FLDesignPrinciple_CodingAssignment

June 25, 2024

1 Coding Assignment - "FL Design Principle"

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.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)
    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 and add labels.
    for node in G_FMI.nodes:
        ax.scatter(coords[node,1], coords[node,0], color='black', 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='black', 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')

    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 computes the average of the local loss
# incurred by given local odel parameters.
def compute_train_err(graph, localparam):

    nrnodes = len(graph.nodes)
    tmp = 0

    for iter_i in range(nrnodes):
        predictions = np.ones((graph.
↪nodes[iter_i]['samplesize'],1))*localparam[iter_i]
        local_loss = mean_squared_error(graph.nodes[iter_i]['y'], predictions)
        tmp += local_loss

    train_err = tmp / nrnodes

    return train_err


# The function computes the total variation
# of local model parameters.
```

```
def compute_totalvariation(graph,localparam):

    nrnodes = len(graph.nodes)
    tmp = 0

    total_var = 0
    for u, v in graph.edges():
        total_var += (localparam[u] - localparam[v])**2

    return total_var


# 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
```

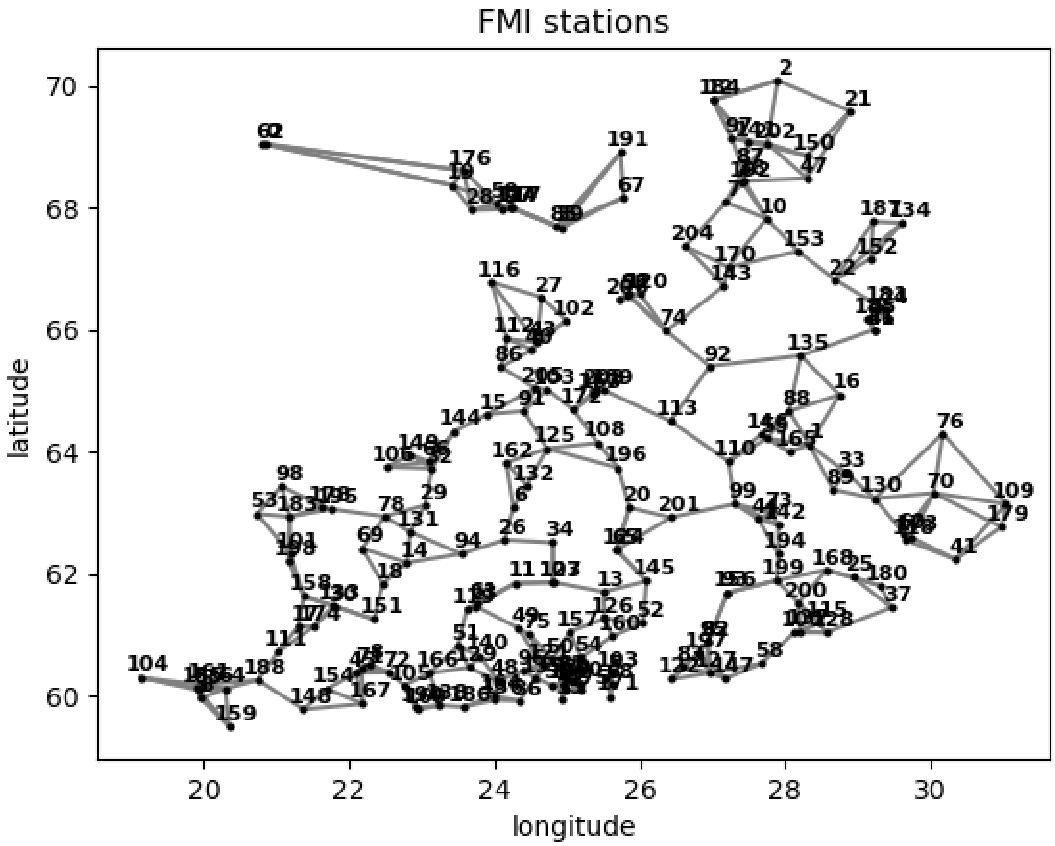
1.2 2 Data

1.2.1 2.1 Dataset

[3]:
```python
# 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))

