



HAL
open science

Introduction to Online Machine Learning for Embedded Systems (ESP32)

Christophe Cerin, Mamadou Sow, Muhammad Sanaullah Kayani

► To cite this version:

Christophe Cerin, Mamadou Sow, Muhammad Sanaullah Kayani. Introduction to Online Machine Learning for Embedded Systems (ESP32). Compas 2024 Conférence francophone d'informatique en Parallélisme, Architecture et Système, IETR (Institut d'Electronique et des Technologies du numéRique) et le LS2N (Laboratoire des Sciences du Numérique de Nantes) - Nantes Université, Jul 2024, Nantes, France. hal-04680048v1

HAL Id: hal-04680048

<https://hal.science/hal-04680048v1>

Submitted on 28 Aug 2024 (v1), last revised 29 Aug 2024 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Introduction to Online Machine Learning for Embedded Systems (ESP32)

Christophe Cerin*, Mamadou SOW*, Muhammad Sanaullah Kayani

*Sorbonne Paris Nord University, LIPN, 99 avenue J.B. Clément, 93430 Villetaneuse France
Grenoble Alpes University, 621 avenue Centrale, 38400 Saint-Martin-d'Hères France
cerin@lipn.univ-paris13.fr, mamadou.sow@lipn.univ-paris13.fr,
muhammad-sanaullah.kayani@etu.univ-grenoble-alpes.fr

Abstract

Due to the explosive growth of data traffic in the Internet of Things systems (IoT), machine learning and data-driven approaches are expected to become a key factor in fueling the development of wireless networks beyond 5G (B5G). Standard machine learning approaches require centralizing training data on a single data center such as a cloud. However, due to privacy constraints and limited communication resources for data transmission, it is impractical for all wireless devices to transmit all their collected data to a data center that can use the collected data to implement centralized machine learning algorithms for data analysis and inference. This has led to the emergence of a fast-growing research area, called Edge Learning, which can deeply integrate two major areas : wireless communication and machine learning. Our work revolves around implementing online machine learning algorithms at the edge, specifically the k-means clustering algorithm as a use case on ESP32-WROOM microcontroller.

ESP32 can work with sensors for things like measuring temperature, but it can also handle more complicated jobs like executing "complex" machine learning (ML) algorithms. We're particularly interested in its industry-leading performance in electronic integration, power consumption, and connectivity. Using its built-in WIFI that uses very little power, the ESP32 receives data from a dataset, corresponding to a building located at Grenoble, and processes the k-means algorithms under study. The data we process on the board, comprising CO2 volume and temperature readings, is obtained from an MQTT server, which implements the publication-subscription paradigm. This paradigm allows for asynchronous and non-blocking message communication, enhancing efficiency.

We aim to study models capable of continual learning from new data, without revisiting past data. One of the K-Means algorithms we study iteratively groups unlabeled data into K clusters based on cluster centers (centroids). The data assigned to each cluster is determined such that their average distance from their respective centroid is minimized. In this case, we develop with the Arduino toolchain. In another case, to efficiently put in place our software code, we utilize the Micropython-ulab environment on ESP32, providing a robust platform for coding ML algorithms. We are also studying River, a dedicated Python online machine learning tool, but not capable of running on ESP32-like boards. Finally, we provide lessons learned and insights into our use case. The presentation aims to discuss the difficulties in these tasks, both from a technical point of view and from an algorithmic one.

Mots-clés : Online machine learning algorithms, ESP32 and microcontrollers architectures, Micropython and Arduino ecosystems, Edge computing.

1. Introduction

Our internship aims to develop a model capable of continuous learning from sequential data streams on embedded systems, ensuring adaptation to new information without revisiting past data. We focus on a model that can adapt to concept drift and suit real-world production contexts, especially event-based ones, while seamlessly integrating into the embedded systems programming ecosystem. Online machine learning presents challenges such as catastrophic forgetting, where previous knowledge is abruptly lost upon learning new information. To address these challenges, we explore ecosystems such as micropython-ulab, Scikit-learn, and River, offering out-of-core implementations and user-friendly libraries for machine learning on streaming data. River, a combination of Creme and scikit-multiflow, provides a wide range of algorithms for continual learning and streaming data processing, encompassing linear models, decision trees, anomaly detection, clustering, and more.

