Trabajo Fin de Grado
en Ingeniería de Tecnologías industriales

Detección de anomalías en placas de acero utilizando técnicas de inteligencia artificial

Autor: Ignacio González Peris

Tutor: Francisco José Simois Tirado

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025

Trabajo Fin de Grado
en Ingeniería de Tecnologías Industriales

Detección de anomalías en placas de acero utilizando técnicas de inteligencia artificial

Autor:
Ignacio González Peris

Tutor:
Francisco José Simois Tirado
Profesor titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2025

Trabajo Fin de Grado: Detección de anomalías en placas de acero utilizando técnicas de inteligencia artificial

Autor: Ignacio González Peris

Tutor: Francisco José Simois Tirado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2025

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Quisiera expresar mi gratitud al profesor Francisco José Simois, cuyo acompañamiento ha sido decisivo para la culminación de este Trabajo Fin de Grado. Su orientación técnica, su constante disposición para resolver dudas y, sobre todo, su confianza en mis capacidades han impulsado cada etapa del proyecto. Agradezco también a mis compañeros y familia, cuyo apoyo incondicional ha proporcionado el ánimo necesario para superar los retos que surgieron durante la investigación. Finalmente, extiendo mi reconocimiento a la Escuela y al personal de laboratorio por facilitar los recursos y el entorno académico que hicieron posible este trabajo.

Ignacio González Peris

Grado en ingeniería de tecnologías industriales

Sevilla, 2025

Resumen

El objetivo del presente documento consiste en la predicción de defectos en placas de acero utilizando herramientas de machine learning. La base de datos empleada proviene de la serie Tabular Playground de Kaggle, diseñada para practicar técnicas de machine learning y feature engineering sobre datos sintéticos. La competición consiste en anticipar la probabilidad de siete tipos de fallos: Pastry, Z_Scratch, K_Scatch, Stains, Dirtiness, Bumps y Other_Faults. Empleando únicamente datos tabulares, es decir, sin recurrir a imágenes. Para ello, se aprovechan librerías de Python como Pandas y Scikit-Learn en las fases de limpieza, generación de variables y modelado, buscando maximizar el promedio del AUC-ROC de cada clase.

El análisis se estructura en tres etapas principales: exploración inicial de los datos para comprender distribuciones y relaciones, ingeniería de características dirigida a capturar patrones relevantes y entrenamiento de modelos supervisados con ajuste de hiperparámetros. Finalmente, se evalúa el rendimiento mediante curvas ROC y se genera el archivo de envío en formato CSV con las probabilidades predichas para cada ID. Más allá de la puntuación, este TFG pone en práctica los conocimientos adquiridos en metodología de machine learning, desde el preprocesamiento hasta la interpretación de resultados.

.

Abstract

This work explores modern machine-learning techniques for automated quality control in the steel industry through the Kaggle "Steel Plate Defect Prediction – TPS S4 E3" challenge. Using a fully synthetic, tabular dataset of 19 219 samples with 27 engineered features, we model the joint probability of seven common defects: Pastry, Z-Scratch, K-Scratch, Stains, Dirtiness, Bumps and Other Faults, without relying on image data. A reproducible pipeline integrates rigorous exploratory analysis, targeted feature engineering and class-imbalance mitigation, followed by a comparison of baseline linear models, Random Forests and gradient-boosted trees. The best LightGBM ensemble achieves a mean macro AUC-ROC of 0.883 across cross-validation folds, demonstrating competitive performance while retaining interpretability. Beyond leaderboard scores, the study highlights practical considerations (reproducibility, deployment readiness and explainability) essential for translating data-centric research into industrial inspection systems.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

A^*	Conjugado
c.t.p.	En casi todos los puntos
c.q.d.	Como queríamos demostrar
■	Como queríamos demostrar
e.o.c.	En cualquier otro caso
e	número e
IRe	Parte real
IIm	Parte imaginaria
sen	Función seno
tg	Función tangente
arctg	Función arco tangente
sen	Función seno
$\sin^x y$	Función seno de x elevado a y
$\cos^x y$	Función coseno de x elevado a y
Sa	Función sampling
sgn	Función signo
rect	Función rectángulo
Sinc	Función sinc
∂y ∂x	Derivada parcial de y respecto
$x_\circ$	Notación de grado, x grados.
Pr(A)	Probabilidad del suceso A
SNR	Signal-to-noise ratio
MSE	Minimum square error
:	Tal que
<	Menor o igual
>	Mayor o igual
\	Backslash
⇔	Si y sólo si

1 INTRODUCCIÓN

Durante décadas, la industria siderúrgica ha intentado responder una pregunta tan simple como incómoda: ¿cómo saber si una lámina de acero, aparentemente perfecta, esconde un defecto? En sus comienzos, la respuesta fue decididamente artesanal: ojos entrenados, normas técnicas y mucha experiencia acumulada. La inspección visual manual fue, por largo tiempo, el bastión del control de calidad. Sin embargo, como todo oficio basado en la percepción humana, esta práctica también arrastraba sus límites: fatiga, subjetividad y errores inevitables.

Con la llegada de la automatización y el auge de la informática, el escenario cambió radicalmente . [1] Aparecieron métodos automáticos que, como centinelas incansables, analizaban datos estructurados, ya fueran cifras frías derivadas de sensores o imágenes transformadas por algoritmos. Antes de que el deep learning tomara por asalto la escena tecnológica, se empleaban técnicas de procesamiento digital de señales e imágenes para extraer características clave: formas, niveles de luminancia o transformadas en frecuencia [2]. A partir de ahí, los datos eran pasados por el tamiz de clasificadores tradicionales. Entre los enfoques clásicos se destacaban la segmentación por umbrales, las transformadas de Fourier o los filtros de Gabor, todos diseñados para iluminar las imperfecciones ocultas en la textura del metal [2]. A su vez, se aplicaban métodos estadísticos para detectar anomalías y desviaciones sutiles en la superficie. [1]

Poco a poco, la inteligencia artificial (la "vieja escuela" del aprendizaje automático) empezó a ganar terreno. Hacia finales de los años noventa y principios de los 2000 [3], se introdujeron algoritmos supervisados aplicados a datos estructurados . [4, 3, 5] La idea era sencilla y ambiciosa a la vez: dejar que las máquinas aprendieran a reconocer defectos a partir de ejemplos previos. Surgieron modelos como los árboles de decisión, las redes neuronales con una sola capa oculta o las célebres máquinas de vectores de soporte (SVM), [6] entrenadas con atributos numéricos que describían, la superficie de las placas de acero. Eran modelos aún primitivos si los comparamos con las redes profundas actuales, pero marcaron el primer paso serio hacia la automatización del juicio técnico.

El panorama se volvió más interesante con la llegada de la década de 2010 [4]. La disponibilidad de conjuntos de datos públicos y el avance de las técnicas de minería de datos despertaron un renovado entusiasmo investigador. Un hito particular fue la publicación en 2010 del conjunto de datos Steel Plates Faults en el repositorio de la UCI . [7] Este conjunto (una suerte de "cartilla escolar" para algoritmos) incluye 27 atributos físicos de placas de acero y el tipo de defecto asociado. Desde entonces, se ha convertido en un campo de pruebas para numerosos clasificadores desarrollados en la literatura especializada. Entre las variables se encuentran medidas como índices de forma, niveles de luminosidad o dimensiones, que permiten inferir, con razonable precisión, el tipo de falla en la fabricación. [7, 8, 2]

Los enfoques más tradicionales para la detección de defectos en datos tabulares recurrían a modelos "poco profundos", en contraposición a las actuales redes neuronales profundas . [6] El arsenal incluía análisis de varianza, control estadístico de procesos y algoritmos de clasificación diseñados para predecir automáticamente el tipo de defecto. Con el tiempo, modelos genéricos como Random Forest , las SVM o las redes de pocas capas demostraron ser más consistentes que el ojo humano [4, 1, 6]. Investigaciones previas también exploraron la combinación de análisis de imágenes (segmentación espacial o frecuencial) con clasificadores automáticos clásicos . [6]

Estos modelos permitieron reducir la dependencia de reglas técnicas o inspecciones visuales. Sin embargo, su rendimiento seguía atado al ingenio humano: dependían en gran medida de características diseñadas por expertos [2]. Eran sistemas eficaces, pero también frágiles fuera de su contexto. Esta limitación impulsó una segunda ola de investigaciones: la búsqueda de enfoques más robustos, menos sensibles al entorno y, en definitiva, más "inteligentes". [9]

1.1 Objetivos

El propósito general de este Trabajo Fin de Grado es demostrar la aplicación rigurosa de técnicas modernas de Machine Learning (ML) para la detección automática de defectos en placas de acero, utilizando el reto

Steel Plate Defect Prediction – Kaggle TPS S4 E3 (marzo 2024) [10] como caso de estudio. Para alcanzar dicho propósito se plantean los siguientes objetivos específicos::

1. Diseñar un flujo reproducible de ciencia de datos que abarque descarga, trazabilidad y versionado de los datos
2. Explorar y caracterizar el conjunto de datos sintéticos proporcionado por Kaggle mediante un análisis descriptivo y visual
3. Implementar técnicas de feature engineering y selección de variables que mejoren la capacidad predictiva sin sacrificar interpretabilidad.
4. Desarrollar y comparar distintos modelos predictivos incluyendo enfoques lineales, bagging (Random Forest) y gradient boosting (LightGBM/XGBoost), evaluándolos con validación cruzada estratificada y métrica AUC-ROC media por clase
5. Realizar fine tunning de los modelos mediante un ajuste de los hiperparámetros
6. Reflexionar críticamente sobre los resultados obtenidos, las limitaciones del dataset sintético y las líneas de mejora futura, dejando definida una hoja de ruta para trabajos posteriores. El objetivo no es, por tanto, alcanzar las mejores scores dentro del leaderboard si no realizar una comparación de diferentes modelos y técnicas de ML.

2 FUNDAMENTACIÓN TEÓRICA Y ESTADO DEL ARTE

2.1 Aprendizaje supervisado y clasificación multi-etiqueta

En este apartado, dedicado al aprendizaje supervisado aplicado a la detección de defectos en la industria siderúrgica, se ha descrito con detalle un marco de clasificación multi-etiqueta en el que, partiendo de la premisa de que una misma lámina de acero puede exhibir simultáneamente múltiples imperfecciones: rayaduras (scratches), manchas, deformaciones o suciedad, [6, 8] se pone de manifiesto la insuficiencia de los clasificadores multi-class tradicionales que obligan a asignar una única categoría predominante (o la etiqueta «sin defectos») a cada pieza inspeccionada [8]. De este modo, al permitir que cada instancia reciba varios rótulos extraídos de un conjunto predefinido de categorías, el enfoque multi-etiqueta introduce para cada etiqueta una variable binaria que indica la presencia o ausencia del defecto correspondiente y posibilita, por tanto, predecir combinaciones completas de fallas que resultan esenciales para la posterior toma de decisiones correctivas en planta, tal y como se subraya en la literatura de aprendizaje multi-etiqueta. [11, 12, 8]

Históricamente, cuando los sistemas de inspección automatizada no contemplaban esta capacidad, la solución habitual consistía en agregar clases compuestas o genéricas; un ejemplo paradigmático es la categoría «Other_Faults» [1, 9], utilizada en conjuntos de datos industriales para agrupar láminas que no encajaban limpiamente en un único tipo de defecto o que presentaban varias anomalías simultáneas. Si bien esta estrategia permitía etiquetar de algún modo dichas piezas, su principal limitación residía en la vaguedad del resultado, pues no informaba sobre qué defectos concretos estaban presentes. La adopción de modelos multi-etiqueta solventa esta carencia al evitar categorías cajón de sastre y al posibilitar que, frente a una muestra ambigua, el sistema indique explícitamente, por ejemplo, Defectos = {Rayadura, Manchas}, proporcionando así una descripción más rica y accionable de la calidad del producto. [12]

Por otro lado, la relevancia práctica de la clasificación multi-etiqueta se ve reforzada por la posibilidad de explotar las correlaciones entre defectos, puesto que determinados fallos tienden a coexistir debido a causas comunes del proceso de laminado (una desalineación, por ejemplo, puede inducir simultáneamente rayaduras y abolladuras) y el conocimiento de tales co-ocurrencias , además de mejorar la capacidad predictiva del modelo al capturar patrones conjuntos, contribuye al diagnóstico de las causas raíz [8]. No obstante, incluso un enfoque multi-etiqueta sencillo que trate cada defecto de manera independiente aporta ya la ventaja fundamental de identificar todos los tipos de fallas presentes en una misma muestra y, en consecuencia, de reflejar con mayor fidelidad el estado real de cada producto, garantizando así un control de calidad integral en entornos manufactureros donde la presencia de defectos múltiples es la norma y no la excepción. [12]

2.2 Algoritmos relevantes

Se describen, a continuación, los algoritmos que, según la literatura reciente, resultan más adecuados para la clasificación multi-etiqueta de defectos en acero cuando se dispone de un conjunto de datos tabulares de tamaño amplio, manteniendo como premisa que, además de entrenar clasificadores específicamente diseñados para múltiples etiquetas, es viable adaptar algoritmos tradicionales de clasificación multi-clase a fin de predecir, de manera simultánea, más de un defecto:

2.2.1 Bosques aleatorios (Random Forest)

En primer lugar, los Bosques Aleatorios (Random Forest), constituidos por un conjunto de árboles de decisión generados a partir de subconjuntos aleatorios de muestras y atributos, han demostrado una elevada robustez frente al ruido y una notable capacidad para manejar características heterogéneas [3, 8, 9]; de hecho, al tratar

problemas multi-etiqueta, puede optarse por entrenar un bosque independiente por cada etiqueta o recurrir a variantes de árbol que admiten múltiples salidas, siendo esta última opción la que permite obtener, para cada defecto, una predicción binaria basada en criterios de división adaptados, por ejemplo, la entropía multi-etiqueta [8].

La eficacia de esta técnica queda reflejada en los trabajos de Nkonyana et al. [9], quienes informan de que Random Forest superó a SVM y redes neuronales en la clasificación de siete tipos de defectos, resultado atribuible a la naturaleza ensemble del método y a la reducción del sobreajuste que proporciona la selección aleatoria de características en cada árbol. [9]

2.2.2 Máquinas de Gradient Boosting

Por otro lado, las Máquinas de Gradient Boosting, representadas por implementaciones como XGBoost o LightGBM [13, 12, 6], se han popularizado gracias a su elevada precisión en datos tabulares y a la eficiencia computacional que ofrecen las versiones modernas [12]; estos algoritmos construyen secuencialmente árboles de decisión donde cada nuevo modelo corrige los errores del conjunto precedente y, en escenarios multi-etiqueta, suelen aplicarse bajo el esquema de binary relevance, entrenando un clasificador independiente para cada defecto [12].

No obstante, pese a tratar cada etiqueta de forma separada, su alto poder predictivo, combinado con la posibilidad de ajustar pesos o funciones objetivo para mitigar el desbalanceo de clases, los convierte en una opción competitiva en entornos industriales de gran volumen de datos; además, la estrategia de classifier chains permite captar parcialmente las dependencias entre etiquetas sin requerir modificaciones específicas del algoritmo de boosting. [12]

Investigaciones recientes (por ejemplo, el estudio de Agrawal y Adane [14] demuestran que un Random Forest estándar y una versión de este algoritmo optimizada mediante Principal Component Analysis (PCA) alcanzan alrededor del 76 % de exactitud, lo que respalda la eficacia de los ensambles de árboles en general y, por extensión, subraya el potencial de los métodos de gradient boosting, que pertenecen a la misma familia de enfoques. [14]

Finalmente, se destaca que la elección entre entrenar un único modelo capaz de emitir múltiples salidas o una colección de clasificadores independientes dependerá, en última instancia, de la magnitud del conjunto de datos disponible, del grado de desbalanceo presente y del impacto computacional que el diseñador de la solución esté dispuesto a asumir. [14]

2.2.3 Regresión logística (multi-etiqueta)

Dentro del conjunto de clasificadores lineales, la regresión logística resulta especialmente atractiva por su sencillez matemática y la interpretabilidad directa de sus coeficientes; en un contexto multi-etiqueta, la práctica habitual consiste en entrenar un modelo logístico binario por cada tipo de defecto, de forma que, para cada instancia, se obtiene una probabilidad asociada a la presencia de dicho defecto, asumiendo, como hipótesis de primer orden, la independencia estadística entre etiquetas. [8, 4]

Este planteamiento proporciona una línea base pertinente gracias a su rapidez de entrenamiento, a la facilidad para calibrar las probabilidades derivadas y a la posibilidad de ajustar los umbrales de decisión que determinan si una probabilidad estimada se traduce finalmente en la asignación de la etiqueta; no obstante, su naturaleza estrictamente lineal puede limitar la captación de relaciones más complejas entre atributos y defectos, especialmente cuando dichas relaciones presentan comportamientos no lineales pronunciados [4].

Aun así, en sistemas de inspección industrial, la logística puede integrarse como clasificador por defecto para cada etiqueta y combinarse posteriormente, mediante reglas de negocio o métricas de coste, con modelos no lineales que aporten poder predictivo adicional, preservando de esta forma la trazabilidad de las decisiones modeladas. [12]

2.2.4 Máquinas de Vectores de Soporte (SVM) multi-etiqueta

Las SVM han constituido un pilar clásico en la clasificación de defectos debido a su capacidad para explotar espacios de alta dimensión y definir fronteras de decisión con óptimo margen; sin embargo, al generar por defecto modelos binarios o multi-clase, la extensión a escenarios multi-etiqueta suele materializarse a través de la estrategia uno contra todos, entrenando una SVM independiente por defecto y obteniendo así una predicción binaria para cada etiqueta. [8, 6] Estudios específicos sobre el Steel Plates Faults dataset muestran que configuraciones one against one (OAO) logran precisiones próximas al 86 % lo que las sitúa en el rango alto de resultados publicados, si bien esta eficacia viene acompañada de un coste computacional que crece linealmente con el número de etiquetas. [4]

Existen formulaciones avanzadas, como Rank-SVM o enfoques de anotación extendida (extended decision label annotation), que tratan de incorporar dependencias inter-etiqueta, aunque su adopción industrial sigue siendo limitada por la complejidad de entrenamiento y la dificultad de parametrización. [11]

En la práctica, entrenar varias SVM binarias continúa resultando una opción viable cuando el número de defectos no supera la decena, siempre que se disponga de recursos de cómputo suficientes y se acepten las simplificaciones inherentes a la pérdida de información relacional entre etiquetas. [14]

2.2.5 k-Nearest Neighbors adaptado (ML-kNN)

El algoritmo de los k vecinos más cercanos ofrece una vía alternativa basada en aprendizaje perezoso, donde la predicción de cada muestra se apoya directamente en la distribución de etiquetas observada en su vecindad; la variante ML-kNN propuesta por Zhang y Zhou . [11] incorpora un tratamiento bayesiano que, para cada etiqueta, estima la probabilidad de pertenencia a partir del recuento de vecinos que poseen dicha etiqueta y de un término a priori. A diferencia del kNN convencional, que votaría por una única clase mayoritaria, ML-kNN calcula, etiqueta por etiqueta [4, 8], una puntuación que permite etiquetar simultáneamente múltiples defectos sin necesidad de entrenar clasificadores separados. [11]

Su implementación es conceptualmente sencilla y puede servir de referencia cuando el número de atributos es moderado y están correctamente normalizados; sin embargo, comparte con el kNN tradicional la sensibilidad a la dimensionalidad y a la escala de los atributos, lo que exige aplicar técnicas de reducción o estandarización previas para evitar la degradación del rendimiento. [11]

En manufactura tabular con combinaciones recurrentes de defectos, ML-kNN resulta útil para capturar patrones locales que otros métodos globales podrían obviar, aunque no figura entre los algoritmos más citados en la literatura reciente sobre placas de acero, motivo por el cual se recomienda considerarlo como complemento exploratorio más que como solución final única. [11]

2.2.6 Redes Neuronales (Perceptrón Multicapa)

Las redes neuronales artificiales de tipo perceptrón multicapa (MLP) constituyen una alternativa particularmente flexible para abordar la clasificación multi-etiqueta, dado que permiten incorporar de forma natural varias neuronas de salida con funciones de activación sigmoide y, por tanto, generar para cada defecto una probabilidad independiente comprendida entre 0 y 1.

En un conjunto de datos tabulares con 35 atributos y 7 etiquetas de defecto, una arquitectura formada por unas pocas capas densas es capaz de aprender relaciones no lineales complejas entre las variables de entrada y las etiquetas, así como de captar eventuales correlaciones entre ellas cuando se utiliza una función de pérdida conjunta, por ejemplo, la entropía cruzada binaria agregada sobre todas las etiquetas, que penaliza la predicción incorrecta del vector completo de salidas. [14]

Trabajos industriales previos, citados por [14], confirman la idoneidad de esta aproximación: Mary [8] informó de un acierto aproximado del 75 %, mientras que Zhang et al. [8] alcanzaron valores en torno al 77 % con arquitecturas densas comparables, resultados que, si bien no superan de forma clara a los obtenidos con SVM o árboles, evidencian la competitividad de las redes densas tradicionales [8]. La disponibilidad de frameworks modernos simplifica hoy la construcción y ajuste de modelos neuronales multi-etiqueta, integrando técnicas de regularización (dropout, normalización, etc.) para mitigar el sobreajuste típico de datasets estructurados de tamaño medio [8]; no obstante, debe señalarse que, a diferencia de árboles de decisión o regresiones logísticas, las redes neuronales se presentan como cajas negras, un factor a considerar en entornos industriales donde la interpretabilidad de las causas de un defecto posee un valor operativo relevante. [14]

2.2.7 Otros enfoques

Más allá de los algoritmos previamente descritos, existen estrategias específicas para la clasificación multi-etiqueta que merecen atención [4]. Entre las técnicas de transformación del problema, el enfoque Binary Relevance ,que consiste en entrenar un clasificador independiente por etiqueta, se ha convertido en estándar de facto por su simplicidad y por hallarse implícito en múltiples aplicaciones (por ejemplo, al desarrollar una SVM, un árbol o una red por cada tipo de defecto). En contraposición, los Classifier Chains, propuestos en 2010, buscan explotar las correlaciones entre etiquetas concatenando una secuencia de clasificadores en la que cada modelo incorpora como variables adicionales las predicciones generadas por los clasificadores anteriores; esta metodología puede mejorar la calidad predictiva cuando las dependencias inter-etiqueta son significativas, aunque requiere especial cuidado para evitar la propagación de errores a lo largo de la cadena [4]. Existen, asimismo, algoritmos que modifican el modelo base para manejar directamente conjuntos de etiquetas, como variantes de árboles de decisión que emplean medidas de impureza multi-label (p. ej., entropía conjunta) o métodos orientados a ranking, entre ellos Calibrated Label Ranking, que transforma la tarea en varios problemas de ordenación binaria [4]. En la práctica industrial de detección de defectos, la preferencia suele inclinarse hacia algoritmos estándar con ajustes mínimos, debido a su facilidad de implementación y mantenimiento; sin embargo, los avances en investigación amplían continuamente el repertorio de técnicas disponibles, ofreciendo oportunidades para incrementar la precisión y la robustez de los sistemas de predicción multi-etiqueta en escenarios productivos cada vez más exigentes. [12]

2.3 Trabajos previos y referentes en la literatura

Resumiendo el estado del arte sobre la detección automática de defectos en acero, y tomando como referencia exclusiva los trabajos previamente citados en la literatura, se ha llevado a cabo una síntesis cronológica que permite, introduciendo los resultados originales de cada autor (precisión alcanzada, algoritmo empleado y naturaleza de los datos), identificar la evolución metodológica desde los enfoques clásicos hasta las propuestas híbridas más recientes. [1] [4] [6] [8] [11]

En una primera etapa (2010-2015), caracterizada por la disponibilidad inicial de conjuntos públicos como Steel Plates Faults, se demostró que la aplicación de máquinas de soporte vectorial. por ejemplo, el estudio de Thirukovalluru et al. [22], con precisiones próximas al 75 % resultaba viable, aunque insuficiente para superar la barrera del 80 %;. De manera paralela, se ensayaron árboles de decisión C4.5 y redes neuronales de retropropagación Mary, [22] o feed-forward Zhang et al. [12], alcanzándose rangos de 75-79 %, lo que insinuaba

que la capacidad de los árboles para capturar no linealidades confería una ligera ventaja sobre otros métodos individuales. [22,12]

En una segunda etapa (2016-2020) dominada por los métodos ensemble, autores como Nkonyana et al. (2019) y Srivastava [21] verificaron que la combinación de clasificadores (principalmente Random Forest y AdaBoost) elevaba la robustez predictiva hasta valores en torno a 78-79 %, mientras que la inclusión de filtros de selección de características, tal como el mRMR empleado por Zhang et al. [12], permitía depurar variables irrelevantes y alcanzar incluso 79.3 % de acierto[12]; en este mismo periodo, las tentativas de optimización de algoritmos más simples Mohamed y Samsudin [9,21], corroboraron que, sin tales estrategias de mejora, su rendimiento permanecía relegado a márgenes de 66-72 %. [21]

Una tercera etapa (2020-2023) marcada por la irrupción del aprendizaje profundo en dominios tradicionalmente tabulares evidenció, mediante el uso de redes LSTM Agrawal y Adane [14], los modelos secuenciales alcanzaban rendimientos (≈75.6 %) comparables a los árboles y, por tanto, no justificaban la complejidad adicional salvo incremento sustancial de datos o atributos; simultáneamente, la propuesta de ensambles de árboles con reducción de dimensionalidad (PDTF e I-PDTF) sólo reportó mejoras discretas hasta 76.1 % [14]. Un aspecto singular de las investigaciones recientes es la problemática asociada a la clase Other_Faults: Shu et al. (2023) introdujeron la Extended Label Annotation (ELA), enriqueciendo la información de entrenamiento de un SVM y un árbol C4.5 y obteniendo ~77.5 %, lo que sugiere que refinar la representación de etiquetas heterogéneas puede mitigar el impacto del desbalance.

