[1. Fundamentación Teórica y Estado del Arte 1](#_Toc459688056)

[1.1. Detección de defectos en la industria del acero 1](#_Toc1402083987)

[1.2. Aprendizaje supervisado y clasificación multi-etiqueta 1](#_Toc88877230)

[1.3. Algoritmos relevantes (Random Forest, otros métodos) 1](#_Toc899498022)

[1.4. Trabajos previos y referentes en la literatura 1](#_Toc244820993)

[2. Entendiendo el Dataset 1](#_Toc1127783626)

[2.1. Origen y fuentes de los datos (Competición Kaggle) 1](#_Toc642876884)

[2.2. Descripción de las variables y etiquetas de defecto 1](#_Toc1487980566)

[2.3. Naturaleza sintética del dataset y consideraciones especiales 1](#_Toc1234356700)

[2.4. Formato de los datos y requisitos de la competición 1](#_Toc1027571478)

[3. Preparación del Entorno 1](#_Toc1998631354)

[3.1. Librerías y versiones de Python utilizadas 1](#_Toc607018965)

[3.2. Entorno de desarrollo (Anaconda, notebooks, etc.) 1](#_Toc1933514586)

[4. Análisis Exploratorio de Datos (EDA) 1](#_Toc857798182)

[4.1. Estadísticas descriptivas 2](#_Toc1006124455)

[4.2. Visualizaciones de distribución y correlación 2](#_Toc1350130431)

[4.3. Identificación y tratamiento de valores faltantes 2](#_Toc1921473421)

[4.4. Detección de outliers y calidad de datos 2](#_Toc888513426)

[5. Preprocesamiento 2](#_Toc1389927668)

[5.1. Limpieza y transformación de los datos 2](#_Toc850339335)

[5.2. Ingeniería de características (feature engineering) 2](#_Toc479636866)

[5.3. Manejo del desbalance de clases 2](#_Toc206216698)

[5.4. Particionado de datos (entrenamiento, validación, test) 2](#_Toc242108988)

[6. Modelado 2](#_Toc1043273594)

[6.1. Algoritmos considerados 2](#_Toc2140003877)

[6.2. Implementación de la clasificación multi-etiqueta 2](#_Toc1755652388)

[6.3. Random Forest y otros modelos comparados 2](#_Toc17684963)

[6.4. Selección de parámetros iniciales 2](#_Toc498932729)

[7. Optimización 2](#_Toc1742287433)

[7.1. Tuning de hiperparámetros con Optuna 3](#_Toc573008301)

[7.2. Selección de características (feature selection) 3](#_Toc839460029)

[7.3. Estrategias adicionales de optimización (regularización, ensembles, etc.) 3](#_Toc163407153)

[8. Evaluación 3](#_Toc2033613951)

[8.1. Métricas clave (AUC, exactitud, etc.) 3](#_Toc926636786)

[8.2. Procedimiento de validación (k-fold, hold-out, etc.) 3](#_Toc1618240792)

[8.3. Resultados obtenidos y comparación entre modelos 3](#_Toc353159056)

[8.4. Interpretación y análisis de importancia de variables 3](#_Toc674903774)

[9. Conclusiones y Trabajo Futuro 3](#_Toc76032738)

[9.1. Discusión de hallazgos y limitaciones 3](#_Toc309819206)

[9.2. Aplicaciones prácticas e implicaciones en la industria 3](#_Toc1993507172)

[9.3. Líneas de mejora y posibles continuaciones 3](#_Toc168251680)

[10. Referencias 3](#_Toc1600058863)

# 1. Fundamentación Teórica y Estado del Arte

## 1.1. Detección de defectos en la industria del acero

La industria siderúrgica ha implementado diversas estrategias a lo largo del tiempo para detectar defectos en productos de acero. Inicialmente, la inspección manual visual era la forma predominante de control de calidad, apoyada en normas y criterios humanos para identificar imperfecciones superficiales. Con el avance de la automatización y la informática, surgieron métodos automatizados basados en sensores y visión artificial para analizar datos estructurados (mediciones numéricas derivadas del proceso o de imágenes procesadas). Antes del auge del *deep learning*, muchas soluciones empleaban técnicas de procesamiento digital de señales e imágenes para extraer características (por ejemplo, índices de forma, medidas de luminancia, transformadas en frecuencia) y luego aplicaban algoritmos de clasificación convencionales. Entre los enfoques clásicos se incluyen técnicas de segmentación por umbrales, transformadas de Fourier y filtros de Gabor para resaltar discontinuidades, así como métodos estadísticos para detectar desviaciones en texturas. Gradualmente, estas estrategias dieron paso al uso de la inteligencia artificial clásica (aprendizaje automático tradicional) en los sistemas de control de calidad.

A finales de la década de 1990 y comienzos de los 2000, empezaron a explorarse algoritmos de aprendizaje supervisado sobre datos estructurados para mejorar la detección de defectos. Se introdujeron métodos como árboles de decisión, redes neuronales de una capa oculta y máquinas de vectores de soporte (SVM) entrenados con atributos numéricos que describían la superficie o propiedades físicas de las placas de acero. Estos primeros sistemas “inteligentes” eran aún limitados, pero sentaron las bases de la automatización del control de calidad. Hacia la década de 2010, con la disponibilidad de conjuntos de datos públicos y la madurez de técnicas de *data mining*, aumentó el interés investigador en aplicar algoritmos más avanzados. Un ejemplo notable es el conjunto de datos *Steel Plates Faults* donado en 2010 al repositorio UCI, el cual contiene mediciones físicas (27 atributos) de placas de acero y su tipo de defecto asociado. Desde su publicación, este conjunto ha sido ampliamente utilizado como banco de pruebas para distintos clasificadores en la literatura. En estos datos, cada muestra está caracterizada por medidas como índices de forma, luminosidad y dimensiones de la placa, que permiten inferir el tipo de falla de fabricación presente. (1)

Los enfoques tradicionales para detección de fallas con datos tabulares se basaban en modelos supervisados “poco profundos”, en contraposición a las redes neuronales profundas actuales. Por ejemplo, se aplicaron técnicas de análisis de varianza y control estadístico de procesos para identificar factores que condujeran a defectos, complementadas con algoritmos de clasificación para predecir automáticamente el tipo de defecto. Con el tiempo, modelos de aprendizaje automático genéricos como bosques aleatorios, SVM o redes neuronales de pocas capas demostraron ser eficaces para capturar patrones en las mediciones numéricas de producción y detectar defectos con mayor consistencia que la inspección humana. Estudios previos señalan que, además de métodos basados en procesamiento de imágenes (segmentación espacial/frecuencial), se emplearon modelos de clasificación automática “clásicos” como **Random Forest, SVM y redes neuronales superficiales** para identificar defectos en superficies metálicas. Estos algoritmos permitían aprovechar la información estructurada (por ejemplo, propiedades geométricas o físicas medidas por sensores) para discernir si una pieza presentaba defectos y de qué tipo, reduciendo la dependencia en reglas fijas o inspección visual. No obstante, dichos métodos tradicionales a menudo se optimizaban para escenarios específicos y su exactitud dependía en gran medida de las características predefinidas por expertos. Esto motivó investigaciones posteriores para encontrar enfoques más generales y robustos.

## 1.2. Aprendizaje supervisado y clasificación multi-etiqueta

En problemas clásicos de clasificación de defectos, cada pieza o producto suele asignarse a una única clase de defecto predominante (o a la clase "sin defectos"). Sin embargo, en el entorno industrial del acero es común que una misma pieza presente múltiples defectos simultáneamente. Por ejemplo, una lámina de acero podría exhibir a la vez rayaduras (*scratches*) y manchas, o deformaciones y suciedad. En estos casos, la suposición de exclusividad de clase de los clasificadores *multi-class* tradicionales resulta inadecuada. Surge así la necesidad de enfoques de clasificación *multi-etiqueta*, donde a cada instancia (placa de acero inspeccionada) se le pueden asignar varios rótulos de defecto entre un conjunto de categorías posibles.

La clasificación multi-etiqueta se diferencia de la multi-clase en que no obliga a elegir una única categoría; por el contrario, permite predecir un subconjunto de categorías relevantes por muestra. Cada etiqueta se interpreta como una variable binaria (defecto presente/ausente), y el objetivo del modelo es predecir correctamente la combinación de defectos presentes. Este enfoque es especialmente relevante en acero u otras manufacturas donde los defectos no son mutuamente excluyentes. Desde un punto de vista operativo, identificar *todas* las fallas de una pieza es crucial para tomar acciones correctivas completas (por ejemplo, reprocesar una plancha con grietas y oxidación requiere abordar ambas imperfecciones).

Históricamente, cuando los sistemas no soportaban multi-etiqueta, la solución era introducir categorías compuestas o genéricas. Un ejemplo es la clase "Otros" empleada en algunos conjuntos de datos industriales para agrupar piezas que no encajan limpiamente en un solo tipo de defecto. En el caso de las placas de acero, la categoría “Other\_Faults” se usó para designar piezas con características atípicas o múltiples defectos a la vez. Si bien esto permitía clasificar *algo* dichas piezas, el resultado era impreciso pues no informaba qué defectos específicos estaban presentes. La clasificación multi-etiqueta aborda este problema evitando categorías cajón de sastre; en lugar de etiquetar una muestra ambigua simplemente como "Otros", el modelo multi-etiqueta ideal indicaría, por ejemplo: Defectos = {Rayadura, Manchas} en una misma observación. De esta forma, se obtiene una descripción más rica y accionable de la calidad del producto.

