Assignment A2 — FL Design Principle

CS-E4740 Federated Learning

Read Carefully

There is no submission. Your work is assessed via a multiple-choice quiz.

The quiz will test the numerical quantities listed below. You must compute them according to the specification in this document and print them in your Python script.

The assessed quantities are:

1. **System statistics**
   - the average node degree $\overline{d}$ of the constructed federated learning network (FL network) $\mathcal{G}$ with nodes being Finnish Meteorological Institute (FMI) stations appearing in the CSV,
   - connectedness of $\mathcal{G}$, i.e., determine if $\mathcal{G}$ a connected
2. **Federated linear regression results**
   For each value of the regularization parameter $\alpha \in \{0, 1, 100\}$:
   - the normalized parameter variation V,
   - the average local validation error across all nodes.

No other quantities are graded. In particular, intermediate results, plots, or additional metrics are *ignored* by the quiz.

Python Environment

To ensure reproducibility, the following environment is recommended:

- Python $\geq$ 3.11
- `numpy` == 2.3.4
- `scikit-learn` == 1.7.2 (for `LinearRegression`)
- `networkx` == 3.5 (for representing the node set)
- `matplotlib` == 3.10.7 (optional; plotting not required)

You can verify installed package versions via:

```
python -m pip show numpy scikit-learn networkx matplotlib
```

1 Purpose

This assignment revolves around generalized total variation minimization (GTVMin) as a design principle for federated learning (FL) systems. Using daily weather observations from FMI stations, each node trains a local linear model whose model parameters are coupled to those of neighbouring nodes via a graph-structured regularizer (i.e., generalized total variation (GTV)).

By formulating federated linear regression as GTVMin, the assignment examines how the GTVMin parameter α controls the trade-off between independent local learning ($\alpha = 0$) and strongly aligned models over the FL network.

2 Data

We use the same raw data as in the previous assignment. In particular, you must use the following CSV file (authoritative source):

https://github.com/alexjungaalto/FederatedLearning/blob/main/Edition2026/assignments/fmidata.csv

Each row corresponds to **a specific day at a specific FMI weather station**. The CSV file contains the following columns:

`station`	station name (string)
`lat`	latitude (float)
`lon`	longitude (float)
`day`	date in `YYYY-MM-DD` format
`tmax`	daily maximum temperature (float); used as label y
`tmin`	daily minimum temperature (float); used as feature x

The following snippet illustrates the structure of the CSV file:

```
station,lat,lon,day,tmax,tmin
Alajärvi Möksy,63.08898,24.26084,2025-12-31,-15.3,-22.0
Alajärvi Möksy,63.08898,24.26084,2026-01-01,-16.5,-22.3
Porvoo Kilpilahti satama,60.30373,25.54916,2026-01-01,-3.2,-7.8
```

All temperatures are measured in degrees Celsius (°C). **You must not modify or replace the dataset**.

3 FL network

You must construct an FL network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose

- nodes $\mathcal{V} = \{1, \ldots, n\}$ represent FMI weather stations occurring in the CSV, and
- edges $\mathcal{E}$ connect geographically nearby stations (see below).

3.1 Step 1: Determine the node set from the CSV

Parse the CSV file and identify the set of FMI stations with measurement records in the file. Let n denote the number of such stations.

Assign each station a unique consecutive node index

$$i \in \{1, \ldots, n\}.$$

Node indices are fixed and must be used consistently throughout the assignment. In particular, the index i uniquely identifies a specific FMI weather station.

