Modelo Cliente-Servidor sin Control de Estado para Aprendizaje Profundo de Datos en Dispositivos IoT Aplicados a Parámetros Ambientales

Ing. Javier A. Ouret y Luciano Parodi

UCA - Facultad de Ingeniería y Ciencias Agrarias.
Ingeniería en Informática - Cátedra Protocolos de Internet
javierouret@uca.edu.ar
luciano.parodi97@gmail.com

Resumen

El crecimiento exponencial de dispositivos IoT requiere de la investigación y desarrollo de nuevas arquitecturas para la gestión de protocolos de acceso a sensores, operaciones cliente servidor y el análisis de datos con múltiples parámetros relacionados y gran volumen. El objetivo de este trabajo es investigar y proponer un modelo cliente-servidor sin control de estado, con brokers MQTT y arquitectura REST, de acceso a sensores IoT para el aprendizaje profundo de datos ambientales. Los sensores son accesibles en tiempo real por medio de gateways GNSS (con acceso a redes celulares LTE-M1, WiFi mesh o Lorawan), son monitoreados y gestionados con protocolos SNMP/Netconf [15]. Se realizan correcciones con datos ambientales externos obtenidos por geolocalización de estaciones meteorológicas. Utilizando K-NN, K-Means y GMM para el aprendizaje automático (supervisado y no supervisado) de los parámetros seleccionados se define el nivel de riesgo del lugar en base a la concentración de CO y CO2 del modelo. Con la información obtenida se pueden realizar acciones de corrección (o alarma) sobre otros dispositivos IoT para regular la ventilación del lugar y la capacidad operativa.

Palabras Clave: Internet of Things, IoT Sensor, Machine Learning, MQTT, RestAPI.

Introducción

La Internet de las Cosas (*IoT, Internet of Things) es un concepto que engloba a la interconexión digital de objetos cotidianos, sensores, software y otras tecnologías con internet, con objetos en lugar de con personas, permitiendo la gestión entre dispositivos y el ambiente para mejorar la calidad de vida de las personas. Se estima que hay más de 14 mil millones de dispositivos IoT* conectados en la actualidad, con una proyección de 27 mil millones para el 2025 [1]. En la actualidad son limitados los mecanismos unificados para acceder y configurar cualquier dispositivo de IoT en Internet y hay muchos temas a mejorar, como la falta de visibilidad y de integración a la seguridad de las redes, el uso de protocolos abiertos que no están totalmente probados, la inyección en la red de grandes volúmenes de datos o la vulnerabilidad de las APIs. Los protocolos existentes, como SNMP, están orientados a darnos información sobre

lo que ocurre en cada equipo y modificar parámetros, Protocolos como MQTT [2] de la norma OASIS [3] son usados para el transporte ligero de mensajes bajo el esquema de publicación/suscripción, utilizando poco código y un mínimo de ancho de banda. El proceso de automatización para la captura y agrupación (clustering) de los datos de los dispositivos IoT puede usar técnicas de aprendizaje profundo supervisado y no supervisado.

Descripción del informe

El presente informe tiene como finalidad mostrar de forma detallada los avances y aportes realizados en el marco de la investigación. La versión resumida se encuentra dentro de la sección número 4 del informe: “Ejecución del modelo cliente-servidor y análisis de mediciones”.

Los aportes se dividen en tres secciones: almacenamiento, presentación y análisis y clasificación de los datos provenientes de un sensor.

Materias incluidas: Programación Web – Base de Datos – Materias de Programación/ Informática (aportes generales para buenas prácticas de programación aplicadas a Python).

1- Almacenamiento

Para el almacenamiento de datos provenientes del sensor se diseñó una base de datos en MySQL. La misma será utilizada para facilitar la futura visualización, traslado y análisis de los datos extraídos.

En las imágenes de la Figura 1-A y 1-B se puede ver el código del archivo *datos_sensor.sql*, utilizado para la creación de la base de datos y la inserción de los datos en la misma.

```
--
-- Table structure for table `datos_sensor`
--

CREATE TABLE `datos_sensor` (
  `id` int(11) NOT NULL,
  `co2` decimal(20,2) DEFAULT NULL,
  `co2_corregido` decimal(20,2) NOT NULL,
  `temp` decimal(20,2) DEFAULT NULL,
  `hum` decimal(20,2) DEFAULT NULL,
  `fecha` text NOT NULL,
  `lugar` text NOT NULL,
  `altura` decimal(10,0) NOT NULL,
  `presion` decimal(8,2) NOT NULL,
  `presion_nm` decimal(8,2) NOT NULL,
  `temp_ext` decimal(20,2) NOT NULL,
  `temp_ref` decimal(8,2) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci;

--
-- Dumping data for table `datos_sensor`
--

INSERT INTO `datos_sensor` (`id`, `co2`, `co2_corregido`, `temp`, `hum`, `fecha`, `lugar`, `altura`, `presion`, `presion_nm`, `temp_ext`, `temp_ref`) VALUES
(1, '600.21', '646.14', '29.06', '59.79', '22-Dec-2022 (23:50:04.232289)', 'C480', '0', '954.00', '1013.21', '33.00', '298.15'),
(2, '597.52', '639.22', '29.06', '59.74', '22-Dec-2022 (23:50:04.232289)', 'C480', '0', '960.00', '1013.21', '23.00', '298.15'),
(3, '598.72', '623.70', '29.10', '59.79', '22-Dec-2022 (23:50:04.232289)', 'C480', '0', '986.00', '1013.21', '12.00', '298.15'),
(4, '593.85', '603.87', '29.07', '59.71', '22-Dec-2022 (23:50:04.232289)', 'C480', '0', '1010.00', '1013.21', '20.00', '298.15'),
(5, '584.37', '573.29', '29.10', '59.78', '22-Dec-2022 (23:50:04.232289)', 'C480', '0', '1047.00', '1013.21', '25.00', '298.15'),
(6, '566.56', '611.90', '29.09', '59.62', '22-Dec-2022 (23:50:04.232289)', 'C480', '0', '951.00', '1013.21', '16.00', '298.15'),
(7, '563.09', '583.06', '29.11', '59.55', '22-Dec-2022 (23:50:04.232289)', 'C480', '0', '992.00', '1013.21', '28.00', '298.15'),
(8, '562.21', '563.92', '29.09', '59.60', '22-Dec-2022 (23:50:04.232289)', 'C480', '0', '1024.00', '1013.21', '12.00', '298.15'),
(9, '562.30', '585.07', '29.05', '59.66', '22-Dec-2022 (23:50:04.232289)', 'C480', '0', '987.00', '1013.21', '24.00', '298.15'),
(10, '567.84', '577.34', '29.03', '59.59', '22-Dec-2022 (23:50:04.232289)', 'C480', '0', '1010.00', '1013.21', '29.00', '298.15'),
(11, '577.30', '595.95', '29.10', '59.54', '22-Dec-2022 (23:50:04.232289)', 'C480', '0', '995.00', '1013.21', '26.00', '298.15'),
(12, '576.71', '573.94', '29.07', '59.52', '22-Dec-2022 (23:50:09.074721)', 'C480', '0', '1032.00', '1013.21', '27.00', '298.15'),
```

Fig. 1-A. Código del archivo 'datos_sensor.sql'

```
--
-- Indexes for dumped tables
--


--
-- Indexes for table `datos_sensor`
--
ALTER TABLE `datos_sensor`
  ADD PRIMARY KEY (`id`);


--
-- AUTO_INCREMENT for dumped tables
--


--
-- AUTO_INCREMENT for table `datos_sensor`
--
ALTER TABLE `datos_sensor`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=21774;
COMMIT;
```

Fig. 1-B. Código del archivo 'datos_sensor.sql'

La base de datos cuenta con las siguientes columnas:

- Id: un valor entero que identifica individualmente cada fila. Se incrementa automáticamente al agregar un nuevo valor a la base de datos.
- Co2: Valor de c02 extraído del sensor.
- C02_corregido: Valor de c02 corregido. Este campo se calcula en base a otros datos extraídos del sensor que explicaremos más adelante.
- Temp: Valor de temperatura extraído del sensor.
- Hum: Valor de humedad extraído del sensor.
- Fecha: Fecha y hora en la que se realizó la medición de los datos del sensor.
- Lugar: Lugar donde se extrajeron los datos.
- Altura: Altura respecto al nivel del mar del lugar donde se extrajeron los datos.
- Presión: Valor de presión extraído del sensor.
- Presion_nm: Presión barométrica de referencia (1013,21 hPa a nivel del mar) utilizada para el cálculo del c02 corregido.
- Temp_ext: Valor de temperatura externa del lugar en donde se midieron los datos.
- Temp_ref: Temperatura de referencia (generalmente 25°C, convertida a absoluta 298,15 para °C) utilizadas para el cálculo del c02 corregido.

2- Presentación

Para mejorar la visualización de los datos del sensor, se desarrolló una página web mediante Node.js. En ella, se presenta una tabla que muestra todos los datos previamente guardados en la base de datos.

```
var mysql = require('mysql')
var connection = mysql.createConnection({
  host: '127.0.0.1',
  user: 'root',
  password: [redacted]
  database: 'datos_sensor'

})
connection.connect((err) => {
  if (err) {
    console.log(err)
    return
  }
  console.log('Database connected')
})
module.exports = connection
```

Fig. 2. Código del archivo 'database.js'

El código de la Figura 2 se encarga de establecer una conexión a una base de datos y exportar esa conexión para ser utilizada en otros archivos o módulos.

