Universidad ORT Uruguay
Facultad de Ingeniería

Inteligencia Artificial Explicable: tecnología y transparencia para la Industria 4.0

Entregado como requisito para la obtención del título de Master en Big Data

Lua Iusim - 212851
Pía Marchesano - 159680

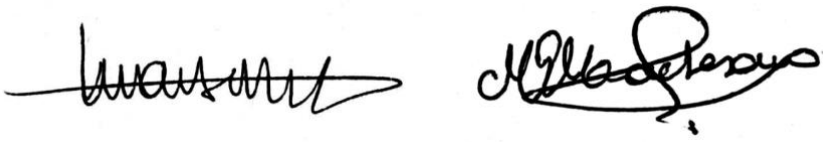
Tutor: Sergio Yovine

2023

Declaración de autoría

Nosotras, Lua Iusim y Pía Marchesano, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el Proyecto Final del Master en Big Data;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



Lua Iusim	Pía Marchesano
30/03/2023	30/03/2023

Agradecimientos

A Sergio Yovine por compartir sus conocimientos y guiarnos a lo largo de este proyecto.

A nuestros docentes y compañeros del Master en Big Data, por la formación y el intercambio.

A nuestros afectos, por el sostén y la compañía invaluable.

Abstract

En el marco de la Industria 4.0, resulta un desafío para las empresas hacer uso de la tecnología de forma de optimizar sus procesos, aumentando su rentabilidad y no perdiendo competitividad en el mercado.

El mantenimiento predictivo utiliza el potencial de la Industria 4.0 para aportar información relevante sobre las máquinas a futuro.

Las herramientas tecnológicas de inteligencia artificial exploradas, permiten desarrollar estrategias dentro del mantenimiento predictivo, basado en el monitoreo continuo de las máquinas, permitiendo realizar su mantenimiento cuando es necesario, minimizando costos de mantenimiento no programado y maximizando la vida útil de los componentes. Dentro de las aplicaciones del aprendizaje automático al matenimiento predictivo, se hace hincapié en resolver problemas de clasificación y de vida útil restante (RUL).

La forma en la que se encuentran disponibles los datos, condiciona las herramientas tecnológicas necesarias para la generación de valor a partir de ellos. En el presente trabajo se aborda uno de los problemas más frecuentes en el área de análisis de fallas: el desbalance de los datos. Para esto se toman estrategias como creación de una función de costos, variación de un umbral de probabilidad y aplicación de técnicas de desbalance.

En esta línea, se desarrollan modelos de *Machine Learning*, en donde el mejor es capaz de predecir con una *accuracy* de 99%, una *precision* de 86% y una *recall* de 90%, si una máquina fallará o no a partir de sus condiciones operativas.

Se aborda el tema de la Inteligencia Artificial Explicable (XAI) para datos tabulares, cuyo objetivo es brindar transparencia, interpretabilidad e imparcialidad a los resultados que generan los modelos de *Machine Learning*, con el fin de que puedan ser fácilmente comprensibles por el ser humano.

También se aborda la explicabilidad de modelos en formato de series temporales, formato de interés para el área de estudio. Los modelos son generados a partir de un estudio previamente realizado, que tiene como solución a un problema de cálculo de la RUL, una red neuronal recurrente, en particular LSTM.

Ambos modelos de explicabilidad están basados en *Shapley Values*, enfoque para aplicar la XAI que, con base en la teoría de juegos de coalición, permite entender las variables más influyentes en las predicciones.

Los modelos desarrollados superan en *performance* y profundidad explicativa a la bibliografía disponible para el caso de datos tabulares y, setean un precedente para la explicabilidad a partir de modelos complejos para datos en forma de serie temporal.

Palabras clave

Industria 4.0; Mantenimiento predictivo; Inteligencia Artificial Explicable; Aprendizaje automático; Desbalance de clases; Detección de fallas; Vida útil restante (RUL); Valores de Shapley

Índice

1. Introducción

1.1. Industria 4.0

En el contexto de la Industria 4.0 [1], [2], [3], origen de la Cuarta Revolución Industrial, resulta de suma importancia comenzar a incorporar tecnologías en las industrias. La Industria 4.0 combina técnicas avanzadas de producción y operaciones con tecnologías inteligentes que se integrarán en las organizaciones, las personas y los activos. Esta revolución se encuentra marcada por la aparición de nuevas tecnologías, que deberán ser identificadas por las organizaciones para satisfacer sus necesidades y así no perder competitividad en el mercado.

La nueva revolución industrial supone un cambio que proporciona acceso en tiempo real a los datos y la inteligencia de negocio, transformando de forma integral la gestión. En este sentido, las empresas deben cambiar la forma en la que fabrican y distribuyen sus productos. Es por esto que las fábricas están integrando nuevas tecnologías volviéndose fábricas inteligentes, equipándose con sensores avanzados, *software* integrado y robótica que recopilan y analizan datos y permiten una mejor toma de decisiones.

En línea con lo anterior, existe un flujo de información y acciones PDP - *physical-to-digital-to-physical* - que implica capturar datos del mundo físico y crear un registro digital, interpretar los mismos mediante analítica avanzada, análisis de escenario e inteligencia artificial y aplicar algoritmos para traducir las decisiones del mundo digital a información que estimule acciones y cambios en el mundo físico.

El desarrollo de las fábricas inteligentes brinda la oportunidad de analizar los datos recopilados en las plantas de producción, proporcionando herramientas para el mantenimiento predictivo con el fin de minimizar el tiempo que los equipos están sin operar.

1.2. Mantenimiento industrial

Una de las principales tareas asociadas a los procesos industriales es el mantenimiento de la maquinaria, también conocido como mantenimiento industrial [4], [5], [6], [7], [8]. Resulta menester para las organizaciones invertir recursos y mecanismos en el mantenimiento para alargar la vida útil de los equipos, así como para reducir costes.

El objetivo del mantenimiento industrial es asegurar la producción continua y predecible de los activos industriales y su calidad. Este puede entenderse como el conjunto de actividades y acciones técnicas y administrativas destinadas a mantener o reparar un activo para lograr un óptimo funcionamiento de las instalaciones, equipos y maquinaria.

La inversión en mantenimiento genera beneficios para las organizaciones como prevenir accidentes laborales, evitar o disminuir pérdidas por paros en la producción, extender el ciclo de vida de los equipos y reducir costes, entre otros. De este modo se puede evitar un negativo impacto financiero, así como garantizar la excelencia operativa, la integridad de los operadores y la competitividad de la empresa.

El mantenimiento industrial puede clasificarse en tres tipos: correctivo, preventivo y predictivo.

1.2.1. Mantenimiento correctivo

El mantenimiento correctivo es una respuesta reactiva a los fallos en la maquinaria. El fin de este tipo de mantenimiento es corregir los defectos en los equipos una vez que han dejado de funcionar, es decir, corrige una vez que se ha detectado la avería. Las piezas defectuosas son reemplazadas cuando dejan de ser funcionales, garantizando que se utilicen durante la totalidad de su vida útil. Es la estrategia de mantenimiento más sencilla pero más costosa. El tiempo de inactividad y los costos de mantenimiento no planificados afectan a la productividad y el rendimiento.

1.2.2. Mantenimiento preventivo

El mantenimiento preventivo, también conocido como mantenimiento en función del tiempo implica la planificación de acciones de mantenimiento. Este tipo de mantenimiento implementa medidas sistemáticas de sustitución de piezas cada un intervalo de tiempo determinado, independientemente de la condición del componente. El mantenimiento preventivo tiene como justificación que suele ser más económico reemplazar una pieza en concreto a tener que reparar una máquina entera cuando esta se rompa. Sin embargo, esto puede generar que se tomen acciones innecesarias y, en ocasiones, un aumento de los costos operativos por sustituir componentes que aún podrían durar más tiempo.

1.2.3. Mantenimiento predictivo

El mantenimiento predictivo se basa en el monitoreo continuo de la condición, salud e integridad de una máquina, permitiendo realizar el mantenimiento únicamente cuando es necesario. El mantenimiento predictivo utiliza el potencial de las nuevas tecnologías de la industria 4.0 para aportar información relevante sobre las máquinas. Permite la detección temprana de fallas mediante la utilización de herramientas predictivas basadas en datos históricos, con técnicas de aprendizaje automático, factores de integridad y métodos de inferencia estadísticas, entre otros.

El objetivo del mantenimiento predictivo es optimizar el equilibrio entre el mantenimiento correctivo y preventivo, permitiendo el reemplazo justo a tiempo de los componentes, minimizando el costo del mantenimiento no programado y maximizando la vida útil del componente.

Si bien el mantenimiento predictivo es una de las soluciones de mantenimiento más complejas de implementar, trae acompañado una serie de ventajas y beneficios para aquellas organizaciones que lo lleven a cabo. Entre ellas se destacan: maximizar la vida útil de los componentes, reducir las pérdidas de producción no programadas, optimización de los recursos y prevención de posibles fallas.

1.2.3.1. Aplicaciones del aprendizaje automático al mantenimiento predictivo

Con el fin de poder aplicar el mantenimiento predictivo de manera adecuada resulta importante utilizar el aprendizaje automático para encontrar patrones ocultos en los datos. Para ello es importante entender qué tipo de datos se debe recopilar, también se debe definir con qué frecuencia se leerán los sensores, cómo se almacenarán los datos y cómo serán procesados.

La cantidad y la calidad de los datos serán la principal limitación. Resulta de suma importancia contar con una buena cantidad de ejemplos de fallas. En general los fallos son poco comunes y resulta necesario almacenar todo hasta que se registren suficientes ejemplos. Asimismo, el etiquetado es el factor principal que determina la calidad de los datos. Se debe registrar eventos como fallas categorizadas en sus tipos, mantenimientos, cortes de energía y cualquier acontecimiento que pueda afectar las mediciones.

El aprendizaje automático puede ser aplicado al mantenimiento predictivo en diferentes modelos como *Remaining Useful Life* (RUL), *Classifier* y *Unsupervised Anomaly Detection*.

RUL

Un predictor RUL dará como resultado una aproximación de tiempo antes de que ocurra una falla, es decir, la cantidad de vida útil restante de un componente.

Classifier

Este modelo predecirá si una pieza fallará o no. También puede enmarcarse como un problema de clasificación multiclase para saber qué tipo de falla esperar. En particular, si la predicción está enmarcada en una ventana de días, se denomina *Window Classifier*.

Unsupervised Anomaly Detection

Se utiliza principalmente en datos sin etiquetar, donde se puede estudiar el comportamiento para detectar anomalías que probablemente impliquen fallas. Estos métodos son difíciles de validar.

Los campos de trabajo del presente documento serán *Classifier* y *RUL*.

1.3. Inteligencia Artificial Explicable (XAI)

Con el uso creciente de los modelos de *Machine Learning*, especialmente de *Deep Learning*, se vuelve cada vez más importante poder explicar sus predicciones. La explicabilidad de la inteligencia artificial tiene como objetivo brindar información que permita a los desarrolladores de modelos evaluar la robustez y solidez de los mismos.

Si bien es común que muchos modelos de *machine learning* sean una caja negra, la falta de una comprensión razonable de cómo funcionan estos modelos no permite que los proyectos tengan éxito. Es importante poder responder a la pregunta: ¿cómo toma las decisiones el modelo?

Existe un *trade-off* entre el rendimiento de un modelo y su interpretabilidad. Ciertos modelos sencillos pueden explicar cómo han llegado a una predicción en particular. Sin embargo, modelos más complejos, que producen un mejor rendimiento, son percibidos como caja negra, ya que resulta difícil explicar las decisiones que ha tomado.

La Inteligencia Artificial Explicable (XAI) [9], [10], [11], [12] es un conjunto de métodos y procesos que permite entender cómo y por qué un algoritmo toma decisiones o realiza predicciones. En otras palabras, permite la descripción de un modelo de inteligencia artificial en profundidad, contribuyendo a la precisión del resultado, la imparcialidad y sobre todo generando transparencia.

Entonces, la Inteligencia Artificial Explicable resulta crucial para una organización que busque aplicar modelos que explican o predicen fenómenos en producción. Dejar de ver los modelos como caja negra y entender de dónde provienen los resultados, no solo generará confianza en los mismos, sino que aportará inigualable valor conceptual para la resolución de problemas.

XAI es un esfuerzo continuo en el campo de la inteligencia artificial para aumentar los métodos y resultados del *machine learning* con una explicación comprensible para los humanos. Para que las organizaciones confíen en el resultado de un modelo deben entender la explicación del mismo.

La transparencia, la capacidad de hacer preguntas y la facilidad de comprender las decisiones por parte de un humano, son las claves para la interpretación de un modelo.

En particular para un caso de mantenimiento industrial, en donde se busque predecir una falla en una máquina o el momento adecuado de recambio de un repuesto, existirá una diferencia sustancial entre un modelo que entregue un tiempo de vida útil o fecha de recambio, contra uno que indique además qué variables tomó en cuenta para calcularlo, brindando entonces información adicional que pueda luego convertirse en estrategias o planes de acción para extender ese tiempo de recambio.

Por otra parte, la explicabilidad puede también ayudar a los desarrolladores a garantizar que el sistema funciona de forma esperable, mitigar riesgos legales, de cumplimiento, seguridad y reputación de la inteligencia artificial en producción. Incluso estas técnicas, en ocasiones, son utilizadas para depurar modelos.

Suele decirse que "*La IA explicable es uno de los requisitos clave para implementar la IA responsable*" [13].

Hasta el desarrollo de esta rama de la inteligencia artificial, la evaluación de los modelos entrenados se basa en gran medida en herramientas como matrices de confusión, curvas ROC, parámetros como *precision*, *recall*, *accuracy* y *F1-scores*: todas propiedades estadísticas. Estas herramientas muestran que un modelo se comportará de manera predecible, es decir, hay un desempeño estadístico conocido que podrá ser considerado replicable, siempre que se utilice en datos comparables a los utilizados en los conjuntos de entrenamiento, validación y *test*.

Sin embargo, dichos métodos de evaluación no permiten conocer el rendimiento de un modelo en un solo punto de datos. Como se ha comentado anteriormente, en general no es de conocimiento qué combinación de factores es responsable de un resultado dado y por lo tanto tampoco cuando el resultado puede ser considerado confiable.

Ahora bien, dentro de la explicabilidad de modelos, existen las explicaciones globales y las locales. Las primeras, sirven para describir el comportamiento del algoritmo en general, es decir, qué variables contribuyen al modelo para tomar decisiones y cómo afectan los valores de esas variables a las predicciones en general. Las segundas, se utilizan para explicar el comportamiento del algoritmo para casos concretos, es decir, cómo afectan los valores de cada variable a una predicción determinada, permitiendo abordar la falta de profundidad y transparencia que se explicaba en el párrafo anterior.

En suma, a partir del análisis de los modelos se puede obtener el efecto de cada característica (*feature*) en el global de las predicciones del modelo y el efecto de cada *feature* para una predicción en particular.

1.3.1. SHAP

Uno de los enfoques utilizados para aplicar la inteligencia artificial explicable es SHAP (*Shapley Additive exPlanations*) que, con base en la teoría de juegos de coalición, permite entender qué variables tienen más influencia en las predicciones [14], [15].

Este enfoque calcula la contribución de cada característica a la predicción final mediante el valor de Shapley (*Shapley Values*) [16].

Cada característica o *feature* tendrá asignado un valor de Shapley que, cuanto más grande es su valor absoluto, más influencia tiene la variable en el resultado final.

En este contexto, se considerará como juego a la tarea de predicción para una sola observación x_0 y como jugadores a los *features* de x_0.

Ahora bien, los *Shapley Values* se calculan considerando todas las posibles combinaciones de jugadores y se determina la contribución de cada jugador en cada combinación. Luego, se promedia esta contribución a lo largo de las combinaciones posibles.

En suma, en el contexto de modelos predictivos:

- Juego: tarea de predicción para una sola observación x_0
- Jugadores: *features* de x_0 que colaboran para recibir la "ganancia" o "pago".
- Pago o ganancia por jugador: contribución de cada característica, es decir, cuánto influye el *feature* en la predicción del modelo. Es el *Shapley Value* de cada *feature*.
- Pago o ganancia total: suma de los *Shapley Values* de todos los jugadores en x_0.

El cálculo de *Shapley Values* es costoso computacionalmente. Sin embargo, existen ciertas propiedades que ayudan a optimizar los algoritmos:

- Simetría: si dos jugadores son iguales, sus valores de shapley son iguales.
- Aditividad: si hay un juego que se puede dividir en dos, los componentes Shapley también se pueden descomponer.
- Jugador nulo: si un jugador no aporta valor al juego, su valor es 0.
- Eficiencia: la ganancia total siempre es la suma de las contribuciones de cada jugador. Por lo tanto, sumando los *Shapley Values* de las variables para una observación x_0, se obtiene la diferencia entre la predicción en x_0 y el *baseline* (esperanza de todas las predicciones).

$$\textit{Ganancia total} = \textit{Shapley Values } (x_0) = f(x_0) - E(f(x))$$

Siendo *f* el modelo y *f(x)* la predicción.

La esperanza se estima como la media de un conjunto de datos, definido en función del costo computacional en el que se desee incurrir.

Considerando estas propiedades, al agrupar los *Shapley Values* puede observarse el comportamiento global del modelo. En otras palabras, sumando los valores Shapley que se obtienen en cada predicción por cada *feature*, es posible observar el comportamiento de cada variable en la totalidad del set de datos y entregar un resultado general. Esto convierte al algoritmo de explicabilidad en uno además de local, global.

1.4. Objetivos

En el contexto de mantenimiento predictivo se explorarán modelos de *Machine Learning* y métodos de explicabilidad. Los campos de trabajo serán clasificación y RUL. Para el primero, se utilizará un *dataset* tabular, formato habitual para este tipo de problemas, mientras que para el segundo, un *dataset* en forma de series temporales, formato en el que suelen presentarse los datos para el cáclulo de la vida útil restante.

La estructura del trabajo será la siguiente:

- Desarrollo y evaluación de modelos predictivos para set de datos tabular.
- Desarrollo de un modelo explicable para el set de datos tabular.
- Comparación de los modelos obtenidos con el *paper "Explainable Artificial Intelligence for Predictive Maintenance Applications"* [17].
- Desarrollo de un modelo de explicabilidad para el set de datos en formato series temporales.

Los detalles técnicos de implementación se encontrarán en los anexos que serán referenciados a lo largo del texto.

2. Caso: Detección de fallas en datos tabulares

En el presente capítulo se trabajará con un *dataset* de mantenimiento predictivo de máquinas - *predictive maintenance* -. La elección surge de la existencia del *paper* [17] y del afán de profundizar en los hallazgos obtenidos para un problema de detección de fallas.

2.1. Descripción del conjunto de datos

El conjunto de datos se encuentra disponible en la plataforma *UCI Machine Learning Repository* [18] y es de naturaleza sintética, debido a que los conjuntos de datos reales dentro del campo de mantenimiento predictivo resultan difíciles de obtener.

A partir de él, se explorará en la generación de algoritmos de clasificación binaria y multiclase y el objetivo será predecir la variable *Target* o *Failure Type* según corresponda. La clasificación multiclase se profundizará únicamente en el anexo.

El *dataset* consta de 10000 puntos de datos almacenados como filas y 10 variables. Las mismas son:

- UDI: número de fila.
- Product ID: consta de una letra L, M o H como variantes de calidad del producto y un número de serie específico de la variante.
- Type: consta de una letra L, M o H como variantes de calidad del producto, refieren a *Low*, *Medium* o *High* respectivamente.
- Air Temperature (K): temperatura del aire.
- Process Temperature (K): temperatura del proceso.
- Rotational Speed (rpm): velocidad rotacional.
- Torque (Nm): torque.
- Tool Wear (min): desgaste de la herramienta.
- Target: variable binaria que toma valores de 1 si la máquina incurrió en una falla y 0 en caso contrario.
- Failure Type: variable que describe qué tipo de falla presentó la máquina. Sus valores pueden ser *Heat Dissipation Failure (HDF)*, *Overstrain Failure (OSF)*, *Power Failure (PWF)*, *Tool Wear Failure (TWF)*, *Random Failure (RNF)* o *No Failure*.

2.2. Ingeniería de atributos

Antes de comenzar con la generación de algoritmos resulta menester realizar un análisis exploratorio de datos (EDA) [19], [20], el cual tendrá como objetivo analizar y explorar los datos, descubrir patrones, identificar anomalías y valores atípicos, sugerir hipótesis y verificar supuestos.

Para ello, se explorará en la distribución de las variables que componen el *dataset*, así como en la presencia de valores NA, registros duplicados, correlación entre variables y búsqueda de patrones.

Luego de profundizar en el análisis[1], se concluye la ausencia de datos NA y de registros duplicados. Por otra parte, se considera despreciable la presencia de *outliers* por lo que se decide no realizar modificaciones en el *dataset* a partir de dicho análisis, así como tampoco a partir del análisis de correlación entre variables.

