



Práctica Final de APA

PREDICCIÓN DE FALLO EN ROBOT

Por: Víctor Ramírez Arimaha y Adrià Cebrián Ruiz

Facultat Informàtica de Barcelona (FIB)
3 de enero de 2026

Contents

1	Introducción y objetivos	2
2	Preparación de los datos para entrenamiento	2
2.1	Ingeniería de Características: Lags Temporales	2
2.2	Preprocesamiento y Limpieza	2
2.3	Desbalanceo de datos	2
3	Estudio de los datos	3
3.1	Correlación Espacial y Redundancia	3
3.2	Análisis de Autocorrelación (Justificación de Lags)	4
3.3	Relevancia de Características con el Target	4
3.4	Reducción de Dimensionalidad y Estructura Latente	4
4	Metodología de selección de modelos	6
5	Modelos Lineales	6
5.1	Regresión Logística	6
5.2	QDA (Análisis Discriminante Cuadrático)	7
5.3	SVM Kernel Lineal	8
5.4	Conclusión modelos lineales	8
6	Modelos No Lineales	9
6.1	SVM con Kernel RBF	9
6.2	Random Forest	10
6.3	Red Neuronal (Multilayer Perceptron)	10
7	Comparativa de modelos	11
7.1	Resumen Cuantitativo	11
7.2	Precisión vs. Sensibilidad	12
7.3	El problema de la Clase 2 (Protective Stop)	12
8	Combinación de modelos (Ensembles)	12
8.1	Stacking Classifier	12
8.2	Voting Classifier	13
8.3	Conclusión	14
9	Conclusiones y dificultades encontradas	14
	Referencias	15

1 Introducción y objetivos

En este trabajo hemos desarrollado y evaluado diferentes modelos de aprendizaje automático para detectar y clasificar fallos en un brazo robótico UR3. Específicamente, hemos implementado y analizado 6 modelos distintos, 3 lineales y 3 no lineales.

El problema planteado consiste en una tarea de aprendizaje supervisado de clasificación multiclas. El objetivo es predecir el estado de un robot (normalidad frente a diversos tipos de fallos) basándose en las lecturas temporales de sensores situados en sus 5 articulaciones.

Debido a la naturaleza del problema, la preparación de los datos tiene un papel muy importante.

2 Preparación de los datos para entrenamiento

Un aspecto crítico de este problema es la naturaleza secuencial y agrupada de los datos. Las mediciones se organizan en *ciclos de operación*, lo que implica que las muestras dentro de un mismo ciclo están correlacionadas temporalmente. Por tanto, el reto no solo reside en alcanzar una alta precisión, sino en garantizar una metodología de validación robusta que evite el *data leakage* (fuga de datos) entre ciclos. Para ello, hemos optado por una estrategia de validación basada en `GroupShuffleSplit`, asegurando que si un ciclo pertenece al conjunto de entrenamiento, ninguna de sus muestras futuras aparezca en el conjunto de testing.

2.1 Ingeniería de Características: Lags Temporales

La decisión más relevante en el preprocesamiento ha sido la introducción de *lag features* (retardos). Dado que cada fila del dataset original representa una "foto" estática del estado de los sensores, se pierde la información sobre la dinámica del movimiento (velocidad, aceleración, cambios bruscos de temperatura).

Para capturar esta inercia física, hemos transformado el dataset para que cada instancia incluya no solo el estado actual t , sino también los estados $t - 1, \dots, t - 5$. Esta ventana temporal de 5 segundos se justifica en la sección de Análisis de Autocorrelación, basada en el análisis de autocorrelación parcial (PACF).

2.2 Preprocesamiento y Limpieza

Antes de la generación de modelos, se han aplicado los siguientes pasos:

- **Limpieza:** Eliminación de columnas no informativas (Timestamp, índices) y filas con valores nulos generados por el desplazamiento de los lags.
- **Estandarización:** Dado que los sensores manejan magnitudes físicas muy dispares (Amperios vs Grados Celsius vs Velocidad), se ha aplicado `StandardScaler` para normalizar todas las variables a media 0 y desviación estándar 1, paso crucial para modelos como SVM y Redes Neuronales.

2.3 Desbalanceo de datos

El conjunto de datos presenta un fuerte desequilibrio de clases, característico de entornos industriales donde la operación normal predomina sobre los fallos. La variable objetivo se ha construido combinando las columnas de fallo originales (`grip_lost` y `ProtectiveStop`):

- **Clase 0 (OK):** Funcionamiento normal.
- **Clase 1 (Grip Lost):** Fallo de agarre de la pieza.
- **Clase 2 (Protective Stop):** Parada de seguridad.

Cabe destacar la existencia teórica de una **Clase 3**, correspondiente a la ocurrencia simultánea de ambos fallos. Aunque este escenario es físicamente posible y representa un estado válido del sistema, su prevalencia en el conjunto de datos es estadísticamente insignificante (casos anecdóticos).

Desde el punto de vista del aprendizaje automático, mantener una clase con un número tan extremadamente reducido de muestras es contraproducente: impide la generalización, dificulta la validación cruzada

estratificada (algunos *folds* podrían quedarse sin muestras de esta clase) y fuerza al modelo a memorizar ruido en lugar de aprender patrones. Por consiguiente, se ha decidido excluir estas instancias puntuales del entrenamiento para priorizar la robustez en la detección de los modos de fallo principales.

Para mitigar el desbalanceo restante entre la clase OK y los fallos individuales, hemos adoptado dos estrategias:

- 1. Cost-Sensitive Learning (Pesos de Clase):** Para los modelos de *Scikit-Learn* (SVM, Random Forest, Regresión Logística), hemos utilizado el parámetro `class_weight='balanced'`. Esto no modifica los datos, sino que ajusta la función de pérdida para penalizar proporcionalmente más los errores cometidos en las clases minoritarias.
- 2. Data-Level Approach (SMOTE):** Exclusivamente para el entrenamiento de la Red Neuronal (MLP), hemos aplicado el método de SMOTE. SMOTE interpola entre instancias de fallo vecinas en el espacio para generar nuevas muestras sintéticas con las que entrenar al modelo, de esta manera podemos equilibrar las clases.

3 Estudio de los datos

Antes de proceder al modelado, realizamos un análisis exploratorio para entender la estructura de las correlaciones espaciales (entre sensores) y temporales (autocorrelación).

3.1 Correlación Espacial y Redundancia

Calculamos la matriz de correlación de Pearson sobre las variables "raw". Como se observa en la Figura 1, existe una fuerte multicolinealidad entre ciertos grupos de sensores (por ejemplo, las temperaturas de las distintas articulaciones tienden a subir al unísono, y las corrientes están fuertemente ligadas a la velocidad).

