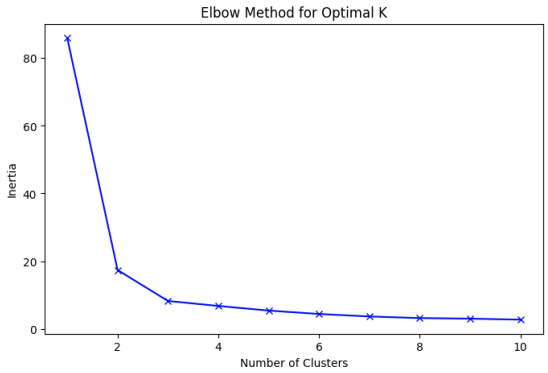
Code JupyterLab Python 3 (ipykernel)

[8]:
```
inertia = []
K = range(1, 11)
```

[9]:
```
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_normalized)
    inertia.append(kmeans.inertia_)
```

Visusalisasi metode Elbow

[10]:
```
plt.figure(figsize=(8, 5))
plt.plot(K, inertia, 'bx-')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.show()
```



Clustering Dengan K optimal

jupyter Praktikum K-means Last Checkpoint: 4 days ago

File Edit View Run Kernel Settings Help Trusted

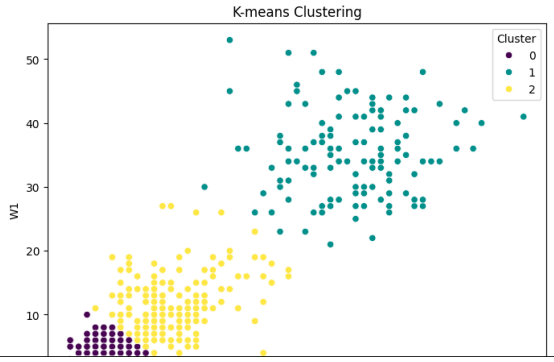
Code JupyterLab Python 3 (ipykernel)

