s1290018_Exercise13

July 31, 2023

1 Mining Frequent Patterns from Air Pollution Data

1.0.1 Task1 - (b

[2]:

```
from PAMI.extras.DF2DB import denseDF2DB as pro
import pandas as pd

# Load the dataset
df = pd.read_csv('apdata.csv')

# Replace NaN values with zero
df = df.fillna(0)

# delete timestamp column
df = df.drop(columns = 'timestamp')

# Convert DataFrame to float data type
df = df.astype(float)

# Replace values greater than or equal to 100 with zero
df[df >= 100] = 0

# Objective: convert the above dataframe into a transactional database with␣
 ↪items whose value is greater than or equal 15
db = pro.denseDF2DB(inputDF=df, thresholdValue=15, condition='>=')

# Convert and store the dataframe as a transactional database file
db.createTransactional(outputFile='PM24HeavyPollutionRecordingSensors.csv')

# Getting the fileName of the transactional database
print('The output file is saved at ' + db.getFileName())
```

```
The output file is saved at PM24HeavyPollutionRecordingSensors.csv
```

1.0.2 Task1 - (c

[1]:

```python
#import the frequent pattern mining algorithm
from PAMI.frequentPattern.basic import FPGrowth as alg

#inputFile = 'fileName'
inputFile = 'PM24HeavyPollutionRecordingSensors.csv'

#specify the constraints used in the model
minSup=200

#create the object of the mining algorithm
obj = alg.FPGrowth(inputFile, minSup)

#start the mining process
obj.startMine()

#Print the number of interesting patterns generated
print("Total number of Frequent Patterns:", len(obj.getPatterns()))

#Save the generated patterns in a file
obj.save('frequentPatterns.txt')

# Determine the memory consumed by the mining algorithm
print("Total Memory in RSS", obj.getMemoryRSS())

# Determine the total runtime consumed by the mining algorithm
print("Total ExecutionTime in seconds:", obj.getRuntime())
```

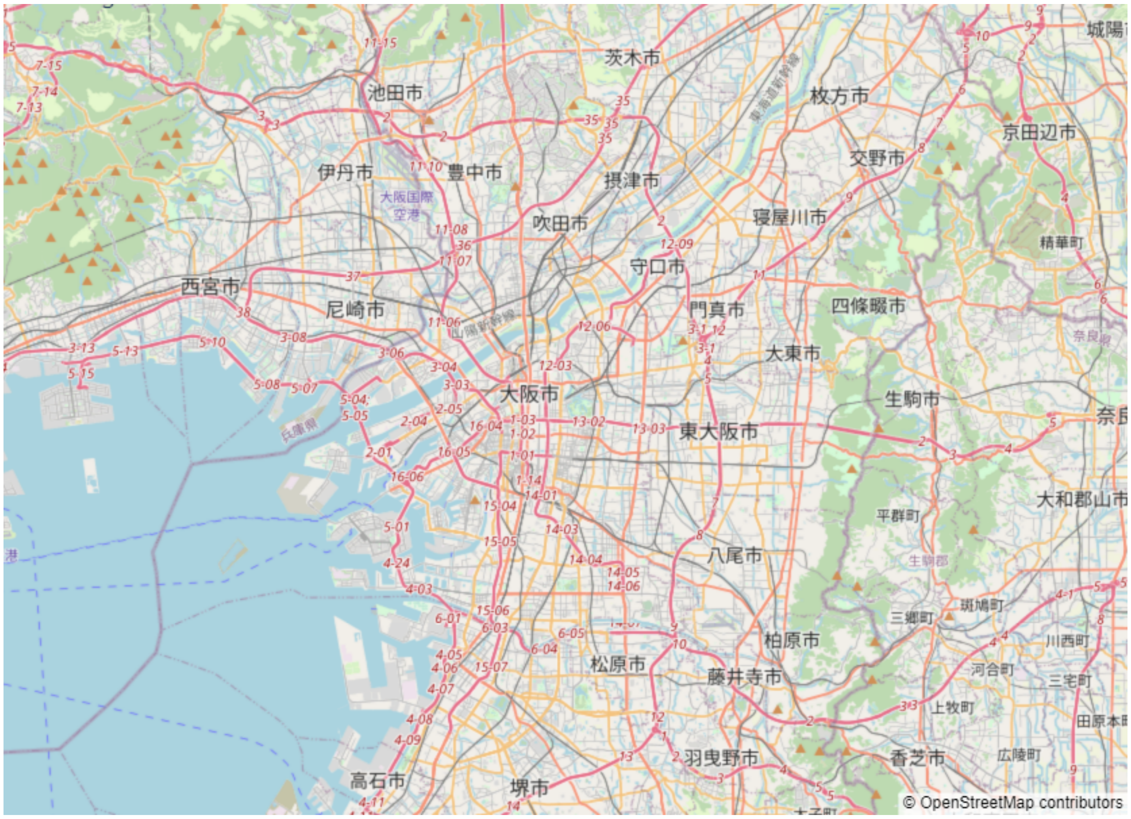
```
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
Total number of Frequent Patterns: 441
Total Memory in RSS 167165952
Total ExecutionTime in seconds: 0.5712528228759766
```

1.0.3 Task1 - (d

[4]:

```python
import plotly.express as px
import pandas as pd

# Read the 'frequentPatterns.txt' file and extract point coordinates and occurrence counts
frequent_patterns_file = 'frequentPatterns.txt'
with open(frequent_patterns_file, 'r') as f:
    lines = f.readlines()

data = []
```

```
for line in lines:
    points_str, count_str = line.strip().split(':')
    count = int(count_str)
    points = points_str.split('\t')

    for point in points:
        lon, lat = point.replace('POINT(', '').replace(')', '').split()
        data.append((float(lon), float(lat), count))

# Create a DataFrame from the extracted data
df = pd.DataFrame(data, columns=['longitude', 'latitude', 'occurrence_count'])

# Find the longest pattern by sorting the DataFrame by 'occurrence_count' in␣
 ↪descending order
longest_pattern = df.sort_values(by='occurrence_count', ascending=False).head(1)

# Create the Open Street Map visualization using Plotly Express
fig = px.scatter_mapbox(
    longest_pattern,
    lat='latitude',
    lon='longitude',
    size='occurrence_count',  # Size of the points based on occurrence count
    hover_name='occurrence_count',  # Display occurrence count on hover
    center={'lat': 34.686567, 'lon': 135.52000},
    zoom=10,
    height=600,
    width=800
)

fig.update_layout(mapbox_style='open-street-map')
fig.update_layout(margin={"r": 0, "t": 0, "l": 0, "b": 0})
fig.update_layout(title_text="Longest Pattern")

# Set the Mapbox token (you need to replace 'your_mapbox_token' with your␣
 ↪actual token)
fig.update_layout(
    mapbox=dict(
        accesstoken='your_mapbox_token',
    )
)

# Show the interactive map
fig.show()
```



© OpenStreetMap contributors

[ ]: