Estudio de tolerancia a errores en sensores empleados para detección de ocupación de habitantes de un recinto empleando técnicas de aprendizaje automático

Carlos Binker[1], Hugo Tantignone[1], Lautaro Lasorsa[1], Guillermo Buranits[1], Eliseo Zurdo[1], Maximiliano Frattini[1]

[1]Universidad Nacional de La Matanza, Florencio Varela 1903 (B1754JEC) -- San Justo, Buenos Aires, Argentina

{cbinker, htantignone, laulasorsa, gburanits, eazurdo, mfrattini}@unlam.edu.ar

Resumen. En este paper se consideró un caso de estudio que propuso la obtención de un modelo predictivo que determina la ocupación de habitantes de un recinto en función de ciertas variables de entrada, como ser temperatura, humedad, luz, CO2 y humedad relativa, obtenidas a partir de un dataset externo. El caso de estudio pretende determinar que bien tolera el modelo la falla de los sensores que representan a cada una de las variables predictoras indicadas, sin que el modelo se dé cuenta de dicha falla. A tal efecto se plantean dos escenarios. En el primer escenario, se considera que un sensor falla de manera abrupta, es decir que esa variable que representa en realidad queda deshabilitada. En cambio, en el segundo escenario, a cada sensor que falla en vez de deshabilitarlo, se le introduce un error (un ruido), y se observa como éste ruido se traslada en la predicción de la ocupación de habitantes de un recinto.

Palabras claves: Machine learning, TensorFlow, IoT, dataset, Random Forest

1 Introducción

1.1 Problemática a resolver

La idea inicial de este trabajo consiste en la obtención de un modelo predictivo (empleando técnicas de machine learning [1]), que a partir de un conjunto de datos (dataset) [2] el cual está conformado por valores de temperatura, dióxido de carbono, humedad, luz y humedad relativa que están asociados con una serie de sensores específicos para tal fin, sea capaz de predecir la ocupación o no de personas dentro de un recinto. Por lo tanto, este modelo predictivo poseerá un conjunto de variables de entrada predictoras y una variable de salida dependiente que será la indicación de la existencia o no de personas, tal como lo muestra la Figura 1.

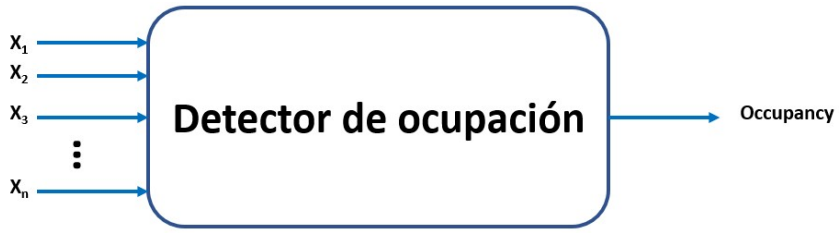

Figura 1. Modelo predictor de ocupación de habitantes generado por machine learning

En este caso de estudio se plantearon dos escenarios bien diferenciados. En el primero se consideró que un sensor falla en su totalidad, es decir podemos suponer que dicha variable predictora está ausente, y por lo tanto por ser cinco las variables predictoras se consideró el entrenamiento de un modelo para 31 subconjuntos de dichas variables predictoras, excluyendo el conjunto vacío obviamente. Dado que el número de variables predictoras del dataset bajo estudio es relativamente pequeño, el costo computacional para este primer escenario resulta relativamente bajo y por ende perfectamente realizable. Luego en el segundo escenario se consideró la situación en donde cada sensor no falla abruptamente (es decir el sensor no está ausente), sino que se le introduce un ruido que simula un error en su comportamiento. Dicho ruido se introduce realizando un promedio ponderado entre el valor de la variable en el dataset y una distribución normal con misma media y varianza que los datos de entrenamiento. Se ampliará más adelante durante el desarrollo de la experiencia. A nivel práctico se empleó Python [3] por ser un lenguaje altamente desarrollado para su empleo en machine learning.