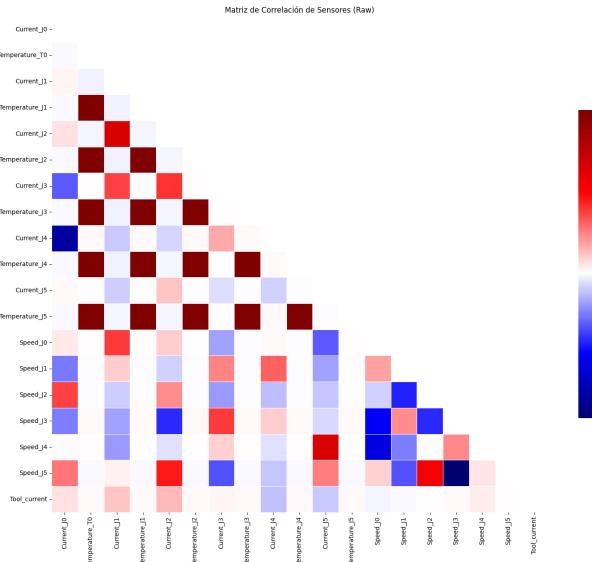


Figure 1: Matriz de correlación entre sensores.

En el heatmap anterior se puede observar la correlación entre las distintas variables. Se puede encontrar una alta correlación entre la temperatura de los distintos sensores, es decir, cuando la temperatura de un sensor aumenta, la de los otros sensores también tiende a aumentar. Exactamente lo mismo ocurre con la corriente y la velocidad, cuando la velocidad de un sensor aumenta, su velocidad también lo hace. Mientras que en la parte inferior derecha, podemos ver como la velocidad de las articulaciones se afectan también entre

si, esto tiene lógica, pues los movimientos de esta clase de maquinas suelen ser muy controlados, por lo que no suelen haber múltiples articulaciones moviéndose a la vez

Esta alta redundancia justifica el uso posterior de técnicas de reducción de dimensionalidad o regularización (como L2 en Ridge/SVM) para evitar problemas de inestabilidad en los modelos lineales.

3.2 Análisis de Autocorrelación (Justificación de Lags)

Para determinar científicamente el tamaño de la ventana de historia necesaria, analizamos los gráficos de Autocorrelación (ACF) y Autocorrelación Parcial (PACF) de los sensores físicos.

Mientras que sensores como la corriente mostraron un comportamiento cercano al ruido blanco (baja memoria), los sensores de temperatura mostraron una clara estructura autoregresiva. Observando el PACF (Figura 2), que aísla la correlación directa eliminando efectos intermedios, detectamos que la significancia estadística decae rápidamente tras los primeros instantes.

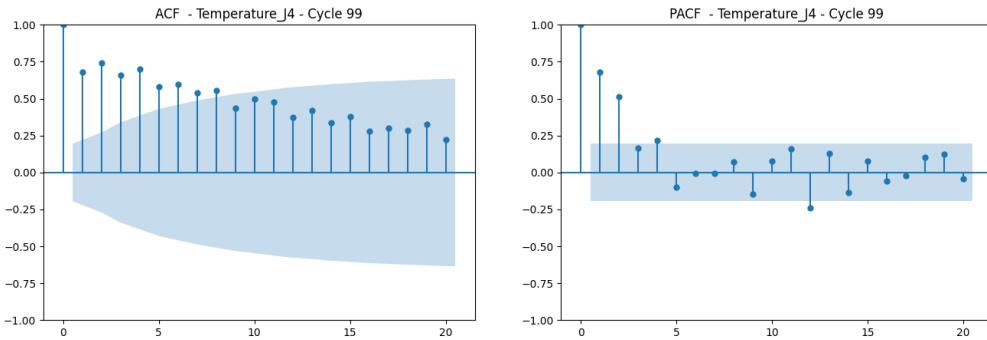


Figure 2: Gráfico de Autocorrelación para la temperatura de la articulación 4.

Como se observa en la figura, el gráfico de ACF (izquierda) muestra un decaimiento lento, indicativo de una fuerte dependencia temporal acumulada: la temperatura no cambia instantáneamente, por lo que el valor actual guarda relación con una larga historia de valores pasados.

Sin embargo, el gráfico de PACF (derecha) nos revela la dependencia directa real. Observamos picos de correlación muy significativos en los Lags 1 y 2 (coeficientes > 0.5), y una actividad residual que se mantiene relevante hasta el Lag 4. A partir del quinto retraso, las correlaciones oscilan mayoritariamente dentro del intervalo de confianza (zona sombreada), indicando que aportan poca información predictiva nueva.

Basándonos en esto, fijamos `NUM_LAGS = 5`. Esta ventana es suficiente para capturar la inercia del sistema sin introducir excesivo ruido ni aumentar innecesariamente la dimensionalidad del problema.

3.3 Relevancia de Características con el Target

Finalmente, tras aplicar los lags y el escalado, analizamos la correlación lineal directa de cada característica con la variable objetivo. Como muestra la Figura 3, los lags inmediatos ($t - 1$) de las variables de velocidad y par motor resultaron ser los predictores más potentes para distinguir los fallos, validando nuestra hipótesis de que la dinámica temporal es clave para la detección de anomalías.

3.4 Reducción de Dimensionalidad y Estructura Latente

Dado que la ingeniería de características (lags) expandió el espacio de entrada a 114 dimensiones, analizamos la redundancia y la separabilidad intrínseca de los datos mediante técnicas de proyección.

Análisis de Componentes Principales (PCA): Al aplicar PCA sobre el conjunto de datos estandarizado, observamos que es posible retener el **95% de la varianza explicada** utilizando solo **40 componentes principales**. Esto implica un ratio de compresión de **2.9x**, confirmando la alta redundancia lineal existente entre los sensores (especialmente temperaturas y sus retardos), tal como sugería la matriz de correlación inicial.

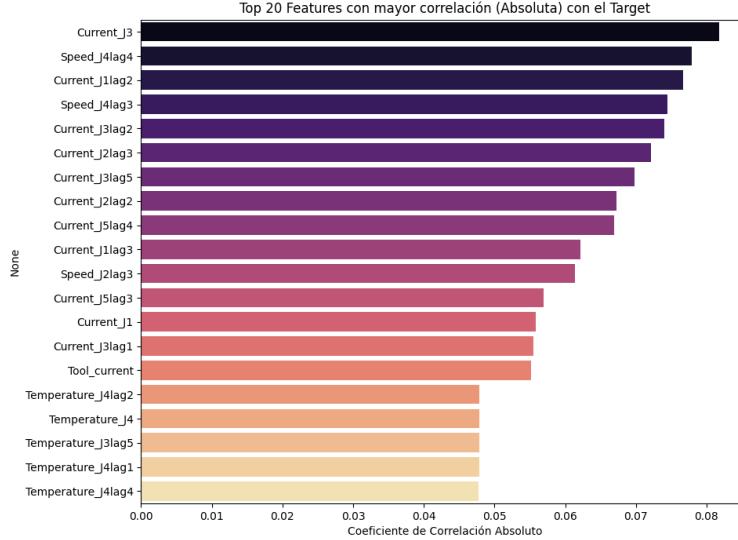


Figure 3: Top 20 características con mayor correlación absoluta con el Target.