# Add node attributes: station name, feature, and label.
yglobal = np.array([])

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]['X'] = X # The feature matrix for local dataset at node i
    G_FMI.nodes[i]['name'] = station # The name of the i-th weather station
    G_FMI.nodes[i]['coord'] = (station_data.Latitude.unique()[0], station_data.
↪Longitude.unique()[0]) # The coordinates of the i-th weather station
    G_FMI.nodes[i]['y'] = y  # The label vector for local dataset at node i
    yglobal = np.append(yglobal, y)

# Add edges between each station and its nearest neighbors.
# NOTE: the node degree might be different for different nodes.
numneighbors = 3
G_FMI = add_edges(G_FMI, numneighbors=numneighbors)

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



1.3 3. Model

1.3.1 3.1 Student task #1 - Training error and total variation

[5]:
```python
# Define the regularization parameter
gtvmin_alpha = 1

###################TODO###################
# TODO: 1. Generate the Laplacian matrix for the empirical graph.
#       2. Build matrix Q (Eq. (3.18)) and vector q (Eq. (3.19)) for the␣
↪quadratic objective
#          function in GTVMin (see Eq. (3.17)) in the Lecture Notes.
#       3. Use the zero-gradient condition (see Lecture Notes) to compute a␣
↪solution for the GTVMin instance.

raise NotImplementedError
# L_FMI =
# Q =
# q =
# hat_w =
```

```python
# Calculate and print the training error and the total variation.
print("Training error:", compute_train_err(G_FMI, hat_w))
print("Total variation:", compute_totalvariation(G_FMI, hat_w))

# Get the coordinates of each weather station
coords = nx.get_node_attributes(G_FMI, 'coord')
coords = np.array(list(coords.values()))

# Visualize the learnt model paramters in scatter plot using
# the longitue value as horizontal axis.
plt.scatter(coords[:,1], hat_w)
plt.xlabel("Longitude ot station")
plt.ylabel("Local model parameter $\widehat{w}$")
plt.show()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
Cell In[5], line 10
      2 gtvmin_alpha = 1
      4 ####################TODO#####################
      5 # 1. Generate the Laplacian matrix for the empirical graph.
      6 # 2. Build matrix Q (Eq. (3.18)) and vector q (Eq. (3.19)) for the␣
 ↪quadratic objective
      7 #    function in GTVMin (see Eq. (3.17)) in the Lecture Notes.
      8 # 3. Use the zero-gradient condition (see Lecture Notes) to compute a␣
 ↪solution for the GTVMin instance.
---> 10 raise NotImplementedError
     11 # L_FMI =
     12 # Q =
     13 # q =
   (…)
     16
     17 # Calculate and print the training error and the total variation.
     18 print("Training error:", compute_train_err(G_FMI, hat_w))

NotImplementedError:
```

1.3.2 3.2 Student task #2 - The connectivity of the empirical graph

[ ]:

```python
####################TODO#####################
# TODO: Construct different variants G_FMI using the above function add_edges()␣
 ↪with
#       different choices for the parameter "nrneighbors" (1, 2, 3, and 4) and
#       determine for each value if G_FMI is connected.
```

```
raise NotImplementedError
```

1.3.3 3.3 Student task #3 - GTVMin alpha impact

[ ]:
```
###################TODO###################
# TODO: Construct G_FMI for nrneighbors=5 and learn local model parameters␣
 ↪using
#       GTVMIN with different choices for alpha (1, 10, 100, and 1000).
#       For each choice of alpha, determine the average local loss (training␣
 ↪error)
#       and the total variation of the learnt local model parameters.

raise NotImplementedError
```

FLDesignPrinciple_RefSol

June 25, 2024

1 Reference Solution for Assignment "FL Design Principle"

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.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)
    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 and add labels.
    for node in G_FMI.nodes:
        ax.scatter(coords[node,1], coords[node,0], color='black', 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='black', 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')

