Estudio comparativo de diferentes modelos de aprendizaje automático y análisis de la robustez de dichos modelos frente a fallas provocadas en los sensores en el caso de detección de incendios para ser aplicados a dispositivos de computación de borde

Carlos Alberto Binker[12], Hugo Tantignone[12], Lautaro Lasorsa[12]
Guillermo Buranits[12], Eliseo Zurdo[12], Maximiliano Frattini[12]
[1]Departamento de Ingeniería e Investigaciones Tecnológicas
[2]Universidad Nacional de La Matanza, Florencio Varela 1903 (B1754JEC) -- San Justo, Provincia de Buenos Aires, Argentina
{cbinker, htantignone, laulasorsa, eazurdo, gburanits, mfrattini }@unlam.edu.ar

Resumen

El aprendizaje automático (machine learning) aplicado a IoT (Internet of Things) brinda a los dispositivos de borde la toma de decisiones sin necesidad de comunicarse con un servidor central, a este paradigma se lo conoce como computación de borde. Para ello es necesario realizar un modelo liviano que pueda ejecutarse en un dispositivo con pocas prestaciones de hardware (tal como Arduino, ESP32, Raspberry, etc), comparado con un servidor centralizado en la nube. En primer lugar se confeccionará un caso de estudio en donde se compararán diferentes modelos de aprendizaje automático para ver su eficacia detectando fuego (utilizando para ello una batería de sensores provenientes de un dataset externo). Concretamente, el caso de estudio que se plantea parte de un dataset tomado de kaggle.com, y a partir del mismo se ensayarán diferentes modelos que predecirán una alarma de incendio dentro de un recinto en función de doce variables de entrada, como ser la temperatura, humedad, dióxido y monóxido de carbono, y varios otros gases relevantes. Posteriormente, a partir de la obtención del mejor modelo, se proveerá de un nuevo valor agregado sustancial a dicho caso de estudio, como ser el de evaluar qué tan bien resisten dichos modelos de aprendizaje automático estudiados a la falla de alguno de los 12 sensores utilizados en nuestro dataset. Este modelo deberá ser implementado utilizando la plataforma de código abierto TensorFlow, que es un framework muy empleado en el desarrollo de modelos para dispositivos IoT, que pueden emplearse para realizar computación de borde.

1. Introducción

1.1 Problemática a resolver

Se planteará un caso de estudio que consiste en dos fases. En la primera fase, la obtención de un modelo empleando técnicas de aprendizaje automático (machine learning [1]), que a partir de un conjunto de datos (dataset) [2] conformado por valores de concentración de gases medidos con una serie de sensores específicos para tal fin, sea capaz de predecir la presencia o no de fuego dentro de un recinto. Por lo tanto, este modelo predictivo poseerá un conjunto de variables de entrada predictoras y una variable de salida dependiente que será la indicación de la existencia o no de fuego. Por lo tanto la salida de nuestro sistema será una variable booleana a la que denominaremos Fire Alarm (ver Figura 1). Pero no sólo se estudiará un único modelo de aprendizaje automático aplicado a nuestro dataset para el detector de fuego mencionado, sino que lo que se pretende también, es probar diferentes modelos alternativos basados tanto en redes neuronales como así también en árboles de decisión (Ramdom Forest) [3] y así entonces poder determinar el mejor modelo en base a estas técnicas mencionadas que mejor se adapte a representar nuestro detector de incendios. Luego en una segunda fase, una vez obtenido el mejor de los modelos para nuestro detector de incendios se evaluará qué tan bien resisten estos diferentes modelos de aprendizaje automático a la falla en alguno de los 12 sensores utilizados en nuestro dataset. A nivel práctico se empleará Python [4] por ser un lenguaje altamente desarrollado para su empleo en machine learning y en el cual se encuentra desarrollado Tensorflow.

Se entiende que la problemática analizada en este estudio es importante debido a que uno de los campos de aplicación de la tecnología IoT es la seguridad, y dentro de esta se ubica la detección de incendios. Los motivos para utilizar dispositivos de borde independientes de un servidor central son claros:

- Los incendios o sus causas por sí mismos pueden impedir dicha comunicación con el servidor, o sencillamente impedir que éste funcione.
- Permite instalar estos dispositivos en vehículos o en instalaciones aisladas, que no posean una conexión estable a un servidor o computador de gran potencia de cómputo.
- Vuelve a los dispositivos de borde en unidades autosuficientes.

- Por otro lado, la robustez frente a fallos en los componentes de los sensores es clave para garantizar la seguridad de los dispositivos empleados. En especial teniendo en cuenta que si estos fallos se detectan en inspecciones periódicas, el dispositivo debe seguir brindando seguridad entre la falla del sensor y su detección.

Figura 1. Modelo predictor de fuego generado por machine learning

1.2 Planteo de los posibles escenarios una vez obtenido el mejor modelo predictor de fuego

En esta segunda fase de nuestro caso de estudio, se implementarán dos posibles escenarios, que a su vez cada uno se dividirá en doce sub escenarios. El primer escenario consistirá en que uno de los doce sensores deje de funcionar abruptamente, y por ende se evaluará la eficacia a partir del mejor modelo resultante obtenido para la primera fase de nuestro estudio, pero en base sólo a las 11 variables restantes. En este caso no se evaluarán más modelos porque habría que hacer una versión de cada modelo para cada conjunto de sensores y eso es computacionalmente muy costoso para los alcances de este estudio. En el segundo escenario, el sensor que falla dará información errónea, pero el modelo no sabrá que éste falla. Se evaluará entonces cómo se comportan en este segundo escenario los modelos de Random Forest, Regresión Logística Simple y una red neuronal más compleja.

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

A continuación se presenta un cuadro representativo del dataset donde constan las variables de entrada y la salida a predecir a la que denominamos Fire Alarm, ver Figura 2.

	Temperature[C]	Humidity[%]	TVOC[ppb]	eCO2[ppm]	Raw H2	Raw Ethanol	Pressure[hPa]	PM1.0	PM2.5	NC0.5	NC1.0	NC2.5	Fire Alarm
0	20.000	57.36	0	400	12306	18520	939.735	0.00	0.00	0.00	0.000	0.000	0
1	20.015	56.67	0	400	12345	18651	939.744	0.00	0.00	0.00	0.000	0.000	0
2	20.029	55.96	0	400	12374	18764	939.738	0.00	0.00	0.00	0.000	0.000	0
3	20.044	55.28	0	400	12390	18849	939.736	0.00	0.00	0.00	0.000	0.000	0
4	20.059	54.69	0	400	12403	18921	939.744	0.00	0.00	0.00	0.000	0.000	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
62625	18.438	15.79	625	400	13723	20569	936.670	0.63	0.65	4.32	0.673	0.015	0
62626	18.653	15.87	612	400	13731	20588	936.678	0.61	0.63	4.18	0.652	0.015	0
62627	18.867	15.84	627	400	13725	20582	936.687	0.57	0.60	3.95	0.617	0.014	0
62628	19.083	16.04	638	400	13712	20566	936.680	0.57	0.59	3.92	0.611	0.014	0
62629	19.299	16.52	643	400	13696	20543	936.676	0.57	0.59	3.90	0.607	0.014	0

