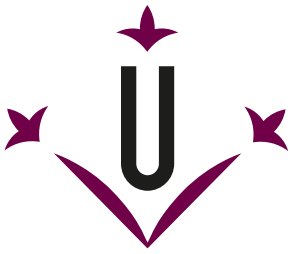
Universidad de Lleida
Escuela Politécnica Superior
Prof. Josep Lluís Lérida



Aplicación de imágenes hiperespectrales HSI-NIR para la clasificación de granos de trigo contaminados

Christian López García

10 de octubre de 2023

Índice de contenido

Índice de código

Índice de figuras

Índice de tablas

Glosario de acrónimos

BIL Bandas intercaladas por líneas.

CSV Comma Separated Values.

DON Deoxinivalenol.

ML Machine Learning.

NIR-HSI Near-InfraRed HyperSpectral Imaging.

PCA Principal Component Analysis.

PNF Polynomial Features.

SSC Standard Scaler.

1. Introducción

Nuestro proyecto está enfocado en detectar un problema de seguridad alimentaria determinando la contaminación por Deoxinivalenol (DON), una toxina producida por hongos del género Fusarium, tales como Fusarium graminearum y Fusarium culmorum.

Es interesante como sociedad que los alimentos que consumimos directa o indirectamente pasen un control de calidad, pues dicha toxina tiene efectos negativos sobre la salud humana y animal [6].

1.1. Motivación

Se nos presentó la posibilidad de aportar de forma remunerada en este proyecto y además utilizarlo como trabajo de final de grado. Otro de los principales motivantes ha sido la utilización de Python, pues este proyecto me ha facilitado la exploración de nuevas librerías y profundización de algunas librerías que hemos utilizado en clase en proyectos de ML.

1.2. Objetivos

El principal objetivo es realizar un proceso automatizado de análisis de imágenes hiperespectrales, el cual nos muestre qué granos de trigo están contaminados por la toxina. Emulando una parte del proceso productivo en una cinta transportadora, es decir, tratando de optimizar el tiempo de ejecución de los análisis y preparando el programa para ser ejecutado durante un tiempo indefinido.

Los pasos que hemos discernido del objetivo principal para realizar proyecto son los siguientes:

1. Entender los datos de los resultados de laboratorio.
2. Entender el existente proceso de análisis de imágenes hiperespectrales.
3. Tratar de mejorar el actual proceso de análisis de las imágenes.
4. Entender el actual proceso de entranamiento y análisis de modelos de ML.
5. Tratar de mejorar el preprocesado de los datos, tanto como el entrenamiento de nuevos modelos para tratar de obtener mejores resultados.
6. Entrenar buenos modelos de ML que, según las imágenes hiperespectrales, sean capaces de predecir si un grano está contaminado.
7. Una vez entrenados los modelos básicos, tratar de ajustarlos a los datos con *Hyperparameter tuning*.
8. Añadir nuevos modos de ejecución del programa para emular la carga continua de

imágenes que habría en un caso de uso real, además de prepararlo para que se pueda ejecutar por un tiempo indefinido como en un caso real, guardando los resultados de las predicciones.

1.3. Estructura del documento

El documento tiene un orden lógico. Primeramente, introduciremos el problema real indicando cómo se toman los datos en laboratorio, cómo analizaremos estos datos y cómo la ley ampara este ámbito.

A continuación, explicaremos los conceptos básicos de ML que hemos utilizado para la realización del proyecto, pondremos algunos ejemplos y comentaremos las principales fases de un proyecto de ML.

Antes de diseñar el proyecto, habiendo analizado los datos de partida, probaremos diferentes formas de preprocesarlos para que los modelos entrenados con estos tengan mejores resultados.

Por último, analizaremos y compararemos los resultados, tratando de ver la viabilidad de aplicarlos en una aplicación de tiempo real.

Además, podemos encontrar algunas definiciones adicionales, acrónimos y el anexo.

2. Análisis del problema

Para ello, tenemos una base de datos de imágenes hiperespectrales con formato BIL (**figura 8**), de la cual hemos extraído los píxeles que forman los granos, por otro lado, tenemos otra base de datos con el estado de contaminación de estos mismos granos.

2.1. NIR-HSI

Near-InfraRed HyperSpectral Imaging (NIR-HSI) es una tecnología rápida, no destructiva y precisa que nos permite hacer inspecciones de calidad, la cual ha demostrado su potencial en los últimos años [2]. Es una técnica de imagen química basada en la espectroscopia de reflectancia (la luz reflejada por los materiales), la cual es capaz de caracterizar compuestos orgánicos y algunos minerales [13].

Como hemos comentado en la introducción, nuestro objetivo es conseguir un modelo que prediga lo mejor posible qué granos de una Imagen hiperespectral contienen granos contaminados con DON.

2.2. Separación del grano contaminado

El valor de contaminación Deoxinivalenol () de un grano lo obtenemos de hacer un proceso químico que no entra dentro del propósito de este proyecto. Lo único que nos interesa es que el valor de la contaminación es un valor real. Además, sabemos que desde el laboratorio se considera que un grano está contaminado a partir de una concentración de $1250 \mu g/kg$. De esta forma, aunque es un valor real, podemos utilizarlo como tal o considerar solamente si está contaminado o no. Es decir, considerarlo como un problema de Clasificación o de Regresión.

Aunque tenemos el valor continuo de contaminación, podemos reemplazarlo directamente por una columna booleana que indique si está contaminado. De esta forma pasaríamos de un problema de regresión, el cual nos permite predecir valores continuos, a uno de clasificación para predecir valores discretos, si está contaminado o no.

3. Marco teórico, conceptos clave del ML

> 'ML: the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data' [12]

Es decir, un sistema que es capaz de inferir patrones de unos datos y realizar predicciones sobre nuevos datos a raíz de los patrones encontrados.

3.1. Tipos de aprendizaje y modelos

Hay cuatro tipos de aprendizaje en el ML: supervisado, no supervisado, semi-supervisado y de refuerzo [10].