Finalmente, en la frontera actual del conocimiento, Yilmaz Eroglu y Guleryuz [14] presentaron un ensemble de árboles de decisión con regresión logística integrada (Logistic Model Tree Forest) que, tras aplicar un preprocesamiento ENN para equilibrar la clase minoritaria, estableció un nuevo máximo de 86.7 % de exactitud, superando en más de siete puntos porcentuales al Random Forest convencional (~79.5 %) [14] y evidenciando que la integración de clasificadores clásicos con técnicas de balanceo específicas constituye una vía eficaz para incrementar la precisión global.

En síntesis, y de forma análoga a lo observado en otros ámbitos de ingeniería, la progresión desde modelos sencillos hacia esquemas ensemble y estrategias híbridas ha permitido, mediante un análisis detallado de la naturaleza multi-etiqueta y el desbalance entre tipos de defecto, acercar el rendimiento de la clasificación de defectos en acero a niveles notablemente elevados, confirmando que la innovación sobre algoritmos clásicos, más que la adopción indiscriminada de redes profundas, resulta decisiva en escenarios de datos estructurados de tamaño moderado.

3 DESCRIPCIÓN DEL DATASET

3.1 Origen y fuentes de los datos (Competición Kaggle)

El conjunto de datos utilizado en el presente estudio procede íntegramente de la competición Steel Plate Defect Prediction perteneciente a la Tabular Playground Series – Season 4, Episode 3, organizada en la plataforma Kaggle; el reto, abierto entre el 1 y el 31 de marzo de 2024, fue concebido con el propósito de facilitar la práctica de técnicas de machine learning sobre datos tabulares de naturaleza sintética, garantizando al mismo tiempo la ausencia de restricciones de confidencialidad y la uniformidad de las condiciones de evaluación. [10]

3.2 Descripción de las variables y etiquetas de defecto

El fichero de entrenamiento, suministrado en formato CSV, consta de 19 219 registros × 35 columnas; una de dichas columnas corresponde a un identificador único (id), veintisiete representan características numéricas, entre las que se incluyen coordenadas, áreas y perímetros normalizados, transformaciones logarítmicas e índices derivados, y las siete restantes actúan como variables objetivo binarias, indicando la presencia o ausencia de cada tipo de defecto. El fichero de prueba mantiene la misma estructura de atributos (28 columnas, es decir, id más las 27 features), pero omite las etiquetas para permitir la inferencia ciega. Las clases de defecto está cada una registrada en una columna independiente cuya semántica se conserva íntegramente respecto de la literatura previa. Estas son [15]:

1. Pastry: Parche irregular y rugoso que rompe la uniformidad del brillo; suele formarse cuando escoria, cascarilla o salpicaduras metálicas se adhieren a la chapa aún caliente y dejan una "mancha pastosa" sin surco profundo.

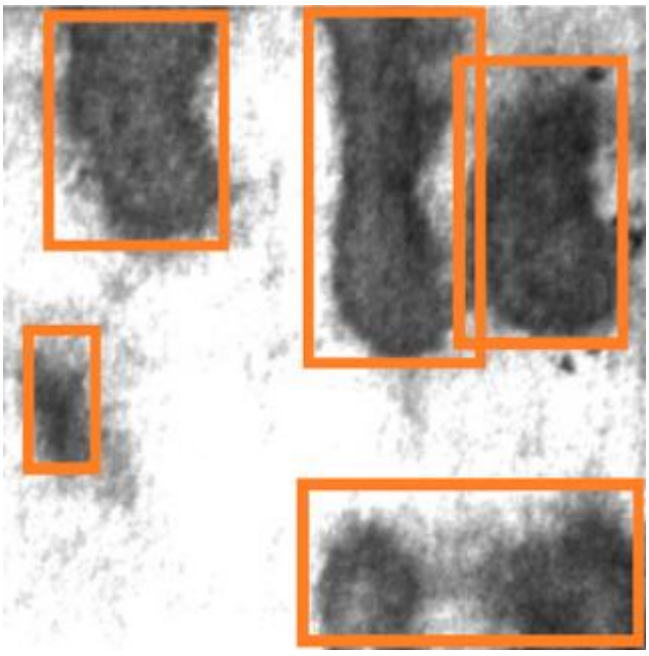


Figura 3-1. Parche rugoso (patch) análogo a un defecto Pastry: escoria adherida que rompe el brillo [20]

2. Z_Scratch: Arañazo lineal en tres tramos que dibuja la letra **Z**. Se produce cuando una partícula dura queda atrapada entre rodillo y placa y se arrastra en zig-zag durante el laminado.

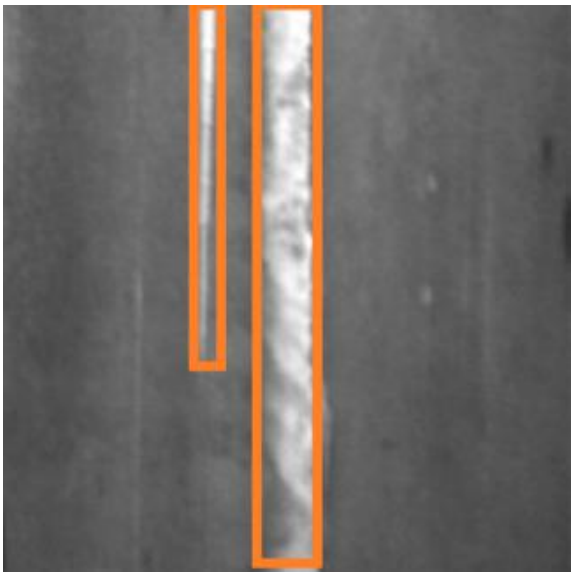


Figura 3-2. Arañazo en varios tramos, basta una zona con trazo en zig-zag para ilustrar Z_Scratch [20]

3. K_Scatch: Arañazo multisegmento con forma de **K**: un trazo principal y dos ramas oblicuas. Indica contactos repetidos con el mismo cuerpo extraño o rebabas sueltas en el transportador.

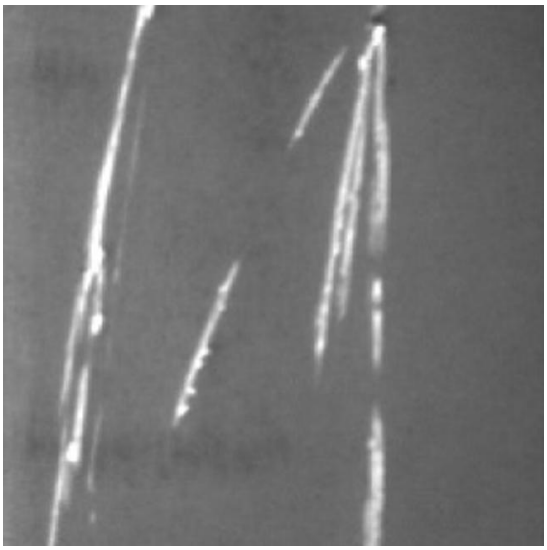


Figura 3-3. Conjunto de arañazos oblicuos que forman una K; corresponde a K_Scatch [3]

4. Stains: Manchas o decoloraciones sin relieve apreciable, fruto de oxidación puntual, restos de lubricante quemado o ataques químicos; alteran la reflectancia pero apenas modifican la geometría.

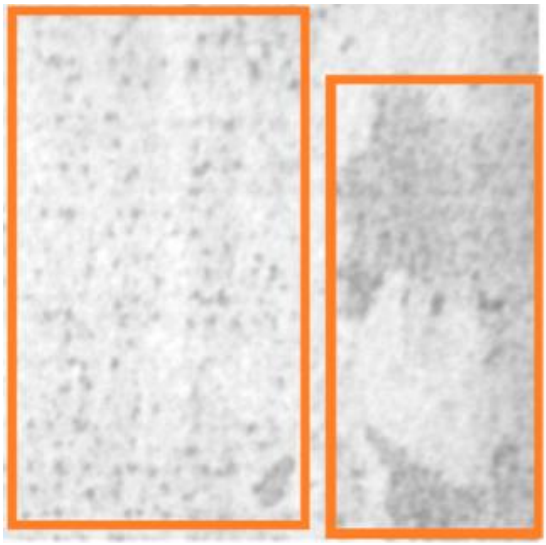


Figura 3-4. Mancha/discoloración superficial sin relieve visible, equivalente a Stains. [20]

5. Dirtiness: Depósitos difusos de polvo, calamina u óxidos sueltos que ensucian amplias zonas y reducen la luminosidad promedio; suelen deberse a una limpieza insuficiente entre fases de procesado

Figura 3-5. Depósitos difusos de calamina ejemplo real de superficie sucia, Dirtiness [2]

6. Bumps: Pequeñas protuberancias (bultos) que sobresalen de la superficie por inclusiones internas, chispas solidificadas o impactos de partículas calientes; geométricamente son elevaciones frente a los huecos de otros defectos.

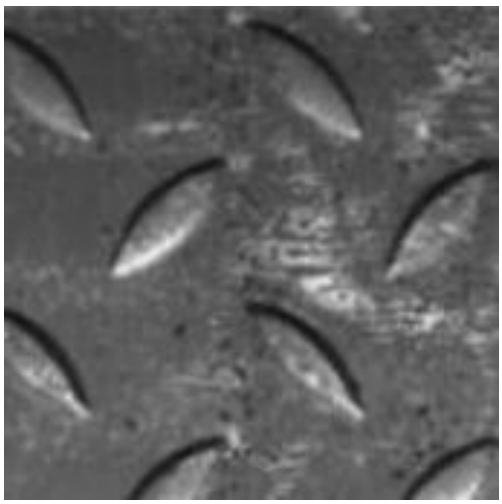


Figura 3-6. Protuberancias que sobresalen del plano, Ilustración de Bumps. [2]

7. Other_Faults: Categoría “cajón de sastre” para cualquier imperfección no incluida en las seis anteriores (p.ej. picaduras, microgrietas, abolladuras, exfoliaciones); por esa amplitud es la clase más frecuente del conjunto original.

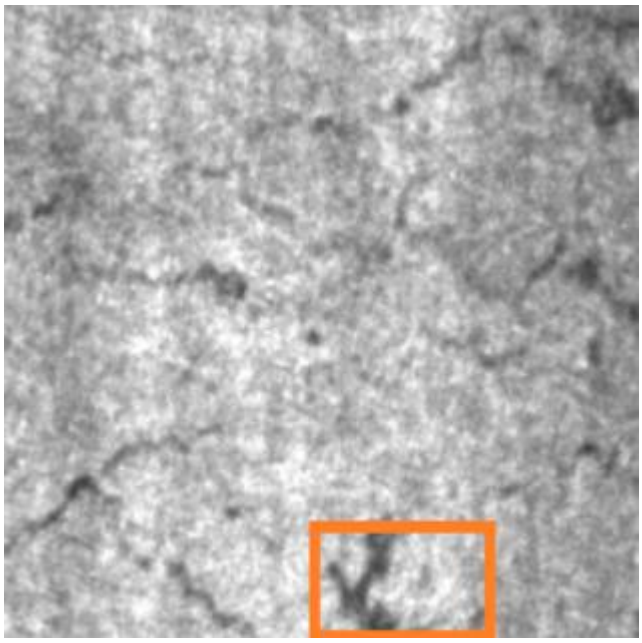


Figura 3-7. Imperfección variada (crazing) incluida en la categoría Other_Faults [2]

3.3 Naturaleza sintética del dataset y consideraciones especiales

De acuerdo con la información provista por el organizador, los registros fueron generados de forma sintética a partir de distribuciones inspiradas en inspecciones reales de placas de acero, lo que garantiza simultáneamente (i) la ausencia de datos confidenciales y (ii) un compromiso adecuado entre complejidad y ligereza ≈ 19 k observaciones para propósitos didácticos. No obstante, esta procedencia implica dos cuestiones clave:

- La transferencia potencialmente limitada a entornos industriales reales, donde las señales pueden presentar ruido adicional y patrones menos marcados.
- La necesidad de una interpretación cautelosa de modelos y transformaciones, dado que la "historia" física subyacente a cada feature no se encuentra documentada de manera explícita, extremo que obliga a fundamentar con detalle cualquier conclusión extraída a partir de los pesos o importancias obtenidos

3.4 Formato de los datos y requisitos de la competición

El formato original suministrado es un CSV con separador de coma, codificación UTF-8 y encabezados explícitos; para la fase de evaluación, la plataforma exige un archivo de envío que contenga la columna *id* seguida de las siete probabilidades estimadas (en idéntico orden a las etiquetas enumeradas), por ejemplo:

- id,Pastry,Z_Scratch,K_Scatch,Stains,Dirtiness,Bumps,Other_Faults

  19219,0.12,0.03,0.07,0.01,0.04,0.08,0.02
- Métrica oficial: media (macro) del AUC-ROC individual de cada una de las siete clases.

La métrica oficial utilizada para el ranking es la media macro del AUC-ROC calculada de forma individual para cada clase [10]. En el apartado 7.1 se explicará con mayor detalle en que consiste. Con este marco se delimitan las condiciones necesarias para el análisis exploratorio y la construcción del pipeline de machine learning que se detallarán en los capítulos posteriores.

4 ANÁLISIS EXPLORATORIO DE DATOS (EDA)

Código en notebook 01_eda.ipynb [22]. Este apartado se basa en el trabajo de [14, 15, 17]

4.1 Estadísticas descriptivas

En la fase de análisis exploratorio de datos (EDA) se ha elaborado, a partir de la totalidad de variables numéricas del conjunto de entrenamiento, un compendio de los principales estadísticos descriptivos: conteo de observaciones (N), media, mediana, valores mínimo y máximo, desviación típica, percentiles 25 % y 75 %, así como el rango intercuartílico (IQR), con el propósito de detectar de forma temprana posibles valores extremos, sesgos pronunciados en la distribución o discrepancias de escala que pudieran requerir un proceso posterior de normalización. La información resultante se ha sintetizado en una tabla que, mediante un código cromático (Rojos para desviación estándar superior a media, amarillos para mínimos y máximos elevados según regla de iqr y azul oscurto para iqr extremos comparados con el resto de variables) resalta aquellas columnas cuya dispersión relativa (por ejemplo, desviaciones estándar significativamente elevadas frente a la media) o cuyos valores mínimo y máximo pudieran sugerir errores de captura o la existencia de outliers.

	count	mean	std	min	25%	50%	75%	max	iqr
id	19219.000000	9609.000000	5548.191747	0.000000	4804.500000	9609.000000	14413.500000	19218.000000	9609.000000
X_Minimum	19219.000000	709.854675	531.544189	0.000000	49.000000	777.000000	1152.000000	1705.000000	1103.000000
X_Maximum	19219.000000	753.857641	499.836603	4.000000	214.000000	796.000000	1165.000000	1713.000000	951.000000
Y_Minimum	19219.000000	1849756.040012	1903553.850679	6712.000000	657468.000000	1398169.000000	2368032.000000	12987661.000000	1710564.000000
Y_Maximum	19219.000000	1846605.345439	1896295.137914	6724.000000	657502.000000	1398179.000000	2362511.000000	12987692.000000	1705009.000000
ixels_Areas	19219.000000	1683.987616	3730.319865	6.000000	89.000000	168.000000	653.000000	152655.000000	564.000000
X_Perimeter	19219.000000	95.654665	177.821382	2.000000	15.000000	25.000000	64.000000	7553.000000	49.000000
Y_Perimeter	19219.000000	64.124096	101.054178	1.000000	14.000000	23.000000	61.000000	903.000000	47.000000
_Luminosity	19219.000000	191846.678235	442024.694057	250.000000	9848.000000	18238.000000	67978.000000	11591414.000000	58130.000000
_Luminosity	19219.000000	84.808419	28.800344	0.000000	70.000000	90.000000	105.000000	196.000000	35.000000
_Luminosity	19219.000000	128.647380	14.196976	39.000000	124.000000	127.000000	135.000000	253.000000	11.000000
of_Conveyer	19219.000000	1459.350747	145.568687	1227.000000	1358.000000	1364.000000	1652.000000	1794.000000	294.000000
fSteel_A300	19219.000000	0.402674	0.490449	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000
fSteel_A400	19219.000000	0.596337	0.490644	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000
e_Thickness	19219.000000	76.213122	53.931960	40.000000	40.000000	69.000000	80.000000	300.000000	40.000000
Edges_Index	19219.000000	0.352939	0.318976	0.000000	0.058600	0.238500	0.656100	0.995200	0.597500
Empty_Index	19219.000000	0.409309	0.124143	0.000000	0.317500	0.413500	0.494600	0.927500	0.177100
quare_Index	19219.000000	0.574520	0.259436	0.008300	0.375750	0.545400	0.818200	1.000000	0.442450
ide_X_Index	19219.000000	0.030609	0.047302	0.001500	0.006600	0.009500	0.019100	0.665100	0.012500
ges_X_Index	19219.000000	0.614749	0.222391	0.014400	0.451600	0.636400	0.785700	1.000000	0.334100
ges_Y_Index	19219.000000	0.831652	0.220966	0.105000	0.655200	0.964300	1.000000	1.000000	0.344800
Global_Index	19219.000000	0.591899	0.482050	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000
LogOfAreas	19219.000000	2.473475	0.760575	0.778200	1.949400	2.227900	2.814900	4.554300	0.865500
.og_X_Index	19219.000000	1.312667	0.467848	0.301000	1.000000	1.146100	1.431400	2.997300	0.431400
.og_Y_Index	19219.000000	1.389737	0.405549	0.000000	1.079200	1.322200	1.707600	4.033300	0.628400
tation_Index	19219.000000	0.102742	0.487681	-0.988400	-0.272700	0.111100	0.529400	0.991700	0.802100
osity_Index	19219.000000	-0.138382	0.120344	-0.885000	-0.192500	-0.142600	-0.084000	0.642100	0.108500
noidOfAreas	19219.000000	0.571902	0.332219	0.119000	0.253200	0.472900	0.999400	1.000000	0.746200

Figura 4-1. Estadísticas descriptivas

Por otro lado, se ha determinado la distribución de las siete etiquetas binarias de defecto, Pastry, Z_Scratch, K_Scatch, Stains, Dirtiness, Bumps y Other_Faults, cuantificando tanto el número absoluto de piezas afectadas como su porcentaje relativo respecto al total de muestras; este balance de clases revela un desbalance notable, ya que ciertos defectos (Dirtiness o Stains) presentan una incidencia muy reducida frente a otros (Bumps u Other_Faults) relativamente más frecuentes. Por otro lado, se ha podido apreciar que el numéro de filas con más

de un defecto es prácticamente despreciable (menos de 30), es probable que esto se deba a que piezas irregulares con defectos múltiples se colocasen directamente en la etiqueta Other_faults en el dataset original abriendo la posibilidad de transformar el problema de multi-etiqueta a multi-clase si se eliminasen estas filas. Esta estrategia si bien es interesante y ha dado buenos resultados en el ranking, en la práctica son modelos menos generales y dependientes del contexto de la competición.

Tabla 4–1 Desbalance de clases

Tipo de defecto	positivos	Porcentaje (%)
Pastry	1,466	7.63
Z_Scratch	1,150	5.98
K_Scatch	3,432	17.86
Stains	568	2.96
Dirtiness	485	2.52
Bumps	4,763	24.78
Other_Faults	6,558	34.12

Finalmente, se ha analizado la repercusión directa de dicho desbalance en la estrategia de evaluación y entrenamiento, destacándose la conveniencia de implementar técnicas de mitigación específicas, ponderación de clases u oversampling con el fin de evitar que los modelos subsecuentes se inclinen hacia las clases mayoritarias y se preserve la representatividad de los defectos minoritarios. En conjunto, este diagnóstico preliminar proporciona una visión clara de la estructura global de los datos y sienta las bases para anticipar y corregir potenciales inconvenientes antes de la fase de modelado.

4.2 Visualizaciones de distribución y correlación

Posteriormente se ha elaborado visualizaciones, sobre la base del resumen estadístico previamente descrito, un conjunto de representaciones gráficas encaminadas a examinar, en primer lugar, la morfología univariante de las distribuciones, mediante histogramas con curvas de densidad y, en segundo término, la relación entre dichas variables y la presencia de cada defecto mediante diagramas de caja; finalmente, se ha generado un mapa de calor de correlaciones de Pearson. El principal objetivo es allanar el terreno para realizar con más información la ingeniería de features en el preprocesamiento.

4.2.1 Distribuciones univariantes

En esta primera fase de la visualización se han generado, para cada variable numérica del conjunto de entrenamiento, histogramas acompañados de curvas de densidad (KDE) que permiten discernir la forma de la distribución: simetría, sesgos a la derecha o a la izquierda, multimodalidades o colas larga (skew) y, al mismo tiempo, detectar acumulaciones de valores en los extremos que podrían revelar la presencia de outliers o límites técnicos (p. ej., saturación de sensores), de modo que la información obtenida orienta la eventual aplicación de transformaciones logarítmicas o escalados con el fin de mejorar la normalidad de los datos y homogeneizar sus rangos). A modo de ejemplo se muestran 3 gráficas descriptivas de la situación general aunque en la carpeta reports/visualizations [22] se pueden visualizar para todas las variables.

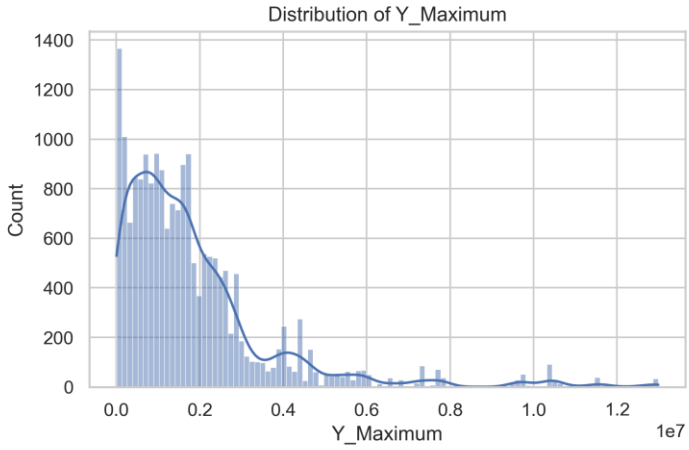


Figura 4-2. Distribución KDE de Y máximo.

La distribución de Y_Maximum muestra una alta concentración de registros en valores bajos y una cola prolongada hacia la derecha. La continuidad de los valores altos sugiere que no son errores de captura, sino defectos detectados lejos del origen de coordenadas; Dado que X y Y representan las coordenadas del bounding-box que encierra cada defecto (no las dimensiones completas de la placa), descartar estos valores extremos implicaría perder información valiosa sobre anomalías que ocurren en la zona final de la placa o sobre defectos excepcionalmente largos

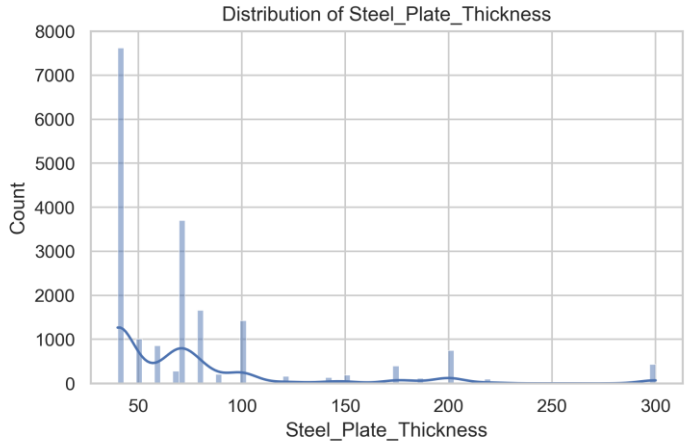


Figura 4-3. Distribución KDE del grosor de placa

Se puede apreciar una distribución multimodal con un alto grado de asimetría y discreta, probablemente esto último se deba a la utilización de espesores estándar en la producción de placas.

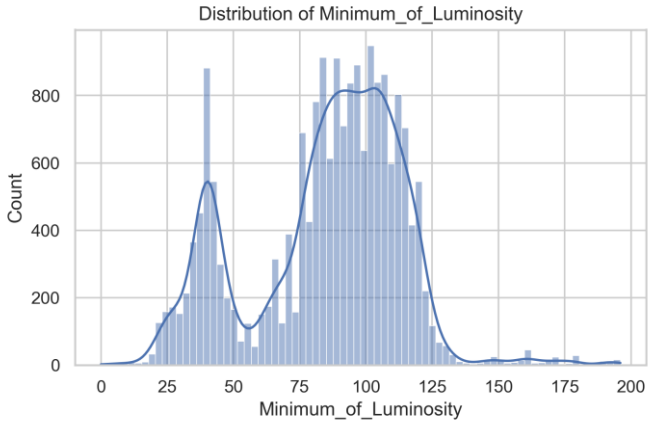


Figura 4-4. Distribución KDE del mínimo de luminosidad.