62630 rows × 13 columns

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

Como puede observarse se trata de doce variables de entrada. Las mismas son: temperatura en °C, humedad en %, TVOC (Total Volatile Organic Compound, medido en partes por billón). Los compuestos orgánicos son sustancias químicas que contienen carbono y se encuentran en todos los seres vivos, estos compuestos orgánicos volátiles se convierten fácilmente en vapores o gases. Junto con el carbono, contienen elementos como hidrógeno, oxígeno, flúor, cloro, bromo, azufre o nitrógeno. Luego tenemos eCO2 (dióxido de carbono equivalente, medido en partes por millón), Raw H2 (gas Hidrógeno) y Raw Ethanol (gas Etano); el término Raw hace mención de una detección de dichos gases en crudo, es decir sin ningún filtrado o procesamiento, luego tenemos la presión en hectopascales; inmediatamente tenemos PM1.0 y PM2.5 que se trata de la detección de partículas materiales de diámetros de 1 y 2,5 micrómetros o más pequeñas aún. Estas partículas provienen de diversas fuentes, incluidos los procesos de combustión, las actividades industriales, las emisiones vehiculares y fuentes naturales como el polvo y el polen. Luego a continuación tenemos NC0.5, NC1.0 y NC2.5 que miden el número de concentración actual de partículas materiales con diámetros iguales o inferiores a 0,5, 1,0 y 2,5 micrómetros respectivamente.

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

https://www.kaggle.com/datasets/deepcontractor/smoke-detection-dataset

Kaggle.com es un site de uso público que reúne cientos de datasets de diversas temáticas. En nuestro caso elegimos una temática relacionada con el IoT, porque este estudio reiteramos una vez más, constituye el paso preliminar para la puesta práctica del modelo obtenido en el dispositivo de borde. Observar la Figura 3 que resume de qué manera se ha llevado a cabo esta importación del dataset.

```
[ ]  from google.colab import drive
     #drive.mount('/content/drive')
     drive.mount("/content/drive", force_remount=True)
     path = '/content/drive/MyDrive/IA IOT/'

     Mounted at /content/drive

[ ]  ruta_archivo = path + 'smoke_detection_iot.csv'
     print(ruta_archivo)

     /content/drive/MyDrive/IA IOT/smoke_detection_iot.csv
```

Figura 3. Importación del dataset de Kaggle.com a nuestro Google drive

2. Explicación teórica y descripción de los modelos a emplear en nuestro caso de estudio

2.1 Descripción teórica del problema

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

Así, la f(x) queda planteada de la siguiente forma:

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

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

El objetivo de los siguientes modelos es buscar la mejor forma de estimar dichas probabilidades condicionales.

2.2 Regresión logística simple.

En este modelo se plantea:

$$P(Y=1|X=x) = sigmoide(\theta^T * x + \beta)$$

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

$$L(x) = \frac{1}{1+e^{-x}}$$

En este modelo los parámetros entrenables, es decir que se pueden modificar para adaptar el modelo a los datos de estudio, son θ y β.

Para implementar este modelo se utilizará una red neuronal [5] de una única capa. Esta capa poseerá tantas neuronas como variables de entrada, es decir en nuestro caso serán doce neuronas. También se incorporará una capa de normalización de los datos de entrada a fin de mejorar la performance (convergencia del modelo). A la hora de implementar el modelo la capa de normalización ocupa el rol de la capa de entrada.

2.3 Redes neuronales como familia de modelos

Las redes neuronales están compuestas por neuronas organizadas en capas. Cada neurona *s* se comporta individualmente de forma similar a la regresión logística:

- Tiene un conjunto de datos de entrada, que pueden ser los datos de entrada del modelo o la salida de otras neuronas, llamémosle X_s.
- Tiene parámetros entrenables θ_s y β_s.
- Tiene una función de activación, que suele ser una función sigmoide Fs . En los modelos que estudiaremos todas las neuronas utilizan la misma función de activación, la función logística comentada en el apartado anterior.
- Cada neurona toma el valor de evaluar $F_s(\theta_s{}^T * x_s + \beta_s)$

Finalmente, hay una última neurona de salida, cuya salida se interpreta como la predicción de $P(Y=1|X=x)$

El objetivo de tener redes neuronales con más de una capa, frente a la regresión logística simple, es poder detectar patrones no lineales entre los parámetros, de una forma flexible que no demanda suponer cuáles serán esos patrones no lineales de antemano. Por eso las funciones de activación necesitan ser no lineales.

2.3 Red neuronal de dos capas. Capa de entrada más una capa oculta

En este modelo se agrega una segunda capa densa a la de entrada. Acá ya hay un hiperparámetro a considerar, que es el número de neuronas a incluir en esta segunda capa. Luego, para comparar, se hará variar en una grilla el posible tamaño de esta capa intermedia (capa oculta) para ver cómo cambia el desempeño de este modelo. A estas capas intermedias que tienen por entrada a todas las neuronas de la capa anterior se las llama capas densas.

Los hiperparámetros, como se menciona en este caso al número de neuronas, son aquellos que en vez de modificarse durante el proceso de entrenamiento del modelo, forman parte de la definición del mismo. Éstos no se entrenan, sino que se comparan los distintos modelos entre sí durante el proceso de validación cruzada.

2.4 Red neuronal de n capas

En esta parte lo que haremos es ir variando el número de capas intermedias del modelo, cuyo tamaño será el que obtuvo el mejor desempeño en validación cruzada en el modelo de una única capa oculta.

2.5 Ajuste del hiperparámetro batch_size

Un hiperparámetro que no se ha considerado es el batch_size (que constituye el tamaño del lote de datos empleados para el entrenamiento del modelo). El uso del entrenamiento por batches mejora la eficiencia del modelo en etapas tempranas pero también se corre el riesgo de evitar que el modelo alcance la máxima precisión posible. Por lo tanto se agregará a los modelos ya entrenados nuevas épocas de entrenamiento. Se tomarán 50 épocas pero ahora considerando un batch_size = 1024 y posteriormente se tomaron 500 épocas con un batch_size igual al tamaño de los datos de entrenamiento. Habrá que tener en cuenta que estas 500 épocas adicionales serán individualmente mucho más cortas que cada una de las 50 épocas originales por no haber una división en batches. Lo que se busca comparar aquí es el impacto en las métricas de este entrenamiento adicional.

2.6 Árboles de decisión y Bosque Aleatorio

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

- Partiendo del conjunto de datos completo, se busca la variable predictora que de forma aislada mejor sirva para dividir los datos entre una y otra

categoría. Es decir, la variable Xi que permita establecer un valor de corte Vi tal que usando como criterio para clasificación de xi<Vi o xi>Vi, ocurran la menor cantidad de malas asignaciones posibles.

- Posteriormente, se divide el conjunto de datos entre los clasificados por este criterio como 0 y los clasificados como 1.
- Luego, con cada uno de estos subconjuntos se construyen sub-árboles de decisión que serán los hijos del nodo actual. Conectados cada uno por una arista que representa el resultado de la clasificación en el nodo raíz.
- Si el conjunto de datos considerado está compuesto por nodos de una única categoría, el proceso termina y ése es un nodo hoja del árbol.