1.2 Descripción de las variables de entrada y de los sensores asociados

El dataset empleado para nuestro caso de estudio fue extraído del siguiente site de Kaggle:

https://www.kaggle.com/datasets/kukuroo3/room-occupancy-detection-data-iot-sensor

Kaggle.com es un sitio web de uso público que reúne cientos de datasets de diversas temáticas. En nuestro caso elegimos una temática relacionada con el IoT, porque este estudio constituirá el paso preliminar para la puesta práctica del modelo obtenido en un dispositivo de borde, como ser un ESP32 o una Raspberry Pi. A continuación se presenta un cuadro representativo del dataset donde constan las variables de entrada y la salida a predecir. Como puede observarse se trata de 5 (cinco) variables de entrada. Las mismas son: temperatura, humedad, Luz, dióxido de carbono y humedad relativa. Por lo tanto la salida de nuestro sistema será una variable booleana a la que denominaremos *Occupancy,* siguiendo la nomenclatura brindada por el dataset (ver Figura 2).

```
data = pd.read_csv("Occupancy.csv")
# Elimino el timestamp porque no me interesa usar esta información
data = data.drop(columns=["date"])
data
```

[22]

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
0	23.7000	26.2720	585.200000	749.200000	0.004764	1
1	23.7180	26.2900	578.400000	760.400000	0.004773	1
2	23.7300	26.2300	572.666667	769.666667	0.004765	1
3	23.7225	26.1250	493.750000	774.750000	0.004744	1
4	23.7540	26.2000	488.600000	779.000000	0.004767	1
...	...	...	...	...	...	...
20555	20.8150	27.7175	429.750000	1505.250000	0.004213	1
20556	20.8650	27.7450	423.500000	1514.500000	0.004230	1
20557	20.8900	27.7450	423.500000	1521.500000	0.004237	1
20558	20.8900	28.0225	418.750000	1632.000000	0.004279	1
20559	21.0000	28.1000	409.000000	1864.000000	0.004321	1

20560 rows × 6 columns

Figura 2. Vista del dataset a emplear en nuestro modelo predictor de ocupación de habitantes

Como puede observarse en la Figura 2, se ha eliminado el timestamp ya que su valor no aporta información al modelo y podría causar una fuga de información entre los datos de validación y los datos de entrenamiento.

2 Explicación teórica y descripción de los modelos a emplear

En nuestro caso de estudio se emplearon dos modelos predictivos, a saber: regresión logística simple [4] y Random Forest [5].

2.1 Descripción teórica del problema

Formalmente se tiene una variable dependiente binaria Y con un conjunto de variables independientes predictoras X. Se busca una función f(x) que prediga el valor de Y, minimizando la probabilidad de clasificación incorrecta. Así, la f(x) queda planteada de la siguiente forma:

$$f(x) = \begin{cases} 1 \; si \; P(Y = 1|X = x) \geq P(Y = 0|X = x) \\ 0 \; si \; P(Y = 0|X = x) \geq P(Y = 1|X = x) \end{cases}$$

Donde P(Y=1 | X=x) y P(Y=0 | X=x) son las probabilidades de que Y valga 1 y 0 respectivamente condicionadas a que X tiene el valor x.

2.2 Regresión logística simple

En este modelo se plantea (ver 1). Las funciones sigmoides son una familia de funciones que se caracteriza por ser acotadas, diferenciables, con derivada positiva en cada punto y con un único punto de inflexión (raíz de la segunda derivada). En general, se usa una función particular de esta familia que es la función logística L(x), (ver 2).

$$P(Y = 1|X = x) = sigmoide(\theta^T * x + \beta) \qquad \textbf{(1)} \qquad L(x) = \frac{1}{1+e^{-x}} \qquad \textbf{(2)}$$

En este modelo los parámetros entrenables, es decir que se pueden modificar para adaptar el modelo a los datos de estudio, son θ y β. La función de pérdida utilizada en el entrenamiento de este modelo, es decir la función que el modelo intenta explícitamente minimizar, es la función de entropía cruzada binaria (ver 3), estando f(x) definida en (4).

$$Perdida(\theta,\beta) = \frac{-1}{N}\sum_{i=1}^{N} y_i * \log\big(f(X_i)\big) + (1-y_i) * \log\left(1 - \mathrm{f(X_i)}\right) \quad (\mathbf{3})$$

$$f(x) = L(\theta^T * x + \beta) = \frac{1}{1 + e^{-(\theta^T * x + \beta)}} \quad (\mathbf{4})$$