    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 computes the average of the local loss
# incurred by given local odel parameters.
def compute_train_err(graph, localparam):

    nrnodes = len(graph.nodes)
    tmp = 0

    for iter_i in range(nrnodes):
        predictions = np.ones((graph.
↪nodes[iter_i]['samplesize'],1))*localparam[iter_i]
        local_loss = mean_squared_error(graph.nodes[iter_i]['y'], predictions)
        tmp += local_loss

    train_err = tmp / nrnodes

    return train_err


# The function computes the total variation
# of local model parameters.
```

```python
def compute_totalvariation(graph,localparam):

    nrnodes = len(graph.nodes)
    tmp = 0

    total_var = 0
    for u, v in graph.edges():
        total_var += (localparam[u] - localparam[v])**2

    return total_var


# 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
```

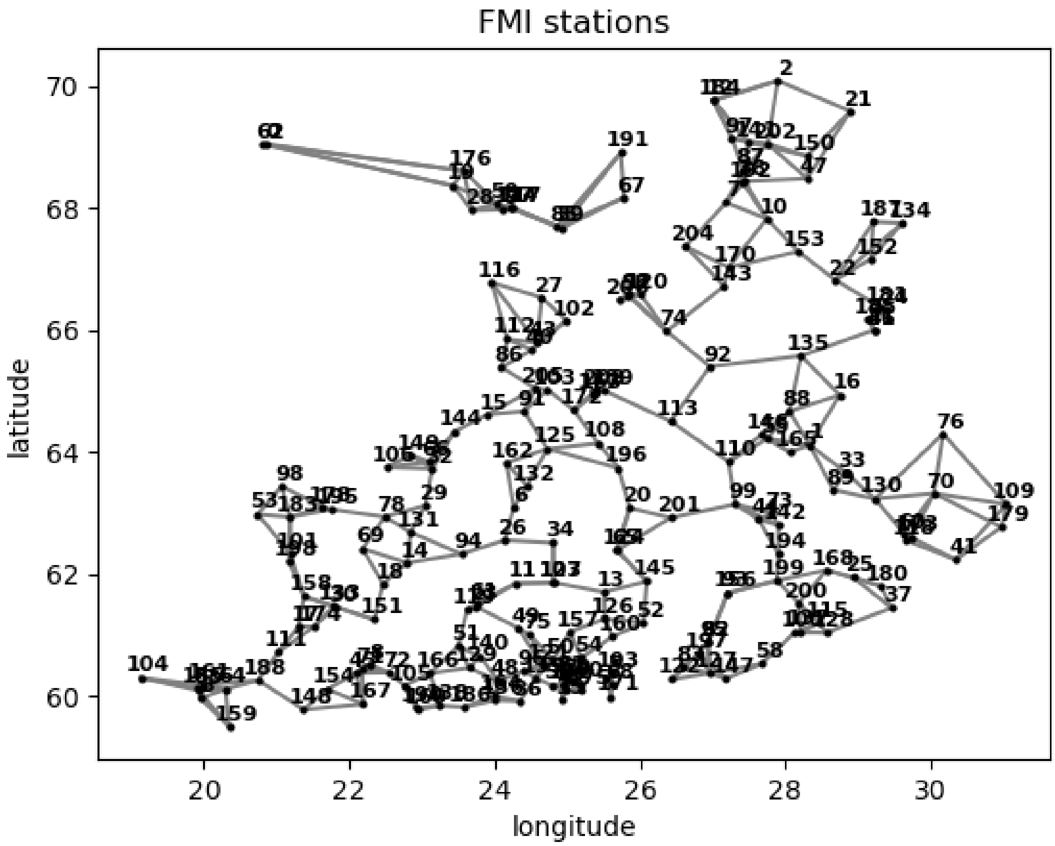
1.2 2 Data

1.2.1 2.1 Dataset

[3]:
```python
# 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))

