

**UNIVERSIDAD EUROPEA  
MIGUEL DE CERVANTES**

**ESCUELA POLITÉCNICA SUPERIOR**

**TITULACIÓN:  
MÁSTER UNIVERSITARIO EN GESTIÓN Y ANÁLISIS  
DE GRANDES VOLÚMENES DE DATOS: BIG DATA**



**TRABAJO FIN DE MÁSTER**

**Estudio del efecto de la deriva de  
sensores para gases en modelos  
de ML**

**AUTOR**

**Daniel García Teba**

**TUTOR**

**Miguel Ángel Gómez López**

**VALLADOLID, octubre de 2020**

# Índice de contenidos

<b>1. Objetivos del trabajo</b>	<b>6</b>
<b>2. Ánalysis de la situación</b>	<b>7</b>
<b>3. Obtención, procesado y almacenamiento de los datos</b>	<b>11</b>
3.1. Procesado . . . . .	11
3.2. Exploratory Data Analysis . . . . .	11
3.3. Nota sobre los sensores . . . . .	15
3.4. Evolución visual del drift . . . . .	18
<b>4. Diseño e implementación de los modelos o técnicas necesarias</b>	<b>24</b>
4.1. Modelo secuencial . . . . .	24
4.1.1. Modelo simplificado . . . . .	24
4.1.2. Modelo completo . . . . .	25
4.1.3. Efecto del tipo de sensor . . . . .	27
4.2. Algoritmos no supervisados . . . . .	31
<b>5. Conclusiones y planes de mejora</b>	<b>36</b>
<b>A. Código utilizado</b>	<b>37</b>
A.1. Código Exploratory Data Analysis . . . . .	37
A.2. Código Red neuronal secuencial . . . . .	41
A.3. Código LGBM . . . . .	45
A.4. Código Randonforest . . . . .	49
A.5. Código aprendizaje no supervisado . . . . .	52
A.6. Código Utilities . . . . .	55
<b>Referencias</b>	<b>62</b>

# Índice de figuras

2.1.	Esquema de sistema de adquisición de datos . . . . .	9
2.2.	Respuesta del sensor . . . . .	10
2.3.	Descomposición de las señales temporales . . . . .	10
3.1.	Número de muestras de gas por Batch. El número de muestras ensayadas en cada lote es muy desigual, donde los lotes 1,4,5,8 y 9 cuentan con muchas menos mediciones que el resto. . . . .	14
3.2.	Número de muestras de cada gas en total . . . . .	15
3.3.	Recuento del numero de muestras disponibles. El eje de abscisas no es continuo, solo muestra de forma ordenada de menor a mayor las concentraciones disponibles en el dataset. . . . .	16
3.4.	Recuento del numero de muestras disponibles, agrupadas por in- tervalos. De esta forma es más visual poder apreciar el rango de variación de la concentración para cada gas. . . . .	17
3.5.	Número de muestras por gas . . . . .	18
3.6.	Señales de los sensores coloreadas por tipo. . . . .	19
3.7.	Correlation map dataset . . . . .	20
3.8.	Correlation map de 4 sensores de cada tipo . . . . .	21
3.9.	Señales disponibles para Gas 2, concentraciones 1-100, evolucion para cada batch . . . . .	22
3.10.	Media de las señales disponibles para Gas 2, concentraciones 1-100, evolucion para cada batch . . . . .	23
4.1.	Matriz de confusión del modelo utilizando todas las mediciones, entrenando al 70/30. Muy buenos resultados. . . . .	25
4.2.	Evolución del drift con los resultados de validación . . . . .	26
4.3.	Matriz de confusión utilizando todos los datos disponibles de los batchs 1 a 9. Los datos de entramiento y datos de validacion se han divido al 70-30 . . . . .	27

4.4. Matriz de confusión enfrentando el modelo a los datos nuevos proporcionados en le batch 10. El modelo sigue siendo preciso, pero no accurate, ya que confunde los gases.Podemos ver que 2 y 5 son catalogados como gas3. . . . .	28
4.5. Matrix de confusión para cada tipología de sensor . . . . .	29
4.6. Accuracy y precision . . . . .	30
4.7. Modelo LGBM-1sensor . . . . .	31
4.8. Modelo LGBM-6sensores . . . . .	32
4.9. Resultados KMeans Clustering . . . . .	33
4.10. Resultados TSNE . . . . .	34
4.11. Resultados PCA modelo simplificado . . . . .	34
4.12. Resultados PCA modelo batch 1 y 10 . . . . .	35

# Índice de tablas

3.1.	Distribución de los lotes a lo largo del tiempo. . . . .	12
3.2.	Numero de gases ensayados por batch. Notar que el gas 6 en los lotes 3,4 y 5 no está presente. . . . .	13
3.3.	Rango de variación de la concentraciones en ppmv utilizadas en las mediciones de los diferentes gases. . . . .	13
3.4.	Par de gas-concentración qué más mediciones se han realizado. Existen menos muestras disponibles para el par 2-200, pero se trata de suficientes muestras para poder entrenar un modelo. . . . .	16
4.1.	Modelo de red neuronal secuencial . . . . .	24

## Capítulo 1

# Objetivos del trabajo

El fenómeno *drift* o en español deriva es un problema importante que impide el uso fiable de sensores de gas, ya que con el tiempo se suceden diferentes efectos que alteran la respuesta del sensor ante el mismo estímulo.

Para estudiar este problema existe en *UCI Data Repository* un dataset, llamado *Gas Sensor Array Drift Dataset Data Set*, que puede ser descargado desde <https://archive.ics.uci.edu/ml/datasets/Gas+Sensor+Array+Drift+Dataset>.

Este trabajo va a ilustrar la complejidad de los datos de este dataset, compuesto por las mediciones de 16 sensores, detectando 6 tipos de gases a lo largo de 36 meses. Cada medición de un gas genera 16 series temporales, de las cuales de cada una se han extraído 8 *features*. Esto hace un total de 128 componentes para cada medición. Para cada medición es conocido el gas que se ha ensayado y su nivel de concentración.

En este trabajo se centrará en la tarea de clasificación de cada tipo de gas, dejando de lado la tarea de predecir dadas las 128 componentes el gas y su concentración.

Con este tipo de información, se va a estudiar los resultados que pueden ofrecer las redes neuronales para resolver este problema.

Se planteará un algoritmo de clasificación no supervisada, para después demostrar que el fenómeno de deriva de los sensores tiene un gran importancia en la precisión del modelo.

En este trabajo se utilizará Keras y TensorFlow para el diseño de las redes neuronales, en el entorno Python. Para los algoritmos no supervisados se ha elegido de la librería SciKit el cluster KMeans. Para modelos ML supervisados se va a analizar las bondades de RandomForest y LightGBM.

## Capítulo 2

# Análisis de la situación

El principal objetivo del dataset de Gas Drift de UCI es proporcionar un bechmark donde probar las diferentes soluciones y algoritmos para afrontar el problema de la deriva de sensores de gas, también conocidos como narices electrónicas.

En la Figure 2.1 (Zhao y cols., 2019) podemos ver la disposición del sistema de adquisición para construir el dataset que finalmente está disponible en UCI.

El gas pasa por la placa de adquisición, donde las señales temporales generadas por cada sensor (Figura 2.2) se descomponen en dos componentes de continua y 6 componentes de transitorio, 3 caracterizando la entrada del gas, y 3 caracterizando la salida. En la Figura 2.3 se resumen estas carateristicas. Para más detalle del significado de las componentes consultar (Vergara y cols., 2011)

Por tanto el dataset proporcionado caracteriza a cada gas con 128 componentes obtenidas de los diferentes sensores, 1 componente que especifica la concentración presente de ese gas en la medición, y el batch al que pertenece la medición, es decir, en qué meses se realizó. En total tendríamos 2 variables objetivo (gas y concentración), 128 características, y una componente temporal (batch id).

Esto presenta un reto para entrenar un algoritmo de clasificación, por varios motivos:

- ▷ Para un mismo gas, los sensores capturaran diferentes respuestas para diferentes concentraciones
- ▷ Una medición de un gas a una concentración dada, no será igual en los primeros meses de medición que en los últimos, debido a la deriva del sensor.
- ▷ Se han usado 4 tipos de sensores, y dentro de cada tipo se usaron 4 calibraciones diferentes (Zhao y cols., 2019) por lo que podría darse la situación de que algún sensor no sea capaz de detectar el paso del gas, o se vea saturado,

dando una medición no valida para clasificar.

En este trabajo primero comprobaremos que

- ▷ Existe esta deriva de los sensores. Comparación de la misma señal a lo largo de los meses.
- ▷ Esta deriva afecta a los modelos de clasificación. Modelo de clasificación muy sencillo con una red neuronal secuencial.
- ▷ Existe correlación entre las mediciones de los 16 sensores, ya que se trata de 16 mediciones del mismo fenómeno. Es más, las 8 componentes la descomposición propuesta de la señal, no son independientes entre sí.



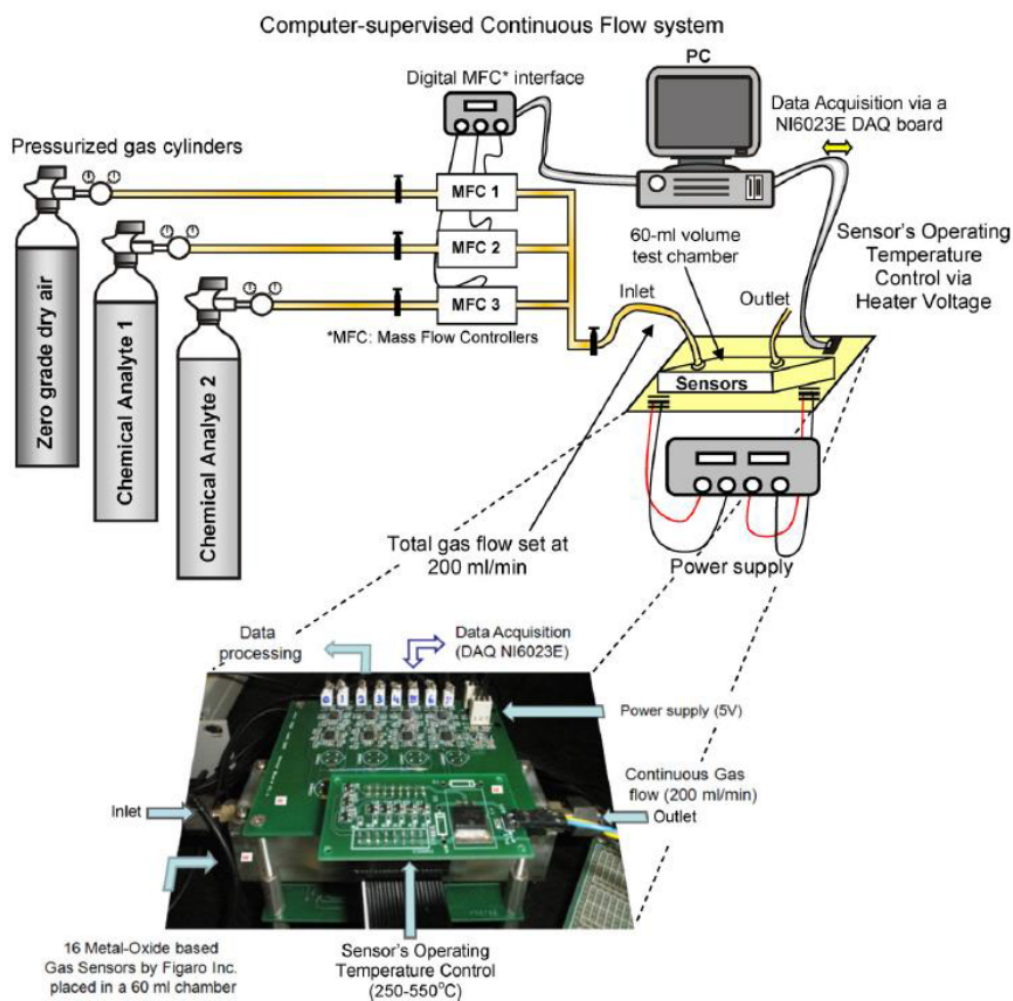


Figura 2.1: Esquema del sistema de adquisición, donde pueden verse los 16 sensores de gas Figaro.

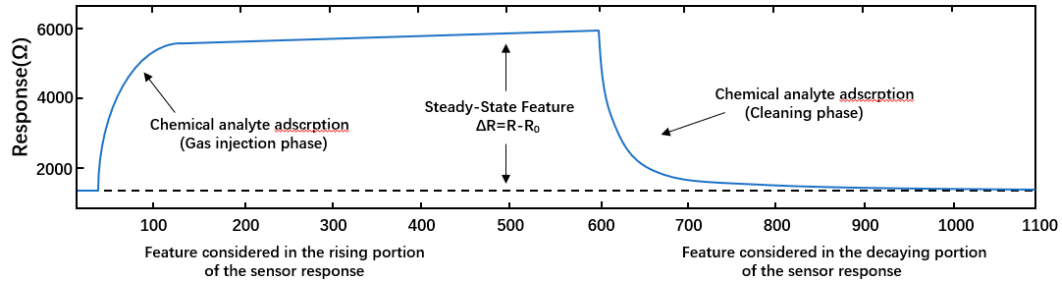


Figura 2.2: Información típica de la respuesta del sensor. Respuesta típica de una sustancia química a base de óxido metálico sensor a 30ppmv de acetaldehído. La curva muestra las tres fases de una medición: línea de base medición (realizada con aire puro), medición del gas de prueba (cuando se inyecta el analito químico, en forma de gas, a la cámara de prueba) y fase de recuperación (durante la cual el sensor se expone nuevamente a aire puro; el tiempo de recuperación suele ser mucho más largo que la inyección de gas(Vergara y cols., 2011))

Steady-State features	Transient features	
	rising portion	decaying portion
$\Delta R$	$MAXema_{\alpha=0.001}$	$MINema_{\alpha=0.001}$
$  \Delta R  $	$MAXema_{\alpha=0.01}$	$MINema_{\alpha=0.01}$
	$MAXema_{\alpha=0.1}$	$MINema_{\alpha=0.1}$

Figura 2.3: Descripción de la descomposición de las señales temporales generadas por los sensores. Más información en (Vergara y cols., 2011)

## Capítulo 3

# Obtención, procesamiento y almacenamiento de los datos

Los datos provienen del Gas Drift Dataset del repositorio UCI, donde el objetivo era tratar de detectar el drift (la deriva) de los sensores a lo largo de los meses. Están disponibles un total de 13910 mediciones.

Los datos están disponibles para su descarga desde UCI data repository. en 10 archivos formato *.dat*. Cada archivo *.dat* contiene una codificación tipo libsvm *x:v*, donde *x* representa el número de característica y *v* el valor real de la característica. Por ejemplo, en 1; 10.000000 1: 15596.162100 2: 1.868245 3: 2.371604 4: 2.803678 5: 7.512213 ... 128:-2.654529

X sería 1;10.000, que nos dice que se trata del gas 1 con una concentración de 10, y a continuación las 128 componentes obtenidas de los sensores. con 129 columnas, donde la primera nos informa del gas y la concentración, y el resto es la información obtenida del sensor.