En esta ilustración se puede apreciar una distribución multimodal con dos picos claros, esto se puede deber a una convivencia de dos subpoblaciones o procesos diferentes, por ejemplo: distintos tipos de materiales o diferencias en las condiciones de medición en las placas de acero. Presenta además una pequeña asimetría con una pequeña cola hacia la derecha.

4.2.2 Boxplots por clase de defecto

A fin de explorar la relación entre cada variable y la presencia de los distintos defectos, se han construido diagramas de caja (boxplots) bajo la modalidad uno-versus-resto, comparando la distribución de cada característica entre piezas que presentan o no un determinado defecto; En estos gráficos se pueden apreciar, en algunas variables, desplazamientos significativos en la mediana (línea interior en la caja) y en el rango

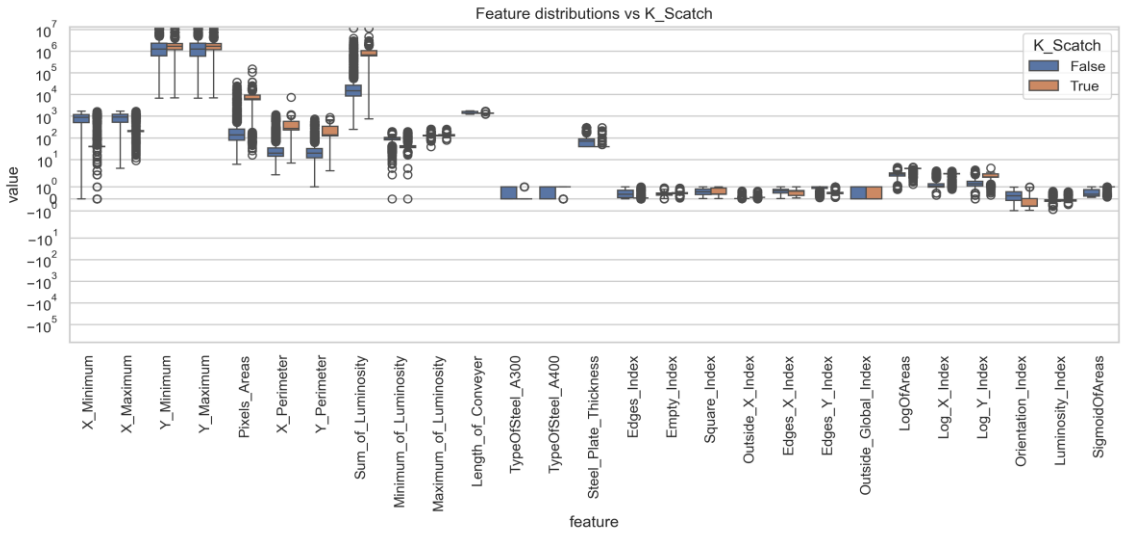


Figura 4-5. Boxplot por defecto (K-Scratch).

intercuartílico (línea superior e inferior de cada caja) con respecto al grupo sin defecto, lo que sugiere un potencial valor predictivo de estas variables y orienta tanto la selección de variables como el diseño de futuros modelos. Los círculos blancos representan valores que se encuentran fuera del rango intercuártilico

En este ejemplo los rectángulos naranjas muestran los valores de todas las variables cuando K-Scratch está presente y los azules cuando no, en este caso llama la atención dos cosas: por un lado, se puede apreciar un elevado número de outliers en diversas variables lo que sugiere que es necesario realizar transformaciones si se quieren implementar modelos lineales (esto algo que ya se había podido apreciar en los diagramas KDE). Por otro lado, hay ciertas variables cómo TypeOfSteel_A300 / A400 que muestran una separación casi perfecta cuando se presenta el defecto, por lo que con total seguridad serán relevantes para identificar el defecto.

4.2.3 Matriz de correlación

Finalmente, se ha elaborado un mapa de calor de correlaciones de Pearson que evidencia la existencia de varios pares de variables con coeficientes superiores a 0,8 en valor absoluto, indicativo de redundancia informativa, y por tanto justificativo de técnicas de reducción de dimensionalidad (PCA) o de regularización, así como de otras variables con correlaciones muy bajas que podrían aportar información singular o ruido; este análisis complementa los estadísticos numéricos y las visualizaciones univariantes anteriores, estableciendo una base sólida para las etapas subsiguientes de ingeniería de características y construcción de modelos.

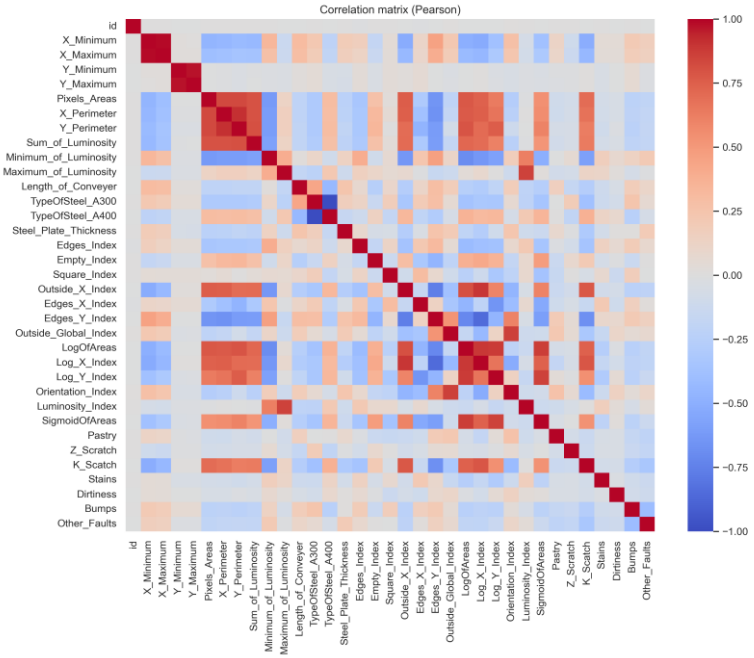


Figura 4-6. Correlación de las variables

Se puede apreciar una correlación casi perfecta entre las variables, X/Y_min con X/Y_max; Pixels_Areas, X_Perimeter y Y_Perimeter también están bastante correlacionadas (Áreas y perímetros crecen de forma conjunta). Por otro lado, Typeofsteel_A300 y A400 son complementarias (Azul oscuro) es necesario eliminar una (Para modelos que no soportan multicolinealidad perfecta cómo regresión logística es imprescindible)

4.3 Identificación de valores faltantes

No se ha identificado ningún valor faltante, si bien se ha decidido realizar una visualización para usos futuros y documentar la ausencia de impacto inmediato en la calidad de los datos. El diagnóstico de valores ausentes se ha abordado mediante la confección de una tabla que, para cada variable, detalla el número absoluto y el porcentaje de registros nulos respecto al total, a fin de priorizar aquellas columnas cuya carencia de información supere un umbral crítico y, llegado el caso, justificar su descarte o la aplicación de técnicas de imputación que permitan estimar los valores faltantes.

	n_missing	percent
id	0	0.0
X_Minimum	0	0.0
X_Maximum	0	0.0
Y_Minimum	0	0.0
Y_Maximum	0	0.0
Pixels_Areas	0	0.0
X_Perimeter	0	0.0
Y_Perimeter	0	0.0
Sum_of_Luminosity	0	0.0
Minimum_of_Luminosity	0	0.0
Maximum_of_Luminosity	0	0.0
Length_of_Conveyer	0	0.0
TypeOfSteel_A300	0	0.0
TypeOfSteel_A400	0	0.0
Steel_Plate_Thickness	0	0.0
Edges_Index	0	0.0
Empty_Index	0	0.0
Square_Index	0	0.0
Outside_X_Index	0	0.0
Edges_X_Index	0	0.0
Edges_Y_Index	0	0.0
Outside_Global_Index	0	0.0
LogOfAreas	0	0.0
Log_X_Index	0	0.0
Log_Y_Index	0	0.0
Orientation_Index	0	0.0
Luminosity_Index	0	0.0
SigmoidOfAreas	0	0.0
Pastry	0	0.0
Z_Scratch	0	0.0
K_Scatch	0	0.0
Stains	0	0.0
Dirtiness	0	0.0
Bumps	0	0.0
Other_Faults	0	0.0

Figura 4-7. Valores faltantes por features.

4.4 Detección de outliers

La detección de valores atípicos se ha efectuado aplicando la regla clásica basada en el rango intercuartílico (IQR = Q3- Q1), esto es, considerando outliers aquellos registros que se sitúan por debajo de Q1 – 1,5 × IQR o por encima de Q3 + 1,5 × IQR, lo que ha dado lugar a una tabla donde se consigna, por variable, el número de observaciones extremas y, con ello, se identifica cualquier columna con una concentración desproporcionada de valores anómalos, posible indicio de errores de medición, etiquetado incorrecto o condiciones de operación inusuales pero legítimas.

Tabla 4–2 Número de outliers

Tipo de defecto	positivos
X_Minimum	0
X_Maximum	0
Y_Minimum	1,118
Y_Maximum	1,112
Pixels_Areas	3,722
X_Perimeter	3,717
Y_Perimeter	2,785
Sum_of_Luminosity	3,826
Minimum_of_Luminosity	211
Maximum_of_Luminosity	1,292
Length_of_Conveyer	0
TypeOfSteel_A300	0
TypeOfSteel_A400	0
Steel_Plate_Thickness	2,173
Edges_Index	0
Empty_Index	61

Square_Index	0
Outside_X_Index	3,641
Edges_X_Index	0
Edges_Y_Index	25
Outside_Global_Index	0
LogOfAreas	421
Log_X_Index	3,325
Log_Y_Index	8
Orientation_Index	0
Luminosity_Index	999
SigmoidOfAreas	0

La regla del IQR es necesaria para identificar outliers, ya que, al basarse en los cuartiles de la propia distribución, se ajusta de manera más robusta a la dispersión real de los datos. Realizar, por ejemplo, un corte al 99 % (es decir eliminar aquellas variables que superen el umbral del 99 %) es arbitrario y puede ser demasiado estricto o demasiado laxo según la asimetría de cada variable: una variable con cola ligera tal vez no tenga valores por encima del percentil 99 que sean verdaderamente anómalos, mientras que otra muy sesgada podría contener outliers que caigan por debajo del umbral si solo mirásemos el p99.

4.5 Conclusiones clave del EDA

4.5.1 Ausencia de valores faltantes y limpieza previa

El dataset raw (19 219 filas × 35 columnas) no presenta nulos, por lo que no es necesaria ninguna etapa de imputación. Las variables objetivo binarias se quedaron intactas; el resto de features continuas requieren tratamiento de escala y outliers.

4.5.2 Heterogeneidad de escalas y asimetrías marcadas

Algunas variables (p. ej. perímetros, áreas, luminosidad) exhiben colas derechas muy largas y rangos que abarcan varios órdenes de magnitud. Para modelos lineales/redes, se aplicarán log1p, seguidas de escalado robusto. Para LightGBM o random Forest basta un capado ligero (winsorización P1–P99). En el preprocesamiento se verá con más detalle.

4.5.3 Presencia de outliers legítimos

Entre el 10 % y el 20 % de los valores de área, perímetro y luminosidad se encuentran fuera del rango intercuartílico típico. Sin embargo, estos valores extremos no son errores de sensores ni anomalías, sino que podrían corresponder precisamente a defectos inusualmente grandes o brillantes, es decir, los casos que el modelo debe aprender a detectar. Por tanto, no se eliminarán. En su lugar, se aplicarán transformaciones suaves

(logaritmo + winsorización) para atenuar su impacto en el modelo sin perder la información útil que contienen.
.

4.5.4 Desbalance multilabel y su impacto

La prevalencia varía de un 34 % (Other_Faults) a apenas un 2,5 % (Dirtiness). Aunque el AUC-ROC no se ve afectado directamente por el desbalance, la métrica del concurso, la media de AUC por clase, obliga al modelo a tratar también a las clases minoritarias, lo que plantea un reto adicional de aprendizaje y estabilidad. .

4.5.5 Grupos de features redundantes

- **Coordenadas** (X_Minimum–X_Maximum, Y_Minimum–Y_Maximum, TypeOfsteel300, TypeOfsteel400) con correlación ~ $|r| \approx 1$, (Puede ser interesante la combinación en nuevas variables derivadas como Width, Height, X_Center, Y_Center.
- **Área–Perímetro–Luminosidad** altamente correlacionados →Puede ser interesante conservar versión *log* de área, un perímetro agregado y Mean_Luminosity = Sum / Area. Aunque eliminar variables puede mejora rendimiento de logreg, empeora ligeramente los otros modelos ya que todos realizan el mismo preprocesamiento

4.5.6 Señal principal según boxplots y correlaciones

El tamaño (área, perímetros) y la intensidad luminosa son los predictores más fuertes en casi todos los defectos, especialmente Bumps y Other_Faults. La posición relativa al borde aporta información en rasguños (K_Scatch, Z_Scratch): crear variables de distancia normalizada al borde. Clases minoritarias como Dirtiness y Stains solo muestran diferencias sutiles en índices y brillo máximo → aprovechar modelos no lineales y losses que favorezcan aprender esas distinciones.

4.5.7 Acciones de feature engineering derivadas

- Transformaciones de escala y winsorización para atenuar la asimetría sin perder los valores extremos.
- Nuevas variables:
    1. Width, Height, Aspect_Ratio, X_Center_norm, Y_Center_norm,
    2. Mean_Luminosity, Max_to_Mean_Luminosity,
    3. indicadores binarios de “muy grande” (por encima de P95).

4.5.8 Preparación para modelado

- Montar un ColumnTransformer reproducible que incorpore todas las transformaciones anteriores.
- Validar con k-fold estratificado multilabel, monitorizando AUC-ROC por clase.

Estas decisiones se documentan en detalle en el siguiente apartado para garantizar la reproducibilidad del preprocesamiento, y las variables imputadas se almacenan en la carpeta data/interim/ para su posterior reutilización. [14]

5 PREPROCESAMIENTO

El preprocesamiento de los datos es una fase crítica en todo proyecto de aprendizaje automático. Su objetivo es preparar los datos en un formato óptimo para el entrenamiento de modelos, minimizando sesgos, reduciendo la varianza y asegurando la reproducibilidad. En este proyecto, se ha diseñado un pipeline modular que encapsula todos los pasos previos al modelado, facilitando su mantenimiento y evaluación. [14, 15, 16, 17]

5.1 Limpieza y transformación de los datos

Para asegurar la integridad de los datos y prepararlos adecuadamente para el modelado, se diseñó un pipeline con las siguientes transformaciones (en los siguientes apartados se detallarán en profundidad):

- Generación de nuevas características (FeatureGenerator)[16]: Este paso automatiza la creación de variables derivadas a partir de combinaciones útiles de columnas originales. Permite incorporar relaciones no lineales o proporcionales que pueden ser predictivamente relevantes.
- Eliminación de columnas redundantes (ColumnDropper)[16]: Se eliminaron variables altamente correlacionadas o redundantes, como Sum_of_Luminosity o Edges_X_Index, que no aportaban nueva información y podían inducir multicolinealidad.
- Winsorización y transformación logarítmica [16]: Para las variables más asimétricas (LOG_COLS), primero se aplicó una winsorización (limitación de valores extremos) para mitigar el impacto de outliers. Posteriormente, se aplicó una transformación logarítmica para reducir la asimetría y estabilizar la varianza.

Por último se creó un column transformer dónde se aplicó a cada variable la transformación (o no) que le corresponde en base a las gráficas obtenidas en el EDA.

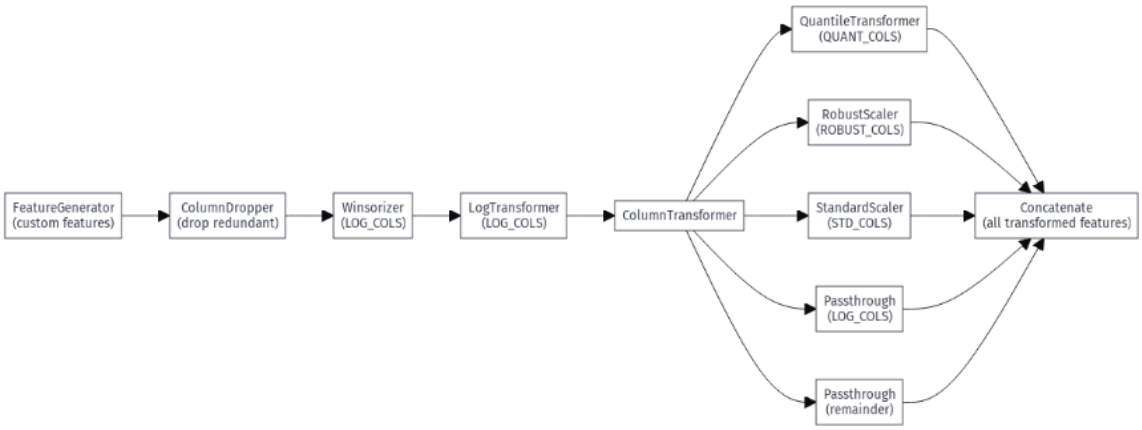


Figura 5-1. Diagrama de flujo del preprocesamiento

5.2 Ingeniería de características (feature engineering)

La ingeniería de características consiste en transformar o crear nuevas variables a partir de los datos existentes, con el fin de capturar relaciones relevantes que los modelos puedan explotar.

5.2.1 Feature Generator

Este paso se encapsuló en el componente FeatureGenerator, una clase personalizada dentro del pipeline de preprocesamiento. Este transformador se encarga de crear nuevas features a partir de las ya existentes que podrían resultar clave a la hora de detectar defectos. Las diferentes features creadas son prácticas habituales en bounding-box representations [15]. Entre ellas se han decidido crear:

5.2.1.1 Cálculo de dimensiones geométricas

Se extraen las variables Width y Height a partir de las coordenadas mínimas y máximas de cada defecto. Estas dimensiones permiten capturar la forma básica del defecto. A partir de ellas se deriva también el Aspect_Ratio (ancho entre alto), útil para distinguir formas alargadas de formas más compactas.

5.2.1.2 Normalización de posiciones

Se generan las variables X_Center_norm y Y_Center_norm, que representan las coordenadas del centro del defecto normalizadas por el ancho y alto total de la imagen. Esta normalización permite que la posición sea comparable entre defectos, independientemente del tamaño absoluto de la imagen.

5.2.1.3 Relaciones de intensidad de luminosidad

Se calcula Mean_Luminosity como la media de luminosidad por píxel. También se introduce Max_to_Mean_Luminosity, una medida de contraste relativa que indica si hay un pico de intensidad en relación con el valor medio. Esta relación puede ser indicativa de ciertos tipos de fallos localizados.

5.2.1.4 Identificación de placas inusualmente grandes

Se añade la variable binaria is_very_large, que marca si el área (Pixels_Areas) supera el percentil 95 de toda la muestra. Este tipo de banderas puede ayudar a los modelos a distinguir casos extremos o anómalos.

	Width	Height	Aspect_Ratio	X_Center_norm	Y_Center_norm	Mean_Luminosity	Max_to_Mean_Luminosity	is_very_large
0	8	101	0.079208	0.319412	0.073237	117.263156	1.407092	0
1	147	46	3.195652	0.066176	0.033790	958.371427	0.132516	0
2	14	12	1.166667	0.930588	0.192910	97.390243	1.273228	0
3	13	12	1.083333	0.947353	0.064381	111.061727	0.855380	0
4	8	17	0.470588	0.540000	0.089781	123.953333	1.064917	0

Figura 5-2. Ejemplo de los 5 primeros valores de las nuevas variables

Se ha decido eliminar las siguientes columnas: *'X_Maximimum', 'X_Minimum', 'Y_Maximimum', 'Y_Minimum', 'Sum_of_Luminosity', 'Edges_X_Index', 'SigmoidOfAreas', 'Luminosity_Index', 'TypeOfSteel_A300'*.

Esta decisión se debe a la alta correlación con otras variables (ej. Edges_X_Index', correlación ~0.97 con Edges_Index) o por su poca relevancia en el resultado tras comprobar la importancia de cada feature.

5.3 Escalado de características

5.3.1 Winsorizer y LogTransformer

Además de FeatureGenerator, se utilizaron estos dos transformadores personalizados (sobre las mismas variables) [16]:

- **Winsorizer**: Transformador que mitiga el impacto de valores extremos (outliers) en las columnas más sesgadas, limitando sus valores entre los percentiles 0.1 y 99.9. Esto evita que valores atípicos dominen el comportamiento de los modelos sensibles a escalas o varianzas. Su principal ventaja es que mantiene las filas con outliers sin comprometer las estimaciones de los modelos.

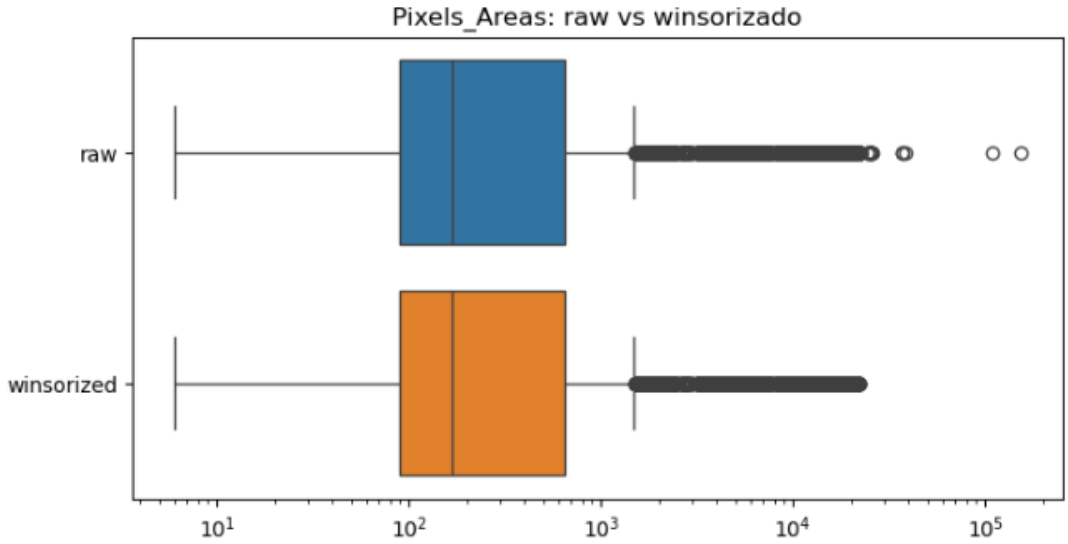


Figura 5-3. Winsorizado de Pixels_Areas

- **LogTransformer**: aplica una transformación logarítmica (log1p para evitar los valores nulos). a columnas altamente asimétricas. Esto reduce la dispersión y mejora la linealidad de las relaciones con las variables objetivo, especialmente útil en técnicas lineales o redes neuronales.

$$\text{log1p} = \log(x + 1)$$

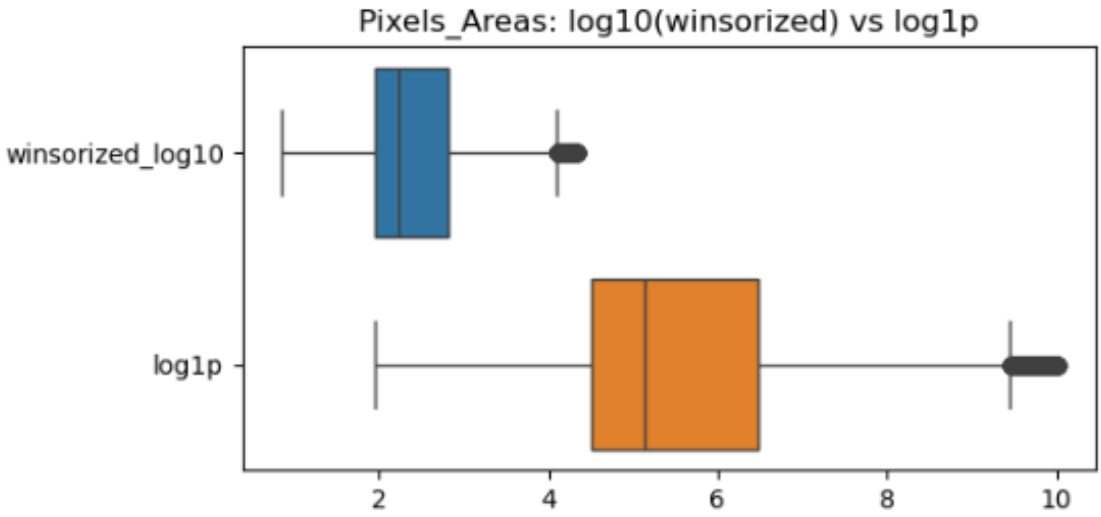


Figura 5-4. Winsorizado y transformación logarítmica de Pixel_Areas

Todas estas transformaciones se aplicaron sobre los siguiente atributos: *'Max_to_Mean_Luminosity', 'Mean_Luminosity', 'Outside_X_Index', 'Pixels_Areas', , 'Sum_of_Luminosity', 'X_Perimeter', 'Y_Center_norm', 'Y_Perimeter'.*

5.3.2 Column transformer

Una vez generadas y transformadas las variables más asimétricas y con más outliers, es fundamental escalar los valores de las demás variables antes de alimentar los modelos. Esto se debe a que muchos algoritmos de aprendizaje automático (como regresión logística, SVMs o redes neuronales) son sensibles a las escalas relativas de las variables. Para abordar esta tarea de forma modular y controlada, se empleó un ColumnTransformer [16] que permite aplicar distintos métodos de escalado a diferentes subconjuntos de columnas, dependiendo de su distribución estadística, sensibilidad a outliers y rol en el modelo.