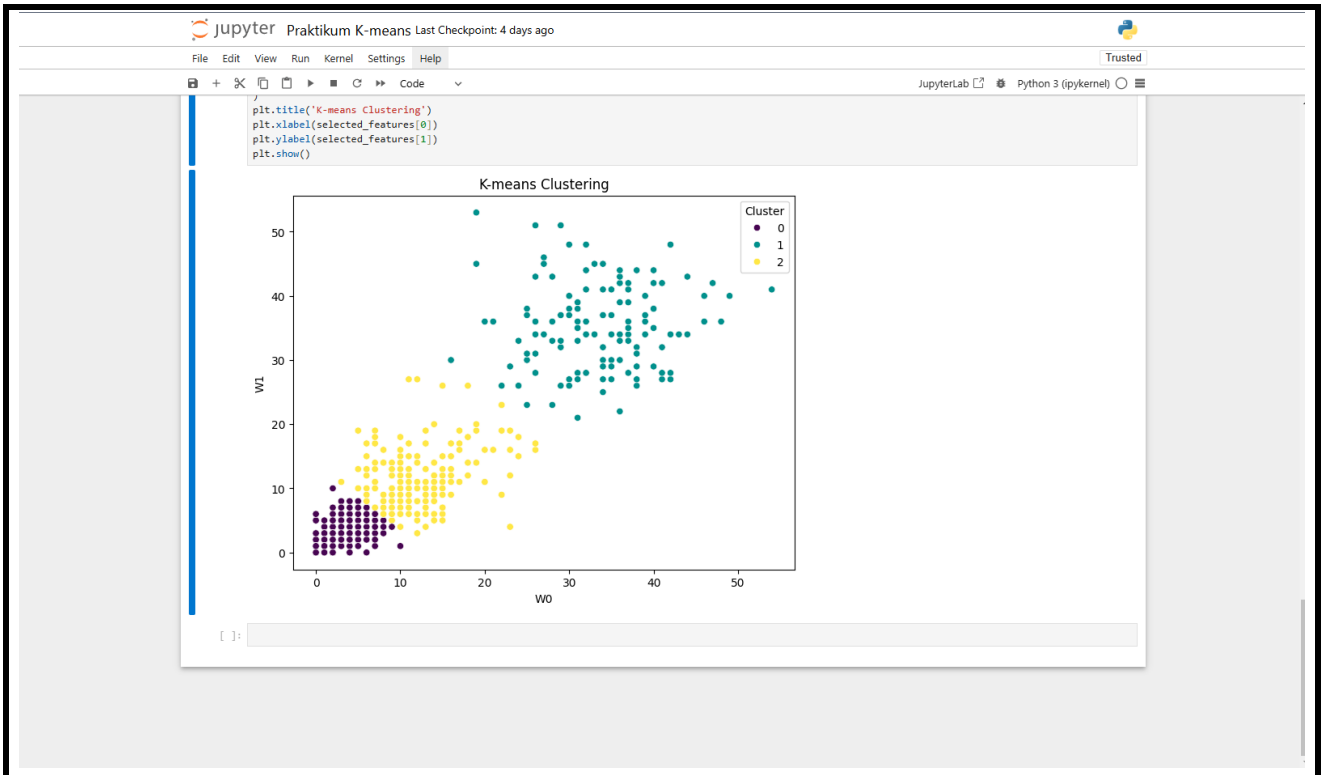
Clustering Dengan K optimal

[11]:
```
optimal_clusters = 3
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
data['Cluster'] = kmeans.fit_predict(data_normalized)
```

Visualisasi Hasil Clustering

[12]:
```
plt.figure(figsize=(8, 6))
sns.scatterplot(
    x=data_selected.iloc[:, 0],
    y=data_selected.iloc[:, 1],
    hue=data['Cluster'],
    palette='viridis'
)
plt.title('K-means Clustering')
plt.xlabel(selected_features[0])
plt.ylabel(selected_features[1])
plt.show()
```





Penjelasan

1. Import Library

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt

import seaborn as sns

- Mengimpor library yang diperlukan:
    - pandas untuk pengolahan data.
    - numpy untuk operasi numerik.
    - KMeans dari sklearn untuk algoritma clustering.
    - MinMaxScaler untuk normalisasi data.
    - matplotlib dan seaborn untuk visualisasi.

2. Memuat Dataset

data = pd.read_csv('sales.csv')

print("Dataset Preview:")

print(data.head())

- Membaca file dataset sales.csv ke dalam dataframe data.
- Menampilkan beberapa baris pertama dataset dengan data.head().

3. Informasi Dataset

print("\nInformasi Dataset:")

print(data.info())

- Menampilkan informasi tentang dataset, seperti tipe data setiap kolom dan jumlah nilai yang tidak kosong (non-null).

4. Memeriksa Kolom

print(data.columns)

- Menampilkan daftar nama kolom di dataset.

5. Pemilihan Fitur

selected_features = ['W0', 'W1']

data_selected = data[selected_features]

- Memilih dua kolom W0 dan W1 sebagai fitur yang akan digunakan dalam algoritma clustering.

6. Normalisasi Data

scaler = MinMaxScaler()

data_normalized = scaler.fit_transform(data_selected)

- Menggunakan MinMaxScaler untuk menormalisasi data sehingga nilainya berada dalam rentang 0 hingga 1. Hal ini penting untuk memastikan setiap fitur memiliki kontribusi yang seimbang dalam algoritma K-means.

7. Metode Elbow

```
inertia = []

K = range(1, 11)

for k in K:

    kmeans = KMeans(n_clusters=k, random_state=42)

    kmeans.fit(data_normalized)

    inertia.append(kmeans.inertia_)
```

- Menggunakan metode Elbow untuk menentukan jumlah cluster yang optimal.
- inertia mencatat total jarak kuadrat antara setiap titik data dan pusat cluster-nya untuk setiap nilai kk.

8. Visualisasi Metode Elbow

```
plt.figure(figsize=(8, 5))

plt.plot(K, inertia, 'bx-')

plt.xlabel('Number of Clusters')

plt.ylabel('Inertia')

plt.title('Elbow Method for Optimal K')

plt.show()
```

- Membuat grafik metode Elbow untuk mengidentifikasi jumlah cluster yang optimal berdasarkan perubahan inertia.

9. Clustering dengan K Optimal

```
optimal_clusters = 3

kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)

data['Cluster'] = kmeans.fit_predict(data_normalized)
```

- Menentukan jumlah cluster optimal (dalam contoh, k=3k = 3).
- Melakukan clustering menggunakan K-means dan menambahkan hasil cluster ke dataframe data.