Visualización con t-SNE: Para evaluar la separabilidad no lineal, proyectamos los datos a 2 dimensiones utilizando t-SNE (t-Distributed Stochastic Neighbor Embedding).

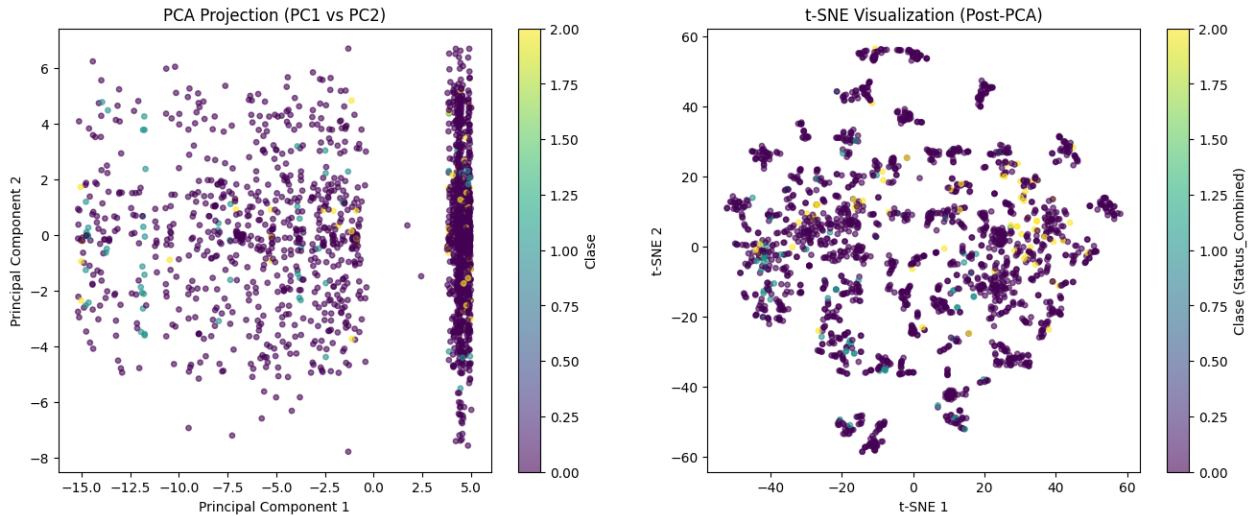


Figure 4: Proyección PCA (izq) vs t-SNE (der) coloreada por clase.

Como se observa en la Figura 4:

- **PCA (Izquierda):** Las clases de fallo (puntos amarillos y azules) no forman cúmulos distintos, sino que están dispersas y solapadas con la clase normal (puntos morados). Esto anticipa la dificultad que tendrán los modelos puramente lineales para trazar fronteras de decisión efectivas.
- **t-SNE (Derecha):** Al permitir relaciones no lineales, t-SNE revela una estructura local más rica. No existe realmente una buena separación, pero podemos ver que dentro de los puntos morados (clase OK), hay "islas" de puntos amarillos y azules, de manera que se puede empezar a distinguir entre las dos clases de fallo. Esto seguramente sea un indicativo de que los modelos no lineales vayan a funcionar mejor para este dataset.

4 Metodología de selección de modelos

Para seleccionar una buena combinación de hiperparámetros de cada uno de los distintos modelos estudiados hemos seguido la misma metodología, probar distintas combinaciones de hiperparámetros y seleccionar el mejor modelo que, con validación cruzada, demuestra los mejores resultados en la métrica del f1-score en el macro avg.

Debido al gran desbalance presente en las clases, la métrica de precisión puede ser engañosa debido a que un modelo prediciendo solo comportamiento normal tendría una puntuación muy buena, además, como queremos darle la misma importancia a cada una de las clases, hemos tomado la decisión de usar el f1-score en macro para la selección de las mejores combinaciones de hiperparámetros.

Por último, para ser consistentes en el tema de la agrupación por ciclos, la validación cruzada sigue la estrategia de StratifiedGroupKFold con 5 splits para asegurar la independencia de grupos (por ciclos) y la estratificación (mantener la misma proporción de clases, importante debido al desbalanceo ya mencionado).

5 Modelos Lineales

Con los modelos lineales nuestro objetivo es establecer cierta baseline de rendimiento. El uso de estos modelos es necesario debido a que, tal vez, nos encontramos con el caso de que el problema es lo suficientemente simple como para poder detectar y clasificar razonablemente bien los fallos del robot.

A continuación entraremos en más detalle sobre los resultados obtenidos en cada uno de los modelos no lineales.

5.1 Regresión Logística

Los mejores hiperparámetros encontrados en este caso han sido:

'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'

Análisis de Resultados: Los resultados no son demasiado prometedores, el modelo obtiene un 0.38 en el f1-score de macro. Esto resulta una mejora de 0.15 respecto al estudio preliminar del dataset mas no deja de ser una puntuación bastante baja.

- **Grip Lost (Clase 1):** El modelo alcanza un buen recall de 0.79, sin embargo se para con una precisión pésima de solo 0.2. Esto nos indica que el modelo tiene una cantidad exagerada de falsos positivos, en la matriz de confusión se puede apreciar que un 17% de las veces que no tenía ningún fallo decía que había perdido fuerza de agarre. Porcentualmente tal vez no parezca demasiado pero, al haber un desbalance tan exagerado, realmente hay muchos casos.
- **Protective Stop (Clase 2):** En este caso, contamos con un recall y precisión muy similares a los de la clase anterior.

Importancia de Atributos: En cuanto a los atributos que el modelo considera más importante vemos que la temperatura domina el ranking, ningún atributo de otro tipo se ha colado en el top 20.