### 3.1 Procesado

La lectura de los archivos *.dat* se ha realizado utilizando una librería que se ha creado para cargar de forma rápida y cómoda todos los datos en un Dataframe de Pandas. El código adjunto en Apéndices ??.

### 3.2 Exploratory Data Analysis

Los datos se nos presentan en 10 lotes, correspondientes a experimentos a lo largo de tres años, donde se ensayaron 6 diferentes gases a diferentes concentraciones.

Batch ID	Month IDs
Batch 1	Months 1 and 2
Batch 2	Months 3, 4, 8, 9 and 10
Batch 3	Months 11, 12, and 13
Batch 4	Months 14 and 15
Batch 5	Month 16
Batch 6	Months 17, 18, 19, and 20
Batch 7	Month 21
Batch 8	Months 22 and 23
Batch 9	Months 24 and 30
Batch 10	Month 36

Tabla 3.1: Distribución de los lotes a lo largo del tiempo.

Los gases que se estudiaron son los siguientes:

- ▷ Ethanol
- ▷ Ethylene
- ▷ Ammonia
- ▷ Acetaldehyde
- ▷ Acetone
- ▷ Toluene

Los lotes contienen una cantidad de muestras desigual, ni los 6 gases de estudio están presentes en todos los lotes. La tabla 3.2 muestra el numero de ensayos para cada gas.

En la Tabla3.2 podemos ver que el número de muestras en cada lote es desigual. En los lotes 3, 4 y 5 el gas 6 no está presente. A la hora de crear un dataset de entrenamiento, convendría generar un lote donde haya un numero equitativo de muestras de todos los gases, y si las mediciones no son distantes en el tiempo podremos ver el efecto de la deriva si el algoritmo entrenado con los primeros lotes falla para cada vez más conforme nos alejamos en el tiempo.

La Figura 3.1 muestra la cantidad de gases ensayados por lote, mientras que la Figura3.5 muestra el numero de mediciones totales sobre cada gas.

En la Tabla 3.3 los rangos de concentración para cada gas, han sido también diferentes.

Batch ID	GAS1	GAS2	GAS3	GAS4	GAS5	GAS6
1	90	98	83	30	70	74
2	164	334	100	109	532	5
3	365	490	216	240	275	0
4	64	43	12	30	12	0
5	28	40	20	46	63	0
6	514	574	110	29	606	467
7	649	662	360	744	630	568
8	30	30	40	33	143	18
9	61	55	100	75	78	101
10	600	600	600	600	600	600

Tabla 3.2: Numero de gases ensayados por batch. Notar que el gas 6 en los lotes 3,4 y 5 no está presente.

GAS	Minimo	Máximo	Media	StdDesv
1	2.5	600	114.95	86.64
2	2.5	300	116.1	79.89
3	2.5	1000	323.55	272.02
4	2.5	300	126.32	76.71
5	10	1000	228.57	217.38
6	1	230	47.66	32.58

Tabla 3.3: Rango de variación de la concentraciones en ppmv utilizadas en las mediciones de los diferentes gases.

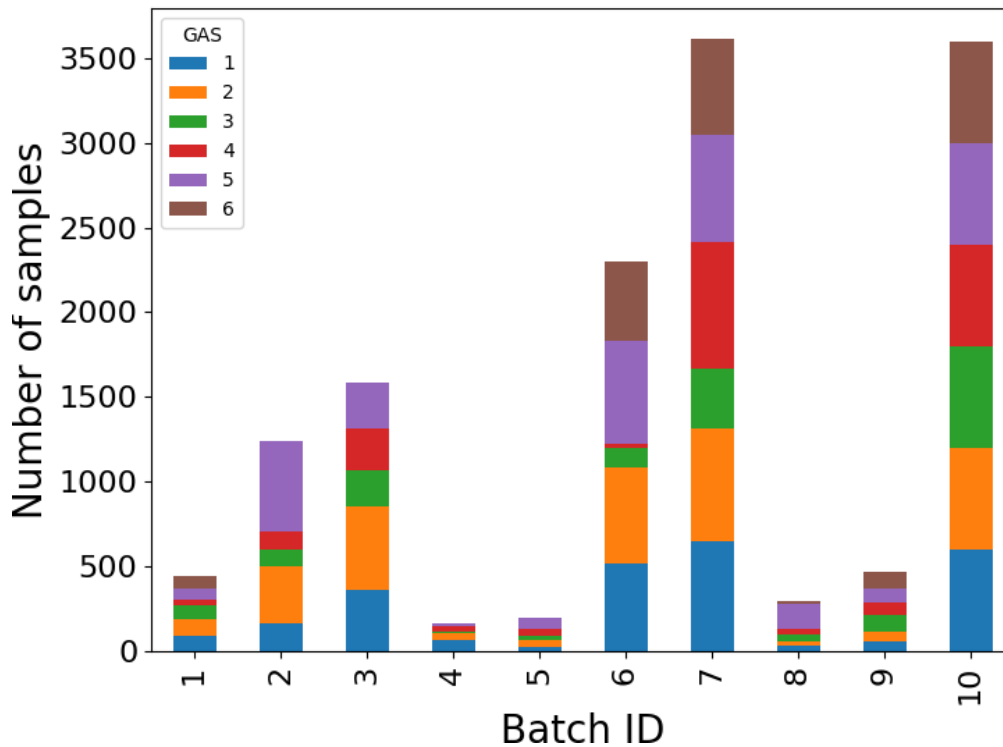


Figura 3.1: Número de muestras de gas por Batch. El número de muestras ensayadas en cada lote es muy desigual, donde los lotes 1,4,5,8 y 9 cuentan con muchas menos mediciones que el resto.

En la Figura3.3 se ha realizado un recuento de cuantas muestra para cada par gas-concentración están disponibles dentro de las 13910 muestras.

A la vista de la distribución de concentraciones disponibles, para eliminar el efecto de la concentración en la señal generada por el gas, utilizaremos la concentración más ensayada para cada gas.

Esta información es necesario tenerla en cuenta a la hora de entrenar nuestro modelo, ya que si el rango de variación de los datos es dispar, será recomendable realizar una normalización de los mismos para evitar perder la información contenida en las variables con un rango menor.

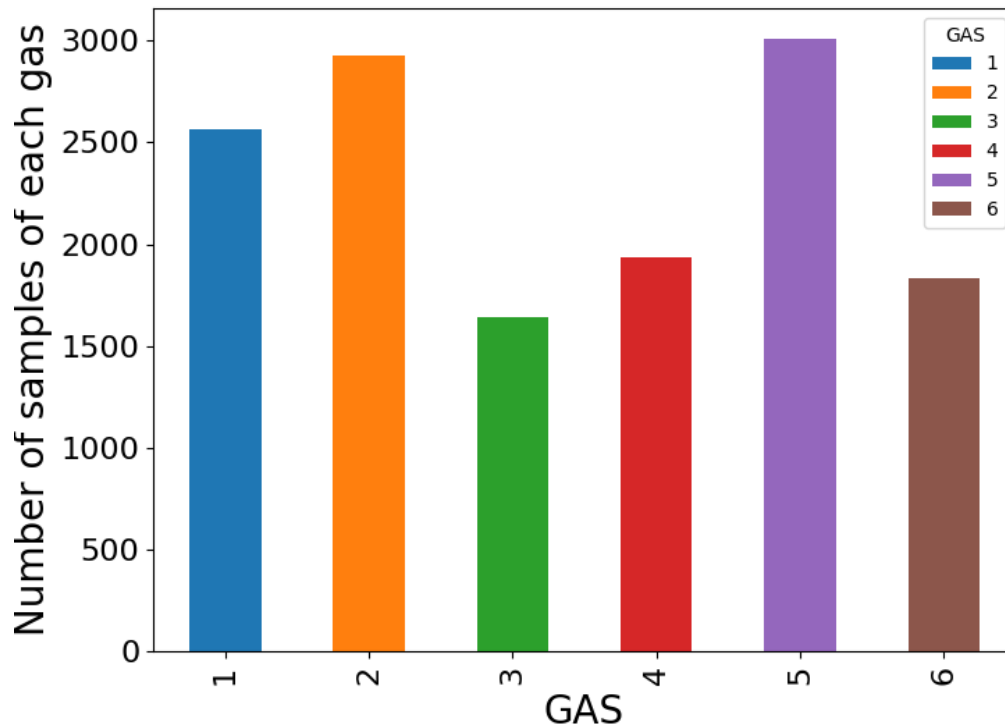


Figura 3.2: Número de muestras de cada gas en total

### 3.3 Nota sobre los sensores

Hay que tener presente que los 16 sensores están generando señales similares y correlacionadas. En la Figura3.6 queda reflejado cómo las diferencias entre cada sensor/calibración son claras.

Esto es un problema a la hora de alimentar a los modelos con variables independientes. En la Figura3.7 se ha representado el mapa de correlación de las 128 componentes. Para mayor claridad en la Figura3.8 se han representado las features correspondientes a los sensores de diferentes tipos. Puede verse de forma clara cómo existe una correlación fuerte, tanto positiva como negativa, entre las variables que alimentarán el modelo.

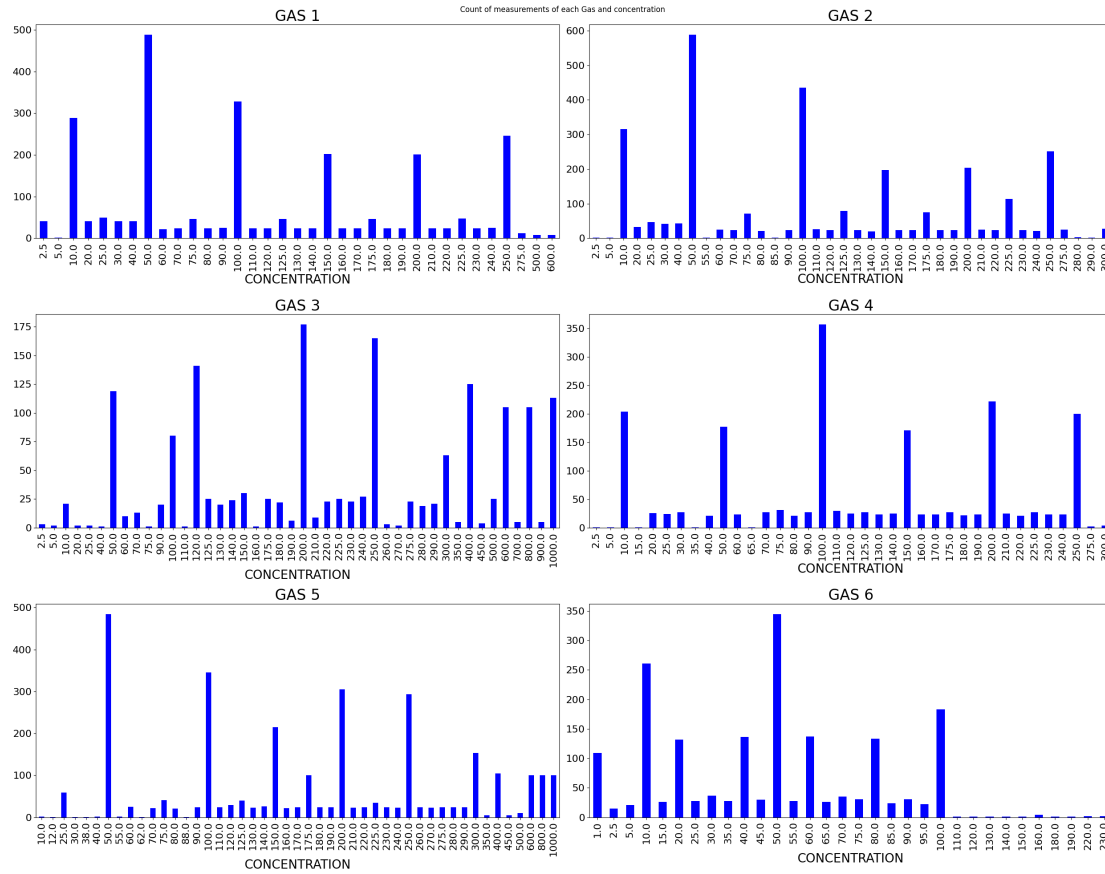


Figura 3.3: Recuento del numero de muestras disponibles. El eje de abscisas no es continuo, solo muestra de forma ordenada de menor a mayor las concentraciones disponibles en el dataset.

GAS	CONCENTRACIÓN	Numero de muestras
1	50	488
2	50	588
3	200	177
4	100	357
5	50	485
6	50	345

Tabla 3.4: Par de gas-concentración qué más mediciones se han realizado. Existen menos muestras disponibles para el par 2-200, pero se trata de suficientes muestras para poder entrenar un modelo.



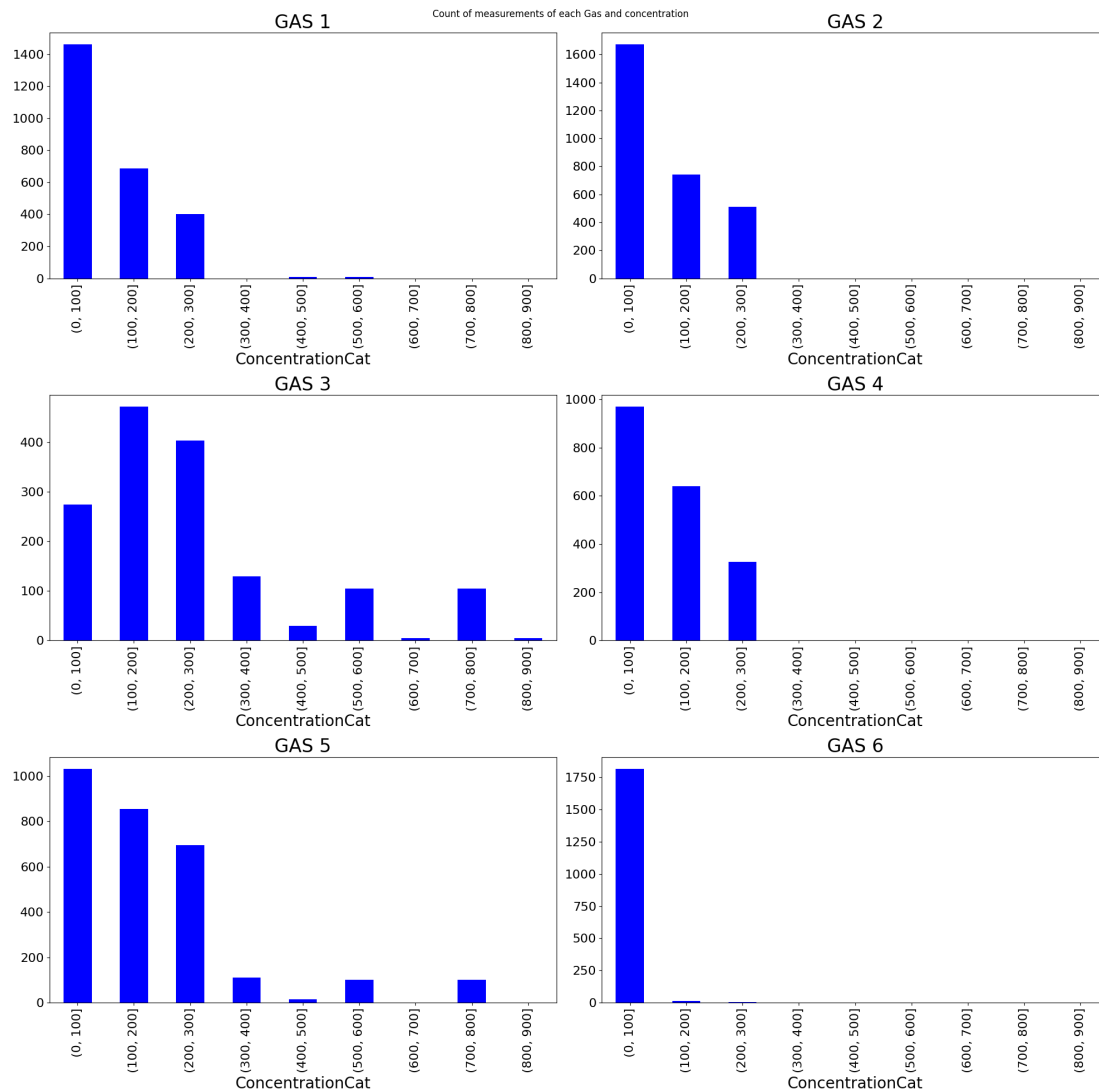


Figura 3.4: Recuento del numero de muestras disponibles, agrupadas por intervalos. De esta forma es más visual poder apreciar el rango de variación de la concentración para cada gas.