Al observar la cantidad de puntos que presentan las variables *Target* y *Failure Type* en cada una de sus clases, se aprecia que existen inconsistencias. Existen registros que según la variable *Target* han fallado y según la variable *Failure Type* no y viceversa. Por este motivo y con el fin de que los datos sean consistentes, se eliminan aquellos registros que presenten incongruencias. Luego de esta limpieza se obtiene un conjunto de datos de 9973 registros, de los cuáles 330 fallan y 9643 no.

[1] Ver Anexo 1 – Ingeniería de atributos para clasificación binaria

Luego de las transformaciones realizadas, se obtiene el *dataset*[2] con el que se trabajará en la generación de algoritmos. Se procede a observar la distribución de la variable a predecir.

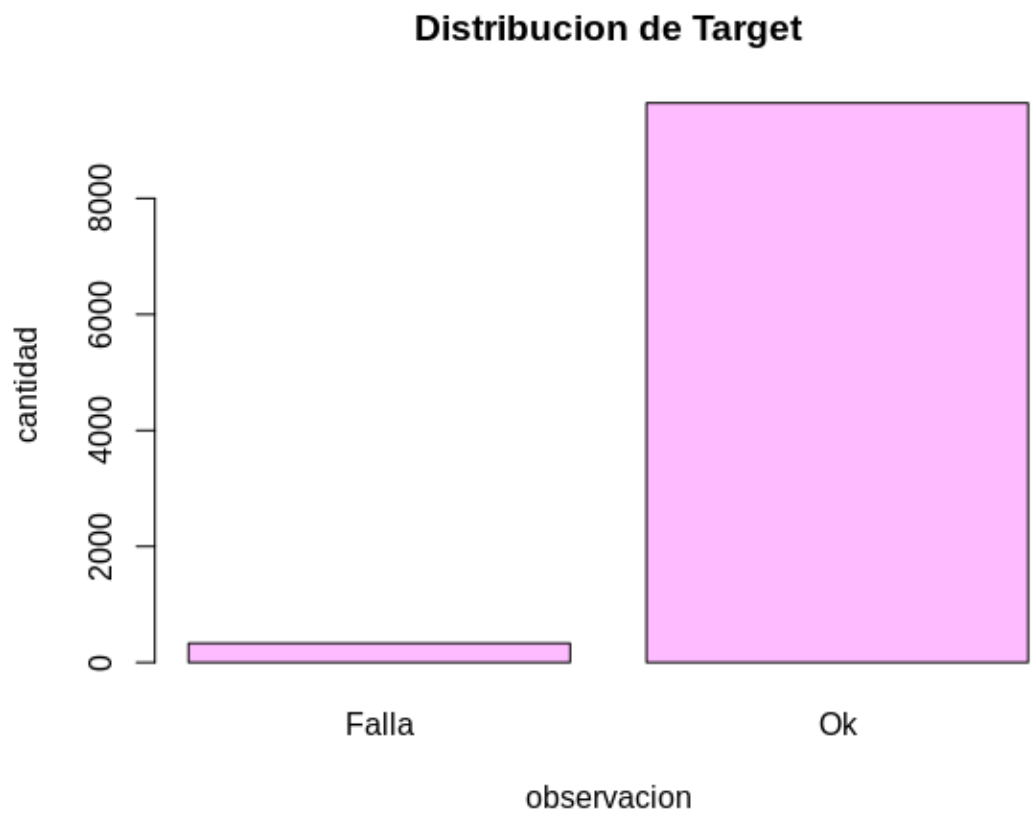


Figura 1. Distribución de la variable *Target*

El set de datos presenta un desbalance entre sus clases, existen considerablemente más observaciones para los cuáles el valor es “Ok” en comparación con los que el valor es “Falla”. Este punto deberá ser tenido en cuenta al momento de generar algoritmos ya que podría condicionar de forma negativa el entrenamiento de los modelos, por lo que deberán explorarse acciones de mitigación o eliminación de este desbalance.

[2] Ver Anexo 2 – Dataset para clasificación binaria

2.3. Clasificación binaria

La disparidad en las frecuencias de las clases observadas puede tener un impacto negativo en el ajuste de los modelos para problemas de clasificación. Las técnicas o estrategias[3] para resolver el desequilibrio serán:

- Desarrollo de una función de costos.
- Variación de umbral de forma de impactar en la matriz de confusión.
- Estrategias de balanceo aplicadas a los datos de entrenamiento.

2.3.1. Métrica objetivo y función de costos

La métrica que se utilizará para optimizar los modelos será *cost*. Esta será una función de costo definida previamente en el apartado *utils*[4], que penalizará aquellas predicciones que resulten más costosas para la empresa en caso de ser incorrectas. La función de costos, que es una función de pesos, se ha definido, en primera instancia, de la siguiente forma:

- Sumar 10 para aquellos casos en los que se prediga "Falla" y en realidad no existiese.
- Restar 25,5 para aquellos casos en los que se prediga "Falla" correctamente.

La función de costos utilizada en esta solución es tal que le indica al algoritmo que la empresa podría ahorrar 25,5 unidades al predecir correctamente, mientras que le costaría 10 unidades más el predecir incorrectamente.

El análisis y discusión sobre las magnitudes de los pesos de la función de costos será realizado en el punto 2.4.1.3.

[3] Ver Anexo 3 – Estrategias para resolver el desbalance
[4] Ver Anexo 10 – Código Utils

2.3.2. Umbral

La existencia de un umbral de probabilidad que define a partir de qué valor una observación es considerada de una clase u otra, impacta en la distribución de los resultados. Entonces, *df.thr_vec* será un valor o vector de valores que la función de costos utilizará para encontrar el mejor resultado. Al variar el umbral, se varía la probabilidad por encima de la cual cada observación es considerada como "Falla". Al bajar el umbral, es posible disminuir el efecto que genera el desbalance del *dataset*. En primera instancia el vector de umbrales será definido como: seq(0.25,0.30,by=0.01), es decir una secuencia que comienza en 0,25, termina en 0,30, y se recorre con paso 0,01.

2.3.3. Estrategias de balanceo de clases

Las técnicas o estrategias [21] para resolver el desequilibrio serán aplicadas únicamente al conjunto de entrenamiento. En este trabajo serán las siguientes:

- *Down-sampling*: esta técnica selecciona un subconjunto aleatorio de la clase mayoritaria en el conjunto de entrenamiento para que la frecuencia de clase coincida con la clase minoritaria.
- *Up-sampling*: esta técnica selecciona un muestreo aleatorio con reemplazo de la clase minoritaria para que tenga el mismo tamaño que la clase mayoritaria.
- *Hybrid methods*: estas técnicas reducen la cantidad de muestras de la clase mayoritaria y sintetizan nuevas observaciones de la clase minoritaria. En este trabajo se aplicará *ROSE*.

2.4. Algoritmos para clasificación binaria

Con el fin de resolver el problema de clasificación se realiza una investigación[5] de modelos mediante la utilización de algoritmos de Naive Bayes, Árboles de Decisión, Random Forest y Boosting Machine.

A partir de esto, se seleccionarán los mejores modelos para poder trabajar en la optimización de los mismos. Visualmente:

	Umbral	Costo en test	Accuracy	Precision	Recall
Naive Bayes (naive_bayes)	0,2728	-0,2487	0,967	0,500	0,485
Random Forest (rf)	0,2840	-0,6525	0,984	0,689	0,939
Random Forest (ranger)	0,2500	-0,6424	0,983	0,674	0,939
Boosting Machine (gbm)	0,2540	-0,3501	0,974	0,621	0,545
Boosting Machne (xgbTree)	0,2820	-0,5702	0,982	0,692	0,818
Árboles de Decisión (rpart)	0,2760	-0,1179	0,971	0,833	0,152

Tabla 1. Resultados obtenidos para los modelos de clasificación

Dado que se busca minimizar los costos, los modelos que mejor satisfacen esta necesidad son *rf*, *ranger* y *xgbTree*. Estos, también son aquellos que obtienen mejor *recall*, métrica seleccionada en este trabajo como una de las principales para evaluar la bondad del modelo. Dado que *rf* y *ranger* son dos algoritmos de Random Forest, se seguirá adelante únicamente con *rf*.

Se decide realizar un zoom a los valores de cost de *rf* y *xgbTree* para evaluar si puede existir *overfitting*.

	Costo en train	Costo en test
rf	-0,5985	-0,6525
xgbTree	-0,5442	-0,5702

Tabla 2. Costos en *train* y *test* para *rf* y *xgbTree*

[5] Ver Anexo 4 – Desarrollo de modelos para clasificación binaria

A partir de los valores expuestos en la tabla, no se puede presumir que exista *overfitting*, por lo que se continuará con la optimización de estos modelos.

2.4.1. Optimización de resultados

El objetivo de este subcapítulo es mejorar los resultados obtenidos. Esto se intentará a través de estrategias de balanceo, cambios de umbral y cambios en la función de costos.

2.4.1.1. Estrategias de balanceo de clases

Una de las técnicas de optimización que se utilizarán es la aplicación de estrategias de balanceo de clases. Se aplicará *down-sampling*, *up-sampling* y *ROSE* para el algoritmo *rf*.

	Costo en test	Accuracy	Precision	Recall
sin estrategia	-0,6525	0,984	0,689	0,939
down-sampling	0,6805	0,848	0,178	1,000
up-sampling	-0,5712	0,979	0,630	0,879
ROSE	0,7508	0,841	0,172	1,000

Tabla 3. Resultados de los modelos con *rf* y estrategias de balanceo de clases

En todas las métricas evaluadas, el mejor modelo se obtiene sin la aplicación de estrategias de balanceo. Ahora bien, resulta interesante comentar los resultados obtenidos.

Si bien todos han obtenido buenos valores de *recall*, es decir, todos han logrado identificar como "Falla" a más del 87% de las mismas, los valores de *precision* y *cost* no indican que sean buenos modelos.

En particular, *down-sampling* y *ROSE* obtienen valores de *cost* positivos, lo que se puede leer como que le están agregando dicho costo a la empresa, en vez de permitir un ahorro. Al obtener una *recall* de 1, se podría considerar que estos modelos son buenos, ya que logran identificar a todas las fallas como tales. Sin embargo, dada la baja *precision* que obtienen, se puede suponer que están catalogando como "Falla" a muchas observaciones que no lo son. Esto puede apreciarse mejor en las matrices de confusión en el apartado del anexo[6].

Se confirma que estos modelos catalogan a muchas observaciones como "Falla" incorrectamente.

Respecto a la estrategia de *up-sampling*, los valores obtenidos son buenos. Sin embargo, no logran superar al algoritmo sin estrategia.

2.4.1.2. Cambios en el vector de umbrales

La probabilidad de cada observación de ser considerada como "Falla" varía en función del vector de umbrales. Al bajar el umbral, más observaciones tenderán a ser clasificadas como "Falla". Análogamente, al subirlo, menos observaciones tenderán a ser clasificadas como "Falla".

El vector de umbrales utilizado originalmente es seq(0.25,0.30,by=0.01). Se explorará que resultados se obtienen utilizando los vectores seq(0.10,0.20,by=0.01) y seq(0.30,0.60,by=0.01).

[6] Ver Anexo 5 – Matrices de confusión de la optimización de resultados

Los valores obtenidos para el algoritmo *rf* son los siguientes:

	Umbral	Costo en test	Accuracy	Precision	Recall
seq(0.25,0.30,by=0.01)	0,2840	-0,6525	0,984	0,689	0,939
seq(0.10,0.20,by=0.01)	0,1980	-0,5923	0,978	0,608	0,939
seq(0.30,0.60,by=0.01)	0,3740	-0,7172	0,992	0,857	0,909

Tabla 4. Resultados de los modelos con *rf* y cambios en el vector de umbrales

Al bajar el umbral hay más observaciones catalogadas como "Falla". Sin embargo, esto no ha provocado que más observaciones sean catalogadas correctamente, si no que existen más observaciones que no fallan catalogadas como que sí. Esto puede apreciarse en las matrices de confusión[7], así como también en la disminución del ahorro en *cost* y al obtener una *precision* más baja.

Por otro lado, al subir el umbral, menos observaciones que fallan han sido catalogadas como tal, provocando una disminución en la *recall*. No obstante, este modelo es el que ha obtenido **menor costo en *test*** en comparación con todos los modelos generados hasta el momento (**-0,71**). También es de los que ha obtenido una de las mejores *precision*.

Este modelo logra identificar a más del 90% de las fallas, siendo también de los modelos que tiene menos errores en términos de categorizar observaciones "Ok" como "Falla".

Los valores obtenidos para el algoritmo *xgbTree* son los siguientes:

	Umbral	Costo en test	Accuracy	Precision	Recall
seq(0.25,0.30,by=0.01)	0,2820	-0,5702	0,982	0,692	0,818
seq(0.10,0.20,by=0.01)	0,1880	-0,5256	0,976	0,596	0,848
seq(0.30,0.60,by=0.01)	0,4240	-0,6349	0,990	0,897	0,788

Tabla 5. Resultados de los modelos con *xgbTree* y cambios en el vector de umbrales

[7] Ver Anexo 5 – Matrices de confusión de la optimización de resultados

Sucede de manera similar que con *rf*. Al bajar el umbral más observaciones son categorizadas como “Falla”, aunque solo una observación más de “Falla” ha sido categorizada correctamente. Esto provoca que aumente la *recall* y disminuya la *precision*.

Por el contrario, al subir el umbral, menos observaciones son categorizadas como “Falla”, aunque, en su mayoría, las observaciones que no logran pasar este umbral son aquellas que estaban siendo asignadas incorrectamente. De este modo, este sería el modelo que obtuvo menor costo y mayor *precision* de los tres.

En conclusión, en ambos algoritmos se aprecia que al bajar el umbral existen más observaciones clasificadas como “Falla”. Sin embargo, la cantidad de fallas correctamente predichas no han aumentado, sino que ha aumentado la cantidad que no fallan y son clasificadas como que sí. A su vez, al aumentar el umbral, la cantidad de fallas correctamente predichas no disminuye significativamente, mientras que logra clasificar mejor a aquellas que no dan falla.

2.4.1.3. Cambios en la matriz de costos

La función de costos, al ser la métrica seleccionada para optimizar, influye directamente en la manera en la que el algoritmo clasifica las observaciones. De este modo, se decide explorar la devolución del algoritmo al modificar esta función.

Es pertinente mencionar que en este problema de clasificación se ha definido que:

- La clase positiva indica que la máquina ha fallado (Target = “Falla”)
- La clase negativa indica que la máquina no ha fallado (Target = “Ok”)

Entonces, originalmente los valores eran los siguientes:

- *false positive*: 10
- *true positive*: -25,5

Se explora con los siguientes cambios:

- Opción 1:
  - *false positive*: 1
  - *true positive*: -1
- Opción 2:
  - *false positive*: 50
  - *true positive*: -25,5
- Opción 3:
  - *false positive*: 50
  - *true positive*: -90
- Opción 4
  - *false positive*: 20
  - *true positive*: -1000

Al colocar valores más bajos en *true positive* se recompensa en mayor medida al algoritmo al acertar si una observación falla. Al incrementar el peso de *false positive* sucede lo contrario, es decir, se penaliza aún más al algoritmo al predecir una falla cuando no es cierto.

Los resultados obtenidos para el algoritmo *rf* son los siguientes:

	Accuracy	Precision	Recall
FP=10, TP=-25.5	0,984	0,689	0,939
FP=1, TP=-1	0,984	0,689	0,939
FP=50, TP=-25,5	0,985	0,705	0,939
FP=50, TP=-90	0,984	0,689	0,939
FP=20, TP=-1000	0,984	0,689	0,939

Tabla 6. Resultados de los modelos con *rf* y cambios en la función de costos

Los cambios en la función de costos no han generado cambios significativos al momento de realizar la predicción. El que ha obtenido mejores resultados es aquel que utiliza la función de costos con *false positive* = 50 y *true positive* = -25,5.

Los resultados obtenidos para el algoritmo *xgbTree* son los siguientes:

	Accuracy	Precision	Recall
FP=10, TP=-25.5	0,982	0,692	0,818
FP=1, TP=-1	0,982	0,692	0,818
FP=50, TP=-25,5	0,985	0,705	0,939
FP=50, TP=-90	0,982	0,692	0,818
FP=20, TP=-1000	0,982	0,692	0,818

Tabla 7. Resultados de los modelos con *xgbTree* y cambios en la función de costos

Al igual que con *rf*, no se aprecian grandes cambios al variar la matriz de pesos. Sin embargo, nuevamente se concluye que el modelo que obtiene mejores resultados es aquel que utiliza *false positive* = 50 y *true positive* = -25,5.

En suma, se concluye que el modelo que utiliza una función de costos que recompensa al algoritmo al estimar correctamente una falla, y penaliza en mayor medida al decir que existirá una falla cuando no es cierto, es aquel que obtiene mejores resultados. En otras palabras, en términos absolutos, en el caso en el que el valor *de false positive* es mayor que el de *true positive*, se logra obtener un mejor modelo.

2.4.1.4. Conclusiones de la optimización

En el apartado de optimización de resultados se han aplicado estrategias de balanceo, cambios en el vector de umbrales y cambios en la función de costos. A partir de ello, se concluye que la aplicación de técnicas de balanceo de clases no ha obtenido una mejora de resultados con respecto a los obtenidos sin ellas. Sin embargo, al aplicar un vector de umbrales con valores más elevados se ha logrado mejorar la bondad del modelo. Sucede lo mismo con el cambio en la función de costos, donde se penaliza al predecir incorrectamente en mayor medida que lo que se recompensa al predecir correctamente.

De este modo, se decide volver a correr los algoritmos *rf* y *xgbTree* con el vector de umbrales seq(0.30,0.60,by=0.01) y función de costos con *false positive* = 50 y *true positive* = -25,5. Los resultados obtenidos con la combinación del vector de umbrales y función de costos descritas anteriormente no han superado los valores anteriores.

En conclusión el mejor modelo es *rf* con técnica de *cross validation* de 5 *folds*, vector de umbrales seq(0.30,0.60,by=0.01) y función de costos *false positive* = 10 y *true positive* = -25,5. Obteniendo, a partir de un umbral de 0,37, un costo en test de -0,71, *accuracy* de 0,99, *precision* de 0,86 y *recall* de 0,90. El hiperparámetro de este modelo es *mtry*, que refiere al número de predictores seleccionados aleatoriamente en cada árbol. En este caso, el valor seleccionado es 6. El bosque aleatorio tiene 500 árboles, valor seteado por *default*.

3. Explicabilidad para el caso: Detección de fallas en datos tabulares

3.1. Explicabilidad global

Para la explicabilidad del caso, se parte de la base de que los tres mejores resultados obtenidos fueron con los algoritmos: *rf*, *xgbTree* y *ranger*. Se trabajará con este último debido a la dificultad de aplicación durante el desarrollo del código para los otros dos algoritmos, considerando que la bondad del modelo es adecuada.

Se utilizará la librería *fastshap* para R[8] [15], [22], [23], la cual calculará los *Shapley Values* de forma rápida en comparación a otras implementaciones.

Utilizando una muestra representativa de datos se obtienen los siguientes gráficos.

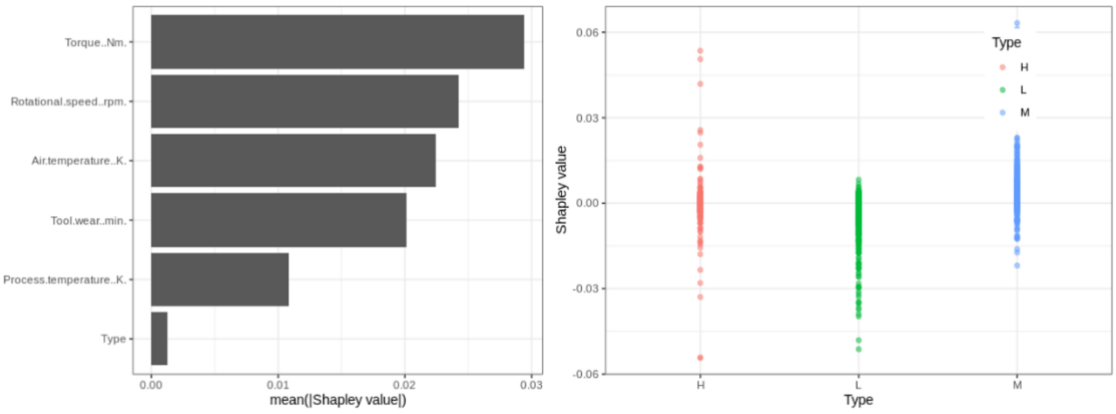


Figura 2. Explicabilidad global para Detección de fallas en datos tabulares

[8] Ver Anexo 7 – Fastshap para R

En el gráfico de la izquierda se observa el *Shapley Value* promedio de cada variable de predicción para el conjunto de datos sobre el que fueron calculados. Esto indicaría que, de forma general, la variable *Torque* es la que tiene más influencia o importancia en las predicciones de la variable *Target*. La segunda característica en términos de contribución es *Rotational Speed*, luego *Air Temperature* y así sucesivamente.

Por otra parte, el gráfico de la derecha, llamado *Dependence Plot*, sirve para observar la distribución de los *Shapley Values* del *dataset* segregados según una variable en particular. En este caso, a modo de ejemplo, la variable elegida fue *Type*.