Funcionamiento del código:

- connection.connect((err) => {...}): Se llama al método **connect** de un objeto **connection**. El método **connect** toma una función de devolución de llamada como argumento. Esta función de devolución de llamada se ejecuta una vez que se intenta establecer la conexión a la base de datos, ya sea que la conexión tenga éxito o falle.
  Si hay un error se ejecutará el código dentro de la condición **if**. En caso contrario, si la conexión tuvo éxito se continuará con el resto del código.
- module.exports = connection: Al final del código, se exporta el objeto **connection** para que pueda ser utilizado en otros archivos o módulos que requieran esta conexión a la base de datos. Esto permite reutilizar la conexión en diferentes partes de la aplicación.

```
// Importación de módulos
var express = require('express')
var path = require('path')
var createError = require('http-errors')
var cors = require('cors')
var bodyParser = require('body-parser')
var app = express()

// Configuración de recursos estáticos
app.use(express.static(__dirname+'/'))

// Conexión a la base de datos MySQL
var dbMySQLNode = require('./database')

// Configuración del motor de plantillas y directorio de vistas
app.set('views', path.join(__dirname, '/'))
app.set('view engine', 'ejs')

// Configuración de análisis de solicitudes HTTP
app.use(bodyParser.json())
app.use(
  bodyParser.urlencoded({
    extended: true,
  }),
)

// Configuración de CORS
app.use(cors())

// Ruta principal
app.get('/', (req, res) => {
  res.render('index')
})

// Ruta para obtener datos desde la base de datos
app.get('/fetch', function (req, res) {
  dbMySQLNode.query('SELECT * FROM sensor ORDER BY id ', function (
    error,
    response,
  ) {
    if (error) {
      res.json({
        msg: error,
      })
    } else {
      res.json({
        msg: 'Data successfully fetched',
        sensor: response,
      })
    }
  })
})

// Inicio del servidor en el puerto 3000
app.listen(3000, function () {
  console.log('Node app is being served on port: 3000')
})
module.exports = app
```

Fig. 3. Código del archivo 'app.js'

Este código establece la configuración básica del servidor Express, incluyendo la gestión de archivos estáticos, la configuración del motor de plantillas y el análisis de los cuerpos de las solicitudes HTTP.

Funcionamiento del código:

1. Importación de módulos: Se importan los módulos necesarios para el funcionamiento del servidor, como Express, path, createError, cors y bodyParser.
2. Configuración de recursos estáticos: Se indica la ubicación de los recursos estáticos (archivos HTML, CSS, imágenes, etc.) que serán servidos por el servidor.
3. Conexión a la base de datos MySQL: Se establece la conexión con la base de datos MySQL utilizando el módulo "database".
4. Configuración del motor de plantillas y directorio de vistas: Se configura el motor de plantillas EJS y se especifica la ubicación de las vistas.
5. Configuración de análisis de solicitudes HTTP: Se configura el análisis de las solicitudes HTTP entrantes para poder procesar los datos enviados en el cuerpo de la solicitud.
6. Configuración de CORS: Se habilita el soporte para CORS (Cross-Origin Resource Sharing), lo cual permite que el servidor responda a solicitudes de dominios diferentes al dominio del servidor.
7. Ruta principal: Se define una ruta principal ("/") que renderiza la vista "index" al recibir una solicitud GET.
8. Ruta para obtener datos desde la base de datos: Se define una ruta ("/fetch") que realiza una consulta a la base de datos MySQL para obtener los datos de la tabla "sensor" y los devuelve como respuesta en formato JSON.
9. Inicio del servidor: Se inicia el servidor en el puerto 3000 y se muestra un mensaje en la consola indicando el puerto en el que se está ejecutando.
10. Exportación del módulo: Se exporta la aplicación Express para que pueda ser utilizada en otros archivos.

Estructura de archivos utilizada:

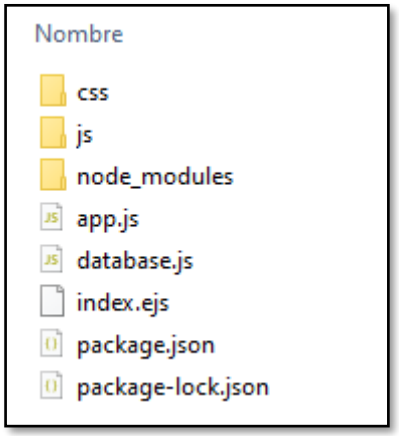

Fig. 4. Estructura de la carpeta de archivos

- Las carpetas **css** y **js** contienen los archivos de CSS y Javascript utilizados para darle formato al archivo **index.ejs**
- La carpeta **node_modules** es el directorio donde se almacenan las dependencias del proyecto. Cuando se instalan las dependencias del proyecto utilizando el comando **npm install**, Node.js crea automáticamente la carpeta **node_modules** y descarga e instala las bibliotecas y módulos de terceros especificados en el archivo package.json.
- El archivo **index.ejs** contiene el cuerpo de la página web. La extensión "EJS" significa "Embedded JavaScript", ya que permite incrustar código JavaScript en el archivo de plantilla. Los archivos **.ejs** contienen marcadores o etiquetas especiales que se utilizan para generar contenido dinámico.
  En este caso es un archivo de una tabla simple en HTML con datos insertados desde la base de datos.
- El archivo **package.json** define las dependencias y metadatos del proyecto, mientras que el archivo **package-lock.json** asegura una instalación consistente y determinista de las dependencias.

Vista final de la página web:

Fig. 5. Página web con la tabla de datos

Nota: El código desarrollado y detalle de los prototipos está disponible en: https://github.com/jaouret/IoT01 . Enviar email a javierouret@uca.edu.ar para reci-bir el permiso de lectura.

3- Análisis y clasificación

El siguiente objetivo de la investigación es el de realizar minería de datos de las redes de sensores operativas utilizando el aprendizaje automático de los parámetros previamente seleccionados. Para lograrlo se utilizaron tres algoritmos en Python:

- K-Nearest-Neighbor (KNN): clasificador de aprendizaje supervisado no paramétrico. Utiliza la proximidad para hacer clasificaciones o predicciones sobre la agrupación de un punto de datos individual. Puede usarse para clasificar nuevas muestras (valores discretos) o para predecir (regresión, valores continuos).

- K-Means: algoritmo de clasificación no supervisada (clusterización) que agrupa objetos en "k grupos" basándose en sus características. El agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el centroide de su grupo o cluster.

- Gaussian Mixture Model (GMM): modelo probabilístico que considera que las observaciones sigan una distribución probabilística formada por la combinación de múltiples distribuciones normales. Puede entenderse como una generalización de K-Means - que en lugar de asignar cada observación a un único cluster, obtiene una distribución de probabilidad de pertenencia a cada uno.

Cada algoritmo utilizará 20.000 valores extraídos del sensor y almacenados en la base de datos. Los resultados serán graficados para visualizar la distribución de los datos de C02 corregido y la separación de las clases o clústeres obtenidas. Como resultado, sabremos qué algoritmo o configuración será más apropiada para trabajar con los datos extraídos del sensor a futuro.

Los atributos a tener en cuenta a la hora de graficar los resultados de los algoritmos serán: la concentración de CO2, temperatura y humedad. Los gráficos 2D contienen como variable la temperatura medida para cada valor de C02 y los gráficos 3D, además de la temperatura, contienen la variable humedad para cada valor de C02.

La clasificación de concentración de C02 que vamos a tomar como referencia es la siguiente:

CONCENTRACIÓN	EFECTO
200 - 450 ppm	Concentración atmosférica típica
600 - 800 ppm	Calidad del aire interno aceptable
900-1200 ppm	Calidad del aire interno tolerable
2000- 5000 ppm	Límite promedio de exposición en un período de ocho horas
6000 - 30000 ppm	Preocupación, solo exposición breve
3 - 8%	Incremento de la frecuencia respiratoria, dolor de cabeza
> 10%	Náuseas, vómitos, pérdida de conocimiento
> 20%	Pérdida de conocimiento repentina, muerte

Tabla 1. Efectos de la concentración de CO2 Una elevada concentración conduce a la asfixia por desplazamiento de oxígeno. [9][10]

Encabezado y cuerpo del código

Al inicio de nuestro código en Python importamos las librerías necesarias para utilizar todos los algoritmos propuestos.

