FLFlavors_CodingAssignment

June 25, 2024

1 Assignment "FL Main Flavors"

1.1 1. Preparation

1.1.1 1.1 Libraries

[ ]:
```
import numpy as np
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import mean_squared_error

# We will use networx objects to store empircial graphs, local datasets and␣
↪models
import networkx as nx
from sklearn.neighbors import kneighbors_graph
from numpy import linalg as LA
```

1.1.2 1.2 Helper functions

[ ]:
```
# The function generates a scatter plot of nodes (=FMI stations) using
# latitude and longitude as coordinates.
def plotFMI(G_FMI):
    num_stations = len(G_FMI.nodes)
    colors = np.array(['black', 'green', 'red', 'brown', 'deeppink',
                       'blue', 'olive', 'gray', 'orange', 'purple'])
    coords = [G_FMI.nodes[i]['coord'] for i in range(num_stations)]
    df_coords = pd.DataFrame(coords,columns=['latitude','longitude'])
    coords = np.hstack((df_coords["latitude"].to_numpy().
↪reshape(-1,1),df_coords["longitude"].to_numpy().reshape(-1,1)))
    # Create a plot
    fig, ax = plt.subplots()
    # Draw nodes
    for node in G_FMI.nodes:
        color = colors[G_FMI.nodes[node]['cluster']]
```

```
        ax.scatter(coords[node,1], coords[node,0], color=color, s=4, zorder=5) # zorder ensures nodes are on top of edges
        ax.text(coords[node,1]+0.1, coords[node,0]+0.2, str(node), fontsize=8, ha='center', va='center', color=color, fontweight='bold')
    # Draw edges
    for edge in G_FMI.edges:
        ax.plot([coords[edge[0],1],coords[edge[1],1]], [coords[edge[0],0],coords[edge[1],0]], linestyle='-', color='gray', alpha=0.5)

    ax.set_xlabel('longitude')
    ax.set_ylabel('latitude')
    ax.set_title('FMI stations')
    plt.show()


# The function connects each FMI station with
# the nearest neighbours.
def add_edges(graph, numneighbors=4):
    coords = [graph.nodes[i]['coord'] for i in range(num_stations)]
    df_coords = pd.DataFrame(coords,columns=['latitude','longitude'])
    coords = np.hstack((df_coords["latitude"].to_numpy().reshape(-1,1),df_coords["longitude"].to_numpy().reshape(-1,1)))
    A = kneighbors_graph(coords, numneighbors, mode='connectivity', include_self=False)
    nrnodes = len(graph.nodes)
    for iter_i in range(nrnodes):
        for iter_ii in range(nrnodes):
            if iter_i != iter_ii :
                if A[iter_i,iter_ii]> 0 :
                    graph.add_edge(iter_i, iter_ii)
    return graph

# The function below extracts a feature and label from each row
# of dataframe df. Each row is expected to hold a FMI weather
# measurement with cols "Latitude", "Longitude", "temp", "Timestamp"
# returns numpy arrays X, y.
def ExtractFeaureMatrixLabvelVector(data):
    nrfeatures = 7
    nrdatapoints = len(data)
    X = np.zeros((nrdatapoints, nrfeatures))
    y = np.zeros((nrdatapoints, 1))

    # Iterate over all rows in dataframe and create corresponding feature vector and label
    for ind in range(nrdatapoints):
```

```
        # latitude of FMI station, normalized by 100
        lat = float(data['Latitude'].iloc[ind])/100
        # longitude of FMI station, normalized by 100
        lon = float(data['Longitude'].iloc[ind])/100
        # temperature value of the data point
        tmp = data['temp'].iloc[ind]
        # read the date and time of the temperature measurement
        date_object = datetime.strptime(data['Timestamp'].iloc[ind], '%Y-%m-%d␣
↪%H:%M:%S')
        # Extract year, month, day, hour, and minute. Normalize these values
        # to ensure that the features are in range [0,1].
        year = float(date_object.year)/2025
        month = float(date_object.month)/13
        day = float(date_object.day)/32
        hour = float(date_object.hour)/25
        minute = float(date_object.minute)/61
        X[ind,:] = [lat, lon, year, month, day, hour, minute]
        y[ind,:] = tmp

    return X, y
```