Como parte de nuestros datos están etiquetados, podemos utilizar tanto el aprendizaje supervisado como el semi-supervisado. En un sistema supervisado se requiere que todos los datos estén etiquetados con la categoría deseada, en nuestro caso la contaminación. Sin embargo, para el semi-supervisado no es necesario que todos los datos estén etiquetados, pues utilizan tanto los datos etiquetados como los que no (datos parcialmente etiquetados) para tratar de generalizar mejor [10].

No utilizaremos el no supervisado, pues consiste en un conjunto de técnicas utilizadas para agrupar automáticamente datos no etiquetados a base de similitudes sin tener supervisión previa. El objetivo es que los datos dentro de un mismo grupo sean más similares entre sí que con los otros grupos [4]. Sin embargo, como tenemos una cantidad más o menos considerable de información etiquetada, no nos hace falta.

Tampoco utilizaremos el aprendizaje por refuerzo, pues no se encuentra implementado dentro de la librería que utilizamos *sklearn* y tampoco está dentro de los objetivos del proyecto.

3.2. Fases del desarrollo de un proyecto de ML

Nuestro proyecto ha tenido las fases que nombramos a continuación:

1. Obtención de datos (*apartado 4.1*)
    a) Recolección de información de los BIL.
2. Preparación de datos (*apartado 4.2*)
    a) Selección de métodos de preprocesado
3. Entrenamiento de los modelos (*apartado 4.3*)
    a) Entrenamiento 'básico' de modelos
    b) Selección y refinamiento

4. Evaluación de resultados (*apartado 5*)
5. Monitoreo de los modelos

Para generalizar la hoja de ruta de un proyecto es más o menos como la de la *figura 1*. Cabe destacar que el proceso de entrenar un modelo de ML, como bien muestra la imagen, es un proceso cíclico en búsqueda de mejores resultados a base de probar metodologías distintas, añadir pasos, el cambio de requisitos, etc.

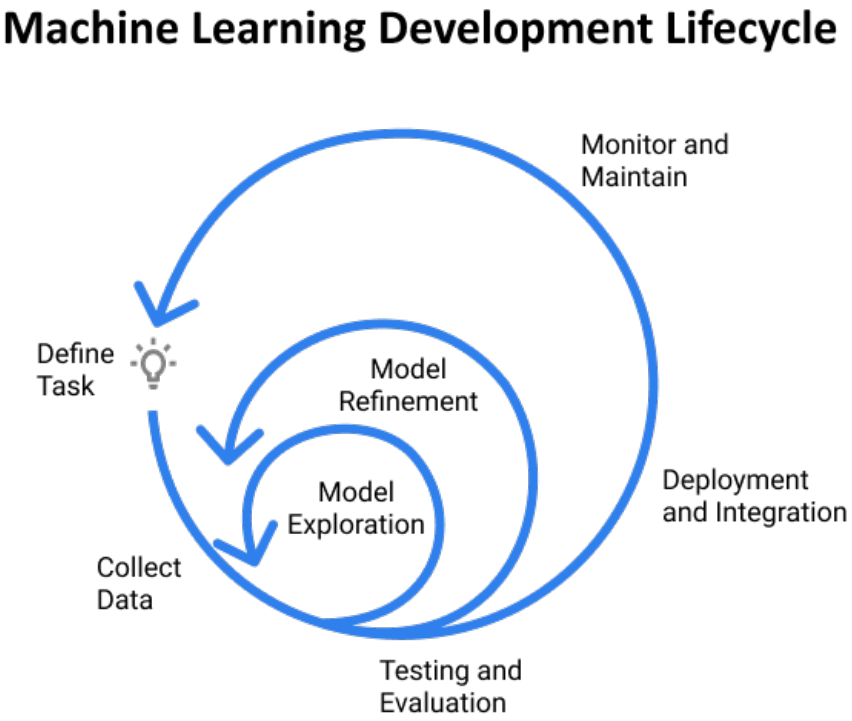


Figura 1: Esquema del proceso del desarrollo de un proyecto de ML, fuente [11]

3.3. Métricas de evaluación de modelos

Antes de continuar con el entrenamiento de modelos, su mejora y la selección de pasos del preprocesado; necesitamos algún tipo de métrica para evaluar y comparar los modelos. A continuación, explicaremos las que hemos utilizado para los diferentes tipos de modelos.

3.3.1. Clasificación supervisada y semi-supervisada

Primeramente, explicaremos las partes de una matriz de confusión como la que podemos ver en la *figura 2*. Podemos ver que tiene cuatro partes:

- *True Positive (tp)*, las muestras que el modelo clasifica como positivas que realmente son positivas.

- *False Positive (fp)*, las muestras que el modelo clasifica como positivas que realmente son negativas.
- *True Negative (tn)*, las muestras que el modelo clasifica como negativas que realmente son negativas.
- *False Negative (fn)*, las muestras que el modelo clasifica como negativas que realmente son positivas.

Estos cuatro valores son la base del resto de métricas que comentaremos a continuación.

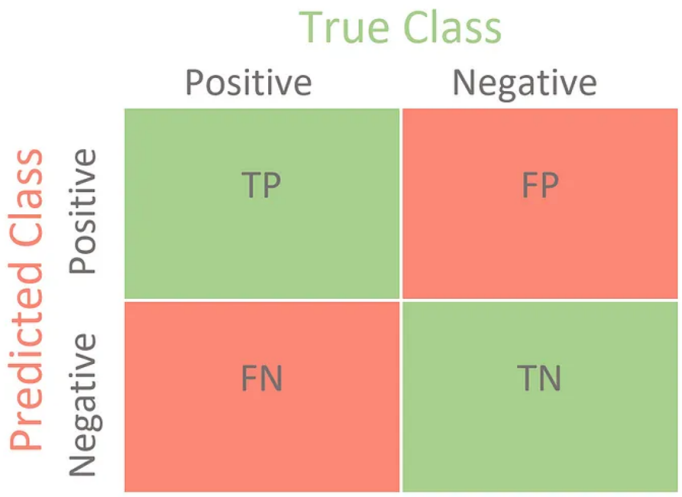


Figura 2: Ejemplo de matriz de confusión, fuente [5]