Otro motivo para la relevancia de la multi-etiqueta es la posibilidad de explotar correlaciones entre defectos. En la práctica, ciertos defectos pueden tender a ocurrir juntos debido a causas comunes en el proceso de fabricación. Por ejemplo, una desalineación en el laminado podría causar simultáneamente rayaduras y abolladuras; conocer esta co-ocurrencia puede ayudar tanto en la predicción (el modelo aprende patrones combinados) como en el diagnóstico de causas raíz. La literatura de aprendizaje multi-etiqueta enfatiza la importancia de considerar la dependencia entre etiquetas para mejorar la precisión predictiva. No obstante, incluso un enfoque sencillo de multi-etiqueta que trate cada defecto por separado ya proporciona la ventaja fundamental de detectar todos los tipos de fallas presentes en una muestra. En resumen, en contextos industriales donde la realidad de las piezas defectuosas es compleja, la clasificación multi-etiqueta es necesaria para reflejar fielmente el estado de cada producto y garantizar un control de calidad integral.

## 1.3. Algoritmos relevantes (Random Forest, otros métodos)

El abordaje de un problema multi-etiqueta puede realizarse mediante la adaptación de algoritmos de clasificación tradicionales o mediante métodos diseñados específicamente para múltiples etiquetas. En el caso de datos tabulares con decenas de atributos (como medidas físicas, químicas o geométricas), muchos de los algoritmos clásicos de aprendizaje supervisado han demostrado ser eficaces, ya sea tratando cada etiqueta por separado o modelando todas las etiquetas conjuntamente. A continuación, se revisan los principales algoritmos empleados y sus consideraciones en tareas multi-etiqueta relevantes a defectos de acero:

* **Bosques aleatorios (*Random Forest*)**: Consisten en múltiples árboles de decisión entrenados sobre muestras aleatorias de datos y subconjuntos de atributos. Para problemas multi-etiqueta, una estrategia común es entrenar un modelo por cada etiqueta (bosque aleatorio para cada tipo de defecto) o utilizar variantes de árboles que soportan múltiples salidas. Los Random Forest se han utilizado ampliamente en la detección de defectos por su robustez ante datos ruidosos y su capacidad de manejar atributos heterogéneos. En la literatura de fallas en acero, los bosques aleatorios han sobresalido en rendimiento; por ejemplo, Nkonyana et al. (2019) reportan que Random Forest superó a SVM y redes neuronales en la clasificación de 7 tipos de defectos [mdpi.com](https://www.mdpi.com/2075-1702/11/7/679#:~:text=Nkonyana%20et%20al.%20,70). Su naturaleza de ensemble tiende a generar buenos resultados generales, y al manejar cada árbol una selección de características reduce el riesgo de sobreajuste. En contexto multi-etiqueta, los bosques pueden extenderse fácilmente a predecir múltiples etiquetas, obteniendo una predicción binaria por defecto, típicamente usando criterios de división adaptados (como la entropía multi-etiqueta) o simplemente construyendo modelos independientes por etiqueta.
* **Máquinas de Gradient Boosting**: Métodos de *boosting* como **XGBoost** o **LightGBM** han ganado popularidad por lograr alta exactitud en datos tabulares. Estos algoritmos construyen secuencialmente modelos (árboles de decisión generalmente) donde cada nuevo árbol corrige errores del conjunto anterior. En clasificación multi-etiqueta, se suelen aplicar en un esquema de *binary relevance*, es decir, entrenando un modelo de boosting separado para cada etiqueta de defecto. Pese a tratar cada etiqueta individualmente, su elevado poder predictivo puede resultar en un desempeño competitivo, y debido a la rapidez y eficiencia de implementaciones modernas (LightGBM destaca en grandes volúmenes de datos), son viables para problemas industriales. Una ventaja es que manejan bien datos desbalanceados mediante ajuste de pesos o objetivos personalizados, algo útil si ciertos defectos son mucho más raros que otros. Si bien no existe una modificación específica de XGBoost/LightGBM exclusiva para multi-etiqueta en tabular, combinados con enfoques de transformación del problema (por ejemplo, entrenar varios clasificadores o utilizar estrategias de *classifier chains*), pueden captar algunas dependencias entre etiquetas. En investigaciones recientes de defectos de acero, métodos de boosting han sido incluidos entre los mejores: Agrawal y Adane (2022) obtuvieron aproximadamente un 76% de precisión usando Random Forest y técnicas basadas en árboles optimizados (*PCA-based decision tree forest*), demostrando la eficacia de los ensambles de árboles en esta tarea
* **Regresión logística (multi-etiqueta)**: La regresión logística es un clasificador lineal ampliamente utilizado por su simplicidad e interpretabilidad. En un escenario multi-etiqueta, se puede entrenar un modelo logístico binario por cada defecto, produciendo para cada instancia una probabilidad de presencia del defecto. Esta estrategia asume independencia entre defectos (es un enfoque de primer orden), pero sirve como línea base razonable. Su ventaja es la rapidez y la capacidad de calibrar probabilidades, aunque puede quedarse corta si las relaciones entre atributos y la ocurrencia de defectos son no lineales o complejas. Aun así, como parte de un sistema de detección, la logística puede emplearse para cada etiqueta y luego combinando las salidas (por ejemplo, etiquetando como defectuoso aquel tipo cuya probabilidad supere cierto umbral). Cabe señalar que la calibración de umbrales es importante en modelos logísticos multi-etiqueta, para decidir a partir de qué probabilidad se considera que un defecto está presente. Este modelo, aunque no fue el más destacado en trabajos previos sobre placas de acero, sí ha sido utilizado en otros contextos industriales multi-etiqueta por su interpretabilidad.
* **Máquinas de Vectores de Soporte (SVM) multi-etiqueta**: Las SVM han sido un pilar en la clasificación de defectos debido a su capacidad para manejar espacios de alta dimensión y encontrar fronteras de decisión óptimas. No obstante, una SVM estándar produce un clasificador binario o multi-clase; para extenderla a multi-etiqueta se suele emplear igualmente la táctica de *uno contra todos* (entrenar una SVM independiente por etiqueta). Investigaciones sobre defectos en acero muestran resultados competitivos de SVM en configuraciones multi-clase tradicionales, llegando a precisiones del orden de 73-75% en la predicción de la etiqueta de defecto principal. En multi-etiqueta, cada SVM puede concentrarse en distinguir la presencia o ausencia de un tipo de defecto específico. Existen también formulaciones más avanzadas como Rank-SVM, donde la SVM se entrena para ordenar etiquetas relevantes sobre irrelevantes, incorporando cierto manejo de correlación; sin embargo, dichas variantes son menos comunes en la práctica industrial por su complejidad. Por lo general, la simplicidad de entrenar múltiples SVM binarias funciona adecuadamente, aunque a costa de ignorar la interdependencia entre defectos. Un punto a considerar es el costo computacional: entrenar muchas SVM puede ser pesado si el dataset es gr ande, pero con 5–10 etiquetas (como en nuestro caso de 7 defectos) es factible. Algunos trabajos recientes propusieron variaciones de SVM adaptadas; por ejemplo, Shu et al. (2023) emplean SVM con una técnica de “extended decision label annotation (ELA)”, logrando alrededor de 77% de exactitud en la clasificación de defectos. Esto sugiere que ligeras modificaciones en la forma de anotar o presentar las etiquetas a la SVM pueden mejorar su rendimiento en contextos multi-etiqueta complejos.
* **k-Nearest Neighbors adaptado (ML-kNN)**: El algoritmo de *k vecinos más cercanos* (kNN) puede extenderse a multi-etiqueta mediante adaptaciones como **ML-kNN**. En este enfoque, para una muestra dada, se buscan sus k vecinos más similares en el conjunto de entrenamiento y luego se determina la probabilidad de cada etiqueta en función de cuántos de esos vecinos poseen dicha etiqueta. Zhang y Zhou (2007) propusieron ML-kNN incorporando inferencia bayesiana para estimar la pertenencia a cada clase dada la vecindad. A diferencia del kNN tradici onal (que votaría por una única clase mayoritaria), ML-kNN calcula para cada posible etiqueta una puntuación de pertenencia considerando la frecuencia de esa etiqueta entre los k vecinos y una modelización de a priori. Este método es sencillo de implementar y a veces sirve de referencia, aunque puede verse afectado por la **dimensionalidad y la escala** de los atributos (problema común en kNN). En datos tabulares de manufactura, si el número de atributos es moderado (decenas) y están bien normalizados, ML-kNN puede capturar patrones locales útiles, especialmente cuando existen combinaciones frecuentes de defectos. No es de los algoritmos más reportados en el ámbito específico de placas de acero, pero es conceptualmente importante en la caja de herramientas de clasificación multi-etiqueta.
* **Redes Neuronales (Perceptrón Multicapa)**: Una red neuronal artificial de tipo perceptrón multicapa (MLP) puede manejar naturalmente salidas multi-etiqueta al tener múltiples neuronas de salida con funciones de activación sigmoide (para producir una probabilidad entre 0 y 1 por etiqueta). De hecho, en aprendizaje profundo es común tratar tareas multi-label de esta forma. En datos tabulares, una red neuronal de unas pocas capas densas puede aprender relaciones no lineales entre los 35 atributos y las 7 etiquetas de defecto. La ventaja de las redes es su flexibilidad para modelar interacciones complejas entre atributos; incluso pueden captar correlaciones entre etiquetas si se les provee la señal adecuada (por ejemplo, usando una función de pérdida conjunta como entropía cruzada binaria sumada para todas las etiquetas, que incentiva la predicción correcta de todo el vector de etiquetas). Trabajos previos en la industria del acero han probado redes neuronales simples: por ejemplo, Mary (2018) utilizó una red neuronal de retropropagación logrando alrededor de 75% de acierto, mientras que Zhang et al. (2018) alcanzaron ~77% con una red similar. Estos resultados, en línea con SVM y árboles, mostraban que las redes densas tradicionales podían rendir bien, aunque sin superarlos claramente. Con el auge de frameworks modernos, hoy es sencillo construir un modelo neuronal multi-etiqueta y ajustarlo con técnicas de regularización (dropout, etc.) para evitar sobreajuste en datasets estructurados relativamente pequeños. Un aspecto a vigilar es la interpretabilidad: a diferencia de árboles de decisión o regresiones, las redes son cajas negras, lo cual puede ser relevante en entornos industriales donde se valora entender qué factores contribuyen a una decisión de defecto.
* **Otros enfoques**: Además de los anteriores, existen métodos específicos de clasificación multi-etiqueta dignos de mención. Las técnicas de transformación del problema, como Binary Relevance (entrenar un clasificador independiente por etiqueta) y Classifier Chains (entrenar una secuencia de clasificadores donde cada uno incorpora las predicciones de etiquetas anteriores como características), ofrecen formas simples pero efectivas de abordar la multi-etiqueta. Binary Relevance es muy utilizado por su simplicidad y fue implícitamente la estrategia adoptada en muchos trabajos (entrenando, por ejemplo, una SVM por defecto, un árbol por defecto, etc.). Los Classifier Chains, propuestos en 2010, buscan explotar correlaciones encadenando las etiquetas; aunque poderosos, requieren cuidado para evitar propagar errores en la cadena. Por otro lado, existen algoritmos que modifican directamente el algoritmo base para manejar conjuntos de etiquetas, como variantes de árboles de decisión con medidas de impureza *multi-label* (ej. calculando entropía conjunta de varias etiquetas), o métodos basados en descomposición de errores (por ejemplo, Calibrated Label Ranking que convierte la tarea en varios problemas de ranking binario). En el ámbito de defectos industriales, la mayoría de aplicaciones han preferido algoritmos estándar con adaptaciones sencillas (por su facilidad de implementación en entornos productivos), pero los avances en investigación ofrecen una gama amplia de opciones para mejorar la calidad de las predicciones multi-etiqueta.