1.2 2 Data

1.2.1 2.1 Dataset

[ ]:
```
# Import the weather measurements.
data = pd.read_csv('Assignment_MLBasicsData.csv')

# We consider each temperature measurement (=a row in dataframe) as a
# separate data point.
# Get the numbers of data points and the unique stations.
num_stations = len(data.name.unique())
num_datapoints = len(data)
```

1.2.2 2.2 Empirical graph

[ ]:
```
##################TODO###################
# TODO:
# 1. Construct the empirical graph G_FMI as a networkx.Graph() object.
# 2. Add a single node for each station.
# 3. Each node i must have the following attributes:
#   'samplesize' - the number of measurements of the i-th weather station,
#   'name' - the name of the i-th weather station,
#   'coord' - the coordinates of the i-th weather station,
#   'X' - the feature matrix,
#   'y' - the label vector,
#   'cluster' - the index of the cluster to which node i belongs to
```

```
raise NotImplementedError
# G_FMI =


# Add edges between each station and its nearest neighbors.
# NOTE: the node degree might be different for different nodes.
numneighbors = 4
G_FMI = add_edges(G_FMI, numneighbors=numneighbors)
print("The empirical graph is connected:", nx.is_connected(G_FMI))

# Visualize the empirical graph.
plotFMI(G_FMI)
```

1.3 3. Model

1.3.1 3.1 Main hyperparameters

[ ]:
```
# Define the number of clusters and the random seed.
k = 10
seed = 4740
```

1.3.2 3.2 Student task #1 - K-Means with coordinates as a representation vector.

[ ]:
```
##################TODO###################
# TODO: 1. Create a 2-dimensional representation vector
#          with entries being the latitude and longitude
#          of each FMI station.
#       2. Cluster the nodes of G_FMI using the Python class sklearn.cluster.
 ↪KMean.
#       3. Store the cluster index in the nodes' attribute 'cluster'.

raise NotImplementedError


# Plot the clustered graph.
plotFMI(G_FMI)
```

[ ]:
```
##################TODO###################
# TODO: 1. Compute the average temperature for each cluster.
#       2. Calculate the average (over all nodes) squared
#          error loss (see the Lecture Notes 6.7).

raise NotImplementedError
# avg_error =

# Print the average error.
```

```python
print(f"The average squared loss over all datapoints is {avg_error}")
```

1.3.3 3.3 Student task #2 - K-Means with GMM parameters as a representation vector.

[ ]:
```python
# Define the number components for the GMM.
n_components = 2

###################TODO###################
# TODO: 1. Fit the GaussianMixture() model
#          to each node in the G_FMI. Use
#          the pre-defined n_componentes and
#          random_state (seed) values.
#       2. Extract the parameters of the fitted
#          model.
#       3. Create a 2-dimensional representation vector
#          of the shape (207, 114) with entries being the GMM parameters.
#       4. Cluster the nodes of G_FMI using the Python class sklearn.cluster.
 ↪KMean.
#       5. Store the cluster labels in the nodes' attribute 'cluster'.
# HINT: GMM parameters can be extracted with
#          .means_ - returns the matrix with
#                    entries being the mean vectors
#                    of each mixture component,
#          .covariances_ - returns the list of covariance matrices
#                          of each mixture component,
#          .weights_ - returns the weights of each mixture components.
#       Use .ravel() to flatten all parameters and .concatenate()
#       to stack them together.
#       Therefore, the stacked parameters of each node have the shape (114, ).
#       The raveled parameters are in the following order: means, covariances,␣
 ↪weights.

raise NotImplementedError



# Plot the clustered graph.
plotFMI(G_FMI)
```

[ ]:
```python
###################TODO###################
# TODO: 1. Compute the average temperature for each cluster.
#       2. Calculate the average (over all nodes) squared
#          error loss (see the Lecture Notes 6.7).
# NOTE: You can copy your implementation from the cell above.

raise NotImplementedError
```

```
# avg_error =


# Print the average error.
print(f"The average squared loss over all datapoints is {avg_error}")
```

1.3.4 3.4 Student task #3 - K-Means with eigenvectors of the Laplacian matrix as a representation vector.