# Add node attributes: station name, feature, and label.
yglobal = np.array([])

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]['X'] = X # The feature matrix for local dataset at node i
    G_FMI.nodes[i]['name'] = station # The name of the i-th weather station
    G_FMI.nodes[i]['coord'] = (station_data.Latitude.unique()[0], station_data.
 ↪Longitude.unique()[0]) # The coordinates of the i-th weather station
    G_FMI.nodes[i]['y'] = y  # The label vector for local dataset at node i
    yglobal = np.append(yglobal, y)

# Add edges between each station and its nearest neighbors.
# NOTE: the node degree might be different for different nodes.
numneighbors = 3
G_FMI = add_edges(G_FMI, numneighbors=numneighbors)

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



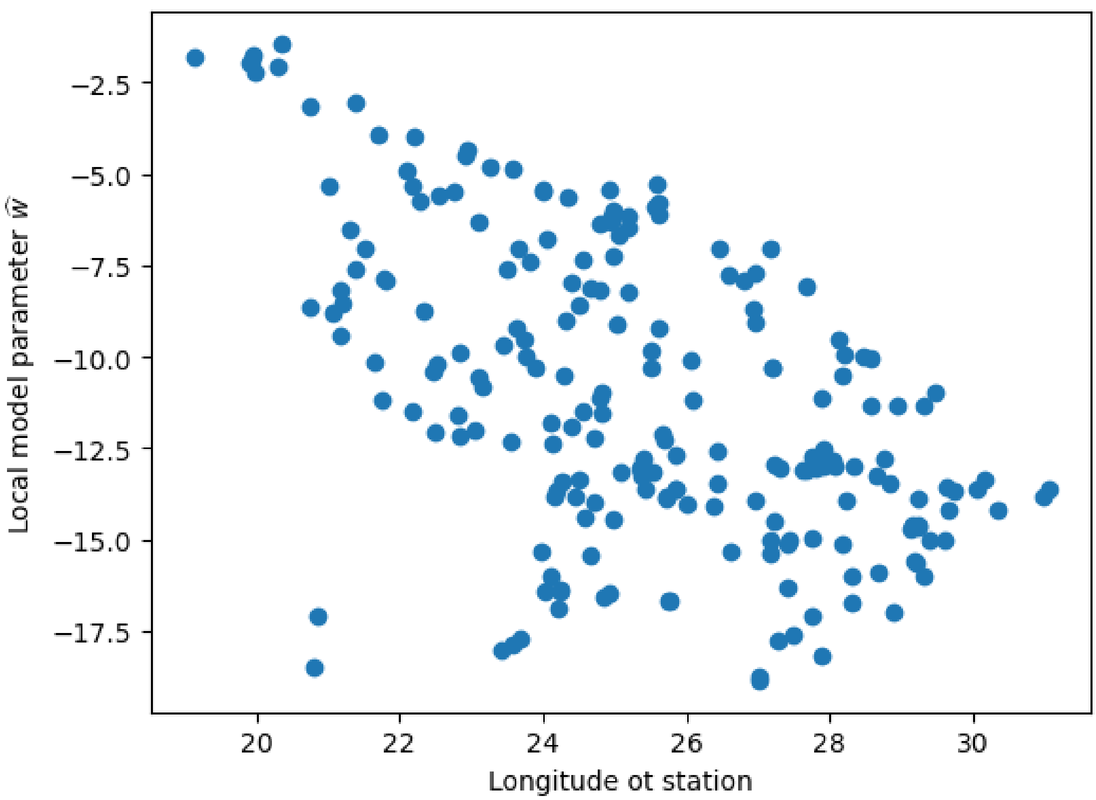
1.3 3. Model

1.3.1 3.1 Student task #1 - Training error and total variation

[5]:
```python
# Define the regularization parameter
gtvmin_alpha = 1

# Generate the Laplacian matrix for the empirical graph.
L_FMI = nx.laplacian_matrix(G_FMI).toarray()

# Build matrix Q and vector q  (Eq. (3.18)) for the quadratic objective
# function in GTVMin (see Eq. (3.17)) in the Lecture Notes.
Q = np.eye(num_stations) + gtvmin_alpha * L_FMI
q = np.zeros(num_stations)
for iter_node in range(num_stations):
    q[iter_node] = -2*np.sum(G_FMI.nodes[iter_node]['y']) / G_FMI.nodes[iter_node]['samplesize']

# Use the zero-gradient condition (see Lecture Notes) to compute a solution for the GTVMin instance.
```

```python
hat_w = LA.inv(Q).dot(-0.5*q)

# Calculate and print the training error and the total variation.
print("Training error:", compute_train_err(G_FMI, hat_w))
print("Total variation:", compute_totalvariation(G_FMI, hat_w))

# Get the coordinates of each weather station
coords = nx.get_node_attributes(G_FMI, 'coord')
coords = np.array(list(coords.values()))

# Visualize the learnt model paramters in scatter plot using
# the longitue value as horizontal axis.
plt.scatter(coords[:,1], hat_w)
plt.xlabel("Longitude ot station")
plt.ylabel("Local model parameter $\widehat{w}$")
plt.show()
```

```
Training error: 32.77241810495208
Total variation: 203.28666126559543
```



1.3.2 3.2 Student task #2 - The connectivity of the empirical graph

[6]:
```
for n_neighbors in [1,2,3,4] :
    G_FMI = nx.Graph()
    G_FMI.add_nodes_from(range(0, num_stations))
    for i,station in enumerate(data.name.unique()):
        # find rows for the same FMI station and form new dataframe df