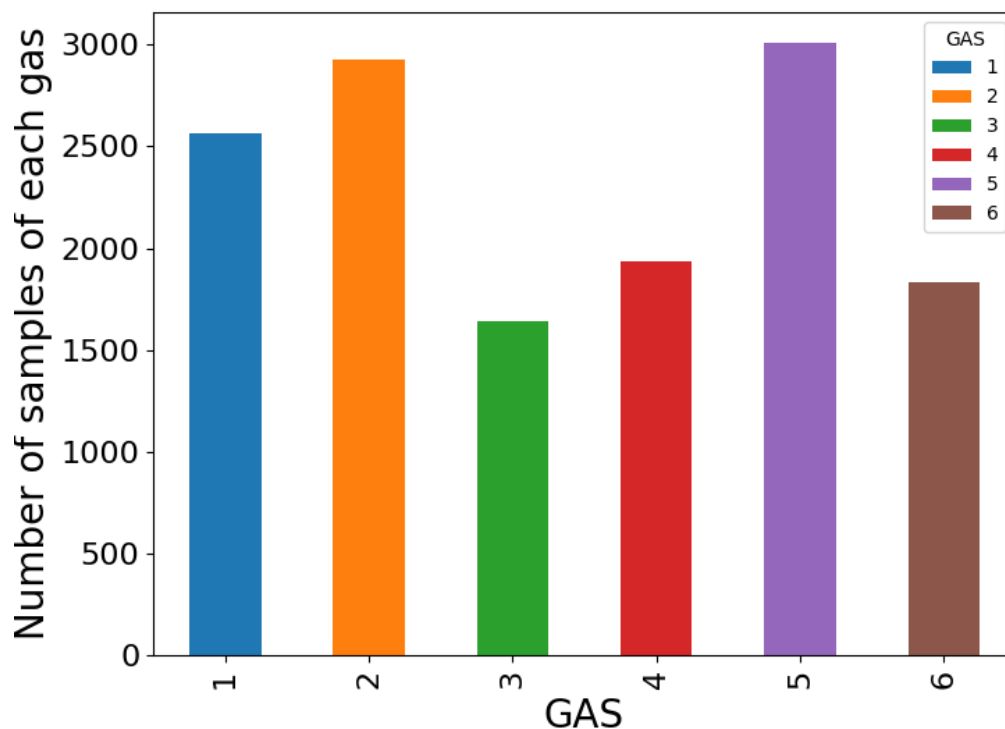


Figura 3.5: Número de muestras de cada gas en total

### 3.4 Evolución visual del drift

Se va a proceder a dibujar la señal generada por el sensor 1 de un par gas-concentración, en los diferentes batches, y así ver la deriva de la señal generada. Escogemos para tal tarea el par gas 2 con concentraciones entre 1 y 100.

Dado que las componentes estacionarias y transitorias tienen magnitudes muy diferentes, para ayudar a su visualización realizamos un escalado estándar.

No puede apreciarse a simple vista el efecto de la deriva del sensor. Si realizamos la media de todas las mediciones realizadas en cada batch 3.10, vemos que la firma del gas 2 no cambia de forma visible.

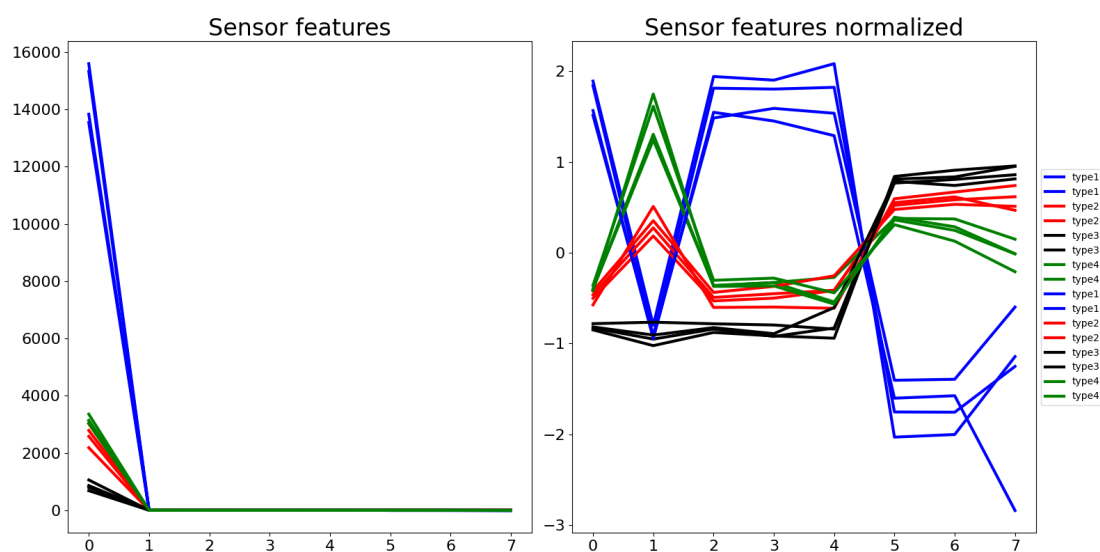


Figura 3.6: En el ejeX tenemos las 8 componentes que definen la señal de un sensor. Las señales están coloreadas según el tipo de sensor, al que se le ha denominado Tipo 1,2,3 y 4. A la izq la img muestra las señales tal como están presentes en el dataset, y a la drch están normalizadas.

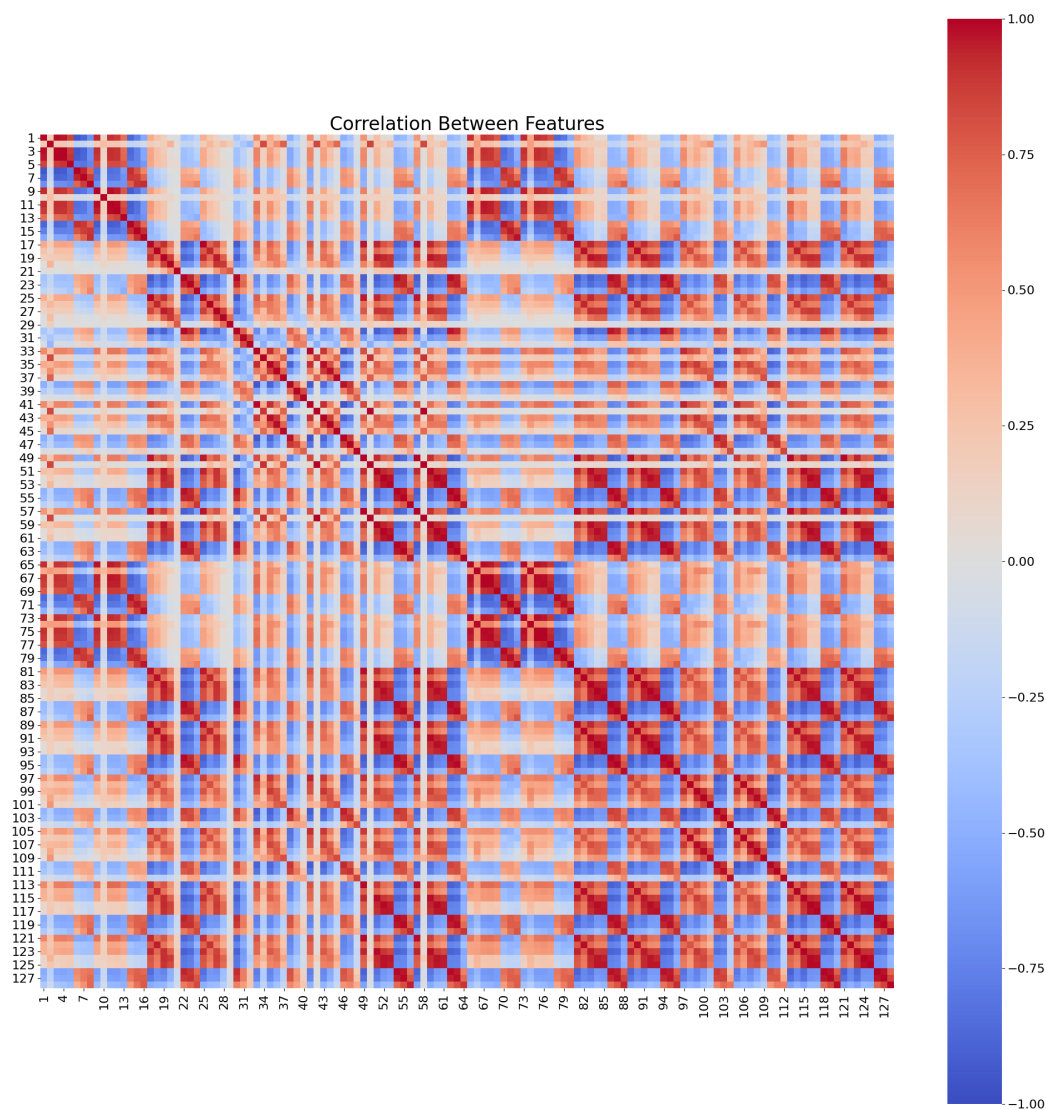


Figura 3.7: Correlación entre las 128 componentes del dataset.

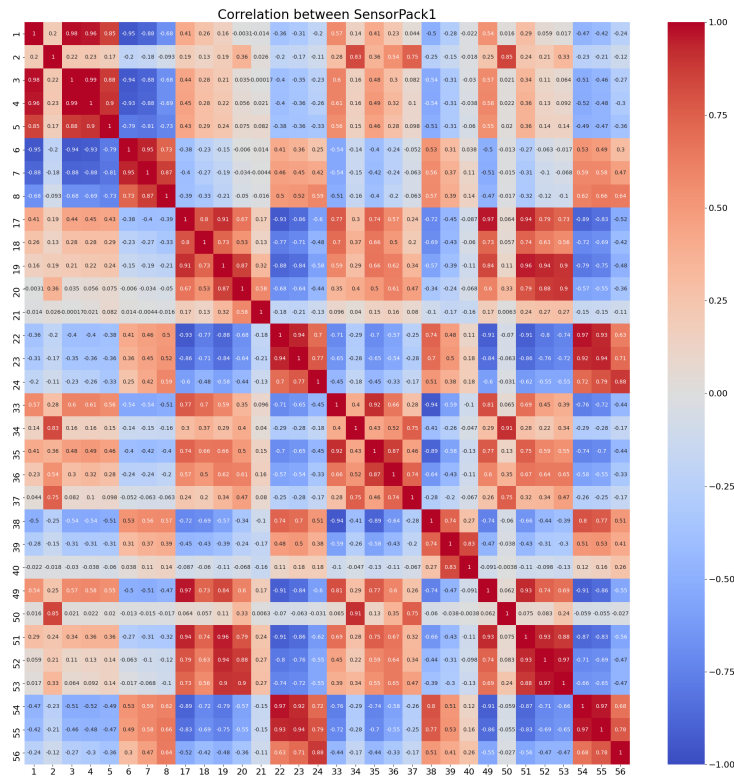


Figura 3.8: Correlación entre las 128 componentes de 4 sensores de diferentes tipos.

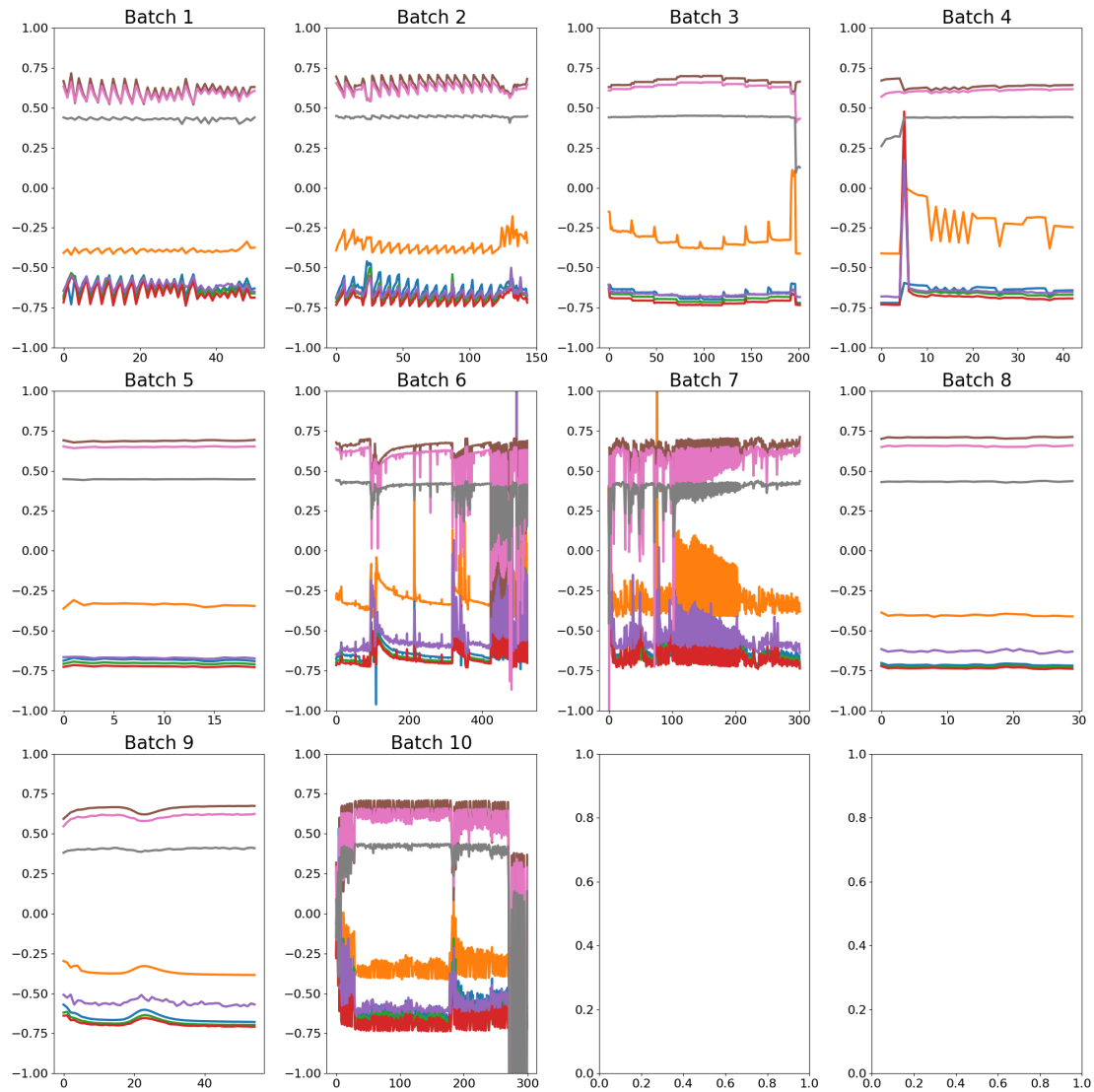


Figura 3.9: Para cada batch, se han representado todas las señales disponibles para el Gas 2 y concentraciones en el rango de 1 a 100 ppvm. A simple vista no puede apreciarse la deriva del sensor.