## 1.4. Trabajos previos y referentes en la literatura

La detección automática de defectos en el acero ha sido objeto de numerosos estudios, especialmente a partir de datos estructurados como el conjunto *Steel Plates Faults*. A continuación se sintetiza la evolución de resultados y enfoques en la literatura, haciendo hincapié en aquellos con rigor académico en inglés o español.

**Primeros estudios (2010–2015):** Tras la disponibilidad de datos públicos, investigadores comenzaron evaluaciones comparativas de algoritmos. Thirukovalluru et al. (2016) aplicaron SVM a la clasificación de defectos de placas de acero, obteniendo precisiones en torno al 75%. Estos primeros esfuerzos mostraron que métodos basados en máquinas de soporte vectorial eran viables. Sin embargo, la necesidad de explorar más técnicas quedó patente dado que la exactitud no superaba el 80%. Paralelamente, algunos estudios exploraron árboles de decisión y ensambles: por ejemplo, se empleó el algoritmo C4.5 y sus variantes, aunque con resultados aún en el rango medio de 70%. En esta época inicial también se publicaron trabajos con **redes neuronales simples**; Mary (2018) reportó ~75.3% de acierto con una red de retropropagación, y Zhang et al. (2018) lograron ~77.3% con una red neuronal feed-forward, comparándola además con un árbol CART que alcanzó ~79%[mdpi.com](https://www.mdpi.com/2075-1702/11/7/679#:~:text=Mohamed%20et%20al.%20,2016%20Support%20vector%20machine%2075.27). Esto sugería que los **árboles de decisión** (posiblemente debido a su capacidad de manejar no linealidades e interacciones en datos tabulares) ya despuntaban ligeramente sobre otros métodos individuales.

**Avances con métodos de ensemble (2016–2020):** Hacia finales de la década de 2010, los **métodos de conjunto (ensembles)** ganaron protagonismo. Diversos trabajos mostraron que combinar múltiples clasificadores o árboles mejoraba la robustez de la predicción. Nkonyana et al. (2019) llevaron a cabo una evaluación extensa con **Random Forest, SVM y redes neuronales**, encontrando que el Random Forest producía el mejor desempeño (77.8% de exactitud) en la identificación de los 7 tipos de defectos. En la misma línea, Srivastava (2019) probó algoritmos como árboles de decisión, bosques aleatorios, AdaBoost y kNN, obteniendo su mayor precisión con Random Forest (~79.4%) seguido de cerca por AdaBoost (~78.4%) . Estos resultados confirmaron la efectividad de los ensambles basados en árboles para capturar las complejas relaciones en los datos de fabricación del acero. Asimismo, algunos autores combinaron técnicas de preprocesamiento y selección de características: Zhang et al. (2018) aplicaron un filtro *mRMR* (mínima redundancia, máxima relevancia) antes de entrenar un árbol de decisión, logrando hasta 79.3% de acierto, lo que indica que una depuración de variables irrelevantes puede beneficiar la detección de defectos. En 2021, Mohamed y Samsudin exploraron algoritmos base (Naive Bayes, kNN, árbol de decisión) con estrategias de mejora como información gain o algoritmos evolutivos, obteniendo resultados en el rango de 66–72%; estos confirmaron que métodos más simples requieren optimización adicional para acercarse al rendimiento de ensambles.

**Enfoques recientes y aprendizaje profundo (2020–2023):** Con la irrupción del *deep learning*, era natural intentar aplicarlo también a datos tabulares de manufactura, aunque este dominio no siempre saca el mayor provecho de las redes profundas debido al tamaño de los datos y la naturaleza de los atributos. Agrawal y Adane (2022) investigaron el uso de una red LSTM (memoria a corto-largo plazo) para predecir defectos, encontrando un desempeño de ~75.6%, similar al de árboles y bosques tradicionales. Adicionalmente, propusieron ensambles de árboles (PDTF e I-PDTF en sus siglas en inglés) integrando técnicas de reducción de dimensiones, con pequeñas mejoras hasta ~76.1%. Estos trabajos sugieren que, para datos estructurados de tamaño moderado, las técnicas de *machine learning* clásicas competen fuertemente con redes profundas, a menos que se disponga de muchos más datos o se generen atributos adicionales. Un aspecto destacado en investigaciones recientes es el manejo de la mencionada clase "Other\_Faults". Shu et al. (2023) introducen el concepto de **anotación extendida de etiquetas (Extended Label Annotation, ELA)** para enriquecer la información de entrenamiento de un SVM y un árbol C4.5, logrando aproximadamente 77.5% de exactitud. Aunque los detalles de ELA exceden este resumen, la mejora sugiere que enriquecer la forma en que se presentan las etiquetas (posiblemente indicando la presencia de múltiples defectos o subclases) ayuda al modelo a aprender patrones más diferenciados, atacando el problema de la heterogeneidad dentro de la clase "otros".

Por otra parte, la literatura en visión por computador ha abordado la detección multi-etiqueta de defectos en acero utilizando imágenes (p.ej., conjuntos NEU-DET o Severstal), aplicando redes convolucionales y transformers. Si bien esos trabajos se enfocan en imágenes, comparten con nuestro caso la necesidad de identificar múltiples defectos concurrentes. Najafi et al. (2022) por ejemplo, utilizaron un Vision Transformer para clasificación multi-etiqueta de defectos superficiales en acero, enfrentando retos de desbalance e interpretación de resultados. Esto evidencia que el concepto de multi-etiqueta está tomando relevancia en distintos frentes del control de calidad industrial.

**Estado del arte actual:** Los continuos esfuerzos por mejorar la precisión llevaron recientemente a propuestas novedosas. Yilmaz Eroglu y Guleryuz (2025) presentaron un algoritmo ensemble basado en árboles de decisión con regresión logística integrados (denominado **Logistic Model Tree Forest**). Al evaluar su modelo en el conjunto de fallas de placas de acero, alcanzaron una exactitud promedio del **86.7%**, superando en más de 7 puntos porcentuales al Random Forest tradicional (~79.5%). Este avance significativo se logró mediante la combinación de clasificadores múltiples y la mitigación del sesgo debido al desbalance de clases (aplicaron un preprocesamiento ENN para equilibrar la clase *Other\_Faults* con las demás). El resultado marca un nuevo hito en la detección de defectos con datos tabulares, acercando el rendimiento a niveles muy altos y demostrando el beneficio de innovar sobre algoritmos clásicos para adaptarlos mejor al dominio.