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from matplotlib.colors import ListedColormap
from matplotlib import colors
import matplotlib.pyplot as plt
import mysql.connector

# Declaración de datos y variables
X = []
y = []
grafico_3d = []
conteo=0
valor_KNN=3      #valor de KNN
valor_KMEANS=3   #valor de K-Means (rango de colores aplicado de 3 a 6)
valor_GMM=3      #valor de GMM
N=20000          #cantidad de datos a utilizar
```

Fig. 6. Librerías importadas en Python y declaración de variables [4] [5] [6] [7] [8]

Para graficar los datos de los algoritmos accedimos nuevamente a los datos almacenados dentro de la base de datos como se muestra en la Figura 7 a continuación. De todos los datos extraídos, utilizamos los valores de C02, Temperatura y Presión para el cálculo de C02 corregido mediante la siguiente fórmula:

$$\text{CO2 corregido (ppm)} = \text{CO2 medido(ppm)} \times \left(\frac{\text{Temp medida (°K)} \times \text{P referencia (Hpa)}}{\text{Presión medida (Hpa)} \times \text{Temp referencia (°K)}} \right)$$

Fórmula 1. Cálculo del C02 corregido

```python
# Credenciales de conexión a la base de datos
config = {
    'host': '127.0.0.1',
    'user': 'root',
    'password': [redacted]
    'database': 'datos_sensor'
}

# Conexión a la base de datos
cnx = mysql.connector.connect(**config)
cursor = cnx.cursor()

# Consulta para extraer los datos necesarios de la tabla
query = "SELECT id, co2, temp, hum, presion FROM sensor"
cursor.execute(query)

# Recorrer los resultados de la consulta
for (id, co2, temp, hum, presion) in cursor:

    if presion != 0:
        #-----Datos extraidos-----
        Tactual = float(temp)           #°C - Temperatura extraida por el sensor en °C
        Tmedida= Tactual + 273.15       #°K - Tmedida = temperatura absoluta actual, °C +
        ppmC02medido = float(co2)       #ppm - Valor de ppm C02 medido por el sensor
        Pmedida = float(presion)        #hPa - pmedida = Presión actual, en las mismas uni
        Humedad = float(hum)            #g/m^3 - Humedad extraida por el sensor en gramos

        #-----Datos de referencia-----
        Tref = 298.15               #°K // Tref = temperatura de referencia, generalmente
        Pref = 1013.207             #hPa // pref = presión barométrica de referencia, norm

        #-----Cálculos-----
        ppmC02corregido = ppmC02medido * ((Tmedida*Pref)/(Pmedida*Tref))
        #print(f"ID: {id}, Temperatura: {Tmedida}, Cálculo: {round(ppmC02corregido)}")

        # Agregar los datos calculados al arreglo X y al arreglo grafico_3d
        X.append([ppmC02corregido,Tmedida])
        grafico_3d.append([ppmC02corregido,Tmedida,Humedad])
```

Fig. 7. Código para la extracción de datos de la Base de Datos en Python

KNN

Para poder aplicar KNN lo primero que debemos hacer es agregarles etiquetas a los valores obtenidos de C02 corregido. En este caso vamos a utilizar números del 0 al 5 para etiquetar los diferentes rangos de concentraciones de C02 como se muestra a continuación:

```
# Agregar los datos calculados al arreglo X
X.append([ppmC02corregido,Tmedida])

# Agregar las etiquetas de los valores calculados al arreglo Y
if 0 <= ppmC02corregido < 500:
    y.append(0)
elif 500 <= ppmC02corregido < 700:
    y.append(1)
elif 700 <= ppmC02corregido < 1000:
    y.append(2)
elif 1000 <= ppmC02corregido < 2500:
    y.append(3)
elif 2500 <= ppmC02corregido < 5000:
    y.append(4)
else:
    y.append(5)
    print("Guardando valor de C02 ¡MUY ALTO!", ppmC02corregido)
```

Fig. 8. Etiquetado de valores de C02 corregido

```
#---------------Cálculos de KNN y su gráfico 2D---------------

# Conversor de la matriz a un array Numpy
X = np.array(X)
y=np.array(y)

# Crea y entrena el clasificador KNN
print("Calculando KNN...")
knn = KNeighborsClassifier(n_neighbors=valor_KNN)
knn.fit(X, y)

x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))

# Predecir la clase de todos los puntos de la malla
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Grafica los colores de la región de decisión del clasificador KNN
plt.figure()
plt.pcolormesh(yy, xx, Z, cmap='coolwarm')

# Grafica los puntos
plt.scatter(X[:, 1], X[:, 0], c=y, edgecolor='k', cmap='coolwarm')
plt.ylabel('C02 Corregido ppm')
plt.xlabel('Temperatura °K')
plt.title(f'Clasificador KNN {valor_KNN}')

plt.show()
```

Fig. 9. Código utilizado para realizar el gráfico básico del algoritmo KNN

KNN, al ser un algoritmo de aprendizaje supervisado, requiere ejemplos de datos de entrenamiento con sus respectivas etiquetas como se muestra en la Figura 8. Durante la fase de entrenamiento, el algoritmo KNN aprende los patrones y relaciones entre los datos de entrada y sus etiquetas correspondientes.

La Figura 9 muestra el código utilizado para graficar los datos y las etiquetas utilizando el algoritmo de KNN.

Funcionamiento del código:

1. "Conversor de la matriz a un array Numpy": Se utiliza la función **np.array()** de la biblioteca NumPy para convertir las variables **X** e **Y** en matrices en formato NumPy. Esto permite utilizar las funciones y operaciones proporcionadas por NumPy en lugar de las estructuras de datos estándar de Python.

2. "Crea y entrena el clasificador KNN": Se crea una instancia del clasificador K-Nearest Neighbors (KNN) utilizando **KNeighborsClassifier()** del módulo **neighbors** de la biblioteca scikit-learn. El parámetro **n_neighbors** indica el número de vecinos más cercanos que se tendrán en cuenta en la clasificación. Luego, el clasificador se entrena con los datos **X** e **y** utilizando el método **fit()**.

   Se calculan los límites mínimo y máximo para los ejes x e y basados en los datos en **X**. Se agregan pequeños valores de desplazamiento (+0.1 y -0.1) para asegurarse de que los puntos de datos no se encuentren exactamente en los límites del gráfico.

   Se utiliza la función **np.meshgrid()** para crear una malla de puntos en los rangos de los ejes x e y establecidos previamente. Esto se hace utilizando el método **np.linspace()** para generar una secuencia de valores equidistantes en el rango de cada eje.

3. "Predecir la clase de todos los puntos de la malla": Se utiliza el clasificador KNN entrenado (**knn**) para predecir la clase de todos los puntos en la malla creada. Se utiliza **np.c_ []** para combinar los valores de **xx** e **yy** en una matriz bidimensional de coordenadas (una para cada punto en la malla).

   La función **predict()** del clasificador KNN se utiliza para realizar las predicciones. Esta función realiza el cálculo de la distancia euclidiana para determinar la clase de los puntos de datos en la malla generada.

$$d_E(P,Q) = \sqrt{(p_1-q_1)^2 + (p_2-q_2)^2 + \cdots + (p_n-q_n)^2} = \sqrt{\sum_{i=1}^{n}(p_i-q_i)^2}.$$

Fórmula 2. Cálculo de la distancia euclidiana utilizado por el algoritmo de KNN [12]

Se ajusta la forma (**reshape()**) del arreglo de predicciones **Z** para que coincida con la forma de la malla **xx**. Esto es necesario para que los datos de predicción se puedan visualizar correctamente.

4. “Grafica los colores de la región de decisión del clasificador KNN”: **plt.figure()** crea una nueva figura para el gráfico. Una figura es el lienzo en el que se representarán los elementos gráficos, como los ejes, las líneas y las etiquetas.

   Se utiliza **plt.pcolormesh()** para graficar los colores de la región de decisión del clasificador KNN. La función **pcolormesh()** toma como argumentos las coordenadas **yy** y **xx** de la malla, y los valores de predicción **Z**. Se utiliza el mapa de colores 'coolwarm' para visualizar las clasificaciones.

5. "Grafica los puntos": Se utiliza **plt.scatter()** para graficar los puntos de datos en el gráfico. Los puntos se extraen de **X** utilizando indexación de matriz y se asigna un color (**c**) basado en los valores de **y**. El argumento **edgecolor='k'** establece el color de los bordes de los puntos en negro.

   Por último, se utiliza **plt.ylabel()**, **plt.xlabel()** y **plt.title()** para agregar etiquetas a los ejes y un título al gráfico.

En el algoritmo KNN el valor de K (representado por la variable **valor_KNN)**) representa el número de vecinos más cercanos tomados en cuenta para la predicción o clasificación. Es un parámetro que debe especificarse con un valor entero para indicar el número de vecinos más cercanos que se utilizarán en la clasificación.

Al aumentar el valor de K, se considerarán más vecinos en la decisión de clasificación, permitiendo suavizar las fronteras de decisión y reducir el efecto de puntos atípicos. Sin embargo, un valor de K demasiado grande puede llevar a un modelo más complejo y menos sensible a patrones locales en los datos.

La elección del valor óptimo de K depende del conjunto de datos y del problema específico. Para encontrar un valor óptimo fueron graficados 20.000 datos con diferentes valores de K con cada algoritmo.

Resultados obtenidos:

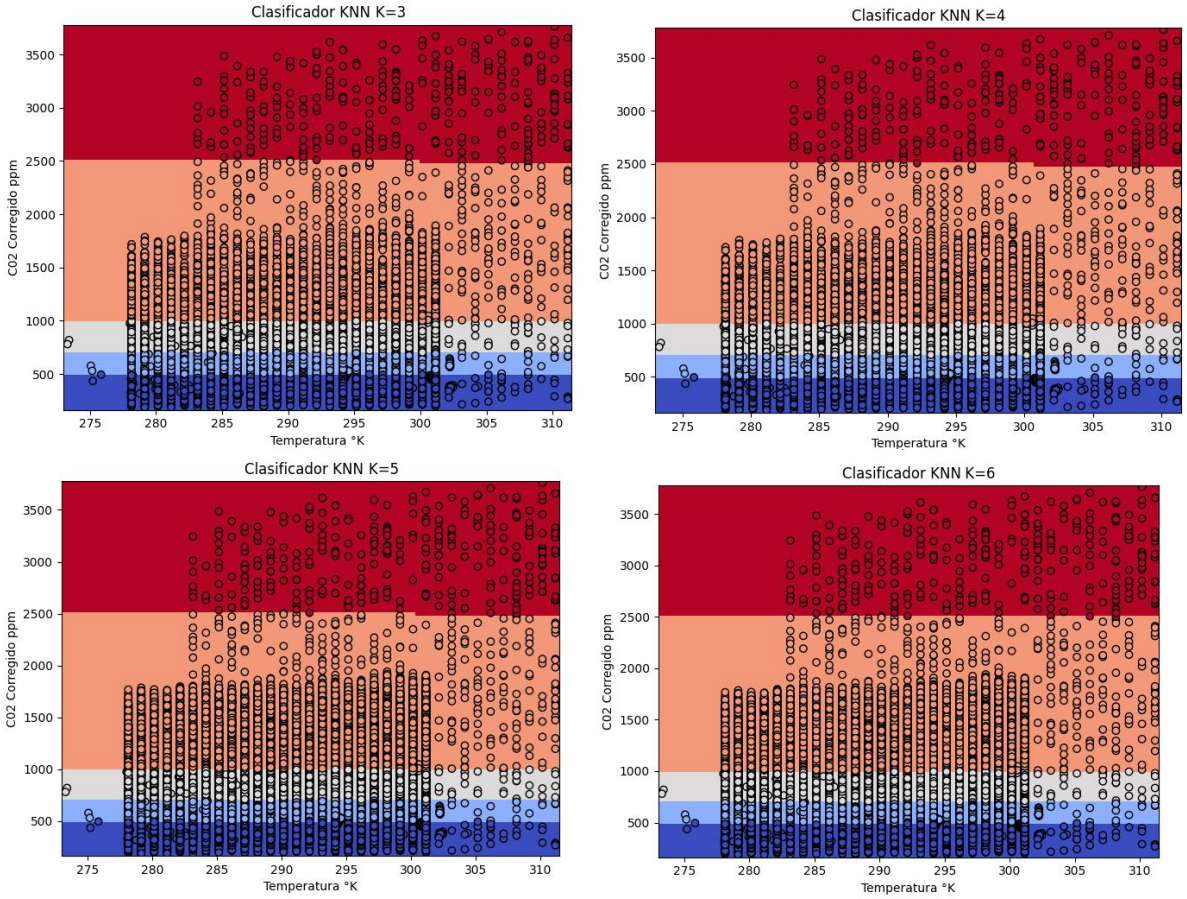

Fig. 10. Gráficos del algoritmo KNN utilizando valores de K= 3,4,5 y 6

El gráfico sobre la región de decisión incluye los colores representativos a las diferentes clases de puntos. Como resultado, se pueden distinguir las áreas donde se agrupan los puntos con distintas etiquetas, facilitando la clasificación visual de los datos. Esto permite: entender cómo se agrupan los datos, cómo se pueden separar, y también comparar diferentes clasificadores y parámetros.

Los diferentes valores de K, en este caso, no mostraron una gran diferencia ya que el 100% de los datos poseen etiquetas.

Para poner a prueba las predicciones de nuestro algoritmo dividiremos el conjunto de datos en “Datos de Prueba” y “Datos de entrenamiento” mediante la función **train_test_split()** de la librería **sklearn.model_selection** [16]. Luego calcularemos la precisión del modelo comparando las predicciones (**y_pred**) con las etiquetas reales (**y_test**) utilizando la función **accuracy_score()** de la librería **sklearn.metrics** [17]. A continuación, se presenta el código que refleja todos los pasos mencionados anteriormente:

```python
# Conversor de la matriz a un array Numpy
X = np.array(X)
y=np.array(y)

# Importar train_test_split de scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Dividimos los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.95, train_size=0.05)

# Crea y entrena el clasificador KNN
print("Calculando KNN...")
knn = KNeighborsClassifier(n_neighbors=valor_KNN)
knn.fit(X_train, y_train)

# Hacemos predicciones con los datos de prueba y evaluamos su precisión
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Precisión: {accuracy}')
print("Precisión del modelo:", knn.score(X_test, y_test))

# Definir los rangos de las variables x e y para crear una malla de puntos
x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))

# Predecir la clase de todos los puntos de la malla
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Grafica los puntos
plt.figure()
plt.pcolormesh(yy, xx, Z, cmap='coolwarm')
plt.scatter(X_test[y_test != y_pred, 1], X_test[y_test != y_pred, 0], c='k', marker='x', label='Mal clasificados')
plt.scatter(X_test[y_test == y_pred, 1], X_test[y_test == y_pred, 0], c=y_test[y_test == y_pred], edgecolor='k',
            cmap='coolwarm', marker='o', label='Clasificados correctamente')
plt.ylabel('C02 Corregido ppm')
plt.xlabel('Temperatura °K')
plt.title(f'Clasificador KNN {valor_KNN} (Puntuación: {accuracy:.4f})')
plt.show()
```

Fig. 11. Código utilizado para realizar el testeo del algoritmo KNN

Resultados obtenidos:

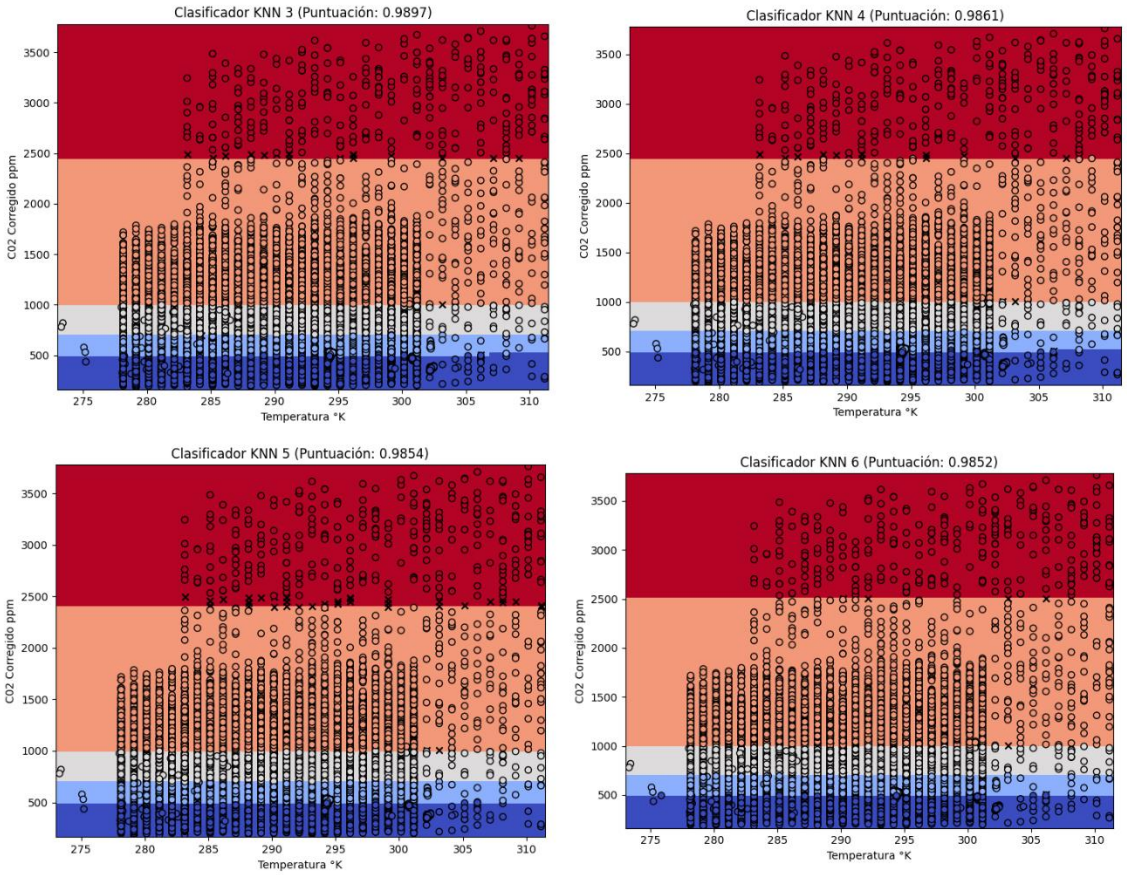

Fig. 12. Gráficos del testeo del algoritmo KNN en utilizando valores de K= 3,4,5 y 6

Utilizando 5% de los datos para entrenamiento y el 95% para prueba obtuvimos los siguientes valores de precisión:

- K=3 puntuación 0.9897
- K=4 puntuación 0.9861
- K=5 puntuación 0.9854
- K=6 puntuación 0.9852

Como se puede ver en los gráficos, los límites son muy parecidos al etiquetado normal, esto se ve reflejado en los altos resultados de precisión del modelo. Los errores de clasificación tienden a concentrarse en las proximidades de los límites de las regiones debido al enfoque de KNN, el cual utiliza los puntos cercanos como referencia para predecir las etiquetas.

KNN demuestra ser una herramienta valiosa en la clasificación de datos utilizando como base un conjunto de datos previamente etiquetada para su entrenamiento.

La precisión de KNN para el etiquetado de datos se ve influenciada por el tamaño y la distribución del conjunto de entrenamiento utilizado. Encontrar un tamaño y una distribución de datos ideal para el entrenamiento podría hacer que este algoritmo sea más preciso para esta investigación.

K-MEANS

Como fue explicado anteriormente, K-Means es un algoritmo de clasificación no supervisada que agrupa los datos en “k grupos” basándose en sus características. A diferencia de KNN, K-Means no necesita etiquetas en los datos de entrada. El algoritmo se enfoca en encontrar los centroides (puntos representativos) para los grupos formados en función de la similitud de las características de los puntos de datos.

En esta oportunidad utilizaremos nuevamente el arreglo X generado con los datos del sensor:

```
#---------------Cálculos de K-Means y su gráfico 2D---------------

# Conversor de la matriz a un array Numpy
X = np.array(X)

# Crea y entrena el algoritmo K-means
print("Calculando K-Means...")
kmeans = KMeans(n_clusters=valor_KMEANS, n_init=5, random_state=4)
kmeans.fit(X)

# Obtiene las etiquetas de los clústeres y los centroides
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Crea una malla para la visualización
x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))

# Predecir la clase de todos los puntos de la malla
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure()

#Grafica los colores de la región de decisión del algoritmo K-Means
plt.pcolormesh(yy, xx, Z, cmap='coolwarm')
plt.scatter(X[:, 1], X[:, 0], c=labels, edgecolor='k', cmap='coolwarm')

# Grafica los centroides de cada sección
plt.scatter(centroids[:, 1], centroids[:, 0], marker='*', s=300, c='black', edgecolor='k')

#Configura los títulos y etiquetas de los ejes
plt.xlabel('Temperatura °K')
plt.ylabel('C02 Corregido ppm')
plt.title('Algoritmo K-means')

plt.show()
```

Fig. 13. Código utilizado para realizar el gráfico 2D del algoritmo K-Means

Funcionamiento del código:

1. "Conversión de la matriz a un array Numpy": Nuevamente se utiliza la función **np.array()** de la biblioteca NumPy para convertir la matriz **X** en un array Numpy. Esto proporciona una estructura de datos eficiente para realizar operaciones numéricas en los datos.

2. "Crea y entrena el algoritmo K-Means": Se crea una instancia del algoritmo K-Means utilizando la clase **KMeans** de la biblioteca sklearn.cluster. Se especifica el número de clústeres mediante el parámetro proporcionado **n_clusters=valor_KMEANS** y un estado inicial dado por **n_init** y **random_state**.
   Se entrena el algoritmo K-Means utilizando el método **fit()** y los datos de entrada **X**.
   Los objetos se representan con vectores reales de d dimensiones (x1,x2,...,xn) y el algoritmo k-means mediante la función **fit()** construye **k** grupos donde se minimiza la suma de distancias de los objetos, dentro de cada grupo S={S1,S2,...,Sk}, a su centroide. El problema se puede formular de la siguiente forma:

$$\min_{\mathbf{S}} E(\boldsymbol{\mu}_i) = \min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

Fórmula 3. Cálculo de minimización de suma de distancias utilizado por el algoritmo de K-Means [13]

3. "Obtiene las etiquetas de los clústeres y los centroides": Se obtienen las etiquetas de los clústeres asignadas a cada punto de datos mediante el atributo **labels_** del objeto **kmeans**.

   Se obtienen los centroides de cada clúster mediante el atributo **cluster_centers_** del objeto **kmeans**.

4. "Crea una malla para la visualización": Similar al caso de KNN, se crea una malla utilizando la función **np.meshgrid()** de NumPy, generando una cuadrícula de puntos en los ejes x e y.

5. "Predecir la clase de todos los puntos de la malla": Se utiliza el algoritmo K-means entrenado (**kmeans**) para predecir la clase de todos los puntos en la malla generada mediante la función **predict()**.

6. Se da forma a los resultados para que coincidan con las dimensiones de la malla utilizando la función **reshape()**.

7. El resto de los pasos para graficar son similar al algoritmo de KNN. La única diferencia que se le agregan los centroides a cada gráfico. Los mismos tendrán forma de estrella y serán de color negro.

Los gráficos de K-Means proporcionan una representación visual de cómo los datos se agrupan en diferentes clústeres y cómo se distribuyen en el espacio de características. Esto ayuda a comprender mejor la estructura y separabilidad de los datos.

En cada gráfico se podrán distinguir diferentes cantidads de regiones en base al valor de K (**valor_KMEANS**) utilizado para cada uno.

Resultados obtenidos:

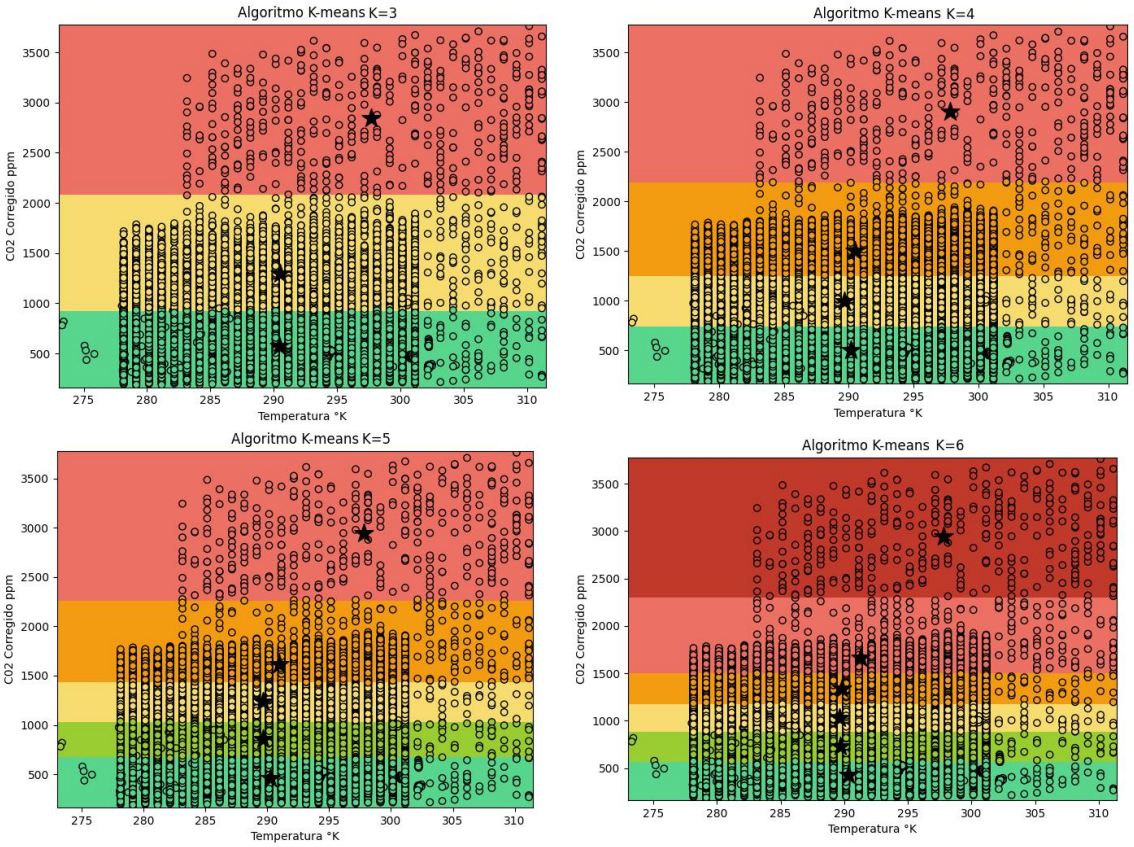

Fig. 14. Gráficos del algoritmo K-Means en 2D utilizando valores de K= 3,4,5 y 6

Los puntos que se encuentran dentro de una misma región de decisión se consideran similares en términos de características. Estas regiones se generan en base a la ubicación de los centroides y la asignación de puntos a los clústeres correspondientes.

Los gráficos resultantes se asemejan bastante a la clasificación de referencia que se encuentran en la Tabla 1, en especial los de K=5 y K=6. Esto se debe a la gran cantidad y variedad de datos utilizados.

K-MEANS en 3D

En un espacio bidimensional los datos se representan utilizando dos variables, permitiendo así una visualización más simple en un plano. A su vez, se pueden identificar y analizar patrones de agrupación en función de la relación entre estas dos variables.

Por otro lado, al utilizar un espacio tridimensional, se agrega una dimensión adicional a la representación de los datos. Para nuestro conjunto de datos vamos a utilizar las variables: temperatura, humedad y concentración de CO2 para identificar grupos o clusters utilizando K-Means 3D que no solo consideran la relación entre temperatura y el C02 corregido, sino también la influencia de la humedad en la formación de los grupos.

K-Means en 3D nos brinda una representación más completa y detallada de la estructura de los datos al considerar múltiples variables. Como consecuencia, esto puede revelar patrones y relaciones más complejas entre las variables y potencialmente llevar a una mejor comprensión de los datos y las agrupaciones encontradas.