El método de Bosque Aleatorio consiste en construir un gran número de árboles de decisión utilizando para cada uno subconjuntos aleatorios de las observaciones y de las variables predictoras. Para evaluarse en un caso concreto, el Bosque Aleatorio evalúa todos los árboles de decisión que lo componen y se queda con la categoría que obtiene la mayoría de votos entre los árboles de decisión.

El crear muchos árboles independientes tiene por objetivo permitir que cada uno detecte aspectos distintos del problema, y a su vez evitar el sobreajuste.

Este modelo se diferencia de los otros porque llega directamente al paso predictivo sin pasar por una estimación de las funciones de probabilidad condicional.

El módulo TensorFlow Random Forest [6] implementa este modelo de una forma sencilla y fácil de utilizar.

3. Etapas a desarrollar en la primera fase de nuestro caso de estudio

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

3.1 Obtención del mejor modelo. Utilización de los datos de entrenamiento

Se aplica el mismo dataset de entrenamiento a los diferentes modelos ya enunciados anteriormente (60% del dataset original, es decir se computarán 37578 registros sobre un total de 62630). En el entrenamiento se buscará ajustar los parámetros del modelo para minimizar el valor de una función de perdida (loss), que mide el desempeño del modelo en los datos de entrenamiento. En este caso, utilizamos la entropía cruzada como función de costo [7].

3.2 Validación del mejor modelo. Utilización de datos para validación cruzada

Con un nuevo subconjunto de datos diferentes a los usados en el entrenamiento, en este caso se tomará un 20% del dataset original (12526 registros), y se los aplicará a los distintos modelos ya entrenados a fin de obtener el mejor de los modelos, y comprobar de esta manera la solvencia de cada modelo.

Para comparar a los modelos entre sí y obtener el mejor, se utilizará como criterio la precisión, qué en las gráficas aparecerá con la leyenda *accuracy* (para los datos de entrenamiento) y *val_accuracy* (cuando se consideren los datos para validación cruzada). Es decir, la cantidad de observaciones correctamente clasificadas sobre la cantidad de observaciones totales.

Sobre esto, hay que tener en cuenta que al modificar el valor de un hiperparámetro se genera un modelo diferente aunque sea de la misma familia, por lo tanto la validación cruzada también elige el valor óptimo (entre los utilizados) de los hiperparámetros.

3.3 Evaluación del mejor modelo. Utilización de datos para la prueba del modelo

Ahora con el subconjunto restante de datos (que es el otro 20% del dataset) se procede a evaluar el comportamiento del modelo. Esto permite dar una estimación confiable de la efectividad del modelo seleccionado utilizando los datos de validación cruzada.

3.4 Repetición de los puntos anteriores incorporando ruido (error por dispersión del sensor)

Dado que los sensores que podrían ser utilizados pueden introducir cierto ruido en las mediciones (error del sensor por dispersión), se torna interesante observar la robustez del modelo frente a datos que tengan ese ligero ruido incorporado. Para incorporar el ruido, se hará lo siguiente: en el dataset completo, se tomará la desviación estándar de cada columna, es decir de cada sensor. A cada columna se le sumará una variable normal de esperanza 0 y desviación estándar igual al 10 % de la desviación estándar de la variable. Posteriormente se dividirá el dataset y se realizará el mismo procedimiento visto para el caso sin ruido.

4. Desarrollo de la experiencia

4.1 Descripción de la plataforma de trabajo a emplear

Se empleará para este caso de estudio la plataforma Google Colab [8], la cual forma parte de la suite de aplicaciones de Google. Esta plataforma permite obtener recursos tales como GPU, disco y memoria RAM y por lo tanto evita la utilización de un hardware local de grandes prestaciones.

4.2 Importación de módulos y datos

En primer lugar para la realización del estudio es necesario importar las siguientes librerías de código abierto: numpy (manipulación numérica en Python) [9], pandas (manejo de datasets) [10], tensorflow (librería para aprendizaje automático) [11] , seaborn (manipulación de gráficos) [12], matplotlib (Idem seaborn) [13], prettytable (para imprimir datos en formato de tabla dibujada en consola) [14] y tensorflow_decision_forests [15] (modelos de árboles de decisión). En segundo lugar hay que crear un diccionario para mapear los nombres de las columnas originales del dataset a los nuevos nombres, quedando ahora el dataset como se ilustra en la Figura 4.

Variable	Covarianza	R2
Humidity[%]	0.3998	0.1599
Raw Ethanol	-0.3407	0.1160
Pressure[hPa]	0.2498	0.0624
TVOC[ppb]	-0.2147	0.0461
Temperature[C]	-0.1639	0.0269
NC0.5	-0.1281	0.0164
PM1.0	-0.1106	0.0122
Raw H2	0.1070	0.0115
eCO2[ppm]	-0.0970	0.0094
PM2.5	-0.0849	0.0072
NC1.0	-0.0828	0.0069
NC2.5	-0.0577	0.0033

Figura 7. Covarianzas y coeficiente de correlación R2 de cada una de las variables predictoras

4.3 Análisis exploratorio de los datos

La primera etapa, antes de definir un modelo, es realizar un análisis exploratorio de los datos. En los siguientes gráficos no se observa que individualmente ninguna variable sea buena predictora de Fire Alarm (ver Figuras 5, variables X1 a X6 y Figura 6, variables X7 a X12 respectivamente). A continuación, el siguiente paso es observar la covarianza y los coeficientes de correlación R2 entre cada una de las variables predictoras y la variable a predecir, lo cual arroja el resultado que puede observarse en la Figura 7. De dicha tabla se desprende que las variables capturan individualmente muy poco de la varianza de Fire Alarm. Esto pone en evidencia la necesidad de utilizar un modelo compuesto por varias de las variables predictoras.

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	Y
0	20.000	57.36	0	400	12306	18520	939.735	0.00	0.00	0.00	0.000	0.000	0
1	20.015	56.67	0	400	12345	18651	939.744	0.00	0.00	0.00	0.000	0.000	0
2	20.029	55.96	0	400	12374	18764	939.738	0.00	0.00	0.00	0.000	0.000	0
3	20.044	55.28	0	400	12390	18849	939.736	0.00	0.00	0.00	0.000	0.000	0
4	20.059	54.69	0	400	12403	18921	939.744	0.00	0.00	0.00	0.000	0.000	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
62625	18.438	15.79	625	400	13723	20569	936.670	0.63	0.65	4.32	0.673	0.015	0
62626	18.653	15.87	612	400	13731	20588	936.678	0.61	0.63	4.18	0.652	0.015	0
62627	18.867	15.84	627	400	13725	20582	936.687	0.57	0.60	3.95	0.617	0.014	0
62628	19.083	16.04	638	400	13712	20566	936.680	0.57	0.59	3.92	0.611	0.014	0
62629	19.299	16.52	643	400	13696	20543	936.676	0.57	0.59	3.90	0.607	0.014	0