A continuación, se detallan los grupos de variables y las razones por las que se aplicó cada técnica de escalado:

5.3.2.1 QuantileTransformer (QUANT_COLS)

Variables: Edges_Index, Edges_Y_Index, Height, Length_of_Conveyer, Width, Aspect_Ratio

Estas variables presentan distribuciones muy sesgadas o con colas pesadas, pero sin valores extremos atípicos aislados. Para normalizarlas sin afectar su orden relativo, se aplicó el QuantileTransformer [16] con salida de distribución normal. Este método es especialmente útil porque:

- Convierte cualquier distribución en una distribución aproximadamente gaussiana.
- Respeta el orden de los datos originales (es una transformación monótona).
- Mejora el rendimiento en modelos que asumen normalidad (por ejemplo, regresión logística o redes con activaciones simétricas).

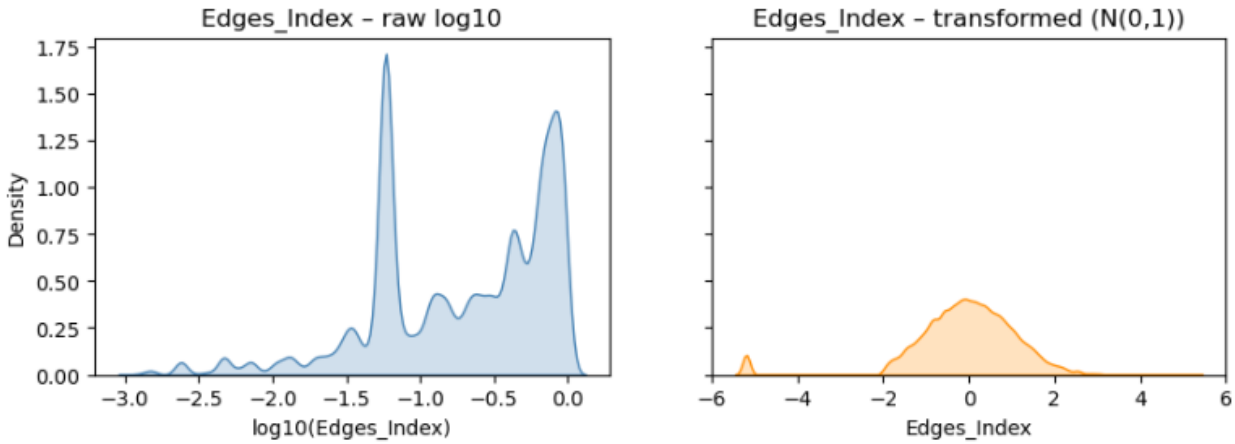


Figura 5-5. Transformación quantílica de Edges_index

5.3.2.2 RobustScaler (ROBUST_COLS)

Variables: LogOfAreas, Log_X_Index, X_Center_norm, Y_Center_norm

Estas variables, aunque ya transformadas (logarítmicamente o generadas), siguen presentando valores extremos o outliers. El RobustScaler [16] se basa en la mediana y el rango intercuartílico (IQR), lo que lo hace resistente a valores atípicos. Es una opción preferente cuando:

- No se quiere perder la estructura original (como con recortes o cuantiles).
- Se espera que las colas extremas contengan información, pero no se desea que dominen la escala.

$$X_{\text{scaled}} = \frac{X - \text{Median}(X)}{\text{IQR}(X)}$$

5.3.2.3 StandardScaler (STD_COLS)

Variables: Empty_Index, Log_Y_Index, Minimum_of_Luminosity, Orientation_Index, Outside_Global_Index, Square_Index, , Steel_Plate_Thickness

Estas variables no mostraban ni una fuerte asimetría ni outliers destacados. Por tanto, se escalan de forma clásica (media 0, varianza 1) para asegurar una magnitud comparable entre ellas [16]. Esto es especialmente relevante cuando:

- Se combinan con otras variables en modelos lineales.
- Se quiere evitar que variables con mayor varianza dominen la optimización del modelo.

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}$$

5.3.2.4 Passthrough para variables log-transformadas (LOG_COLS)

Variables como Pixels_Areas, X_Perimeter, Maximum_of_Luminosity, Outside_X_Index, entre otras, ya habían sido transformadas con log1p tras aplicar winsorización. Al haber reducido su rango y asimetría, se decidió no escalarlas de nuevo, manteniéndolas con passthrough [16]. Esta decisión evita aplicar transformaciones redundantes y posibles distorsiones innecesarias. Para las variables no mencionadas en columna transformer se ha colocado un remainder: "passthrough" por lo que tampoco serían transformadas.

5.4 Pipelines y reproducibilidad

Todo el proceso de preprocesamiento ha sido encapsulado en un único pipeline [16]. Esto tiene varias ventajas:

- Reproducibilidad: garantiza que los mismos pasos se apliquen en entrenamiento, validación y test, evitando fugas de información.
- Modularidad: cada transformación es una etapa independiente, lo que permite ajustes finos y tests por separado.
- Facilidad de integración: el pipeline se conecta fácilmente con el modelo final, permitiendo el uso conjunto en procesos de validación cruzada o exportación.

5.5 Manejo del desbalance de clase

5.5.1 Estratificación multilabel

Antes de entrenar se generan los pliegues de validación conservando simultáneamente la distribución de todas las etiquetas. De este modo, incluso las clases más raras están representadas en cada partición y se evita que algún pliegue se quede sin ejemplos positivos, lo que garantiza métricas comparables entre pliegues. Pesos de clase en el entrenamiento.

Para cada etiqueta se calcula un peso proporcional [16] a su rareza (cuanto menos frecuente, mayor peso) y se incorpora a la función de pérdida del modelo. Esto penaliza con más intensidad los errores en la clase minoritaria sin alterar las frecuencias reales, ayudando a que el algoritmo aprenda patrones útiles para detectar esos casos escasos.

5.5.2 Oversampling

Se ha implementado un oversampling opcional mediante RandomOversample() del módulo imbalanced-learn, solo sobre la parte de entrenamiento de cada pliegue, se duplican aleatoriamente las observaciones de las etiquetas más escasas (en este proyecto, Pastry y Dirtines ya que las clases minoritarias alcanzan un buen ROC-AUC, modelos como lgbm y logreg apenas mejoran un 0,001 y Random forest empeora ligeramente) hasta alcanzar un número fijo de positivos. Así se equilibra la proporción de clases que ve el modelo durante el ajuste, sin contaminar la validación.

6 MODELADO

En este capítulo se describe el código de (03_model_training.ipynb). Se ha implementado usando la información procedente y funciones procedentes de [13,16,17]

6.1 Flujo principal del trainer

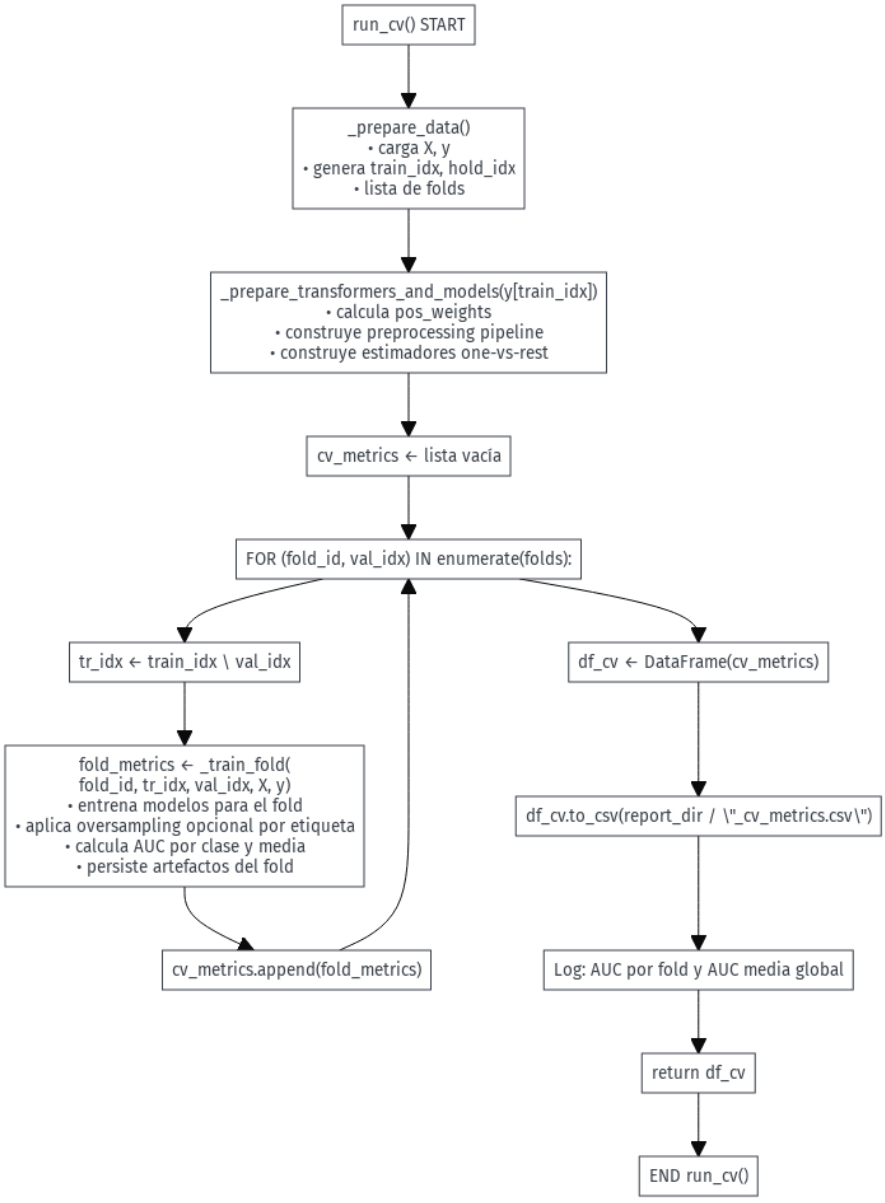


Figura 6-1. Diagrama de flujo del trainer

En los siguientes apartados se detalla cada una de las diferentes etapas principales del flujo con mayor detalle.

6.2 Particionado y preparación de datos (entrenamiento, validación, test)

6.2.1 Hold-out inicial para test final

Al inicio del proceso se aparta un subconjunto de observaciones que, por diseño, permanecerá reservado en exclusiva para la evaluación definitiva del modelo una vez concluidas tanto la fase de ajuste de hiperparámetros como la validación intermedia, de modo que dicho conjunto (denominado en lo sucesivo test final) no interviene en ninguna iteración de búsqueda ni en operación de validación cruzada, garantizando con ello que la métrica calculada sobre él refleje la capacidad real de generalización del algoritmo frente a datos absolutamente inéditos.

La segregación temprana de este test final elude cualquier filtración de información: cada decisión relativa a la arquitectura, a la sintonía de hiperparámetros o a las transformaciones aplicadas se adopta sin exposición previa a esos ejemplos, y de esta manera la evaluación post-entrenamiento carece de sesgo derivado del contacto previo con el conjunto reservado, proporcionando así una estimación imparcial y operacionalmente relevante sobre la idoneidad del modelo para su eventual despliegue o comparación con propuestas alternativas.

6.2.2 K-fold estratificado multilabel

Una vez excluido el test definitivo, el resto de los datos se destina a la fase combinada de entrenamiento y validación intermedia, para la cual se implementa una validación cruzada en k pliegues que, iterativamente, asigna uno de los pliegues al rol de validación mientras los restantes se emplean como entrenamiento; además, dado que la tarea es de naturaleza multilabel, es decir, cada registro puede portar simultáneamente varias etiquetas, la partición se diseña de manera que cada pliegue preserve, en la medida de lo estadísticamente posible, la distribución original de todas las etiquetas, incluidas las menos frecuentes, reproduciendo así la complejidad inherente al conjunto global. Se ha utilizado iterative-stratification 0.1.9 para la generación de los splits.

La estrategia de validación cruzada estratificada multilabel permite explotar al máximo la información disponible, obteniendo múltiples estimaciones de la métrica de interés y mitigando la dependencia de una sola partición; al asegurar que cada iteración contiene representantes de todas las etiquetas, se evita la ausencia o la sobre-representación de clases minoritarias, lo que se traduce en evaluaciones consistentes y mutuamente comparables entre pliegues y, en consecuencia, en una estimación más robusta de la capacidad de generalización del modelo durante el ajuste.

6.2.3 Fijar semillas y guardar splits

Se emplea una semilla aleatoria fija en todas las operaciones que implican estocasticidad, desde la partición inicial hasta cualquier procedimiento de muestreo o de transformación con componente aleatoria, de modo que, ante ejecuciones repetidas bajo las mismas condiciones y con la misma semilla, las divisiones en entrenamiento, validación y el test final se repliquen exactamente, lo que resulta esencial para la rastreabilidad y la reproducibilidad íntegra de los experimentos.

La reproducibilidad posibilita la comparación fidedigna de resultados entre ejecuciones, el intercambio transparente de experimentos con terceros y la verificación empírica de mejoras introducidas en el pipeline; al fijar la semilla se garantiza que cualquier divergencia observada provenga exclusivamente de modificaciones en el modelo, en el preprocesado o en la lógica de entrenamiento, y no de fluctuaciones inherentes a particiones distintas, de modo que, aun sin almacenar explícitamente los índices en archivos separados, la configuración de la semilla única permite regenerar de forma determinista los mismos splits en futuras ejecuciones.

6.3 Preparación de transformadores y modelos

En esta fase se establecen las bases que permiten convertir los datos crudos en una representación operativa óptima para el aprendizaje automático y, de forma paralela, se configuran los estimadores que abordarán de manera independiente cada una de las siete etiquetas de defecto incluidas en el problema. El objetivo último es garantizar que, a lo largo de todo el flujo de entrenamiento y validación, se preserve la reproducibilidad de los

pasos, se controle el desbalanceo inherente a las clases minoritarias y se mantenga una arquitectura modular que facilite futuras extensiones.

Para cada etiqueta binaria defecto frente a no defecto, se ha calculado un peso equilibrado positivo/negativo derivado de la distribución empírica observada en el conjunto de entrenamiento. Concretamente, para la etiqueta k se obtiene la relación mostrada en la ecuación correspondiente, empleándose un umbral mínimo ε que previene la asignación de pesos nulos, requisito indispensable para ciertos algoritmos gradientes (por ejemplo, LightGBM). De este modo:

- *Compensación del desbalance*: los defectos poco frecuentes reciben un factor de penalización superior, evitando la trivialización de la predicción.
- *Robustez frente a falsos negativos*: al ponderar con mayor severidad los errores en clases minoritarias, se refuerza la detección de defectos críticos.
- *Flexibilidad específica por etiqueta*: cada clase dispone de su propio coeficiente, ajustado a la prevalencia registrada.

$$w_{\text{neg}},\ w_{\text{pos}} = \text{compute\_class\_weight}(\text{"balanced"}, \{0,1\}, y^{(k)}) \quad \Longrightarrow \quad \text{peso}_k = \frac{w_{\text{neg}}}{w_{\text{pos}}}$$

6.3.2 Pipeline de preprocesamiento

Se construye un único objeto de tipo Pipeline que encadena, de modo reproducible y sin filtrado de datos de validación, todas las transformaciones necesarias sobre las características. El contenido y las trasformaciones se encuentran detallados en el apartado anterior. La importancia de integrar la pipeline de preprocesamiento radica en:

- *Prevención de fuga de datos (data leakage):* las estadísticas de transformación se ajustan sólo con el conjunto de entrenamiento de cada pliegue.
- *Reproducibilidad:* Le otorga al modelo la capacidad de ejecutar el pipeline completo (limpieza, ingeniería de características y escalado) de forma idéntica en cualquier fase o entorno, asegurando consistencia de resultados
- *Modularidad y limpieza:* cualquier cambio en el preprocesamiento se concentra en un único módulo.

6.3.3 Estimadores one vs rest

Considerando que una misma observación puede presentar simultáneamente más de un defecto (configuración multi-etiqueta) [16], se adopta la estrategia **"one-vs-rest"**. Así, se crea un clasificador independiente por etiqueta, al que se le suministran los pesos calculados en el epígrafe 7.3.1 como hiperparámetro ligado a la función de pérdida interna. Este planteamiento resulta especialmente pertinente porque:

- *Aísla la especialización por defecto,* permitiendo que cada modelo refine sus fronteras de decisión sobre un único tipo de anomalía;
- *Garantiza homogeneidad en la entrada*, toda vez que los siete modelos comparten el pipeline descrito en 7.3.2;
- *Facilita la extensibilidad*, reservando la posibilidad de incorporar, en el futuro, técnicas de muestreo o arquitecturas alternativas sin comprometer la integridad del esquema general.

En conjunto, esta preparación permite tratar de forma elegante el desbalance, garantizar la calidad de los datos y abordar la complejidad multi-label con una arquitectura clara y modular.

6.3.4 Integración en el pipeline de entrenamiento

La generación de particiones, tanto la separación hold-out inicial como los pliegues destinados a la validación cruzada, ha quedado plenamente integrada en el flujo automatizado de entrenamiento. Al iniciarse el proceso, se cargan los datos, se reserva el subconjunto de prueba definitivo y se construyen los índices correspondientes a cada pliegue. En la iteración propia del entrenamiento, se emplean exclusivamente los registros de aprendizaje y validación de la partición en curso, de modo que el conjunto de prueba permanece inaccesible hasta la evaluación final, garantizando así la objetividad de las métricas reportadas.

6.4 Algoritmos implementados:

6.4.1 Random Forest

Se ha utilizado el algoritmo Random Forest [3,16,17], técnica de ensamblado que, al combinar un elevado número de árboles de decisión independientes entrenados mediante Bootstrap: cada árbol se entrena con una versión diferentes de los datos permitiendo el remplazo (algunas filas se duplican y otras se eliminan de manera que cado uno es diferente a los anteriores), esto permite obtener un modelo robusto y con menor varianza que un árbol único. Específicamente, cada árbol se ha construido con una muestra aleatoria del conjunto de entrenamiento y, adicionalmente, con una selección aleatoria de características en cada nodo, lo que introduce la diversidad necesaria para que el voto mayoritario (en clasificación) o el promedio (en regresión) proporcione una predicción final más estable.

6.4.2 Gradient Boosting Machines (LightGBM)

Por otro lado, se ha implementado el método de Gradient Boosting [13,17], concretamente mediante la librería LightGBM. [19] A diferencia del enfoque anterior, los árboles se construyen de forma secuencial, de manera que cada nuevo árbol corrige los residuos (expresados como gradientes negativos de la función de pérdida) generados por el conjunto de árboles previo. El procedimiento iterativo se formaliza, para cada etapa m, a través de la optimización:

$$f_m(x) = \arg\min_f \sum_i \nabla_{y_i} L\left(y_i, \hat{y}_i^{(m-1)} + f(x_i)\right),$$

donde L es la función de pérdida y ∇_{y_i} su gradiente con respecto a la predicción anterior.

6.4.3 Regresión logística

La regresión logística es un método utilizado para modelar la probabilidad de que una variable de respuesta binaria (por ejemplo, defect = Yes/No) pertenezca a una categoría determinada, en función de una o más variables predictoras. [16,17]

A diferencia de una regresión lineal, que podría producir probabilidades negativas o mayores que uno (lo cual no tiene sentido en un contexto probabilístico), la regresión logística emplea la función logística, que asegura que las predicciones estén siempre entre 0 y 1:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

Esta función tiene una forma característica de "S", ajustándose de manera más natural al rango de probabilidades real y generando predicciones coherentes sin importar el valor de la variable explicativa.

Matemáticamente, la regresión logística modela la relación mediante el logaritmo de las odds (*log-odds* o *logit*):

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X$$

Existen diversas técnicas de penalización para evitar el sobreajuste del modelo. Tras una optimización de hiperparámetros se ha tomado l1 (lasso). Qué consiste en la suma de valores absolutos de los coeficientes con el objetivo de desincentivar modelos con valores complejos.

$$\sum_j |\beta_j|$$

6.4.4 Redes neuronales (Perceptrón Multicapa)

Finalmente, se ha implementado una red neuronal Perceptrón Multicapa (MLP), concebida como una arquitectura feed-forward fluye en una única con al menos una capa oculta. Cada neurona lleva a cabo una transformación lineal de sus entradas, seguida de una función de activación no lineal (ReLU o tanh), tal como se describe en la expresión [16,17]:

$$a^{(l)} = \varphi\left(W^{(l)}a^{(l-1)} + b^{(l)}\right)$$

donde W y b son los pesos y sesgos de la capa l y φ la función de activación seleccionada. El ajuste de los parámetros se ha efectuado mediante retropropagación y descenso de gradiente, empleando variantes como Adam, y se han incorporado técnicas de regularización con el fin de evitar el sobreajuste y garantizar la convergencia del modelo.

En conjunto, la integración de estos algoritmos ha permitido disponer de un abanico de enfoques complementarios, cuyos resultados se han comparado posteriormente en términos de capacidad predictiva y robustez, siguiendo la misma lógica analítica aplicada en el resto del estudio.

6.5 Implementación de la clasificación multi-etiqueta

En esta sección se comparan las dos aproximaciones consideradas para abordar la naturaleza multi-etiqueta del problema, a saber, la envoltura MultiOutputClassifier proporcionada por scikit-learn y el entrenamiento de clasificadores independientes, valorándose sus implicaciones prácticas en términos de implementación, flexibilidad y carga computacional. [16]

6.5.1 Opción A: MultiOutputClassifier

La clase MultiOutputClassifier [16] permite, en teoría, encapsular un único estimador base y aplicar de manera automática el ajuste (fit) y la predicción (predict) sobre cada etiqueta binarizada.

Esta decisión obedece a que, aun cuando la API resultante es extraordinariamente concisa (basta con una sola llamada a fit/predict), el esquema exige que todas las salidas compartan exactamente el mismo conjunto de hiperparámetros, lo que restringe la posibilidad de ajustar, por ejemplo, valores diferenciados de pos_weight por etiqueta o de ejecutar procesos de hyper-parameter optimisation (HPO) individualizados.

6.5.2 Opción B: Clasificadores independientes

La estrategia alternativa consiste en generar, para cada una de las siete etiquetas, un estimador autónomo (LightGBM, Random Forest, Logistic Regression o Multi-Layer Perceptron) a través de la función build_estimator() [16], que devuelve la citada lista de clasificadores; posteriormente, En Trainer.py se itera secuencialmente sobre ella. Este planteamiento, aunque conlleva un código ligeramente más extenso y obliga a agregar manualmente las salidas para la construcción del submission.csv, ofrece ventajas sustanciales:

- posibilidad de cargar y almacenar ficheros de parámetros óptimos específicos (best_params_<label>.json) por defecto.
- ajuste diferenciado de scale_pos_weight o class_weight en función de la prevalencia de cada defecto, y
- paralelización del entrenamiento sin dependencias cruzadas.

6.5.3 Justificación de la elección estándar

Finalmente, se ha adoptado la opción B como enfoque de referencia, atendiendo a los siguientes argumentos:

- LightGBM carece de soporte nativo para problemas multi-etiqueta, de modo que la capacitación de siete modelos independientes constituye la única vía factible para explotar sus prestaciones.
- El esquema propuesto otorga libertad para desbalancear cada salida de forma óptima y efectuar ajustes finos por clase, circunstancia imprescindible en un contexto donde la distribución de defectos presenta fuertes asimetrías.
- En síntesis, aun sacrificando cierta concisión en la implementación, la adopción de clasificadores independientes maximiza la flexibilidad hiperparamétrica, facilita la gestión del desbalanceo y conserva la posibilidad de paralelizar el flujo de entrenamiento, cualidades que resultan decisivas para satisfacer los requisitos de precisión y reproducibilidad establecidos en este trabajo.

7 OPTIMIZACIÓN

En este capítulo se describe el código de (04_hyperparameter_optimization.ipynb) basado en [16,17,18]

7.1 Métrica de optimización

Conviene precisar en qué consiste exactamente la ROC-AUC (Receiver Operating Characteristic - Area Under the Curve). [17] Para cada etiqueta binaria se construye la curva ROC trazando la tasa de verdaderos positivos (TPR = TP / [TP + FN]) frente a la tasa de falsos positivos (FPR = FP / [FP + TN]) mientras se barre todo el rango de umbrales posibles sobre la probabilidad estimada por el modelo. El área bajo esa curva se calcula como la integral definida de la función TPR(FPR) entre 0 y 1. Numéricamente suele aproximarse mediante la regla del trapecio y representa la probabilidad de que el clasificador asigne una puntuación mayor a una muestra positiva que a otra negativa tomada al azar. El valor resultante está acotado entre 0.5 (rendimiento aleatorio) y 1.0 (clasificación perfecta) y, a diferencia de la exactitud, permanece invariante ante desequilibrios de clase, motivo por el cual se adopta como métrica oficial de la competición y como criterio interno de optimización.