En síntesis, la literatura académica muestra una evolución desde métodos sencillos hacia enfoques más complejos y personalizados al problema de defectos en acero. Inicialmente, técnicas como SVM, árboles o redes de una capa lograban resultados decentes (~70-75%). Con la introducción de ensambles y el refinamiento de características, se rozó el 80% de precisión a finales de la década de 2010. Experimentos con deep learning puro no superaron claramente ese umbral, lo que sugiere que para la escala de datos disponible, los métodos basados en árboles y ensambles seguían siendo la opción preferente. Finalmente, las contribuciones más recientes incorporan estrategias híbridas (combinar modelos lineales con árboles, o técnicas de etiquetado mejoradas) que han elevado sustancialmente el techo de desempeño. Esto subraya la importancia de un análisis exhaustivo del problema: comprender la naturaleza multi-etiqueta, el desbalance entre tipos de defectos y las relaciones entre atributos ha permitido diseñar soluciones de aprendizaje supervisado cada vez más efectivas para la industria del acero.

Referencias seleccionadas: Cabe destacar la diversidad de fuentes académicas consultadas, incluyendo artículos en inglés y español. Entre ellos se encuentran revisiones generales de aprendizaje multi-etiqueta, estudios específicos sobre detección de fallas en acero con distintos algoritmos, y desarrollos modernos publicados en revistas internacionales que abordan mejoras sustanciales en este campo. Estas referencias brindan un sustento teórico y práctico al análisis, asegurando que el contenido presentado esté alineado con el conocimiento científico actual y las mejores prácticas documentadas en la detección de defectos mediante aprendizaje supervisado y clasificación multi-etiqueta.

# 2. Entendiendo el Dataset

## 2.1. Origen y fuentes de los datos (Competición Kaggle)

El conjunto de datos procede de la competición *Steel Plate Defect Prediction* de la Tabular Playground Series – Season 4, Episode 3 organizada en Kaggle. El reto estuvo abierto del 1 de marzo al 31 de marzo de 2024 (23:59 UTC) y forma parte de los desafíos mensuales diseñados para practicar técnicas de *machine learning* con datos tabulares sintéticos.([kaggle.com](https://www.kaggle.com/competitions/playground-series-s4e3?))

## 2.2. Descripción de las variables y etiquetas de defecto

* **Tamaño del fichero de entrenamiento**: **19 219 filas × 35 columnas**; una de ellas es un identificador (id), **27** son *features* numéricas (coordenadas, áreas, perímetros, transformaciones logarítmicas, índices normalizados, etc.) y **7** columnas son objetivos binarios que indican la presencia de cada defecto.([surajwate.com](https://surajwate.com/blog/steel-plate-defect-prediction/?), [ejurnal.ubharajaya.ac.id](https://ejurnal.ubharajaya.ac.id/index.php/jiforty/article/download/2485/1692/6339?))
* **Tamaño del fichero de prueba**:**28 columnas** (mismos atributos pero sin las etiquetas).
* **Etiquetas de defecto**
  1. Pastry
  2. Z\_Scratch
  3. K\_Scatch
  4. Stains
  5. Dirtiness
  6. Bumps
  7. Other\_Faults

## 2.3. Naturaleza sintética del dataset y consideraciones especiales

El organizador generó los registros de forma **sintética** a partir de distribuciones inspiradas en casos reales de inspección de placas de acero. Esto garantiza la ausencia de información confidencial y un equilibrio razonable entre complejidad y ligereza (≈ 19 k registros). Sin embargo, implica dos cuestiones clave:

1. **Transferencia limitada** a entornos industriales reales: las señales pueden ser más limpias y los patrones más marcados que en producción.
2. **Evaluación académica**: al desconocer la “historia” real detrás de cada *feature*, resulta esencial explicar las transformaciones y la interpretación de los modelos con cautela.

## 2.4. Formato de los datos y requisitos de la competición

* **Formato original**: CSV con separador de coma, codificación UTF‑8 y encabezados.
* **Archivo de envío**: CSV con las columnas id + las siete probabilidades estimadas, en idéntico orden que las etiquetas, 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.([kaggle.com](https://www.kaggle.com/competitions/playground-series-s4e3?utm_source=chatgpt.com))

Con esta información se establece un marco claro para el análisis exploratorio y el diseño del pipeline de *machine learning* descritos en los capítulos posteriores.

# 3. Preparación del Entorno

## 3.1. Librerías y versiones de Python utilizadas

## 3.2. Entorno de desarrollo (Anaconda, notebooks, etc.)

Se listan las herramientas y librerías clave (p. ej., NumPy, pandas, scikit-learn, Optuna, Matplotlib) y cómo se configuró el entorno de trabajo.

# 4. Análisis Exploratorio de Datos (EDA)

## 4.1. Estadísticas descriptivas

El análisis exploratorio comienza con un resumen estadístico de todas las variables numéricas presentes en el conjunto de entrenamiento. Se han calculado los principales estadísticos descriptivos: conteo total de observaciones (N), media, mediana, mínimo, máximo, desviación típica, percentiles 25% y 75%, así como el rango intercuartílico (IQR). Esto permite identificar rápidamente si existen variables con valores extremos, sesgos evidentes en su distribución o escalas muy diferentes que puedan requerir normalización.

A continuación, se muestra la tabla resumen de estadísticas numéricas. Es importante destacar cualquier variable que presente alta dispersión (por ejemplo, desviaciones estándar elevadas respecto a la media), presencia de valores mínimos o máximos inusuales o posibles errores de captura. En estos casos, se han marcado las celdas con color para facilitar su detección.

Por otro lado, se ha calculado también la distribución de clases para las siete etiquetas de defecto: Pastry, Z\_Scratch, K\_Scatch, Stains, Dirtiness, Bumps y Other\_Faults. Cada una de estas etiquetas es binaria, representando la presencia o ausencia del defecto correspondiente. La tabla de balance de clases muestra el número total de piezas defectuosas por tipo y su porcentaje relativo respecto al total de muestras.

Este balance de clases permite observar que el conjunto de datos presenta un **desbalance significativo**, con algunos defectos (como Dirtiness o Stains) apareciendo en un porcentaje muy bajo del total de piezas, mientras que otros (Bumps o Other\_Faults) son algo más frecuentes. Esta característica tendrá un impacto directo en la estrategia de evaluación y entrenamiento, ya que será necesario utilizar técnicas que gestionen adecuadamente este desbalance, como ponderación de clases o técnicas de oversampling.

En conjunto, este análisis proporciona una primera visión clara de la estructura general de los datos y sienta las bases para identificar problemas potenciales antes del modelado.

## 4.2. Visualizaciones de distribución y correlación

Una vez obtenido el resumen estadístico, se procede al análisis gráfico de las distribuciones univariantes y las relaciones entre variables. Este apartado tiene como objetivo identificar posibles sesgos, multimodalidades, transformaciones necesarias (como logaritmos o escalados), y descubrir patrones de correlación que puedan influir en el rendimiento de los modelos.

**Distribuciones univariantes**

Se han generado histogramas con curvas de densidad (KDE) para cada variable numérica del conjunto de entrenamiento. Estos gráficos permiten visualizar la forma de cada distribución: si es simétrica, sesgada hacia la derecha o izquierda, bimodal, o si presenta colas largas. Algunas de las variables presentan distribuciones **asimétricas** y con **colas prolongadas**, lo que sugiere que una transformación logarítmica podría ser útil para mejorar la normalidad de los datos.

En muchas de las variables se observan acumulaciones de valores en los extremos, lo que puede indicar presencia de outliers o límites técnicos (por ejemplo, sensores que saturan en ciertos valores máximos).

**Boxplots por clase de defecto**

Para explorar cómo se relaciona cada variable con la presencia de defectos, se han generado diagramas de caja (boxplots) comparando la distribución de cada característica según si una pieza presenta o no un defecto determinado (uno vs resto). Estos gráficos ayudan a identificar variables que podrían ser predictivas, al mostrar diferencias significativas entre grupos.

Por ejemplo, en el caso del defecto K\_Scatch, se observan diferencias claras en la mediana y el rango intercuartílico de ciertas variables respecto a las piezas sin ese defecto. Este tipo de hallazgos son útiles para orientar el feature selection y el diseño del modelo.

**Matriz de correlación**

Finalmente, se ha generado un mapa de calor (heatmap) con las correlaciones de Pearson entre todas las variables numéricas. Este tipo de análisis permite detectar variables altamente correlacionadas entre sí, lo que puede indicar redundancia de información y justificar la aplicación de técnicas como reducción de dimensionalidad (PCA) o eliminación de colinealidades.

En este dataset se identifican varios pares de variables con correlaciones superiores a 0.8 en valor absoluto, lo cual deberá tenerse en cuenta a la hora de construir modelos lineales o árboles con regularización. Por otro lado, algunas variables muestran muy baja correlación con el resto, lo cual puede indicar que aportan información única o simplemente ruido.

Este análisis visual complementa los estadísticos numéricos anteriores y proporciona una base sólida para las siguientes etapas de ingeniería de características y selección de variables.