3.2 Step 2: Create an empty `NetworkX` graph

Store the FL network $\mathcal{G}$ as an undirected `networkx.Graph`:

```
import networkx as nx

G = nx.Graph()   # empty undirected graph
```

3.3 Step 3: Add nodes and static node attributes

For each FMI station, add one node to the FL network using its assigned index $i \in \{1, \dots, n\}$. Each node must store the following attributes:

- `station`: station name (string, from the CSV),
- `lat`: latitude (float),
- `lon`: longitude (float).

```
# Example: add nodes with required attributes

stations = [
    ("Helsinki", 60.1756, 24.9443),
    ("Espoo",       60.1767, 24.8056),
    ("Vantaa",    60.2934, 25.0378),
    ("Turku",    60.4518, 22.2666),
    ("Tampere",     61.4427, 23.7610),
    ]

for node_id, (name, lat, lon) in enumerate(stations, start=1):

        G.add_node(node_id,station=name,lat=lat,lon=lon)

# Sanity check

print(G.number_of_nodes())  # should print len(stations)
```

3.4 Step 4: Compute pairwise geographic distances

Compute the distance between two nodes i and i' via the Euclidean distance between their geographic coordinates:

$$d(i,i') \triangleq \sqrt{(\texttt{lat}_i - \texttt{lat}_{i'})^2 + (\texttt{lon}_i - \texttt{lon}_{i'})^2}.$$

3.5 Step 5: Add edges via 3-nearest neighbor

- For each node i, identify the 3 distinct nodes $i' \neq i$ with the smallest distances $d(i,i')$.
- Add an undirected edge between i and each of these 3 nearest neighbors.
- Each edge $\{i,i'\} \in \mathcal{E}$ has unit weight, i.e., $A_{i,i'} = 1$
- Note that some nodes might end up having more than 3 neighbors
- Compute the average node degree $\overline{d} = (1/n) \sum_{i=1}^{n} d^{(i)}$ of the resulting FL network
- Verify if the resulting FL network is a connected graph (e.g., using `nx.is_connected()`)

3.6 Step 6: Construct and store local datasets

For each node i, construct a station-specific training set $\mathcal{D}^{(i,\mathrm{t})}$ and validation set $\mathcal{D}^{(i,\mathrm{v})}$ and store them (in the form of `numpy.ndarray`) as `networkx` node attributes.

In particular, for each node i, construct

- `trainset`: a `NumPy` array containing all but the *last day* of measurements for the FMI station i,

- `valset`: a `NumPy` array containing the latest daily measurement at station i.

Each row of the arrays corresponds to one data point (a day at some FMI station). The feature vector of a data point is

$$\mathbf{x} = (\underbrace{1}_{x_1}, \underbrace{t_{\rm min}}_{x_2})^T,$$

where $x_1 = 1$ is a *dummy feature* used to represent the intercept term, and $t_{\rm min}$ denotes the daily minimum temperature. The label of the data point is the daily maximum temperature $t_{\rm max}$.

Columns must be ordered consistently across all nodes, and the resulting arrays must be stored directly as node attributes, accessible via

- `G.nodes[i]["trainset"]` and
- `G.nodes[i]["valset"]`.

3.7 Step 7: Local model as a node attribute

For each node i, associate a local linear regression method that is based on the assumption

$$\underbrace{t_{\rm max}}_{y} \approx \underbrace{w_1^{(i)} + w_2^{(i)} t_{\rm min}}_{h(\mathbf{x})}. \qquad (1)$$

The assumption (1) suggests to train a linear model $h(\mathbf{x}) = \mathbf{x}^T \mathbf{w}^{(i)}$ with model parameters $\mathbf{w}^{(i)} = \big(w_1^{(i)}, w_2^{(i)}\big)^T$. **Note:** The linear model is applied to the feature vector $\mathbf{x} = \big(1, t_{\rm min}\big)^T$ of a data point. The first feature $x_1 = 1$ is a dummy feature which is required to capture the intercept term $w_1^{(i)}$ in (1). We stress that each node i (representing some FMI station) learns a separate linear model with model parameters $\mathbf{w}^{(i)} = \big(w_1^{(i)}, w_2^{(i)}\big)^T \in \mathbb{R}^2$.