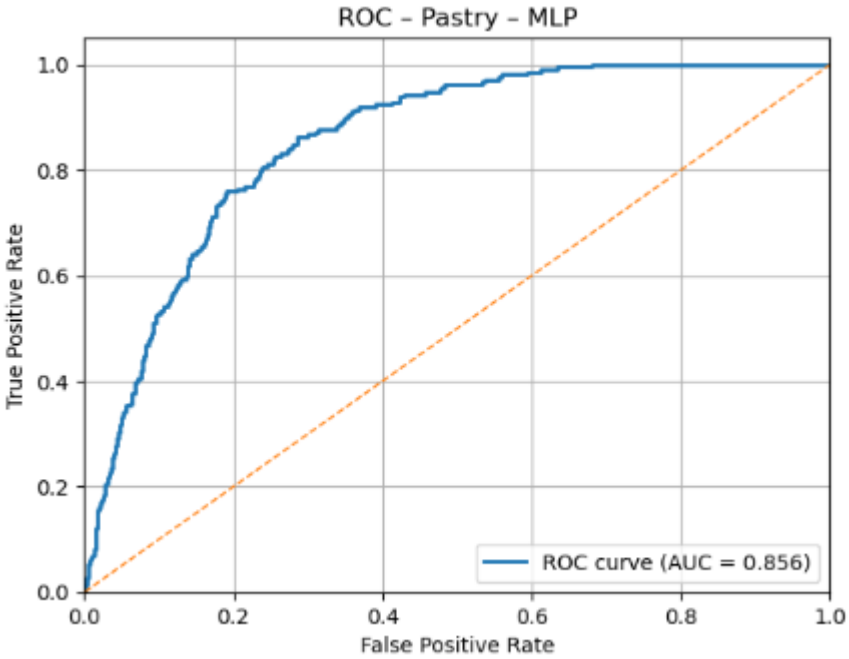


Figura 7-1. Area Under the ROC Curve de red neuronal Perceptrón Multicapa

Cualquier curva que se encuentre cerca de la recta amarilla discontinua no extrae señal. ROC-AUC es el área total por debajo la línea azul. Nota: Aunque permanezca invariante ante desequilibrios de clase el modelo en el entrenamiento sigue necesitando estrategias para manejar el desbalance. De no ser así se tendería a no predecir nunca defectos minoritarios (La métrica del concurso es la media de ROC-AUC).

7.2 Selección de parámetros iniciales

Los hiperparámetros son las diferentes configuraciones posibles con las que se puede entrenar un modelo. Por ejemplo: Número de árboles en un random forest, profundidad máxima del árbol, número de neuronas en una capa oculta mlp, etc. El rendimiento de un modelo puede variar en gran medida dependiendo de la selección de estos. Es por este motivo que realizar un ajuste es de vital importancia para garantizar buenos resultados.

El objetivo, por tanto, es explorar distintas combinaciones midiendo su rendimiento con un conjunto de validación. La idea principal es dividir los datos en k pliegues de manera que a la hora de probar valores se vayan rotando y cada Split participe una vez en la validación y en el resto de las iteraciones, en el entrenamiento. Es imprescindible no tomar los datos reservados para el test final para no provocar estimaciones engañosas y fugas de información. (validación cruzada)

Finalmente se obtendría la configuración que ofrezca el equilibrio óptimo entre precisión y capacidad de generalización.

Antes de la optimización fina (siguiente apartado) se definió un punto de partida reproducible para cada familia de modelos, inspirándonos en recomendaciones clásicas de la literatura y en buenas prácticas de sus implementaciones de referencia (scikit-learn 1.4 y LightGBM 4.3). [3,13,16,19]

Estos parámetros base se codifican en src/models/builders.py como base_params; si el experimento anterior dejó en disco un best_params_<model>.json, la función _load_best_params los sobreescribe dinámicamente, manteniendo la coherencia del circuito de experimentación.

Tabla 7–1. Hiperparámetros iniciales

Modelo	Parámetro	Valor inicial	Justificación
LightGBM (LGBMClassifier)	n_estimators	**500**	Compromiso entre sesgo y varianza con coste de cómputo aceptable.
	learning_rate	**0.05**	Recomendado por Ke et al. [13] como valor "seguro" en problemas tabulares medianos.
	num_leaves	**64**	Aproximadamente 2 × √(n_features) para evitar sobre-ajuste prematuro.
	max_depth	**-1**	Sin restricción; se regulariza vía *subsampling*.
	subsample, colsample_bytree	**0.8 / 0.8**	Bagging/grafting ligero que reduce varianza sin disparar el error de sesgo.
Random Forest (RandomForestClassifier)	n_estimators	**400**	Breiman [3] sugiere > 200 árboles para estabilizar OOB-error; 400 asegura convergencia con hardware actual.
	max_depth	**None**	Cada árbol crece hasta pureza; el ensemble limita el *overfitting*.
Regresión Logística (LogisticRegression)	penalty	**ℓ2**	Regularización estándar cuando n ≫ p.
	solver	**lbfgs**	Convergencia robusta en problemas multiclase y líneas

Modelo	Parámetro	Valor inicial	Justificación
			base grandes.
	max_iter	**2000**	Asegura convergencia tras estandarizar y *one-hot* sin penalizar excesivamente el tiempo.
MLP (Red Neuronal Multicapa)	hidden_layer_sizes	**(128, 64)**	Arquitectura "2-layer ReLU" habitual en tablas sintéticas de tamaño medio.
	activation	**relu**	Buen compromiso entre expresividad y tiempo de inferencia.
	solver	**adam** con learning_rate_init = 0.001	Ajuste canónico que converge rápido en datasets medianos.
	batch_size	**256**, max_iter = 200, alpha = 1e-4	Valores guía de scikit-learn para evitar *over-fitting* temprano.
	early_stopping	Sí (validation_fraction = 0.1, n_iter_no_change = 10)	Detiene el entrenamiento cuando no hay mejora significativa.

7.3 Tuning de hiperparámetros con Optuna

Entre las diferentes alternativas, la optimización bayesiana [17,18] es un método inteligente para encontrar los mejores hiperparámetros de un modelo de machine learning sin tener que probar todas las combinaciones posibles. En lugar de explorar de manera aleatoria o exhaustiva, este enfoque se basa en ir "aprendiendo" a partir de los resultados de los experimentos anteriores. Cada vez que se evalúa una combinación de hiperparámetros, el algoritmo actualiza su conocimiento sobre qué zonas del espacio de búsqueda parecen más prometedoras, concentrando los esfuerzos en aquellas configuraciones que tienen más probabilidades de mejorar el rendimiento.

Optuna, una de las librerías más utilizadas para este propósito, implementa esta optimización de forma muy flexible y automática. El proceso se organiza en ensayos llamados trials, donde cada uno corresponde a un conjunto específico de hiperparámetros que se prueban en el modelo. Tras cada *trial*, Optuna mide la calidad de la solución (por ejemplo, la ROC-AUC obtenida en validación cruzada) y utiliza un modelo probabilístico interno que estima qué combinaciones pueden producir resultados aún mejores. Este mecanismo permite que las siguientes pruebas no sean al azar, sino guiadas por la experiencia acumulada durante la búsqueda. [18]

Una ventaja importante de Optuna es que gestiona de forma automática el almacenamiento de los experimentos, la elección de las próximas configuraciones y el descarte anticipado de pruebas que no parecen prometedoras, una técnica conocida como poda temprana. Gracias a esta combinación de aprendizaje progresivo y eficiencia, se logra reducir significativamente el tiempo necesario para encontrar una configuración óptima. Además, su integración con scikit-learn y LightGBM facilita aplicarla en proyectos reales sin complicaciones adicionales. [18]

Para maximizar el promedio de ROC-AUC en las siete etiquetas, se diseñó un proceso de optimización basado en Optuna con los siguientes componentes:

Figura 7-2. Flujo principal de optimización

7.4 Función objetivo con validación multinivel

Con el objetivo prioritario de maximizar el valor medio de la métrica ROC-AUC correspondiente a las siete etiquetas previstas en la competición de Kaggle, se ha implementado un procedimiento de optimización asistido por Optuna que, introduciendo los parámetros semilla definidos en la sección 7.1 y manteniendo la coherencia con la pipeline de pre-procesado, permite obtener de manera automática y reproducible el conjunto de hiperparámetros que conduce al mejor desempeño del sistema de clasificación multilabel. Dicho procedimiento, estructurado en torno a una función objetivo con validación multinivel, calcula en cada ensayo la media de ROC-AUC macro sobre k particiones internas y, para ello, emplea MultiOutputClassifier en los casos de Random Forest y Regresión Logística (Ensemble pendiente: no se implementó stacking por límites de tiempo; estudios previos en competiciones similares sugieren mejoras de 0.002-0.005 AUC al combinar GBMs y redes tabulares. mientras que para LightGBM genera una lista de siete modelos independientes, preservando en ambos casos el esquema binary relevance ya adoptado en el resto del proyecto; además, cada iteración actualiza los pesos de clase (pos_weights), construye el estimador mediante build_model_from_trial, aplica la canalización común de pre-procesado y devuelve el valor de la métrica objetivo, lo que garantiza la trazabilidad completa del experimento.

7.5 Gestión de resultados

En cuanto a la gestión persistente de los resultados, cada estudio se almacena automáticamente en una base SQLite (hpo_<model>.db), lo que facilita tanto la reanudación de la búsqueda como el análisis posterior de las trayectorias de optimización; al concluir, los mejores hiperparámetros se exportan en ficheros best_params_<model>.json, sin prefijos ni estructuras adicionales, de modo que el módulo builders.py los pueda cargar directamente en la fase de entrenamiento definitiva, cerrando así un flujo end-to-end en el que la intervención manual se reduce a la mera definición de los rangos iniciales y al análisis interpretativo de los resultados. En consecuencia, la búsqueda de hiperparámetros queda plenamente automatizada, alineada con los objetivos de maximización de la métrica multilabel y respaldada por una trazabilidad que garantiza la reproducibilidad de la investigación.

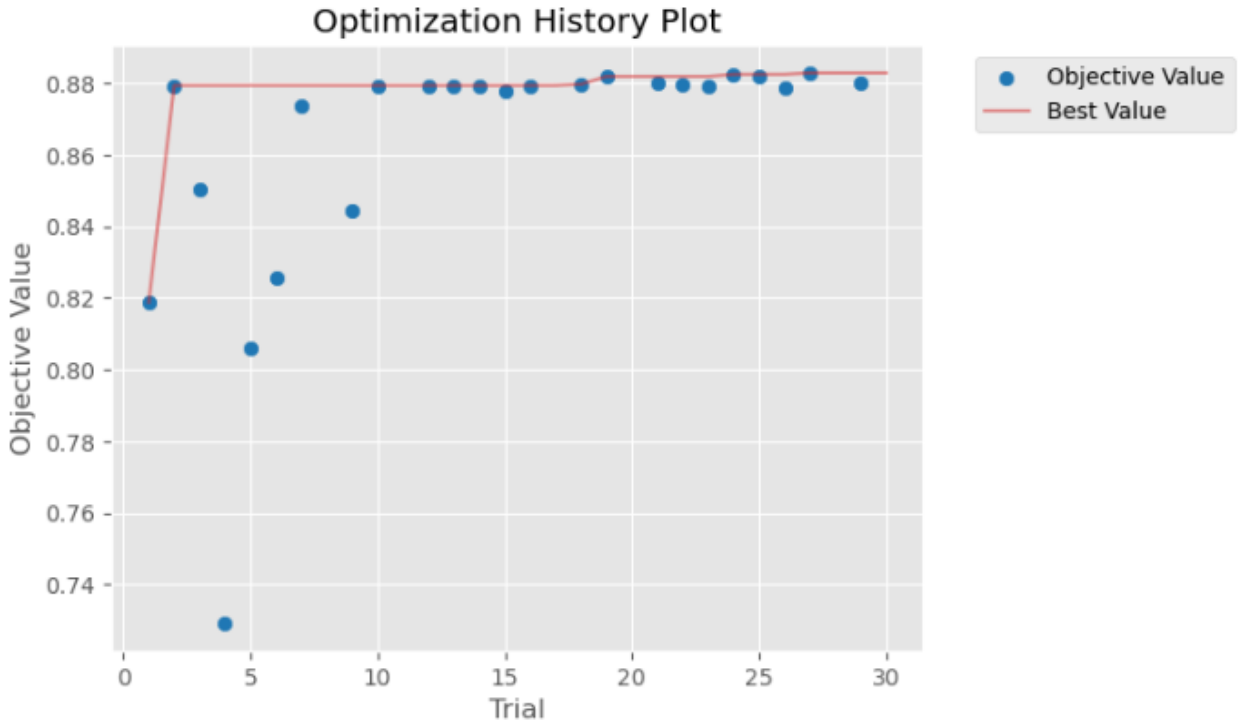


Figura 7-3. Evolución del ROC-AU por cada trial en lgbm

La gráfica representa el "Optimization History Plot" que genera Optuna para visualizar cómo evoluciona la búsqueda de hiperparámetros:

- Eje X (Trial): número de ensayo; cada trial corresponde a un conjunto distinto de hiperparámetros que se entrena y evalúa.
- Eje Y (Objective Value): valor de la métrica objetivo (en tu caso, la media de AUC-ROC) que se obtiene tras validar ese trial.
- Puntos azules: rendimiento individual de cada ensayo.
- Línea roja: el mejor valor alcanzado hasta ese momento (se actualiza si algún punto supera al anterior).

Se puede observar un arranque exploratorio: en los tres o cuatro primeros ensayos los resultados varían bastante (de ~0.73 a ~0.88). Esto refleja la fase inicial, donde el muestreador prueba combinaciones más dispersas para aprender el terreno.

Mejora rápida y estabilización: a partir del trial 2–3 se descubre un conjunto de hiperparámetros con AUC ≈ 0.88. La línea roja se eleva casi al tope y, desde entonces, apenas mejora unas centésimas: el algoritmo ya explora alrededor de esa zona prometedora.

Plateau: del ensayo 10 en adelante todos los puntos oscilan muy cerca del 0.88. Significa que la búsqueda ha convergido; el modelo no encuentra configuraciones mucho mejores y solo afina detalles menores.

En conjunto, la curva muestra que Optuna:

Encuentra una buena solución muy pronto (eficiencia de la optimización bayesiana) y utiliza las pruebas posteriores para confirmar y pulir ese rendimiento, manteniendo la estabilidad sin grandes saltos.

8 EVALUACIÓN

En este apartado se se describe el código de (05_feature_importance. ipynb) basado en [13,16,19]

8.1 Importancia de variables

LightGBM evalúa la importancia de cada feature principalmente según la ganancia acumulada que proporciona esa variable al dividir nodos en los árboles durante el entrenamiento. Se ha guardado durante el entrenamiento las variables que se usaron junto a una suma de las ganancias de esa variable (posteriormente normalizadas) [19]

Tabla 8–1 Variables más importantes en el defecto Pastry

Nº	Variable	Importancia
1	Length_of_Conveyer	0.057
2	Minimum_of_Luminosity	0.055
3	Empty_Index	0.055
4	Edges_Index	0.053
5	Max_to_Mean_Luminosity	0.050
6	Outside_X_Index	0.050
7	Orientation_Index	0.048
8	Mean_Luminosity	0.045
9	Square_Index	0.039
10	Log_Y_Index	0.038

Tabla 8–2 Variables más importantes en el defecto Z_Scratch

Nº	Variable	Importancia
1	Steel_Plate_Thickness	0.077
2	Empty_Index	0.073
3	Length_of_Conveyer	0.059
4	Minimum_of_Luminosity	0.050
5	X_Maximum	0.050
6	Edges_Index	0.043
7	X_Minimum	0.042

8	X_Center_norm	0.042
9	Orientation_Index	0.039
10	Outside_X_Index	0.038

Tabla 8–3 Variables más importantes en el defecto K_Scratch

Nº	Variable	Importancia
1	Minimum_of_Luminosity	0.056
2	Empty_Index	0.054
3	Edges_Y_Index	0.049
4	Maximum_of_Luminosity	0.048
5	Length_of_Conveyer	0.045
6	X_Center_norm	0.044
7	Max_to_Mean_Luminosity	0.042
8	Y_Center_norm	0.041
9	Log_Y_Index	0.041
10	Edges_Index	0.038

Tabla 8–4 Variables más importantes en el defecto Stains

Nº	Variable	Importancia
1	Steel_Plate_Thickness	0.096
2	Minimum_of_Luminosity	0.076
3	LogOfAreas	0.067
4	Log_Y_Index	0.056
5	Maximum_of_Luminosity	0.053
6	Edges_Index	0.052
7	Length_of_Conveyer	0.051
8	Y_Perimeter	0.050
9	Pixels_Areas	0.049
10	Aspect_Ratio	0.042

Tabla 8–5 Variables más importantes en el defecto Dirtiness

Nº	Variable	Importancia
1	Steel_Plate_Thickness	0.092
2	Orientation_Index	0.085
3	Edges_Index	0.072
4	Minimum_of_Luminosity	0.070
5	Aspect_Ratio	0.055
6	Length_of_Conveyer	0.052
7	X_Center_norm	0.040
8	X_Maximum	0.040
9	X_Minimum	0.039
10	Square_Index	0.037

Tabla 8–6 Variables más importantes en el defecto Bumps

Nº	Variable	Importancia
1	Steel_Plate_Thickness	0.069
2	Length_of_Conveyer	0.062
3	Empty_Index	0.054
4	Aspect_Ratio	0.046
5	Minimum_of_Luminosity	0.046
6	Y_Center_norm	0.043
7	Edges_Index	0.042
8	Mean_Luminosity	0.041
9	Outside_X_Index	0.040
10	X_Center_norm	0.039

Tabla 8–7 Variables más importantes en el defecto Other_Faults

Nº	Variable	Importancia
1	Empty_Index	0.058
2	Y_Center_norm	0.054
3	Max_to_Mean_Luminosity	0.052
4	Minimum_of_Luminosity	0.052
5	Edges_Index	0.049
6	Mean_Luminosity	0.048
7	Steel_Plate_Thickness	0.043
8	Aspect_Ratio	0.042
9	Length_of_Conveyer	0.042
10	Orientation_Index	0.041

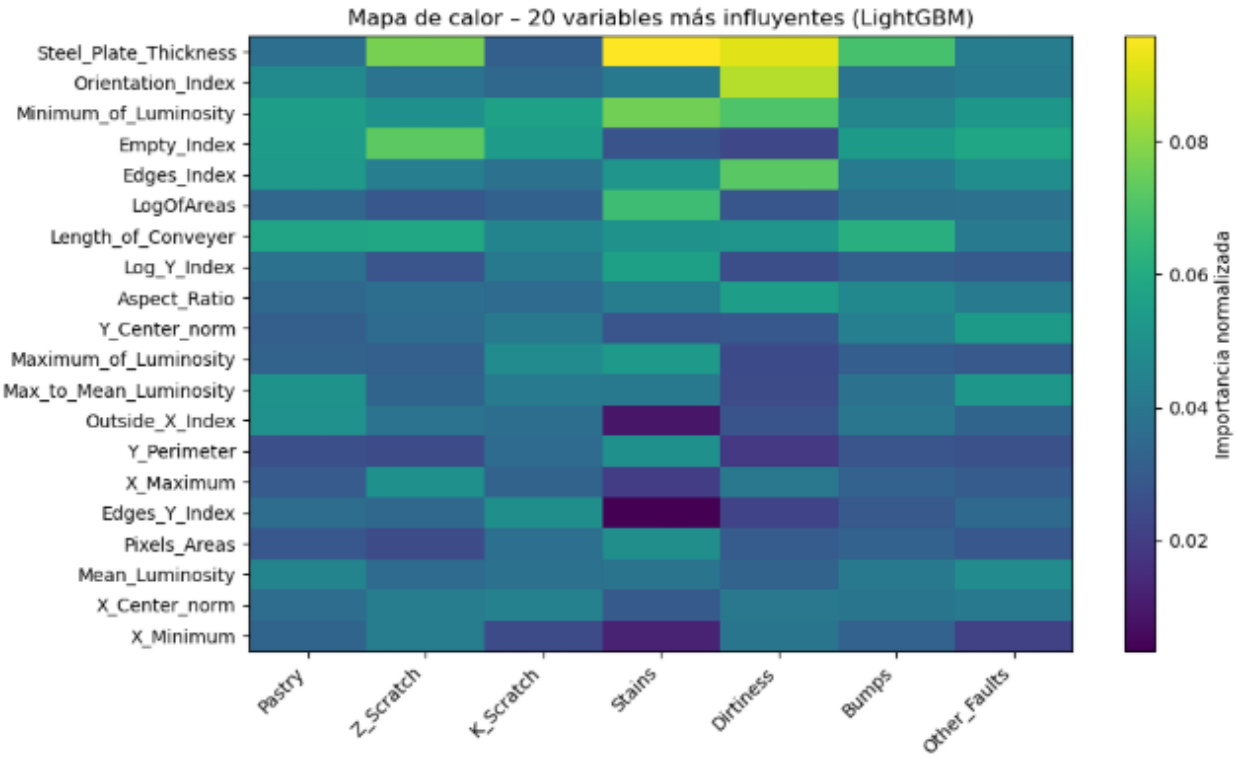


Figura 8-1. Mapa de calor 20 variables más influyentes

Se puede apreciar que las variables generadas en el preprocesamiento como Aspect_ratio, Y_Center_norm, etc. Son influyentes en varios defectos.

8.2 Conclusiones del preprocesamiento

Tabla 8–8 Comparativa Auc con y sin preprocesamiento

model	preprocesamiento	Mean_auc
lgbm	no	0.885514
lgbm	sí	0.886909
lr	no	0.783910
lr	sí	0.850710
mlp	no	0.558199
mlp	sí	0.865843
rf	no	0.883222
rf	sí	0.884474

- Sin las transformaciones los modelos lineales (p.e: logreg) no convergen y empeoran significativamente.
- Para modelos basados en arboles como random forest y lightboost el preprocesamiento no es necesario.
- El orden de transformaciones (winsorizar → log → scaler) reduce asimetrías, outliers y colas extremas.
- Las features derivadas añaden información clave (tamaño relativo, posición y luminosidad normalizada).
- El pipeline resultante es reproducible y encaja directamente en el modelo, evitando el data leakage. [15]

8.3 Resultados obtenidos

A lo largo del proyecto se evaluaron cuatro familias de modelos: Regresión Logística (baseline lineal), Random Forest (árboles en bagging) y LightGBM (gradiente boosting), red neural mlp, mediante validación cruzada estratificada de 5 folds. Para facilitar la comparación de los diferentes modelos , en la Tabla 9-1 se resumen las métricas promedio de 5 folds (0-4), que son las que se utilizaron sistemáticamente para elegir hiperparámetros.

Tabla 8–9. AUC de lgbm

Fold	**Pastry**	**Z_Scratch**	**K_Scratch**	**Stains**	**Dirtiness**	**Bumps**	**Other_Faults**	**Mean AUC**
0	0.86242	0.96306	0.98224	0.98637	0.88217	0.80326	0.71800	0.88536
1	0.85648	0.96000	0.98362	0.99279	0.88222	0.82247	0.70225	0.88569
2	0.87640	0.96379	0.98413	0.99456	0.85302	0.81475	0.70935	0.88514
3	0.87809	0.96136	0.98582	0.99175	0.82781	0.80302	0.68670	0.87636
4	0.84367	0.95523	0.98572	0.99319	0.88530	0.81252	0.70512	0.88296

Media global lgbm: 0.8837

Mejores hiperparámetros:


```
{
  "n_estimators": 612,
  "learning_rate": 0.01072281429140794,
  "num_leaves": 194,
  "max_depth": 7,
  "min_child_weight": 0.2466259045229786,
  "subsample": 0.9927990065176265,
  "colsample_bytree": 0.6147785539690324,
  "reg_alpha": 0.03601145064808459,
  "reg_lambda": 0.00042061065726512664
}
```


Tabla 8–10. AUC de logreg

Fold	Pastry	Z_Scratch	K_Scratch	Stains	Dirtiness	Bumps	Other_Faults	Mean AUC
0	0.86053	0.91331	0.97856	0.99066	0.82578	0.75026	0.65980	0.85413

1	0.85146	0.91581	0.97914	0.98859	0.85109	0.74170	0.65981	0.85537
2	0.84906	0.89298	0.97820	0.98519	0.87907	0.76286	0.65934	0.85810
3	0.85055	0.89459	0.97691	0.98836	0.84902	0.75173	0.65005	0.85160
4	0.87782	0.91186	0.98097	0.98229	0.82034	0.77072	0.66477	0.85840

Media AUC logreg: 0.85182

Mejores hiperparámetros:

{
 "n_estimators": 1000,
 "min_samples_split": 4,
 "min_samples_leaf": 7
}

Tabla 8–11. AUC de rf

Fold	Pastry	Z_Scratch	K_Scratch	Stains	Dirtiness	Bumps	Other_Faults	Mean AUC
0	0.85433	0.94812	0.98321	0.99144	0.87888	0.81546	0.71234	0.88365
1	0.84912	0.95123	0.98467	0.99212	0.88134	0.80245	0.69867	0.88132
2	0.86210	0.95756	0.98734	0.99345	0.88267	0.80922	0.70543	0.88423
3	0.86045	0.95089	0.98512	0.99078	0.77912	0.79234	0.68845	0.86477
4	0.83445	0.93989	0.98344	0.99412	0.89367	0.80867	0.69734	0.87845

Media Globlal rf: 0.87875.