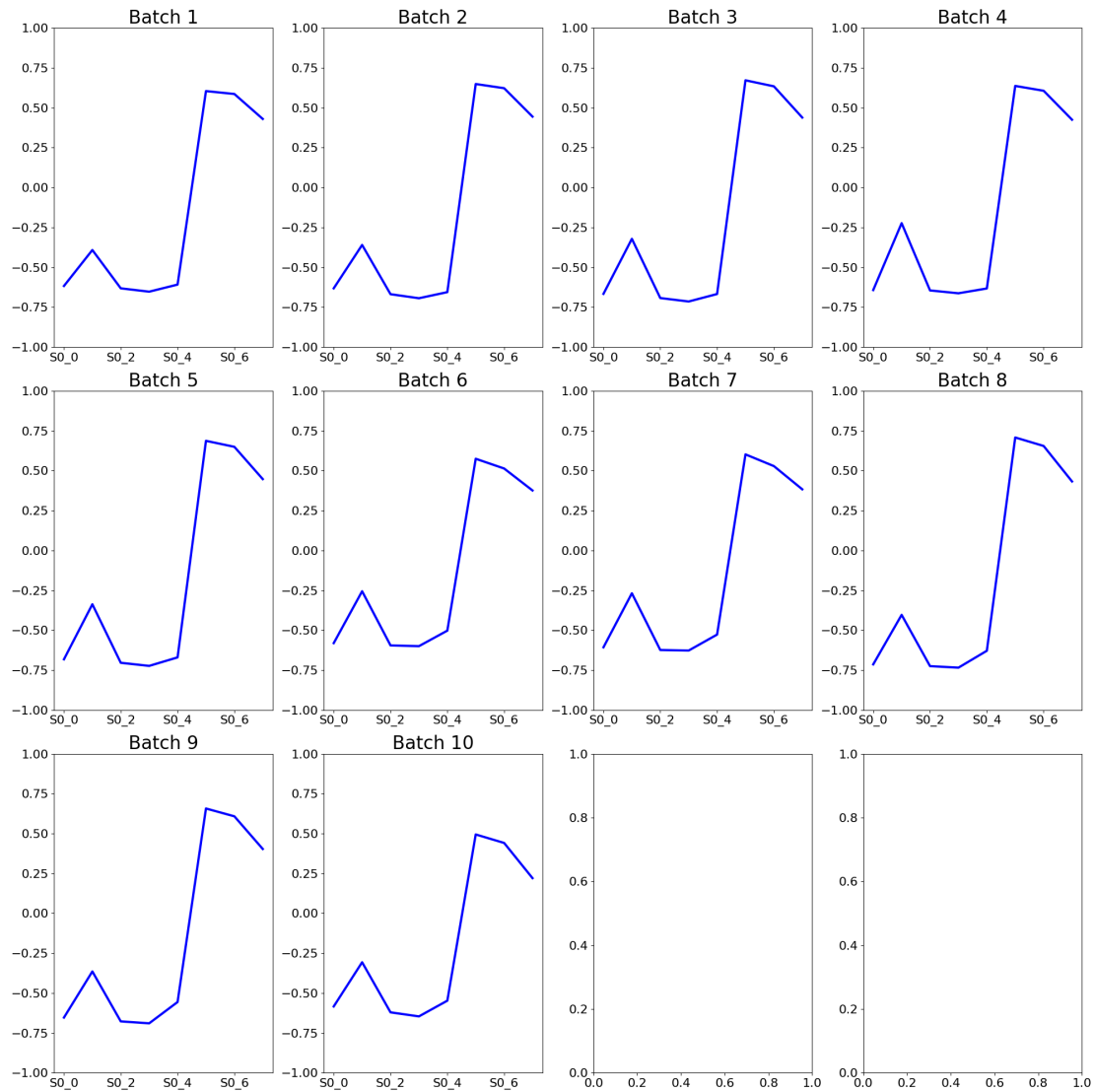


Figura 3.10: Para cada batch, se ha calculado la media de todas las señales disponibles para el Gas 2 y concentraciones en el rango de 1 a 100 ppm. En el ejeX, la nomenclatura  $SX\_Y$  significa SensorX, featureY, indexado en cero.

## Capítulo 4

# Diseño e implementación de los modelos o técnicas necesarias

### 4.1 Modelo secuencial

En este capítulo se ah diseñado una red neuronal muy sencilla, con itención de observar los efectos que el drift puede tener en el accuracy y el loss del modelo. El esquema de la red neuronal se presenta en la tabla 4.1

#### 4.1.1 Modelo simplificado

Primero, modelo más simple posible. Los datos de entrenamiento serán las mediciones de 1 sensor y como valores objetivo tomamos pares gases-concentración . Es decir, reducimos las mediciones a 6 parejas gas-concentración únicas tomadas por

Model: "sequential"		
Layer (type)	Output Shape	Param
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 32)	4128
dense_1 (Dense)	(None, 32)	1056
dense_2 (Dense)	(None, 6)	198
Total params: 5,382		
Trainable params: 5,382		
Non-trainable params: 0		

Tabla 4.1: Modelo de red neuronal secuencial



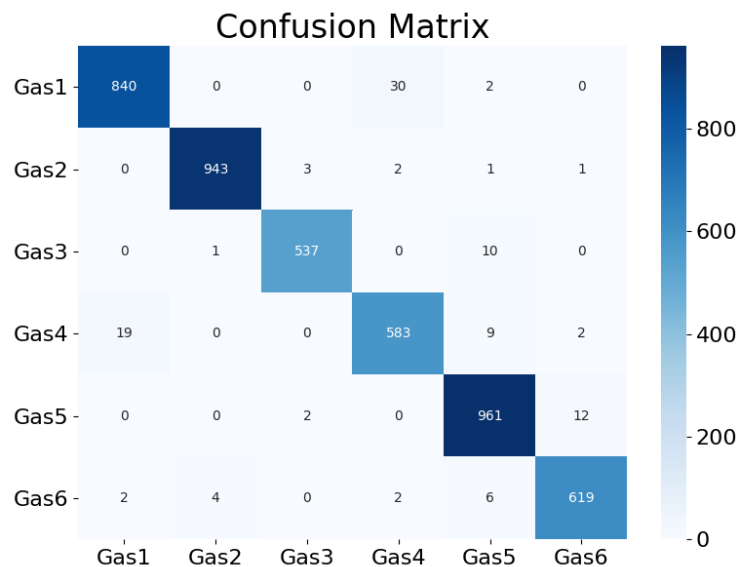


Figura 4.1: Matriz de confusión del modelo utilizando todas las mediciones, entrenando al 70/30. Muy buenos resultados.

un único sensor.

Reducimos así las variables de estudio, y podemos ver el efecto del drift en las mediciones de un gas dado. Utilizando todos los datos, división entre entrenamiento y validación al 70/30, obtenemos la matriz de confusión de la Figura4.1

Sabemos que nuestro modelo a podido aprender a diferenciar entre los gases, pero no es realmente útil, ya que el modelo está aprendiendo cómo es un gas tanto en el primer mes de mediciones como en el último. Es decir, este modelo daría malos resultados si probamos con nuevas mediciones.

#### 4.1.2 Modelo completo

Si este mismo modelo lo entrenamos con todos los datos disponibles de los sensores, el primer batch, y medimos el accuracy y el loss frente a los siguientes batch, obtenemos las Figuras4.2. Se observa claramente cómo tanto el accuracy decae conforme intentamos predecir valores más lejanos.

Esto nos indica que la deriva es acumulativa, conforme mas nos alejamos de los meses de entrenamiento peor accuracy tiene el modelo.

Debemos mencionar que estos resultados se han obtenido utilizando los datos de

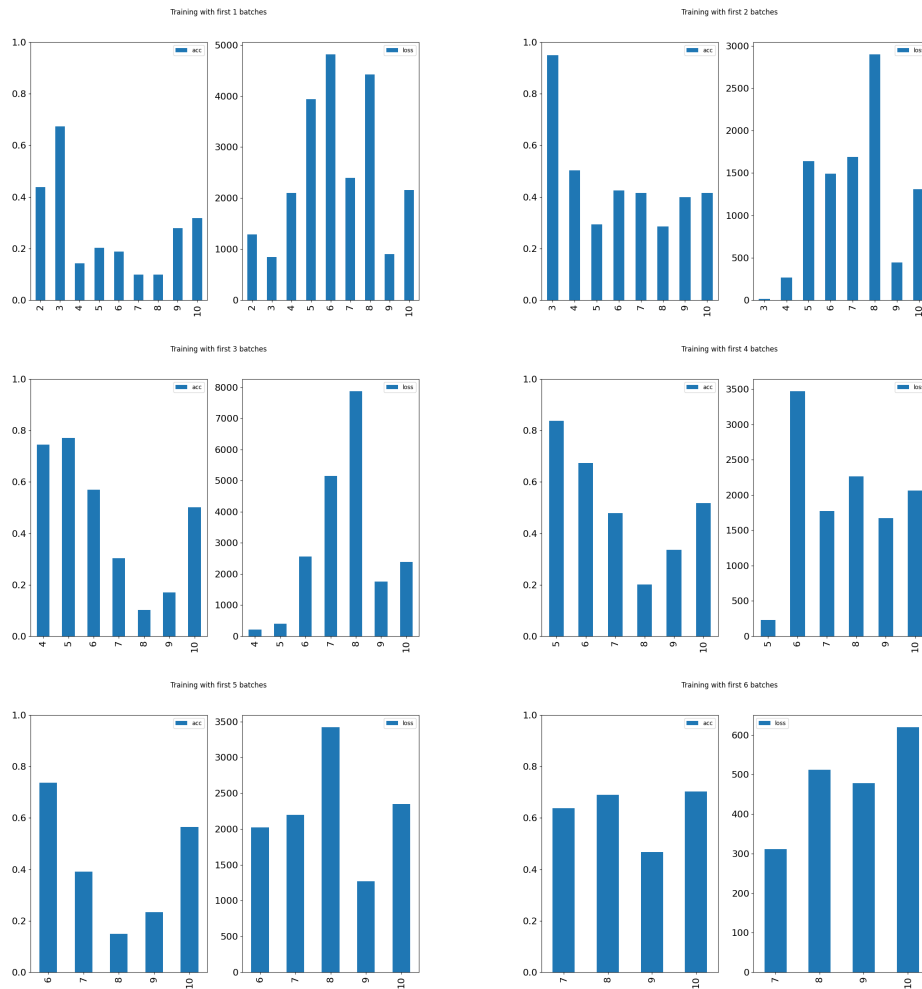


Figura 4.2: Evolución del drift con los resultados de validación. Utilizando un numero N de lotes como entrenamiento, y calculando su accuracy y loss frente a los siguientes batchs de forma individual.

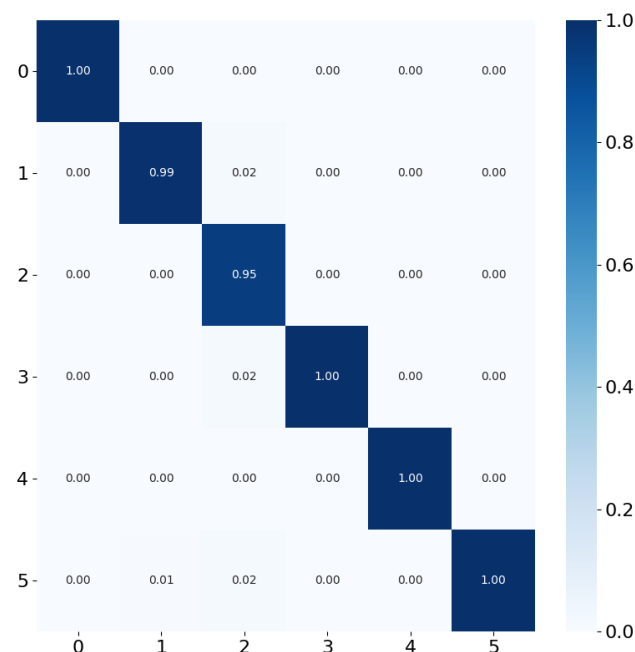


Figura 4.3: Matriz de confusión utilizando todos los datos disponibles de los batches 1 a 9. Los datos de entrenamiento y datos de validación se han dividido al 70-30

los 16 sensores, que como ya se mencionó guardan una fuerte correlación.

Esto podría hacer que el modelo no diera importancia al resto de variables, ya que siempre es necesario entrenar los modelos con variables independientes. La multicolinealidad reduce la precisión de los coeficientes de estimación, lo que debilitaría los modelos de regresión, y por tanto cualquier modelo basado en regresión: redes neuronales, random forest, etc.

Sin embargo, es posible que este efecto se vea compensado por la información añadida que sí que aporta, ya que el gas a una concentración dada pasaría de estar definido por 8 variables a 128.

Al utilizar RandomForest y LightGBM se han encontrado la misma evolución. Cabe destacar que LightGBM es increíblemente rápido para entrenar.

#### 4.1.3 Efecto del tipo de sensor

Vamos a entrenar 4 modelos, uno utilizando las mediciones de cada tipo de sensor, y compararemos las matrices de confusión, para ver si hay sensores que no consiguen detectar un gas específico. Denominemos al tipo de sensor A,B,C y D.