62630 rows × 13 columns

Figura 4. Mapeo del dataset con nuevos nombres en columnas de las variables independientes

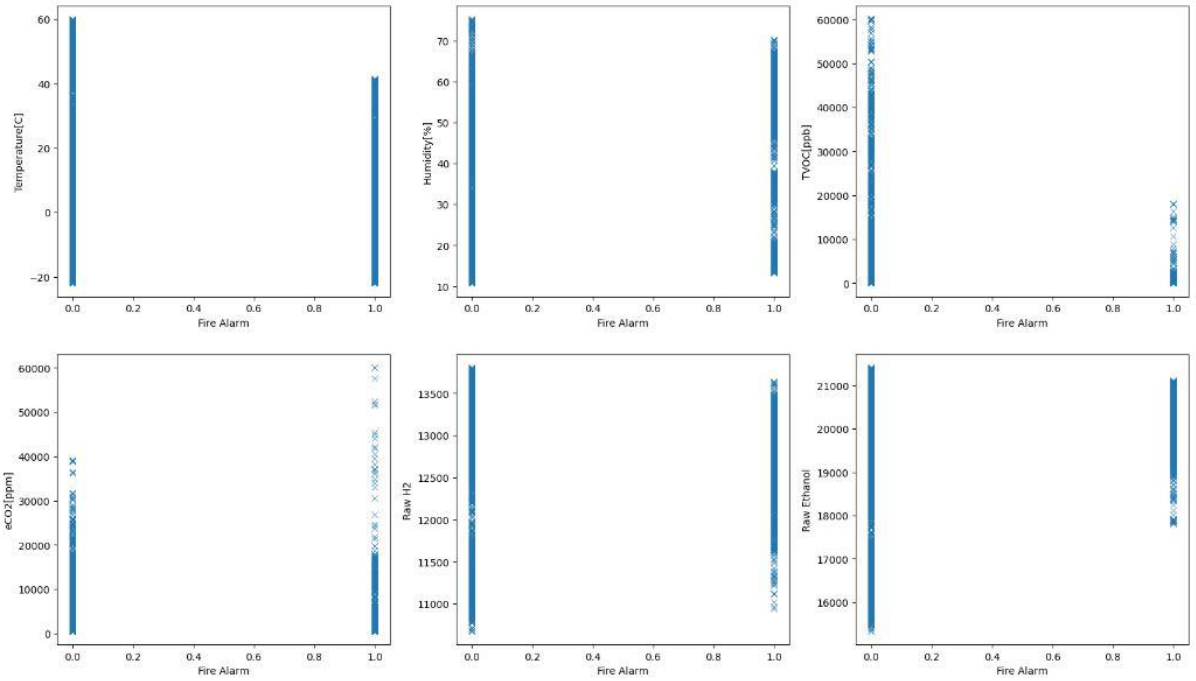

Figura 5. Variables predictoras (Temperature a Raw Ethanol – x1 a x6)

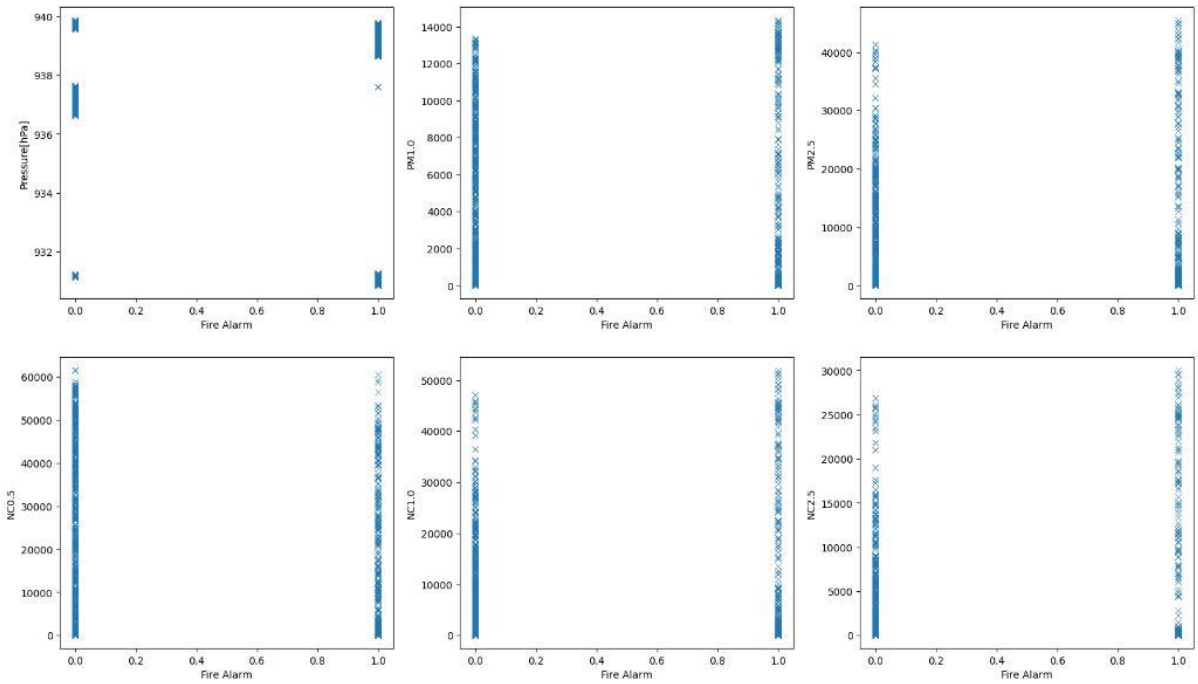

Figura 6. Variables predictoras (Presión a NC2.5 – x7 a x12)

5. Resultados obtenidos para la obtención del mejor modelo predictivo (sensores con comportamiento normal, sin fallas)

5.1 Regresión logística simple

Como se ve en la Figura 8 (en el gráfico de la izquierda), hay un punto donde es difícil distinguir las líneas debido a que éstas convergen a valores muy cercanos a 0.90. Debido a esto, es oportuno hacer también un gráfico de log (1-precisión), es decir el logaritmo (natural en este caso) de la tasa de errores. En el gráfico de la derecha se puede apreciar mejor como el modelo mejora en la fase madura del entrenamiento. Como puede apreciarse en este caso, la precisión alcanza un nivel sustancialmente alto tanto en los datos de entrenamiento como en los datos de validación cruzada. Por lo tanto esto nos permite inferir que no hay un caso de sobreajuste (overfitting).

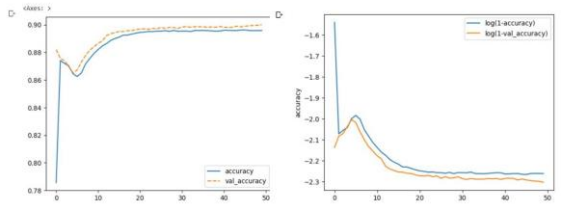

Figura 8. Predicción de Fire Alarm para los datos de entrenamiento y de validación cruzada - 90%

5.2 Regresión logística de dos capas

De los resultados obtenidos se observa que la precisión máxima obtenida es 99,1538 % y se da para 40 (cuarenta) neuronas, por lo tanto éste es el número de neuronas a considerar en la segunda capa, ver Figura 9. Como se puede apreciar en dicha figura, el aumentar considerablemente el tamaño de esta capa intermedia no lleva a una mejora en el desempeño en los datos de Validación Cruzada.