In smart buildings, prioritizing residents' data privacy and security may prevent data transmission to external cloud servers (such as Amazon or Alibaba Cloud). This shift requires redesigning building data management systems, processes, and controllers. As a result, it raises further research into optimizing the use of computing, networking, and storage resources owned by building occupants and visitors, along with evaluating the performance of embedded devices within the building. These aspects are vital for efficiently utilizing on-site data for learning purposes. Inside a building, a CO₂ sensor and temperature sensor generate data, motivating our interest in clustering the data to identify anomalies and applying regression techniques for analysis. By tackling these challenges, our internship aims to enhance the learning capabilities of embedded systems for practical implementation in real-world scenarios, while optimizing power consumption utilizing low-power devices such as the ESP32.

1.1. Methodology

An MQTT server gathers real-time data including Co₂ volume and temperature measurement from the sensor and sends it to the ESP32-embedded device for further processing. We have set up MicroPython, uLab, and other required libraries on the ESP32 to implement online machine-learning algorithms. We are collecting power consumption data into our matrices for further optimization, and then clustering the data using online K-means algorithms. Online machine learning sequentially updates the best predictor using incoming data. To validate the result, we compare the performance of River, a user-friendly ML library for streaming data, with our developed clustering algorithm, evaluating quality-oriented metrics like Jaccard and Jini indexes, and metrics for concept drift estimation.

2. Related work

The literature on data stream clustering algorithms emphasizes their importance in partitioning observations into clusters to reduce data complexity. These algorithms are designed for situations where data flows continuously, presenting difficulties related to memory and time limitations, particularly in IoT and edge computing environments. We study one existing algorithm [2]. It outlines two-phase schemes for data stream clustering, with online components processing data points and offline components generating clusters. Methodological considerations include models for data streams and sliding windows to handle potentially infinite and non-stationary data.

Ghesmoune, M. and Lebbah [3] provide valuable insights into modern data stream clustering methods, categorizing them by their clustering strategies. While they explore hierarchical stream methods like neural gas, we focus on partitioning-based clustering, notably the k-means algorithm, which relies on distance metrics to form clusters. Our approach emphasizes simplicity, sidestepping the requirement for intricate data structures and optimizations. Moreover, we capitalize on the efficiency of the MQTT server and its data publication rate to enhance batch processing in our work.

3. Problem statement and algorithmic solution

3.1. Problem statement

Sensors continuously transmit data to a server using the MQTT protocol as shown in Figure 1. The “Processor for clustering” subscribes to these messages from the server and conducts the clustering process. The publisher (MQTT server) and the clustering processor are low-cost machines. It’s feasible for a single machine e.g. ESP32 to handle both tasks, allowing data collection and clustering services to run on the same device. This co-location reduces communication overhead by maintaining data production and processing nearby. We’ve chosen to illustrate only one processing machine for clarity, but multiple sources can supply data to this machine. Therefore, we haven’t addressed scalability and architecture dimensioning. Our proposed architecture is additionally suitable for using phones or tablets as processing machines, given that the MQTT protocol is already supported on these devices.