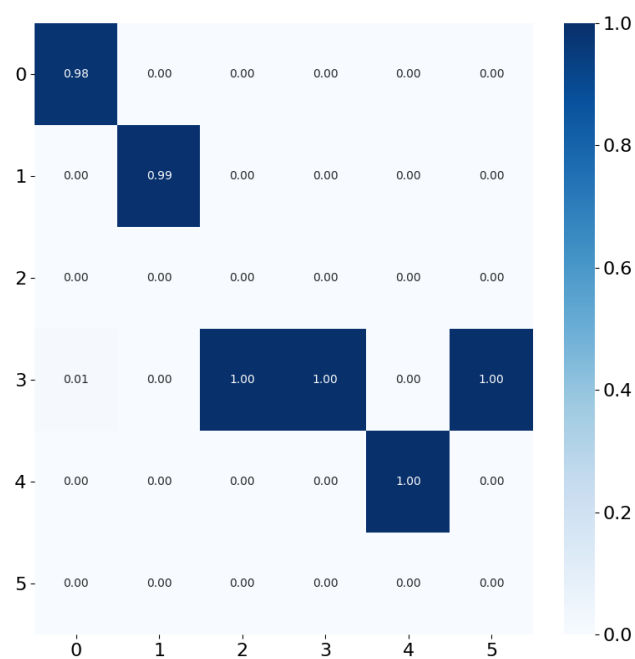


Figura 4.4: Matriz de confusión enfrentando el modelo a los datos nuevos proporcionados en le batch 10. El modelo sigue siendo preciso, pero no accurate, ya que confunde los gases.Podemos ver que 2 y 5 son catalogados como gas3.

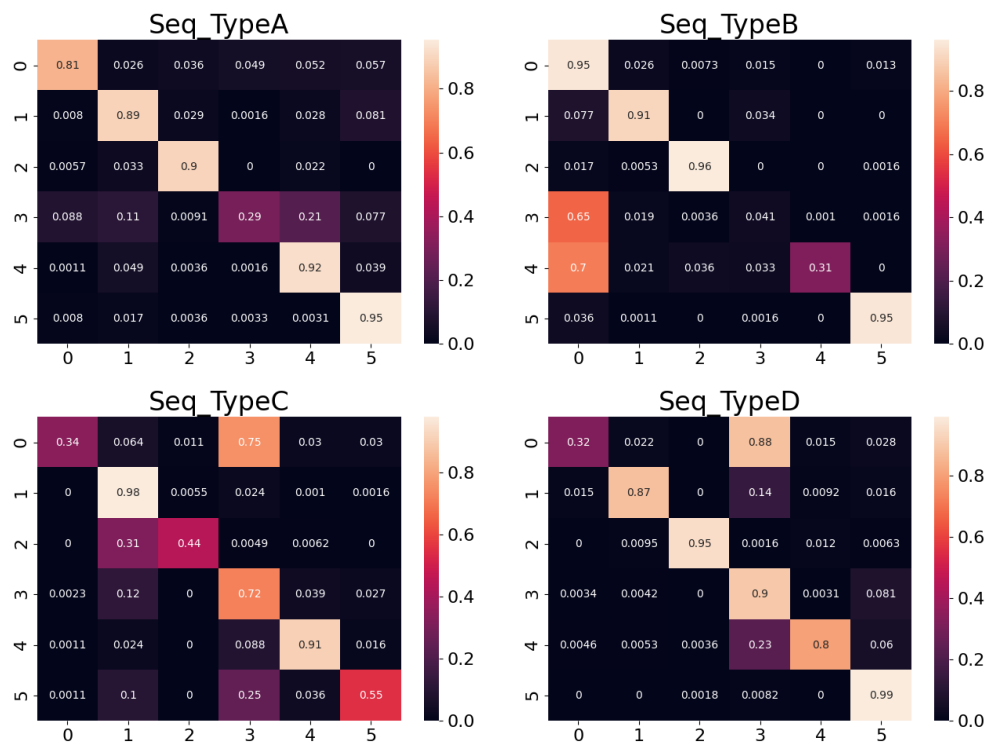


Figura 4.5: Matriz de confusión utilizando solo sensores tipo A en la img sup izq, solo sensores tipo B en la img sup drech y así sucesivamente. El sensor tipo A predice bastante bien todos los tipos de gases, los sensores B no consiguen detectar dos gases, y los tipo C y D fallan mucho al detectar un gas en concreto.

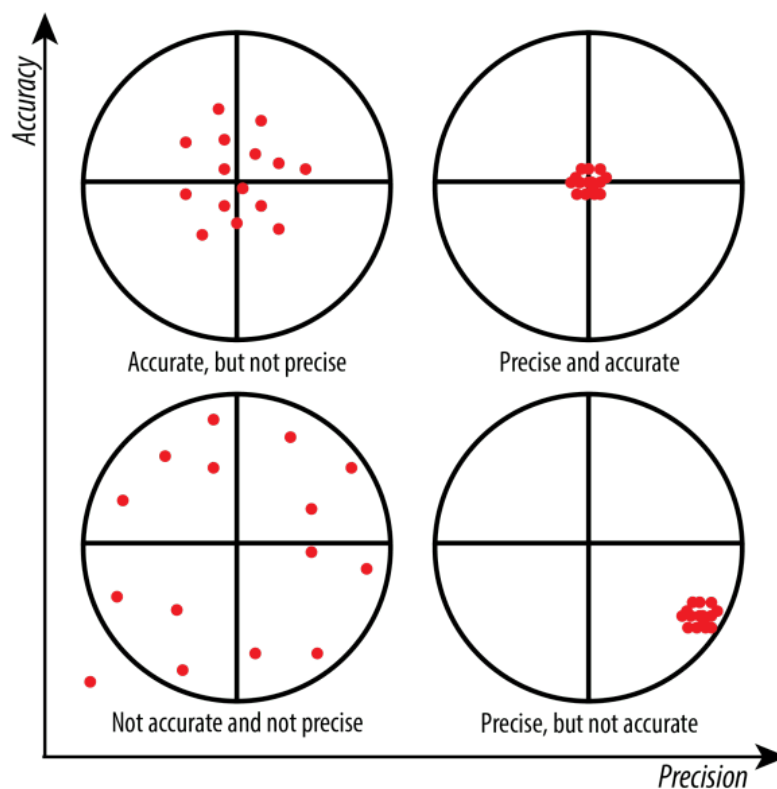


Figura 4.6: Es importante diferenciar entre accuracy y precision, ya que según de que forma fallen nuestros modelos, ya que podría aportarnos información sobre el problema al que nos enfrentamos.

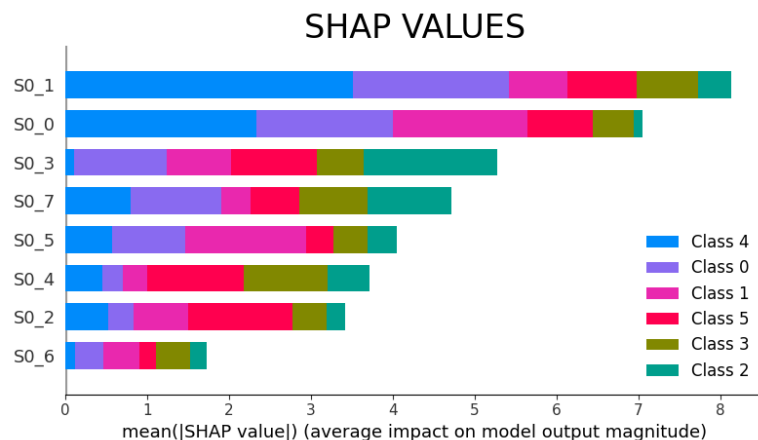


Figura 4.7: Modelo de LGBM para un sensor. Prácticamente todas las variables tienen influencia en la predicción.

Las diferencias de sensibilidad entre cada sensor del mismo tipo influyen en sus predicciones. La efectividad de los modelos basados en cada sensor es peor que el modelo total. Pero cabe la posibilidad de que el modelo de 128 dimensiones esté desechando muchas dimensiones, no utilizando la información proporcionada por todos los sensores.

La Figura 4.5 nos da mucha información sobre qué está ocurriendo, ya que el modelo de tipoB no es accuracy, pero sí precise, ya que los gases 3, y 4 los está clasificando como gas0. (ver Figura 4.6)

Si utilizamos RandomForest o LightGBM, podemos representar los shap values, y ver cuales son las variables que más están influyendo en el resultado de la predicción. Los shap values para un modelo entrenado únicamente con un sensor son razonables, 4.7, pero si utilizamos las 128 componentes disponibles vemos que el modelo está desechando mucha información, Figura 4.8

## 4.2 Algoritmos no supervisados

Para este apartado se ha elegido dos algoritmos de clasificación no supervisada, KMeans clustering y TSNE.

Para el algoritmo de KMeans clustering se ha realizado la Figura 4.9. En la imagen drcha se puede apreciar cómo hay mediciones que están bien diferenciadas del resto, mientras que hay un gran conjunto de ellas que el algoritmo no ha podido separarlas, ya que se trata de mediciones muy similares, pero sin embargo e trata

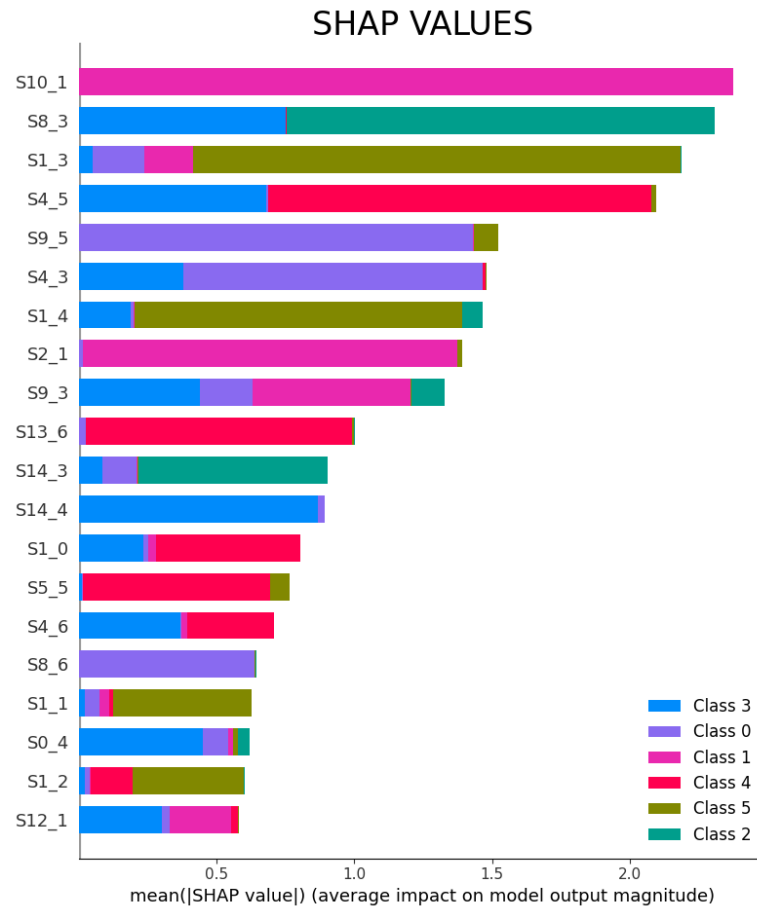


Figura 4.8: Modelo de LGBM para todos los sensores. podemos observar que está tomando componentes aleatorias de cada sensor. De las 128 componentes, podríamos quedarnos únicamente con las 20 primeras y los resultados de la predicción serían similares.



de gases distintos.

Esto pone de manifiesto la complejidad del problema, ya que si las señales para cada gas fueran suficientemente diferentes entre sí, el algoritmo de clustering no habría encontrado problema en diferenciarlas.

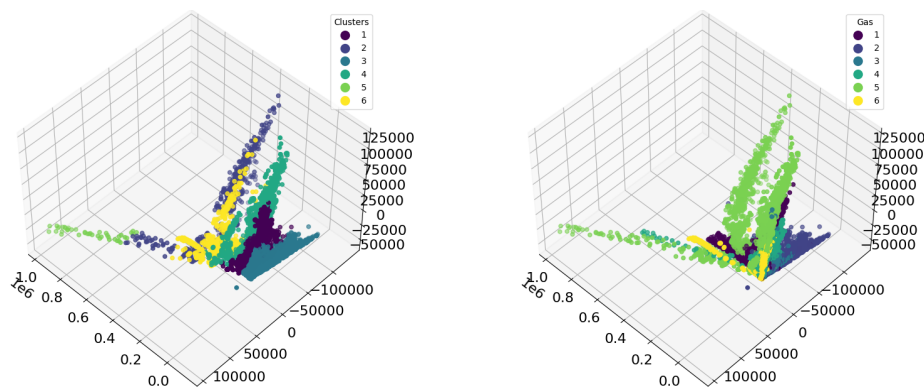


Figura 4.9: En ambas imágenes los puntos están distribuidos según los planos propuestos por PCA. En la imagen a la izquierda podemos ver los grupos propuestos por el algoritmo de KMeans Clustering, mientras que a la derecha se ha coloreado cada punto según a qué gas pertenece dicha medición.

Otro algoritmo de clasificación no supervisada, TSNE *t-distributed stochastic neighbor embedding*, nos ha dado como resultado la Figura 4.10, donde se ha coloreado cada punto en función del gas al que pertenece. Esta imagen pone de manifiesto cómo hay mediciones que a pesar de pertenecer a otros gases, son tan similares a otras que acaban en un clúster que no les corresponde.

Si reducimos la dificultad del problema, creando un dataset con las mediciones de un solo sensor, concentraciones por debajo de 100, y utilizando el primer batch, obtenemos los gráficos de las Figuras ??

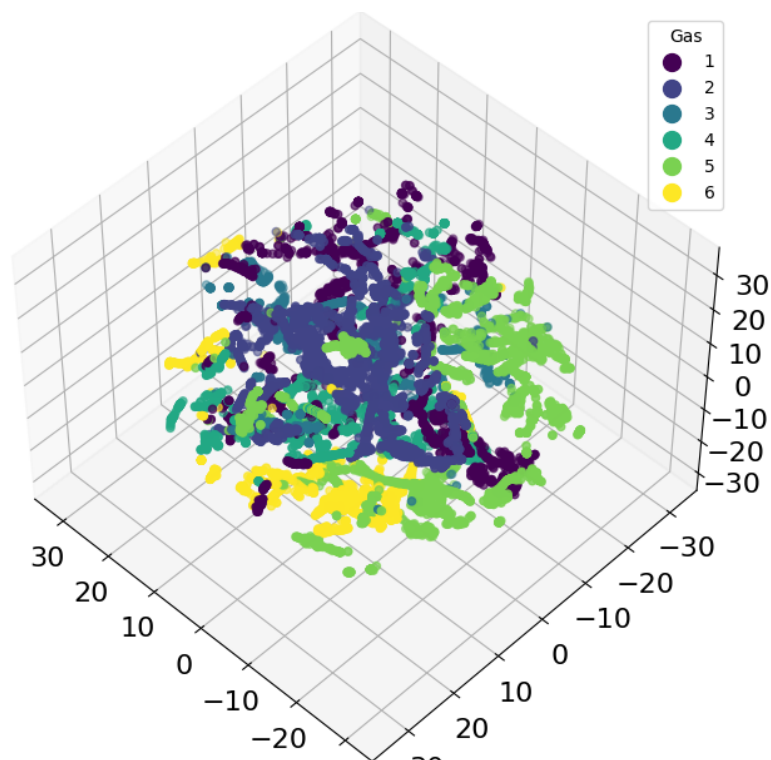


Figura 4.10: Resultados TSNE. Se han coloreado los puntos según a qué gas pertenece. Se observa que no es capaz de separar los clusters de forma unívoca para cada medición.