[ ]:
```
##################TODO###################
# TODO: 1. Construct the Laplacian matrix of G_FMI.
#       2. Compute the eigenvalues and eigenvectors
#          of the Laplacian matrix.
#       3. Sort both the eigenvalues and the eigenvectors
#          in ascending order.
#       4. Use the first k eigenvectors as
#          a representation vector.
#       5. Cluster the nodes of G_FMI using the Python class sklearn.cluster.
 ↪KMean.
#       6. Store the cluster labels in the nodes' attribute 'cluster'.

raise NotImplementedError


# Plot the clustered graph.
plotFMI(G_FMI)
```

[ ]:
```
##################TODO###################
# TODO: 1. Compute the average temperature for each cluster.
#       2. Calculate the average (over all nodes) squared
#          error loss (see the Lecture Notes 6.7).
# NOTE: You can copy your implementation from the cell above.

raise NotImplementedError
# avg_error =


# Print the average error.
print(f"The average squared loss over all datapoints is {avg_error}")
```

[ ]:
```

```

FLFlavors_RefSol

June 25, 2024

1 Reference Solution for Assignment "FL Main Flavors"

1.1 1. Preparation

1.1.1 1.1 Libraries

[1]:
```
import numpy as np
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import mean_squared_error

# We will use networx objects to store empircial graphs, local datasets and␣
↪models
import networkx as nx
from sklearn.neighbors import kneighbors_graph
from numpy import linalg as LA
```

1.1.2 1.2 Helper functions

[2]:
```
# The function generates a scatter plot of nodes (=FMI stations) using
# latitude and longitude as coordinates.
def plotFMI(G_FMI):
    num_stations = len(G_FMI.nodes)
    colors = np.array(['black', 'green', 'red', 'brown', 'deeppink',
                        'blue', 'olive', 'gray', 'orange', 'purple'])
    coords = [G_FMI.nodes[i]['coord'] for i in range(num_stations)]
    df_coords = pd.DataFrame(coords,columns=['latitude','longitude'])
    coords = np.hstack((df_coords["latitude"].to_numpy().
↪reshape(-1,1),df_coords["longitude"].to_numpy().reshape(-1,1)))
    # Create a plot
    fig, ax = plt.subplots()
    # Draw nodes
    for node in G_FMI.nodes:
        color = colors[G_FMI.nodes[node]['cluster']]
```

```
        ax.scatter(coords[node,1], coords[node,0], color=color, s=4, zorder=5) ␣
↪# zorder ensures nodes are on top of edges
        ax.text(coords[node,1]+0.1, coords[node,0]+0.2, str(node), fontsize=8,␣
↪ha='center', va='center', color=color, fontweight='bold')
    # Draw edges
    for edge in G_FMI.edges:
        ax.plot([coords[edge[0],1],coords[edge[1],1]],␣
↪[coords[edge[0],0],coords[edge[1],0]], linestyle='-', color='gray', alpha=0.
↪5)

    ax.set_xlabel('longitude')
    ax.set_ylabel('latitude')
    ax.set_title('FMI stations')
    plt.show()


# The function connects each FMI station with
# the nearest neighbours.
def add_edges(graph, numneighbors=4):
    coords = [graph.nodes[i]['coord'] for i in range(num_stations)]
    df_coords = pd.DataFrame(coords,columns=['latitude','longitude'])
    coords = np.hstack((df_coords["latitude"].to_numpy().
↪reshape(-1,1),df_coords["longitude"].to_numpy().reshape(-1,1)))
    A = kneighbors_graph(coords, numneighbors, mode='connectivity',␣
↪include_self=False)
    nrnodes = len(graph.nodes)
    for iter_i in range(nrnodes):
        for iter_ii in range(nrnodes):
            if iter_i != iter_ii :
                if A[iter_i,iter_ii]> 0 :
                    graph.add_edge(iter_i, iter_ii)
    return graph

# The function below extracts a feature and label from each row
# of dataframe df. Each row is expected to hold a FMI weather
# measurement with cols "Latitude", "Longitude", "temp", "Timestamp"
# returns numpy arrays X, y.
def ExtractFeaureMatrixLabvelVector(data):
    nrfeatures = 7
    nrdatapoints = len(data)
    X = np.zeros((nrdatapoints, nrfeatures))
    y = np.zeros((nrdatapoints, 1))

    # Iterate over all rows in dataframe and create corresponding feature␣
↪vector and label
    for ind in range(nrdatapoints):
```

```
        # latitude of FMI station, normalized by 100
        lat = float(data['Latitude'].iloc[ind])/100
        # longitude of FMI station, normalized by 100
        lon = float(data['Longitude'].iloc[ind])/100
        # temperature value of the data point
        tmp = data['temp'].iloc[ind]
        # read the date and time of the temperature measurement
        date_object = datetime.strptime(data['Timestamp'].iloc[ind], '%Y-%m-%d␣
↪%H:%M:%S')
        # Extract year, month, day, hour, and minute. Normalize these values
        # to ensure that the features are in range [0,1].
        year = float(date_object.year)/2025
        month = float(date_object.month)/13
        day = float(date_object.day)/32
        hour = float(date_object.hour)/25
        minute = float(date_object.minute)/61
        X[ind,:] = [lat, lon, year, month, day, hour, minute]
        y[ind,:] = tmp

    return X, y
```

