DWM

5th DWM Aprioro algo in Python

```
data = [
        ['T100',['I1','I2','I5']],
        ['T200',['I2','I4']],
        ['T300',['I2','I3']],
        ['T400',['I1','I2','I4']],
        ['T500',['I1','I3']],
        ['T600',['I2','I3']],
        ['T700',['I1','I3']],
        ['T800',['I1','I2','I3','I5']],
        ['T900',['I1','I2','I3']]
        ]
init = []
for i in data:
    for q in i[1]:
        if(q not in init):
            init.append(q)
init = sorted(init)
print(init)


support_count = 0.4
s = int(support_count*len(init))
s


from collections import Counter
c = Counter()
for i in init:
    for d in data:
        if(i in d[1]):
            c[i]+=1
print("C1:")
for i in c:
    print(str([i])+": "+str(c[i]))
print()
l = Counter()
for i in c:
    if(c[i] >= s):
        l[frozenset([i])]+=c[i]
print("L1:")
for i in l:
    print(str(list(i))+": "+str(l[i]))
print()
pl = l
```

```
pos = 1
for count in range (2,1000):
    nc = se()
    set()
    temp = list(l)
    for i in range(0,len(temp)):
        for j in range(i+1,len(temp)):
            t = temp[i].union(temp[j])
            if(len(t) == count):
                nc.add(temp[i].union(temp[j]))
    nc = list(nc)
    c = Counter()
    for i in nc:
        c[i] = 0
        for q in data:
            temp = set(q[1])
            if(i.issubset(temp)):
                c[i]+=1
    print("C"+str(count)+":")
    for i in c:
        print(str(list(i))+": "+str(c[i]))
    print()
    l = Counter()
    for i in c:
        if(c[i] >= s):
            l[i]+=c[i]
    print("L"+str(count)+":")
    for i in l:
        print(str(list(i))+": "+str(l[i]))
    print()
    if(len(l) == 0):
        break
    pl = l
    pos = count
print("Result: ")
print("L"+str(pos)+":")
for i in pl:
    print(str(list(i))+": "+str(pl[i]))
print()
```

6th Data discretization and visualization

```
import pandas as pd
import matplotlib.pyplot as plt

# Create a sample dataset
```

```
data = {'value': [15, 28, 32, 40, 52, 58, 60, 70, 75, 80, 90, 100]}
df = pd.DataFrame(data)

# Define bin edges
bin_edges = [0, 30, 60, 100]

# Discretize the data using pandas.cut()
df['bin'] = pd.cut(df['value'], bins=bin_edges, labels=['Low', 'Medium', 'High

# Print the bins
print("Bins:")
print(pd.cut(df['value'], bins=bin_edges))

# Plot histogram
plt.figure(figsize=(8, 6))
plt.hist(df['value'], bins=bin_edges, color='blue', alpha=0.7, edgecolor='blac
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of Value')
plt.xticks(bin_edges)
plt.show()

# Plot scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(df['value'], df['bin'], color='green', alpha=0.7)
plt.xlabel('Value')
plt.ylabel('Bin')
plt.title('Scatter Plot of Value vs. Bin')
plt.yticks(df['bin'].unique())
plt.show()
```

7th(Weka Tools)

There's nothing for weka tools as all the parts are done by the software like classification clustering discretization and also the visualzation too

8th (Bayesian algo)

```
data = [
    {"index": 0, "outlook": "Sunny", "play": "Yes"},
    {"index": 1, "outlook": "Rainy", "play": "No"},
    {"index": 2, "outlook": "Overcast", "play": "Yes"},
    {"index": 3, "outlook": "Sunny", "play": "No"},
    {"index": 4, "outlook": "Rainy", "play": "No"},
    {"index": 5, "outlook": "Sunny", "play": "Yes"},
    {"index": 6, "outlook": "Overcast", "play": "Yes"},
    {"index": 7, "outlook": "Rainy", "play": "Yes"},
    {"index": 8, "outlook": "Sunny", "play": "No"},
    {"index": 9, "outlook": "Sunny", "play": "Yes"},
    {"index": 10, "outlook": "Rainy", "play": "Yes"},
    {"index": 11, "outlook": "Overcast", "play": "Yes"},
    {"index": 12, "outlook": "Sunny", "play": "No"},
    {"index": 13, "outlook": "Overcast", "play": "No"}
]

play_yes_count = len([item for item in data if item["play"] == "Yes"])
play_no_count = len(data) - play_yes_count

outlook_sunny = len([item for item in data if item["outlook"] == "Sunny"])
sunny_yes = len([item for item in data if item["outlook"] == "Sunny" and item[
sunny_no = outlook_sunny - sunny_yes

outlook_rainy = len([item for item in data if item["outlook"] == "Rainy"])
rainy_yes = len([item for item in data if item["outlook"] == "Rainy" and item[
rainy_no = outlook_rainy - rainy_yes

outlook_overcast = len([item for item in data if item["outlook"] == "Overcast"
overcast_yes = len([item for item in data if item["outlook"] == "Overcast" and
overcast_no = outlook_overcast - overcast_yes

c1 = play_yes_count / len(data)
c2 = play_no_count / len(data)

x1 = sunny_yes / play_yes_count
x2 = sunny_no / play_no_count

f1 = c1 * x1
f2 = c2 * x2

if f1 > f2:
    print("PLAY = YES")
else:
    print("PLAY = NO")
```