## 4.3. Identificación de valores faltantes

El análisis de valores faltantes es una etapa fundamental dentro del preprocesamiento de datos, ya que la presencia de datos ausentes puede afectar negativamente la calidad del modelo si no se maneja adecuadamente. En este paso se identifican las variables con registros nulos, se cuantifica su impacto y se proponen estrategias de imputación o descarte.

**Identificación y cuantificación**

Se ha generado una tabla que muestra, para cada variable, el número absoluto y el porcentaje de registros faltantes respecto al total. Esta información permite priorizar las variables más afectadas por la ausencia de datos y descartar aquellas cuya falta de información supera un umbral crítico (por ejemplo, 30% o más).

Adicionalmente, se ha generado un mapa de calor (heatmap) de valores faltantes, que permite visualizar patrones sistemáticos: por ejemplo, si varios campos tienden a faltar al mismo tiempo, lo que podría indicar un problema específico en la captura de datos de ciertas etapas del proceso industrial.

En este conjunto concreto, Al ser una base de datos sintética. Las variables no presentan ningún valor faltante, aunque se ha decido mantener el mapa de calor para futuro uso.

## 4.4. Detección de outliers

La calidad de los datos es un aspecto clave en cualquier proyecto de machine learning, ya que errores, inconsistencias o valores extremos pueden deteriorar tanto el proceso de entrenamiento como la capacidad de generalización del modelo. Este apartado se centra en la detección de valores atípicos (outliers) y en la evaluación general de la integridad del conjunto de datos.

**Detección de valores atípicos**

Para identificar valores atípicos se ha aplicado la regla clásica basada en el rango intercuartílico (IQR). Esta técnica considera como outliers aquellos valores que se encuentran por debajo de Q1 - 1.5 × IQR o por encima de Q3 + 1.5 × IQR, donde Q1 y Q3 son los percentiles 25 y 75 de la distribución, respectivamente.

La tabla correspondiente muestra el número de valores atípicos detectados por variable. Este análisis permite identificar columnas con una cantidad desproporcionada de extremos, que podrían corresponder a errores de medición, datos mal etiquetados, o bien condiciones reales pero inusuales del proceso industrial.

Además del enfoque univariado, se recomienda en etapas posteriores aplicar métodos multivariantes como Isolation Forest o Local Outlier Factor (LOF), especialmente si se detectan interacciones complejas entre variables que generan observaciones inusuales sólo al considerarse en conjunto.

**Evaluación de la calidad de los datos**

Además de los outliers, se ha llevado a cabo una revisión de la calidad general del dataset, atendiendo a los siguientes aspectos:

* **Consistencia de tipos y formatos**: se verificó que no haya columnas con mezclas de tipos (por ejemplo, numérico y texto), ni valores mal codificados.
* **Rangos válidos y plausibles**: se comprobaron los límites de las variables físicas para asegurarse de que los valores estén dentro de lo razonable según el dominio industrial.
* **Duplicados**: se examinó la presencia de filas duplicadas, que podrían surgir por errores de recolección o agregación de datos.
* **Errores sistemáticos**: no se detectaron columnas con desplazamientos, saturaciones o patrones constantes sospechosos que indiquen fallos en sensores o sistemas de entrada.

**Tratamiento aplicado**

El tratamiento de los valores atípicos se ha realizado con criterios específicos según el impacto detectado:

* **Capping / Winsorización**: en variables con outliers leves pero frecuentes, se ha optado por limitar los valores extremos al percentil 1 y 99 para evitar su influencia desproporcionada.
* **Eliminación selectiva de registros**: sólo en casos en los que se confirmó que un valor era erróneo y no representativo del fenómeno real.
* **Conservación justificada**: en situaciones donde el valor extremo reflejaba condiciones de operación válidas, aunque poco frecuentes.

Toda acción ha sido documentada de forma transparente, y el conjunto de datos resultante ha sido almacenado en la carpeta data/interim/ para asegurar la trazabilidad entre los datos originales y los modificados.

## 4.5. Conclusiones clave del EDA

**Ausencia de valores faltantes y limpieza previa**

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

**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 o transformaciones tipo Yeo–Johnson, seguidas de escalado robusto. Para GBDT bastará un capado ligero (winsorización P1–P99).

**Presencia de outliers legítimos**

* Entre el 10–20 % de los valores de áreas, perímetros y luminosidad caen fuera del rango IQR estándar.
* Dado que representan defectos grandes o muy brillantes, no se eliminarán; en su lugar, se harán transformaciones suaves (log + winsorización) para preservar la señal.

**Desbalance multilabel y su impacto**

* La prevalencia varía de 34 % (*Other\_Faults*) a 2,5 % (*Dirtiness*).
* La métrica del concurso (media de AUC-ROC por clase) exige que las clases minoritarias no queden penalizadas; se plantea:
  + *Iterative stratification* en train/valid.
  + Uso de *class weights* y/o oversampling (SMOTE, RandomOverSampler) focalizado en las minoritarias.
  + Posible uso de Focal Loss o pérdidas equilibradas.

 **Grupos de features redundantes**

* **Coordenadas** (X\_Minimum–X\_Maximum, Y\_Minimum–Y\_Maximum) con correlación ~ |r| ≈ 1 → se recomiendan las variables derivadas Width, Height, X\_Center, Y\_Center.
* **Área–Perímetro–Luminosidad** altamente correlacionados → conservar versión *log* de área, un perímetro agregado y Mean\_Luminosity = Sum / Area.
* **Índices geométricos** (Edges\_\*, Outside\_\*, Square\_Index, Orientation\_Index) forman un subespacio correlacionado; se mantendrán 2–3 ejemplares tras evaluación de importancia.

**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.

**Acciones de feature engineering derivadas**

* Transformaciones de escala y winsorización para atenuar la asimetría sin perder los valores extremos.
* Nuevas variables:
  + Width, Height, Aspect\_Ratio, X\_Center\_norm, Y\_Center\_norm,
  + Mean\_Luminosity, Max\_to\_Mean\_Luminosity,
  + indicadores binarios de “muy grande” (por encima de P95).
* Evaluar reducción de redundancia con VIF o PCA solo en los grupos muy correlacionados.

**Preparación para modelado**

* Montar un ColumnTransformer reproducible que incorpore todas las transformaciones anteriores.
* Validar con k-fold estratificado multilabel, monitorizando AUC-ROC y PR-AUC por clase.
* Documentar en el TFG la relación entre cada transformación y su beneficio empírico (importancias, ROC univariadas).

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.

# 5. Preprocesamiento

## 5.1. Limpieza y transformación de los datos

Para homogeneizar las escalas, atenuar asimetrías y conservar la señal de casos extremos, implementamos el siguiente flujo:

1. **Carga y verificación inicial**
   * Leemos data/raw/playground-series-s4e3/train.csv y test.csv en pandas.DataFrame.
   * Confirmamos en el apartado anterior que no existen valores nulos en ninguna columna (df.isna().sum() == 0), por lo que no es necesaria ninguna imputación previa.
2. **Generación de features derivadas**
   * Aplicamos FeatureGenerator(img\_width=1700, img\_height=12900000) para añadir:
   * Estas variables sintetizan posición, forma y intensidad, y quedan registradas junto a las originales.
3. **Winsorización de colas**
   * Instanciamos Winsorizer(cols=LOG\_COLS, lower\_pct=1, upper\_pct=99), donde]
   * En fit(), el transformador calcula percentiles P1 y P99 de cada columna; en transform(), recorta los valores por debajo de P1 y por encima de P99.
   * De esta forma preservamos los casos de defectos muy grandes o brillantes, limitando su influencia sobre la varianza de modelos sensibles.
4. **Transformación logarítmica**
   * Sobre el resultado del winsorizado, aplicamos LogTransformer(cols=LOG\_COLS), que ejecuta np.log1p en las mismas cinco columnas.
   * Esta operación reduce drásticamente la asimetría de sus distribuciones, como se ilustra en los boxplots comparativos (notebook 02).
5. **Escalado cuantílico**
   * Finalmente, utilizamos QuantileTransformer(output\_distribution="normal", random\_state=42) para todas las variables continuas.
   * Cada feature se transforma de modo que su distribución empírica se aproxime a una Gaussiana estándar, lo cual mejora la convergencia de modelos lineales y redes neuronales.
6. **Implementación en pipeline**
   * Encapsulamos todas las etapas anteriores en un único objeto de sklearn.pipeline.Pipeline, definido en src/pipeline/preprocessing.py:
   * Este pipeline garantiza que el mismo flujo —features derivadas, winsorización, log y escalado— se aplique de forma idéntica en fase de entrenamiento, validación y test, evitando discrepancias o fugas de datos.

Con estas etapas hemos conseguido un conjunto de variables con:

* Rango controlado (colas recortadas sin eliminar registros).
* Distribuciones aproximadamente normales en las continuas.
* Nuevas features que capturan posición, forma y brillo en formatos comparables.

Este preprocesamiento establece una base sólida y reproducible para el modelado, tal como se detallará en el apartado 5.5.