Si se observa el primer gráfico, este indica que la variable *Type*, es aquella que tiene un *Shapley Value* promedio más cercano a cero. A partir del nuevo gráfico - *Dependence Plot* - podría decirse que los *Shapley Values* de la variable *Type*, se encuentran concentrados alrededor de 0 para productos con *Type* H (calidad alta), pero concentrados alrededor de *Shap Values* con valores absolutos un poco más altos para el caso de *Type* L (calidad baja) o *Type* M (calidad media).

Pensando en el problema en cuestión, el cual refiere a una máquina sobre la cual se desea entender qué variables son aquellas que tendrán mayor incidencia en la generación o no de una falla, parece tener sentido - a priori - que el *Torque* o la *Rotational Speed* puedan definir o ser variables críticas para entender si la máquina fallará o no. Por ejemplo, estas variables podrían indicar el desgaste de alguna pieza o algún problema en un motor.

Por otra parte, parece tener sentido que productos de alta calidad estén más alejados de indicar una próxima falla que aquellos que son de calidad media o baja, siempre y cuando la calidad del producto dependa exclusivamente de las condiciones operativas y no sea un seteo a elección.

Independientemente de cual sea el caso, se busca demostrar cómo estas herramientas de explicabilidad globales comienzan a agregar aún más sentido a la predicción. A partir de ellas se pueden generar conclusiones o premisas que expliquen el por qué de los resultados del algoritmo, no solo denotando transparencia sino permitiendo el debate y la profundidad conceptual para un mayor entendimiento de la predicción y del problema.

3.2. Explicabilidad local

Utilizando las mismas herramientas de código[9] que para el caso de explicabilidad global, es posible profundizar sobre un caso puntual. En este sentido, se buscará analizar cuáles son los *features* de mayor influencia para una predicción en particular, identificada por el *Product ID*.

Se analizarán dos casos puntuales: L50199 que presenta falla y M14860 que no falla.

Para el caso ID L50199 se observan a continuación los *Shapley Values* obtenidos.

	feature	shapley.value
Type	Type=L	-0.02795515
Air.temperature..K.	Air.temperature..K.=300.5	0.01567622
Process.temperature..K.	Process.temperature..K.=309.8	-0.01936854
Rotational.speed..rpm.	Rotational.speed..rpm.=1379	-0.17187808
Torque..Nm.	Torque..Nm.=54.2	-0.21170864
Tool.wear..min.	Tool.wear..min.=207	-0.39449286

Figura 3. *Shapley Values* y valores de *features* del caso L50199

El algoritmo *ranger* aplicado para la generación del modelo de explicabilidad devuelve la probabilidad de que un punto de observación no presente una Falla. Aquellas variables que influyen negativamente son las que aumentan la probabilidad de que dicha observación falle. Del mismo modo, aquellas que influyen de manera positiva son las que disminuyen la probabilidad de que la observación falle.

[9] Ver Anexo 7 – Fastshap para R

Ahora bien, a partir de la función autoplot, es posible graficar al igual que se hizo para la explicabilidad global. Puede verse que *Tool Wear* es la variable que más incidencia tuvo en el resultado y lo hizo de forma negativa, es decir generando que la probabilidad de fallar aumente. Por otro lado, *Air Temperature* es la variable que menos incidencia tuvo en el resultado, contribuyendo de forma positiva.

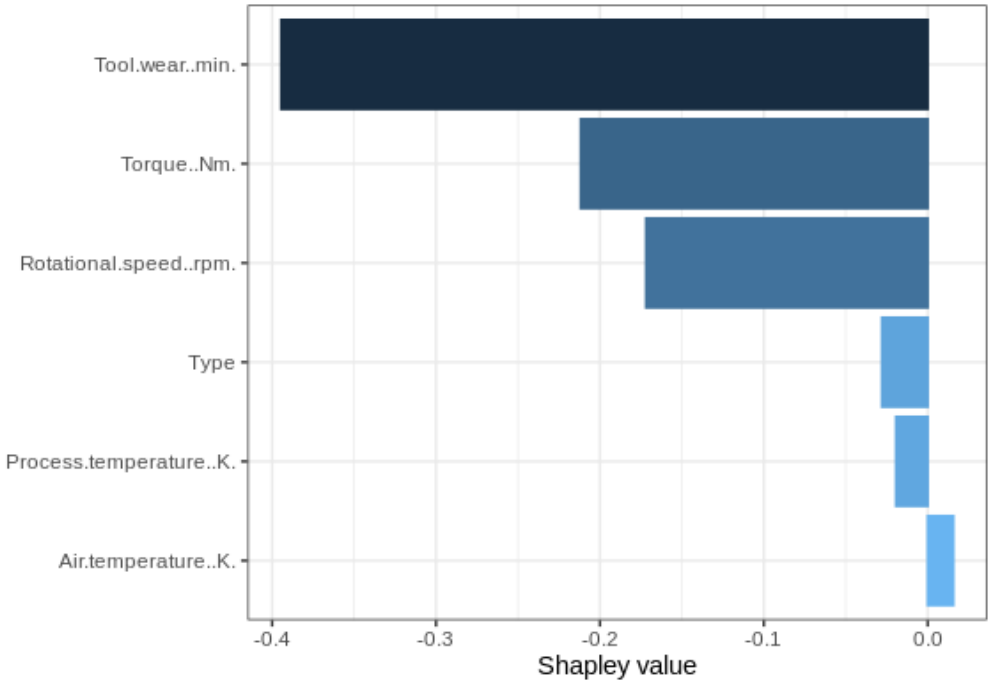


Figura 4. Gráfico de los *Shapley Values* del caso L50199

Resulta interesante comparar este caso de explicabilidad puntual, con la explicabilidad global calculada anteriormente.

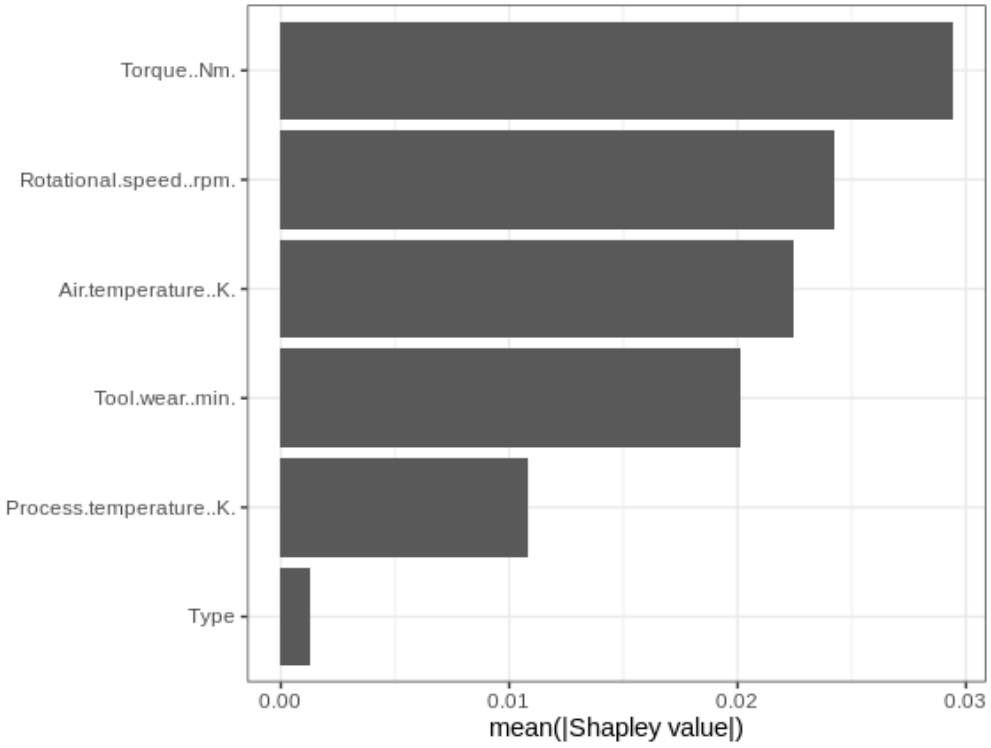


Figura 5. Gráfico de los *Shapley Values* globales

Puede apreciarse, como una de las grandes diferencias, que en la explicabilidad global la variable *Air Temperature* es la tercera más influyente, mientras que para el caso puntual es la menos influyente. Es decir, la incidencia de los *features* en la predicción, podría llegar a ser sustancialmente diferente en algunos casos y estarse perdiendo información de no profundizar en casos puntuales.

La explicabilidad global resulta una herramienta valiosa para abordar el problema desde una mirada macro, pero como en la mayoría de los problemas, únicamente una mirada macro no es suficiente para comprender algunas particularidades.

A modo de ejemplo, para el caso del *Product ID* L50199, el valor de *Tool Wear* se aleja mucho de la media de todas las observaciones. Por este motivo, es que esta variable podría cobrar importancia al momento de generar una predicción para este caso puntual, siendo la característica que mayor incidencia tiene, mientras que en el global es la cuarta. En contraposición, en relación a *Air Temperature*, el valor del caso analizado es 300,5 K, mientras que el valor promedio de dicha variable es 300,0 K. Siendo valores similares, esta característica podría no haber tenido tanta influencia en la predicción del punto de observación en cuestión. Asimismo, es el único *feature* que tiene una contribución positiva, bajando la probabilidad de que este punto falle.

Se realiza el mismo ejercicio para el punto de observación con el *product ID* M14860.

	feature	shapley.value
Type	Type=M	0.000160914
Air.temperature..K.	Air.temperature..K.=298.1	0.015259996
Process.temperature..K.	Process.temperature..K.=308.6	-0.003759060
Rotational.speed..rpm.	Rotational.speed..rpm.=1551	-0.001932613
Torque..Nm.	Torque..Nm.=42.8	0.019251406
Tool.wear..min.	Tool.wear..min.=0	0.003984390

Figura 6. *Shapley Values* y valores de *features* del caso M14860

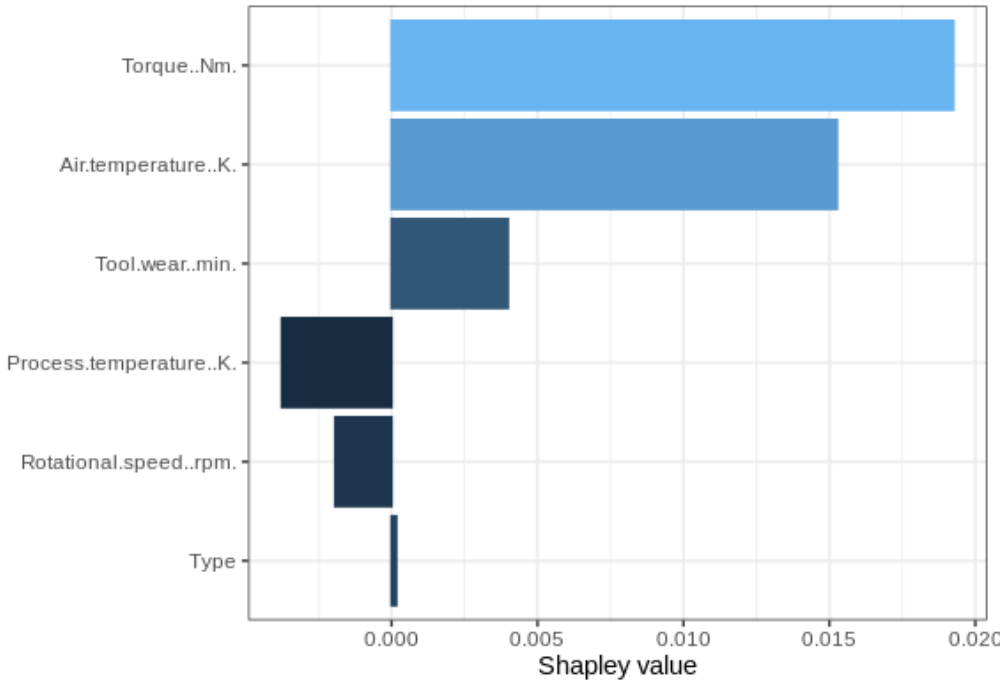


Figura 7. Gráfico de los *Shapley Values* del caso M14860

Del gráfico y de los *Shapley Values* obtenidos, se desprende que *Torque* y *Air Temperature* son los *features* que más han influido en la predicción de esta observación. Estos dos, al igual que *Tool Wear* y *Type*, han obtenido valores de Shap positivos, es decir, inclinan la predicción hacía la clase "Ok". Sin embargo, los valores de *Process Temperature* y *Rotational Speed* son negativos, inclinando la predicción hacia la clase "Falla".

4. Comparación con otro enfoque de explicabilidad

El *paper* [17] realiza una comparativa sobre la *performance* de un modelo de explicabilidad y una interfaz explicativa, ambos entrenados sobre el set de datos tabular. Se profundizará sobre el mismo y se lo comparará con los resultados obtenidos en el presente trabajo.

El *paper* presenta la siguiente estructura:

- Presentación de resultados del modelo *bagged trees ensemble classifer*
- Estimación de la importancia de los predictores para *bagged trees ensemble classifier*
- Desarrollo e implementación de un modelo de explicabilidad
- Desarrollo e implementación de una interfaz explicativa

4.1. Bagged Trees Ensemble Classifier

De forma de seguir el hilo conductor del trabajo realizado por Stefan Matzka [17], resulta importante comentar brevemente qué pasos previos a los modelos de explicabilidad se realizaron durante su estudio.

En primera instancia, se entrenó un modelo *bagged trees ensemble classifier* obteniéndose resultados considerados satisfactorios para el propósito del *paper*.

El artículo presenta un modelo con una *recall* de 70,8%, mientras que el mejor modelo desarrollado para abordar la clasificación binaria en el presente documento muestra una *recall* del 90,9%. En cuanto a la *precision*, el modelo del artículo presenta una tasa del 86,7%, mientras que la obtenida para *rf* es del 85,7%. A su vez, el modelo desarrollado con el algoritmo *ranger* presenta una *accuracy* similar a la del artículo. Con relación a la *precision*, la obtenida por *ranger* es significativamente menor a la del mismo, pudiéndose investigar su mejora a través de la variación del umbral. Repecto a la *recall*, es el que presenta mejor valor de los tres.

	Accuracy	Precision	Recall
bagged trees ensemble	0,983	0,867	0,708
rf	0,992	0,857	0,909
ranger	0,983	0,674	0,939

Tabla 8. Resultados obtenidos con el modelo desarrollado en el *paper* vs. *rf* y *ranger* desarrollados en el presente trabajo

4.2. Estimación de la importancia de los predictores

Con el fin de comenzar a explicar el modelo implementado - *bagged trees ensemble classifier* -, el *paper* [17] detalla el resultado de una estimación de la importancia de cada *feature*, con base en la métrica y estimación descrita e implementada en *Mathworks, "Estimates of predictor importance for classification ensemble of decision trees"* [24], el cual utiliza *Statistics and Machine Learning Toolbox*. A continuación los resultados obtenidos a partir de este análisis.

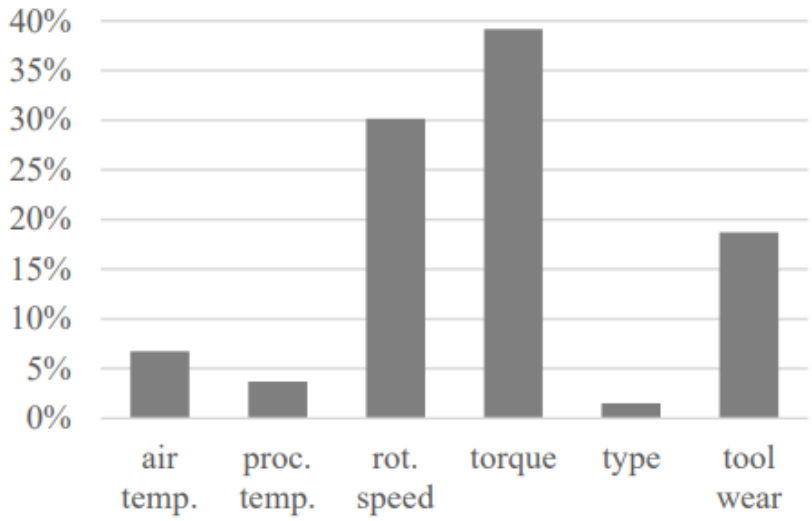


Figura 8. Gráfico de la importancia relativa de cada *feature* utilizado en el *bagged trees ensemble.* Descrito e implementado en [24]. Fig. 2 de [17].

Entonces, utilizando *Statistics and Machine Learning Toolbox*, se estima que las variables *Torque* y *Rotational Speed* dominan en influencia el proceso de clasificación de falla, mientras que ambas temperaturas (*Air Temperature* y *Process Temperature*) y la variable *Type* no parecen tener demasiada importancia. Esto sería a priori, un análisis de explicabilidad global del modelo.

Si se compara este resultado con el obtenido en el desarrollo del presente trabajo con *Shapley Values* (ver Figura 5), se observa que en ambos casos *Torque* y *Rotational Speed* son las variables más influyentes para la predicción. Por otra parte, con el método de *Shapley Values*, *Air Temperature* también tiene una influencia significativa a diferencia de lo obtenido en el caso de estudio del *paper*.

4.3. Modelo de explicabilidad

A pesar de la deficiencia de explicaciones conceptuales sobre la forma o camino a través de la cual un modelo realiza una determinada predicción, algunos métodos de aprendizaje automático son fáciles de interpretar al ojo humano, como es el caso de árboles de decisión.

Como modelo de explicabilidad, el *paper* [17] entrena un conjunto de quince árboles de decisión, cada uno limitado por un máximo de cuatro nodos para una fácil interpretación, es decir, cada árbol se entrena utilizando como máximo cuatro de las seis características o *features* disponibles.

A modo de ejemplo, el siguiente árbol fue entrenado como parte del modelo de explicabilidad con los *features*: *Torque*, *Rotational Speed* y *Tool Wear*.

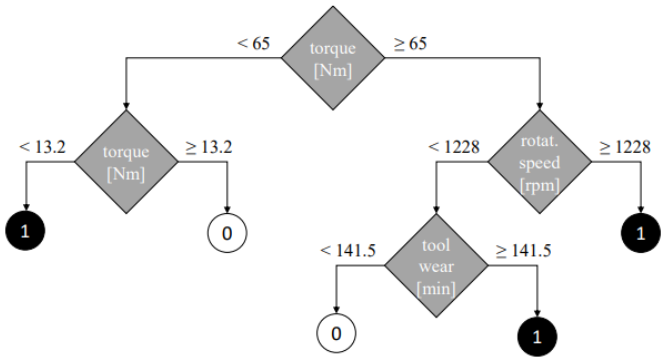


Figura 9. Árbol de decisión (#1). Fig. 1 de [17]

El *paper* enfatiza en la idea de que este árbol de decisión puede ser fácilmente comprensible para personal de mantenimiento entrenado. En otras palabras, una persona que comprenda del campo estudiado puede leerlo como "valores pequeños de torque, por debajo de 13,2 Nm predicen una falla en la máquina. A su vez, altos valores de torque, por encima de 65 Nm, cuando son combinados con una velocidad rotacional superior a 1228 rpm o un valor de desgaste de la herramienta mayor a 141,5 min también generan falla".

Este caso funciona como ejemplo de la importancia de la explicabilidad dentro del marco conceptual del problema, no solo como aportador de transparencia del modelo, sino como herramienta de generación de acciones en busca de soluciones. El personal capacitado que cuente con esta información podrá mejorar la toma de decisiones en relación al mantenimiento de las máquinas.

Por otra parte, cabe reafirmar a partir de este ejemplo, que cada predicción y su forma de alcanzarla puede haber seguido un camino lógico que contenga un umbral sobre una determinada variable como regla de decisión. En este caso, para valores de *Torque* menores a 65 nM, las otras dos variables presentes en el árbol no inciden significativamente en el resultado final. Ahora, si el *Torque* tomara valores mayores a 65 nM, las otras dos variables sí comienzan a ser definitorias para la predicción o no de una falla.

Entonces, si se estuviera trabajando únicamente con la explicabilidad global de un modelo sin una regla de decisión, o sin la posibilidad de profundizar sobre un caso puntual (como se ha debatido en el punto 3.2), el modelo explicaría la influencia de *Torque* como medida promedio de todos los casos considerados, pero podrían quedar excluidos o camuflados los casos minoritarios definidos por el umbral, pudiéndose perder información importante. En otras palabras, el modelo de explicabilidad global arrojaría como variables con mayor incidencia a aquellas que han participado en la toma de decisiones de la mayoría de los casos. De este modo, si la mayoría de los casos presentaran valores de *Torque* menores a 65 Nm, se consideraría esta variable como la más influyente, presentando a *Rotational Speed* y *Tool Wear* como variables que han tenido poca incidencia. Por el contrario, si la mayoría de los casos presentarán valores de *Torque* menores a 65 Nm, probablemente se consideraría que las variables *Rotational Speed* y *Tool Wear* han contribuido de forma significativa a la predicción.