        df = data[data.name==station]
        G_FMI.nodes[i]['coord'] = (df.Latitude.unique()[0],df.Longitude.
↪unique()[0]) # coordinates of i-th weather station
    G_FMI = add_edges(G_FMI, numneighbors = n_neighbors)
    print(f"The minimum number of nearest neighbors is {n_neighbors}, the graph␣
↪is connected: {nx.is_connected(G_FMI)}")
```

```
The minimum number of nearest neighbors is 1, the graph is connected: False
The minimum number of nearest neighbors is 2, the graph is connected: False
The minimum number of nearest neighbors is 3, the graph is connected: False
The minimum number of nearest neighbors is 4, the graph is connected: True
```

1.3.3 3.3 Student task #3 - GTVMin parameter impact

[7]:
```
# Create a networkX graph
G_FMI = nx.Graph()

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

# Add node attributes: station name, feature, and label
yglobal = np.array([])

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]['X'] = X # The feature matrix for local dataset at node i
    G_FMI.nodes[i]['name'] = station # The name of the i-th weather station
    G_FMI.nodes[i]['coord'] = (station_data.Latitude.unique()[0], station_data.
↪Longitude.unique()[0]) # The coordinates of the i-th weather station
    G_FMI.nodes[i]['y'] = y  # The label vector for local dataset at node i
    yglobal = np.append(yglobal, y)
```

```
nrneighbors = 5
G_FMI = add_edges(G_FMI,numneighbors=nrneighbors)

L_FMI = nx.laplacian_matrix(G_FMI).toarray()

alpha_vals = [1,10,100,1000]

for gtvmin_alpha in alpha_vals:
    Q = np.eye(num_stations) + gtvmin_alpha * L_FMI
    q = np.zeros(num_stations)
    for iter_node in range(num_stations):
        q[iter_node] = np.sum(G_FMI.nodes[iter_node]['y']) / G_FMI.
 ↪nodes[iter_node]['samplesize']
    # use the zero-gradient condition (see Lecture Notes) to compute a solution␣
 ↪to the GTVMin instance
    hat_w = LA.inv(Q).dot(q)

    # Calculate and print the training error and the total variation.
    print("Alpha value:", gtvmin_alpha)
    print("Training error:", compute_train_err(G_FMI, hat_w))
    print("Total variation:", compute_totalvariation(G_FMI, hat_w), "\n")
```

```
Alpha value: 1
Training error: 33.54684568071317
Total variation: 291.85650430820164

Alpha value: 10
Training error: 36.74944232826125
Total variation: 66.97955846449986

Alpha value: 100
Training error: 45.38185127582437
Total variation: 5.553538235518733

Alpha value: 1000
Training error: 50.6961893635968
Total variation: 0.08962935002391109
```

[ ]:

```

```