De esta forma podemos definir un estimador de la probabilidad (ver 5)

$$\hat{P}(Y = 1|X = x) = f(x) = \frac{1}{1+e^{-(\theta^T * x + \beta)}} \quad \mathbf{(5)}$$

2.3 Árboles de decisión y bosque aleatorio (Random Forest)

Otro modelo que existe es el de Random Forest como evolución de los árboles de decisión. Un árbol de decisión se construye recursivamente, y en el caso del problema de clasificación el proceso es el siguiente:

1. Partiendo del conjunto de datos completo, se busca la variable predictora que de forma aislada mejor sirva para dividir los datos entre una y otra categoría. Es decir, la variable X_i que permita establecer un valor de corte V_i tal que usando como criterio para clasificación de $x_i < V_i$ o $x_i > V_i$, ocurran la menor cantidad de malas asignaciones posibles.
2. Posteriormente, se divide el conjunto de *datos* entre los clasificados por este criterio como 0 (cero) y los clasificados como 1 (uno).
3. Luego, con cada uno de estos subconjuntos se construyen sub árboles de decisión que serán los hijos del nodo actual. Conectados cada uno por una arista que representa el resultado de la clasificación en el nodo raíz.
4. Si el conjunto de datos considerado está compuesto por nodos de una única categoría, el proceso termina y ése es un nodo hoja del árbol.

El método de Random Forest consiste en construir un gran número de árboles de decisión utilizando para cada uno subconjuntos aleatorios de las observaciones y de las variables predictoras. Para evaluarse en un caso concreto, el Random Forest evalúa todos los árboles de decisión que lo componen y se queda con la categoría que obtiene la mayoría de votos entre los árboles de decisión. El crear muchos árboles independientes tiene por objetivo permitir que cada uno detecte aspectos distintos del problema, y a su vez evitar el sobreajuste (overfitting). Este modelo se diferencia de los otros porque llega directamente al paso predictivo sin pasar por una estimación de las funciones de probabilidad condicional. El módulo TensorFlow Random Forest [6] implementa este modelo de una forma sencilla y fácil de utilizar.

3 Etapas a desarrollar

A continuación enunciamos las diferentes etapas por las que deberá transitar nuestro caso de estudio. Para todo trabajo que busque evaluar un modelo predictivo es necesario tener al menos dos, pero se recomiendan tres, subconjuntos de los datos originales.

3.1 División del dataset

Se dividirán los datos de entrada en tres subconjuntos elegidos aleatoriamente:

- Datos de entrenamiento: 60% del dataset original (12336 observaciones)
- Datos de validación cruzada: 20% del dataset original (4112 observaciones)
- Datos de prueba: 20% del dataset original (4112 observaciones)

Los datos de entrenamiento serán utilizados para entrenar los modelos y los datos de validación cruzada se utilizarán para comparar los modelos entre si en cada uno de los siguientes escenarios planteados a continuación. Se reservaron datos para una etapa de prueba como parte de las buenas practicas en ciencia de datos pero finalmente no fueron utilizados.

3.2 Análisis exploratorio de datos

Antes de entrenar y evaluar los modelos se realizará un análisis manual de los datos de entrenamiento para observar posibles puntos importantes en los datos.

3.3 Supresión de variables

En esta sección se busca simular que un subconjunto de los sensores no está disponible, y se entrenarán y evaluarán los modelos utilizando el resto de las variables predictoras (columnas del dataset). Al haber 5 variables predictoras hay 31 subconjuntos no vacíos de variables predictoras.

3.4 Incorporación de ruido

En esta sección se simula que hay defectos en las mediciones de los sensores pero que el sistema no nota éstos. Por lo tanto, se entrenan los modelos una única vez con los datos de entrenamiento originales. Posteriormente, para cada uno de los 31 subconjuntos no nulos de variables y para distintas intensidades de error (25%, 50%, 75% y 100%) se reemplazará a cada observación afectada por un promedio ponderado entre el valor original y una distribución normal cuya media y desvío estándar coinciden con los que tiene dicha variable predictora en el dataset de entrenamiento.

4 Desarrollo de la experiencia

4.1 Descripción de la plataforma de trabajo a emplear

Se empleará para este caso de estudio el editor de código Vscode. La versión de Python empleada fue la 3.11, creándose un entorno virtual para tal fin, con la finalidad de

establecer correctamente las dependencias de los paquetes de librería instalados para python. Se utilizó una laptop con 32 GB de memoria RAM, micro Intel core I7 de octava generación. Las extensiones principales necesarias a instalar fueron jupyter notebook y WSL.