You must use the `scikitlearn` class `LinearRegression` to implement a linear regression method for learning the model parameter $w_1^{(i)}, w_2^{(i)}$. In particular, each node must store a `LinearRegression` object from the `scikit-learn` library as a node attribute named `model`.

Important: The `LinearRegression` object must be configured to exclude the intercept (bias) term by setting `fit_intercept=False`, since we instead include a dummy feature $x_1 = 1$ for each data point.

At this stage, no assumptions are made about how the model is trained or validated; these aspects will be specified later.

Summary of required node attributes:

attribute	description
`station`	station name
`lat`	latitude
`lon`	longitude
`trainset`	local training set (`NumPy` array)
`valset`	local validation set (`NumPy` array)
`model`	`LinearRegression` object (without intercept)

4 Federated linear regression

You have to implement a simple federated linear regression method which is obtained by applying block-coordinate optimization to the GTVMin instance

$$\min_{\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}\in\mathbb{R}^d} \sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2$$
$$\text{with } L_i\left(\mathbf{w}^{(i)}\right) = (1/|\mathcal{D}^{(i,\mathrm{t})}|) \sum_{(\mathbf{x},y)\in\mathcal{D}^{(i,\mathrm{t})}} \left(y - (w_1^{(i)} + w_2^{(i)} x_2)\right)^2. \tag{2}$$

In particular,

- For each node i, initialize $\mathbf{w}^{(i)}$ by plain linear regression using only the local training set $\mathcal{D}^{(i,\mathrm{t})}$.
- For $t = 0, 1, \ldots, 99$, perform the following synchronous updates at all nodes:

$$\mathbf{w}^{(i,t+1)} \in \arg\min_{\mathbf{w}\in\mathbb{R}^2} L_i(\mathbf{w}) + \alpha \sum_{i'\in\mathcal{N}^{(i)}} \left\|\mathbf{w} - \mathbf{w}^{(i',t)}\right\|_2^2. \tag{3}$$

- Store the learnt model parameters $\widehat{\mathbf{w}}^{(i)} = \mathbf{w}^{(i,100)}$ which are needed for computing the results in the next task.

This update can be implemented *without custom optimizers* by applying `LinearRegression.fit()` to an augmented version of the local training set $\mathcal{D}^{(i,\mathrm{t})}$. You might also need to use non-uniform sample weights in `LinearRegression.fit()`.

Note:

- The update (3) must be executed *synchronously* at each node i, i.e., using neighbours' model parameters $\mathbf{w}^{(i',t)}$, for $i' \in \mathcal{N}^{(i)}$.
- Make sure that your Python implementation does not accidentally overwrite model parameters $\mathbf{w}^{(i',t)}$ which are still needed on the right-hand side of the update (3) at some other nodes i.

5 Results

Run the above federated linear regression method for different choices of the regularization parameter α:

- $\alpha = 0$ (no coupling),
- $\alpha = 1$ (mild coupling),
- $\alpha = 100$ (strong coupling).

For each choice of α, determine

- the normalized variation of model parameters

$$V = (1/\overline{d}) \frac{\sum_{\{i,i'\}\in\mathcal{E}} \left\|\widehat{\mathbf{w}}^{(i)} - \widehat{\mathbf{w}}^{(i')}\right\|_2^2}{\sum_{i\in\mathcal{V}} \left\|\widehat{\mathbf{w}}^{(i)}\right\|_2^2}.$$

Here, $\overline{d}$ denotes the average node degree of the FL network.

- the average local validation error

$$(1/n) \sum_{i \in \mathcal{V}} \underbrace{\frac{1}{|\mathcal{D}^{(i,\mathrm{v})}|} \sum_{(\mathbf{x},y) \in \mathcal{D}^{(i,\mathrm{v})}} \big(y - (\widehat{w}_1^{(i)} + \widehat{w}_2^{(i)} x_2)\big)^2}_{\text{validation error at node } i}. \tag{4}$$