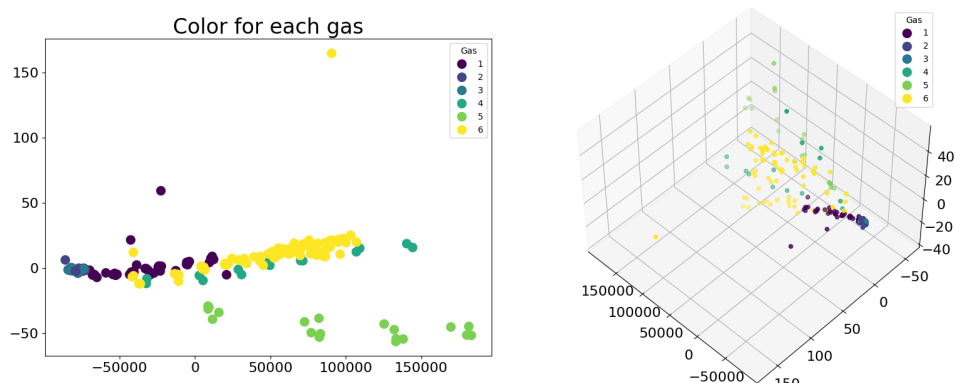


Figura 4.11: Resultados PCA para los datos del batch 1, sensor1 y concentraciones por debajo de 100ppmv. Se han coloreado los puntos según a qué gas pertenece. A la izq se ha reducido la dimensionalidad a 2d, y a la derecha a 3d. Los fases aparecen en clusters bien diferenciados

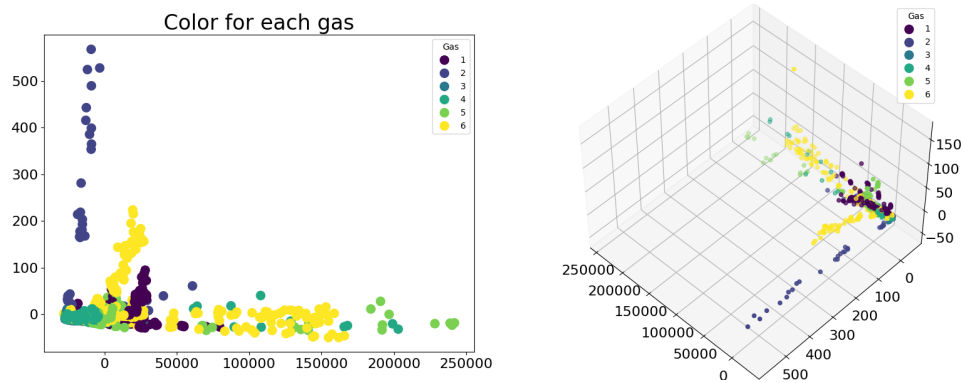


Figura 4.12: Resultados PCA para los datos del batch 1 y 10, sensor1 y concentraciones por debajo de 100ppmv. Se han coloreado los puntos según a qué gas pertenece. A la izq se ha reducido la dimensionalidad a 2d, y a la drecha a 3d. Los puntos que han aparecido con respecto a la imagen anterior no se han agrupado con el cluster del batch 10, si no que han generado una nueva rama, lo que nos indica que son lo suficiente diferentes como para estar agrupadas aparte.

## Capítulo 5

# Conclusiones y planes de mejora

La deriva o drift en los sensores tiene un efecto muy negativo en la capacidad de predicción de los modelos de regresión, que no puede obviarse. En el dataset de estudio, *UCI Gas Sensor Array Drift Dataset Data Set* hemos comprobado tanto con redes neuronales, randomForest o LightGBM que su precisión se ve drásticamente reducido debido al drift.

Los métodos de clasificación basados en redes neuronales son lentos en entrenar, y su accuracy se va reduciendo conforme nos alejamos de los datos de entrenamiento.

Los métodos basados en RandomForest o LightGBM entrenan con mucha rapidez, pero no son inmunes al efecto del drift y el accuracy desciende con mediciones distantes entre sí. Todos los métodos anteriormente mencionados fallan en precisión a causa del drift, no en accuracy, catalogando unos gases de forma errónea en la categoría de otro gas.

Los métodos de aprendizaje no supervisado básicos que se han probado, KMeans y TSNE tienen un rendimiento mucho peor que las redes neuronales o los RF o LGBM. TSNE es muy costoso computacionalmente.

Este trabajo ha puesto de manifiesto la complejidad del problema de clasificación. Es necesario un estudio de variables, que no se ha realizado en este trabajo, para poder generar un modelo únicamente con variables independientes no correlacionadas entre sí. Se ha analizado que los sensores son diferentes entre sí, y que dentro del mismo tipo tienen calibraciones diferentes que los hacen insensibles a ciertos gases.

En (Zhao y cols., 2019) han propuesto un modelo ensemble basado en SVM y LSTM para reducir el efecto del drift en la pérdida de precisión. En un próximo trabajo con más tiempo nos gustaría tratar de reproducir sus resultados.

## Apéndice A

# Código utilizado

### A.1 Código Exploratory Data Analysis

```
1 """
2 Exploratory Data Analysis.
3 Para explorar el dataset de gases, vamos a representar
4     - Cuantos gases hay en cada batch
5     - calcular tabla conteo de:
6         - batch vs gases
7         - Cuantas muestras hay para cada gas.
8 """
9 import pandas as pd
10 import matplotlib.pyplot as plt
11
12 from python.LoadUciData import load_data, load_data_scaled,
13     calculate_bins_concentration
14 from python.StandardFigure import save_figure
15 pd.set_option('display.max_columns', 10)
16
17 def plot_count_per_batch_and_gas(df_gas):
18     props = df_gas.groupby("Batch ID")['GAS'].value_counts(normalize=
19 False).unstack()
20     ax = props.plot(kind='bar', stacked='True', figsize=(8, 6))
21     ax.set_ylabel('Number of samples')
22     return plt.gcf()
23
24 def plot_sample_count_per_gas(df_gas):
25     props = df_gas.groupby('GAS')['GAS'].value_counts(normalize=False
26 ).unstack()
27     ax = props.plot(kind='bar', stacked='True', figsize=(8, 6))
28     ax.set_ylabel('Number of samples of each gas')
29     return plt.gcf()
```

```

29
30
31 def concentration_plot_count(ax, df_gas, gas=1):
32     df = df_gas.copy()
33     df_signal = df[df['GAS'] == gas]
34     df_signal = df_signal.drop(columns=['GAS', 'Batch ID'])
35     df_in = df_signal.groupby(by='CONCENTRATION')
36
37     #ax = df_in.count().plot(style='.-', ax=ax, color='b')
38     ax = df_in.count().plot.bar(ax=ax, color='blue')
39     ax.get_legend().remove()
40     ax.title.set_text('GAS ' + str(gas))
41
42
43 if __name__ == '__main__':
44
45     # Load .dat files
46     df_gas = load_data()
47
48     ## Tables
49     # Show samples per Batch and Gas
50     gas_batch_group = pd.crosstab(df_gas['Batch ID'], df_gas['GAS']).
sort_index()
51     print('\n', gas_batch_group.to_markdown())
52
53     # Show concentration range statistics per GAS
54     pivot = pd.pivot_table(df_gas,
55                             index=['GAS'],
56                             values='CONCENTRATION',
57                             aggfunc=['min', 'max', 'mean', 'std', '
count'])
58     pivot.round(2)
59     print('\n', pivot.round(2).to_markdown())
60
61     # Calculate which concentration is more common for each gas
62     df_c = df_gas[['GAS', 'CONCENTRATION']].value_counts()
63     df_c1 = df_c.reset_index()
64     df_c1 = df_c1.rename(columns={0: 'count'})
65     idx = df_c1.groupby(['GAS'])['count'].transform(max) == df_c1['
count']
66     result = df_c1[idx].sort_values(by='GAS')
67     print(result)
68
69     ## Plots
70
71     # Show samples count per GAS
72     fig = plot_count_per_batch_and_gas(df_gas)
73     save_figure(fig, 'Step0_Count_Batch_Gas')
74     plt.show()

```

```

75 fig = plot_sample_count_per_gas(df_gas)
76 save_figure(fig, 'Step0_Count_Gas')
77 plt.show()
78
79
80 ## Concentration plot (takes time to plot)
81 fig, axes = plt.subplots(3, 2, figsize=(25, 20))
82 fig.suptitle('Count of measurements of each Gas and concentration
83 ')
84 for i, ax in enumerate(axes.flatten(), start=1):
85     print(i)
86     concentration_plot_count(ax, df_gas, gas=i)
87 plt.tight_layout()
88 plt.show()
89 save_figure(fig, 'Step0_Concentration Distribution per gas')
90
91 # Group concentration
92 df_gas['ConcentrationCat'] = pd.cut(df_gas['CONCENTRATION'], bins
93 =range(0, 1000, 100))
94
95 fig, axes = plt.subplots(3, 2, figsize=(20, 20))
96 fig.suptitle('Count of measurements of each Gas and concentration
97 ')
98 for gas, ax in enumerate(axes.flatten(), start=1):
99     print(gas)
100     df = df_gas.copy()
101     df_signal = df[df['GAS'] == gas]
102     df_signal = df_signal.drop(columns=['GAS', 'Batch ID', '
103 CONCENTRATION'])
104     df_in = df_signal.groupby(by='ConcentrationCat')
105
106     #ax = df_in.count().plot(style='.-', ax=ax, color='b')
107     ax = df_in.count().plot.bar(ax=ax, color='blue')
108     ax.get_legend().remove()
109     ax.title.set_text('GAS ' + str(gas))
110 plt.show()
111 save_figure(fig, 'Step0_Concentration Distribution per gas_binned
112 ')
113
114
115 ## Para el gas 2, concentracion 50, plotear la señal de un sensor
116 en los
117 # diferentes batch
118 # Escalamos las muestras
119 df_sca_gas = load_data_scaled()
120 df_sca_gas = calculate_bins_concentration(df_sca_gas)
121
122 #Representamos las medias
123 fig, axes = plt.subplots(3, 4, figsize=(20, 20))
124 fig.subplots_adjust(top=0.8)

```

```

118     #fig.suptitle('Evolution of Gas2-Concentration50 in every batch')
119
120     for batch, ax in enumerate(axes.flatten(), start=1):
121         if batch > 10:
122             break
123         print(batch)
124         df = df_sca_gas.copy()
125         df_signal = df[(df['GAS'] == 2) & (df['Batch ID'] == batch)]
126
127         df_signal = df_signal.reset_index().drop(columns=['
ConcentrationCat'])
128         df_signal = df_signal.iloc[:, :8]
129
130         # ax = df_in.count().plot(style='.-', ax=ax, color='b')
131         ax = df_signal.mean().plot(ax=ax, color='blue')
132         #ax.get_legend().remove()
133         ax.title.set_text('Batch ' + str(batch))
134         ax.set_ylim([-1, 1])
135     fig.subplots_adjust(top=2)
136     plt.tight_layout()
137     plt.show()
138     save_figure(fig, 'Step0_Evolution_of_signal_sensor1_mean')
139
140     # Representamos las muestras
141     fig, axes = plt.subplots(3, 4, figsize=(20, 20))
142     #fig.suptitle('Evolution of Gas2-Concentration50 in every batch')
143     fig.subplots_adjust(top=1.2)
144     for batch, ax in enumerate(axes.flatten(), start=1):
145
146         print(batch)
147         df = df_sca_gas.copy()
148         df_signal = df[(df['GAS'] == 2) & (df['Batch ID'] == batch)]
149         df_signal = df_signal.set_index('ConcentrationCat').loc[50]
150         df_signal = df_signal.reset_index().drop(columns=['
ConcentrationCat'])
151         df_signal = df_signal.iloc[:, :8]
152
153         # ax = df_in.count().plot(style='.-', ax=ax, color='b')
154         ax = df_signal.plot(ax=ax)
155         ax.get_legend().remove()
156         ax.title.set_text('Batch ' + str(batch))
157         ax.set_ylim([-1, 1])
158         if batch==10:
159             break
160         handles, labels = ax.get_legend_handles_labels()
161         lgd = fig.legend(handles, labels, bbox_to_anchor=(2, 0), loc='
lower right')
162         lgd.FontSize = 18;
163         plt.subplots_adjust(left=0.07, right=0.93, wspace=0.25, hspace

```



```
=0.35)
164
165 plt.tight_layout()
166 plt.show()
167 save_figure(fig, 'Step0_Evolution_of_signal_sensor1')
```

Listing A.1: Realiza el analisis exploratorio de los datos disponibles

## A.2 Código Red neuronal secuencial

```
1 import tensorflow as tf
2 from tensorflow import keras
3 from tensorflow.python.keras.callbacks import TensorBoard
4 from time import time
5 import os
6 import pandas as pd
7 from abc import ABC, abstractmethod
8 from sklearn.model_selection import train_test_split
9
10
11 class SeqModel:
12     """ Sequential Neutal Net."""
13     FOLDER = 'models/sequential'
14
15     def __init__(self):
16         pass
17
18     def save_model(self, name):
19         self.model.save(os.path.join(self.FOLDER, name))
20
21     def load_model(self, name):
22         self.model = keras.models.load_model(os.path.join(self.FOLDER
23 , name))
24
25     def _gen_model_seq(self):
26         model = keras.Sequential([
27             keras.layers.Flatten(input_shape=(self.x_size, 1)),
28             keras.layers.Dense(32, activation='relu'),
29             keras.layers.Dense(32, activation='relu'),
30             keras.layers.Dense(self.y_size)
31         ])
32         return model
33
34     def _gen_and_complile_model(self):
35         model = self._gen_model_seq()
36         print(model.summary())
37         model.compile(optimizer='adam',
38                       loss=tf.keras.losses.CategoricalCrossentropy(
```

```

38         from_logits=True),
39         metrics=['accuracy'])
40     return model
41
42     def model_train(self, X_train, y_train):
43         # TensorFlow and tf.keras
44         self.x_size = X_train.shape[1]
45         self.y_size = y_train.shape[1]
46         model = self._gen_and_compile_model()
47         tb = TensorBoard(log_dir="logs/{}".format(time()))
48         model.fit(X_train, y_train, epochs=50, callbacks=[tb])
49         self.model = model
50
51     def model_evaluate(self, X_test, y_test):
52         model = self.model
53         test_loss, test_acc = model.evaluate(X_test, y_test, verbose
54 =2)
55         print('\nTest accuracy:', test_acc)
56         return test_loss, test_acc
57
58     def get_model(self):
59         return self.model
60
61 class AbstractSequentialModel(ABC):
62     @abstractmethod
63     def split_data(self):
64         return self.X_train, self.X_test, self.y_train, self.y_test
65
66     def train_and_save_model(self, modelname):
67         seq = SeqModel()
68         seq.model_train(self.X_train, self.y_train)
69         seq.save_model(modelname)
70
71
72 class SeqModelSimple(AbstractSequentialModel):
73
74     def __init__(self, df):
75         self.df = df
76
77     def split_data(self):
78         ## First, we will NOT use concentration data
79         df_gas = self.df
80         gas_X = df_gas.drop(columns=['Batch ID', 'GAS', '
81 CONCENTRATION']).to_numpy()
82         gas_y = pd.get_dummies(df_gas['GAS'], drop_first=False)
83         self.X_train, self.X_test, self.y_train, self.y_test =
84         train_test_split(gas_X, gas_y, test_size=0.33, random_state=42)