Antes de explicar estas métricas, comentaremos otras más básicas que son utilizadas para calcular que utilizaremos [14]. *Sensitivity* permite saber la probabilidad de que una predicción positiva sea realmente positiva.

$$tp/(tp + fn) \tag{1}$$

Specificity permite saber la probabilidad de que una predicción negativa sea realmente negativa.

$$tn/(tn + fp) \tag{2}$$

Recall indica la capacidad del modelo para encontrar todas las muestras positivas.

$$tp/(tp + fn) \tag{3}$$

Precision indica la precisión de las predicciones positivas, es decir, la habilidad del clasificador para no predecir como positiva una muestra que es negativa.

$$tp/(tp + fp) \tag{4}$$

Con estas métricas básicas podemos definir las que realmente compararemos, ya que tienen en cuenta el balanceo del Dataset:

fbeta score es la media harmónica ponderada entre *Precision* y *Recall*. El parámetro *beta* representa la importancia de *Recall* por encima de *Precision*. Es decir, $beta > 1$ le da más importancia a *Recall* y, contrariamente, $beta < 1$ le da más peso a *Precision*. [9]

$$f_\beta = (1 + \beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall} \tag{5}$$

Balanced accuracy es la media aritmética de *Sensitivity* y *Specificity*. Se utiliza principalmente cuando tratamos con datos desbalanceados. [3]

$$balanced\_accuracy = \frac{sensitivity + specificity}{2} \tag{6}$$

4. Diseño del proyecto, primera iteración

4.1. Análisis del problema y de los datos

Considerando que los datos contenidos en los archivos BIL que hemos recibido desde el laboratorio contienen 168 columnas con información de las diferentes longitudes de onda [18] y además el nivel de contaminación DON de cada grano, podemos generar un Dataset con los datos de cada grano, haciendo la media de cada longitud de onda por cada píxel del mismo grano. Una vez hemos extraído la información de todos los archivos BIL y la hemos guardado en un archivo CSV, podemos entrenar los modelos.

Cabe recalcar que para los modelos de regresión nos hemos guardado el valor de contaminación DON explícitamente, para los de clasificación si el valor DON supera los límites de contaminación y hemos preparado un Dataset aparte con algunos archivos Bandas intercaladas por líneas (BIL) que contienen granos que no han sido analizados químicamente para el entrenamiento semi-supervisado.

En esta primera iteración nos hemos centrado en intentar encontrar un buen modelo de clasificación y mejorar sus resultados lo máximo posible.

4.2. Preprocesado de datos

Ahora que tenemos los datos en un solo archivo CSV, el siguiente paso es preparar los datos. Para ello, hemos realizado el siguiente proceso:

1. Eliminación de columnas
2. Codificación
3. Valores atípicos (*outliers*)
4. Balanceo de datos
5. Separación en datos de entreno y de prueba (*train-test-split*)
6. Preprocesado común, (*pipeline*)
   a) Separación de datos aplicando la primera derivada
   b) Aumento de dimensionalidad (*Polynomial Features (PNF)*)
   c) Estandarización (*Standard Scaler (SSC)*)
   d) Reducción de dimensionalidad (*Principal Component Analysis (PCA)*)

4.2.1. Eliminación de columnas, codificación y valores atípicos

Existen datos que nos interesan como medida de seguridad, pero no nos interesa que un modelo entrene con ellos, ya que podría inferir patrones irreales o incluso memorizárselos

y hacer Overfitting. En nuestro caso, un ejemplo sería una columna que indicase el número identificador del grano o el archivo del cual se han extraído los datos.

La codificación consiste en transformar columnas con datos categóricos en columnas de datos numéricos, pues los modelos de ML funcionan mejor con valores numéricos. En nuestro caso, la única columna con valores no numéricos era la columna de la contaminación, que podía tomar los valores *{B, C}*, así que lo codificamos manualmente como se muestra a continuación:

$$B \longrightarrow 0, C \longrightarrow 1$$

Los valores atípicos u *outliers* son aquellos valores inusuales en los datos que pueden distorsionar nuestros análisis estadísticos. Sin embargo, se debe tener en cuenta que puede haber mucha variación en la naturaleza de nuestro problema. Por ello, se deben diferenciar los *outliers* que se pueden incluir en los datos y los que no. En nuestro caso, los hemos incluido todos, pues al tener cada grano 168 columnas de datos, la mayoría tenía alguna columna que no entraba dentro de lo "normal', como por ejemplo en la *figura 3*. Por lo que después de probar a quitar todos los granos que alguna de sus columnas fuera un *outlier* (con el *código 2*), nos quedamos sin datos.

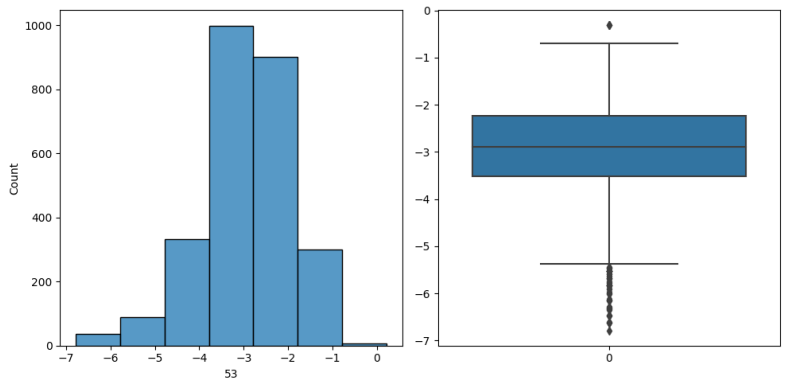


Figura 3: Representación de los *outliers* de la columna 53 en un gráfico de velas, fuente propia, *código 1*

4.2.2. Balanceo de datos

Podemos ver en la *figura 4* que los datos no están balanceados, es decir, que la columna que nos interesa 'Contaminación' no tiene un número de datos similares en cada clase. En nuestro caso, como en general es más complicado encontrarse un grano contaminado, tenemos más granos sanos.