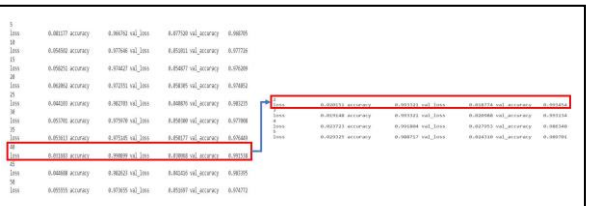

Figura 9. Predicción de Fire Alarm para los datos de entrenamiento y de validación cruzada - 90%

5.3 Regresión logística de n capas

De acuerdo a la salida arrojada se verifica que con 2 capas se obtiene la máxima precisión y ésta asciende al 99,3454 %, ver Figura 9.

5.4 Consideración sobre el hiperparámetro Batch_size

Comparando las métricas entre el modelo lineal y el modelo de 2 capas de 40 neuronas para 50 épocas con batch_size igual a 32, 50 épocas con batch_size igual a 1024 y para 500 épocas con batch_size igual a 37578, se observa que ambas precisiones (entrenamiento y validación cruzada) no sufren prácticamente variaciones. Ver Figura número 10.

5.5 Árboles de decisión (Ramdom Forest)

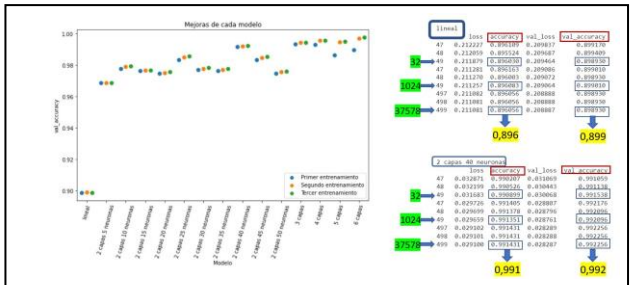

Figura 10. Mejoras de cada modelo con 3 tipos de entrenamiento variando batch_size y épocas

A continuación, en la Figura 11 se observa el árbol de decisión. Éste método resultó ser el mejor de todos los modelos con una precisión en torno al 99,992 %.

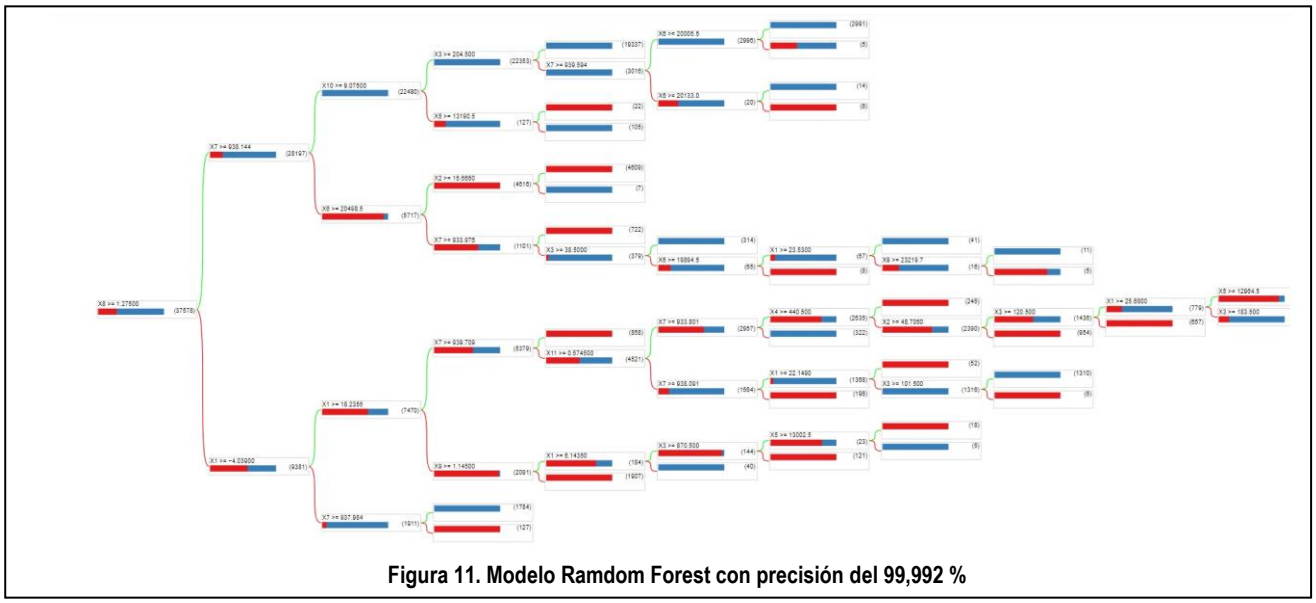

Figura 11. Modelo Ramdom Forest con precisión del 99,992 %

5.6 Evaluación del mejor modelo obtenido

A continuación tomamos el modelo obtenido por la técnica de Ramdom Forest (que fue el mejor de todos en cuanto a su precisión) y lo evaluamos aplicándole ahora el set de datos de prueba, que constituye el 20% restante del dataset. De esta manera le exponemos datos que el modelo nunca observó. Los resultados se resumen en la Figura 12, la cual se indica a continuación:

```
El mejor modelo es : Random Forest
38/38 [==============================] - 1s 14ms/step - loss: 0.0000e+00 - accuracy: 0.9999
13/13 [==============================] - 0s 14ms/step - loss: 0.0000e+00 - accuracy: 0.9999
13/13 [==============================] - 0s 13ms/step - loss: 0.0000e+00 - accuracy: 0.9998
Tiene:
Train Accuracy:                 [0.0, 0.9999201893806458]
Cross Validation Accuracy:      [0.0, 0.9999201893806458]
Test Accuracy:                  [0.0, 0.9997605085372925]
```

Figura 12. Obtención de la precisión para los datos de entrenamiento, validación y prueba para Random Forest

5.7 Incorporación de ruido a los modelos obtenidos en el estudio

Observar ahora la Figura 13 en donde se sintetiza cómo afecta la precisión producto del ruido (por la dispersión del sensor) en todos los modelos estudiados:

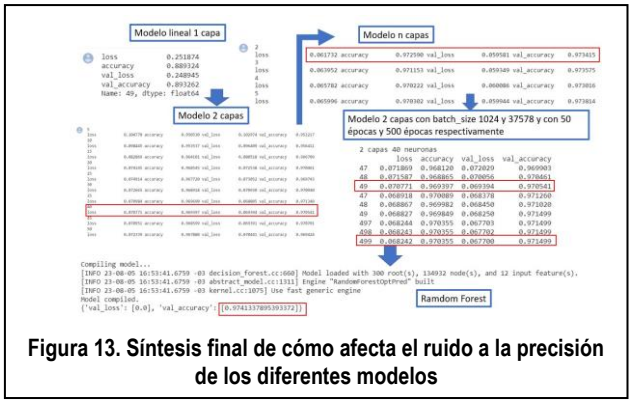

Figura 13. Síntesis final de cómo afecta el ruido a la precisión de los diferentes modelos

Como puede apreciarse en la figura anterior, la precisión de los modelos (a partir del de dos capas) baja en torno al 97%, lo cual no constituye una pérdida significativa de la precisión de los modelos evaluados, teniendo en cuenta la dispersión introducida por los sensores.