Mejores hiperparámetros:

{
 "lr_penalty": "l1",
 "lr_C": 4.291057738420302
}

Tabla 8–12. AUC de mlp

Fold	Pastry	Z_Scratch	K_Scratch	Stains	Dirtiness	Bumps	Other_Faults	Mean AUC
0	0.842666	0.947118	0.979115	0.985648	0.844476	0.764036	0.683186	0.864178
1	0.856610	0.924725	0.970116	0.977263	0.841509	0.800497	0.701836	0.867508
2	0.863960	0.945971	0.979966	0.987199	0.806450	0.794449	0.686674	0.866381
3	0.878320	0.933051	0.978610	0.986500	0.814956	0.776216	0.673848	0.863714
4	0.840300	0.917283	0.980485	0.990758	0.877519	0.796728	0.696562	0.870734

Media Globlal mlp: 0.8665

Mejores hiperparámetros:

{
"n_layers": 3,
"hidden_units": 128,
"activation": "tanh",
"alpha": 0.00016480446427978953,
"learning_rate_init": 0.0011207606211860567
}

8.4 Interpretación y comparación entre modelos

Se muestran los resultados de AUC promedio para cada defecto, así como un análisis de la importancia de características.

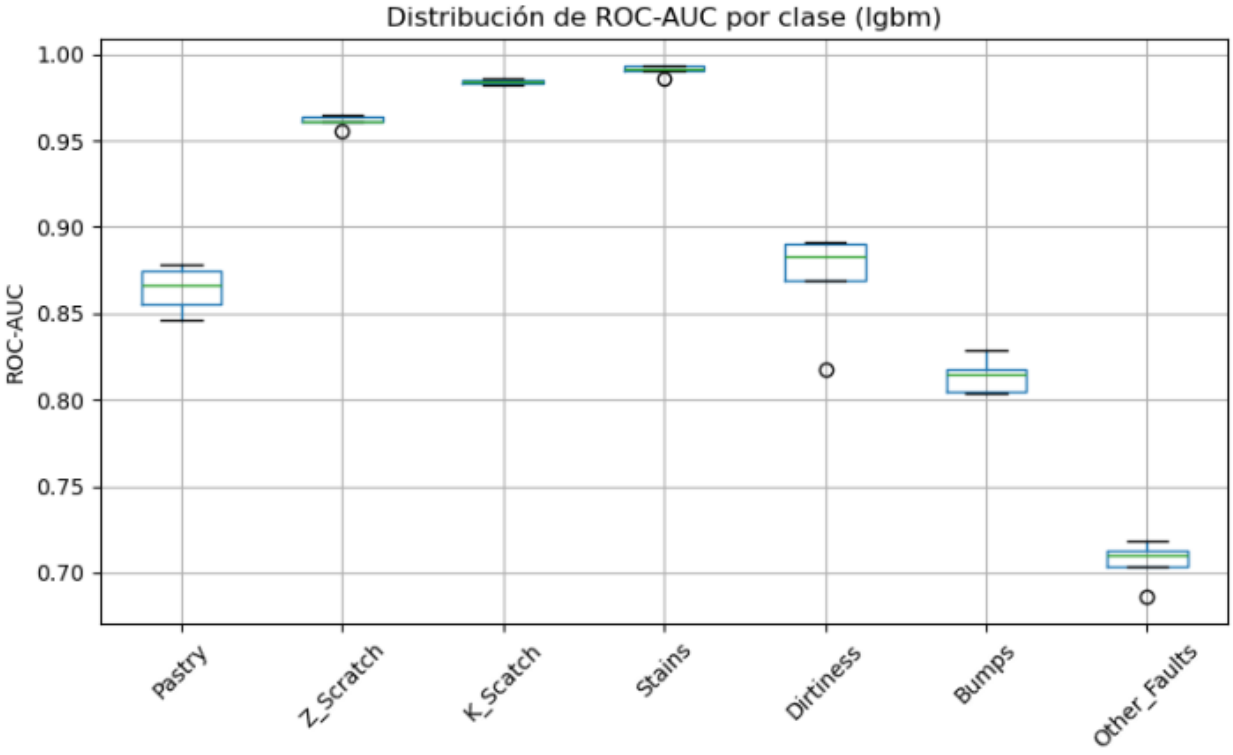


Figura 8-2. Auc medio por defecto (lgbm)

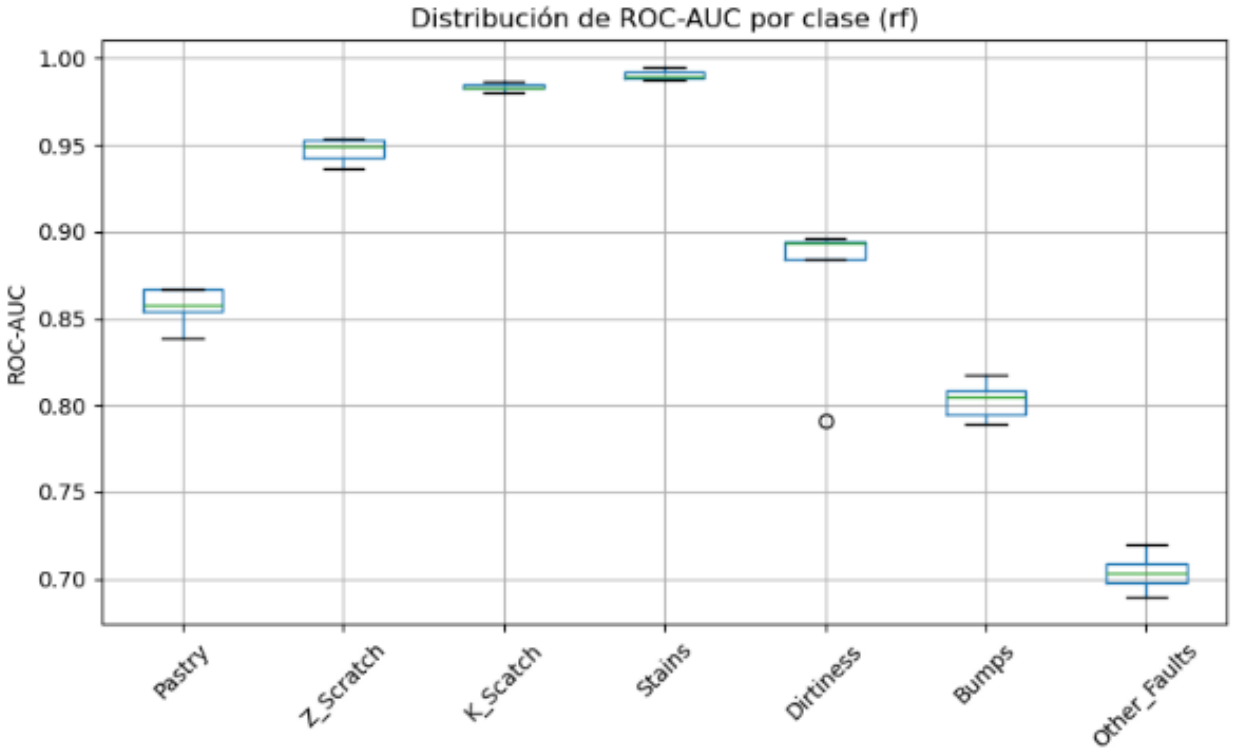


Figura 8-3. Auc medio por defecto (rf)

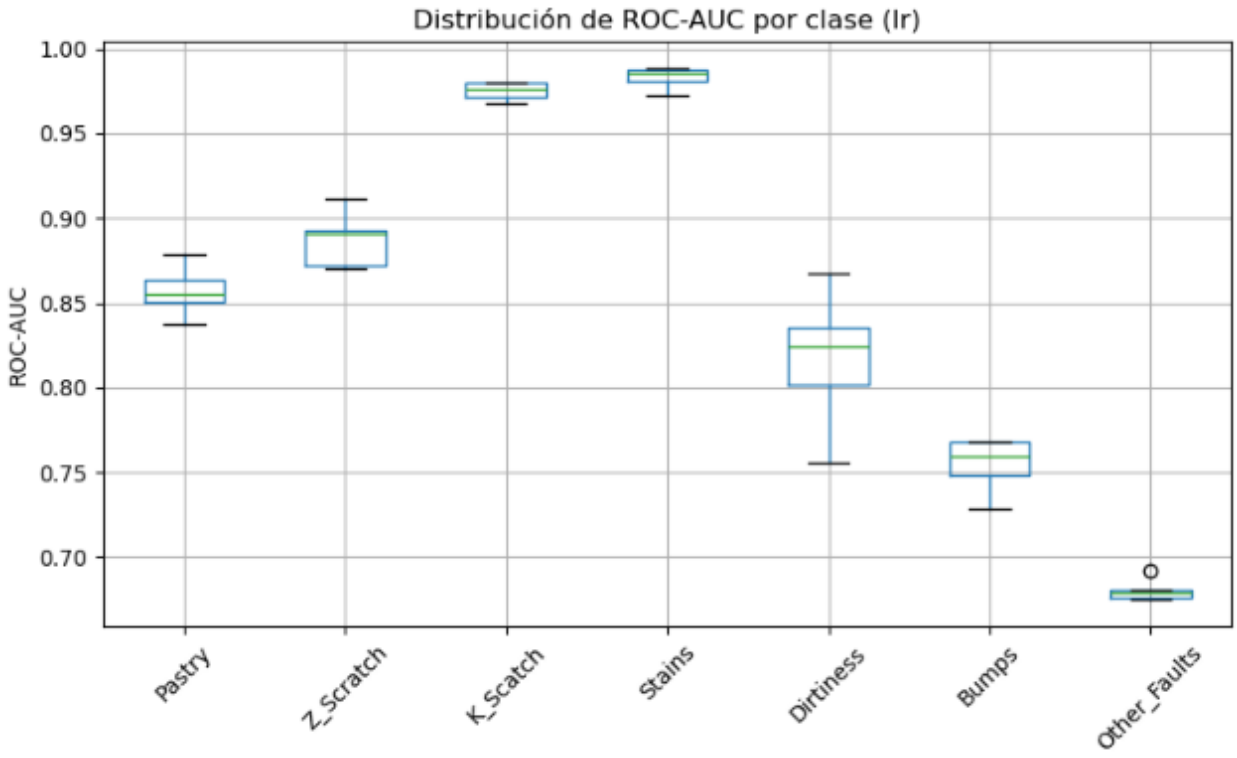


Figura 8-4. Auc medio por defecto (logreg)

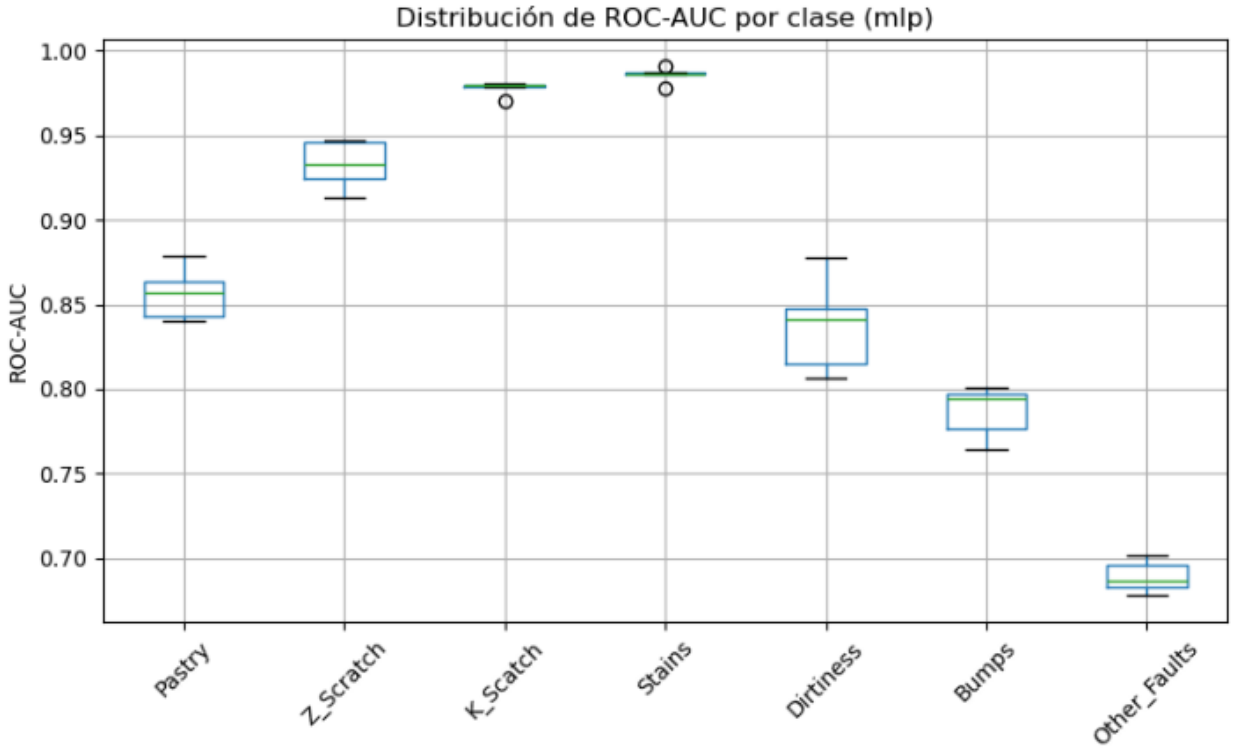


Figura 8-5. Auc medio por defecto (mlp)

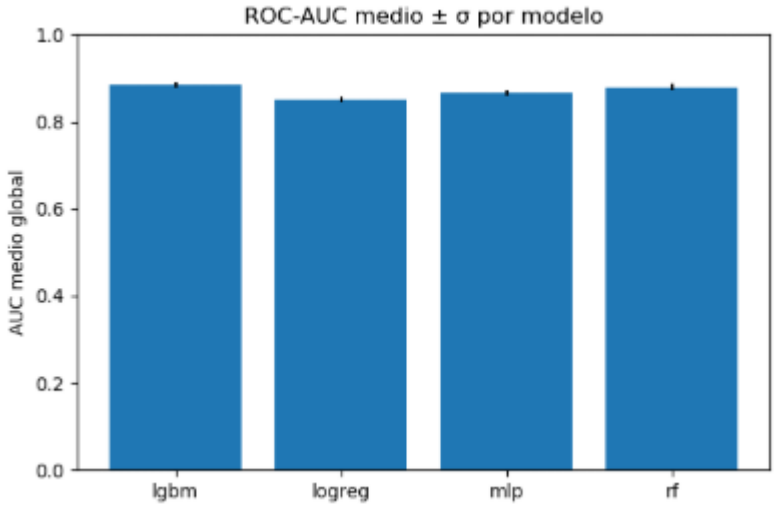


Figura 8-6. ROC-AUC medio sobre el conjunto de validación (hold out)

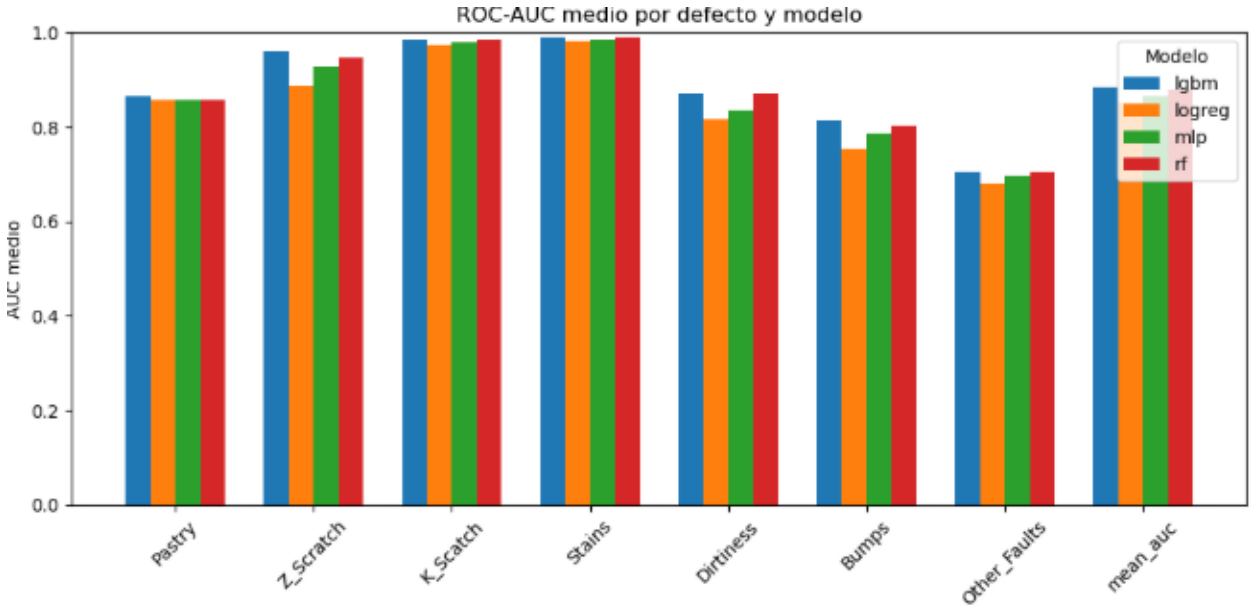


Figura 8-7. Auc medio por defecto sobre el conjunto de validación (hold out) de todos los modelos

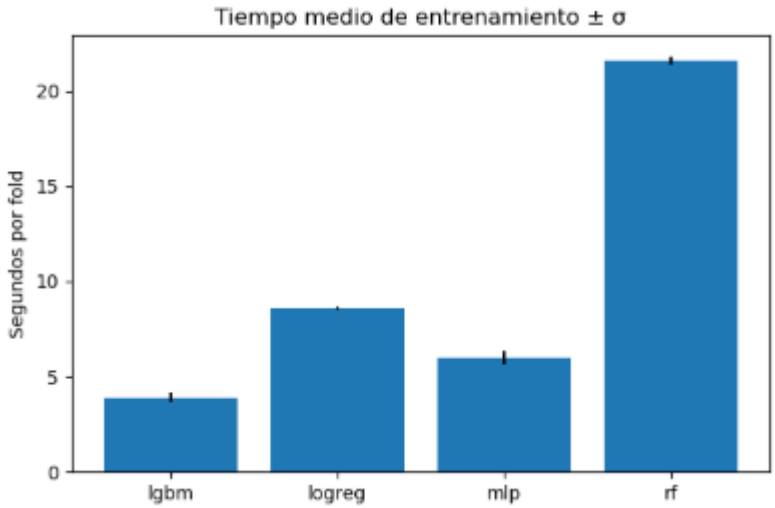


Figura 8-8. Tiempo de entrenamiento de cada modelo por fold

Puntos clave:

- LightGBM lidera consistentemente todas las clases; la ganancia media frente a RF es de +0.004 AUC, y frente a la logística, +0.027 AUC.
- El tiempo de entrenamiento por fold se mantiene en segundos; incluso en entornos de producción con inferencia en línea, los cuatro modelos son viables.
- Los resultados completos se encuentran en reports/tables/ metrics.csv.

8.5 Discusión de hallazgos y limitaciones

- LightGBM como mejor compromiso: alcanza la mayor AUC en la submission de Kaggle tanto en validación (0.883) como en el leaderboard público (0.893), sin incurrir en sobreajuste (gap privado 0.883). [10]. (En kaggle hay dos scores diferentes el público representa un 20% del test final mientras que el privado solo se revela al final de la competición 80% restante) La mejor puntuación de la competición fue 0.89782 en el leaderboard público por usuario Jeonghyeon Yun y 0.88977 en el privado por usuario kailai

Figura 8-9. Puntuación en Kaggle

- Dataset sintético: el conjunto de datos procede de la Tabular Playground Series y no refleja por completo la producción real.
- Desbalance de etiquetas: la estratificación por etiqueta mantuvo estabilidad, el oversampling empeoró ligeramente la puntuación.

8.6 Aplicaciones prácticas e implicaciones en la industria

Aplicando la regla del 80-20, podemos lograr la mayor parte del beneficio con una fracción del esfuerzo. Para una primera prueba en un entorno real conviene usar Random Forest: funciona bien sin tener que preparar mucho los datos, tolera distintas escalas y valores vacíos, y se puede convertir con facilidad en código C para integrarlo directamente en los controladores de planta (PLCs). Además, permite entender de forma sencilla qué variables influyen más en cada predicción.

Si esta prueba inicial demuestra su utilidad, el siguiente paso natural es pasar a LightGBM o XGBoost. Estos modelos suelen aportar un poco más de precisión sin disparar los tiempos de cálculo (en torno a una milésima de segundo por registro en un procesador actual) y pueden empaquetarse en formato Open Neural Network Exchange (ONNX), lo que facilita su despliegue en distintas plataformas.

9 CONCLUSIONES Y LÍNEAS FUTURAS DE INVESTIGACIÓN

El flujo de trabajo diseñado, desde la limpieza y transformación de los datos hasta la optimización, ha demostrado ser determinante para obtener resultados sólidos y reproducibles. Tras comparar varias familias de algoritmos, el modelo basado en árboles de decisión (LightGBM) ofreció el mejor equilibrio entre precisión y rapidez, alcanzando un AUC medio de 0,88 sin mostrar indicios de sobreajuste. Este rendimiento confirma que un preprocesado cuidadoso, junto con una evaluación coherente del desbalanceo de clases, puede mitigar la pérdida de información y mejorar la generalización aun cuando se trabaja con datos sintéticos. Aun así, el carácter artificial del conjunto empleado deja abiertas dudas sobre la traslación directa de estas conclusiones a un entorno industrial real [8].

De cara al futuro sería valioso ensayar combinaciones de modelos distintas, por ejemplo, agrupando varios algoritmos y entrenando un meta-modelo con hiperpárametros diferentes por defecto que decida a quién "hacer caso" en cada escenario; estas técnicas de ensamblado suelen aportar pequeñas mejoras adicionales de precisión. También convendría incorporar herramientas de interpretabilidad más avanzadas que detallen la influencia de cada variable, facilitando el diagnóstico de defectos por parte del personal de calidad. Finalmente, probar el sistema con datos procedentes de la línea de producción permitiría medir su robustez ante el ruido real y, si fuera necesario, ajustar de nuevo los parámetros para mantener el rendimiento a lo largo del tiempo.

De cara a continuar puliendo el sistema, un siguiente paso razonable sería revisar a fondo los resultados etiqueta por etiqueta: construir matrices de confusión y curvas ROC permitiría detectar, por ejemplo, si algún defecto concreto se confunde sistemáticamente con "Other Faults" y ajustar los umbrales de decisión o los pesos de clase en consecuencia. En paralelo, podría probarse un preprocesado "más fino" diferenciado para cada modelo, se han mantenido variables con correlación alta que empeoran algunos métodos como la regresión logística. Finalmente, convendría explorar dos o tres modelos complementarios al LightGBM (p.ej. CatBoost o XGboost); incluso si la ganancia bruta en AUC es pequeña [16, 17, 19], el contraste de importancias y la calibración conjunta de probabilidades aportarían una imagen más nítida de dónde está aún el margen de mejora.

Una posible extensión sería revisar la calibración de probabilidades, especialmente si se desea utilizar los scores del modelo más allá del ranking puro. Aunque la métrica AUC-ROC utilizada en este trabajo evalúa la capacidad discriminativa (es decir, si los ejemplos positivos obtienen scores más altos que los negativos), no garantiza que las probabilidades predichas sean numéricamente fiables. En otras palabras, un valor de 0.9 no implica que el defecto ocurra en el 90 % de los casos similares. Técnicas de calibración como Platt Scaling (una regresión logística aplicada a los scores) o Isotonic Regression (un ajuste monótono no paramétrico) permiten corregir esta desviación de forma post-hoc. Estas pueden integrarse mediante el wrapper CalibratedClassifierCV de scikit-learn [16], y serían especialmente útiles si se desea fijar umbrales operativos explícitos por clase o comparar directamente scores entre etiquetas con distinta prevalencia.

REFERENCIAS

[1] M. A. A. Kazemi, S. Hajian, y N. Kiani, "Quality control and classification of steel plates faults using data mining," *Applied Mathematics & Information Sciences Letters*, vol. 6, no. 2, pp. 59–67, 2018, doi: 10.18576/amisl/060202.

[2] W. Zhao, F. Chen, H. Huang, D. Li, y W. Cheng, "A new steel defect detection algorithm based on deep learning," *Computational Intelligence and Neuroscience*, vol. 2021, Art. ID 5592878, 13 pp., Mar. 2021, doi: 10.1155/2021/5592878.

[3] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[4] I. D. Kordatos and P. Benardos, "Comparative analysis of machine-learning algorithms for steel-plate defect classification," *Int. J. Mechatronics Manuf. Syst.*, vol. 15, no. 4, pp. 246–263, Nov. 2022, doi: 10.1504/IJMMS.2022.127211.

[5] S. García, J. Luengo, and F. Herrera, *Data Preprocessing in Data Mining*, Intelligent Systems Reference Library, vol. 72. Cham, Switzerland: Springer, 2015, doi: 10.1007/978-3-319-10247-4.