1.2 2 Data

1.2.1 2.1 Dataset

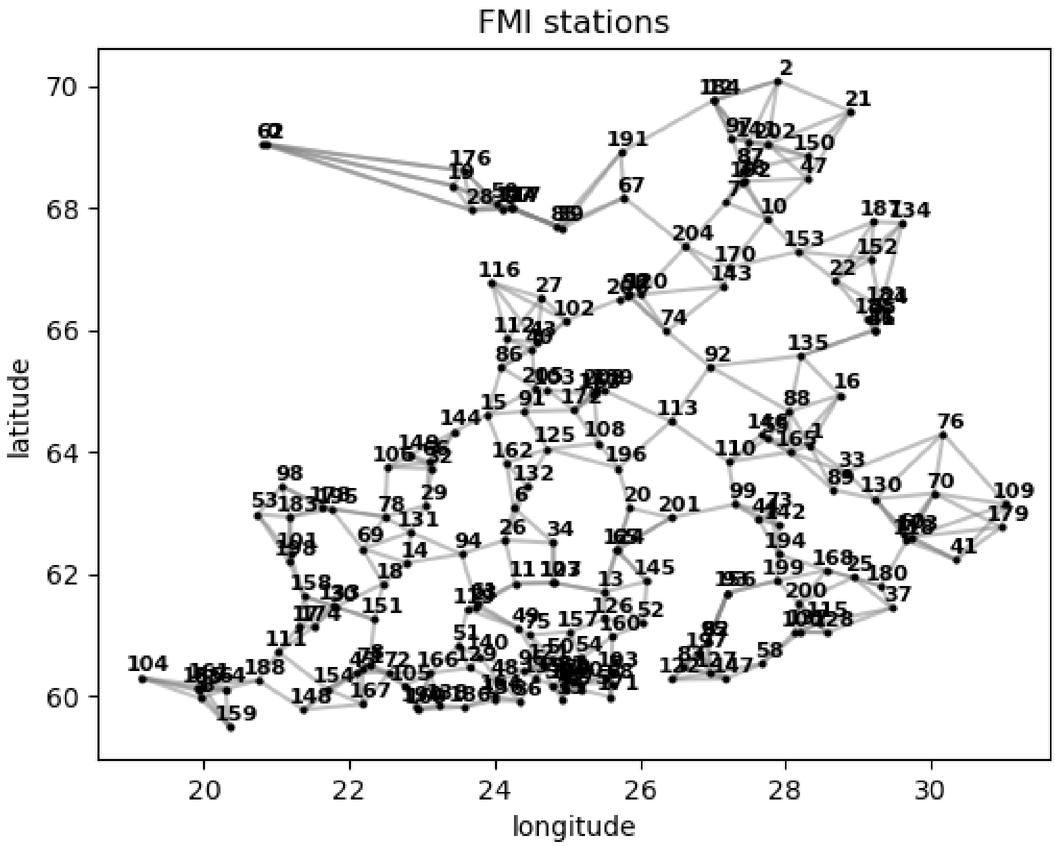
[3]:
```
# Import the weather measurements.
data = pd.read_csv('Assignment_MLBasicsData.csv')

# We consider each temperature measurement (=a row in dataframe) as a
# separate data point.
# Get the numbers of data points and the unique stations.
num_stations = len(data.name.unique())
num_datapoints = len(data)
```