```
#-----Cálculos de K-Means y su gráfico 3D-----

# Conversión de la lista de datos en una matriz Numpy para facilitar su manipulación
grafico_3d = np.array(grafico_3d)

# Creación de una figura y un eje 3D para la visualización
print("Calculando K-Means 3D...")
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Instanciación del algoritmo K-Means con el número de clusters especificado
kmeans = KMeans(n_clusters=valor_KMEANS, n_init=10, random_state=4).fit(grafico_3d)

# Obtención de los centroides de los clusters
centroids = kmeans.cluster_centers_

# Asignación de una etiqueta a cada punto según el cluster al que pertenece
labels = kmeans.labels_

#Grafica los puntos en 3D
ax.scatter(grafico_3d[:, 0], grafico_3d[:, 1], grafico_3d[:, 2], c=labels, cmap='coolwarm', edgecolor='k')

#Grafica los centroides en 3D
ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], marker='*', s=300,
c='black', edgecolor='k')

#Configura los títulos y etiquetas de los ejes
ax.set_xlabel('CO2 Corregido ppm')
ax.set_ylabel('Temperatura °K')
ax.set_zlabel('Humedad g/m^3')
ax.set_title('Algoritmo K-means en 3D')

plt.show()
```

Fig. 15. Código utilizado para realizar el gráfico 3D del algoritmo K-Means

Funcionamiento del código:

1. "Conversión de la lista de datos en una matriz Numpy para facilitar su manipulación": La lista de datos **grafico_3d** se convierte en una matriz Numpy mediante la función **np.array()** para facilitar su manipulación.
2. "Creación de una figura y un eje 3D para la visualización": Se crea una figura y un eje tridimensional utilizando la biblioteca **matplotlib**.
3. "Instanciación del algoritmo K-Means con el número de clusters especificado": Se instancia el algoritmo K-Means mediante la clase **KMeans** y se especifica el número de clusters (valor_KMEANS), el número de inicializaciones (n_init) y la semilla aleatoria (random_state=4). Luego, se ajusta el modelo K-Means a los datos **grafico_3d** mediante el método **fit()**.
4. "Obtención de los centroides de los clusters": Se obtienen los centroides de los clusters mediante el atributo **cluster_centers_** del modelo K-Means.
5. "Asignación de una etiqueta a cada punto según el cluster al que pertenece": Se asigna una etiqueta a cada punto en los datos según el cluster al que pertenece, utilizando el atributo **labels_** del modelo K-Means.
6. "Grafica los puntos y centroides en 3D": Los puntos se grafican en el espacio tridimensional utilizando la función **scatter()** del eje tridimensional. Se utiliza el array **grafico_3d** para especificar las coordenadas x, y, z de los puntos y las etiquetas **labels** para asignar colores a cada punto.

   Los centroides también pueden ser graficados descomentando la línea correspondiente. Se utilizan asteriscos como marcadores, se especifica el tamaño y el color de los centroides.
7. El resto de los pasos para graficar son similar al algoritmo de K-Means 2D.

Los gráficos de K-Means nos brindarán una representación visual de la estructura de agrupamiento de los datos y su distribución en el espacio de características. La cantidad de regiones distintas en cada gráfico depende del valor de K utilizado.

Resultados obtenidos:

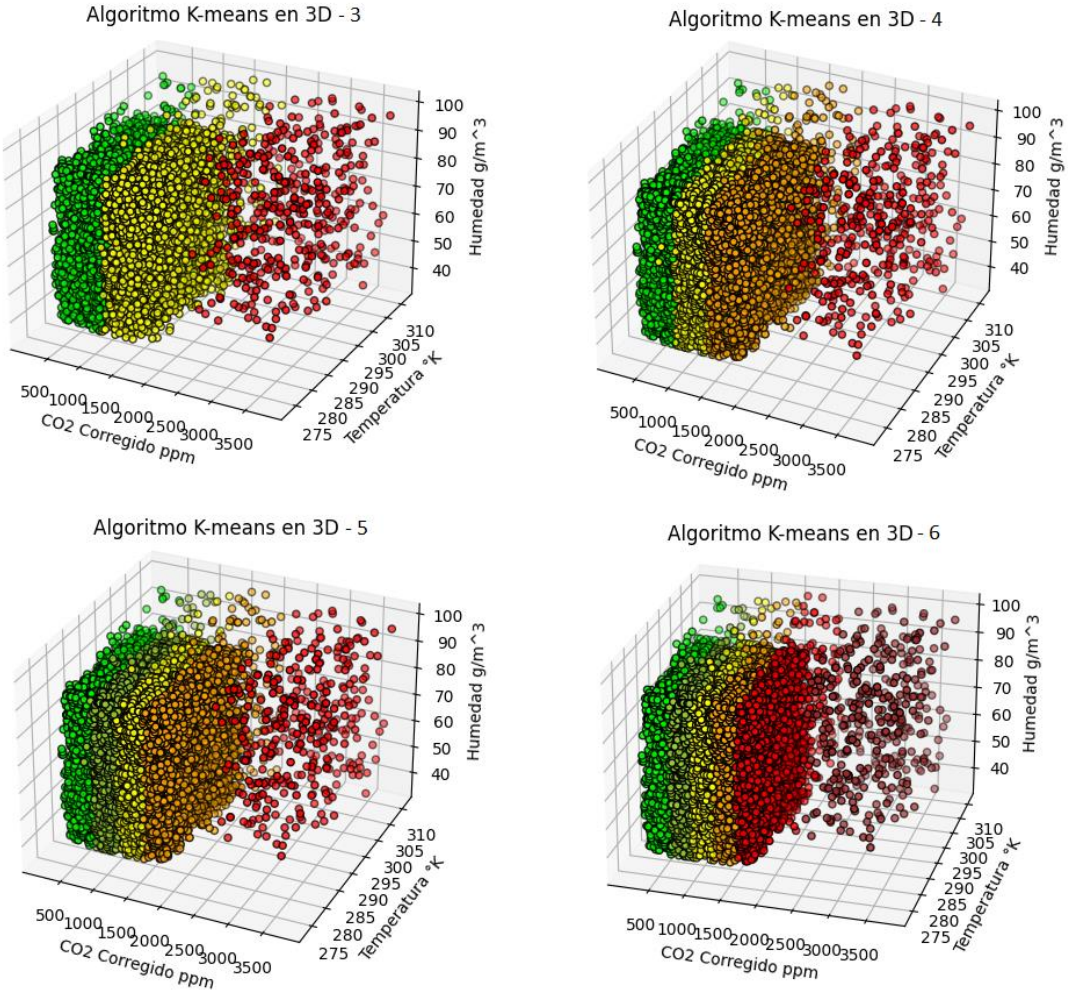

Fig. 16. Gráficos del algoritmo K-Means en 3D utilizando valores de K= 3,4,5 y 6

El resultado de las regiones obtenidas, tomando como referencia el valor de C02 corregido, es muy similar al de los gráficos con dos variables. Los resultados se asemejan nuevamente a la clasificación de referencia que se encuentran en la Tabla 1, en especial los de K=5 y K=6.

Tras haber analizado los gráficos, arribamos a la conclusión de que K-Means puede ser de gran utilidad a la hora de clasificar los datos obtenidos por el sensor, independientemente de la cantidad de variables utilizadas.

Es importante destacar que este algoritmo nos permitiría ahorrar tiempo y recursos al automatizar el proceso de etiquetado, en especial cuando se manejan grandes volúmenes de datos y se requiere de una clasificación rápida y precisa.

GMM

Por último, utilizaremos el algoritmo GMM (Gaussian Mixture Model). Este modelo es muy similar a K-Means, no obstante, posee marcadas diferencias en la forma en que los algoritmos abordan la agrupación de datos:

- K-Means: Este algoritmo busca asignar cada punto de datos al cluster más cercano en función de la distancia euclidiana. Utiliza centroides como representantes de los clusters y minimiza la suma de las distancias al cuadrado entre los puntos y sus centroides asignados.
- GMM: En cambio, GMM asume que los datos provienen de una mezcla de distribuciones gaussianas subyacentes. En lugar de asignar puntos a clusters discretos, GMM estima las distribuciones de probabilidad de cada cluster y calcula las probabilidades de pertenencia de cada punto a cada cluster. Utiliza el algoritmo de Expectation-Maximization (EM) para estimar los parámetros de las distribuciones gaussianas. [11]

Cabe destacar que la principal diferencia entre GMM y K-Means radica en la naturaleza de los clusters asignados. Mientras que K-Means produce clusters rígidos y con límites definidos, GMM permite que los puntos de datos tengan una pertenencia probabilística a múltiples clusters, lo que brinda mayor flexibilidad en la asignación. Esto nos proporcionara una perspectiva diferente de los datos que queremos estudiar en esta investigación.