6. Resultados obtenidos considerando sensores defectuosos. Segunda fase del caso de estudio

6.1 Primer escenario en donde el sensor falla de manera abrupta

En este caso, como sólo se busca comparar entre modelos y no tener un proceso de selección de modelos como los que ya realizamos anteriormente, es que únicamente se dividirá el dataset en dos. Un 70% para datos de entrenamiento y un 30% para datos de prueba o testing. Estos datos empleados para la prueba del modelo cumplen la función que usualmente hacían los datos de validación cruzada ya descriptos, por eso resultará importante que los datos de prueba no se usen para entrenar a los modelos, que sean estos diferentes, para evitar de esta manera premiar el sobreajuste. Por lo tanto, en este escenario se evaluarán doce posibilidades. En cada una se considerará el dataset con una columna faltante y se expondrá la precisión del modelo en los datos de prueba. Recordemos que la evaluación sólo se realiza en este escenario para el modelo de Ramdom Forest, por ser éste el mejor de los modelos evaluados obtenidos anteriormente en la primera fase de nuestro caso de estudio. Observar la figura 14 para mayor detalle de lo narrado. Como se ve en dicha figura, el modelo de Random Forest en este ejemplo es robusto y la anulación de un único sensor no reduce significativamente la precisión del modelo para la detección de fuego.

Sensor anulado	Precisión Comp.
-	0.999787
Temperature[C]	0.999787
Humidity[%]	0.999734
TVOC[ppb]	0.999042
eCO2[ppm]	0.999734
Raw H2	0.999734
Raw Ethanol	0.999734
Pressure[hPa]	0.999681
PM1.0	0.999734
PM2.5	0.999734
NC0.5	0.999734
NC1.0	0.999734
NC2.5	0.999734

Figura 14. Precisión obtenida para los datos de prueba considerando la anulación abrupta de cada sensor

6.2 Segundo escenario donde el sensor falla dando valores aleatorios

En este segundo escenario, primeramente se van a entrenar cuatro modelos utilizando el set de entrenamiento completo, y luego se probará cómo se comportan cuando se agrega el ruido debido a la falla en cada uno de los sensores. Pero este ruido ahora pretenderá simular un sensor defectuoso. Se detallará más adelante en este escenario a continuación. El primer modelo a entrenar en este nuevo contexto será una *regresión logística simple*, que en TensorFlow se implementa como una red neuronal con una única capa. Se evaluará para 100 épocas y para 250 épocas con valores de batch_size de 32 y 1024 respectivamente. Recordemos que el batch_size es el tamaño del lote de datos para entrenar el modelo. Los valores de 1371 y 43 respectivamente surgen de tomar el 70% de 62630 que es el número de filas de nuestro dataset, valor que da 43841, y luego de dividir a este valor por 32 y por 1024 se obtienen los valores respectivos de 1371 y 43, aproximándose al valor entero más cercano.

Para este modelo y las redes neuronales se utilizó una capa de normalización [16], cuya función es modificar los datos de entrada con la siguiente fórmula:

$$Entrada\_Normalizada = \frac{Entrada - Media_{Entrada}}{Desvio\_Estandar_{Entrada}}$$

Esta fórmula se aplicó a cada columna de forma independiente. Esto tiene el propósito de hacer más eficiente el proceso de entrenamiento.

6.2.1 Primer modelo. Resultados obtenidos para regresión logística simple

Obsérvese en la figura 15 que la precisión para este modelo de red neuronal de una capa no mejora sustancialmente al pasar de 100 épocas a 250 épocas y su valor ronda aproximadamente en un 90% .

```
Epoch 96/100
1371/1371 [==============================] - 2s 2ms/step - loss: 0.2061 - accuracy: 0.8951
Epoch 97/100
1371/1371 [==============================] - 2s 2ms/step - loss: 0.2061 - accuracy: 0.8948
Epoch 98/100
1371/1371 [==============================] - 3s 2ms/step - loss: 0.2061 - accuracy: 0.8955
Epoch 99/100
1371/1371 [==============================] - 3s 2ms/step - loss: 0.2060 - accuracy: 0.8956
Epoch 100/100
1371/1371 [==============================] - 3s 2ms/step - loss: 0.2059 - accuracy: 0.8949
43/43 [==============================] - 0s 4ms/step - loss: 0.2047 - accuracy: 0.8952
Epoch 246/250
43/43 [==============================] - 0s 3ms/step - loss: 0.2047 - accuracy: 0.8953
Epoch 247/250
43/43 [==============================] - 0s 3ms/step - loss: 0.2047 - accuracy: 0.8954
Epoch 248/250
43/43 [==============================] - 0s 3ms/step - loss: 0.2047 - accuracy: 0.8953
Epoch 249/250
43/43 [==============================] - 0s 4ms/step - loss: 0.2047 - accuracy: 0.8954
Epoch 250/250
43/43 [==============================] - 0s 3ms/step - loss: 0.2047 - accuracy: 0.8953
```

Figura 15. Precisión obtenida para los datos de prueba para la red neuronal de una capa

6.2.2 Segundo modelo. Resultados obtenidos para la red neuronal multicapa sin dropout

Para este entrenamiento se usaron 6 capas ocultas con 25 neuronas por capa. Se obtuvieron los siguientes resultados, ver Figura 16.

```
Epoch 96/100
1371/1371 [==============================] - 3s 2ms/step - loss: 0.0072 - accuracy: 0.9982
Epoch 97/100
1371/1371 [==============================] - 3s 2ms/step - loss: 0.0080 - accuracy: 0.9977
Epoch 98/100
1371/1371 [==============================] - 4s 3ms/step - loss: 0.0101 - accuracy: 0.9971
Epoch 99/100
1371/1371 [==============================] - 4s 3ms/step - loss: 0.0084 - accuracy: 0.9975
Epoch 100/100
1371/1371 [==============================] - 4s 3ms/step - loss: 0.0107 - accuracy: 0.9965
Epoch 115/250
43/43 [==============================] - 0s 4ms/step - loss: 0.0028 - accuracy: 0.9997
Epoch 116/250
43/43 [==============================] - 0s 4ms/step - loss: 0.0028 - accuracy: 0.9997
Epoch 117/250
43/43 [==============================] - 0s 6ms/step - loss: 0.0028 - accuracy: 0.9997
Epoch 118/250
43/43 [==============================] - 0s 6ms/step - loss: 0.0033 - accuracy: 0.9995
Epoch 119/250
43/43 [==============================] - 0s 6ms/step - loss: 0.0029 - accuracy: 0.9996
```

Figura 16. Precisión obtenida para los datos de prueba para la red neuronal multicapa sin dropout

Observar que en este caso (para el caso de 250 épocas), la ejecución se cortó antes, exactamente a las 119 épocas. Esto es así porque el entrenamiento usa una herramienta automática llamada *callback* [17] que cuando lleva varias épocas sin mejorar su función de pérdida, corta de forma automática la ejecución. Esto evita el entrenamiento inútil. En este caso la precisión es superior al 99,9 %.