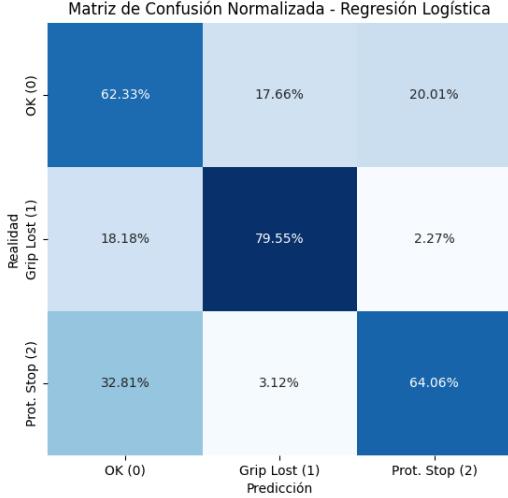


Figure 5: Matriz de Confusión (LogReg)

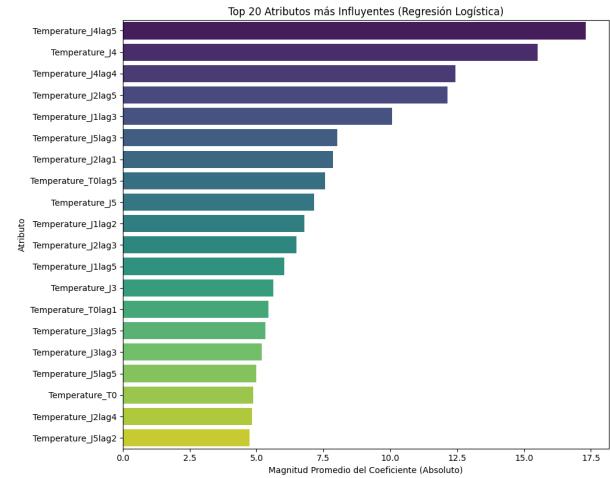


Figure 6: Top 20 Features (Coeficientes)

5.2 QDA (Análisis Discriminante Cuadrático)

El QDA modela las clases como distribuciones normales multivariantes con matrices de covarianza independientes, lo que permite generar fronteras de decisión curvas. Dada la alta multicolinealidad de nuestros sensores, fue necesario estabilizar la inversión de matrices optimizando el parámetro de regularización (`reg_param`), obteniendo un valor óptimo de **0.01**.

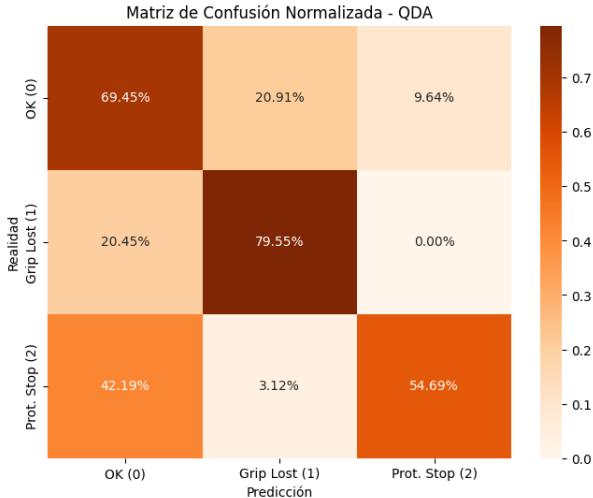


Figure 7: Matriz de Confusión (QDA)

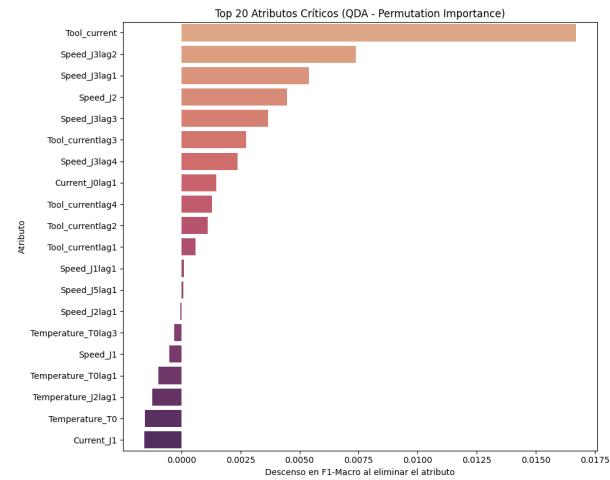


Figure 8: Importancia por Permutación

Análisis de Resultados: Como muestra la Figura 7 y el reporte de clasificación, el QDA se comporta como un modelo extremadamente sensible y poco preciso.

- **Grip Lost (Clase 1):** El modelo alcanza un excelente **Recall del 80%**, detectando la gran mayoría de los fallos de agarre. Esto sugiere que el cambio en la varianza de los datos durante este fallo es muy marcado. Sin embargo, su precisión es de solo 0.09, generando una tasa masiva de falsos positivos.
- **Protective Stop (Clase 2):** El rendimiento es moderado (Recall 0.55), indicando que la distribución de estos paros se solapa significativamente con la operación normal, dificultando su separación bajo la asunción gaussiana.

Importancia de Atributos: La Figura 8 muestra que la decisión del modelo se basa mucho en corriente de la herramienta (`Tool_current`) y sus retardos. Nuestra teoría es que al focalizarse tanto en esta variable, el modelo detecta muy bien los problemas de pinza, pero pierde robustez general. Además, existen barras con importancia negativa, de manera que parecen haber bastantes variables que introducen ruido en el modelo.

5.3 SVM Kernel Lineal

Las Máquinas de Vectores de Soporte (SVM) con kernel lineal buscan encontrar el hiperplano óptimo que maximiza el margen de separación entre las clases. A diferencia de la regresión logística, que minimiza una pérdida probabilística, el SVM se centra en los puntos más difíciles de clasificar (los vectores de soporte) cerca de la frontera de decisión. Los mejores hiperparámetros encontrados para este modelo han sido: 'C': 1, 'dual': False, 'loss': 'squared-hinge', 'penalty': 'l2'.

Análisis de Resultados: El reporte de clasificación sitúa al SVM como el mejor modelo entre los lineales con un f1-score en macro de 0.48 (respecto a un 0.37 y un 0.41 de los modelos anteriores).

- **Grip Lost (Clase 1):** El modelo, a diferencia de los otros lineales, tiene el peor recall de todos en la clase 1, sin embargo, tiene la mejor precisión con diferencia (0.23, sigue siendo muy baja pero una mejora considerable), de ahí que tenga el mejor f1-score.
- **Protective Stop (Clase 2):** Igual que con la clase 1, el recall es muy bajo en comparación con el resto de modelos no lineales, sin embargo, una vez más, notamos una mejora considerable en la precisión (0.22).

