Machine Learning for IOT

Homework 1

*****DUE DATE: 26 Nov (h23:59)*****

Submission Instructions:

Each group will send an e-mail to daniele.jahier@polito.it and valentino.peluso@polito.it (in cc) with subject *ML4IOT25 GroupN* (replace *N* with the group ID). Attached to the e-mail a single ZIP archive (*.zip*) named *HW1_GroupN.zip* (replace *N* with the group ID) containing the following files:

1. The code deliverables specified in the text of each exercise.
2. One-page pdf report, titled *GroupN_Homework1.pdf*, organized in different sections (one for each exercise). Each section should motivate the main adopted design choices and discuss the outcome of the exercise.

Late messages, or messages not compliant with the above specs, will be automatically discarded.

Exercise 1: Timeseries Processing for Memory Optimization (3 points)

1.1 Memory optimization for Temperature & Humidity Monitoring System (2pt)

On the RPI, develop a Python script (ex1.py) to measure temperature and humidity using the DHT-11 sensor, and upload the collected data to Redis. The script should adhere to the following specifications:

- Create two timeseries called *mac_address:*temperature and *mac_address:*humidity.
- Set the **data acquisition interval** for both timeseries to **2 seconds**.
- Set the **retention period** for the two timeseries to **30 days**.
- Create three additional timeseries for temperature called *mac_address:*temperature_min, *mac_address:*temperature_max, and *mac_address:*temperature_avg. These timeseries should store the minimum, maximum, and average temperature values, calculated every **1 hour**, with a **retention period** of **365** days.
- Create three additional timeseries for humidity called *mac_address:*humidity_min, *mac_address:*humidity_max, and *mac_address:*humidity_avg. These timeseries should store the minimum, maximum, and average humidity values, calculated every **1 hour**, with a **retention period** of **365** days.
- Enable compression for all the timeseries.
- The script should be run from the command line and take as input the following arguments:
  - --host (*str*): the Redis Cloud host.
  - --port (*int*): the Redis Cloud port.
  - --user (*str*): the Redis Cloud username.
  - --password (*str*): the Redis Cloud password.

1.2 Reporting (1pt)

In the PDF report:

- Answer the following question: How much memory (in GB) is needed to provide this monitoring service for 1000 clients?
- Consider the average compression ratio for estimating the required memory.
- Report and discuss the calculations made.

Code Deliverables

a) A single Python script named *ex1.py* that contains the code of 1.1. The code is intended to be run on the Raspberry PI without installing additional software or dependencies on the provided SD card. Moreover, the script must include all necessary classes and methods needed for its correct execution.

Exercise 2: Voice Activity Detection Optimization & Deployment (3 points)

2.1 VAD Optimization (1pt)

- In Deepnote, optimize the hyperparameter values of Voice Activity Detection (VAD) implementation based on spectrogram features introduced in the notebook "HW1: Voice Activity Detection" from the *Material* project (link). The VAD class specifications are as follows:

Class Name **VAD**

Description Classify an audio file as *silence* or *non-silence* using spectrogram features. The VAD algorithm consists of the following steps:

1. Calculate the spectrogram of the audio.
2. Convert the amplitude values to decibels (dB).
3. For each time frame of the spectrogram, compute the average amplitude across all frequency bins.
4. For each time frame, compute the relative amplitude with respect to the frame with minimum average amplitude.
5. Identify frames where the relative amplitude exceeds a predefined threshold (*dBthres*) and mark these frames as non-silence.
6. Calculate the total duration, in seconds, of the non-silence segment.
7. If this duration exceeds a specified threshold (*duration_thres*), classify the entire audio file as *non-silence*; otherwise, classify it as *silence*.

Init Args

- **sampling_rate**: *int*. Sampling rate of the input.
- **frame_length_in_s**: *float*. Frame length (in seconds) of the STFT.
- **frame_step_in_s:** *float*. Frame step (in seconds) of the STFT; must be less than or equal to *frame_length_in_s.*
- **dBthres**: *float*. dB threshold.
- **duration_thres**: *float*. Non-silence duration threshold in seconds.

Methods

is_silence

Args

- **audio**: 1D *Tensor* of type *float32* storing a normalized audio.

Returns

- **is_silence**: An *int* equal to 1, if the audio has been classified as silence, 0 otherwise.

- Use the *vad-dataset* available in Deepnote to evaluate the accuracy of the VAD using different hyperparameter values.
- Deploy the VAD class to the RPI using the script *vad_latency.py* available in *Portale delle Didattica* to measure the median latency for different hyperparameter values.
- Find a configuration for the hyperparameters of the *VAD* object such that **all** the constraints below are met:
  - Accuracy on the *vad-dataset* > 97.6%
  - Median Latency on the RPI < 25 ms

2.2 VAD Deployment (1pt)

On the RPI, develop a Python script (ex2.py) to measure temperature and humidity, controlled through a Voice User Interface (VUI) based on VAD.

The system should perform data collection every 2 seconds and display the results on the command line (without uploading data to Redis). The VUI will enable or disable data collection based on voice commands. The script should follow the requirements below:

- Initially, the data collection is disabled.
- The VUI always runs in background and continuously records audio data with the USB microphone. Configure the audio recording with 1 channel, 16-bit depth (int16), and a sampling rate of 48 kHz.
- Every second, the VUI analyzes the last second of recorded audio to detect the presence of speech using the *is_silence* method from the VAD class.
- Before passing the audio data to the *is_silence* method, applies the following transformations:
  - Cast to np.float32
  - Downsample to 16 kHz
  - Convert to TensorFlow tensor
  - Squeeze the tensor
  - Normalize the tensor
- If the VAD returns *non-silence*, toggle the data collection state (enabled if it was disabled, or vice versa).
  - After toggling, stay in the new state for at least 5 seconds before checking for further state changes.
- If the VAD returns *silence*, maintain the current state.
- When enabled, the system should measure temperature and humidity every 2 seconds and print the data to the command line.
- The script should be executable from the command line.

Example:

```
python ex2.py
```

2.3 Reporting (1pt)

In the PDF report:

- Report and discuss the search space used to discover the VAD hyperparameters compliant with the constraints.
- Add a table reporting the selected values of the VAD hyperparameters (*frame_length_in_s*, *frame_step_ins*, *dBFSthres*, *duration_thres*).
- Comment the table discussing how each hyperparameter affects accuracy and/or latency.
- Elaborate on the fundamental choices needed to match the target constraints and explain why the selected configuration ensures faster processing than the default settings of the *vad_latency.py* script.

Code Deliverables

- A single Python script named *ex2.py* that contains the code of 2.2. The code is intended to be run on the Raspberry PI without installing additional software or dependencies on the provided SD card. Moreover, the script must include all necessary classes and methods needed for its correct execution.