Al tener las clases desbalanceadas tenemos varias opciones:

1. Usar métodos de evaluación que tengan en cuenta el desbalance de las clases.

2. Balancear el Dataset utilizando tanto *undersampling* como *oversampling* (explicados en el *apartado 6.1*).

Para esta primera iteración, nos es más sencillo utilizar métricas que tengan en cuenta el número de instancias de cada clase a la hora de evaluar los resultados, es decir, que tengan en cuenta el desbalance del Dataset.

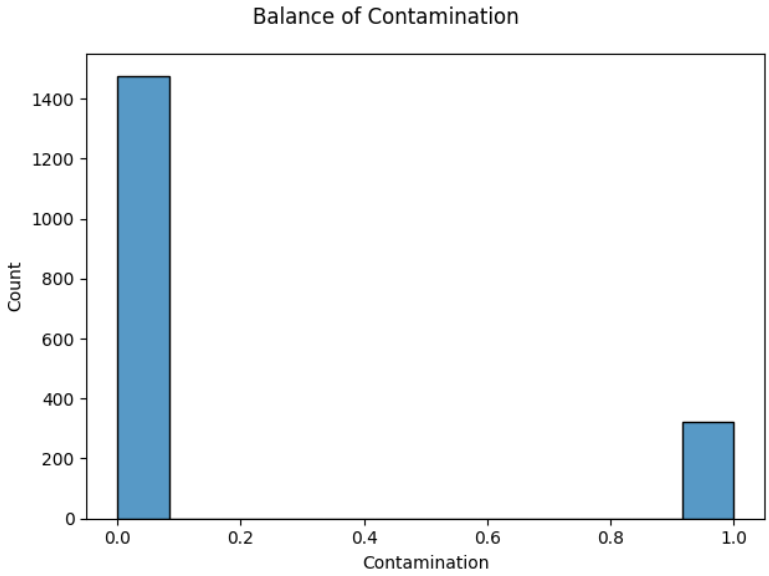


Figura 4: Balanceo de la clase objetivo del Dataset, fuente propia

4.2.3. Separación de datos de entreno y de prueba

En todo el proyecto hemos estado utilizando la librería sklearn, esta tiene un submódulo con una función sklearn.model_selection.train_test_split para separar el Dataset en entreno y prueba. De todas las opciones, cabe recalcar una que hemos habilitado, pues como en el Dataset original las etiquetas no están balanceadas y hay muchos más granos sanos que contaminados, es importante que los granos contaminados estén igualmente representados tanto en el conjunto de prueba como en el de entreno. Esto se consigue con la opción *stratify*.

4.2.4. Preprocesado común

Después de balancear los datos, debemos transformarlos para que tengan algunas propiedades que suelen ser preferibles a la hora de entrenar ciertos modelos. Es decir, hay modelos que les vendrá mejor tener los datos estandarizados, por ejemplo, y otros que no hará falta o será contraproducente, al final será la prueba y error lo que nos diga qué pasos son preferibles.

Antes de probar diferentes formas de preprocesar los datos, hemos preparado un entorno para entrenar una batería de modelos distintos para ver su efectividad.

Un primer paso, recomendado desde el laboratorio, ya que era lo que utilizaban ellos en su modelo estadístico, es aplicar la primera derivada. El aplicar la derivada permite mínimamente separar los granos contaminados de los que no. Para probarlo, podemos comparar los resultados de entrenar con o sin derivar en la *tabla 1* y, efectivamente, obtenemos resultados algo mejores.

	Derivative							No preprocessing						
Model Name	Train score	Test score	f1score	f0.5score	f2score	ROC/AUC score	Balanced accuracy	Train score	Test score	f1score	f0.5score	f2score	ROC/AUC score	Balanced accuracy
XGBoost	82	82	0	0	0	50	50	82	82	0	0	0	50	50
Stochastic Gradient Descent	18	18	30.508	21.531	52.326	50	50	18	18	30.508	21.531	52.326	50	50
Random Forest	99.926	87.778	49.541	69.948	38.352	66.531	66.531	99.556	84.889	52.778	57.057	49.096	70.069	70.069
Quadratic Discriminant Analysis	100	82	0	0	0	50	50	100	82	0	0	0	50	50
Multi-Layer Perceptron	83.259	80.667	8.421	14.599	5.917	51.114	51.114	82	82	0	0	0	50	50
Linear Discriminant Analysis	85.333	78.222	20.968	25.692	17.711	53.96	53.96	84.889	78	22.047	26.415	18.919	54.306	54.306
LightGBM	81.852	72	47.934	40	59.794	71.846	71.846	75.778	64	37.692	30.74	48.708	62.632	62.632
K-Neighbors	100	86.889	56.296	63.973	50.265	71.289	71.289	100	91.778	74.83	79.71	70.513	82.46	82.46
Hist Gradient Boosting	78.444	70.444	44.813	37.448	55.785	68.97	68.97	72.815	63.556	33.871	28.037	42.77	58.988	58.988
Extra Trees	99.63	90.444	69.504	76.324	63.802	78.756	78.756	94.667	81.111	56.853	51.376	63.636	76.438	76.438
Decision Tree	56.963	58.889	41.27	31.957	58.244	67.224	67.224	82	82	0	0	0	50	50
						61.79	61.79						59.53572727	59.53572727

Tabla 1: Comparación de los resultados de entrenar utilizando la derivada y sin preprocesar. Fuente propia.