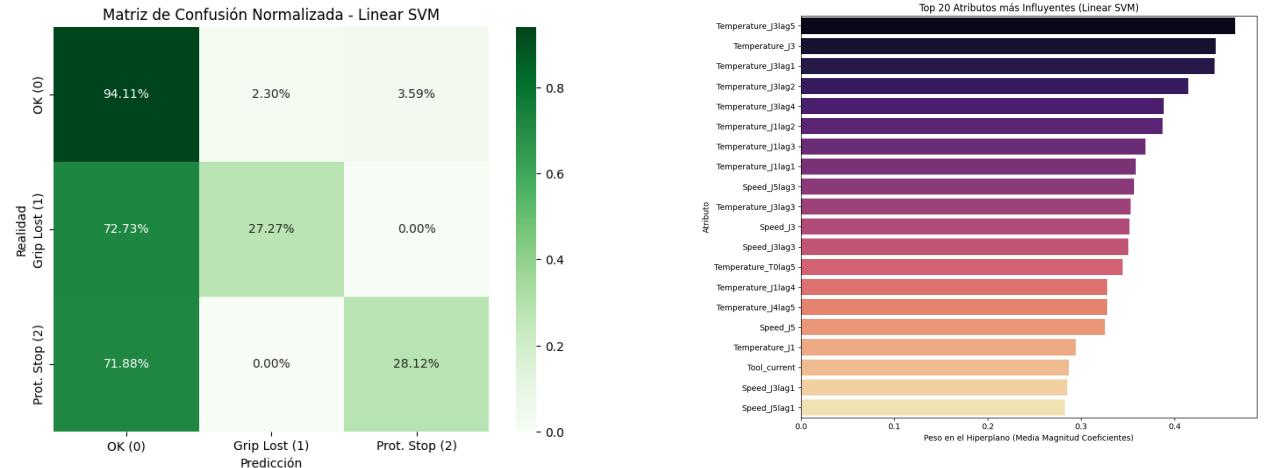


Figure 9: Matriz de Confusión (SVM Lineal)

Figure 10: Top 20 Features (Pesos del Hiperplano)

Análisis de Resultados: El SVM lineal suele ofrecer resultados más robustos que la regresión logística en espacios de alta dimensión (como el nuestro con 100 features tras los lags). La importancia de atributos es similar a la de la regresión logística, la temperatura domina el ranking mientras que, a diferencia de la regresión, la velocidad toma un papel importante a la hora de la clasificación.

5.4 Conclusión modelos lineales

Si nos mantenemos con la premisa de que el mejor modelo es el que demuestra mejores resultados en el f1-score del macro entonces el mejor modelo lineal encontrado es el SVM. Sin embargo, aquí hay que tener en cuenta alguna otra consideración adicional, ¿es preferible que el modelo indique que ha habido un fallo aunque no sea verdad antes que no lo indique y si haya? Esto depende totalmente del tipo de tarea que esté realizando el brazo robótico, si reportar demasiadas veces que hay un fallo solo por las dudas reduce mucho la velocidad de trabajo y eso es lo que buscamos, entonces nos quedariamos con el SVM, sin embargo, si lo

que queremos es tener muy en cuenta los fallos, arriesgandonos a reducir mucho la productividad, entonces el QDA sería el mejor modelo por el recall demostrado (aunque sea similar e incluso algo inferior al de la regresión, la estabilidad del modelo es algo mejor, lo que lo compensa).

6 Modelos No Lineales

Tras establecer la base con modelos lineales, exploramos arquitecturas no lineales para capturar patrones de datos más complejos. Dado que los sensores presentan interdependencias físicas (como la relación entre temperatura, par y velocidad), estos modelos buscan identificar anomalías que no son evidentes mediante simples combinaciones lineales de las variables, es decir, patrones complejos.

A continuación, analizamos los resultados obtenidos con modelos de kernel, conjuntos de árboles y redes neuronales, evaluando su capacidad para mejorar la precisión en las clases más desbalanceadas.

6.1 SVM con Kernel RBF

Utilizamos una support vector machine con el kernel RBF Gausiano, que funciona en un espacio con dimensión infinita de características, cosa que nos permite encontrar patrones que no sean lineales. Esto nos puede permitir encontrar patrones complejos en el entorno.

Con `class_weight='balanced'` para contrarrestar la class imbalance, implementamos además búsqueda de hiperparámetros con GridSearch, y encontramos que la mejor configuración era la siguiente:

- $C = 100$ y $\gamma = \text{'auto'}$

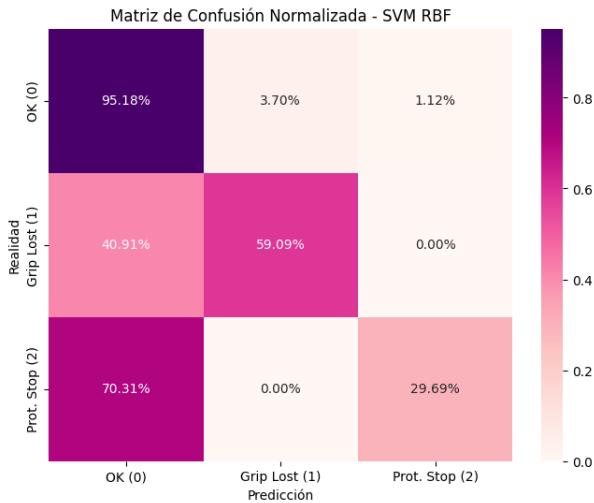


Figure 11: Matriz de Confusión (SVM RBF)

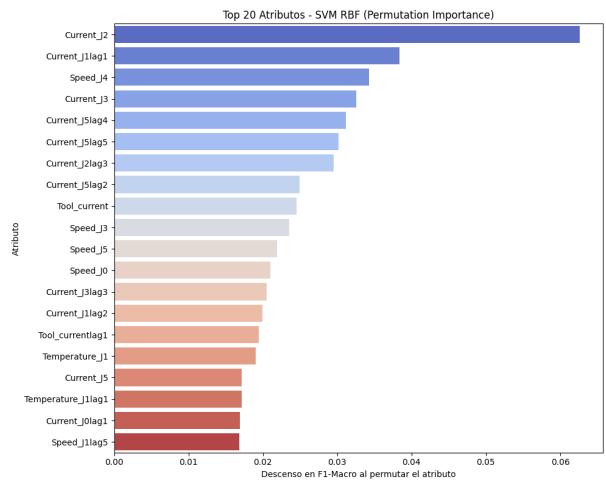


Figure 12: Importancia por Permutación

Análisis de Resultados: A pesar de la flexibilidad teórica del kernel RBF y el alto coste de penalización ($C = 100$), el modelo presenta dificultades significativas para caracterizar los fallos de seguridad, como se observa en la Figura 11:

- **Debilidad en Protective Stop (Clase 2):** Este es el punto crítico del modelo. La matriz muestra que el **70.31%** de las paradas de emergencia reales son clasificadas erróneamente como estado normal (OK). Con un *Recall* de apenas 0.30, el SVM tiende a "ignorar" estos eventos, probablemente porque, en el espacio transformado, los vectores de soporte de esta clase siguen estando demasiado "mezclados" con la clase mayoritaria.
- **Grip Lost (Clase 1):** El desempeño es superior, detectando correctamente el **59%** de los casos. Sin embargo, su precisión es baja (muchos falsos positivos), lo que sugiere que la frontera de decisión para esta clase es difusa.