1.2.2 2.2 Empirical graph

[4]:
```
# Create a networkX graph
G_FMI = nx.Graph()

# Add a one node per station
G_FMI.add_nodes_from(range(0, num_stations))

for i, station in enumerate(data.name.unique()):
    # Extract data of a certain station
    station_data = data[data.name==station]

    # Extract features and labels
```

```
    X, y = ExtractFeaureMatrixLabvelVector(station_data)

    localsamplesize = len(y)
    G_FMI.nodes[i]['samplesize'] = localsamplesize # The number of measurements of the i-th weather station
    G_FMI.nodes[i]['name'] = station # The name of the i-th weather station
    G_FMI.nodes[i]['coord'] = np.array([station_data.Latitude.unique()[0], station_data.Longitude.unique()[0]]) # The coordinates of the i-th weather station
    G_FMI.nodes[i]['X'] = X # The feature matrix for local dataset at node i
    G_FMI.nodes[i]['y'] = y  # The  label vector for local dataset at node i
    G_FMI.nodes[i]['cluster'] = 0 # The cluster to which the node is assigned (default value = 0)

# Add edges between each station and its nearest neighbors.
# NOTE: the node degree might be different for different nodes.
numneighbors = 4
G_FMI = add_edges(G_FMI, numneighbors=numneighbors)
print("The empirical graph is connected:", nx.is_connected(G_FMI))

# Visualize the empirical graph.
plotFMI(G_FMI)
```

```
The empirical graph is connected: True
```



1.3 3. Model

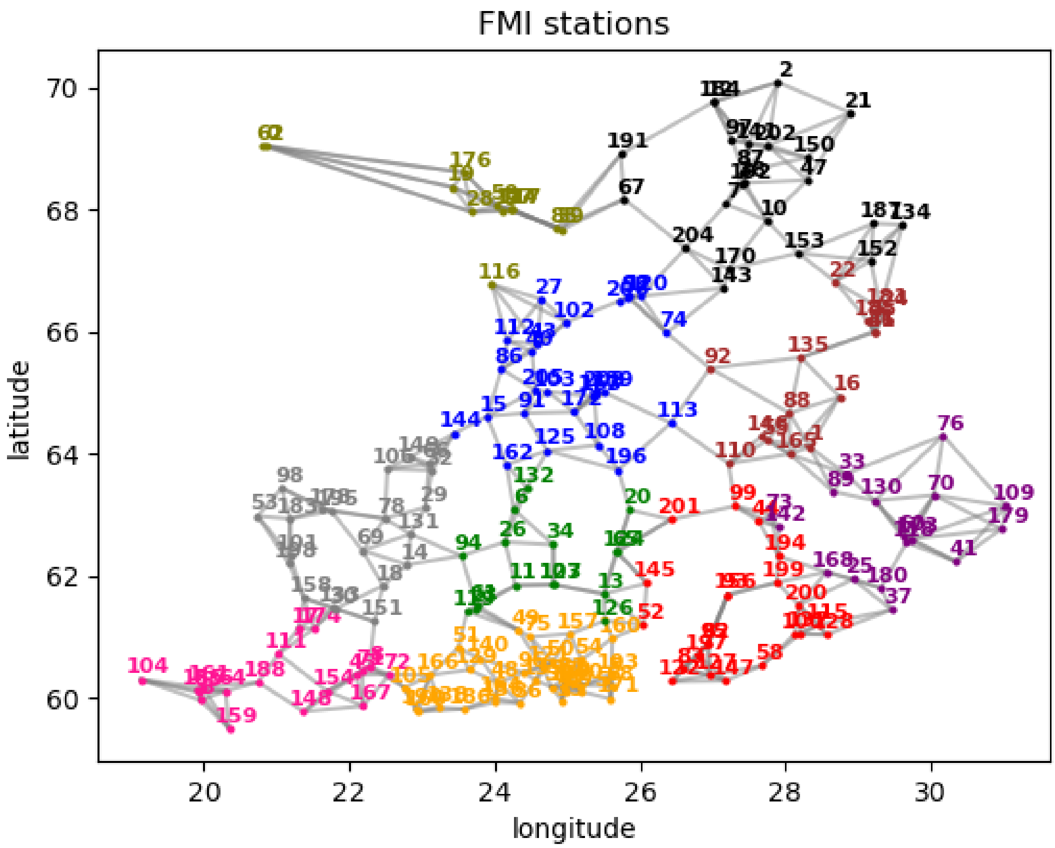
1.3.1 3.1 Main hyperparameters

[5]:
```
# Define the number of clusters and the random seed.
k = 10
seed = 4740
```

1.3.2 3.2 Student task #1 - K-Means with coordinates as a representation vector.