El siguiente paso que podemos probar es aplicar la transformada, como los resultados habiendo derivado los datos son mejores, transformaremos los datos después de derivarlos. Podríamos probar todas las permutaciones de los diferentes pasos de preprocesado, sin embargo, en esta primera iteración tan solo nos interesa encontrar un resultado decente. Podríamos hacerlo en una segunda o tercera iteración si lo vemos necesario. La transformada consiste en transformar cada columna para que tengan una forma gaussiana [16]. Como estamos utilizando *sklearn*, tenemos una clase que nos aplica la transformada sklearn.preprocessing.PowerTransformer, por defecto esta clase además te aplica una estandarización de los datos, pero la hemos deshabilitado para aplicarla en otro paso. Podemos ver los resultados de entrenar utilizando la derivada y transformada en la *tabla 2*, podemos ver que los resultados son peores, así que continuaremos las pruebas con la derivada.

Como hemos comentado antes, el proceso de estandarización de los datos lo realizaremos ahora con sklearn.preprocessing.StandardScaler. La estandarización consiste en, columna por columna, restarles la media de los valores (centrar los valores en torno al 0) y dividir por la varianza (para que la desviación tienda a 1). De esta forma tenemos los datos ya preparados para

Polynomial features,

	Transformer and derivative							Derivative						
Model Name	Train score	Test score	f1score	f0.5score	f2score	ROC/AUC score	Balanced accuracy	Train score	Test score	f1score	f0.5score	f2score	ROC/AUC score	Balanced accuracy
XGBoost	82	82	0	0	0	50	50	82	82	0	0	0	50	50
Stochastic Gradient Descent	18	18	30.508	21.531	52.326	50	50	18	18	30.508	21.531	52.326	50	50
Random Forest	99.926	87.778	49.541	69.948	38.352	66.531	66.531	99.926	87.778	49.541	69.948	38.352	66.531	66.531
Quadratic Discriminant Analysis	100	83.333	19.355	34.884	13.393	55.149	55.149	100	82	0	0	0	50	50
Multi-Layer Perceptron	82	82	0	0	0	50	50	83.259	80.667	8.421	14.599	5.917	51.114	51.114
Linear Discriminant Analysis	85.185	76.667	18.605	21.978	16.129	52.529	52.529	85.333	78.222	20.968	25.692	17.711	53.96	53.96
LightGBM	81.852	72	47.934	40	59.794	71.846	71.846	81.852	72	47.934	40	59.794	71.846	71.846
K-Neighbors	100	80.222	25.21	32.189	20.718	56.143	56.143	100	86.889	56.296	63.973	50.265	71.289	71.289
Hist Gradient Boosting	78.444	70.444	44.813	37.448	55.785	68.97	68.97	78.444	70.444	44.813	37.448	55.785	68.97	68.97
Extra Trees	99.63	90	68.966	74.184	64.433	78.967	78.967	99.63	90.444	69.504	76.324	63.802	78.756	78.756
Decision Tree	56.963	58.889	41.27	31.957	58.244	67.224	67.224	56.963	58.889	41.27	31.957	58.244	67.224	67.224
						60.669	60.669						61.79	61.79

Tabla 2: Comparación de los resultados de entrenar utilizando, transformando y derivando los datos; frente a solo derivando. Fuente propia.

StandardScaler

PCA

Pipeline

4.3. Entrenamiento de modelos, selección e *hypertuning*

Ahora que tenemos los datos preparados, podemos entrenar modelos. Aunque hay muchos tipos de modelos preparados en la librería, cada uno funciona de una forma. Podríamos investigar cuál funciona mejor con nuestros datos, sin embargo, como el Dataset que tenemos no es muy grande, podemos entrenar todos sin perder mucho tiempo y mirar cuál tiene mejores resultados. Estos resultados del primer entrenamiento los podemos ver en la *tabla 3*

	Model Name	Train score	Test score	Recall	Precision	f1score	f0.5score	f2score	ROC/AUC score	Balanced accuracy
mlp	Multi-Layer Perceptron	91.902	90.707	86.15	94.817	90.276	92.947	87.754	90.714	90.714
xgb	XGBoost	92.226	88.904	84.211	92.966	88.372	91.072	85.827	88.911	88.911
rf	Random Forest	94.678	88.766	83.38	93.478	88.141	91.267	85.221	88.773	88.773
lgbm	LightGBM	91.856	88.488	84.488	91.867	88.023	90.29	85.867	88.494	88.494
qda	Quadratic Discriminant Analysis	87.043	88.488	77.839	98.944	87.132	93.854	81.308	88.503	88.503
et	Extra Trees model	85.886	85.437	78.116	91.558	84.305	88.512	80.479	85.447	85.447
knn	K-Neighbors	100.0	83.218	70.36	94.776	80.763	88.625	74.182	83.236	83.236
hgb	Hist Gradient Boosting	81.259	82.802	76.177	87.859	81.602	85.245	78.258	82.811	82.811
lda	Linear Discriminant Analysis	80.102	82.802	73.961	89.899	81.155	86.185	76.68	82.814	82.814
sgd	Stochastic Gradient Descent	76.955	80.999	73.961	86.129	79.583	83.385	76.112	81.008	81.008
dt	Decision Tree	72.975	76.144	62.05	86.486	72.258	80.172	65.766	76.164	76.164

Tabla 3: Resultados del primer entrenamiento de modelos. Fuente propia.

Una vez hemos entrenado estos modelos vemos que hay muchos que hacen *overfitting*,

para verlo más claramente podemos ver sus curvas de aprendizaje en la *figura 5*. La que más claramente se ve es en la *figura 5g*, nos hemos de fijar en la diferencia de resultados en el Dataset sobre el que se ha entrenado y el de prueba. Para evitar el Overfitting vamos a seleccionar los mejores modelos y a hacer *hyperparameter tuning*.