[6] A. Mishra, F. Harrou, and Y. Sun, "Explainable artificial-intelligence-based fault diagnosis and insight extraction for steel manufacturing," *arXiv preprint* arXiv:2008.04448, Aug. 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2008.04448

[7] M. Buscema, S. Terzi, and W. Tastle, *Steel Plates Faults Data Set* [dataset], UCI Machine Learning Repository, 2010. [Online]. Available: https://archive.ics.uci.edu/dataset/198/steel+plates+faults

[8] W. Weng, Y.-W. Li, Y. Song, and X. Jin, "Multi-label classification: review and opportunities," *Data Sci. Manag.*, vol. 6, no. 2, pp. 71–86, 2023, doi: 10.1016/j.dsm.2022.12.001

[9] T. Nkonyana, Y. Sun, B. Twala, and E. Dogo, "Performance evaluation of data-mining techniques in steel manufacturing industry," *Procedia Manufacturing*, vol. 35, pp. 623–628, 2019, doi: 10.1016/j.promfg.2019.06.004.

[10] Kaggle, *Steel Plate Defect Prediction – Playground Series Season 4, Episode 3* [online], 2024. Available: https://www.kaggle.com/competitions/playground-series-s4e3

[11] M.-L. Zhang and Z.-H. Zhou, "ML-KNN: a lazy-learning approach to multi-label learning," *Pattern Recognition*, vol. 40, no. 7, pp. 2038–2048, Jul. 2007, doi: 10.1016/j.patcog.2006.12.019.

[12] N. Endut, W. M. A. F. Hamzah, and I. Ismail, "A systematic literature review on multi-label classification based on machine-learning algorithms," *TEM Journal*, vol. 11, no. 2, pp. 658–666, May 2022, doi: 10.18421/TEM112-20.

[13] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, vol. 30, G. Guyon et al., Eds., Long Beach, CA, USA: Curran Associates, Inc., 2017, pp. 3146–315.

[14] B. Ghasemkhani, R. Yilmaz, D. Birant, and R. A. Kut, "Logistic Model Tree Forest for steel plates faults prediction," *Machines*, vol. 11, no. 7, Art. 679, Jun. 2023, doi: 10.3390/machines11070679.

[15] nazimcherpanov, "*STEEL PLATE DEFECT PREDICTION [Kaggle Notebook],*" 2023. [Online]. Available: https://www.kaggle.com/code/nazimcherpanov/0-8922-steel-plate-defect-prediction. [Accessed 03 Julio 2025].

[16] Scikit-learn Developers, *Scikit-learn: Machine Learning in Python* [online], INRIA, 2024. Available: https://scikit-learn.org/stable/.

[17] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications*

in R, 2nd ed. New York, NY, USA: Springer, 2021, doi: 10.1007/978-1-0716-1418-1..

[18] Optuna Development Team, *Optuna – A Hyperparameter Optimization Framework* [online], 2025. Available: https://optuna.org/

[19] Microsoft, *LightGBM Documentation* [online], 2024. Available: https://lightgbm.readthedocs.io/en/latest/

[20] K. Song, S. Hu, and Y. Yan, "Automatic recognition of surface defects on hot-rolled steel strip using scattering convolution network," *J. Comput. Inf. Syst.*, vol. 10, no. 7, pp. 3049–3055, 2014.

[21] A. K. Srivastava, "Comparison analysis of machine learning algorithms for steel plate fault detection," *Int. Res. J. Eng. Technol. (IRJET)*, vol. 6, no. 5, pp. 1231–1234, May 2019.

[22] I. G. Peris, *tfg_steel_plate_defects_v1* [online], GitHub, 2024. Available: https://github.com/ignagp34/tfg_steel_plate_defects_v1

ÍNDICE DE CONCEPTOS

Tabla 0–1 Conceptos

Concepto	Significado
Area Under the Curve	Medida que cuantifica el rendimiento de un clasificador binario a partir del área de una curva.
Adam	Es un optimizador avanzado que mejora el descenso de gradiente. Adam ajusta automáticamente la velocidad con la que se mueven los pesos (la tasa de aprendizaje) en cada paso y cada parámetro, ayudando a que el entrenamiento sea más rápido y estable
Area Under the ROC Curve	Variante completa de AUC que refiere específicamente a la curva ROC (Receiver Operating Characteristic). Para ello se mide la tasa de verdaderos positivos vs la tasa de falsos positivos.
Aprendizaje supervisado	Paradigma de Machine Learning donde un modelo aprende a mapear entradas a salidas a partir de ejemplos etiquetados. En este contexto, el objetivo es predecir la presencia de defectos conocidos.
Árboles de decisión	Algoritmos supervisados que modelan decisiones mediante divisiones jerárquicas de atributos. Cada nodo refleja una regla sobre un atributo que conduce a una predicción en las hojas.
Binary Relevance/ one vs rest	Estrategia que consiste en entrenar un clasificador independiente por cada etiqueta, asumiendo independencia entre ellas. Es simple y ampliamente utilizada en escenarios multi-etiqueta. Sinónimo de one vs rest en el contexto de multi-etiqueta.
Bootstrap	Utilizado en random forest para la creación de árboles de clasificación, Consiste en particionar los datos de entrenamiento por árbol independiente permitiendo el remplazo de ellos.
Classifiers chains	Técnica que encadena clasificadores de forma secuencial, añadiendo las predicciones previas como variables de entrada al siguiente modelo, para capturar dependencias entre etiquetas. Enfoque alternativo a Binary Relevance.
Exploratory Data Analysis	Proceso de análisis exploratorio previo al modelado. Incluye visualizaciones, análisis de correlación y detección de outliers.
Ensemble Learning Algorithm / Enfoques ensemble	Término genérico que hace referencia a métodos que combinan múltiples modelos base.
Edited Nearest Neighbors	Técnica de limpieza de ruido en datos, eliminando instancias mal clasificadas por kNN. Se cita como opción en la preparación del dataset.
Extended Decision Label Annotation	Estrategia que enriquece la representación de etiquetas heterogéneas (p.ej., Other_Faults) para mejorar la capacidad predictiva y mitigar el

Concepto	Significado
	impacto de la vaguedad en la clase.
Arquitectura feed-forward	La información fluye en una sola dirección, desde la entrada hasta la salida, sin bucles ni retroalimentación. Cada capa recibe la salida de la anterior y la pasa a la siguiente.
Función de Pérdida Binaria Agregada	Suma de las pérdidas (por ejemplo, entropía cruzada binaria) calculadas de forma independiente por cada etiqueta, optimizando simultáneamente todas las salidas.
Data Leakage	Problema que ocurre cuando información del conjunto de validación/test contamina el proceso de ajuste de transformaciones o modelos, generando una evaluación artificialmente optimista.
Desbalance de clases	Situación en la que algunos defectos aparecen con mucha menor frecuencia que otros. Provoca sesgos si no se mitigan mediante técnicas como pesos de clase u oversampling.
Dropout	Técnica de regularización en redes neuronales, Durante el entrenamiento, desactiva aleatoriamente algunas neuronas en cada paso. Esto obliga a la red a aprender representaciones más generales y robustas.
Estratificación Multi-Label	Técnica de partición que conserva proporcionalmente la distribución de todas las etiquetas en los pliegues de validación, garantizando representatividad incluso de las clases minoritarias.
Focal loss	Función de pérdida que reduce el peso de los ejemplos bien clasificados y enfoca el aprendizaje en los casos difíciles o minoritarios, modulando la contribución de cada muestra mediante un factor que penaliza más los errores de predicción alta confianza.
Gradient Boosting Machines	Algoritmos de ensamblado que entrenan secuencialmente árboles de decisión corrigiendo errores previos.
Iterative stratification	Técnica de particionado que reparte las muestras de un conjunto de datos multilabel entre los pliegues de validación conservando de forma proporcional no solo la frecuencia individual de cada etiqueta, sino también sus coocurrencias más frecuentes, garantizando que todas las combinaciones relevantes estén representadas en cada pliegue.
Interquartile Range	Medida estadística que ayuda a identificar outliers. Se utiliza en la limpieza de datos (preprocesamiento).
Kernel Density Estimation	Técnica no paramétrica para estimar la distribución de probabilidad de una variable. Empleada en EDA para visualizar distribuciones.
Label Powerset	Técnica que trata cada combinación de etiquetas como una clase única en un problema multi-clase, útil cuando el número de combinaciones posibles es pequeño.
Long Short-Term Memory	Tipo de red neuronal recurrente capaz de capturar dependencias temporales.

Concepto	Significado
LightGBM	Implementación optimizada de Gradient Boosting que usa histogramas y técnicas de hoja inteligente para acelerar el entrenamiento y reducir el consumo de memoria.
Multi-Label k Nearest Neighbors	Algoritmo de clasificación multilabel basado en kNN.
Multi-Layer Perceptron	Red neuronal feed-forward con varias capas ocultas. Considerada como modelo de referencia en la clasificación de defectos.
One-Against-One Support Vector Machine	Estrategia de clasificación multiclase que entrena un SVM por cada par de clases.
Open Neural Network Exchange	Formato abierto para exportar modelos ML. Se cita como opción de interoperabilidad de modelos entrenados.
Overfitting o sobreajuste	Ocurre cunado un modelo aprende demasiado bien los datos del entrenamiento de manera que se adapta al ruido.
Oversampling	Estrategia que consiste en replicar observaciones de las clases minoritarias dentro del conjunto de entrenamiento, con el objetivo de balancear la proporción de etiquetas.
Pesos de Clase (Class Weights)	Ajustes aplicados a la función de pérdida del modelo para penalizar más los errores en clases poco frecuentes y reducir el sesgo hacia la predicción de la clase mayoritaria.
Principal Component Analysis	Técnica de reducción de dimensionalidad que proyecta datos en componentes principales.
Pipeline	Secuencia estructurada de pasos implementados en un objeto único (por ejemplo, Pipeline de scikit-learn), que encapsula todo el flujo de transformaciones y permite aplicarlo de forma idéntica en entrenamiento y validación.
Random forest	Algoritmo de ensamblado basado en la agregación de múltiples árboles de decisión entrenados sobre muestras y subconjuntos de variables aleatorios, robusto frente a ruido y overfitting.
Red Neuronal (Perceptrón Multicapa)	Arquitectura con una o más capas ocultas que permite aprender relaciones no lineales entre variables y producir salidas independientes por etiqueta mediante activaciones sigmoides.
Reproducibilidad	Capacidad de ejecutar el pipeline completo (limpieza, ingeniería de características y escalado) de forma idéntica en cualquier fase o entorno, asegurando consistencia de resultados.
Clasificación Multi-Etiqueta	Tarea de predicción en la que una muestra puede pertenecer simultáneamente a múltiples categorías. A diferencia de la clasificación multi-clase, se generan salidas binarias por cada etiqueta.
SHapley Additive exPlanations	Método para interpretar predicciones de modelos complejos, asignando importancia a cada feature. Utilizado para explicar predicciones del

Concepto	Significado
	modelo final.
Synthetic Minority Over-sampling Technique	Técnica de generación de ejemplos sintéticos para balancear clases minoritarias. Considerada en la etapa de preprocesamiento.
Standard Deviation	Medida de dispersión. Utilizada en la exploración y normalización de variables.
Support Vector Machine	Clasificadores que buscan el hiperplano con mayor margen que separe las clases en un espacio de alta dimensión. Son eficaces para patrones complejos en datos tabulares.
Tree-structured Parzen Estimator	Método bayesiano de optimización de hiperparámetros. Utilizado en la librería Optuna para ajustar modelos.
Transformadas de Fourier, filtros de Gabor y segmentaciones	En este contexto, conjunto de métodos para transformar imágenes de la superficie de la placa de acero en atributos cuantitativos. Extraen características visuales relevantes.
XGBoost	Algoritmo de boosting muy popular por su precisión y capacidad de regularización, eficaz en tareas de clasificación multi-etiqueta mediante enfoque Binary Relevance.

GLOSARIO

AUC: Area Under the Curve ..... x, i, iii, i, 12, 14, 17, 18, 11, 13, 10, 11, 12, 13
AUC-ROC: Area Under the ROC Curve ..... x, i, 14, 17, 18
CPU: Central Processing Unit 13
CSV: Comma Separated Values ..... xi, 10, 14
EDA: Exploratory Data Analysis ..... ii, 10, 16, 19
ELA: Ensemble Learning Algorithm 14
ENN: Edited Nearest Neighbors 14
GBMs: Gradient Boosting Machines 12
IQR: Interquartile Range 10, 15, 16, 17, 22
KDE: Kernel Density Estimation ..... i, 12, 14
LSTM: Long Short-Term Memory 14
ML: Machine Learning ..... ii, 11, 12, 10
ML-kNN; Multi-Label k Nearest Neighbors 12
MLP: Multi-Layer Perceptron 12, 13, 11
N: Número de instancias 10
OAO-SVM: One-Against-One Support Vector Machine 12
ONNX: Open Neural Network Exchange 13
PCA: Principal Component Analysis 11, 14
PLCs: Programmable Logic Controllers 13
SHAP: SHapley Additive exPlanations 13
SMOTE: Synthetic Minority Over-sampling Technique 13
STD: Standard Deviation 23
SVM: Support Vector Machine ..... ii, 11, 12, 13, 14
TPE: Tree-structured Parzen Estimator 12
UCI: University of California, Irvine 11, 10
UTF-8: Unicode Transformation Format – 8 bit 14

ANEXO A. ESTRUCTURA DEL PROYECTO Y FRAGMENTOS DE CÓDIGO

A.1 Estructura de carpetas del proyecto

```
tfg-steel-plate-defects/
├── data/
│   ├── raw/                    # Datos originales de Kaggle
│   │   └── playground-series-s4e3/
│   ├── interim/                # Datos limpios o transformados (outliers, imputación)
│   └── external/               # Recursos externos (documentos, benchmarks, etc.)
├── notebooks/
│   ├── 01_eda.ipynb            # Análisis exploratorio
│   ├── 02_feature_engineering.ipynb  # Ingeniería de características
│   ├── 03_model_training.ipynb     # Entrenamiento de modelos
│   ├── 04_hyperparameter_optimization.ipynb     # Optimización de hiperpárametros
│   └── 05_feature_importance.ipynb     # Comparación de modelos y features
├── src/
│   ├── __init__.py
│   ├── config.py               # Rutas y parámetros globales
│   ├── data/
│   │   ├── loading.py          # Carga de CSVs y rutas
│   │   ├── split.py            # Generación de splits (hold-out y k-fold)
│   │   └── preprocessing.py       # Pipeline de limpieza y transformaciones
│   ├── features/
│   │   └── transformers.py        # FeatureGenerator, ColumnDropper, escalado, etc.
│   ├── models/
│   │   ├── builders.py         # Función build_estimator por tipo de modelo
│   │   ├── hpo.py              # Optimización de hiperpárametros
│   │   └── trainer.py          # Clase Trainer: fit, predict, evaluación
│   ├── visualization/          # Gráficas reutilizables (SHAP, AUC por clase, etc.)
│   └── cli/
│       └── train.py            # Entry point con Typer para entrenamiento por terminal
├── models/                     # Pesos entrenados (.pkl), parámetros .json
├── reports/
│   ├── figures/                # Gráficos exportados para la memoria
│   └── tables/                 # CSVs con métricas y resultados
├── docs/                       # Borradores de memoria
└── environment.yml
```

A.2 Fragmentos clave del código

A.2.1 FeatureGenerator (transformers.py)

[5, 15, 16, 22]

```
class FeatureGenerator(BaseEstimator, TransformerMixin):
    """
    Genera features derivadas sin filtrar datos de validación/test.
    Añade columnas que describen:
    - forma y tamaño del defecto (width, height, aspect ratio)
    - posición relativa (centro normalizado)
    - intensidad luminosa (media y pico relativo)
    - flag binario de área muy grande (P95)
    """

    def __init__(self, img_width: int = 1700, img_height: int = 12_800_000):
        self.img_width = img_width
        self.img_height = img_height

    def fit(self, X, y=None):
        # Guarda el percentil 95 del área para usarlo como umbral fijo
        self.feature_names_in_ = np.asarray(X.columns, dtype=object)
        self.p95_ = np.percentile(X["Pixels_Areas"], 95)
        return self

    def transform(self, X):
        X_ = X.copy()

        # Tamaño del defecto (en píxeles)
        X_["Width"] = X_["X_Maximum"] - X_["X_Minimum"]
        X_["Height"] = X_["Y_Maximum"] - X_["Y_Minimum"]
        X_["Aspect_Ratio"] = X_["Width"] / (X_["Height"] + 1e-6)

        # Posición relativa (centro de la bounding-box)
        X_["X_Center_norm"] = (X_["X_Minimum"] + X_["X_Maximum"]) / 2 / self.img_width
        X_["Y_Center_norm"] = (X_["Y_Minimum"] + X_["Y_Maximum"]) / 2 / self.img_height

        # Intensidad media y contraste local
        X_["Mean_Luminosity"] = X_["Sum_of_Luminosity"] / (X_["Pixels_Areas"] + 1e-6)
        X_["Max_to_Mean_Luminosity"] = (
            X_["Maximum_of_Luminosity"] / (X_["Mean_Luminosity"] + 1e-6)
        )

        # Flag binario: ¿es un defecto inusualmente grande?
        X_["is_very_large"] = (X_["Pixels_Areas"] > self.p95_).astype(int)

        return X_

    def get_feature_names_out(self, input_features=None):
        if input_features is None:
            input_features = self.feature_names_in_
        extra = np.array(
            [
                "Width",
                "Height",
                "Aspect_Ratio",
                "X_Center_norm",
```

```
            "Y_Center_norm",
            "Mean_Luminosity",
            "Max_to_Mean_Luminosity",
            "is_very_large",
        ],
        dtype=object,
    )
    return np.concatenate([np.asarray(input_features, dtype=object), extra])
```

A.2.2 Función build_estimator (builders.py)

[16,19,22]

```
def build_estimator(name: str, pos_weights: List[float]):
    """
    Devuelve una lista de estimadores (uno por etiqueta).

    Parameters
    ----------
    name : {"lgbm", "rf", "logreg", "mlp"}
        Algoritmo base a utilizar.
    pos_weights : list[float]
        Peso asignado a la clase positiva de cada etiqueta, útil para balanceo.
    """
    n_labels = len(pos_weights)

    # LightGBM: boosting eficiente para datos tabulares
    if name == "lgbm":
        base_params = dict(
            n_estimators=500,
            learning_rate=0.05,
            num_leaves=64,
            subsample=0.8,
            colsample_bytree=0.8,
            random_state=RANDOM_STATE,
            n_jobs=N_JOBS,
            verbosity=-1,
            min_gain_to_split=0.0,
        )
        base_params.update(_load_best_params("lgbm"))  # Optuna override
        base_params = _filter_params(LGBMClassifier, base_params)
        return [
            LGBMClassifier(**base_params, scale_pos_weight=pos_weights[k])
            for k in range(n_labels)
        ]

    # Random Forest: modelo en bagging robusto al ruido
    if name == "rf":
        base_params = dict(
            n_estimators=400,
            random_state=RANDOM_STATE,
            n_jobs=N_JOBS,
        )
        base_params.update(_load_best_params("rf"))
        base_params = _filter_params(RandomForestClassifier, base_params)
        return [
            RandomForestClassifier(
                **base_params,
                class_weight={0: 1, 1: pos_weights[k]}
```

```
            )
            for k in range(n_labels)
        ]

    # Regresión logística con ajuste de solver según penalización
    if name in {"logreg", "lr", "logistic"}:
        base_params = dict(
            penalty="l2",
            solver="lbfgs",
            C=1.0,
            max_iter=2000,
            random_state=RANDOM_STATE,
            n_jobs=N_JOBS,
        )
        base_params.update(_load_best_params("logreg"))

        # Si se usa L1 o elasticnet, se fuerza solver compatible
        if base_params.get("penalty") in {"l1", "elasticnet"} and \
           base_params.get("solver") not in {"saga", "liblinear"}:
            base_params["solver"] = "saga"

        base_params = _filter_params(LogisticRegression, base_params)
        return [
            LogisticRegression(
                **base_params,
                class_weight={0: 1, 1: pos_weights[k]}
            )
            for k in range(n_labels)
        ]

    # MLP (red neuronal densa) con regularización y early stopping
    if name == "mlp":
        base_params = dict(
            hidden_layer_sizes=(128, 64),
            activation="relu",
            solver="adam",
            learning_rate_init=0.001,
            batch_size=256,
            max_iter=200,
            alpha=1e-4,
            early_stopping=True,
            validation_fraction=0.1,
            n_iter_no_change=10,
            random_state=RANDOM_STATE,
            verbose=False,
        )
        base_params.update(_load_best_params("mlp"))
        base_params = _filter_params(MLPClassifier, base_params)
        return [MLPClassifier(**base_params) for _ in range(n_labels)]

    # Si el modelo no está implementado
    raise ValueError(
        f"Modelo '{name}' no soportado. Elige entre 'lgbm', 'rf', 'logreg' o 'mlp'."
    )
```

A.2.3 Método run_cv() – Entrenamiento en validación cruzada (K-Fold)

[16, 19, 22]

```
def run_cv(self) -> pd.DataFrame:
    """
    Ejecuta validación cruzada k-fold para el modelo actual.
    Guarda las métricas por fold y devuelve un DataFrame con los resultados.
    """

    # 1. Carga de datos y generación de pliegues -------------------------
    X, y, train_idx, hold_idx, folds = self._prepare_data()

    # 2. Inicialización del pipeline y modelos ---------------------------
    self._prepare_transformers_and_models(y[train_idx])

    # 3. Entrenamiento por fold ------------------------------------------
    cv_metrics: List[Dict[str, float]] = []
    for fold_id, val_idx in enumerate(folds):
        # Separa train y validación para este fold
        tr_idx = np.setdiff1d(train_idx, val_idx)
        self.logger.info(
            "Fold %s – train %s · val %s", fold_id, len(tr_idx), len(val_idx)
        )

        # Entrena modelos y calcula AUC por clase en este fold
        fold_metrics = self._train_fold(fold_id, tr_idx, val_idx, X, y)
        cv_metrics.append(fold_metrics)

    # 4. Agrega métricas y guarda resultados -----------------------------
    df_cv = pd.DataFrame(cv_metrics)
    df_cv.to_csv(self.report_dir / f"{self.cfg.model_name}_cv_metrics.csv", index=False)

    self.logger.info("Mean AUC per fold:\n%s", df_cv[["fold", "mean_auc"]])
    self.logger.info("Global mean AUC = %.5f", df_cv["mean_auc"].mean())

    return df_cv
```

ANEXO B. ESTRUCTURA DEL PROYECTO Y FRAGMENTOS DE CÓDIGO

B.1 Notebook 01_eda.ipynb

Análisis Exploratorio de Datos (EDA)

En este notebook cargaremos los datos de entrenamiento y generaremos **inline** todas las visualizaciones del proyecto Steel Plate Defect Prediction, aprovechando las funciones de `src/visualization/eda.py`.

[3]:
```
%matplotlib inline
import sys, os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

project_root = os.path.abspath(os.path.join(os.getcwd(), '..'))
if project_root not in sys.path:
    sys.path.insert(0, project_root)

sns.set_theme(style="whitegrid")
plt.rcParams.update({"figure.autolayout": True})

from src.visualization.eda import load_data

df = load_data(
    "../data/raw/playground-series-s4e3/train.csv"
)
print(f"Dimensiones del dataset: {df.shape}")
df.head()
```

```
Dimensiones del dataset: (19219, 35)
```

[3]:

	id	X_Minimum	X_Maximum	Y_Minimum	Y_Maximum	Pixels_Areas	X_Perimeter	Y_Perimeter	Sum_of_Luminosity	Minimum_of_Luminosity	...	Orientation_Index
0	0	584	590	909972	909977	16	8	5	2274	113	...	-0.5000
1	1	808	816	728350	728372	433	20	54	44478	70	...	0.7419
2	2	39	192	2212076	2212144	11388	705	420	1311391	29	...	-0.0105
3	3	781	789	3353146	3353173	210	16	29	3202	114	...	0.6667
4	4	1540	1560	618457	618502	521	72	67	48231	82	...	0.9158

5 rows × 35 columns

Figura 0-1. Head con dimensiones del dataset y primeras 5 filas

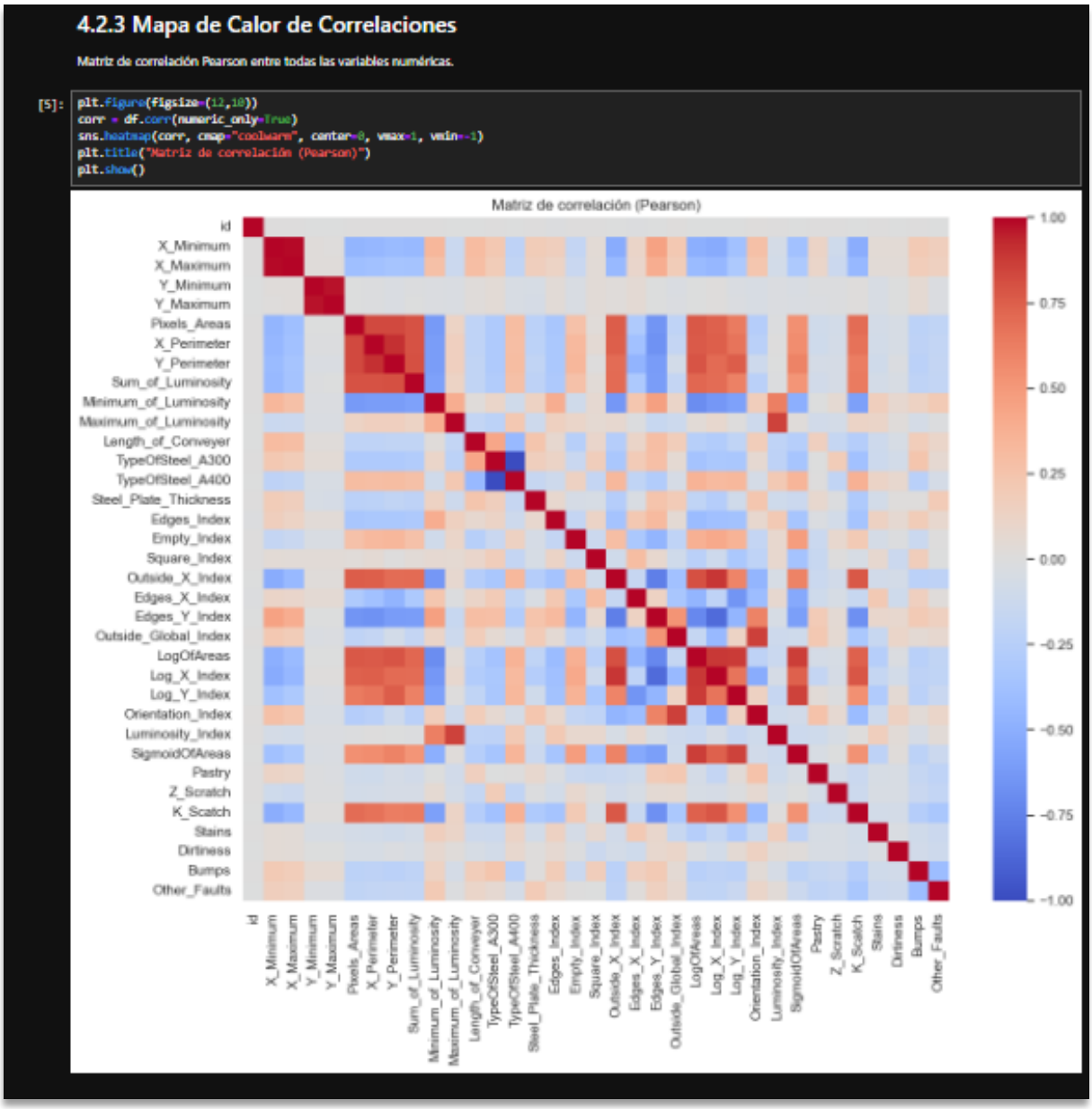


Figura 0-2. Matriz de correlación Pearson entre las variables.