[6]:
```
# Create a list of all coordinates.
coordinates = np.array([G_FMI.nodes[i_node]['coord'] for i_node in G_FMI.nodes])

# Fit the K-Means to the representation vector.
kmeans = KMeans(n_clusters = k, random_state = seed, n_init = 'auto')
kmeans.fit(coordinates)

# Assign cluster labels to the nodes in the graph.
for i, node in enumerate(G_FMI.nodes):
    G_FMI.nodes[node]['cluster'] = kmeans.labels_[i]
```

```python
# Plot the clustered graph.
plotFMI(G_FMI)
```



[7]:
```python
# Create the storage for average temperatures.
avg_temperatures = np.zeros(k)
for cluster in range(k):
    # Define the variable to store the number of data points in the cluster.
    cluster_cnt = 0
    for node in G_FMI.nodes:
        if G_FMI.nodes[node]['cluster'] == cluster:
            avg_temperatures[cluster] += np.sum(G_FMI.nodes[node]['y'])
            cluster_cnt += G_FMI.nodes[node]['samplesize']
    avg_temperatures[cluster] /= cluster_cnt

# Average error over all nodes in the graph.
avg_error = 0
num_datapoints = 0
for node in G_FMI.nodes:
    cluster_avg_temp = avg_temperatures[G_FMI.nodes[node]['cluster']]
```

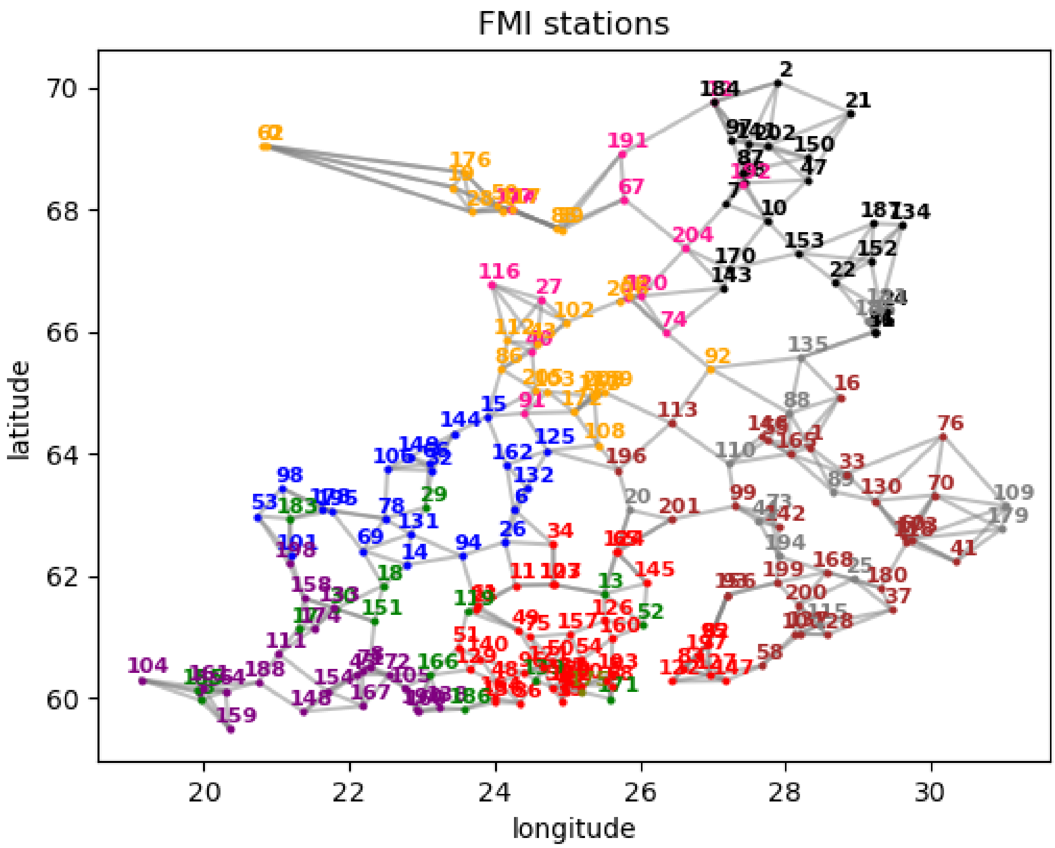
```
    node_error = np.sum((G_FMI.nodes[node]['y'] - cluster_avg_temp) ** 2)
    num_datapoints+=G_FMI.nodes[node]['samplesize']
    avg_error += node_error
    # print(f"The node {node} from the cluster {G_FMI.nodes[node]['cluster']}")
    # print(f"The average cluster temperatuer is {cluster_avg_temp}")
    # print(f"The node error is {node_error}\n")
avg_error /= num_datapoints

# Print the average error.
print(f"The average squared loss over all data points is {avg_error}")
```

```
The average squared loss over all data points is 36.41977783638262
```

1.3.3 3.3 Student task #2 - K-Means with GMM parameters as a representation vector.