10. Visualisasi Hasil Clustering

```
plt.figure(figsize=(8, 6))
sns.scatterplot(
    x=data_selected.iloc[:, 0],
    y=data_selected.iloc[:, 1],
    hue=data['Cluster'],
    palette='viridis'
)
plt.title('K-means Clustering')
plt.xlabel(selected_features[0])
plt.ylabel(selected_features[1])
plt.show()
```

- Membuat visualisasi scatter plot untuk menunjukkan hasil clustering. Data diwarnai berdasarkan cluster.

Penjelasan mengenai dataset sales.csv

Dataset **sales.csv** terdiri dari 811 baris dan 107 kolom. Berikut adalah ringkasan strukturnya:

- **Kolom-kolom**:
    - Ada satu kolom dengan tipe data objek bernama **Product_Code**, yang kemungkinan merupakan pengenal unik untuk produk.
    - Sisanya adalah kolom numerik (52 tipe float64 dan 54 tipe int64), yang kemungkinan berisi data kuantitatif terkait dengan karakteristik atau performa produk.
- **Contoh Data**: Berikut adalah beberapa kolom dari dataset:
    - **W0 – W51**: Kolom-kolom ini tampaknya merepresentasikan angka atau metrik tertentu untuk setiap produk.
    - **Normalized 0 - Normalized 51**: Kolom ini mungkin hasil normalisasi dari data mentah.

Screenshot praktikum Q-learning

jupyter Q learning Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help Not Trusted

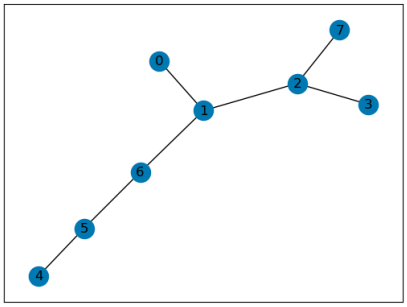
Code JupyterLab Python 3 (ipykernel)

[1]:
```
import numpy as np
import pylab as plb

#titik yang mana mengghubungan antara node cara membaca berarti (0,1) = saling berhubungan
points_list = [(0,1),(1,6),(5,6),(5,4),(1,2),(2,3),(2,7)]
```

[2]:
```
goal = 7

import networkx as nx
G = nx.Graph()
G.add_edges_from(points_list)
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos)
nx.draw_networkx_edges(G,pos)
nx.draw_networkx_labels(G,pos)
plb.show()
```



[3]:
```
#titik maksimal matriks, ini wajib didefenisikan sebagai jalur pembelajaran nantinya
MATRIX_SIZE = 8 #sesuai dengan jumlah dengan jumlah nodes nya

#membuat titik-titik awal menjadi matriks 8*8
```

jupyter Q learning Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

[3]:
```
#titik maksimal matriks, ini wajib didefenisikan sebagai jalur pembelajaran nantinya
MATRIX_SIZE = 8 #sesuai dengan jumlah dengan jumlah nodes nya

#membuat titik-titik awal menjadi matriks 8*8
```
```
R = np.matrix(np.ones(shape=(MATRIX_SIZE,MATRIX_SIZE)))
print(R)

R *= -1
print(R)
```

```
[[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. -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.]]
```

[4]:
```
for point in points_list:

    if point[1] == goal:
        R[point]=100

    else:
        R[point] = 0

    if point[0] == goal:
        R[point[::-1]] = 100
    else:
        R[point[::-1]]=0

#mendefisikan 100 dalam matrik
R[goal,goal]=100
R
```

[4]:
```
matrix([[ -1.,   0.,  -1.,  -1.,  -1.,  -1.,  -1.,  -1.],
```

jupyter Q learning Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help

Not Trusted

Code

JupyterLab Python 3 (ipykernel)

[4]:
```
matrix([[ -1.,   0.,  -1.,  -1.,  -1.,  -1.,  -1.],
        [  0.,  -1.,   0.,  -1.,  -1.,  -1.,   0.,  -1.],
        [ -1.,   0.,  -1.,   0.,  -1.,  -1.,  -1., 100.],
        [ -1.,  -1.,   0.,  -1.,  -1.,  -1.,  -1.],
        [ -1.,  -1.,  -1.,  -1.,  -1.,   0.,  -1.,  -1.],
        [ -1.,  -1.,  -1.,  -1.,   0.,  -1.,   0.,  -1.],
        [ -1.,   0.,  -1.,  -1.,  -1.,   0.,  -1.,  -1.],
        [ -1.,  -1.,   0.,  -1.,  -1.,  -1.,  -1., 100.]])
```