6.2.3 Tercer modelo. Resultados obtenidos para la red neuronal multicapa con dropout

El D*ropout* [18] es una técnica utilizada en el aprendizaje profundo (deep learning) y las redes neuronales para regularizar el modelo y prevenir el sobreajuste (overfitting). El sobreajuste ocurre cuando una red neuronal se adapta demasiado a los datos de entrenamiento y se tiene dificultades para generalizar correctamente en datos nuevos y no vistos. El dropout es una estrategia efectiva para abordar este problema. Por lo tanto, el concepto de dropout implica apagar aleatoriamente (es decir, eliminar temporalmente) un conjunto de unidades o neuronas en una capa durante cada iteración del entrenamiento. Esto significa que, en cada paso hacia adelante y hacia atrás del entrenamiento, algunas de las neuronas no contribuirán a la propagación hacia adelante ni a la retropropagación de

gradientes. Esto introduce un grado de incertidumbre en la red neuronal y evita que las neuronas se vuelvan demasiado especializadas en la tarea de entrenamiento. El dropout se aplica típicamente en las capas ocultas de una red neuronal y se controla mediante una *tasa de dropout*, que es un valor que determina la probabilidad de que una neurona se apague durante una iteración de entrenamiento. Por ejemplo, si se establece una tasa de dropout de 0.50, cada neurona tiene un 50% de probabilidad de ser apagada en cada iteración. La principal ventaja del dropout es que ayuda a prevenir el sobreajuste al forzar a la red neuronal a aprender representaciones más robustas y generalizables de los datos. Además, el dropout a menudo conduce a modelos más robustos que son mejores en la generalización a nuevos datos de prueba. En resumen, el dropout es una técnica de regularización en redes neuronales que implica la aleatorización temporal de neuronas durante el entrenamiento para mejorar la capacidad de generalización del modelo. Cabe destacar también que esta técnica de dropout se aplica en esta situación puntual para ver si se logra una mayor robustez del modelo frente a los errores de ruido producidos en cada sensor debido a un mal funcionamiento del mismo. Por el contrario no ha sido necesario aplicar esta técnica en la primera fase de nuestro caso de estudio, en donde el objetivo era encontrar el mejor modelo, porque ya los datos de validación cruzada tenían un comportamiento similar a los datos empleados para el entrenamiento de dichos modelos. Y eso daba ya la pauta que en esa situación no existía por lo tanto sobreajuste.

En este caso concreto, se utilizaron 3 capas de dropout y la distribución de las 11 capas totales fue la siguiente:

1. Capa de normalización
2. Dropout(0.5)
3. Capa densa de 25 neuronas
4. Capa densa de 25 neuronas
5. Capa densa de 25 neuronas
6. Dropout(0.5)
7. Capa densa de 25 neuronas
8. Capa densa de 25 neuronas
9. Capa densa de 25 neuronas
10. Dropout(0.5)
11. Capa de salida.

Los resultados obtenidos para esta red multicapa considerando el dropout pueden observarse a continuación en la Figura 17.

6.2.4 Cuarto modelo. Resultados obtenidos para Ramdom Forest

Los resultados en este caso son los mismos que se han obtenido para la primera fase de nuestro caso de estudio. Ver punto 5.6.

6.2.5 Comparativa de los cuatro modelos enunciados

Para la comparación se busca utilizar un criterio que sea aplicable a todos los sensores sin que su lógica necesite ser adaptada para cada caso particular.

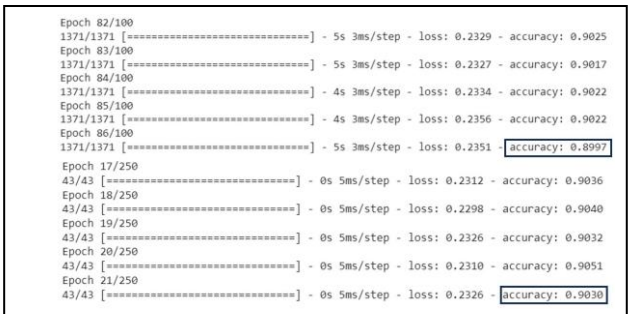

Figura 17. Precisión obtenida para los datos de prueba para la red neuronal multicapa con dropout

Concretamente, para simular la falla en el vector Xi se creará un vector Ui, donde cada entrada que representa cada sensor sigue una distribución uniforme [19] que estará comprendida entre los valores mínimo y máximo de Xi dentro de todo el dataset. Esto para que, vista de forma aislada, esta nueva observación siempre tenga sentido.

Luego, se elegirá un parámetro α y se remplazarán los valores de Xi por:

$$O_{i,\alpha} = (1-\alpha) * X_i + \alpha * U_i$$

Analizando la expresión $O_{i,\alpha}$, puede inferirse que cuando $\alpha = 0$ entonces $O_{i,\alpha} = X_i$, en donde esto representa al sensor sin fallar, es decir que los datos observados guardan relación con los datos medidos dados por el dataset original. Por el contrario cuando $\alpha = 1$ entonces $O_{i,\alpha} = U_i$, es decir la variación aleatoria del sensor sigue la distribución uniforme y nos estaría representando al sensor fallando en su totalidad. En este contexto, α representa qué tan comprometido está el sensor en cuanto a la falla aleatoria se refiere. Se verá para cada sensor como cada modelo se comporta para distintos valores de α. Cabe destacar que estas pruebas se realizaron con los datos de testing.

A continuación, en la figura 18 puede observarse la comparativa mencionada de los diferentes modelos y como la falla de cada sensor por separado contribuye con la precisión de cada modelo, determinando así la robustez de cada modelo frente a la falla de cada sensor.