## 5.3. Manejo del desbalance de clases

En nuestro problema multilabel, la frecuencia de las etiquetas varía significativamente: de ≈ 34 % en *Other\_Faults* a ≈ 2.5 % en *Dirtiness*. Dado que la métrica oficial —la media de ROC-AUC por clase— otorga igual peso a cada etiqueta, es crucial evitar que las clases minoritarias queden infrarrepresentadas en el aprendizaje. A continuación, detallamos las estrategias implementadas:

1. **Estratificación multilabel**
   * Utilizamos la librería iterstrat y su clase IterativeStratification(n\_splits=5, order=1) para dividir el dataset de entrenamiento en 5 folds.
   * Cada fold mantiene la misma proporción de positivos y negativos en las 7 etiquetas, garantizando que tanto clases mayoritarias como minoritarias estén representadas en cada partición.
2. **Pesos de clase en el entrenamiento**
   * Calculamos pesos inversamente proporcionales a la prevalencia de cada etiqueta k:
   * En LightGBM se pasa scale\_pos\_weight = N\_negative / N\_positive; en XGBoost scale\_pos\_weight análogo; en modelos de scikit-learn se usa el parámetro class\_weight; y en Keras, el diccionario class\_weight de model.fit().
3. **Oversampling focalizado**
   * Para reforzar el aprendizaje de las clases raras, aplicamos **RandomOverSampler** de imblearn con sampling\_strategy="not majority", duplicando ejemplos de todas las clases que no sean la mayoritaria hasta acercarlas al 50 % de la clase más frecuente.
   * De manera alternativa, probamos **SMOTE** para sintetizar muestras nuevas en el espacio de features continuas, y **MLSMOTE** (de smote\_variants) cuando era necesario conservar dependencias multilabel.
4. **Pérdidas equilibradas y focal loss**
   * En redes neuronales experimentamos con **Focal Loss** (γ ≈ 2) para penalizar menos los ejemplos fáciles de clases mayoritarias y concentrarnos en los difíciles de las minoritarias.
   * Asimismo, evaluamos la **Class-Balanced Loss** (Cui et al., 2019) que ajusta la contribución de cada clase según su número de ejemplos.
5. **Monitorización y validación por clase**
   * Durante el entrenamiento, registramos por separado el **ROC-AUC** y **PR-AUC** para cada etiqueta en el set de validación.
   * Aceptamos una estrategia de muestreo o pesos solo si **no degrade** el desempeño en las clases menos frecuentes (por ejemplo, que la AUC de *Dirtiness* no caiga por debajo de 0.75).

Con este conjunto de prácticas aseguramos que el modelo dedique capacidad de aprendizaje a todas las etiquetas de forma equilibrada, mejorando la robustez y evitando el sesgo hacia las clases más abundantes.

## 5.4. Particionado de datos (entrenamiento, validación, test)

Para garantizar evaluaciones rigurosas y evitar fugas de información, seguimos un procedimiento en dos etapas: separación de un conjunto de test independiente y particionado estratificado multilabel para entrenamiento y validación.

1. **Hold-out inicial para test final**
   * Reservamos un 15 % del dataset original como **conjunto de test final**, que no participa en ningún ajuste de parámetros ni optimización.
   * Resultado:
     + **X\_rem, y\_rem** (85 % datos) → para entrenamiento/validación
     + **X\_test, y\_test** (15 % datos) → evaluación final sin tocar.
2. **K-fold estratificado multilabel**
   * Sobre X\_rem/y\_rem, aplicamos **IterativeStratification** del paquete iterstrat:
   * Este método preserva de forma aproximada las proporciones de positivos y negativos en cada etiqueta en todos los folds.
3. **Fijar semillas y guardar splits**
   * Utilizamos random\_state=42 en todas las funciones que admiten semilla para garantizar reproducibilidad.
   * Tras generar los índices de cada fold, los guardamos en disco:
   * Esto permite reconstruir exactamente las mismas particiones si fuera necesario.
4. **Integración en el pipeline de entrenamiento**
   * En src/models/training.py disponemos de una función train\_and\_evaluate() que, para cada fold:
     + Carga los índices guardados.
     + Extrae X\_train, y\_train, X\_val, y\_val.
     + Aplica el preprocessor (fit en entrenamiento, transform en validación).
     + Ajusta el modelo y calcula métricas por clase.
   * Ejemplo esquemático:

python

Con este esquema de particionado, combinamos un **test final** completamente aislado con un **k-fold** que respeta las características multilabel de nuestros datos, asegurando que las métricas de validación reflejen un rendimiento realista y sin sesgos de muestreo

## 5.5. Pipelines y reproducibilidad

Para asegurar que todo el flujo de preprocesamiento y modelado sea replicable, transparente y fácil de compartir en el TFG, hemos adoptado las siguientes prácticas:

1. **Modularización del código**
   * **Transformadores independientes**  
     ­– src/features/transformers.py contiene clases especializadas (FeatureGenerator, Winsorizer, LogTransformer) que implementan cada paso de limpieza y generación de features.
   * **Construcción del pipeline**  
     ­– src/pipeline/preprocessing.py expone la función build\_preprocessing\_pipeline(), que devuelve un objeto sklearn.pipeline.Pipeline con las etapas ordenadas.
   * **Entrenamiento acoplado al pipeline**  
     ­– En src/models/training.py se importa el preprocesador y se encadena con el modelo elegido (e.g., LightGBM, CatBoost) usando un segundo Pipeline o aplicando manualmente fit\_transform y transform.
2. **Gestión de dependencias**
   * El fichero environment.yml especifica versiones concretas de Python y librerías clave:
   * requirements.txt complementa con paquetes PyPI adicionales (e.g., smote\_variants, tensorflow-addons si se usa Focal Loss).
3. **Persistencia de artefactos**
   * **Split indices**  
     ­– Índices de train/validation para cada fold se guardan con joblib.dump en la carpeta splits/, de modo que siempre podamos reproducir las mismas particiones.
   * **Preprocessor y modelos**  
     ­– El objeto preprocessor y los modelos finales se serializan (joblib.dump) en models/preprocessor.pkl y models/model\_fold\_{i}.pkl.
   * **Resultados de validación**  
     ­– Métricas de cada fold (ROC-AUC, PR-AUC por etiqueta) se vuelcan en CSV dentro de reports/tables/ para su análisis y comparación.

# 6. Modelado

## 6.1. Algoritmos considerados

Para abordar la predicción multilabel de defectos en placas de acero, hemos seleccionado un conjunto de algoritmos que cubren distintas familias de modelos y que, a partir de los hallazgos del EDA y del preprocesamiento, se ajustan bien a los retos del problema: heterogeneidad de escalas, asimetrías marcadas, multicolinealidad y desbalance de clases.

1. **Random Forest**
   * **Justificación**:
     + Puede manejar directamente variables con escalas dispares y valores extremos tras la winsorización, sin necesidad de transformar todas las features a distribuciones normales.
     + Tolerante a la multicolinealidad: construye árboles sobre subconjuntos aleatorios de variables, reduciendo el riesgo de sobreajuste por features redundantes (áreas vs. perímetros, índices geométricos).
     + Fácil de interpretar mediante importancias de variable agregadas y parciales, lo que facilita documentar en el TFG cómo cada transformación (e.g. log-scale, ratios de luminosidad, features de posición) aporta señal.
   * **Uso en multilabel**: se configurará con class\_weight="balanced" para cada clasificador binario (One-vs-Rest).
2. **Gradient Boosting Machines (LightGBM / XGBoost)**
   * **Justificación**:
     + Capturan patrones no lineales e interacciones complejas sin necesidad de generar manualmente polinomios o interacciones — clave dado que el EDA mostró correlaciones débiles con las etiquetas (|r| < 0.35).
     + Manejan bien escalas dispares y outliers moderados; recomendamos aplicar un capado ligero y mantener también versiones log-transformadas para que el modelo elija la representación más informativa.
     + Permiten asignar scale\_pos\_weight o pesos de clase específicos por etiqueta, lo que resulta fundamental para no penalizar las clases raras (*Dirtiness*, *Stains*).
   * **Configuración inicial**: se probarán valores moderados de learning\_rate (0.05–0.1), num\_leaves (31–127) y max\_depth (–1 o entre 6 y 12), junto a los pesos calculados según prevalencia.
3. **Regresión logística (con regularización Elastic-Net)**
   * **Justificación**:
     + Sirve como **baseline lineal** para cuantificar cuánto aportan las transformaciones y las features derivadas en comparación con un modelo simple.
     + La penalización L1/L2 ayuda a reducir la multicolinealidad y seleccionar automáticamente un subconjunto de variables — útil para validar la relevancia de Aspect\_Ratio, Mean\_Luminosity o las flags de “muy grande”.
     + En combinación con StandardScaler o QuantileTransformer, puede ofrecer resultados competitivos en clases mayores, aunque no capture completamente las interacciones no lineales de las minoritarias.
   * **Implementación**: se utilizará LogisticRegression(solver='saga', penalty='elasticnet', l1\_ratio=0.5, class\_weight='balanced').
4. **Redes neuronales feed-forward (FFNN)**
   * **Justificación**:
     + Permiten diseñar arquitecturas personalizadas y utilizar pérdidas especializadas (Focal Loss, Class-Balanced Loss) para centrar el entrenamiento en las clases más difíciles.
     + Con lot-normalization y *dropout*, pueden gestionar la heterogeneidad de escala y aprender representaciones latentes que combinen información de tamaño, forma y luminosidad.
     + Actúan como contrapunto al GBDT, ya que su rendimiento depende más de las transformaciones y del preprocesamiento numérico.
   * **Configuración inicial**: 2–3 capas ocultas (64–128 unidades), activación ReLU, optimizador Adam con tasa de aprendizaje 1e-3, batch size 256.