Volviendo al modelo de explicabilidad desarrollado por el *paper*, para generar los quince árboles de decisión, a cada árbol se le entregaron cuatro *features*. A partir de esto, en el entrenamiento del modelo hubo casos donde el mismo utilizó las cuatro variables otorgadas y en otros casos en donde prescindió de algunas de ellas. El siguiente cuadro explica este punto en profundidad, en donde 1 representa los *features* que fueron utilizados por el árbol de decisión, 0 cuando no fueron utilizados y el espacio en blanco indica que no fueron provistos al algoritmo.

tree num.	air temp.	proc. temp.	rot. speed	torq.	type	tool wear
1			1	1	0	1
2		0		1	0	1
3		0	1		1	1
4		0	1	1		0
5		0	1	1	0	
6	0			1	0	1
7	1		1		0	1
8	0		1	1		0
9	1		1	1	0	
10	0	0			0	0
11	0	0		1		1
12	0	0		1	1	
13	1	1	1			0
14	1	1	1		0	
15	1	1	1	1		
∑	5	3	10	10	2	6

Figura 10. *Features* utilizados en el entrenamiento de los árboles de decisión. Tabla II de [17]

De la tabla se desprende rápidamente que hay variables que son utilizadas siempre que están disponibles, por ejemplo *Rotational Speed*. Además, sumando las variables utilizadas columna a columna, se puede tener un estimado de la importancia de cada *feature*.

A continuación, se toma un subconjunto de 20 puntos de datos, detallados en Figura 11. Intentando abarcar una muestra que contenga todos los tipos de falla y tipos de producto disponibles, se eligen los datos aleatoriamente dentro de cada categoría.

UID	prodID	TWF	HDF	PWF	OSF	BTC
2672	M17531	1				0
3866	H33279	1				1
6341	H35754	1				0
8358	L55537	1				0
9019	L56198	1				0
3237	M18096		1			0
4079	H33492		1			1
4174	M19033		1			1
4327	L51506		1			1
4502	L51681		1			1
464	L47643			1		1
1493	M16352			1		1
3001	H32414			1		1
7537	L54716			1		1
8583	M23442			1		1
250	L47429					1
3020	L50199				1	1
5400	H34813				1	1
7592	M22451				1	1
9660	L56839				1	1

Figura 11. Puntos de datos utilzados para la evaluación del desempeño explicativo. Tabla IV de [17]

Las columnas describen un UID propio del *dataset*, el *product ID* referente a cada observación, el tipo de falla (con un 1 que denota el correspondiente a cada caso) y en la columna BTC cual fue la predicción por parte del *bagged trees ensemble classifier.*

Se entregan los puntos elegidos y detallados en la Figura 11 a los quince árboles de decisión entrenados como modelo de explicabilidad. Si al menos dos árboles de decisión devuelven un resultado positivo, es decir, predicen correctamente, se toma como modelo de explicabilidad el desarrollado a partir de esos árboles.

A partir de esto, se construye la Figura 12, en donde para cada observación elegida se detalla el modo de falla, cuál fue el resultado del *bagged trees ensemble classifier* (BTC), qué árboles predijeron bien el resultado (*Trees*) y la explicabilidad a partir de ellos (*Explanation*).

UID	Mode	BTC	Trees	Explanation	Scr
2672	TWF	0		none	n/a
3866	TWF	1		none	− −
6341	TWF	0		none	n/a
8358	TWF	0		none	n/a
9019	TWF	0		none	n/a
3237	HDF	0		none	n/a
4079	HDF	1		none	− −
4174	HDF	1		none	− −
4327	HDF	1	13-**15**	torque < 65 Nm rotSpeed < 1380 rpm airTemp < 301.5 K procTemp < 310.5 K	+ +
4502	HDF	1			+ +
464	PWF	1	1-**8**,9, 11-15	torque < 13.2 Nm	+
1493	PWF	1		none	− −
3001	PWF	1	1,2,4, 5,6,**8**, 9,11, 12,15	torque > 65 Nm rotSpeed > 1229 rpm	+ +
7537	PWF	1	1-5, 6-**8**, 9-15	torque < 13.2 Nm	+
8583	PWF	1	1,2, 4-6,**8**, 9,11, 12,15	torque > 65 Nm rotSpeed > 1229 rpm	+ +
250	OSF	1	2,6, **11**		+ +
3020	OSF	1	2,3, 6,11		+ +
5400	OSF	1	2,6, **11**	torque > 53.6 Nm tool wear > 194.5 min	+ +
7592	OSF	1	2,6, **11**		+ +
9660	OSF	1	2,3, 6,**11**		+ +

Figura 12. Explicación otorgada por los árboles de decisión para los datos seleccionados.

Tabla V de [17]

Se pueden encontrar llamativas detecciones como ser que el modo de falla TWF no logra ser explicado a partir de este modelo, así como tampoco tres casos del modo de falla HDF. Teniendo en cuenta estas y otras consideraciones, el *paper* define como explicaciones útiles a partir del modelo a 9 casos de los 20 puntos elegidos.

Resulta interesante comparar algunos de estos 20 puntos elegidos, con el modelo de explicabilidad local desarrollado en el presente trabajo.

Por ejemplo, a partir del modelo de explicabilidad del *paper*, para el *Product ID* L50199 (UID 3020), el cual fue profundizado secciones atrás, se alcanza la teoría de que hubo falla debido principalmente a dos variables: *Torque* y *Tool Wear*. Este resultado coincide con el calculado por los *Shapley Values* en el presente trabajo en donde estas dos variables, también alcanzan los valores absolutos más altos de Shapley (ver Figura 4).

A pesar de este resultado aparentemente concordante, los *Shapley Values* denotan que hay cierta importancia también en la variable *Rotational Speed*. Si se profundiza en cuáles árboles crearon el modelo del *paper*, se observa que ellos fueron los números: 2, 3, 6 y 11.

tree num.	air temp.	proc. temp.	rot. speed	torq.	type	tool wear
1			1	1	0	1
2		0		1	0	1
3		0	1		1	1
4		0	1	1		0
5		0	1	1	0	
6	0			1	0	1
7	1		1		0	1
8	0		1	1		0
9	1		1	1	0	
10	0	0			0	0
11	0	0		1		1
12	0	0		1	1	
13	1	1	1			0
14	1	1	1		0	
15	1	1	1	1		
∑	5	3	10	10	2	6

Figura 13. *Features* utilizados en el entrenamiento de los árboles de decisión que definen la explicabilidad del caso L50199. Tabla IV de [17]

Para estos cuatro árboles, la variable *Rotational Speed* fue entregada solo una vez, en el árbol 3, y utilizada por el mismo como regla de decisión. Esto podría denotar una de las deficiencias de este modelo de explicabilidad: con el fin de ser interpretable por el ser humano, la cantidad de nodos está limitada y por lo tanto el modelo no siempre cuenta con todas los features para tomar o explicar sus decisiones.

A modo de continuar profundizando en la comparación, se analizará un nuevo caso puntual.

Para el product ID L54716 (UID 7537), se obtienen los siguientes valores de Shapley.

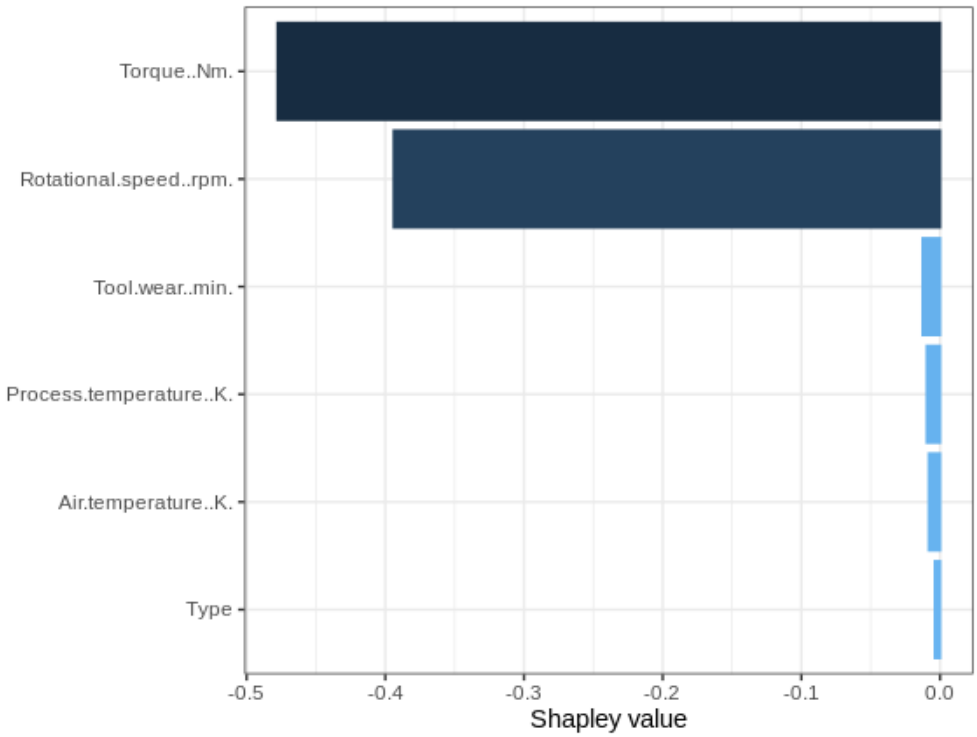


Figura 14. Gráfico de los *Shapley Values* del caso L54716

Este caso resulta interesante de ver porque según *Shapley Values* la incidencia de las variables en la predicción es muy marcada: *Torque* y *Rotational Speed* son las más influyentes.

Según el método de explicabilidad del paper la única variable clave es *Torque*. *Rotational Speed* no se encuentra detallada a modo de umbral en la columna *"Explanation"* de la Figura 12, a pesar de que para este análisis se utilizaron los quince árboles en donde *Rotational Speed* sí tuvo la mayor sumatoria, al igual que *Torque*.

tree num.	air temp.	proc. temp.	rot. speed	torq.	type	tool wear
1			1	1	0	1
2		0		1	0	1
3		0	1		1	1
4		0	1	1		0
5		0	1	1	0	
6	0			1	0	1
7	1		1		0	1
8	0		1	1		0
9	1		1	1	0	
10	0	0			0	0
11	0	0		1		1
12	0	0		1	1	
13	1	1	1			0
14	1	1	1		0	
15	1	1	1	1		
∑	5	3	10	10	2	6

Figura 15. Sumatoria de las veces que fueron utilizadas las variables *Rotational* Speed y Torque por los 15 árboles de decisión. Tabla IV de [17]

4.4. Interfaz explicativa

Hasta ahora se comentó sobre el desarrollo del primer enfoque de explicabilidad en [17] para el set de datos en cuestión. El nuevo enfoque en [17], refiere al análisis de la desviación normalizada de cada *feature* como interfaz explicativa y se detalla a continuación.

Para esto, cada *feature* es normalizado a una esperanza de 0 y una desviación estándar de 1 (segunda fila). En la siguiente tabla se muestra esto para la primera instancia de cada modo de falla en comparación al valor absoluto de cada *feature* (primera fila) y se marca en negrita las dos máximas desviaciones absolutas para cada *feature*.

UID	Mode	air temp.	proc. temp.	rot. speed	torq.	tool wear
2672	TWF	299,7	309,3	1399	41,9	221
		-0,15	-0,47	**-0,78**	0,19	**1,78**
3237	HDF	300,8	309,4	1342	62,4	113
		0,40	-0,41	**-1,10**	**2,25**	0,08
464	PWF	297,4	308,7	2874	4,2	118
		-1,30	-0,88	**7,45**	**-3,59**	0,16
250	OSF	298	308,3	1405	56,2	218
		-1,00	-1,15	-0,75	**1,63**	**1,73**

Figura 16. Valores de los *features* y sus desviaciones estándar normalizadas. Tabla VI de [17]

Entonces, la explicación estará basada en las máximas desviaciones. Por ejemplo, para el caso UID 2672, la explicación según el *paper* puede leerse como “La clasificación ‘falla’ se debe a un *Tool Wear* alto de 221 min y a una baja *Rotational Speed* de 1399 rpm”.

En definitiva, con este nuevo método, trabajando sobre los 20 datos escogidos, se desprenden las siguientes *explanations*.

UID	Mode	BTC	Explanation	Scr
2672	TWF	0	high tool wear of 221 mins low rot. speed of 1399 rpm	+
3866	TWF	1	high tool wear of 228 mins high air temp. of 302.6 K	+
6341	TWF	0	high tool wear of 210 mins low rot. speed of 1397 rpm	+
8358	TWF	0	high tool wear of 210 mins low rot. speed of 1397 rpm	+
9019	TWF	0	high tool wear of 217 mins low air temp. of 297.3 K	+
3237	HDF	0	high torque of 62.8 Nm low rot. speed of 1342 rpm	+
4079	HDF	1	high torque of 62.8 Nm low rot. speed of 1294 rpm	+
4174	HDF	1	high air temp. of 302.2 K low rot. speed of 1346 rpm	+ +
4327	HDF	1	high torque of 55.8 Nm low rot. speed of 1362 rpm	+
4502	HDF	1	high torque of 54.0 Nm low rot. speed of 1307 rpm	+ +
464	PWF	1	high rot. speed of 2874 rpm low torque of 4.2 Nm	+
1493	PWF	1	high torque of 58.5 Nm low air temp. of 298 K	+
3001	PWF	1	high torque of 72.8 Nm low rot. speed of 1324 rpm	+
7537	PWF	1	high rot. speed of 2579 rpm low torque of 12.5 Nm	+
8583	PWF	1	high torque of 72.8 Nm low proc. temp. of 308.1 K	+
250	OSF	1	high tool wear of 218 mins high torque of 56.2 Nm	+ +
3020	OSF	1	high tool wear of 207 mins high torque of 54.2 Nm	+ +
5400	OSF	1	high tool wear of 218 mins high air temp. of 302.8 K	+
7592	OSF	1	high torque of 61.3 Nm high tool wear of 202 mins	+ +
9660	OSF	1	high torque of 61.9 Nm high tool wear of 216 mins	+ +

Figura 17. Explicación otorgada por la desviación normalizada de los *features* para los datos seleccionados. Tabla VII de [17]

Profundizando nuevamente en los puntos *Product ID* L50199 (UID 3020) y *Product ID* L54716 (UID 7537), se observa lo siguiente:

Para el *Product ID* L50199 la interfaz explicativa considera definitorios los *features Tool Wear* y *Torque* al igual que el modelo de explicabilidad desarrollado inicialmente en el *paper*. Sin embargo, no considera la variable *Rotational Speed* como significativa, en contraposición con el método de *Shapley Values.*

A diferencia del primer modelo de explicabilidad desarrollado en el *paper*, donde para el *Product ID* L54716 únicamente considera como variable explicativa el *Torque*, la interfaz explicativa considera como definitorios los *features Rotational Speed* y *Torque*, al igual que los *Shapley Values* desarrollados en el presente trabajo.

4.5. Comentarios finales

En síntesis, el artículo [17], realiza una comparativa sobre la *performance* de un modelo de explicabilidad, creado a partir de 15 árboles de decisión, y una interfaz explicativa, utilizando la desviación normalizada de los *features*, ambos entrenados sobre el set de datos en cuestión.

El modelo de explicabilidad desarrollado en el *paper*, a través del método de árboles de decisión, es capaz de explicar de manera útil 9 de los 20 datos elegidos de forma aleatoria, mientras que el modelo de explicabilidad del presente trabajo con *Shapley Values* permite hacerlo no solo para datos específicos sino para todos aquellos que formen parte del *dataset*. A su vez, la interfaz explicativa también logra brindar una explicabilidad para todos los datos, aunque las mismas son consideradas por Matzka [17] como de menor calidad explicativa.

Por otra parte, el modelo de explicabilidad del *paper*, por su propia naturaleza, no siempre utiliza todos los *features* disponibles, a diferencia del desarrollado a partir de *Shapley Values*.

Tanto el modelo de explicabilidad como la interfaz explicativa del *paper*, además de definir variables críticas en el proceso de predicción, establecen rangos o umbrales como regla de decisión. Esto no se encuentra presente en el modelo basado en *Shapley Values*, pudiendo ser considerado una deficiencia. Sin embargo, es importante considerar que, existen herramientas complementarias para trabajar con los *Shapley Values*, como los *Parcial Dependence Plots*, que muestran la distribución de los valores de un *feature* en particular, de forma de tener una imagen más completa de la influencia de la misma en el modelo.

El modelo desarrollado en el presente trabajo es capaz de utilizarse en cualquier set de datos, no solo en el escogido en este caso. Es decir, el modelo de explicabilidad a partir de *Shapley Values* es agnóstico, a diferencia de los modelos desarrollados en el *paper* que son específicos. Este punto es considerado como un posible trabajo futuro por el autor del artículo, el cual expresa que en el futuro intentará aplicar *local interpretable model-agnostic explanations*.

Otra de las ventajas del método *Shapley Values* por sobre el desarrollado en el *paper* como método de explicabilidad – a partir de 15 árboles de decisión –, es que, en el primero, a cada caso puntual le corresponde una explicación individual, mientras que en el segundo, varios casos puntuales comparten la misma explicación.

5. Caso: Remaining Useful Life en series temporales

En el próximo apartado, se abordará el tema de la explicabilidad de un modelo aplicado a un conjunto de datos representado como serie temporal. Koen Peters, un usuario de la plataforma *Towards Data Science* [25], ha creado un modelo de predicción de RUL utilizando una red neuronal LSTM a partir del conjunto de datos que será descrito a continuación. Este modelo se considerará una caja negra, y se trabajará específicamente en el desarrollo de un modelo de explicabilidad para el mismo.

5.1. Descripción del conjunto de datos

El *Prognostics Center of Excellence Data Set Repository* de la *NASA* [26] es un repositorio de datos que se centra exclusivamente en conjuntos de datos de pronóstico. En su mayoría, son datos de series temporales que van desde un estado nominal previo hasta un estado fallido.

En este repositorio se encuentra el *Turbofan Engine Degradation Simulation*. Este es una simulación de la degradación del motor, llevado a cabo utilizando el sistema de simulación de aeropropulsión modular comercial (C-MAPSS). A partir de esta simulación se generan cuatro conjuntos de datos bajo diferentes combinaciones de condiciones operativas y modos de falla.

Los conjuntos de datos han sido publicados en el 2008 y su popularidad y relevancia sigue vigente hasta el día de hoy. En el año 2022 se han publicado 343 artículos de investigación al respecto [27].

Turbofan Dataset presenta cuatro conjuntos de datos. En estos conjuntos, los motores funcionan correctamente al principio pero a medida que avanza el tiempo desarrollan fallas. Los datos son provistos ya separados en *train* y *test*. En los conjuntos de entrenamiento, los motores funcionan hasta el fallo, mientras que en los conjuntos de prueba la serie temporal finaliza en algún momento antes del fallo. De este modo, el objetivo es predecir la RUL de cada motor.

Como se ha mencionado anteriormente, *Remaining Useful Life* (RUL) o vida útil restante es el período de tiempo que se espera que una máquina opere antes de requerir reparación o reemplazo. Al tomar en cuenta la RUL, se puede programar el mantenimiento, optimizar la eficiencia operativa y evitar el tiempo de inactividad no planificado.

Para cada conjunto de datos se cuenta con 3 archivos distintos:

- Train: conjunto de datos de entrenamiento
- Test: conjunto de datos de prueba
- RUL: RUL del conjunto de datos de prueba, variable a predecir

Los conjuntos de datos provistos son los siguientes:

- FD001: 1 condición operativa y 1 modo de falla
- FD002: 6 condiciones operativas y 1 modo de falla
- FD003: 1 condición operativa y 2 modos de falla
- FD004: 6 condiciones operativas y 2 modos de falla

Koen Peters desarrolla un LSTM [25] para el conjunto de datos FD004, es decir, el set de datos más complejo de los cuatro.

El *dataset* FD004 consta de 26 variables:

- Número de unidad de motor: unit_nr
- Tiempo en ciclos: time_cycles
- 3 variables de configuración operativas: setting_1, setting_2 y setting_3
- 21 sensores: s_n°sensor

5.2. Modelo LSTM para predecir RUL

LSTM es un tipo de red neuronal recurrente (RNN). Las RNN son un tipo de red especializada en el procesamiento de datos secuenciales, que comparte características entre nodos y permite la utilización de salidas anteriores como entradas a nodos.

Para el desarrollo del modelo, se le realizan ciertas transformaciones [28] al conjunto de datos:

- Calcula la RUL como la diferencia entre el máximo de ciclos de un motor y la cantidad de ciclos que ya han ocurrido agregándose como una nueva columna.
- Define las condiciones de operación en función de las variables de configuración operativas.
- Realiza una estandarización de los valores de los sensores por condición operativa. En otras palabras, agrupa todos los registros de una sola condición operativa y aplica un escalador estándar, lo que permite comprender la magnitud de las diferencias absolutas.
- Elimina algunos sensores, ya que no presentan cambios a medida que avanzan los ciclos, ergo no aportan valor para entrenar el modelo ni para generar la predicción. Se conservan los datos de 14 sensores.
- Aplica suavizamiento exponencial a los valores de los sensores.