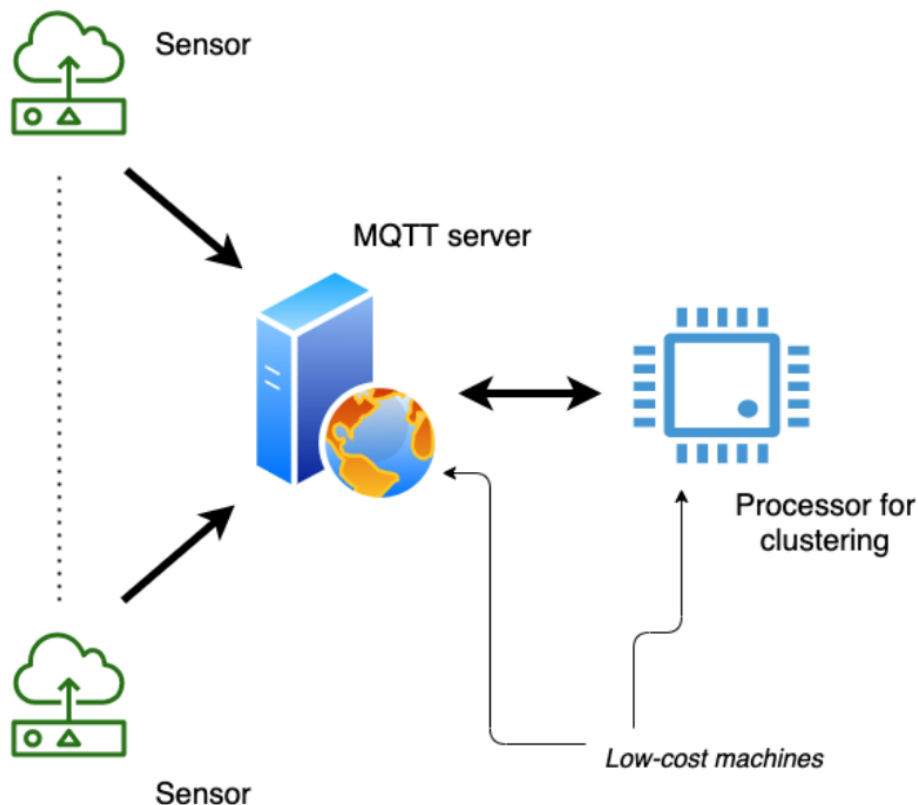


FIGURE 1 – Sensors Data Collection and Processing

3.2. Proposed Solution

The Stream window clustering algorithm [1] efficiently processes streaming data by partitioning it into fixed-size windows and adapting to its dynamic nature. Initially, it employs k-means or k-means++ clustering to identify patterns within each window. Subsequently, data points are sorted based on coordinates for streamlined processing. Regularly removing points maintains data diversity, with new ones from the stream replacing them. Continuous centroid updates mirror the changing data distribution. Utilizing proven algorithms and possibly implementing multithreading guarantees scalability and efficiency, rendering it well-suited for real-time processing of large-scale data streams.

Many existing streaming clustering algorithms cannot serve as baseline algorithms because they are not designed to meet the requirements of low-cost machines or memory-constrained devices. Therefore, comparing them would not be fair. In summary, the key properties of our generic framework are as follows :

- The algorithm is based on offline clustering, allowing for traditional methods like k-means to be utilized.
- When repeating the clustering process, only a small fraction of data ($p \ll W$) needs to be inserted, enabling control of memory space.
- Each sorting step can potentially be performed in parallel, if hardware permits, aiding in preserving diversity in the data.
- The concept of eliminating a fraction of data regularly corresponds to maintaining diversity in the input.

4. Experiments

This section overviews the testing devices employed and outlines the experimental design. Our experimentation involves applying online k-means clustering to evolving CO2 and temperature data collected from various zones within the IMAG building, with a focus on detecting concept drift for subsequent analysis.

4.1. Test Equipment and Tools

We utilized the low-cost machine to achieve our objective, our experimental setup includes the following hardware and software components :

4.1.1. ESP-WROOM-32D

We programmed the Micropython-ulab collaboratory on ESP32 to achieve our results. It's a powerful and versatile Wi-Fi and Bluetooth module developed by Espressif Systems. It features a dual-core Tensilica LX6 microprocessor with a clock speed of up to 240 MHz, offered robust processing capabilities for a wide range of applications while minimizing energy consumption.