Cada uno de estos algoritmos se integrará dentro de un esquema **One-vs-Rest**, de modo que se entrene un clasificador binario por etiqueta, usando el mismo pipeline de preprocesamiento (features derivadas, winsorización, log y escalado cuantílico). De esta manera garantizamos comparabilidad directa entre modelos y podemos justificar, en la sección de resultados, cómo cada familia de algoritmos aprovecha las transformaciones y maneja los desafíos específicos de este TFG.

## 6.2. Implementación de la clasificación multi-etiqueta

Para abordar el problema de predecir simultáneamente siete defectos en placas de acero, se optó por una estrategia por “problem transformation” multilabel. Con ella, cada etiqueta se trata como una tarea de clasificación binaria independiente, manteniendo a la vez la capacidad de evaluación conjunta mediante la métrica promedio de ROC-AUC.

* **MultiOutputClassifier (scikit-learn)**  
  Este envoltorio (MultiOutputClassifier) aloja un clasificador base que se entrena de forma paralela para cada una de las siete etiquetas. La ventaja de este enfoque es su simplicidad de uso y su integración directa con el API de scikit-learn, permitiendo aplicar sin cambios los métodos habituales (fit, predict\_proba, score).
* **Clasificadores independientes en LightGBM**  
  Dado que LightGBM permite ajustar el parámetro scale\_pos\_weight por etiqueta para compensar desbalances, en lugar de usar MultiOutputClassifier con LGBM se emplea una lista de siete objetos LGBMClassifier. Cada uno se entrena individualmente, recibiendo su propio valor de scale\_pos\_weight (proporción de negativos respecto a positivos) calculado en el conjunto de entrenamiento. Esto maximiza la flexibilidad al tratar etiquetas muy desbalanceadas sin perder el paralelismo en la evaluación de métricas multilabel.

En todos los casos, la evaluación intermedia se realiza mediante validación cruzada estratificada multilabel (utilizando **IterativeStratification**), lo que garantiza que cada pliegue mantiene proporciones similares de positivos y negativos en cada una de las siete clases.

## 6.3. Random Forest y otros modelos comparados

Se seleccionaron tres familias de modelos para comparar rendimiento y robustez:

1. **Random Forest**
   * Conformado por un ensamble de árboles de decisión entrenados sobre bootstrap samples.
   * Maneja de forma nativa datos de entrada con distribuciones no gaussianas y es resistente al ruido de características.
   * Se usa class\_weight="balanced\_subsample" para reequilibrar cada árbol según la frecuencia de etiquetas en su muestra.
2. **Regresión Logística**
   * Implementada con solver “saga” y penalización elasticnet para combinar L1 (selección de variables) y L2 (estabilidad).
   * Su linealidad la hace muy interpretable y rápida de entrenar, pero depende de un escalado adecuado de las variables y no captura interacciones complejas sin ingeniería previa.
3. **LightGBM**
   * Algoritmo de boosting basado en histogramas optimizado para gran velocidad y consumo de memoria.
   * Ajusta automáticamente interacciones de segundo orden y maneja datos categóricos de forma natural.
   * Se configura un clasificador por etiqueta, cada uno con scale\_pos\_weight específico, y parámetros base de 500 iteraciones y tasa de aprendizaje moderada (0.05).

Cada modelo aporta un perfil distinto: Random Forest aporta robustez general, Regresión Logística interpretabilidad y LightGBM eficiencia y capacidad de modeling de patrones complejos.

## 6.4. Selección de parámetros iniciales

Antes de la etapa de optimización fina (descrita en el siguiente apartado), se fijaron valores base de hiperparámetros con criterios de experiencia y recomendaciones de la literatura:

* **Random Forest**
  + n\_estimators = 300: número de árboles suficiente para estabilizar la curva de generalización sin incurrir en coste computacional excesivo.
  + max\_depth = None: permite una profundización hasta que todas las hojas estén puras, mitigada por min\_samples\_leaf implícito en scikit-learn.
* **Regresión Logística**
  + solver = "saga", max\_iter = 2000: garantiza convergencia incluso con penalización mixtas y múltiples etiquetas.
  + penalty = "elasticnet", l1\_ratio = 0.5: equilibrio inicial entre selección de variables y regularización suave.
* **LightGBM**
  + n\_estimators = 500, learning\_rate = 0.05, num\_leaves = 63: valores comunes para datasets de tamaño medio, que ofrecen un buen trade-off entre profundidad de cada árbol y velocidad de aprendizaje.
  + subsample = 0.8, colsample\_bytree = 0.8: muestreo parcial de instancias y características para reducir overfitting.

Los pesos de clase (scale\_pos\_weight) se calcularon de forma inversamente proporcional a la prevalencia de cada defecto en el conjunto de entrenamiento, garantizando que las instancias minoritarias tuvieran un impacto equivalente durante el ajuste. Con estos parámetros iniciales se completó la fase de benchmarking, antes de lanzar la búsqueda de hiperparámetros avanzada.

# 7. Optimización

## 7.1. Tuning de hiperparámetros con Optuna

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

1. **Función objetivo multinivel**
   * Cada “trial” de Optuna define un conjunto de valores para los hiperparámetros del modelo (p. ej. número de estimadores, profundidad máxima, tasa de aprendizaje, parámetros de regularización, etc.).
   * Se ejecuta validación cruzada estratificada multilabel (IterativeStratification) en *n* folds (por defecto 5), ajustando el pipeline de preprocesamiento **fit\_transform** sólo en los índices de entrenamiento de cada fold y transformando el conjunto de validación con **transform**.
   * Para LightGBM en particular, se genera un clasificador independiente por etiqueta, cada uno con su propio scale\_pos\_weight calculado según la prevalencia de esa etiqueta en el entrenamiento.
2. **Métrica de optimización**
   * En cada fold se calcula la ROC-AUC para cada una de las siete etiquetas, y se promedia (“macro‐average”).
   * El valor de la función objetivo es la media de esas 5 ROC-AUC promedio de fold. De este modo, Optuna maximiza directamente la métrica oficial de la competición.
3. **Sampler y poda temprana**
   * Se usa **TPESampler** con semilla fija para garantizar reproducibilidad en la búsqueda.
   * **MedianPruner** conduce pruning tras un número mínimo de folds completados, descartando configuraciones que ya no puedan superar la mejor prueba actual y reduciendo el coste computacional.
4. **Gestión de resultados**
   * Cada estudio persiste su base de datos en SQLite (reports/hpo/hpo\_{modelo}.db), permitiendo reanudar búsquedas interrumpidas.
   * Al finalizar se vuelcan en JSON los mejores parámetros (best\_params\_{modelo}.json), que luego se integran en el pipeline de entrenamiento final.

Con este esquema, la búsqueda de hiperparámetros queda automatizada, reproducible y centrada en maximizar la ROC-AUC promedio multilabel.

## 7.2. Selección de características (feature selection)

A pesar de contar con un conjunto de variables derivadas y escaladas, se exploran técnicas de reducción de dimensionalidad para mejorar la interpretabilidad y, en ocasiones, la generalización:

1. **Importancia de variables basada en modelo**
   * Se entrena un Random Forest (o un LightGBM base) con los parámetros iniciales y se extrae la importancia de cada feature (feature\_importances\_).
   * Se prueban dos umbrales:
     + **Umbral fijo** (p. ej. retirar variables con importancia < 0.01).
     + **Umbral dinámico** (eliminar el 10 % de features menos relevantes).
2. **Métodos wrapper**
   * **Recursive Feature Elimination (RFE)** con un estimador base ligero (p. ej. regresión logística con L1) para seleccionar de forma iterativa el subconjunto óptimo que maximice la ROC-AUC en validación.
   * **Sequential Feature Selector** (hacia adelante y hacia atrás) para comparar rendimiento incremental.
3. **Métodos embebidos**
   * Regularización L1 en modelos lineales (Lasso) para forzar coeficientes a cero y descartar variables irrelevantes de forma automática.
   * En LightGBM/XGBoost, usar parámetros de penalización (reg\_alpha, reg\_lambda) para inducir sparsity en la estructura del árbol.

Cada una de estas técnicas se valida mediante el mismo procedimiento de validación cruzada multilabel, comparando la métrica de ROC-AUC promedio antes y después de la reducción de features. Se escoge finalmente el método que ofrezca la mejor relación entre parsimonia y rendimiento.