Se pueden ver más detalles de la implementación en el repositorio del proyecto [17]

4.2 Importación de módulos y datos

En primer lugar para la realización del estudio es necesario importar las siguientes librerías de código abierto: pandas (manejo de datasets) [7], numpy (manipulación numérica en Python) [8], tensorflow (librería para aprendizaje automático) [9], tensorflow_decision_forests (modelos de árboles de decisión) [10], seaborn (manipulación de gráficos) [11], matplotlib (Idem seaborn) [12], sklearn (análisis y minería de datos) [13]. Otra cosa importante que debe realizarse es fijar una semilla para que todas las ejecuciones del notebook sean iguales, tal como se observa en la Figura 3.

```
seed = 2024

# Establecer la semilla para la generación de números aleatorios en Python
random.seed(seed)

# Establecer la semilla para numpy
np.random.seed(seed)

# Establecer la semilla para tensorflow
tf.random.set_seed(seed)
```

Figura 3. Establecimiento de una semilla para la generación de números aleatorios en Python

4.3 Análisis exploratorio de los datos

La primera etapa, antes de definir un modelo, es realizar un análisis exploratorio de los datos. A continuación se muestra un gráfico muy importante que toma los datos de entrenamiento y permite ver la distribución de cada variable para los casos positivos y los negativos. Observar a continuación la Figura 4.

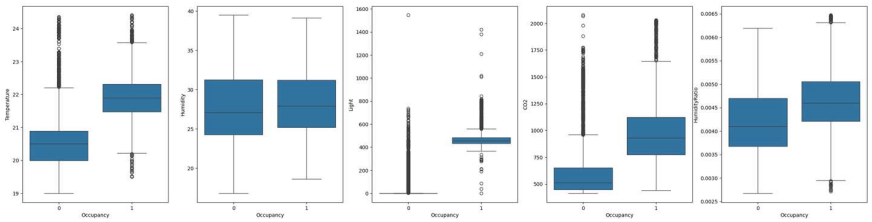

Figura 4. Distribución de las variables predictoras tomando los datos de entrenamiento

De cada variable se observa lo siguiente:

- Temperatura: Parece que está más cálido en los momentos en los que hay gente.
- La humedad no muestra grandes diferencias
- La luz tiene distribuciones completamente distintas en ambos casos. Hay mucha más gente en los momentos que hay luz.
- El CO2 también tiene una aparente correlación positiva con la presencia de personas.
- El ratio de humedad tampoco muestra diferencias tan grandes, aunque más notorias que en el nivel absoluto de humedad.

El problema que puede haber es que varios de estos factores en vez de ser consecuencia y algo que permita detectar si hay personas, sean en realidad una causa de que la gente asista o una consecuencia de una causa de que la gente asista, por ejemplo, de la hora (hay más calor y luz de día que de noche).

5 Resultados obtenidos

A continuación se muestran los diferentes modelos predictivos y sus comparaciones

5.1 Regresión logística simple

En los siguientes gráficos podemos ver como mejoran las métricas (pérdida y precisión) sobre los datos (de entrenamiento y de validación) a lo largo de las iteraciones.

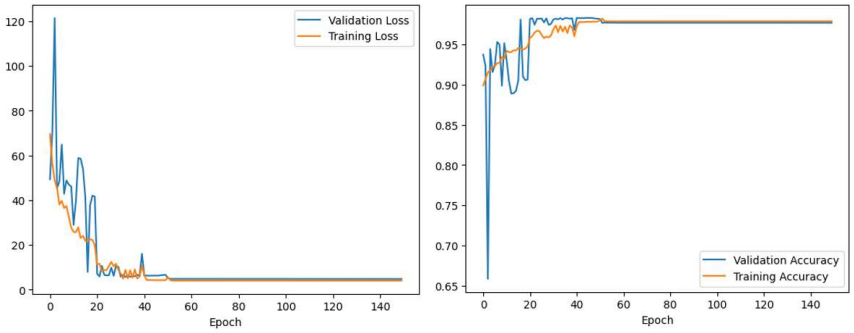

Figura 5. Vista de la Pérdida y Precisión sobre los datos de entrenamiento y de validación

