Machine Learning for IoT

Homework 1

*****DUE DATE: 28 Nov (h23:59)*****

Submission Instructions:

Each group will send an e-mail to andrea.calimera@polito.it and valentino.peluso@polito.it (in cc) with subject *ML4IOT24 TeamN* (replace *N* with the team ID). Attached with the e-mail a single ZIP archive (*.zip*) named *HW1_TeamN.zip* (replace *N* with the team ID) containing the following files:

1. The code deliverables specified in the text of each exercise.
2. One-page pdf report, titled *TeamN_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: Voice Activity Detection Optimization & Deployment (3 points)

1.1 VAD Optimization (1pt)

In Deepnote, optimize the hyper-parameters of the *VAD* object (*LAB2 – Exercise* 3) such that **all** the constraints below are met:

- Accuracy on the *vad-dataset* > 98.5%
- Median Latency savings > 20%

Latency savings must be computed with respect to a reference *VAD* instantiated with the following hyper-parameters:

- Sampling rate: 16 kHz
- Frame length: 50 ms
- # of Mel bins: 80
- Lower frequency: 0 Hz
- Upper frequency: 8 kHz
- dbFS threshold: -35
- Duration threshold: 0.2 seconds

and using the following formula: *100 × (reference – optimized) / reference*.

Note: Measure accuracy and median latency in Deepnote using the *benchmark_vad* method defined in the *Voice Activity Detection* notebook.

1.2 VAD Deployment (1pt)

On your PC, develop a Python script that continuously records audio data and stores it on disk only if it contains speech. Follow the instructions reported below:

- In VS Code, write a Python script to record audio with your PC and the integrated/USB microphone. Set the # of channels to 1, the resolution to *int16*, and the sampling frequency to 16 kHz (if not supported by your microphone, apply resampling).
- In the same script, integrate the *VAD* class for processing recorded *numpy* arrays. For converting *numpy* arrays to *Tensorflow* tensors use the *tf.convert_to_tensor* method:

```
tf_indata = tf.convert_to_tensor(indata, dtype=tf.float32)
```

Remember to squeeze and normalize *tf_indata* before feeding it to the *is_silence* method of the *VAD* object.

- **Every 0.5 seconds** (in parallel with the recording), check if the **last second** of recorded audio contains speech using an instance of the *VAD* class with the hyper-parameters of 1.1. If *is_silence* returns 0, store the audio data on disk using the timestamp as the filename, otherwise discard it. For the implementation of this functionality, define a global audio buffer initialized with all zero values and updated periodically every 0.5 seconds.
- Stop the recording and exit when the Q key is pressed.
- The script should be run from the command line interface and should take as input a single argument called *--device* (int) that specifies the ID of the microphone used for recording.

Example:

```
python ex1.py --device 0
```

1.3 Reporting (1pt)

In the PDF report:

- Describe the methodology adopted to discover the *VAD* hyper-parameters compliant with the constraints.
- Add a table reporting the selected values of the *VAD* hyper-parameters (*frame_length_in_s*, *num_mel_bins*, *lower_frequency*, *upper_frequency*, *dbFSthres*, *duration_thres*).
- Comment the table and explain which hyper-parameters affect accuracy and latency, respectively. Elaborate on the fundamental choices needed to match the target constraints and explain why the selected configuration ensures faster processing than the reference implementation.

Code Deliverables

- A single Python script named *ex1.py* that contains the code of 1.2. The code is intended to be run on a laptop and must use only the packages that get installed with the *requirements.txt* provided in *lab1-getting-started*. Moreover, the script should contain all the classes and methods needed for its correct execution.

Exercise 2: Memory-constrained Timeseries Processing (3 points)

2.1 Memory-constrained Battery Monitoring System (2pt)

Starting from the solution of *LAB1 – Exercise 2.1*, design and develop a memory-constrained battery monitoring system with the following specifications:

- Set the acquisition period of *mac_address:*battery and *mac_address:*power to 1 second.
- Create a new timeseries called *mac_address:*plugged_seconds that, every hour, automatically stores how many seconds the power has been plugged in the last hour.
- Set a retention period of 1 day for *mac_address:*battery and *mac_address:*power and of 30 days for *mac_address:*plugged_seconds.
- How many clients can be monitored with this configuration considering a database with a maximum of 100 MB of memory? Create all the timeseries with compression activated and consider the average compression ratio for calculating the maximum number of clients.
- The script should be run from the command line interface and should 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.

2.2 Reporting (1pt)

In the PDF report:

- Explain how you created and set up the *mac_address:*plugged_seconds timeseries.
- Answer the question in the previous section, reporting and commenting on the calculations made.

Code Deliverables

a) A single Python script named *ex2.py* that contains the code of 2.1. The code is intended to be run on a laptop and must use only the packages that get installed with *requirements.txt* provided in *lab1-getting-started*. Moreover, the script should contain all the classes and methods needed for its correct execution.