```
#---------------Cálculos de GMM y su gráfico 2D---------------

# Crear instancia del modelo GMM y ajustar a los datos
print("Calculando GMM 2D...")
gmm = GaussianMixture(n_components=valor_GMM, random_state=4)
gmm.fit(X)

# Predecir la clase de cada punto en los datos
labels_gmm = gmm.predict(X)

# Graficar los datos con los colores basados en las etiquetas de clase predichas por GMM
plt.scatter(X[:,1], X[:,0], c=labels_gmm, cmap='jet')
plt.xlabel('C02 Corregido ppm')
plt.ylabel('Temperatura °K')
plt.title('Algoritmo GMM 2D')

plt.show()
```

Fig. 17. Código utilizado para realizar el gráfico 2D del algoritmo GMM

Funcionamiento del código:

1. “Crear instancia del modelo GMM y ajustar a los datos”: Se crea una instancia del modelo Gaussian Mixture Model (GMM) utilizando la clase **GaussianMixture** de la biblioteca scikit-learn. El primer parámetro

n_components indica el número de componentes o clústeres que se desea ajustar en el modelo. Luego, el modelo se ajusta a los datos de entrada X utilizando el método **fit(X)**.
El método **fit()** en el algoritmo GMM (Gaussian Mixture Model) utiliza el algoritmo de Expectation-Maximization (EM) para ajustar el modelo a los datos. El algoritmo EM es utilizado para estimar los parámetros del modelo, es decir, los pesos, las medias y las matrices de covarianza de las componentes gaussianas.

Esperanza (Expectation): Con los parámetros conocidos de las distribuciones normales y la probabilidad 'π' de pertenencia a un Cluster, asignar cada objeto al Cluster con el mayor valor de probabilidad de pertenencia:

$$argmax \sum_{k=1}^{K} \pi_k \cdot P_k(X|, \vec{\mu}, \vec{\sigma})$$

Fórmula 4. Cálculo de Esperanza utilizado por el algoritmo GMM [14]

Maximización (Maximization): Calcular con los objetos asignados a cada Cluster el valor de los parámetros (π_k, μ_k y σ_k):

$$\pi_k = \frac{Objetos\ en\ el\ Cluster}{Total\ Objetos}$$

$$\mu_k = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \quad y \quad \sigma_k = \sqrt{\frac{1}{n-1} \cdot \sum_{n=1}^{n} (x_i - \mu_k)^2}$$

Fórmula 5. Cálculo de Maximización utilizado por el algoritmo GMM [14]

2. "Predecir la clase de cada punto en los datos": Se utiliza el modelo GMM entrenado para predecir la clase de cada punto en los datos de entrada X. El método **predict(X)** asigna una etiqueta de clase a cada punto en función de la distribución de probabilidad estimada por el modelo GMM.

3. Graficar los datos con los colores basados en las etiquetas de clase predichas por GMM: Se realiza una visualización gráfica de los datos utilizando el gráfico de dispersión (scatter plot) de la biblioteca matplotlib. Los puntos se representan en el gráfico utilizando las características del eje x e y de los datos de entrada X. Los colores de los puntos se asignan en función de las etiquetas de clase predichas por el modelo GMM, utilizando el argumento c=**labels_gmm**. Además, se configuran los títulos y etiquetas de los ejes utilizando los métodos **xlabel**, **ylabel** y **title**.
Finalmente, se muestra el gráfico resultante utilizando el método **show()** de matplotlib.

A continuación, se observa como en cada gráfico, al igual que ocurría con K-Means, se distinguen diferentes cantidades de regiones en base al valor de K (**valor_GMM** en este caso) utilizado para cada uno.

Resultados obtenidos:

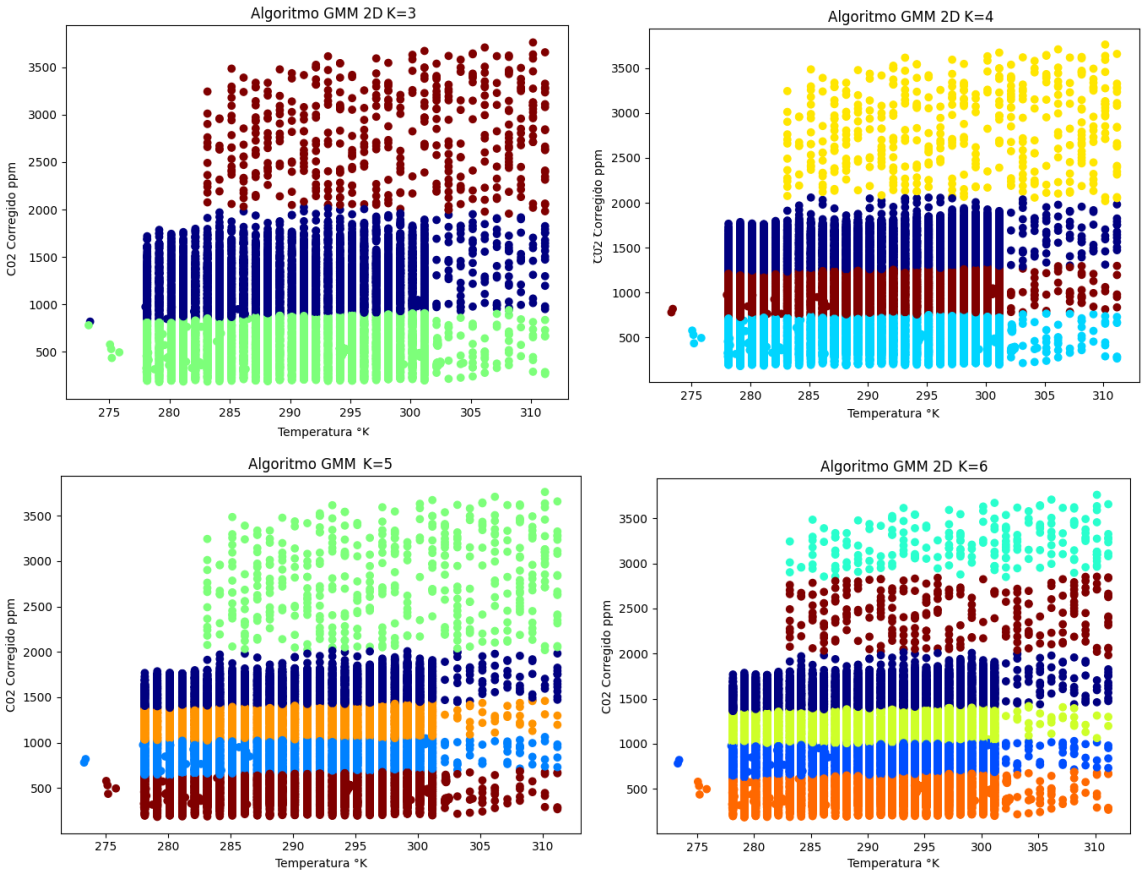

Fig. 18. Gráficos del algoritmo GMM en 2D utilizando valores de K= 3,4,5 y 6

Detectamos que los gráficos se asemejan bastante a los gráficos obtenidos con el algoritmo de K-Means, no obstante, si se observa con más detenimiento, los límites entre regiones no son tan rectos y definidos. Estos poseen ondulaciones, formas irregulares y contornos suaves. Esto se debe a que K-Means asume que los clústeres son esféricos y de tamaño similar, lo que puede limitar su capacidad para capturar formas de agrupamiento más complejas o datos con superposición.

Por otro lado, GMM es un modelo de mezcla de distribuciones gaussianas, que tiene en cuenta la forma de los datos y puede capturar agrupamientos más flexibles, permitiendo que los clústeres tengan formas y tamaños diferentes.

Antes de arribar a una conclusión respecto de los resultados obtenidos de ambos algoritmos, analizaremos cómo son los resultados de los gráficos con GMM en 3D.

GMM en 3D

En esta oportunidad, utilizaremos el algoritmo de GMM, pero ahora en tres dimensiones. La principal diferencia entre GMM 2D y GMM 3D radica en la complejidad y la capacidad de modelado. GMM 3D puede capturar relaciones más complejas y patrones en los datos al considerar una dimensión adicional. De esta forma, nos permitirá identificar clústeres y formar grupos influenciados por la tercera variable que agregamos: la humead.