En pos de preparar los datos para entrenar el modelo se crean dos funciones para la generación de datos de *train*. El objetivo es la generación de secuencias. La primera función está diseñada para recibir un dataframe conteniendo datos de un único motor, mientras que la segunda es una función que permite generar secuencias para varios motores.

El autor del artículo define un largo de secuencia de 30, indicando que las secuencias contendrán 30 valores. De este modo, cada una contiene valores de sensores de 30 ciclos de un mismo motor. Otra función será de generación de datos de *test*, la cual se utilizará en la creación de *test_array*, arreglo que contendrá una fila por motor, con secuencias de 30 ciclos con valores de 14 sensores.

Cabe aclarar que la red es entrenada con todos los datos disponibles en *train*. Esto es, todas las secuencias de largo 30 que son generadas por motor. Estas están contenidas en *train_array*. Ahora bien, al momento de predecir, se utilizará la última secuencia de 30 ciclos de cada motor, almacenadas en *test_array*.

5.3. Explicabilidad para LSTM

Una vez entrenado el modelo, se busca brindar explicabilidad al mismo. Para esto, se utilizará DeepExplainer[10] [29], [30], [31], [32], método que aproxima los valores de shap para modelos de aprendizaje profundo.

5.3.1. Explicabilidad global

Se considera importante desarrollar un modelo que sirva como explicabilidad global, es decir, que ofrezca una idea general de explicabilidad independientemente del motor.

Se define entonces la variable “muestra”, a la cual se le asignará el número de motores que se desea tomar en cuenta para la explicabilidad global. Esto definirá el tamaño del array *test_array*, el cual contiene la última secuencia de 30 tiempos para cada motor junto con el valor de cada sensor para esos tiempos.

Se calculan luego los *Shapley Values* para los motores seleccionados. Luego, se calcula el promedio de los *Shapley Values* de estos motores para cada tiempo. A partir de esto, considerando la totalidad de los motores (248)[11], se obtiene el siguiente gráfico.

[10] Ver Anexo 8 – SHAP para Python
[11] Ver Anexo 9 – Explicabilidad para LSTM

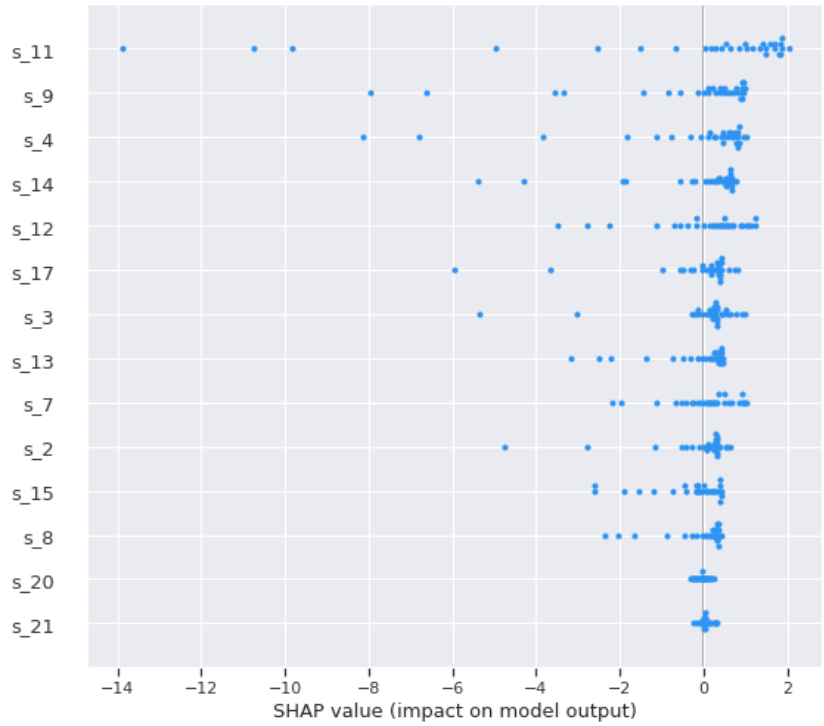


Figura 18. Explicabilidad global para LSTM promedio de los *Shapley Values* de los 248 motores. Cada punto corresponde a un instante de tiempo

El gráfico muestra para cada sensor, la distribución de los *Shapley Values* en los 30 instantes de tiempo. A su vez, ordena los sensores o *features* según su influencia general en la predicción. Se observa que el sensor 11 es el más influyente en el resultado, seguido por el sensor 9 y así sucesivamente.

Resulta importante agregar que, la variable “muestra” definirá la cantidad de motores a tener en cuenta para aproximar un resultado global y por lo tanto, generará valores más exactos cuanto más grande sea su valor, pero a su vez, su elección será un compromiso con el costo computacional que genere.

En el presente trabajo el número total de motores - 248 -, no genera un costo computacional alto, su tiempo de ejecución es menor a 2 minutos. De todas formas, se prueba con un número menor – 10 –, para observar el resultado.

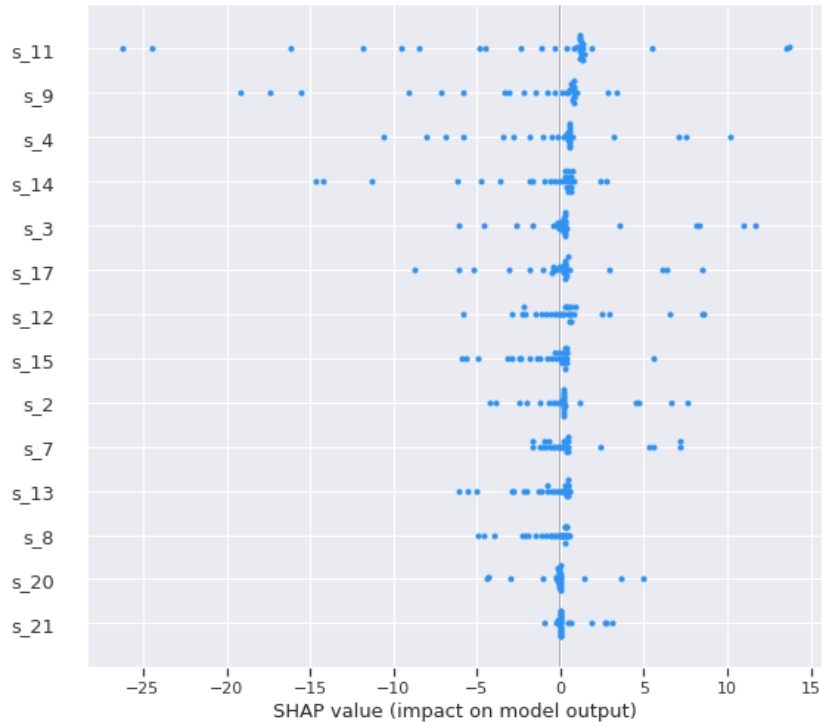


Figura 19. Explicabilidad global para LSTM promedio de los *Shapley Values* de 10 motores. Cada punto corresponde a un instante de tiempo

Como puede apreciarse, no se alcanza exactamente el mismo resultado. Con la totalidad de los motores, el sensor 12 es el quinto más importante, mientras que tomando como muestra solo 10 motores, es el séptimo. Sin embargo, los cuatros primeros sensores más influyentes han sido los mismos en los dos casos.

El costo computacional podría ser una limitación en un *dataset* que contemple mayor cantidad de datos.

Los gráficos anteriores muestran la explicabilidad por sensor, donde cada punto representa un instante de tiempo. Otra manera de analizar la explicabilidad global de los datos, puede ser, además de independizarse del motor, independizarse del tiempo. Entonces, se promedian los *Shapley Values* correspondientes a cada instante de tiempo. De esta manera se obtiene una matriz de 1x14 que únicamente contiene un valor de Shapley para cada sensor.

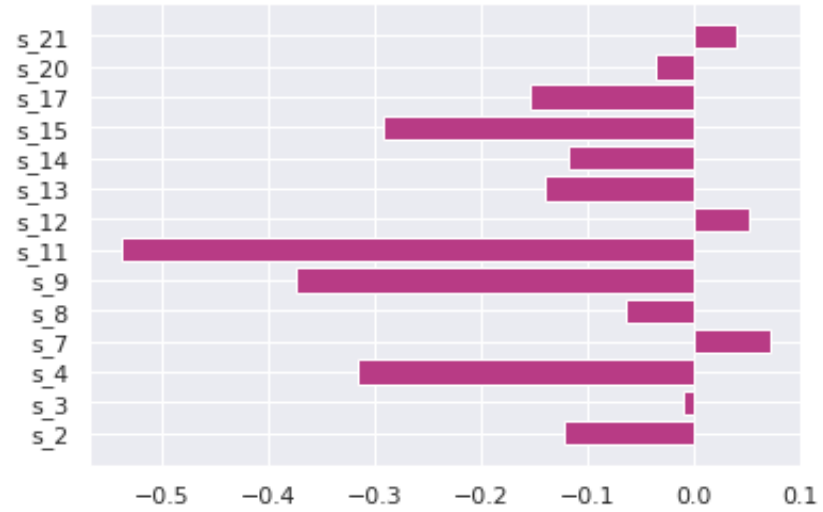


Figura 20. Explicabilidad global para LSTM promedio de los *Shapley* Values de los 248 motores y los 30 instantes de tiempo

De él se desprende que el sensor 11 es el más influyente en la predicción, seguido del sensor 9 y del sensor 4, los mismos sensores catalogados como más influyentes en los dos gráficos anteriores de explicabilidad global.

5.3.2. Explicabilidad local

La explicabilidad local en este caso será profundizar en la influencia de los *features* (sensores) para la predicción de la RUL en un motor en particular.

Para esto, se desarrolla un "buscador"[12], en donde se asigna el motor y para el cual específicamente se calcularán luego los *Shapley Values*. Este aspecto se entiende relevante para la utilización de las herramientas en un caso real, ya que no parecería muy útil entender qué variables fueron claves para el cálculo de la RUL por parte de un modelo, sin poder consultar o saber en qué motor influyeron. De hecho, los motores podrían tener diferentes características entre ellos que aporten al análisis o búsqueda de soluciones si se deseara trabajar sobre la extensión de su *useful life*.

Eligiendo el motor 1, se obtiene el gráfico a continuación.

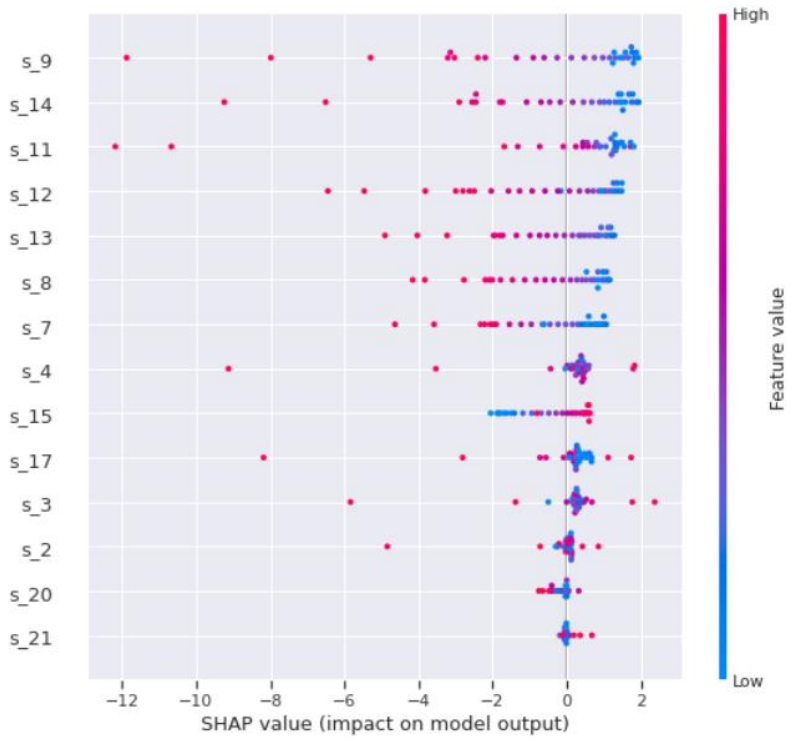


Figura 21. Explicabilidad local para el motor 1

[12] Ver Anexo 9 – Explicabilidad para LSTM

A este nuevo gráfico se le ha agregado los valores de cada sensor, lo que permite que además de ordenar los sensores en órden de influencia según *Shapley Values* – siendo el sensor 9 el de mayor contribución a la predicción y el sensor 21 el de menor –, generar una escala de colores evidenciando para cada punto si el valor del *feature* es alto o bajo en función de su distribución.

Se puede afirmar que para el sensor 9, que tiene la mayor influencia en el resultado de RUL según los valores de Shap, valores elevados – color rojo –tienen una contribución negativa al valor de RUL, mientras que los valores bajos – color azul – tienen una contribución positiva.

Se realiza nuevamente el ejercicio para el motor número 7, obteniéndose el siguiente gráfico.

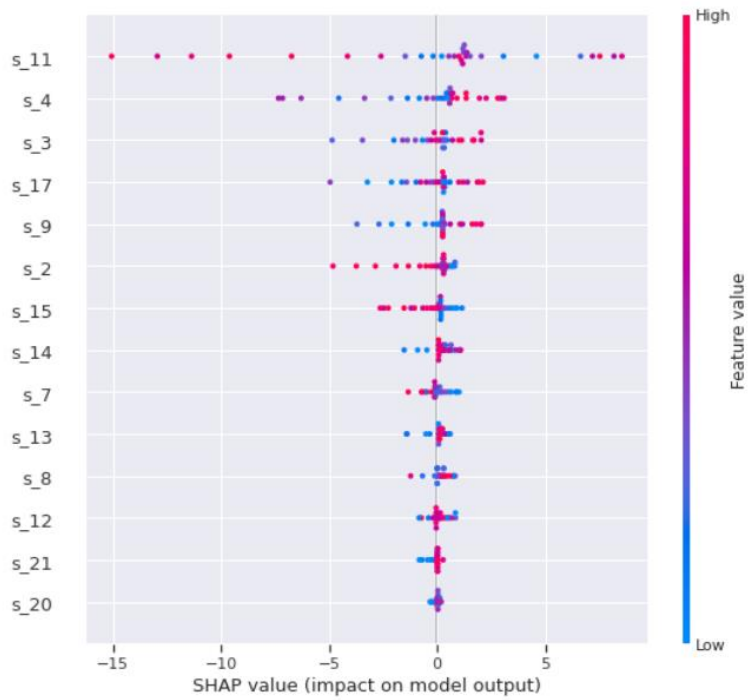


Figura 22. Explicabilidad local para el motor *7*

Para el caso del motor 7, los sensores que resultan más influyentes en el valor de la RUL son el sensor 11 y 4, mientras que aquellos tienen menor influencia son el sensor 21 y 20.

Si se comparan los resultados con el motor número 1, se observa que el sensor más influyente en ambos casos no es el mismo, pero, el sensor más influyente para el motor 7, coincide con el de la explicabilidad global. Esto nuevamente es un ejemplo de la importancia de poder profundizar más allá de una mirada global, aportando transparencia y comprensión a las decisiones que ha tomado el modelo desarrollado.

6. Conclusiones

En primer lugar, el proyecto ha servido para explorar algunas áreas de interés que no fueron estudiadas en profundidad durante el Master en Big Data, permitiendo ampliar el conocimiento y enriquecer aún más los aprendizajes obtenidos durante el desarrollo del mismo. En este sentido, se ha podido comprender e interiorizar la aplicabilidad de la Inteligencia Artificial Explicable.

Se desarrollaron modelos de *Machine Learning* para una serie de datos tabulares orientada a mantenimiento predictivo, obteniéndose buenos valores de *recall*, *precision* y costo en *test*.

Uno de los desafíos presentados para el desarrollo de estos modelos, fue el desbalance de clases detectado en el set de datos tabular, problema que se concluye que puede ser resuelto a través de estrategias como la búsqueda de un umbral o la definición de una función de costos, en conjunto con la aplicación de algoritmos de ensamble, obteniendo resultados superiores a aquellos alcanzados mediante la utilización de técnicas de desbalance de clases como son *up-sampling* o *down-sampling*.

Se profundizó en el concepto de explicabilidad, pudiendo demostrar su importancia en la resolución de problemas en el área de trabajo escogida. Para esto, se desarrolló un modelo de explicabilidad global y local para el set de datos tabular, obteniéndose mayor profundidad en comparación a la bibliografía disponible sobre el mismo *dataset*.

Por otra parte, se entiende que en las empresas de carácter industrial los datos podrían encontrarse como una serie temporal y por lo tanto, esto supone un nuevo desafío para su explotación o interpretación. De este modo, se decidió explorar un modelo de mantenimiento predictivo ya resuelto, para ampliar el campo de la explicabilidad, obteniéndose herramientas de visualización comprensibles tanto globales como locales.

Ambos modelos de explicabilidad desarrollados en el presente trabajo, se consideran extrapolables a otros casos de aplicabilidad dentro del mismo contexto y, en conjunto con el resto de los modelos desarrollados, marcan un antecedente para el mantenimiento predictivo en la industria 4.0.

Una vez concluido el trabajo, se considera que existen algunas oportunidades de mejora y trabajo a futuro. Entre ellas se encuentran la idea de desarrollar un modelo de explicabilidad para el problema de clasificación multiclase, la oportunidad de desarrollar un modelo de explicabilidad para datos almacenados en forma de serie temporal que no esté basado en redes neuronales y el afán de seguir profundizando en la alta gama de visualizaciones que ofrece la librería SHAP como foco para la explicabilidad.

7. Referencias bibliográficas

[1] Deloitte España, "¿Qué es la Industria 4.0?," [En línea]. Available: https://www2.deloitte.com/es/es/pages/manufacturing/articles/que-es-la-industria-4.0.html. [Último acceso: Agosto 2022].

[2] IBM, "¿Qué es la Industria 4.0?," [En línea]. Available: https://www.ibm.com/es-es/topics/industry-4-0. [Último acceso: Agosto 2022].

[3] C. d. P. B. "La Industria 4.0: ¿Una nueva industria?," 4 Setiembre 2021. [En línea]. Available: https://www.radioreloj.cu/revista-semanal/la-industria-4-0-una-nueva-industria/. [Último acceso: Agosto 2022].

[4] B. S. "Diferencias entre el mantenimiento predictivo y el tradicional," [En línea]. Available: https://www.predictive-sigma.com/2021/02/25/diferencias-entre-el-mantenimiento-predictivo-y-el-tradicional/#:~:text=El%20preventivo%20implementa%20medidas%20sistem%C3%A1ticas,dr%C3%A1sticamente%20los%20costes%20de%20mantenimiento. [Último acceso: Agosto 2022].

[5] J. B. J. O. y G. H. , "Predictive maintenance using Machine Learning," 3 Setiembre 2020. [En línea]. Available: https://tryolabs.com/blog/2020/09/03/predictive-maintenance-using-machine-learning. [Último acceso: Agosto 2022].

[6] Dynamox, "Conozca sobre los 4 tipos de mantenimiento industrial," 20 Agosto 2021. [En línea]. Available: https://dynamox.net/es/blog/cuatro-tipos-mantenimento-industrial. [Último acceso: Agosto 2022].

[7] Seguas, "La importancia del mantenimiento en instalaciones industriales: ¿Qué es el mantenimiento industrial?," [En línea]. Available: https://www.seguas.com/la-importancia-del-mantenimiento-en-instalaciones-industriales/#%3A~%3Atext%3DEl%20mantenimiento%20industrial%20se%20puede%2Cque%20componen%20esas%20instalaciones%20industriales. [Último acceso: Octubre 2022].

[8] Eurofins, "¿Qué diferentes tipos de mantenimiento existen en una empresa?," 21 Junio 2020. [En línea]. Available: https://www.eurofins-environment.es/es/diferentes-tipo-de-mantenimiento-existen-empresa/#%3A~%3Atext%3DDependiendo%20del%20trabajo%20a%20realizar%2C%3A%20preventivo%2C%20correctivo%20y%20predictivo. [Último acceso: Octubre 2022].

[9] Opensistemas, "¿Qué es la Inteligencia artificial explicable (XAI) y por qué es tan importante?," 4 Noviembre 2020. [En línea]. Available: https://opensistemas.com/que-es-la-inteligencia-artificial-explicable-xai-y-por-que-es-tan-importante/. [Último acceso: Enero 2023].

[10] A. S. "Interpretabilidad en Modelos de Machine Learning," 20 Marzo 2022. [En línea]. Available: https://www.linkedin.com/pulse/interpretabilidad-en-modelos-de-

machine-learning-antonio-soto/?originalSubdomain=es. [Último acceso: Enero 2023].

[11] Na8, “Interpretación de Modelos de Machine Learning,” 30 Abril 2019. [En línea]. Available: https://www.aprendemachinelearning.com/interpretacion-de-modelos-de-machine-learning/. [Último acceso: Enero 2023].

[12] M. J. V. J. L. y D. M. , “Feature Importance for Time Series Data: Improving KernelSHAP,” presented at the 3rd ACM International Conference on AI in Finance, 2022.