```

```

83         return self.X_train, self.X_test, self.y_train, self.y_test
84
85
86 class SeqModelWithConcentration(AbstractSequentialModel):
87
88     def __init__(self, df):
89         self.df = df
90
91     def split_data(self):
92         ## Use concentration data
93         df_gas = self.df
94         gas_X = df_gas.drop(columns=['Batch ID', 'GAS']).to_numpy()
95         gas_y = pd.get_dummies(df_gas['GAS'], drop_first=False)
96         self.X_train, self.X_test, self.y_train, self.y_test =
train_test_split(gas_X, gas_y, test_size=0.33, random_state=42)
97         return self.X_train, self.X_test, self.y_train, self.y_test
98
99
100 if __name__ == '__main__':
101     model = keras.Sequential([
102         keras.layers.Flatten(input_shape=(128, 1)),
103         keras.layers.Dense(32, activation='relu'),
104         keras.layers.Dense(32, activation='relu'),
105         keras.layers.Dense(6)
106     ])
107
108     model.summary()

```

Listing A.2: Red neuronal secuencial

```

1  """
2  Comprobamos el rendimiento de las redes neuronales para realizar la
3  tarea de clasificacion
4  """
5
6  import numpy as np
7  import pandas as pd
8
9  import tensorflow as tf
10
11 from sklearn.metrics import confusion_matrix
12
13 import matplotlib.pyplot as plt
14 import seaborn as sns
15
16 from python.LoadUciData import load_data
17 from python.StandardFigure import save_figure
18 from python.SeqModel import SeqModel, SeqModelSimple,

```

```
SeqModelWithConcentration
19
20
21 def plot_conf(model, X_test, y_test):
22
23     # Plot the confusion matrix
24     f = plt.figure(figsize=(8, 6))
25     # First add a Softmax layer to get probabilities.
26     probability_model = tf.keras.Sequential([model, tf.keras.layers.
Softmax()])
27     predictions = probability_model.predict(X_test)
28     y_test_values = np.argmax(y_test.values, axis=1)
29     gas_predictions = np.argmax(predictions, axis=1)
30     confusion = confusion_matrix(y_test_values, gas_predictions)
31
32     label_gas = ['Gas1', 'Gas2', 'Gas3', 'Gas4', 'Gas5', 'Gas6']
33     df_conf = pd.DataFrame(data=confusion, columns=label_gas, index=
label_gas)
34     sns.heatmap(df_conf, annot=True, fmt="d", cmap='Blues')
35     plt.title('Confusion Matrix')
36     plt.yticks(rotation=0)
37     plt.show()
38     return f
39
40
41 if __name__ == '__main__':
42
43     # Check the results for the sequential Neural Net
44     # Load data
45     df_gas = load_data()
46
47     mod1 = SeqModelSimple(df_gas)
48     X_train, X_test, y_train, y_test = mod1.split_data()
49     #mod1.train_and_save_model('ModelSimple')
50
51     mod2 = SeqModelWithConcentration(df_gas)
52     X_train2, X_test2, y_train2, y_test2 = mod2.split_data()
53     #mod2.train_and_save_model('ModelWithConcentration')
54
55
56     seq = SeqModel()
57     seq.load_model('ModelSimple')
58     f = plot_conf(seq.get_model(), X_test, y_test)
59     save_figure(f, 'ConfMatrix_ModelSimple')
60
61     seq = SeqModel()
62     seq.load_model('ModelWithConcentration')
63     f = plot_conf(seq.get_model(), X_test2, y_test2)
```

```
64 save_figure(f, 'ConfMatrix_ModelWithConcentration')
```

Listing A.3: Red neuronal secuencial para clasificar gases

### A.3 Código LGBM

```
1 import lightgbm as lgb
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 import shap
5 import pandas as pd
6 import numpy as np
7
8 from python.LoadUciData import load_data
9 from python.LoadSensorData import sensor_features_column,
   sensor_type_dict
10 from python.StandardFigure import save_figure
11
12 from sklearn.model_selection import train_test_split
13 from sklearn.metrics import confusion_matrix
14
15
16 def plot_conf(conf):
17     fig = plt.figure(figsize=(8, 8))
18     ax = sns.heatmap(conf, annot=True, fmt=".02f", cmap='Blues')
19     plt.yticks(rotation=0)
20     plt.show()
21     return fig
22
23
24 def train_lgbm(X_train, y_train):
25     lgbm = lgb.LGBMClassifier(objective='multiclass', random_state=5)
26     lgbm.fit(X_train, y_train)
27     return lgbm
28
29
30 def lgbm_conf_shap(X_train, X_test, y_train, y_test):
31     lgbm = train_lgbm(X_train, y_train)
32     y_pred = lgbm.predict(X_test)
33     conf = confusion_matrix(y_test, y_pred)
34     conf_percent = conf.T/y_test.value_counts().sort_index().values
35     plot_conf(conf_percent)
36
37     # explain the model's predictions using SHAP
38     # (same syntax works for LightGBM, CatBoost, scikit-learn and
   spark models)
39     print('Start shapely')
40     model = lgbm
```

```

41 explainer = shap.TreeExplainer(model)
42 shap_values = explainer.shap_values(X_train)
43
44 fig = plt.figure()
45 plt.title("SHAP VALUES")
46 shap.summary_plot(shap_values, X_train, plot_type="bar")
47 return fig
48
49
50 if __name__ == '__main__':
51
52     df = load_data()
53
54     # Keep only one pair of Gas and Concentration
55     """
56     | GAS      | CONCENTRATION | count |
57     |-----:|-----:|-----:|
58     | 1        | 50            | 488   |
59     | 2        | 50            | 588   |
60     | 3        | 200           | 177   |
61     | 4        | 100           | 357   |
62     | 5        | 50            | 485   |
63     | 6        | 50            | 345   |
64     """
65     # Get count of each pair value
66     df_c = df[['GAS', 'CONCENTRATION']].value_counts()
67     df_c1 = df_c.reset_index()
68     df_c1 = df_c1.rename(columns={0: 'count'})
69     idx = df_c1.groupby(['GAS'])['count'].transform(max) == df_c1['
count']
70     result = df_c1[idx].sort_values(by='GAS')
71     print(result)
72
73     # Total of measurements we are using is 2440.
74
75     dict_gas_concentration = {1: 50,
76                             2: 50,
77                             3: 200,
78                             4: 100,
79                             5: 50,
80                             6: 50}
81
82     df_temp = pd.DataFrame()
83     for g, c in dict_gas_concentration.items():
84         df_select = df[(df['GAS'] == g) & (df['CONCENTRATION'] == c)]
85         df_temp = df_temp.append(df_select)
86
87     # Select only one sensor
88     X = df_temp.iloc[:, :8]

```

```
89     y = df_temp['GAS']
90
91
92     X_train, X_test, y_train, y_test = train_test_split(X, y,
93                                                         test_size
94                                                         =0.3,
95                                                         random_state
96                                                         =42)
97     fig = lgbm_conf_shap(X_train, X_test, y_train, y_test)
98     save_figure(fig, 'Step4_LGBM_one_sensor')
99
100     ### Now let's use all sensor data
101     # Select all sensors
102     X = df_temp.iloc[:, :128]
103     y = df_temp['GAS']
104
105     X_train, X_test, y_train, y_test = train_test_split(X, y,
106                                                         test_size
107                                                         =0.3,
108                                                         random_state
109                                                         =42)
110     fig = lgbm_conf_shap(X_train, X_test, y_train, y_test)
111
112     ### Okay, now let's train with batch 1 to 3, and predict batch 4
113     df_temp_train = df_temp[df_temp['Batch ID'].isin
114                             ([1,2,3,4,5,6,7,8])]
115     df_temp_test = df_temp[df_temp['Batch ID'].isin([9])]
116
117     X_train = df_temp_train.iloc[:, :128]
118     y_train = df_temp_train['GAS']
119
120     X_test = df_temp_test.iloc[:, :128]
121     y_test = df_temp_test['GAS']
122
123     lgbm_conf_shap(X_train, X_test, y_train, y_test)
124     save_figure(fig, 'Step4_LGBM_all_data')
125
126     # Now we get that the drift has changed the measurement of gas 3
127     # and 6,
128     # and the model get totally confuse. It still precise, but not
129     # accurate.
130
131     # maybe is because, as shown the shapely plot, it decision is
132     # based only in a few features
133     X = df_temp.iloc[:, :128]
134     y = df_temp['GAS']
135
136     X_train, X_test, y_train, y_test = train_test_split(X, y,
137                                                         test_size
```

```

=0.33,
130                                     random_state
=42)
131     lgbm_conf_shap(X_train, X_test, y_train, y_test)
132
133     ### Okay, now let's train with batch 1 to 5, and predict batch 6
134     df_temp_train = df_temp[df_temp['Batch ID'].isin([1, 2, 3, 4, 5])
135 ]
136     df_temp_test = df_temp[df_temp['Batch ID'].isin([6])]
137
138     X_train = df_temp_train.iloc[:, :128]
139     y_train = df_temp_train['GAS']
140
141     X_test = df_temp_test.iloc[:, :128]
142     y_test = df_temp_test['GAS']
143
144     lgbm_conf_shap(X_train, X_test, y_train, y_test)
145
146     # maybe if we train 4 models, one for each type of sensor, and
147     # get a votation
148     range_cols_typeA = [sensor_features_column(n) for n in
149 sensor_type_dict['A']]
150     idx_cols = np.r_[range_cols_typeA].ravel()
151
152     df_sA = df_temp.iloc[:, idx_cols]
153     df_feat = df_temp[['GAS', 'CONCENTRATION', 'Batch ID']]
154     dfA_full = pd.concat([df_sA, df_feat], axis=1)
155
156     df_A_train = dfA_full[dfA_full['Batch ID'].isin([1, 2, 3, 4,
157 5,6,7,8])]
158     df_A_test = dfA_full[dfA_full['Batch ID'].isin([9])]
159
160     X_train = df_A_train.drop(columns= ['GAS', 'CONCENTRATION', '
161 Batch ID'])
162     X_test = df_A_test.drop(columns= ['GAS', 'CONCENTRATION', 'Batch
163 ID'])
164
165     y_train = df_A_train['GAS']
166     y_test = df_A_test['GAS']
167
168     lgbmA = train_lgbm(X_train, y_train)
169     y_pred = lgbmA.predict(X_test)
170     lgbmA.score(X_test, y_test)
171     confA = confusion_matrix(y_test, y_pred)
172
173     # tendria que elegir las concetracion por rangos.

```

Listing A.4: Red neuronal secuencial para clasificar gases



## A.4 Codigo Randonforest

```
1 import os
2 from sklearn.ensemble import RandomForestClassifier
3 from sklearn.model_selection import GridSearchCV
4 import numpy as np
5 import joblib
6
7 class RandomForestMod:
8     """ RandomForestClasificador."""
9
10    FOLDER = 'models/randomforest'
11
12    def __init__(self):
13        pass
14
15    def save_model(self, name):
16        joblib.dump(self.model, os.path.join(self.FOLDER, name))
17
18    def load_model(self, name):
19        self.model = joblib.load(os.path.join(self.FOLDER, name))
20
21    def _gen_model_base(self):
22        model = RandomForestClassifier(max_depth=2, random_state=0)
23        return model
24
25    def _gen_model(self):
26        model = RandomForestClassifier(max_depth=80,
27                                     max_features=3,
28                                     min_samples_leaf=3,
29                                     min_samples_split=12,
30                                     n_estimators=200,
31                                     random_state=0)
32        return model
33
34    def model_search(self, X_train, y_train):
35        model = self._gen_model_base()
36
37        param_grid = {
38            'bootstrap': [True],
39            'max_depth': [80, 90, 100, 110],
40            'max_features': [2, 3],
41            'min_samples_leaf': [3, 4, 5],
42            'min_samples_split': [8, 10, 12],
43            'n_estimators': [100, 200, 300, 1000]}
44        # Instantiate the grid search model
45        grid_search = GridSearchCV(estimator=model,
46                                   param_grid=param_grid,
```

```

47         cv=3, n_jobs=-1, verbose=2)
48     grid_search.fit(X_train, y_train)
49     grid_search.best_params_
50     self.best_grid = grid_search.best_estimator_
51     self.model = model
52
53     def model_train(self, X_train, y_train):
54         model = self._gen_model()
55         model.fit(X_train, y_train)
56         self.model = model
57         return model
58
59     def evaluate(self, test_features, test_labels):
60         predictions = self.model.predict(test_features)
61         errors = abs(predictions - test_labels)
62         mape = 100 * np.mean(errors / test_labels)
63         accuracy = 100 - mape
64         print('Model Performance')
65         print('Average Error: {:.4f} degrees.'.format(np.mean(errors
)))
66         print('Accuracy = {:.2f}%.'.format(accuracy))
67
68         return accuracy
69
70     def model_evaluate(self, X_test, y_test):
71         grid_accuracy = self.evaluate(self.best_grid, X_test, y_test
)
72         return grid_accuracy
73
74     def get_model(self):
75         return self.model
76
77
78 if __name__ == '__main__':
79     rf = RandomForestMod()

```

Listing A.5: Clases para cargar datos en memoria

```

1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 import tensorflow as tf
6 from sklearn.model_selection import train_test_split
7 from sklearn.metrics import confusion_matrix
8
9 from python.LoadUciData import load_data
10 from python.LoadSensorData import get_sensors_list, get_sensor
11 from python.StandardFigure import save_figure

```

```

12 from python.RandomForestClasification import RandomForestMod
13
14
15 if __name__ == '__main__':
16
17     #Load data
18     df = load_data()
19
20     df = get_sensors_list([0, 1, 8, 9])
21     df = get_sensors_list([2, 3, 10, 11])
22     df = get_sensors_list([4, 5, 12, 13])
23     df = get_sensors_list([6, 7, 14, 15])
24
25     for col in ['GAS']:
26         df[col] = df[col].astype('category')
27
28     df['is_train'] = np.where(np.logical_and(df['Batch ID'] >= 1, df[
29 'Batch ID'] <= 8), True, False)
30     df['is_test'] = np.where(np.logical_and(df['Batch ID'] >= 9, df[
31 'Batch ID'] <= 9), True, False)
32
33     df_train = df[df['is_train'] == True]
34     df_test = df[df['is_test'] == True]
35
36     X_train = df_train.drop(columns=['GAS', 'Batch ID', '
37 CONCENTRATION'])
38     y_train = df_train['GAS']
39
40     X_test = df_test.drop(columns=['GAS', 'Batch ID', 'CONCENTRATION'
41 ])
42     y_test = df_test['GAS']
43
44     #X_train, X_test, y_train, y_test = train_test_split(X, y,
45 test_size=0.33)
46
47
48     rfm = RandomForestMod()
49     rfm.model_train(X_train.values, y_train.values)
50     rfm.save_model('RF1')
51
52     rf = rfm.get_model()
53     rf.score(X_test, y_test)
54     y_pred = rf.predict(X_test)
55     conf = confusion_matrix(y_test, y_pred)
56     conf_percent = conf.T/y_test.value_counts().sort_index().values
57
58     fig = plt.figure(figsize=(8, 8))
59     ax = sns.heatmap(conf_percent, annot=True, fmt=".02f", cmap='