4.1.2. MQTT Server

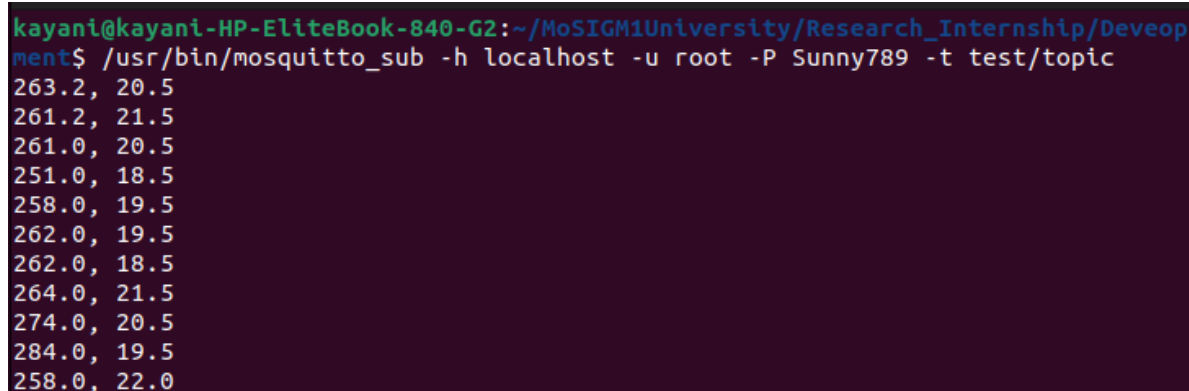
An MQTT (Message Queuing Telemetry Transport) server, also known as an MQTT broker, is a crucial component in the MQTT communication protocol, which is designed for lightweight, efficient message exchange in IoT and embedded systems. It's widely used in IoT applications, such as smart buildings, for efficient real-time data transmission between sensors and processing units. Popular implementations include Mosquitto, HiveMQ, and EMQX.

4.1.3. Setting of MQTT Client

To configure the MQTT client to connect to an MQTT broker, Broker Address (Hostname or IP), Port Number, Username, and Password (If authentication is enabled) are required to establish

a connection between the devices. After configuring the client, you can publish or subscribe the messages to MQTT topics using the `mosquitto_pub` or `mosquitto_sub` commands given below.

- `mosquitto_pub -h <10.188.186.20> -p <1883> -u <root> -t <test/topic> -m <message>`
- `mosquitto_sub -h <10.188.186.20> -p <1883> -u <root> -t <test/topic>`



```
kayani@kayani-HP-EliteBook-840-G2:~/MoSIGM1University/Research_Internship/Deveopment$ /usr/bin/mosquitto_sub -h localhost -u root -P Sunny789 -t test/topic
263.2, 20.5
261.2, 21.5
261.0, 20.5
251.0, 18.5
258.0, 19.5
262.0, 19.5
262.0, 18.5
264.0, 21.5
274.0, 20.5
284.0, 19.5
258.0, 22.0
```

FIGURE 2 – Setting MQTT client to subscribe data

4.1.4. Setup Thonny IDE

This is the IDE we use to perform multiple experiments for our project. Thonny is an Integrated Development Environment (IDE) specifically designed to make Python programming more accessible and user-friendly. It supports virtual environment management, allowing users to isolate projects and dependencies for better organization and reproducibility. So, it's easy for us to install a library on ESP32 as it provides built-in internal support to install libraries and plugins. Furthermore, establishing a connection of micropython-ulab over the ESP32 board is very convenient compared to a manual shell tool. You can upload your code on ESP32 in simple clicks and execute your code over there without flashing ESP32. These all features caught our attention to choose Thonny IDE for our research project.