[5]:
```
Q = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))

# learning parameter
gamma = 0.8

initial_state = 1

#Aksi yang akan diambil dari vector
def available_actions(state):
    current_state_row = R[state,]
    av_act = np.where(current_state_row >= 0)[1]
    return av_act

available_act = available_actions(initial_state)

#Mesin memutuskan untuk mesin selanjutnya
def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_act,1))
    return next_action

action = sample_next_action(available_act)

#menyimpan hasil dari langkah yang diambil
def update(current_state, action, gamma):

    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]

    Q[current_state, action] = R[current_state, action] + gamma * max_value
    print('max_value', R[current_state, action] + gamma * max_value)

    if (np.max(Q) > 0):
```

jupyter Q learning Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help

Not Trusted

Code

JupyterLab Python 3 (ipykernel)

[5]:
```
Q = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))

# learning parameter
gamma = 0.8

initial_state = 1

#Aksi yang akan diambil dari vector
def available_actions(state):
    current_state_row = R[state,]
    av_act = np.where(current_state_row >= 0)[1]
    return av_act

available_act = available_actions(initial_state)

#Mesin memutuskan untuk mesin selanjutnya
def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_act,1))
    return next_action

action = sample_next_action(available_act)

#menyimpan hasil dari langkah yang diambil
def update(current_state, action, gamma):

    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]

    Q[current_state, action] = R[current_state, action] + gamma * max_value
    print('max_value', R[current_state, action] + gamma * max_value)

    if (np.max(Q) > 0):
        return(np.sum(Q/np.max(Q)*100))
    else:
        return (0)

update(initial_state, action, gamma)
```

```
max_value 0.0
```

[5]:
```
0
```

jupyter Q learning Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

```
[6]: # Training
scores = []
for i in range(700):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_act = available_actions(current_state)
    action = sample_next_action(available_act)
    score = update(current_state,action,gamma)
    scores.append(score)
    print ('Score:', str(score))

print("Trained Q matrix:")
print(Q/np.max(Q)*100)

# Testing
current_state = 0
steps = [current_state]

while current_state != 7:

    next_step_index = np.where(Q[current_state,] == np.max(Q[current_state,]))[1]

    if next_step_index.shape[0] > 1:
        next_step_index = int(np.random.choice(next_step_index, size = 1))
    else:
        next_step_index = int(next_step_index)

    steps.append(next_step_index)
    current_state = next_step_index

print("Most efficient path:")
print(steps)

plb.plot(scores)
plb.show()
```

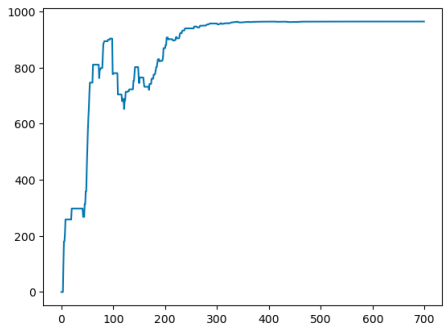
```
max_value 0.0
Score: 0
max_value 0.0
Score: 0
max_value 0.0
Score: 0
max_value 0.0
Score: 0
max_value 100.0
Score: 100.0
max_value 80.0
```

jupyter Q learning Last Checkpoint: 3 hours ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

```
max_value 255.808822063558
Score: 964.0245475163156
Trained Q matrix:
[[  0.          63.99118836   0.           0.           0.
    0.           0.           0.        ]
 [ 51.17428716   0.          79.98898546   0.           0.
    0.          51.17428716   0.        ]
 [  0.          63.96785895   0.          63.99118836   0.
    0.           0.          99.98623182]
 [  0.           0.          79.98898546   0.           0.
    0.           0.           0.        ]
 [  0.           0.           0.           0.           0.
   40.93942973   0.           0.        ]
 [  0.           0.           0.           0.          32.75154378
    0.          51.17428716   0.        ]
 [  0.          63.96785895   0.           0.           0.
   40.93942973   0.           0.        ]
 [  0.           0.          79.98898546   0.           0.
    0.           0.         100.        ]]
Most efficient path:
[0, 1, 2, 7]
```



[ ]:

Penjelasan

1. Import Library dan Definisi Titik

import numpy as np

import pylab as plb

```
# Titik-titik yang saling terhubung (0,1) berarti node 0 terhubung ke node 1
points_list = [(0,1),(1,6),(5,6),(5,4),(1,2),(2,3),(2,7)]
```

- Mengimpor numpy untuk operasi numerik dan pylab untuk visualisasi.
- Mendefinisikan daftar hubungan antar node dalam bentuk pasangan koordinat (graph).