```

```
Blues')
56 plt.yticks(rotation=0)
57 plt.show()
58
59 from sklearn import metrics
60 # Model Accuracy, how often is the classifier correct?
61 print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

Listing A.6: Clases para cargar datos en memoria

## A.5 Código aprendizaje no supervisado

```
1 """
2 Use PCA, Kmeans and TSNE to reduce the dimensionality and classify
   gases.
3 """
4
5 import seaborn as sns
6 import matplotlib.pyplot as plt
7 from mpl_toolkits.mplot3d import Axes3D
8
9 from sklearn.preprocessing import StandardScaler
10 from sklearn.cluster import KMeans
11 from sklearn.decomposition import PCA
12 from sklearn.manifold import TSNE
13 from sklearn.metrics import confusion_matrix
14
15 from python.LoadUciData import load_data,
   calculate_bins_concentration
16 from python.StandardFigure import save_figure
17 from python.Step0_1_DataExploration import get_sensors_array
18
19
20 def apply_KMeans_2d(X,y, name):
21     pca = PCA(n_components=2)
22     pca.fit(X, y)
23     xp = pca.transform(X)
24
25     number_of_clusters = 6
26     km = KMeans(n_clusters=number_of_clusters)
27     # Normally people fit the matrix
28     y_pred = km.fit_predict(X)
29     #igualamos las categorias a los indices 1 al 6 del dataframe
30     y_pred = y_pred + 1
31
32     #Plots
33     fig, ax = plt.subplots(figsize=(8, 6))
34     ax.set_title('Color for each cluster')
```

```

35 scatter = ax.scatter(xp[:, 0], xp[:, 1], c=y_pred)
36 legend1 = ax.legend(*scatter.legend_elements(),
37                     loc="upper right", title="Clusters")
38 plt.show()
39 save_figure(fig, f'Step0_3_Color for each cluster_2d_{name}')
40
41 fig, ax = plt.subplots(figsize=(8, 6))
42 ax.set_title('Color for each gas')
43 scatter = ax.scatter(xp[:, 0], xp[:, 1], c=y)
44 legend1 = ax.legend(*scatter.legend_elements(),
45                     loc="upper right", title="Gas")
46 plt.show()
47 save_figure(fig, f'Step0_3_Color for each gas_{name}')
48
49 fig = plt.figure(3, figsize=(8, 6))
50 conf = confusion_matrix(y, y_pred)
51 sns.heatmap(conf, annot=True, fmt='d')
52 plt.title(f"Confusion matrix_{name}")
53 plt.show()
54
55
56 def apply_KMeans_3d(X,y, name):
57     pca = PCA(n_components=3)
58     pca.fit(X, y)
59     xp = pca.transform(X)
60
61     number_of_clusters = 6
62     km = KMeans(n_clusters=number_of_clusters)
63     # Normally people fit the matrix
64     y_pred = km.fit_predict(X)
65     #igualamos las categorias a los indices 1 al 6 del dataframe
66     y_pred = y_pred + 1
67
68     fig = plt.figure(figsize=(8,6))
69     plt.clf()
70     ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
71     scatter = ax.scatter(xp[:, 0], xp[:, 1], xp[:, 2], c=y_pred)
72     legend1 = ax.legend(*scatter.legend_elements(), loc="upper right",
73                         title="Clusters")
74     plt.show()
75     save_figure(fig, f'Step0_3_Color for each cluster_3d_{name}')
76
77     fig = plt.figure(figsize=(8, 6))
78     ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
79     scatter = ax.scatter(xp[:, 0], xp[:, 1], xp[:, 2], c=y)
80     legend1 = ax.legend(*scatter.legend_elements(), loc="upper right",
81                         title="Gas")
82     plt.show()
83     save_figure(fig, f'Step0_3_Color for each gas_3d_{name}')

```

```

82
83 fig = plt.figure(3, figsize=(8, 6))
84 conf = confusion_matrix(y, y_pred)
85 sns.heatmap(conf, annot=True, fmt='d')
86 plt.title(f"Confusion matrix_{name}")
87 plt.show()
88
89
90 def apply_tsne(X, y, name):
91     print('tsne2d')
92     X_embedded = TSNE(n_components=2).fit_transform(X)
93     fig, ax = plt.subplots(figsize=(12, 8))
94     ax.set_title('TSNE 2d Batch1, Sensor1, Concentration less 100ppmv')
95     scatter = ax.scatter(X_embedded[:, 0], X_embedded[:, 1], c=y,
96 label=y.unique())
97     legend1 = ax.legend(*scatter.legend_elements(), loc="upper right",
98 title="Gas")
99     plt.show()
100     save_figure(fig, f'Step0_3_TSNE_2d_{name}')
101
102     print('tsne3d')
103     X_embedded = TSNE(n_components=3).fit_transform(X)
104     fig = plt.figure(figsize=(8, 6))
105     ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azimuth=134)
106     ax.scatter(X_embedded[:, 0], X_embedded[:, 1], X_embedded[:, 2],
107 c=y)
108     ax.set_title('TSNE 3d Batch1, Sensor1, Concentration less 100ppmv')
109     legend1 = ax.legend(*scatter.legend_elements(), loc="upper right",
110 title="Gas")
111     plt.show()
112     save_figure(fig, f'Step0_3_TSNE_3d_{name}')
113
114
115 if __name__ == '__main__':
116     # Load data
117     df = load_data()
118
119     #keep only values of concentration less or equal than 100
120     df = df[df['CONCENTRATION'] <= 100]
121
122     #Keep only first batch
123     df = df[df['Batch ID'] == 1]
124
125     # keep only data from sensor1
126     X = df.iloc[:, :8]
127     y = df['GAS']

```

```
125 apply_KMeans_2d(X, y, 'Batch1_Sensor1_Conc less 100ppmv')
126 apply_KMeans_3d(X, y, 'Batch1_Sensor1_Conc less 100ppmv')
127 apply_tsne(X, y, 'Batch1_Sensor1_Conc less 100ppmv')
128
129
130 ## Now lets keep batch 1 and 10
131 # Load data
132 df = load_data()
133 df = df[df['CONCENTRATION'] <= 100]
134 df = df[df['Batch ID'].isin([1,10])]
135
136 # keep only data from sensor1
137 X = df.iloc[:, :8]
138 y = df['GAS']
139
140 apply_KMeans_2d(X, y, 'Batch1and10_Sensor1_Conc less 100ppmv')
141 apply_KMeans_3d(X, y, 'Batch1and10_Sensor1_Conc less 100ppmv')
142 apply_tsne(X, y, 'Batch1_Sensor1_Conc less 100ppmv')
143
144 ### Now all the data available
145 # Load data
146 df = load_data()
147
148 # keep only data from sensor1
149 X = df.iloc[:, :128]
150 y = df['GAS']
151
152 apply_KMeans_2d(X, y, 'All data')
153 apply_KMeans_3d(X, y, 'All data')
154 apply_tsne(X, y, 'All data')
```

Listing A.7: Clases para cargar datos en memoria

## A.6 Código Utilities

```
1 import os
2 import pandas as pd
3 import re
4 import numpy as np
5 from python.FileUtils import get_list_of_files_with_extension
6 from sklearn.preprocessing import StandardScaler
7
8 class LoadDatFile:
9     """
10     This class aims to load a .dat files from UCI
11     https://archive.ics.uci.edu/ml//datasets/Gas+Sensor+Array+Drift+
12     Dataset
13     , and returns a pandas.dataframe object
```

```

13
14     :arg .dat file
15     :return df
16     """
17
18     def __init__(self, file):
19         self.file = file
20
21     @property
22     def batch_number(self):
23         base = os.path.basename(self.file)
24         name, ext = os.path.splitext(base)
25         num = re.findall(r'\d+', name)[0]
26         # num = num.zfill(2)
27         return int(num)
28
29     @property
30     def df(self):
31         df = pd.read_table(self.file, engine='python', sep='\s+\d+: ',
32                             header=None)
33         df['Batch ID'] = self.batch_number
34         return df
35
36 class GasDataFrame:
37     """ Process the .dat file to get all the information contained:
38     - Gas, concentration and measures."""
39
40     def __init__(self, file):
41         self.file = file
42
43     @property
44     def df(self):
45         df_raw = LoadDatFile(self.file).df
46         return self._add_gas_info(df_raw)
47
48     @staticmethod
49     def _add_gas_info(df):
50         df[['GAS', 'CONCENTRATION']] = df.iloc[:, 0].str.split(";",
51                             expand=True, )
52         df.drop(df.columns[0], axis=1, inplace=True)
53         df['GAS'] = df['GAS'].astype('int')
54         df['CONCENTRATION'] = df['CONCENTRATION'].astype('float')
55         return df
56
57 class LoadDatFolder:
58     """
59     This class aims to load all .dat files contained in a folder,

```



```
60     gives each file a GasDataframe format and concats all in a pandas
        .dataframe object with
61
62     :inputs: folder with many .dat files
63     :return df
64     """
65     def __init__(self, folder):
66         self.folder = folder
67
68     @property
69     def df(self):
70         files = get_list_of_files_with_extension(self.folder, 'dat')
71         df_full = pd.DataFrame()
72         for f in files:
73             dftemp = GasDataframe(f).df
74             df_full = df_full.append(dftemp)
75         return df_full
76
77
78 def load_data():
79     folder = r'data_uci/driftdataset'
80     df_gas = LoadDatFolder(folder).df
81
82     #Rename sensor columns
83     col_names_dict = {}
84     i = 1
85     for sensor in range(0, 15 + 1):
86         for feature in range(0, 7 + 1):
87             col_names_dict[i] = f'S{sensor}_{feature}'
88             i = i + 1
89
90     df_gas = df_gas.rename(columns=col_names_dict)
91     return df_gas
92
93
94 def load_data_scaled():
95     # Load data
96     df = load_data()
97
98     # init scaler
99     sc = StandardScaler()
100
101     # Scale only sensor data
102     sensor_features = df.iloc[:, :128]
103     sc.fit(sensor_features)
104     data_sc = sc.transform(sensor_features)
105
106     # Get the unscaled info
107     info = df[['Batch ID', 'GAS', 'CONCENTRATION']].values
```

```
108
109 # Merge scaled data and the info into a pandas dataframe.
110 data = np.concatenate([data_sc, info], axis=1)
111 df_sca_gas = pd.DataFrame(data, columns=df.columns)
112 for col in ['GAS', 'Batch ID']:
113     df_sca_gas[col] = df_sca_gas[col].astype('int').astype('
category')
114 return df_sca_gas
115
116 def calculate_bins_concentration(df):
117     # Create ConcentrationCat column.
118     df['ConcentrationCat'] = pd.cut(df['CONCENTRATION'], bins=range
(0, 1000, 100))
119     return df
120
121
122
123 if __name__ == '__main__':
124     file_data = r'data_uci/driftdataset/batch1.dat'
125     lf = LoadDatFile(file_data)
126     my_dataframe = lf.df
127
128     gdf = GasDataFrame(file_data)
129     my_dataframe_gas = gdf.df
130
131     folder = r'data_uci/driftdataset/'
132     ldf = LoadDatFolder(folder)
133     my_dataframe_full = ldf.df
134
135     my_df_scaled = load_data_scaled()
```

Listing A.8: Clases para cargar datos en memoria

```
1 import pandas as pd
2 import numpy as np
3 from python.LoadUciData import load_data
4
5 sensor_type_dict = {'A': [0, 1, 8, 9],
6                     'B': [2, 3, 10, 11],
7                     'C': [4, 5, 12, 13],
8                     'D': [6, 7, 14, 1]}
9
10 def get_sensor_by_type(letter):
11     """
12     Select the sensors as type, A,B,C or D
13     and returns the sensors features (N*8), Batch ID, GAS and
14     concentration
15     :arg list of integers
16     :return pandas dataframe
```

```
16     """
17     return get_sensors_list(sensor_type_dict[f'{letter}'])
18
19
20 def get_sensors_list(list_of_n):
21     """
22     Get the N sensor features (N*8), Batch ID, GAS and concentration
23     ,
24     by index
25     :arg list of integers
26     :return pandas dataframe
27     """
28
29     df = load_data()
30     sensors_columns = []
31     for n in list_of_n:
32         ix = list(sensor_features_column(n))
33         sensors_columns.extend(ix)
34     #sensors_columns = np.r_[1:10, 15, 17, 50:100]
35     df_features_n = df.iloc[:, sensors_columns]
36
37     df_gas = df[['Batch ID', 'GAS', 'CONCENTRATION']]
38     return pd.concat([df_features_n, df_gas], axis=1)
39
40 def get_sensor(n):
41     """ Get the sensor 8 features, Batch ID, GAS and concentration,
42     by index
43     """
44
45     df = load_data()
46     df_features = get_sensor_features(df, n)
47     df_gas = df[['Batch ID', 'GAS', 'CONCENTRATION']]
48     return pd.concat([df_features, df_gas], axis=1)
49
50 def get_sensor_features(df, n):
51     return df.iloc[:, sensor_features_column(n)]
52
53
54 def sensor_features_column(n):
55     if 0 <= n <= 15:
56         return range(8*n, 8*n + 8)
57     else:
58         return None
59
60
61 def get_sensor_by_col_name(n):
62     """
63     Get the sensor 8 features, Batch ID, GAS and concentration,
```

```
64     by column names
65     """
66     df = load_data()
67     filter_col = [col for col in df if col.startswith(f'S{n}')]
68     df_sensor = df[filter_col]
69     df_gas = df[['Batch ID', 'GAS', 'CONCENTRATION']]
70     return pd.concat([df_sensor, df_gas], axis=1)
71
72
73 if __name__ == '__main__':
74
75     df_sensor_7 = get_sensor(7)
76     df = get_sensors_list([0, 2, 4, 6])
77
78     #Comparamos que ambos metodos son equivalentes
79     df_index = get_sensor(2)
80     df_name = get_sensor_by_col_name(2)
81     print(df_index.equals(df_name))
```

Listing A.9: Clases para cargar datos en memoria

```
1 import os
2
3 def get_list_of_files(folder):
4     """Returns a list with the files contained in that folder"""
5     ls_files = os.listdir(folder)
6     path_files = [os.path.join(folder, f) for f in ls_files]
7     return path_files
8
9 def get_list_of_files_with_extension(folder, ext):
10    """Returns a list with the files with the specified extension {
11    ext} contained in that folder."""
12    path_files = get_list_of_files(folder)
13    ls_ext = [f for f in path_files if f.endswith(ext)]
14    return ls_ext
```

Listing A.10: Clases para cargar datos en memoria



## Referencias

- Vergara, A., Ayhan, T., Vembu, S., Huerta, R., Ryan, M., y Homer, M. (2011, 01). Gas sensor drift mitigation using classifier ensembles.. doi: 10.1145/2003653.2003655
- Vergara, A., Vembu, S., Ayhan, T., Ryan, M. A., Homer, M. L., y Huerta, R. (2012). Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators, B: Chemical*, 166-167, 320–329. Descargado de link doi: 10.1016/j.snb.2012.01.074
- Zhao, H., Li, L., Xiao, W., Meng, Z., Han, y Yu, H. (2019, 09). Sensor drift compensation based on the improved lstm and svm multi-class ensemble learning models. *Sensors*, 19, 3844. doi: 10.3390/s19183844