Importancia de Atributos: Al utilizar un kernel no lineal, tenemos que usar *Permutation Importance* para estimar el efecto de cada parámetro, observamos que el modelo basa sus decisiones principalmente en las corrientes de las articulaciones 2 y 3 (*Current_J2*, *Current_J3*) y la velocidad de la articulación 4. A diferencia del QDA, que se centraba en la herramienta, el SVM distribuye su atención en la dinámica del brazo completo.

6.2 Random Forest

Random Forest es un método de ensamble que entrena una multitud de árboles de decisión y promedia sus predicciones. Lo elegimos por su capacidad nativa para modelar relaciones no lineales y su robustez frente al ruido y outliers. Los mejores hiperparámetros encontrados para este modelo han sido: 'class-weight': 'balanced', 'max-depth': 10, 'min-samples-split': 2, 'n-estimators': 5

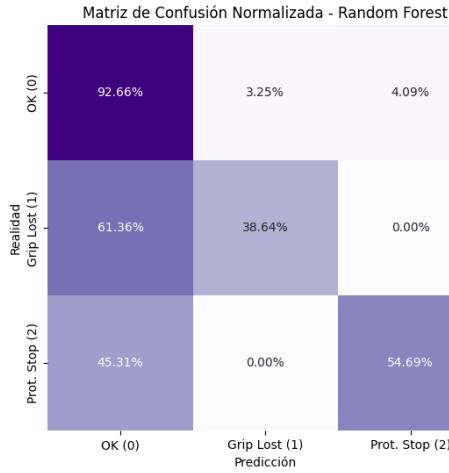


Figure 13: Matriz de Confusión (Random Forest)

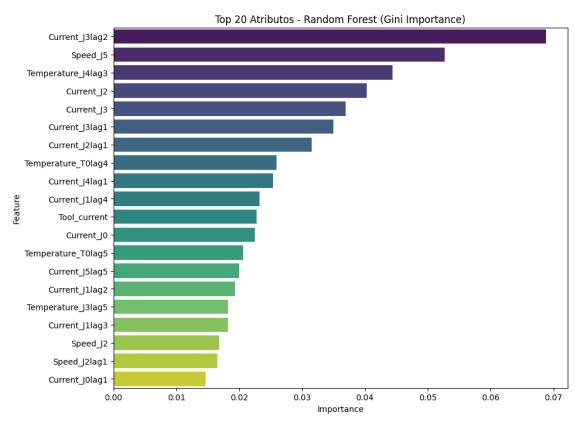


Figure 14: Importancia de Características (Gini)

Análisis de Resultados:

- **Debilidad en Protective Stop (Clase 2):** Este es el punto más importante del modelo, a diferencia del resto, consigue un éxito considerable no solo en el recall, si no también en la precisión. En comparación con el qda que consigue un 0.55 de recall y un 0.17 de precisión, nos encontramos con que obtenemos el mismo recall con casi el doble de precisión (0.32) lo cual es una mejora bastante considerable.
- **Grip Lost (Clase 1):** El equilibrio en esta clase es superior a los modelos lineales, si lo comparamos con el svm lineal (el modelo lineal más equilibrado), tenemos que, con un recall de 0.39 (respecto al 0.27 del svm) obtenemos la misma precisión de 0.23. Sin embargo, si lo comparamos con el resto de los modelos no lineales, vemos que es el peor con un margen considerable en esta clase.

Importancia de Atributos: A diferencia de los modelos lineales (exceptuando el qda), el random forest le da mucha importancia a la corriente que pasa por cada joint (articulación) del brazo robótico al igual que el rbf.

6.3 Red Neuronal (Multilayer Perceptron)

Implementamos un Perceptrón Multicapa (MLP) utilizando *PyTorch*. Debido a la severidad del desbalanceo, para este modelo utilizamos una estrategia de sobremuestreo (**SMOTE**), que genera muestras sintéticas de las clases minoritarias (usando $k = 5$ vecinos).

La arquitectura de la red se definió a partir de una búsqueda de hiperparámetros utilizando *Random Search* para explorar combinaciones de estos.

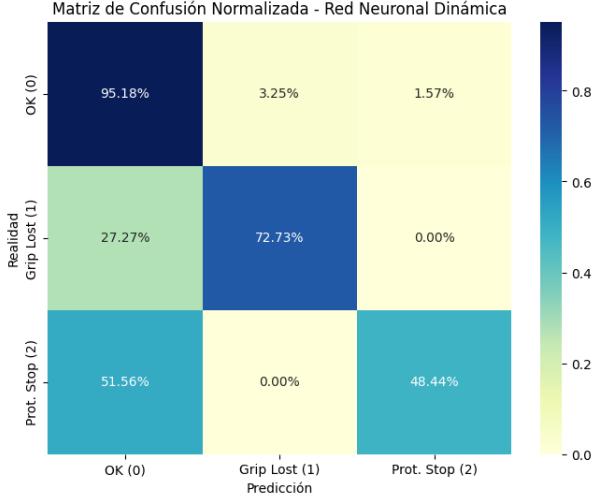


Figure 15: Matriz de Confusión (Red Neuronal)

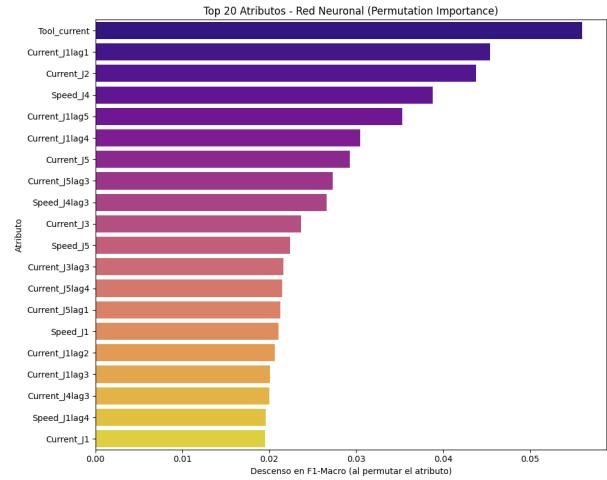


Figure 16: Importancia por Permutación

Análisis de Resultados: La mejor configuración encontrada fue una red con los siguientes hiperparámetros: `hidden_1: 256, hidden_2: 32, dropout: 0.2, lr: 0.001, batch_size: 64, epochs: 50`

- **Detección de Grip Lost (Clase 1):** La red alcanza un *Recall* del 73%, superando ampliamente a los modelos lineales. Sin embargo, solo tiene una precisión de (36%), indicando que hay confusión con la clase OK.
- **Dificultad con Protective Stop (Clase 2):** A diferencia de la clase 1, la red sufre para generalizar en las paradas de emergencia (Recall de solo 48%). Quizá SMOTE no es capaz de generar muestras sintéticas no ruidosas, o puede que simplemente sea una clase especialmente complicada.