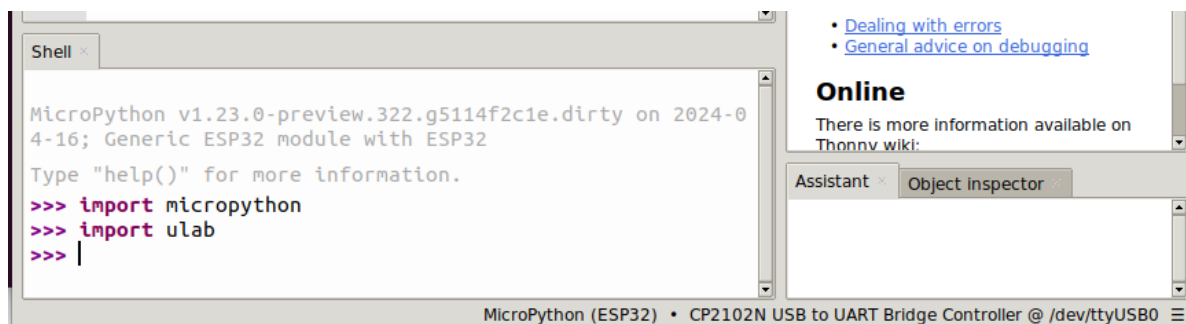


FIGURE 3 – Micropython-ulab on ESP32

4.2. Experiment on ESP32 to apply an Online K-Means using Micropython

Online K-means clustering is a variation of the traditional K-means algorithm, designed for dynamic and incremental data. In this method, we define a target number k , which refers to the number of centroids needed in the dataset. A centroid represents the central point of a cluster. Each data point is incrementally assigned to a cluster, minimizing the sum of squares within the cluster. Unlike the traditional approach, where the algorithm first identifies k centroids and then allocates each data point to the nearest cluster, online K-means updates the centroids iteratively as new data points are processed. This allows the centroids to adjust dynamically to new data. The “means” in K-means refers to the average of the data points within each cluster, effectively finding the center of gravity for each cluster.

In our approach to applying online K-means clustering, we process data received from an on-line stream in discrete windows. Each window is handled sequentially, allowing for continuous and incremental clustering. As new data arrives in each window, we perform the clustering process on that window, updating the centroids iteratively to reflect the latest data points. This dynamic updating ensures that the clustering model remains responsive to changes in the data distribution over time.

In addition to clustering, we implement a mechanism to detect drift in the data stream. Detecting drift is crucial for further analysis, as it helps identify significant changes in the underlying data patterns. By monitoring for drift, we can adjust our clustering model to maintain accuracy and relevance in changing data environments.

4.2.1. Data Exploration

We utilized the Seaborn and Matplotlib modules to visualize data distributions from multiple sources, such as CO2 levels, temperature, and clusters. Seaborn offers a handy feature called Distplot, which combines a histogram with a line plot to show how the data is spread out. This type of plot, also called a distribution diagram, helps to see the range and variability in the data. Seaborn’s Distplot is especially handy for getting an overall view of how continuous data variables are distributed.