2. Visualisasi Graph

```
goal = 7

import networkx as nx
G = nx.Graph()
G.add_edges_from(points_list)
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos)
plb.show()
```

- Menggunakan networkx untuk membangun dan memvisualisasikan graph.
- goal = 7 adalah node tujuan dalam graph.
- Visualisasi graph dilakukan dengan posisi node dihitung menggunakan spring_layout.

3. Matriks Reward Awal

```
MATRIX_SIZE = 8  # Jumlah node
R = np.matrix(np.ones(shape=(MATRIX_SIZE, MATRIX_SIZE))) * -1
print(R)
```

- Membuat matriks reward berukuran 8x8 (sesuai jumlah node), dengan nilai awal -1 untuk semua elemen.

4. Mengatur Nilai Reward

```
for point in points_list:
```

```
    if point[1] == goal:
        R[point] = 100
    else:
        R[point] = 0
    if point[0] == goal:
        R[point[::-1]] = 100
    else:
        R[point[::-1]] = 0


R[goal, goal] = 100
R
```

- Mengatur nilai reward:
    - Jika suatu node terhubung langsung ke node tujuan (goal), maka reward-nya adalah 100.
    - Hubungan lain diatur menjadi 0.
    - Node tujuan memiliki reward 100 untuk dirinya sendiri.

5. Inisialisasi Matriks Q dan Parameter

```
Q = np.matrix(np.zeros([MATRIX_SIZE, MATRIX_SIZE]))


# Parameter pembelajaran
gamma = 0.8
initial_state = 1
```

- Membuat matriks Q berukuran sama dengan matriks R, diinisialisasi dengan nilai 0.
- gamma adalah faktor diskon, memengaruhi seberapa jauh reward masa depan dipertimbangkan.
- initial_state adalah node awal.

6. Fungsi untuk Menentukan Aksi

```
def available_actions(state):
    current_state_row = R[state, ]
    av_act = np.where(current_state_row >= 0)[1]
    return av_act


def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_actions_range, 1))
    return next_action
```

- available_actions: Menentukan aksi yang tersedia dari suatu state berdasarkan matriks R.
- sample_next_action: Memilih aksi berikutnya secara acak dari daftar aksi yang tersedia.

7. Fungsi Update Q-matrix

```
def update(current_state, action, gamma):
    max_index = np.where(Q[action, ] == np.max(Q[action, ]))[1]
    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size=1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]


    Q[current_state, action] = R[current_state, action] + gamma * max_value
    if np.max(Q) > 0:
        return np.sum(Q / np.max(Q) * 100)
    else:
        return 0
```

- Fungsi ini memperbarui nilai matriks Q berdasarkan aturan Q-learning: $Q[s,a]=R[s,a]+\gamma \cdot \max_{f_0}(Q[s',a'])Q[s, a] = R[s, a] + \gamma \cdot \max(Q[s', a'])$
- Menggunakan nilai maksimal dari Q untuk state berikutnya sebagai bagian dari perhitungan.

8. Training Model

```
scores = []
for i in range(700):
    current_state = np.random.randint(0, Q.shape[0])
    available_act = available_actions(current_state)
    action = sample_next_action(available_act)
    score = update(current_state, action, gamma)
    scores.append(score)

print("Trained Q matrix:")
print(Q / np.max(Q) * 100)
```

- Model dilatih selama 700 iterasi.
- Di setiap iterasi, dipilih state dan aksi secara acak, lalu matriks Q diperbarui.
- Hasil matriks Q dilatih diubah ke skala persentase.

9. Testing dan Jalur Efisien

```
current_state = 0
steps = [current_state]

while current_state != 7:
    next_step_index = np.where(Q[current_state, ] == np.max(Q[current_state, ]))[1]
    if next_step_index.shape[0] > 1:
        next_step_index = int(np.random.choice(next_step_index, size=1))
    else:
        next_step_index = int(next_step_index)
    steps.append(next_step_index)
    current_state = next_step_index
```

print("Most efficient path:")

print(steps)

- Menentukan jalur paling efisien dari node awal ke node tujuan (7) berdasarkan matriks Q yang telah dilatih.

10. Visualisasi Skor

plb.plot(scores)

plb.show()

- Membuat grafik perubahan skor selama proses pelatihan untuk menunjukkan peningkatan kinerja.