Machine Learning for IoT

Homework 1

Matteo Donadio
Student ID: s303903
Politecnico di Torino

Alessandro Maria Feri'
Student ID: s305949
Politecnico di Torino

Mattia Sabato
Student ID: s305849
Politecnico di Torino

I. Exercise 1

A. GridSearch

In order to identify the right combination of hyperparameters, yielding the optimal trade-off between performances and computational cost, and complying with the specified constraints ($accuracy \geq 98.5, latency\_saving \geq 20\%$), we conducted a GridSearch over the spaces outlined in Table I, for a total of 324 combinations.

ID	Hyperparameter	Search Space
-	`sampling_rate`	{16000}
`[h1]`	`frame_length`	{0.016, 0.032, 0.064}
`[h2]`	`mel_bins`	{8, 10, 12}
`[h3]`	`lower_freq`	{0, 80}
`[h4]`	`upper_freq`	{6000, 8000}
`[h5]`	`dbFS_thresh`	{-38, -40, -42}
`[h6]`	`dur_thresh`	{0.04, 0.05, 0.06}

TABLE I: *Hyperparameters Search Spaces*

We ended up with the configuration shown in Table II, where we report for simplicity just the IDs of the hyperparameters, and their performance with respect to the *accuracy* (*acc*) and the median *latency_saving* (*ls*).

B. Correlations with accuracy and latency

In defining our search spaces we considered the theoretical impact of each hyperparameter on both accuracy and latency, thus being able to trim the less promising solutions. Regarding `mel_bins`, it covers a paramount importance when considering accuracy, since a too small value would lead to loss of details, whereas an excessively high value would yield a detrimental too fine grained information, potentially related to noise. Furthermore, increasing `mel_bins` would result in higher computational costs and, consequently, latency. Regarding the `frame_length`, our choice was to consider values that allowed us to work with powers of 2, and so to leverage the notable efficiency gain due to the FFT, which allowed to drastically decrease the latency. The range comprised between `lower_freq` and `upper_freq` has been defined considering the usual frequency range related to speech, and it did not actually have a relevant impact neither on the accuracy nor on the latency. The `dbFS_thresh`, instead, impacts significantly on the accuracy, since it is crucial in determining whether there is silence or not. Finally, `dur_thresh` mainly impacts the accuracy, and it has been tuned so to consider the right interval to avoid both false positive and false negatives.

`[h1]`	`[h2]`	`[h3]`	`[h4]`	`[h5]`	`[h6]`	*acc*	*ls*
0.032	12	0	8000	-42	0.05	98.78	31.26

TABLE II: *Best Hyperparameters and related performances*

II. Exercise 2

In order to design and implement a memory-constrained battery monitoring system, we established a time series labeled `mac_address:plugged_seconds` which, every hour, automatically stores the duration in seconds for which the power has been plugged in during the previous hour, with a retention time of 30 days. The data addition process utilizes the `createrule` method, which performs an aggregation by summing the binary values linked to the power stored in the corresponding time series and calculates the cumulative duration every hour. Considering the constrained memory scenario (100MB storage), we made several assumptions: we considered only the average compression ratio, neglected the header size and assumed that the memory usage is always a multiple of the chunk size. To estimate the maximum number of clients the database could handle within the given constraint, we have proceeded considering the memory occupied by the two time series with keys `mac_address:battery` and `mac_address:power`, which both have a retention time of one day, and both save data every second. Regarding the total number of records associated with the last two time series, this is equal to:

$$24\frac{[h]}{[day]} \times 60\frac{[min]}{[h]} \times 60\frac{[rec]}{[min]\,[ts]} \times 2\,[ts] = 172800\frac{[rec]}{[day]}$$

Given the aforementioned acquisition and retention periods of `mac_address:plugged_seconds`, its maximum total number of records will be:

$$30\frac{[day]}{[month]} \times 24\frac{[rec]}{[day]} = 720\frac{[rec]}{[month]}$$

Finally, the maximum total number of records per client would be the sum of the related records from the `battery` and `power` time series with the ones coming from `plugged_seconds`, resulting in a total number of records equal to $172800 + 720 = 173520$. The uncompressed memory is approximately 173520×16 bytes $\approx$ 2.65MB which, thanks to the Gorilla compression algorithm, that has an average compression ratio of 90%, turns into $2.65\text{MB} - 90\% = 0.265\text{MB}$ memory occupied. So, in a worst case scenario, the system is able to sustain a maximum number of clients equal to:

$$\left\lfloor \frac{100\,[\text{MB}]}{0.265\,[\text{MB}]\,/\,[client]} \right\rfloor = 377\,[client]$$