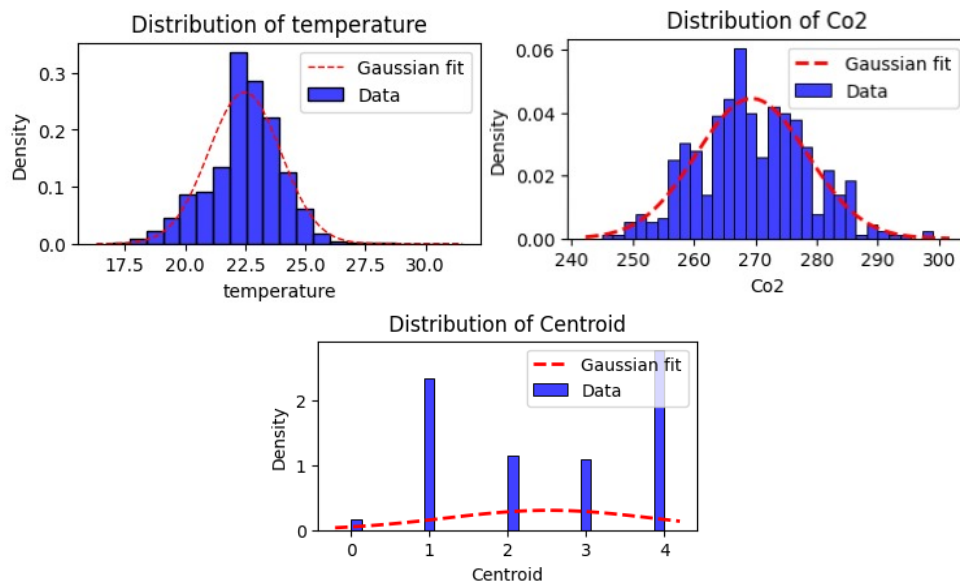


FIGURE 4 – Distribution of Data

4.2.2. Applying K-Means Clustering with Micropython-ulab and K=5

K is the number of clusters to form as well as the number of centroids to generate. The initialization method with k-means allows to selection of the initial cluster centroids using sampling based on an empirical probability distribution of the point's contribution to the overall inertia.

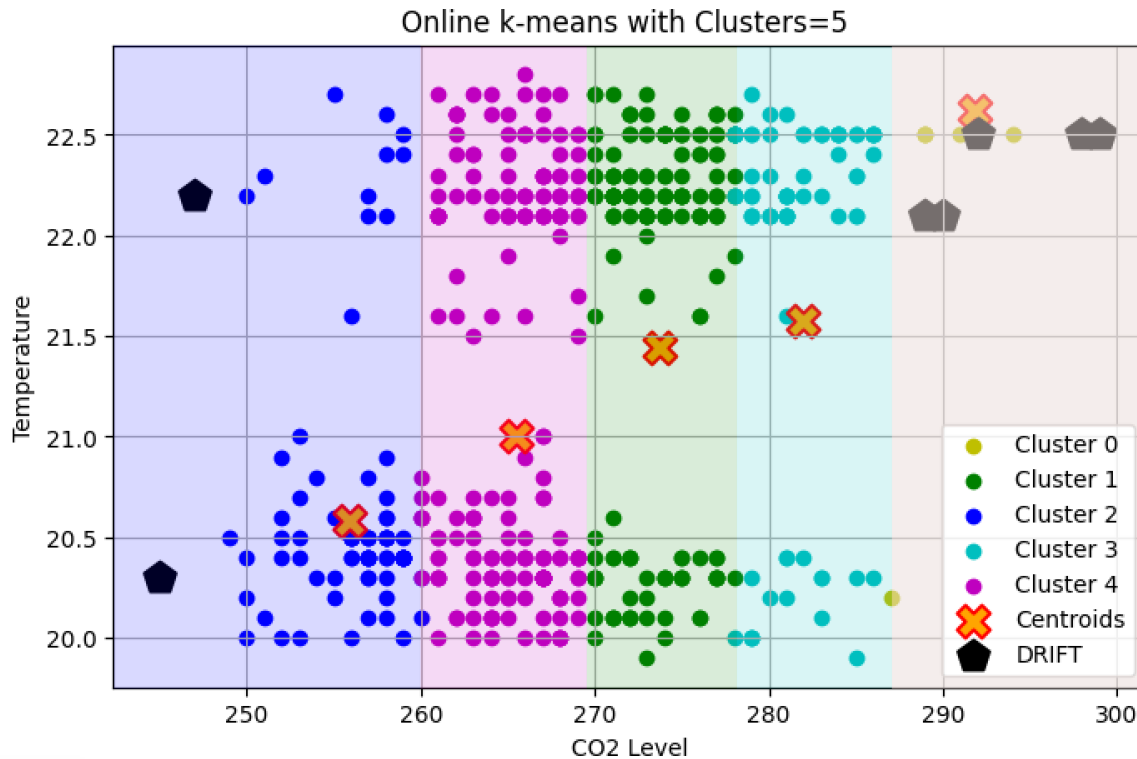


FIGURE 5 – Applying KMeans Clustering with Micropython-ulab and K=5

The scatter plots in the form of a cloud of points thus show how the Temperature variable is affected by the CO2 concentration variable. We cannot deduce either a linear or non-linear correlation. While we observe a linear evolution of the 5 centroids

From the result of online k means with distance method we can analyze that the data stream is coming continuously and we don't exactly know the exact number of clusters at initial stage. The purpose of this testing is to estimate the processing and power capacity of ESP32 using micropython-ulab. Our subsequent experiment will explore DBSTREAM Clustering, which operates on density rather than distance, eliminating the need to specify the value of k initially.