9th (K means Clustering)

```
# Importing necessary libraries
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Generating random data points for demonstration
np.random.seed(0)
X = np.random.rand(100, 2)  # 100 data points with 2 features each

# Initializing KMeans with the number of clusters you want to form
kmeans = KMeans(n_clusters=3)

# Fitting the data to the KMeans algorithm
kmeans.fit(X)

# Getting the centroids and labels based on the trained KMeans algorithm
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

# Visualizing the data points and centroids
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', s=200, linewidths=3,
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('KMeans Clustering')
plt.show()
```

10th(Agglomerative Clustering)

```
import numpy as np
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

plt.scatter(x, y)
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
```

```
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))

hierarchical_cluster = AgglomerativeClustering(n_clusters=2, affinity='euclide
labels = hierarchical_cluster.fit_predict(data)

plt.scatter(x, y, c=labels)
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

x = [18,22,25,27,42,43]

y = [18,22,25,27,42,43]

data = list(zip(x, y))

x_labels = [str(point) for point in data]

linkage_data = linkage(data, method='single' , metric='euclidean')
dendrogram(linkage_data, labels =x_labels)

plt.show()

import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

x = [18,22,25,27,42,43]

y = [18,22,25,27,42,43]

data = list(zip(x, y))

x_labels = [str(point) for point in data]

linkage_data = linkage(data, method='complete' , metric='euclidean')
dendrogram(linkage_data, labels =x_labels)

plt.show()
```

11th(Pagerank algo)

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.DiGraph()

G.add_edges_from([('A', 'D'), ('B', 'C'), ('B', 'E'), ('C', 'A'),
                                ('D', 'C'), ('E', 'D'), ('E', 'B'), ('E', 'F')
                                ('E', 'C'), ('F', 'C'), ('F', 'H'), ('G', 'A')
                                ('G', 'C'), ('H', 'A')])

plt.figure(figsize =(10, 10))
nx.draw_networkx(G, with_labels = True)

hubs, authorities = nx.hits(G, max_iter = 50, normalized = True)
# The in-built hits function returns two dictionaries keyed by nodes
# containing hub scores and authority scores respectively.

print("Hub Scores: ", hubs)
print("Authority Scores: ", authorities)

def calculate_hits(adjacency_matrix, max_iterations=50, tolerance=0.0001):
    num_pages = len(adjacency_matrix)
    hubs = [1] * num_pages
    authorities = [1] * num_pages

    for _ in range(max_iterations):
        new_authorities = [0] * num_pages
        new_hubs = [0] * num_pages
        norm = 0

        # Update authorities
        for i in range(num_pages):
            for j in range(num_pages):
                if adjacency_matrix[j][i] == 1:
                    new_authorities[i] += hubs[j]
            norm += new_authorities[i] ** 2

        norm = norm ** 0.5

        # Normalize authorities
        for i in range(num_pages):
            new_authorities[i] /= norm

        # Update hubs
        for i in range(num_pages):
            for j in range(num_pages):
                if adjacency_matrix[i][j] == 1:
                    new_hubs[i] += new_authorities[j]
            norm += new_hubs[i] ** 2
        norm = norm ** 0.5

        # Normalize hubs
        for i in range(num_pages):
            new_hubs[i] /= norm

        # Check for convergence
        authority_diff = sum(abs(new_authorities[i] - authorities[i]) for i in
        hub_diff = sum(abs(new_hubs[i] - hubs[i]) for i in range(num_pages))

        if authority_diff < tolerance and hub_diff < tolerance:
            authorities = new_authorities
            hubs = new_hubs
            break

        authorities = new_authorities
        hubs = new_hubs

    return hubs, authorities

# Sample web graph represented as an adjacency matrix
adjacency_matrix = [
    [0, 0, 1, 1, 0, 0, 0, 0],
    [0, 0, 1, 0, 1, 1, 0, 0],
    [1, 0, 0, 1, 1, 1, 1, 0],
    [0, 0, 1, 0, 0, 0, 1, 0],
    [0, 0, 0, 1, 0, 0, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0]
]

# Example usage
hubs, authorities = calculate_hits(adjacency_matrix, max_iterations=50)
print("Hub Scores:", hubs)
print("Authority Scores:", authorities)


# Hub Scores:  [
#   0.24843673779840283,
#   0.33551684797364123,
#   0.6303066764793214,
#   0.24843673779840283,
#   0.33551684797364123,
#   0,
#   0.10104182354556279,
#   0.10104182354556279
# ]


# Authority Scores:  [
#   0.3312982358905198,
#   0,
#   0.3312982358905198,
#   0.4832818070546864,
#   0.3844010293340567,
#   0.3844010293340567,
#   0.4832818070546864,
#   0.1335366804492604
# ]
```