Homework 1 - Group 11

Feraud Elisa (s295573), Scalenghe Luca (s303452), Tchouamou Pangop Christelle Elsa (s295753)

Machine Learning For IoT, Politecnico di Torino, A.A.2022-2023

1 Exercise 1.3 Reporting

We found the hyperparameters using a grid-search in order to get feedback and confront it with what we knew about STFT. The inspection of the code of get_spectrogram() gave us important information on which parts were the most critical with respect to the metrics.
The grid-search was done using 4 parameters, specifically:

- downsampling_rate, which is the process of reducing the sample rate by an integer factor. A signal can be downsampled by a factor of Q by retaining every Qth sample and discarding the remaining samples.This gives a loss in quality of the wave, but up until 8kHz the sampling frequency for voice is acceptable.
- frame_length_in_s, which is the dimension of the window in which you compute the Fourier transformation.
- dbFSthres, is the threshold that tells when to discriminate a signal in noise or in speech
- duration_thres, is the quantity of time an audio file had to be classified as not silence in order to label the file audio as voice.

The STFT implementation uses the FFT which is much faster. FFT splits the matrix multiplication needed in the Fourier transformation into smaller and simpler ones recursively. This brings the complexity from O(N^2) to a much faster O(N*logN), but this can only happen when the input size is a square of two. An example can be:
sampling_frequency = 16000 and frame_len_in_s = 0.032
The number of samples used for computing the transform for that chunk is 16000*0.032 = 512 = 2^9.
In table 1 we reported the selected values of the VAD hyper-parameters.

The accuracy is affected by all the hyper-parameters, while the average latency is affected by the downsampling_rate and the frame_lenght_in_s. As shown in the yellow box, a lower sampling frequency gave a huge increase in avg_latency even if theoretically this should not be the case because there are less computations to do when computing the transform. We think that much of the time spent is because of the preprocessing steps used to downsample and prepare the data. Using a frame_length_in_s that, multiplied by the sampling_rate, does not give a square of two number impacted on the latency as you can see in the orange cells, this is because the STFT could not use the FFT implementation effectively. By analyzing the code it is pretty obvious that dbFSthres and duration_thres do not affect avg_latency, this is clearly visible for the dbFSthres in the blue cells. The best results obtained are those in the green lines. The good choice of values for frame_length_in_s enables the FFT, thus faster computation.
It is worth mentioning that the measures taken with Deepnote were not always stable so this result could be not replicable with a single trial, maybe because Deepnote does not give always the same computing power. The parameters used for the second part of the exercise are:
downsampling_rate = 16000, frame_length_in_s = 0.016, dbFSthres = -120, duration_thres = 0.06

2 Exercise 2.2 Reporting

The timeseries mac_address:plugged_seconds stores one value every 24 hours. This will contain how many seconds the power has been plugged in that time period. This is done by using the aggregation rule that follows:
redis_client.ts().createrule(mac_power_name, mac_plugged_s_name, aggregation_type = 'sum', bucket_size_msec = bucket_duration_in_ms). The aggregation strategy is a sum because every second in which the power was plugged has in the original timeseries a 1 otherwise 0, by summing them you obtain how many seconds the laptop was loading in that day. The bucket_size_msec contains the value of 24 hours expressed in milliseconds. In order to set the retention period of the timeseries mac_address:battery and mac_address:power and occupy less than 5 MB, we did the following calculations:

1. considering the average compression ratio of 90% the uncompressed memory is 50M because X - 0.90X = 5MB.
2. The number of records will be $\frac{50MB}{16bytes}$ because each record is 16 bytes (8 for timestamp and 8 for the value). This corresponds to 3276800 records. Since records are stored every second, the number of record corresponds to the number of seconds. In milliseconds it will be 3276800*1000.

To set the retention period with the memory upperbound of 1MB for the mac_address:plugged_seconds we noticed that having the same conditions we could take the number of records obtained before and divide it by 5 because the upper bound was $\frac{1}{5}$ of before. The result was 655360 records. Since we only store 1 value every 24 hours the retention period in milliseconds will be 655360*24h*60min*60s*1000.

downsampling_rate	frame_length_in_s	dbFSthres	duration_thres	accuracy	avg_time_ms
8000	0,008	-120	0,05	98,44%	33,164
8000	0,016	-120	0,05	98,44%	34,614
8000	0,032	-120	0,05	98,44%	34,430
8000	0,064	-120	0,05	88,89%	34,885
16000	0,015	-130	0,06	98,44%	10,120
16000	0,015	-130	0,05	98,44%	10,186
16000	0,015	-125	0,1	97,89%	10,199
16000	0,015	-125	0,05	98,44%	10,238
16000	0,015	-130	0,1	98,44%	10,256
16000	0,015	-120	0,1	97,67%	10,332
16000	0,015	-120	0,06	97,89%	10,362
16000	0,015	-120	0,05	98,11%	10,449
16000	0,015	-125	0,06	98,56%	10,658
16000	0,035	-130	0,1	98,78%	11,463
16000	0,035	-130	0,05	98,44%	11,512
16000	0,035	-130	0,06	98,44%	11,596
16000	0,035	-125	0,1	98,33%	11,629
16000	0,035	-125	0,06	98,00%	11,655
16000	0,035	-125	0,05	98,00%	11,739
16000	0,035	-120	0,1	98,33%	11,788
16000	0,035	-120	0,06	98,11%	11,879
16000	0,035	-120	0,05	98,11%	12,026
16000	0,008	-120	0,05	97,67%	9,953
16000	0,016	-120	0,05	98,22%	9,722
16000	0,032	-120	0,05	98,33%	9,647
16000	0,064	-120	0,05	88,89%	9,502
16000	0,016	-130	0,1	98,44%	8,800
16000	0,032	-130	0,06	98,56%	8,844
16000	0,016	-130	0,06	98,33%	8,889
16000	0,016	-130	0,05	98,33%	8,890
16000	0,032	-125	0,1	98,44%	8,917
16000	0,016	-125	0,06	98,33%	8,949
16000	0,016	-125	0,1	98,11%	8,957
16000	0,032	-125	0,06	97,89%	8,991
16000	0,016	-120	0,05	98,20%	8,817
16000	0,032	-120	0,05	98,30%	8,810
16000	0,016	-120	0,06	98,20%	8,842
16000	0,032	-120	0,06	98,30%	8,850
16000	0,016	-120	0,1	97,50%	8,811
16000	0,032	-120	0,1	98,20%	8,767
16000	0,032	-50	0,05	53,00%	9,581
16000	0,032	-60	0,05	69,44%	9,577
16000	0,032	-70	0,05	82,56%	9,560
16000	0,032	-80	0,05	89,56%	9,315
16000	0,032	-90	0,05	93,78%	9,239
16000	0,032	-100	0,05	96,56%	9,205
16000	0,032	-110	0,05	97,78%	9,215
16000	0,032	-120	0,05	98,33%	10,611
16000	0,032	-130	0,05	98,56%	9,147
16000	0,032	-140	0,05	95,67%	9,238
16000	0,032	-150	0,05	88,67%	9,224
16000	0,032	-160	0,05	88,89%	9,450
16000	0,032	-170	0,05	88,89%	9,438
16000	0,032	-180	0,05	88,89%	9,404

Figure 1: Table 1