4.3.1. Selección de modelos e *hyperparameter tuning*

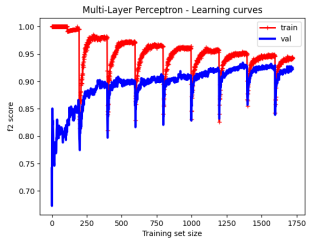
Antes de elegir el mejor modelo, explicaremos las métricas con los que los evaluamos. Para ello,

Antes de ajustar los modelos, seleccionaremos aquellos con los mejores resultados. Viendo la *tabla 3* vemos que los cinco mejores modelos han sido los siguientes:

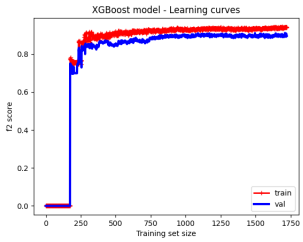
1. Multi-Layer Perceptron
2. XGBoost
3. Random Forest
4. LightGBM
5. Quadratic Discriminant Analysis

Antes de continuar, descartaremos el *Random Forest*, pues es de los que más tarda en entrenar junto con el *Multi-Layer Perceptron*, sin embargo, a diferencia del segundo, obtenemos peores resultados, aunque se podría trabajar con él de todas formas.

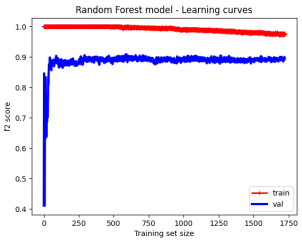
De los cuatro modelos con los que nos hemos quedado al final, solamente dos hacen Overfitting, de todas formas intentaremos mejorar los resultados de los cuatro. Para ello, primero haremos RandomizedSearchCV. Una vez hemos encontrado buenos resultados en el rango de parámetros definido, utilizamos GridSearchCV



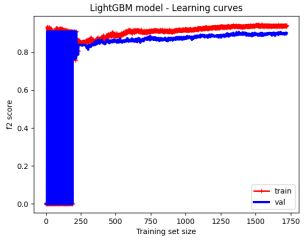
(a) Curva de aprendizaje del Multilayer Perceptron



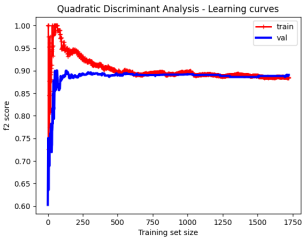
(b) Curva de aprendizaje del XGBoost



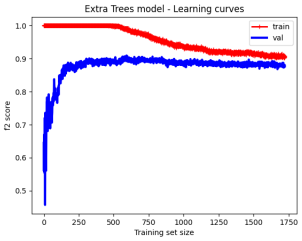
(c) Curva de aprendizaje del Random Forest



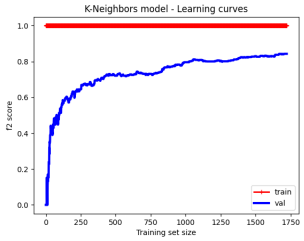
(d) Curva de aprendizaje del LightGBM



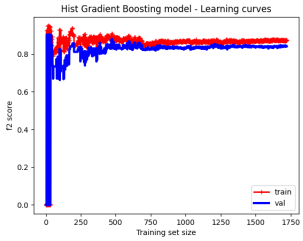
(e) Curva de aprendizaje del Quadratic Discriminant Analysis



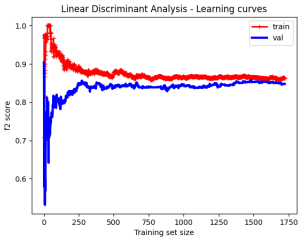
(f) Curva de aprendizaje del Extra Trees



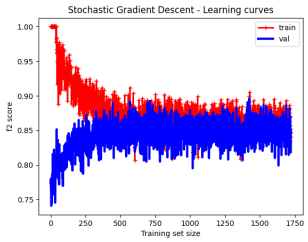
(g) Curva de aprendizaje del K-Neighbors



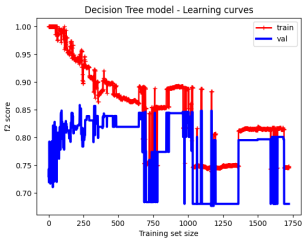
(h) Curva de aprendizaje del Hist Gradient Boosting



(i) Curva de aprendizaje del Linear Discriminant Analysis



(j) Curva de aprendizaje del Stochastic Gradient Descent



(k) Curva de aprendizaje del Decision Tree

Figura 5: Curvas de aprendizaje de los modelos entrenados según el tamaño del Dataset de entrenamiento. Fuente propia, *código 3*

5. Análisis y comparación de resultados

6. Segunda iteración

6.1. Balanceo de datos

La toma de nuevos datos reales es costosa, y como teníamos relativamente pocos datos para entrenar el balanceo es esencial. Podemos hacer unos entrenamientos y comparar los resultados para ver si es necesario balancear, los resultados los podemos ver en la *tabla 4*. Fijándonos en las columnas a partir de *f1score*, ya que estas son las métricas que ignoran el balanceo de los datos, podemos ver que, son mejores los modelos entrenados sobre el Dataset balanceado.

	Balanced dataset					Unbalanced dataset				
Model Name	Train score	Test score	f1score	f0.5score	Balanced accuracy	Train score	Test score	f1score	f0.5score	Balanced accuracy
XGBoost model	92.051	87.669	87.055	89.736	87.669	82	82	0	0	50
Stochastic Gradient Descent	79.584	78.862	76.506	81.988	78.862	46.37	47.778	34.54	25.985	59.003
Random Forest model	94.399	87.398	86.58	90.09	87.398	99.926	85.333	35.294	54.545	60.705
Quadratic Discriminant Analysis	87.94	87.669	86.191	92.871	87.669	81.778	79.333	34.043	37.383	59.937
Multi-Layer Perceptron	89.205	85.908	85.014	88.376	85.908	86.889	81.111	17.476	26.627	53.794
Linear Discriminant Analysis	82.746	80.759	78.743	84.026	80.759	82.815	80.667	2.247	4.425	49.669
LightGBM model	92.141	87.127	86.409	89.402	87.127	57.778	52	39.665	29.857	65.914
K-Neighbors model	100	83.333	81.048	88.314	83.333	100	85.111	46.4	56.42	65.869
Hist Gradient Boosting model	82.294	81.436	79.764	84.322	81.436	73.556	62	33.463	27.389	58.522
Extra Trees model	86.224	84.688	82.695	89.701	84.688	95.481	83.111	52.5	52.897	70.912
Decision Tree model	80.352	79.268	76.279	83.503	79.268	82	82	0	0	50

Tabla 4: Tabla comparativa de los resultados de entrenar los modelos con los datos balanceados y sin balancear, después de preprocesar los datos. Fuente propia.