Importancia de Atributos: Utilizamos *Permutation Importance* para interpretar el modelo (Figura 16). La red tiene una distribución de importancias bastante equilibrada, no hay una importancia extremadamente dominante. Le da mucho peso a la velocidad y corriente de algunas articulaciones en diferentes Lags, indicativo de que utiliza la dinámica temporal para la toma de decisiones.

7 Comparativa de modelos

Tras analizar individualmente los 6 modelos propuestos, en esta sección sintetizamos los resultados para identificar patrones generales, fortalezas y debilidades. La comparación se centra en la capacidad de generalización (F1-Score Macro) y en el compromiso entre detectar fallos y evitar falsas alarmas.

7.1 Resumen Cuantitativo

La Tabla 1 recoge las métricas clave obtenidas en el conjunto de test. Se observa una clara divisoria entre las familias de modelos: los algoritmos no lineales superan sistemáticamente a los lineales, confirmando la hipótesis planteada tras la visualización t-SNE (la frontera de decisión no es un hiperplano simple).

Modelo	Tipo	Macro F1	Rec. (Grip)	Prec. (Grip)	Rec. (P. Stop)	Prec. (P. Stop)
Regresión Logística	Lineal	0.37	0.80	0.10	0.66	0.11
QDA	Lineal	0.41	0.80	0.09	0.55	0.17
SVM Lineal	Lineal	0.48	0.27	0.23	0.28	0.22
SVM (RBF)	No Lineal	0.57	0.59	0.28	0.30	0.49
Random Forest	No Lineal	0.55	0.39	0.23	0.55	0.32
Red Neuronal (SMOTE)	No Lineal	0.61	0.73	0.42	0.31	0.38

Table 1: Comparativa de rendimiento detallada por tipo de fallo.

7.2 Precisión vs. Sensibilidad

El principal hallazgo de esta comparativa es el drástico *trade-off* existente entre precisión y sensibilidad (Recall), condicionado por el desbalanceo de clases:

- **Modelos 'Alarmistas' (QDA, Regresión Logística):** Estos modelos obtienen los mejores valores de recall para ambas clases de fallos. Sin embargo, son inviables en un entorno real debido a su bajísima precisión ($\approx 10\%$). Esto significa que, por cada fallo real detectado, el sistema genera 9 falsas alarmas, lo que obligaría a detener el robot constantemente sin motivo.
- **Modelos Conservadores/Equilibrados (SVM Lineal, SVM RBF, RF, NN):** Priorizan la precisión, reduciendo los falsos positivos, pero a costa de ignorar una gran cantidad de fallos reales en su mayoría.

7.3 El problema de la Clase 2 (Protective Stop)

Todos los modelos, exceptuando el random forest, muestran su peor rendimiento en la clase *Protective Stop*. Esto puede indicar que los sensores son insuficientes para predecir los fallos de este tipo. Debido a la falta de contexto en la página del dataset, no sabemos si los fallos de parada se deben exclusivamente al robot, si no fuera el caso, explicaría el porque los modelos en general predicen peor esta clase. Como contexto, los fallos de parada se pueden producir por muchos factores y aunque no todos, algunos son ajenos al robot. Uno de estos posibles fallos es que detecte un choque fuerte en el camino (por ejemplo en una industria podría ser una caja que se ha colocado en el trayecto del brazo o una persona) cosa que no tiene que ver con la mecánica del robot. Debido a que los modelos no son completamente 'tontos' a la hora de predecir este tipo de fallo, es posible que no sea el caso, sin embargo, con que se haya colado alguna instancia donde haya sido por algo ajeno al robot, al haber tanto desbalance, esto podría explicar el porque es consistentemente peor que la predicción en grip lost.

8 Combinación de modelos (Ensembles)

Como se ha podido observar en el apartado anterior, los modelos presentan cada uno distintos defectos y virtudes, algunos tienen una muy buena recall para algunas clases pero una terrible precisión, o quizás alguno es especialmente bueno para predecir una clase en específico. Para aprovechar estas fortalezas complementarias, decidimos implementar y comparar dos estrategias de ensamblaje: *Stacking Classifier* y *Voting Classifier*.

8.1 Stacking Classifier

Implementación: El *Stacking* entrena un meta-modelo (Regresión Logística con `class_weight='balanced'`) que aprende a combinar las predicciones de los modelos anteriores. En nuestro caso encontramos que el mejor Stacking Classifier es el que utiliza *RandomForest* + *NeuralNetwork* + *LinearSVC* como estimadores.

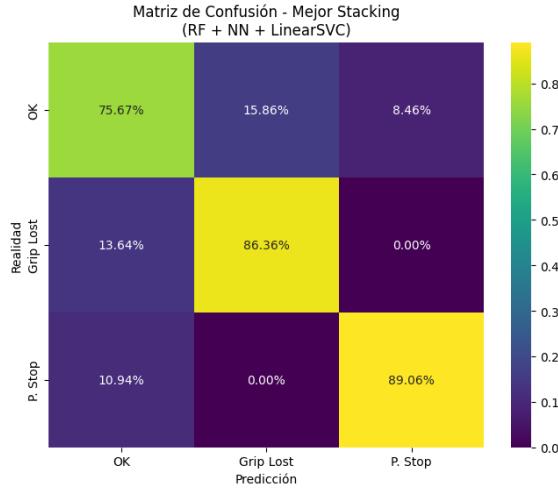


Figure 17: Stacking Inicial (Agresivo)

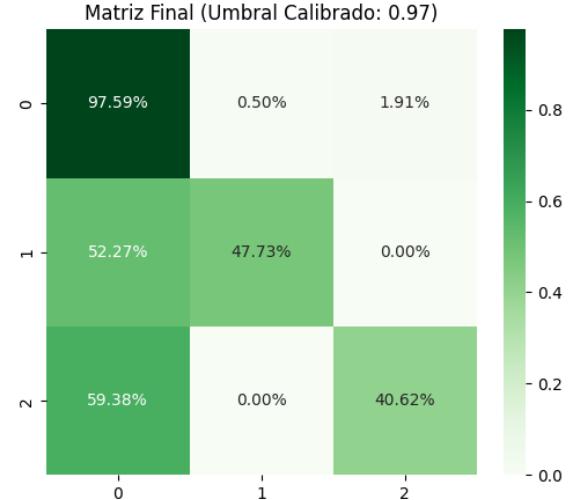


Figure 18: Stacking Calibrado ($\tau = 0.97$)

Resultados y Calibración: El comportamiento inicial del Stacking, visible en la Figura 17, ha resultado ser extremadamente agresivo. Aunque logra identificar una gran proporción de los fallos (diagonal de las clases de fallo con valores altos), también genera una tasa de Falsos Positivos inadmisible en una posible aplicación real (observables en la primera fila de la matriz, donde un porcentaje significativo de ciclos 'OK' fueron clasificados erróneamente como fallos).