[13] IBM, “Inteligencia artificial explicable,” [En línea]. Available: https://www.ibm.com/es-es/watson/explainable-ai. [Último acceso: Enero 2023].

[14] SHAP, “Welcome to the SHAP documentation,” 2018. [En línea]. Available: https://shap.readthedocs.io/en/latest/index.html#:~:text=SHAP%20(SHapley%20Additive%20exPlanations)%20is,papers%20for%20details%20and%20citations).. [Último acceso: Febrero 2023].

[15] B. M. G. “Scalable Shapley Explanations in R: An introduction to the fastshap package,” 02 Octubre 2021. [En línea]. Available: https://bgreenwell.github.io/intro-fastshap/slides.html#1. [Último acceso: Febrero 2023].

[16] C. M. “Aprendizaje automático interpretable: Una guia para hacer que los modelos de caja negra sean explicables.,” 17 Agosto 2021. [En línea]. Available: https://fedefliguer.github.io/AAI/shapley.html. [Último acceso: Febrero 2023].

[17] S. M. “Explainable Artificial Intelligence for Predictive Maintenance Applications,” presented at the 3rd International Conference on Artificial Intelligence for Industries (AI4I), 2020.

[18] UCI Machine Learning Repository, “AI4I 2020 Predictive Maintenance Dataset Data Set,” 2020. [En línea]. Available: https://archive.ics.uci.edu/ml/datasets/AI4I+2020+Predictive+Maintenance+Dataset. [Último acceso: Agosto 2022].

[19] IBM Cloud Education, “Análisis exploratorio de datos,” 25 Agosto 2020. [En línea]. Available: https://www.ibm.com/es-es/cloud/learn/exploratory-data-analysis#:~:text=El%20principal%20objetivo%20de%20EDA,relaciones%20interesantes%20entre%20las%20variables.. [Último acceso: Noviembre 2022].

[20] JMP Statistical Discovery, “Análisis exploratorio de datos,” [En línea]. Available: https://www.jmp.com/es_co/statistics-knowledge-portal/exploratory-data-analysis.html. [Último acceso: Noviembre 2022].

[21] M. K. “The caret Package: 11 Subsampling For Class Imbalances,” 27 Marzo 2019. [En línea]. Available: https://topepo.github.io/caret/subsampling-for-class-imbalances.html. [Último acceso: Octubre 2022].

[22] P. C. “A gentle introduction to SHAP values in R,” 18 Marzo 2019. [En línea]. Available: https://blog.datascienceheroes.com/how-to-interpret-shap-values-in-r/. [Último acceso: Febrero 2023].

[23] B. G. “Plotting Shapley Values,” [En línea]. Available: https://bgreenwell.github.io/fastshap/reference/autoplot.explain.html. [Último acceso: Febrero 2023].

[24] Mathworks, “Estimates of predictor importance for classification ensemble of decision trees,” 2020. [En línea]. Available: https://de.mathworks.com/help/stats/.

[25] K. P. “EXPLORING NASA’S TURBOFAN DATASET: LSTM for predictive maintenance of turbofan engines,” 13 Diciembre 2020. [En línea]. Available: https://towardsdatascience.com/lstm-for-predictive-maintenance-of-turbofan-engines-f8c7791353f3. [Último acceso: Diciembre 2022].

[26] G. O. “Prognostics Center of Excellence Data Set Repository,” [En línea]. Available: https://www.nasa.gov/content/prognostics-center-of-excellence-data-set-repository. [Último acceso: Noviembre 2022].

[27] Google Scholar, 2022. [En línea]. Available: https://scholar.google.com/scholar?q=cmapss+nasa+rul%7Cdiagnosis+-compass&hl=en&as_sdt=0%2C5&as_ylo=2022&as_yhi=2022. [Último acceso: Marzo 2023].

[28] K. P. “/exploring-nasas-turbofan-dataset/8_FD004_LSTM,” 13 Diciembre 2020. [En línea]. Available: https://github.com/kpeters/exploring-nasas-turbofan-dataset/blob/master/8_FD004_LSTM.ipynb. [Último acceso: Febrero 2023].

[29] SHAP, “shap.DeepExplainer,” [En línea]. Available: https://shap-lrjball.readthedocs.io/en/latest/generated/shap.DeepExplainer.html. [Último acceso: Febrero 2023].

[30] SHAP, “Keras LSTM for IMDB Sentiment Classification,” [En línea]. Available: https://shap.readthedocs.io/en/latest/example_notebooks/text_examples/sentiment_analysis/Keras%20LSTM%20for%20IMDB%20Sentiment%20Classification.html?highlight=lstm. [Último acceso: Febrero 2023].

[31] G. M. “How to explain neural networks using SHAP,” 17 Mayo 2021. [En línea]. Available: https://www.yourdatateacher.com/2021/05/17/how-to-explain-neural-networks-using-shap/. [Último acceso: Febrero 2023].

[32] C. M. “Interpretable machine learning: 9.6 SHAP (SHapley Additive exPlanations),” 2 Marzo 2023. [En línea]. Available: https://christophm.github.io/interpretable-ml-book/shap.html. [Último acceso: Marzo 2023].

ANEXO 1 – Ingeniería de atributos para clasificación binaria

Carga de datos

En primer lugar se carga el archivo de datos csv a R y se aplican funciones para comprender la estructura del dataset.

```
raw_data <- read.csv("predictive_maintenance.csv", header = TRUE, sep = ",")

df <- raw_data

dim(df)

str(df)

colnames(df)

lapply(df, class)

summary(df)
```

Se obtiene:

- Cantidad de observaciones: 10000
- Cantidad de variables: 10 (incluyendo *Target* y *Failure Type*)
- Tipos de datos
  - Variables del tipo character: 3
  - Variables del tipo integer: 4
  - Variables del tipo numeric: 3

Tratamiento inicial de los datos

Primeramente se revisa la existencia de valores faltantes en el dataset, obteniéndose que no existen valores NA en el mismo.

```
apply(is.na(df),2,sum)
```

```
                     UDI              Product.ID                    Type      Air.temperature..K.
                       0                       0                       0                       0
Process.temperature..K.  Rotational.speed..rpm.            Torque..Nm.         Tool.wear..min.
                       0                       0                       0                       0
                  Target            Failure.Type
                       0                       0
```

Figura 23. Exploración de valores NA en el *dataset*

Debido a que la variable UDI coincide con el índice de la columna se decide eliminarla.

```
df$UDI <- NULL
```

Se decide observar la cantidad de puntos que presentan las variables *Target* y *Failure Type* en cada una de sus clases, ya que serán las variables a predecir.

```
table(df$Target)
```

```
   0    1
9661  339
```

Figura 24. Distribución de clases de la variable *Target*

```
table(df$Failure.Type)
```

```
Heat Dissipation Failure               No Failure       Overstrain Failure            Power Failure
                     112                     9652                       78                       95
        Random Failures        Tool Wear Failure
                      18                       45
```

Figura 25. Distribución de clases de la variable *Failure Type*

```
table(df$Target, df$Failure.Type)
```

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Random Failures	Tool Wear Failure
0	0	9643	0	0	18	0
1	112	9	78	95	0	45

Figura 26. Distribución de clases de la variables *Target* en conjunto con *Failure Type*

A partir de esto se desprende que hay 9661 máquinas sin falla y 339 con falla según la variable *Target*. Sin embargo, al observar la variable *Failure Type* puede apreciarse que existen inconsistencias en los datos ya que la clase *No Failure* contiene únicamente 9652 observaciones.

Al ampliar el análisis, se aprecia que aquellas observaciones que presentan *Heat Dissipation Failure*, *Overstrain Failure*, *Power Failure* y *Tool Wear Failure* no presentan inconsistencias. Ahora bien, todas las observaciones que presentan *Random Failure* presentan inconsistencias con los datos que se encuentran en la variable *Target*. Por último, 9 observaciones clasificadas como *No Failure* indican que sí han fallado según la variable *Target*.

Por este motivo y con el fin de que todos los datos sean consistentes, se eliminan aquellos registros que en la variable *Target* tengan 0 (no falla) y en *Failure Type* contengan algún valor distinto de *No Failure*, así como los registros que en la variable *Target* tengan 1 (falla) y en *Failure Type* contengan *No Failure*. Luego de esta limpieza se obtiene un conjunto de datos de 9973 registros, de los cuáles 330 fallan y 9643 no.

Luego, en caso que existan, se eliminan los registros duplicados. También, se revisa si existen *Product IDs* duplicados. En ambos casos se concluye que no habían registros duplicados ni registros con el mismo identificador de producto.

Outliers

En pos de evaluar los valores atípicos presentes en el universo de datos seleccionado, se explora qué porcentaje de outliers contiene cada una de las variables cuantitativas.

Los outliers son valores que distan del resto del conjunto de datos. En general, se consideran valores atípicos aquellos que se encuentren alejados un rango intercuartílico y medio por debajo del cuartil 1 o por encima del cuartil 3 del conjunto de datos.

$$q < Q1 - 1{,}5IQR \;\; o \;\; q > Q3 + 1{,}5IQR$$

Debido a que se está trabajando con un universo multivariado y este análisis es univariado, es menester tener cautela al momento de eliminar registros. Por este motivo, se considerarán valores atípicos aquellos que se encuentren alejados tres rangos intercuartílicos, considerados atípicos extremos. Con este criterio se obtiene:

Variable	% outlier
Air.temperature..K.	0,00%
Process.temperature..K.	0,00%
Rotational.speed..rpm.	1,06%
Toque..Nm.	0,00%
Tool.wear..min.	0,00%

Tabla 9. Proporción de *outliers* en las variables cuantitativas

Se aprecia que, la única variable que presenta *outliers* es *Rotational Speed*. Dado que la proporción de valores atípicos en dicha variable es pequeña, no se eliminarán dichos registros ni se imputarán otros valores para ellos.

Exploración y análisis descriptivo

Anteriormente se ha mencionado la importancia de estudiar las relaciones entre las variables. Para la exploración de las variables cuantitativas se utilizará una matriz de correlación.

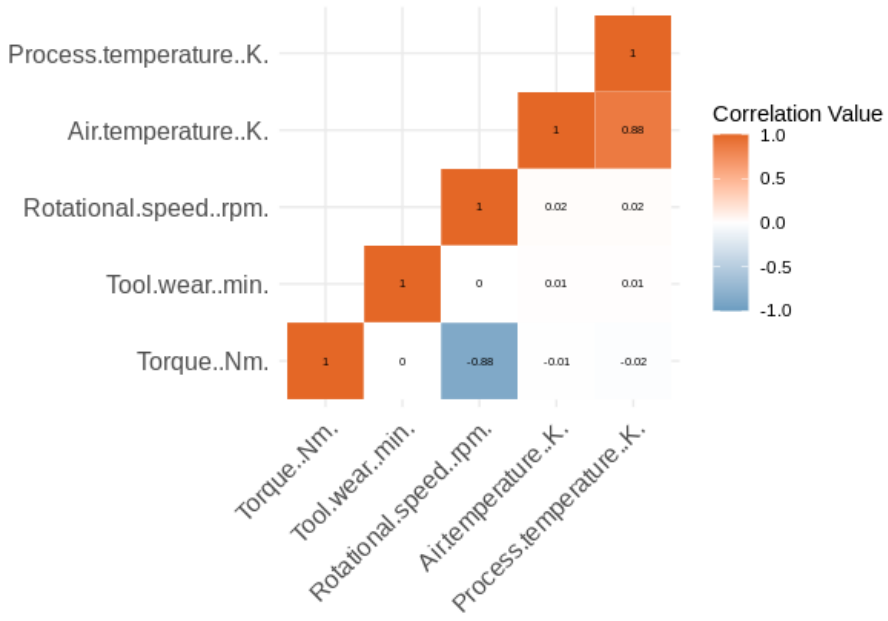


Figura 27. Matriz de correlación entre las variables cuantitativas

De la matriz se desprende que, la mayoría de las variables no presentan correlación entre ellas. Ahora bien, tanto *Torque* y *Rotational Speed* como *Air Temperature* y *Process Temperature* presentan alta correlación. El *Torque* y la *Rotational Speed* presentan alta correlación negativa, mientras que *Air Temperature* y *Process Temperature* presentan alta correlación positiva. Es importante mencionar que correlación no implica causalidad, las variables pueden moverse conjuntamente pero esto no necesariamente es porque una variable cause la otra.

Para continuar con el análisis exploratorio, se grafica la clasificación de la calidad de los productos que fallan y en aquellos que no para evaluar si la misma varía.

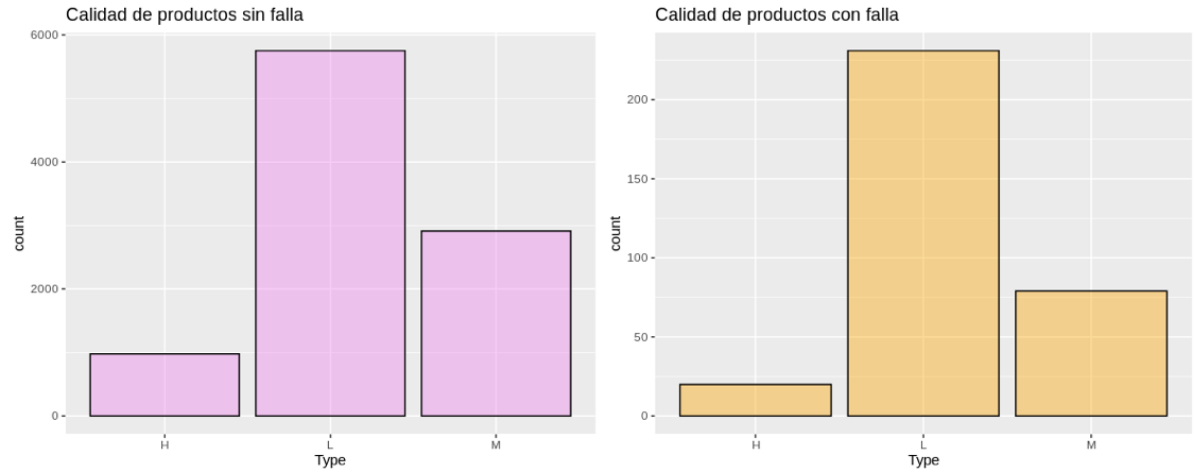


Figura 28. Clasificación según *Type* de productos con y sin falla

Calidad de productos	H	M	L
Con falla	6%	24%	70%
Sin falla	10%	30%	60%

Tabla 10. Proporción de clasificación según *Type* de productos con y sin falla

A partir de los gráficos, se aprecia que la distribución es relativamente similar en productos que han provocado que la máquina falle y en aquellos que no. En ambos casos la clase mayoritaria es la de baja calidad (*Low*), seguida por la clase de calidad media (*Medium*) y como clase minoritaria la de alta calidad (*High*). Ahora bien, al observar la tabla, se puede apreciar como las proporciones de las clases H y M disminuyen, haciendo crecer la clase L. De este modo, se podría considerar que cuando la máquina falla produce productos de menor calidad.

ANEXO 2 – Dataset para clasificación binaria

Una vez finalizado el análisis exploratorio y el tratamiento de los datos, es necesario realizar algunas transformaciones para poder generar algoritmos de clasificación binaria. Asimismo, es importante comprender la distribución de las variables a predecir, en este caso, *Target*.

Se genera un nuevo dataframe a partir de df, el dataframe con el que se ha venido realizando la ingeniería de atributos: *df_clasificacion*.

Dado que la variable *Failure Type* indica el tipo de falla, es necesario eliminarla de este datafame. El tipo de falla es una consecuencia de si el equipo falló o no, y no una causa, por lo que utilizarla como variable predictora sería un error.

Asimismo, la variable *Product ID* no funciona como una variable en sí, sino que es un identificador de producto, por lo que se toma la decisión de asignarlo como nombre de fila del dataset y se elimina como variable.

También, se sustituyen los valores 0 de la variable Target a "Ok" y los 1 a "Falla", facilitando el análisis de los datos. Luego, se castean las variables *Target* y *Type*, ambas a factor.

```
df_clasificacion <- df

df_clasificacion$Failure.Type <- NULL

rownames(df_clasificacion) <- df$Product.ID

df_clasificacion$Product.ID <- NULL

df_clasificacion$Target <-
ifelse(df_clasificacion$Target=='0','Ok','Falla')

df_clasificacion$Target <-
as.factor(df_clasificacion$Target)

df_clasificacion$Type <- as.factor(df_clasificacion$Type)
```

ANEXO 3 – Estrategias para resolver el desbalance

Métrica objetivo y función de costos

Es pertinente mencionar, para empezar, que en el problema de clasificación se ha definido que:

- La clase positiva indica que la máquina ha fallado (*Target* = "Falla")
- La clase negativa indica que la máquina no ha fallado (*Target* = "Ok")

La métrica que se utilizará para optimizar los modelos será *cost*. Esta será una función de costo definida previamente en el apartado *utils*, que penalizará aquellas predicciones que resulten más costosas para la empresa en caso de ser incorrectas. La función de costos, que es una función de pesos se ha definido, en primera instancia, de la siguiente forma:

- Sumar 10 para aquellos casos en los que se prediga "Falla" y en realidad no existiese
- Restar 25,5 para aquellos casos en los que se prediga "Falla" correctamente

De este modo, la empresa que aplique el modelo podría ahorrar 25,5 unidades al predecir correctamente, mientras que le costaría 10 unidades más al predecir incorrectamente. Cabe destacar que, estos valores no se han asignado en función de un estudio de mercado, sino que han sido escogidos al azar siguiendo el criterio de que la función le indique al algoritmo que será recompensado al estimar correctamente una falla y será penalizado, aunque en menor medida, al decir que existirá una falla cuando no es cierto. En la instancia de desarrollo y exploración de algoritmos se evalúa si estos valores generan buenos modelos o si requieren ser modificados.

Teniendo en cuenta que uno de los objetivos y beneficios del mantenimiento industrial es reducir costes es importante conocer cuál es el valor mínimo posible de obtener en *test*. Con los valores de la función de costos y con la partición *train-test* definida, el valor mínimo posible será de -0,84. Conocer este dato será importante al momento de evaluar el valor obtenido según el modelo.

La función a ser utilizada será *df.fn_summary*, basada en la función *fn_summaryCostThr* descripta en *utils*.

ANEXO 4 – Desarrollo de modelos para clasificación binaria

La partición en *train* y *test* se realiza utilizando el 90% de los datos para entrenar y el 10% restante para validación. A su vez, se definen las variables *df.form*, que indica que realizará la predicción de *Target* en función de todas las variables, y *df.metric*, que será utilizada al momento de entrenar los algoritmos, indicando que se buscará minimizar el costo.

```
df.metric <- 'cost'

df.form <- Target ~ .
```

Ahora bien, con el fin de resolver el problema de clasificación se utilizarán algoritmos de Naive Bayes, Árboles de Decisión, Random Forest y Boosting Machine.

Para que los resultados que se obtengan a través de la implementación de estos algoritmos sean reproducibles es necesario sembrar una semilla.

```
set.seed(117)
```

Paquete Caret

La mayoría de los algoritmos que se utilizarán son del paquete Caret. La ventaja de este paquete es que unifica bajo un mismo marco las funciones de distintos paquetes.

En los casos en los que se utiliza el paquete Caret, el entrenamiento de los modelos se realiza con la función *train()*, la cual posee los siguientes argumentos:

- form: detalla la fórmula del modelo a estudiar. Para el caso de la clasificación binaria la fórmula que se utilizará será *Target* ~ ., la cual se encuentra almacenada en la variable *df.form*.
- data: detalla el dataframe que contiene el subconjunto de datos que se utilizará para entrenar el modelo. En este caso, *df.part$train*.
- method: indica el nombre del algoritmo que se empleará.
- trControl: detalla las especificaciones adicionales que se utilizarán en el entrenamiento del modelo. En todos los casos el valor de este campo será generado mediante la función *trainControl()*, en donde se indica si el entrenamiento será con *Cross Validation*, *Repeated Cross Validation*, si la búsqueda de los hiperparámetros será aleatoria (search = "random") o si los mismos tomarán valores previamente definidos (search = "grid"). Por último, el parámetro *summaryFunction* tendrá la función *df.fn_summary*, ya mencionada anteriormente, que el algoritmo buscará minimizar en el entrenamiento.
- tuneGrid/tuneLength: TuneGrid será un vector, en donde se encontrarán los valores elegidos a tomar por los hiperparámetros. TuneLength es utilizado cuando se espera que el algoritmo realice una búsqueda aleatoria, indicando cuántos valores serán evaluados del parámetro principal.
- metric: detalla la métrica utilizada para evaluar la capacidad predictiva del modelo. En este caso será *df.metric*, la cual tiene almacenada la palabra *cost*.

Naive Bayes

Algoritmo generativo que parte del teorema de Bayes y lo simplifica asumiendo independencia condicional entre las variables independientes. Genera un modelo predictivo de clasificación que se basa en que la probabilidad de ocurrencia de cada variable es independiente de las otras. Entonces, Naive Bayes es una técnica basada en la independencia condicional entre los predictores para maximizar la función de utilidad.