Para balancear el Dataset, hacemos dos procedimientos: *undersampling* y *oversampling*. Como su nombre indica, *undersampling* es una técnica para reducir las muestras de la clase mayoritaria, en nuestro caso los granos sin contaminar. Para ello, utilizamos la librería imblearn que fue diseñada específicamente para la clasificación de clases desbalanceadas. En ella, podemos encontrar diferentes algoritmos para reducir las muestras de la clase que queramos de los cuales nos hemos quedado con los EditedNearestNeighbours. Este algoritmo en resumen elimina las muestras que no se parezcan demasiado a sus vecinos [19]. Para hacer *oversampling* utilizamos la librería sdv, más específicamente el GaussianCopulaSynthesizer, el cual utiliza métodos estadísticos clásicos para generar datos sintéticos parecidos a los reales. Una vez generados los datos sintéticos, podemos comprobar su calidad generando un reporte de calidad de la con la misma librería *sdv*, utilizando la función evaluate_quality la cual compara los datos generados con los reales y nos da los resultados de la *figura 6*.

```
Generating report ...
(1/2) Evaluating Column Shapes: : 100%|██████████| 168/168 [00:00<00:00, 1388.12it/s]
(2/2) Evaluating Column Pair Trends: : 100%|██████████| 14028/14028 [01:16<00:00, 184.43it/s]

Overall Quality Score: 97.07%

Properties:
- Column Shapes: 94.38%
- Column Pair Trends: 99.76%
<sdmetrics.reports.single_table.quality_report.QualityReport object at 0x00000294F7A4BC10>
```

Figura 6: Reporte de calidad de los datos generados sintéticamente, fuente propia.

Una vez aplicados los dos algoritmos obtenemos un Dataset balanceado como el de la *figura 7*.

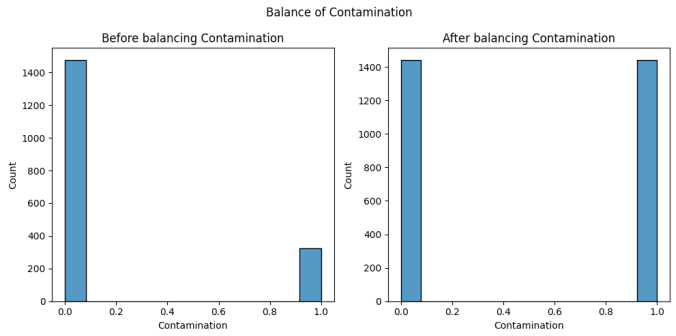


Figura 7: Comparación del balanceo de la clase objetivo del Dataset antes y después de balancear, fuente propia, código 4

7. Comparación de resultados finales

8. Discusión

8.1. Entorno de entrenamiento y ejecución

8.2. Aplicación en entorno real

9. Conclusiones

10. Anexo

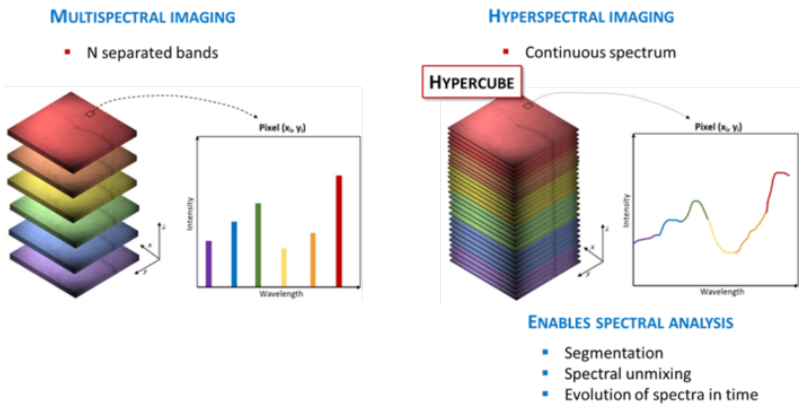


Figura 8: Esquemas para almacenar los valores de píxel reales de una imagen en un archivo BIL, referencia: [8]

Code 1 Gráfico de velas de los outliers

```
def plots_outliers(outliers_df: pd.DataFrame, column=53):
    fig, axes = plt.subplots(ncols=2, figsize=(10, 5))
    sns.histplot(outliers_df[column], binwidth=1, ax=axes[0])
    sns.boxplot(outliers_df[column], ax=axes[1])
    fig.tight_layout()
    plt.show()
```

Code 2 Detección de outliers utilizando *zscore*

```
...
    zscore = np.abs(stats.zscore(df))
    data_clean = df[(zscore < 3).all(axis=1)]
    df = data_clean
...
```

Code 3 Gráfico de curvas de aprendizaje

```
def plots_balancing(df, df_balanced):
    fig, axis = plt.subplots(ncols=2, figsize=(10, 5))
    fig.suptitle(f'Balance of {src.utils.dev_config.OBJECTIVE_COLUMN}')
    axis[0].set_title(f'Before balancing {src.utils.dev_config.OBJECTIVE_COLUMN}')
    axis[1].set_title(f'After balancing {src.utils.dev_config.OBJECTIVE_COLUMN}')
```

```
    plot_balance(df, ax, 0)
    plot_balance(df_balanced, ax, 1)
    f.tight_layout()
    plt.show()
z

def plot_balance(df, axis, axis_number):
    return sns.histplot(df[src.utils.dev_config.OBJECTIVE_COLUMN],
     ↪  ax=axis[axis_number] if axis_number else axis)
```

Code 4 Gráfico de comparación del balanceo

```
def plot_learning_curves(model, model_name, x, y):
    x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2)
    train_errors, val_errors = [], []

    for m in range(5, len(x_train)):
        model.fit(x_train[:m], y_train[:m])
        y_train_predict = model.predict(x_train[:m])
        y_val_predict = model.predict(x_val)
        train_errors.append(metrics.fbeta_score(y_train[:m], y_train_predict,
         ↪  beta=2))
        val_errors.append(metrics.fbeta_score(y_val, y_val_predict, beta=2))

    plt.title(f'{model_name} - Learning curves')
    plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="train")
    plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")
    plt.xlabel("Training set size")
    plt.ylabel("f2 score")
    plt.legend(loc="best")
    plt.show()
```

Definiciones adicionales

Clasificación cuando el problema consiste en la clasificación de un objeto en una clase en base al índice de similitud del objeto con los que forman cada clase [17].