4.2.3. Measure the Capacity of ESP32

We experimented to evaluate the ESP32's performance in processing varying data sizes across different windows using MicroPython. The results are presented in Figure 6.

Based on the table in fig 6, we can conclude that the ESP32 microcontroller can efficiently handle a stream of 5120 messages with optimized batch processing of 128 messages. In contrast, the Arduino can handle a stream of 1024 messages with the same optimized batch processing of 128 messages.

	256 messages		512 messages		1024 messages		5120 messages	
W Slice	It. Max	It. Got	It. Max	It. Got	It. Max	It. Got	It. Max	It. Got
512	x	x	1	1	2	1	10	2
256	1	1	2	2	4	3	20	4
128	2	2	4	4	8	8	40	40
64	4	4	8	8	16	16	80	80
32	8	8	16	16	32	32	160	160

FIGURE 6 – Message Flow, Processing Slice, and Number of Possible Iterations

5. Conclusion

In this research project, we explored the application of online machine-learning algorithms on the ESP32-WROOM microcontroller for real-time data processing. We propose a generic framework for data stream clustering on low-cost machines, focusing on the Internet of Things (IoT) and Edge computing contexts. Our primary focus was on implementing and evaluating the k-means clustering algorithm using CO2 and temperature data collected from various zones within a building. We successfully demonstrated that the ESP32, a low-cost and low-power device, can handle real-time data streams and perform online clustering effectively.

Our experiments revealed several key insights. The ESP32 demonstrated an ability to efficiently manage the online k-means clustering algorithm, underscoring its potential for real-world applications in resource-constrained environments. We observed that the accuracy of clustering results improved with larger window sizes; however, reinitializing clusters at each iteration often led to misclassification. Optimal results were consistently achieved with a well-initialized centroid, which reduced the necessity for multiple iterations. While the k-means algorithm performed adequately under certain conditions, it encountered difficulties with dynamic data streams and required predetermined cluster numbers. In contrast, density-based algorithms like DBSCAN and DBSTREAM emerged as more promising alternatives, offering greater accuracy and flexibility without predefined clusters.

Through our study, we highlighted the importance of addressing concept drift and anomalies in streaming data to maintain model accuracy and reliability. Our findings underscore the potential for implementing sophisticated machine learning algorithms on embedded devices, opening doors to more advanced and autonomous IoT systems.

Future work aims to adapt all River algorithms to the microcontroller context. The ultimate goal is to develop a comprehensive online machine-learning library specifically designed for microcontrollers. Additionally, we plan to create our online machine-learning algorithms depending on the specific scenarios encountered and studied in the LIGLAB.

Applying our developed system in various real-world scenarios, conducting detailed case studies, and exploring its integration with diverse machine learning frameworks and tools will help us assess its practicality and effectiveness across different applications. This comprehensive approach to future research will significantly contribute to edge computing and online machine learning, enabling more intelligent and autonomous IoT systems to operate efficiently even in resource-constrained environments.

Bibliographie

1. Carnein (M.) et Trautmann (H.). – Optimizing data stream representation : An extensive survey on stream clustering algorithms. *Business & Information Systems Engineering (BISE)*, vol. 61, 2019, pp. 277–297.
2. Cérin (C.), Kimura (K.) et Sow (M.). – Data stream clustering for low-cost machines. *Journal of Parallel and Distributed Computing*, 2023.
3. Ghesmoune (M.), Lebbah (M.) et Azzag (H.). – State-of-the-art on clustering data streams. *Big Data Analytics*, vol. 1, 2016, p. 13.

[1] [2] [3]