B.2 Notebook 02_feature_engineering.ipynb

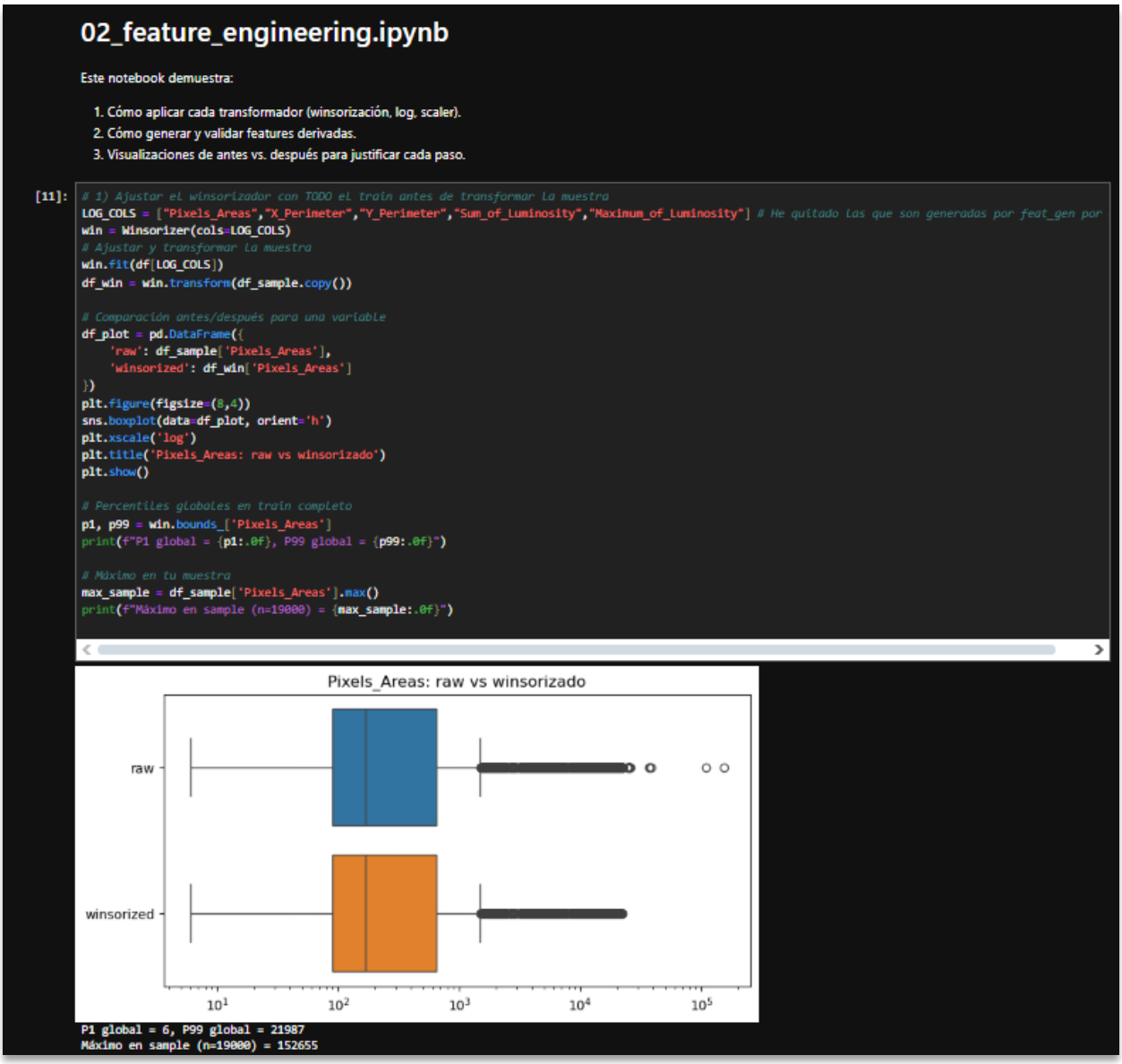


Figura 0-3. Winsorizado variable Pixels_areas.

```
[17]: set_config(transform_output="pandas")

pipe = build_preprocessing_pipeline()
X_trans = pipe.fit_transform(df_sample)
# --- Pixels_Areas ---
df_plot = pd.DataFrame({
    'raw': df_sample[QUANT_COLS[0]],
    'transformed': X_trans[QUANT_COLS[0]]
})

def kde_raw_vs_trans(col):
    # --- preparar datos ---
    raw = df_sample[col].replace(0, np.nan).dropna()          # quitar ceros
    raw_log = np.log10(raw)                                   # Log10 antes del KDE
    trans   = X_trans[col]                                    # N(0,1)

    fig, ax = plt.subplots(1, 2, figsize=(10, 3), sharey=True)

    # ----- raw en log10 -----
    sns.kdeplot(raw_log, ax=ax[0], bw_adjust=0.5, fill=True, color='steelblue')
    ax[0].set_title(f'{col} - raw log10')
    ax[0].set_xlabel(f'log10({col})')

    # ----- transformado N(0,1) -----
    sns.kdeplot(trans,   ax=ax[1], bw_adjust=0.5, fill=True, color='darkorange')
    ax[1].set_title(f'{col} - transformed (N(0,1))')
    ax[1].set_xlabel(col)

    plt.show()

kde_raw_vs_trans(QUANT_COLS[0])
```

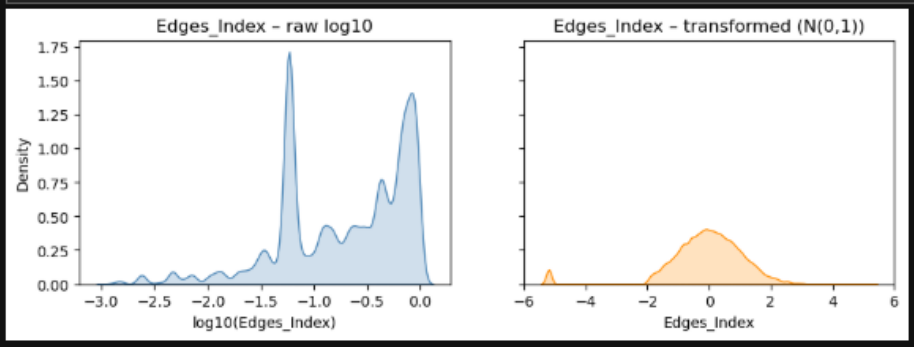
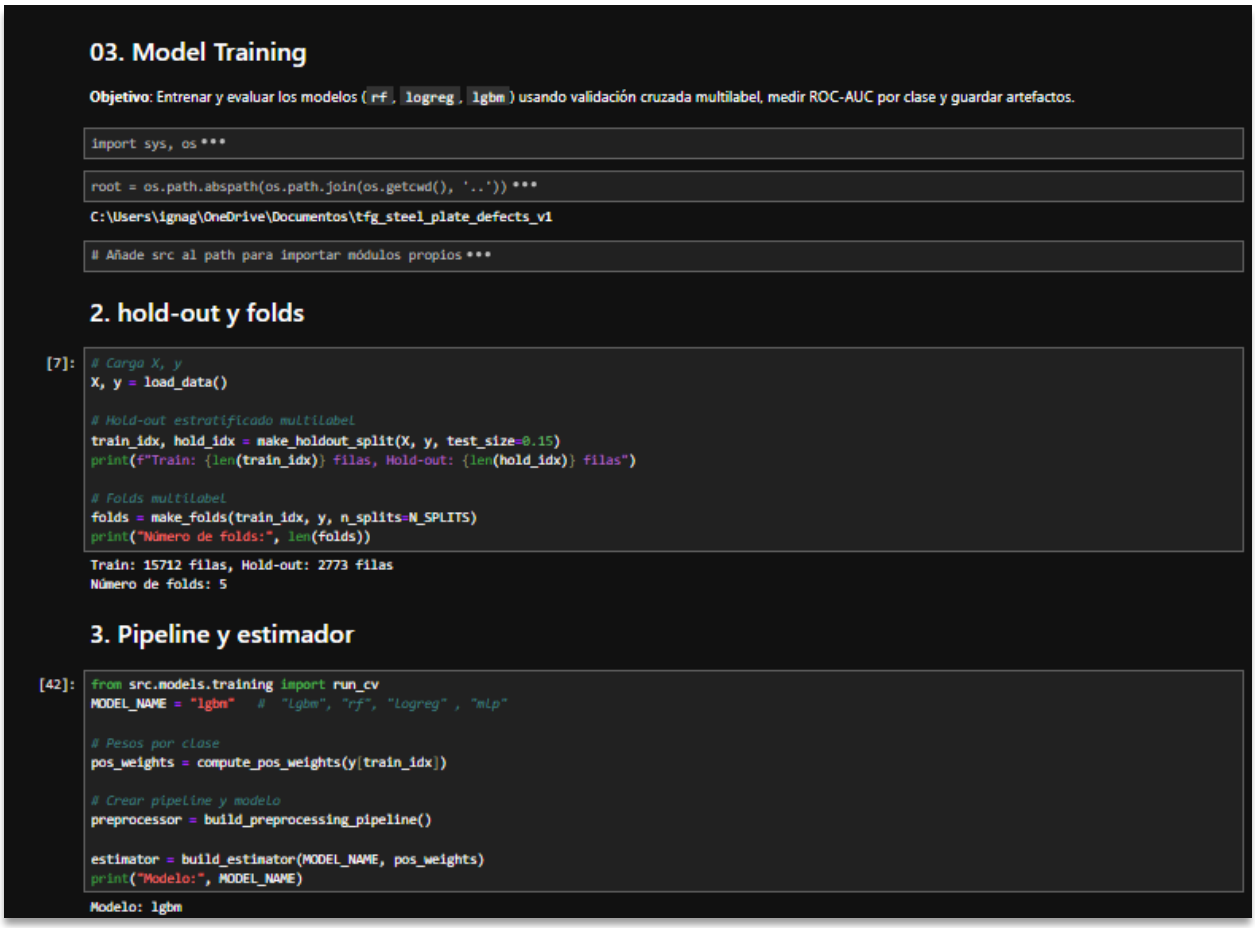


Figura 0-4. Edges_index transformación cuántilica.

Nota: En src/features/transformer.py se ha usado transformación log1p

B.3 Notebook 03_model_training.ipynb



03. Model Training

Objetivo: Entrenar y evaluar los modelos (`rf` , `logreg` , `lgbm`) usando validación cruzada multilabel, medir ROC-AUC por clase y guardar artefactos.

```
import sys, os
```

```
root = os.path.abspath(os.path.join(os.getcwd(), '..'))
```

```
C:\Users\ignag\OneDrive\Documentos\tfg_steel_plate_defects_v1
```

```
# Añade src al path para importar módulos propios
```

2. hold-out y folds

[7]:
```
# Carga X, y
X, y = load_data()

# Hold-out estratificado multilabel
train_idx, hold_idx = make_holdout_split(X, y, test_size=0.15)
print(f"Train: {len(train_idx)} filas, Hold-out: {len(hold_idx)} filas")

# Folds multilabel
folds = make_folds(train_idx, y, n_splits=N_SPLITS)
print("Número de folds:", len(folds))
```

```
Train: 15712 filas, Hold-out: 2773 filas
Número de folds: 5
```

3. Pipeline y estimador

[42]:
```
from src.models.training import run_cv
MODEL_NAME = "lgbm"   #  "lgbm", "rf", "logreg" , "mlp"

# Pesos por clase
pos_weights = compute_pos_weights(y[train_idx])

# Crear pipeline y modelo
preprocessor = build_preprocessing_pipeline()

estimator = build_estimator(MODEL_NAME, pos_weights)
print("Modelo:", MODEL_NAME)
```

```
Modelo: lgbm
```

Figura 0-5. Hold-out inicial, generación de folds y selección del modelo.

5. Resultados y visualización

[23]:
```
# Mostrar resultados completos (incluye train_time_sec para referencia)
display(df_cv)

# Media global de ROC-AUC
mean_auc = df_cv['mean_auc'].mean()
print(f"Media global ROC-AUC: {mean_auc:.4f}")

# Boxplot de AUC por clase (sin la columna de tiempo)
plt.figure(figsize=(8, 5))
classes = [c for c in df_cv.columns
           if c not in ['fold', 'mean_auc', 'train_time_sec']]   # ← excluimos tiempo
df_cv[classes].boxplot(rot=45)
plt.title(f"Distribución de ROC-AUC por clase ({MODEL_NAME})")
plt.ylabel('ROC-AUC')
plt.tight_layout()
plt.show()
```

	Pastry	Z_Scratch	K_Scatch	Stains	Dirtiness	Bumps	Other_Faults	mean_auc	fold	train_time_sec
0	0.850482	0.892724	0.971409	0.980371	0.823877	0.728380	0.679200	0.846635	0	8.617254
1	0.855261	0.891132	0.968038	0.972644	0.835826	0.768329	0.692264	0.854785	1	8.601610
2	0.863521	0.911325	0.976434	0.987185	0.801813	0.768093	0.676456	0.854975	2	8.641689
3	0.878699	0.870036	0.979439	0.985118	0.756076	0.748055	0.675086	0.841787	3	8.547228
4	0.836906	0.871901	0.979623	0.988287	0.867560	0.759250	0.680617	0.854878	4	8.487252

```
Media global ROC-AUC: 0.8506
```

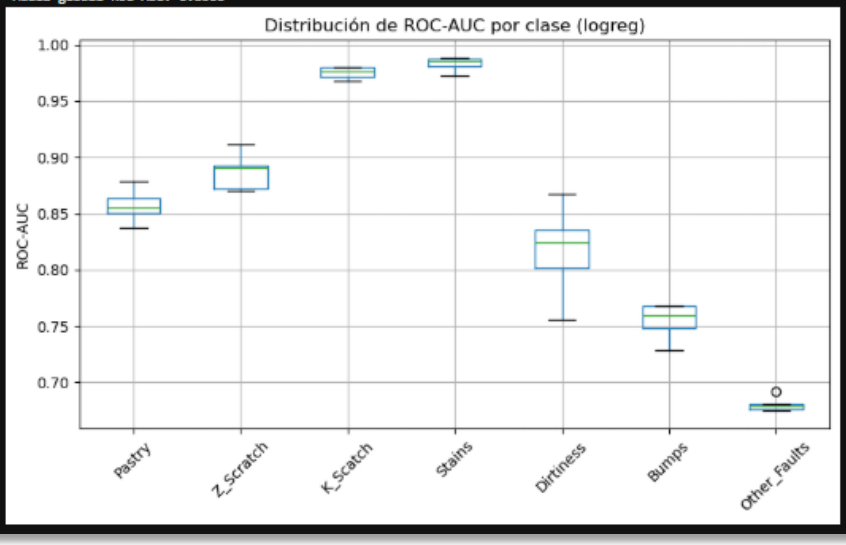


Figura 0-6. Distribución ROC-AUC por defecto en el modelo regresión logística

B.4 04_hyperparameter_optimization.ipynb

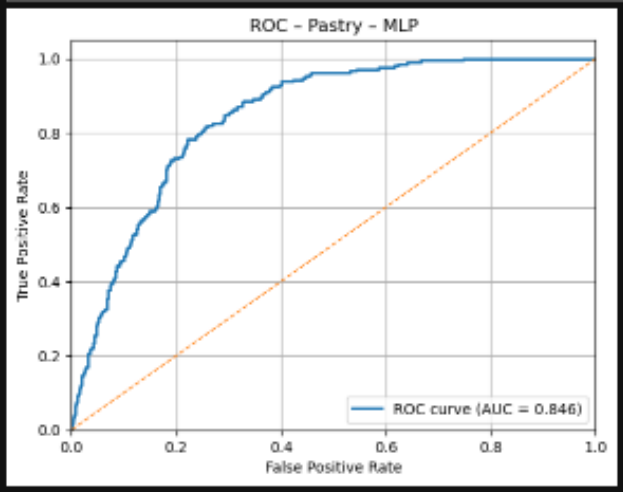
04. Hyper-parameter Optimization (Optuna)

Notebook dedicado exclusivamente a la búsqueda de hiperparámetros para los modelos multilabel de *Steel Plate Defect Prediction*.

- Dependencias clave: `optuna`, `lightgbm`, `scikit-learn`, `hpo.py` (este repo).
- Resultado: archivos `best_params_<modelo>.json` + base de datos `.db` con el estudio completo dentro de `models/hpo/`.

Metrica de optimización: Curva ROC-AUC.

[35]:
```
# ------------------------------------------------------------------
MODEL_NAME = "mlp"      # "lgbm", "rf", "logreg", "mlp"
LABEL_ID   = 0          # 0=Pastry, 1=Z_Scratch, ..., 6=Other_Faults
N_SPLITS   = 1          # entrenamos en todo el train salvo hold-out
# ------------------------------------------------------------------

# 1. Carga y split básico -----------------------------------------------
X, y = load_data()
train_idx, test_idx = make_holdout_split(X, y)
y_train, y_test = y[train_idx], y[test_idx]

# 2. Preprocesado + modelo one-vs-rest para la etiqueta escogida ---------
pos_weights = compute_pos_weights(y_train)
preproc     = build_preprocessing_pipeline()

# solo la etiqueta que nos interesa
clf = build_estimator(MODEL_NAME, pos_weights)[LABEL_ID]

# Ajuste
X_train_p = preproc.fit_transform(X.iloc[train_idx])
X_test_p  = preproc.transform(X.iloc[test_idx])
clf.fit(X_train_p, y_train[:, LABEL_ID])

# 3. Predicciones y curva ROC -------------------------------------------
y_score = clf.predict_proba(X_test_p)[:, 1]
fpr, tpr, _ = roc_curve(y_test[:, LABEL_ID], y_score)
roc_auc = auc(fpr, tpr)

# 4. Plot ---------------------------------------------------------------
plt.figure()
plt.plot(fpr, tpr, lw=2, label=f"ROC curve (AUC = {roc_auc:.3f})")
plt.plot([0, 1], [0, 1], lw=1, linestyle="--")  # línea aleatoria
plt.xlim([0.0, 1.0]); plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.title(f"ROC – {TARGET_COLS[LABEL_ID]} – {MODEL_NAME.upper()}")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



[36]:
```
optuna.logging.set_verbosity(optuna.logging.WARNING)
```

Figura 0-7. Curva ROC-AUC para defecto pastry en el modelo perceptrón multicapa

B.2 05_feature_importance.ipynb

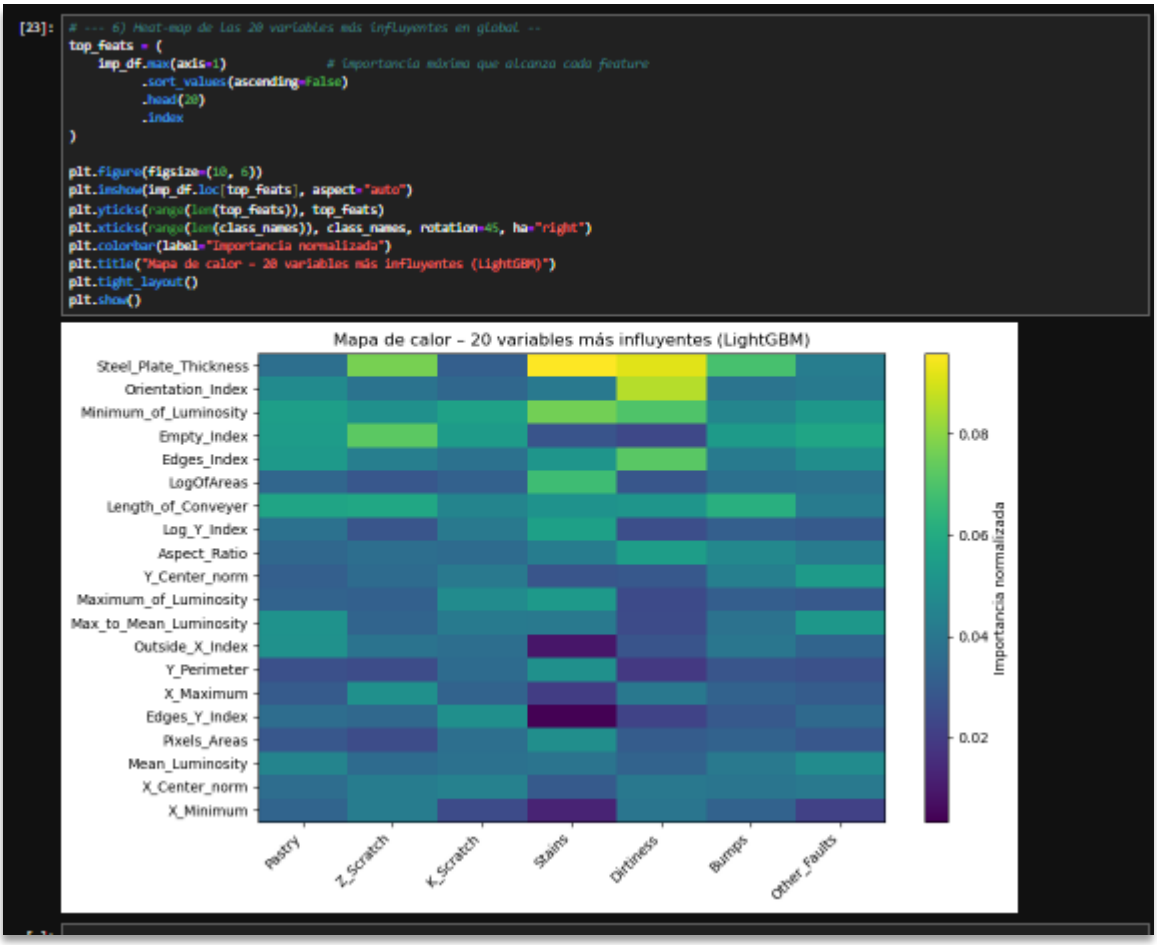


Figura 0-8. Mapa de calor con las variables más importantes

ANEXO C. LIBRERÍAS UTILIZADAS

Tabla 0–1 Librerías de Python empleadas en el proyecto

Tipo	Paquete / Módulo (versión)	Rol dentro del proyecto
Lenguaje	**python 3.9**	Intérprete principal
Biblioteca estándar	argparse, glob, logging, re, sys, datetime, pathlib, json, typing, dataclasses, inspect, warnings	Utilidades nativas para CLI, registro de mensajes, expresiones regulares, fechas, rutas, serialización JSON, tipado, etc. (no requieren instalación)
Cálculo numérico	numpy 2.0.2	Operaciones vectoriales y álgebra lineal
Datos tabulares	pandas 2.2.3	Carga y manipulación de datos
Modelado clásico	scikit-learn 1.6.1	Modelos, métricas, pipelines
Gradient boosting	lightgbm 4.6.0	Modelos de boosting
Optimización	optuna 4.2.1	Búsqueda de hiperparámetros
Desbalanceo	imbalanced-learn 0.12.4	Técnicas de resampling
CV multilabel	iterative-stratification 0.1.9	Splits estratificados multilabel
Visualización	matplotlib 3.9.4, seaborn 0.13.2	Gráficos EDA, curvas ROC, etc.
Serialización	joblib 1.4.2	Guardar y cargar modelos