```
#---------------Cálculos de GMM y su gráfico 3D---------------

# Crear instancia del modelo GMM y ajustar a los datos
print("Calculando GMM 3D...")
gmm = GaussianMixture(n_components=valor_GMM, random_state=4)
gmm.fit(grafico_3d)

# Predecir la clase de cada punto en los datos
labels_gmm = gmm.predict(grafico_3d)

# Graficar los datos en 3D con los colores basados en las etiquetas de clase predichas por GMM
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(grafico_3d[:,0], grafico_3d[:,1], grafico_3d[:,2], c=labels_gmm, cmap='jet')

ax.set_xlabel('CO2 Corregido ppm')
ax.set_ylabel('Temperatura °K')
ax.set_zlabel('Humedad g/m^3')
ax.set_title('Algoritmo GMM 3D')

plt.show()
```

Fig. 19. Código utilizado para realizar el gráfico 3D del algoritmo GMM

Funcionamiento del código:

1. "Crear instancia del modelo GMM y ajustar a los datos": Se crea una instancia del modelo GMM mediante la clase **GaussianMixture**, especificando el número de componentes **valor_GMM** y la semilla aleatoria **random_state=4**.

   El modelo GMM se ajusta a los datos **grafico_3d** mediante el método **fit()**, lo que significa que el modelo aprenderá las distribuciones y parámetros óptimos para los datos de entrada.

2. "Predecir la clase de cada punto en los datos": Se utiliza el método **predict()** para asignar una etiqueta o clase a cada punto en los datos **grafico_3d**. Esto se logra utilizando el cálculo de la probabilidad posterior de pertenencia a cada componente gaussiana.

3. "Graficar los datos en 3D con los colores basados en las etiquetas de clase predichas por GMM": A continuación, se grafican los datos en un gráfico tridimensional utilizando la biblioteca **matplotlib**. Cada punto se

representa en el espacio tridimensional, y se le asigna un color basado en la etiqueta o clase predicha por el modelo GMM.

El gráfico tridimensional se configura con etiquetas adecuadas para los ejes x, y y z, y se le asigna un título descriptivo. Finalmente, el gráfico se muestra en la pantalla con el método **show()**

Nuevamente en cada gráfico se podrá distinguir una diferente cantidad de regiones en base al valor de K (**valor_GMM**) utilizado para cada uno.

Resultados obtenidos:

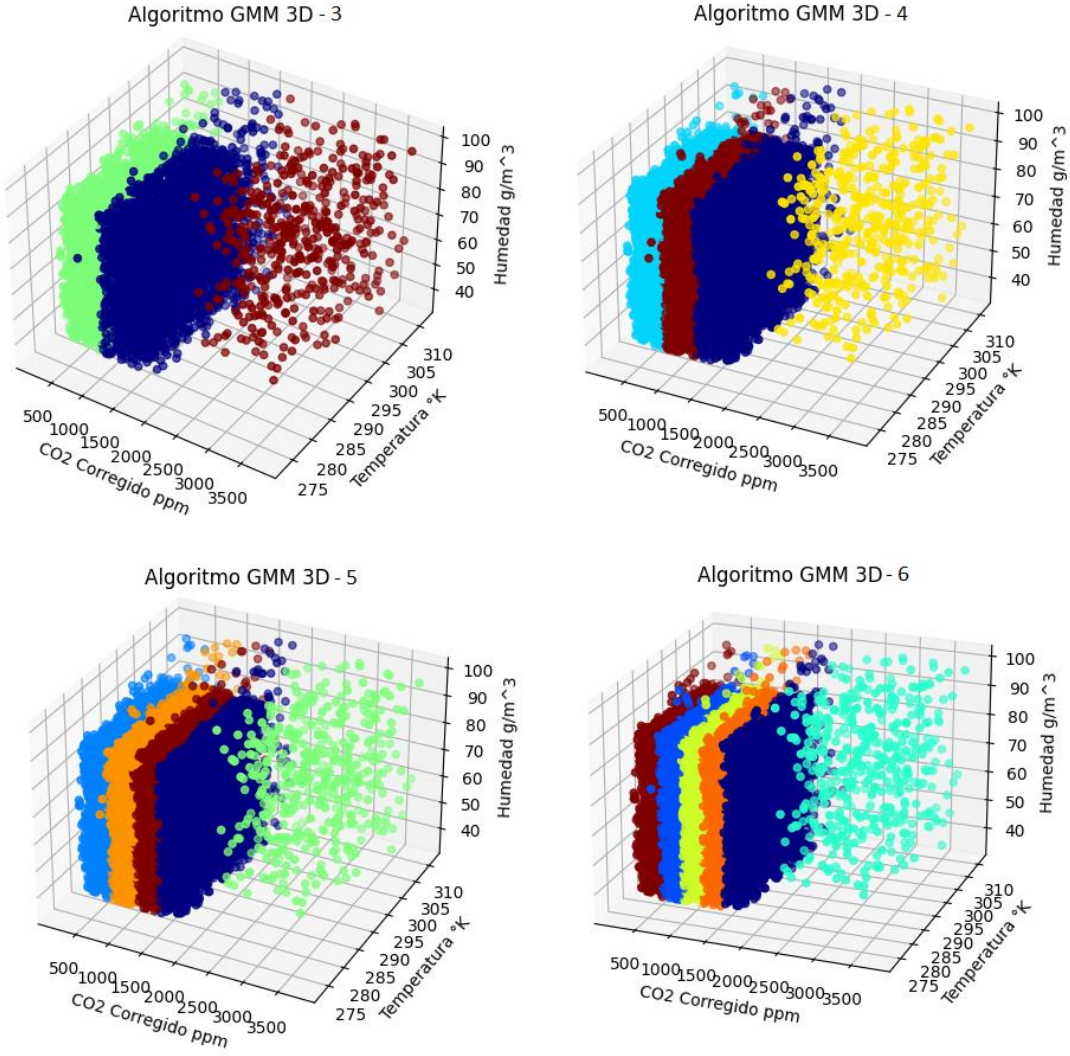

Fig. 20. Gráficos del algoritmo GMM en 3D utilizando valores de K= 3,4,5 y 6

En estos gráficos 3D se pueden apreciar algunas diferencias en las regiones obtenidas respecto del gráfico 2D. Para todos los valores de K, las regiones arriba

de los 2000 C02 corregido fueron agrupadas dentro de una misma región mucho más amplia. Estos resultados se deben a que GMM en 3D capturó relaciones más complejas entre las variables en comparación con el gráfico 2D.

Tanto GMM como K-Means, desmostaron ser algoritmos muy útiles para clasificar datos obtenidos por un sensor, sin importar la cantidad de variables utilizadas.

Observando los resultados de los gráficos, podríamos concluir que efectivamente GMM puede capturar agrupamientos de manera más flexibles. Permite que los clústeres tengan formas y tamaños diferentes, y que los límites de agrupamiento sean más suaves y menos rectos en los gráficos. A su vez, GMM también demostró ser más efectivo a la hora de tratar sets de datos con múltiples variables.

GMM puede ser más adecuado cuando se busca una modelización más flexible de los datos y cuando se desea tener una noción de la probabilidad de pertenencia a cada clúster. Por otro lado, K-Means puede ser más apropiado cuando se tiene un conjunto de datos con estructuras de clúster bien definidas y se busca una solución computacionalmente más eficiente. En este caso, debido a la baja complejidad de los datos utilizado, en conjunto con la clasificación de referencia de la Tabla 1, hacen que K-Means sea la opción más acertada para esta investigación.

Conclusión

Este trabajo concluye la etapa de prototipo del proyecto y se ha logrado la integración de sensores a una arquitectura REST para IoT sobre la red de telefonía celular, utilizando protocolos livianos como MQTT y LTE Cat M para transferir los datos a un gestor intermediario MQTT con un motor SQL, para el análisis de profundo de datos, y el uso de clientes RestAPI con rutas a los clasificadores para luego tomar acciones de corrección de los niveles de CO2 sobre otros actuadores que permitan corregir las dispersiones o, generar los niveles de alarma correspondientes.

Los procesos de clasificación en esta etapa se limitaron a verificar los conjuntos de datos de aprendizaje y el nivel de error para clasificar nuevos datos. Con respecto a los algoritmos de clasificación se obtuvieron resultados similares con K =3,4,5 y 6, una menor tasa de error usando K=5, y tanto con K-mean como con GMM se obtienen grupos de clasificación menos dispersos que con K-NN. La clasificación obtenida coincide con la esperada [Tabla 1] en base a los datos de prueba.

Referencias

1. IOT Analytics. Disponible: https://iot-analytics.com/number-connected-iot-devices.
2. MQTT Message Queuing Telemetry Transport. Disponible; https://mqtt.org/. MQTT Versión 5.0 Estándar OASIS. (marzo 2019).
3. OASIS. Organization for the Advancement of Structured Information Standards. Disponible: https://www.oasis-open.org/
4. Scikit-Learn - Machine Learning in Python. Recuperado el 05/05/23. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.htht
5. Scikit-Learn - Machine Learning in Python. Recuperado el 05/05/23. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans
6. Scikit-learn - Machine Learning in Python. Recuperado el 05/05/23. https://scikit-learn.org/stable/modules/mixture.html#mixture
7. Matplotlib. Recuperado el 05/05/23. https://matplotlib.org/
8. Numpy. Recuperado el 05/05/23. https://numpy.org/
9. Vaisala. https:// es.vaisala.com. Cómo medir el dióxido de carbono. Nota de Aplicación. 2013.
10. T. Teleszewski1 · K. Gładyszewska-Fiedoruk1. “The concentration of carbon dioxide in conference rooms: a simplified model and experimental verification”. Published online: 27 May 2019.
11. "Pattern Recognition and Machine Learning" de Christopher M. Bishop.
12. IBM. KNN. Recuperado el 07/05/23 https://www.ibm.com/topics/knn
13. Universidad de Oviedo. (s.f.). K-Means. Recuperado el 07/05/23 https://www.unioviedo.es/compnum/laboratorios_py/kmeans/kmeans.html
14. RStudio. k-Means Clustering. Recuperado el 07/05/23. https://rstudio-pubs-static.s3.amazonaws.com/705612_158ac73a6c0e42168f051b48c1f81bb5.html
15. J. Ouret, I. Parravicini. Reconfiguración Automática de Parámetros de Calidad de Servicio Dispositivos por medio del Protocolo NETCONF. 48 JAIIO. 2019.