## 7.3. Estrategias adicionales de optimización (regularización, ensembles, etc.)

Además del HPO y la selección de características, se aplican técnicas complementarias:

1. **Regularización y calibración**
   * En redes neuronales se experimenta con **Focal Loss** y **Class-Balanced Loss** para penalizar de forma adaptativa a las clases minoritarias.
   * Se evalúa también **Calibración de probabilidades** (Platt Scaling, Isotonic Regression) sobre el conjunto de validación, mejorando la calidad de las predicciones de probabilidad.
2. **Ensembles y stacking**
   * **VotingClassifier** y **StackingClassifier** de scikit-learn combinan Random Forest, Logistic Regression y LightGBM en un meta-modelo.
   * El meta-modelo se entrena sobre las predicciones de probabilidad de cada base learner, optimizando nuevamente la ROC-AUC promedio multilabel.
3. **Bagging y Boosting combinados**
   * **Bagging sobre LightGBM** (Bootstrap LightGBM): ejecutar múltiples LightGBM con diferente seed y promediar sus predicciones.
   * **Gradient-based One-Side Sampling (GOSS)** y **Exclusive Feature Bundling (EFB)** en LightGBM para acelerar la convergencia y reducir overfitting.
4. **Optimización de thresholds**
   * Dado que la métrica final es probabilística, se analiza la posibilidad de ajustar umbrales de decisión por clase (p. ej. maximizar F1-score en validación) y evaluar el impacto en la ROC-AUC promedio.

Estas estrategias se aplican de forma modular al pipeline final, seleccionando únicamente aquellas que incrementan de manera significativa la AUC media multilabel sin sacrificar tiempos de entrenamiento excesivos ni aumentar demasiado la complejidad del sistema.

# 8. Evaluación

## 8.1. Métricas clave (AUC, exactitud, etc.)

Para cuantificar la calidad de las predicciones multilabel hemos empleado:

* **ROC-AUC (área bajo la curva ROC)** por etiqueta, que refleja la capacidad de cada modelo de discriminar entre positivos y negativos a distintos umbrales, y se reporta también su media aritmética (“macro-average”) para las siete clases.
* **Exactitud (accuracy)** global, aunque en problemas desbalanceados puede resultar engañosa: un alto porcentaje de negativos favorece falsamente a clases mayoritarias.
* **Precisión, recall y F1-score** por etiqueta, útiles para analizar trade-off entre falsos positivos y falsos negativos, especialmente en las clases menos frecuentes.

De todas ellas, la **ROC-AUC macro** fue la métrica principal de selección, alineada con el criterio oficial de la competición.

## 8.2. Procedimiento de validación (k-fold, hold-out, etc.)

El protocolo de evaluación fue el siguiente:

1. **Hold-out final (15 %)**
   * Se reservó un 15 % de las muestras como test independiente, sin usar en ningún ajuste de parámetros ni selección de features.
2. **Validación cruzada estratificada multilabel (5-fold)**
   * Sobre el 85 % restante, aplicamos IterativeStratification(n\_splits=5, order=1) para generar cinco pliegues que preservan la proporción de positivos/negativos en cada una de las siete etiquetas.
   * En cada fold, el pipeline de preprocesamiento (features derivadas → winsorización → log → escalado) se **ajusta** sólo con los datos de entrenamiento y se **aplica** en validación, evitando fugas de información.

Las métricas se calcularon tanto en cada fold (para guiar la optimización de hiperparámetros) como en el hold-out final (para la evaluación definitiva).

## 8.3. Resultados obtenidos y comparación entre modelos

**LightGBM** obtuvo la **mejor media AUC** (≈ 0.858), superando a RF en todas las clases, especialmente en defectos minoritarios como *Dirtiness* y *Bumps*.

**Random Forest** ocupó la segunda posición (media ≈ 0.825), mostrando solidez y buena separación en *Pastry* y *Z\_Scratch*.

**Regresión Logística** rindió algo por debajo, debido a su limitada capacidad de modelar interacciones no lineales.

En el hold-out final, estos porcentajes se mantuvieron con variaciones menores (< 0.02), confirmando la estabilidad del proceso.

(tablas)

## 8.4. Interpretación y análisis de importancia de variables

Para comprender qué información guió las decisiones del modelo, extraímos la **importancia de features** de LightGBM (promedio de feature\_importances\_):

1. **Top 5 de variables más relevantes**
   1. Max\_to\_Mean\_Luminosity
   2. Aspect\_Ratio
   3. Mean\_Luminosity
   4. Pixels\_Areas (tras log-transform)
   5. Y\_Center\_norm
2. **Insights de negocio**
   1. Un valor alto de **Max\_to\_Mean\_Luminosity** indica defectos muy brillantes (por ejemplo, manchas), facilitando su detección frente a zonas homogéneas.
   2. La **forma** capturada por Aspect\_Ratio discrimina eficazmente entre defectos alargados (*Z\_Scratch*, *K\_Scatch*) y otros más puntuales.
   3. La localización central (X\_Center\_norm, Y\_Center\_norm) ayuda a separar defectos que suelen concentrarse en ciertas zonas de la chapa.
3. **Aciertos y errores**
   1. Las clases **mayoritarias** (*Other\_Faults*, *Pastry*) presentan AUC > 0.88 y baja varianza entre folds, reflejando que el modelo las distingue con alta fiabilidad.
   2. **Dirtiness** (≈ 2.5 % de prevalencia) resultó más desafiante (AUC ≈ 0.78 en LightGBM), debido a su similitud con ruidos de iluminación: aquí los errores de falso negativo se concentran en placas con brillo irregular leve.
   3. Los **falsos positivos** se producen mayoritariamente en casos limítrofes, donde la interacción de luminosidad y área no es suficiente para diferenciar manchas pequeñas de suciedad ligera.

En conjunto, el análisis de importancias confirma que las features sintéticas diseñadas (luminosidad normalizada, proporciones de forma y posición) aportan señal discriminativa sólida. Los patrones de error identificados motivan, en futuras iteraciones, explorar transformadores de texturas o CNNs sobre imágenes sintéticas para reforzar la detección de *Dirtiness* y *Stains*.

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

# 9. Conclusiones y Trabajo Futuro

## 9.1. Discusión de hallazgos y limitaciones

 **Rendimiento alcanzado**:

* El mejor modelo (LightGBM) logró una ROC-AUC promedio multilabel de ≈ 0.858 en validación cruzada y se mantuvo estable (±0.02) en el conjunto hold-out final.
* Las etiquetas con mayor prevalencia (*Other\_Faults*, *Pastry*) alcanzaron AUC > 0.88, mientras que las más raras (*Dirtiness*, *Bumps*) quedaron en torno a 0.78–0.81.

 **Interpretación de resultados**:

* Las features derivadas de luminosidad y forma demostraron ser discriminativas: la relación intensidad máxima/media y la razón de aspecto resultaron críticas.
* El preprocesamiento (winsorización, logaritmo y escalado cuantílico) permitió normalizar distribuciones y mitigar el efecto de outliers sin pérdida de ejemplos extremos, beneficiando especialmente a los clasificadores lineales y las redes ligeras.

 **Limitaciones identificadas**:

1. **Desbalance residual**: A pesar de usar pesos y oversampling, las clases muy minoritarias siguen siendo las que peor AUC consiguen.
2. **Ausencia de información espacial detallada**: El dataset sintético no incluye texturas ni relaciones espaciales inter-píxel, limitando la detección fina de defectos como “Dirtiness” con patrones irregulares sutiles.
3. **Alcance del benchmark**: Al tratarse de datos sintéticos, el rendimiento real en líneas de producción reales podría diferir—la distribución de defectos y el ruido de adquisición varían en entornos industriales.

## 9.2. Aplicaciones prácticas e implicaciones en la industria

 **Inspección automatizada**: Un clasificador con AUC ~0.86 puede integrarse como sistema de detección preliminar en la cadena de fabricación, señalando las placas con alta probabilidad de defectos para revisión humana.

 **Reducción de costes**: Al automatizar la inspección, se minimiza la dependencia de operarios expertos, se acelera el proceso y se reduce el porcentaje de falsos negativos que llegan al cliente final.

 **Mantenimiento predictivo**: Los patrones de defectos identificados pueden correlacionarse con condiciones de la línea (velocidad de corte, estado de rodillos, variaciones de temperatura), alimentando sistemas de mantenimiento predictivo y optimizando parámetros de proceso

## 9.3. Líneas de mejora y posibles continuaciones

**Modelos basados en visión**

* Incorporar CNNs o arquitecturas híbridas (tabular + visión) sobre imágenes sintéticas/ reales para aprovechar patrones de textura y bordes más allá de las estadísiticas globales.

**Aumentos de datos específicos**

* Generar transformaciones realistas (ruido gaussiano, distorsiones de iluminación) y aplicar técnicas de *data augmentation* para entrenar modelos más robustos frente a variabilidad de adquisición.

**Optimización de umbrales y calibración**

* Ajustar puertas de decisión por etiqueta maximizando F1-score o minimizando coste de error según las consecuencias industriales de falsos positivos/negativos.

**Feature selection avanzada**

* Probar métodos de *embedded selection* y *auto-ML* que identifiquen automáticamente subconjuntos de variables, reduciendo dimensionalidad y tiempos de inferencia.

**Integración en línea de producción**

* Desplegar un prototipo en tiempo real, midiendo latencia de inferencia y evaluando precisión en un entorno real.

**Monitoreo continuo y retraining**

* Implementar pipelines de MLops que recojan datos de nuevas placas, monitoricen drift en distribuciones y reentrenen periódicamente para adaptarse a cambios en la materia prima o el equipamiento.

# 10. Referencias

Lista completa de fuentes utilizadas, desde artículos académicos, documentación de librerías y enlaces a la competición de Kaggle.

*Lista completa de fuentes utilizadas, desde artículos académicos, documentación de librerías y enlaces a la competición de Kaggle.*