[8]:
```
# Define the number components for the GMM.
n_components = 2

# Fit the nodes to the GMM model and extract the model's parameters.
for node in G_FMI.nodes():

    # Extract node's features.
    node_X = G_FMI.nodes[node]['X']

    # Fit GMM.
    gmm = GaussianMixture(n_components=n_components, random_state = seed)
    gmm.fit(node_X)

    # Get the parameters of the GMM (mean vectors, covariance matricies, and␣
 ↪component weights).
    gmm_params = np.concatenate((np.concatenate((gmm.means_.ravel(), gmm.
 ↪covariances_.ravel())), gmm.weights_))

    # Assign GMM parameters to the node.
    G_FMI.nodes[node]['gmm_params'] = gmm_params

# Get the GMM parameters of all the nodes in the graph.
gmm_params = np.array([G_FMI.nodes[i_node]['gmm_params'] for i_node in G_FMI.
 ↪nodes])

# Fit the K-Means to the representation vector.
kmeans = KMeans(n_clusters = k, random_state = seed, n_init = 'auto')
kmeans.fit(gmm_params)

# Assign cluster labels to the nodes in the graph.
for i, node in enumerate(G_FMI.nodes):
```

```
    G_FMI.nodes[node]['cluster'] = kmeans.labels_[i]

# Plot the clustered graph.
plotFMI(G_FMI)
```



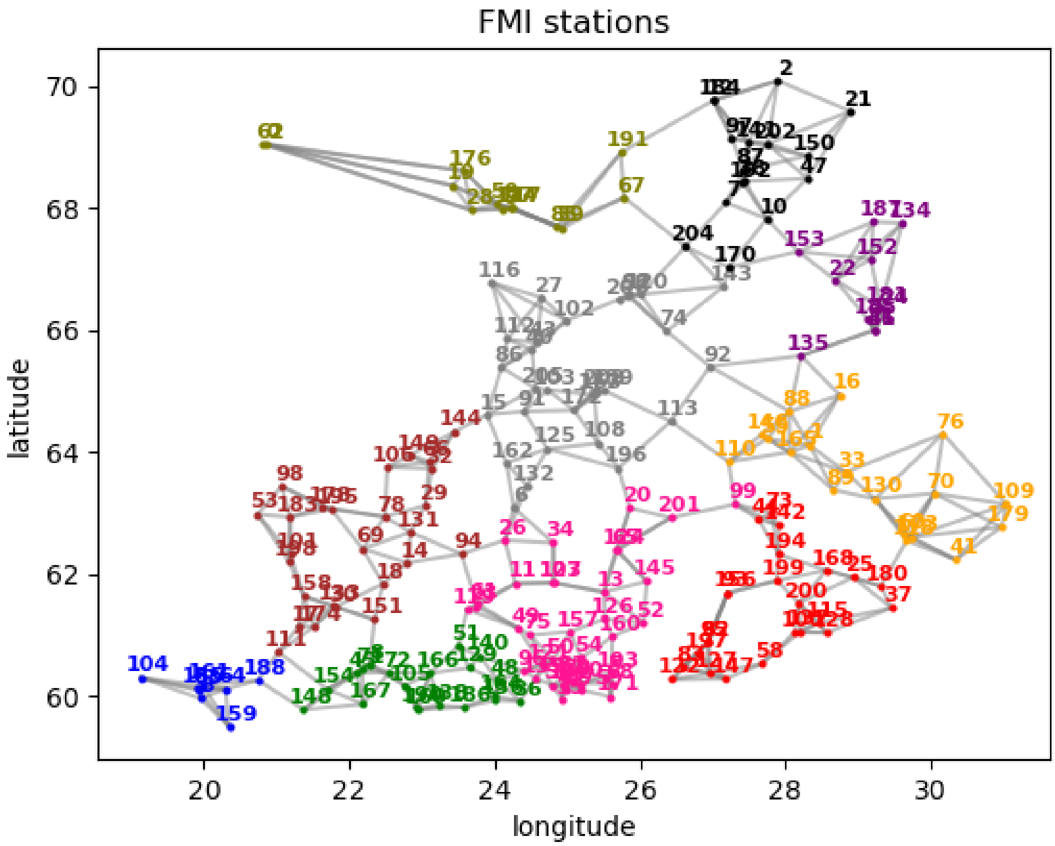
[9]:
```
# Create the storage for average temperatures.
avg_temperatures = np.zeros(k)
for cluster in range(k):
    # Define the variable to store the number of data points in the cluster.
    cluster_cnt = 0
    for node in G_FMI.nodes:
        if G_FMI.nodes[node]['cluster'] == cluster:
            avg_temperatures[cluster] += np.sum(G_FMI.nodes[node]['y'])
            cluster_cnt += G_FMI.nodes[node]['samplesize']
    avg_temperatures[cluster] /= cluster_cnt

# Average error over all nodes in the graph.
avg_error = 0
num_datapoints=0
for node in G_FMI.nodes:
```

```python
    cluster_avg_temp = avg_temperatures[G_FMI.nodes[node]['cluster']]
    node_error = np.sum((G_FMI.nodes[node]['y'] - cluster_avg_temp) ** 2)
    num_datapoints+=G_FMI.nodes[node]['samplesize']
    avg_error += node_error
    # print(f"The node {node} from the cluster {G_FMI.nodes[node]['cluster']}")
    # print(f"The average cluster temperatuer is {cluster_avg_temp}")
    # print(f"The node error is {node_error}\n")
avg_error /= num_datapoints

# Print the average error.
print(f"The average squared loss over all data points is {avg_error}")
```

```
The average squared loss over all data points is 37.87398173145912
```