Primeramente el algoritmo Naive Bayes obtiene un costo en test de -0,0998 y una *recall* de 0,273. Con el fin de mejorar estos resultados, se decide explorar en la utilización de *Cross Validation* y *Repeated Cross Validation*.

	Umbral	Costo en test	Accuracy	Precision	Recall
Sin CV		-0,0998	0,963	0,409	0,273
CV	0,2840	-0,1976	0,965	0,467	0,424
Repeated CV	0,2728	-0,2487	0,967	0,500	0,485

Tabla 11. Resultados obtenidos con los algoritmos de Naive Bayes

Todas las métricas que se monitorean mejoran al aplicar la técnica de *cross validation*. En este caso, y para todos los que utilicen *cross validation* en el presente documento, se aplicará con 5 *folds*.

El repeated *cross validation* aplicado es de *5 folds* con 5 repeticiones. Su aplicación hace que todas las métricas mejoren respecto a la primera corrida de Naive Bayes y respecto a aquella que utiliza *cross validation*.

Las técnicas de *cross validation* y *repeated cross validation* en este caso permitirán optimizar el umbral seleccionado a la hora de entrenar.

A continuación las matrices de confusión:

Naive Bayes sin cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	9	13
	Ok	24	951

Figura 29. Matriz de confusión de Naive Bayes sin *cross validation*

Naive Bayes con cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	14	16
	Ok	19	948

Figura 30. Matriz de confusión de Naive Bayes con *cross validation*.

Naive Bayes con repeated-cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	16	16
	Ok	17	948

Figura 31. Matriz de confusión de Naive Bayes con *repeated cross validation*

Se puede apreciar que al momento de predecir fallas aplicar técnicas de *cross validation* mejora significativamente el resultado. Aplicar *repeated cross validation* mejora levemente la performance respecto a la segunda corrida, obteniendo exactamente los mismos valores para aquellos casos en los que *Target* efectivamente es "Ok".

Entonces, en conclusión, el mejor modelo obtenido utilizando Naive Bayes es aquel que utiliza *Repeated Cross Validation*. Si bien esta técnica obtiene un buen equilibrio *bias-variance*, su costo computacional es elevado, por lo que si el tamaño muestral es muy grande no se recomienda su utilización, ya que debe priorizarse la eficiencia computacional. De este modo, considerando que los resultados no mejoran significativamente respecto a la utilización de *cross validation* y teniendo en cuenta que el objetivo de este estudio es generar modelos que puedan ser utilizados por empresas (que tendrán una gran cantidad de datos), se decide seguir la exploración de algoritmos utilizando *cross validation*.

Random Forest

Método de ensamble que combina árboles de decisión con la metodología de *bagging*, es decir, el *dataset* de entrenamiento dividido en *bags* o bolsas de datos. El algoritmo entrena un clasificador - árbol - para cada bolsa de datos. Cada árbol es entrenado con muestras distintas de datos. El random forest, con el fin de lograr una mayor independencia entre clasificadores, entrena sobre un número pequeño de atributos.

En este método se aplican los algoritmos *randomForest*, *rf* y *ranger*.

	Umbral	Costo en test	Accuracy	Precision	Recall
randomForest (sin CV)		-0,6650	0,993	1,000	0,788
rf (con CV)	0,2840	-0,6525	0,984	0,689	0,939
ranger (sin CV)	0,2500	-0,6424	0,983	0,674	0,939

Tabla 12. Resultados obtenidos con los algoritmos de Random Forest

En este caso no resulta tan sencillo definir cuál de los tres modelos es el mejor, dado que las métricas que se han definido para evaluar la bondad de los mismos (*cost* y *recall*) apuntan a modelos distintos.

Siendo que el mínimo costo posible a obtener en test es -0,84, todos los modelos han obtenido un buen resultado (entre -0,64 y -0,67). Aquel que obtuvo la mejor precision es *randomForest*, con un valor de 1, ha logrado que todas aquellas predicciones que ha catalogado como “Falla” sean aquellas que efectivamente fallen. Sin embargo, al observar la *recall*, esta es mejor en los algoritmos *rf* y *ranger*, indicando que mayor cantidad de los elementos relevantes (fallas) han sido catalogados como tales.

Random forest sin cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	26	0
	Ok	7	964

Figura 32. Matriz de confusión de Random Forest sin *cross validation*

Random forest (rf) con cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	31	14
	Ok	2	950

Figura 33. Matriz de confusión de *rf* con *cross validation*

Random forest (ranger) sin cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	31	15
	Ok	2	949

Figura 34. Matriz de confusión de *ranger* sin *cross validation*

Al observar las matrices de confusión, se aprecia que todas las predicciones de “Falla” de *randomForest* (26) son correctas, catalogando incorrectamente 7 fallas como “Ok”. Ahora bien, *rf* y *ranger* predicen correctamente 31 fallas, catalogando incorrectamente 2 fallas como “Ok” y 14 o 15 “Ok” como fallas respectivamente.

Boosting Machine

Algoritmos de aprendizaje supervisado que construyen clasificadores débiles de forma iterativa utilizando en cada paso el error anterior para perfeccionarse. La idea general es que a través de muchos clasificadores débiles, se construye un clasificador robusto. En otras palabras, Boosting Machine es un método de ensamble que se basa en entrenar predictores débiles en secuencia en busca de generar un clasificador robusto.

En este método se utilizarán los algoritmos *gbm* y *xgbTree*. Ambos son entrenados utilizando la técnica de *cross validation* con 5 *folds*.

El algoritmo *gbm* se ha aplicado utilizando search = “random” y tuneLength = 5, mientras que a *xgbTree* se le ha indicado search = “grid” y una grilla de valores a ser utilizados.

Los hiperparámetros de *xgbTree* son los siguientes:

- max_depth: profundidad máxima del árbol
- nrounds: cantidad de iteraciones
- gamma: parámetro de regularización
- eta: *learning rate*
- colsample_bytree: proporción de *features* (columnas) que se utilizarán para construir cada árbol
- min_child_weight: suma mínima del peso al realizar una partición adicional en un nodo
- subsample: proporción de submuestra de instancias de entrenamiento

Los valores han sido seleccionados en función de la solución a un problema de clasificación con similares características al que se está trabajando: problema de clasificación binaria con clases desbalanceadas y utilizando función de pesos. En primera instancia se utilizará la siguiente grilla, en caso de no obtener buenos resultados se procederá a una exploración de parámetros:

```
grid <- expand.grid(max_depth = 3,
                    nrounds = 700,
                    gamma = 0,
                    eta = 0.1,
                    colsample_bytree = 0.683,
                    min_child_weight = 1,
                    subsample = 1)
```

Los valores obtenidos son los siguientes:

	Umbral	Costo en test	Accuracy	Precision	Recall
gbm	0,2540	-0,3501	0,974	0,621	0,545
xgbTree	0,2820	-0,5702	0,982	0,692	0,818

Tabla 13. Resultados obtenidos con los algoritmos de Boosting Machine

En la tabla puede observarse que *xgbTree* obtiene mejores resultados que gbm en todas las métricas que están siendo evaluadas, especialmente en *cost* y *recall*.

En las matrices de confusión se aprecia que para aquellas observaciones que no fallan la predicción es relativamente similar (953 y 952 correctas y 11 y 12 incorrectas

respectivamente). Ahora bien, en cuanto a las predicciones de “Falla”, *xgbTree* logra predecir correctamente más puntos de observación que *gbm* (27 vs. 18).

Boosting Machine (gbm) con cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	18	11
	Ok	15	953

Figura 35. Matriz de confusión de *gbm* con *cross validation*

Boosting Machine (xgbTree) con cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	27	12
	Ok	6	952

Figura 36. Matriz de confusión de *xgbTree* con *cross validation*

Árboles de decisión

Consiste en encontrar las mejores preguntas a ejecutar sobre las observaciones, de forma de construir el árbol que mejor las clasifique. El árbol está conformado por nodos hoja, donde se define la variable a predecir, y nodos pregunta.

En particular, el algoritmo *rpart*, a medida que avanza el análisis, escoge aquella pregunta que logra nodos hoja en los que todos los elementos tienden a pertenecer a la misma clase. De este modo, si se clasifican todas las observaciones como de una clase, se logra el menor error posible. La complejidad del modelo está dada por la cantidad de preguntas que reciben los datos hasta llegar a la clasificación óptima. A su vez, existen métodos de terminación que indican al algoritmo cuando detenerse, en pos de evitar árboles muy profundos que puedan incurrir en *overfitting*.

El algoritmo utilizado será *rpart*. Si bien este modelo es el que obtiene la mejor *precision* de todos (luego de *randomForest*), sus valores *cost* y *recall* indican que la bondad del ajuste del modelo no es buena. La *precision* indica que el 83% de las predicciones categorizadas como "Falla" son correctas. Sin embargo, este modelo es conservador al momento de categorizar una observación como "Falla", provocando que casi todos los puntos del *dataset* sean categorizados como "Ok". De este modo, únicamente el 15% de las "Fallas" han sido catalogadas como tal.

	Umbral	Costo en test	Accuracy	Precision	Recall
rpart	0,2760	-0,1179	0,971	0,833	0,152

Tabla 14. Resultados obtenidos con el algoritmo de Árboles de Decisión

La matriz de confusión obtenida es la siguiente:

Árboles de decisión (rpart) con cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	5	1
	Ok	28	963

Figura 37. Matriz de confusión de *rpart* con *cross validation*

Con estos resultados, se concluye que este no es un buen modelo para resolver este problema de clasificación.

ANEXO 5 – Matrices de confusión de la optimización de resultados

Estrategia de balanceo de clases para *rf*

Se aplicará *down-sampling*, *up-sampling* y *ROSE* para el algoritmo *rf*.

Random forest con cross-validation y down-sampling

		Referencia	
		Falla	Ok
Predicción	Falla	33	152
	Ok	0	812

Figura 38. Matriz de confusión de *rf* con *cross validation* y *down-sampling*

Random forest con cross-validation y rosé

		Referencia	
		Falla	Ok
Predicción	Falla	33	159
	Ok	0	805

Figura 39. Matriz de confusión de *rf* con *cross validation* y *ROSE*

Random forest (rf) con cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	31	14
	Ok	2	950

Figura 40. Matriz de confusión de *rf* con *cross validation*

Random forest con cross-validation y up-sampling

		Referencia	
		Falla	Ok
Predicción	Falla	29	17
	Ok	4	947

Figura 41. Matriz de confusión de *rf* con *cross validation* y *up-sampling*

Cambios en el vector de umbrales

El vector de umbrales utilizado originalmente es *seq(0.25,0.30,by=0.01)*. Se explorará que resultados se obtienen utilizando los vectores *seq(0.10,0.20,by=0.01)* y *seq(0.30,0.60,by=0.01).*

Algoritmo *rf*

Random forest (rf) con cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	31	14
	Ok	2	950

Figura 42. Matriz de confusión de *rf* con *cross validation*

Random forest con cross-validation y vector 0.1,0.2

		Referencia	
		Falla	Ok
Predicción	Falla	31	20
	Ok	2	944

Figura 43. Matriz de confusión de *rf* con *cross validation* y vector 0.1,0.2

Random forest con cross-validation y vector 0.3,0.6

		Referencia	
		Falla	Ok
Predicción	Falla	30	5
	Ok	3	959

Figura 44. Matriz de confusión de *rf* con *cross validation* y vector 0.3,0.6

Algoritmo *xgbTree*

Boosting Machine (xgbTree) con cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	27	12
	Ok	6	952

Figura 45. Matriz de confusión de *xgbTree* con *cross validation*

Boosting Machine (xgbTree) con cross-validation y vector 0.1,0.2

		Referencia	
		Falla	Ok
Predicción	Falla	28	19
	Ok	5	945

Figura 46. Matriz de confusión de *xgbTree* con *cross validation* y vector 0.1,0.2

Boosting Machine (xgbTree) con cross-validation y vector 0.3,0.6

		Referencia	
		Falla	Ok
Predicción	Falla	26	3
	Ok	7	961

Figura 47. Matriz de confusión de *xgbTree* con *cross validation* y vector 0.3,0.6

Cambios en la matriz de costos

La función de costos, al ser la métrica seleccionada para optimizar, influye directamente en la manera en la que el algoritmo clasifica las observaciones. De este modo, se decide explorar la devolución del algoritmo al modificar esta función.

Originalmente se tenían los siguientes valores:

- *false positive*: 10
- *true positive*: -25,5

Se explora con los siguientes cambios:

- Opción 1:
  - *false positive*: 1
  - *true positive*: -1
- Opción 2:
  - *false positive*: 50
  - *true positive*: -25,5
- Opción 3:
  - *false positive*: 50
  - *true positive*: -90
- Opción 4
  - *false positive*: 20
  - *true positive*: -1000

Algoritmo *rf*

Random forest (rf) con cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	31	14
	Ok	2	950

Figura 48. Matriz de confusión de *rf* con *cross validation*

Random forest con cross-validation
y matriz 1, -1

		Referencia	
		Falla	Ok
Predicción	Falla	31	14
	Ok	2	950

Figura 49. Matriz de confusión de *rf* con *cross validation* y matriz 1,-1

Random forest con cross-validation.
y matriz 50, -25,5

		Referencia	
		Falla	Ok
Predicción	Falla	31	13
	Ok	2	951

Figura 50. Matriz de confusión de *rf* con *cross validation* y matriz 50,-25.5

Random forest con cross-validation
y matriz 50, -90

		Referencia	
		Falla	Ok
Predicción	Falla	31	14
	Ok	2	950

Figura 51. Matriz de confusión de *rf* con *cross validation* y matriz 50,-90

Random forest con cross-validation
y matriz 20, -1000

		Referencia	
		Falla	Ok
Predicción	Falla	31	14
	Ok	2	950

Figura 52. Matriz de confusión de *rf* con *cross validation* y matriz 20,-1000

Algoritmo *xgbTree*

Boosting Machine (xgbTree) con cross-validation

		Referencia	
		Falla	Ok
Predicción	Falla	27	12
	Ok	6	952

Figura 53. Matriz de confusión de *xgbTree* con *cross validation*

Boosting Machine (xgbTree) con cross-validation
y matriz 1, -1

		Referencia	
		Falla	Ok
Predicción	Falla	27	12
	Ok	6	952

Figura 54. Matriz de confusión de *xgbTree* con *cross validation* y matriz 1, -1

Boosting Machine (xgbTree) con cross-validation
y matriz 50, -25,5

		Referencia	
		Falla	Ok
Predicción	Falla	31	13
	Ok	2	951

Figura 55. Matriz de confusión de *xgbTree* con *cross validation* y matriz 50, -25.5

Boosting Machine (xgbTree) con cross-validation
y matriz 50, -90

		Referencia	
		Falla	Ok
Predicción	Falla	27	12
	Ok	6	952

Figura 56. Matriz de confusión de *xgbTree* con *cross validation* y matriz 50, -90

Boosting Machine (xgbTree) con cross-validation.
y matriz 20, -1000

		Referencia	
		Falla	Ok
Predicción	Falla	27	12
	Ok	6	952

Figura 57. Matriz de confusión de *xgbTree* con *cross validation* y matriz 20, -1000

ANEXO 6 – Clasificación multiclase

Dataset para clasificación multiclase

Luego del análisis exploratorio realizado en el capítulo 2, es necesario realizar algunas transformaciones para la generación de algoritmos de clasificación multiclase. Asimismo, es importante comprender la distribución de la variables a predecir, en este caso, *Failure Type*.

Se genera un nuevo dataframe a partir de df, el dataframe con el que se ha realizado la ingeniería de atributos: *df_multiclase*.

Al igual que en el dataframe para clasificación binaria, la variable *Product ID* hace referencia al indicador de los productos, por lo que se asigna como nombre de fila del *dataset* y se elimina como variable. También se elimina la variable *Target*. Luego, se castean las variables *Failure Type* y *Type* a factor.

```
df_multiclase <- df

df_multiclase$Failure.Type <-
as.factor(df_multiclase$Failure.Type)

df_multiclase$Type <- as.factor(df_multiclase$Type)

df_multiclase$Target <- NULL

rownames(df_multiclase) <- df_multiclase$Product.ID

df_multiclase$Product.ID <- NULL
```

Al terminar las transformaciones, es conveniente mirar la distribución de la variable a predecir.

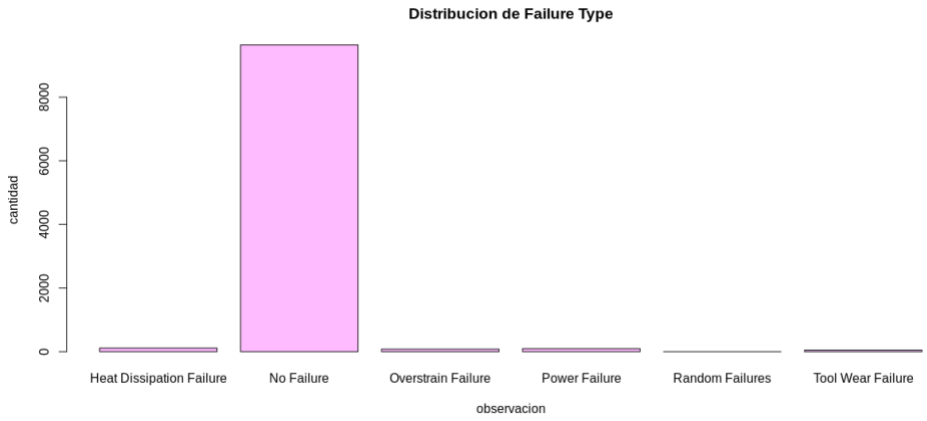


Figura 58. Distribución de la variable *Failure Type*

Se aprecia una notoria clase mayoritaria - *No Failure* -, que contiene significativamente más observaciones que las otras clases. Dado este desbalance, en conjunto con que la clase mayoritaria es aquella que indica que la máquina no ha fallado, se decide eliminar dichas observaciones y no considerarlas dentro del problema de clasificación multiclase. La solución al problema de clasificación binaria ya estará brindando respuesta a si una máquina falló o no, por ende, todas aquellas observaciones que tengan *Target* = "Ok", tendrán *Failure Type* = "*No Failure*". También resulta importante mencionar que la categoría *Random Failures* no contiene ninguna observación.

Una vez eliminadas las observaciones de la clase *No Failure*, se realiza nuevamente el gráfico de barras:

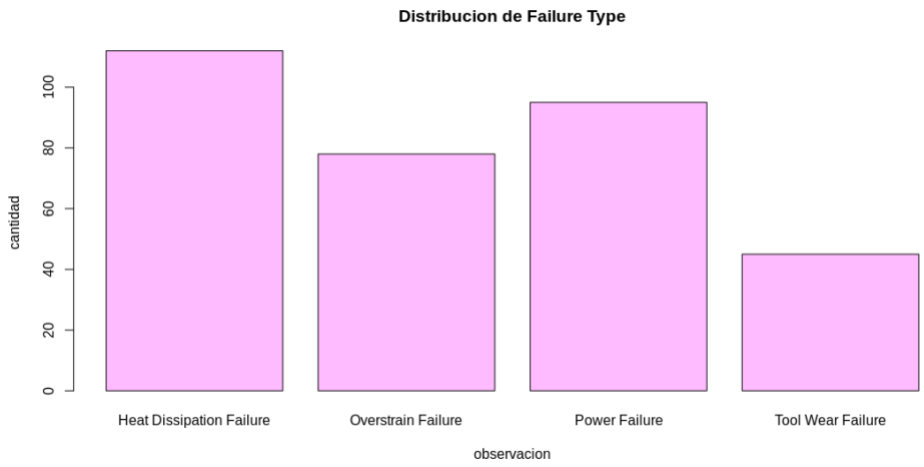


Figura 59. Distribución de la variable *Failure Type* sin *No Failure*

No se aprecia que exista un desbalance. Sin embargo, la diferencia de puntos de observación entre *Tool Wear Failure* (clase con menor frecuencia) y *Heat Dissipation Failure* (clase con mayor frecuencia) puede generar problemas al momento de entrenar los algoritmos y predecir.

Algoritmos para clasificación multiclase

En primer lugar, se realiza la partición en *train* y *test*. En este caso, el 90% de los datos serán utilizados para el entrenamiento de los algoritmos, mientras que el 10% restante serán considerados datos de prueba para evaluar la capacidad predictiva de los modelos en datos nunca antes vistos por los mismos. También se define la variable *df.form*, que indica que se realizará la predicción de *Failure Type* en función de todas las variables.

```
df.form <- Failure.Type ~ .
```

En el problema de clasificación binaria los algoritmos que han obtenido mejor performance han sido Random Forest (*rf*) y Boosting Machine (*xgbTree*). Por este motivo, serán los algoritmos que se utilizarán para intentar resolver el problema de clasificación multiclase.

El objetivo de este apartado es encontrar un modelo que logre clasificar a las observaciones según su tipo de falla - *Failure Type* - correctamente.

Random Forest

Algoritmo de aprendizaje automático que funciona creando varios árboles de decisión y los combina en un bosque aleatorio. Cada árbol de decisión es entrenado con un subconjunto de datos.

Este algoritmo utiliza una estrategia de votación para decidir a qué clase corresponde cada observación. Cada árbol de decisión brinda una predicción y la clase con más votos es seleccionada como la clase predicha.