Dataset o conjunto de datos, corresponde a los contenidos de una única matriz de datos, donde cada columna de la tabla representa una variable en particular, y cada fila representa a un miembro determinado del conjunto de datos que estamos tratando [1].

Deoxinivalenol también conocido como DON o vomitoxina, es una micotoxina relativamente frecuente en cereales como trigo, cebada, avena, etc. Es producido principalmente por los hongos del género *Fusarium*, con mayor frecuencia *Fusarium graminearum* y *Fusarium culmorum* [7].

Imagen hiperespectral imagen formada recopilando información a lo largo de todo el espectro electromagnético. Dividiendo el espectro en muchas bandas más allá de las frecuencias visibles [18].

Overfitting ocurre cuando un modelo se ajusta demasiado a los datos con los que ha entrenado y no generaliza bien.

Regresión cuando el problema no consiste en la clasificación de un objeto en una clase, sino en la predicción de un valor continuo como por ejemplo el peso o el salario [15].

Bibliografía y referencias

[1] *£Qué son datasets y dataframes? | Deusto Formación*.
URL: *https://www.deustoformacion.com/blog/programacion-tic/que-son-datasets-dataframes-big-data#:~:text=Un%20conjunto%20de%20datos%20o,de%20datos%20que%20estamos%20tratando.* (visitado 26-09-2023).

[2] *Application of Near-Infrared Spectroscopy and Hyperspectral Imaging Combined with Machine Learning Algorithms for Quality Inspection of Grape: A Review - PMC*.
URL: *https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9818947/#:~:text=Traditional%20quality%20inspection%20methods%20are,great%20potential%20in%20recent%20years.* (visitado 29-08-2023).

[3] *Balanced Accuracy: When Should You Use It?*
URL: *https://neptune.ai/blog/balanced-accuracy#:~:text=Balanced%20Accuracy%20is%20used%20in,lot%20more%20than%20the%20other.* (visitado 06-10-2023).

[4] *Cluster analysis - Wikipedia*. *https://en.wikipedia.org/wiki/Cluster_analysis*. (Accessed on 09/07/2023).

[5] *Confusion Matrix for Your Multi-Class Machine Learning Model | by Joydwip Mohajon | Towards Data Science*. URL: *https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826* (visitado 04-10-2023).

[6] EFSA Panel on Contaminants in the Food Chain (CONTAM) et al.
«Risks to human and animal health related to the presence of deoxynivalenol and its acetylated and modified forms in food and feed». En: *EFSA Journal* 15.9 (2017), e04718.
DOI: *https://doi.org/10.2903/j.efsa.2017.4718*.
eprint: *https://efsa.onlinelibrary.wiley.com/doi/pdf/10.2903/j.efsa.2017.4718*.
URL: *https://efsa.onlinelibrary.wiley.com/doi/abs/10.2903/j.efsa.2017.4718*.

[7] *ELIKA Seguridad Alimentaria | Deoxinivalenol - ELIKA Seguridad Alimentaria*.
https://seguridadalimentaria.elika.eus/fichas-de-peligros/deoxinivalenol/.
(Accessed on 28/08/2023). Mayo de 2023.

[8] Esri. *Archivos ráster BIL, BIP y BSQArcMap*.
URL: *https://desktop.arcgis.com/es/arcmap/latest/manage-data/raster-and-images/bil-bip-and-bsq-raster-files.htm* (visitado 28-08-2023).

[9] *F-score - Wikipedia*. URL: *https://en.wikipedia.org/wiki/F-score* (visitado 06-10-2023).

[10] Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* OReilly Media, Inc., 2017.

[11] Jordan J. *Organizing machine learning projects: project management guidelines.* Sep. de 2018.
URL: *https://www.jeremyjordan.me/ml-projects-guide/* (visitado 31-08-2023).

[12] *machine learning, n. meanings, etymology and more | Oxford English Dictionary*.
URL: *https://www.oed.com/dictionary/machine-learning_n?tab=factsheet#9938479194* (visitado 31-08-2023).

[13] *NIR Hyperspectral Imaging - Iperion CH*. URL: *http://www.iperionch.eu/project/nir-hyperspectral-imaging/#:~:text=Hyperspectral%20imaging%20is%20a%20chemical,device%20for%20the%20visible%20range).* (visitado 26-09-2023).

[14] *Precision, Recall, Sensitivity, Specificity Very Brief Explanation | by Sean Yonathan T | Analytics Vidhya | Medium*. URL: *https://medium.com/analytics-vidhya/precision-recall-sensitivity-specificity-very-brief-explanation-747d698264ca* (visitado 06-10-2023).

[15] *Regression and Classification | Supervised Machine Learning - GeeksforGeeks*. URL: *https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/* (visitado 26-09-2023).

[16] *sklearn.preprocessing.PowerTransformer scikit-learn 1.3.1 documentation*.
URL: *https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html* (visitado 09-10-2023).

[17] *What are classification problems?*
URL: *https://www.educative.io/answers/what-are-classification-problems* (visitado 26-09-2023).

[18] *What Is Hyperspectral Imaging? | NIREOS*.
https://www.nireos.com/hyperspectral-imaging/. (Accessed on 28/08/2023).

[19] Dennis L Wilson. *Asymptotic properties of nearest neighbor rules using edited data. IEEE Transactions on Systems, Man, and Cybernetics, pages 408421,* 1972.