Para mitigar este exceso de alarmas, intentamos calibrar el umbral de decisión maximizando el *F1-Score* en el conjunto de entrenamiento. El algoritmo converge en un umbral extremadamente alto ($\tau = 0.97$).

Sin embargo, como muestra la Figura 18, esto tiene un efecto contraproducente para la seguridad: el modelo se vuelve muy conservador. Pese a que reduce los falsos positivos (la clase OK recupera un 97% de acierto), la capacidad de detección de la clase crítica *Protective Stop* baja a un 40%, confundiendo la mayoría de paradas reales con funcionamiento normal.

8.2 Voting Classifier

Implementación: Como alternativa más robusta, implementamos un *Voting Classifier*. Este promedia los votos de los modelos que lo componen, en nuestro caso: **Random Forest, Red Neuronal y Regresión Logística** (seleccionados a través de un estudio de las posibles combinaciones de modelos).

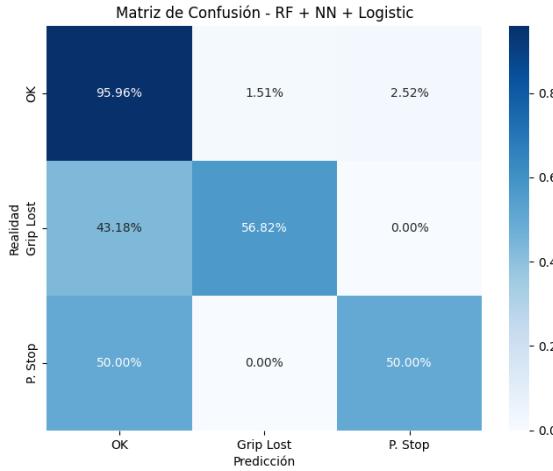


Figure 19: Matriz de Confusión del Voting Classifier (Modelo Final)

Resultados: La Figura 19 ilustra el rendimiento de esta combinación. A diferencia del Stacking calibrado, el Voting logra un equilibrio operativo viable:

- Mantiene una tasa de aciertos muy alta en la clase mayoritaria OK ($> 96\%$), minimizando las interrupciones innecesarias.
- Conserva una capacidad de detección significativa en las clases de fallo, situándose en un punto medio entre el comportamiento conservativo del Stacking calibrado y el ruido del stacking inicial.
- La confusión entre clases de fallo y normalidad se distribuye de manera más uniforme, lo que indica una generalización más sana.

8.3 Conclusión

Tras analizar ambas estrategias, descartamos el *Stacking Classifier* debido a su inestabilidad: oscila entre ser inoperante por exceso de falsas alarmas o inseguro por falta de sensibilidad tras la calibración.

Seleccionamos el **Voting Classifier** como el modelo final del proyecto. Su capacidad para mantener una precisión elevada en condiciones normales sin sacrificar la detección de aproximadamente la mitad de los eventos críticos (sin post-procesado adicional) lo convierte en la base más sólida. Sobre este modelo base, la aplicación posterior de técnicas de filtrado temporal (aprovechando la duración de los fallos) permitiría refinar aún más la robustez del sistema en un entorno de producción. Todo sea dicho, con como está actualmente no consideramos que sea viable aplicarlo en un entorno real, pero nos puede indicar que camino seguir.

9 Conclusiones y dificultades encontradas

Finalmente podemos concluir que, para este dataset, los modelos no lineales probados superan consistentemente a los modelos lineales. Esto nos indica que no es un problema de clasificación simple. Además, nos hemos encontrado con que las puntuaciones del f1, aunque han mejorado en los modelos no lineales respecto los lineales, siguen siendo bastante pobres si se quisiese usar alguno de los modelos en un caso real. Esto se puede deber a muchos factores distintos, entre los que creemos que el gran desbalance entre clases es el principal.

Aunque no esté incluído en el estudio, de forma separada hemos probado a añadir más features en los datos como la diferencia de temperatura respecto las temperaturas anteriores o la desviación estándar del cambio de los datos como la velocidad en los últimos segundos. Estas features parecen teóricamente relevantes pero una vez implementadas no han mejorado ninguno de los modelos, por lo que acabamos decidiendo no implementarlas. Otra feature que hemos debatido si añadir o no sería la de tiempo desde el inicio de ciclo, esta feature si que añadía una mejora sustancial (una diferencia de unos 0.05 en el f1-score de todos los modelos), sin embargo, tras debatirlo, decidimos no añadirla por el hecho de que nos limita los modelos entrenados a solo un patrón de movimiento del robot (el que hace repetidamente en el dataset, cada ciclo es la misma tarea de principio a fin).

References

- [1] Facultat d'Informàtica de Barcelona (FIB) - Universitat Politècnica de Catalunya (UPC). (2025). *Ficha de la asignatura: Aprendentatge Automàtic (APA)*. Disponible en: <https://sites.google.com/upc.edu/aprendentatge-automatic>
- [2] Möller, C., Tauscher, J. P., & Nguemeni, C. (2023). *UR3 CobotOps Dataset*. UCI Machine Learning Repository. Disponible en: <https://archive.ics.uci.edu/dataset/963/ur3+cobotops>
- [3] Scikit-learn Developers. (2024). *Ensemble methods: Random Forests and other randomized tree ensembles*. Documentación oficial. Disponible en: <https://scikit-learn.org/stable/modules/ensemble.html#forest>
- [4] Scikit-learn Developers. (2024). *Support Vector Machines (SVM): Mathematical formulation*. Documentación oficial. Disponible en: <https://scikit-learn.org/stable/modules/svm.html>
- [5] PyTorch Foundation. (2024). *Deep Learning with PyTorch: Neural Networks Tutorial*. Disponible en: https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
- [6] Scikit-learn Developers. (2024). *Time-related feature engineering*. Disponible en: https://scikit-learn.org/stable/auto_examples/applications/plot_cyclical_feature_engineering.html
- [7] Lemaitre, G., Nogueira, F., & Aridas, C. (2024). *Over-sampling methods: SMOTE and ADASYN*. Documentación de Imbalanced-learn. Disponible en: https://imbalanced-learn.org/stable/over_sampling.html
- [8] Scikit-learn Developers. (2024). *Cross-validation iterators for grouped data (GroupShuffle-Split)*. Disponible en: https://scikit-learn.org/stable/modules/cross_validation.html#group-shuffle-split