En este caso, el modelo devuelve la probabilidad de la observación de pertenecer a cada clase. Esta se calcula a partir de la proporción de árboles que predicen cada clase para una observación dada. La clase con mayor probabilidad es seleccionada como la clase a la que pertenece dicha observación.

Se entrena el algoritmo *rf* con 5 folds de *cross-validation* y search = “random”.

```
train_ctrl <- trainControl(method="cv", number=5,
search="random")

model_rf<- train(df.form, data=df.part$train, method="rf",
metric="Accuracy",
trControl=train_ctrl,ntree=741,keep.forest=TRUE,importance=TRU
E)
```

Los resultados obtenidos son los siguientes:

	Accuracy	Precision	Recall
rf	0,879	0,856	0,900

Tabla 15. Resultados obtenidos con el algoritmo *rf* en el problema de clasificación multiclase

Al ser un problema de clasificación multiclase los valores expuestos anteriormente son la *macro avg. precision* y la *macro avg. recall*. Estas métricas permiten evaluar el desempeño del modelo. Sin embargo, pueden llegar a ser engañosas ya que brindan igual peso a todas las clases. Por este motivo, el modelo puede ser mejor para clasificar ciertas clases y estar generando malas predicciones en otras.

De este modo, es importante observar la matriz de confusión, ya que permite visualizar de manera clara y concisa cómo está clasificando el modelo cada clase y cuáles son los errores en los que está incurriendo.

Random forest con p=0,9

		Referencia			
		Heat Dissipation	Overstrain	Power	Tool Wear
Predicción	Heat Dissipation	8	0	0	0
	Overstrain	0	5	2	1
	Power	0	0	12	0
	Tool Wear	0	0	1	4

Figura 60. Matriz de confusión de *rf* multiclase

De la matriz se desprende que este modelo clasifica correctamente a todas las observaciones de *Heat Dissipation Failure* y *Overstrain Failure*. Sin embargo, clasifica mal ciertas observaciones de *Power Failure* y *Tool Wear Failure*.

Boosting Machine

Técnica de aprendizaje automático que combina múltiples modelos de aprendizaje débiles para crear un modelo más fuerte y preciso. Estos se combinan de manera secuencial, donde cada modelo posterior se enfoca en corregir los errores del modelo anterior.

Se entrena el algoritmo *xgbTree*. Los resultados obtenidos son los siguientes:

	Accuracy	Precision	Recall
xgbTree	1,000	1,000	1,000

Tabla 16. Resultados obtenidos con el algoritmo *xgbTree* en el problema de clasificación multiclase

Boosting Machine (xgbTree) con p=0,9

		Referencia			
		Heat Dissipation	Overstrain	Power	Tool Wear
Predicción	Heat Dissipation	8	0	0	0
	Overstrain	0	5	0	0
	Power	0	0	15	0
	Tool Wear	0	0	0	5

Figura 61. Matriz de confusión de *xgbTree* multiclase

Este modelo logra clasificar correctamente a todas las observaciones. Esto quiere decir que el modelo es perfectamente preciso para estos datos.

Optimización de resultados

En este punto se buscará mejorar los modelos obtenidos anteriormente. Para el caso del modelo Boosting Machine, si bien el resultado fue exacto para los datos, igual se buscará optimizarlo para asegurar que el mismo no se encuentra entregando esos resultados como consecuencia de *overfitting*. Las técnicas que se utilizarán son cambios en la partición *train-test* y aplicación de estrategias de balanceo de clases.

Cambios en la partición train-test

La partición *train-test* se refiere a la forma en que se dividen los datos disponibles en conjuntos de entrenamiento y prueba.

La partición óptima depende del conjunto de datos y del problema específico que se esté abordando. En particular, si se trabaja con clases que presentan desbalance o si el conjunto de datos es pequeño, puede ser necesario ajustar la partición en pos de mejorar los resultados del algoritmo.

De este modo, se realiza una nueva partición en *train* y *test*, utilizando el 70% de los datos para entrenamiento y el 30% para prueba.

Los resultados obtenidos con el algoritmo *rf* son los siguientes:

	Accuracy	Precision	Recall
p=0,9	0,879	0,831	0,900
p=0,7	0,848	0,789	0,850

Tabla 17. Resultados obtenidos con el algoritmo *rf* con cambios en la partición *train-test*

Random forest con p=0,7

		Referencia			
		Heat Dissipation	Overstrain	Power	Tool Wear
Predicción	Heat Dissipation	33	0	2	0
	Overstrain	2	14	5	2
	Power	0	1	27	0
	Tool Wear	0	1	2	10

Figura 62. Matriz de confusión de *rf* con partición 70-30

Al modificar la partición *train-test* no se obtienen mejores resultados. Con la partición 90-10 el modelo lograba clasificar correctamente a todas las observaciones que presentaban *Heat Dissipation Failure* y *Overstrain Failure*. Sin embargo, con esta nueva partición el modelo no logra clasificar correctamente en su totalidad a ninguna de las clases.

Al observar la *recall* o *sensitivity* individual de cada clase, se puede apreciar que solo ha mejorado en *Tool Wear Failure*. En otras palabras, el modelo generado con la partición 70-30 clasifica mejor las observaciones que presentan *Tool Wear Failure* que el modelo generado con la partición 90-10.

	Heat Dissipation	Overstrain	Power	Tool Wear
p=0,9	1,00	1,00	0,80	0,80
p=0,7	0,94	0,88	0,75	0,83

Tabla 18. *Recall* individual por clase para *rf*

Para el algoritmo *xgbTree* se obtienen los siguientes resultados:

	Accuracy	Precision	Recall
p=0,9	1,000	1,000	1,000
p=0,7	0,859	0,801	0,866

Tabla 19. Resultados obtenidos con el algoritmo *xgbTree* con cambios en la partición *train-test*

Boosting Machine (xgbTree) con p=0,7

		Referencia			
		Heat Dissipation	Overstrain	Power	Tool Wear
Predicción	Heat Dissipation	34	0	4	1
	Overstrain	1	15	5	1
	Power	0	1	26	0
	Tool Wear	0	0	1	10

Figura 63. Matriz de confusión de *xgbTree* con partición 70-30

En ninguna de las métricas evaluadas se han obtenido mejores resultados. Tampoco se ha logrado mejorar la *recall* individual de cada clase:

	Heat Dissipation	Overstrain	Power	Tool Wear
p=0,9	1,00	1,00	1,00	1,00
p=0,7	0,97	0,94	0,72	0,83

Tabla 20. *Recall* individual por clase para *xgbTree*

Estrategias de balanceo de clases

En el problema de clasificación multiclase no existe un claro desbalance de clases. Sin embargo, se explorará si los resultados mejoran al aplicar este tipo de técnicas. En este caso, se aplicarán las estrategias de *down-sampling* y *up-sampling* para el algoritmo *rf*.

	Accuracy	Precision	Recall
sin estrategia	0,879	0,856	0,900
down-sampling	0,848	0,848	0,917
up-sampling	0,909	0,887	0,950

Tabla 21. Resultados obtenidos con el algoritmo *rf* con estrategias de balanceo

Al aplicar *down-sampling* los resultados empeoran en comparación a los valores obtenidos sin estrategia, con excepción de la *recall* que presenta una leve mejora. Sin embargo, al aplicar *up-sampling* se aprecia que estos mejoran.

Las matrices de confusión obtenidas son las siguientes:

Random forest con down-sampling con p=0,9

		Referencia			
		Heat Dissipation	Overstrain	Power	Tool Wear
Predicción	Heat Dissipation	8	0	3	0
	Overstrain	0	5	1	0
	Power	0	0	10	0
	Tool Wear	0	0	1	5

Figura 64. Matriz de confusión de *rf* con *down-sampling*

Random forest con up-sampling con p=0,9

		Referencia			
		Heat Dissipation	Overstrain	Power	Tool Wear
Predicción	Heat Dissipation	8	0	0	0
	Overstrain	0	5	2	0
	Power	0	0	12	0
	Tool Wear	0	0	1	5

Figura 65. Matriz de confusión de *rf* con *up-sampling*

En todos los casos (sin estrategia, *down-sampling* y *up-sampling*) se clasifica correctamente a las observaciones que presentan *Heat Dissipation Failure* y *Overstrain Failure*. Ahora bien, en el caso sin estrategia no se logra clasificar correctamente a todas las observaciones que presentan *Tool Wear Failure*. Sin embargo, esto se cumple satisfactoriamente en los casos de *down-sampling y up-sampling*. En ninguno de los casos se logra clasificar de manera adecuada a todas las observaciones que presentan *Power Failure*.

Las *recall* individuales por clase son las siguientes:

	Heat Dissipation	Overstrain	Power	Tool Wear
sin estrategia	1,00	1,00	0,80	0,80
down-sampling	1,00	1,00	0,67	1,00
up-sampling	1,00	1,00	0,80	1,00

Tabla 22. *Recall* individual por clase para *rf* con estrategias de balanceo

De este modo, el mejor modelo de *rf* es aquel que utiliza la estrategia de *up-sampling*.

Conclusiones de la optimización

El objetivo de la optimización de resultados es mejorar la capacidad predictiva de los modelos. En este caso, se buscó a partir de cambios en la partición *train-test* y estrategias de balanceo de clases.

Los modelos que han sido entrenados utilizando la partición *train-test* de 70-30 no han logrado conseguir mejores resultados que aquellos generados con la partición 90-10.

En relación a las técnicas de balanceo de clases, la aplicación de la técnica *up-sampling* al algoritmo *rf* permite mejorar la capacidad predictiva de este modelo.

Ahora bien, el modelo que utiliza *rf* con *up-sampling* no logra superar la capacidad predictiva del modelo que utiliza *xgbTree* corrido en primera instancia.

En conclusión, el mejor modelo para predecir qué tipo de falla presentará cada observación es aquel que utiliza el algoritmo *xgbTree* con técnica de *cross-validation* con 5 *folds*, search = “random” y partición *train-test* de 90-10.

ANEXO 7 – Fastshap para R

Para el lenguaje R, existen dos paquetes disponibles que utilizan la misma teoría de *Shapley Values*. Ellos son *shapper* y *fastshap*. En el presente trabajo se utilizará el segundo, el cual es aplicable a cualquier modelo de aprendizaje supervisado y además, es capaz de calcular los *Shapley Values* de forma veloz, en comparación a otras implementaciones.

Se entrena al algoritmo *ranger* y almacena el modelo con el nombre *rfo*. Por otra parte se crea la función *pfun*, la cual se utilizará para almacenar las predicciones del modelo – a través de la función *predict* – de tal forma que luego puedan ser utilizadas por la función *explain*. Al tratarse de una clasificación binaria, la función en cuestión entregará un vector de probabilidades de clase.

```
  set.seed(117)
library(ranger)
rfo <- ranger(Target ~ ., data = df_explicabilidad,
probability = TRUE)
```

```
  pfun <- function(object, newdata) {
 predict(object, data = newdata)$predictions[, 2]}
```

El *dataset* que recibirá la función *explain*, no contendrá la variable *Target*.

```
  X <- subset(df_explicabilidad, select = -Target)
```

Se utiliza finalmente la función *explain* de la librería *fastshap*, la cual calculará los *Shapley Values*. Cabe destacar que la función en cuestión recibe:

- object: modelo ajustado, en este caso *Ranger* (*rfo*)
- X: un objeto R similar a una matriz, sin la variable a predecir (*Target*)
- nsim: número de repeticiones de Monte Carlo a utilizar para estimar cada valor de Shapley. Por defecto toma el valor de 1, solo se utiliza cuando exact = FALSE y cabe destacar que cuanto mayor sea su valor, más exactos serán los resultados.

- adjust: variable lógica que indica si se ajusta o no la suma del valor Shapley estimado para satisfacer la propiedad de aditividad, es decir, si se ajustan los valores de Shapley para igualar la diferencia entre la predicción del modelo para esa muestra y el promedio de predicción sobre todos los datos de entrenamiento (X)
- pred_wrapper: función de predicción que requiere los argumentos *object* y *new data*. Será un argumento necesario siempre que exact = FALSE y la salida de la función será determinada según el caso de estudio. Si se trata de una regresión entregará un vector numérico con los resultados predichos, si se trata de una clasificación binaria o multiclase, entregará un vector con probabilidades de clase pronosticadas para la clase de referencia.
- exact: variable lógica que indica si se deben calcular los valores exactos de Shapley. Existe actualmente únicamente para utilizarse con modelos *lm* y *xgboos*t. Su valor por defecto es FALSE, motivo por el cual no fue especificado en la línea de código.

```
(ex.all <- explain(rfo, X = X, nsim = 99, adjust =
TRUE,  pred_wrapper = pfun))
```

A continuación la vista previa que aparece en la consola al correr el comando anterior. Pueden verse algunos de los *Shapley Values* calculados.

```
# A tibble: 9,973 × 6
         Type Air.temperature..K. Process.temperature..K. Rotational.speed..rpm. Torque..Nm. Tool.wear..…¹
        <dbl>               <dbl>                   <dbl>                  <dbl>       <dbl>          <dbl>
 1  0.000655              0.0136                -0.00135                -0.00283      0.0126         0.0103
 2 -0.000618              0.0118                -0.00313                 0.00627      0.0150         0.00317
 3 -0.00000788            0.00729                0.000328                0.0124      -0.00108        0.0140
 4  0.0000598             0.00486               -0.00209                -0.00466      0.0281         0.00596
 5  0.000538              0.0168                -0.00455                 0.00179      0.0149         0.00305
 6  0.000853              0.00495               -0.00996                 0.0124       0.0226         0.00196
 7  0.00000722            0.0109                -0.00242                 0.0100       0.00791        0.00653
 8 -0.0000782             0.00604                0.000225                0.00764      0.0125         0.00664
 9  0.00340               0.00540               -0.00107                 0.00948      0.0133         0.00246
10 -0.000584              0.00779               -0.00165                 0.0108       0.0106         0.00597
# … with 9,963 more rows, and abbreviated variable name ¹Tool.wear..min.
# ℹ Use `print(n = ...)` to see more rows
```

Figura 65. Vista previa en consola de *Shapley Values* para la explicación global

Se plotean los datos obtenidos con la librería *ggplot2* de forma de visualizar los resultados de forma amigable en busca de obtener conclusiones a partir de ellos.

```
p1 <- autoplot(ex.all)
p2 <- autoplot(ex.all, type = "dependence", feature = "Type",
X = X,color_by = "Type", alpha = 0.5) + theme(legend.position
= c(0.8, 0.8))
gridExtra::grid.arrange(p1, p2, nrow = 1)
```

Para el caso de explicabilidad local se utiliza un código análogo al anterior, con la salvedad de que se selecciona previamente el caso puntual a estudiar y se almacena como dataframe en la variable "puntual".

Por otra parte, también se almacena en la variable res el valor de todas las features del caso puntual elegido en conjunto con su *Shapley Value*. De esta forma se encuentra accesible la información en caso de necesitar consultarla.

Por último, a la función autoplot se le asignará esta vez la variable *type* como "*contribution*".

```
  res <- data.frame("feature" = paste0(names(puntual), "=",
t(puntual)), "shapley.value" = t(ex.puntual))

  autoplot(ex.puntual, type = "contribution")
```

ANEXO 8 – SHAP para Python

El enfoque SHAP (*Shapley Additive exPlanations*) puede desarrollarse a través de la librería shap de Python: una librería de inteligencia artificial explicable que utiliza cálculos del campo de la teoría de juegos para entender qué variables tienen más influencia en las predicciones.

Además, la librería SHAP posee diferentes métodos para calcular los valores de Shapley dependiendo del modelo que quiera explicarse:

- DeepExplainer: Para modelos *de Deep Learning*
- TreeExplainer: Para algoritmos de árboles de decisión como *Decision Tree*, *XGBoost* y *Random Forest*, entre otros
- LinearExplainer: Para modelos lineales como *LinearRegression*
- GradientExplainer: Método que utiliza gradientes para aproximar los valores SHAP (*Shap Values*) en modelos de *Deep Learning*
- KernelExplainer: Método que puede ser utilizado para explicar cualquier tipo de modelo, ya sean modelos lineales, algoritmos de árboles de decisión o modelos de *Deep Learning*

En el presente trabajo, al profundizar en la explicabilidad de una red LSTM, se utiliza el método *Deep Explainer* a partir de la siguiente línea de código.

```
   explainer = shap.DeepExplainer(final_model,
train_array[:100])

   shap_values = explainer.shap_values(test_array[:10],
check_additivity=False)
```

En principio, durante la generación del modelo de explicabilidad se incurrió en un *Assertion Error*. El mismo indica que las explicaciones de SHAP no suman la salida del modelo. De este modo, ha cobrado importancia el parámetro *check_additivity*, parámetro booleano que ejecuta una validación sobre la suma de los valores SHAP. El mismo ha sido seteado en *False*, ya que el error es causado por un error de redondeo.

Una vez resuelto el error, *shap_values* devuelve los valores estimados de SHAP para el modelo aplicado a los datos otorgados a la función. En este caso se le otorgan los 10 primeros conjuntos de test (*test_array[:10]*), en donde cada uno de los 10 conjuntos posee 30 valores de tiempo, cada una con 14 valores de sensores.

Para poder hacer interpretaciones a partir de los *Shap Values*, se recurre a la función *shap.summary_plot* de la siguiente manera.

```
shap.summary_plot(shap_values[0][0],feature_names=remaining_
sensors)
```

Además, la librería Shap posee opciones de visualización, las cuales se cargan en la notebook de trabajo con el comando:

```
shap.initjs()
```

ANEXO 9 – Explicabilidad para LSTM

Explicabilidad global

Para el desarrollo de la explicabilidad global es necesario realizar la suma de los *Shapley Values* de los motores seleccionados y luego realizar un promedio. Para ello, se define la variable "muestra", a la cual se le asigna el número de motores que serán tenidos en cuenta.

```
shap_values=explainer.shap_values(test_array[:muestra],
check_additivity=False)
```

Se calcula entonces el promedio de los *Shap Values* de todos los motores para cada tiempo. Esto se realiza con las siguientes líneas de código.

```
sum_shap_values=np.zeros((30,14))
for i in range(0,30):
    for j in range(0,14):
        for k in range(0,muestra):
            sum_shap_values[i][j]+=shap_values[0][k][i][j]
media_shap_values=sum_shap_values/muestra
```

Explicabilidad local

Con el fin de poder escoger el caso puntual sobre el cuál desarrollar el modelo de explicabilidad, es necesario generar un código que permita elegir un motor y así obtener la explicabilidad de los resultados asociados únicamente a él.

Para esto, se desarrolla un “buscador”, en donde se asigna el motor (ejemplo ['unit_nr'] == 1) y para el cual específicamente se calcularán luego los *Shapley Values*.

```
test_buscador = X_test_interim.loc[X_test_interim['unit_nr'] == 1]
```

ANEXO 10 – Código Utils

```
if(!require(caret)) install.packages('caret')

library(caret)


# Particion

train_dev_partition <- function(df, p = 0.7) {

  size <- round(nrow(df) * p)

  train_idx <- sample.int(nrow(df), size)

  train <- df[train_idx,]

  dev  <- df[-train_idx,]

  return(list(train = train, dev = dev))

}


# Costo con pesos

fn_cost <- function(yhat, y) {

  fp_cost <- 10 * ( yhat == 'Falla' & y == 'Ok' )

  tp_cost <- - 25.5 * ( yhat == 'Falla' & y == 'Falla' )

  return(mean(fp_cost + tp_cost))

}
```

```
fn_summarycost <- function(data, lev = NULL, model = NULL) {
  cost <- fn_cost(yhat = data$pred, y = data$obs)
  c(cost = cost)
}

# error de clasificacion
fn_err_cla <- function(yhat, y) { mean(yhat != y) }

# cost threshold
fn_pred <- function(data, thr = 0.5) {
  factor(ifelse(data[, 'Falla'] > thr, 'Falla', 'Ok'))
}
```

```
fn_summarycostThr <- function(data, thr_vec) {
  cost <- c()
  for(thr in thr_vec){
    pred_thr <- fn_pred(data, thr)
    cot <- fn_cost(yhat = pred_thr, y = data$obs)
    cost <- c(cost, cot)
  }
  c(cost = min(cost), prob_thr = thr_vec[which.min(cost)])
}

fn_results <- function(model) {
  cost <- model$results$cost
  prob_thr <- model$results$prob_thr[which.min(cost)]
  return(list(cost = min(cost), prob_thr = prob_thr))
}
```

```
# matriz de confusion
conf_matrix <- function(y_pred, y_real, positive = 'Falla') {
  cm <- confusionMatrix(data = y_pred,
                        reference = y_real,
                        positive = positive,
                        mode = 'prec_recall')
  return(cm$table)
}
# plot
plot_thr_cost <- function(thr_cost, thr_vec, main = '') {
  plot(thr_cost,
       type = 'l',
       col = 'red',
       main = main,
       ylab = 'Costo',
       xlab = 'Umbral',
       xaxt = 'n')
  axis(1,
       at = seq(1, length(thr_vec)),
       labels = thr_vec)
}
```