5.2 Entrenamiento del modelo empleando Random Forest

Se obtuvieron los siguientes resultados para los valores de entrenamiento y de validación, ver Figura 6.

```
Use /tmp/tmpfebf4oer as temporary training directory
Reading training dataset...
Training dataset read in 0:00:02.915939. Found 12336 examples.
Training model...
Model trained in 0:00:00.772482
Compiling model...
[INFO 24-07-30 21:07:59.9026 -03 kernel.cc:1233] Loading model from path /tmp/tmpfebf4oer/model/ with prefix 37857de78bf14f20
[INFO 24-07-30 21:07:59.9495 -03 decision_forest.cc:734] Model loaded with 300 root(s), 39520 node(s), and 5 input feature(s).
[INFO 24-07-30 21:07:59.9495 -03 abstract_model.cc:1344] Engine "RandomForestOptPred" built
[INFO 24-07-30 21:07:59.9495 -03 kernel.cc:1061] Use fast generic engine
Model compiled.
13/13 [==============================] - 0s 7ms/step
Random Forest model created, Train Accuracy: 0.99660
5/5 [==============================] - 0s 6ms/step
Random Forest model created, Val Accuracy: 0.99003
```

Figura 6. Resultados de la precisión de la ocupación para datos de training y validación

5.3 Comparativa de la performance de ambos modelos (sin la incorporación de ruido en los sensores)

En el siguiente gráfico se puede comparar la performance de ambos modelos predictivos, en donde cada punto corresponde a un subconjunto de las variables predictoras. Tal como puede observarse en el gráfico, en 30 de 31 subconjuntos el Random Forest se comporta mejor y en sólo uno se comporta igual que la Regresión logística. Como se observa, el random forest domina absolutamente a la regresión logística simple. Observar el gráfico de la figura 7.

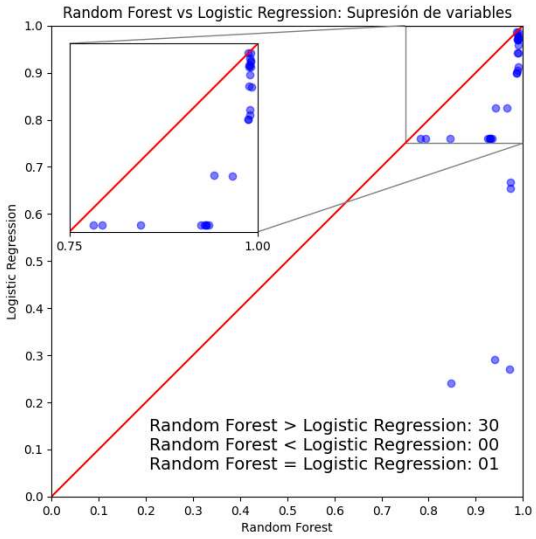

Figura 7. Comparativa de ambos modelos (sin la introducción de ruido en los sensores)

5.4 Incorporación de errores en los sensores

En el siguiente escenario compararemos los modelos pero en vez de deshabilitar algunos sensores (de una forma que es conocida por los modelos), lo que haremos será introducir errores en un subconjunto de los mismos (sin que los modelos sepan).

Al agregar el error, X_i pasa a valer:

$$(1 - \alpha) * X_i + \alpha * N\big(promedio(X), desvio\_standar(X)\big)$$

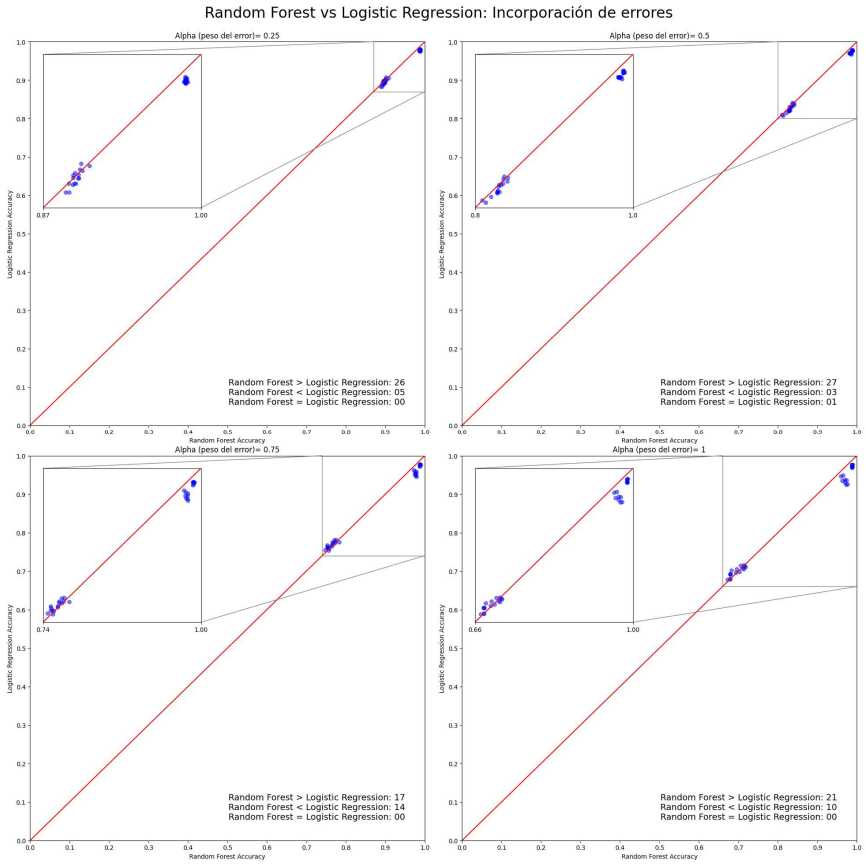

Figura 8. Comparativa de ambos modelos (con la introducción de ruido en los sensores)

Como podemos ver en la figura 8, en este escenario hay más subconjuntos donde la Regresión Logística supera al Random Forest, pero en todos los casos es una minoría de los subconjuntos.

6 Conclusiones y trabajo futuro

En este trabajo se pudo apreciar que el método de Random Forest es tanto más efectivo para la predicción de presencia de personas cómo más robusto a la hora de soportar errores, detectados o no, en los sensores del dispositivo de IoT.

En el futuro será posible buscar verificar estos mismos resultados para otros datasets o baterías de datasets, en pos de lograr la mayor generalidad posible de los mismos. Además, se puede explorar el uso de métodos de aprendizaje automático para la detección de fallas en los sensores.

Otro paso futuro será trasladar el modelo de Ramdom Forest obtenido en una plataforma con una computadora local empleando Jupyter Notebooks a un dispositivo de borde de prestaciones limitadas en cuanto a sus recursos de hardware, como ser un Arduino, un ESP32, una Raspberry Pi, etc. Para ello será necesario el empleo de una librería especial derivada de Tensorflow denominada Tensorflow Lite [14]. También será necesario utilizar MicroPython [15] el cual deberá ser instalado en la memoria flash del dispositivo de borde. Además se necesitará contar con un IDE que permita transferir el código Python al dispositivo como así instalar en la memoria flash del dispositivo el intérprete MicroPython. Entre los IDE más populares se encuentra Thonny [16].

7 Referencias

1. Warden, P., & Situnayake, D. (2019). TinyML: Machine Learning with Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers. O'Reilly Media. D
2. Dataset, https://www.kaggle.com/datasets
3. Mark Lutz. O' REILLY. Programming Python: Powerful Object-Oriented Programming 4th Edición D
4. An Introduction to Statistical Learning: with Applications in R" by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani
5. Tree-based Machine Learning Algorithms: Decision Trees, Random Forests, and Boosting Edición Kindle D
6. Tensorflow Random Forests, https://www.tensorflow.org/decision_forests?hl=es-419
7. Pandas, https://pandas.pydata.org/docs
8. NumPy, https://numpy.org
9. Tensorflow, https://www.tensorflow.org/resources/libraries-extensions
10. Tensorflow decision forests, https://www.tensorflow.org/decision_forests/installation?hl=es-419
11. Seaborn, https://seaborn.pydata.org/
12. Matpoltlib, https://matplotlib.org/
13. Sklearn, https://scikit-learn.org/stable/
14. TinyML: Machine Learning with TensorFlow Lite on Arduino
15. Micropython, https://micropython.org/
16. Thonny, https://thonny.org/
17. Repositorio https://github.com/carlucho1/cacic2024.git