1.3.4 3.4 Student task #3 - K-Means with eigenvectors of the Laplacian matrix as a representation vector.

[10]:
```python
# Construct the Laplacian matrix.
L = nx.laplacian_matrix(G_FMI).toarray()

# Compute eigenvalues and eigenvectors.
eigenvalues, eigenvectors = LA.eig(L)

idx_sorted = np.argsort(eigenvalues)

eigenvalues_sorted = eigenvalues[idx_sorted]
eigenvectors_sorted = eigenvectors.T[idx_sorted]

k_eigen = eigenvectors_sorted[:k]

# Fit the K-Means to the representation vector.
kmeans = KMeans(n_clusters = k, random_state = seed, n_init = 'auto')
kmeans.fit(k_eigen.T)

# Assign cluster labels to the nodes in the graph
for i, node in enumerate(G_FMI.nodes):
    G_FMI.nodes[node]['cluster'] = kmeans.labels_[i]

# Plot the clustered graph.
plotFMI(G_FMI)
```



[11]:
```python
# Create the storage for average temperatures.
avg_temperatures = np.zeros(k)
for cluster in range(k):
    # Define the variable to store the number of data points in the cluster.
    cluster_cnt = 0
    for node in G_FMI.nodes:
        if G_FMI.nodes[node]['cluster'] == cluster:
            avg_temperatures[cluster] += np.sum(G_FMI.nodes[node]['y'])
            cluster_cnt += G_FMI.nodes[node]['samplesize']
    avg_temperatures[cluster] /= cluster_cnt

# Average error over all nodes in the graph.
avg_error = 0
nr_datapoints = 0
for node in G_FMI.nodes:
    cluster_avg_temp = avg_temperatures[G_FMI.nodes[node]['cluster']]
    node_error = np.sum((G_FMI.nodes[node]['y'] - cluster_avg_temp) ** 2)
    avg_error += node_error
    nr_datapoints+=G_FMI.nodes[node]['samplesize']
    # print(f"The node {node} from the cluster {G_FMI.nodes[node]['cluster']}")
```

```
    # print(f"The average cluster temperatuer is {cluster_avg_temp}")
    # print(f"The node error is {node_error}\n")
avg_error /= nr_datapoints

# Print the average error.
print(f"The average squared loss over data points is {avg_error}")
```

```
The average squared loss over data points is 36.976444247108354
```

[ ]:

```

```