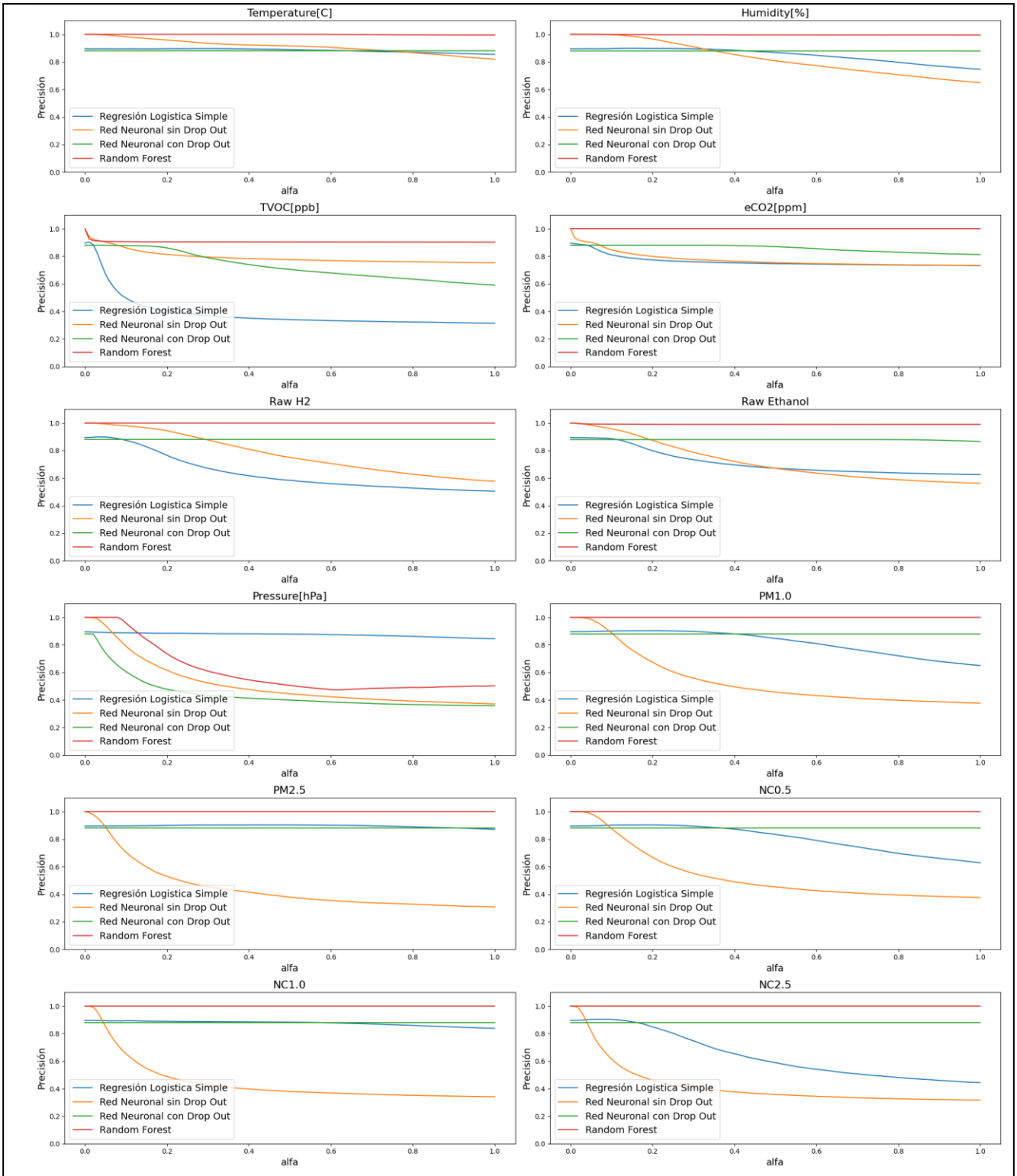

Figura 18. Comparativa de la precisión de cada modelo para cada uno de los sensores fallando mediante una distribución uniforme

Observando los gráficos de la figura 18 podemos decir lo siguiente:

1. Se observa que para todos los sensores excepto el de presión, el modelo Ramdom Forest se comporta mejor que todos los demás modelos al incrementarse el error del sensor dado por el coeficiente α.
2. Sólo los sensores de presión y TVOC muestran un impacto apreciable en la performance del modelo de Random Forest.
3. Se observa también que la red neuronal sin dropout es la más sensible a la introducción de errores en los sensores.

7 Conclusiones y trabajo futuro

Los hiperparámetros juegan un papel fundamental a la hora de analizar el comportamiento de un modelo de una red neuronal en cuanto a su precisión. Concretamente se observa que cuanto mayor es el número de neuronas de una capa intermedia, la precisión del modelo se incrementa, pero esto por supuesto tiene un límite. Por otro lado, también se verificó que el aumentar el número de épocas y el tamaño del lote de datos utilizados para el training del modelo, esta situación no hace que se modifique la precisión de manera sustancial.

En cuanto al modelo Random Forest podemos enunciar que se trata de un modelo robusto tanto frente a la falla detectada de los sensores como a la falla no detectada de los mismos.

El modelo de red neuronal sin dropout es el más sensible frente a la introducción de errores en los sensores, perdiendo en varios casos más del 50% de la precisión.

La regresión logística simple y la red neuronal con dropout son menos sensibles a los errores que la red neuronal sin dropout, pero no menos sensibles que el Random Forest y tienen una performance en el caso base y para errores pequeños sensiblemente menor.

Trabajo futuro:

El trabajo futuro consistirá en poder trasladar el modelo de red neuronal o de Ramdom Forest obtenido en una plataforma de recursos compartidos como lo es Google Colab o Jupyter Notebooks a un dispositivo de borde de prestaciones limitadas en cuanto a sus recursos de hardware. Pero aclaremos que todo este trabajo previo aquí realizado es primordial para pasar a esta etapa posterior.

Para ello será necesario el empleo de una librería especial derivada de Tensorflow denominada Tensorflow Lite [20]. También será necesario utilizar MicroPython [21] el cual deberá ser instalado en la memoria flash del dispositivo de borde. Además se necesitará contar con un IDE que permita transferir el código Python al dispositivo como así instalar en la memoria flash del dispositivo el intérprete MicroPython . Entre los IDE más populares se encuentra Thonny [22].

8 Referencias

[1] Warden, P., & Situnayake, D. (2019). TinyML: Machine Learning with Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers. O'Reilly Media. D.

[2] Dataset.

https://www.kaggle.com/datasets

[3] Tree-based Machine Learning Algorithms: Decision Trees, Random Forests, and Boosting Edición Kindle D.

[4] Mark Lutz. O' REILLY. Programming Python: Powerful Object-Oriented Programming 4th Edición D.

[5] Neural Networks and Deep Learning: A Textbook 1st ed. 2018 Edición de Charu C. Aggarwal (Author) D.

[6] TensorFlow Random Forest:

https://www.tensorflow.org/decision_forests/api_docs/python/tfdf/keras/RandomForestModel

[7] The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning (Information Science and Statistics) 2004th Edición. Reuven Y. Rubinstein (Author), Dirk P. Kroese (Author)

[8] Google Colab. https://colab.research.google.com

[9] NumPy.

https://numpy.org

[10] Pandas.

https://pandas.pydata.org/docs

[11]

https://www.tensorflow.org/resources/libraries-extensions

[12] Seaborn. statistical data visualization.

https://seaborn.pydata.org/

[13] Matplotlib.

https://matplotlib.org/

[14] Prettytable.

https://pypi.org/project/prettytable/

[15] tensorflow_decision_forests

https://www.tensorflow.org/decision_forests

[16] Capa de normalización:

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Normalization

[17] Callback:

https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback

[18] Dropout:

https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9

[19] Probabilidad y Estadística. Aplicaciones y métodos. Georges C. Canavos. Virginia Commonwealth University. Editorial Mc Graw Hill.

[20] Tensorflow Lite.

https://www.tensorflow.org/lite/guide

[21] MicroPython.

https://micropython.org/

[22] Thonny.

https://thonny.org/